

6. 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

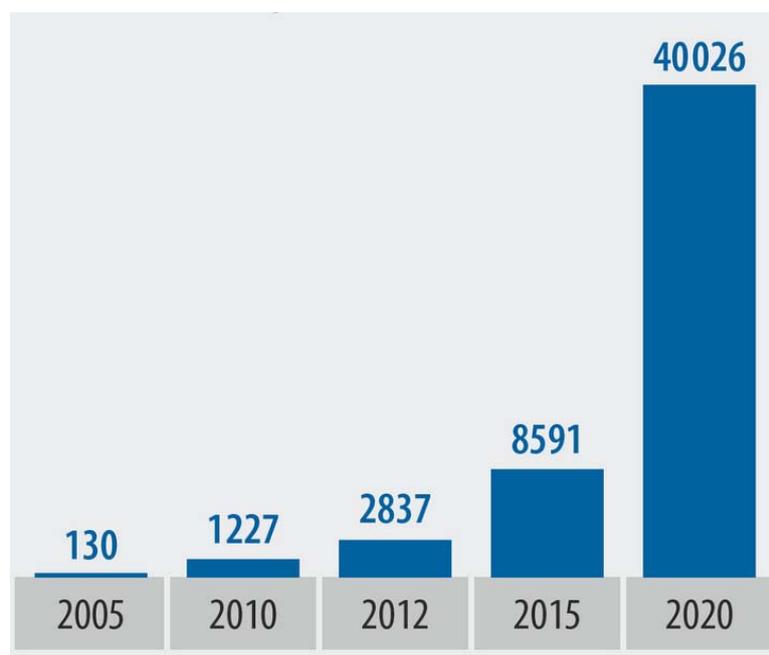


Wie groß ist Big Data?

■ Big wandelt sich schnell

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

■ bis 2020 werden ca 40 ZB jährlich erzeugt



Quelle: IDC

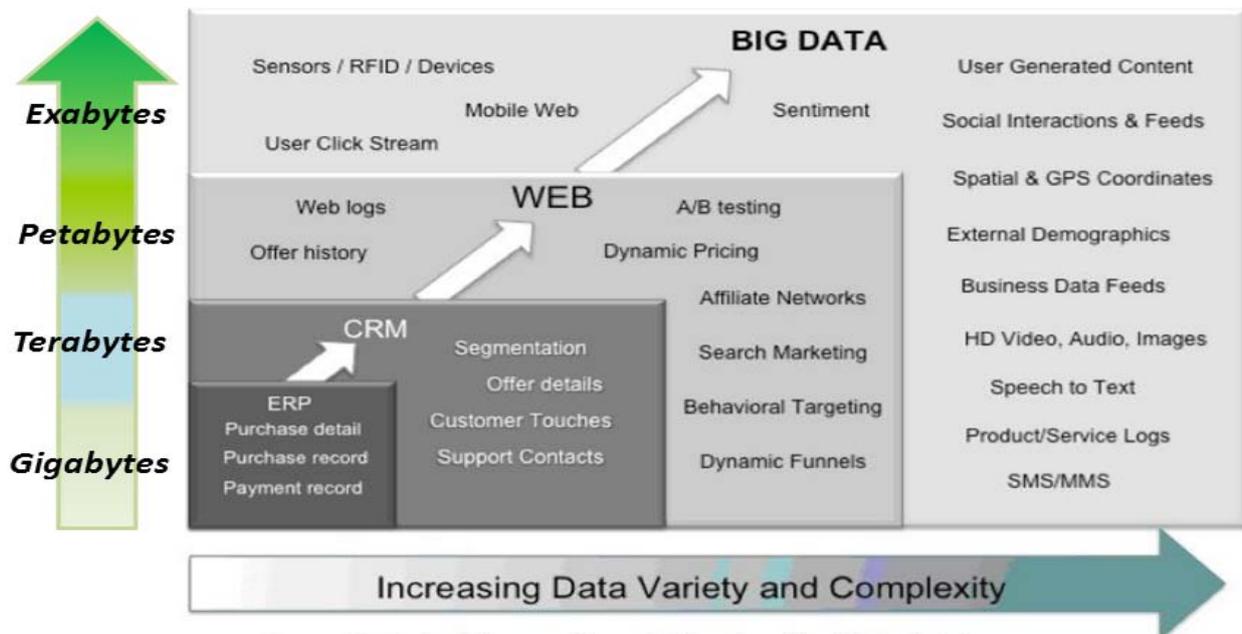


Welche Art von Daten?

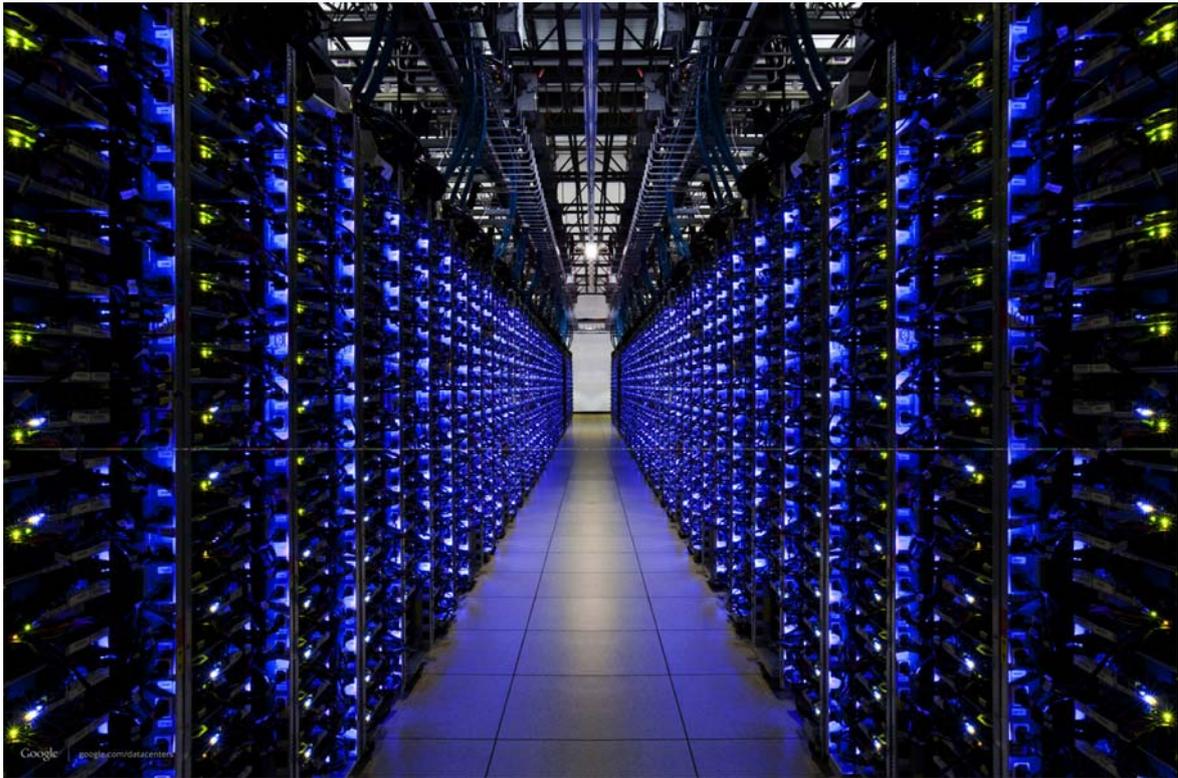
- mobile Endgeräte (Smartphones, Watches, ...):
Multimedia-Daten (Fotos, Videos), Bewegungsdaten, Nachrichten ...
- Unternehmensdaten
 - strukturierte Daten (Datenbanken, Data Warehouses) mit Angaben zu Personal, Kunden, Bestellungen, Rechnungen ...
 - Dokumente, E-Mails
- Web-Daten
 - Web-Seiten + Multimedia-Inhalte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



Datenzunahme



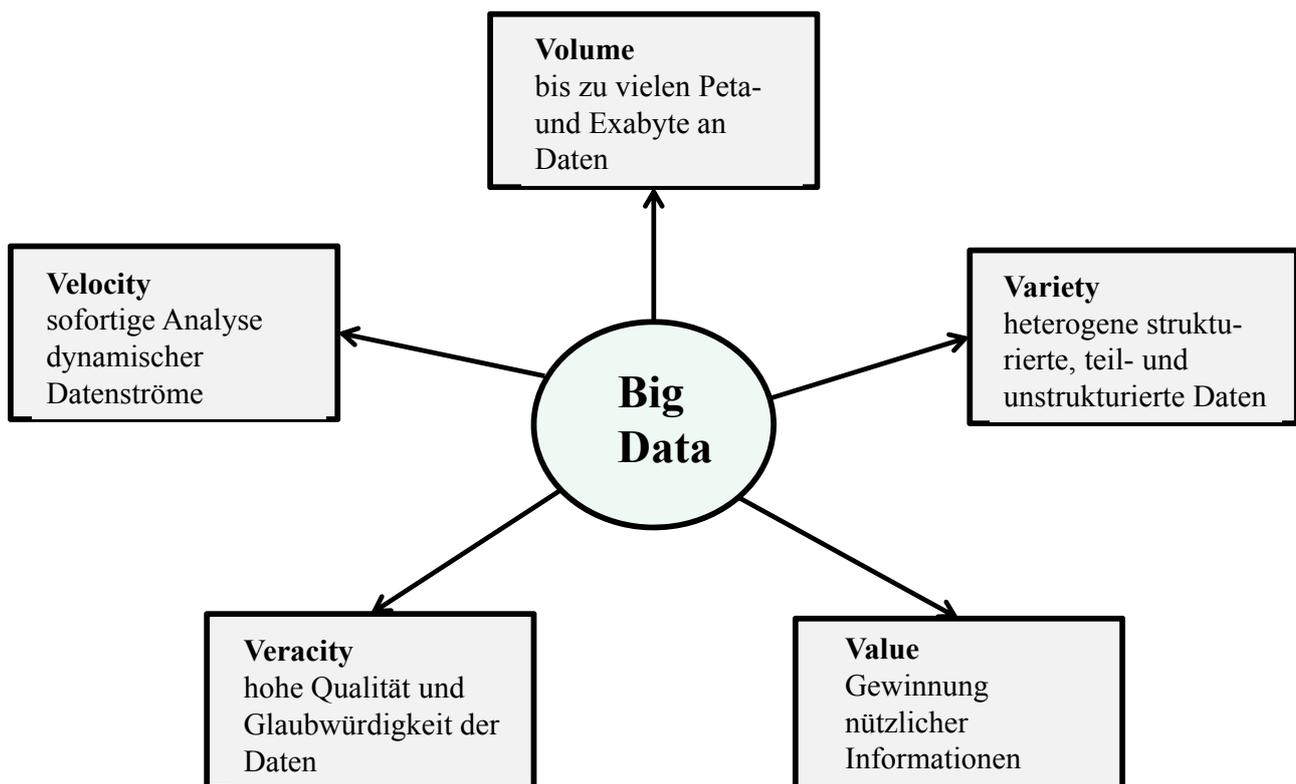
Data Center



Source: Google 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



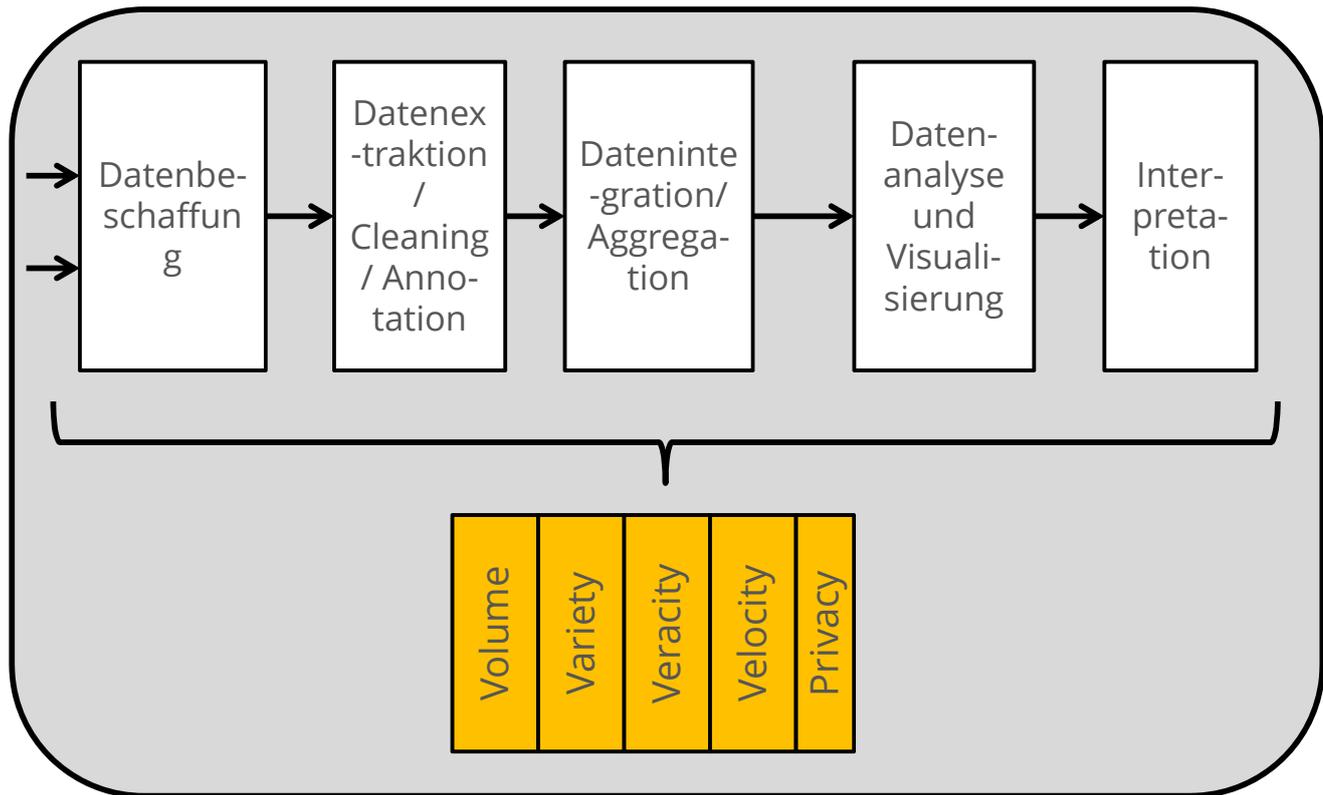
Fraud and Risk



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, ...

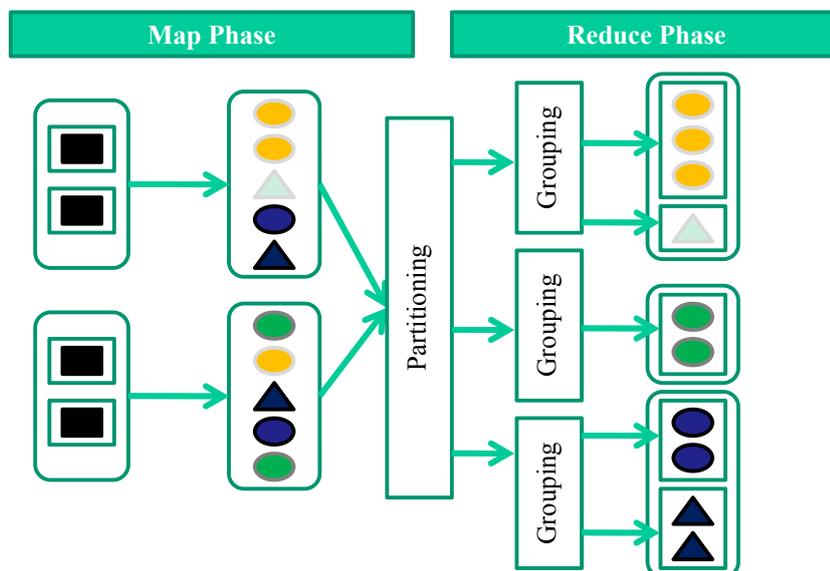
MapReduce

- 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 ...*

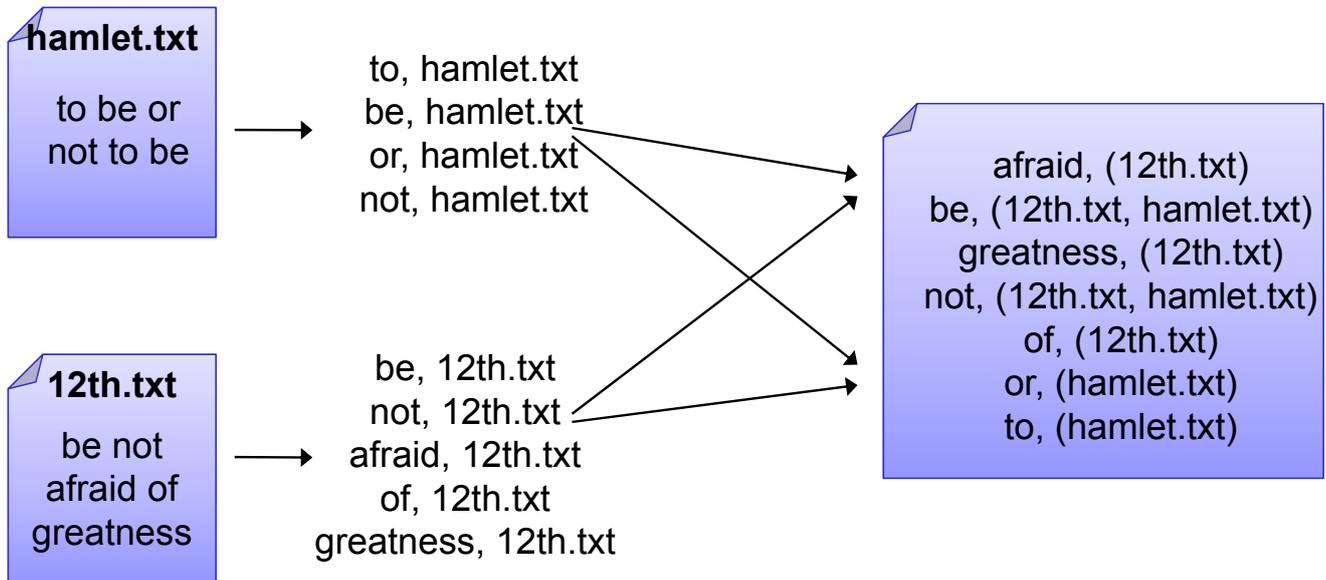


MapReduce

- 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

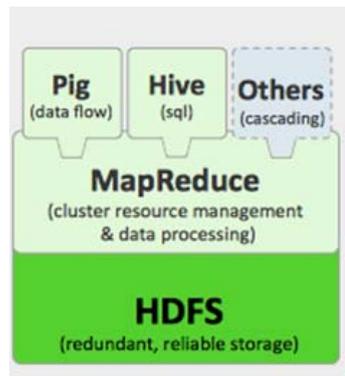


MR-Beispiel: Generierung Text-Index

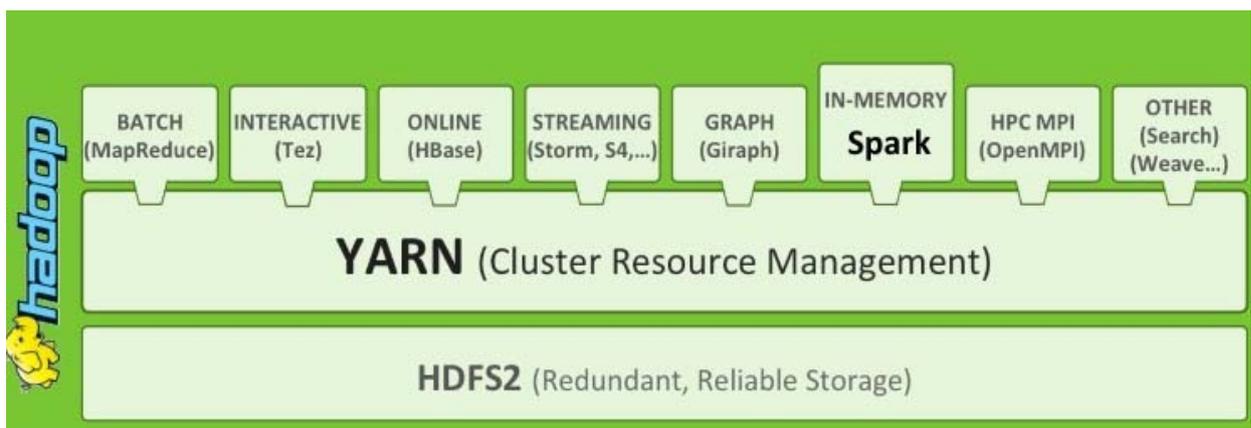


Hadoop-Ökosystem für Big Data

Hadoop 1.0

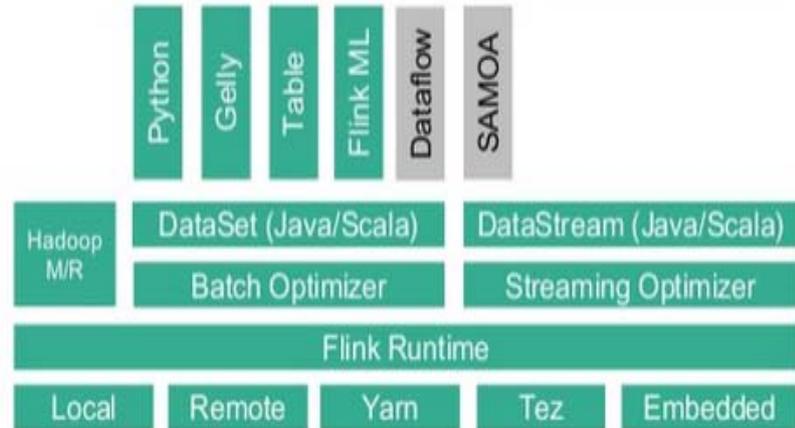
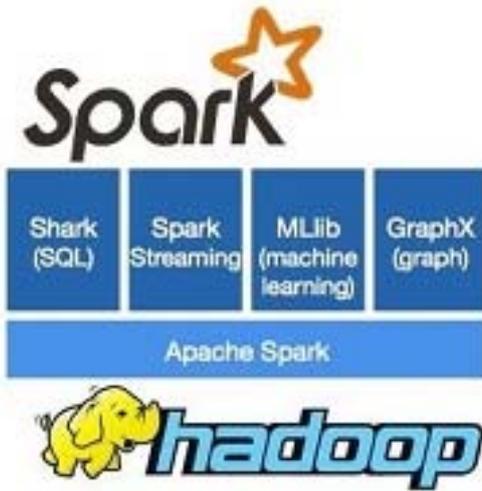


Hadoop 2.x



Spark vs Flink

- In-Memory-Plattformen zur schnelleren Verarbeitung, u.a. für Streaming, Graphanalysen und Machine Learning



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



NoSQL-Datenbanken



- 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 “Not only Sql“ 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

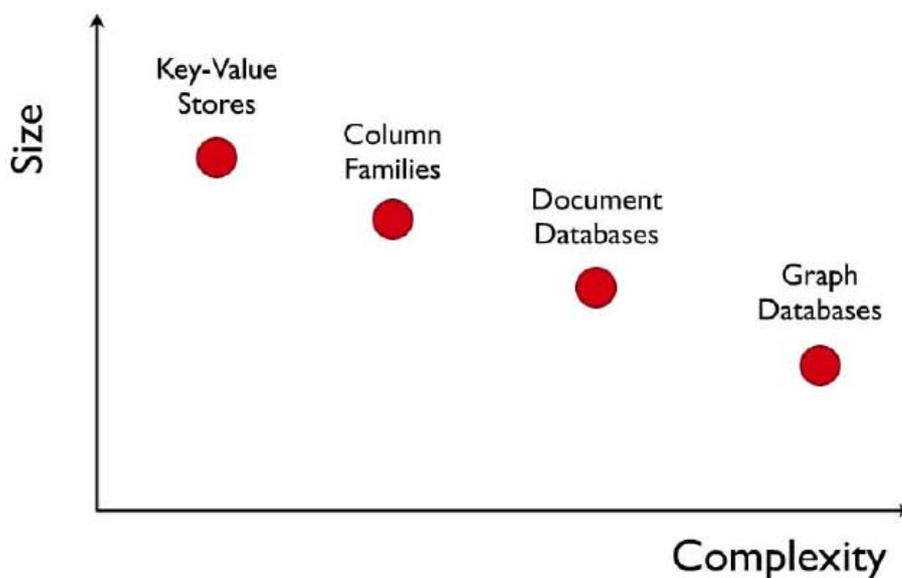
- CouchDB, MongoDB ...
- Speicherung semistrukturierter Daten als Dokument (z.B. JSON)

■ Graph-Datenbanken

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



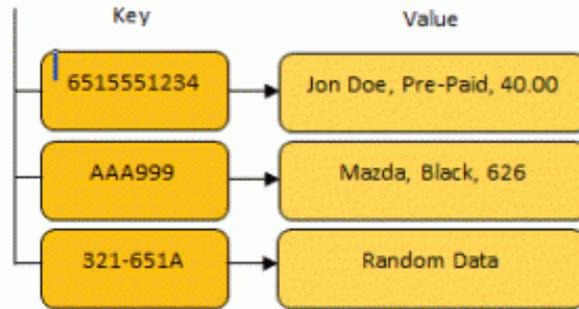
Grobeinordnung NoSQL-Systeme



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)



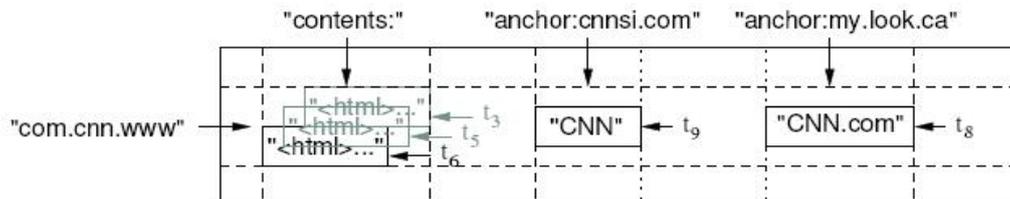
Spaltenorientierte Key-Value Stores

■ 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)



Dokumenten-Datenbanken

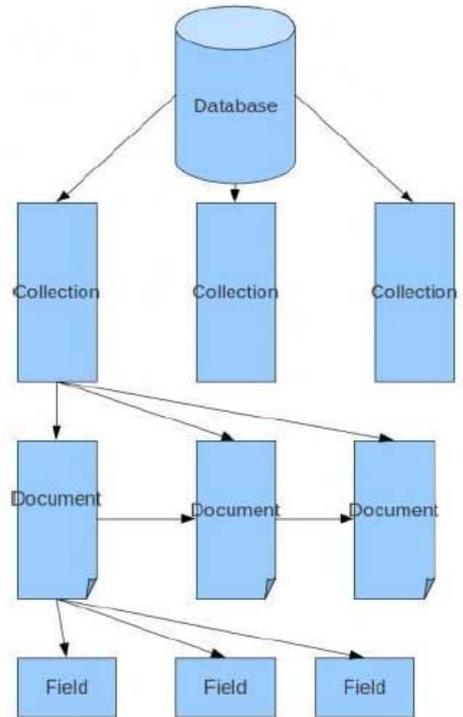
- schemalose Speicherung von Dokumenten, v.a. im JSON-Format
- JSON (JavaScript Object Notation)
 - geschachtelte Objekt-Notation
 - Datentypen: String, Zahl, Array [...], Boolean, Nullwert
 - Objekt { } umfasst Menge von Key-Value (Attribut/Wert-) Paaren
- JSON einfacher als XML, leicht lesbar / schreibbar
- Beispiel (JSON vs. XML)

```
{  
  "Herausgeber": "Mastercard",  
  "Nummer": "1234-5678-9012-3456",  
  "Währung": "EURO",  
  "Inhaber": {  
    "Name": "Mustermann",  
    "Vorname": "Max",  
    "männlich": true,  
    "Hobbys": [ "Reiten", "Golfen", "Lesen" ],  
    "Kinder": [],  
    "Partner": null  
  }  
}
```

```
<Kreditkarte Herausgeber="Mastercard"  
  Nummer="1234-5678-9012-3456"  
  Waehrung="EURO">  
<Inhaber Name="Mustermann" Vorname="Max"  
  maennlich=true Partner="null">  
  <Hobbys>  
    <Hobby>Reiten</Hobby>  
    <Hobby>Golfen</Hobby>  
    <Hobby>Lesen</Hobby>  
  </Hobbys>  
  <Kinder />  
</Inhaber>  
</Kreditkarte>
```



- 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.find();
```

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

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



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

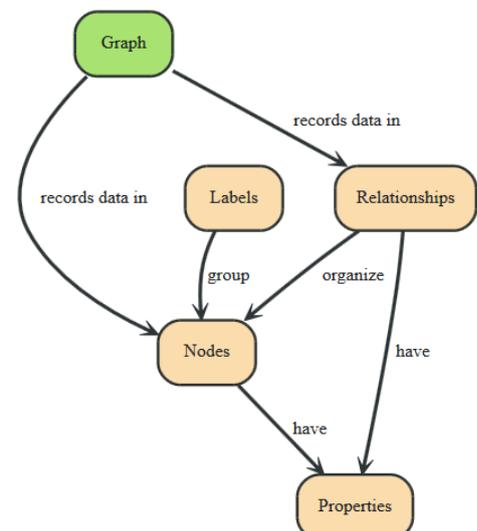
- bessere Unterstützung stark vernetzter Daten als mit Relationenmodell
 - Soziale Netzwerke
 - Protein-Netzwerke
 - stark vernetzte Webdaten / Unternehmensdaten / ...
- Graph-Verwaltung im Relationenmodell oft nicht ausreichend schnell
 - viele Tabellen, viele Joins
 - langsame Traversierung von Kanten
 - 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“)



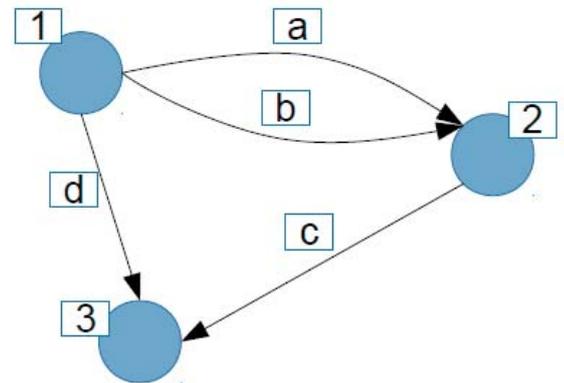
Neo4J



- 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



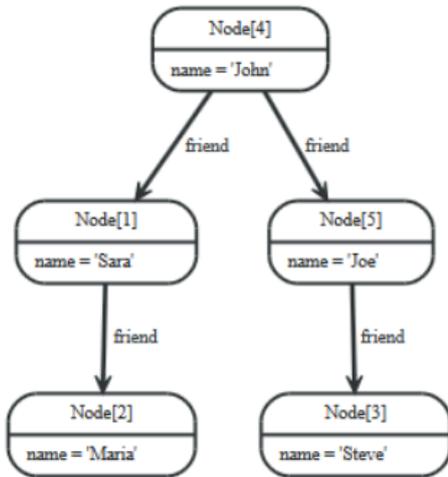
- mehrere, gerichtete Kanten zwischen zwei Knoten möglich (gerichteter „Multigraph“)
 - Labels für Knoten/Kanten
 - Properties für Knoten/Kanten
- konstanter Aufwand zur Traversierung zu Nachbarknoten (statt Join im RM)



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.

Beispiel-Graph



Beispielanfrage (Friend Of Friend)

```

START john=node:node_auto_index(name = 'John')
MATCH john-[:friend]->()-[:friend]->fof
RETURN john, fof
  
```

Ergebnis:

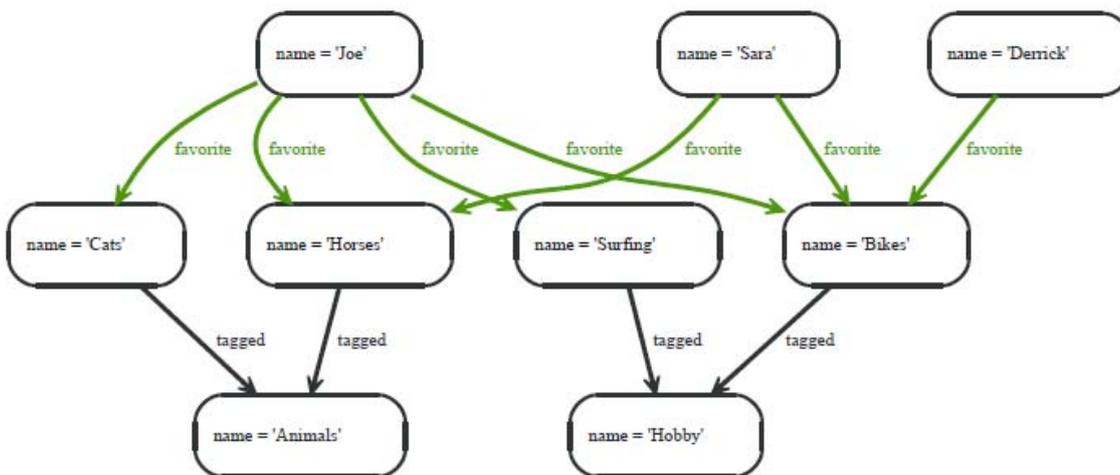
john	fof
Node[4]{name:"John"}	Node[2]{name:"Maria"}
Node[4]{name:"John"}	Node[3]{name:"Steve"}
2 rows	

Quelle: Neo4J Tutorial, <http://docs.neo4j.org>



Beispiel 2

■ finde Personen mit ähnlichen Interessen (Recommendations)



```

START me=node:node_auto_index(name = "Joe")
MATCH me-[:favorite]->myFavorites-[:tagged]->tag
      <-[:tagged]-theirFavorites<-[:favorite]-people
WHERE NOT(me=people)
RETURN people.name as name, count(*) as similar_favs
ORDER BY similar_favs DESC
  
```



Analyse sehr großer Graphen (Big Graph Data)*

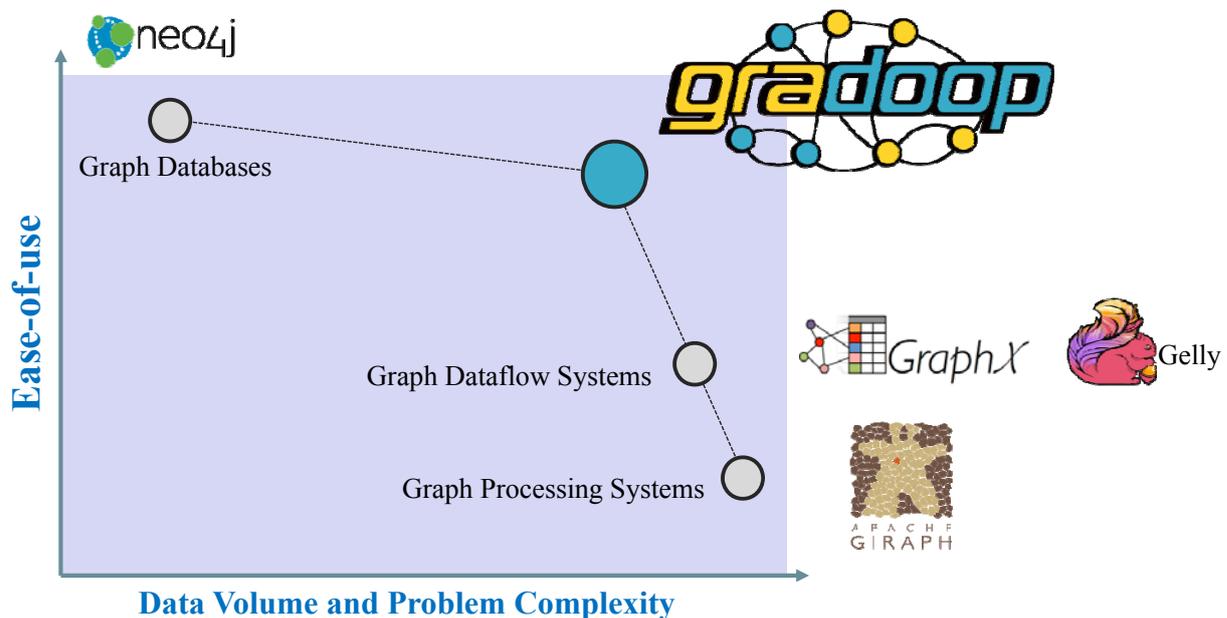
- Graph-DBS fokussieren auf einfachere Auswertungen (OLTP, Queries)
 - (relativ) ausdrucksstarkes Graphdatenmodell: Property-Graphen
 - Skalierbarkeitsprobleme
 - nur begrenztes Graph Mining
- Graph Processing Frameworks (Apache Giraph, Spark GraphX, Flink Gelly, ...)
 - parallele/skalierbare Verarbeitung von sehr großen In-Memory-Graphen
 - nur generische Graphen, keine Query-Sprache
- neuer Forschungsansatz **Gradoop** (UL, 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



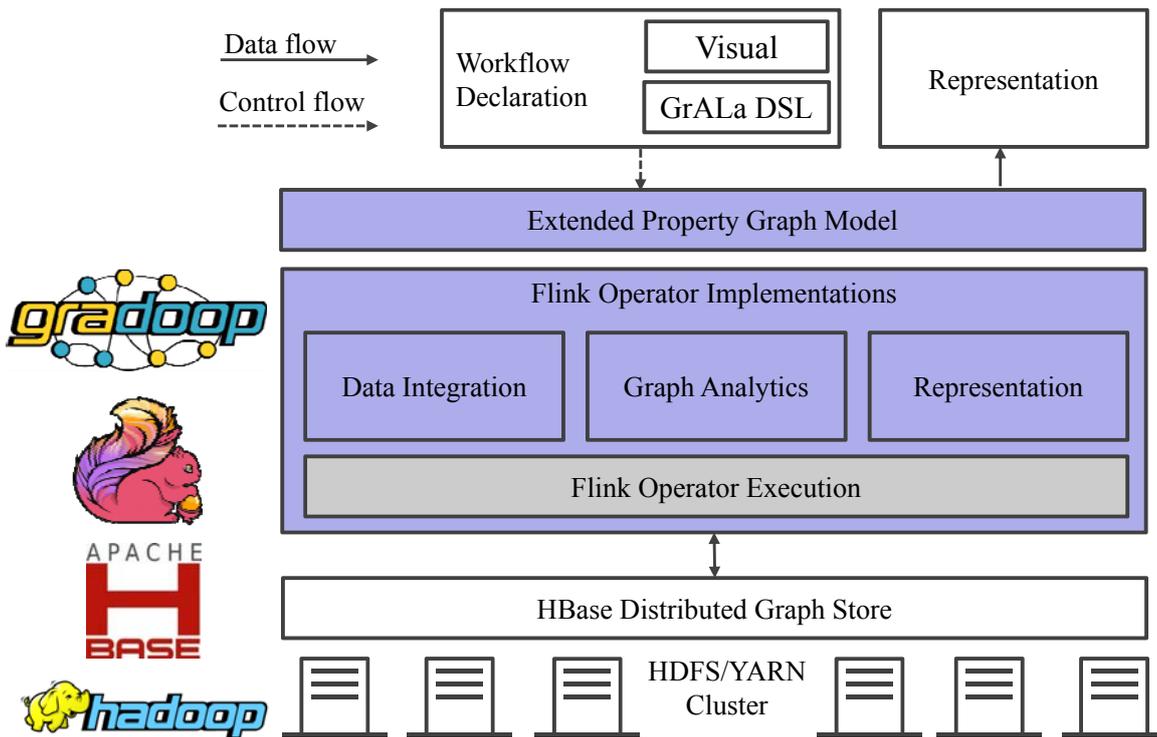
* Junghanns, M.; Petermann, A.; Neumann, M.; Rahm, E.: *Management and Analysis of Big Graph Data: Current Systems and Open Challenges*. In: Handbook of Big Data Technologies, Springer 2017



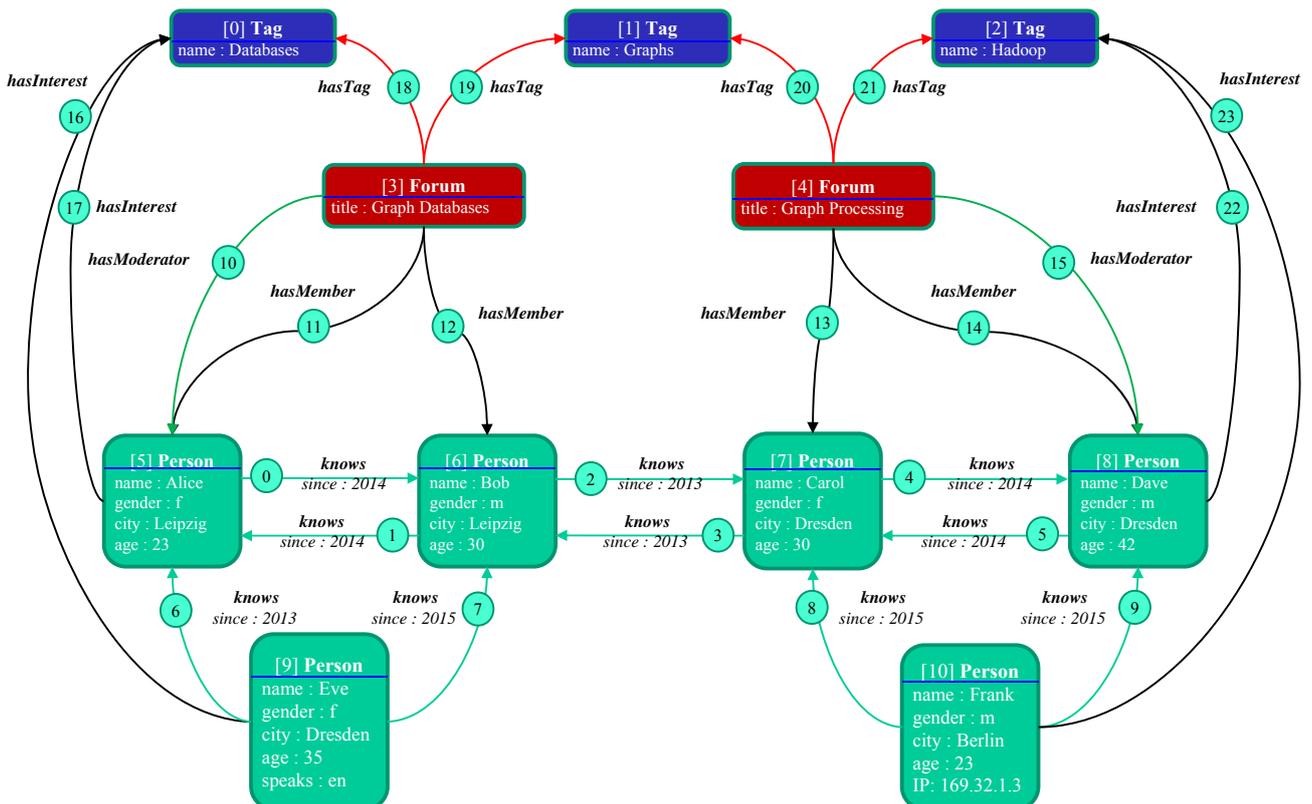
Ansätze zur Graphanalyse



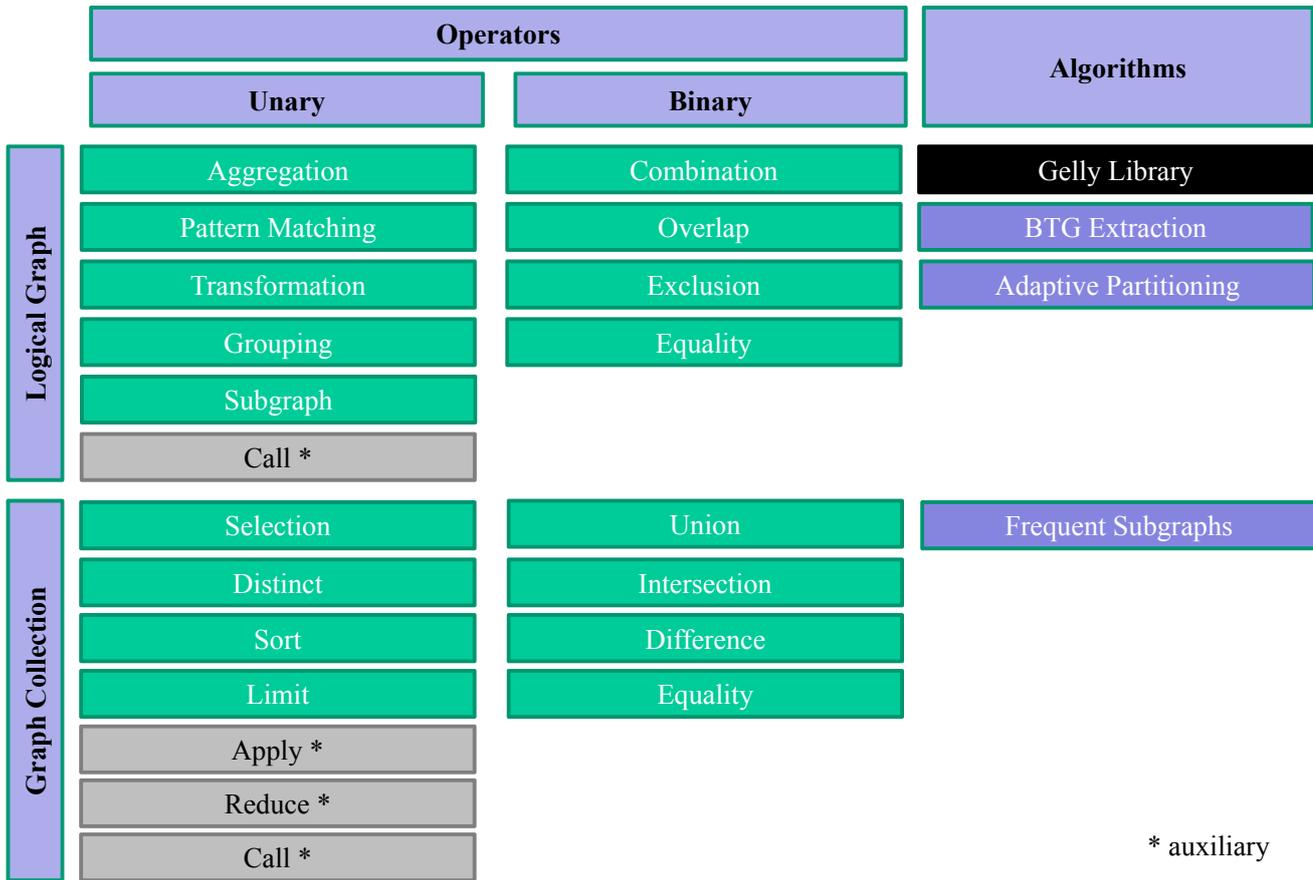
Gradoop-Architektur



Gradoop Beispielgraph



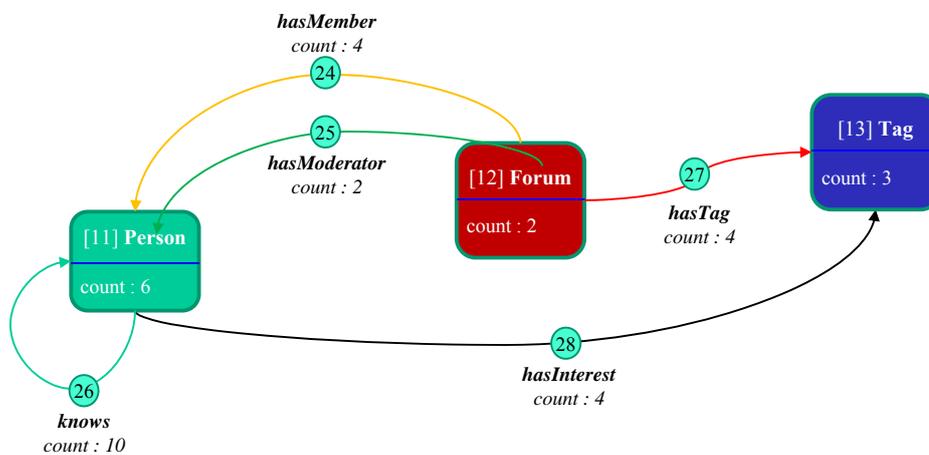
Gradoop-Operatoren



Gradoop Grouping: Typ-Ebene (Schema Graph)

```

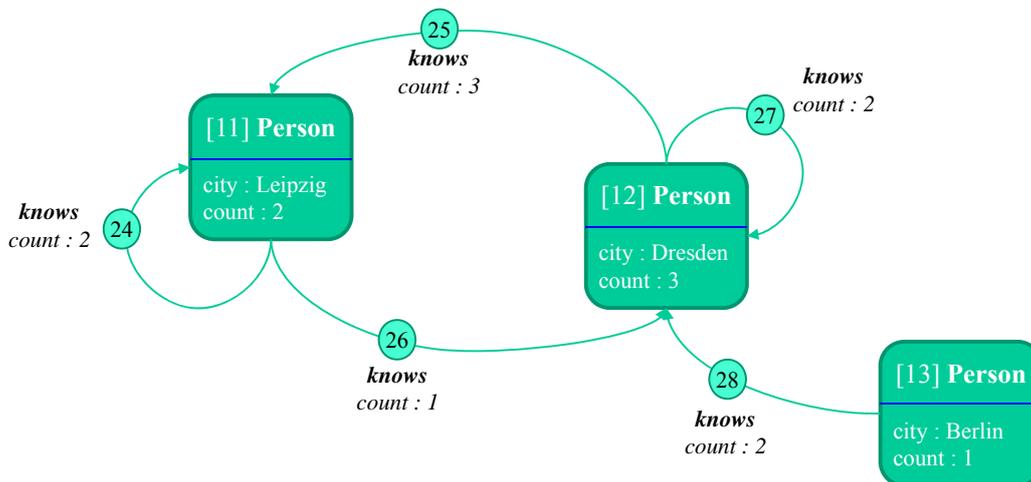
vertexGrKeys = [:label]
edgeGrKeys   = [:label]
sumGraph     = databaseGraph.groupBy(vertexGrKeys, [COUNT()], edgeGrKeys, [COUNT()])
    
```



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	nein
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

