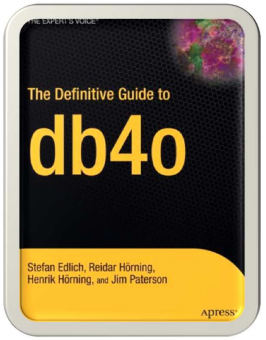
DATABASES = FUN ?

```
select fun, profit from real_world where rational = false
```

Thanks!

Teching Software-Engineering

- **20 years with *Relational Databases***
- **PostgreSQL, SQLite, Oracle ...**



2006

db4o

2004

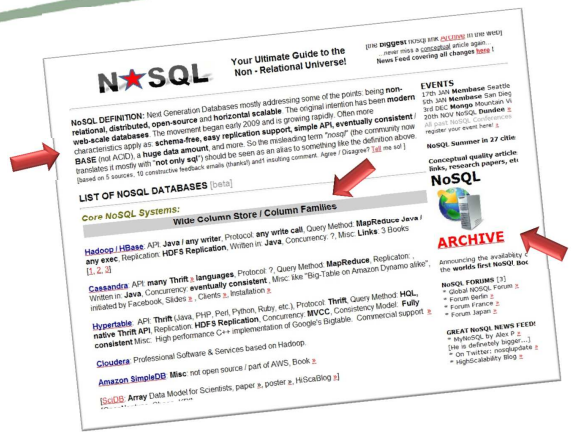
icoodb

2006

2008

http://nosql-database.org

2010



nosqlberlin.de
nosqlfrankfurt.de
nosql powerdays

2010

Oracle, IBM, etc. wollen auch...

NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more. So the misleading term *"nosql"* (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

Und nicht vergessen...

Einfache Installation / Handling + Fun!

non-relational

NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more. So the misleading term *"nosql"* (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

Scale-out!

NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more. So the misleading term *"nosql"* (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

Open-Source

nur indirekte finanzielle Interessen

NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more. So the misleading term *"nosql"* (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

Web-Scale

Historie

Google

NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more. So the misleading term *"nosql"* (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

Schema free

Strange Loop Pattern:

Design for Crash!

ERLANG

NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more. So the misleading term *"nosql"* (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

Replication fun?

```
POST /_replicate HTTP/1.1
{"source":"database","target":"http://example.org/database"}
```

NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more. So the misleading term *"nosql"* (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more. So the misleading term "nosql" (the community now translates it mostly with "not only sql") should be seen as an alias to something like the definition above.

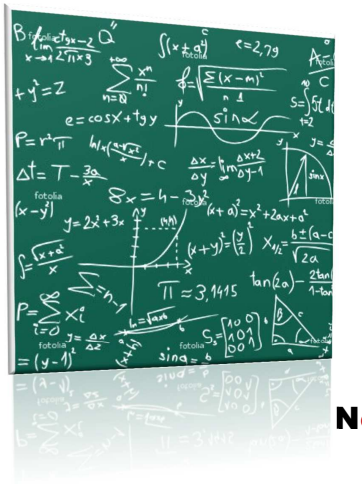
kommt noch...

a NOSQL summer

nosqltapes.com !

JAPAN

NoSQL is specialization!



NoSQL foundations:

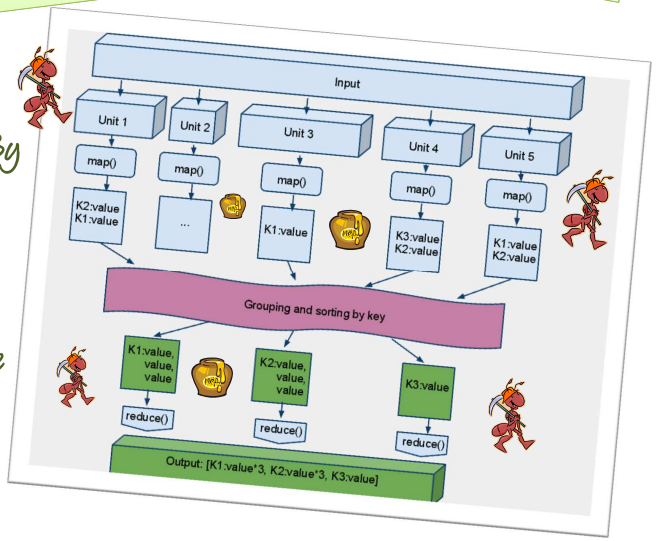
Theorie

funktionale (graph) Dekomposition? Oder…

Schutzpatent ☺

Group By

Aggregate



Programmierung top!

herrlich parallelisierbar

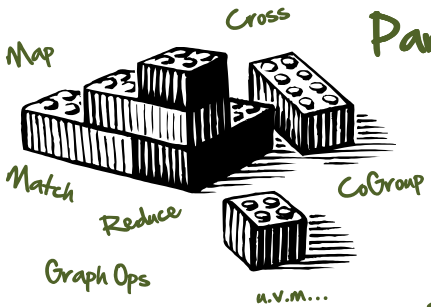
Programmierung nervt!

Nur `large data indexing`

„A giant step back! Incompatible, missing features, not new, …"

Starke Konkurrenz: Stratosphere (TUB), ePic, SwissBox, etc.

Parallelization Contracts



compile, analyze, optimize

auf einer atmenden Cloud!

Eventually Consistent

ACID

- Amazon Dynamo
- MySQL Replikation

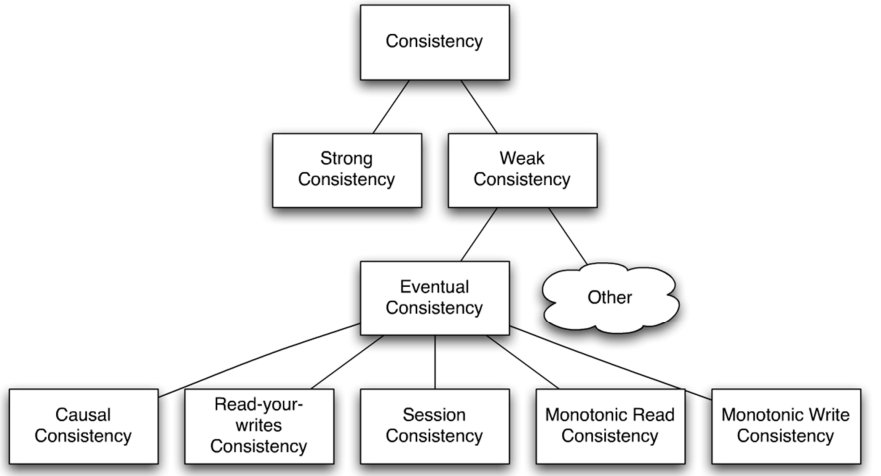
BASE

facebook

myspace

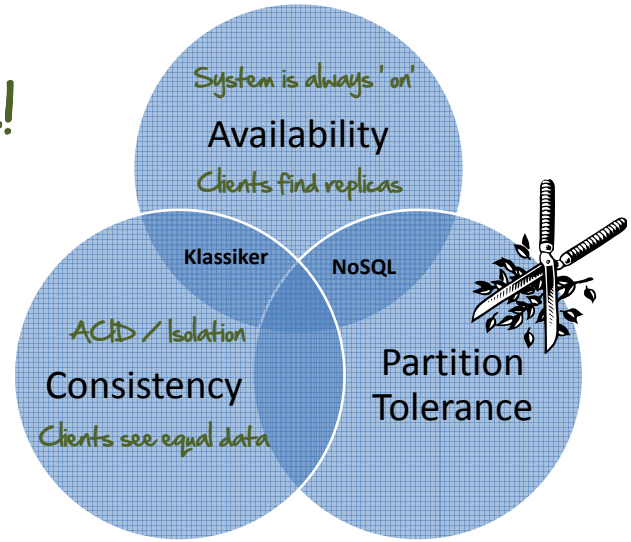
twitter

Consistency Models



CAP Theoreme

Pick 2!



„Don't throw **C** away so easy! It's complex."

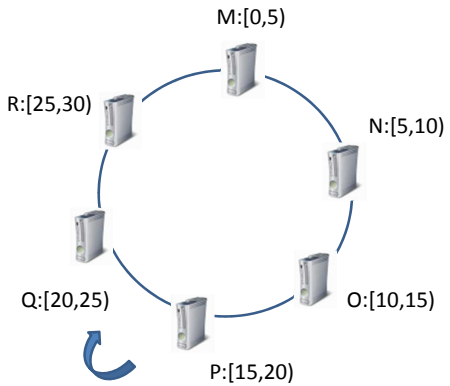
What you really have is:

1. Application errors ★
2. Repetable DBMS errors ★
3. Unrepeatable DBMS errors
4. Operating System errors
5. Hardware failure in cluster
6. Network partition in local cluster
7. A disaster ★
8. WAN failure

- 6 = Network Partition is rare
- 3,4,5,6 is mostly a Single Node
- Algorithms can help!

„give up P rather than sacrificing C. Use VoltDB or NimbusDB"

Consistent Hashing



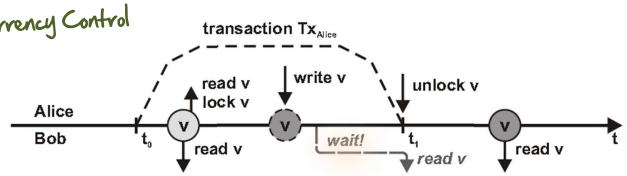
HASH	KNOTEN	REPLIKAT
2	M	N,O
8	N	O,P
10	O	P,Q
17	P	Q,R
22	Q	R,M
26	R	M,N

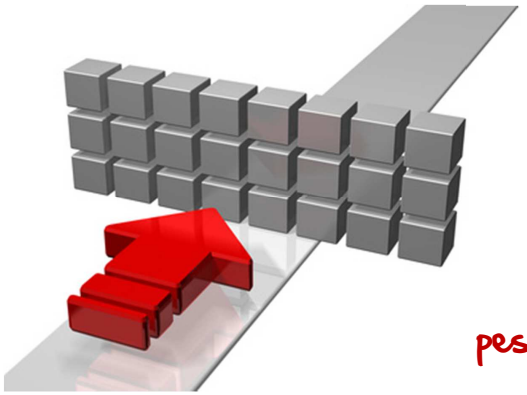
W ✓= 2*W
R ✓= 1*R

- ausfallsicher ✓
- leicht erweiterbar ✓
- gut verteilt / vnodes ✓

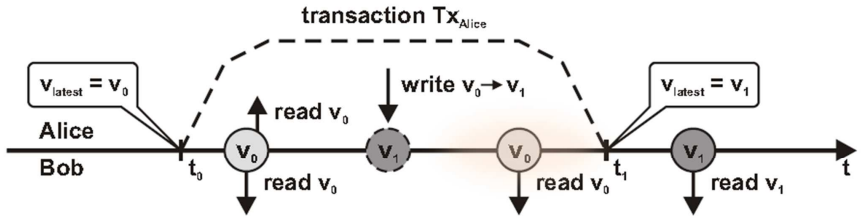
MVCC

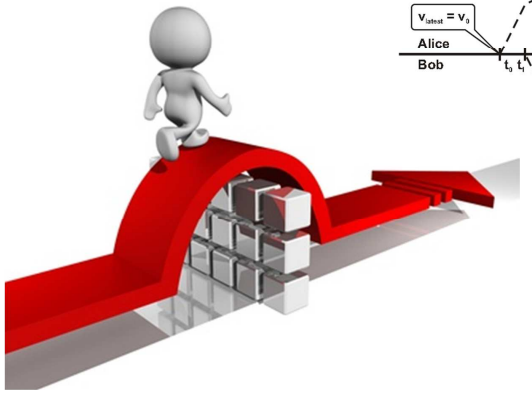
Multi Version Concurrency Control

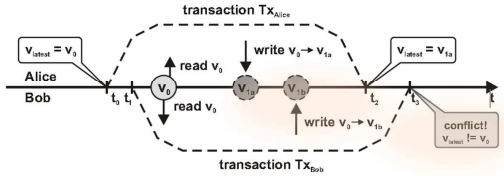




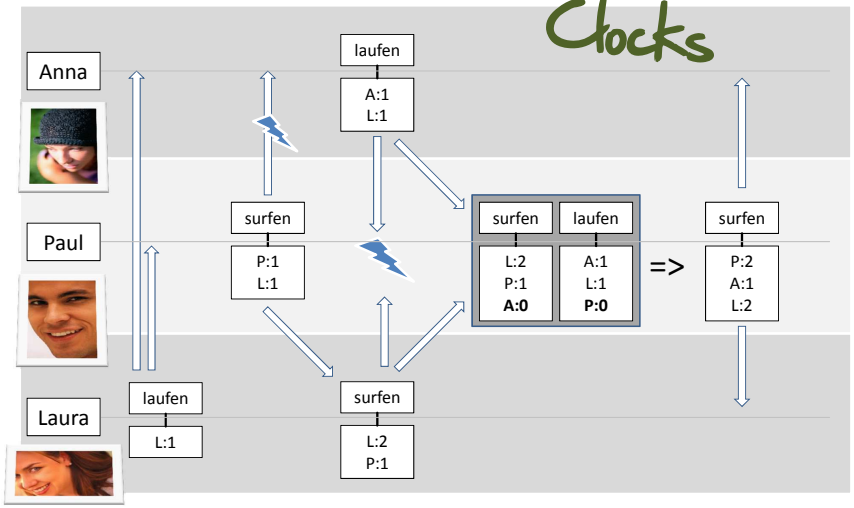
pessimistisches Locking?

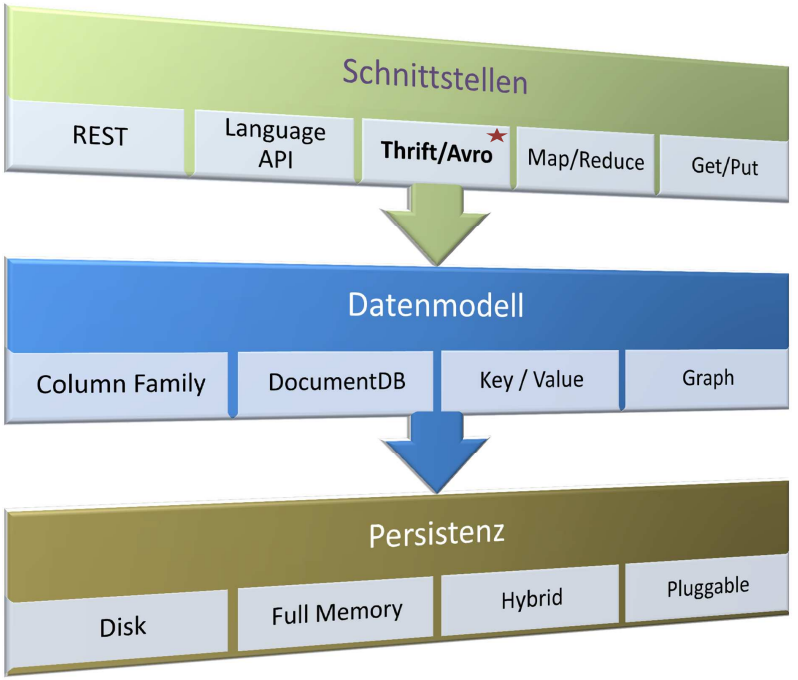






Vector Clocks





Google Protocol Buffers

You write a `.proto` file like this:

```
message Person {
  required int32 id = 1;
  required string name = 2;
  optional string email = 3;
}
```

Then you compile it with `protoc`, the protocol buffer compiler, to produce code in

Then, if you are using C++, you use that code like this:

```
Person person;
person.set_id(123);
person.set_name("Bob");
```

Thrift

```
struct UserProfile {
  1: i32 uid,
  2: string name,
  3: string blurb
}
service UserStorage {
  void store(1: UserProfile user),
  UserProfile retrieve(1: i32 uid)
}
```

=>

```
# Make an object
up = UserProfile(uid=1,
                 name="Mark Slee",
                 blurb="I'll find something to put here."

# Talk to a server via TCP sockets, using a binary protoc
transport = TSocket.TSocket("localhost", 9090)
transport.open()
protocol = TBinaryProtocol.TBinaryProtocol(transport)

# Use the service we already defined
service = UserStorage.Client(protocol)
service.store(up)

# Retrieve something as well
up2 = service.retrieve(2)
```

- JSON!
- Binary data transfer
- automatic RPC generation
- no code generation
- Client + Server tauschen Schema bei Änderung

Daten-Modelle

Relationale DB	Server	Database	Table	Primary Key			
Key Value DB	Cluster	Keyspace		Key	Value		
Column Family DB	Cluster	Table / Keyspace	Column Family	Key	Column Name	Column Value	Super Column optional
Document DB	Cluster	Docspace		Doc Name	Doc Content		
GraphDB	Server	Nodespace	Nodes & Links				

Wide Column Stores / Column Families

```
KEY_1 => COLUMN_FAMILY_ONE => A => Value
                           => B => Value
      => COLUMN_FAMILY_TWO => C => Value

KEY_2 => COLUMN_FAMILY_ONE => A => Value
                           => D => Value
      => COLUMN_FAMILY_TWO => B => Value

KEY_3 => COLUMN_FAMILY_ONE => A => Value
                           => B => Value
                           => F => Value
      => COLUMN_FAMILY_TWO => C => Value
                           => D => Value
```

```
Keyspace:WildWest
CF:Users
  "Jim"   {„reactivity":"2"}
  "Joe"   {„reactivity":"9"   „nick_name":"Little Joe"}
CF:Horses
  "Kelly" {"color":"white"    "owner":"Jim"}
CF:Weapons
  "Colt"  {"anno":"15"        "owner":"Joe"}
CF:Duels
  "High_Noon"  {"player_1":"Jim"   "player_2":"Joe"   "winner":"Joe"}
CF:Messages
  "High_Noon_1" {"from":"Joe"   "to":"Jim"   "text":"Looooser!"}
```

+super

Kunden-ID	Vorname	Nachname	Straße	Stadt	Land	PLZ	Telefon	E-Mail
123	Robert	Schmidt	Hauptstr. 123	Düsseldorf	Deutschland	40210	222-333-4444	
456	Jens	Hansen	Bahnhofstraße 456	Köln	Deutschland	50441	333-444-5555	
789	Doris	Thomas	Schillerstraße 789	München	Deutschland	80331	444-555-6666	dthomas@xyz.com

+N

Column Family

DocumentDBs

Key/ValueDBs

Voldemort, Chordless, Scalaris, Dynamo / Dynomite

GraphDBs

andere

db4o, Versant, Objectivity, Gemstone, Progress, Mark Logic, EMC Momentum, Tamino, GigaSpaces, Hazelcast, Terracotta, ...

hadoop

+ Skalierung = new node
+ Community
+ API
- Replikation
- Aufsetzen, Optimierung, Wartung

amazon SimpleDB

+ stressfreie SaaS Lösung
+ transparent scaling
- UTF-8 String
- Daten liegen bei Amazon
- kein tuning / config

Cassandra

+ Skalierung = new node
+ Replikation
+ Konfiguration (r, w)
- Dokumentation
- Abfragen
- (storage-conf.xml)

Document DBs

Erlang

Views

Lounge

Futon

JSON, semistructured Docs

1 URL Replikation

CouchApps

JS Client

keine Middleware

DB+WebS+App

2. Runde
+= 6,5 Mio $

+ memory mapped, indexes, queries, marketing
- durability, single instance design

67 GB

EC2 Node 66 GB

EC2 Node 66 GB

2 Days off

NoSQL Divergenz: CouchDB = NoNoSQL

NoSQL Konvergenz:

Liquid Data at the bottom (JSON)

besser suchbar

Easy to run,
develop, manage

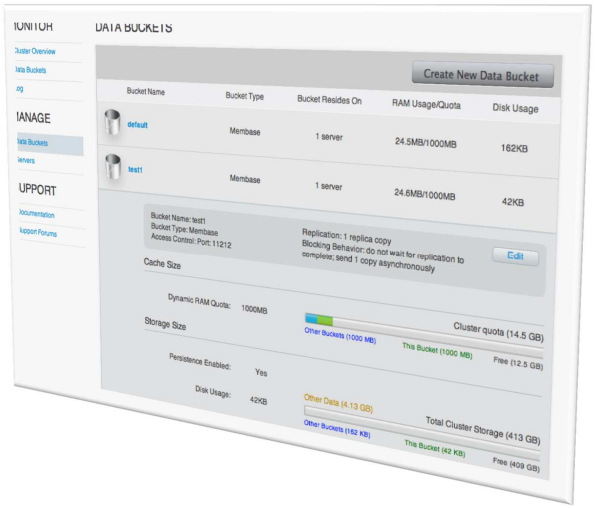
K/V-Stores

Data Structure Server ->

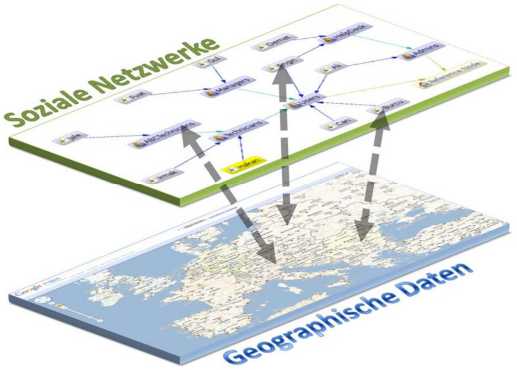
+ sehr schnell > 100.000 / sek
+ konfigurierbarer Disc sync
+ API für eigene Anbindung
+ einfache Replikation
+ hash, list, set, sorted set, messages
+ Installation
  UNIX: 38 sek
  Windows: 18 sek
- noch nicht skalierbar (2.*)

MEMBASE.ORG

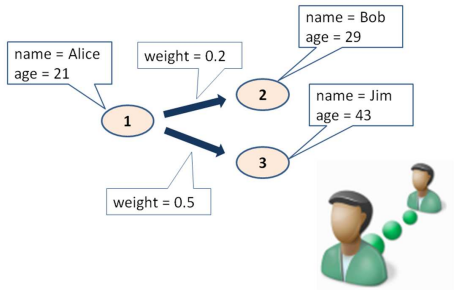
memcached API



GraphDBs



Property Graph



Neo4j, Sones, HyperGraphDB, InfiniteGraph, InfoGrid, Dex, VertexDB, Filament, OrientDB

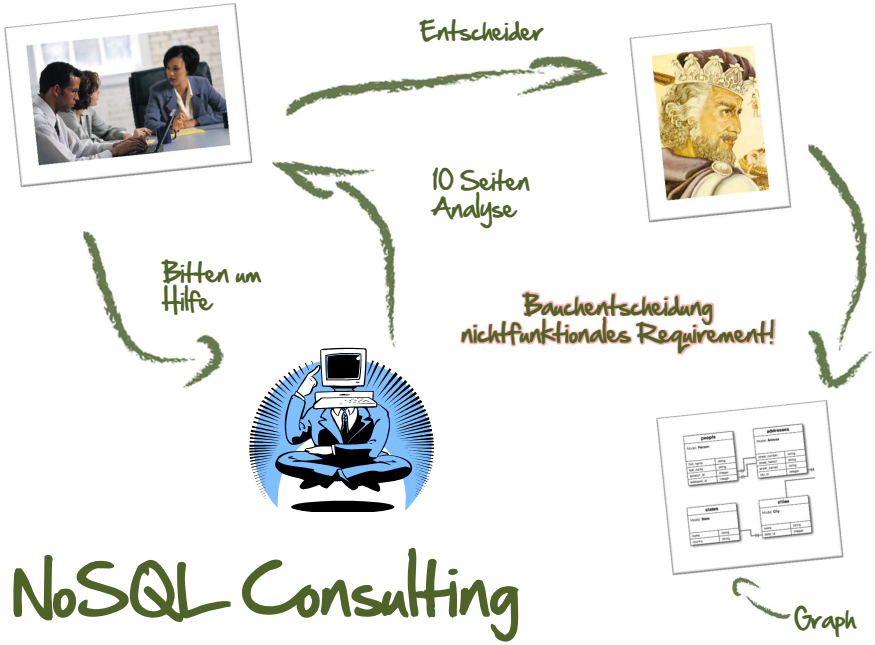
X

DataModel, Query Methods, Languages, License, Protocol

OrientDB

Key / ValueDB + DocumentDB + ObjectDB + GraphDB
+schneller…

> 220 DBs



NoSQL Consulting

Start-Ups ☺

etablierte Unternehmen ☹

1. Data
2. Transactions
3. Performance
4. Queries
5. Architecture
6. other Non-Functional Requirements

Analyse your Data
Domain-Data, Log-Data, Event-Data, Message-Data, critical Data, Business-Data, Meta-Data, temp Data, Session-Data, Geo Data, etc.

Data- / Storage-Model:
relational, column-o, doc-alike, graphs, objects, etc.

What Types / Type-System?
Data-Navigation, Data Amount, Data Komplexity (Deep XML?)

ACID vs. BASE vs. Mixture?
CAP decisions

Performance Dimension Analysis
Latency, Request behaviour, Throughput

Scale-Up vs Scale-Out

Query Requirements
Typical queries, Tools, Ad-Hoc Queries, SQL / LINQ needed, Map/Reduce? ...

Distribution Architecture
local, parallel, distributed / grid, service, cloud, mobile, p2p, ...

Data Access Patterns
read / write distribution, random / sequential, Access Design Patterns

Non Functional Requirements:
Replication, Refactoring Frequency, DB-Support, Qualification / simplicity, Company restrictions, DB diversity (allowed?), Security, Safety / Backup & Restore, Crash Resistance, Licence...

NoSQL

Lessons learned

Think outside the MyOracleSql Box ✓

VoltDB
Vertica
GenieDB
MonetDB
Hadoop++
...

RAM + SSD rocks

RethinkDB


Gustavo Alonso

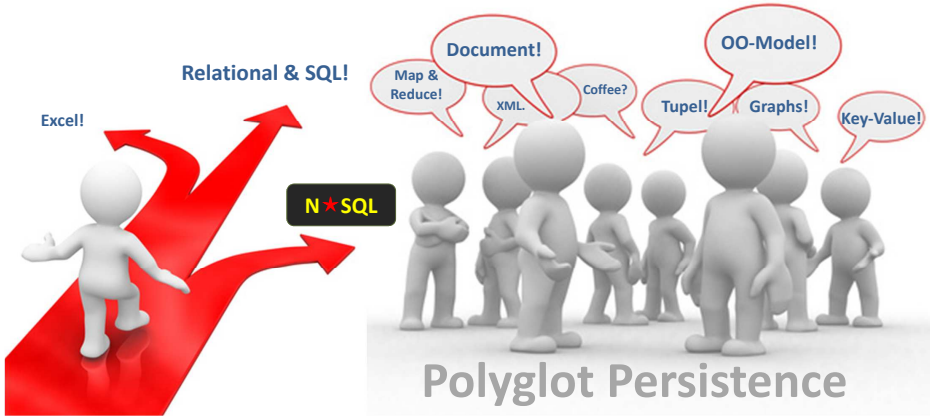
Lot's of >1 PT RAM DBs in CA!

Queries in FPGA !

Tonnen cleverer hybrid Lösungen da!

DaaS => best Mix!

The world is *diverse*! Act accordingly!



Thanks for listening!

http://edlich.de

Diskussion!