Eine Einführung in Apache CouchDB

Java-Forum Stuttgart 2011

Johannes Schneider, cedarsoft GmbH
js@cedarsoft.com
http://blog.cedarsoft.com
http://cedarsoft.com

Vielen
Dank

CouchDB – The VERY Basics

CouchDB – The VERY Basics

Inhalt

NoSQL – (ein paar) theoretische Grundlagen
CouchDB praktisch

Was ist CouchDB?

Apache Top-Level-Projekt
→ Apache 2.0 Lizenz

Implementiert in Erlang (uuuh...)

Dokumentenorientiert (JSON)
Queries per Map/Reduce
Inkrementelle Replikation (Master to Master)
Bi-Direktionales Konfliktmanagement
REST-API

Leichtgewichtig
Kein Treiber
„Internet-fähig“

E6

„RESTful APIs - warum REST mehr als http und XML ist“

Dr. Stefan Schlott

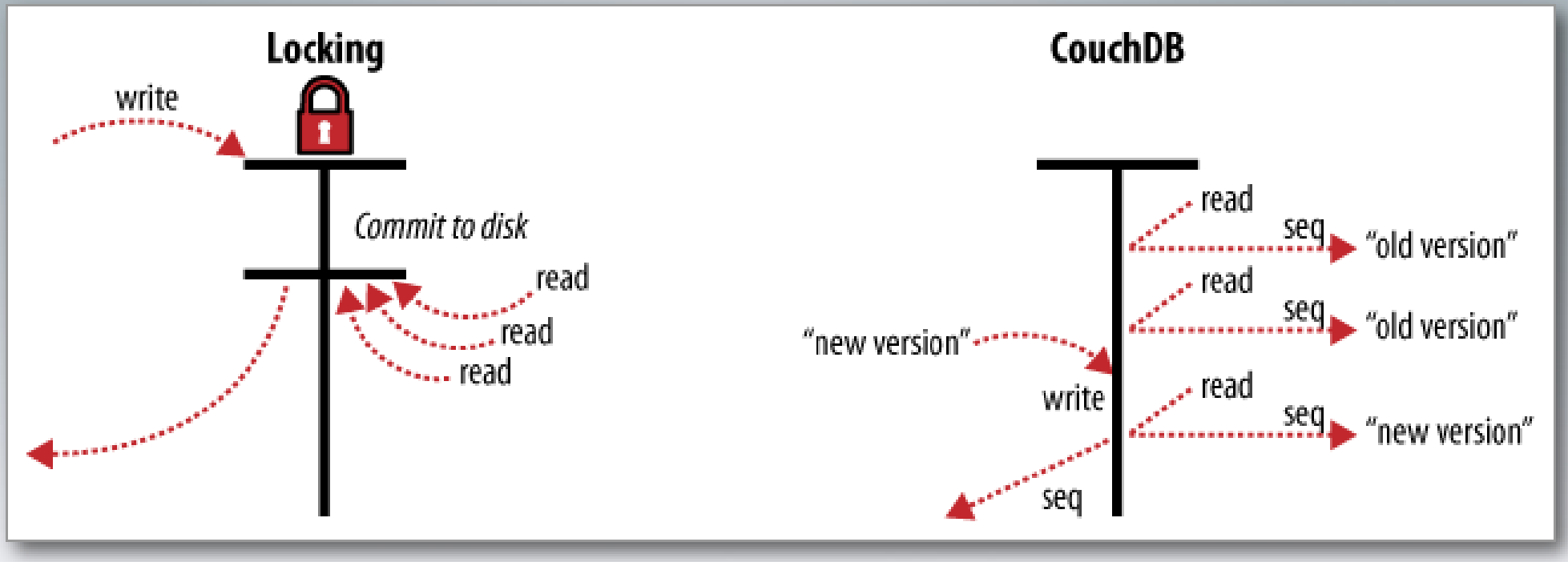
ab 15:35 Uhr

Erste Eindrücke...

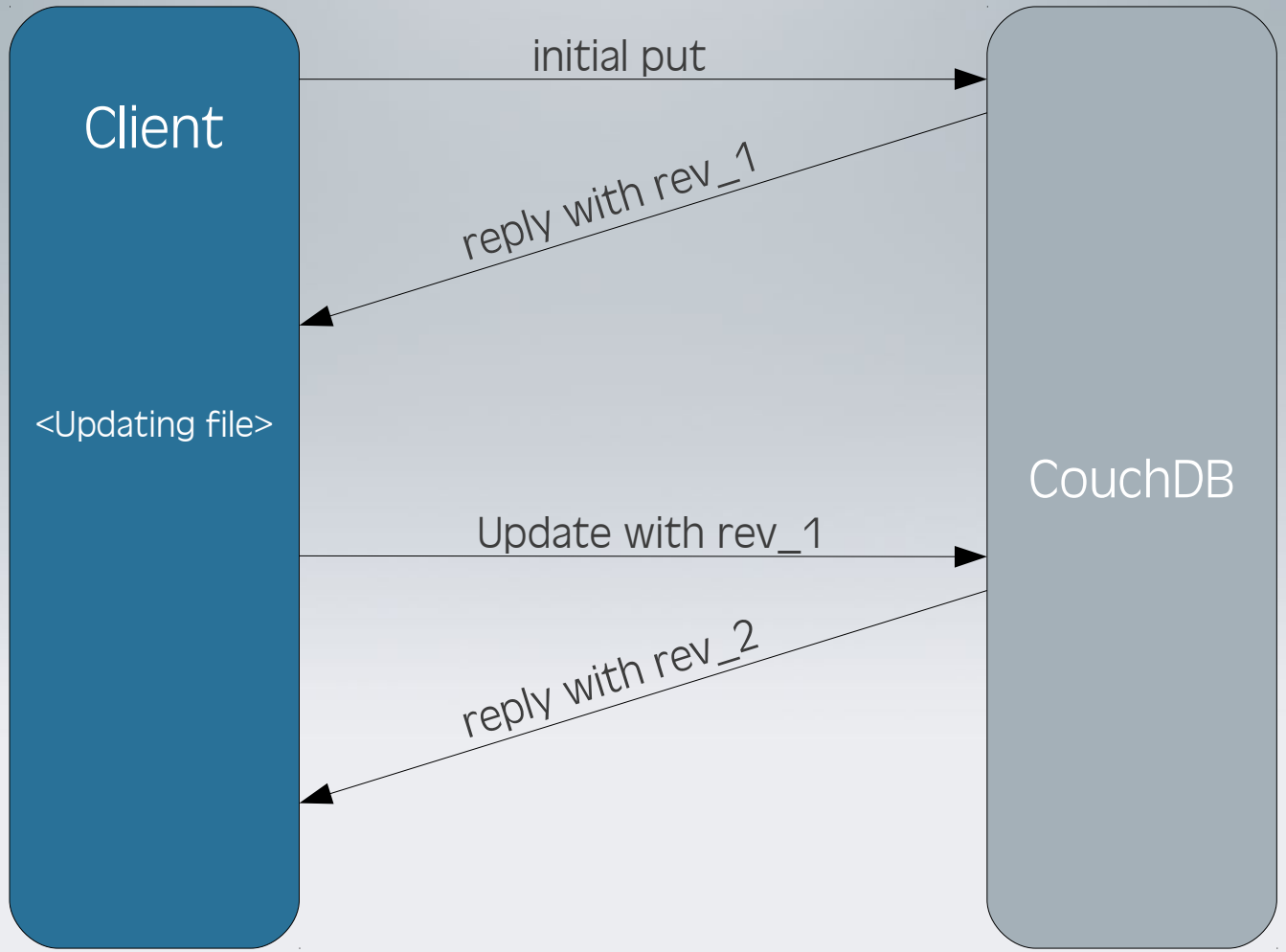
Test-DB
http://couchdb.cedarsoft.com:5984/_utils

→ „sandbox“ für Spielereien

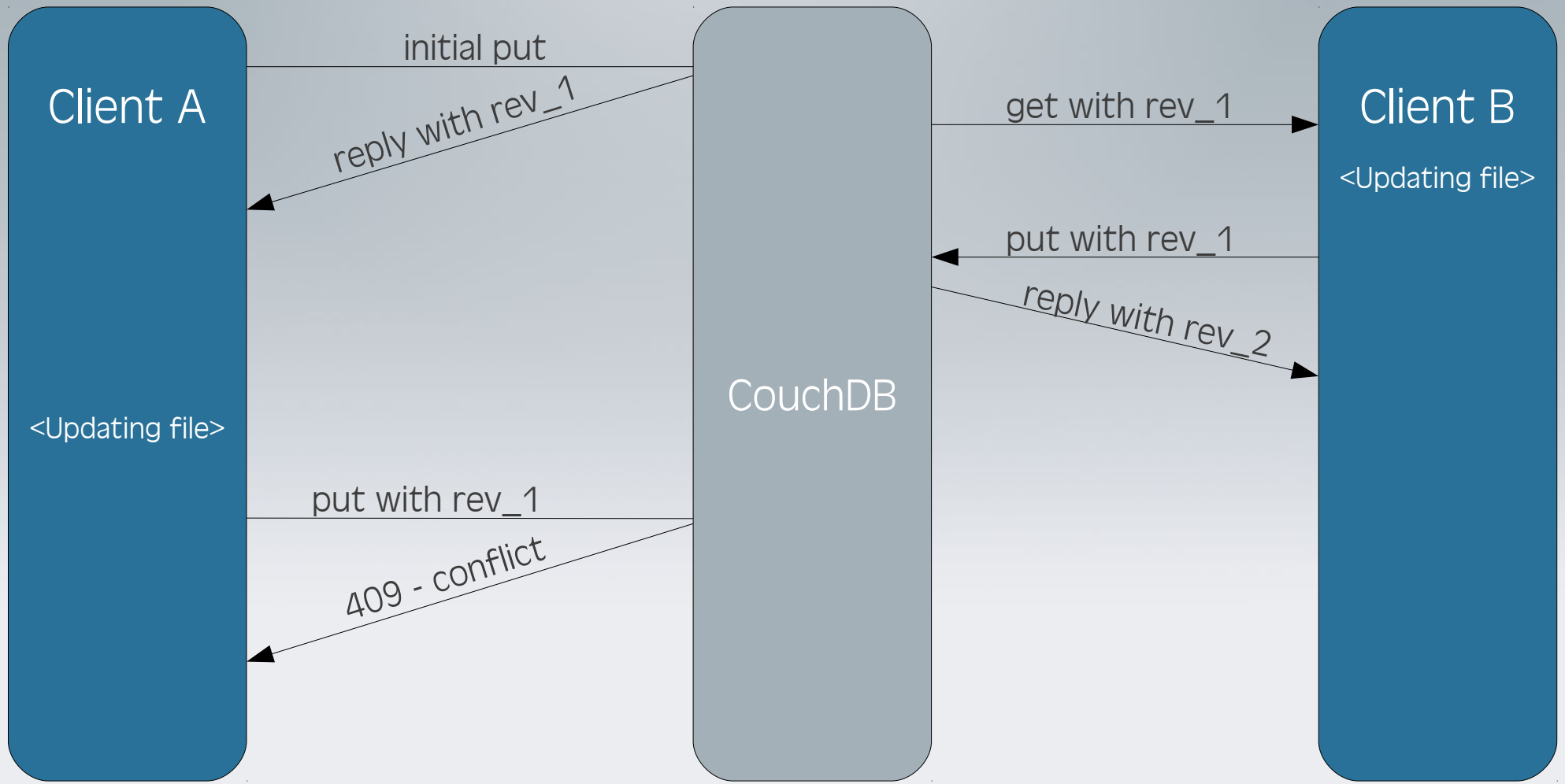
Multi-Version-Concurrency



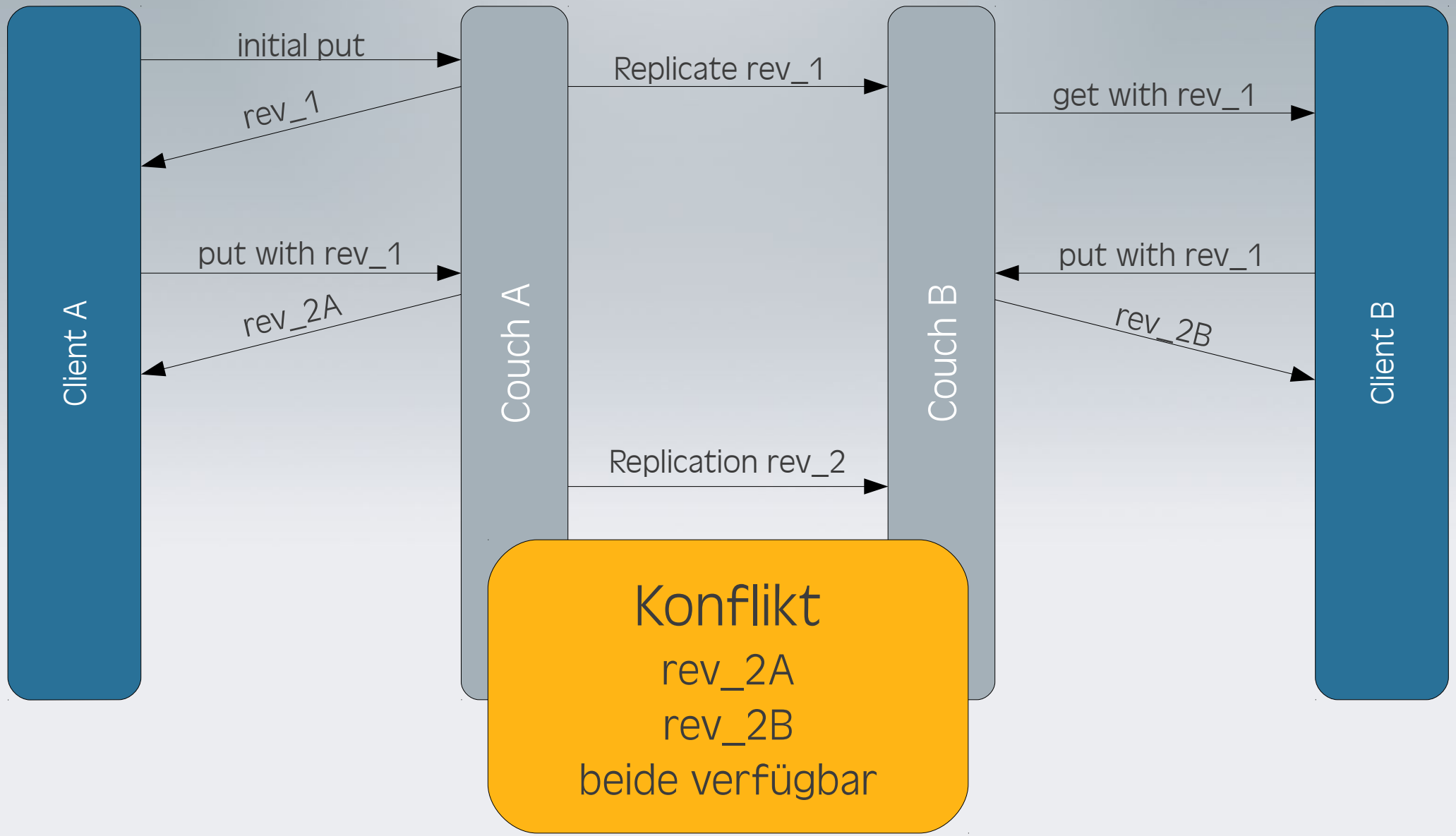
1 Client - 1 Database



2 Clients - 1 Database



2 Clients - 2 Databases



Was ist NoSQL?

„Neu entdeckt“ 2009

„Not only SQL“

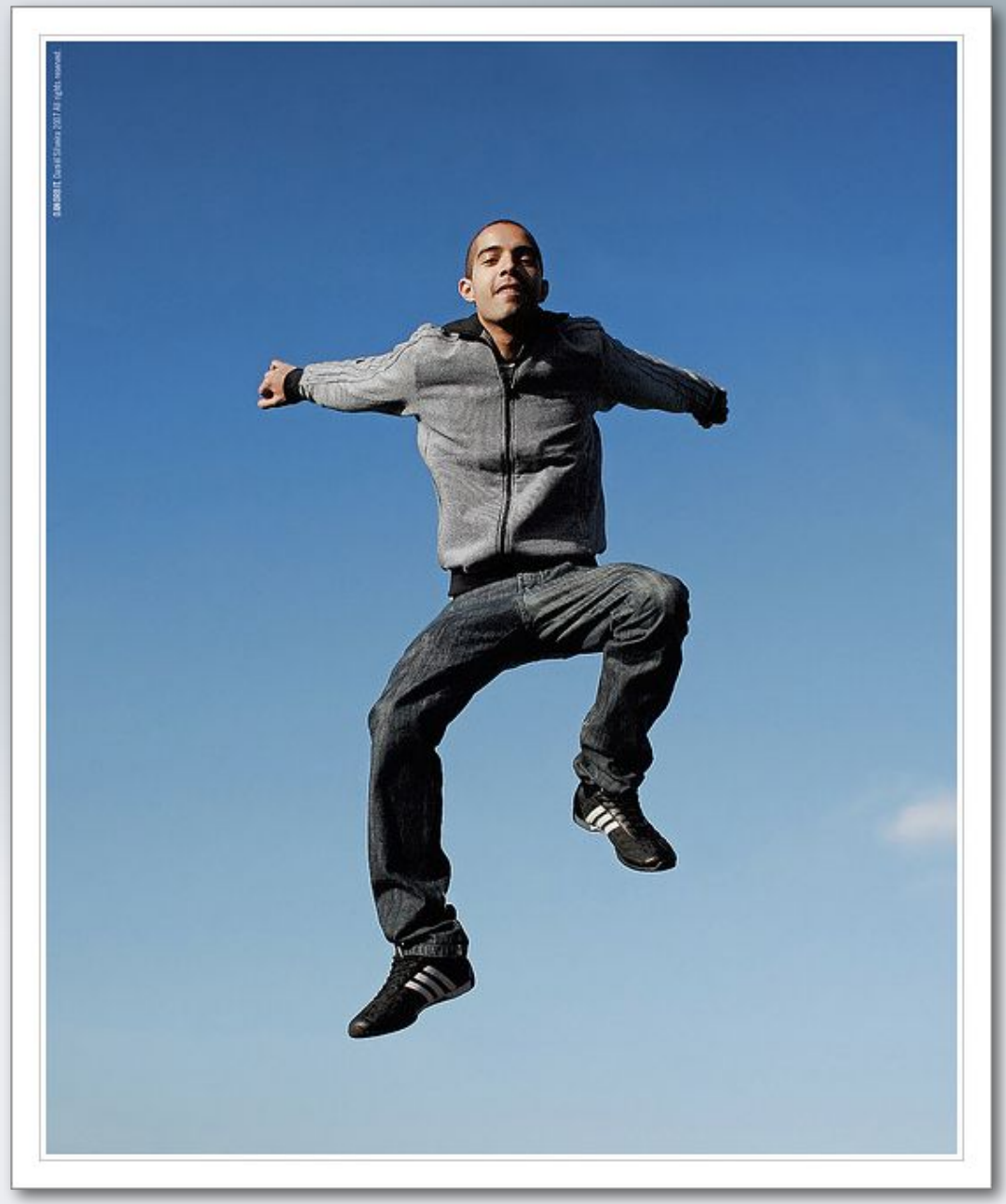
Wir sind geprägt

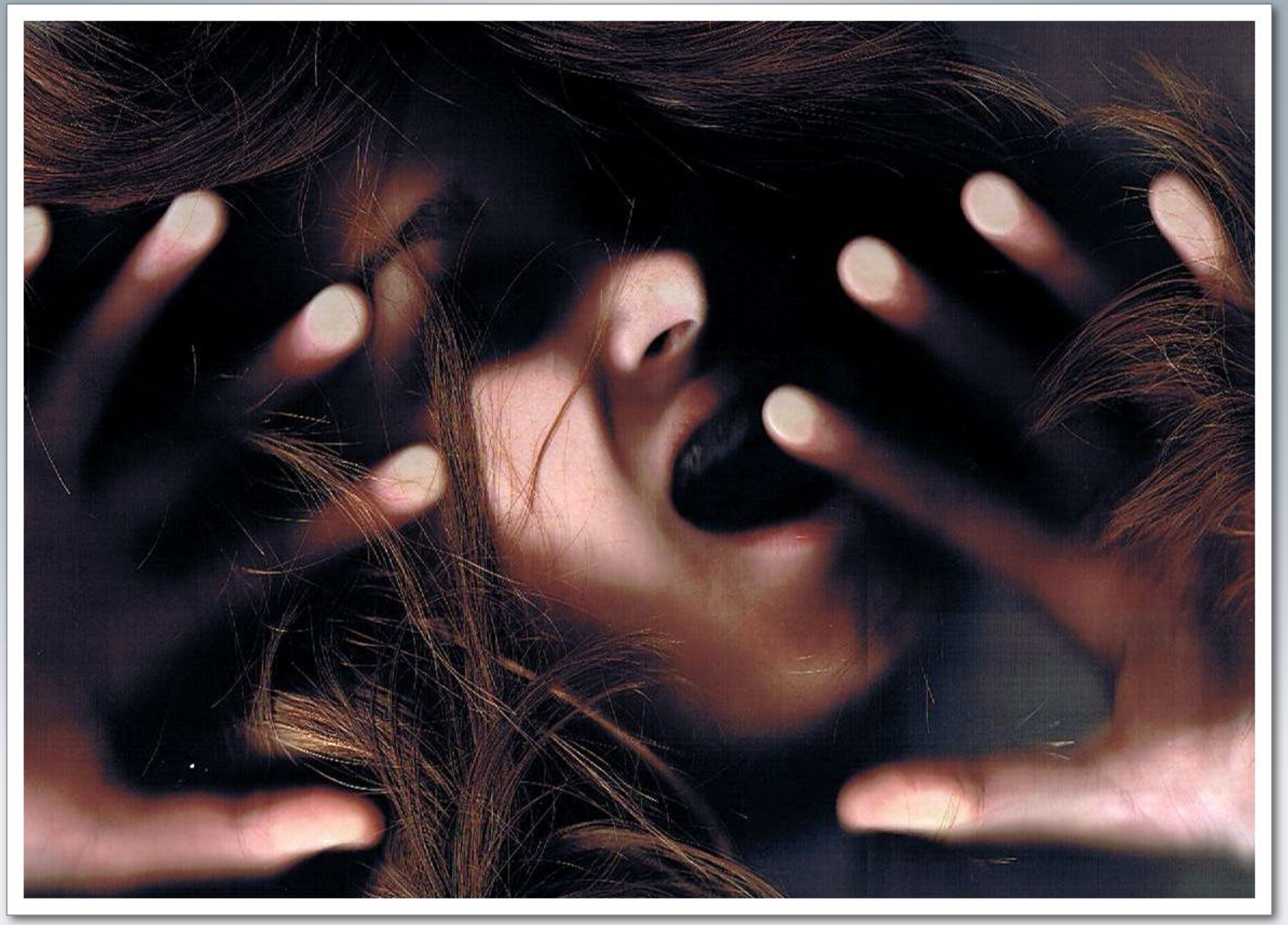
Datenbank == „Relationale Datenbank“

→ gute Erfahrungen

→ gute Tools

→ Know-How





Wo liegt das Problem?

CAP-Theorem

Brewer's Theorem

Consistency

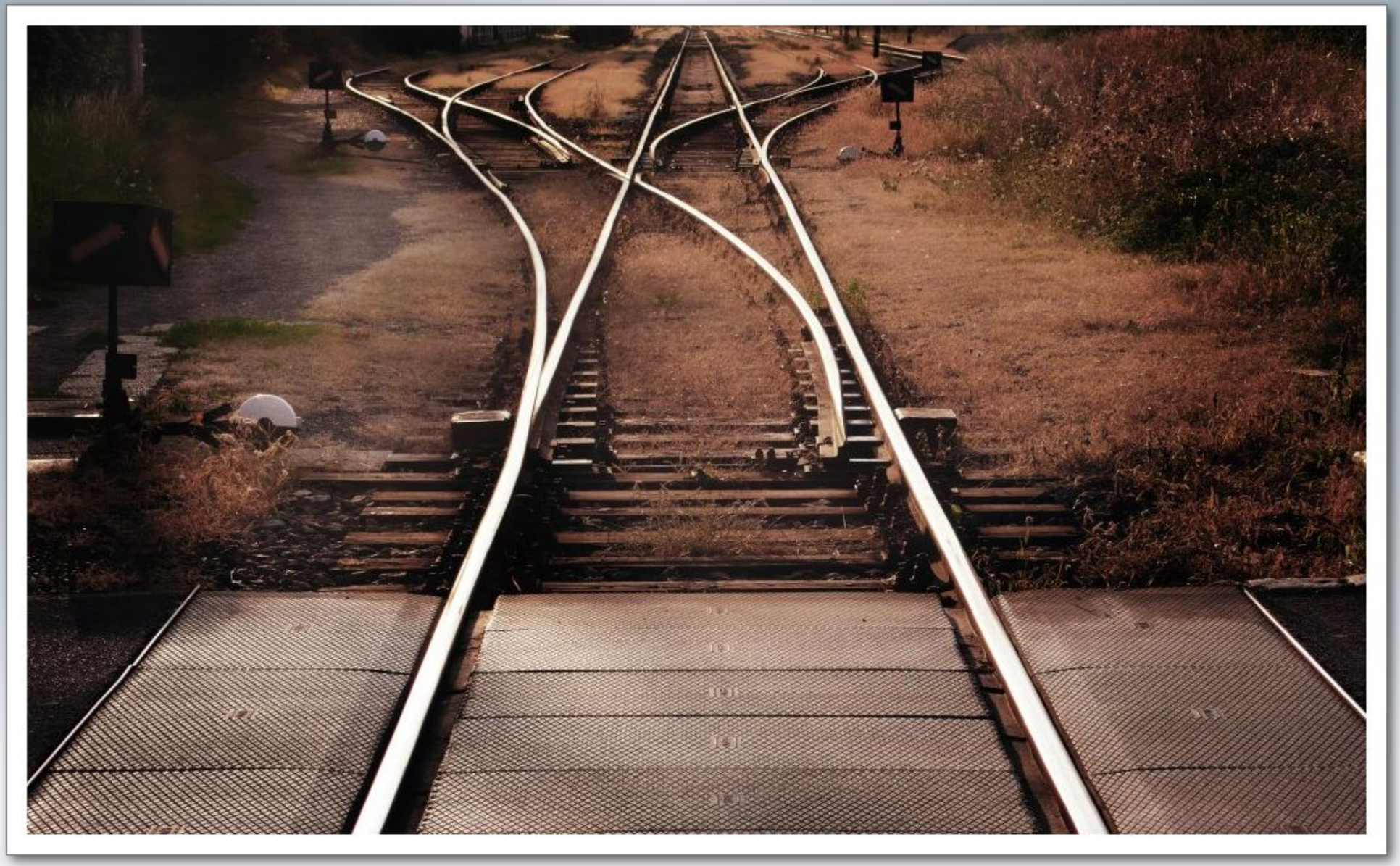
oder

Availability

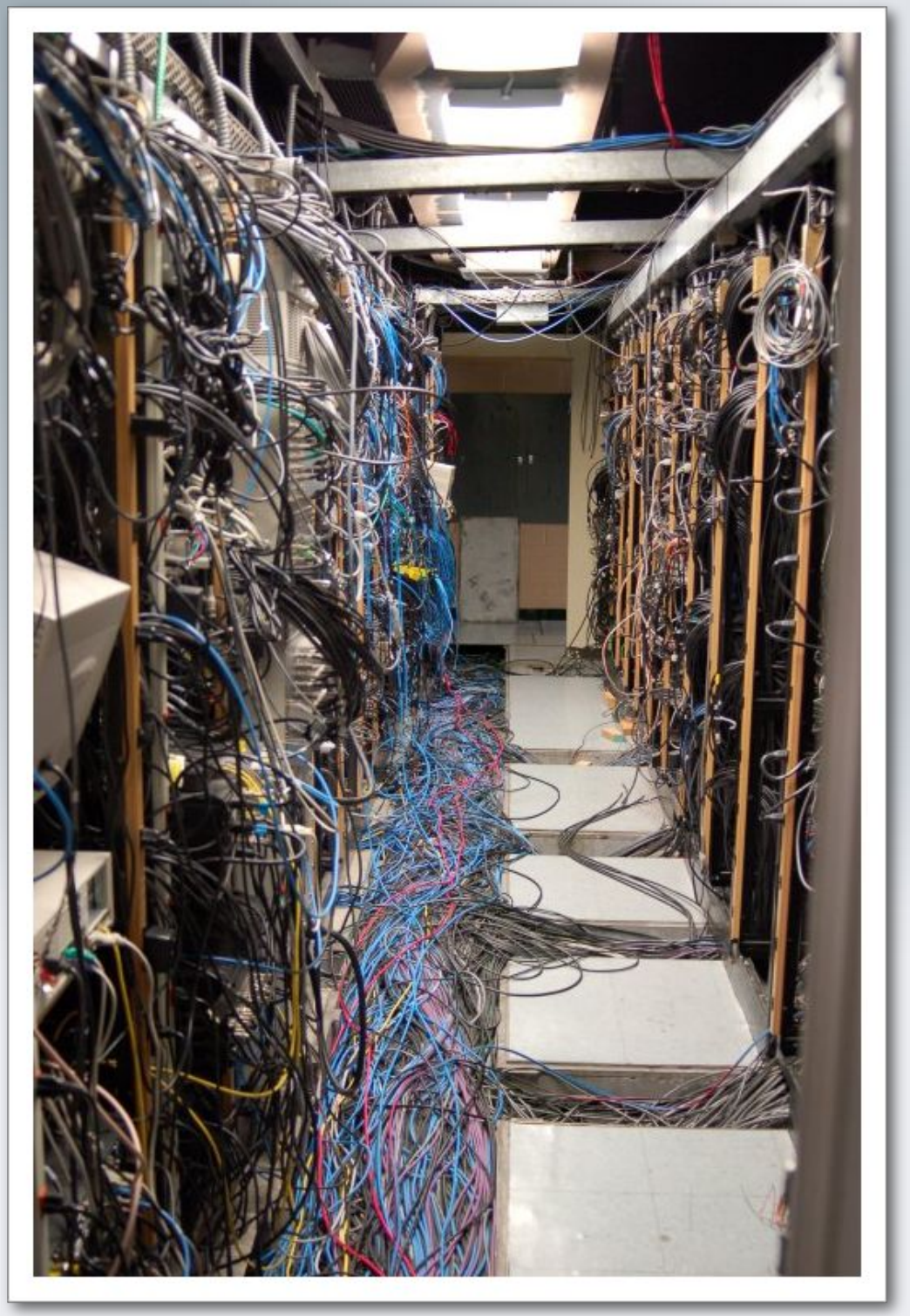
opfern, um

Partition Tolerance

zu erreichen?



Bitte entscheiden Sie sich jetzt....



Partition Tolerance

Funktioniert,
auch wenn das
Netzwerk mal
hustet...

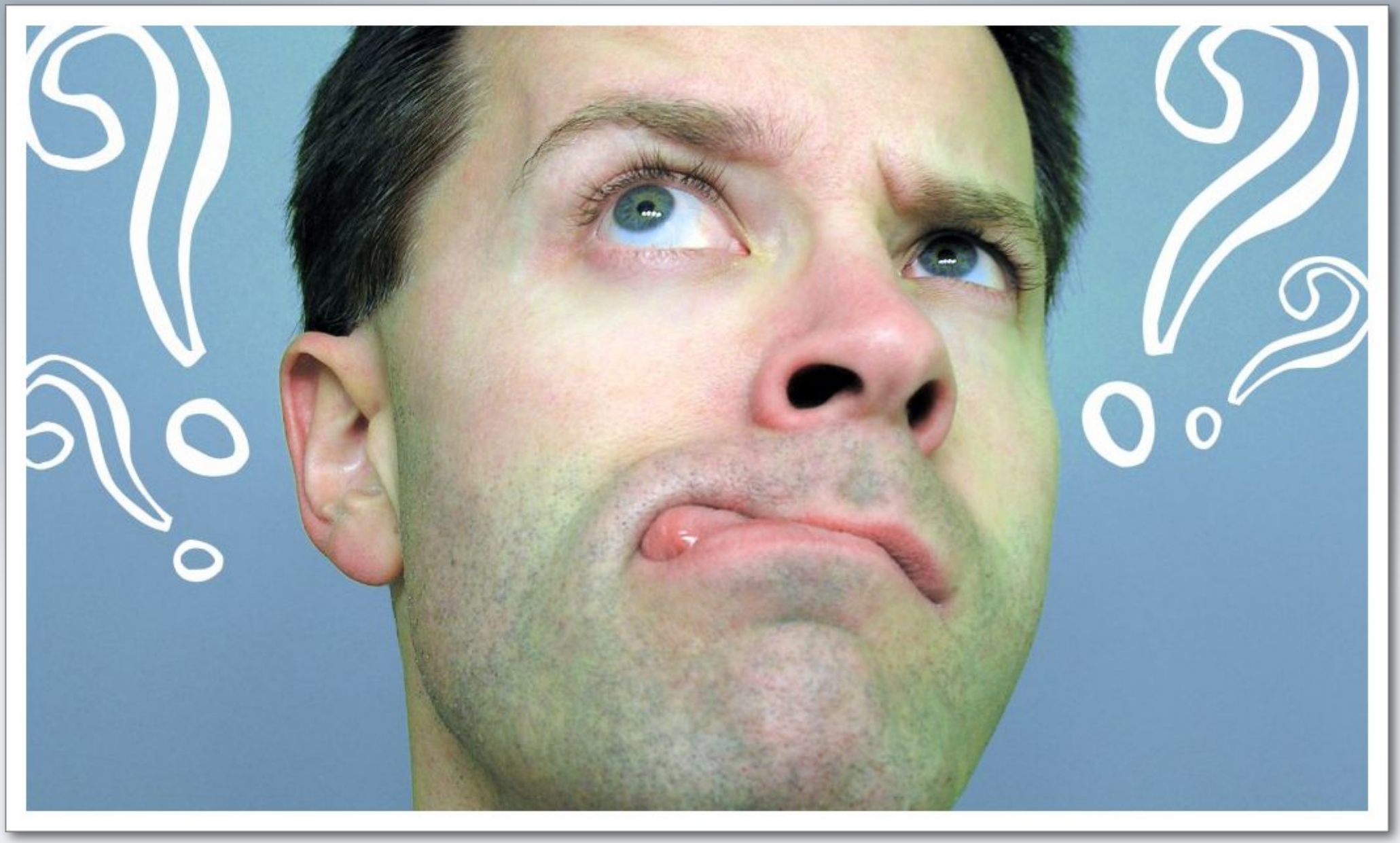
Consistency

Jeder sieht zur selben Zeit
die selben Daten

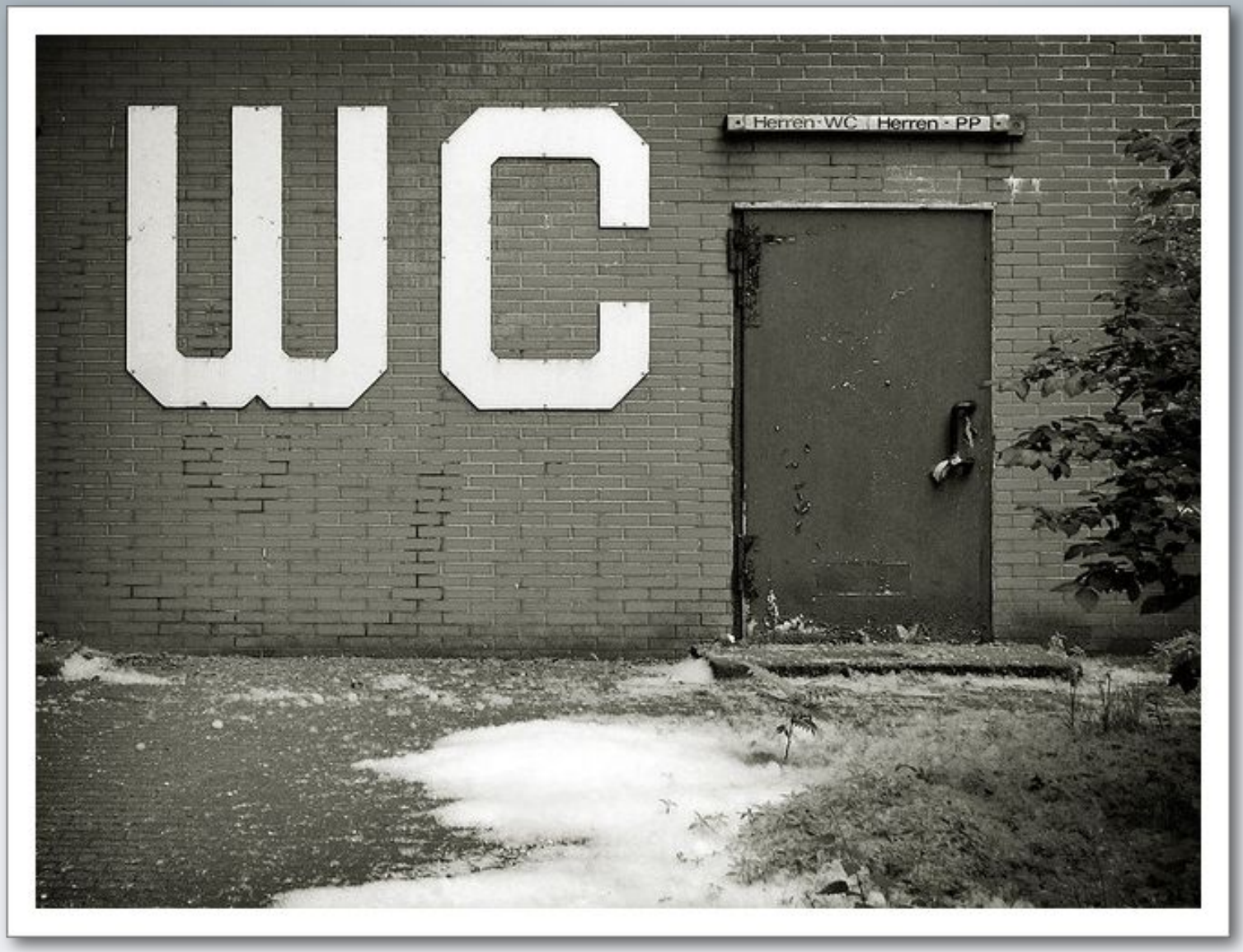
Änderungen sind atomar

Intuition: Absolut unerlässlich

Wirklich?



Availability



Jeder kann
immer Lesen
und Schreiben

Wenn ein Netzwerk beteiligt ist,
welches unter Umständen Pakete verlieren kann...

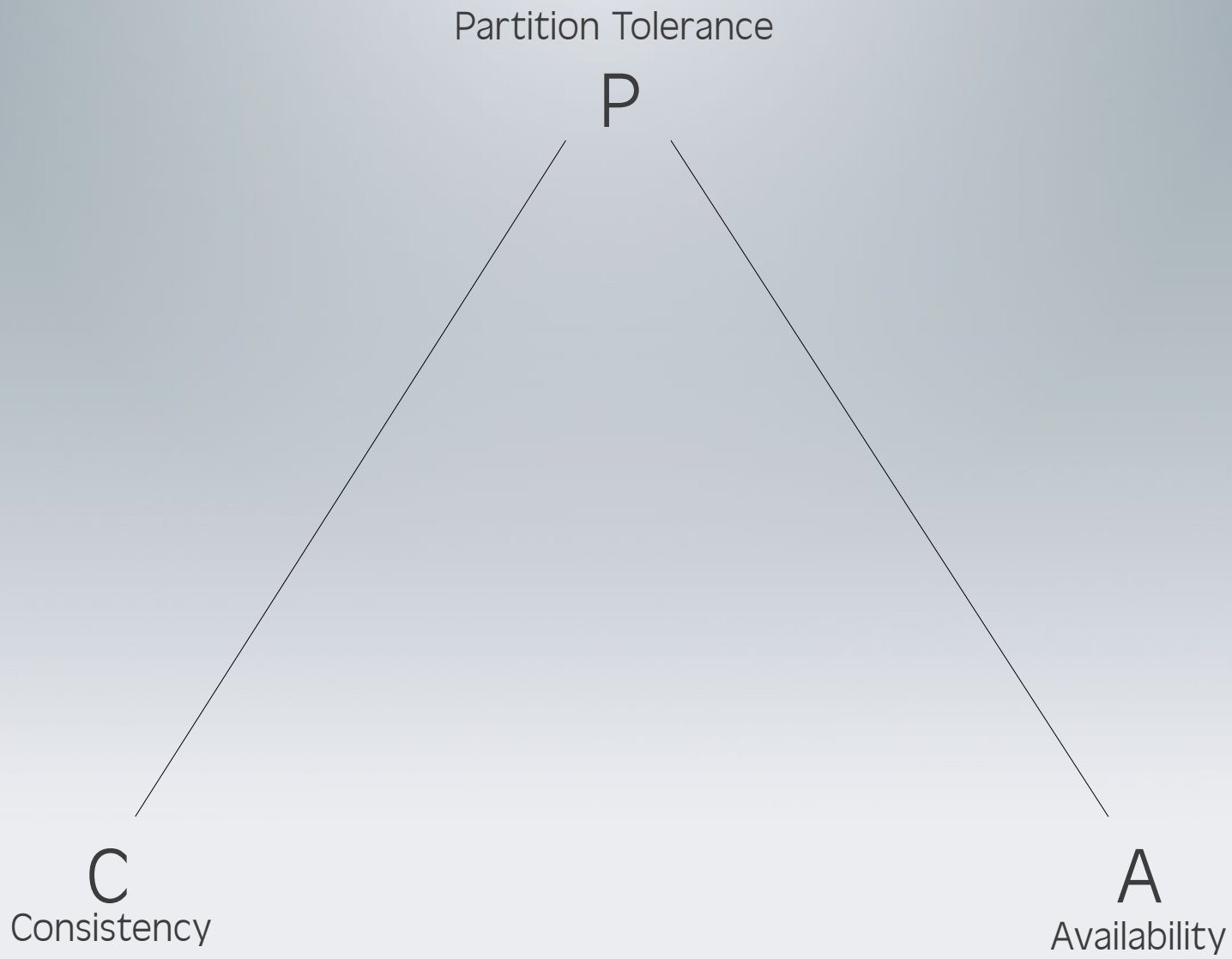
... oder Hardware, die unter Umständen ausfallen kann...

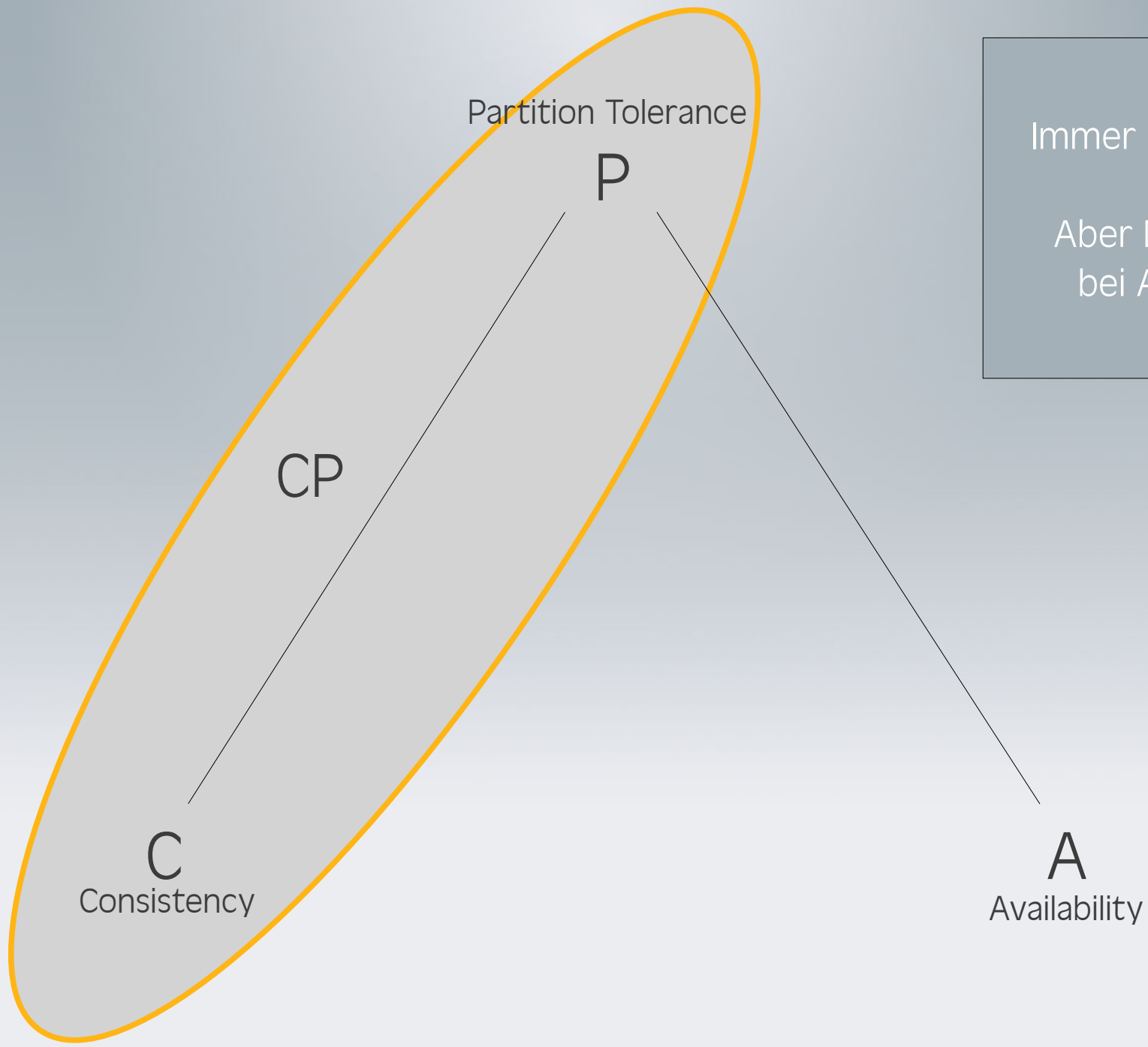
... dann können entweder
Consistency oder Availability
garantiert werden.

→ Consistency + Availability

nur mit perfekter Netzwer-Infrastruktur und Hardware

(Irgendwas geht immer schief)


Partition Tolerance
P
C
Consistency
A
Availability



Immer konsistent.

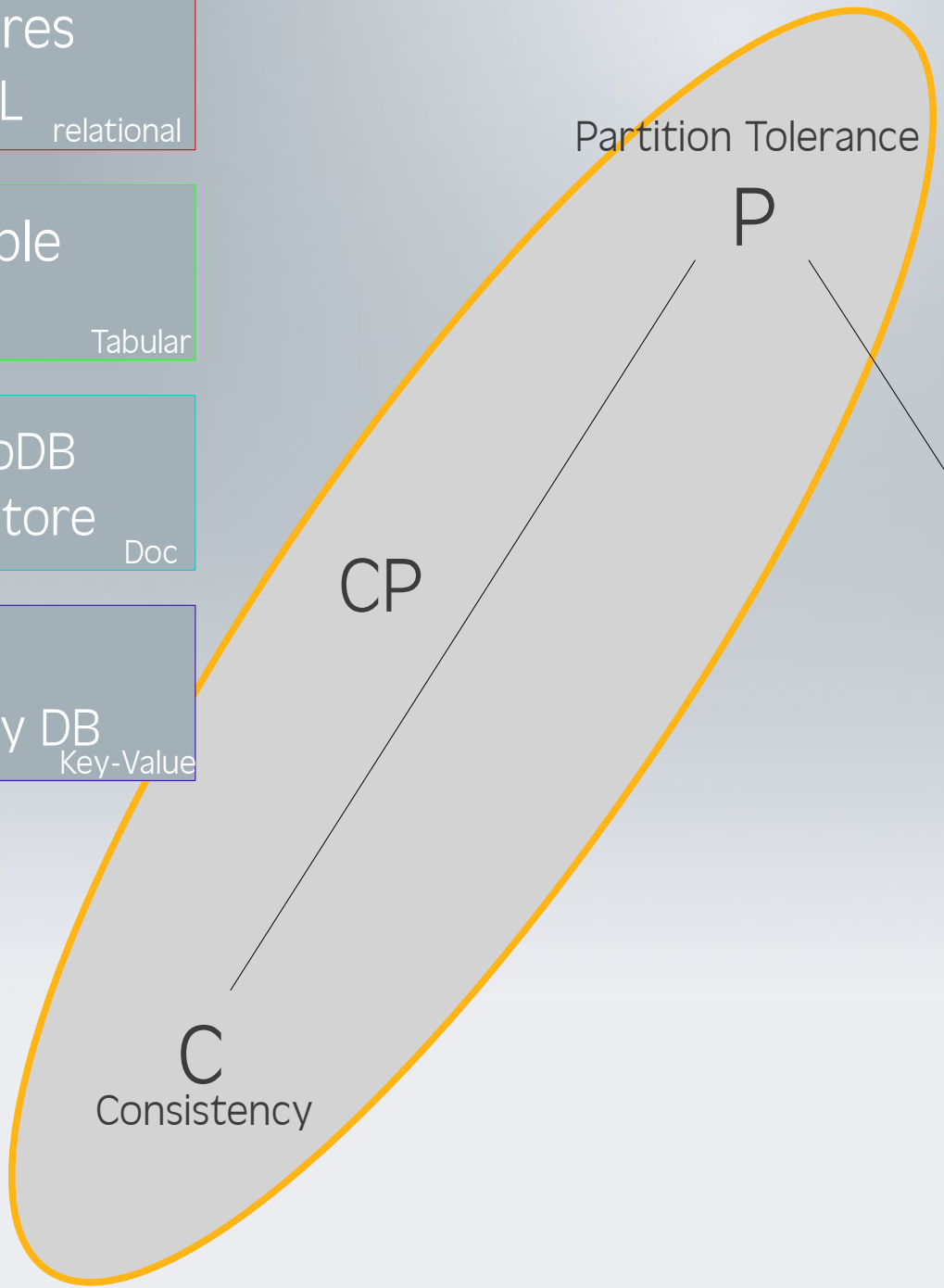
Aber Downtime
bei Ausfällen

Postgres
MySQL
relational

BigTable
Hbase
Tabular

MongoDB
Terrastore
Doc

Redis
Berkley DB
Key-Value



Immer konsistent.

Aber Downtime
bei Ausfällen

weiterer Tradeoff:

Consistency

vs

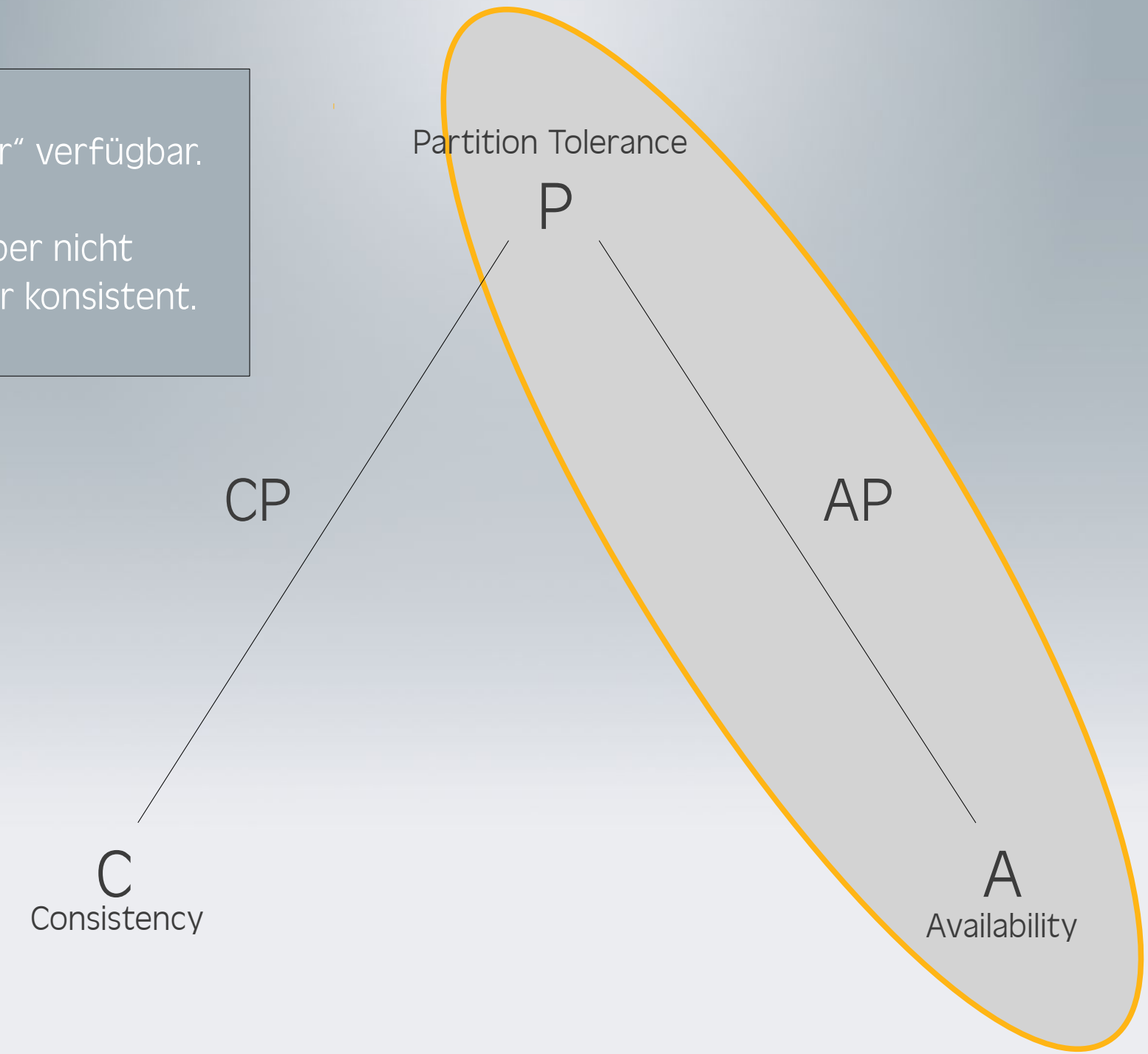
Latenz

Typisches Problem

Zur Hauptzugriffszeit fällt das System aus

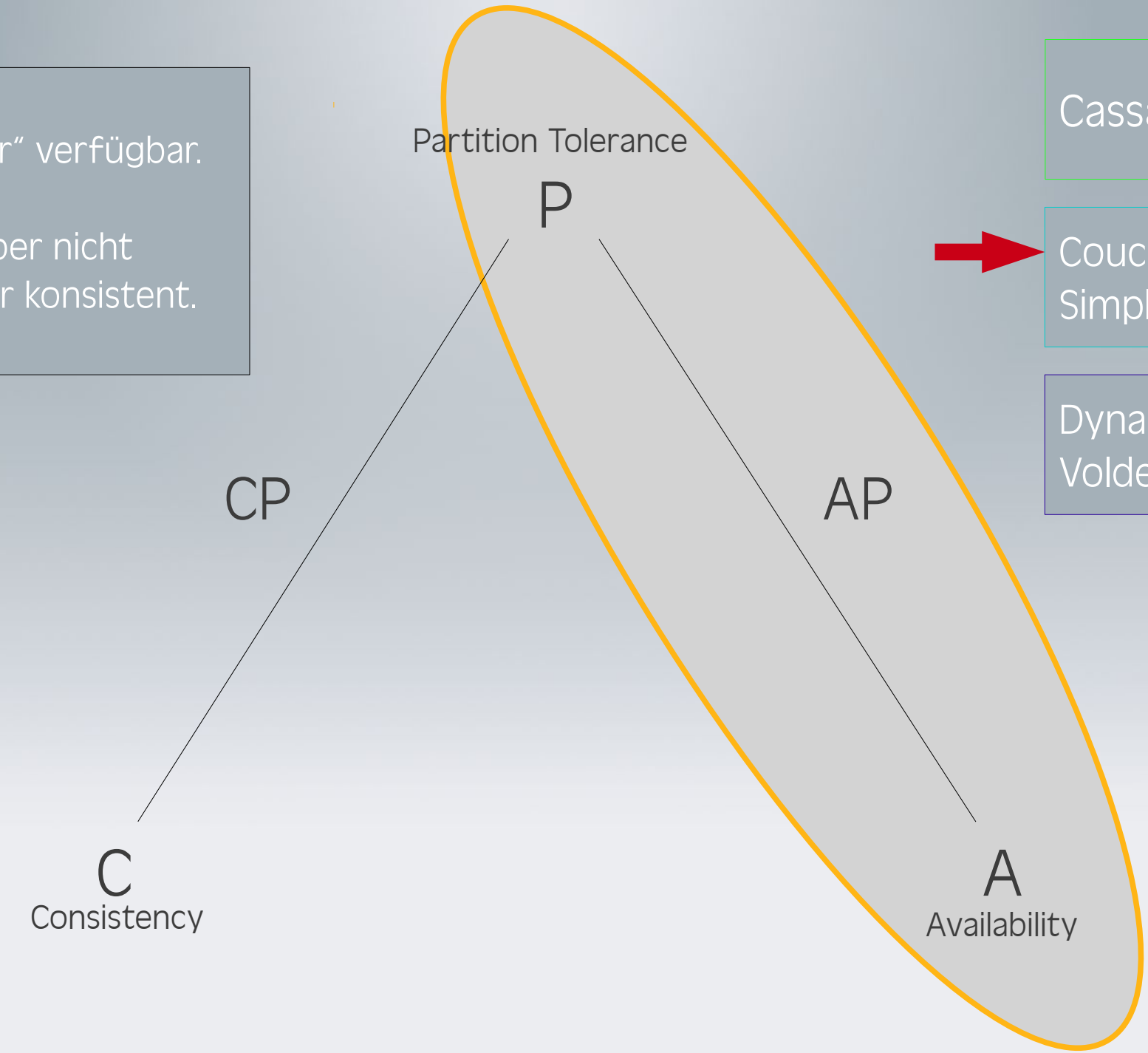
„Immer“ verfügbar.

Aber nicht
immer konsistent.



„Immer“ verfügbar.

Aber nicht
immer konsistent.



Cassandra
Tabular

CouchDB
SimpleDB
Doc

Dynamo
Voldemort
Key-Value

CouchDB „opfert“ Consistency

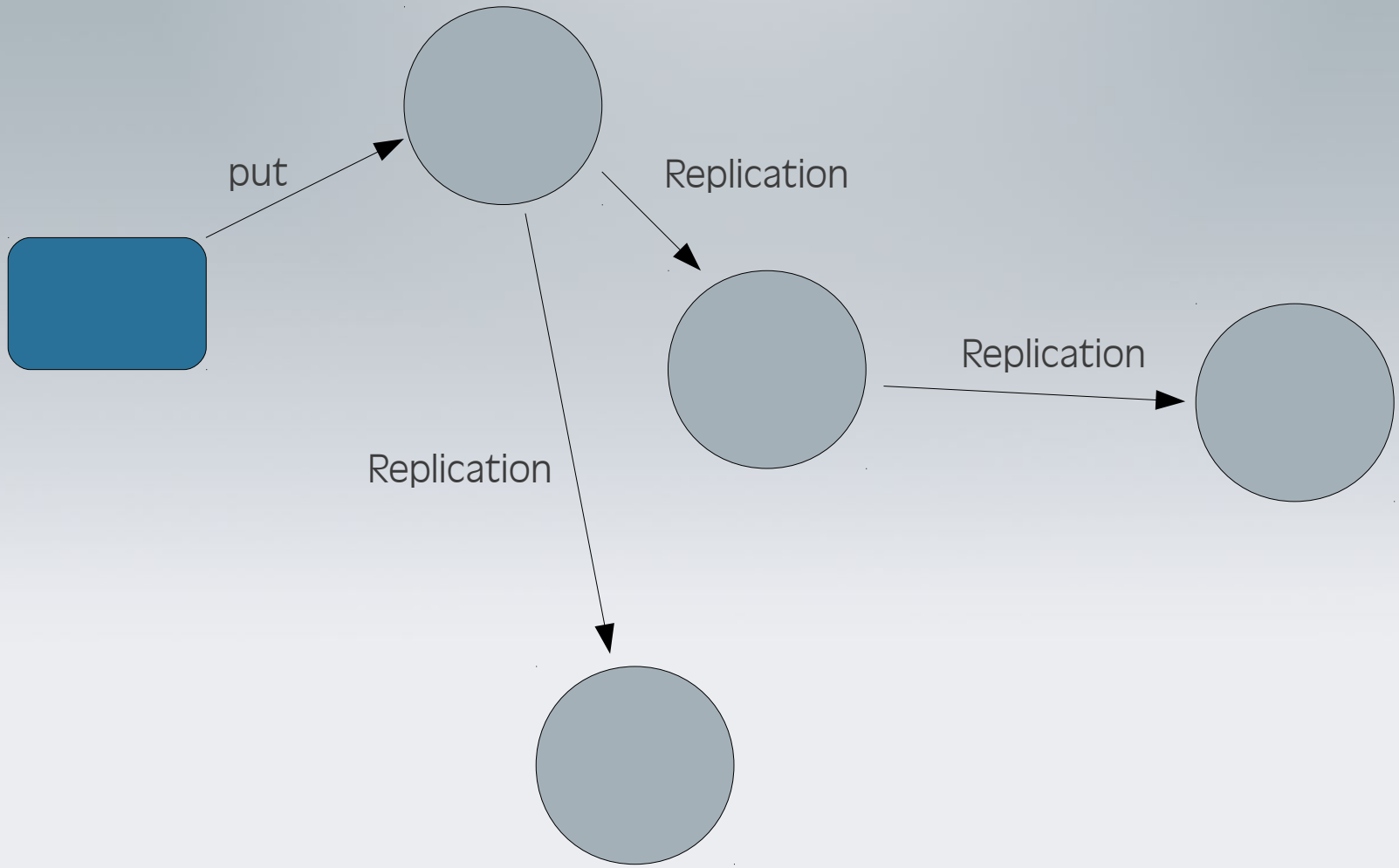
um immer verfügbar zu sein
auch wenn keine Netzverbindung da ist

Anwendung muss dafür vorbereitet sein!

Replication

CouchDB: Alle DBs sind „Master“

Inkrementelles Replizieren



Änderungen werden (eventuell) verzögert propagiert

Eventual Consistency

→ Local Consistency

Konflikte können auftreten

→ Konflikte müssen (manuell) gelöst werden

Konflikte praktisch kaum relevant

wenn

beim Design darauf geachtet wird

Updates möglichst vermeiden

- statt dessen „ergänzen“

Beispiel: Blog-Kommentare

Alle Kommentare in einem Dokument
vs
Pro Kommentar ein Dokument?

Wenn Konflikt-Vermeidung /-Resolving funktioniert, dann

Availability maximieren
Latenz minimieren

„Local Data is King“

Mindestens im LAN

Sowieso auf mobilem Gerät (Notebook, Android, IOS)

Let's talk about

Transactions

Gibt es nicht, braucht man nicht

Gibt es nicht, braucht man nicht

Schreiben eines Dokuments ist (lokal) atomar

Faustregel

Was bei RDBMs in einer Transaktion passiert,
gehört bei CouchDB in ein Dokument

Serious Business

Open your mind

Noch ein kleiner Tipp

C3: 11:10 Uhr

Das nächste große (Java-)Ding

Michael Wiedeking

Große Herausforderungen
in Java-Land?

js@cedarsoft.com

Johannes Schneider, cedarsoft GmbH
js@cedarsoft.com
http://blog.cedarsoft.com
http://cedarsoft.com

Quellen

Visual Guide to NoSQL Systems

http://blog.nahurst.com/visual-guide-to-nosql-systems

CouchDB – The Definitive Guide

http://guide.couchdb.org/

Nachtrag

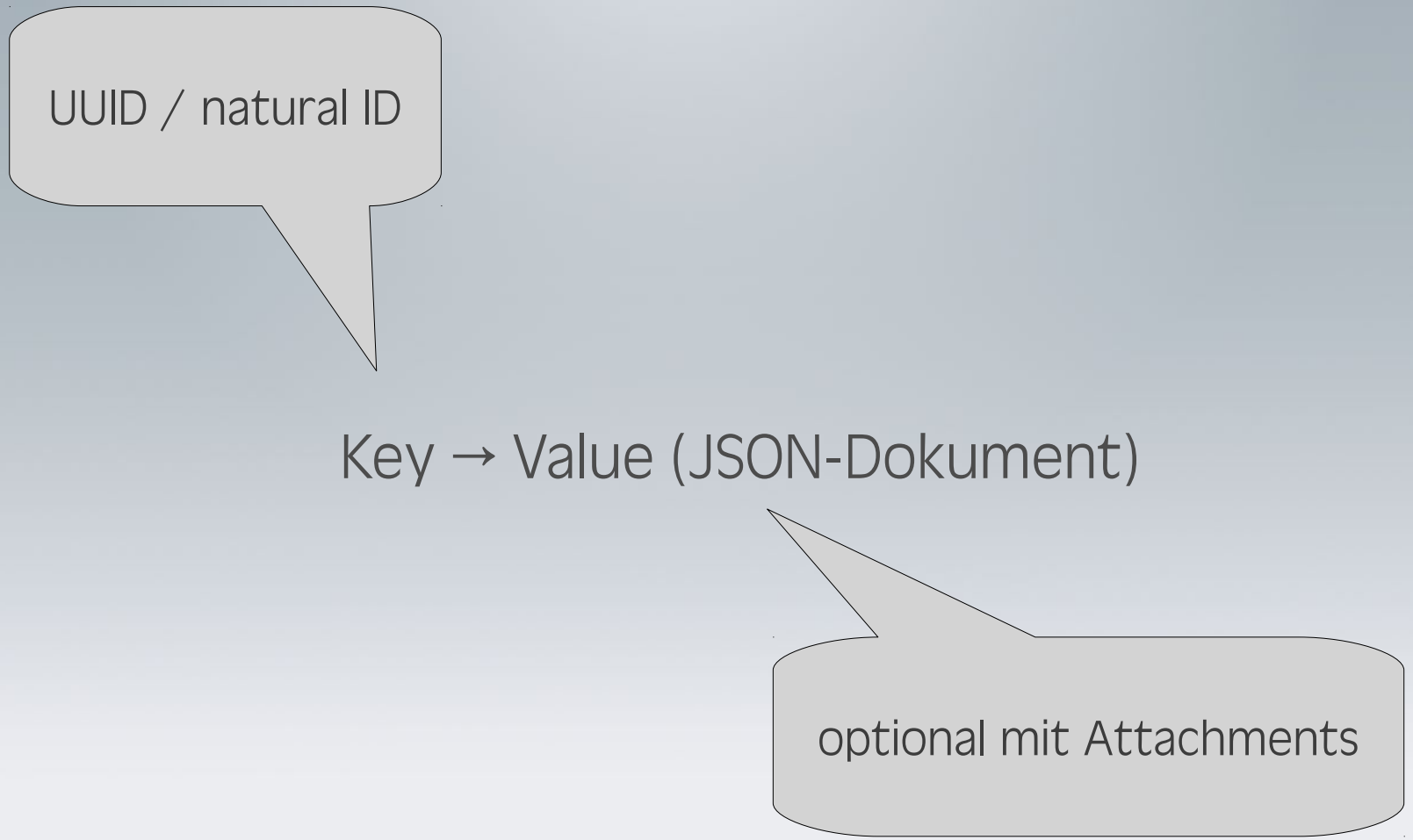
Was kommt in eine DB?

Was gemeinsam abgefragt wird

Security...

Wie speichert CouchDB die Daten?

Key → Value (JSON-Dokument)


UUID / natural ID
Key → Value (JSON-Dokument)
optional mit Attachments

```
{
  "_id": "8629b62422fde9744c80f55c21000b09",
  "_rev": "1-83611c612f3e113912d5c1b6ef21ea77",
  "firstName": "Markus",
  "lastName": "Mustermann"
}
```

```
{
   "_id": "8629b62422fde9744c80f55c21000b09",
   "_rev": "2-ed3408bb4c7fa9cf2d2cb32e34f49f04",
   "firstName": "Markus",
   "lastName": "Mustermann",
   "phones": {
      "mobile": "+49 177 123456789",
      "work": "+49 7122 123456484",
      "private": "+49 7032 123456789"
   },
   "createdAt": "2011-07-07"
}
```

Warum JSON-Dokumente?

Views

Map/Reduce „in“ der Datenbank

Aufbau von B-Trees

Key → JSON
Value → JSON

Query in B-Tree
über Key-Range

GET .../byName?startkey="a"&endkey="z"

JavaScript

(oder andere Sprachen mit Hilfe von View Servern)

```
map: function(doc){
    emit (doc.firstName, doc);
}
```

Resultat

Ein B-Tree nach „firstName“ strukturiert

```
{
  "total_rows" : 2,
  "rows" : [
    {
      "id" : "8629b62422fde9744c80f55c21000b09",
      "key" : "Markus",
      "value" : {
        "_id" : "8629b62422fde9744c80f55c21000b09",
        "_rev" : "3-3d7a92f5f459935df5b92bedc906bd7e"
        "firstName" : "Markus",
        [...]
      }
    },
    {
      "id" : "8629b62422fde9744c80f55c21001a74",
      "key" : "Martha"
      [...]
    }
  ],
  "offset" : 0
}
```

Jedes Dokument wird „gemappt“

pro Dokument können 0..n Einträge erstellt werden

```
{
   "_id": "8629b62422fde9744c80f55c21000b09",
   "_rev": "2-ed3408bb4c7fa9cf2d2cb32e34f49f04",
   "firstName": "Markus",
   "lastName": "Mustermann",
   "phones": {
      "mobile": "+49 177 123456789",
      "work": "+49 7122 123456484",
      "private": "+49 7032 123456789"
   },
   "createdAt": "2011-07-07"
}
```

```
map: function(doc){
    emit (doc.phones.mobile,doc);
    emit (doc.phones.work,doc);
    emit (doc.phones.private,doc);
}
```

Key und Value sind JSON

„Komplexer Key“

```
map: function(doc){
    emit ([doc.lastName, doc.firstName], doc);
}
```

kein Schema

```
map: function(doc){
    if (doc.lastName && doc.firstName) {
        emit ([doc.lastName, doc.firstName], doc);
    }
}
```

Best practise

[...]
"@type": "person",
"@version": 1,
[..]

```
map: function(doc){
    if ( doc['@type'] == "person") {
        emit ([doc.lastName, doc.firstName], doc);
    }
}
```

Reduce

Mehrere Values werden verrechnet

→ aber bitte auch wirklich „reduzieren“

Summen-Bildung

Kontostand

Anzahl Einträge

Statistik

Durchschnitts-Bewertungen

Standard-Abweichung

Min/Max/...

Unique Values

Top N Tags

Queries

Start-Key → End-Key
Reverse
Grouping (mit Level)

Query-Einschränkungen

Keys müssen bekannt sein

→ kein (performantes) Skip

→ keine absolute Pagination (Linked List Pagination)

Queries machen Spaß
Aber es geht nicht alles wie gewohnt

→ Vorher kurz nachdenken

Serious Business: Banken

Konten-Bewegungen mit „Transaktionen“?

Jede Buchung ein Dokument

→ „aktueller“ Kontostand durch Map/Reduce

→ Keine Konflikte möglich

Cleanup?

→ DB pro Konto und Monat/Quartal?

→ initialer Übertrag