NoSQL

Hintergründe und Anwendungen

Andreas Winschu

Inhalt

1. **Motivation**
2. **RDBMS**
3. **CAP Theorem**
4. **NoSQL**
5. **NoSql Overview**
6. **NoSQl Praxis**
7. **Zusammenfassung und Ausblick**

1.Motivation

Datenbanken

- Permanente Sicherung großer Datenmengen
- Effizientes Wiederfinden der Daten (Indexierung)
- Analytische Untersuchungen
- Gleichzeitiges Arbeiten von mehreren Benutzern
- Schutz vor unautorisiertem Zugriff

1.Motivation

Geschichte

60er Jahre

Hierarchisch/Netzwerkartig

- Zeigerstrukturen zwischen Daten
- navigierende Anfragesprachen

70er Jahre

Relational

- Daten in Tabellenstruktur
- standardisierte Datenbanksprache(SQL)

1.Motivation

Geschichte

60er Jahre

Hierarchisch/Netzwerkartig

- Zeigerstrukturen zwischen Daten
- navigierende Anfragesprachen

70er Jahre

Relational

- Daten in Tabellenstruktur
- standardisierte Datenbanksprache(SQL)

1.Motivation

- Relationale Datenbanken sind 40 Jahre alt
- One Size fitts all
  auf alle Probleme angewendet
- Applikationen benutzen ein anderes Datenmodell
- Einige Applikationen haben weniger strikte Anforderungen an Daten

2.RDBMS

Bücher

Buch-ID	Autor	Verlag	Verlagsjahr	Titel	Datum
1	Hans Vielschreiber	Musterverlag	2007	Wir lernen SQL	13.01.2007
2	J. Gutenberg	Gutenberg und Co.	1452	Drucken leicht gemacht	01.01.1452
3	G. I. Caesar	Handschriftverlag	-44	Mein Leben mit Asterix	16.03.1944
5	Galileo Galilei	Inquisition International	1640	Eppur si muove	1641
6	Charles Darwin	Vatikan-Verlag	1860	Adam und Eva	1862
7	Christian Schade	KVG	2008	SQL für Dummies	04.06.2007

BücherNutzer

Nutzer-ID	Buch-ID
10	1
10	2
10	3
12	5
12	6

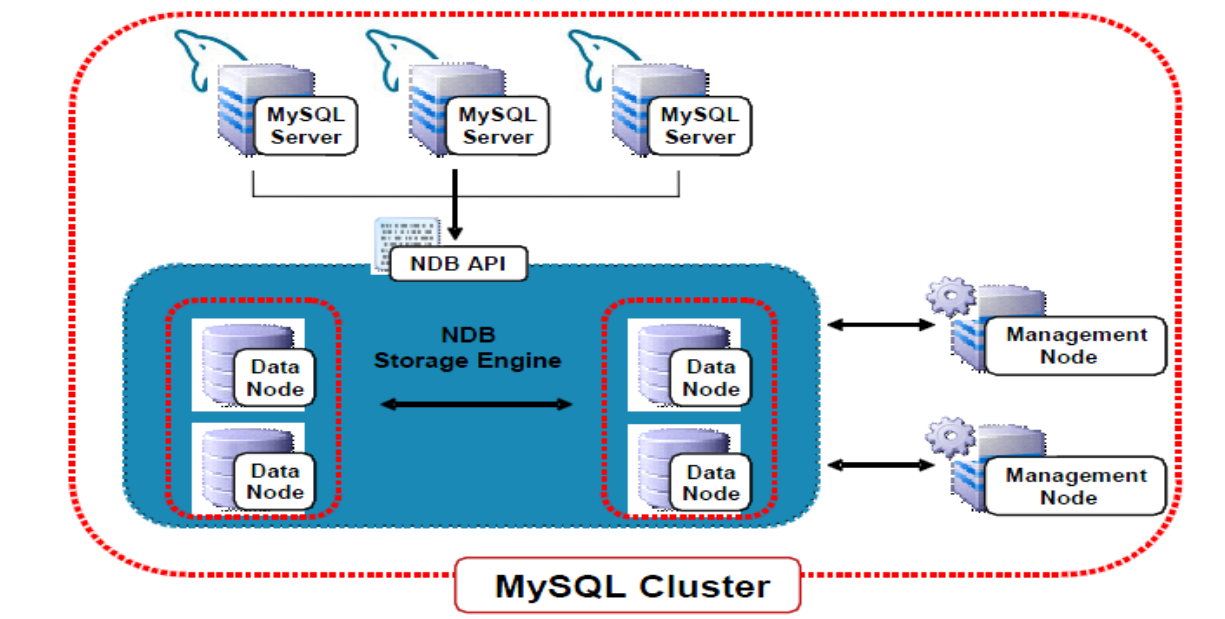
Nutzer

Nutzer-ID	Vorname	Nachname
10	Hans	Vielleser
11	Jens	Mittelleser
12	Erich	Wenigleser

2.RDBMS

Scaling

„Enterprise“ Solutions

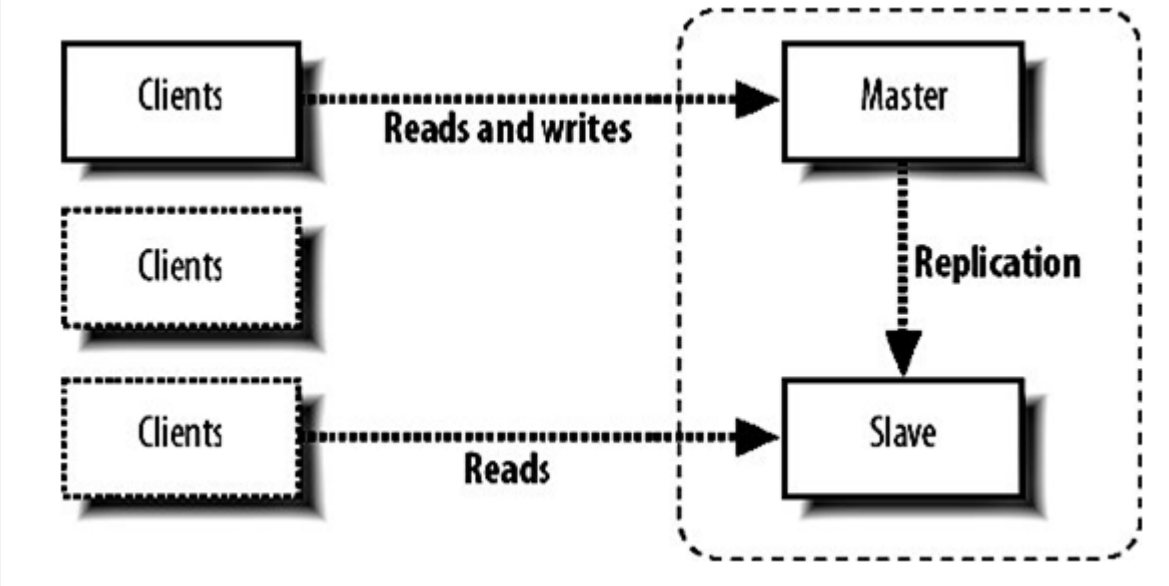


+ Keine Applikationslogik erforderlich

- Teure „Enterpriselösungen“
- Aufwendige JOIN Algorithmen,
  Eignung nur bei „Einfachen JOINs“.

2.RDBMS

Master Slave Replication

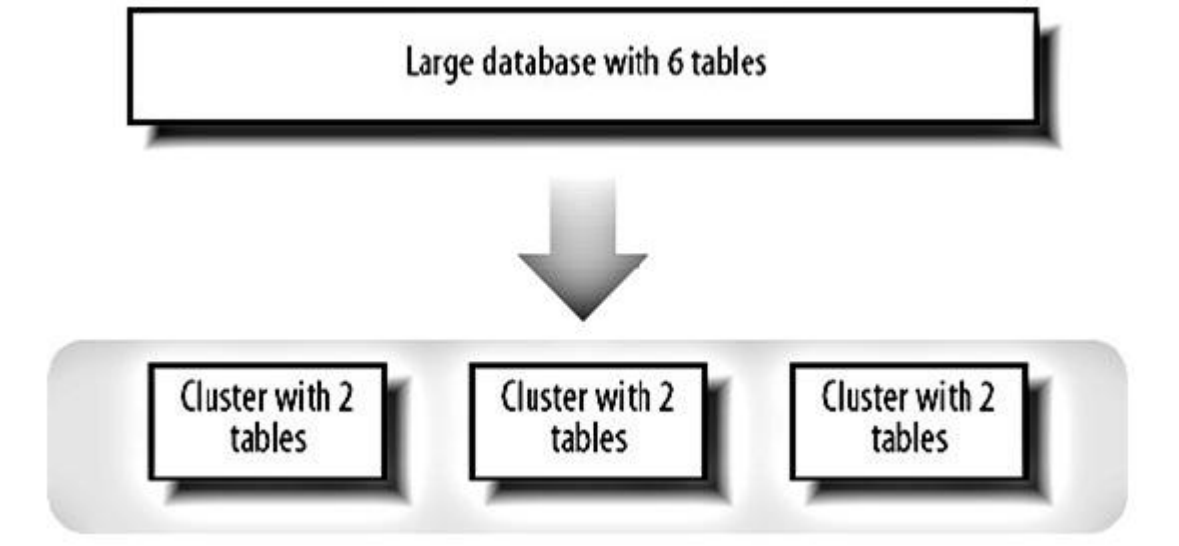


+ Vergleichbar weniger Administration
+ In den Basispacketen vorhanden

- Keine Datenpartionierung
- Schreibzugriffe skalieren nicht

2.RDBMS

Functional Partitioning (Sharding)



- Immens hohe Verteilungslogik auf der Applikationsebene
- Erfordert abgetrennte Datenbereiche
- Denormalisierung
  Verwendung des RDBMS gegen eigentliches Funktionsdesign

2.RDBMS

Transaktionen

Atomicity (Atomarität):
Transaktion wird entweder ganz oder gar nicht ausgeführt

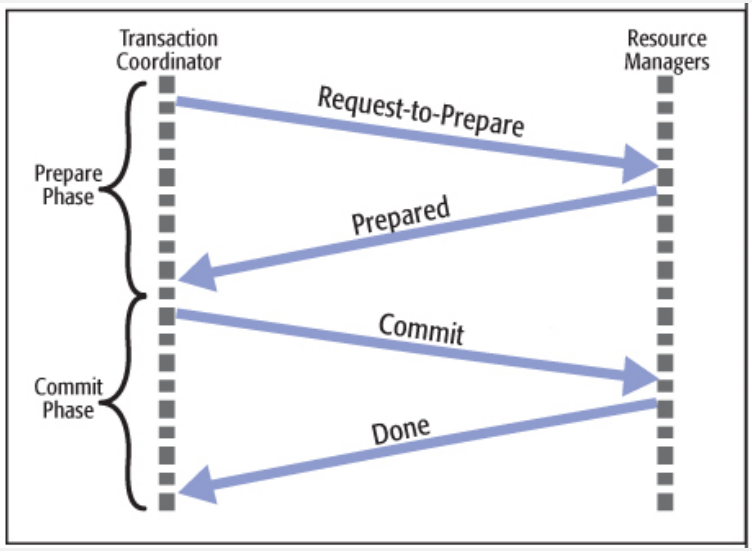
Consistency (Konsistenz / Integritätserhaltung):
Datenbank ist vor Beginn und nach Beendigung einer Transaktion jeweils in einem konsistenten Zustand

Isolation (Isolation):
Nutzer, der mit einer Datenbank arbeitet, sollte den Eindruck haben, dass er mit dieser Datenbank alleine Arbeitet

Durability (Dauerhaftigkeit / Persistenz):
nach erfolgreichem Abschluss einer Transaktion muss das Ergebnis dieser Transaktion „dauerhaft" in der Datenbank gespeichert werden

2.RDBMS

- Garantie der ACID Eigenschaften wird komplex
- **Atomicity** fordert Verteiltes Commit Protokoll (e.g. 2PC)
  Alle Nodes müssen sich einig sein
- **Isolation** fordert Erhaltung der Locks für die gesamte Dauer des Protokolls
  die gesamte Transaktion darf nicht gestört werden durch nebenläufige Transaktionen



Leichtgewichtige Transaktionen, große Netzwerk Roundtrips =>

- Lockerhaltung länger als Arbeitszeit
- „Skyrocketing Lock Competitions“
- Schwund des Transaktionsdurchsatzes

2.RDBMS

Features:

- Festes Datenmodell
- Redundanzvermeidung durch Fremdschlüsselbeziehungen
- Starke Konsistenz
- Transaktionsmodell Garantien

Eignung:

- Finanzprozesse
- ERP Systeme
- Systeme mit wenigen Benutzern

Schwierigkeiten:

- Horizontale Scalierung
- JOIN‘s kosten CPU Power
- Datenmodel passt nicht immer ins relationale Schema

One Size does **NOT** fit all!

3.CAP Theorem

Erik Brewer überlegt 1997 ein Gegenmodel zu **ACID**
BASE – Basically **A**vailable, **S**oft-state, **E**ventually consistent

ACID

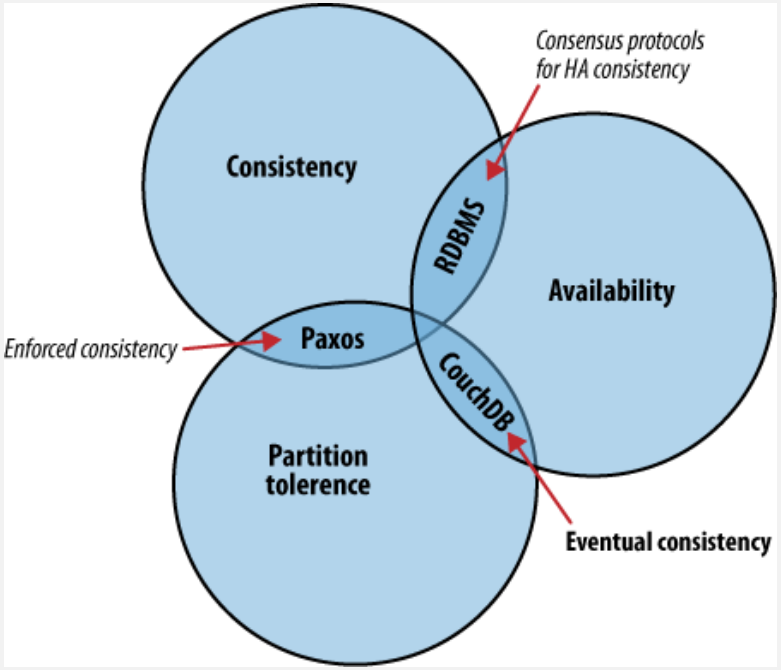
- Starke Konsistenz
- Isolation
- Fokus auf Commit
- Geschachtelte Transaktionen
- Konservativ (pessimistisch)
- Schwierige Umsetzung
- (e.g. Schema, Normalesierung, JOIN‘s)

vs

BASE

- Schwache Konsistenz – altes data OK
- Availability zuerst
- Versuch dein Bestes
- Aggresiv (optimisticsch)
- Leichter!
- Schneller
- Einfacher Umsetzbar

3.CAP Theorem



Erik Brewer stützt notwendigkeit von **BASE** durch ein Theorem

Linear **Consitency**
Totale Ordnung aller Operationen

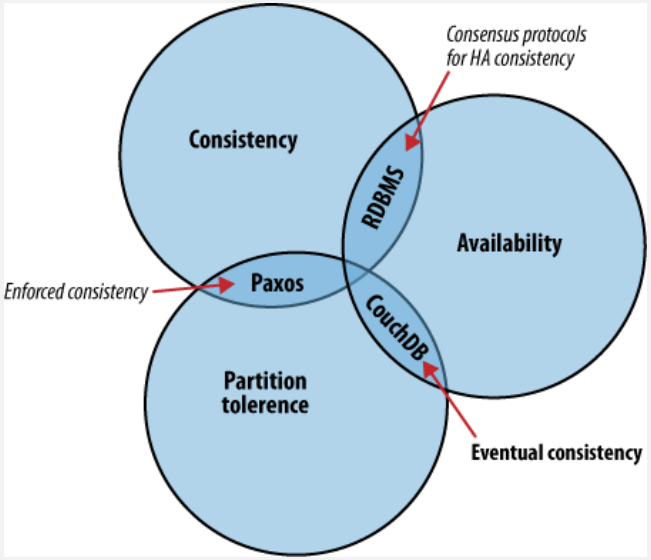
Availibility
Jeder Anfrage an einen intakten Node wir bearbeitet

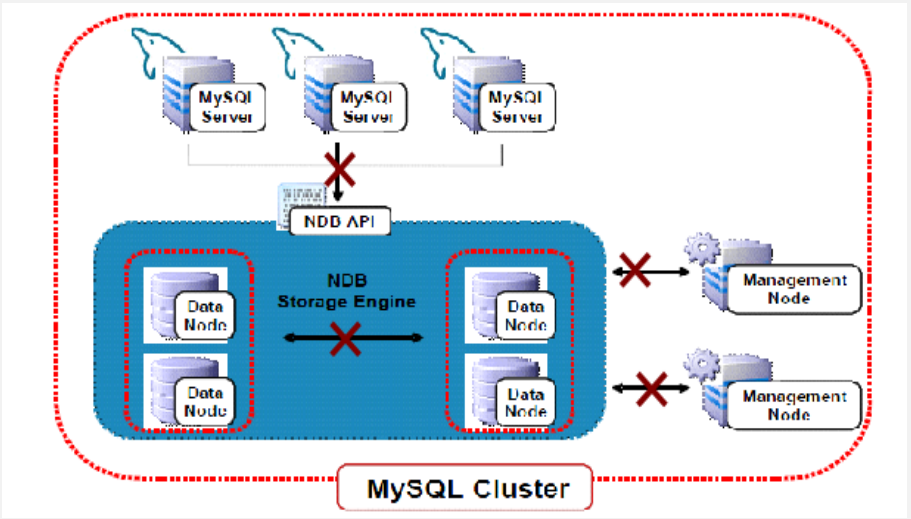
Partitiotolerence
Fehler bei der Kommunikation im verteilten System werden toleriert

Behauptung: nur 2 der 3 Features gleichzeitig erfüllbar
Bewiesen durch Gilbert und Lynch

3.CAP Theorem

AP Class





Es gilt:

a) Das System möchte Partiontoleranz erfüllen
b) Es treten Netzwekfehler auf.

Wenn:

Das System beantwortet zu diesem Zeitpunkt alle Anfragen (Availibility)

Dann:

Die Antworten können nicht konsistent sein, da keine Einigkeit herscht

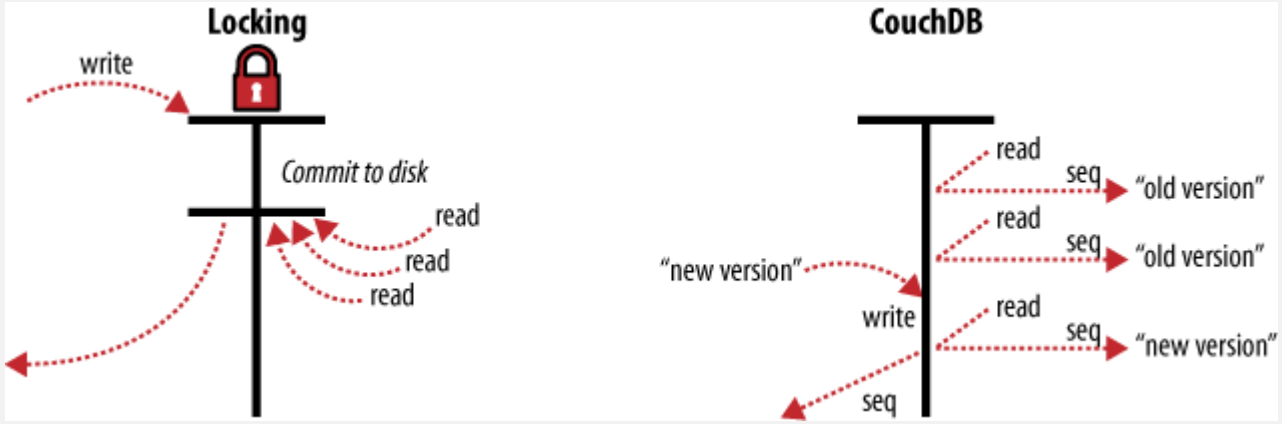
3.CAP Theorem

AP Class

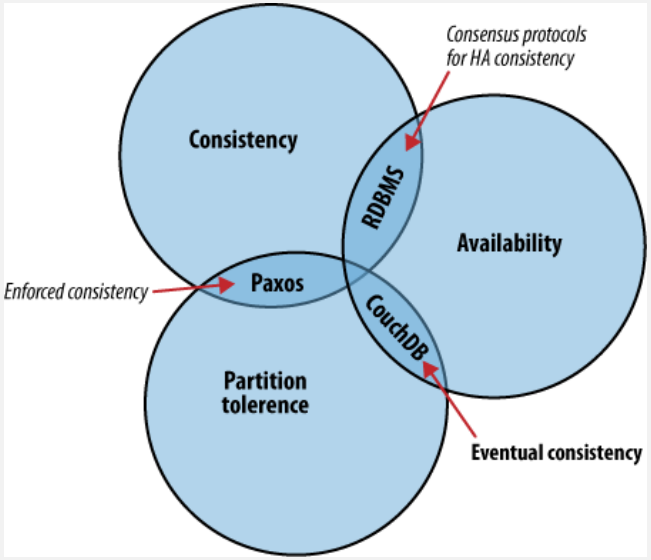
Eventual Consistency

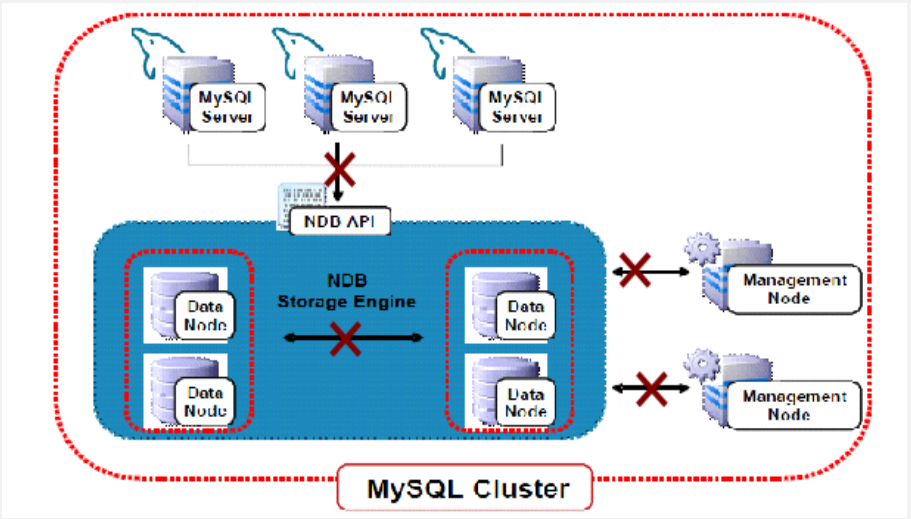
Wenn keine neuen Updates auftreten liefert jeder Zugriff eventuell den aktuellen Wert

- Keine Locks
- Local Consistency
- „Read your own Writes“
- Multi Version Concurency Control (MVCC)



3.CAP Theorem





Es gilt:

a) Das System möchte Partiontoleranz erfüllen
b) Es treten Netzwekfehler auf.

Wenn:

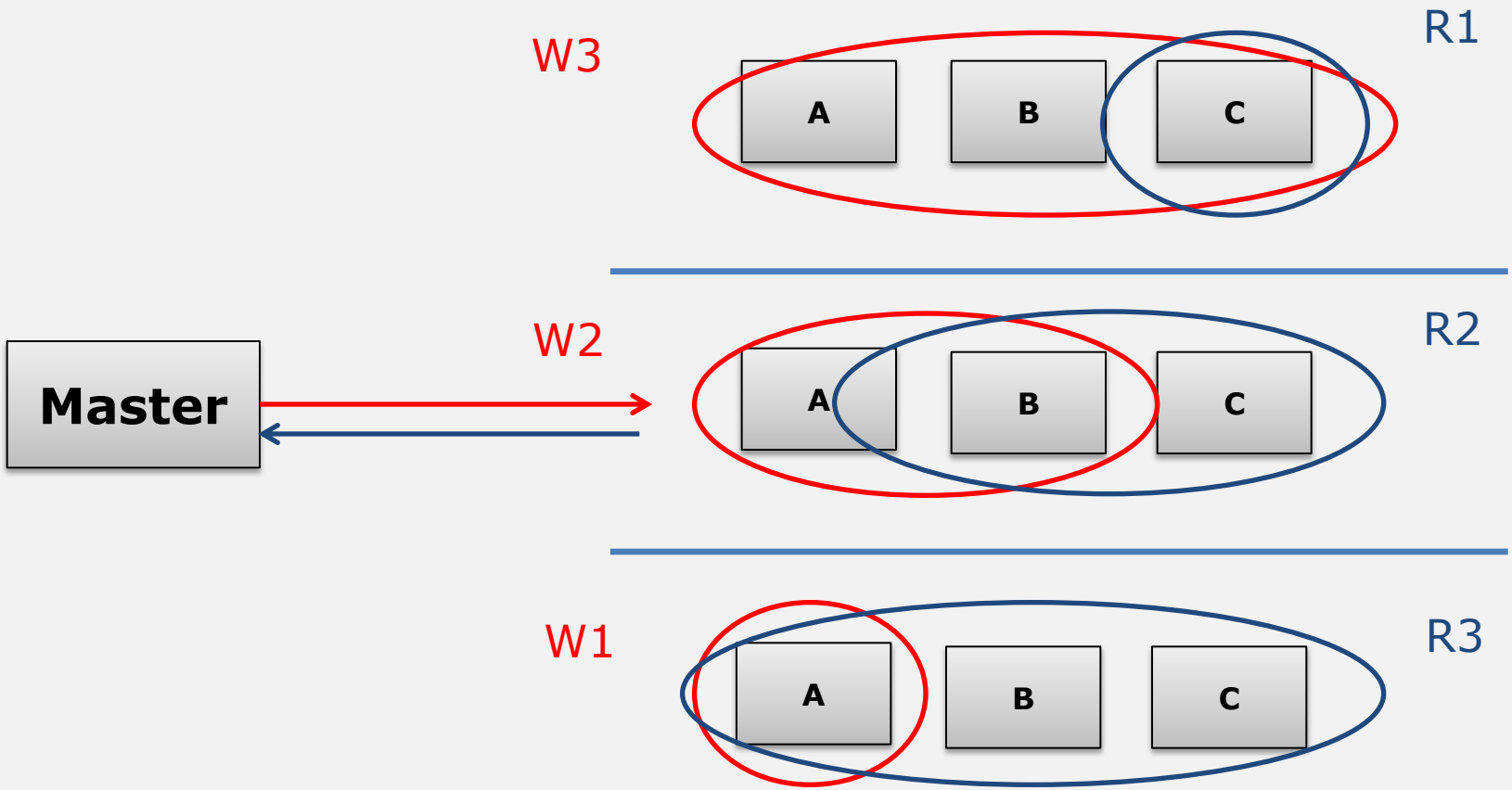
Alle Benutzer des Systems erhalten korrekte und gleiche Daten

Dann:

Das System beantwortet Anfragen solange nicht bis es sich synchronisiert hat

3.CAP Theorem

CP Class

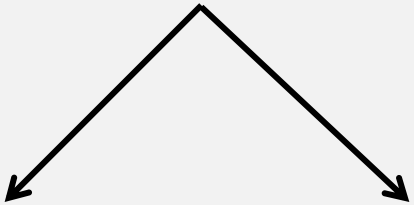


4.NoSQL

- Begriff als erstes 1998 durch Carlo Strozzi verwendet
  - relationale Datnabank, die explizit auf SQL verzichtet
  - No to SQL
- 2009 durch Johan Oskarsson aufgegriffen
  - Last.fm Tagung
  - Techniken für nicht-relationale verteilte Datenbanken
  - Keine neue Technologie
  - Aufleben bekannter BASE Konzepte gemäß aktuellen Anfordrerungen
- NoSQL Bewegung ensteht
  - NoSQL = Not only SQL

4.NoSQL

- Nicht relational
- Schemafrei
- Einfache API's
  (meistens)Verizicht auf SQL, JavaScript, JSON, Views, REST
- Horizontal scalierbar (Shards)
- Map/Reduce Paradigma zu parallelen Verarbeitung
- Replikation zwischen den Schards
- Partition tolerant



AP
Eventual Consistency
MVCC

CP
Write/Read Availabillity Balance
Update in Place

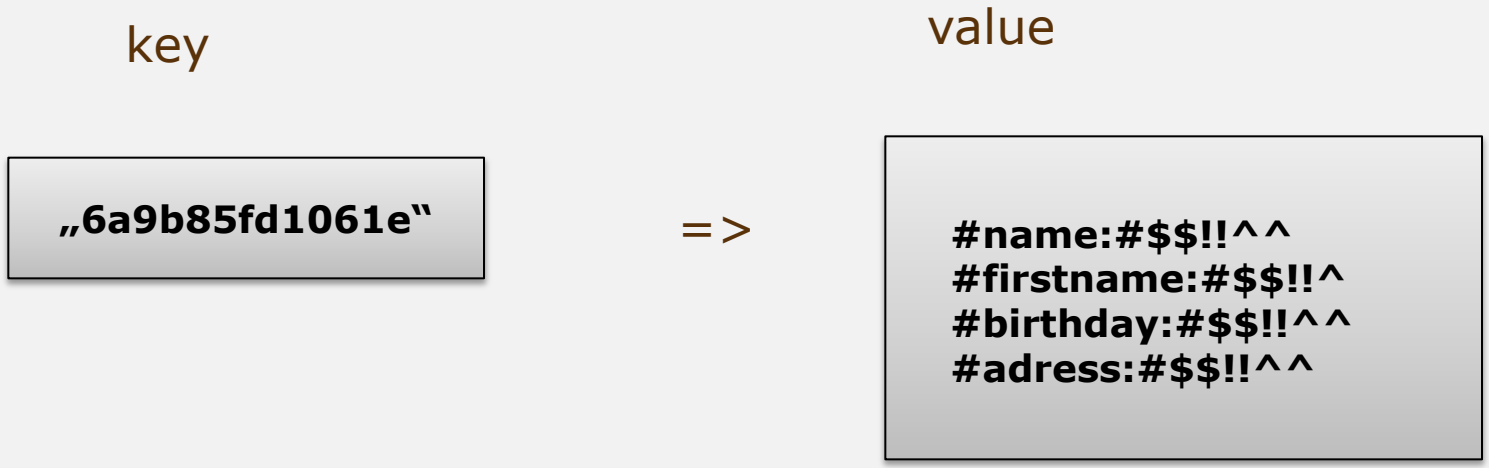
4.NoSQL - Overview

Key/Value Store

- einfache Datenhalter
- Ein Key, Ein Value, Keine Duplikate
- Sehr Schnell
- Verteilter Hash
- Die Datenbank versteht die Values nicht
  BLOB
- Voldemort, Redis, Tokio Cabinet

5.NoSQL - Overview

Key/Value Store

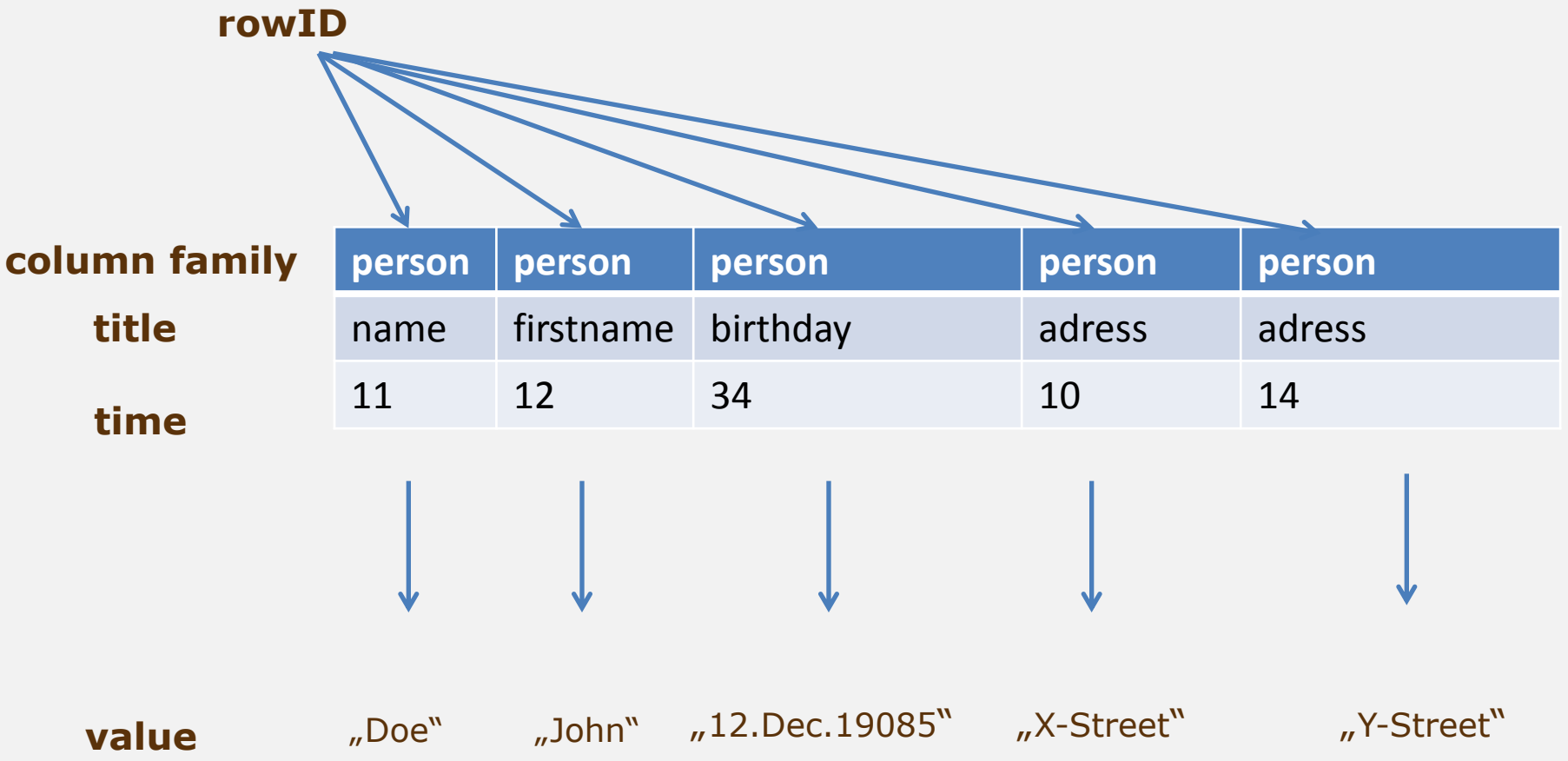


Column-based Store

- Spalten als einfachste Datenhalter
- Verteilte, multidimensonale, sortierte Map
- GoogleBigTable Clones
- GoogleBigTable, Cassandra (Facebook), Apache Hbase
- Angeblich besser bei Analytical Processing, Datawarehouse

4.NoSQL - Overview

Column Store

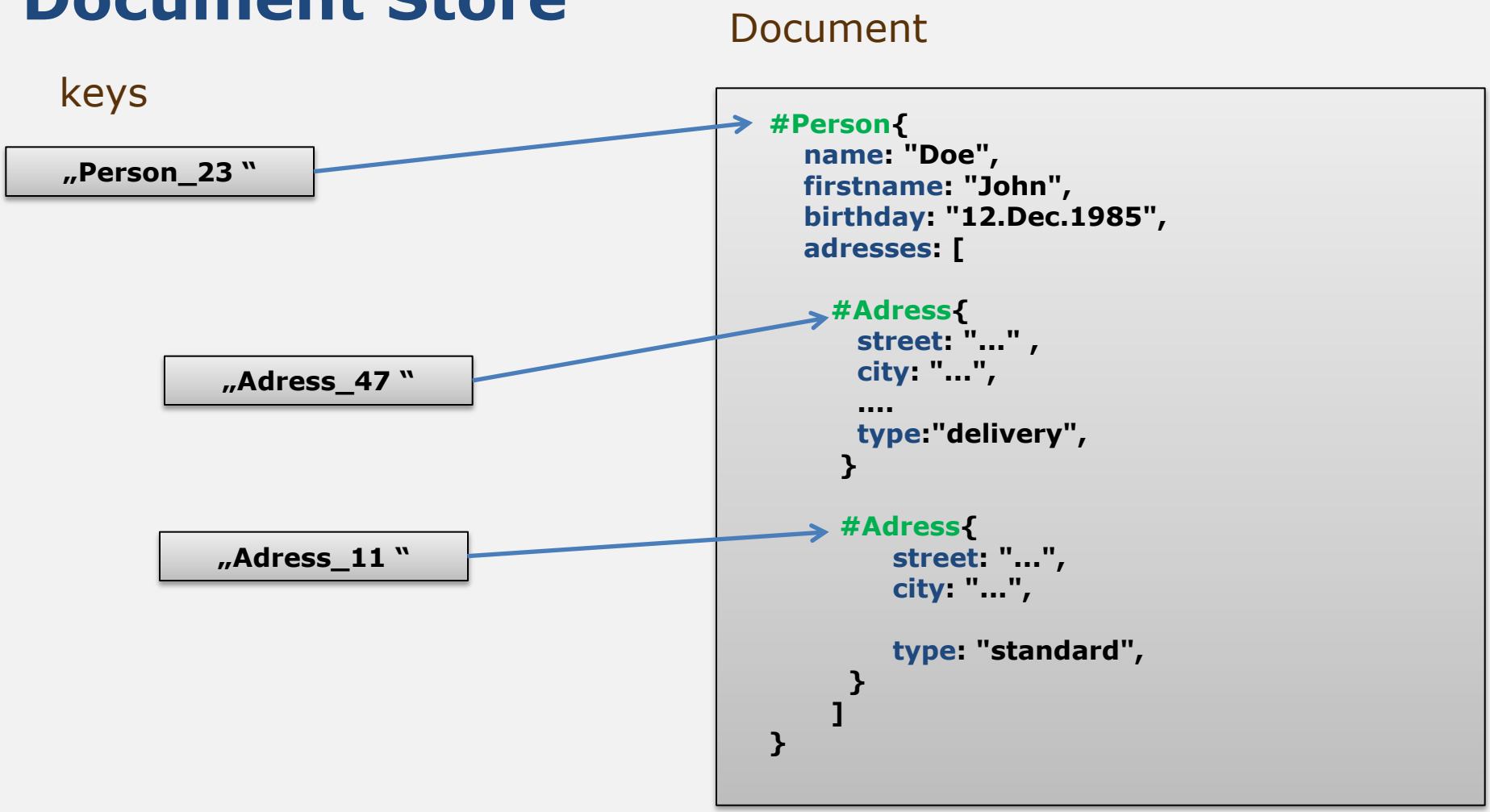


Document Stores

- Key Value Store, aber das Value ist strukturiert und von der DB interpretierbar (Document)
- Datenanfragen nicht nur über Keys, sondern über alle Documentfelder
- Indexes über einzelne Documentfelder
- CouchDB, MongoDB

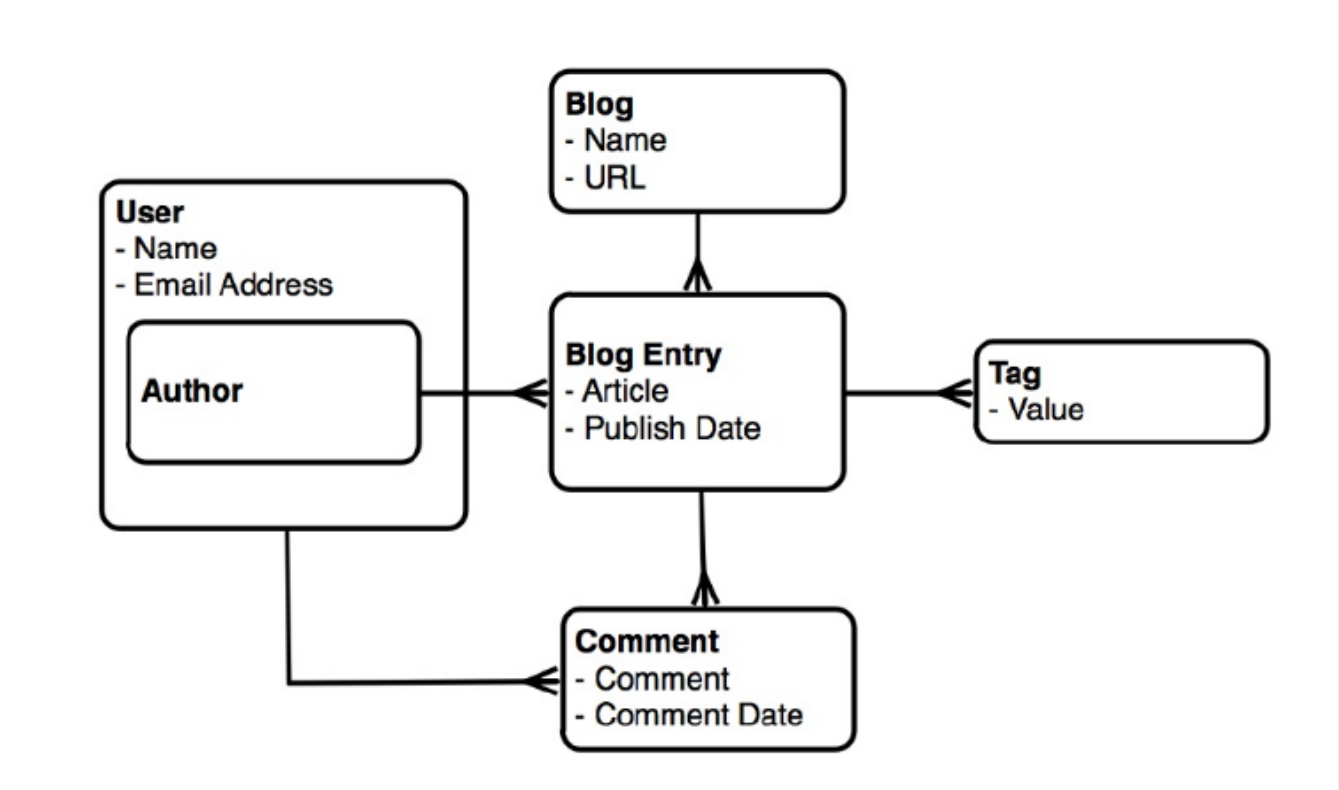
5.NoSQL - Overview

Document Store



Schema Design

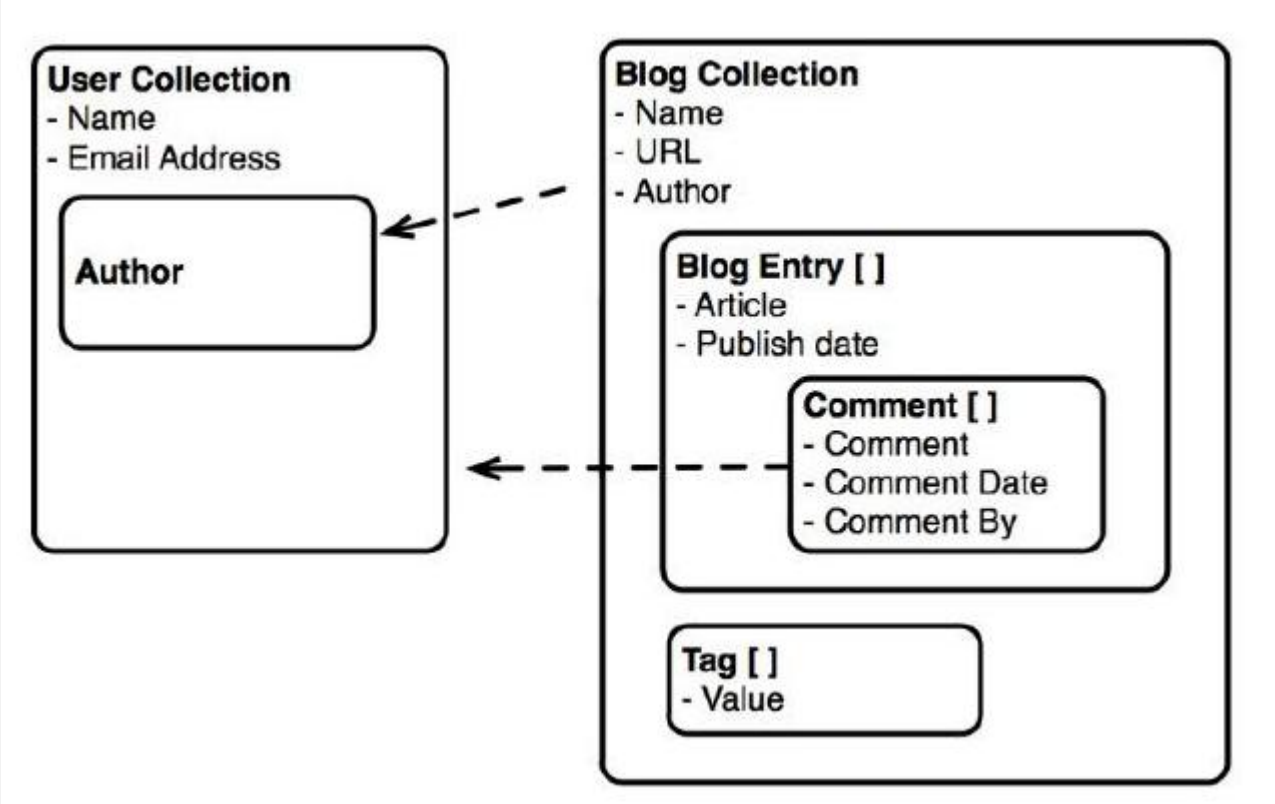
Relational



6.NoSQL Praxis

Schema Design

Denormalisiert(„NoSQL Way")



Embed bevorzugt über **Reference**

6.NoSQL Praxis

Queries

REST

http://127.0.0.1:5984/mydb/_design/blog
GET, PUT, POST, DELETE

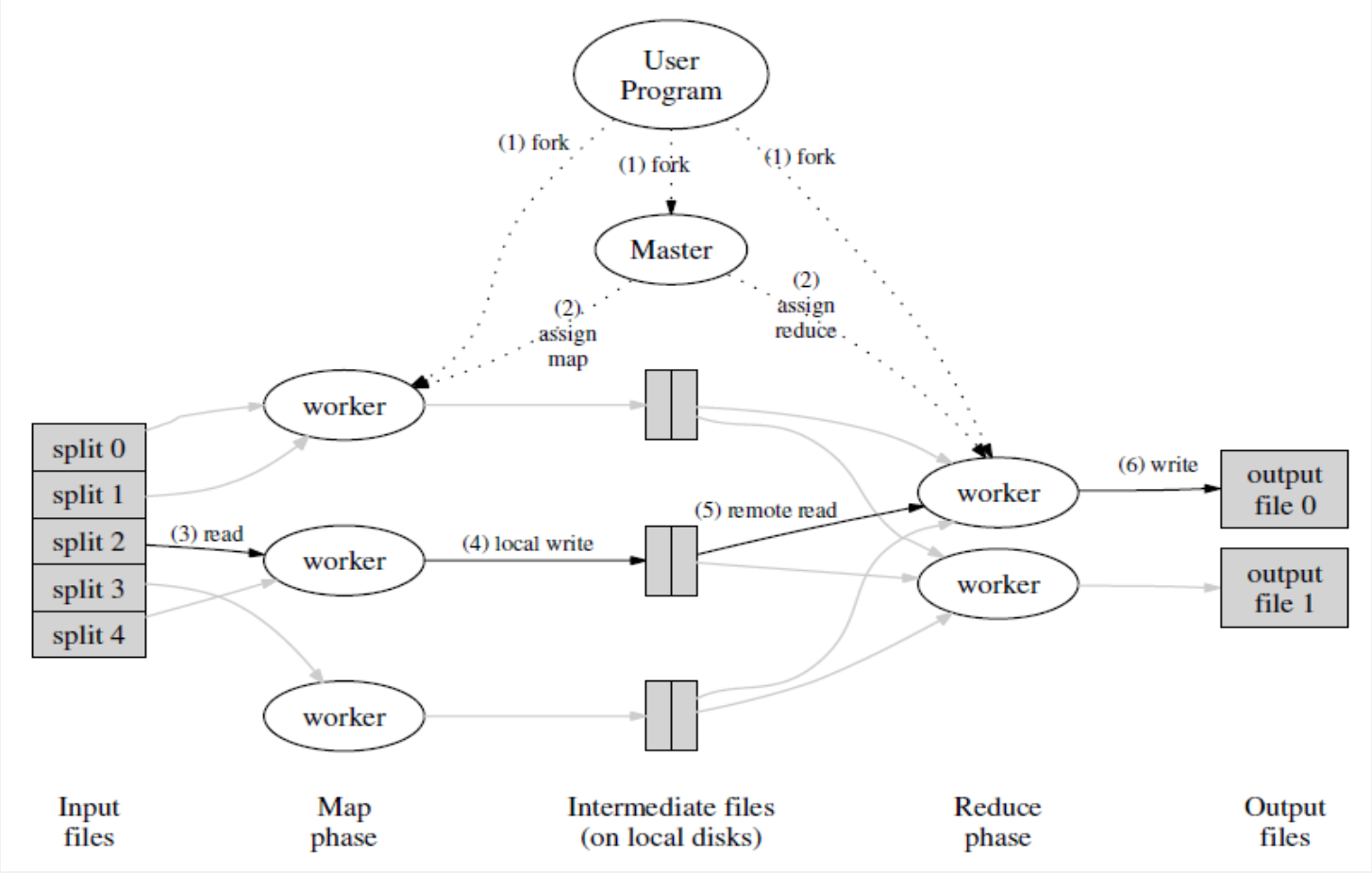
API

db.blogs.find({„author" : "joe"});
db.blogs.find(**$where**: function() { return this.author == “joe”);

- Driver für fast alle Programmiersprachen
- Query **Main** und **Embeded** Objects
- Markups ($all, $or, $exists, $size, <, >, …)
- Embeded Javascipt
  function (){return …}

6.NoSQL Praxis

Map/Reduce



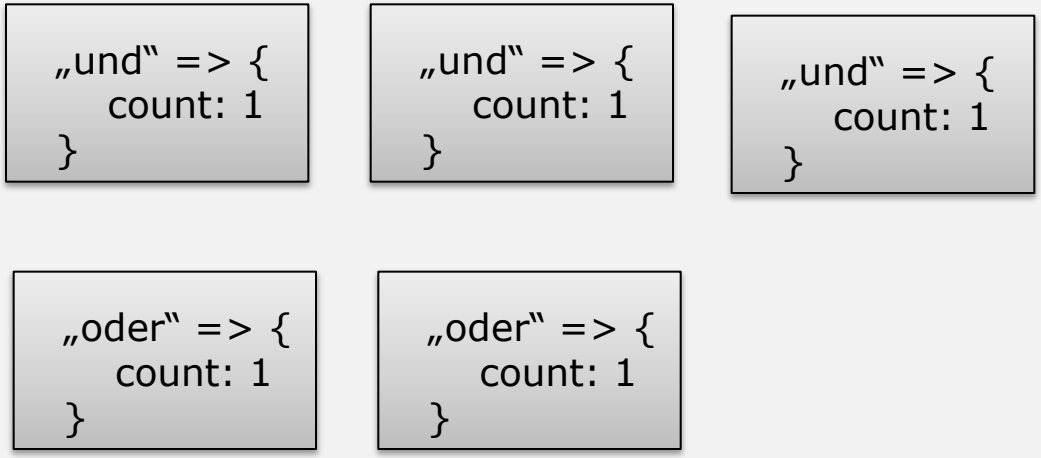
6.NoSQL Praxis

Map/Reduce

Alle Wörter in allen Dokumenten zählen
Anzahl pro Wort Ausgeben (e.g. „und“ => 1590, „oder“ => 17)

Map Function
Durchsuche alle Wörter im Dokument

```
map = function() {
... for (var word in this.text) {
...         emit( word , {count : 1});
... }};
```



6.NoSQL Praxis

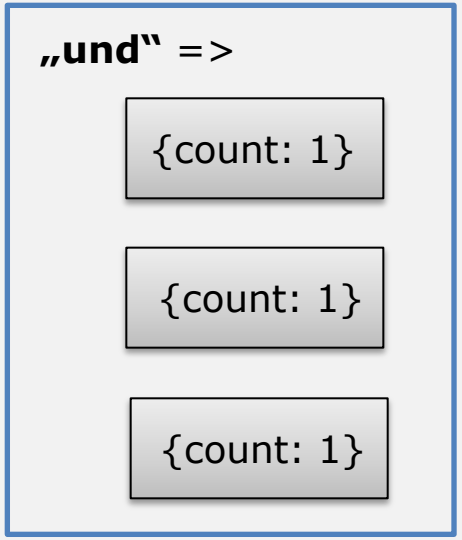
Map/Reduce

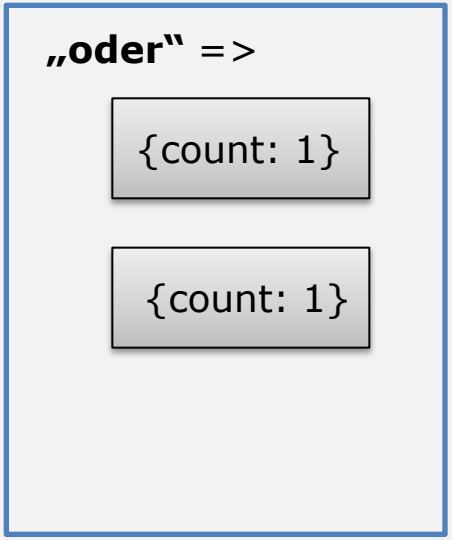
Reduce Function

Die Ergebnisse aus der Suche Bearbeiten

```
reduce = function(key, emits) {
    total = 0;
    for (var i in emits) {
            total += emits[i].count;
     }

    return {"count" : total};
}
```





Jede Node liefert ihr lokales Ergebnis

```
„und“ => {
    count: 3
}
```

```
„oder“ => {
    count: 2
}
```

6.Zusammenfassung und Ausblick

CP Class:

verteilte nicht 100% replizierte Datenbanken

- Filialnetz
- lokale Daten in der Filiale, andere Daten über Netzwerk erreichbar
- Bei Partition(Netzwerkfehler) kann auf eigene Daten konsistent zugegriffen werden

AP Class:

Web 2.0

-Kollektives Blogging, Social Networking

Sync Apps für offline Geräte (Mobiles)

- Mobiles Gerät und Desktop mit jeweils einer DB Replica

Beide:

Real Time Analytics /Datawarehousing

-Map/Reduce Power mehrer Rechner

6.Zusammenfassung und Ausblick

CP Class:

verteilte nicht 100% replizierte Datenbanken

- Filialnetz
- lokale Daten in der Filiale, andere Daten über Netzwerk erreichbar
- Bei Partition(Netzwerkfehler) kann auf eigene Daten konsistent zugegriffen werden

AP Class:

Web 2.0

-Kollektives Blogging, Social Networking

Sync Apps für offline Geräte (Mobiles)

- Mobiles Gerät und Desktop mit jeweils einer DB Replica

Beide:

Real Time Analytics /Datawarehousing

-Map/Reduce Power mehrer Rechner

6.Zusammenfassung und Ausblick

GILT GROUPE

Real Time E-Commerce Analytics
- Hoher Traffic
- Daten in werden in MongoDB gecaptured
- Analyse mit Map/Reduce

Quellen

[1]Andersen Lehnard, Slater, **CouchDB: The Definitive Guide,** O'Reilly, 2010

[2]Chodorof, Dirolf, **MongoDB: The Difinitive Guide**, O'Reilly, 2010

[3]Rick Cattell, **Horizontally Scalable Data Stores,** 2010

[4]Eric Brewer**, Cluster-Based Scalable Network Services,** 1997

[5]Gilber, Lynch, **Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services,** 2002

[6]Jeffrey Dean, Sanjay Ghemawat, **MapReduce: Simplied Data Processing on Large Clusters,** 2004

[7]Stonebarker, **SQL Databases v.NoSQL Databases, 2010**

[8]Stonebarker**, Saying Good-bye to DBMSs, Designing Effective Interfaces, 2010**

[9]Gary Anthes, **Happy Birthday RDBMS**!, http://cacm.acm.org/magazines/2010/5/87252-happy-birthday-rdbms/fulltext, 2010

Quellen

[10]MongoDB, On distributed Consistency, http://blog.mongodb.org/post/475279604/on-distributed-consistency-part-1, 2010

[11]Hendersen, **Building Scalable Websites, O'Reilly**, 2006

[12]Werner Vogels, **Eventually Consistent**, http://queue.acm.org/detail.cfm?id=1466448, 2008