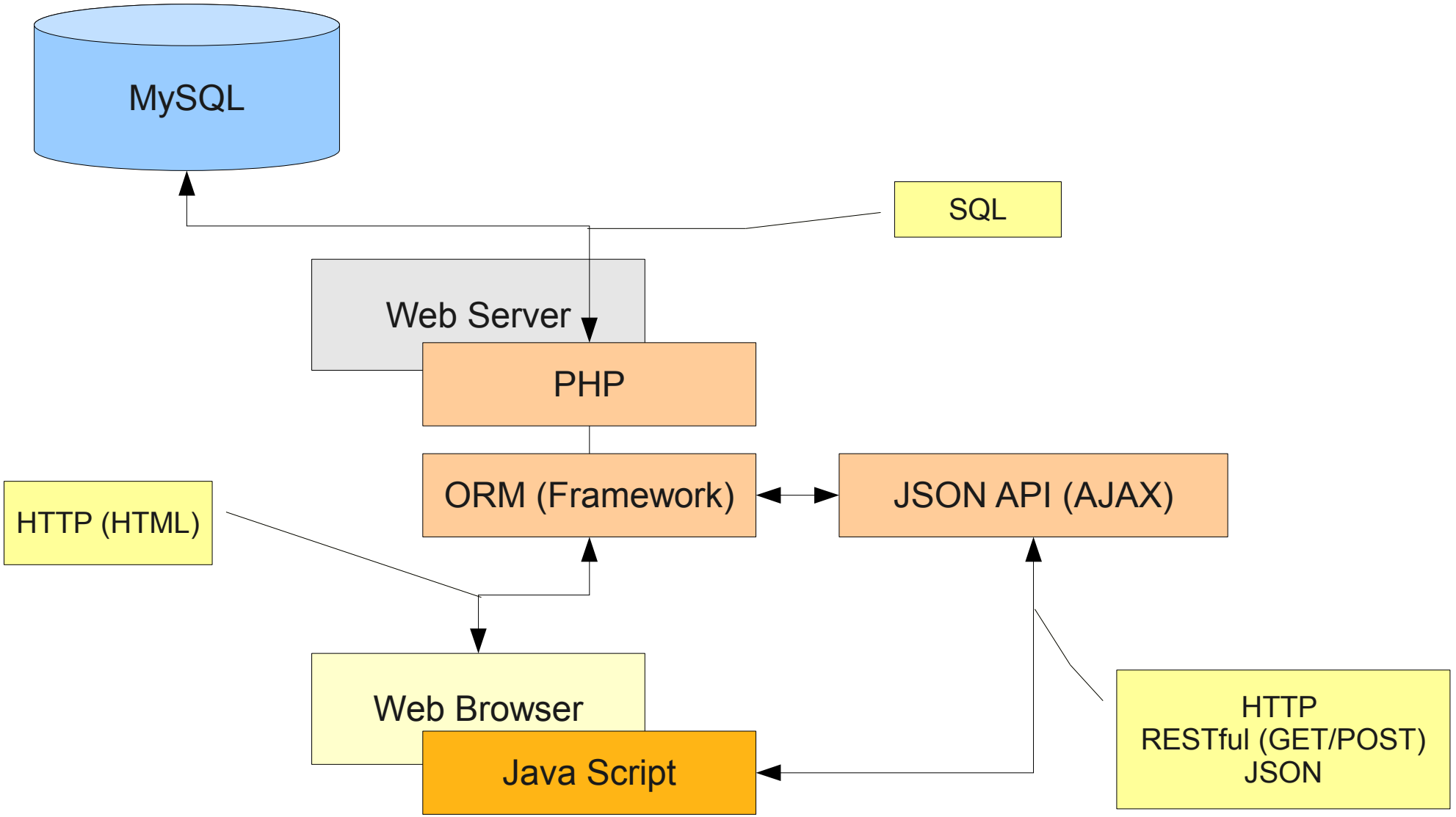
Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

Karl Glatz
Oktober 2009

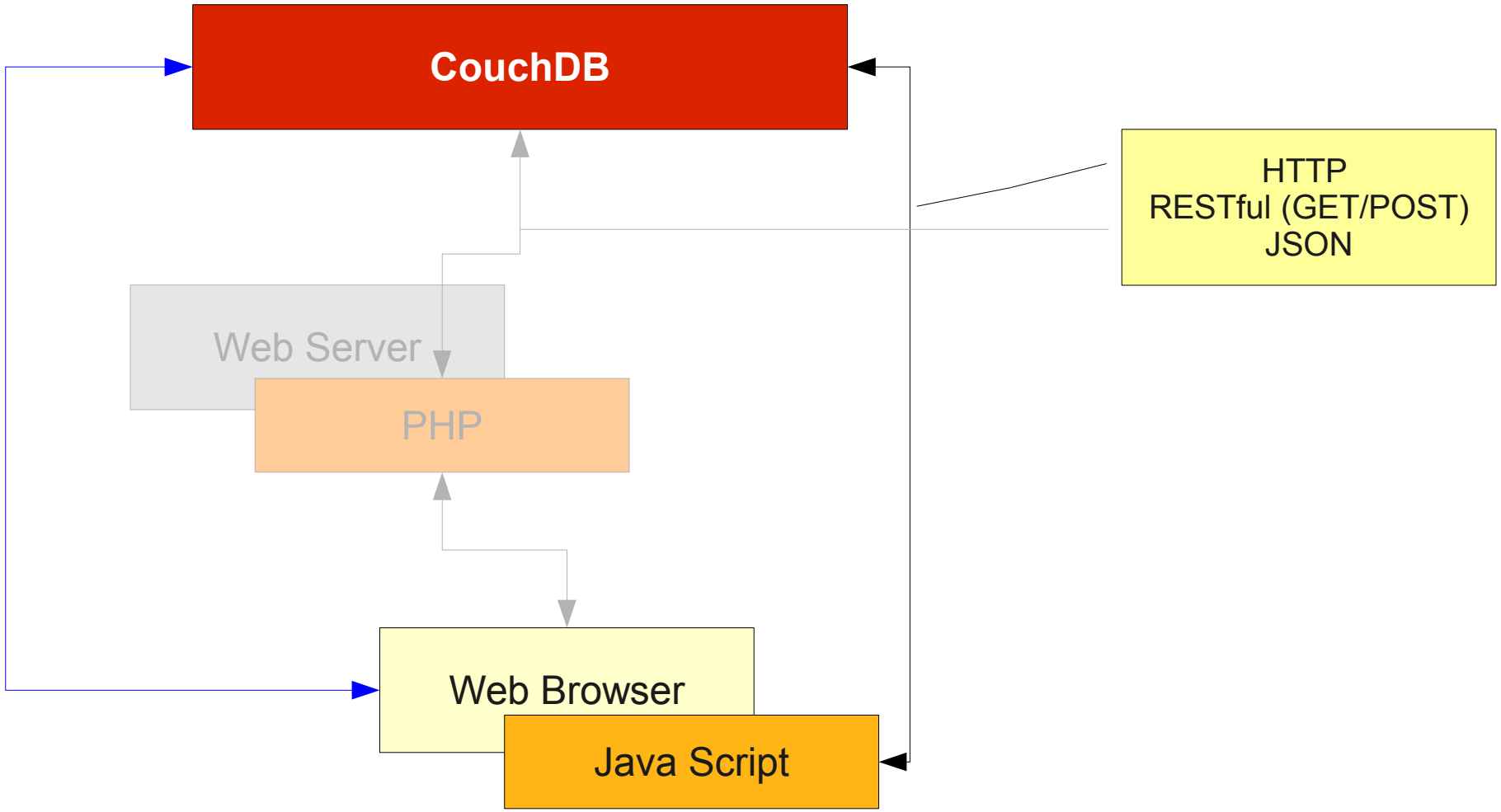
Vorstellung der verteilten NoSQL Datenbank CouchDB

apache CouchDB relax

Web Awendung (AJAX)



Web Anwendung (AJAX)



Inhalt: Einstieg

- **Einstieg in CouchDB**
  - Eigenschaften von CouchDB
  - Warum CouchDB?
  - Das CouchDB Projekt
  - Architektur von CouchDB
  - Unterschiede zu relationalen Datenbanken
  - Beispiel: Zugriffsmechanismen (Query)

Inhalt: Interna & Verteilung

- **CouchDB Interna**
  - Indizes (Views)
  - Gleichzeitigkeit
  - Update, Suche
- **Verteilte Aspekte von CouchDB**
  - Modelle
  - Bi-direktionale Replikation
  - Konfliktmanagement

Inhalt

- Geschwindigkeit
- Pro und Contra CouchDB
- Einsatzgebiete / „Real World“ Anwendungen
- P2P Web: CouchDB Apps
- Zusammenfassung

CouchDB – Was ist das?

- Backronym
  - Cluster
  - of
  - unreliable
  - commodity
  - hardware
  - Data
  - Base

- Überstezung
  - Verbung
  - aus
  - unzuverlässiger
  - Standard-
  - Hardware
  - Daten-
  - Bank

„Google Modell“

Eigenschaften von CouchDB

- Dokumentenorientierte Datenbank
  - ➔ Bekannt als: NoSQL, Key-Value, Property DBs
- Flexible und simple Objekt-Datenstrukur (JSON)
- Kein Schema → keine Einschränkungen
- Lose Verweise (Relationen) wie im Web (Links)
- Programmierbare Indizes (JavaScript)
- HTTP (RESTful) zur Kommunkikation
- Optimiert für die Verteilung auf mehrere Rechner

Warum CouchDB?

- Einfach → Relax!
- Welt besteht aus Dokumenten
  - ➔ z. B. Visitenkarten, Rechnungen etc.
- RDBMS sind komplex → hoher Wartungsaufwand
- RDBMS sind nicht „Web fähig“ (JavaScript)
- Daten und Programmierung „rücken zusammen“
- **Skalierbarkeit** (Hoch und Runter)
  - ➔ Baustein für große und kleine Systeme

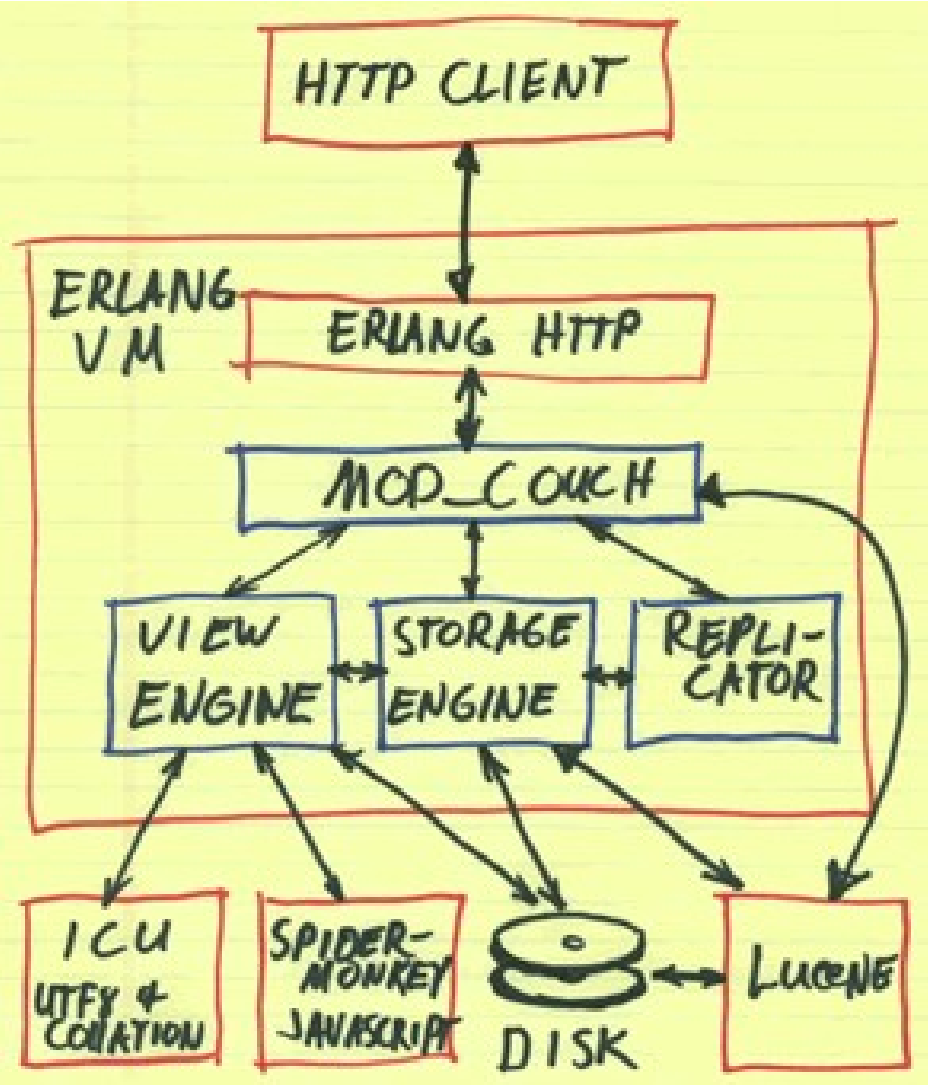
Was CochDB nicht ist!

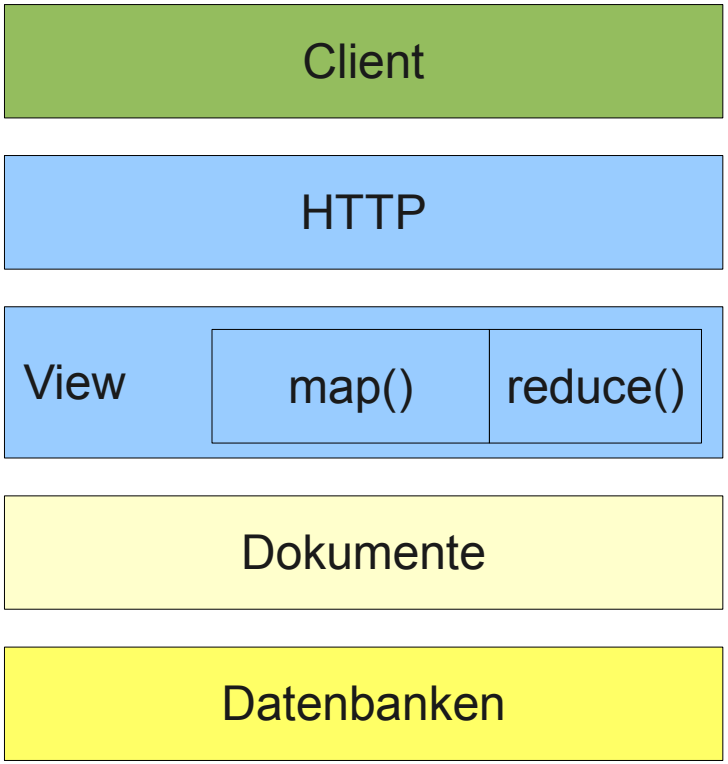
- Eine relationale Datenbank
- Ersatz für RDBMS
  - ➔ Einsatzzweck beachten
- Eine Objektorientierte Datenbank
  - ➔ bzw. Objekt-Relationaler Mapper (ORM)

Das CouchDB Projekt

- OpenSource (Apache 2.0 Lizenz)
- Projekt in Apache Software Foundation
  - ➔ Sichert Projekt: Entwicklung, Qualität, Eigentumsrechte usw.
- Ähnlichkeiten mit Lotus Notes
  - ➔ „Notes done right“
- Programmiert in Erlang
  - ➔ Erlang beherrscht Parallelität als Sprachfeature
- Ca. 13.000 Zeilen Code (MySQL: > 1 Mio)

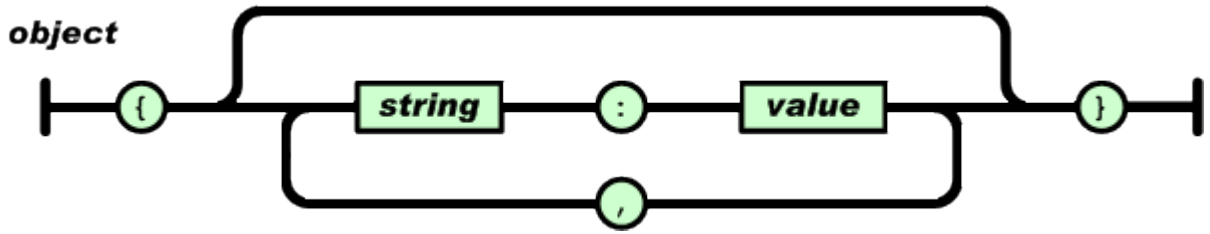
Architekturübersicht

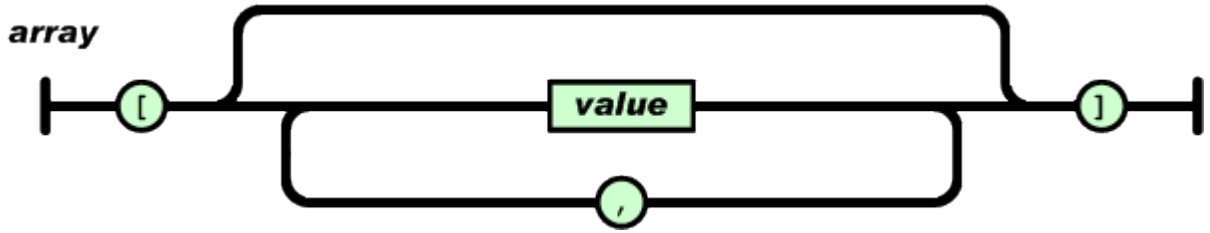




JSON: Objekt-Austausch Format

- JSON: **J**ava**S**cript **O**bject **N**otation





- Simples Format
- In 39 Sprachen verfügbar

RDBMS vs CouchDB

Tabellen vs Dokumente

Schema

blog_posts
id
title
tagline
creator_id
create_date

Daten: Zeilen

id	title	tagline	author_id	create_date
1	How to use ...	Just a How to	3	3. Sept 2009
2	Updates	What i'm doing atm.	2	8. Sept 2009
...	...	...	...	...

Dokumente in DB „blog“

```
{
  _id: "post1",
  _rev : "ASRE",
  type: "post",
  title: "How to use ...",
  tagline: "Just a How to",
  author: "karl",
  create_date: "2. Sept 2009",
  internal_notes: "good blog post"
}
```

```
{
  _id: "post2",
  _rev : "EN3D",
  type: "post",
  title: "Updates",
  tagline: "What i'm doing atm",
  author: "heinz",
  create_date: "8. Sept 2009",
}
```

...

View: Auf Daten zugreifen

- Ein View besteht aus **Map** und (optional) **Reduce** Funktionen (JavaScript)
  - ➔ Anwendung auf alle Dokumente
- View stellt Query Endpunkt dar
- Code in View Ersetzt SQL Abfrage
  - ➔ Sicherheit: Keine SQL-Injektion möglich
- Abfrage Parameter (wie WHERE ...)
  - ➔ All / Key / Range
- Oder: Direktzugriff über Dokument Id
  - ➔ http://localhost:5984/test/BA1F48C5418E4E68E5183D5BD1F06476

Query: Blog Beispiel

- Map Funktion

```
function(doc) {

  if(doc.type == "post")
  {
    emit(doc.author, doc);
  }

}
```

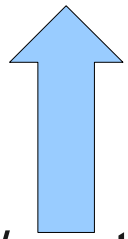
Schlüssel

Wert

{"total_rows":2,
"offset":1,
"rows":[

{"id":"post1",
"key": "kglatz",
"value": {"_id":"post1 8457","_rev":"7-863091422","title":"How to use ...","author":"kglatz","tagline":"Just a How to","create_date":"Mon Oct 05 2009 22:54:30 GMT+0200 (CEST)","type":"post"}}

]}



- Nach Schlüssel wird sortiert
- http://localhost:5984/test1/_design/blog/_view/posts?key="kglatz"

Reduce Funktion

- Reduce dient zur Aggregation von Daten
- Wird auf die Daten aus Map angewandt
- Beispiel: Fotos

```
{"name":"fish.jpg",
"user":"bob",
"type":"jpeg",
"camera":"nikon",
"info":{"width":100,
        "height":200,
        "size":12345},
"tags":["tuna","shark"]}
```

Map

```
function(doc) {
  emit("size", doc.info.size);
}
```

Reduce

```
function(keys, values, rereduce) {
  return sum(values);
}
```

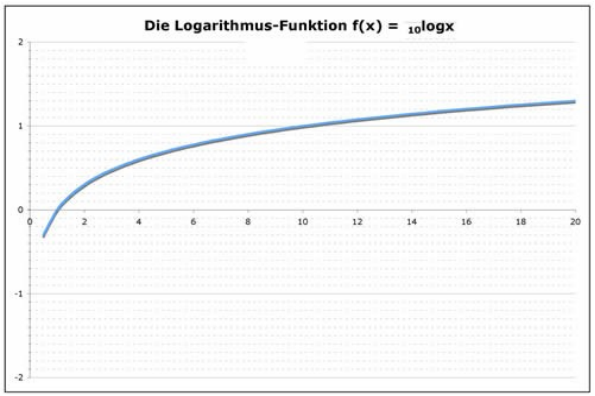
id	key	value	» reduced
1	"size"	12345	
2	"size"	32091	
3	"size"	1253	» 238208
4	"size"	92834	
5	"size"	49287	
6	"size"	50398	

Relationen Modellieren

- Keine Relation
  - Speichern im Dokument
- Relation über _id als Fremdschlüssel
- Effiziente Abfrage (Single Request)
  - Map: Key und Value können JSON Objekte (Arrays) sein → „complex keys“
  - Mit Range (von – bis) Abfragen lassen sich so Relationen nachbauen
- Anleitung: http://www.cmlenz.net/archives/2007/10/couchdb-joins

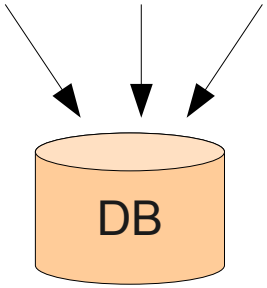
Interna: Views/Indizes

- Views: Schlüssel zu den Daten
- Effizient gespeichert als B-Baum
  - ➔ Zeit: O(log N) für Suche, Einfügen und Löschen
- Simple Zugriffsschicht
- Erzeugung zur Query-Zeit
  - ➔ Langsam bei vielen neuen Dokumenten (meist: Import) → Manuell erzeugen
- Optimierungsmöglichkeiten durch Beschränkung auf Keys



Gleichzeitigkeit

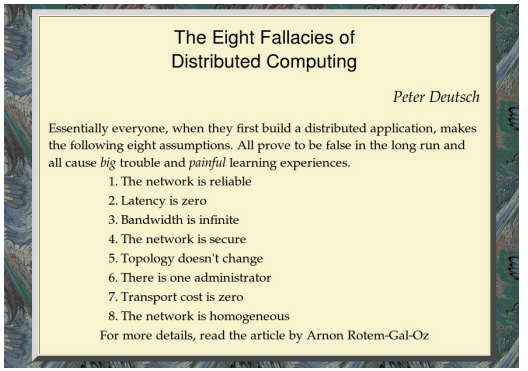
- Locking
  - Schreibzugriff: Warten bis „lock“ erhalten
  - Sperrt (Lese-)Zugriff für alle anderen
  - Anfällig unter hoher Last
- MVCC: Multi Version Concurrency Control
  - Statt überschreiben → Neues Dokument (neue Blöcke auf der Festplatte)
  - Lesezugriff trotzdem möglich (alte Version)
  - Von Last unabhängig
  - Revisionsverwaltung



Verteilte Aspekte

- Architektur ist für Replikation entworfen
  - Eindeutige IDs für Dokument (UUID mit 128 Bit)
  - Revisionsverwaltung (32Bit- Revisions Id)
- Fehlertolerant
  - Akzeptiert mehrere Realitäten → Lokale Konsistenz
- Grundsatz: „Dinge können schief gehen“
  - Fallacies of Distributed Computing
    (Irrtümer der verteilten Datenverarbeitung)

Irrtümer der verteilten Datenverarbeitung


The Eight Fallacies of
Distributed Computing

Peter Deutsch

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause *big* trouble and *painful* learning experiences.

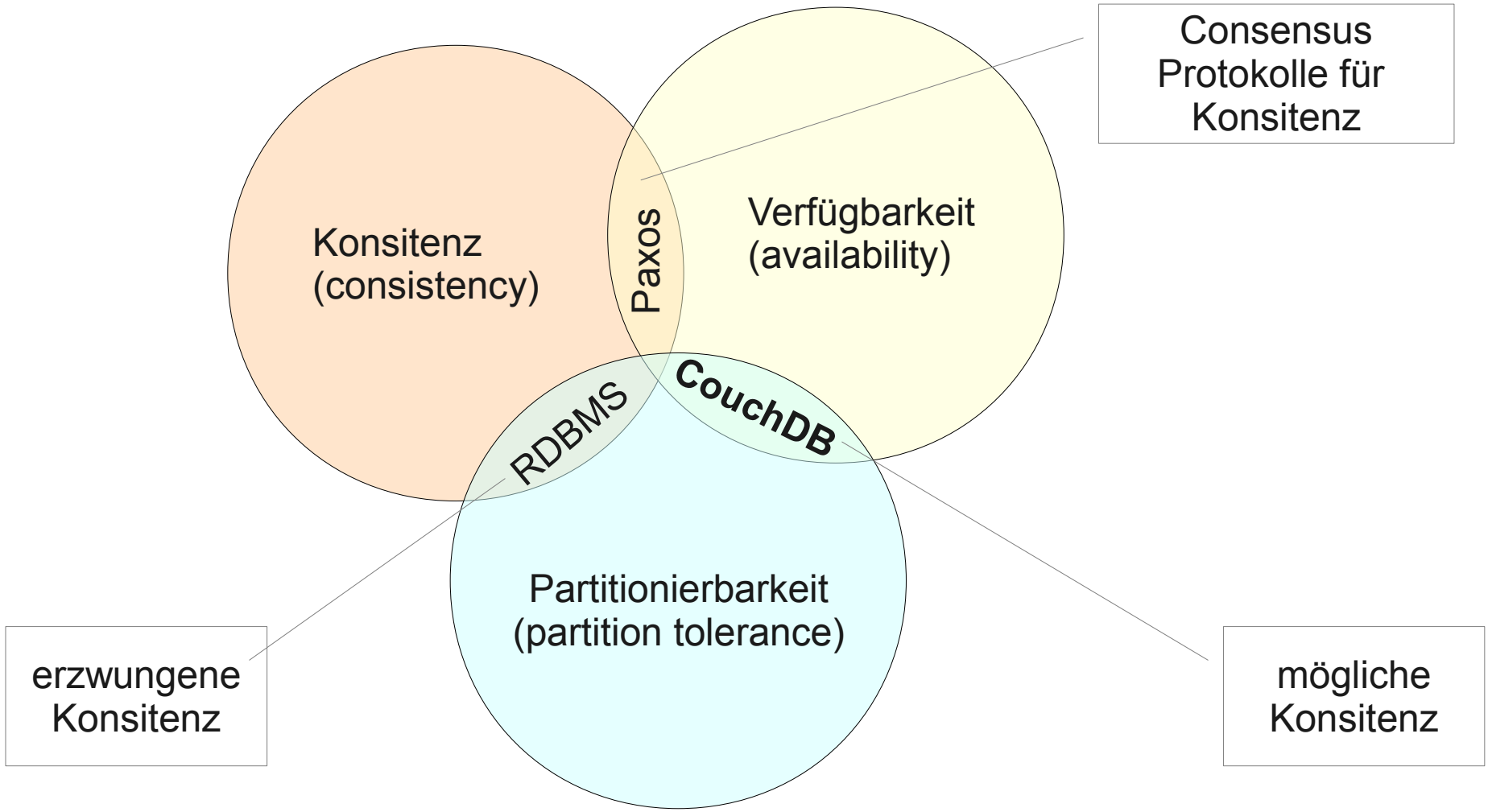
1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

For more details, read the article by Arnon Rotem-Gal-Oz

1. Das Netzwerk ist ausfallsicher
2. Die Latenzzeit ist gleich Null
3. Der Datendurchsatz ist unendlich
4. Das Netzwerk ist sicher
5. Die Netzwerktopologie wird sich nicht ändern
6. Es gibt immer nur einen Netzwerkadministrator
7. Die Kosten des Datentransports können mit Null angesetzt werden
8. Das Netzwerk ist homogen

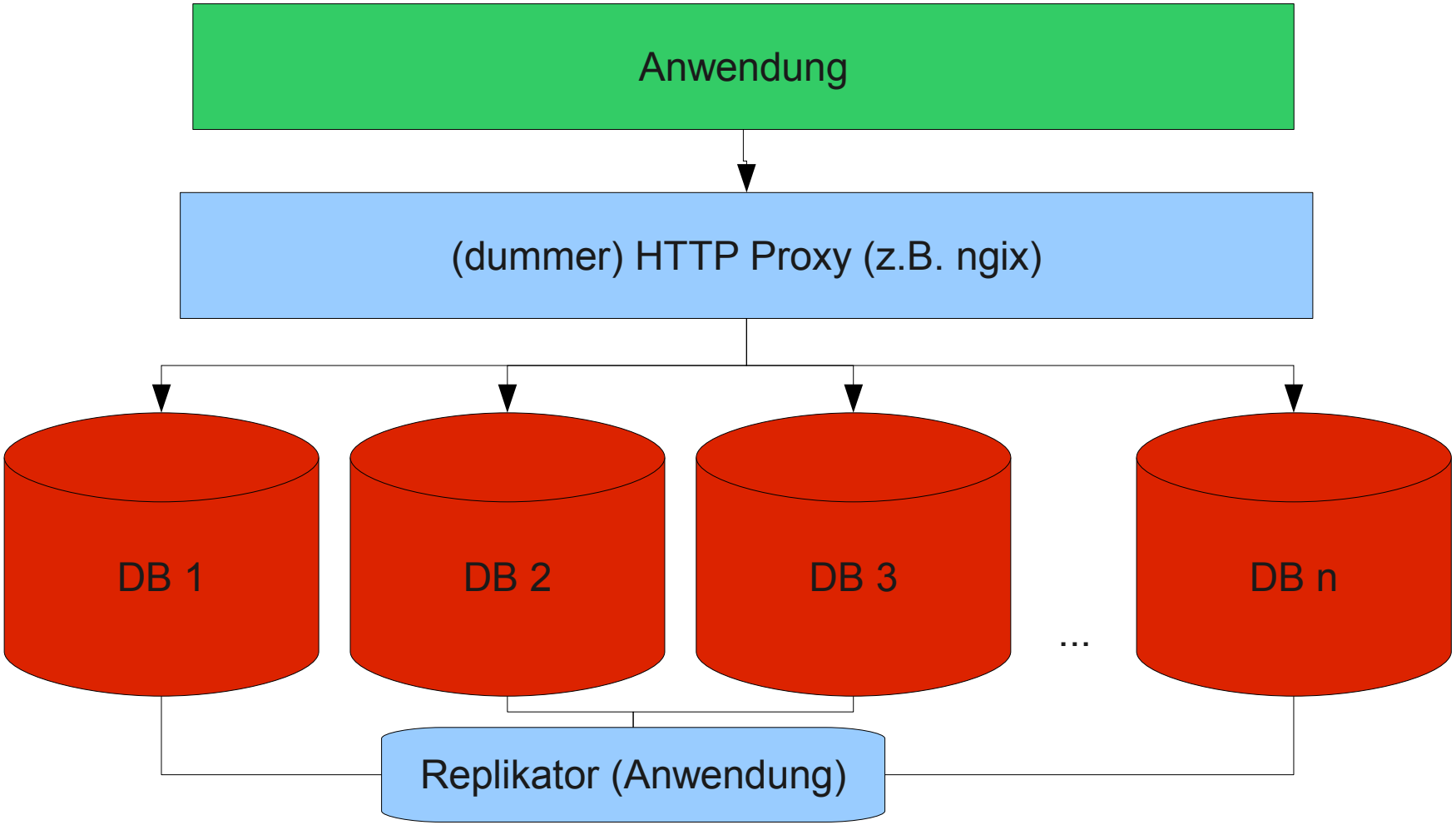
Verteilte Datenbanken

Das CAP Theorem



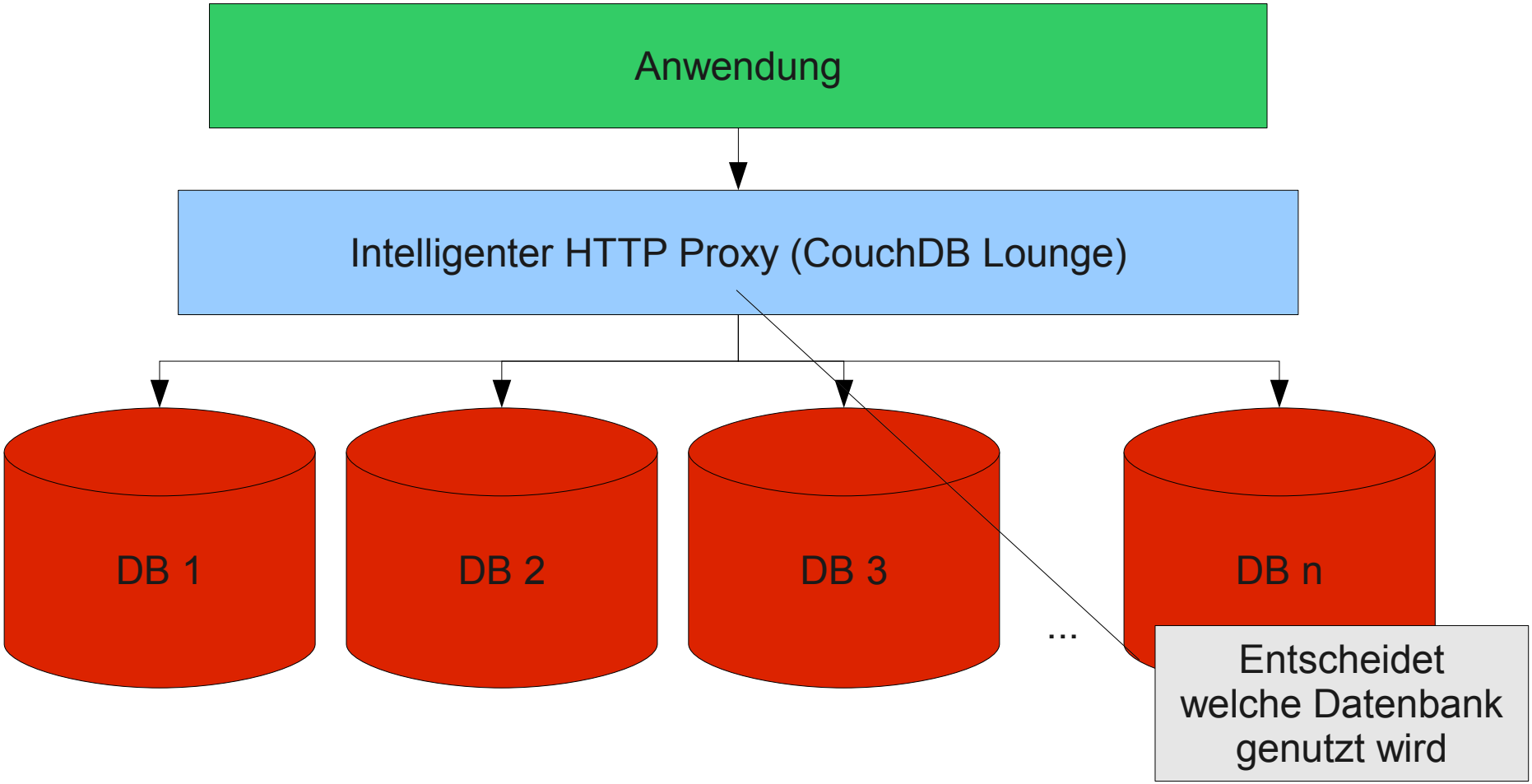
Verteilte Architekturen #1

- Möglichkeit 1: Mehre DBs eine Realität



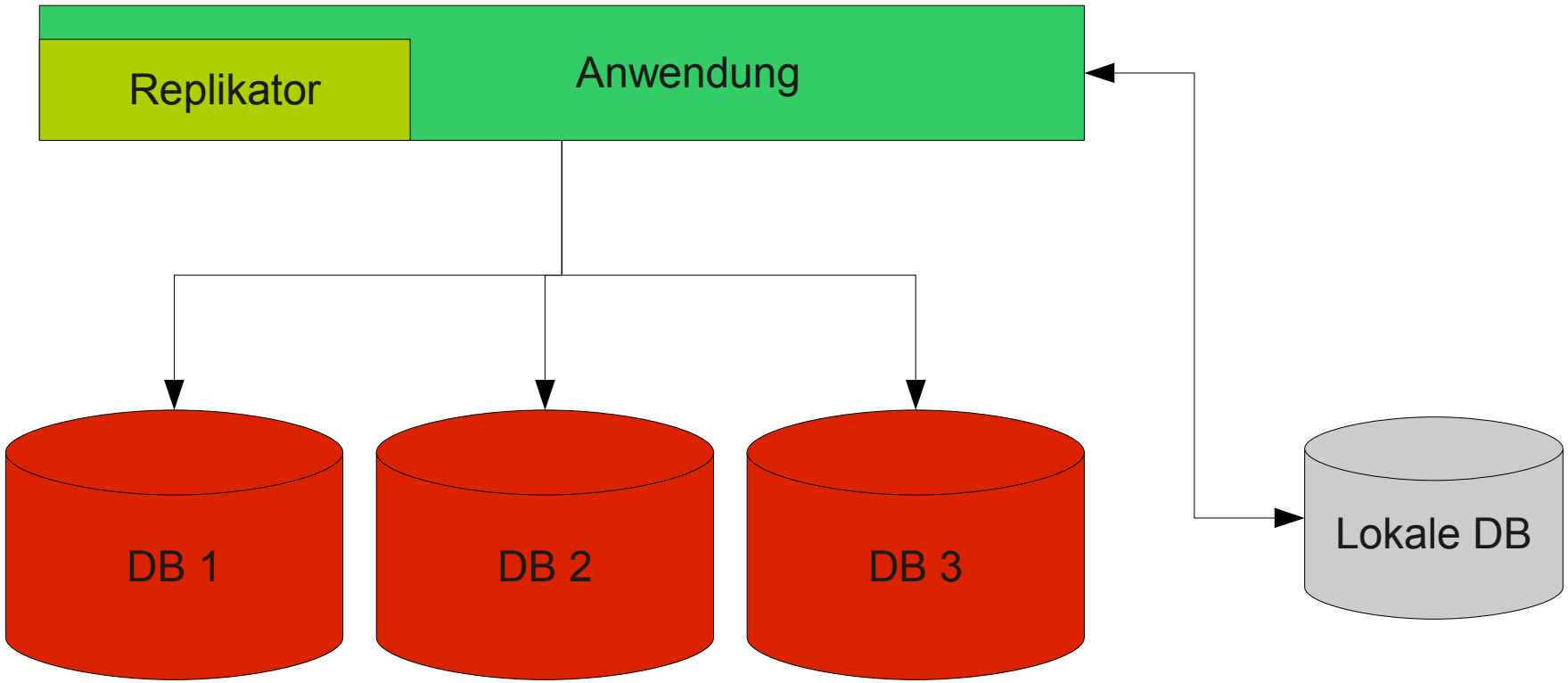
Verteilte Architekturen #2

- Möglichkeit 2: Mehre DBs, mehrere Realitäten



Verteilte Architekturen #3

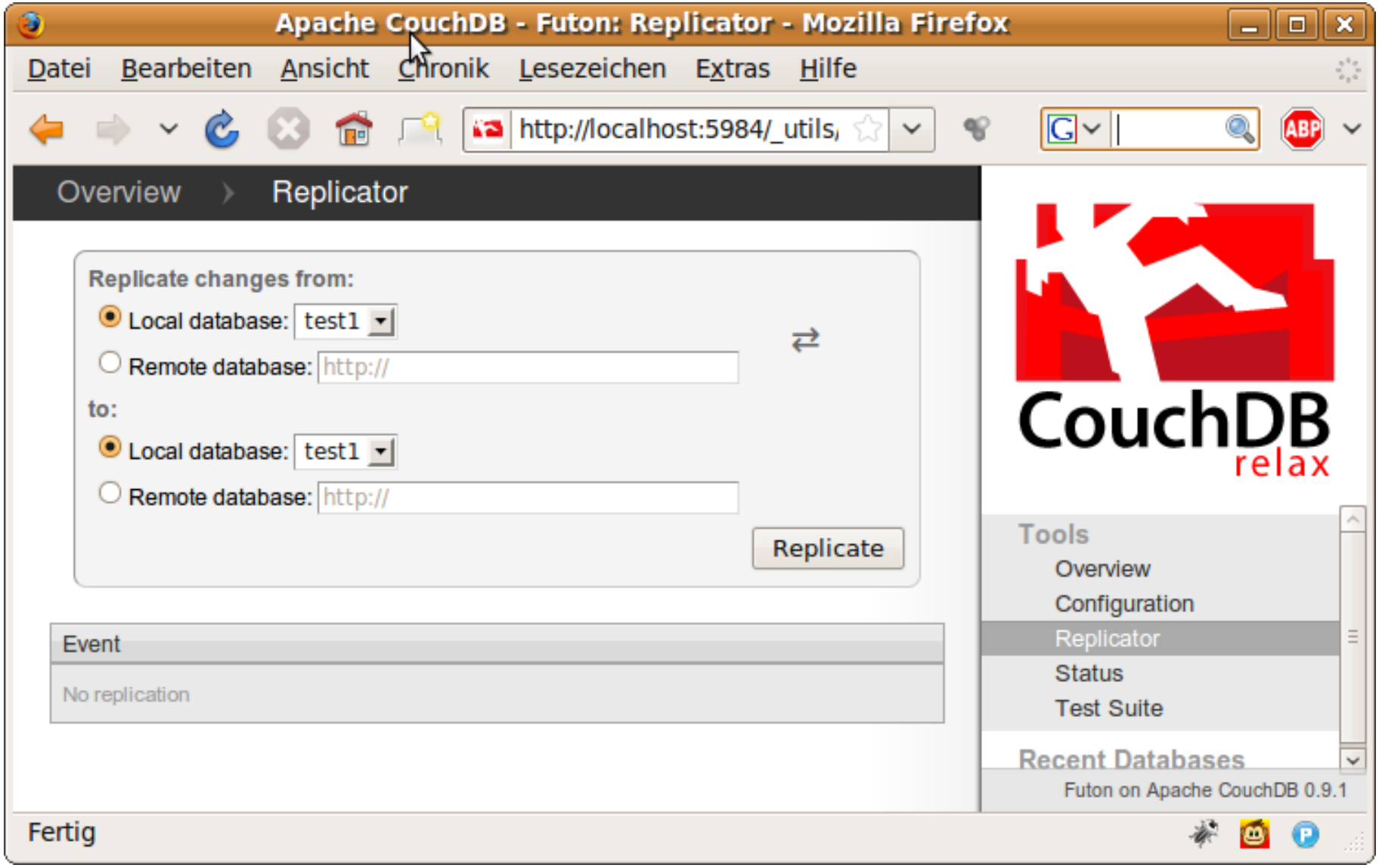
- Möglichkeit 3: Mehre DBs, mehrere Realitäten



Bi-direktionale Replikation

- Replikation muss angestoßen werden
  - ➔ Zusatz: CouchDB Lounge
- Kann unterbrochen werden
  - ➔ Fortsetzung ohne Probleme möglich
- In verteilten Systemen besteht immer lokale Konsitzenz
  - ➔ Anwendungen funktionieren; Daten können veraltet sein
- Konflikte
  - ➔ Besonderer Status (Zustand) des Dokuments

Replikation mit Futon



Geschwindigkeit

- Schnell genug!
- Viele Optimierungsmöglichkeiten noch Offen
  - Status von CouchDB: Beta
- Optimierungen durch Kompromisse
  - Speichern im RAM → weniger Zuverlässig
  - Gleichzeitigkeit: Schnell nur bei einem Request
- Ziel von CouchDB: „Es richtig machen“

Pro und Contra Key-Value DB

Contra

- Spontane Auswertungen von Daten erschwert
- Spontanes ändern von mehreren Dokumenten erschwert (UPDATE …)
- Abbildung von Relationen nicht direkt möglich

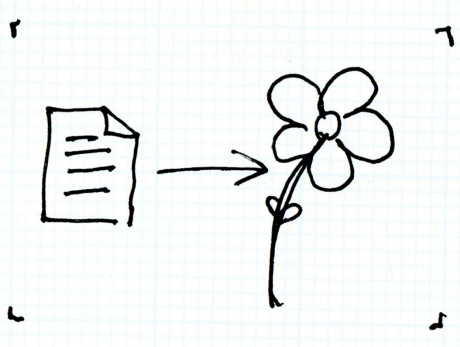
Pro

- Mehr Flexibilität
- Skaliert gut
- Replikation / Verteilung „frei Haus“
- Freiheit für Entwickler
- Dokumentenmodell: Anwendungsdomäne von WebApps
- Hohe Verfügbarkeiten realisierbar

Einsatzgebiete / Anwendungen

- Desktop: Integration in Ubuntu (9.10)
  - Austausch über LAN, „Ubuntu One“ (Profildaten etc.)
- Web Anwendungen
  - Sofa (Blog): http://jchrisa.net/
  - Swinger (Präsentationen)
- Projekte / Produkte
  - BBC: Web Plattform
  - Meebo.com: Kommerzielles Umfrage System

CouchDB Apps: Das P2P Web



- Konzept ähnlich wie IBM Domino (Applikationsserver)
- Anwendungen
  - laufen im Web-Browser (pur JavaScript)
  - in CouchDB als Dokumente gespeichert (Attachments)
  - Daten **und Anwendung** kann repliziert werden (p2p)
  - Offline wie Online („disconnected“)

Weitere „coole“ Features

- Transformationsfunktionen
  - Darstellung als: HTML, RSS, XML etc.
- Validierungsfuktionen
  - Frei programmierbare Validierung von Dokumenten
- Externe Indizes
  - z. B. Lucene – Volltextsuche
- Notification
  - Funktion wird bei Ereignis ausgeführt
- Caching mit HTTP E-Tags

Zusammenfassung

- CouchDB ist ein flexibler Datenspeicher
- Nutzt moderne und offene (Web-)Standards
- Programmierer erhält mehr Freiheit
- Schwächen bei spontanen Auswertungen („ad hoc reporting“)
  - ➔ Weniger geeignet für Finanzanwendungen, Statistik (viel Auswertungen)
- Gut Geeignet für (soziale) Web-Anwendungen
  - ➔ Viele Nutzer, hohe Verfügbarkeit → Cluster HW

CouchDB ist **nicht eine konkrete Lösung** für ein spezielles Problem von verteilten Datenbanken.

→ Sondern: Eine **Zusammenfassung von generischen Mechanismen** zur Erfüllung von spezifischen **Anforderungen** an eine verteilte Datenbank (Anwendung).

Vielen Dank für Ihre Aufmerksamkeit.

Quellen

- Sketch: http://couchdb.apache.org/img/sketch.png
- Oxygen Ions (modifiziert): http://www.oxygen-icons.org/
- CouchDBBuch:http://books.couchdb.org

Konkurrenz: Weitere NoSQL DBs

- Project Voldemort
  - Automatische Replikation / Partitionierung der Daten; Java
- Cassandra
  - Von Facebook, ähnlich wie Googles BigTable
  - Gute Parallelität („write never fails“)
- Solr
  - Auf Volltextsuche optimiert (Lucene; Java)
- MongoDB
  - Auf geschwindigkeit optimiert (C)