

7. Big Data und NoSQL-Datenbanken

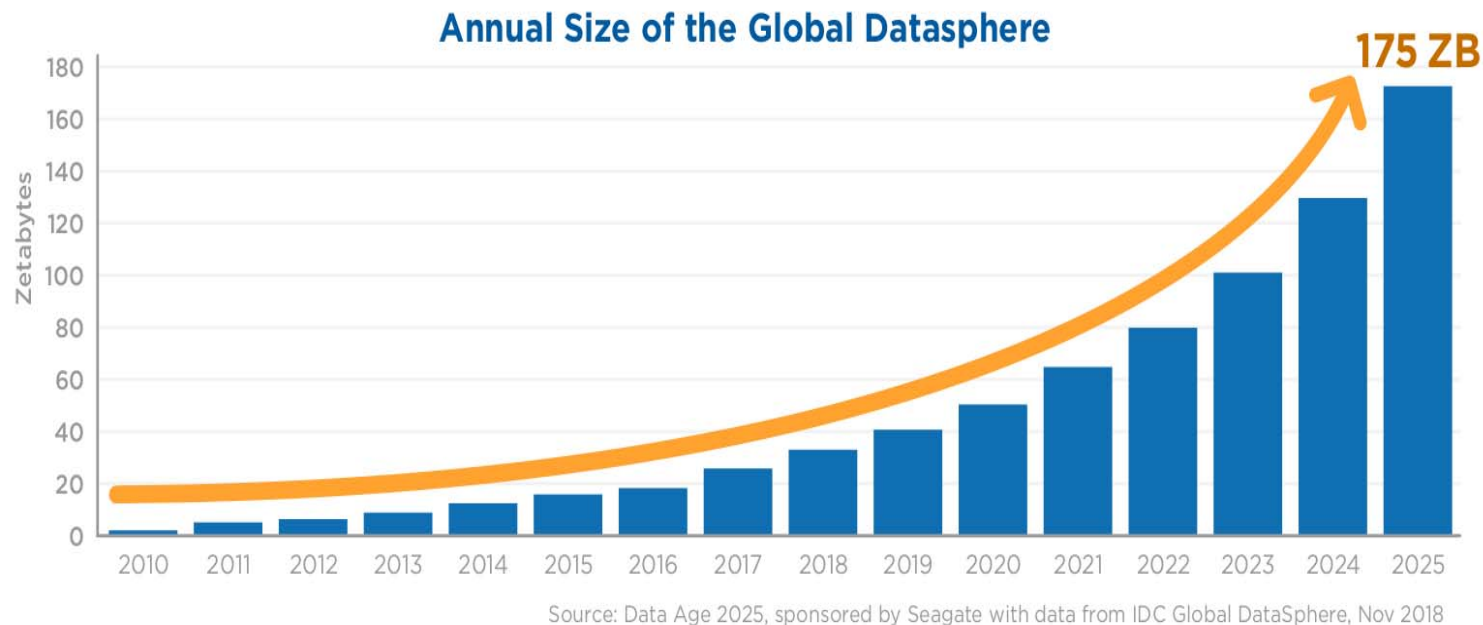
- Motivation Big Data
 - Herausforderungen
 - Einsatzbereiche
- Systemarchitekturen für Big Data Analytics
 - Analyse-Pipeline
 - Hadoop, MapReduce, Spark/Flink
- NoSQL-Datenbanken
 - Eigenschaften
 - Document Stores (MongoDB)
 - Graph-Datenbanken (neo4J)
- parallele/verteilte Graphanalysen / Gradoop



Wie groß ist Big Data?

■ Big wandelt sich schnell

- Gigabytes, Terabytes (10^{12}), Petabytes (10^{15}),
- Exabytes (10^{18}),
- Zettabytes (10^{21}),
- Yottabytes (10^{24}),
- Brontobytes (10^{27}),

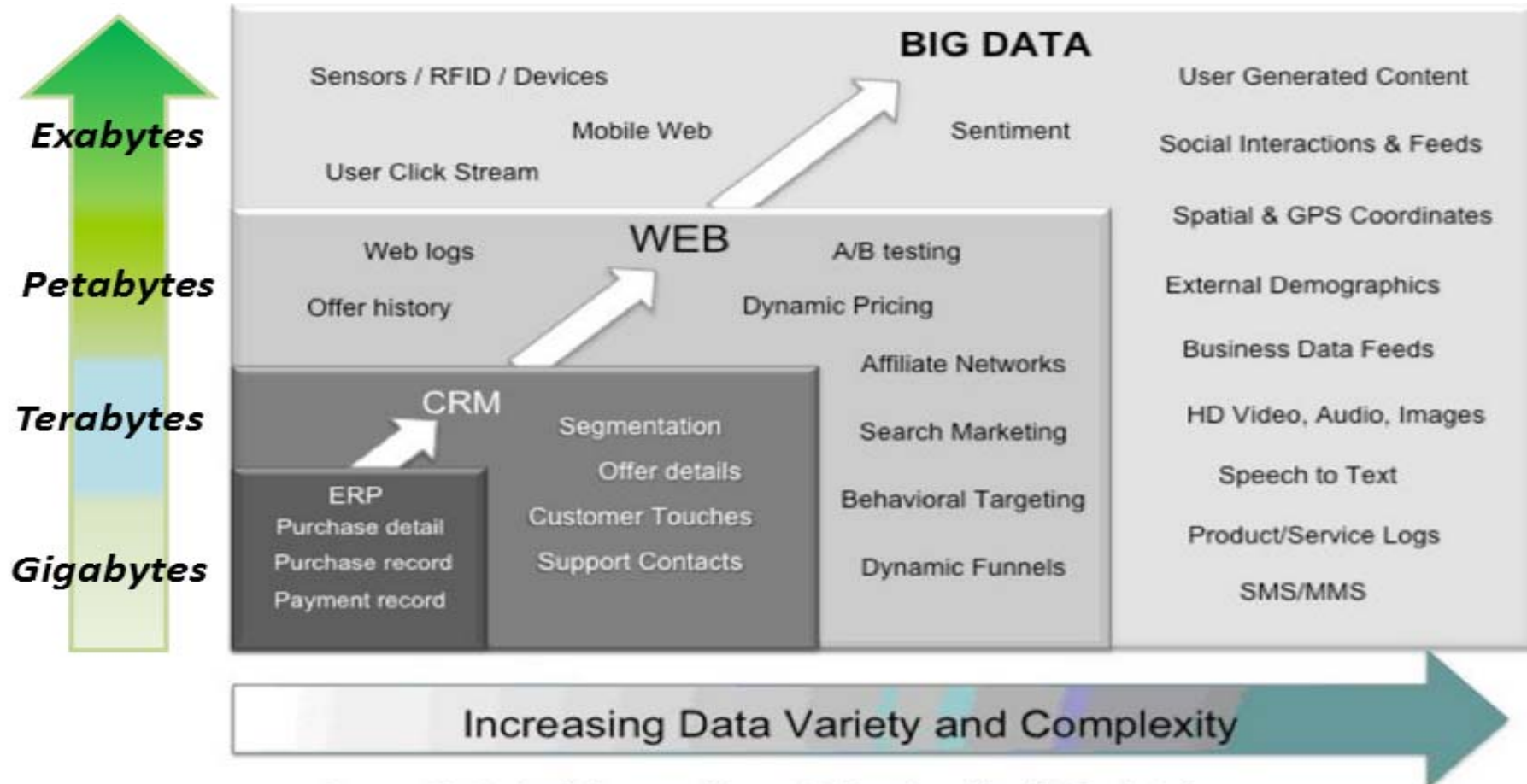


Welche Art von Daten?

- mobile Endgeräte (Smartphones, Watches, ...):
Multimedia-Daten (Fotos, Videos), Bewegungsdaten, Nachrichten ...
- Unternehmensdaten
 - strukturierte Daten (Datenbanken, Data Warehouses) mit Angaben zu Personal, Kunden, Bestellungen, Rechnungen ...
 - Dokumente, E-Mails
- Web-Daten
 - Web-Seiten + Multimedia-Inhalte (Videos, Fotos, ...)
 - Click Logs
- soziale Netzwerke / Kommunikationsplattformen (Facebook, Twitter, LinkedIn, ...)
 - Nutzer, Beziehungen, Nachrichten, Multimedia-Inhalte
- wissenschaftliche Daten z. B. in Klimaforschung, Experimentalphysik, Lebenswissenschaften und Sozialwissenschaften (Digital Humanities)
- Sensor-Daten / eingebettete Systeme
 - vernetzte Produktion/Fertigung (Industrie 4.0)
 - „intelligentes“ Haus, Verkehrsüberwachung, ...



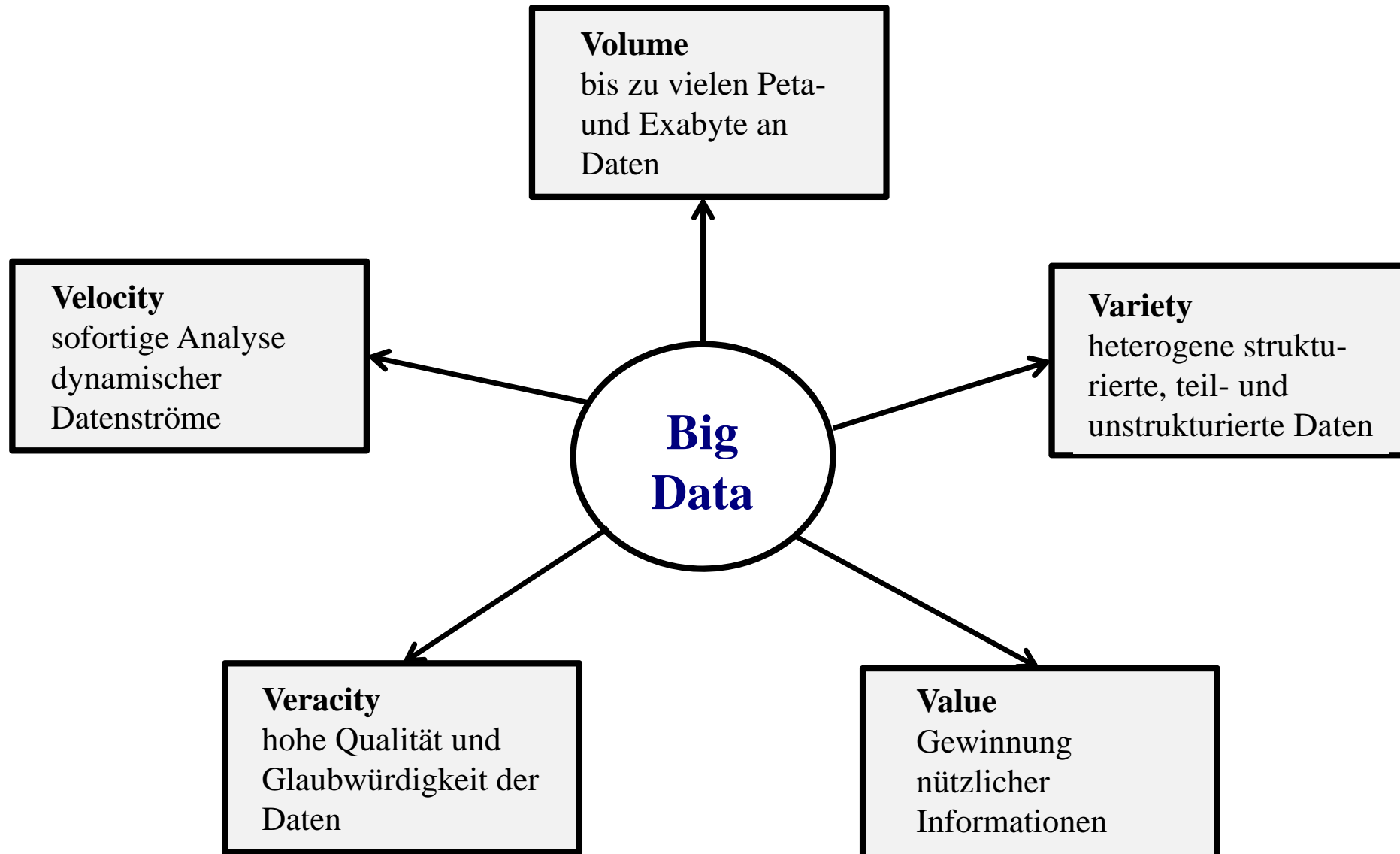
Datenzunahme



Source: Contents of above graphic created in partnership with Teradata, Inc.



Big Data Challenges



Potenziale für Big-Data-Technologien

- Daten sind Produktionsfaktor: „data is the new oil“
 - ähnlich Betriebsmitteln und Beschäftigten
 - essenziell für viele Branchen und Wissenschaftsbereiche
- valide Grundlage für zahlreiche Entscheidungsprozesse
 - Vorhersage/Bewertung/Kausalität von Ereignissen
- kurzfristige Analysen von Realdaten im Geschäftsleben
 - Empfehlungsdienste (Live Recommendations)
 - Analyse/Optimierung von Produktionsprozessen, Lieferketten, etc.
- verbesserte Analysen in zahlreichen wissenschaftlichen Anwendungen



Anwendungsdomänen für Big Data Analytics

Smarter Healthcare



Multi-channel sales



Finance



Log Analysis



Homeland Security



Traffic Control



Telecom



Search Quality



Manufacturing



Trading Analytics



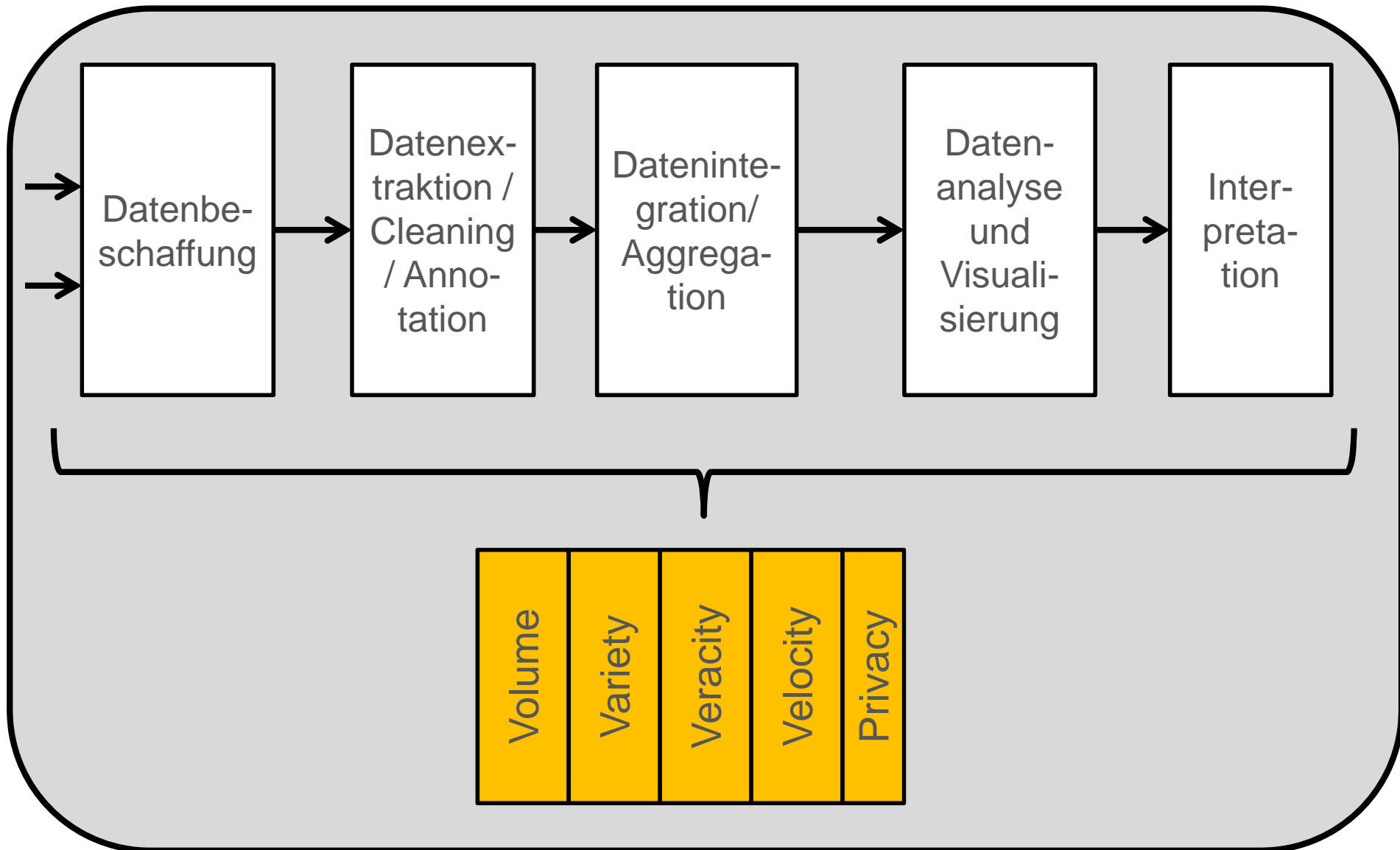
Fraud and Risk



Retail: Churn, NBO



Big Data Analyse-Pipeline



Technologie-Trends

- massiv skalierbare Cloud-Architekturen (Shared Nothing)
- Frameworks zur automatischen Parallelisierung datenintensiver Aufgaben (Hadoop, MapReduce, Apache Spark/Flink)



- Parallelverarbeitung auf unterschiedlichen Stufen
 - Data Center (Cluster)
 - Multi-core server
 - Spezialprozessoren: GPUs /FPGAs
- In-Memory Datenbanken / Data Warehouses
- Spezialsysteme für Streaming-Daten, Graph-Daten, ...

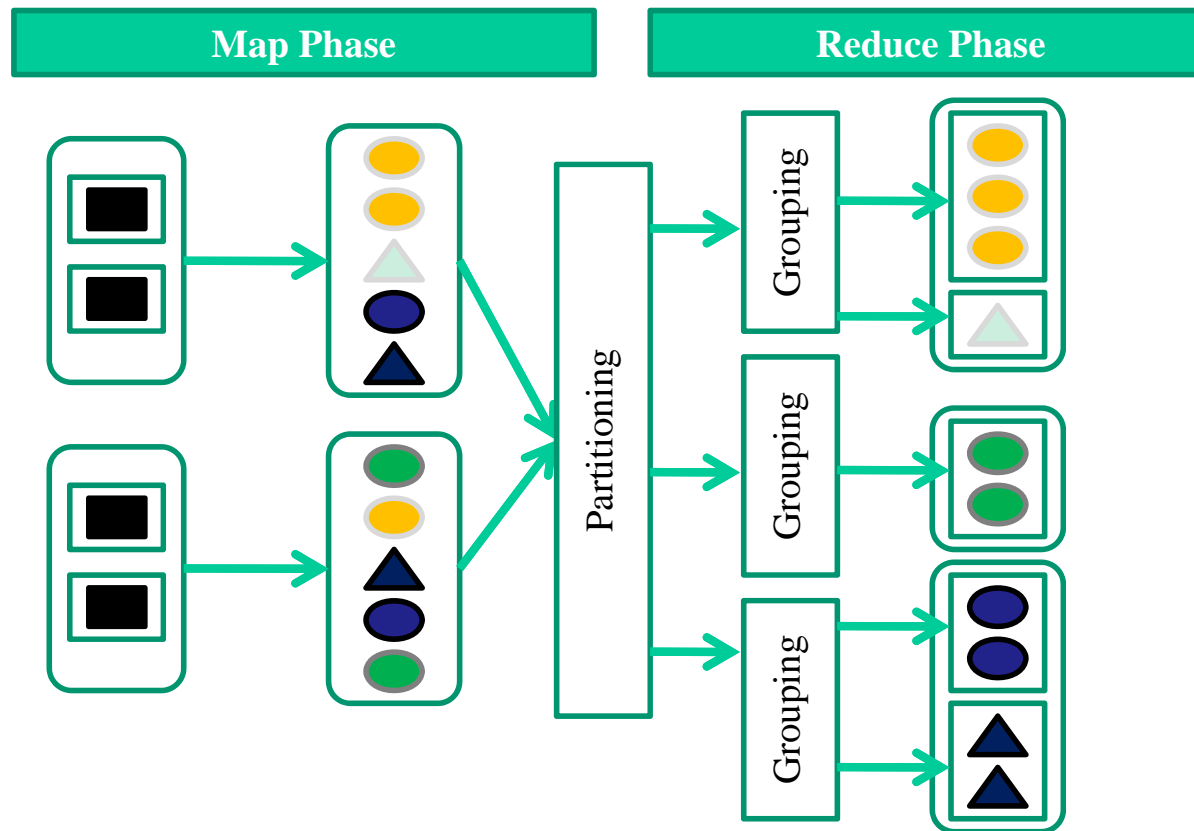
MapReduce

- Framework zur automatischen Parallelisierung von Auswertungen auf großen Datenmengen
 - Entwicklung bei Google
 - populäre Open-Source-Implementierung: Hadoop
- Nutzung v.a. zur Verarbeitung riesiger Mengen teilstrukturierter Daten in einem verteilten Dateisystem
 - *Konstruktion Suchmaschinenindex*
 - *Clusterung von News-Artikeln*
 - *Spam-Erkennung ...*

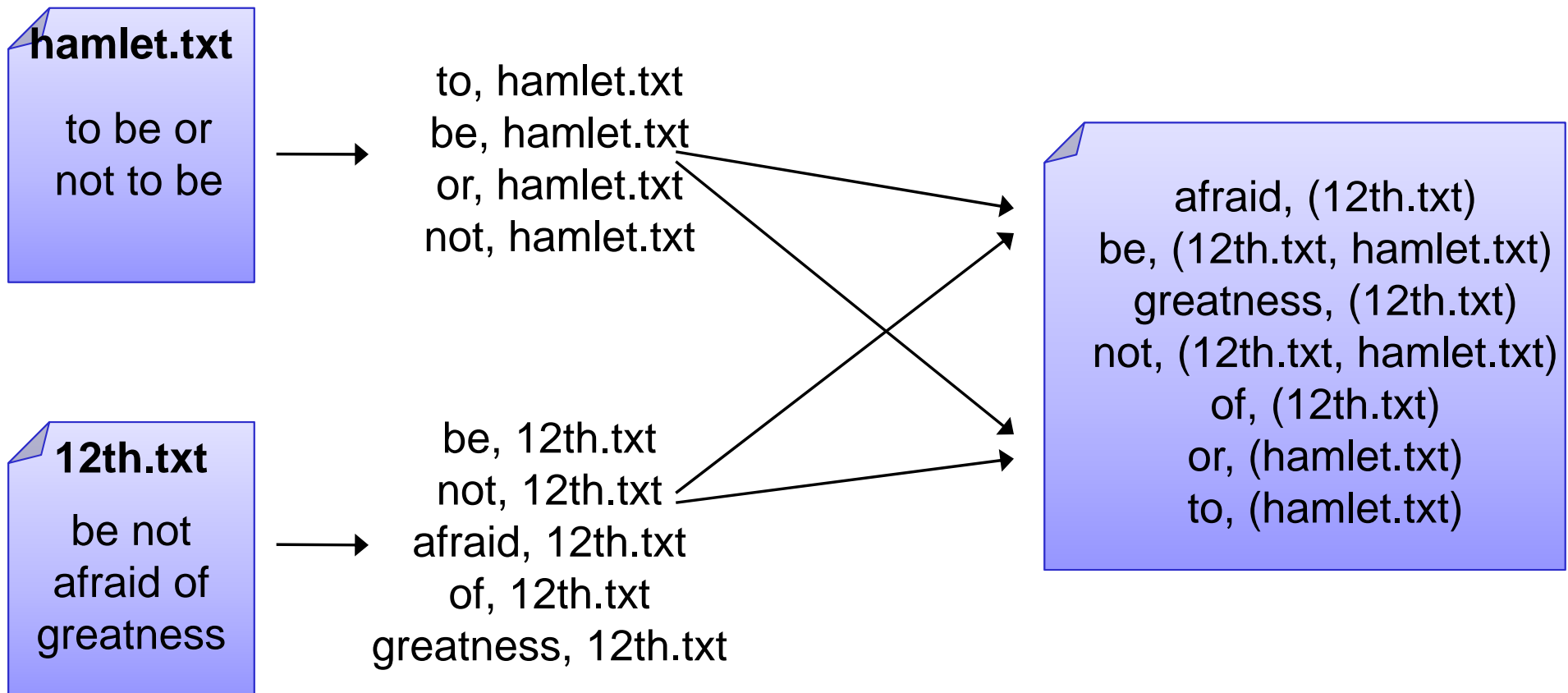


MapReduce

- Verwendung zweier Funktionen: **Map** und **Reduce**
- **Map**-Anwendung pro Eingabeobjekt zur Erzeugung von Key-Value Paaren
 - jedes Key-Value-Paar wird einem Reduce-Task zugeordnet
- **Reduce**-Anwendung für jede Objektgruppe mit gleichem Key

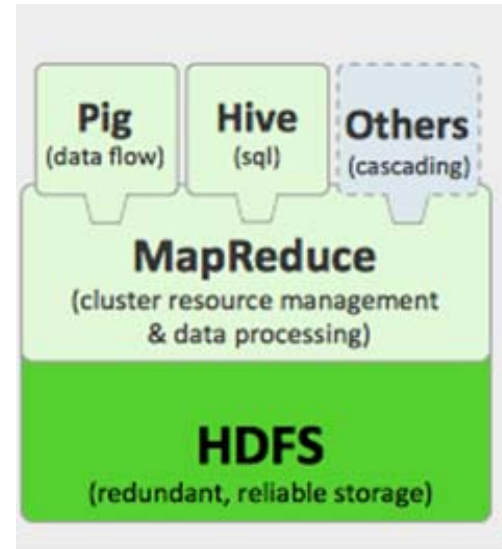


MR-Beispiel: Generierung Text-Index



Hadoop-Ökosystem für Big Data

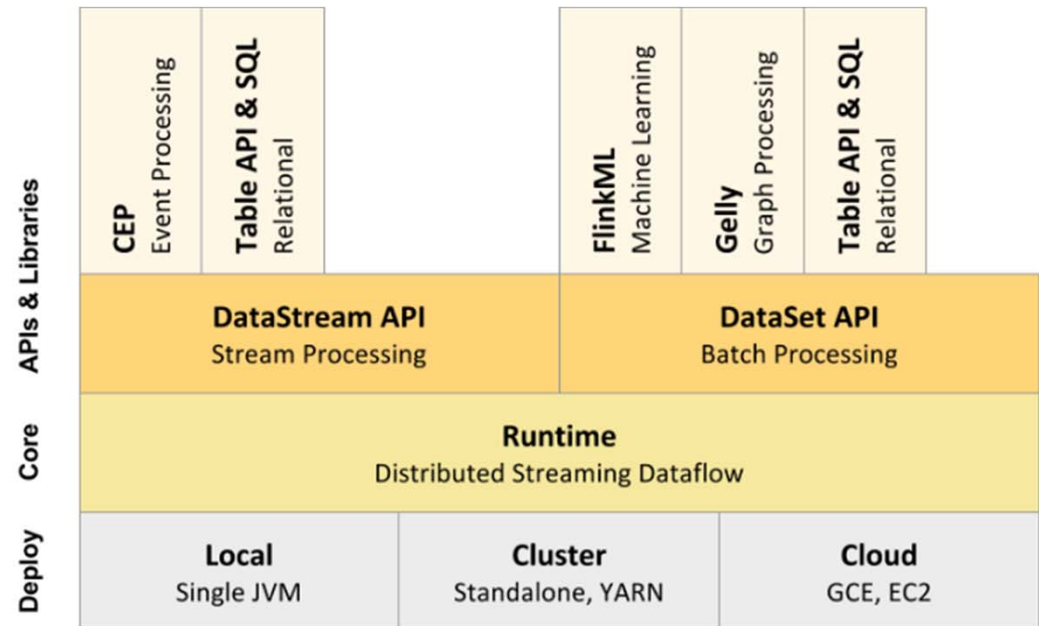
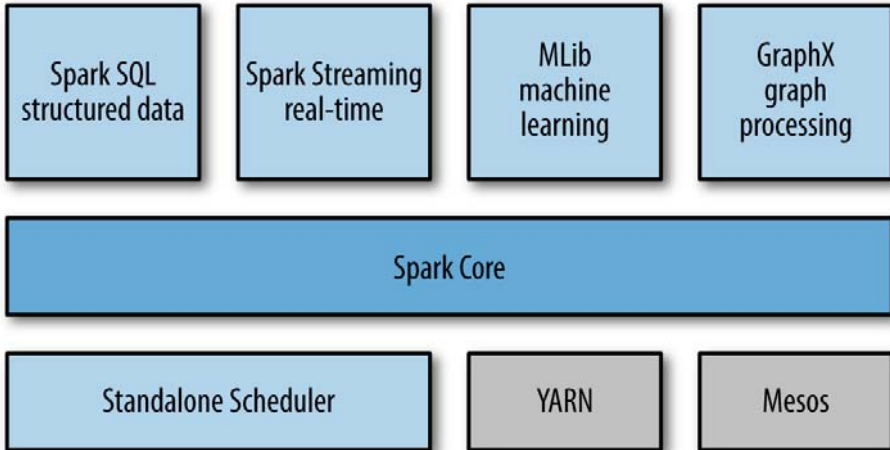
Hadoop 1.0



Hadoop 2.x



Spark vs Flink



NoSQL-Datenbanken

Not
Only SQL



Relationale Datenbanken: Pros and Cons

- universelle Verbreitung und auf absehbare Zeit ungefährdet für die meisten DB-Anwendungen
 - SQL: mächtige, standardisierte, mächtige (deklarative) Query-Sprache
 - ACID
 - reife Technologie
 - automatische Parallelisierung ...
- Schema-getrieben
 - weniger geeignet für semi-strukturierte Daten
 - zu starr für irreguläre Daten, häufige Änderungen
- relativ hohe Kosten, v.a. für Parallele DBS (kein Open-Source System)
- Skalierbarkeitsprobleme für Big Data (Web Scale)
 - Milliarden von Webseiten
 - Milliarden von Nutzern von Websites und sozialen Netzen
- ACID / strenge Konsistenz nicht immer erforderlich



- Definition von *www.nosql-database.org*
 - Datenbanken der nächsten Generation die meist
 - nicht-relational
 - verteilt,
 - open-source und
 - horizontal (auf große Datenmengen) skalierbar sind
- ursprünglicher Fokus: moderne “web-scale” Datenbanken
- Entwicklung seit ca. 2009
- weitere Charakteristika:
 - schema-frei, Datenreplikation, einfache API,
 - eventually consistent / BASE (statt ACID)
- Koexistenz mit SQL
 - “NoSql” wird als “**N**ot **o**nly **S**ql“ interpretiert

Arten von NoSQL-Systemen

■ Key Value Stores

- Amazon Dynamo, Voldemort, Membase, Redis ...
- Speicherung eines Werts (z.B. BLOB) pro nutzer-definiertem Schlüssel bzw. Speicherung von Attribut/Wert-Paaren
- nur einfache key-basierte Lookup/Änderungs-Zugriffe (get, put)

■ erweiterte Record-Stores / Wide Column Store

- Google BigData / Hbase, Hypertable, Cassandra ...
- Tabellen-basierte Speicherung mit flexibler Erweiterung um neue Attribute

■ Dokument-Datenbanken

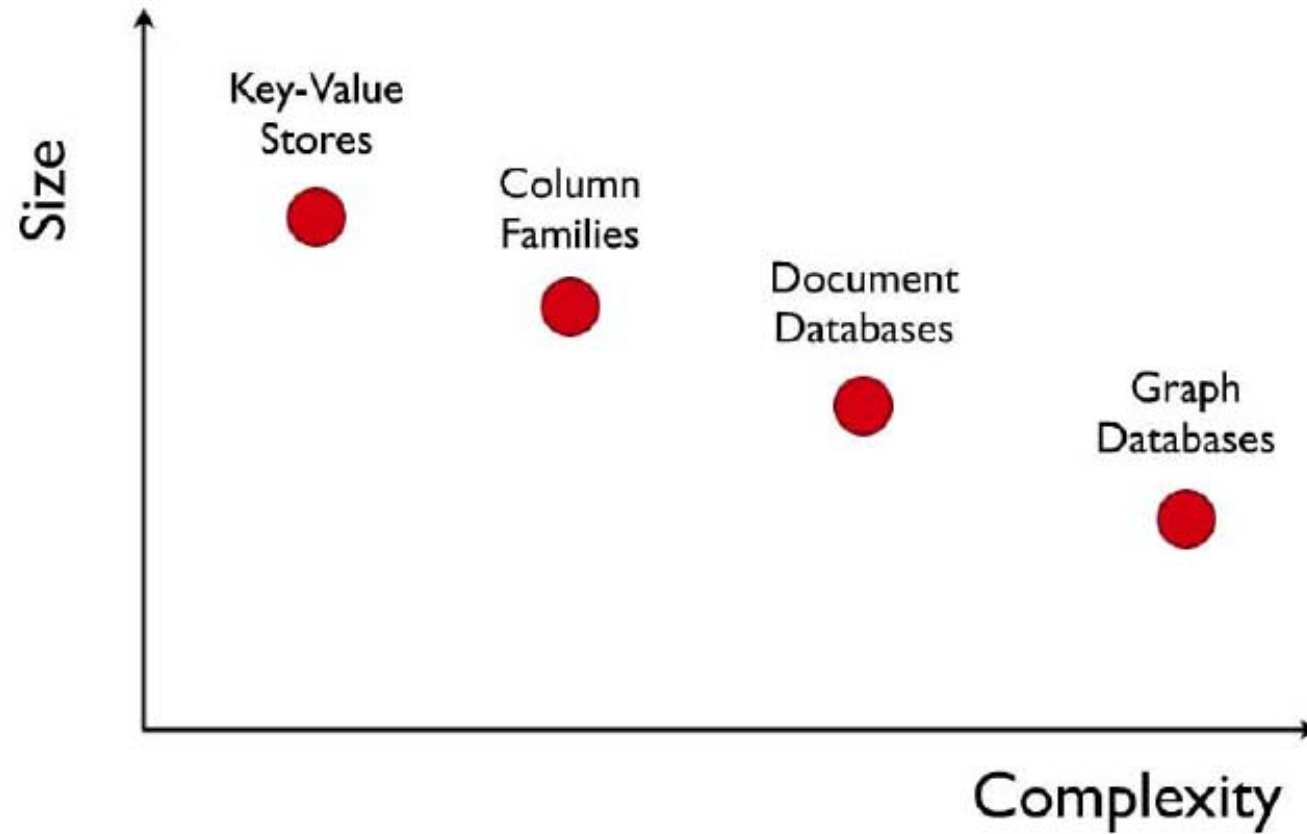
- CouchDB, MongoDB ...
- Speicherung semistrukturierter Daten als Dokument (z.B. JSON)

■ Graph-Datenbanken

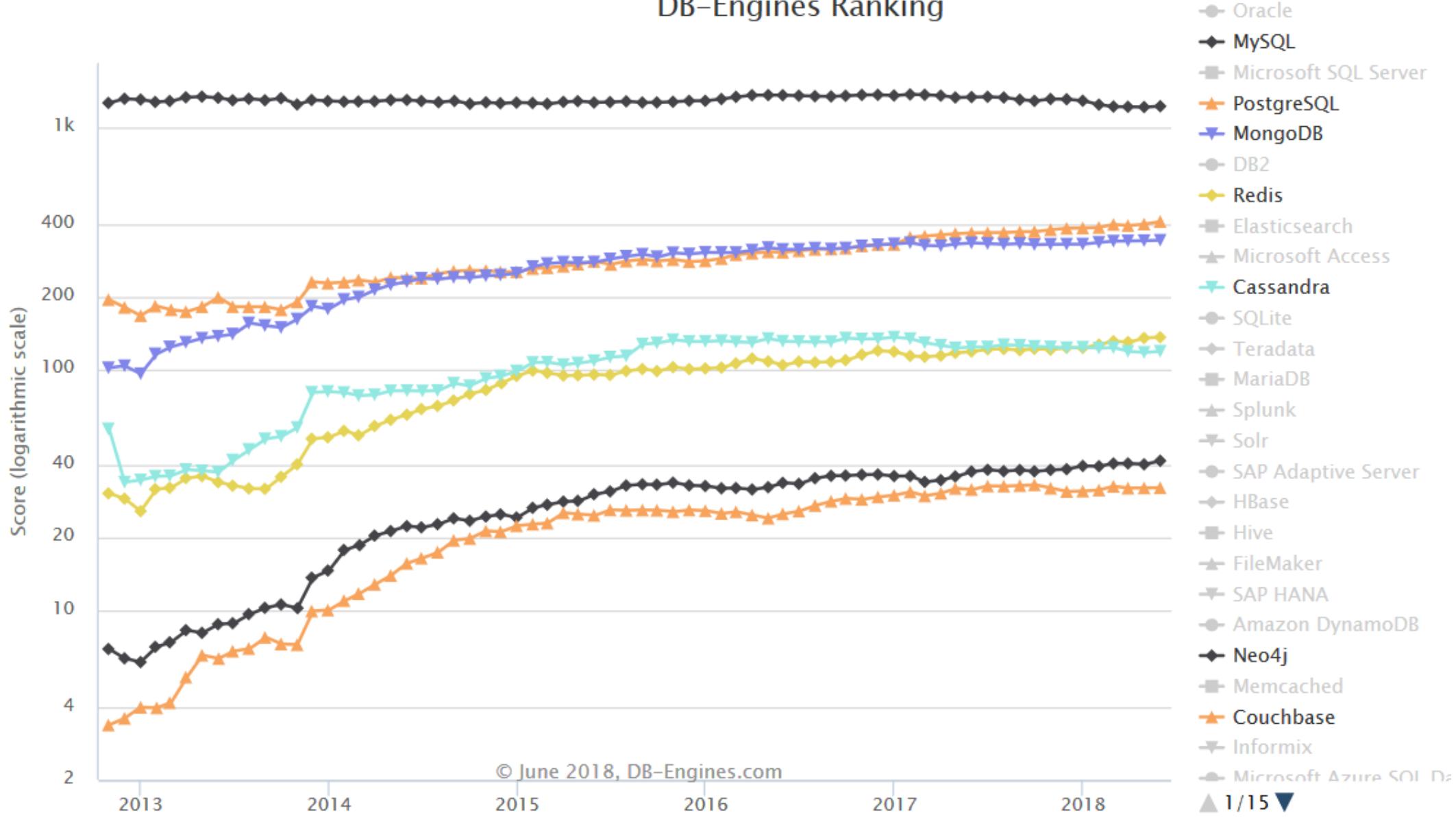
- Neo4J, OrientDB ...
- Speicherung / Auswertung großer Graph-Strukturen



Grobeinordnung NoSQL-Systeme



DB-Engines Ranking



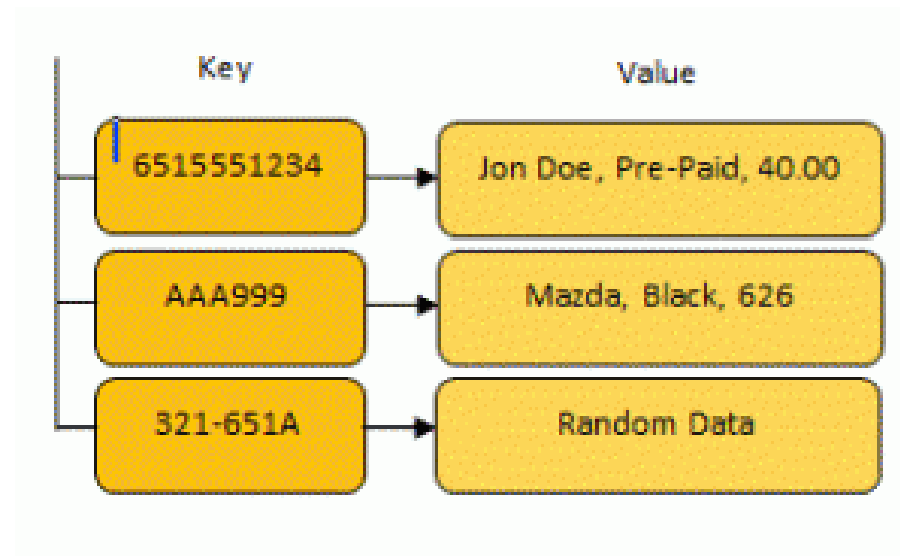
© June 2018, DB-Engines.com

https://db-engines.com/en/ranking_trend



Key-Value Stores

- Speicherung von Schlüssel-Werte-Paaren
 - im einfachsten Fall bleiben Werte systemseitig uninterpretiert (BLOBs)
 - flexibel, kein Schema
 - günstig für wenig strukturierte Inhalte bzw. stark variable Inhalte, z.B. Twitter-Nachrichten, Webseiten etc.
 - keine Verwaltung von Beziehungen



- schnelle Lese/Schreibzugriffe über Schlüssel
 - put (key, value)
 - get (key)
- keine komplexen Queries (z.B. Bereichsabfragen)

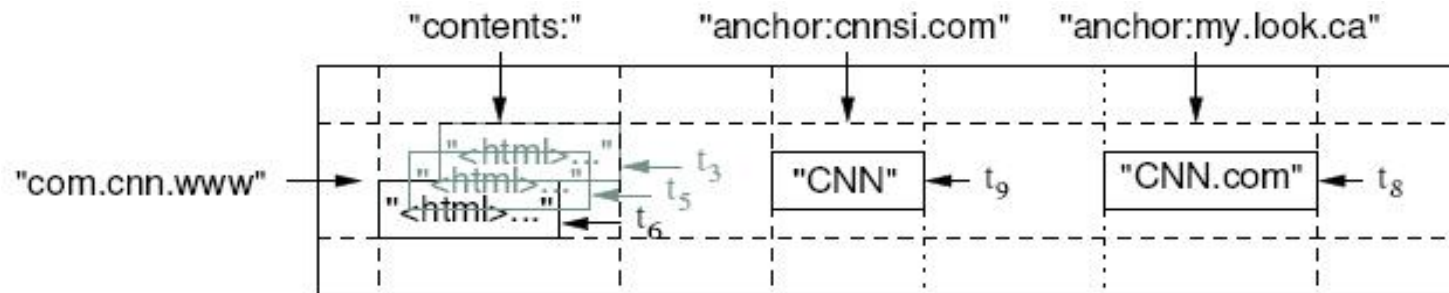
Spaltenorientierte Key-Value Stores

- Vorbild: Google BigTable

- Clones: Hbase, Cassandra, ...

- Ziele (BigTable):

- Milliarden von Zeilen, Millionen von Spalten, Versionierung (z.B. für Web-Seiten)
- einfache Erweiterbarkeit um Spalten (innerhalb vordefinierter Spaltenfamilien)
- mehrdimensionale Keys: Zeilen- und Spalten-Schlüssel, Versionsnr



Row Key	Time Stamp	Column <i>contents</i>	Column Family <i>anchor</i>	
"com.cnn.www"	T9		anchor:cnnsi.com	CNN
	T8		anchor:my.look.ca	CNN.COM
	T6	"<html>.."		
	T5	"<html>.."		



Spaltenorientierte Key-Value Stores (2)

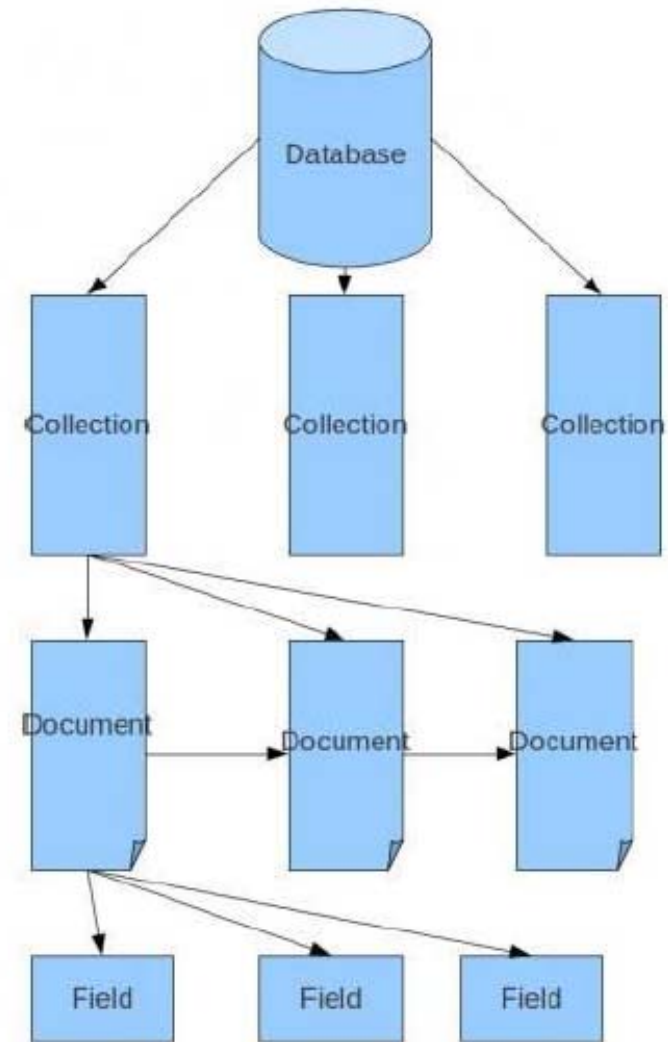
- weiteres Beispiel

Row Key	Column Family: "User"	Column Family: "Article"
"userid1"	name: Username, value: uname1 name: Email, value: uname1@abc.com name: Tel, value: 123-4567	
"userid2"	name: Username, value: uname2 name: Email, value: uname2@abc.com name: Tel, value: 123-4568	name: ArticleId, value:userid2-1 name: ArticleId, value:userid2-2 name: ArticleId, value:userid2-3
"userid3"	name: Username, value: uname3 name: Email, value: uname3@abc.com name: Tel, value: 123-4569	

- hochskalierbar, replizierte Speicherung
- einfache Get/Put-Operationen mit schnellen Zugriffszeiten (Hashing)



- zunehmend Verbreitung findender **Dokumenten-Store**
 - open source-Version verfügbar
 - JSON-Dokumente, gespeichert als BSON (Binary JSON)
- DB besteht aus **Kollektionen** von Dokumenten
- einfache **Anfragesprache**
 - Indexierung von Attributen möglich
 - Map/Reduce-Unterstützung
- **Skalierbarkeit und Fehlertoleranz**
 - Skalierbarkeit durch horizontale Partitionierung der Dokumentenkollektionen unter vielen Knoten (“Sharding”)
 - automatische Replikation mit Konsistenzwahrung
- kein **ACID**, z.B. bzgl Synchronisation
 - Änderungen nur bzgl einzelner Dokumente atomar



Beispiel: relational vs. dokumentenorientiert

RDB:

STUDENTS	
sno	name
1	Peter
2	Tom

POSTS		
postID	topic	pdate
p1	NoSQL	01-09-2012
p2	RelationalDB	02-09-2012

COMMENTS				
commentID	postID	sno	content	cdate
c1	p1	1	Excellent	02-09-2012
c2	p1	2	I do not understand.	03-09-2012
c3	p1	1	This is a good idea!	04-09-2012

MongoDB:

```
{topic: 'NoSQL',
 pdate: 01-09-2012,
 comments: [{name: 'Peter',
             content: 'Excellent',
             cdate: 02-09-2012},
            {name: 'Tom',
             content: 'I do not understand.',
             cdate: 03-09-2012},
            {name: 'Peter',
             content: 'This is a good idea!',
             cdate: 04-09-2012}}
}
```

- keine Beziehungen zwischen Dokumenten (-> keine Joins) sondern geschachtelte Komponenten (ähnlich NF², jedoch ohne Schemazwang)
 - Redundanz bei n:m-Beziehungen



MongoDB: Operationen

RDB		MongoDB
SELECT * FROM students;	↔	db.students.find();
INSERT INTO students ...;	↔	db.students.insert(...);
UPDATE students ...;	↔	db.students.update(...);
DELETE FROM students ...;	↔	db.students.remove(...);
...	↔	...

■ Beispiele:

```
> db.students.insert({sno: 1, name: 'Peter'});
```

```
> db.posts.insert({topic: 'NoSQL', pdate: ...});
```

```
student_a={sno:731, name:'Irving Bonce', address:  
  {street:'38 Glenpark Ave',  
    city:'Dunedin',  
    phone:4535467}};
```

```
> db.students.save(student_a);
```

```
> db.students.find();
```

```
{_id:1, sno:731, name:'Irving Bonce', address:  
  {street:'38 Glenpark Ave',  
    city:'Dunedin',  
    phone:4535467}}
```

MongoDB: Operationen (2)

```
# find the student whose name is "Stephen Hong"
```

```
> db.students.find({name: 'Stephen Hong'});
```

```
# find all students whose last name is 'Hong'
```

```
> db.students.find({name: /Hong$/});
```

```
# find all students who live in Canberra
```

```
> db.students.find({address.city: 'Canberra'});
```

```
# count the number of students
```

```
> db.students.count();
```

```
# find all students who enrolled courses either "SP03" or "SP04"
```

```
> db.students.find({courses: {$in: ['SP03', 'SP04']}},  
                  {sno: 1, name: 1});
```

```
# find all students whose ages are below 20
```

```
> db.students.find({age: {$lt: 20}});
```



Graph-Datenbanken

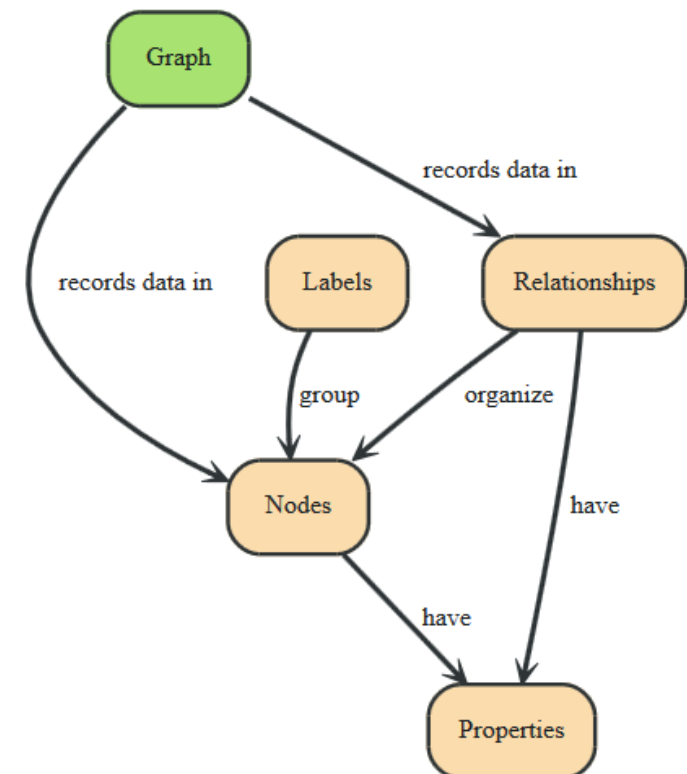
- bessere Unterstützung stark vernetzter Daten als mit Relationenmodell
 - soziale Netzwerke
 - Protein-Netzwerke
 - stark vernetzte Webdaten / Unternehmensdaten / ...
- Graph-Verwaltung im Relationenmodell oft nicht ausreichend schnell
 - viele Tabellen, viele Joins
 - langsame Traversierung von Kanten
 - langsame Umsetzung von Graph-Algorithmen
- Graph-Datenbanken
 - Graph-Datenmodell mit Gleichbehandlung von Entities und Relationships
 - Graph-Anfragesprachen
 - optimierte Graph-Operationen (z.B. finde „friends of friends“)



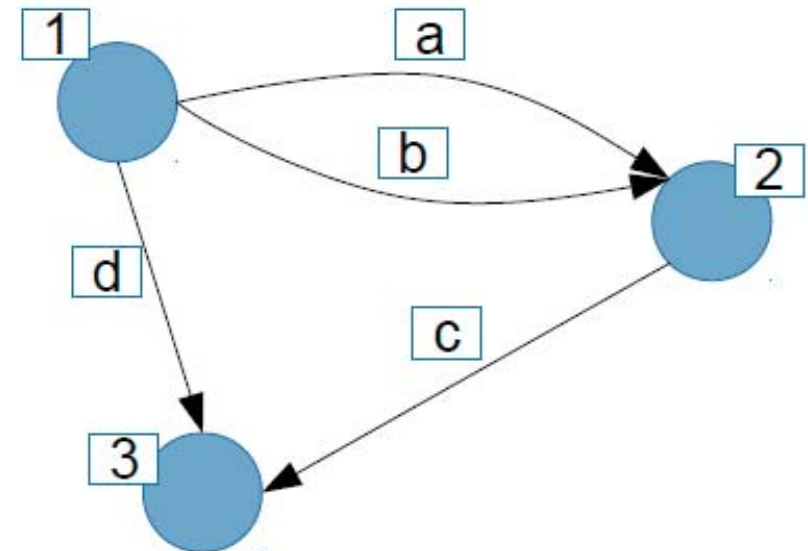
- native Graph-Datenbank von Neo Technology
 - Community Edition (open-source) + kommerzielle Varianten
 - in Java realisiert , Unterstützung weiterer Sprachen (Ruby, Python)
 - ACID-Unterstützung
 - Skalierbarkeit durch Datenreplikation

- zentrale Datenstruktur: Property-Graphen

- Knoten/Kanten haben Label (Id bzw. Typ))
- Knoten und Kanten können Properties haben
- Property: Key-Value-Paar (Key ist vom Typ String)
- gerichtete Kanten



- mehrere, gerichtete Kanten zwischen zwei Knoten möglich (gerichteter „Multigraph“)
 - Labels für Knoten/Kanten
 - Properties für Knoten/Kanten
- konstanter Aufwand zur Traversierung zu Nachbarknoten (statt Join im RM)



Query-Sprache Cypher



■ Merkmale

- Pattern Matching
- OLTP-artige Lese/Änderungsoperationen
- schnelle Traversierungen
- ACID-basierte Updates

■ Klauseln:

- **START**: Starting points in the graph.
- **MATCH**: The graph pattern to match.
- **WHERE**: Filtering criteria.
- **RETURN**: What to return.
- **CREATE**: Creates nodes and relationships.
- **DELETE**: Removes nodes, relationships and properties.
- **SET**: Set values to properties.
- **FOREACH**: Performs updating actions once per element in a list.
- **WITH**: Divides a query into multiple, distinct parts.

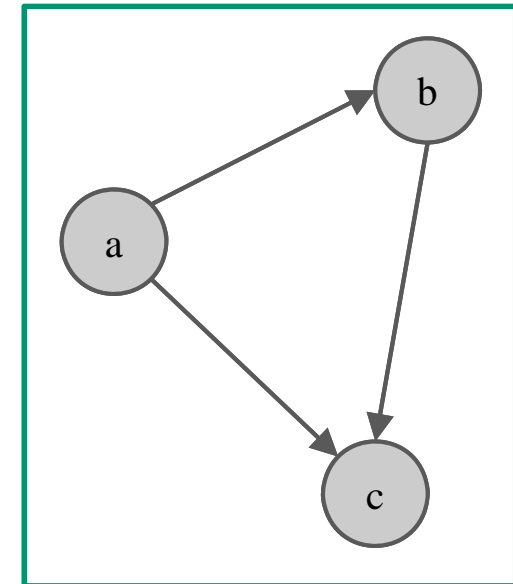
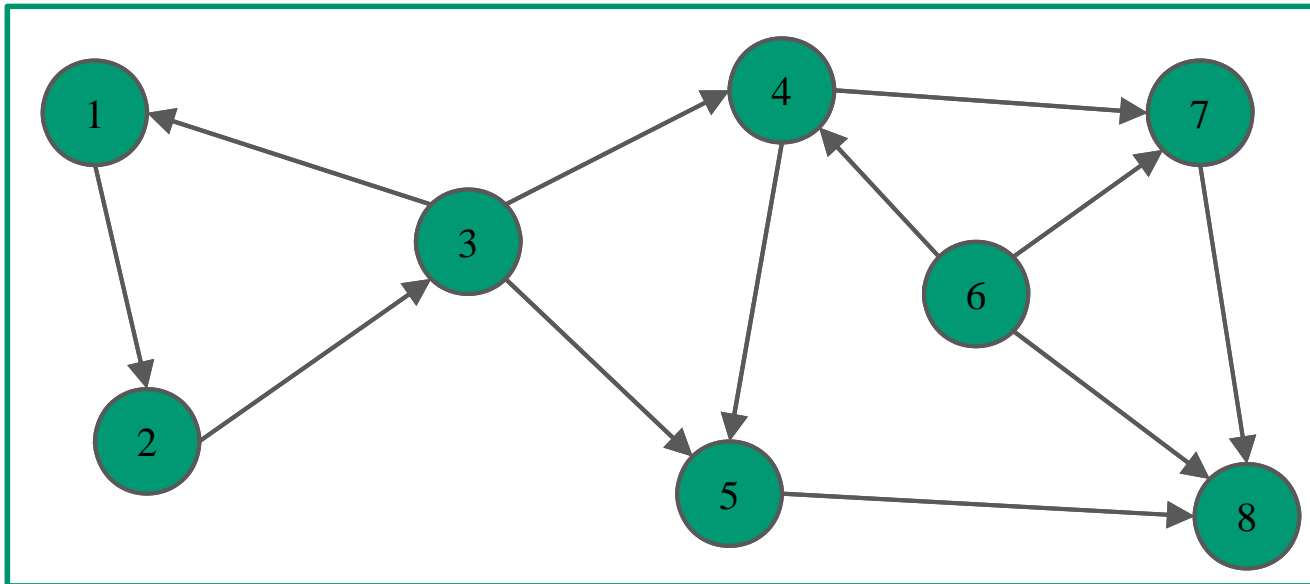


Neo4j: Cypher (2)

```
MATCH (a)-->(b)-->(c),  
      (a)-->(c)  
RETURN a,b,c
```

```
MATCH (a)-->(b)-->(c)<--(a)  
RETURN a,b,c
```

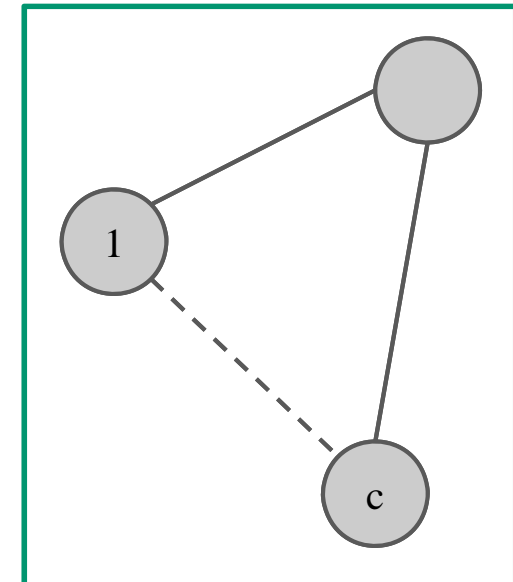
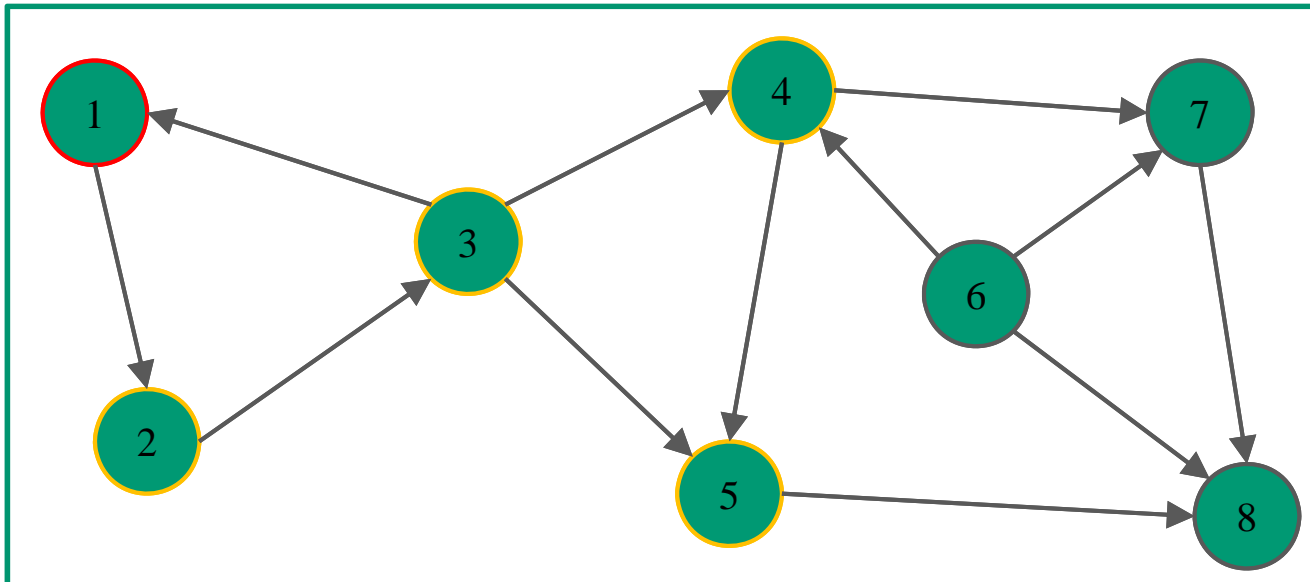
a	b	c
3	4	5
6	4	7
6	7	8



Neo4j: Cypher (3)

```
MATCH (a)--()--(c)
WHERE a.id = 1
RETURN c.id AS c
```

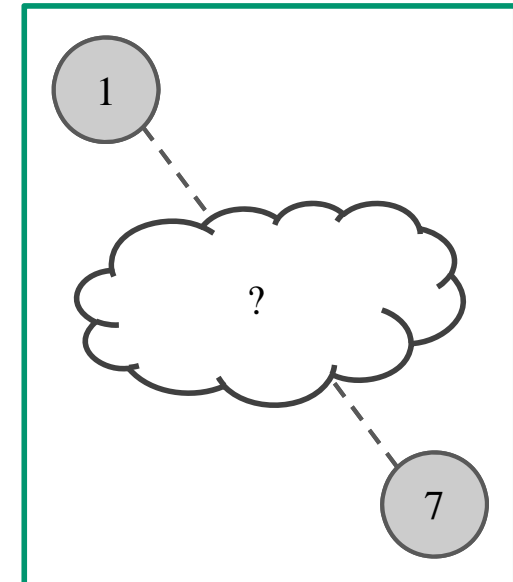
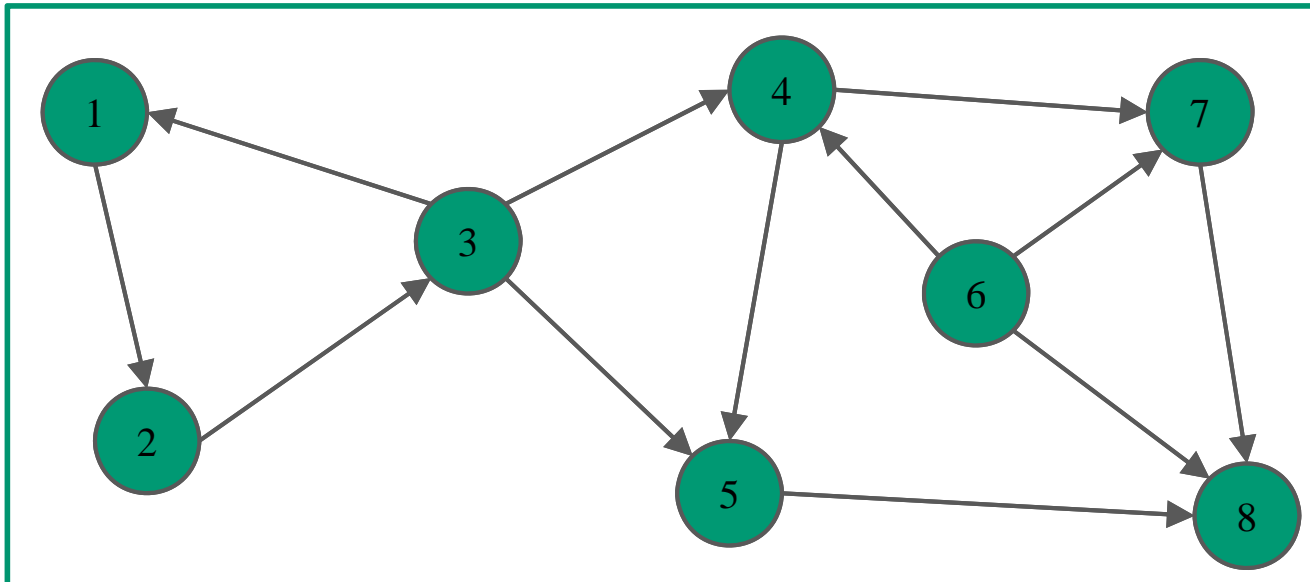
c
3
4
5
2



Neo4j: Cypher (4)

```
MATCH p=(a)-[*1..4]-(b)
WHERE a.id = 1 AND b.id = 7
RETURN nodes(p), length(p)
```

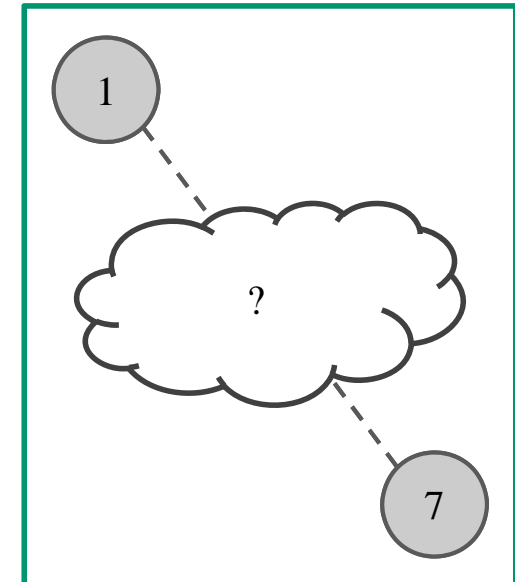
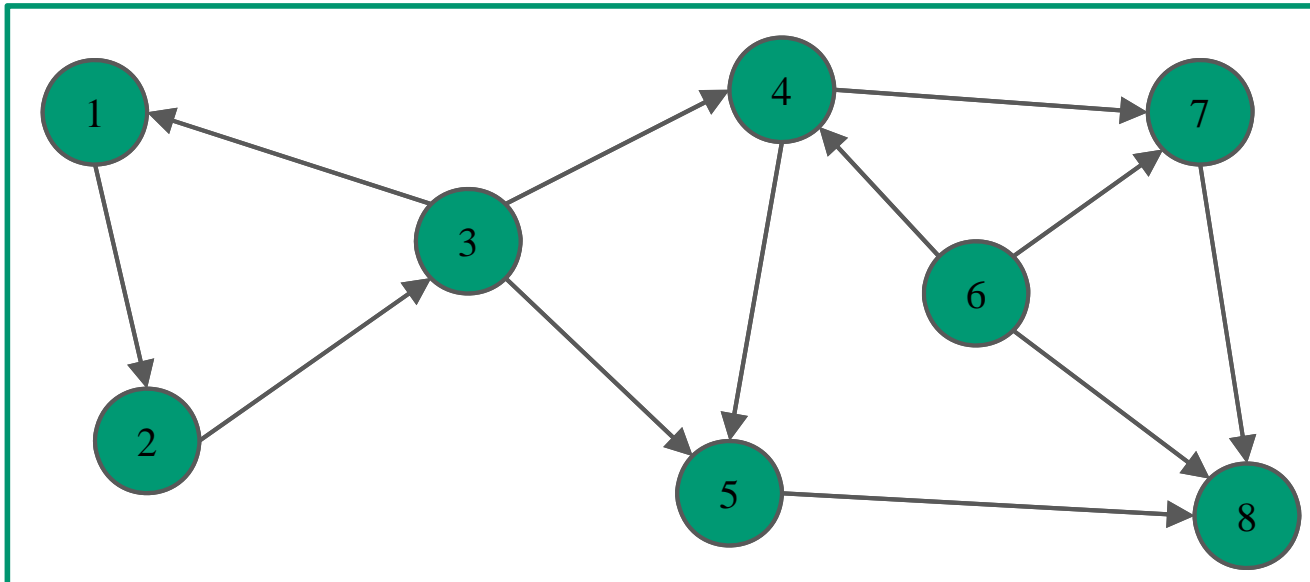
nodes(p)	length(p)
(1,2,3,4,7)	4
(1,3,4,7)	3
(1,3,4,6,7)	4
(1,3,5,8,7)	4
(1,3,5,4,7)	4



Neo4j: Cypher (5)

```
MATCH p=(a)-[*1..4]-(b)
WHERE a.id = 1 AND b.id = 7
RETURN length(p) AS length,
count(p) AS cnt
```

length	cnt
4	4
3	1

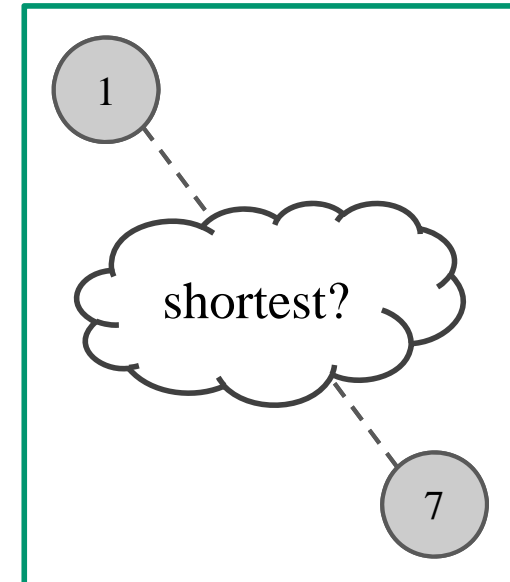
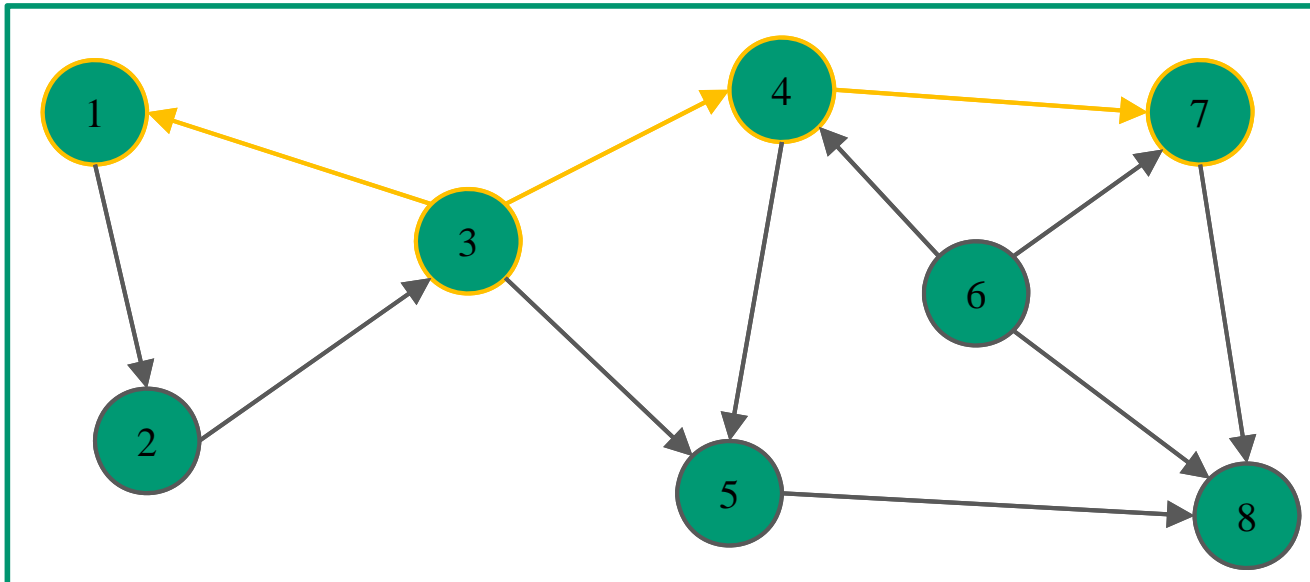


Neo4j: Cypher (6)

```
MATCH p=shortestPath((a)-[*..4]-(b))
WHERE a.id = 1 AND b.id = 7
RETURN nodes(p)
```

nodes(p)

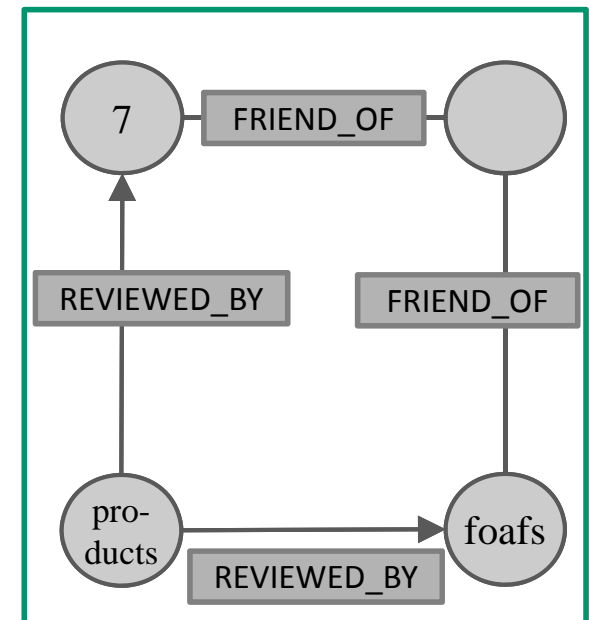
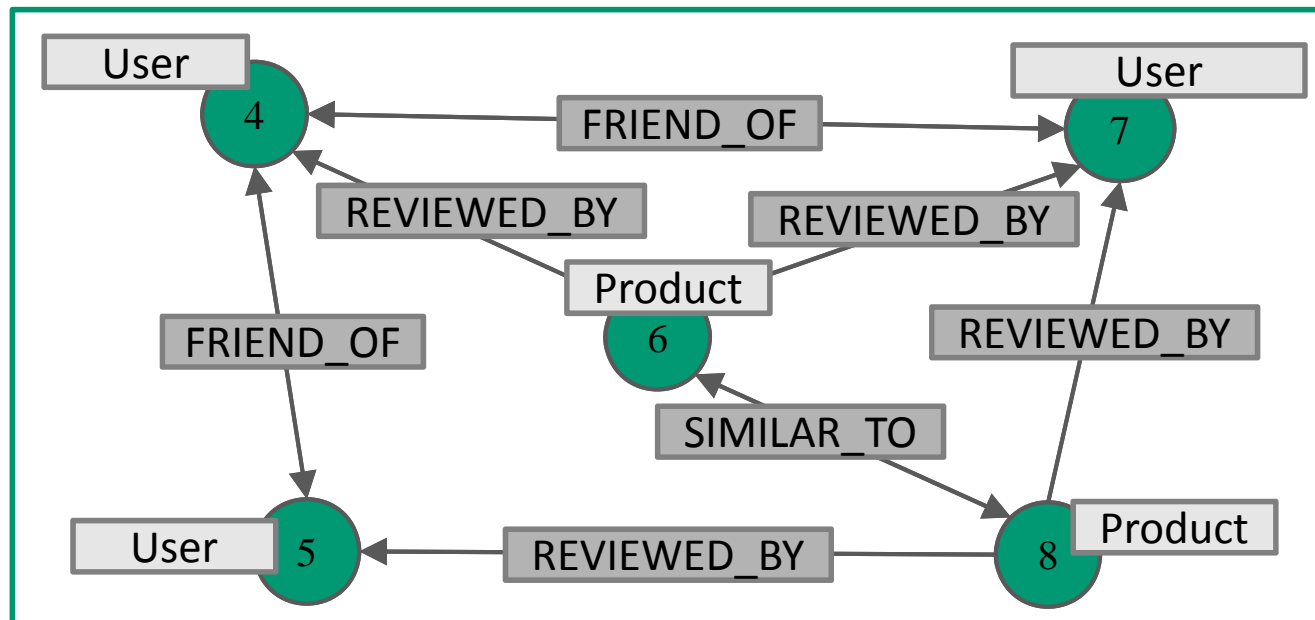
(1,3,4,7)



Neo4j: Cypher (7)

```
MATCH (u:User)-[:FRIEND_OF*2..2]->(foafs),
      (foafs)<-[:REVIEWED_BY]-(products),
      (products)-[:REVIEWED_BY]->(u)
WHERE u.id = 7
AND NOT (u)-[:FRIEND_OF]-(foafs)
RETURN foafs, products
```

foafs	products
5	8



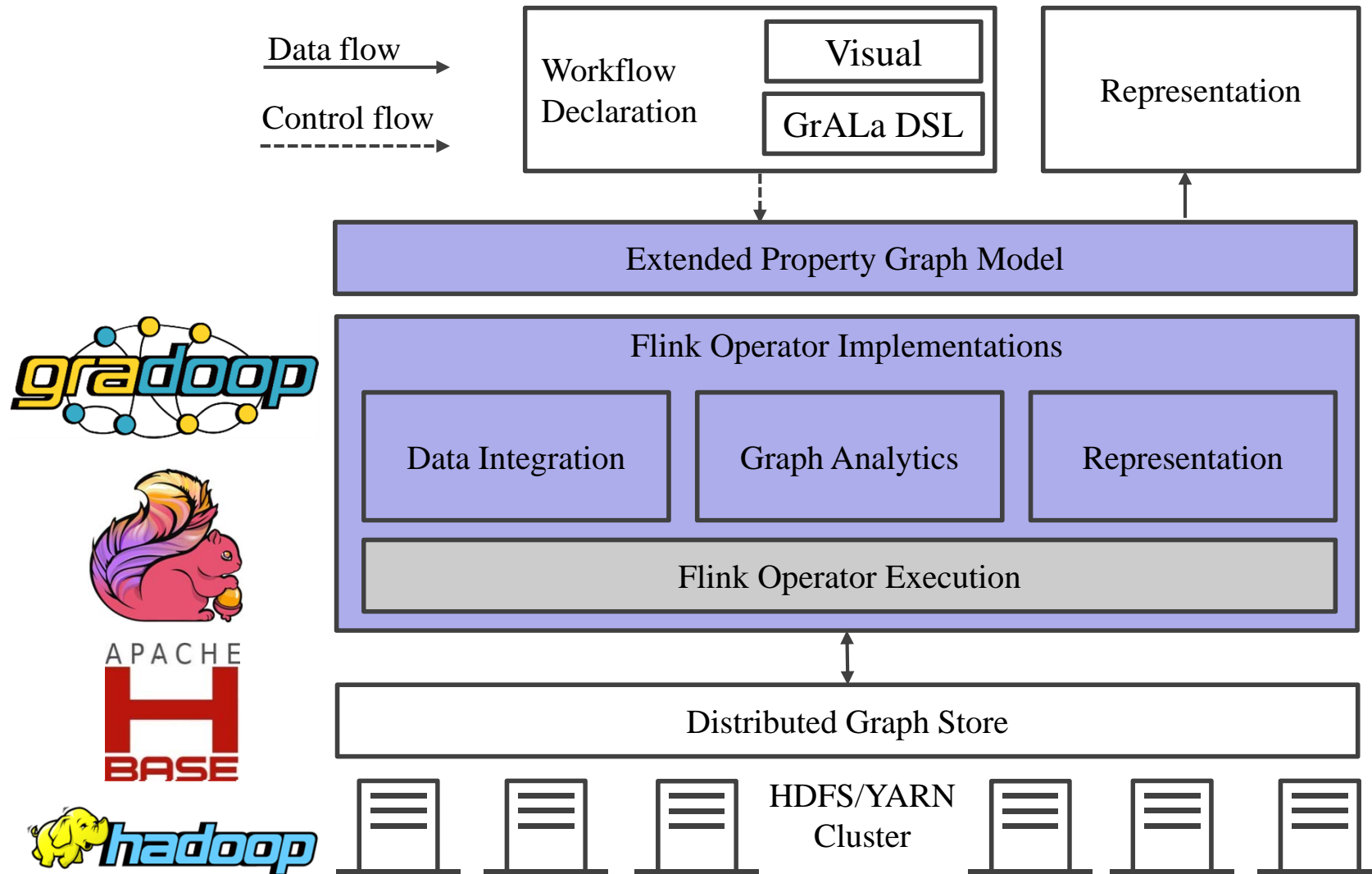
Analyse sehr großer Graphen (Big Graph Data)*

- Graph-DBS fokussieren auf einfachere Auswertungen (OLTP, Queries)
 - (relativ) ausdrucksstarkes Graphdatenmodell: Property-Graphen
 - Skalierbarkeitsprobleme
 - nur begrenztes Graph Mining
- Graph Processing Frameworks (Apache Giraph, Spark GraphX, Flink Gelly, ...)
 - parallele/skalierbare Verarbeitung von sehr großen In-Memory-Graphen
 - nur generische Graphen, keine Query-Sprache
- Forschungsansatz **Gradoop** (Univ. Leipzig, ScaDS)
 - effiziente, verteilte Verwaltung und Analyse großer Mengen von Graphdaten
 - erweitertes Property-Graph Datenmodell (EPGM) mit mächtigen Operatoren (Graph Grouping, Pattern Matching, ...)
 - hoch-skalierbar basierend auf Hadoop / Flink
 - www.gradoop.org

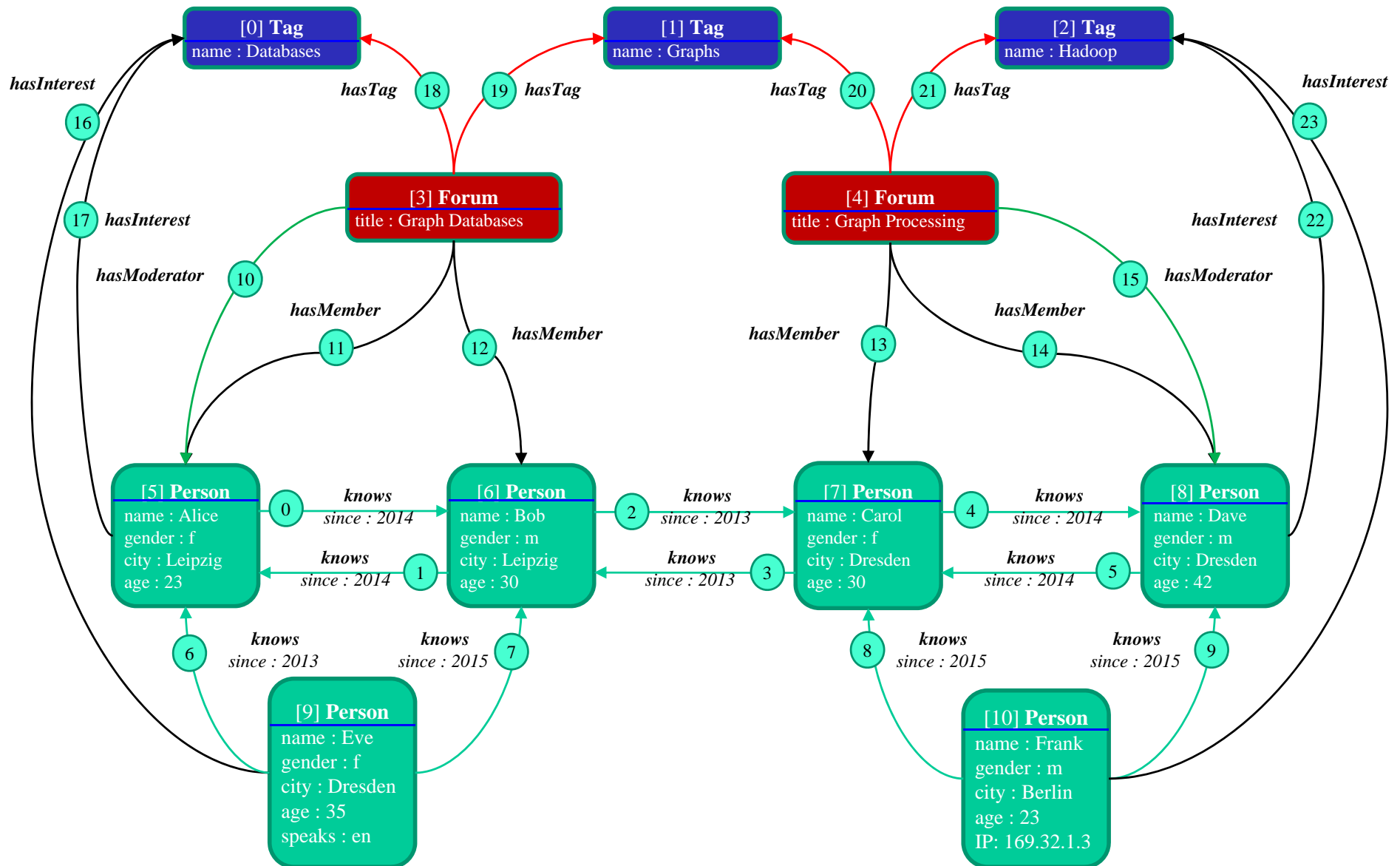


* Junghanns, M.; Petermann, A.; Neumann, M.; Rahm, E.: *Management and Analysis of Big Graph Data: Current Systems and Open Challenges*. In: Handbook of Big Data Technologies, Springer 2017

Gradoop-Architektur



Gradoop Beispielgraph



Gradoop-Operatoren

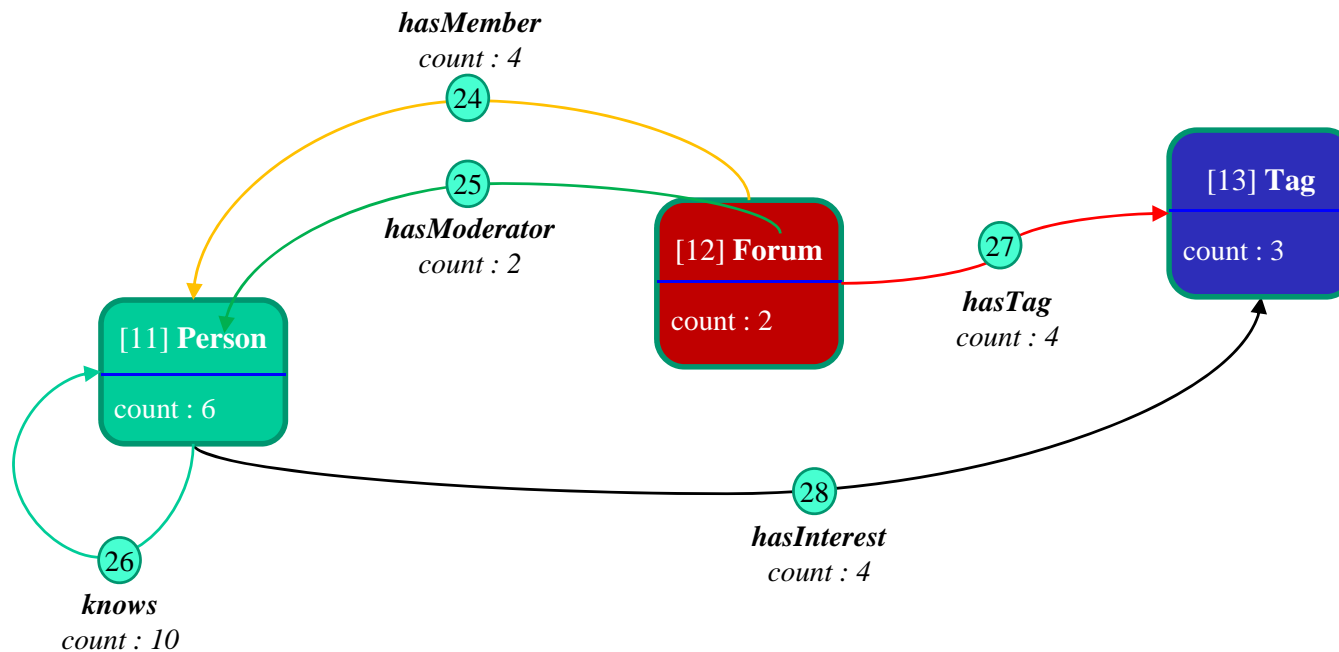
		Operators		Algorithms
		Unary	Binary	
Logical Graph		Aggregation	Combination	Gelly Library
		Pattern Matching	Overlap	BTG Extraction
		Transformation	Exclusion	Adaptive Partitioning
		Grouping	Equality	
		Subgraph		
		Call *		
Graph Collection		Selection	Union	Frequent Subgraphs
		Distinct	Intersection	
		Sort	Difference	
		Limit	Equality	
		Apply *		
		Reduce *		
		Call *		

* auxiliary



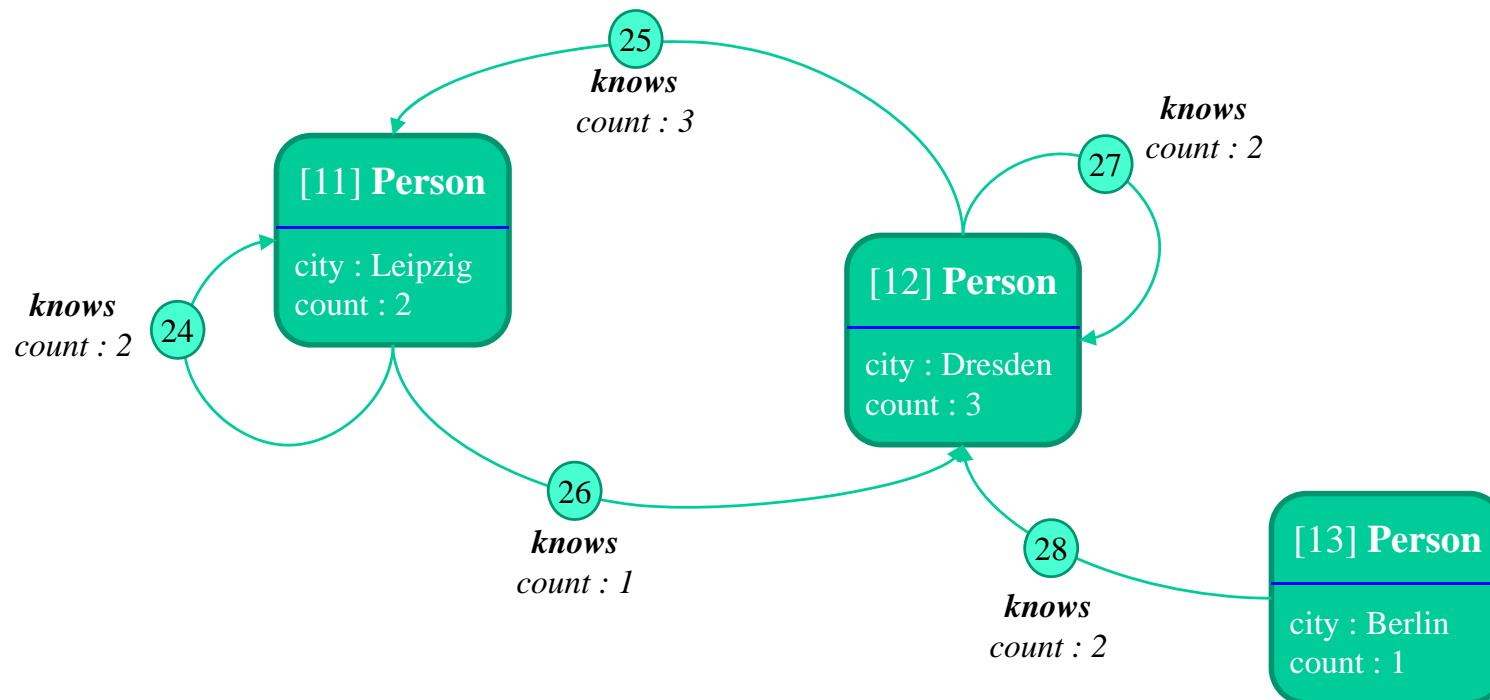
Gradoop Grouping: Typ-Ebene (Schema Graph)

```
vertexGrKeys = [:label]  
edgeGrKeys   = [:label]  
sumGraph     = databaseGraph.groupBy(vertexGrKeys, [COUNT()], edgeGrKeys, [COUNT()])
```




Grouping (2): Property-spezifisch

```
personGraph = databaseGraph.subgraph((vertex => vertex[:label] == 'Person'),  
                                     (edge => edge[:label] == 'knows'))  
vertexGrKeys = [:label, "city"]  
edgeGrKeys   = [:label]  
sumGraph     = personGraph.groupBy(vertexGrKeys, [COUNT()], edgeGrKeys, [COUNT()])
```



Vergleich

	Graph-Datenbanken, z.B. Neo4j	Graph Processing Systems (z.B. Giraph)	Distributed Dataflow Systems (Flink Gelly, Spark GraphX)	
Datenmodell	PGM	generische Graphen	generische Graphen	Extended PGM
Fokus	Queries	Analyse/Mining	Analyse/Mining	Analyse/Mining
Query-Sprache	ja	nein	nein	(ja)
Skalierbarkeit	vertikal	horizontal	horizontal	horizontal
Workflows	nein	nein	ja	ja
Dynamische Graphen / Versionierung	nein	nein	nein	in Arbeit
Visualisierung	(ja)	nein	nein	in Arbeit
Datenintegration	nein	nein	nein	in Arbeit

Zusammenfassung

- **Big Data Herausforderungen**
 - Volume, Variety, Velocity, Veracity, Privacy
 - skalierbare Analysen / Machine Learning
- **unterschiedliche Architekturen: Cloud/Hadoop-Cluster, In-Memory Warehouses, hybride Lösungen**
- **NoSQL**
 - Auslöser: webskalierbares Datenmanagement
 - semistrukturierte schemafreie Daten
 - Verzicht auf SQL/ACID
- **unterschiedliche Systemarten**
 - Key/Value-Stores, erweiterte Record-Stores (Spaltenfamilien)
 - Dokumenten Stores
 - Graph-Datenbanken
- **Hauptproblem für NoSQL: fehlende Standards**
- **skalierbare Graphanalysen: Queries und Mining auf verteilten Plattformen**



DBS2-Abschlussklausur

- **Mittwoch, 24.07.2019, 10 Uhr, Audimax**
 - Voraussetzung: bestandene Zwischenklausur
 - Klausurdauer: 60 Minuten
- **Inhalt: alle Kapitel**
 - Anwendungsprogrammierung / Web-Anbindung (JDBC, PHP)
 - NF², OODBS /ORDBS (UDT, Temporale DB)
 - XML-DB (DTD, XMLSchema, XPath, XQuery)
 - Big Data / NoSQL (Überblick, Cypher)
- **Vorbereitung:**
 - Folien-Skript + Mitschrift
 - Übungsaufgaben
 - LOTS
 - Literatur
 - **Hörsaal-Übung: 08.07. Übersicht + interaktive Schwerpunktsetzung**

