NoSQL-Datenbanken

Graphdatenbanken

Johannes Zschache
Sommersemester 2019

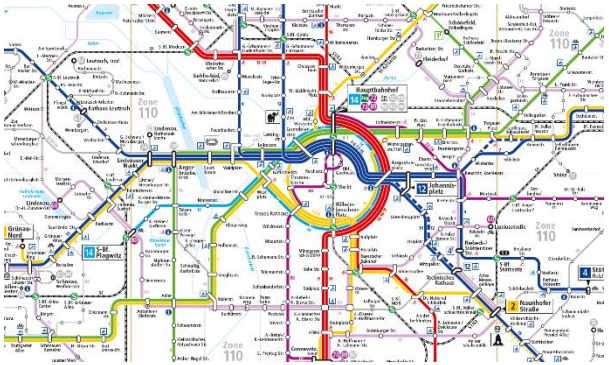
Abteilung Datenbanken, Universität Leipzig
http://dbs.uni-leipzig.de

Inhaltsverzeichnis: Graphdatenbanken

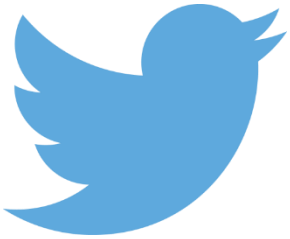
Einführung

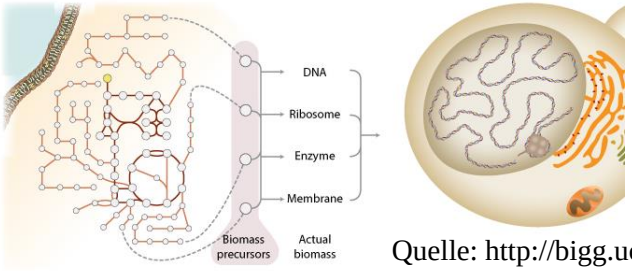
Verstärktes Aufkommen von Graphdaten

- Technologische Netzwerke (Internet, Verkehrsnetze, Stromnetz, …)
- Soziale Netzwerke (Freundschaften, Kommunikation, Krankheiten, …)
- Biologische Netzwerke (Protein-Protein-Interaktion, Jäger-Beute, …)
- Informationsnetzwerke (WWW, Zitiernetzwerke, …)



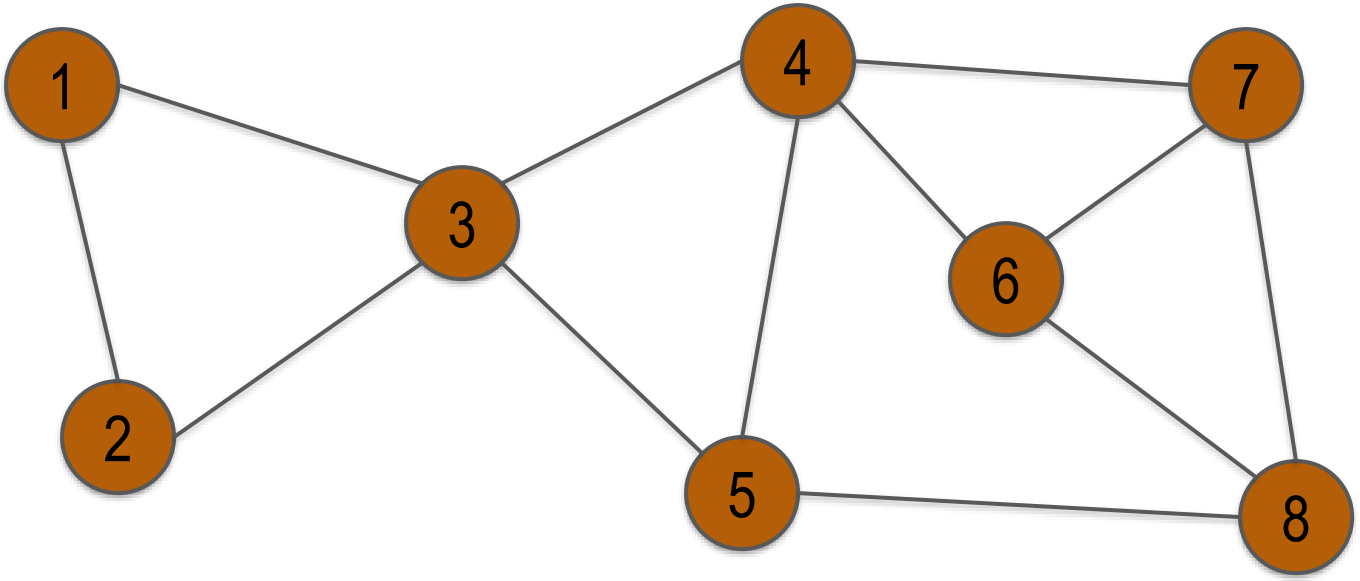
Quelle: http://www.lvb.de





Quelle: http://bigg.ucsd.edu

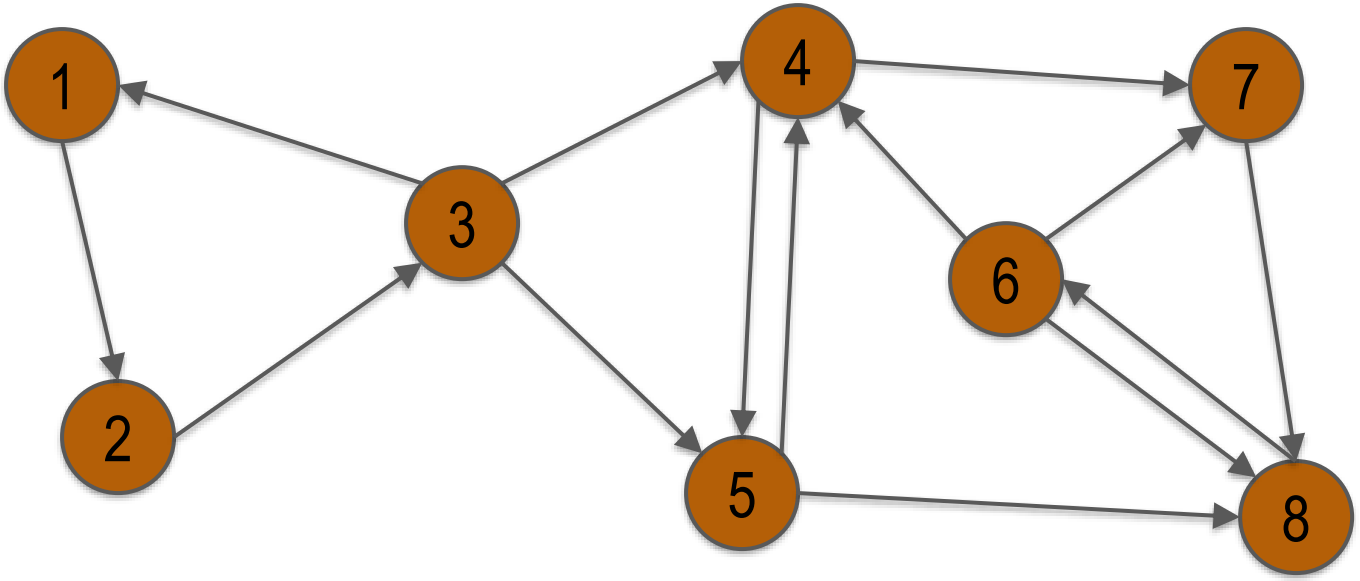
Datenmodell: Graph



$$\boldsymbol{G}raph = (\boldsymbol{V}ertices, \boldsymbol{E}dges)$$

- $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$
- $E = \{\{1,2\}, \{1,3\}, \{2,3\}, \{3,4\}, \{3,5\}, \{4,5\},$
  $\{4,6\}, \{4,7\}, \{5,8\}, \{6,8\}, \{6,7\}, \{7,8\}\}$

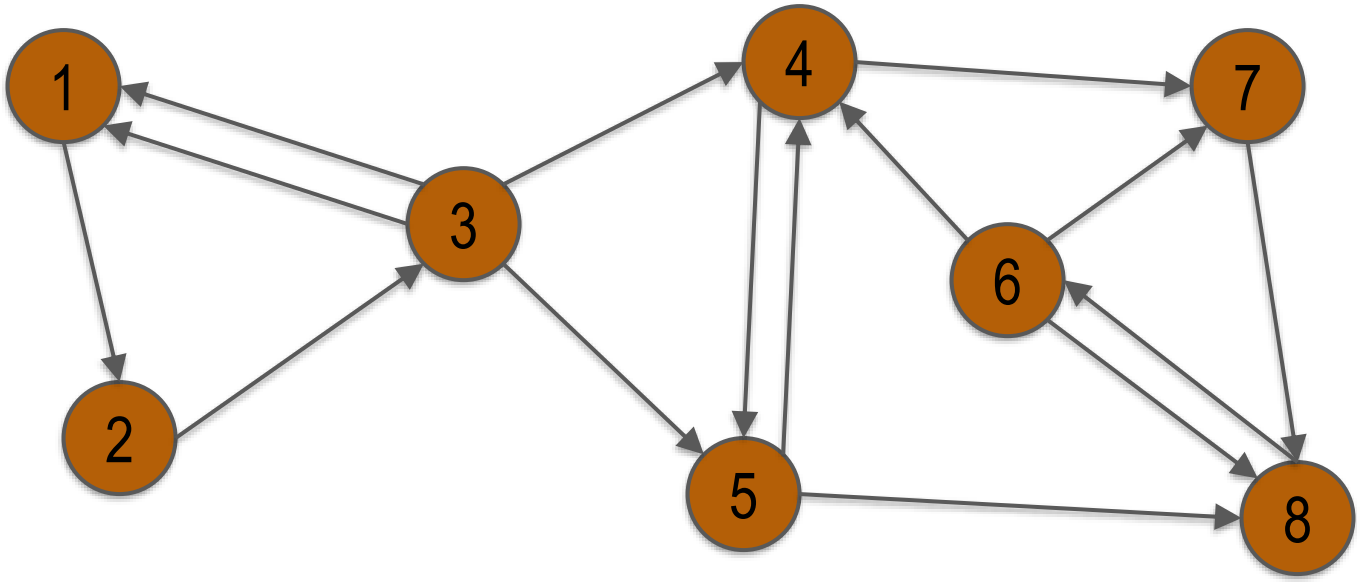
Gerichteter Graph



$$\boldsymbol{G}raph = (\boldsymbol{V}ertices, \boldsymbol{E}dges)$$

- $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$
- E = {(1,2), (2,3), (3,1), (3,4), (3,5), (4,5), (5,4), (4,7), (5,8), (6,4), (6,7), (6,8), (8,6), (7,8)}

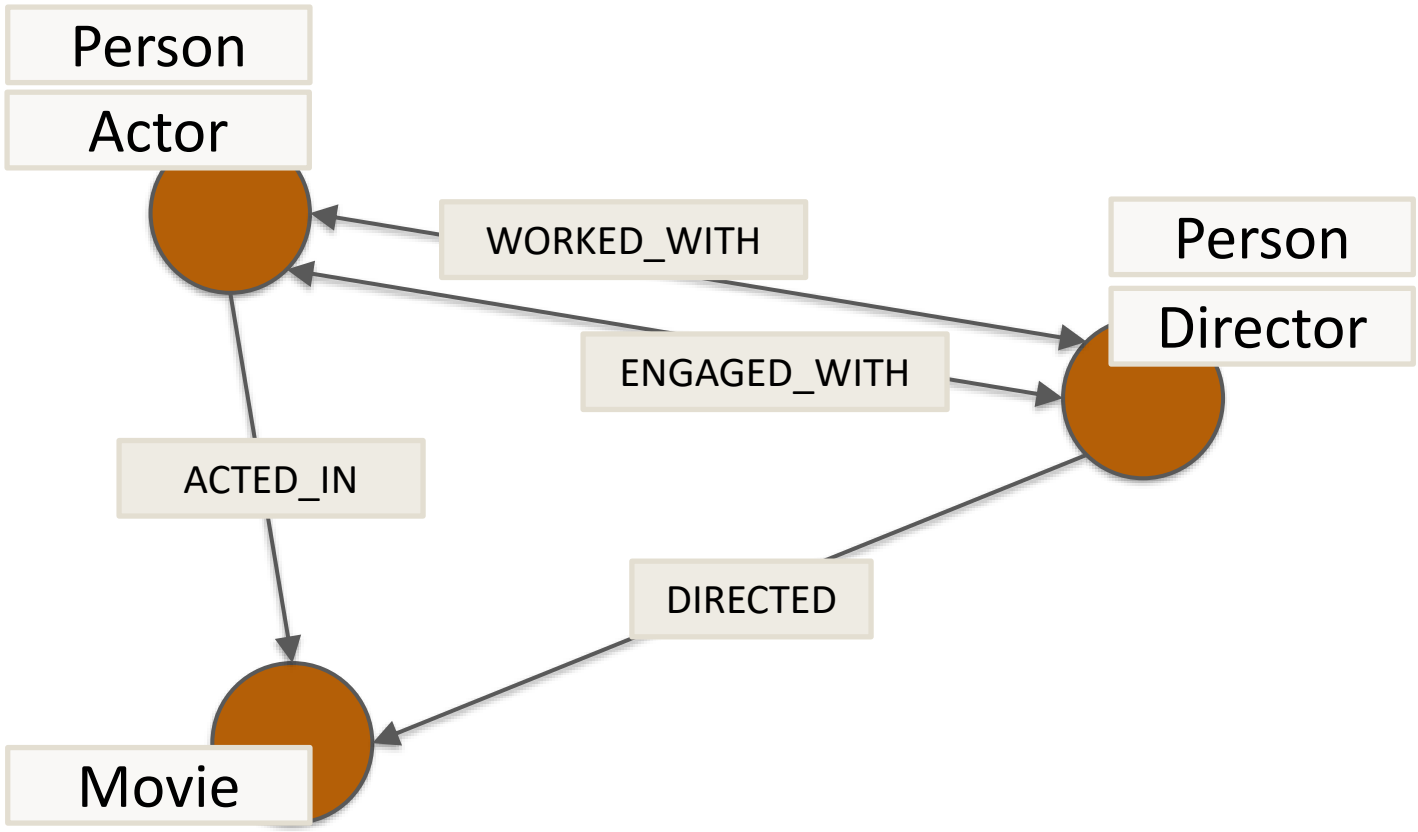
Gerichteter Multigraph



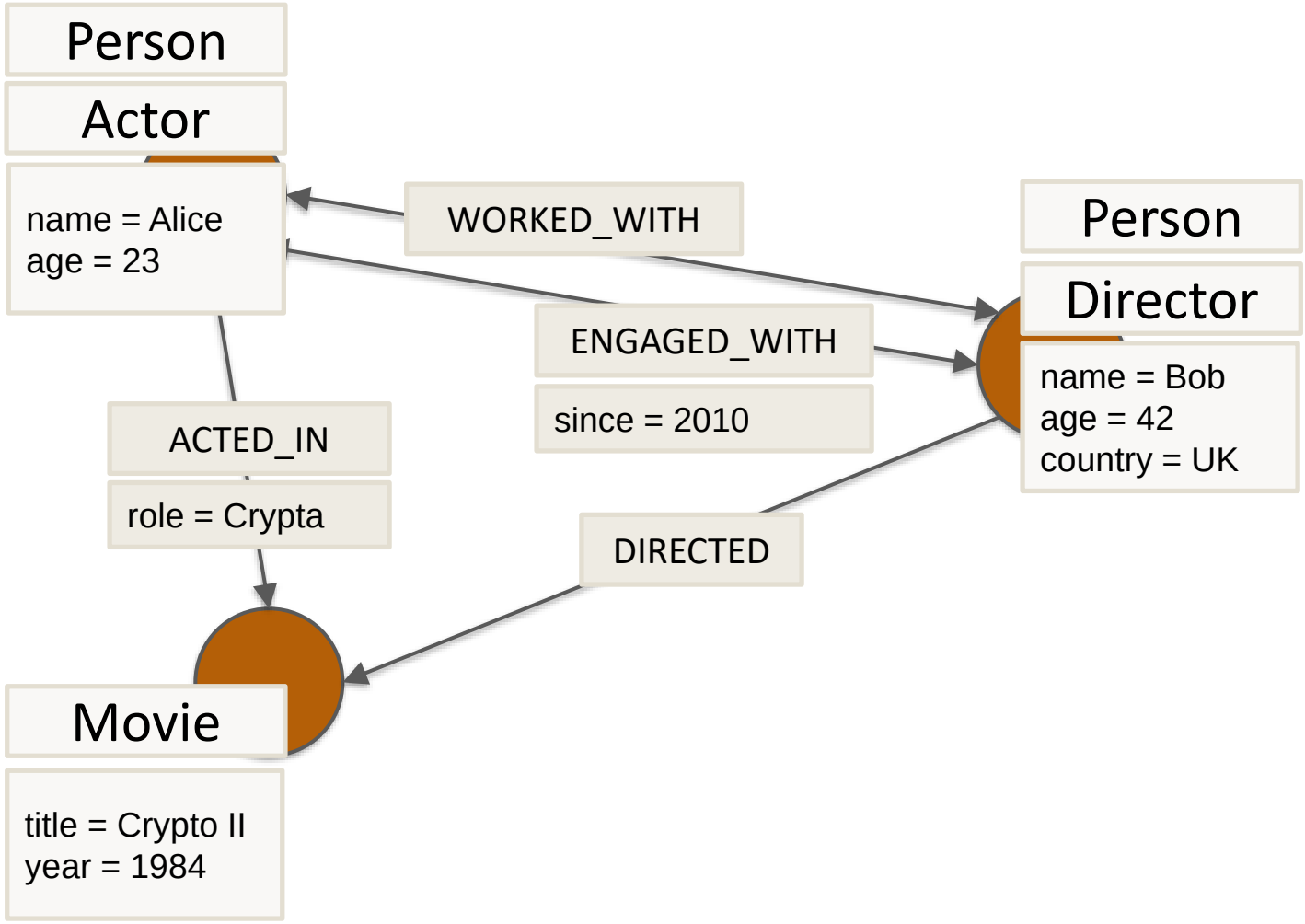
$$\boldsymbol{G}raph = (\boldsymbol{V}ertices, \boldsymbol{E}dges)$$

- $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$
- E = ((1,2), (2,3), (3,1),(3,1), (3,4), (3,5), (4,5), (5,4), (4,7), (5,8), (6,4), (6,7), (6,8), (8,6), (7,8))

Gerichteter Multigraph mit Beschriftungen (Labels)

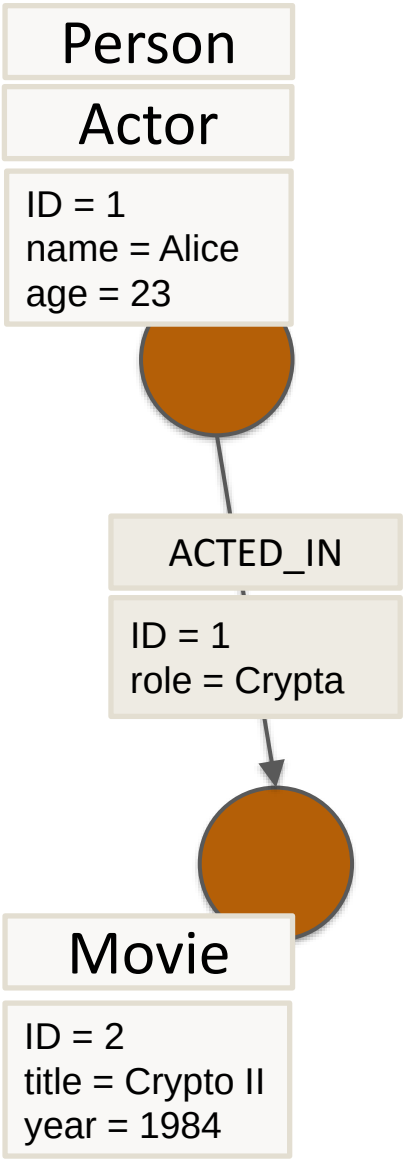


Gerichteter Multigraph mit Beschriftungen und Attributen

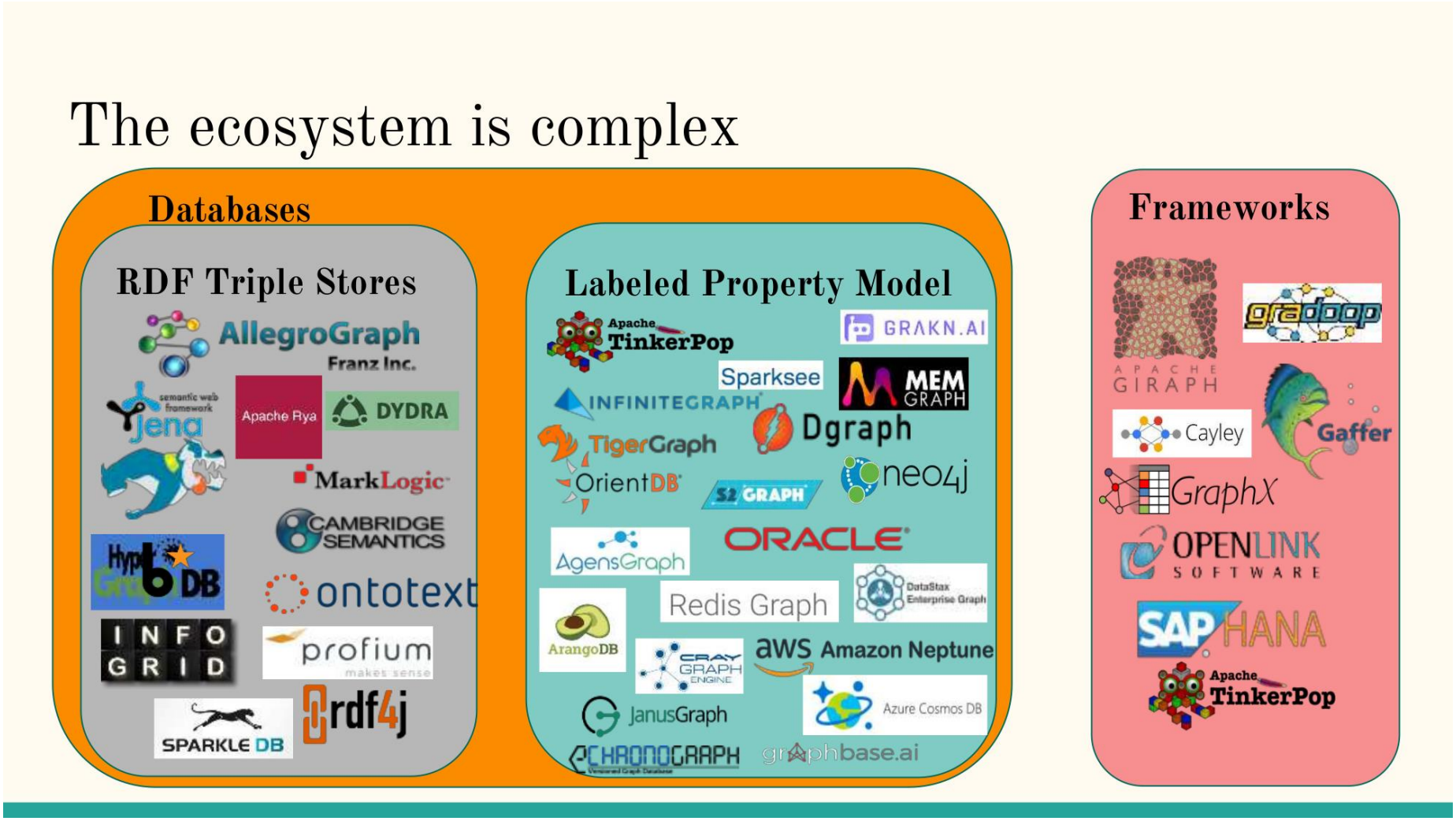


Property Graph Modell

- Gerichteter Multigraph mit Beschriftungen (Labels) und Attributen (Properties)
- Knoten und Kanten haben ID
- Knoten und Kanten besitzen Attribute
  - Annotation mittels Schlüssel-Wert-Paaren
  - Schlüssel: String
  - Wert: Object
- Knoten/Kanten im Allgemeinen **schemafrei**
  - Beliebige Attribute und Datentypen
  - Einschränkung mittels Schema möglich
- Flexibilität
  - Enthält einfachere Graphdatenmodelle
  - Eignung zur Abbildung vieler Netzwerkarten
- Grundlage vieler Graphdatenbanken



Graphdatenmanagement



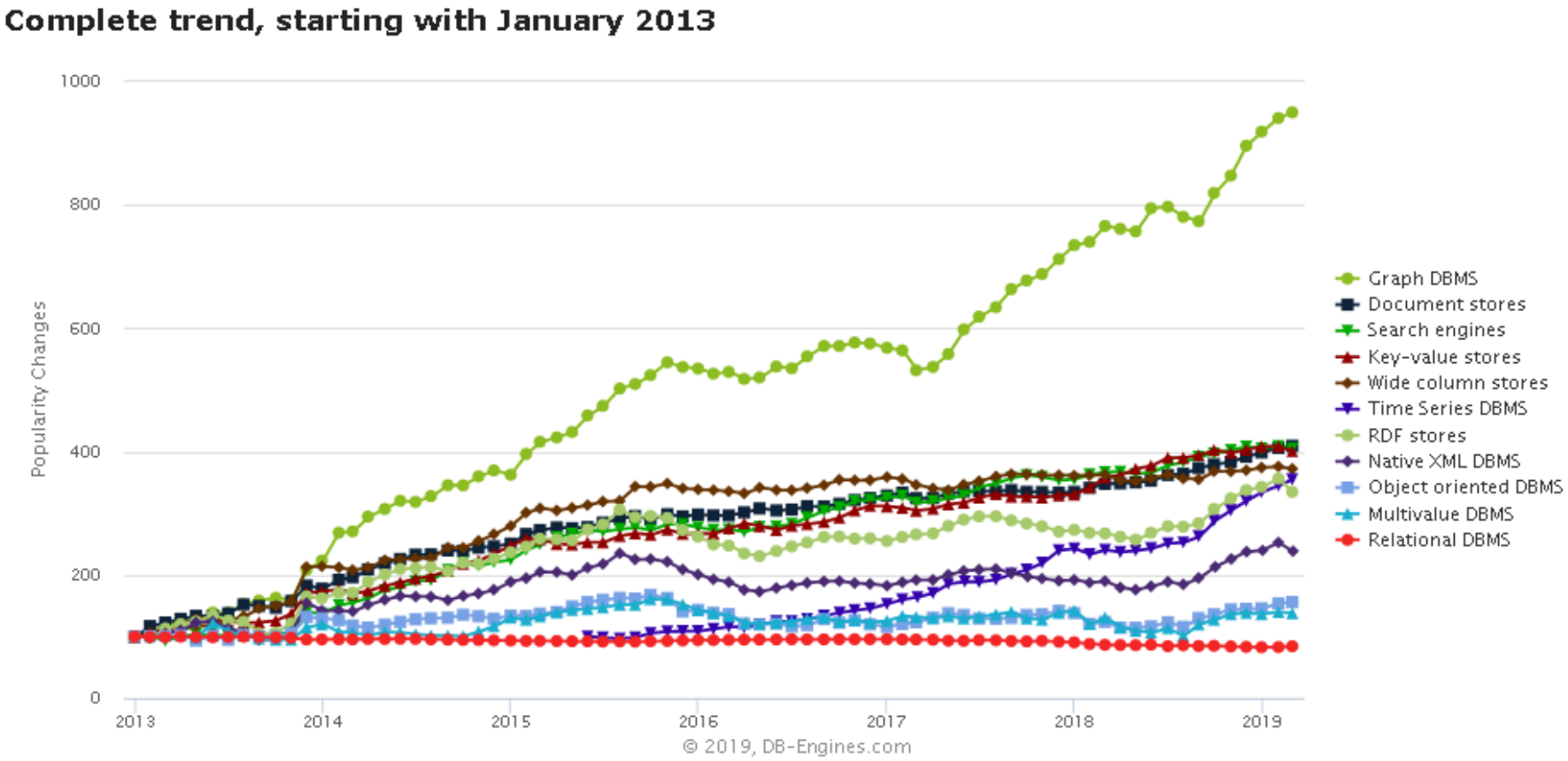
Quelle: https://www.slideshare.net/DaveBechberger/ndc-oslo-2018-a-practical-guide-to-graph-databases-102313908

Graphdatenmanagement: Vergleich

Graphdatenbanken	Graph Processing Framework
Persistenz von und interaktiver Zugriff auf Graphdaten (OLTP)	Verarbeitung umfangreicher Graphdaten
Unmittelbare Ausführung von Anfragen	Batch Processing
Lokaler Bezug zu Knoten-/Kanteninstanzen, z.B. Selektion, Projektion, Traversierung, Mustersuche	Globale Operationen, z.B. Pfadsuche, Partitionierung, Zentralitätsmaße
Unterstützung paralleler Nutzerzugriffe	Keine Unterstützung paralleler Nutzerzugriffe (job-basiert)
Lesender und schreibender Zugriff; Transaktionen (ACID) möglich	Typischerweise nur lesender Zugriff
Eingeschränkte Skalierbarkeit: Vorrangig Replikation; Beschleunigter Zugriff durch Indexierung	Horizontale Skalierbarkeit; Verteilte Berechnungsmodelle, z.B. MapReduce

Graphdatenbanken

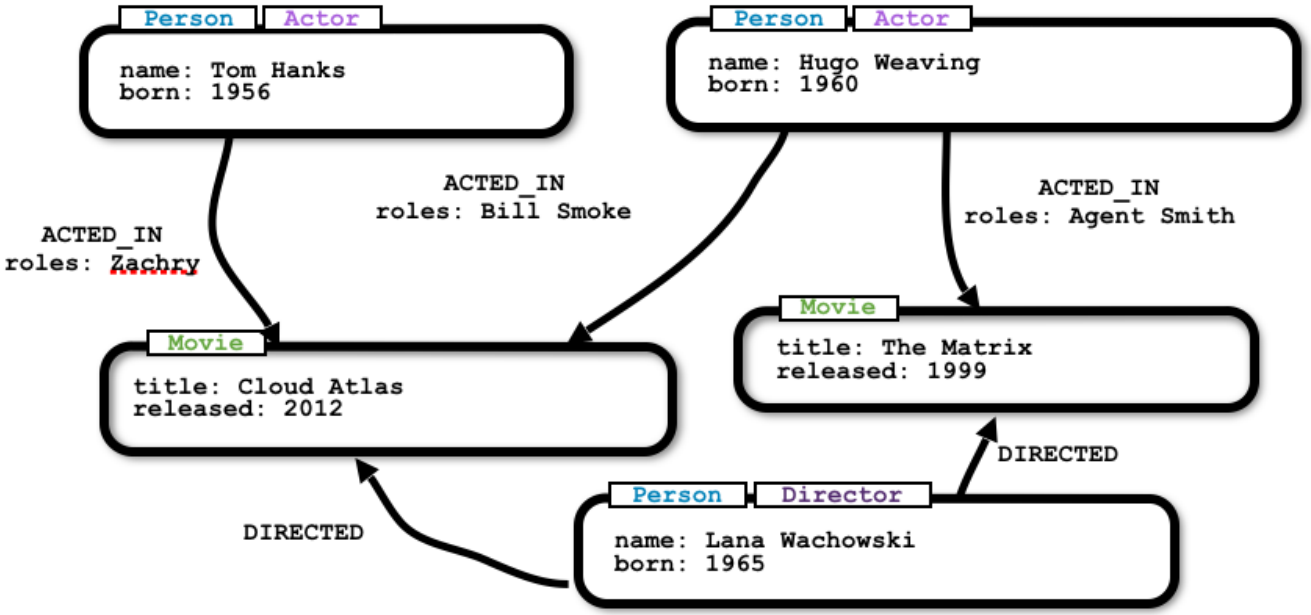
Quelle: http://db-engines.com/en/ranking_categories/



Inhaltsverzeichnis: Graphdatenbanken

GDBMS vs. RDBMS: Modellierung

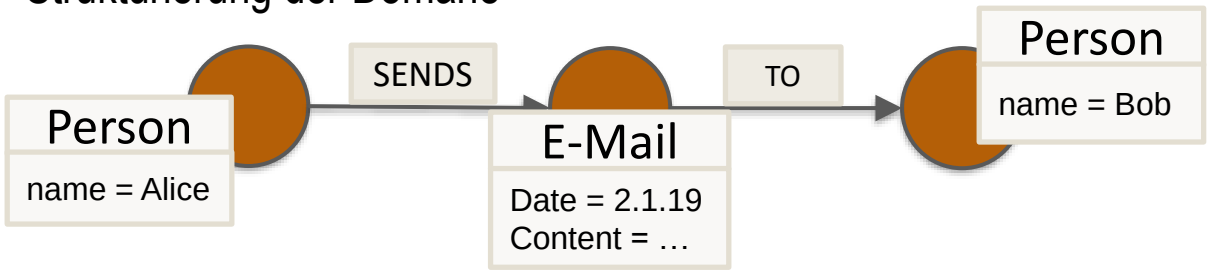
- RDBMS: Entity-Relationship-Modell/UML → Relationenmodell
  - Horizontale/vertikale Partitionierung
  - Normalisierung
  - Assoziationsklassen für m:n-Beziehungen
- GDBMS: Whiteboard-Friendly Modelling



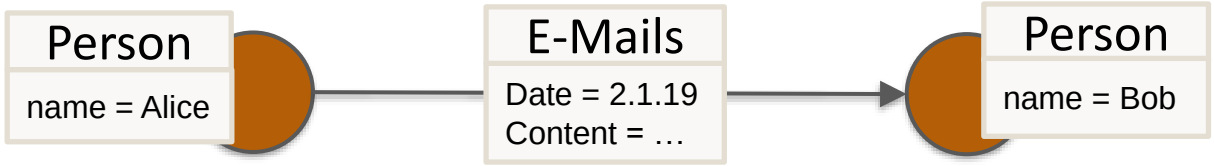
Quelle: https://neo4j.com/developer/guide-data-modeling/#_graph_data_model_whiteboard_friendly

GDBMS vs. RDBMS: Modellierung

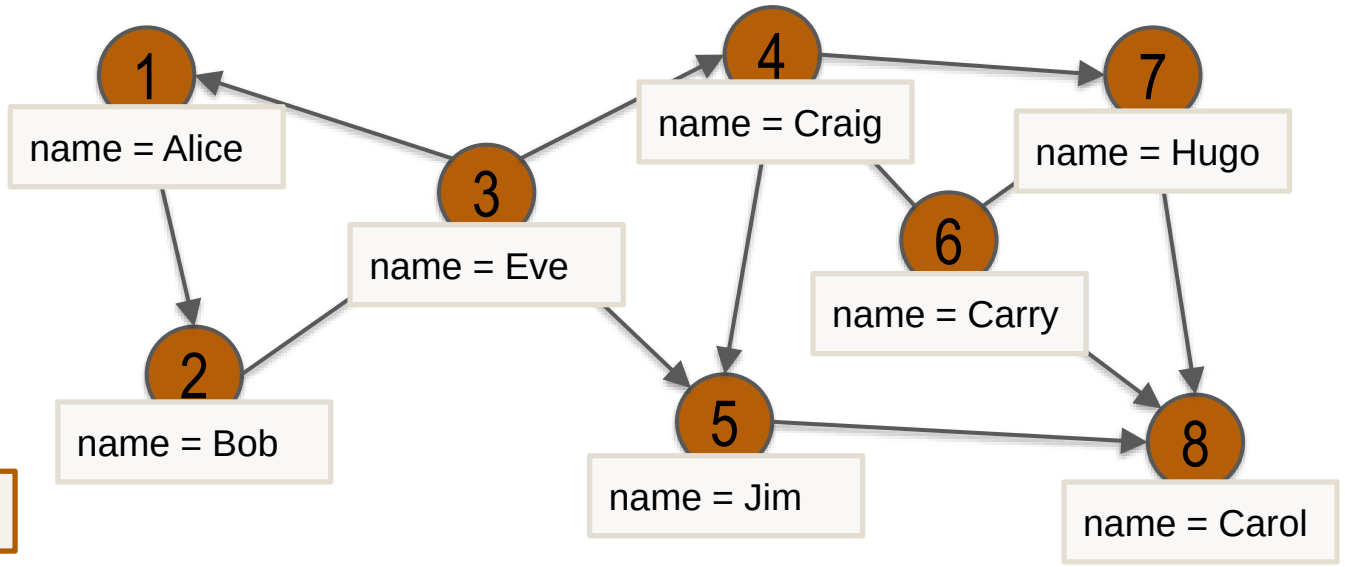
- Schwierigkeit bei Graphen: Identifizierung von Knoten und Kanten
  - Längere Latenzzeiten bei unpassender Modellierung
  - Berücksichtigung der geplanten Anfragen
  - Knoten repräsentieren Entitäten (Objekte, die von Interesse sind)
    - Erkennung der Entitäten über Substantive in natürlicher Sprache
    - Substantive = Label, Eigenschaften = Attribute
  - Kanten repräsentieren Beziehungen zwischen den Entitäten
    - Kommen oft als Verben in natürlicher Sprache vor
    - Strukturierung der Domäne



- Identifizierung nicht immer eindeutig



GDBMS vs. RDBMS: Beziehungen



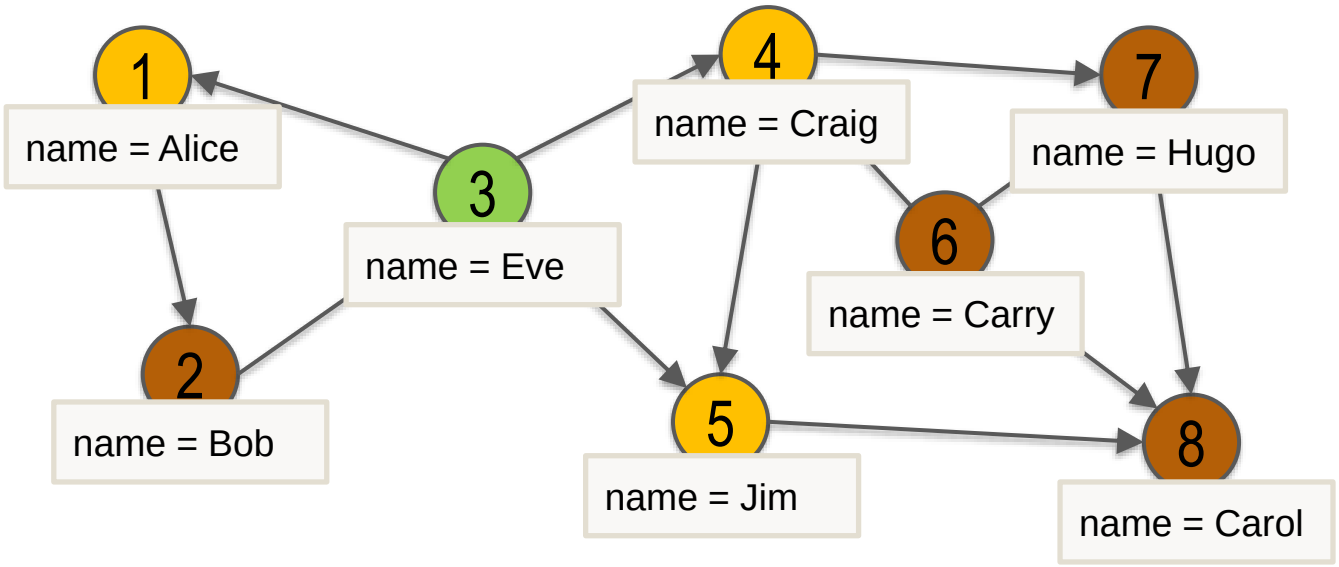
GDBMS

RDBMS

Person	
id	**name**
1	Alice
2	Bob
3	Eve
4	Craig
5	Jim
…	…

PersonFriend	
id1	**id2**
1	2
2	3
3	1
3	4
3	5
…	…

GDBMS vs. RDBMS: Beziehungen



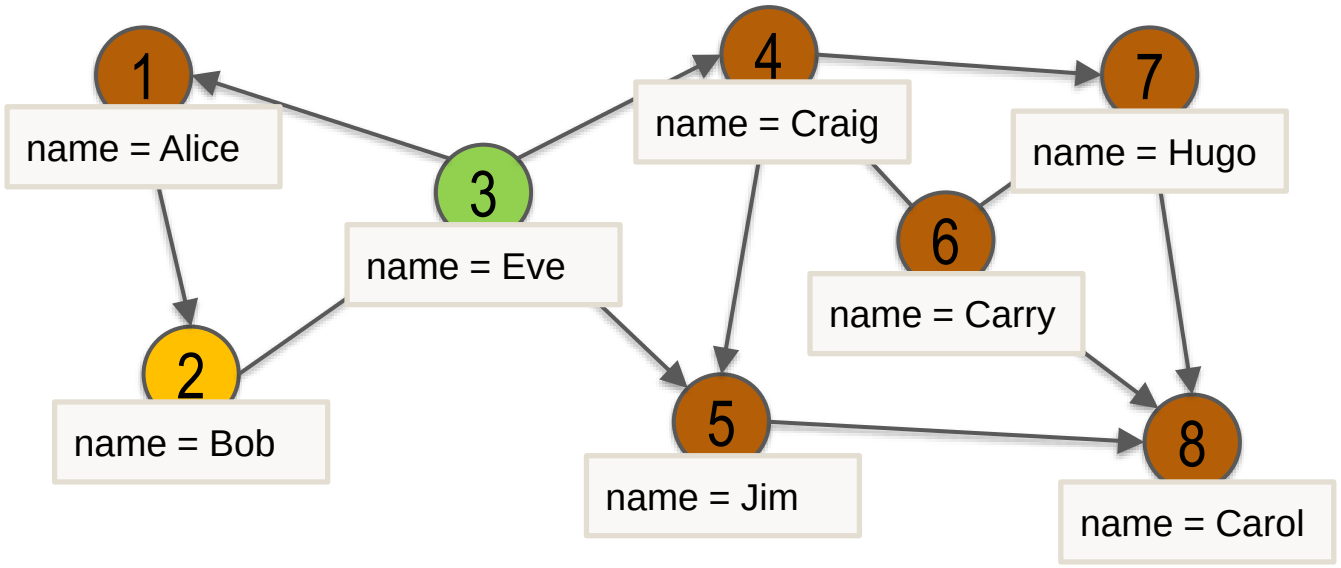
Person	
id	**name**
1	Alice
2	Bob
3	Eve
4	Craig
5	Jim
…	…

PersonFriend	
id1	**id2**
1	2
2	3
3	1
3	4
3	5
…	…

Alle Freunde von Eve?

```
SELECT p1.name
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.id2 = p1.id
JOIN Person p2
  ON PersonFriend.id1 = p2.id
WHERE p2.name = 'Eve'
```

GDBMS vs. RDBMS: Beziehungen



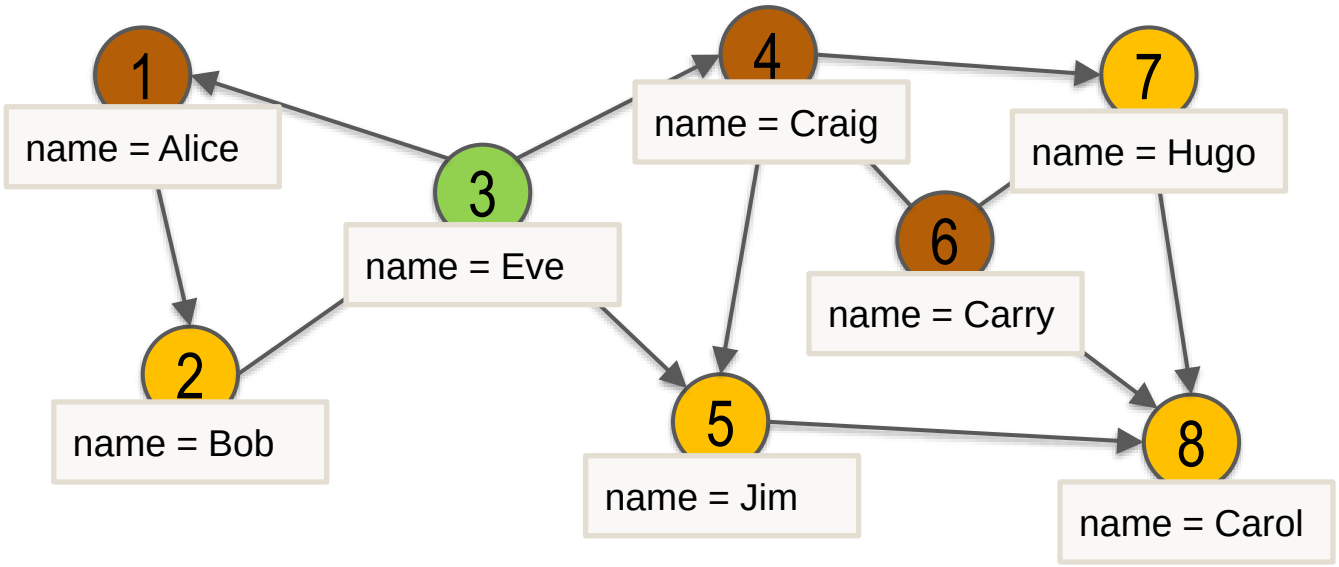
Person	
id	**name**
1	Alice
2	Bob
3	Eve
4	Craig
5	Jim
…	…

PersonFriend	
id1	**id2**
1	2
2	3
3	1
3	4
3	5
…	…

Gegenrichtung:

```
SELECT p1.name
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.id1 = p1.id
JOIN Person p2
  ON PersonFriend.id2 = p2.id
WHERE p2.name = 'Eve'
```

GDBMS vs. RDBMS: Beziehungen



Person	
id	**name**
1	Alice
2	Bob
3	Eve
4	Craig
5	Jim
…	…

PersonFriend	
id1	**id2**
1	2
2	3
3	1
3	4
3	5
…	…

Freundesfreunde?

```
SELECT p1.name
FROM Person p1 JOIN PersonFriend pf1
  ON pf1.id2 = p1.id
JOIN PersonFriend pf2
  ON pf2.id2 = pf1.id1
JOIN Person p2
  ON pf2.id1 = p2.id
WHERE p2.name = 'Eve'
  AND p1.id <> p2.id
```

GDBMS vs. RDBMS: Traversierung

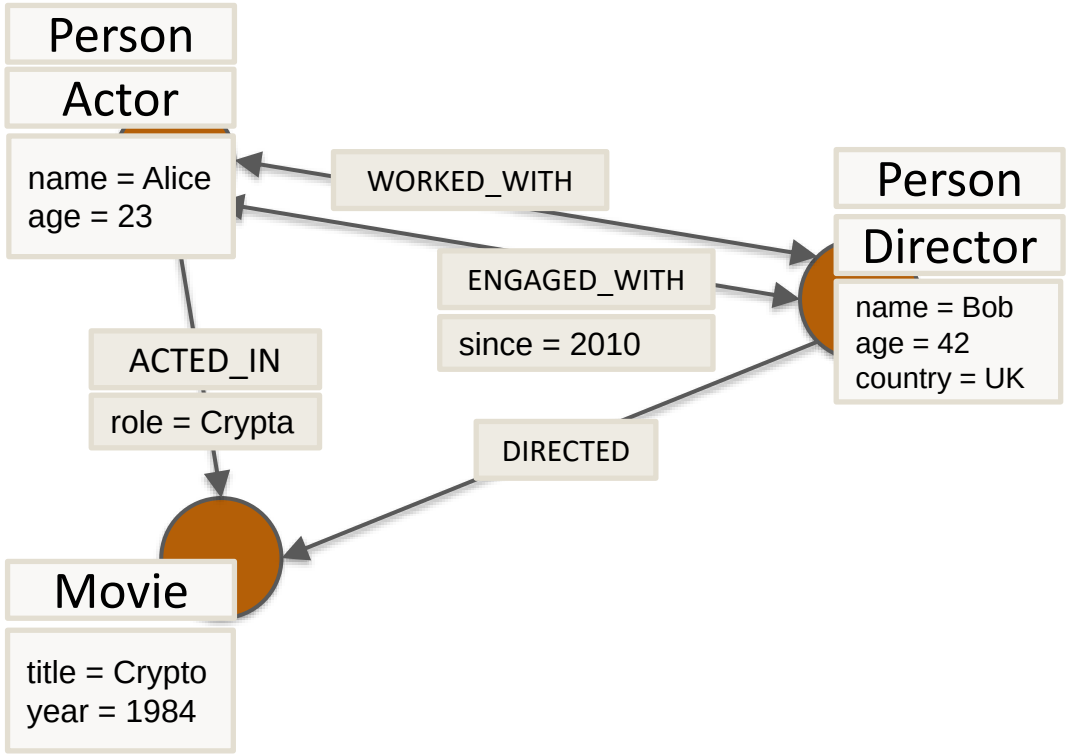
- RDBMS erfordert rekursive JOIN-Berechnung
  - Kanten sind implizit vorhanden (FK-Beziehungen)
  - Abhängigkeit zur Gesamtgröße des Graphen (oder Index notwendig)
- Probleme bei Traversierung großer Datensätzen und einer nicht-trivialen **Rekursionstiefe**
- Experiment von Vukotic und Watt (Online)
  - 1 000 000 Personen
  - ~50 Freunde pro Person
- GDBMS speichert Beziehungen am Knoten
  - Kanten sind explizit vorhanden
  - „Materialisierter Join“ bzw. **Indexfreie Adjazenz**
  - Zugriff auf Kanten unabhängig von der Gesamtgröße des Graphen

```
SELECT p1.name
FROM Person p1
JOIN PersonFriend pf1
  ON pf1.id2 = p1.id
JOIN PersonFriend pf2
  ON pf2.id2 = pf1.id1
JOIN Person p2
  ON pf2.id1 = p2.id
WHERE p2.name = 'Eve'
  AND p1.id <> p2.id
```

Tiefe	MySQL (Sek.)	Neo4j (Sek.)
2	0.016	0.01
3	30.267	0.168
4	1 543.505	1.359
5	Not finished	2.132

GDBMS vs. RDBMS: Schema

- Schema der RDBMS kann zum Problem werden
  - Oft zu starr und zerbrechlich
  - Ergebnis: spärlich besetzte Tabellen (viele NULL-Werte) oder aufwendige Umstrukturierungen
- Property-Graph-Modell: Konzentration auf Beziehungen anstatt Gemeinsamkeiten
  - Verschiedene Knoten-/ Kantentypen (Label)
  - Mehrere Typen pro Knoten/Kante
  - Unterschiedliche Attribute für gleiche Typen
- Typen und Attribute können ohne Aufwand hinzugefügt/entfernt werden

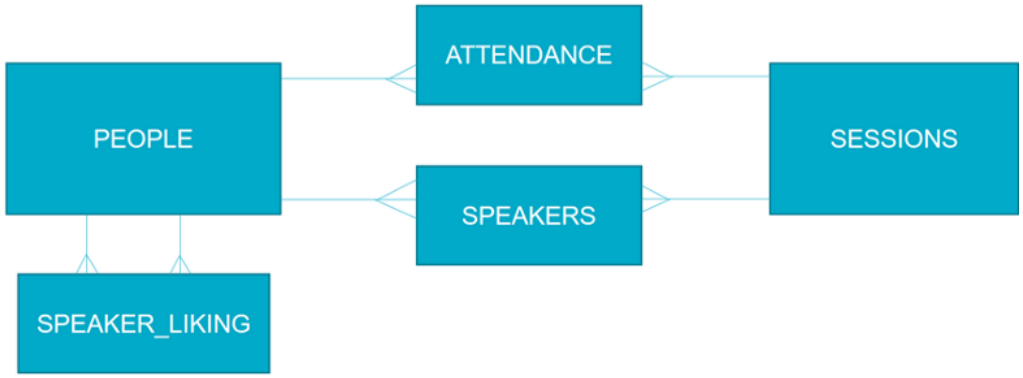


GDBMS vs. RDBMS: Anfragen

- Anfragen in GDBMS durch **Cypher**: je nach Anfrage, besser verständlich und somit leichter änderbar/wartbar
- *Beispiel*: Empfehlungssystem für Konferenzen

SQL

```
SELECT s.title
FROM People p1 JOIN Attendance a1
    ON p1.id = a1.person_id
  JOIN Attendance a2
    ON a2.session_id = a1.session_id
  JOIN Speaker_liking sl
    ON sl.attendee_id = a2.person_id
  JOIN Speaker sp
    ON sl.speaker_id = sp.speaker_id
  JOIN Sessions s
    ON sp.session_id = s.id
WHERE  p1.name = 'Name'
```



Cypher

```
MATCH
(me:People {name: 'Name'}) -[:ATTENDANCE]->
(mySess:SESSIONS) <-[:ATTENDANCE]- (other:People)
-[:SPEAKER_LIKING]-> (sp) - [:SPEAKERS] -> (session)
RETURN session.title
```

Quelle: https://medium.com/oracledevs/building-a-conference-session-recommendation-engine-using-neo4j-graph-database-2365b3b80ad9

GDBMS vs. RDBMS: Anfragen

Typische Anfragen für RDBMS:

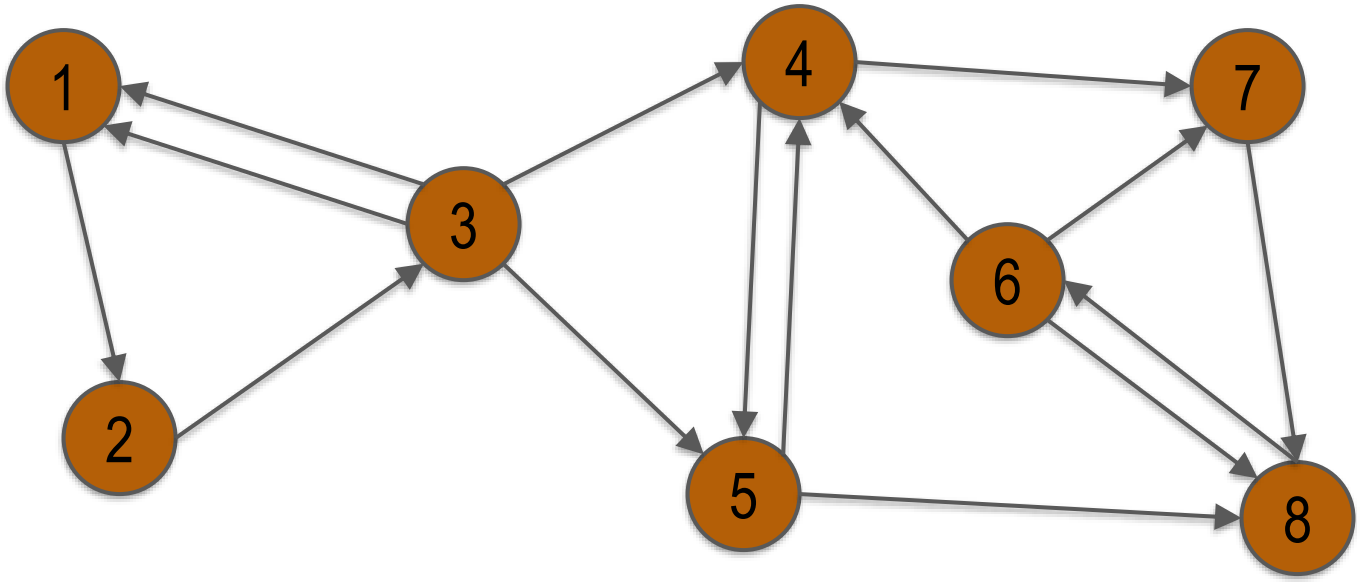
- Finde alle Mitarbeiter der Firma X!
- Finde alle Personen mit Vornamen „John“, die in Leipzig wohnen!
- Wie viele Firmen existieren insgesamt in Leipzig?
- Wie hoch ist das durchschn. Einkommen der Mitarbeiter gruppiert nach Abteilung?
- …

Typische Anfragen für GDBMS:

- Wie ist Firma X mit Firma Y verbunden?
- Über wie viele Personen kennen sich „John“ und „Paula“?
- Welche Filme kann man empfehlen, wenn jemandem der Film „Star Wars“ gefällt?
- Wer ist die einflussreichste Person im Unternehmen?
- …

Inhaltsverzeichnis: Graphdatenbanken

Graphrepräsentation

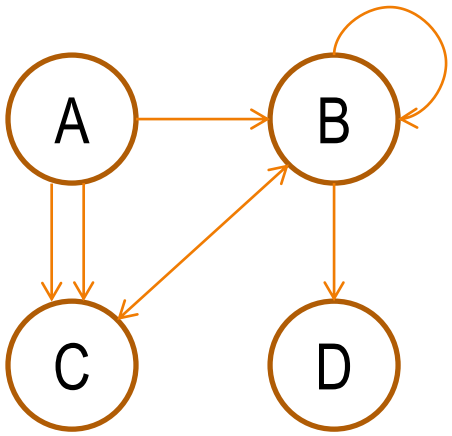


$$\boldsymbol{G}raph = (\boldsymbol{V}ertices, \boldsymbol{E}dges)$$

- $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$
- E = ((1,2), (2,3), (3,1),(3,1), (3,4), (3,5), (4,5), (5,4), (4,7), (5,8), (6,4), (6,7), (6,8), (8,6), (7,8))

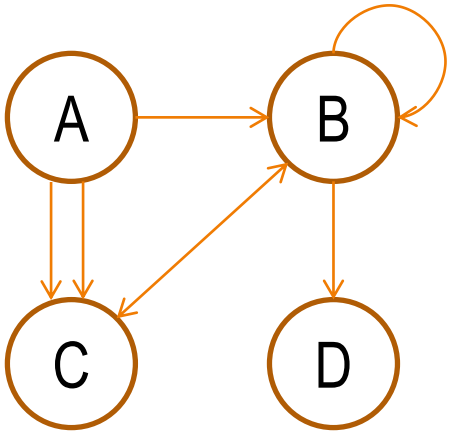
Repräsentation: Adjazenzmatrix

- Adjazenzmatrix für Graph $G = (V, E)$
  - Dimension: $n \times n$ mit $n = |V|$
  - Zelle [u, v] = Anzahl der Kanten von $u \in V$ nach $v \in V$
- Vorteil: Schneller Zugriff auf Kanten, z.B. Prüfung, ob zwei Knoten adjazent (benachbart) sind
- Nachteile
  - Hoher Speicherbedarf (quadratisch, $|V|^2$) für meist spärlich besetzte Matrizen
  - Ineffizientes Lesen aller Kanten eines Knoten (komplette Zeile/Spalte)

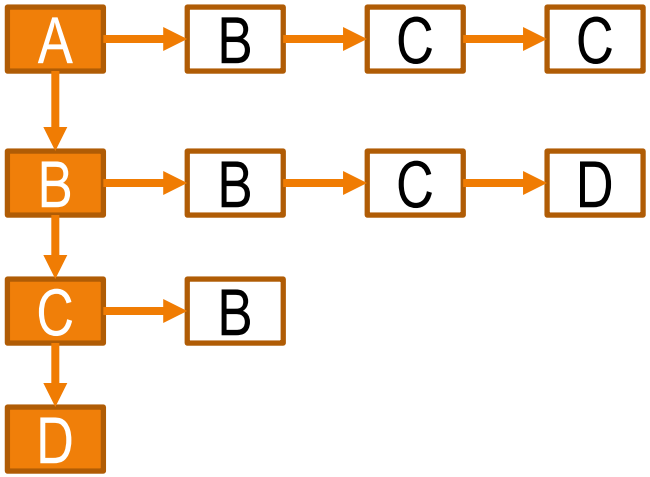


	A	B	C	D
A	0	1	2	0
B	0	1	1	1
C	0	1	0	0
D	0	0	0	0

Repräsentation: Adjazenzliste



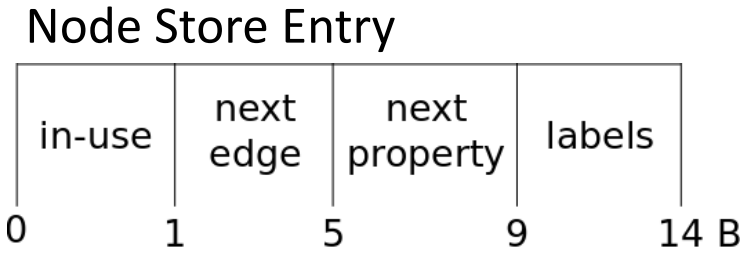
- Adjazenzliste für Graph $G = (V, E)$
  - indizierte Liste aller Knoten $v \in V$
  - jeder Knoten ist Beginn einer Liste aller Nachbarknoten (Kante in Richtung des Nachbarknotens)
- Vorteile
  - Linearer Speicherplatz: $|V| + |E|$
  - Zugriffszeit für alle Kanten eines Knoten abhängig von der Anzahl lokal ausgehender Kanten, nicht von globalen Knotenanzahl
- Nachteile
  - Aufwändigere Implementierung als Adjazenzmatrix
  - Prüfung auf spezifische Kante aufwändiger



Property Graph

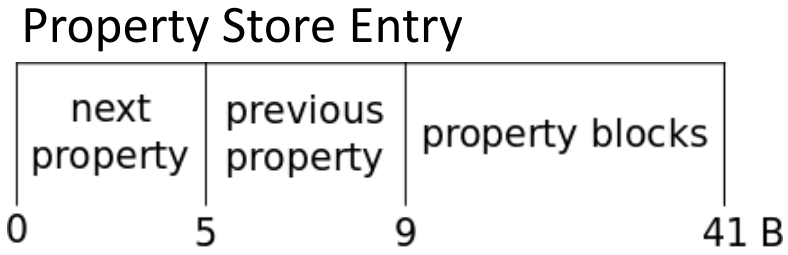
- Verschiedene Architekturen möglich
- Neo4j: Store Files, z.B.
  - Nodes: neostore.nodestore.db
  - Relationships: neostore.relationshipstore.db
  - Properties: neostore.propertystore.db
- Jeder Datenspeicher (Store) besteht aus Einträgen fester Länge
  - Z.B. 15 Byte pro Eintrag im Node-Store
  - ID eines Eintrags entspricht der *relativen Position* innerhalb eines Stores
  - **Vorteil**: Byte-Position wird durch ID * Länge in $O(1)$ ermittelt

1. *Node Store*
   - In-use: Löschen von Knoten
   - Next Edge: ID der ersten Kante
   - Next Property: ID des ersten Attributs
   - Labels: Referenz zu Label Store oder „Inline“

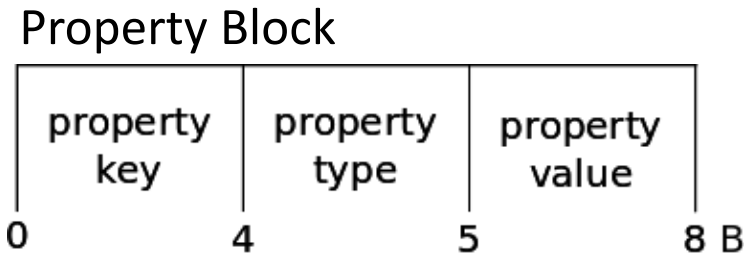


Property Graph

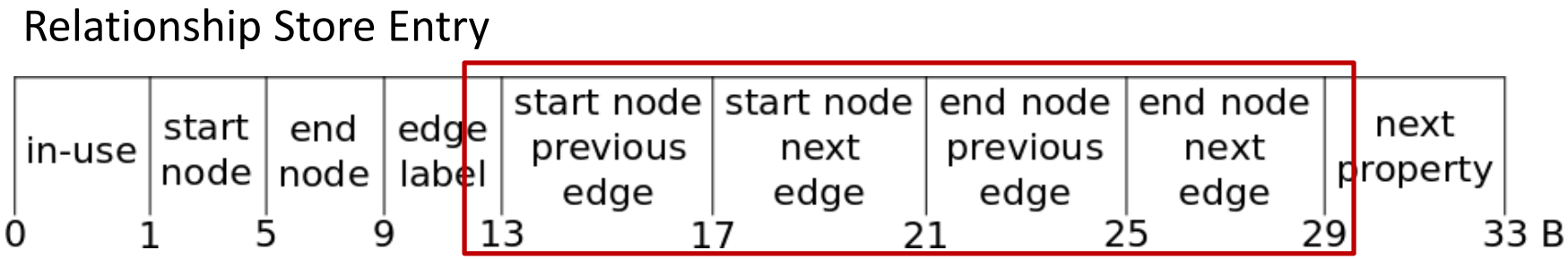
2. *Property Store*
   - Doppelt verkettete Liste von Einträgen
   - Knoten referenziert ersten Eintrag
   - 4 Blöcke pro Eintrag



- *Property Block*
   - Property Key:
     - Pointer auf Index-Datei mit Namen
     - Erlaubt die Wiederverwendung von Namen
     - Weniger Speicherbedarf bei häufig vorkommenden Attributen, wie z.B. last_name
   - Property Type:
     - Primitive Datentypen (Java)
     - Strings
     - Arrays aus primitiven Datentypen
   - Property Value:
     - Inline
     - Refernz auf dynamische Dateien (für Strings und Arrays)



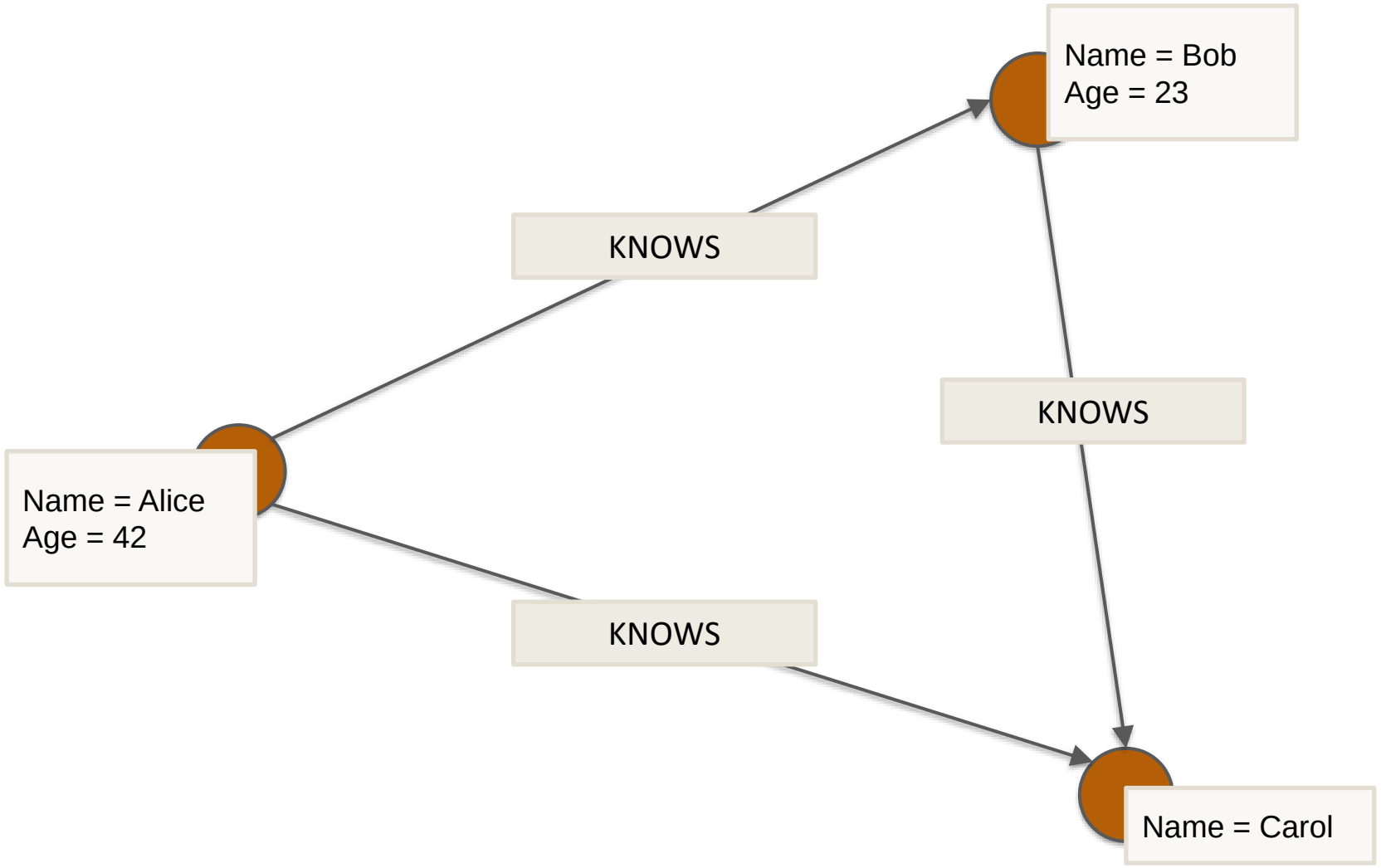
Property Graph



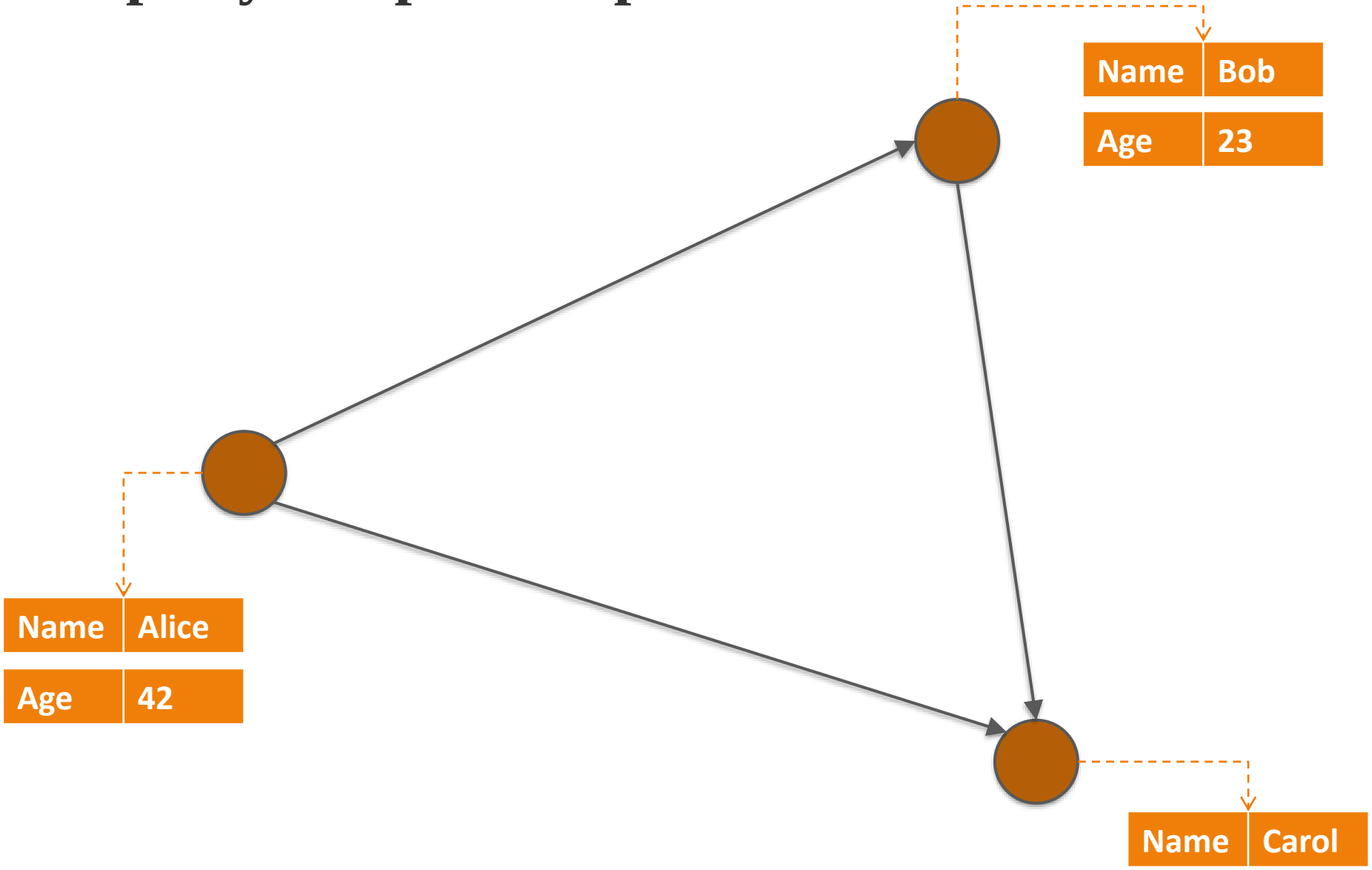
3. *Relationship Store*
   - Start/End Node: Start- und Zielknoten der Kante
   - Jede Kante besitzt Verweise auf vorhergehende und nachfolgende inzidente Kante des Start- und des Zielknotens
   - Jede Kante ist somit Element in drei doppelt verketteten Listen

- „**Indexfreie Adjazenz**“ bzw. Native Graphdatenbank:
  - Jeder Knoten besitzt „direkte“ Referenz zu aus-/eingehenden Kanten
  - Über doppelt verketteten Listen der Kanten: Berechnung aller Nachbarn in O(Anzahl der inzidenten Kanten)
  - Leistungsvorteil bei Traversierung des Graphen, der Suche nach Kanten (mit bestimmten Label) und beim Einfügen/Löschen von Kanten

Property Graph: Beispiel



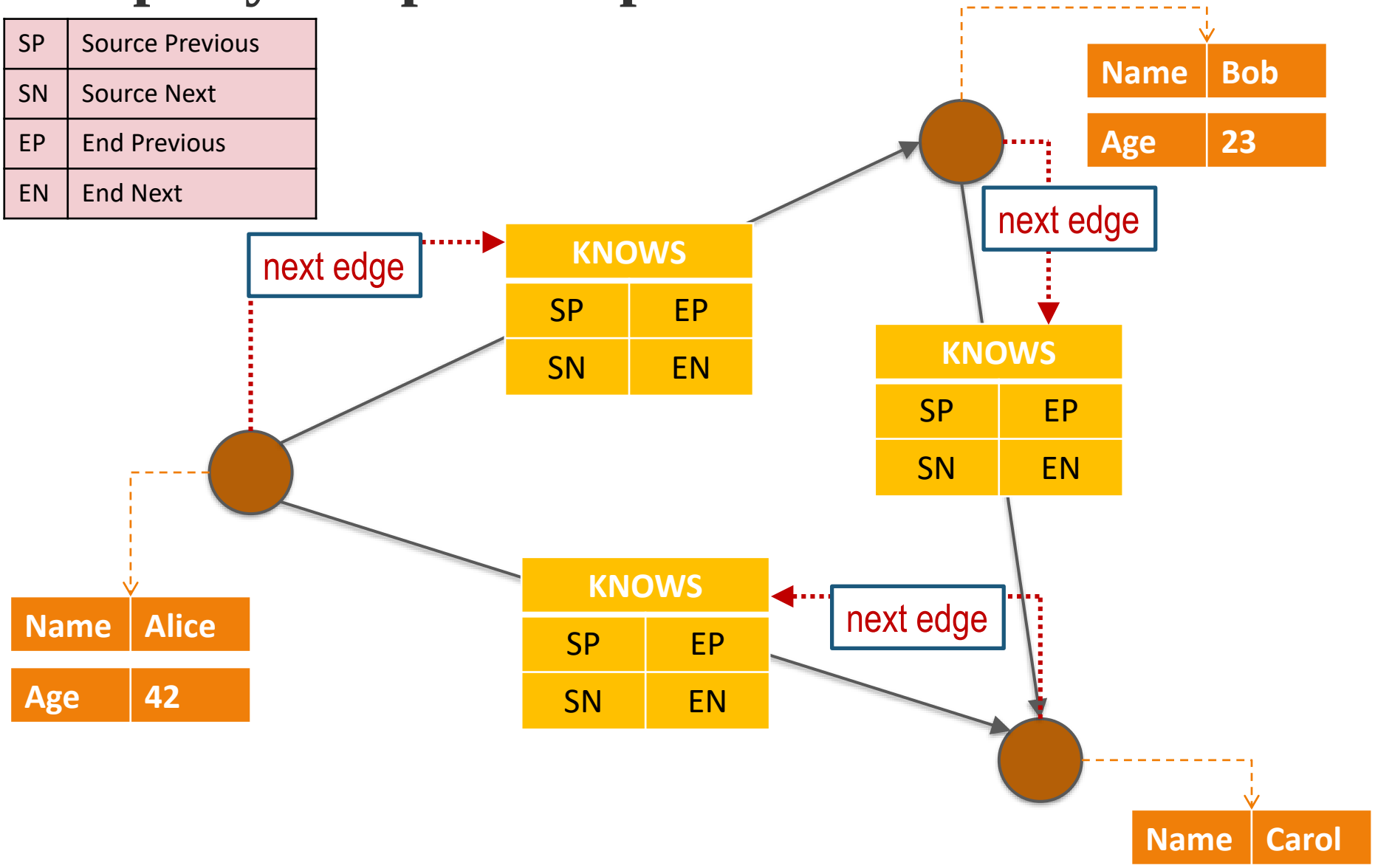
Property Graph: Beispiel



Quelle: Tobias Lindaaker – Neo4j Internals

Property Graph: Beispiel

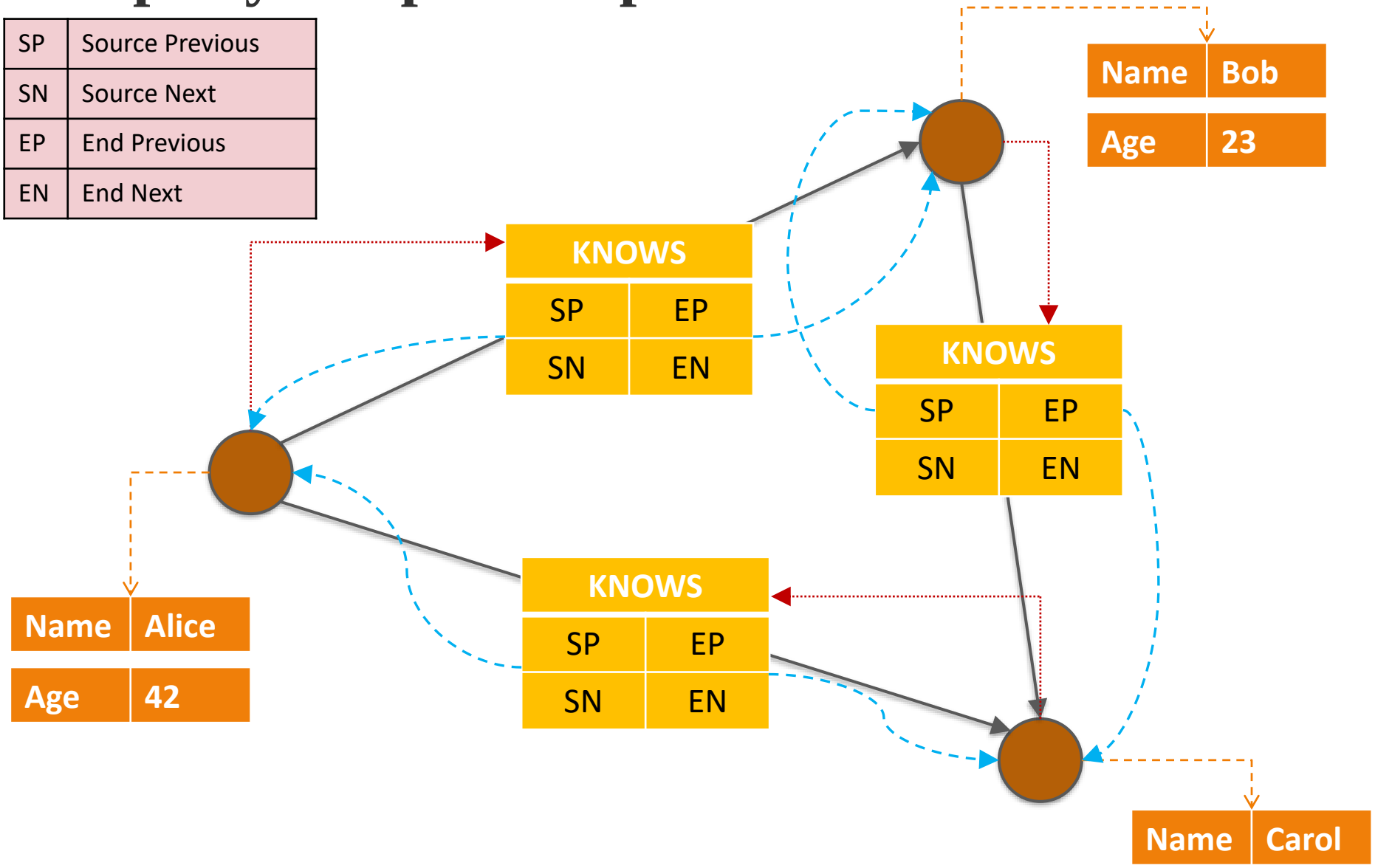
SP	Source Previous
SN	Source Next
EP	End Previous
EN	End Next



Quelle: Tobias Lindaaker – Neo4j Internals

Property Graph: Beispiel

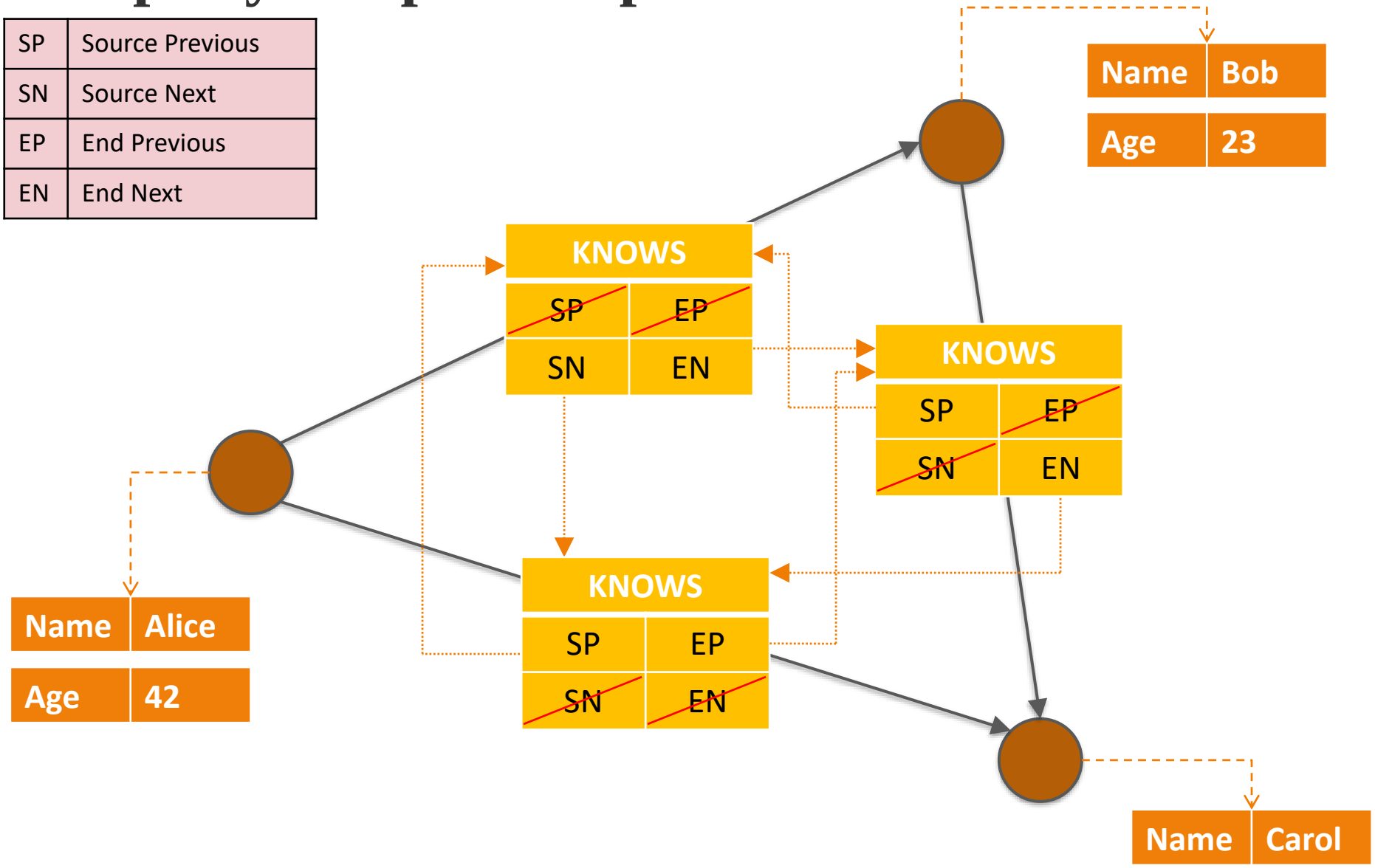
SP	Source Previous
SN	Source Next
EP	End Previous
EN	End Next



Quelle: Tobias Lindaaker – Neo4j Internals

Property Graph: Beispiel

SP	Source Previous
SN	Source Next
EP	End Previous
EN	End Next



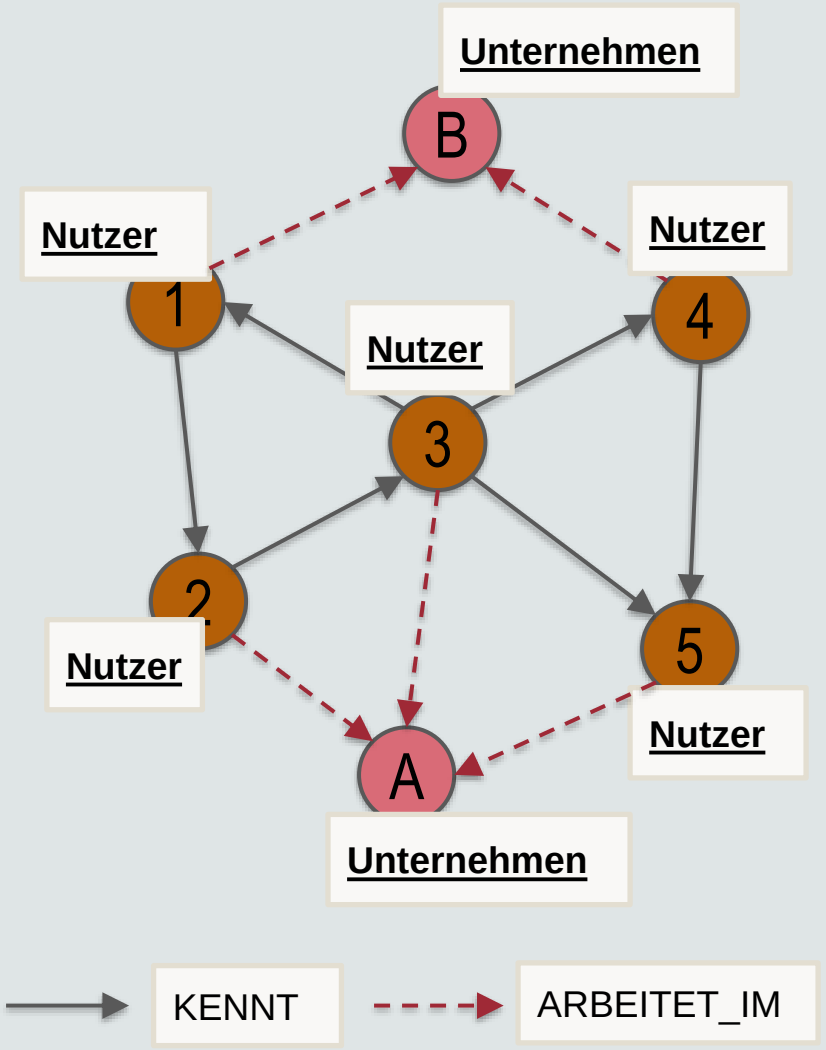
Quelle: Tobias Lindaaker – Neo4j Internals

Übung: arsnova.rz.uni-leipzig.de 81 60 01 90

Wie viele Datenbankzugriffe benötigt eine **Graphdatenbank** (nach obiger Architektur) für folgende Anfrage:

Wie viele der Bekannten des Nutzers mit ID = 3 arbeiten im Unternehmen A?

- Die Richtung der Beziehung KENNT spielt dabei keine Rolle.
- „Nutzer“ und „Unternehmen“ sind Label.
- Der Name des Unternehmens ist als Property gespeichert.



Übung: arsnova.rz.uni-leipzig.de 81 60 01 90

Wie viele Datenbankzugriffe benötigt eine **relationale Datenbank** für die selbe Anfrage? Nehmen Sie an, dass für die vier grau hinterlegten Spalten ein **Hashindex** angelegt wurde (konstante Zugriffszeit).

Wie viele der Bekannten des Nutzers mit ID = 3 arbeiten im Unternehmen A? Die Richtung der Beziehung KENNT spielt dabei keine Rolle.

KENNT

StartKnoten	EndKnoten
1	2
2	3
3	1
3	4
3	5
4	5

ARBEITET_IM

StartKnoten	EndKnoten
1	2
2	1
3	1
4	2
5	1

Unternehmen

ID	Name
1	A
2	B