



UNIVERSITÄT
LEIPZIG

NOSQL DATENBANKEN

Kapitel 3: Graphdatenbanken Teil 1

Christopher Rost

Prof. Dr. Erhard Rahm

rost@informatik.uni-leipzig.de

Sommersemester 2024



INHALT DER VORLESUNG

Kap. 2: Verteilte Datenbanksysteme

- Partitionierung, Replikation, Verfügbarkeit, Serialisierbarkeit, CAP-Theorem

Kap. 3: Graphdatenbanken

- Repräsentation der Daten als Knoten und Kanten mit Properties



Kap. 4: Key-Value Stores

- Kollektion von Key/Value-Paaren



Kap. 5: Document Stores

- Speicherung semistrukturierter Daten als Dokumente (JSON)



Kap. 6: Extensible Record Stores

- Tabellenartige Strukturen mit flexibler Erweiterung der Attribute





INHALTSVERZEICHNIS

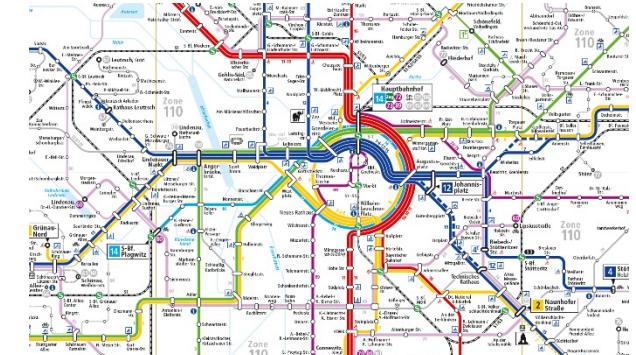
- **Einführung**
- **Vergleich mit relationalen Datenbanksystemen**
- **Repräsentation von Graphdaten**
- **Anfragesprachen**
- **Anwendungen**
- **Beispiel: Neo4j**
- **Das Raft Protokoll**



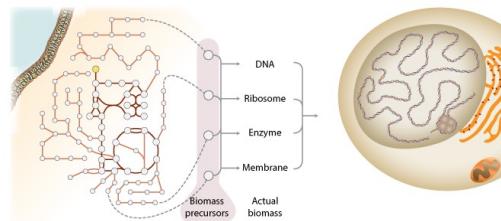
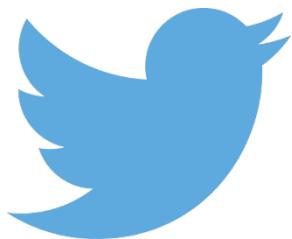
EINFÜHRUNG

Verstärktes Aufkommen von Graphdaten

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



Quelle: <http://www.lvb.de>

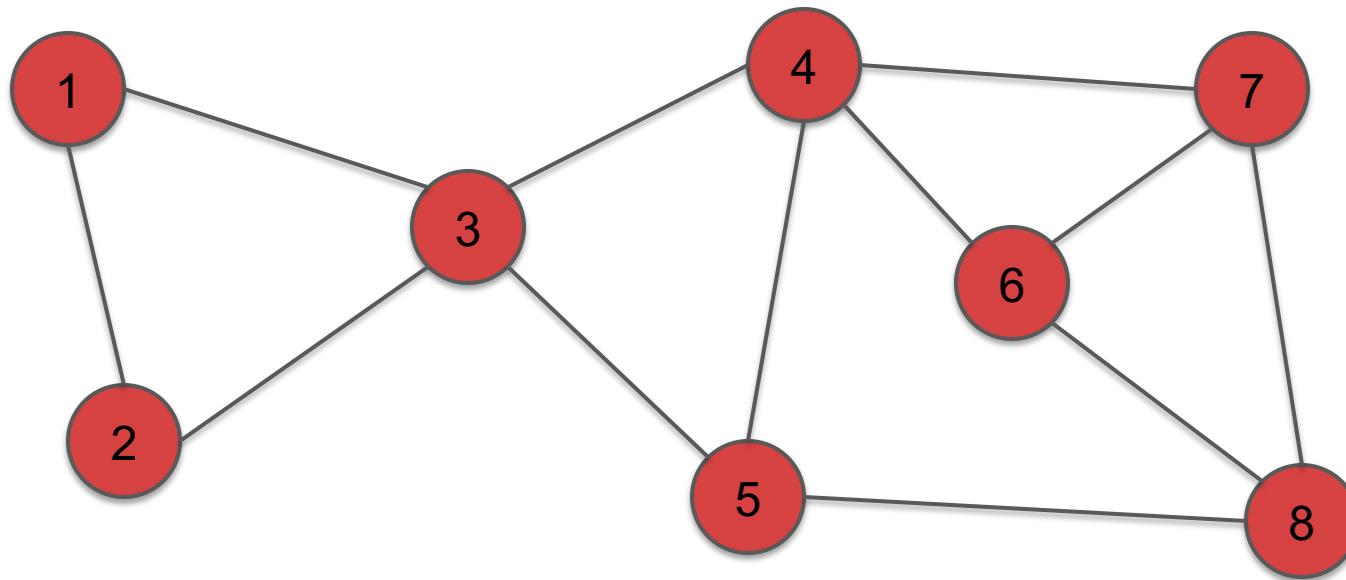


Quelle: <http://bigg.ucsd.edu>





DATENMODELL: GRAPH

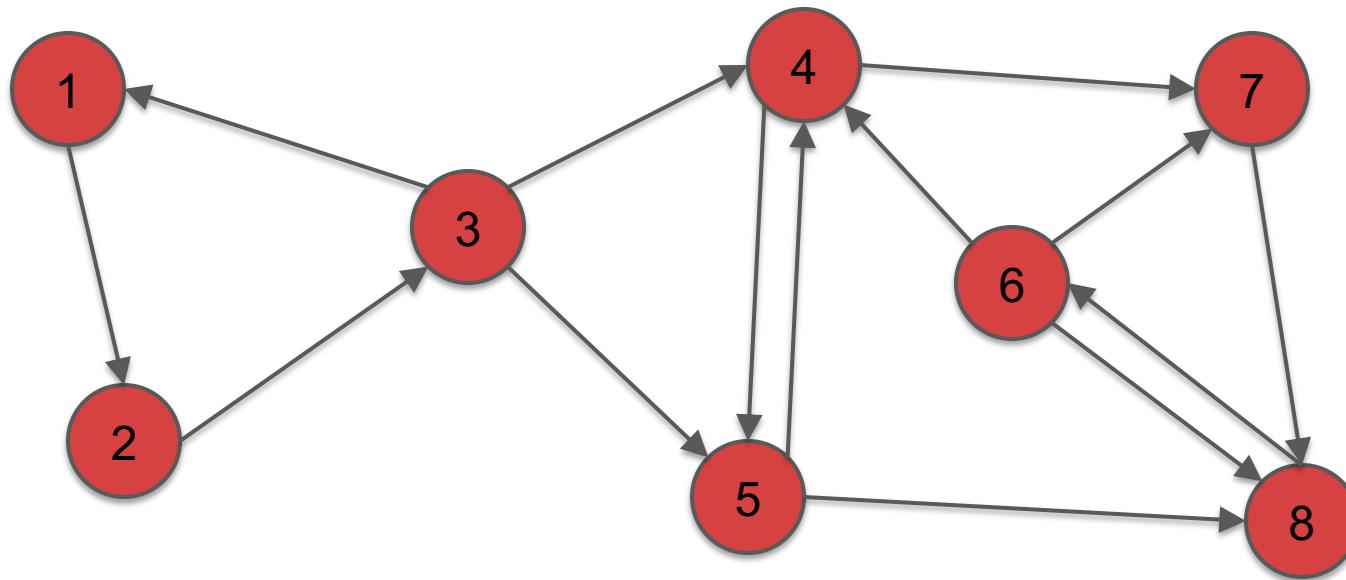


$$Graph = (Vertices, Edges)$$

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



GERICHTETER GRAPH

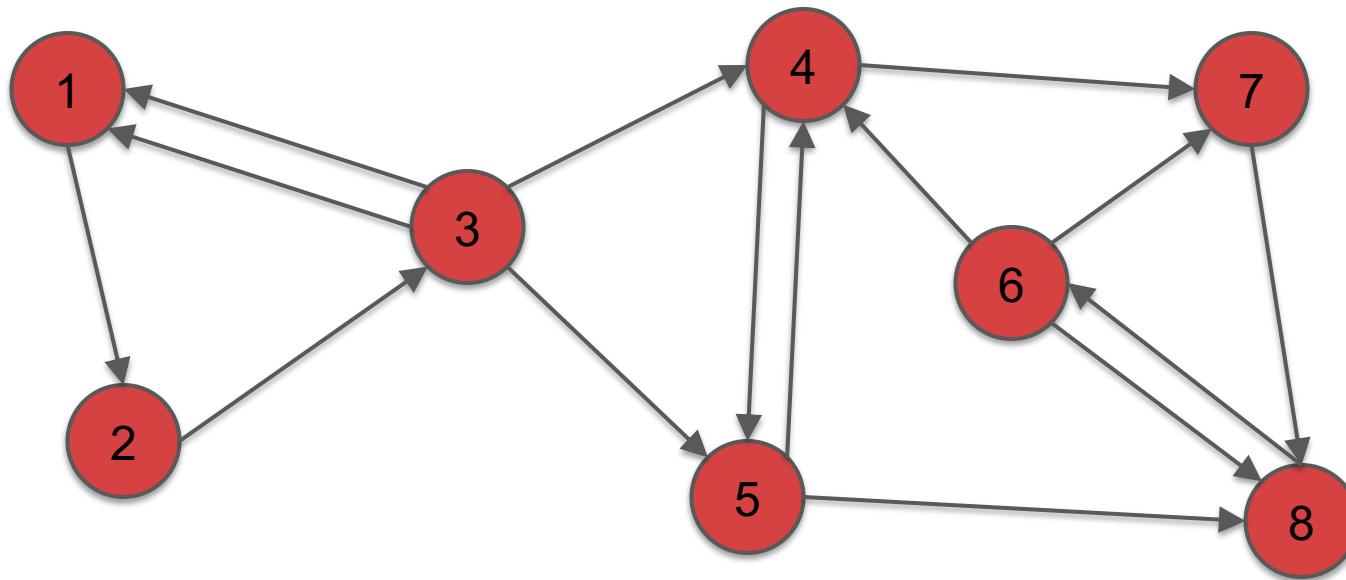


$$Graph = (Vertices, Edges)$$

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



GERICHTETER MULTIGRAPH

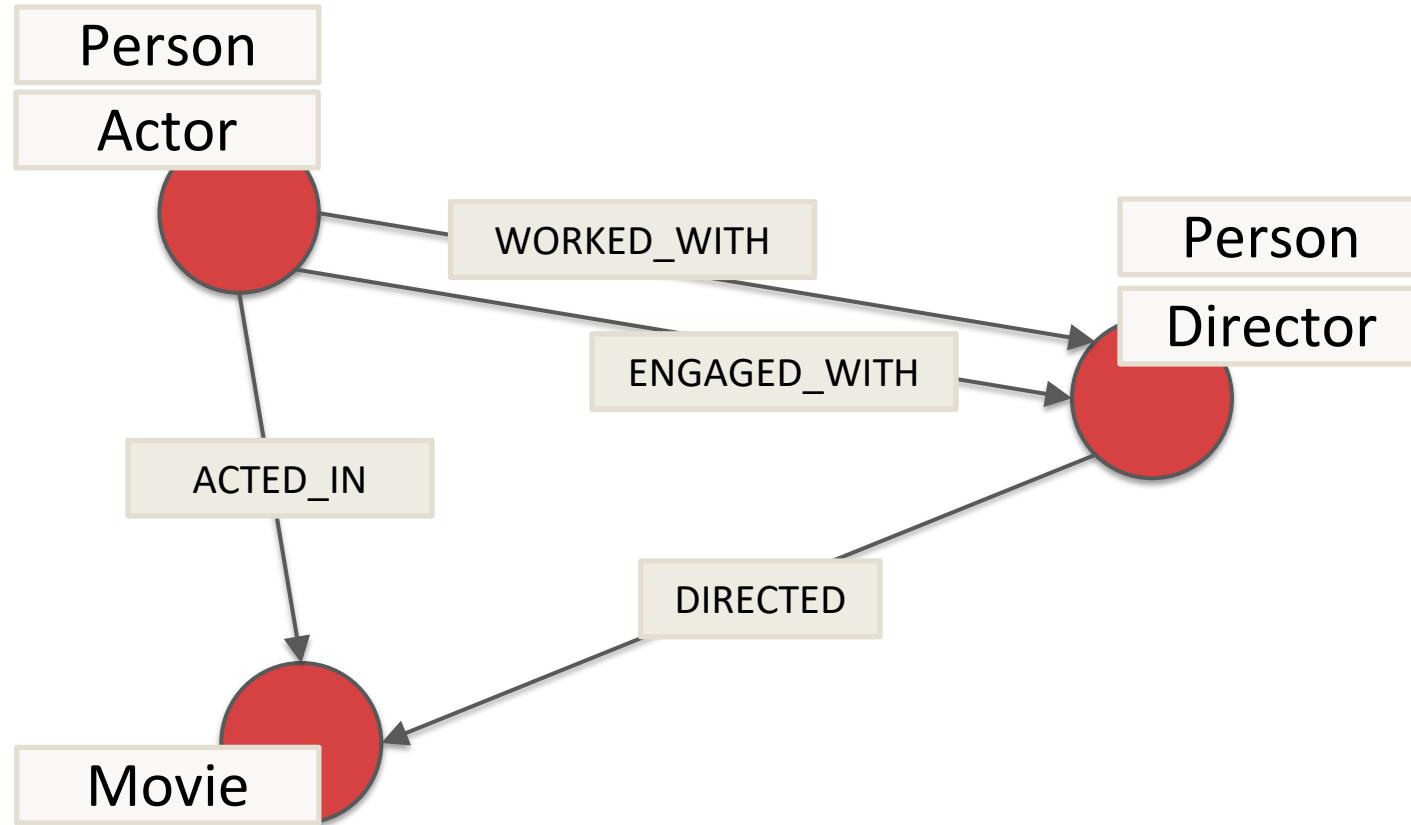


$$Graph = (Vertices, Edges)$$

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



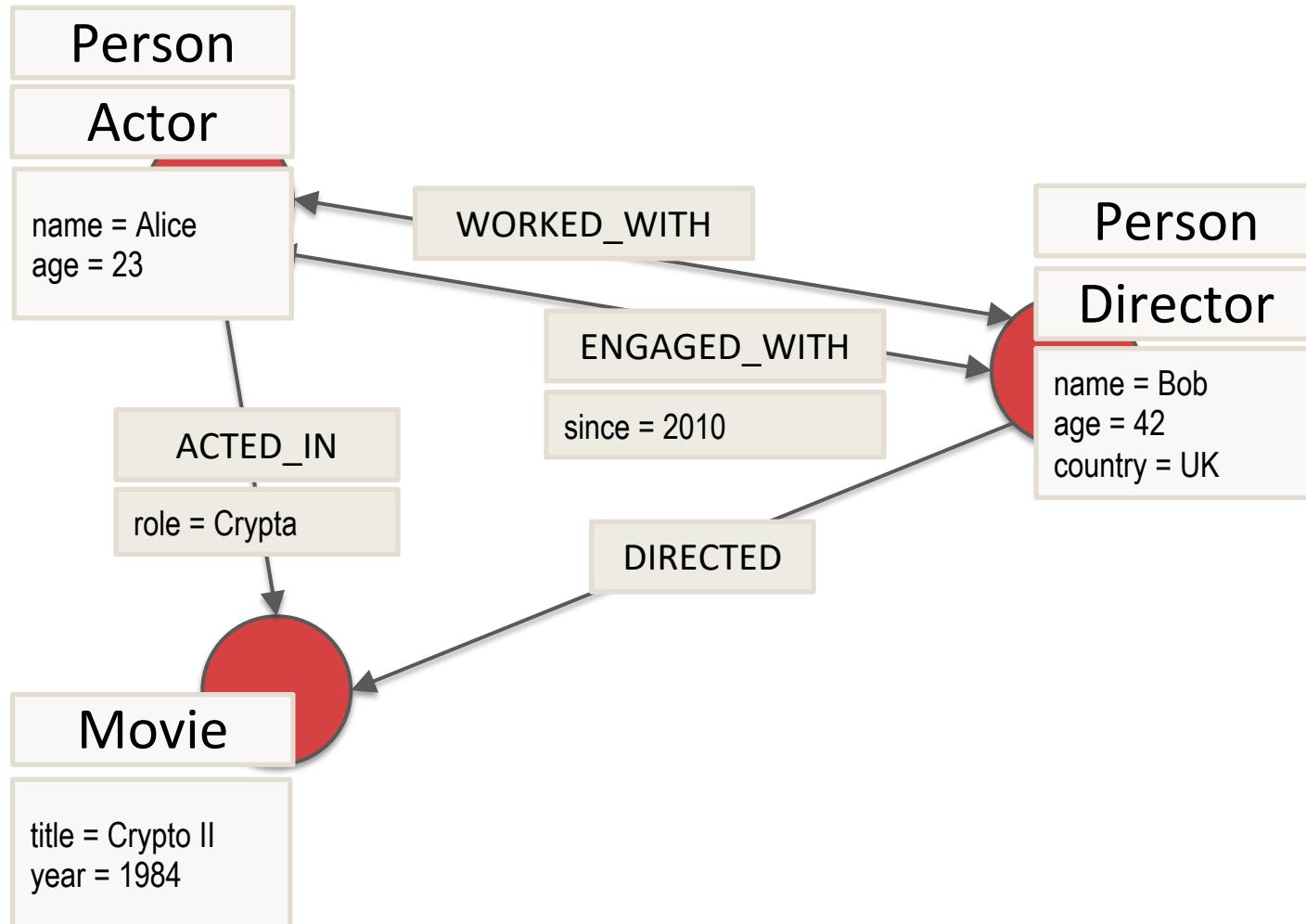
LABLED GRAPH



- gerichtet (directed)
- mit Beschriftung / Label (labeled)



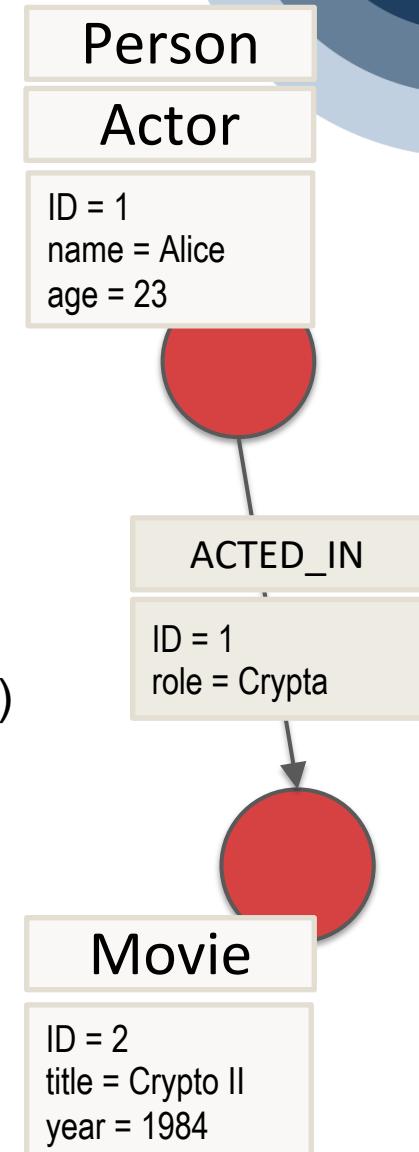
PROPERTY GRAPH





PROPERTY GRAPH MODEL

- Definierte Eigenschaften
 - directed
 - labeled (Knoten und Kanten)
 - multi-graph
 - self-loops
 - optional properties (Knoten und Kanten)
- Properties: Menge von Schlüssel-Wert-Paaren
 - Schlüssel: String
 - Wert: Object (String, Number, Timestamp, Vektoren [NV])
- Knoten/Kanten im Allgemeinen **schemafrei**
 - Beliebige Label, Attribute und Datentypen
 - Aktuelle Forschung: PG-Schemas [Ang23]
- Grundlage vieler Graphdatenbanken
 - Neo4j, Oracle Graph Database, GraphScope, ...

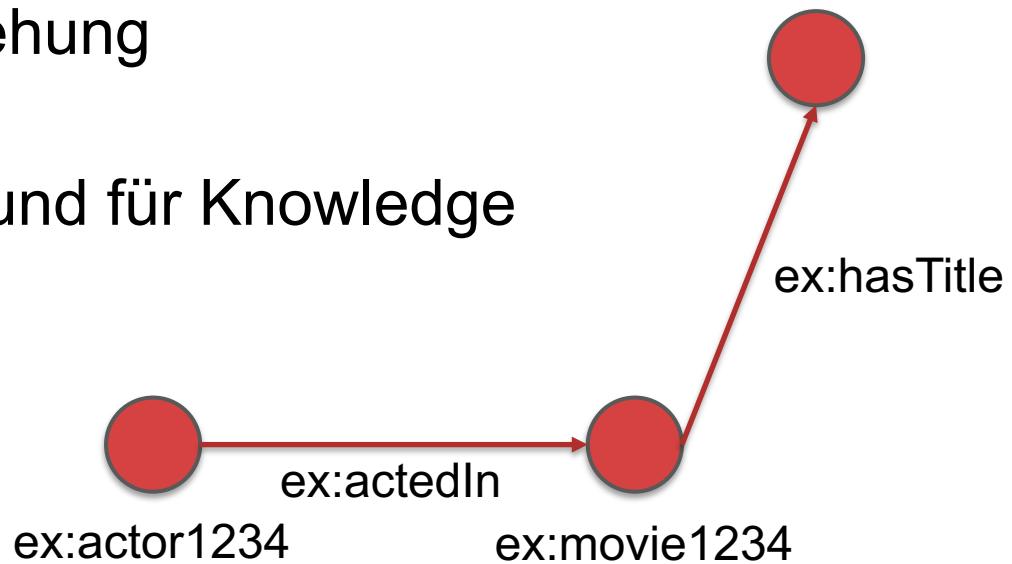




RDF

- Resource Description Framework
- semantisches Graph-Datenmodell
- Tripel = Subjekt, Prädikat, Objekt
- Identifikation via URIs
 - Objekt kann auch Literal oder Zeitstempel sein “Crypto II”^^xsd:string
- Prädikat beschreibt Beziehung
 - durch URI beschrieben
- Für Anwendung im Web und für Knowledge Graphen (KG) optimiert

@prefix ex: <<http://example.org/>> .





GRAPH MODELL - ERWEITERUNGEN

- Zeitliche Graph-Modelle (temporal)
 - Z.B. Temporal Property Graph Model (TPGM)
 - Bitemporale Erweiterung an Knoten und Kanten
 - Wann ist eine Beziehung gültig und wie lange? Wann wurde die Information über diese Beziehung in die Graph-DB eingefügt oder geändert?
- Räumliche Graph-Modelle (spatial)
 - Knoten mit räumlichen Koordinaten
 - Ein-, zwei- oder dreidimensional typisch
 - Wo befindet sich ein Knoten und wie weit sind seine Nachbarn voneinander entfernt?
- Räumlich-Zeitliche Graph-Modelle (spatio-temporal)
 - Erweiterung um Koordinaten und Zeitpunkte / -intervalle
 - Wann ist ein Pfad zwischen zwei entfernten Knoten entstanden?
 - Z.B. Transportationsnetzwerke, Fahrrad-Ausleihen



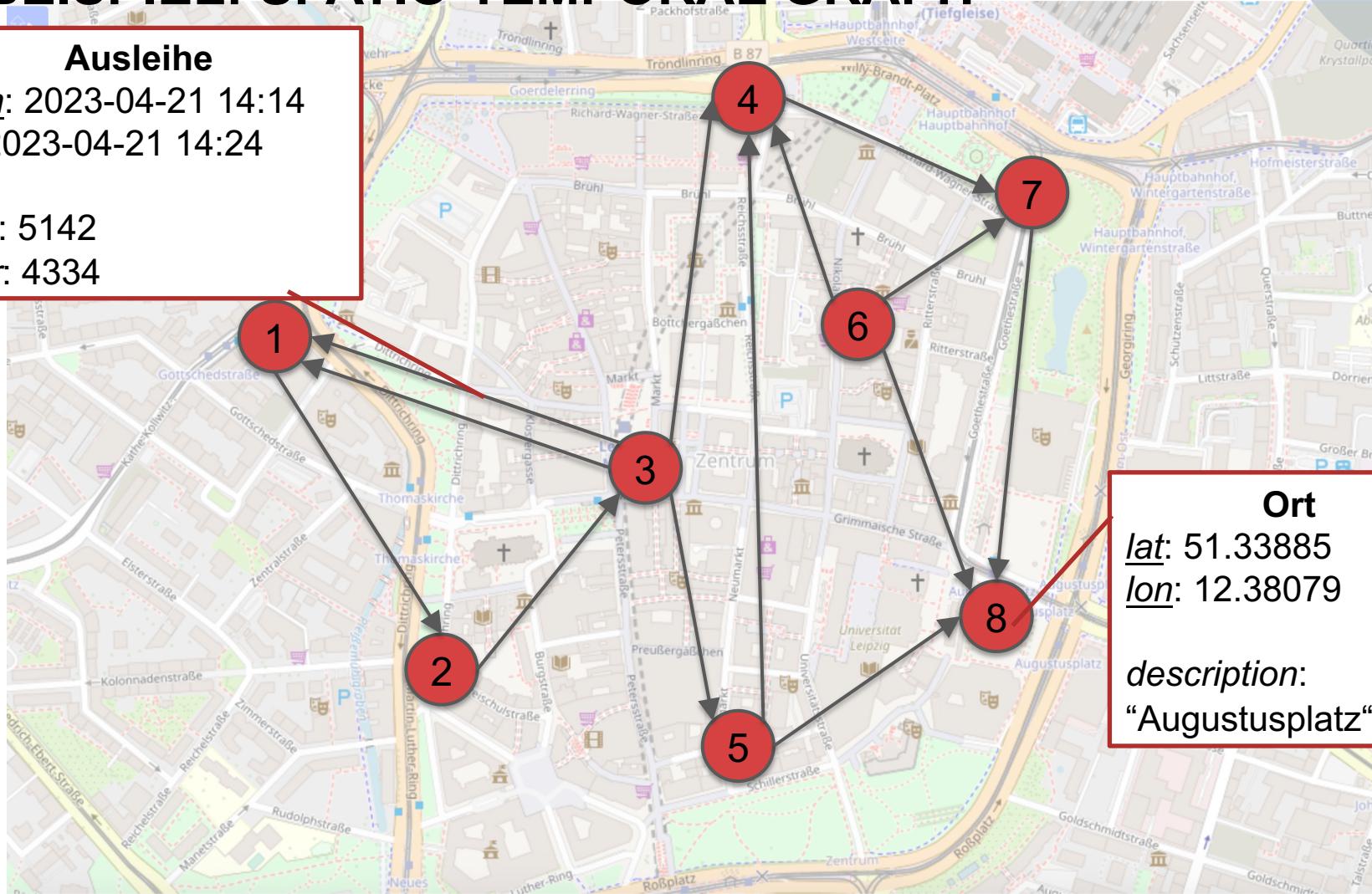


BEISPIEL: SPATIO-TEMPORAL GRAPH

Ausleihe

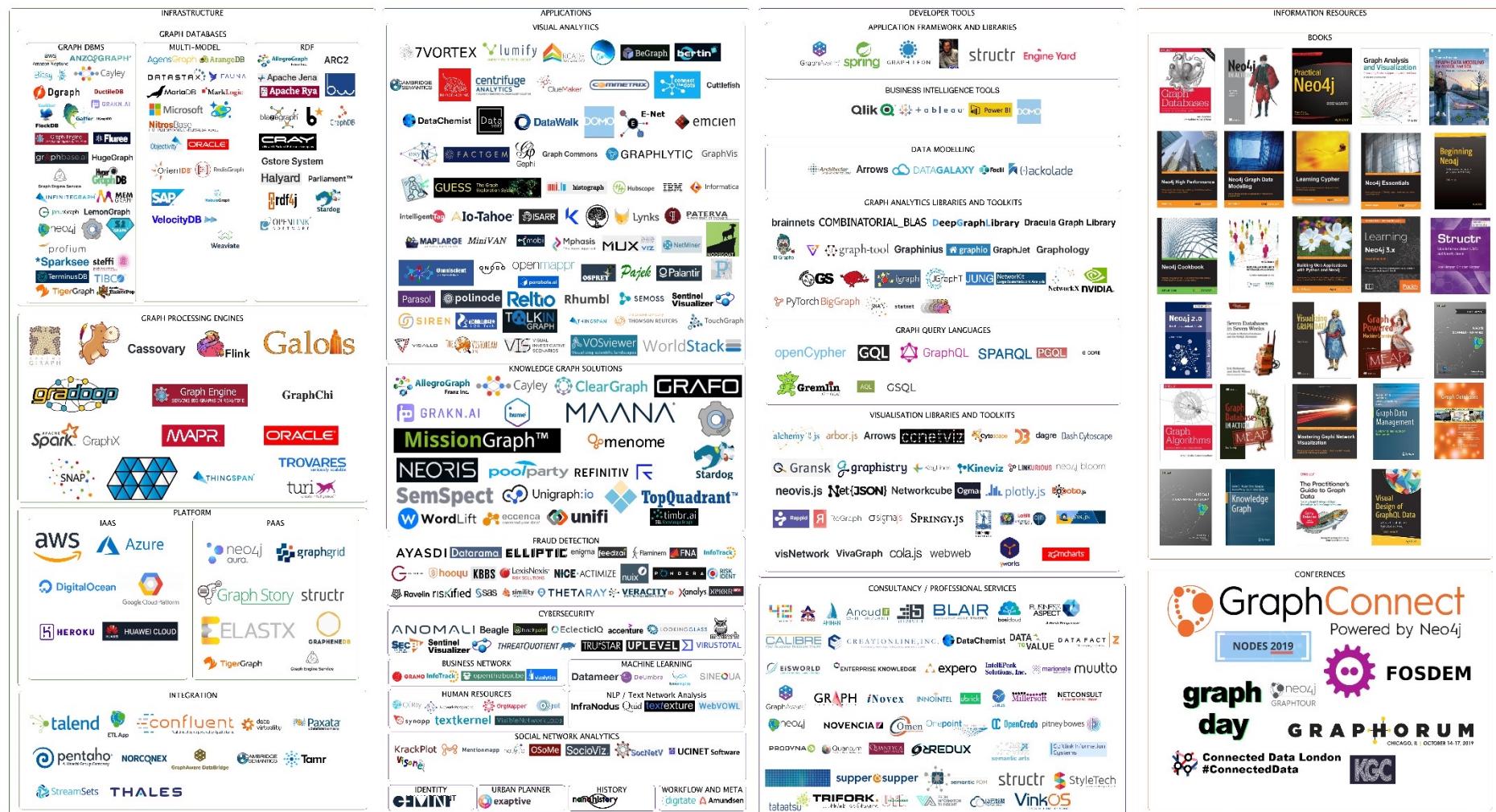
from: 2023-04-21 14:14
to: 2023-04-21 14:24

bike: 5142
user: 4334





GRAPH TECHNOLOGIE LANDSCAPE



2020 v2

Quelle: <https://graphaware.com/graphaware/2020/02/17/graph-technology-landscape-2020.html>

Credits: **GraphCoding**



GRAPH TECHNOLOGIE LANDSCAPE

Semantic Graphs

Follow Semantic standards (RDF/SPARQL)
Some support properties via RDF*



Property Graphs

Support labelled properties with Cypher or proprietary languages



Big Vendors with Graph Support

Support graph with proprietary database



Multi-model Graphs

Support graph with APIs on top of NoSQL DBs



<https://medium.com/swlh/top-seven-hits-and-misses-in-the-graph-database-world-ab98cc785c61>



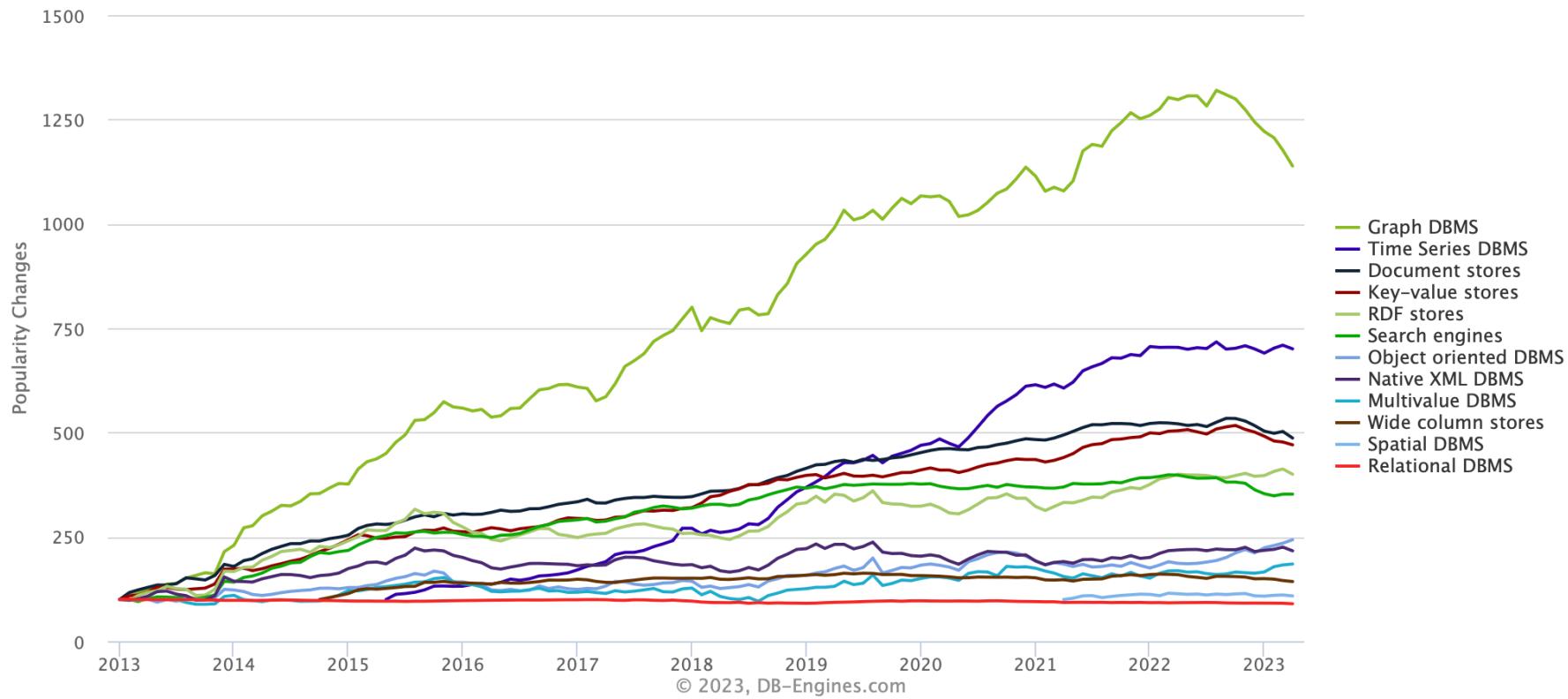
GRAPHDATENMANAGEMENT: VERGLEICH

Graphdatenbanken	Graph Processing Framework
Persistenz von und interaktiver Zugriff auf Graphdaten (OLTP)	Verarbeitung/Analyse umfangreicher Graphdaten
Unmittelbare Ausführung von Anfragen	Batch/Stream-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

Complete trend, starting with January 2013



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



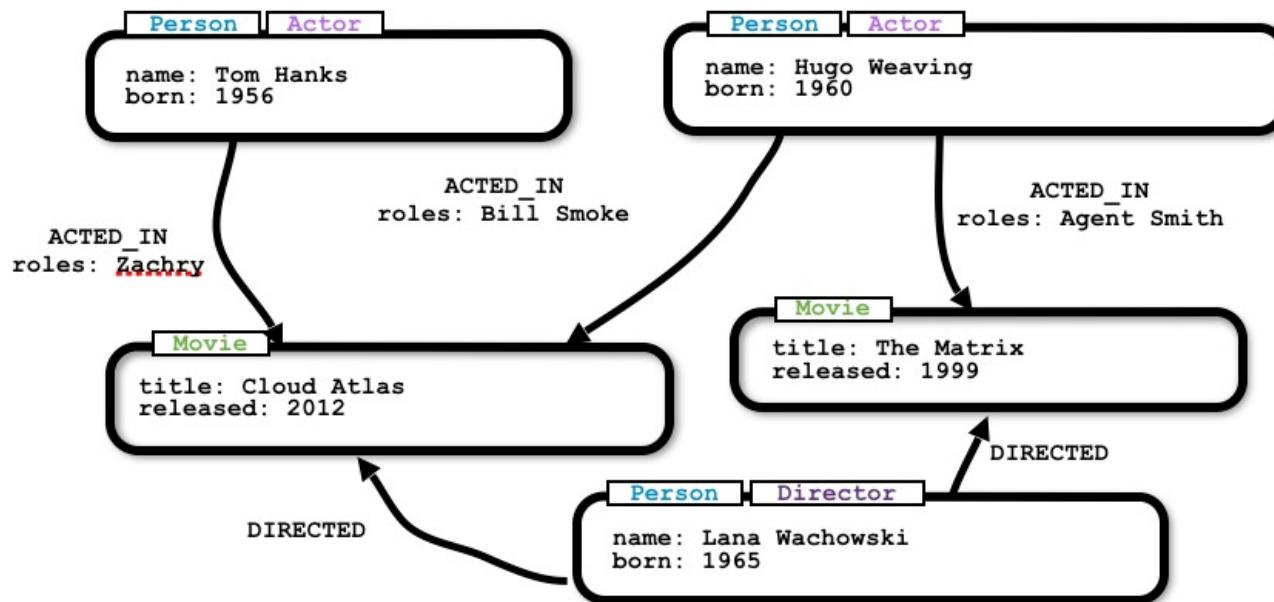
INHALTSVERZEICHNIS

- Einführung
- Vergleich mit relationalen Datenbanksystemen
- Repräsentation von Graphdaten
- Anfragesprachen
- Anwendungen
- Beispiel: Neo4j
- Das Raft Protokoll



GDBMS VS. RDBMS: MODELLIERUNG

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

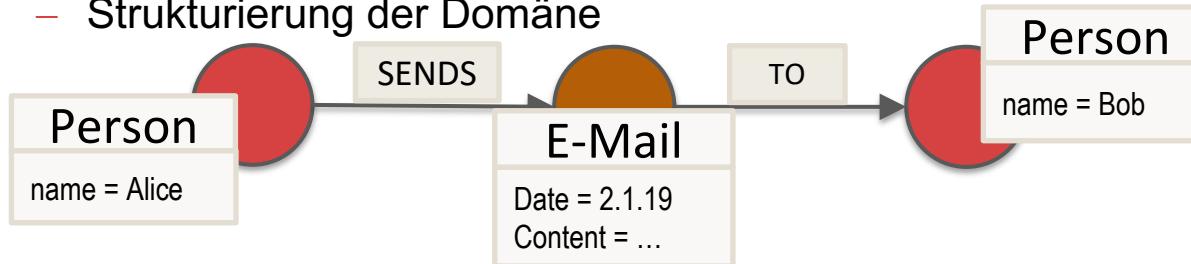


Quelle: https://neo4j.com/developer/guide-data-modeling/#_graph_data_model_whiteboard_friendly



GDBMS VS. RDBMS: MODELLIERUNG

- „Schwierigkeit“ bei Graphen: Identifizierung von Knoten und Kanten
 - Längere Latenzzeiten bei unpassender Modellierung
 - **Wichtig: 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

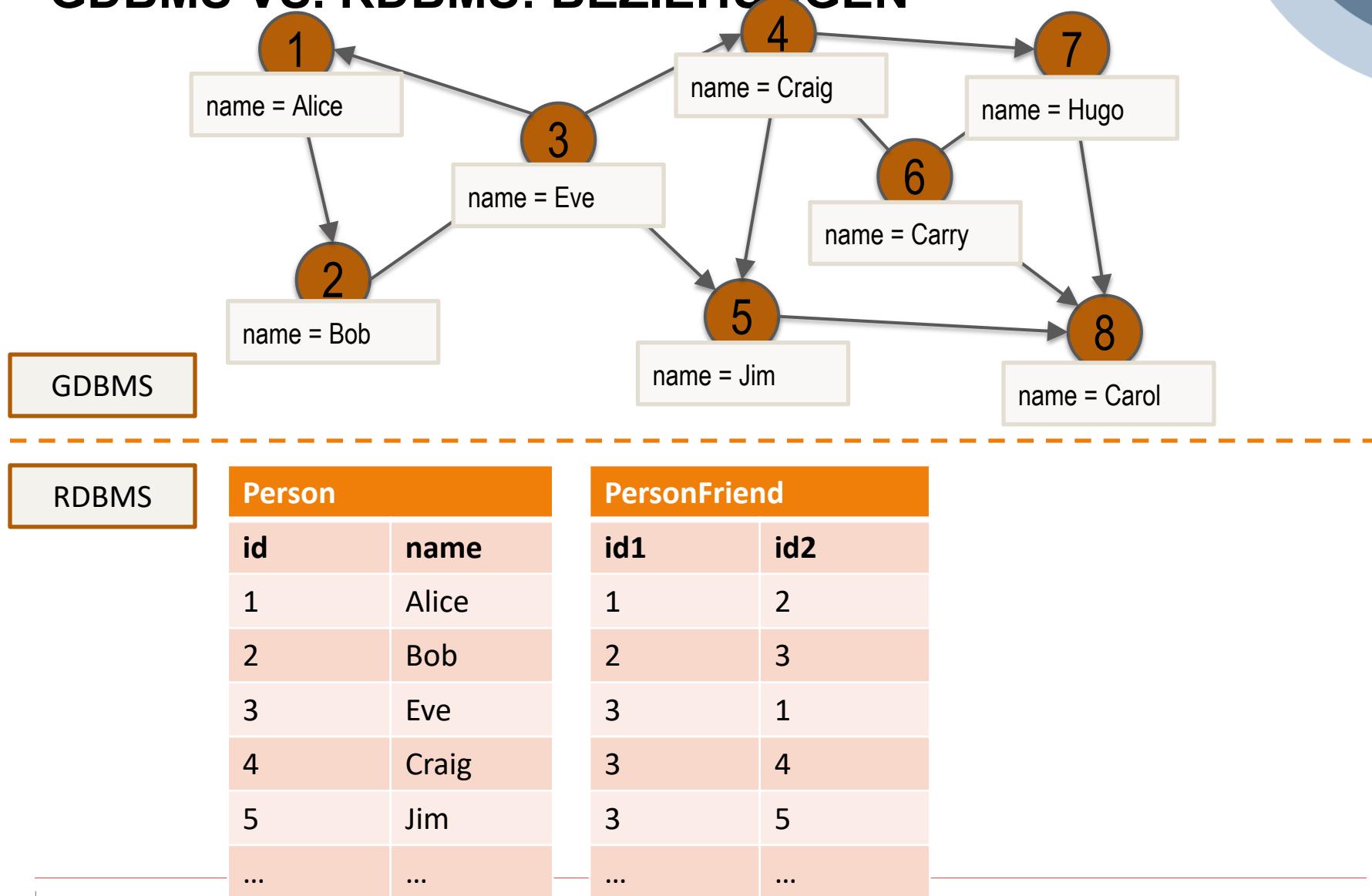


- Beispiel: E-Mail als Knoten vs. E-Mail als Kante



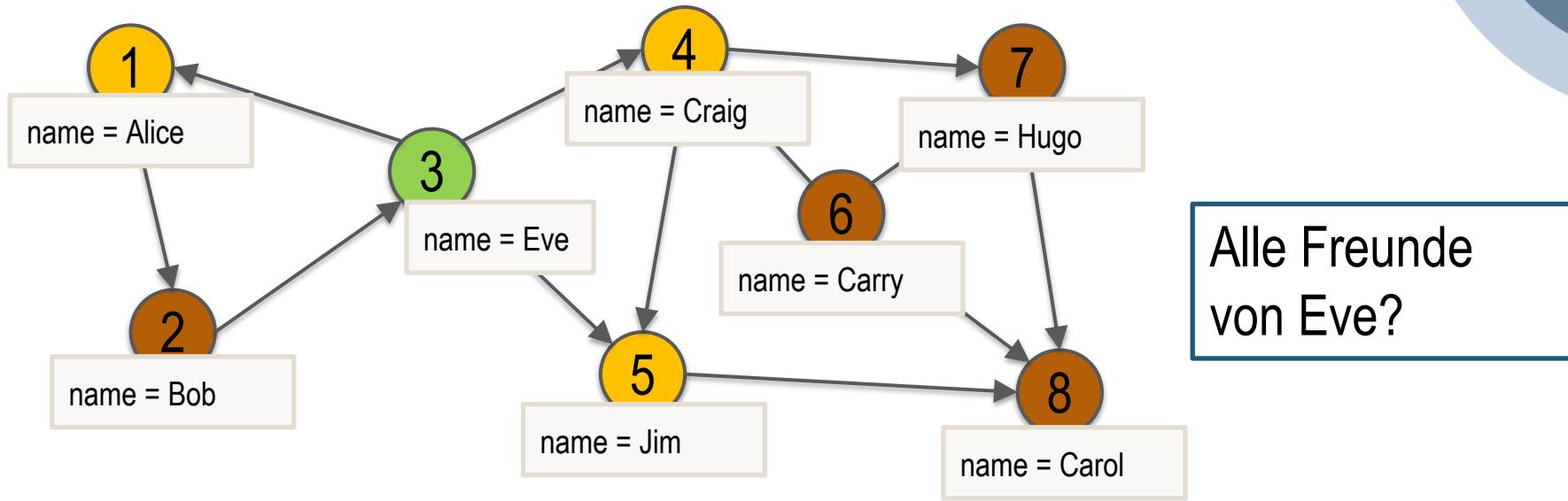


GDBMS VS. RDBMS: BEZIEHUNGEN





GDBMS VS. RDBMS: BEZIEHUNGEN



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

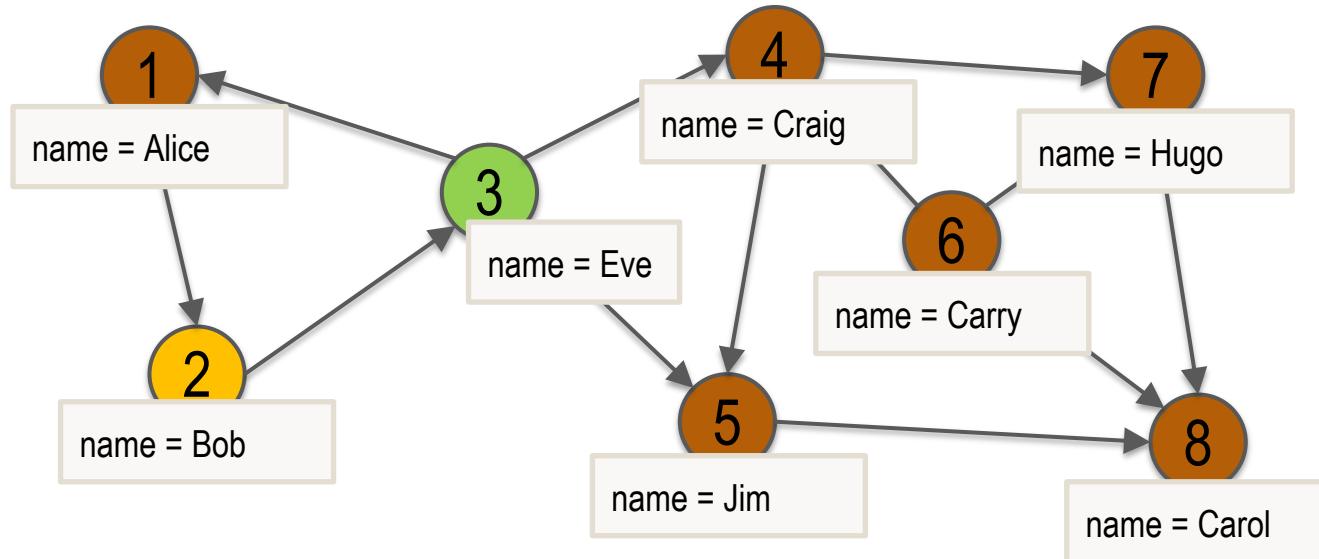
PersonFriend	
id1	id2
1	2
2	3
3	1
3	4
3	5
...	...

```

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



Gegenrichtung:
Wer ist mit Eve
befreundet?

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

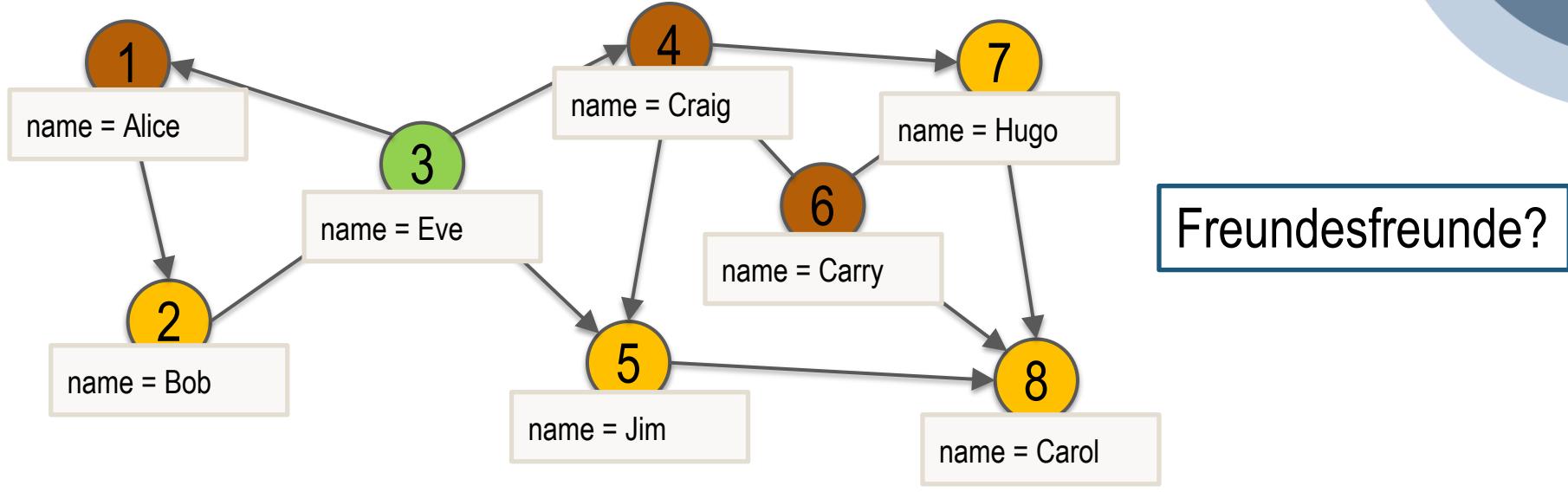
PersonFriend	
id1	id2
1	2
2	3
3	1
3	4
3	5
...	...

```

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
...	...

```

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 [Vuk15]
 - 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

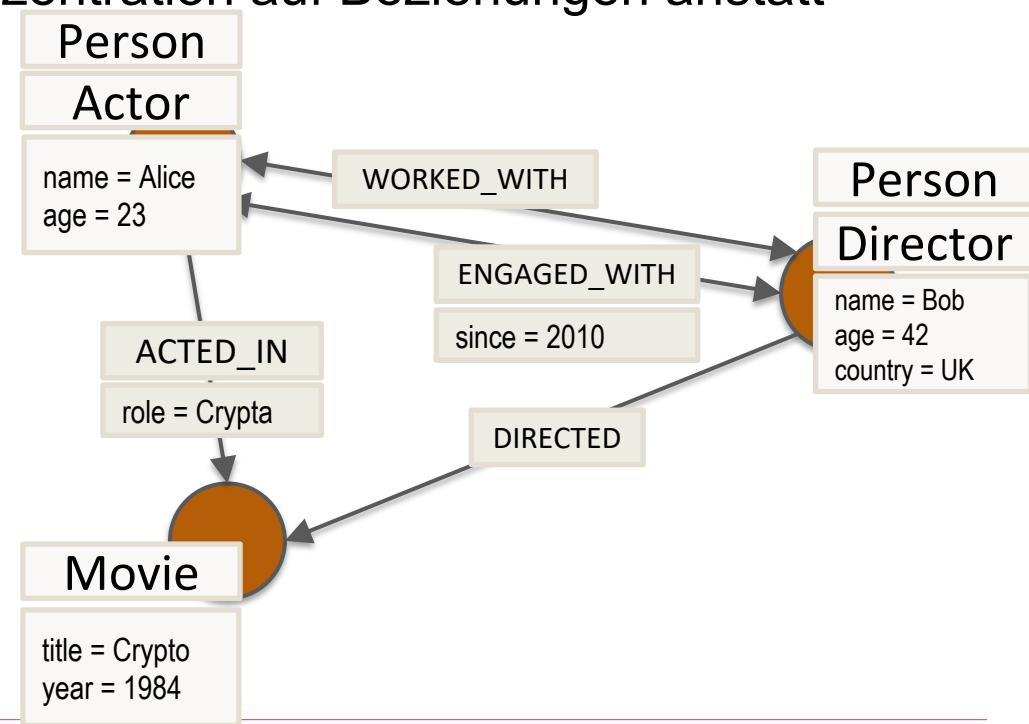
```
SELECT p1.name
FROM Person p1
JOIN PersonFriend pf1
ON pf1.id2 = p1.id
JOIN PersonFriend pf2
ON pf2.id2 = pf1.id1
JOIN Person p2
ON pf2.id1 = p2.id
WHERE p2.name = 'Eve'
AND p1.id <> p2.id
```

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



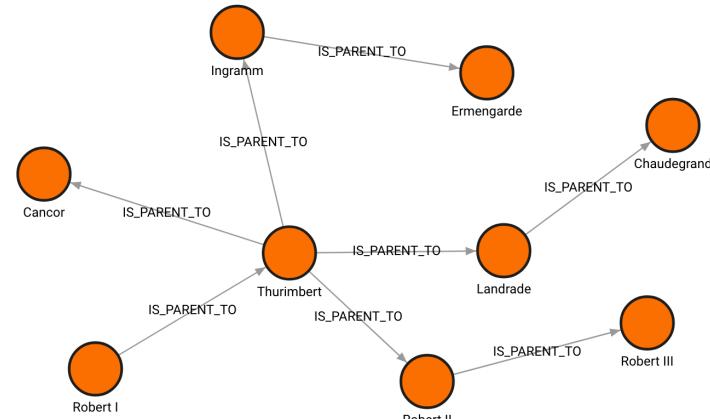


GDBMS VS. RDBMS: ANFRAGEN

- Anfragen in GDBMS durch bspw. **Cypher**: je nach Anfrage, besser verständlich und somit leichter änderbar/wartbar (ASCII-Art)
- *Beispiel*: Finde alle Nachkommen von Thurimbert

SQL

```
WITH RECURSIVE descendants AS
(
  SELECT person
  FROM tree
  WHERE person= 'Thurimbert'
  UNION ALL
  SELECT t.person
  FROM descendants d, tree t
  WHERE t.parent=d.person
)
SELECT * FROM descendants;
```



Cypher

```
MATCH path=(n:Person {name: 'Thurimbert'})-[*]->(m)
RETURN m;
```

Quelle: <https://memgraph.com/blog/graph-database-vs-relational-database>



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?
- ...

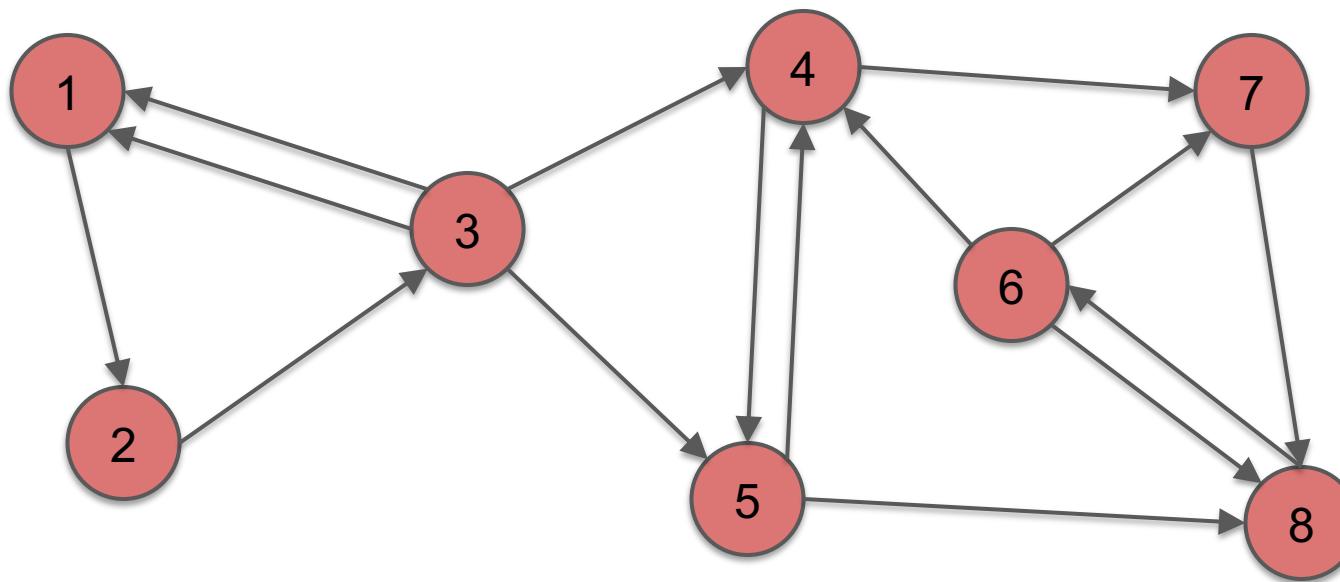


INHALTSVERZEICHNIS

- Einführung
- Vergleich mit relationalen Datenbanksystemen
- Repräsentation von Graphdaten
- Anfragesprachen
- Anwendungen
- Beispiel: Neo4j
- Das Raft Protokoll



GRAPHREPRÄSENTATION



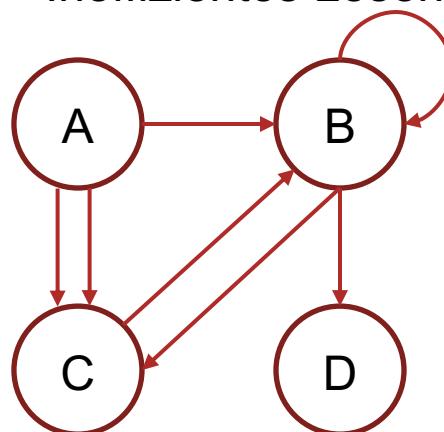
$$Graph = (Vertices, Edges)$$

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



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)



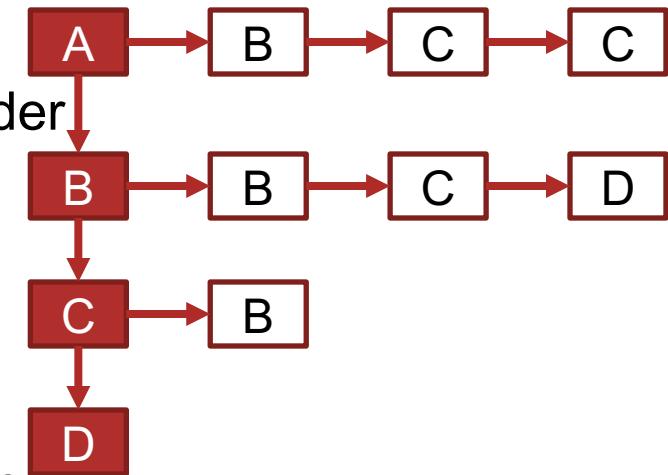
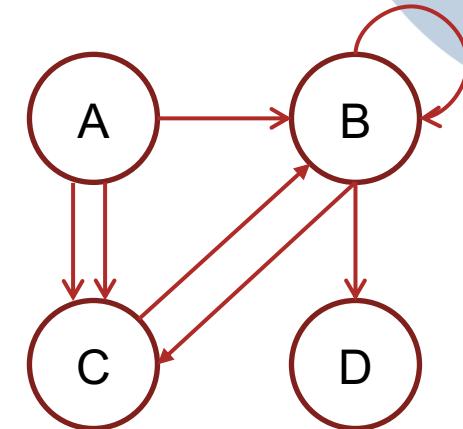
	A	B	C	D
A	0	1	2	0
B	0	1	1	1
C	0	1	0	0
D	0	0	0	0



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
 - Aufwändiger Implementierung als Adjazenzmatrix
 - Prüfung auf spezifische Kante aufwändiger





PROPERTY GRAPH

- Verschiedene Architekturen möglich
- Neo4j: Store Files [N4J], 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“

Node Store Entry

in-use	next edge	next property	labels
0	1	5	9

14 B



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
 - Referenz auf dynamische Dateien (für Strings und Arrays)

Property Store Entry

next property	previous property	property blocks	
0	5	9	41 B

Property Block

property key	property type	property value	
0	4	5	8 B



PROPERTY GRAPH

Relationship Store Entry

in-use	start node	end node	edge label	start node previous edge	start node next edge	end node previous edge	end node next edge	next property
0	1	5	9	13	17	21	25	29

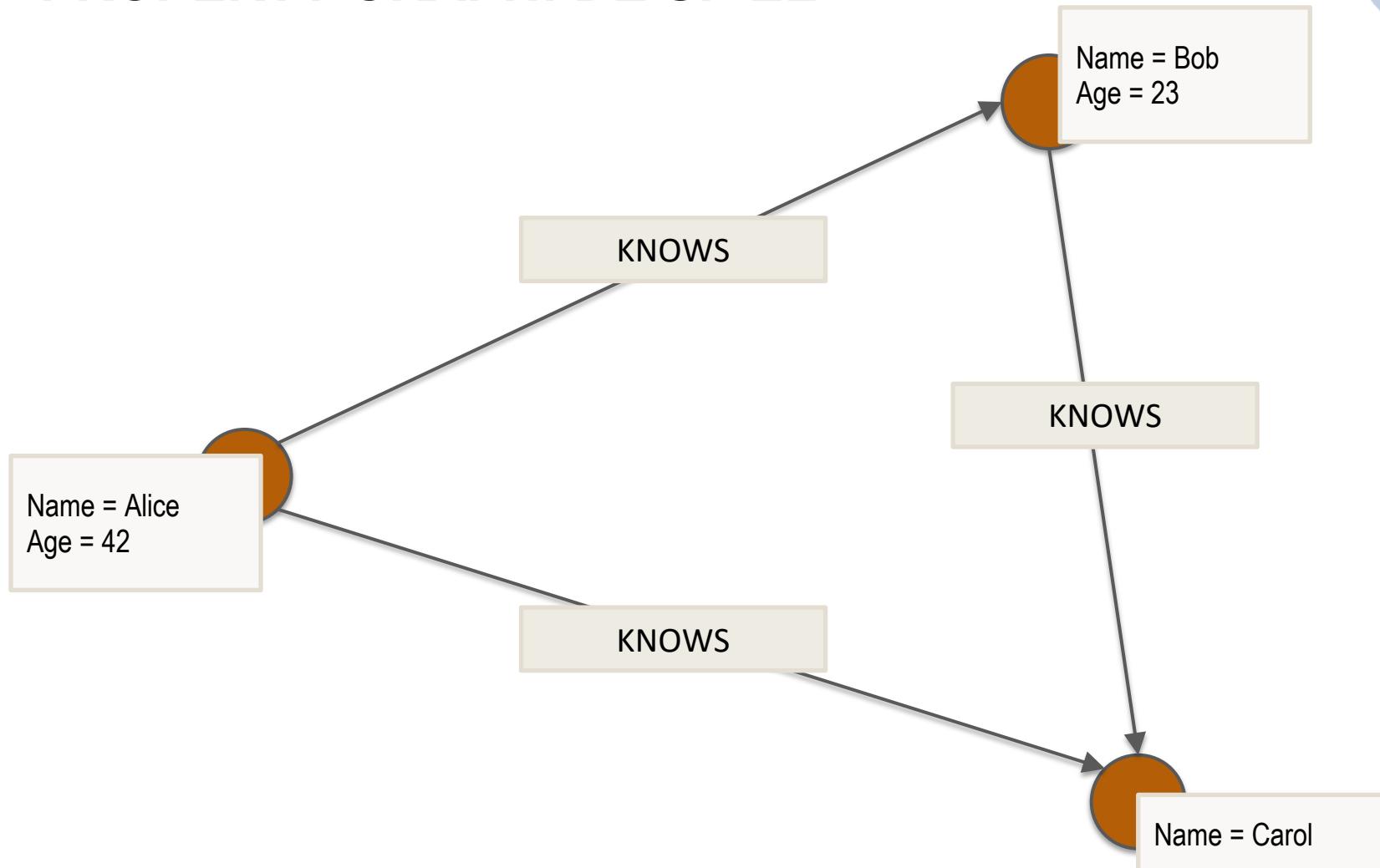
33 B

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 $O(\text{Anzahl der inzidenten Kanten})$
 - Leistungsvorteil bei Traversierung des Graphen, der Suche nach Kanten (mit bestimmten Label) und beim Einfügen/Löschen von Kanten

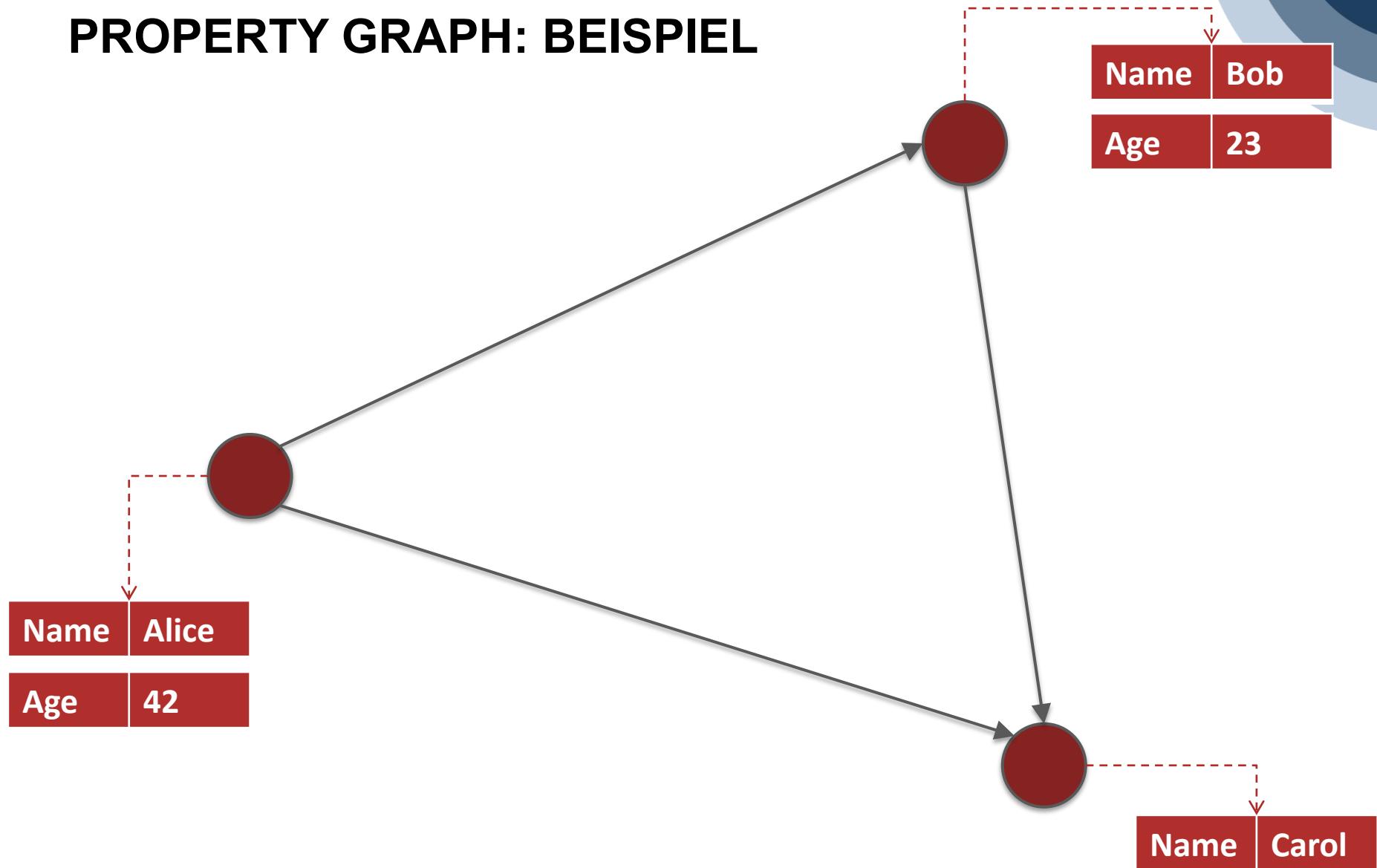


PROPERTY GRAPH: BEISPIEL





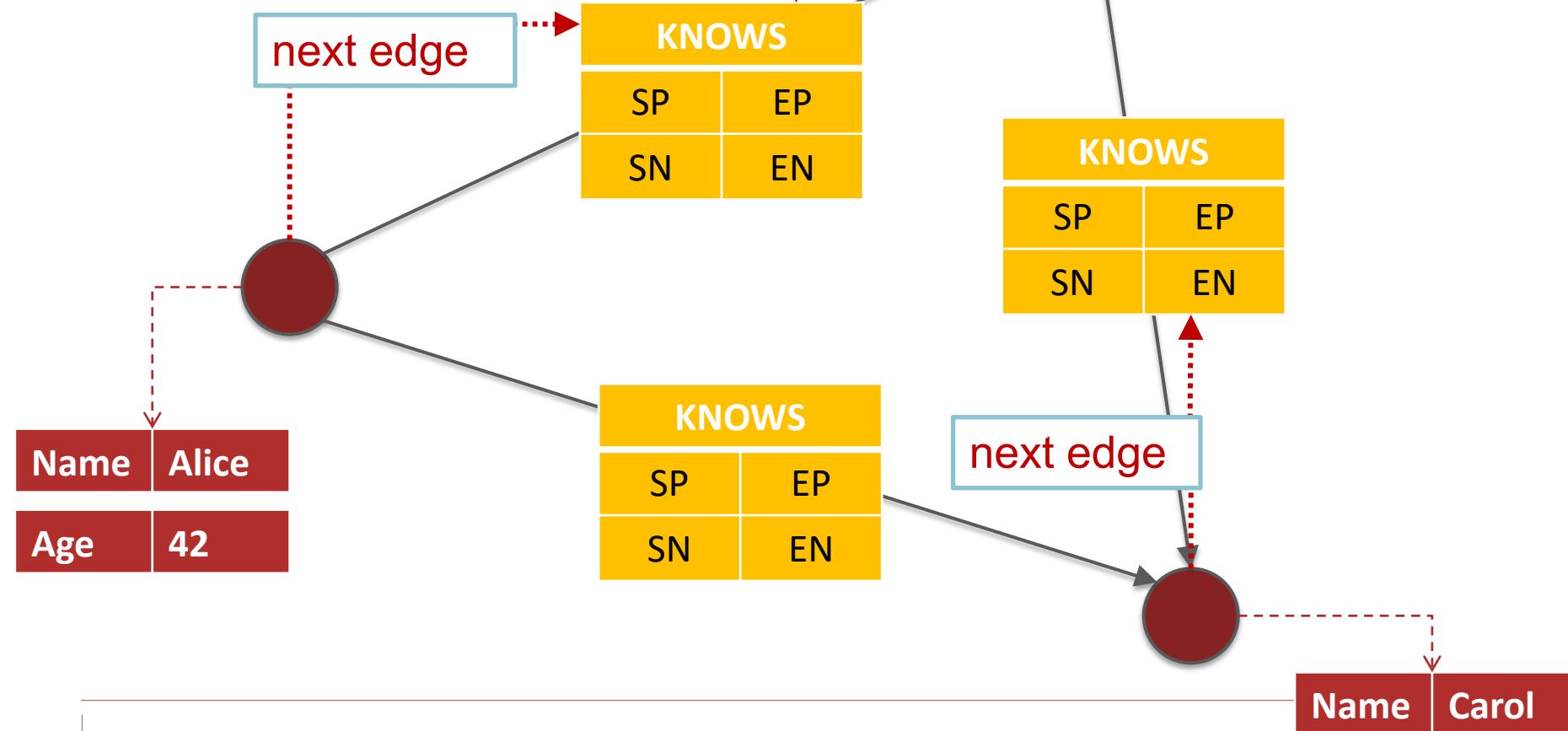
PROPERTY GRAPH: BEISPIEL





PROPERTY GRAPH: BEISPIEL

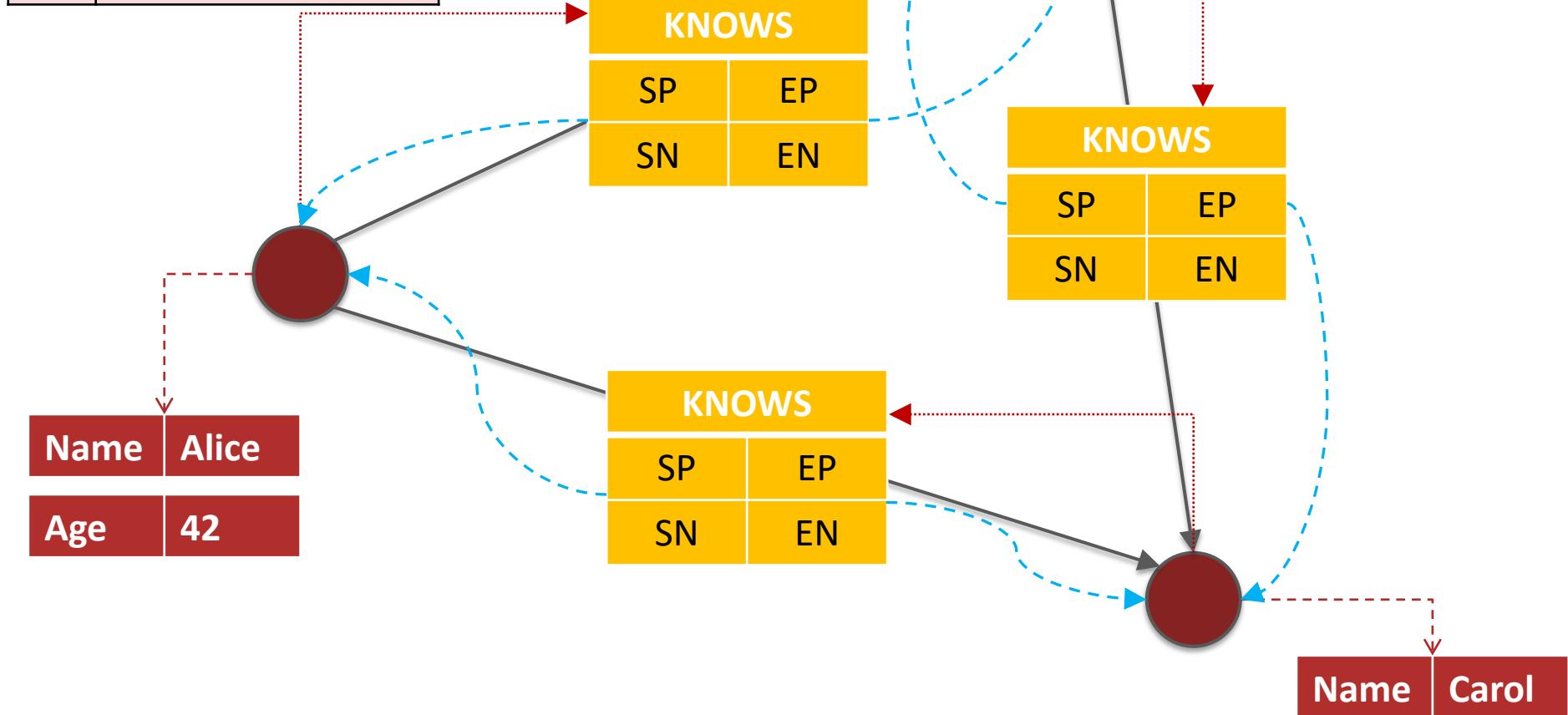
SP	Start Node Previous
SN	Start Node Next
EP	End Node Previous
EN	End Node Next





PROPERTY GRAPH: BEISPIEL

SP	Start Node Previous
SN	Start Node Next
EP	End Node Previous
EN	End Node Next

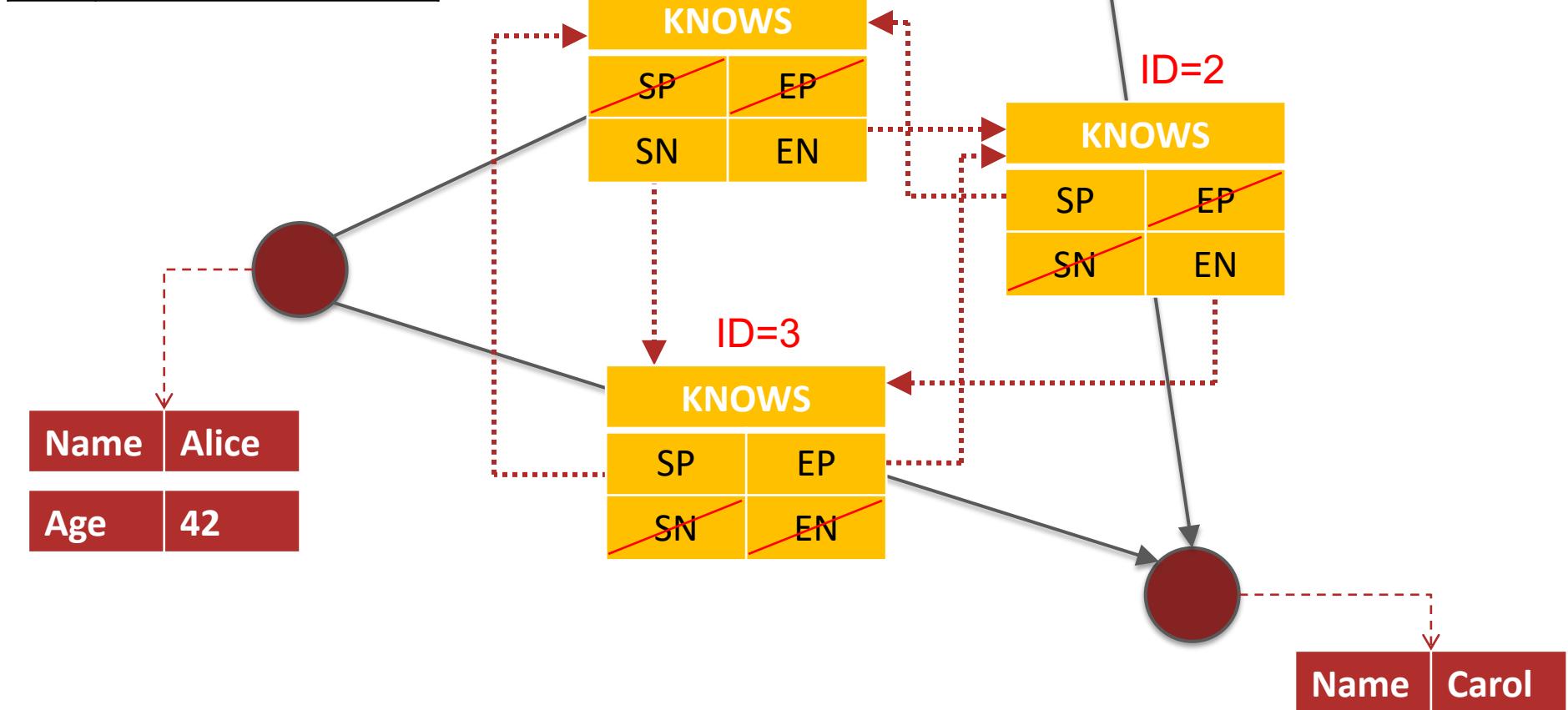




PROPERTY GRAPH: BEISPIEL

SP	Start Node Previous
SN	Start Node Next
EP	End Node Previous
EN	End Node Next

Name	Bob
Age	23





ZUSAMMENFASSUNG

- Verstärktes Aufkommen stark vernetzter umfangreicher und heterogener Daten
- Property Graph Modell als generisches Modell für vernetzte Daten
- GDBMS vs. RDBMS: Einsatz abhängig von Art der Daten und Anfragen
- Natives GDBMS ermöglicht effizienten lokalen Zugriff auf Beziehungen: Indexfreie Adjazenz
- Nächste Vorlesung:
 - Anfragesprachen
 - Anwendungen



LITERATUR

- [Vuk15] Vukotic A, Watt N, Abedrabbo T, Fox D, Partner J. Neo4j in action. Shelter Island: Manning; 2015. https://manning-content.s3.amazonaws.com/download/5/392c9aa-4c64-4c6d-a072-fcf6b73d4ef1/Neo4jinAction_CH01.pdf
- [N4J] <https://neo4j.com/developer/kb/understanding-data-on-disk/>
- [NV] <https://neo4j.com/docs/cypher-manual/current/indexes/semantic-indexes/vector-indexes/>
- [Ang23] Renzo Angles, Angela Bonifati, et al. 2023. PG-Schema: Schemas for Property Graphs. Proc. ACM Manag. Data 1, 2, Article 198 (June 2023), 25 pages. <https://doi.org/10.1145/3589778>



ABSTIMMUNG PRAKTISCHE ÜBUNG NEO4J

- Gruppeneinteilung von heute bis 25.04. im Moodle
 - Alleinige Bearbeitung möglich
- Aufgabenstellung + Beginn der Bearbeitung ab 25.04.