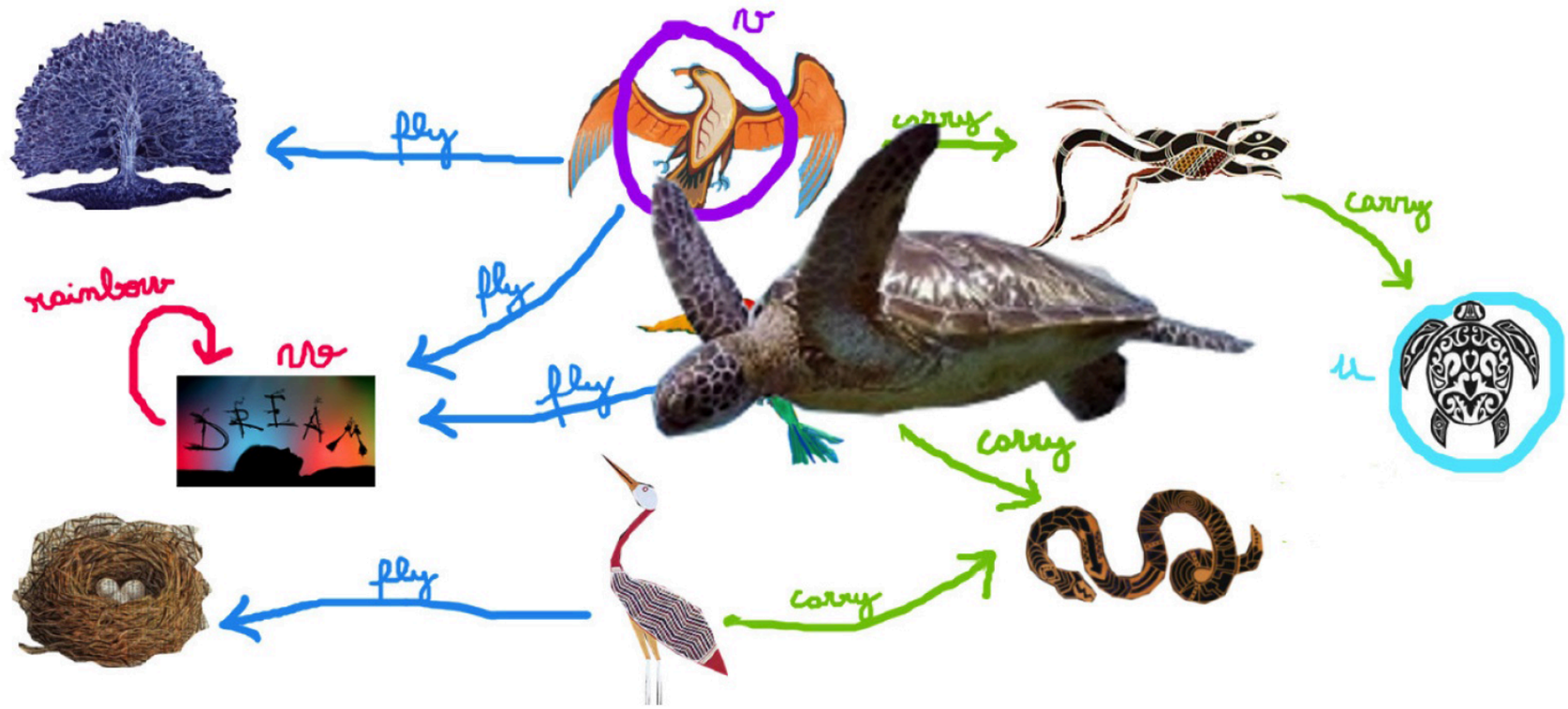
Bases de données spécialisées

Partie 3 : Bases de données orientées graphes



Cristina Sirangelo

Support de cours : Amélie Gheerbrant

IRIF, Université Paris Cité

cristina@irif.fr

Bases de données orientées graphes

- On retrouve des données structurées comme des graphes dans plusieurs applications
  - On vient de voir par exemple les knowledge graphs , mais ce n'est pas le seul exemple
    - Données biologiques, réseaux de telecommunication, réseaux de transport, réseaux sociaux, detection de fraude, systèmes de recommandation, …
  - Requièrent des systèmes de gestion et des langages de requêtes spécifiques (e.g. systèmes de stockages RDF, languages SPARQL)
    - Systems et langages classiques (relationnels) pas adaptés

SGBD relationnels classiques

- Basés sur le modèle relationnel : décomposition des données en relations, ou tables
- Un langage de requêtes standard : SQL
- Données stockées sur disque
- Relations (tables) stockées ligne par ligne
- Système centralisé, avec possibilités limitées de distribution

Force des SGBD relationnels classiques

- Indépendance entre :
  - modèle de données et structures de stockage
  - requêtes déclaratives et exécution
- Requêtes complexes
- Langage d'interrogation standardisé (SQL)
- Optimisation très fine des requêtes, indexes permettant un accès rapide aux données
- Logiciels mûrs, stables, efficaces, riches en fonctionnalités et en interfaces
- Contraintes d'intégrité permettant d'assurer des invariants sur les données
- Gestion efficace de grands volumes de données (gigaoctet, voire téraoctet)
- Transactions (ensembles d'opérations élémentaires garantissant la gestion de la concurrence, l'isolation entre utilisateurs, la reprise sur panne)

Propriétés ACID

Les transactions des SGBD relationnels classiques respectent les propriétés ACID :

Atomicité : l'ensemble des opérations d'une transaction est soit exécuté en bloc, soit annulé en bloc

Cohérence : les transactions respectent les contraintes d'intégrité de la base

Isolation : deux exécutions concurrentes de transactions résultent en un état équivalent à l'exécution sérielle des transactions

Durabilité : une fois une transaction confirmée, les données correspondantes restent durablement dans la base, même en cas de panne

Faiblesse des SGBD relationnels classiques

- Incapable de gérer de très grands volumes de données (de l'ordre du péta-octet)
- Impossible de gérer des débits extrêmes (plus que quelques milliers de requêtes par seconde)
- Peu adapté au distribué (protocoles complexes, peu performants)
- Peu adapté au stockage et à l'interrogation de certains types de données (données hiérarchiques, faiblement structurées, semi-structurées)
- SQL peu optimisé pour certains types de requêtes (requêtes récursives, de chemin, requêtes analytiques)
- Les propriétés ACID entraînent de sérieux surcoûts en latence, accès disques, temps CPU (verrous, journalisation, etc.)
- Performances limitées par les accès disques

NoSQL

- No SQL ou Not Only SQL
- SGBD avec d'autres compromis que ceux faits par les systèmes classiques
- Écosystème très varié
- Fonctionnalités recherchées : modèle de données différent, passage à l'échelle, performances extrêmes, optimisation d'autres types de requêtes (récursives analytiques,…)
- Fonctionnalités abandonnées : (parfois) ACID, (parfois) contraintes, (parfois) requêtes complexes

New SQL

Certaines applications nécessitent :

- un langage de requêtes riches (SQL)
- une conformité aux propriétés ACID
- mais des performances supérieures à celles des SGBD classiques

Solutions possibles :

- Se débarrasser des goulots d'étranglement classiques des SGBD : verrous, journalisation, gestion des caches
- Bases de données en mémoire vive, avec copie sur disque asynchrone
- Une gestion de concurrence sans verrou (MVCC)
- Une architecture distribuée sans partage d'information (shared nothing) et avec équilibrage de charge transparent

Grands types de SGBD

NoSQL = « not only SQL »

Relationnels (SGBDR) : tables, requêtes complexes (SQL), fonctionnalités riches

XML : arbres, requêtes complexes (XQuery), fonctionnalités similaires aux SGBDR

Graphes/Triplets : données graphes, requêtes complexes exprimant des parcours de graphe

Objets : modèle de données très complexe, inspiré de la POO

Documents : les données sont un ensemble de "documents" (objets) hétérogènes, identifiés par une clef, chacun avec sa propre structure . Les languages de requêtes interrogent les meta-données des documents

Clef-Valeur : modèle de données très basique, accent mis sur la performance

Orientés Colonnes : modèle de données entre celui des clef-valeur et des SGBDR ; accent mis sur le parcours ou l'agrégation efficace de colonnes

Grands types de SGBD

Relationnels (SGBDR) : tables, requêtes complexes (SQL), fonctionnalités riches

XML : arbres, requêtes complexes (XQuery), fonctionnalités similaires aux SGBDR

Graphes/Triplets : données graphes, requêtes complexes exprimant des parcours de graphe

Objets : modèle de données très complexe, inspiré de la POO

NoSQL = « not only SQL »

Documents : les données sont un ensemble de “documents” (objets) hétérogènes, identifiés par une clef, chacun avec sa propre structure . Les languages de requêtes interrogent les meta-données des documents

Clef-Valeur : modèle de données très basique, accent mis sur la performance

Orientés Colonnes : modèle de données entre celui des clef-valeur et des SGBDR ; accent mis sur le parcours ou l’agrégation efficace de colonnes

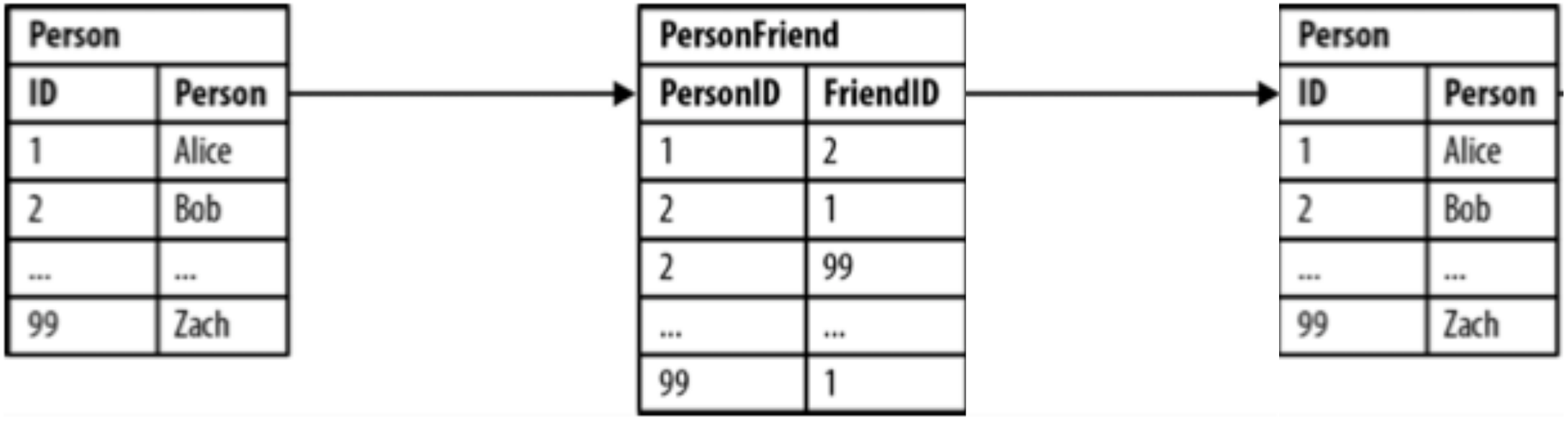
Graphes : de plus en plus au coeur des applications modernes des données

Limites des BD relationnelles et de SQL pour les requêtes de graphe

- Les requêtes de graphe nécessitent des **parcours arbitrairement profonds**
  - Applications naturelles pour certaines données : graphes de réseaux de transport, réseaux sociaux, moteurs de recommandations, télécommunication, etc
- SQL'92 ne peux pas exprimer ces requêtes
- Depuis SQL3 (1999 : ISO/IEC 9075) horizon élargi, requêtes hiérarchiques et récursives
  - en fait en pratique cette « nouveauté » trouve vite ses limites…
  - on gagne en expressivité mais requêtes difficiles à optimiser (et à écrire)
    - organisation des données pas adaptée au "parcours"
      - nécessité d'enchaîner plusieurs jointures

BD relationnelles vs BD graphes : exemple

- Qui sont les amis de Bob ? Facile (index sur PersonId)



Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

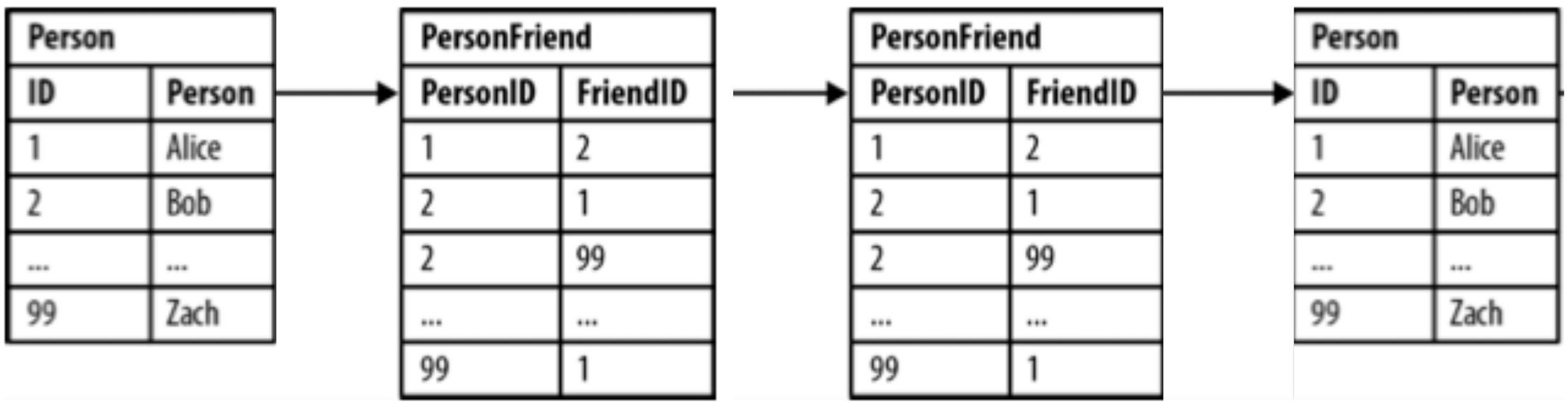
PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend ON
   p1.ID  = PersonFriend.PersonID JOIN
Person p2 ON
   PersonFriend.FriendID = p2.ID
WHERE p1.Person = 'Bob'
```

BD relationnelles vs BD graphes : exemple

- Qui sont les amis des amis de Bob ? Plus problématique.



Person

ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend

PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

PersonFriend

PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

Person

ID	Person
1	Alice
2	Bob
...	...
99	Zach

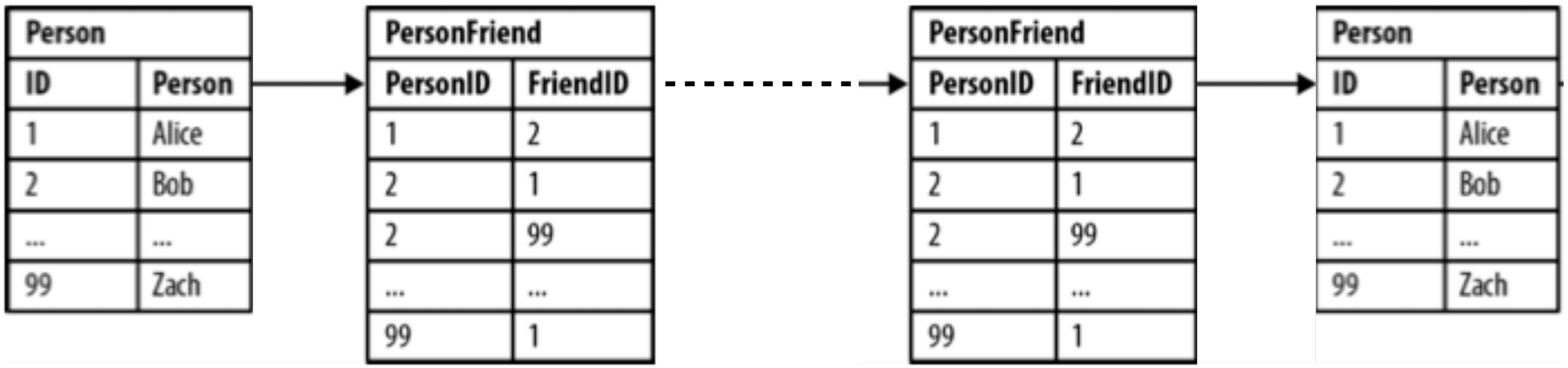
```
SELECT p2.Person AS FRIEND_OF_FRIEND
FROM Person p1 JOIN PersonFriend pf1 ON
   pf1.PersonID = p1.ID JOIN PersonFriend pf2 ON
   pf2.PersonID = pf1.FriendID JOIN Person p2 ON
   pf2.FriendID = p2.ID
WHERE p1.Person = 'Bob' AND p2.ID <> p1.ID
```

Evite de retourner Bob

Détérioration rapide des performances plus on va en profondeur dans le réseau d'amis

BD relationnelles vs BD graphes : exemple

- Avec la recursion SQL peut aller chercher les amis d'amis d'amis…de Bob, sans limite sur le nombre d'amis intermédiaires



Person

ID	Person
1	Alice
2	Bob
…	…
99	Zach

PersonFriend

PersonID	FriendID
1	2
2	1
2	99
…	…
99	1

PersonFriend

PersonID	FriendID
1	2
2	1
2	99
…	…
99	1

Person

ID	Person
1	Alice
2	Bob
…	…
99	Zach

```
WITH RECURSIVE FoF(person1, person2) AS
( SELECT PersonID AS person1, FriendID AS person2  FROM PersonFriend
UNION
   SELECT P.PersonID AS person1, F.person2 AS person 2
   FROM PersonFriend P, FoF F
   WHERE P.FriendID = F.person1 and P.PersonID <> F.person2 )
SELECT P2.nom FROM Person P1, FoF, Person P2
WHERE P1.ID = person1 and P2.ID = person2  AND P1.Person = 'Bob';
```

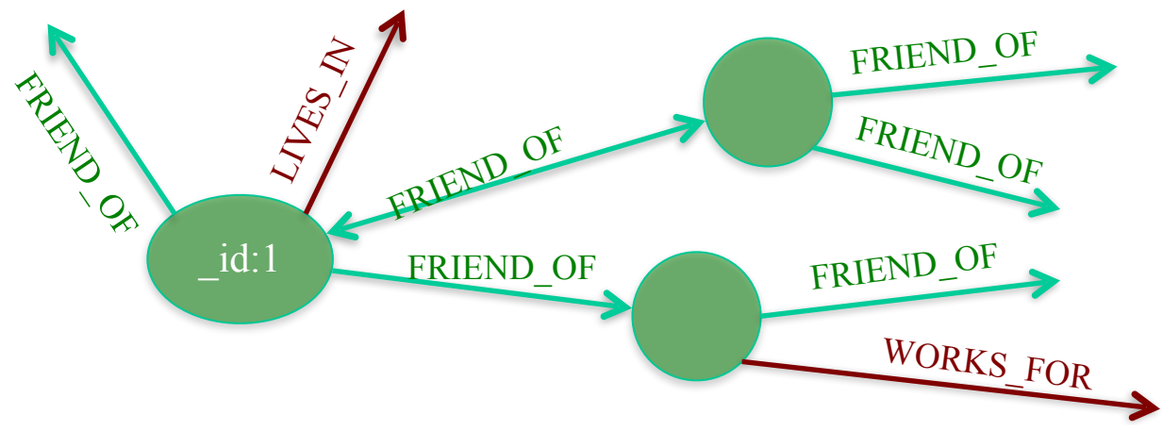
Détérioration rapide des performances plus on va en profondeur dans le réseau d'amis

Le coût des jointures

- Retourner les personnes lié par n amis intermédiaires demande n jointures : temps d'exécution des requêtes vite prohibitif ! (« **join pain** »)
- Secret des bases de données orientées graphe : éviter toutes ces jointures !
  - Grâce à une structuration physique des données adaptée (*graph-native*)
- Et simplifier considérablement l'écriture des requêtes…

BD graphes vs BD relationnelles : coûts

- Coût sur un graphe natif pour les amis à distance 2 :



- En partant d'un noeud, il faut scanner toutes les arrêtes sortantes pour identifier les arrêtes FRIEND_OF
- Ensuite il faut traverser les arrêtes et répéter la recherche pour tous les noeuds atteints
- Si p borne le nombre d'arrêtes sortantes d'un noeud (p<<n<m, avec n le nombre de personnes, m le nombre de paires d'amis), et k borne le nombre d'arrêtes FRIEND_OF sortant d'un noeud (k<<m), alors le coût est $O(p) + O(k * p)$ (temps pour traverser une arrête : constant)
- Des indexes locaux sont normalement utilisés pour accélérer la recherche

BD graphes vs BD relationnelles : coûts

- Coût sur une BD relationnelle pour les amis à distance 2 :

```
SELECT pf2.FriendID AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN PersonFriend pf2 ON
     pf2.PersonID = pf1.FriendID
WHERE pf1.PersonID = 1
```

- Soit m le nombre de paires d'amis, sans indexe on a un coût de $O(m^2)$
- Ok, les indexes réduisent ce coût, ils évitent la recherche linéaire dans une colonne
- Mais quand même. Si le nombre de jointures augmente, le coût devient vite prohibitif et **dépend de la taille entière des tables en jeu**

« Index free adjacency »

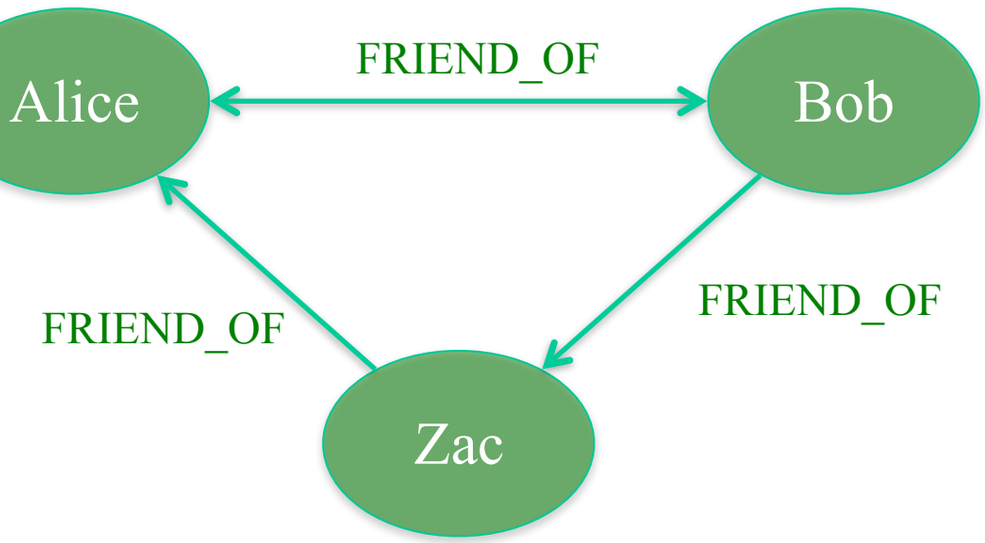
- Dans un moteur de bases de données qui utilise l'adjacence sans index, chaque noeud maintient les références directes de ses noeuds adjacents; chaque noeud agit donc comme un micro-index des autres noeuds à proximité, ce qui est bien plus économique que d'utiliser des indexes globaux.

- i.e. une BD (graphe) G satisfait l'adjacence sans index si l'existence d'une arrête entre deux noeuds v_1 et v_2 dans G peut être testée sur ces noeuds et ne requiert pas d'accès à un index externe global.

- Localement, chaque noeud peut gérer un index spécifique pour accéder à ses noeuds sortants.

BD graphes vs BD relationnelles : exemple

- Modélisation de « ami d'ami » dans une BD graphe



- Les relations dans un graphe forment naturellement des chemins. Les interroger c'est - réellement - traverser le graphe, i.e., suivre ces chemins.
- Dans les BD graphes les opérations basées sur les chemins sont très efficaces (d'autres opérations peuvent être plus compliquées)

BD graphes vs BD relationnelles : expériences pratiques

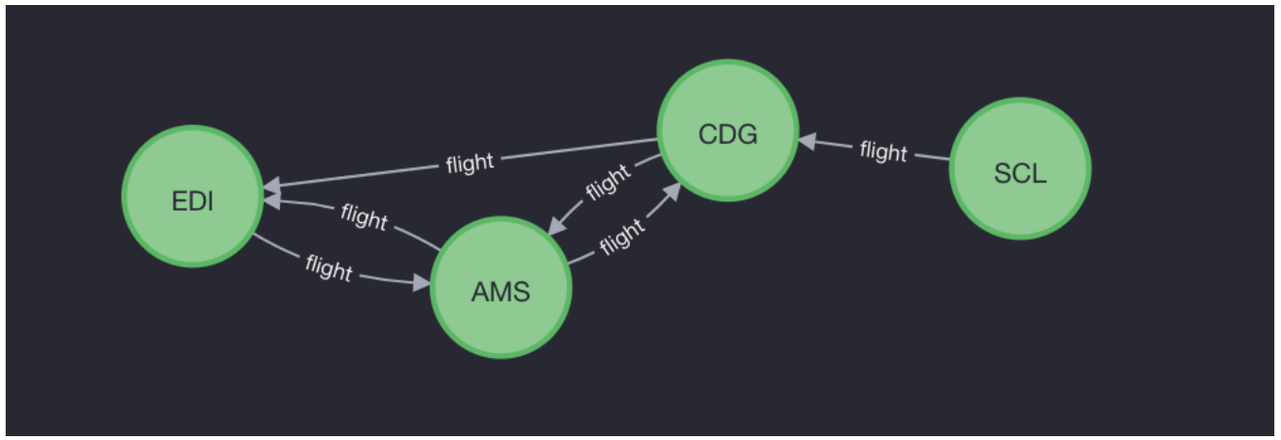
- Résultats expérimentaux : trouver les amis d'amis dans un réseau social, jusqu'à une profondeur maximale de 5, pour un réseau de 1 000 000 personnes (à peu près 50 amis / personne)
  - Etant donné deux personnes choisies au hasard, est-ce qu'il y a un chemin qui les connecte et qui est long d'au plus 5 relations ?

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Tiré de *Neo4j in Action. Jonas Partner, Aleksa Vukotic, and Nicki Watt. MEAP. 2012*

Les bases de données orientées graphes

- Une BD graphe utilise la structure de graphe, les noeuds, les arêtes et les propriétés pour représenter et stocker les données
- Un système de gestion de bases de données orienté graphe offre des méthodes Create, Read, Update et Delete (CRUD) pour accéder et manipuler les données
- Les BD graphe peuvent être utilisées aussi bien dans un cadre analytique OLAP (Online Analytical Processing) que transactionnel OLTP (Online Transactional Processing)
- Les systèmes adaptés à l'OLTP (e.g., neo4j) sont généralement optimisés pour être efficaces sur le plan transactionnel et pour garantir les propriétés ACID

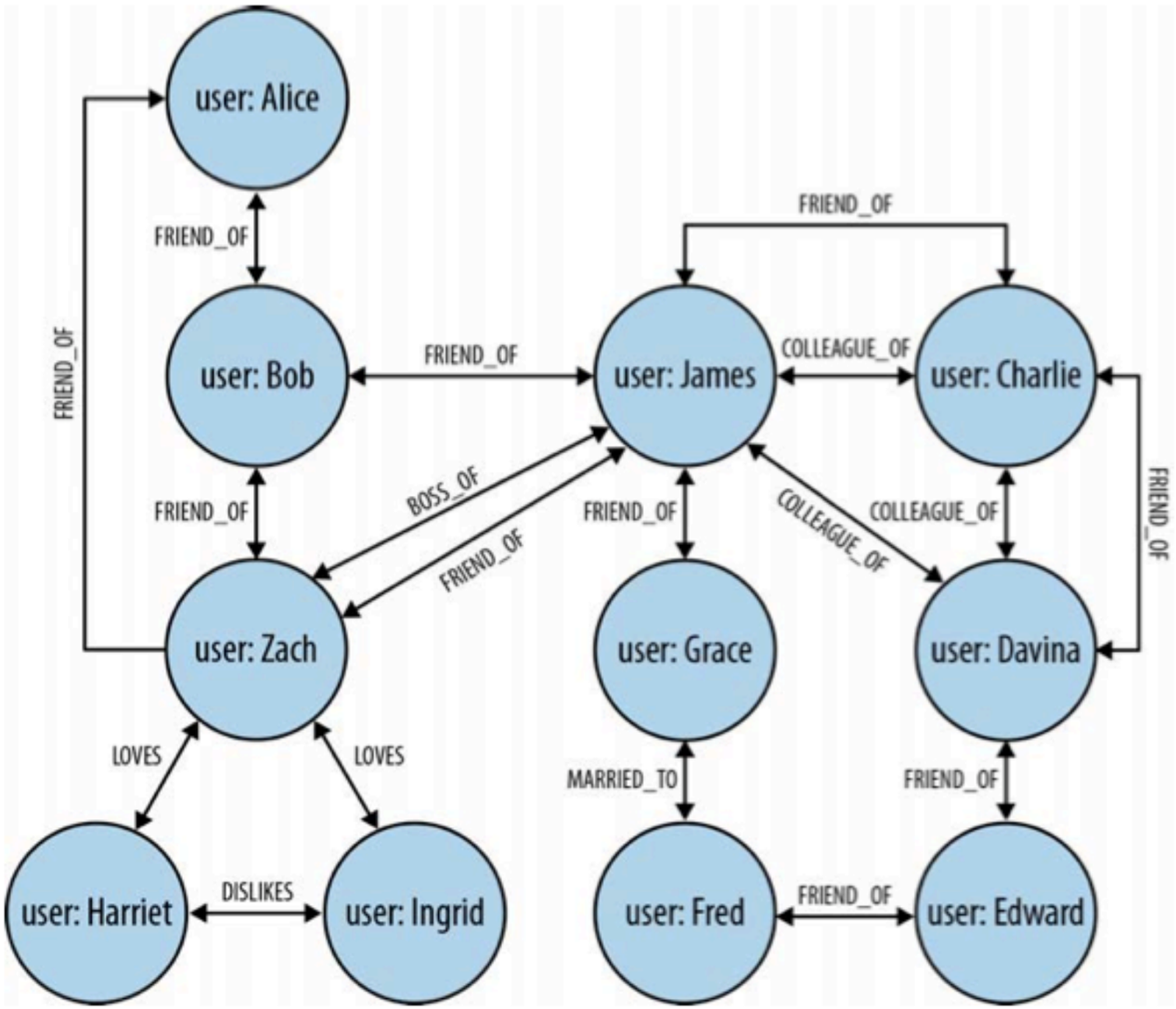


Les bases de données orientées graphes

- Les BD graphe sont conçues sans schéma :
  - adapté à la dynamique du big data : les données peuvent être accumulées progressivement sans imposition à priori d'un schéma rigide prédéfini
  - flexibilité pour des informations différentes avec des propriétés différentes, à chaque niveau de granularité
  - très adapté au traitement des données éparses
    - Attention : ne signifie pas que les aspects intentionnels ne peuvent pas être représentés ! (e.g., contraintes de clef et d'unicité, extraction de schéma a posteriori)
- Les BD graphes peuvent être interrogées via des langages de requête puissants (et standardisés) : la performance vient essentiellement de l'évitement des jointures classiques (performances à relativiser quand même en fonction des requêtes)

Flexibilité des BD graphes

Il est très simple d'incorporer de nouvelles informations dynamiquement



Graph Database Management Systems

- Un Système de gestion de bases de données graphes (SGBDR, GDMS en anglais) est un système gérant des BD graphes. Par exemple :

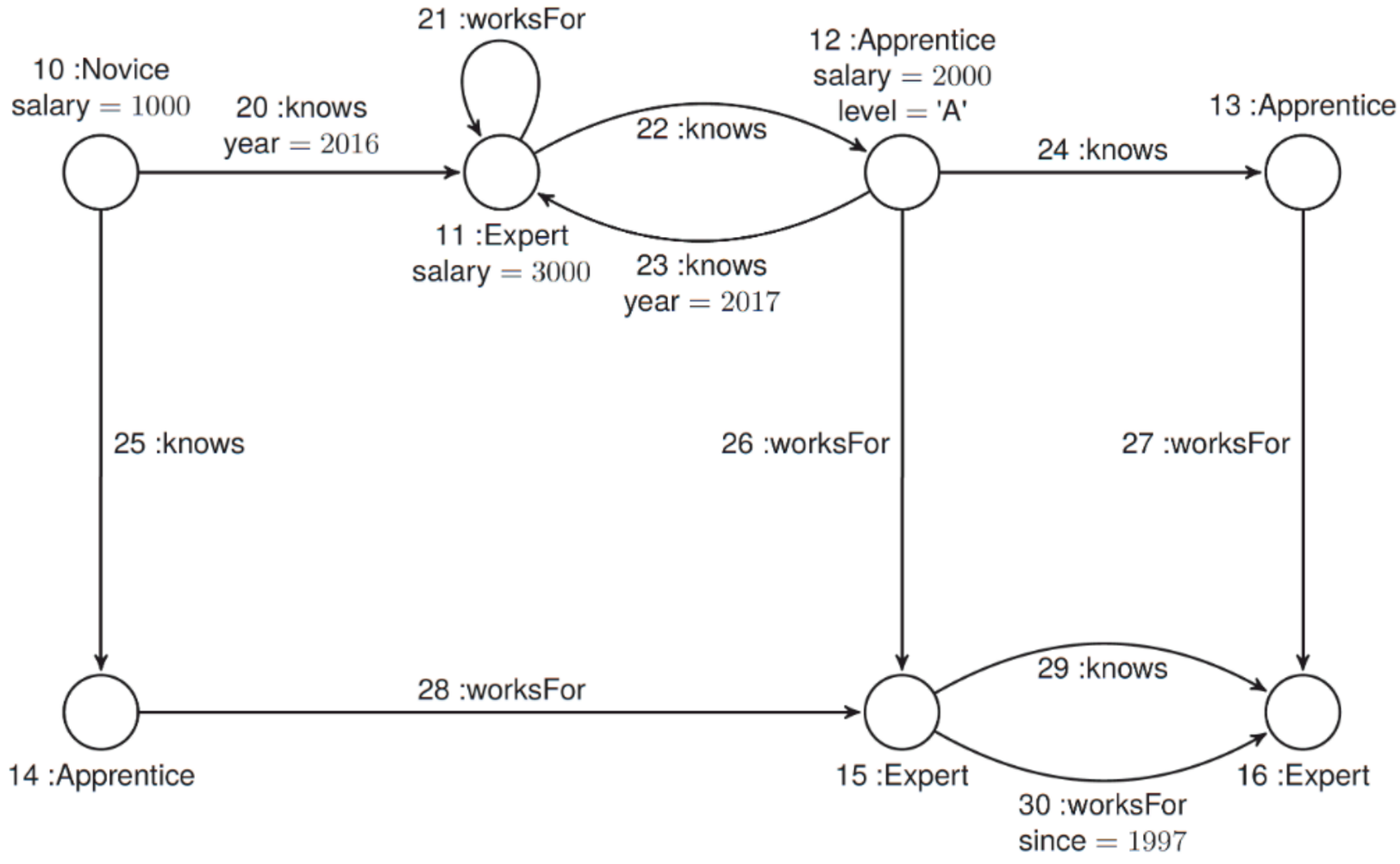
Neo4j
InfiniteGraph
ArangoDB
*Sparksee
sones
OrientDB
TITAN
MEMGRAPH
TigerGraph
Dgraph
Amazon Neptune

Types de BD graphes

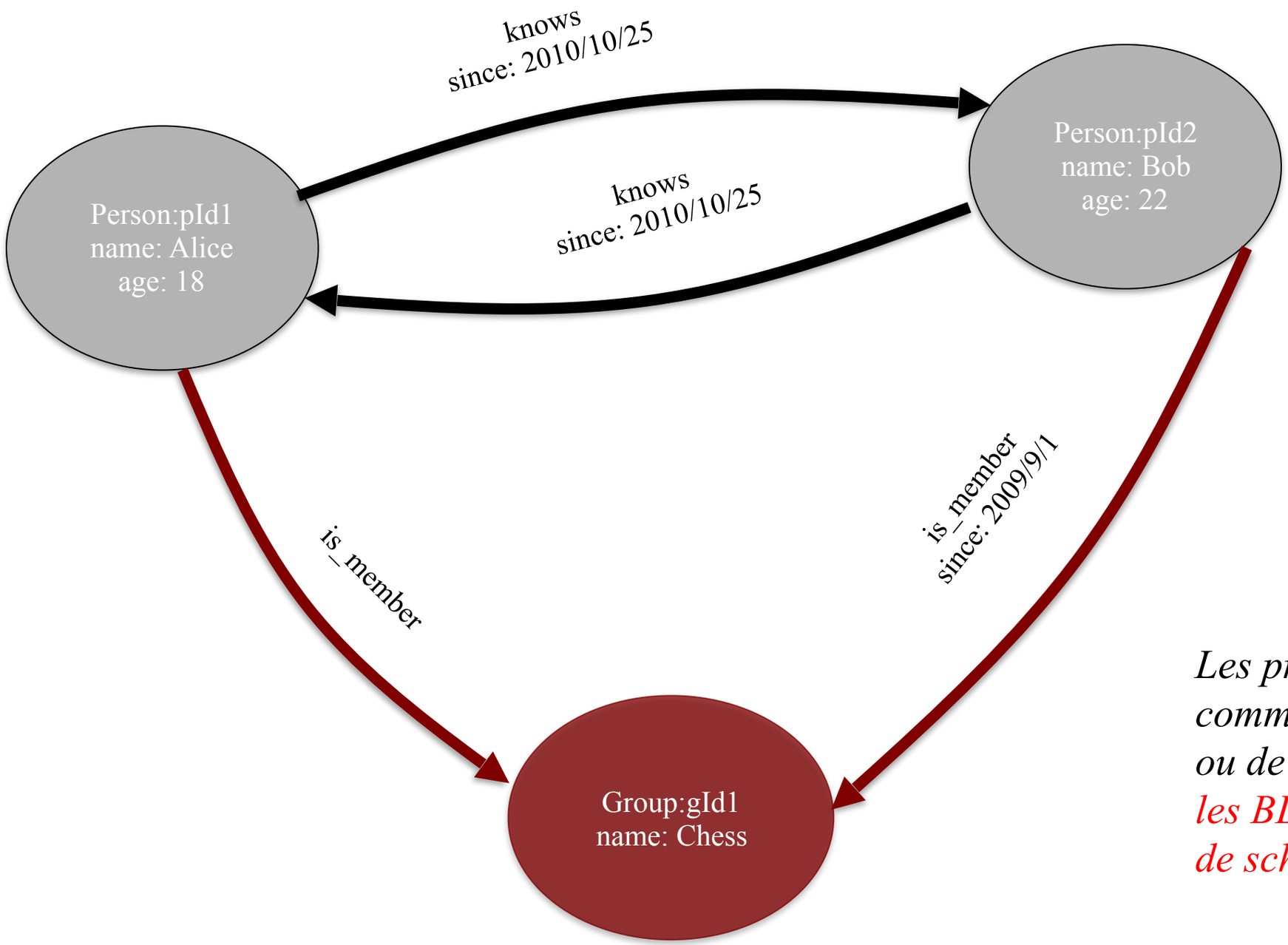
- Il y a différents types de modèles de données graphes basés sur la définition que nous venons voir. Le modele qui s'est dernièrement imposé :
  - Les graphes de propriétés

Les graphes de propriété

- Un graphe de propriété est un multigraphe orienté étiqueté *G = (V, E)* où chaque noeud *v* ∈ *V* et chaque arrête e ∈ *E* peut être associé à un ensemble de paires <*key, value*>, appelé propriétés.
- Chaque arrête représente un lien entre noeuds et est associée à un label (ou étiquette), qui est le nom de la relation (intentionnelle) dont le lien est une instance.



Les graphes de propriété



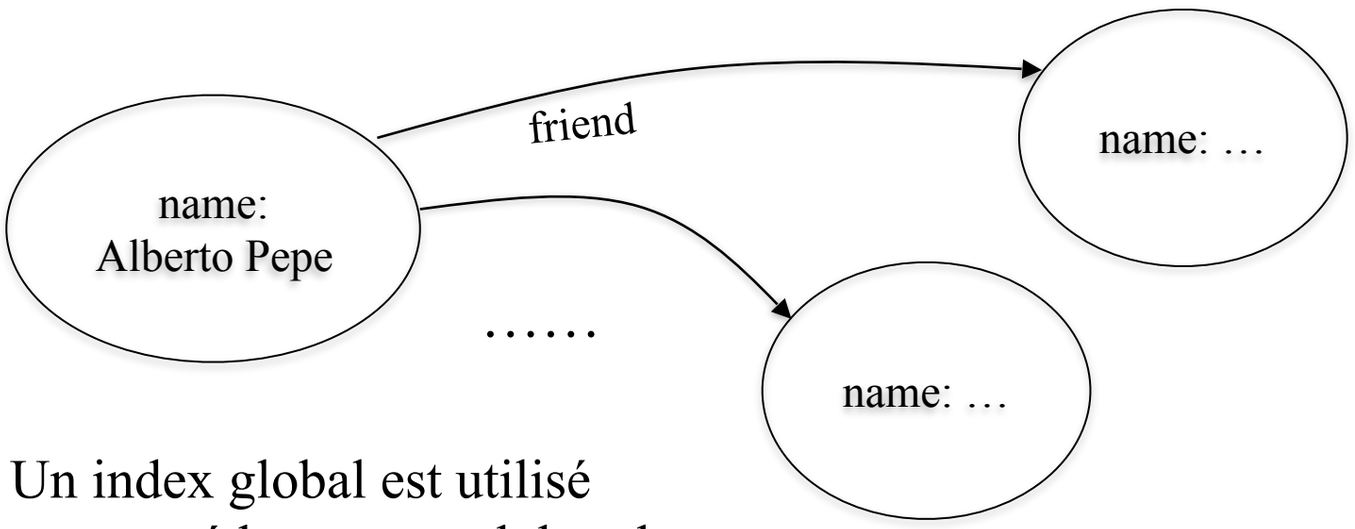
Les propriétés se comportent comme des attributs d'entités ou de relations, mais dans les BD graphes il n'y a pas de schéma rigide a-priori

Interrogation de graphes de propriété

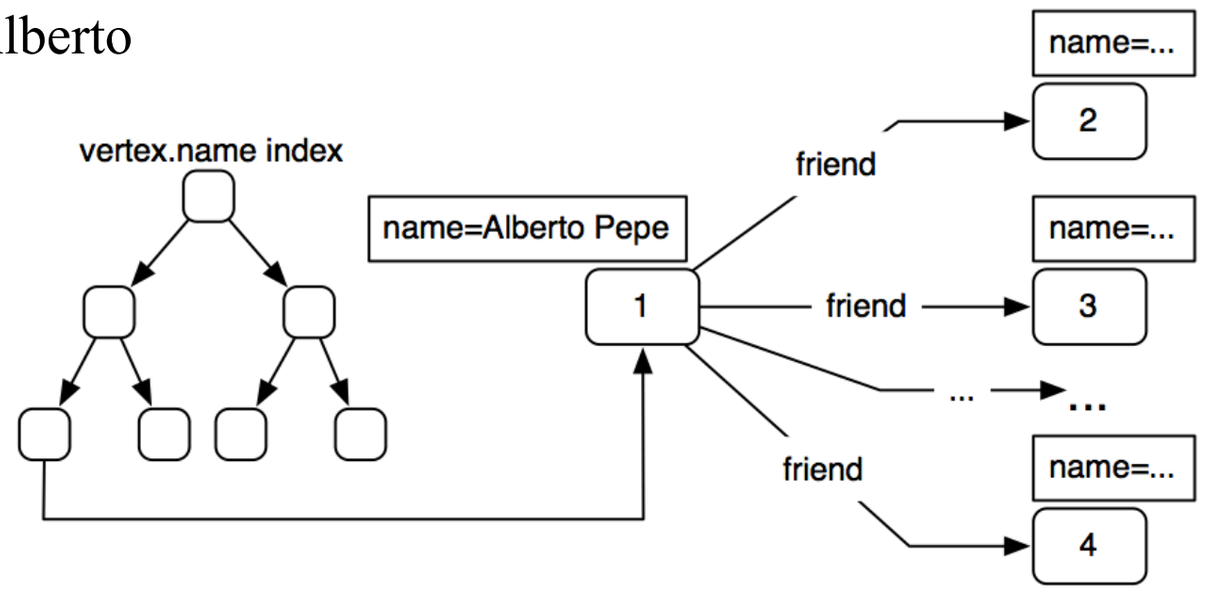
- Les langages de requêtes de base pour les BD graphes n'explorent que leur topologie.
  - Languages théoriques : Regular Path Queries (RPQs)
- Cependant, dans les graphes de propriété on veut aussi accéder aux données stockées sur les noeuds et sur les arrêtes (i.e., les propriétés).
- Les RPQs ne permettent pas ça, mais des langages adaptés (comme Cypher, de Neo4J) permettent l'extraction de propriété
- L'exécution de requêtes accédant aux propriétés, cependant, en plus d'exploiter l'adjacence, s'appuie sur les mécanismes relationnels:
  – Dans le modèle de graphe de propriété, il est commun pour les propriétés des noeuds (et parfois des arrêtes) d'être globalement indexées via des structures d'arbre analogues à celles utilisées dans le cas des BD relationnelles.

Interrogation de graphes de propriété

Les noms des amis d'Alberto Pepe

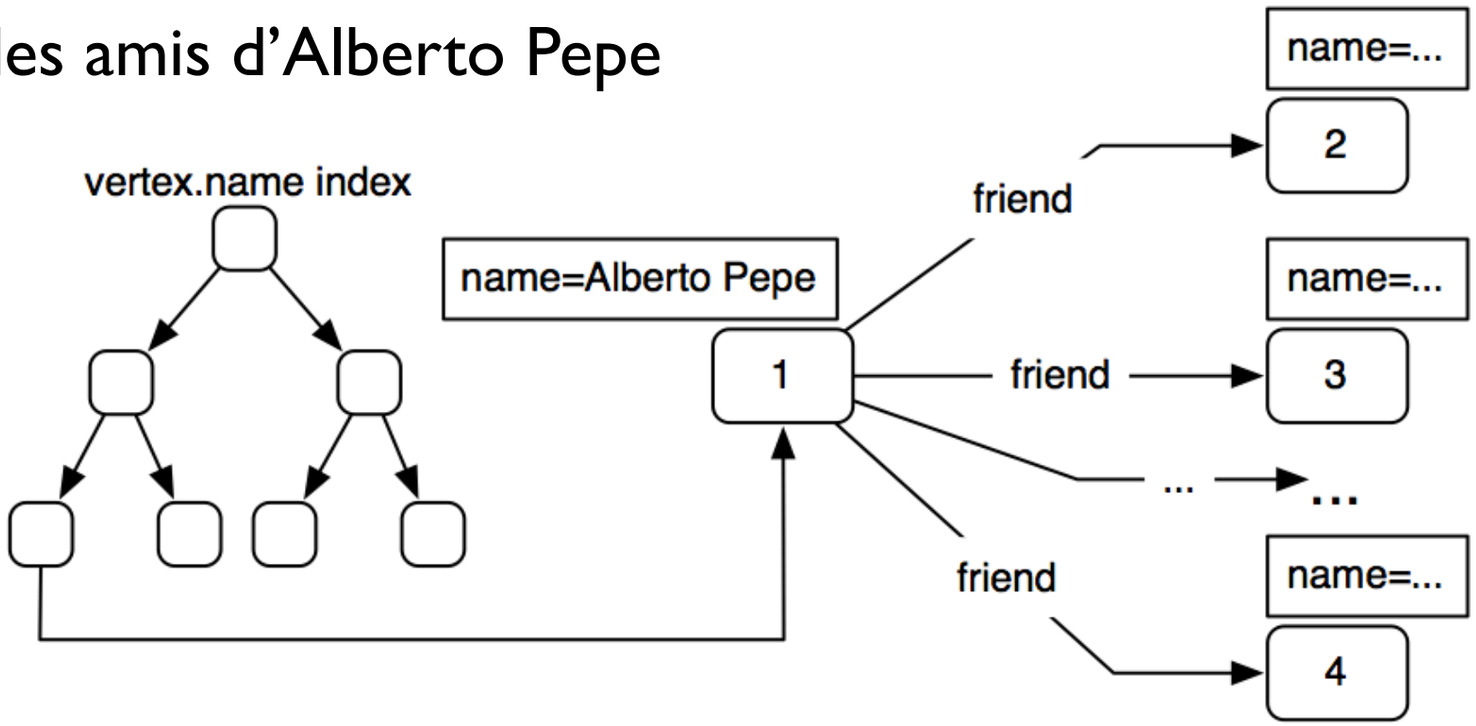


Un index global est utilisé pour accéder au noeud dont la propriété de nom est 'Alberto Pepe'



Interrogation de graphes de propriété

Les noms des amis d'Alberto Pepe



1. Interrogation de l'index vertex.name pour trouver tous les noeuds avec le nom "Alberto Pepe" [O(log2n)] (où n est le nombre de noeuds avec ce nom de propriété)
2. Etant donné le noeud retourné, chercher les k arrêtes friend sortant de ce noeud. [O(k + x)]] (où k est le nombre d'amis et x est le nombre des autres arrêtes sortantes)
   - Remarque : l'avantage par rapport à une representation relationnelle est ici
3. Etant données les k arrêtes friend retournées, chercher les k noeuds à la tête de ces arrêtes [O(k)]
4. Etant donné les k noeuds, obtenir les k noms de propriété de ces noeuds. [O(k*y)] (où y est le nombre de propriétés sur chaque noeud)

Stockage graphe non natif

- Toutes les technologies de BD graphe n'utilisent pas de stockage graphe natif. Le graphe de données est parfois sérialisé sous forme d'une BD relationnelle, orienté objet ou autre.

- Ces GDBMS n'utilisent pas l'adjacence sans index, mais les mécanismes relationnels classiques…

- Exemple : SQL/PGQ permet de faire du graph pattern matching avec SQL (pas encore implémenté à l'échelle industrielle mais ajouté au standard de SQL en 2023)
  - Des avancées récentes sur les algorithmes de jointure (*optimal join algorithms*), permettent de rivaliser en performance

- GDBMS mais seulement du point de vue utilisateur…