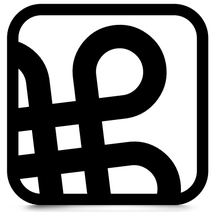


Macoun


Appelgriebsch

Mobiler Einsatz von CouchDB

Andreas Gerlach

Ablauf

- Einleitung: die 3 W's (warum, wieso, weshalb)
- CouchDB 101: ein kurzer Abriss der NoSQL Datenbank
- Beispiel-Szenario
  - HowTo: Synchronisation für native iOS Anwendungen
  - HowTo: Synchronisation für HTML5 Anwendungen

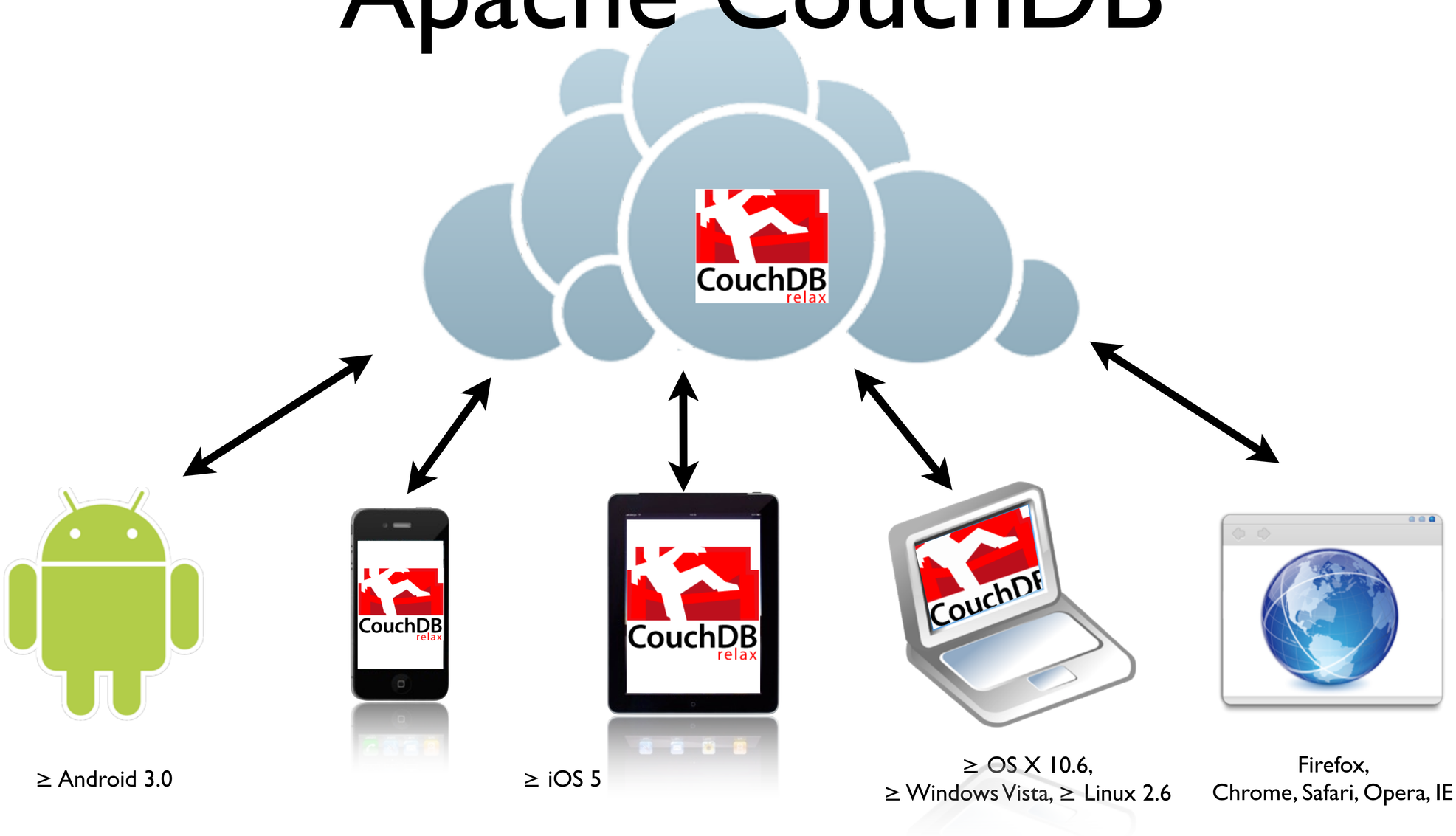
Einleitung

1x1 der Synchronisation

- Daten liegen lokal auf verschiedenen Geräten
- Geräte sind u.U. offline
- Anwendungen müssen offline-fähig sein
- geräte-übergreifender Abgleich der Daten durch zentrale Instanz

Apple's iCloud Service

Apache CouchDB



≥ Android 3.0

≥ iOS 5

≥ OS X 10.6,
≥ Windows Vista, ≥ Linux 2.6

Firefox,
Chrome, Safari, Opera, IE

CouchDB 101

Entwicklungsgeschichte

Entwicklung an CouchDB startet
Damien Katz – Kernentwickler an IBM Lotus Notes – startet das Projekt CouchDB.

CouchDB wird Apache Projekt
CouchDB wird Anfang 2008 als Apache Incubator Projekt aufgenommen.

erstes stabiles Release 1.0
Mitte 2010 erscheint die stabile Version 1.0 der NoSQL Datenbank

Couchbase Inc. gegründet
Damien Katz gründet Ende 2011 die Firma Couchbase Inc. deren Couchbase Server die Vorzüge von CouchDB und Membase vereint.

Cloudant's BigCouch -> Apache CouchDB
Anpassungen von Cloudant für die horizontale Skalierung werden Mitte 2013 in Apache CouchDB überführt.

Eigenschaften

- dokumenten-orientierte NoSQL Datenbank (JSON Format)
- Map-Reduce Abfragen fungieren als Datenbank Views
- Datenspeicherung nutzt Multi-Version Concurrency Control
- Geo-Spatial Unterstützung für die Analyse geographischer Daten
- Ansteuerung über REST API
- Futon Web Anwendung

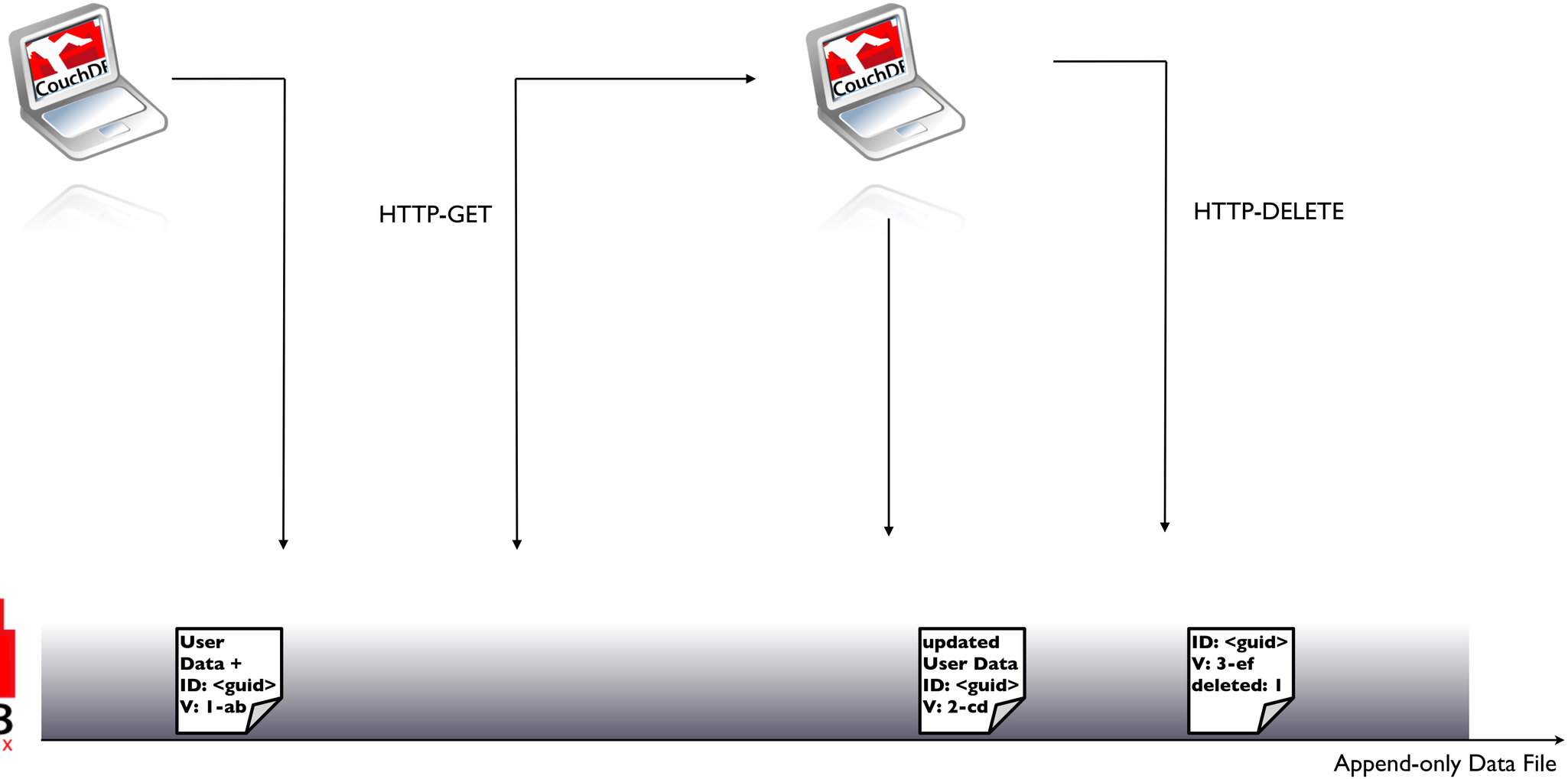
Alleinstellungsmerkmale

- Full-Text Search Anbindung an Apache Lucene / ElasticSearch
- changes-Feed als Protokoll von Datenbank-Änderungen
- Multi-Master Merge-Replikation
- Unterstützung von Anhängen (Attachments) zu Dokumenten
- Couch Apps: self-contained Web Anwendungen in der Datenbank

Schwerpunkte des Vortrages

- Datenspeicherung mittels Multi-Version Concurrency Control
- changes-Feed als Protokoll von Datenbank-Änderungen
- Multi-Master Merge-Replikation

Multi Version Concurrency Control



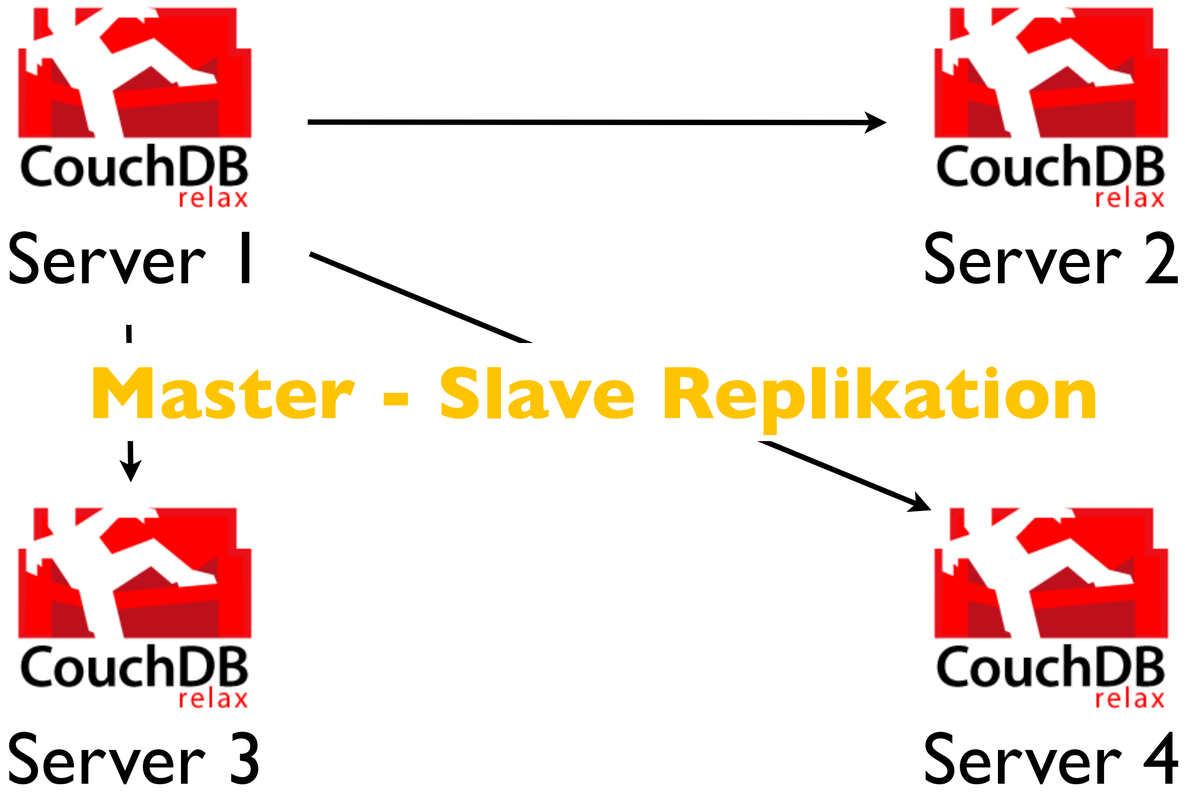
Multi Version Concurrency Control

- jedes Dokument enthält zusätzlich zu den Nutzdaten folgende Meta-Informationen:
  - _id: eine eindeutige ID; entweder benutzerspezifisch oder UUID
  - _rev: eine fortlaufende, eindeutige Versionsnummer + Hashcode des JSON Inhalts
- Dokumente in der Datenbank werden nicht geändert; neue Versionen werden fortlaufend an die Datei angehängt

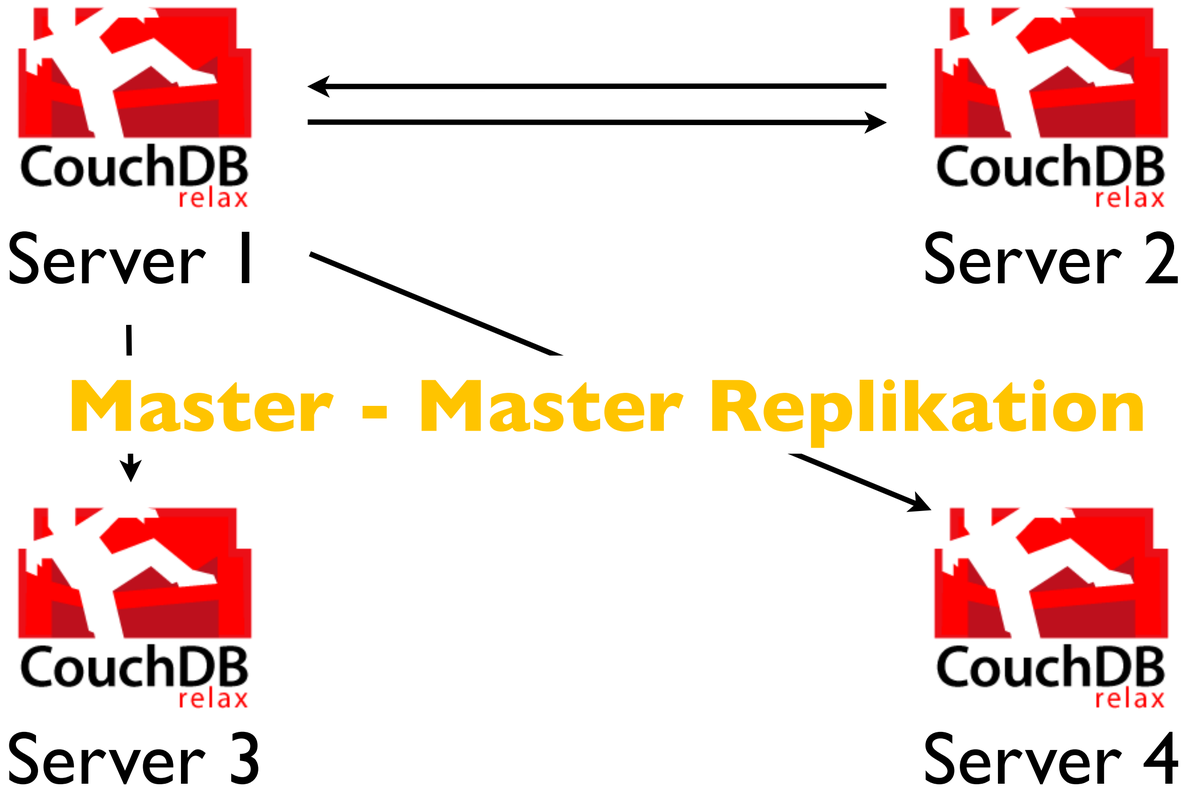
changes Feed

- separater API Aufruf
- basiert auf der append-only Eigenschaft des Datenablagesystems
- unterstützt verschiedene Modi: polling, long polling, continous
- kann mittels Filter-Funktionen kontextspezifisch angepasst werden (bspw. Query-Parameter, aktuell angemeldeter Nutzer, ...)

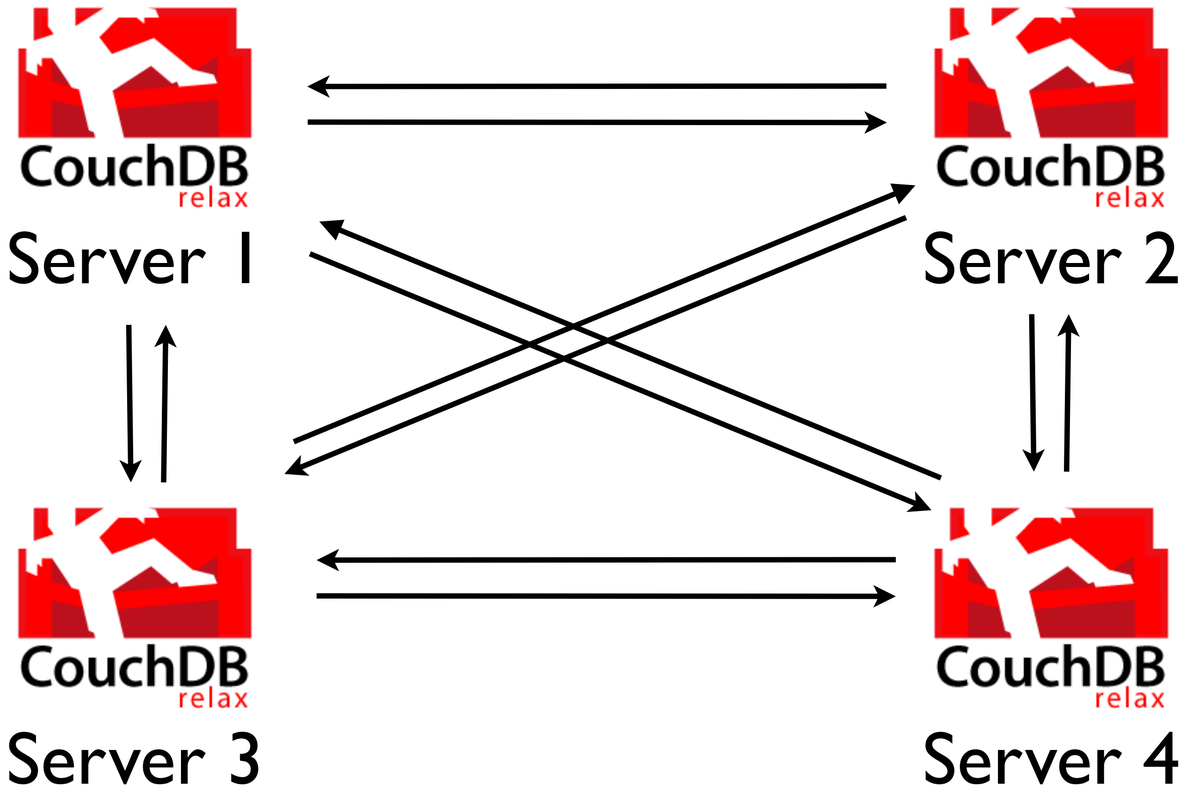
Multi-Master Merge-Replication



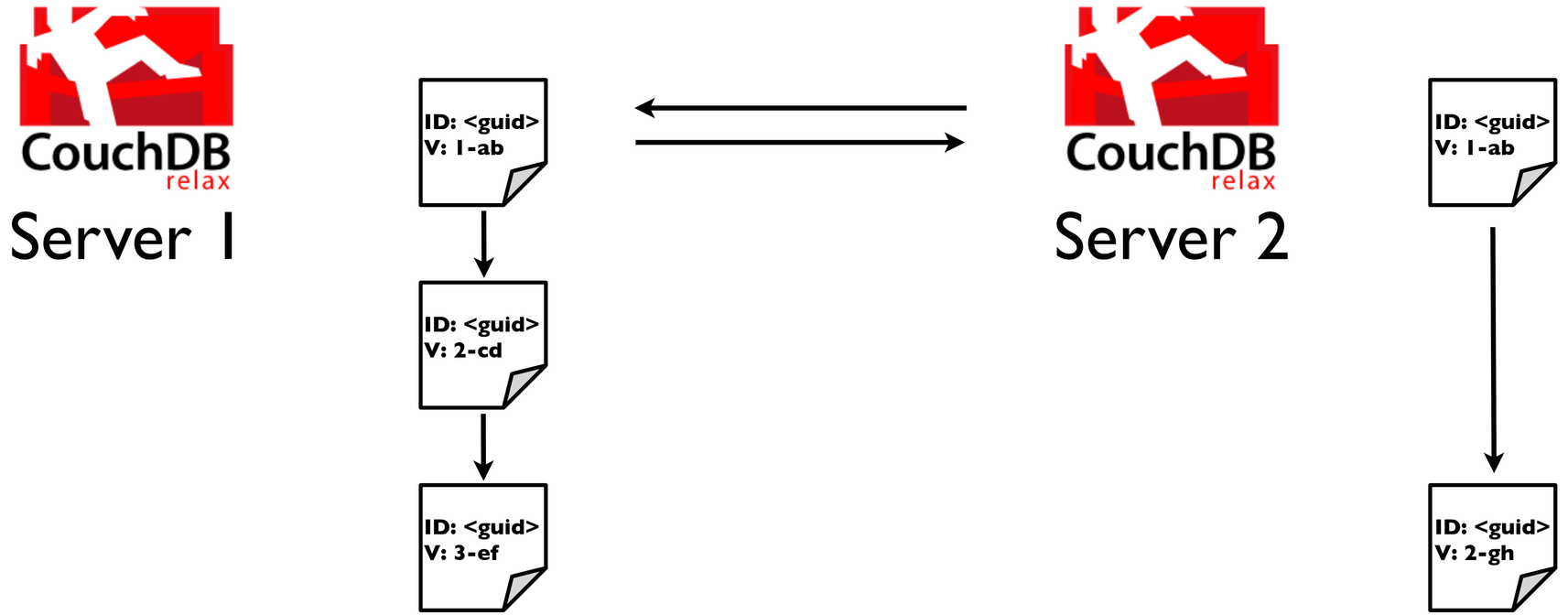
Multi-Master Merge-Replication



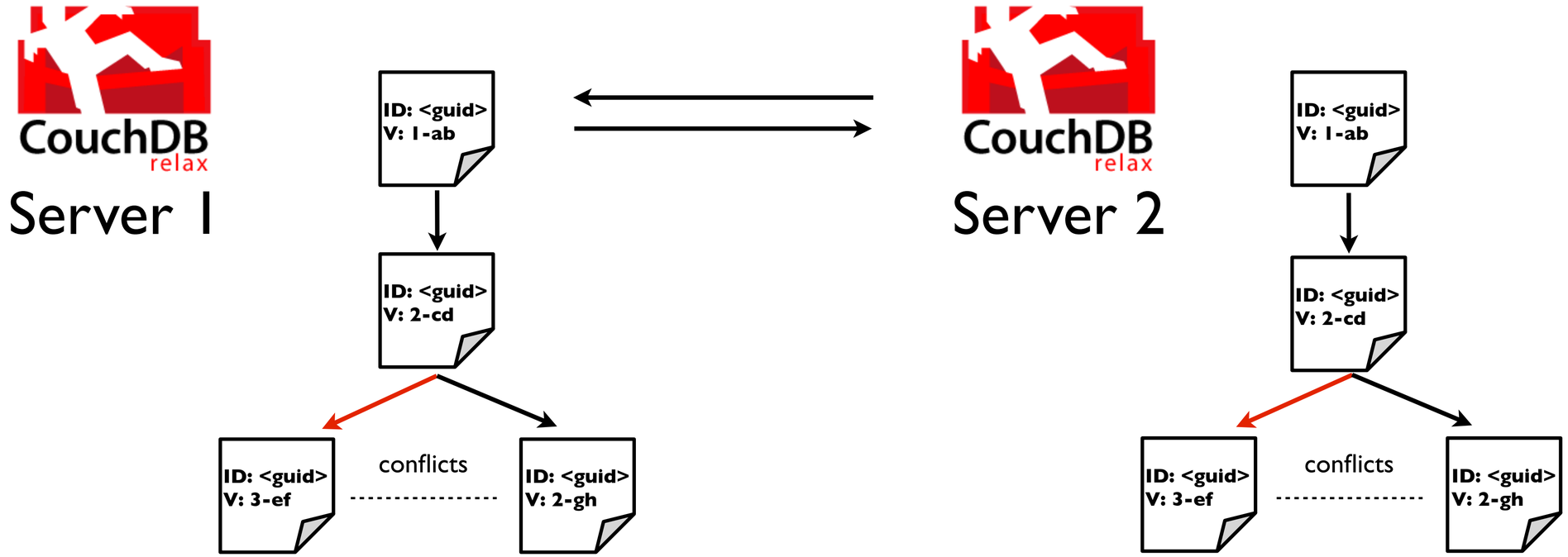
Multi-Master Merge-Replication



Multi-Master Merge-Replication

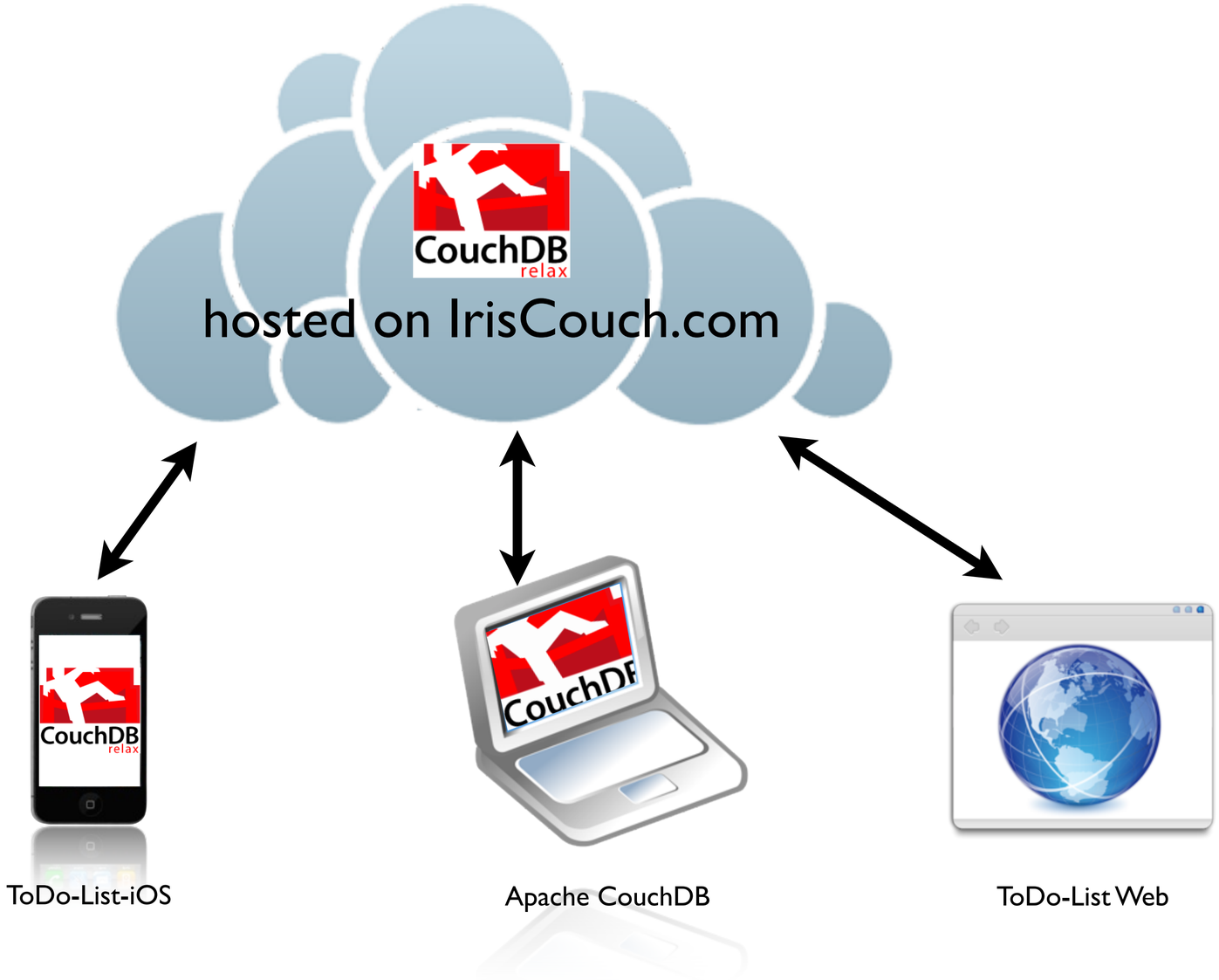


Multi-Master Merge-Replication



Demo

Beispiel-Szenario


CouchDB
relax
hosted on IrisCouch.com
CouchDB
relax
ToDo-List-iOS
Apache CouchDB
ToDo-List Web

CouchBase Lite für iOS

CouchDB API

Objective-C

SQLite

Erste Schritte

- lade Couchbase-Lite für iOS von der Couchbase Webseite
- referenziere CouchbaseLite.framework in Xcode

```
#import <CouchbaseLite/CouchbaseLite.h>
```

- erstelle / benutze eine neue / existierende lokale Datenbank

```
@property(nonatomic, strong)CBLDatabase *database;
...
self.database = [[CBLManager sharedInstance] createDatabaseNamed:
@"todo-items" error: &error];
```

Erste Schritte

- Erstellen einer lokalen Datenbank-View

```
[[self.database viewNamed: @"byDate"] setMapBlock: MAPBLOCK({
    ...
  }) reduceBlock: REDUCEBLOCK({...}) version: @"1.0"];
```

- wobei hierbei das Map-Reduce in Objective-C wie folgt definiert ist:

```
#define MAPBLOCK(BLOCK) ^(NSDictionary* doc, void (^emit)(id key,
id value)){BLOCK}

#define REDUCEBLOCK(BLOCK) ^id(NSArray* keys, NSArray* values, BOOL
rereduce){BLOCK}
```

Erste Schritte

- Abfrage an die Datenbank definieren

```
CBLLiveQuery* query = [[[self.database viewNamed:@"byDate"] query]
asLiveQuery];
```

- Binden der Ergebnisse der Abfrage an einen UITableView

```
@interface MyTblViewController:UIViewController<CBLUITableDelegate>
@property(nonatomic, strong) IBOutlet CBLUITableSource* dataSource;
...
self.dataSource.query = query;
...
- (UITableViewCell *)couchTableSource:(CBLUITableSource *)source
                    cellForRowAtIndexPath:(NSIndexPath *)indexPath;
```

Erste Schritte

- Erstellen eines Dokuments

```
CBLDocument* newDocument = [self.database untitledDocument];
NSMutableDictionary* newProps = [NSMutableDictionary
dictionaryWithDictionary:
          @{@"title": self.titleField.text,
            @"type": @"todo",
            @"finished": @"false", ...}];
```

- Speichern des Dokuments

```
[newDocument putProperties:newProps error:&error];
```

Erste Schritte

- Konfigurieren & Starten der Replikation

```
@property(nonatomic, strong)CBLReplication* pullReplication;
@property(nonatomic, strong)CBLReplication* pushReplication;
...
self.pushReplication = [self.database pushToURL:serverUrl];
self.pullReplication = [self.database pullFromURL:serverUrl];

self.pullReplication.filter = @"app/byUser";
self.pushReplication.continuous =
      self.pullReplication.continuous = YES;

[self.pushReplication start];
[self.pullReplication start];
```

Erste Schritte

- Empfangen von Status-Updates der Replikation

```
NSNotificationCenter* notific = [NSNotificationCenter
defaultCenter];
[notific addObserver: self
            selector:@selector(replicationProgress:)
                name: kCBLReplicationChangeNotification
              object: self.pullReplication];
```

- Interface für Status Updates

```
@property (nonatomic, readonly) CBLReplicationMode mode;
@property (nonatomic, readonly) unsigned completed;
@property (nonatomic, readonly) unsigned total;
@property (nonatomic, readonly, retain) NSError* error;
```

PouchDB JavaScript

CouchDB API

JavaScript

IndexedDB

Erste Schritte

- lade die JavaScript Bibliothek von der PouchDB Webseite
- referenziere die Bibliothek in der eigenen Web Seite

```
<script src="pouchdb-nightly.min.js"></script>
```

- erstelle / benutze eine neue / existierende lokale Datenbank

```
var db = new PouchDB('todo-items');
```

Erste Schritte

- Erstellen eines Dokuments

```
var todo = {
    title: text,
    type: type,
    finished: false
  };
```

- Speichern des Dokumentes

```
db.put(todo, function callback(err, result) {
   ...
});
```

Erste Schritte

- Konfigurieren & Starten der Replikation

```
var remoteCouch = 'http://user:pass@mname.iriscouch.com/todo-
items';

// replication options incl. callback handler
var opts = {continuous: true, filter: ‘app/byUser’,
            complete: onComplete,
            onChange: onChange};

db.replicate.to(remoteCouch, opts);
db.replicate.from(remoteCouch, opts);
```

Tipps & Tricks

- für komplexe Web Anwendungen nutze ein JavaScript MVC Framework (bspw. Backbone.js mit Backbone.PouchDB Plugin)

```
// set global Backbone sync properties
Backbone.sync = BackbonePouch.sync({
    db: Pouch(dbname),
 fetch: 'query'
});

// Adjust id attribute to the one PouchDB uses
Backbone.Model.prototype.idAttribute = '_id';
```

Tipps & Tricks

- ... definiere ein Backbone Model...

```
var Todo = Backbone.Model.extend({
        defaults: function () {
            return {
                type: "todo",
                title: "empty todo...",
                finished: "false"
            };
        },
        ...
```

Tipps & Tricks

- ... eine Collection...

```
var TodoList = Backbone.Collection.extend({
        model: Todo,
        ...
```

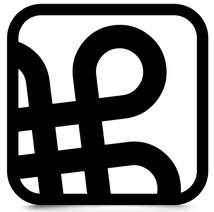
- ... und eine View...

```
var TodoView = Backbone.View.extend({
         tagName: "li",
         template: _.template($('#item-template').html()),
         ...
```

Demo

Fragen?

Vielen Dank



Macoun