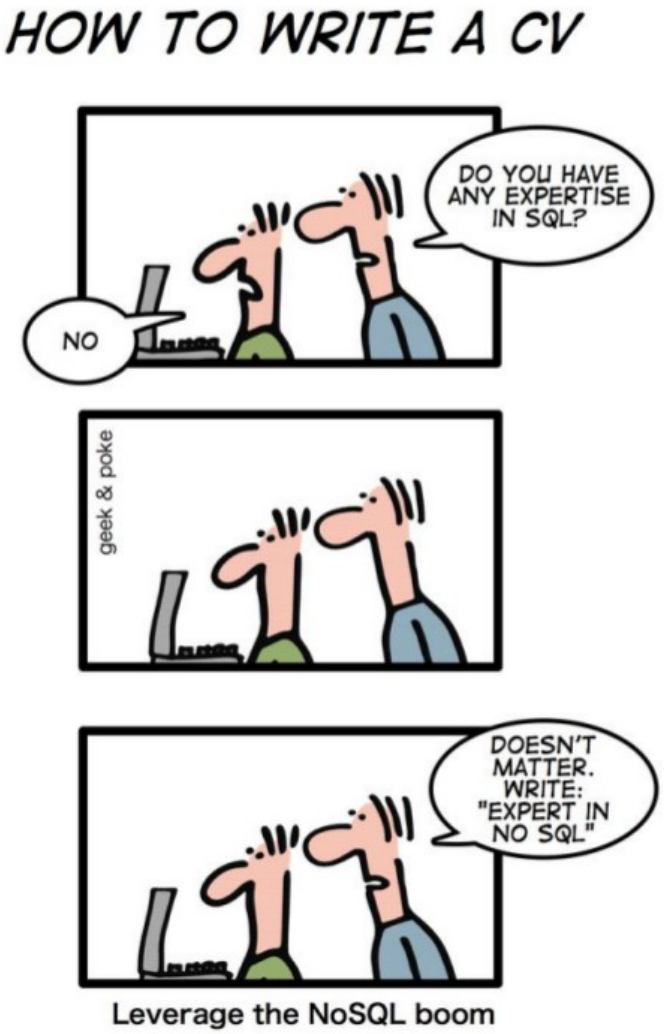
noSQL

„not only SQL“

NoSQL: DAS aktuelle Datenbank-Buzzword

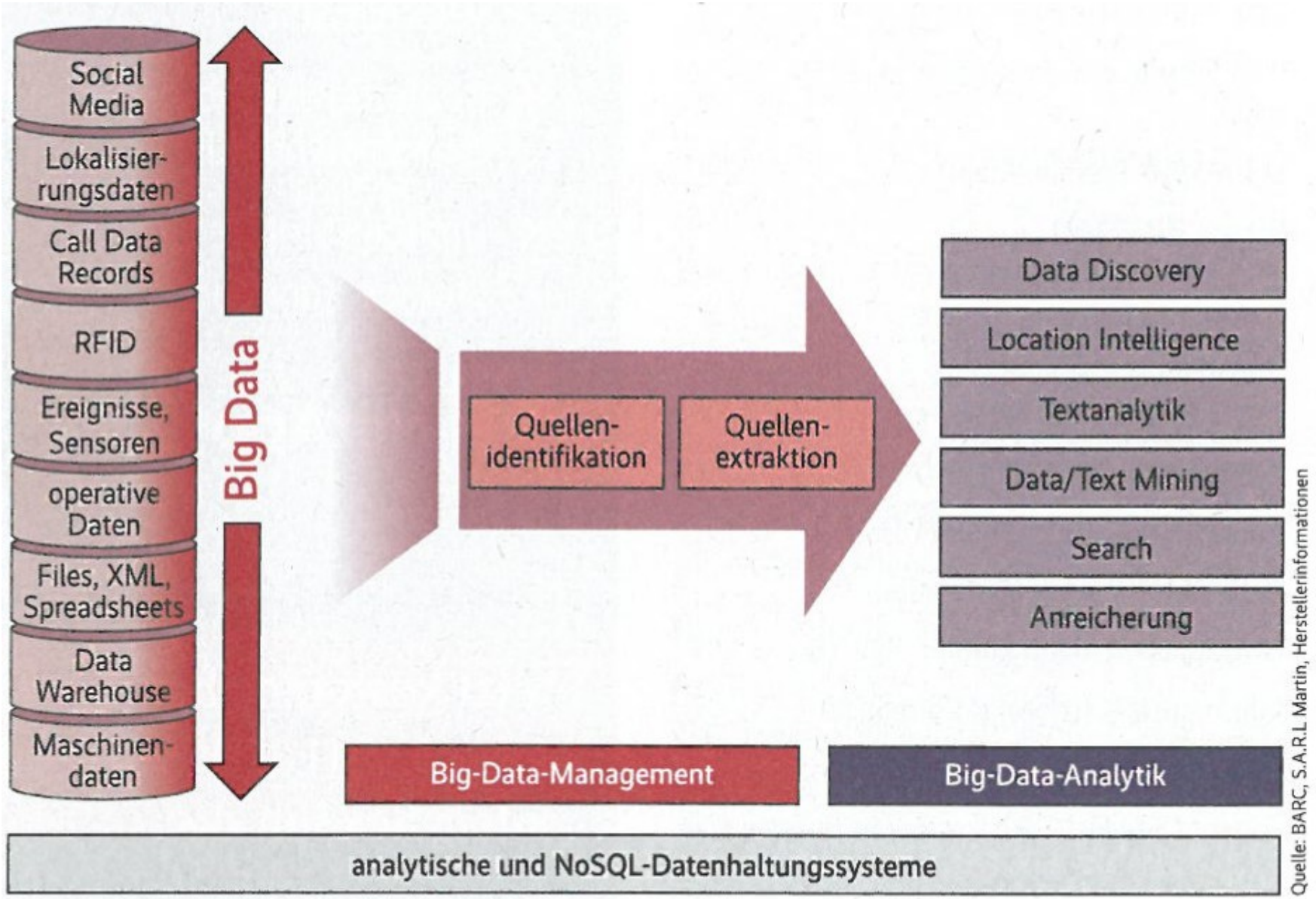


Quelle: http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

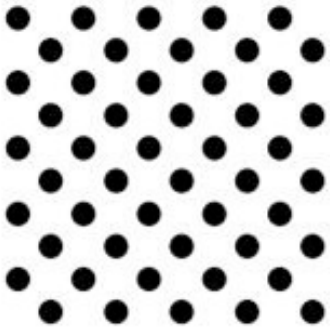
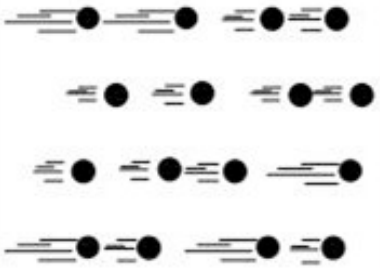
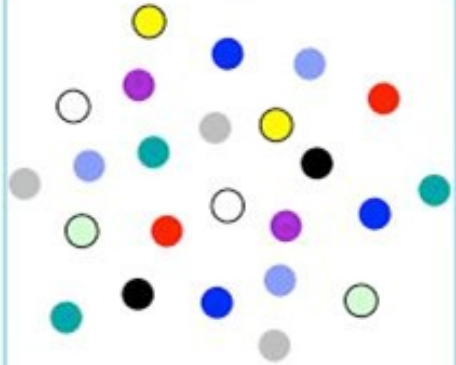
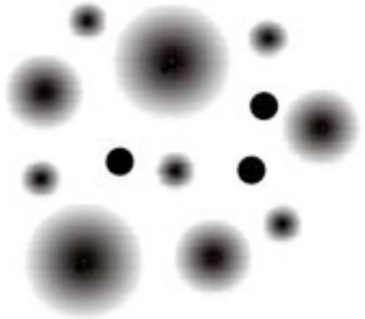
Lehrinhalte

- BigData: Datenquellen, Datenverwendung, …
- Grundlagen noSQL
  - Map/Reduce
  - Kategorisierung
  - Skalierung
  - CAP versus ACID
- LiveCoding
  - CouchDB, MongoDB, Neo4J

BigData – Big Picture



Eigenschaften von BigData: The 4 V‘s

Volume	Velocity	Variety	Veracity*
			
Data at Rest	**Data in Motion**	**Data in Many Forms**	**Data in Doubt**
Terabytes to exabytes of existing data to process	Streaming data, milliseconds to seconds to respond	Structured, unstructured, text, multimedia	Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations

http://contest.trendmicro.com/2013/train.htm

* Aufrichtigkeit, Wahrhaftigkeit

Definition: Not only SQL (noSQL)

Es existiert noch kein einheitliche Definition – ein Variante:

- [Edlich et al: 2011 bzw. http://nosql-database.org/]

Unter NoSQL wird eine neue Generation von Datenbanksystemen verstanden, die meistens einige der nachfolgenden Punkte berücksichtigen:

- Das zugrundeliegende Datenmodell ist nicht relational.
- Das System ist schemafrei oder hat nur schwächere Schemarestriktionen.
- Das System bietet eine einfache API.

aktuell werden komplexere APIs entwickelt

- Die Systeme sind von Anbeginn an auf eine verteilte und horizontale Skalierbarkeit ausgerichtet.
- Aufgrund der verteilten Architektur unterstützt das System eine einfache Datenreplikation.
- Dem System liegt meistens auch ein anderes Konsistenzmodell zugrunde: Eventually Consistent und BASE, aber nicht ACID
- Das NoSQL-System ist Open Source.

inzwischen teilweise „sowohl als auch“

NoSQL: Die Essenz

Datenmodell

- Das zugrundeliegende Datenmodell ist nicht relational.
- Das System ist schemafrei oder hat nur schwächere Schemarestriktionen.

Skalierungsarchitektur

- Die Systeme sind von Anbeginn an auf eine verteilte und horizontale Skalierbarkeit ausgerichtet.
- Aufgrund der verteilten Architektur unterstützt das System eine einfache Datenreplikation.

NoSQL: Die Essenz

Datenmodell

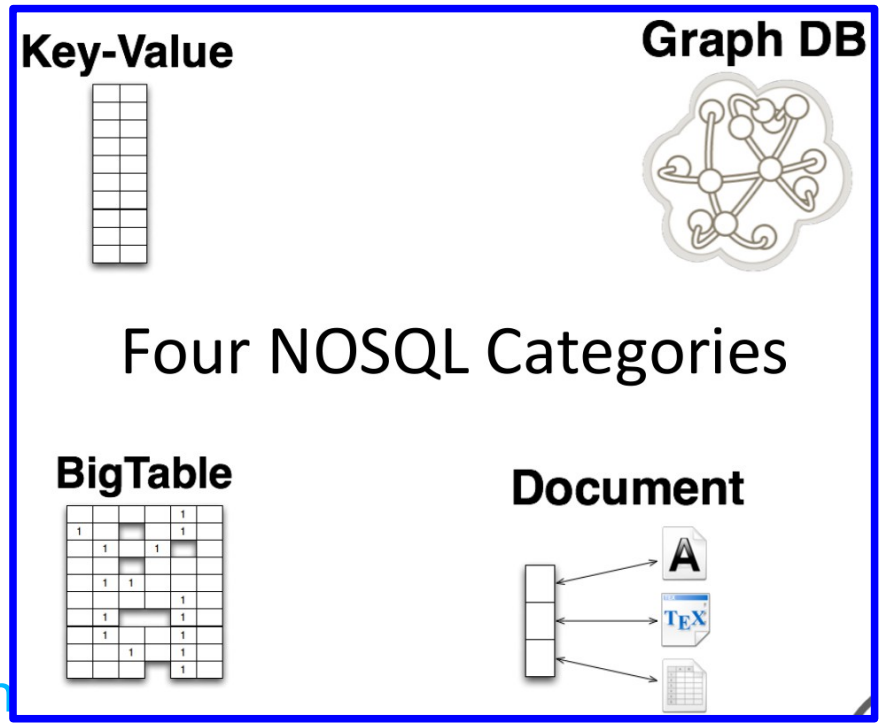
- Das zugrundeliegende Datenmodell ist nicht relational.
- Das System ist schemafrei oder hat nur schwächere Schemarestriktionen.

Skalierungsarchitektur

- Die Systeme sind von Anbeginn an auf eine verteilte und horizontale Skalierbarkeit ausgerichtet.
- Aufgrund der verteilten Architektur unterstützt das System eine einfache Datenreplikation.

(Mögliche) Kategorisierung

- (Noch) keine einheitliche Klassifikation – häufig verwendete Kategorisierung in Anlehnung an http://www.nosql-database.org/
- Core NoSQL Systems:
  - Key-Value Stores
  - Document Stores
  - Column Family Systeme
  - Graph Databases
- Neu (2012): Multimodel Databases
  - Soft NoSQL Systems:
  - Object Databases
  - Grid & Cloud Database Solution
  - XML Databases
  - Multivalue Databases
  - ...



Key/Value-Systeme

- **Datenmodell**
  - Key-Value-Paare mit eindeutigem Key („the big hash table“)
  - Key und Value enthalten Byte-Arrays = beliebige, serialisierte Datentypen (für value auch beliebig komplex)
  - Typische Grundoperationen:
    - set (key, value)
    - value = get (key)
    - delete (key)
- **Indexstrukturen**:
  - Hash-Maps, B*-Bäume auf key
- **Systeme**:
  - Amazon Dynamo / S3, Redis, Riak, Voldemort, ...

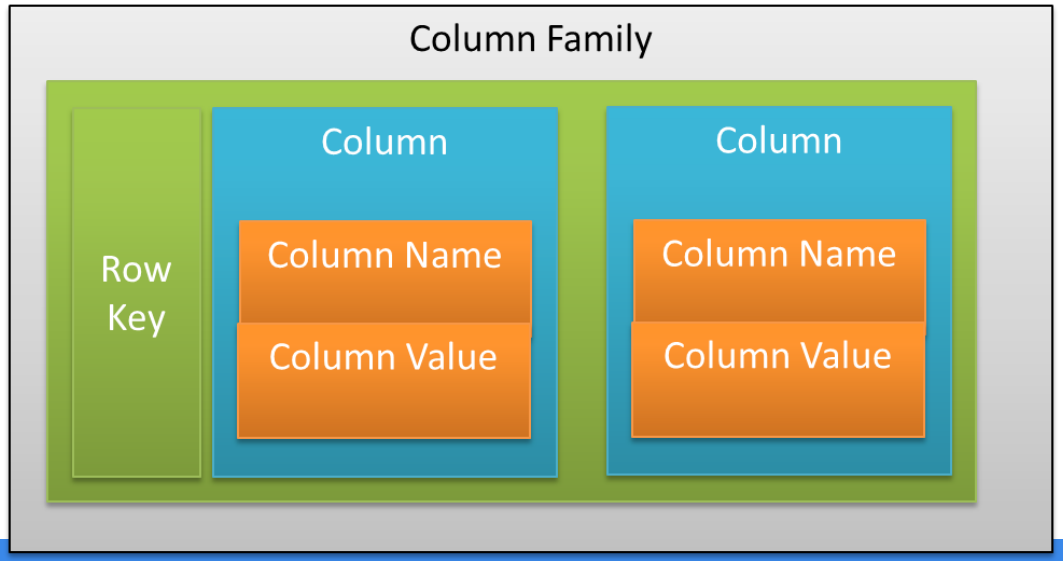
key	value
key	value
key	value
key	value
key	value

Key/Value-Systeme

- Strengths
  - Simple data model
  - Great at scaling out horizontally
    - Scalable
    - Available
- Weaknesses:
  - Simplistic data model
  - Poor for complex data

Column-Family-Systeme

- ähneln manchmal Excel-Tabellen
- beliebige Schlüssel können auf beliebig viele Key/Value-Stores angewendet werden
- Viele Systeme bieten Super-Columns
- Systeme:
  - Hbase, Cassandra, Hypertable, Google BigTable, ...



Beispiel: Column-Family-Systeme

Key	Tweets/00000000-0000-0000-0000-000000000001
Data	

Application	TweetDeck
Private	true
Text	Err, is this on?

Key	Tweets/00000000-0000-0000-0000-000000000002
Data	

App	TweetDeck
Public	true
Text	Well, I guess this is my mandatory hello
Version	1.2

```
get Keyspace1.Standard2['put_username']
set Keyspace1.Standard2['jsmith']['first'] = 'John'
       \            \         \         \          \
        \            \         \_ key    \          \_ value
         \            \                   \_ column
          \_ keyspace  \_ column family

get Keyspace1.Standard2['jsmith']
```

Column-Family-Systeme

- Strengths
  - Data model supports semi-structured data
  - Naturally indexed (columns)
  - Good at scaling out horizontally
- Weaknesses:
  - Unsuited for interconnected data

Document Stores

Datenmodell

- Kleinste logische Einheit: „Dokument“ identifiziert über documentID
- Format i.a. JSON, BSON, YAML, RDF
- Schemafrei, d.h. Anwendung übernimmt Schema-Verantwortung

```
{
   "id": 1,
   "name": “football boot",
   "price": 199,
   "stock": {
      "warehouse": 120,
      "retail": 10
   }
}
```

Indexstrukturen

- B-Baum-Index für documentID
- Teilweise auch B-Baum-Indexe für Datenfelder in Dokumenten

Systeme

- – MongoDB, CouchDB, Riak, ...

Document Stores

- Strengths:
  - Simple, powerful data model (just like SVN!)
  - Good scaling (especially if sharding supported)
- Weaknesses:
  - Unsuited for interconnected data
  - Query model limited to keys (and indexes)
    - Map reduce for larger queries

Difference between key-value and document stores

Key-Value Store

Key	Value
"India"	{"B-25, Sector-58, Noida, India – 201301"
"Romania"	{"IMPS Moara Business Center, Buftea No. 1, Cluj-Napoca, 400606",City Business Center, Coriolan Brediceanu No. 10, Building B, Timisoara, 300011"}
"US"	{"3975 Fair Ridge Drive. Suite 200 South, Fairfax, VA 22033"}

Document Store

The data which is a collection of key value pairs is compressed as a document store quite similar to a key-value store, but the only difference is that the values stored (referred to as "documents") provide some structure and encoding of the managed data. XML, JSON (Java Script Object Notation), BSON (which is a binary encoding of JSON objects) are some common standard encodings.

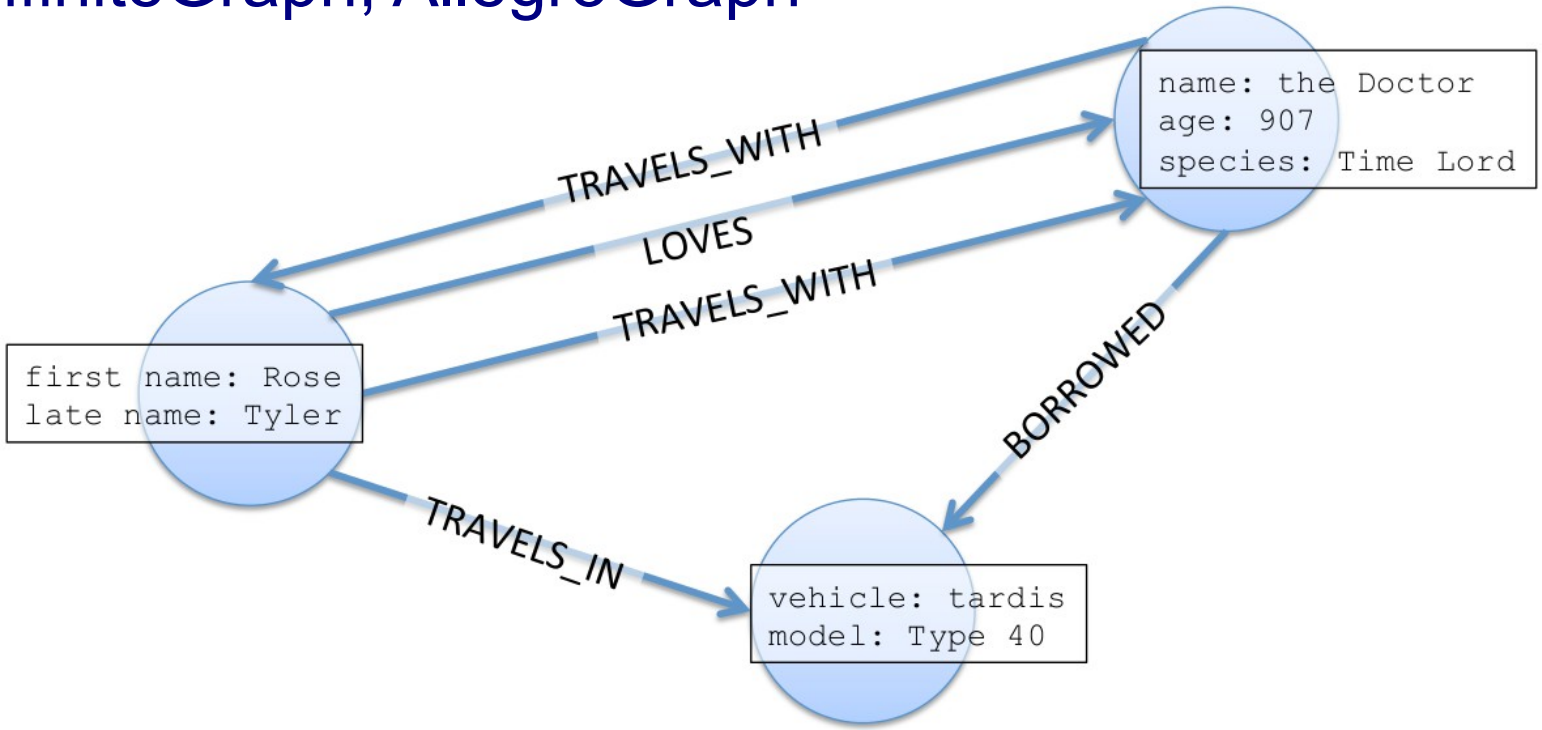
The following example shows data values collected as a "document" representing the names of specific retail stores. Note that while the three examples all represent locations, the representative models are different.

```
{officeName:"3Pillar Noida",
{Street: "B-25, City:"Noida", State:"UP", Pincode:"201301"}
}
{officeName:"3Pillar Timisoara",
{Boulevard:"Coriolan Brediceanu No. 10", Block:"B, Ist Floor", City:
}
{officeName:"3Pillar Cluj",
{Latitude:"40.748328", Longitude:"-73.985560"}
}
```

Quelle: http://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases

Graphdatenbanken

- Graph- oder Baumstrukturen
- Die Knoten und Kanten können Eigenschaften haben
- Systeme: Neo4j, Sones GraphDB, OrientDB, InfiniteGraph, AllegroGraph



Graphdatenbanken

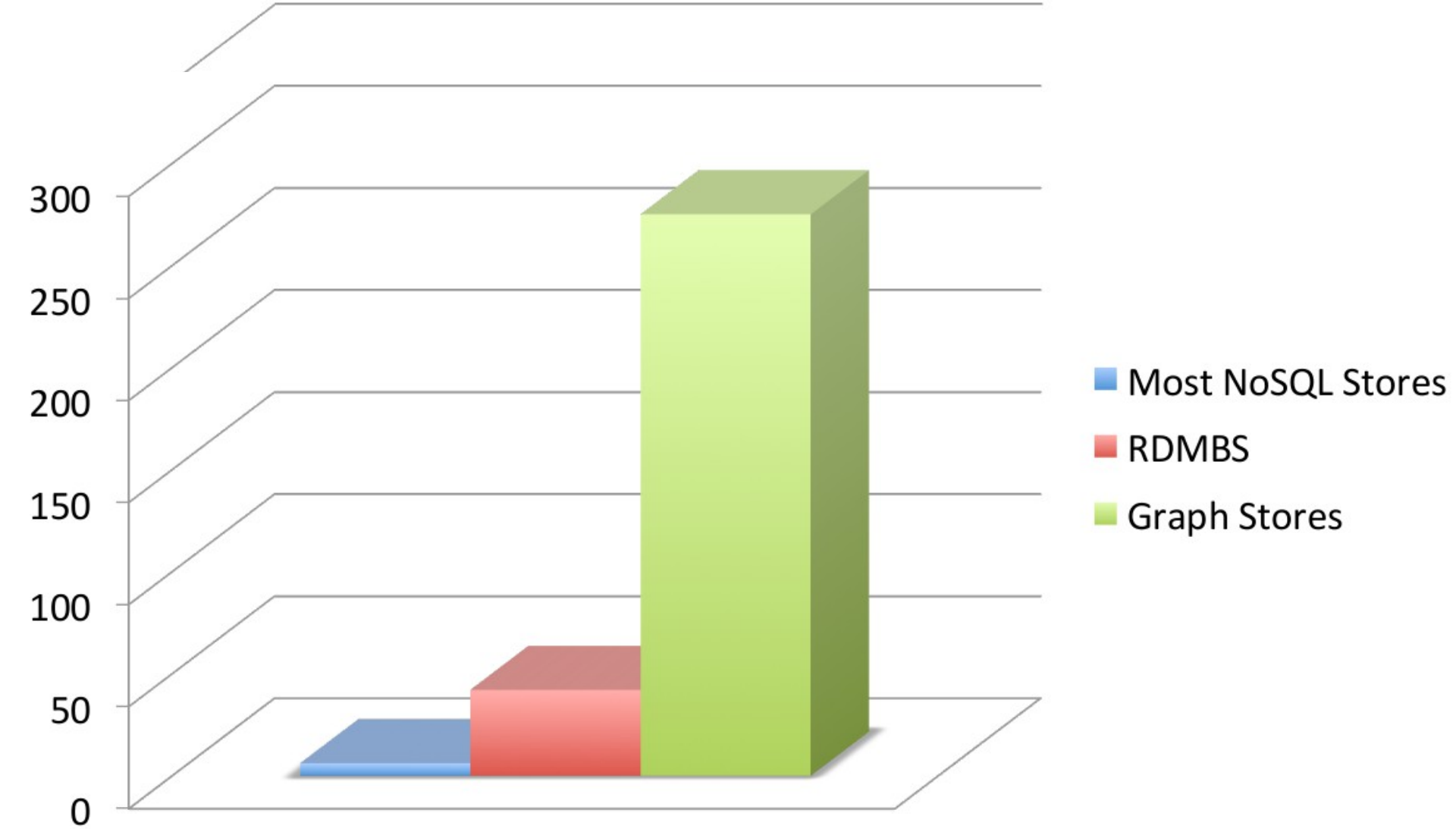
- Strengths
  - Powerful data model
  - Fast
    - For connected data, can be many orders of magnitude faster than RDBMS
- Weaknesses
  - Sharding (sharding, where a large dataset can be broken up and distributed across a number of (typically replicated) shards.)
    - Sharding graphenorientierter Datenbanken ist umständlich

Graphdatenbanken

Meet Leonhard Euler

- Swiss mathematician
- Inventor of Graph Theory (1736)

On maturity of data models



Fehlender Reifegrad bei noSQL Systemen

NoSQL: Die Essenz

Datenmodell

- Das zugrundeliegende Datenmodell ist nicht relational.
- Das System ist schemafrei oder hat nur schwächere Schemarestriktionen.

Skalierungsarchitektur

- Die Systeme sind von Anbeginn an auf eine verteilte und horizontale Skalierbarkeit ausgerichtet.
- Aufgrund der verteilten Architektur unterstützt das System eine einfache Datenreplikation.

Scale up vs. Scale out

Scale up: wenige, große Server

Quelle: ibm.com

Scale out: viele, kleinere (Commodity-)Server

Quelle: eggmusic.com

Scale up vs. Scale out

Scale up

- Vorteile:
  - transparent für DBMS
  - Administrations- aufwand konstant
- Nachteile:
  - Hardware-Kosten
  - Skalierung nur in größeren Stufen möglich → höhere Kosten und ungenutzte Leistung

Scale out

- Vorteile:
  - Kostengünstigere Hardware
  - Skalierung in kleineren Stufen möglich
- Nachteile:
  - Last- und Datenverteilung notwendig
  - Ggf. verteilte Protokolle (2PC, Replikation)
  - Erhöhte Fehlerrate (mehr und einfachere Hardware)
  - Erhöhter Administrationsaufwand

Scale out: CAP-Theorem

Eigenschaften verteilter Datensysteme:

- **Consistency (Konsistenz)**: alle Clients (Anwendungen) haben die gleiche Sicht auf den Datenbestand – auch im Fall von Updates
- **Availability (Verfügbarkeit)**: jeder Request an einen non-failing Knoten führt zu einer Antwort, d.h. ausgefallene Knoten beeinflussen nicht die Verfügbarkeit der anderen Knoten.
- **Partition Tolerance (Ausfalltoleranz):** Systemeigenschaften bleiben auch bei Partitionierung des Netzwerks erhalten (d.h. Knoten können weiter funktionieren auch wenn die Kommunikation mit anderen Knotengruppen verloren gegangen ist)
- → **CAP Theorem** (Eric Brewer, 2000): in verteilten Datensystemen sind zu jeder Zeit nur maximal zwei dieser drei Eigenschaften erreichbar

BASE Konsistenzmodell

- Alternatives Konsistenzmodell zur Lösung des Konflikts des CAP-Theorems
  - Basically Available
  - Soft State
  - Eventually Consistent
- Gegenpart zum klassischen ACID (Atomicity, Consistency, Isolation, Durability)

BASE Consistency Model

Consistency model weaker than **ACID**

Atomicity
Consistency
Isolation
Durability

BASE = Basically Available, Soft state, Eventual consistency

- If a node fails, part of the data will not be available, but the entire data layer stays operational
- The state of the system may change over time, even without input
- The system becomes consistent at some later time

Konsistenz in AP-Systemen?

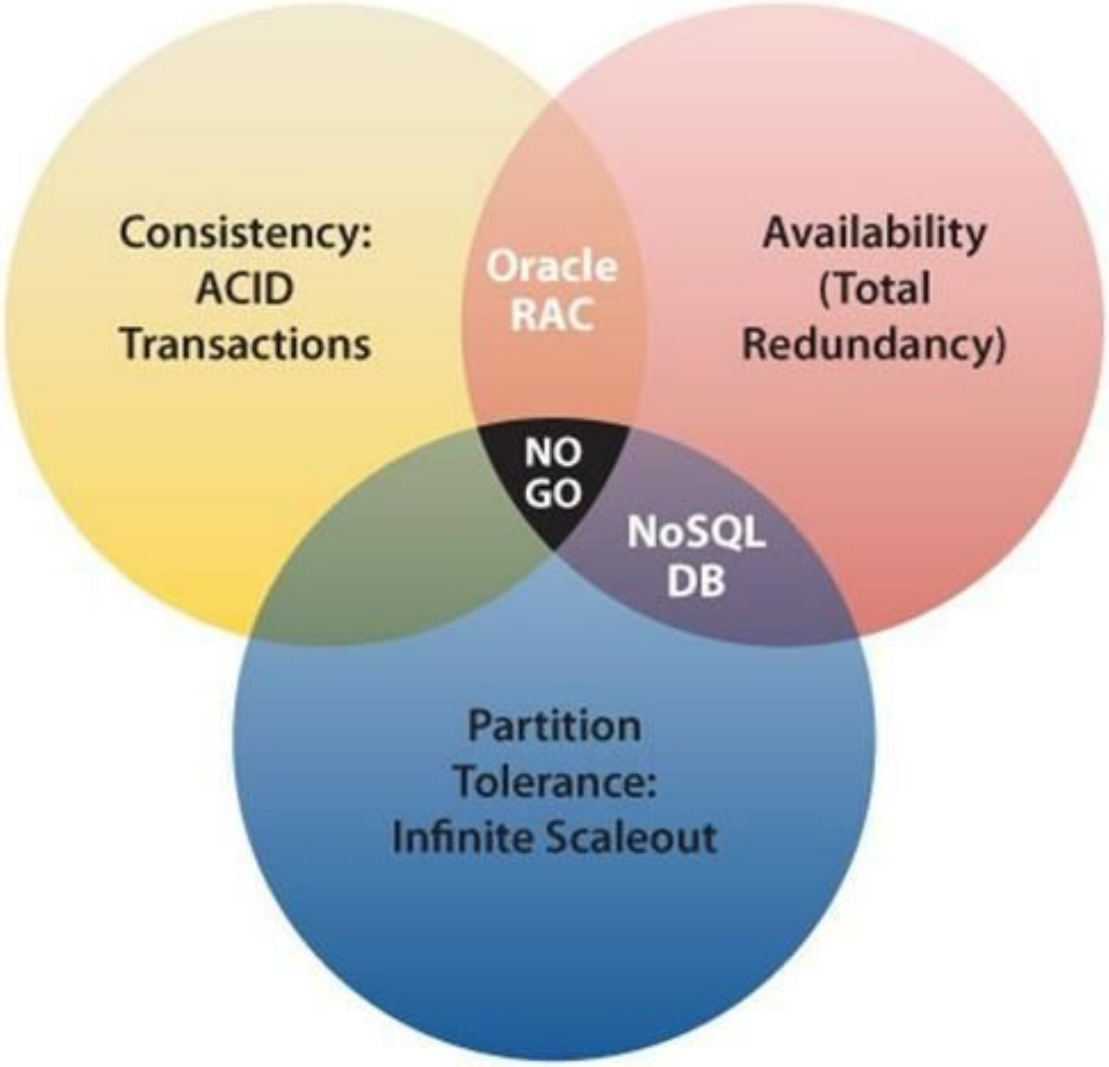
Strong Consistency

- Nach Abschluss eines Updates sehen alle nachfolgenden Zugriffe (auch an anderen Knoten!) den aktuellen Wert (entspricht C in ACID)

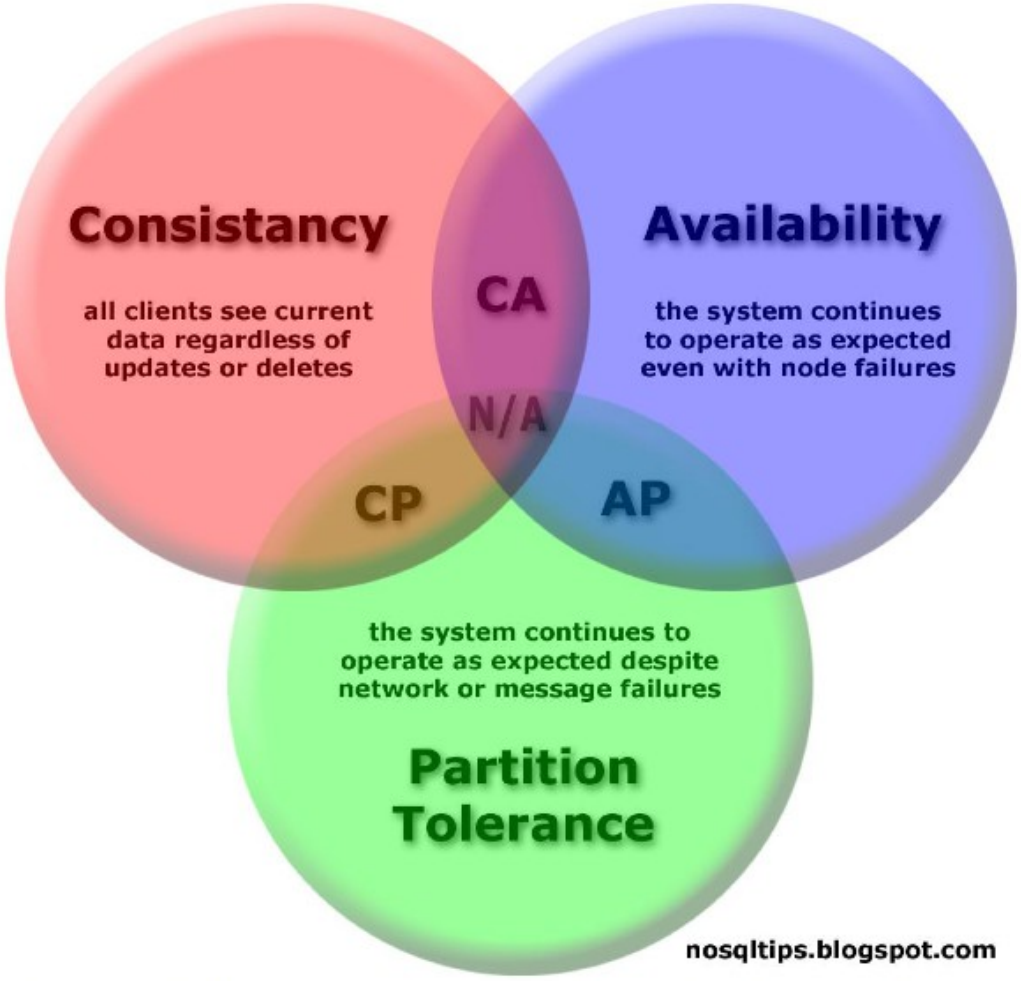
Weak Consistency

- Es ist nicht garantiert, dass nachfolgende Zugriffe den aktuellen Wert sehen
- Spezialform: Eventual Consistency (Vogel, 2008): Es ist garantiert, dass nach einem Zeitfenster schlussendlich (eventually) alle Zugriffe den aktuellen Wert sehen (falls zwischenzeitlich keine weiteren Updates erfolgen)

ACID vs. BASE


Consistency:
ACID
Transactions
Oracle
RAC
Availability
(Total
Redundancy)
NO
GO
NoSQL
DB
Partition
Tolerance:
Infinite Scaleout

CAP-Theorem – Two out of three



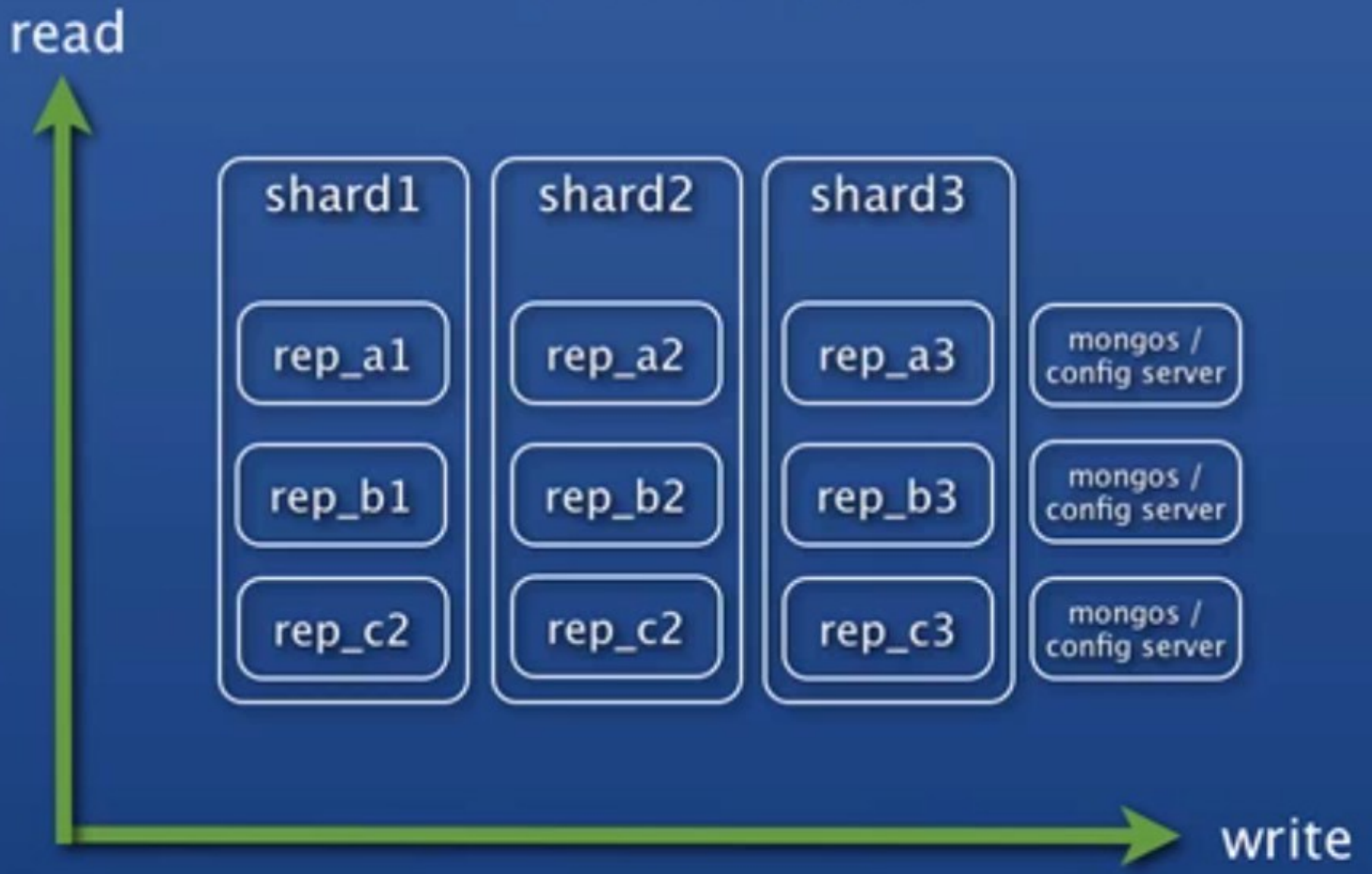
- Um hoch zu skalieren muss man Partitionieren → man muss zwischen Konsistenz und Verfügbarkeit wählen

Eventual Consistency aus Sicht des Client

Praktisch relevante Spezialformen der Eventual Consistency:

- **Read-your-writes Consistency**
  - Jeder Prozess sieht seine eigenen Änderungen (niemals ältere Werte für die geänderten Objekte)
- **Session Consistency**
  - Read-your-writes innerhalb einer Session
- **Monotonic Read Consistency**
  - Wenn ein Prozess einen Wert gelesen hat, sieht er danach nie einen älteren Wert für dieses Objekt.
- **Monotonic Write Consistency**
  - Die Schreiboperationen einer Transaktion werden vom System serialisiert

Scale out


read
shard1
shard2
shard3
rep_a1
rep_a2
rep_a3
rep_b1
rep_b2
rep_b3
rep_c2
rep_c2
rep_c3
mongos / config server
mongos / config server
mongos / config server
write

… to be continued

Weiterführende Links

- http://www.icefaces.org/main/resources/tutorials.iface
- http://component-showcase.icefaces.org/component-showcase/showcase.iface
- !!! http://www.mkyong.com/jsf2
- http://christopher.over-blog.de/article-25780779.html
- http://www.slideshare.net/cagataycivici/jsfandsecurity
- http://e-blog-java.blogspot.com/2011/02/glassfish-based-authentication-with-jsf.html
- https://www.opends.org/wiki/page/GlassfishApplicationServer
- http://facestutorials.icefaces.org/tutorial/validators-tutorial.html#applevelvalidation
- !!! http://www.wi-bw.tfh-wildau.de/~hendrix/JEE_neu.ppt
- !!! http://www.wi-bw.tfh-wildau.de/~hendrix/EJB3.ppt
-
-

Quellen

- Störl, Uta, Hochschule Darmstadt, Vortrag „NoSQL-Datenbanksystem: Revolution oder Evolution“
- Edlich et al, NoSql – Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken, Hanser Verlag, 2. Aufl., 2011
- http://ayende.com/blog/4500/that-no-sql-thing-column-family-databases


HTL LEONDING
Schön, hier zu lernen.