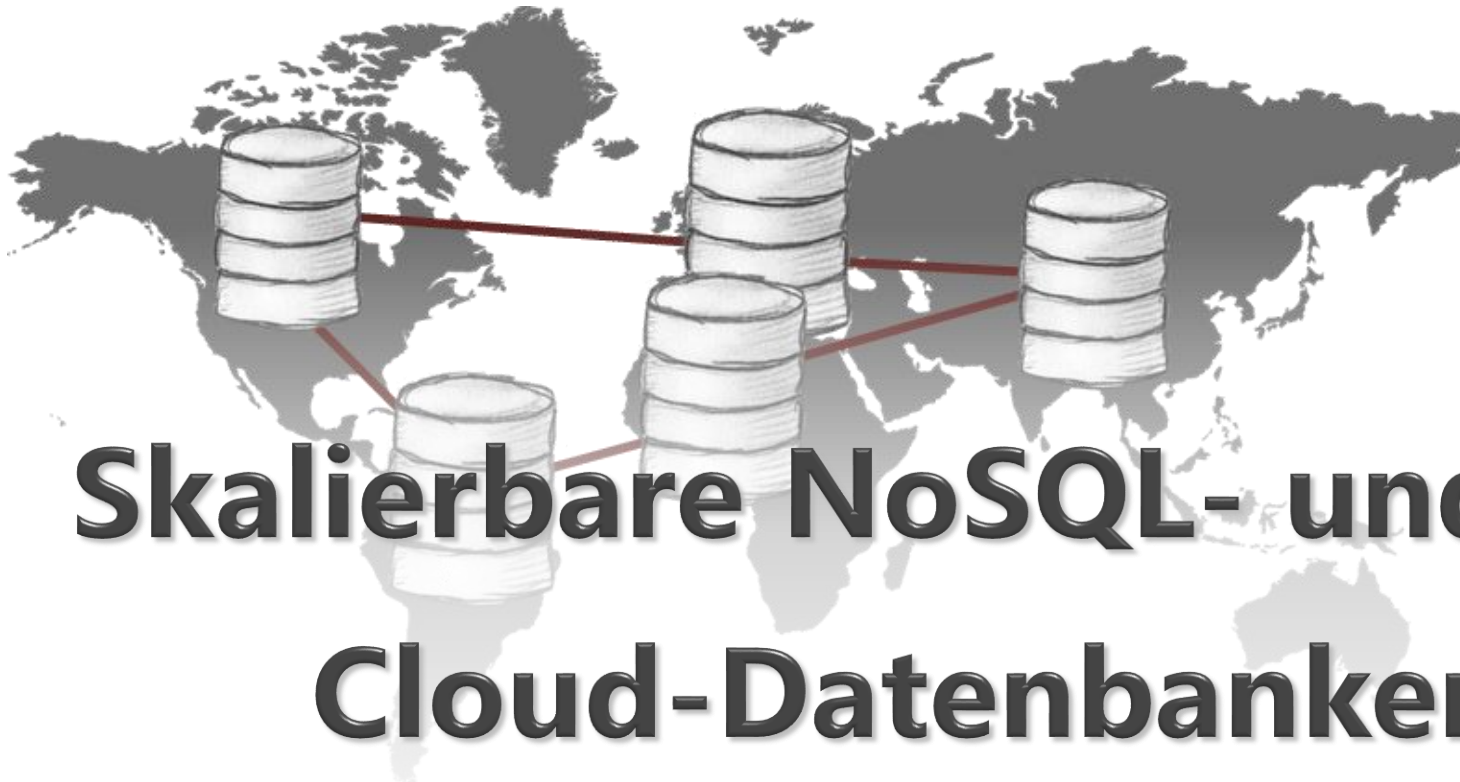


Skalierbare NoSQL- und Cloud-Datenbanken

in Forschung und Praxis

Felix Gessert

Outline

- Foundations: Big Data, Scalability, Availability
- The 4 Classes of NoSQL Databases
- NoSQL Examples: concrete Architectures, Systems, APIs
- Cloud Databases

- Literature
- Motivation
  - Big Data
  - NoSQL
- CAP Theorem
- Impossibility Results
- NoSQL Triangle
- ACID vs BASE

About me

- PhD student (database group university of hamburg)

Research Project for PhD

Orestes

Baqend

Build faster Apps faster.

Cloud Database Startup

The Database Explosion

Sweetspots

RDBMS

General-purpose ACID transactions

Parallel DWH

Aggregations/OLAP for massive data amounts

NewSQL

High throughput relational OLTP

Wide-Column Store

Long scans over structured data

Document Store

Deeply nested data models

Key-Value Store

Large-scale session storage

Graph Database

Graph algorithms & queries

In-Memory KV-Store

Counting & statistics

Wide-Column Store

Massive user-generated content

The Database Explosion

Cloud-Database Sweetspots

Firebase

Realtime BaaS

Communication and collaboration

Amazon RDS

Managed RDBMS

General-purpose ACID transactions

Amazon ElastiCache

Managed Cache

Caching and transient storage

Azure Tables

Wide-Column Store

Very large tables

Amazon DynamoDB

Wide-Column Store

Massive user-generated content

Parse

Backend-as-a-Service

Small Websites and Apps

bonsai

Managed NoSQL

Full-Text Search

Google Cloud Storage

Object Store

Massive File Storage

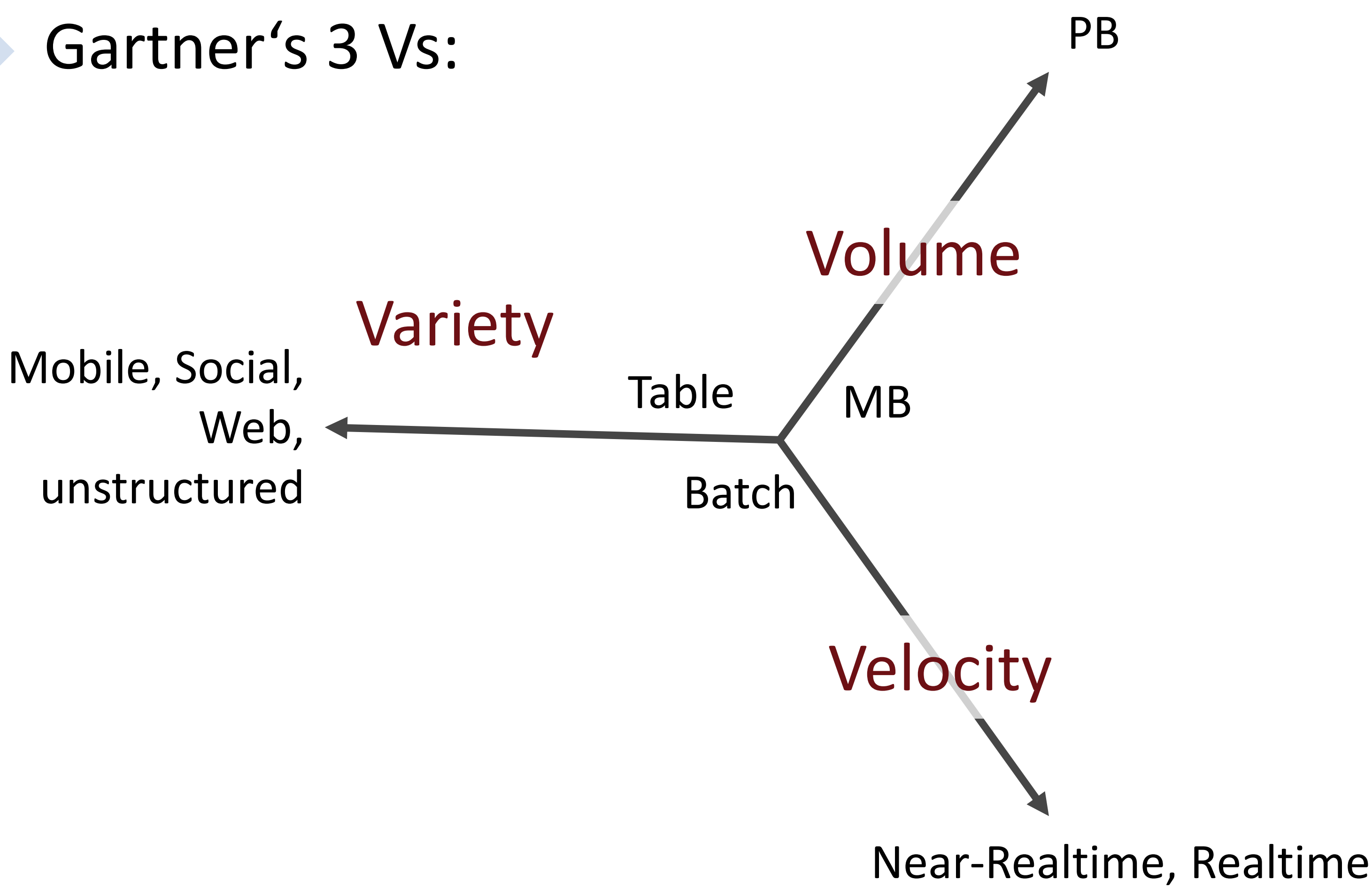
Amazon Elastic MapReduce

Hadoop-as-a-Service

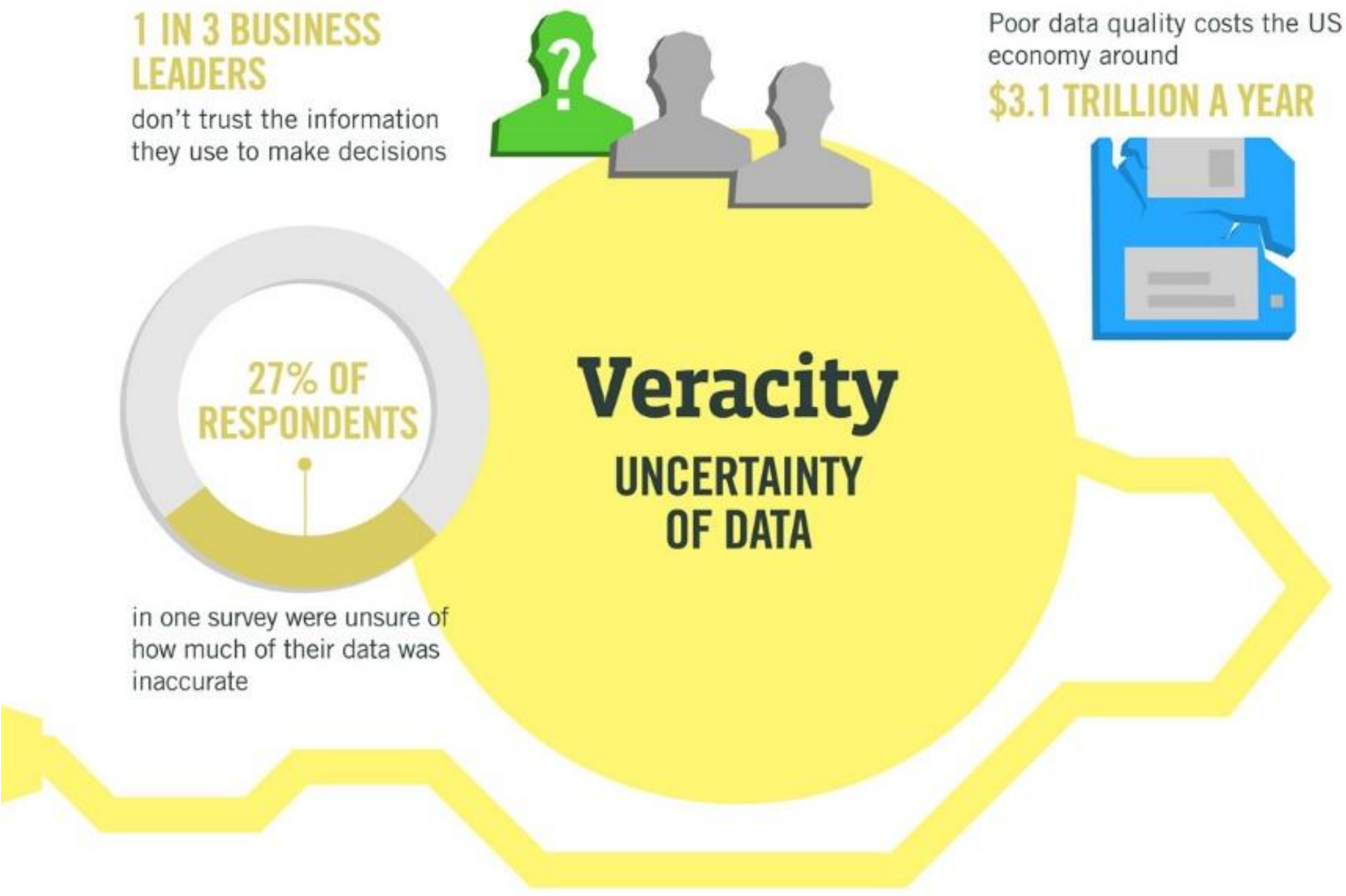
Big Data Analytics

Motivation Big Data

- Gartner's 3 Vs:



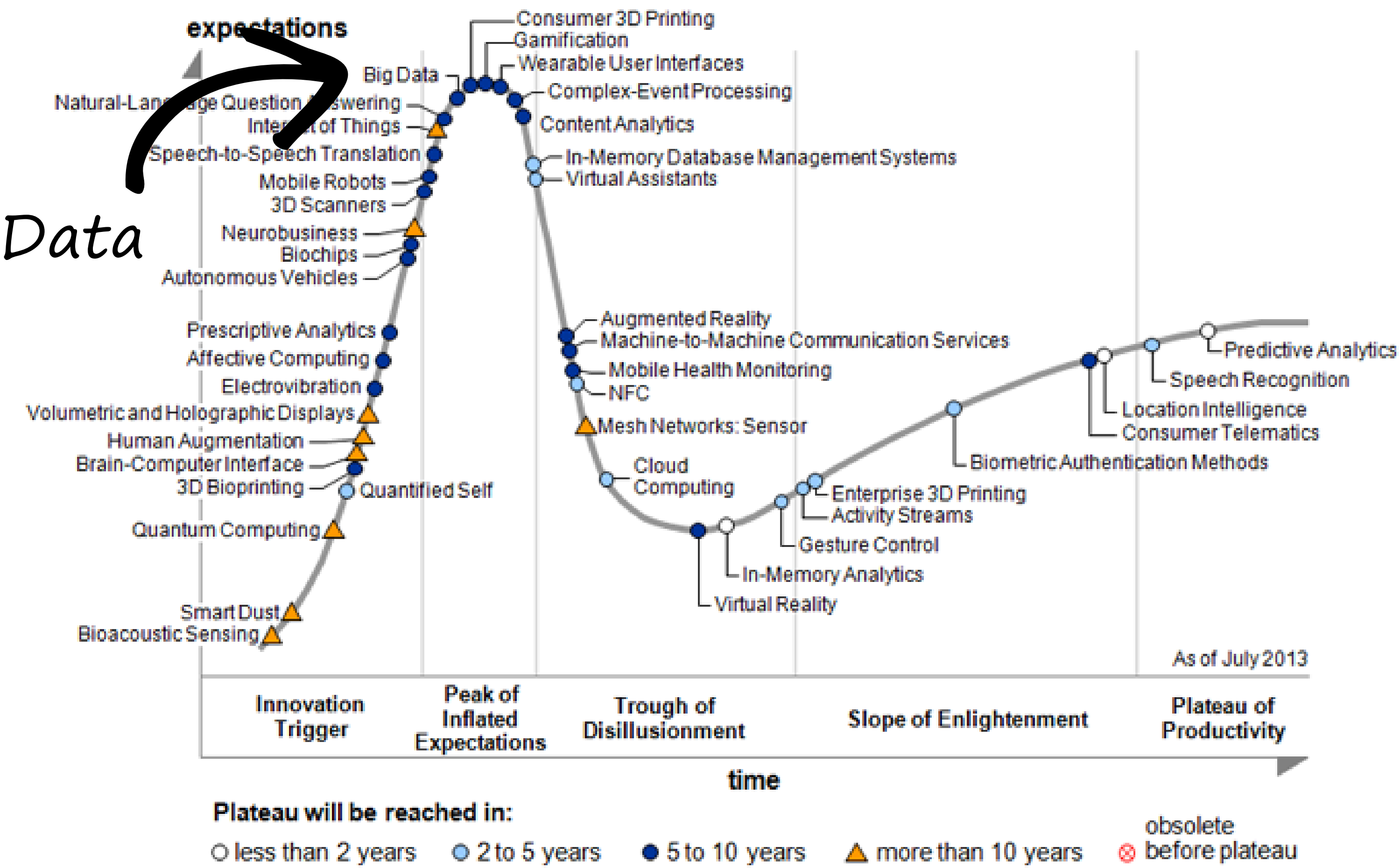
4Vs



IBM Infographic (McKinsey, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS)

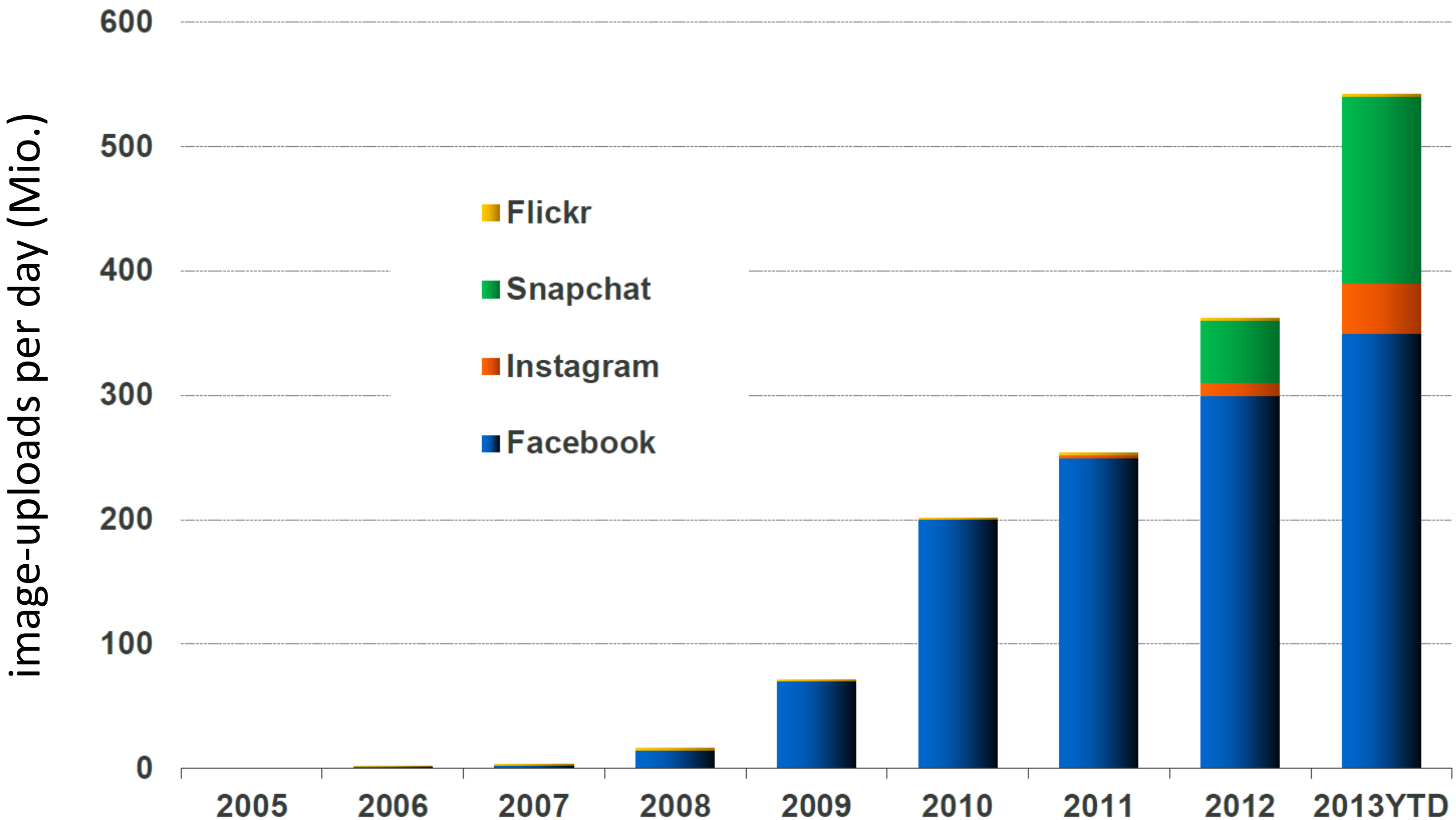
Hype Cycle

Big Data



http://www.gartner.com/newsroom/id/2575515

User-Generated Content: Images



www.kpcb.com/insights/2013-internet-trends

Data flood

Climate Research: The *Deutsche Klimarechenzentrum (DKRZ)* stores **60 PB** climate data.

Archiving: The Internet Archive stores **10 PB** of archived websites.

Gaming: World of Warcraft needs **1.3 PB** for storing the game state. Steam delivers **over 30 PB** of data per month.

Movies: The CGI-effects in Avatar (2009) needed **over 1 PB** storage for rendering.

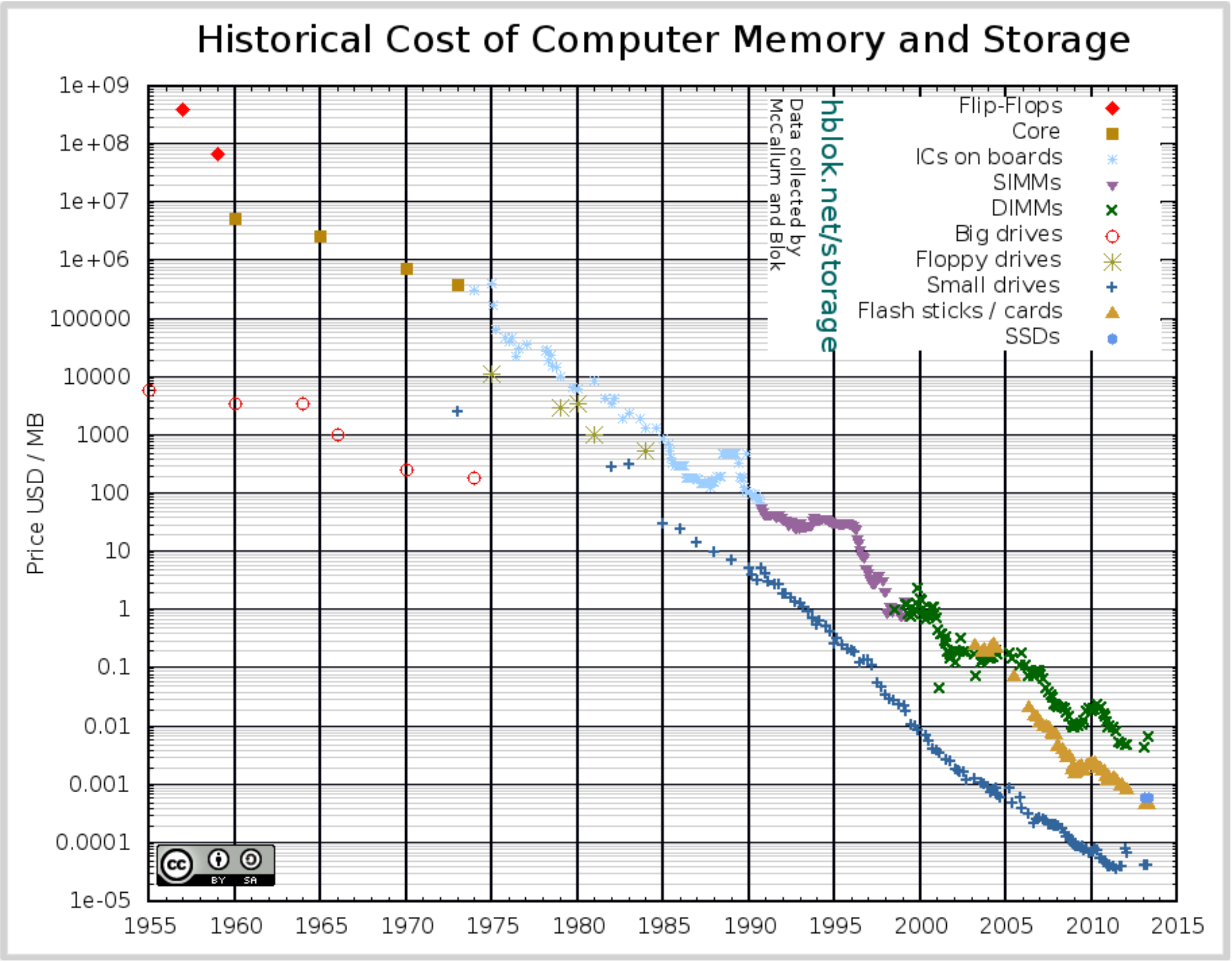
Supercomputing: The Blue Waters Supercomputer is planned to have a storage capacity of **500 PB**.

Particle Physics: In search of the Higgs-Boson CERN gathered **200 PB** of data.

Email: In May 2013 Microsoft announced that for the migration of Hotmail to oulook.com **over 150 PB** user data were transferred.

Wikipedia

Dropping storage costs



HBlock.net

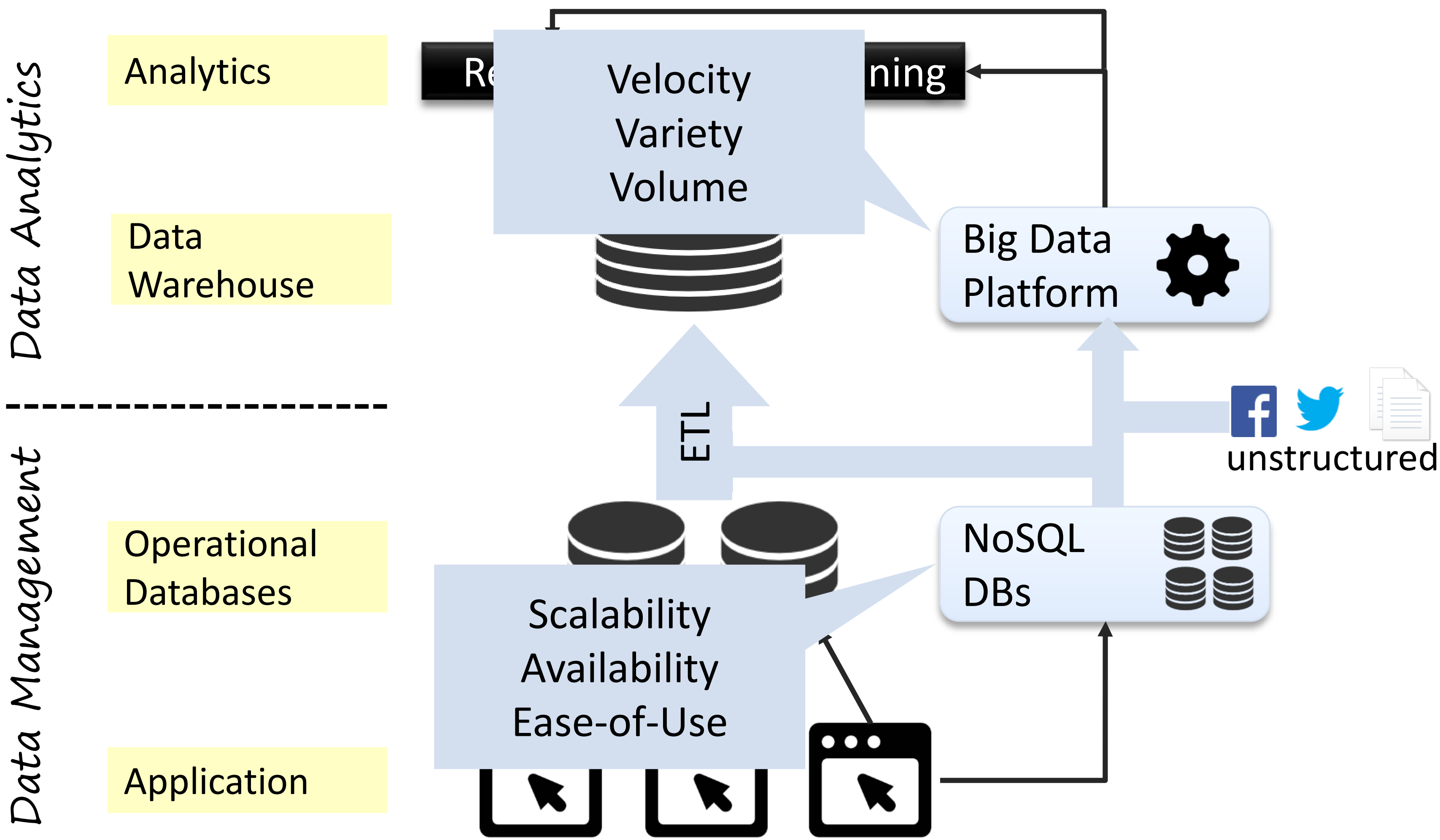
Big Data Defined

- Big Data has two sides:

Big Data

Big Data Management	Big Data Analytics
• **OLTP** • Often referred to as "NoSQL" • e.g. MongoDB, HBase, Cassandra	• **OLAP** • Often referred to as „Big Data“ • e.g. Hadoop, Storm

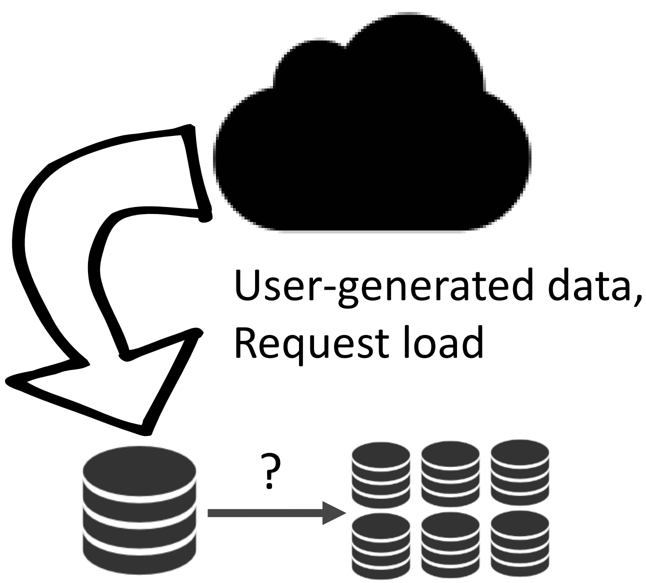
Big Data Architecture



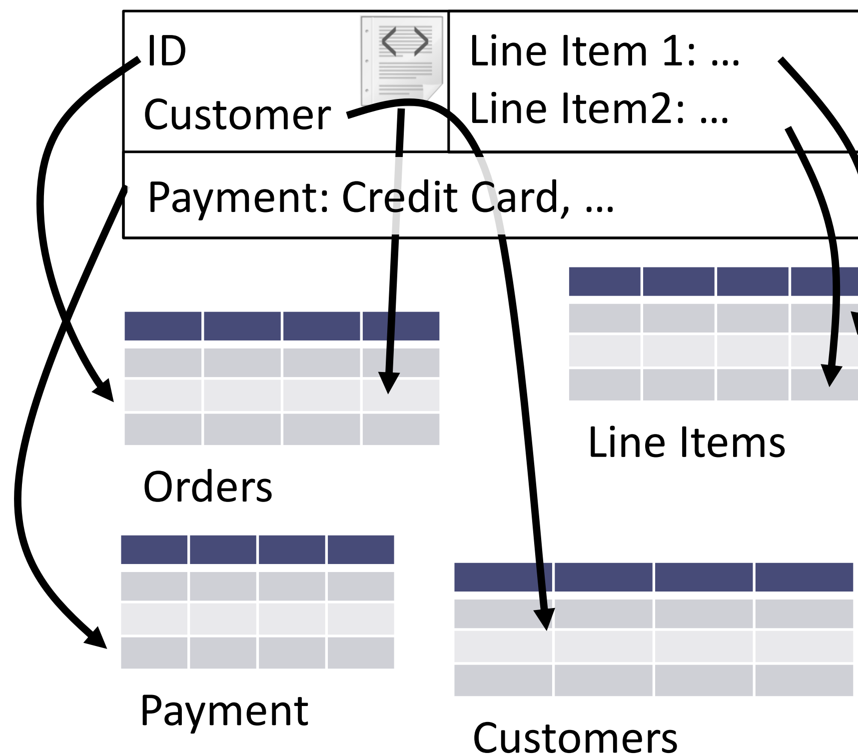
NoSQL Databases

- Two main motivations:

Scalability



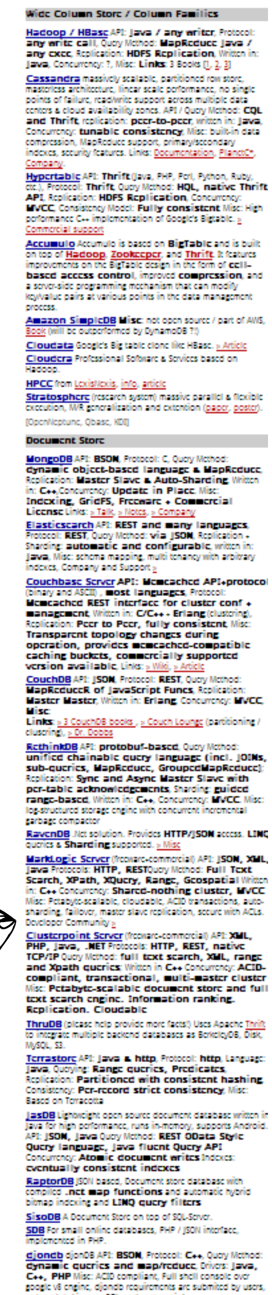
Impedance Mismatch



NoSQL Databases

- „NoSQL“ term coined in 2009
- Interpretation: „**N**ot **O**nly **SQL**“
- Typical properties:
  - Non-relational
  - Open-Source
  - Schema-less (*schema-free*)
  - Optimized for distribution (clusters)
  - Tunable consistency

***NoSQL-Databases.org*:**
Current list has over 150 NoSQL systems



Big Data Analytics

- **Idea**: make existing massive, unstructured data amounts usable

sources

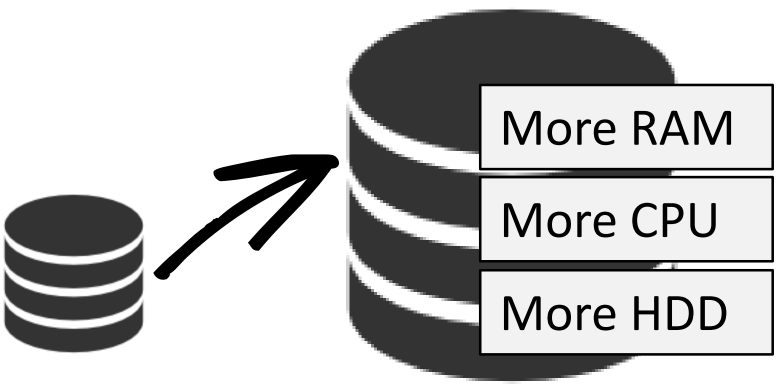
- Structured data (DBs)
- Log files
- Documents, Texts, Tables
- Images, Videos
- Sensor data
- Social Media, Data Services

Analyst, Data Scientist, Software Developer

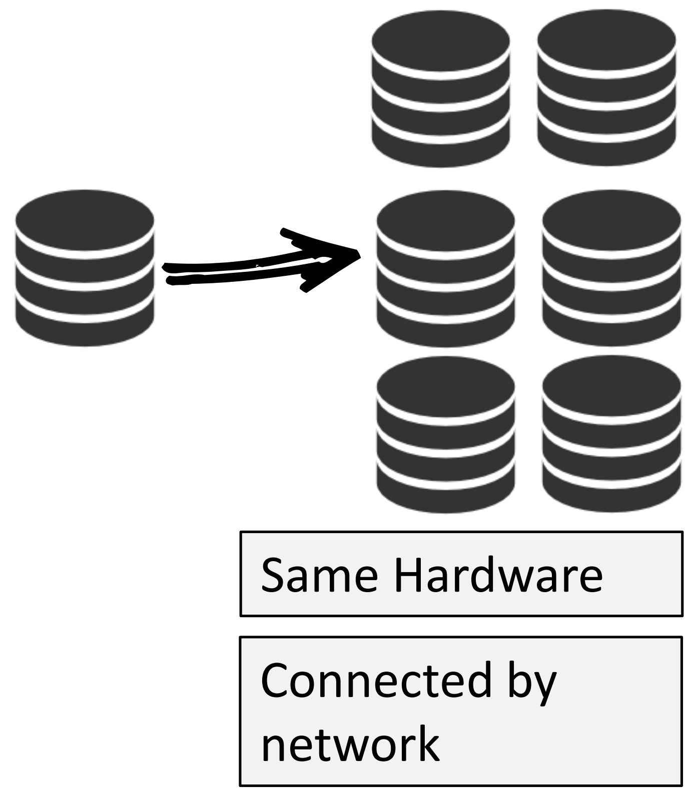
- Statistics, Cubes, Reports
- Recommender
- Classificators, Clustering
- Knowledge

Scale-up vs Scale-out

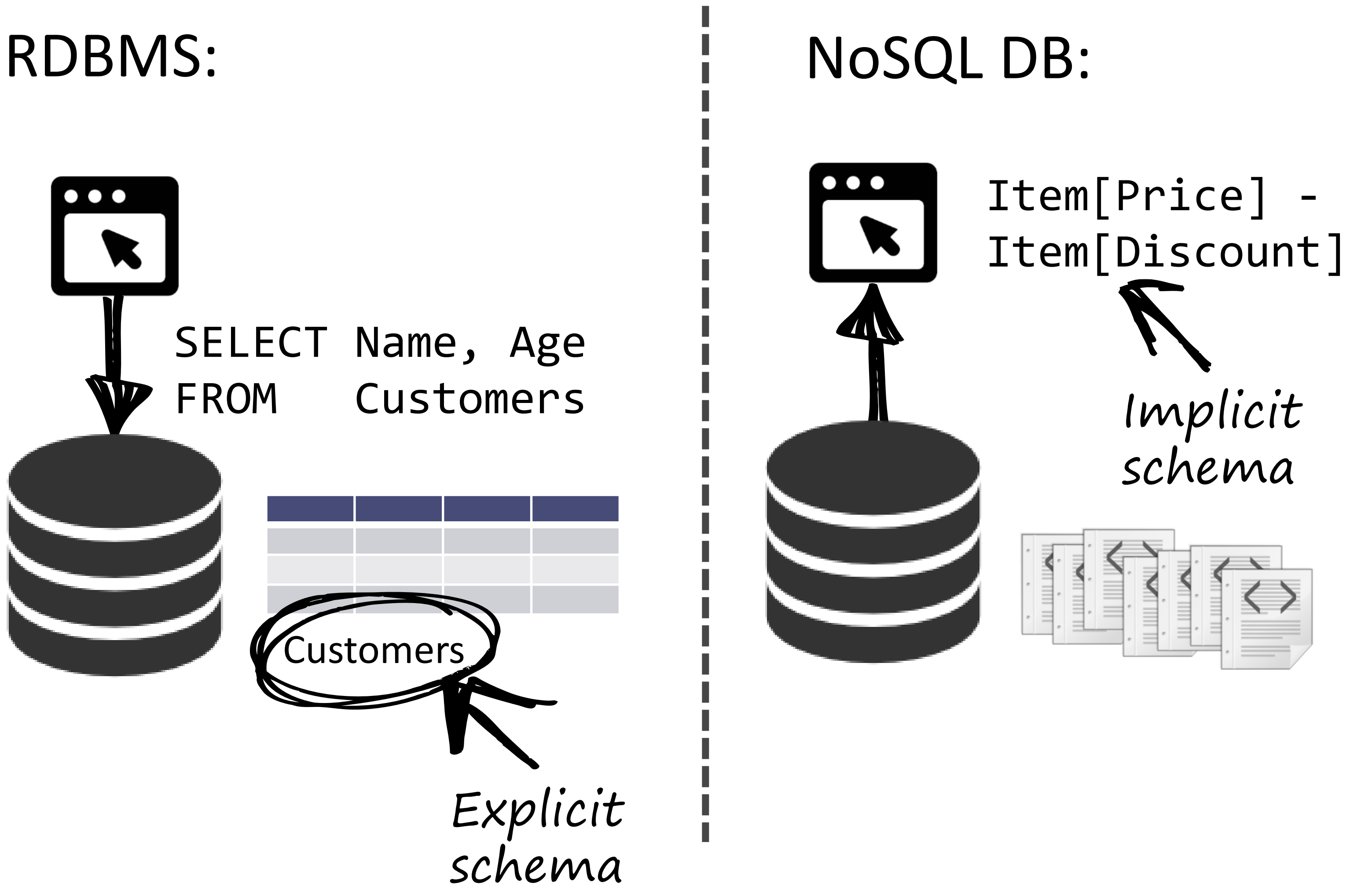
Scale-Up (*vertical* scaling):



Scale-Out (*horizontal* scaling):



Schema-free Data Modeling

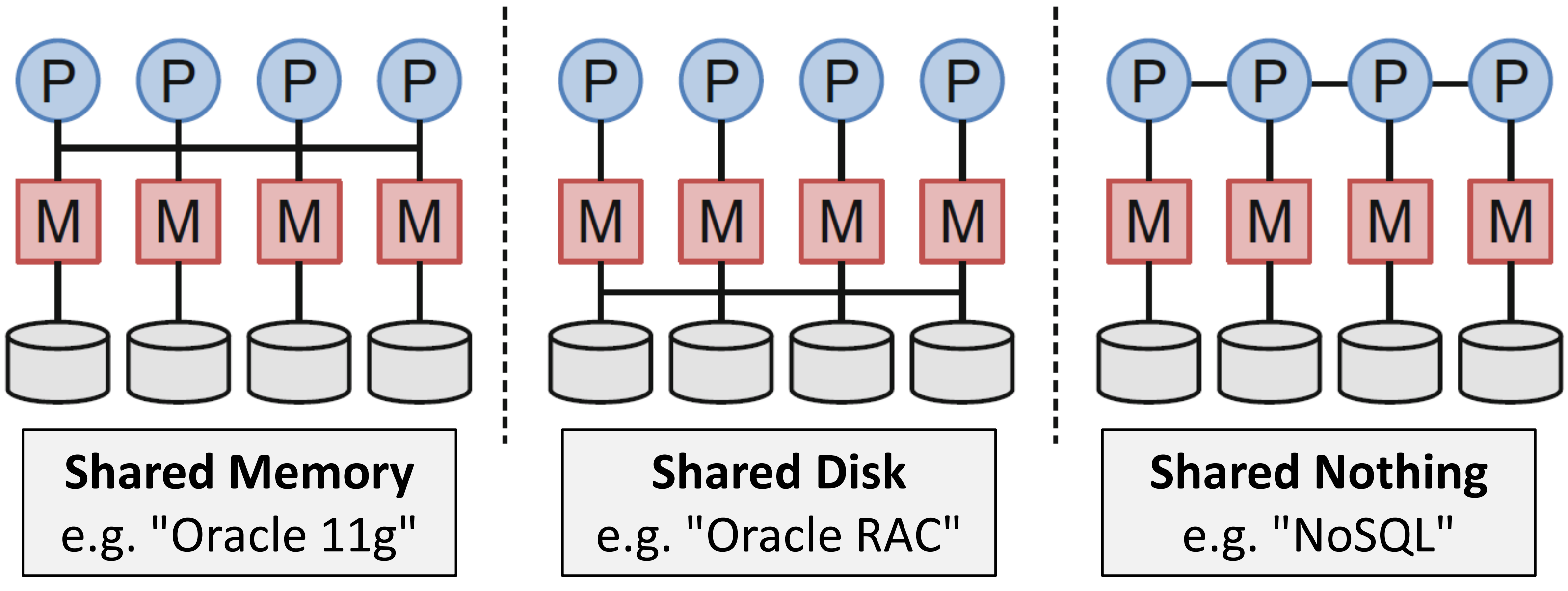


Paradigm Shift

Commercial DBMS	Open-Source DBMS
Specialized DB hardware (Oracle Exadata, etc.)	Commodity hardware
Highly available network (Infiniband, Fabric Path, etc.)	Commodity network (Ethernet, etc.)
Highly Available Storage (SAN, RAID, etc.)	Commodity drives (standard HDDs, JBOD)

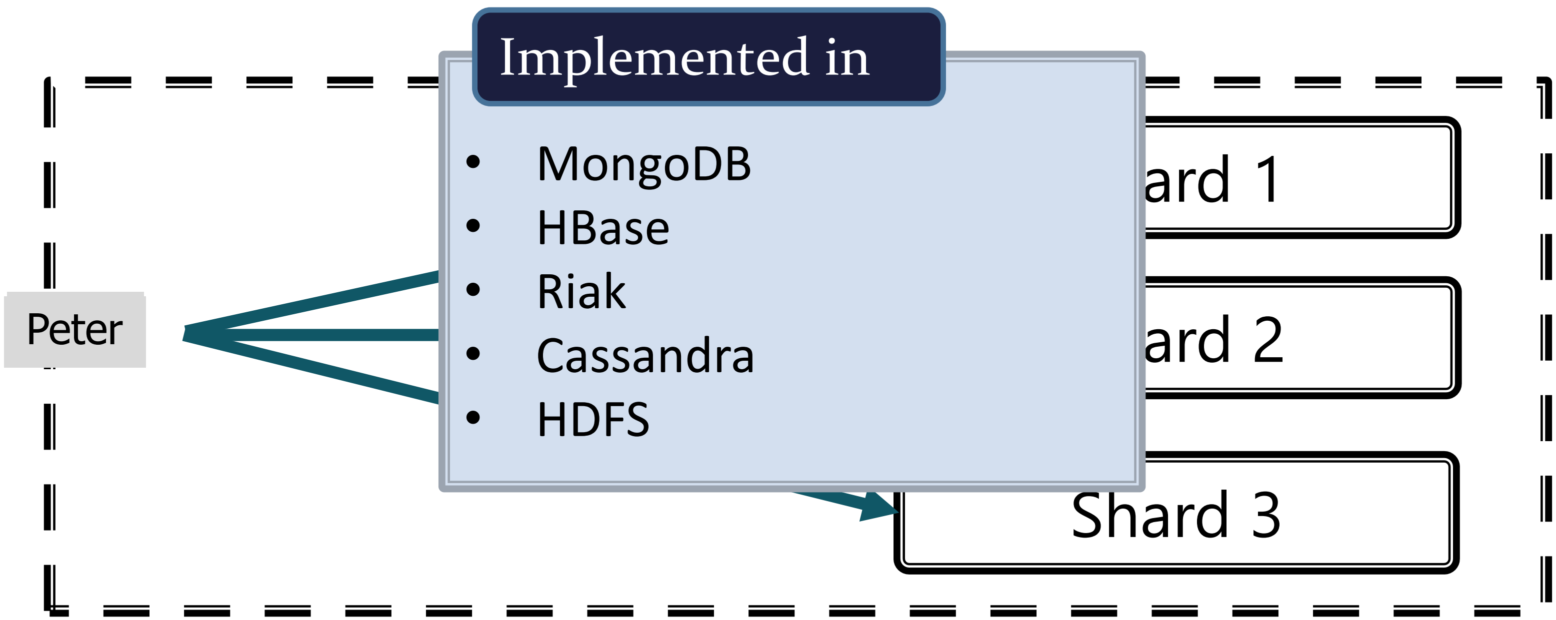
Paradigm Shift

- Shift towards distributed computing architectures



Sharding (aka Partitioning, Fragmentation)

- Horizontal distribution of data over server nodes
- **Paritioning strategies**: Hash-based vs. Range-based
- **Difficulty**: Multi-Shard-Operations (join, aggregation)



Sharding

Hash-based Sharding

- Builds hash of data values (e.g. over the key) to determine a partition (shard) for a data item (tuple)
- **Pro**: Perfectly even distribution
- **Contra**: No data locality – data items are pseudorandomly scattered over partitions

Range-based Sharding

- Assigns ranges defined over fields (shard keys) to partitions
- **Pro**: Data locality preserved (for shard keys)
- **Contra**: distribution might grow uneven → repartitioning/balancing required

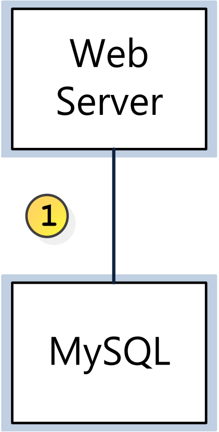
Traditional Sharding

Example: **Tumblr**

- Caching
- Sharding from application

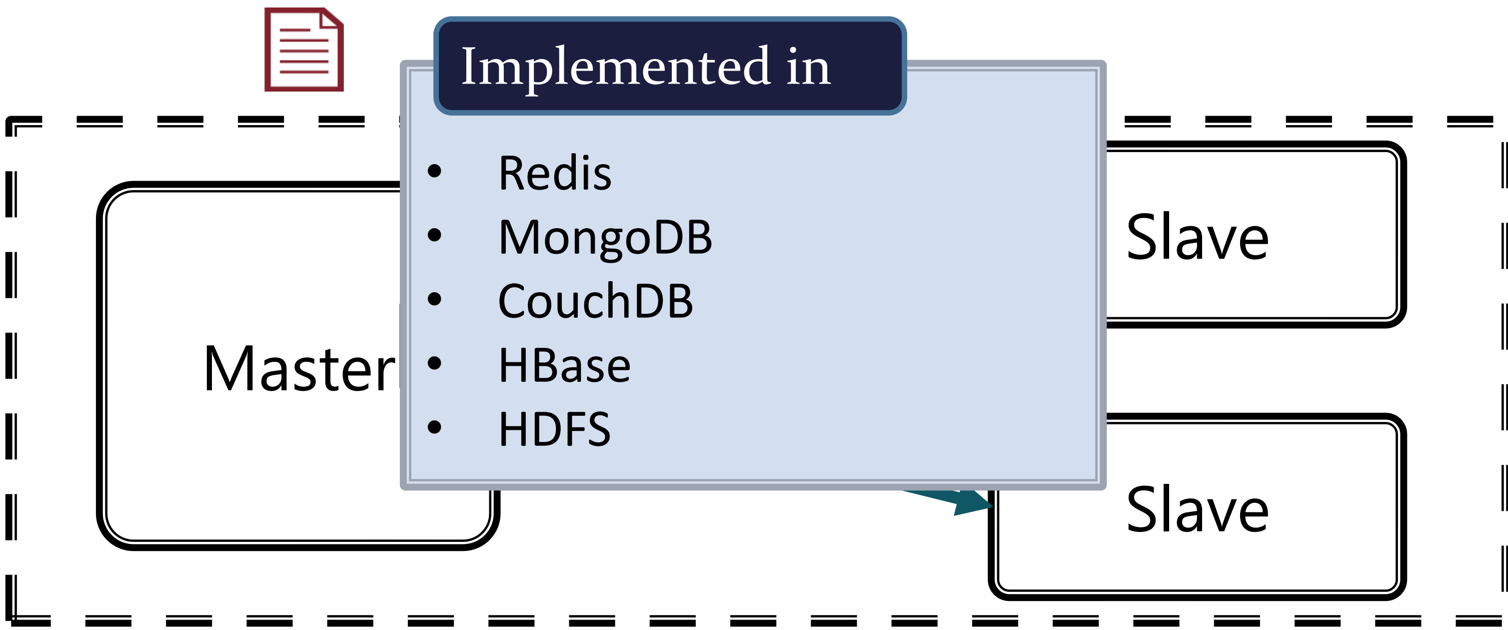
Moved towards:

- Redis
- HBase



Replication

- Stores *N* copies of each data item
- **Consistency model**: synchronous vs asynchronous
- **Coordination**: Multi-Master, Master-Slave



Replication: consistency models

Asynchronous

- Writes are acknowledged immdediately
- Performed through *log shipping* or *update propagation*
- **Pro**: Fast writes, no coordination needed
- **Contra**: Replica data potentially stale (*inconsistent*)

Synchronous

- The node accepting writes synchronously propagates updates/transactions before acknowledging
- **Pro**: Consistent
- **Contra**: needs a commit protocol (more roundtrips), unavaialable under certain network partitions

Replication: coordination

Master-Slave (*Primary Copy*)

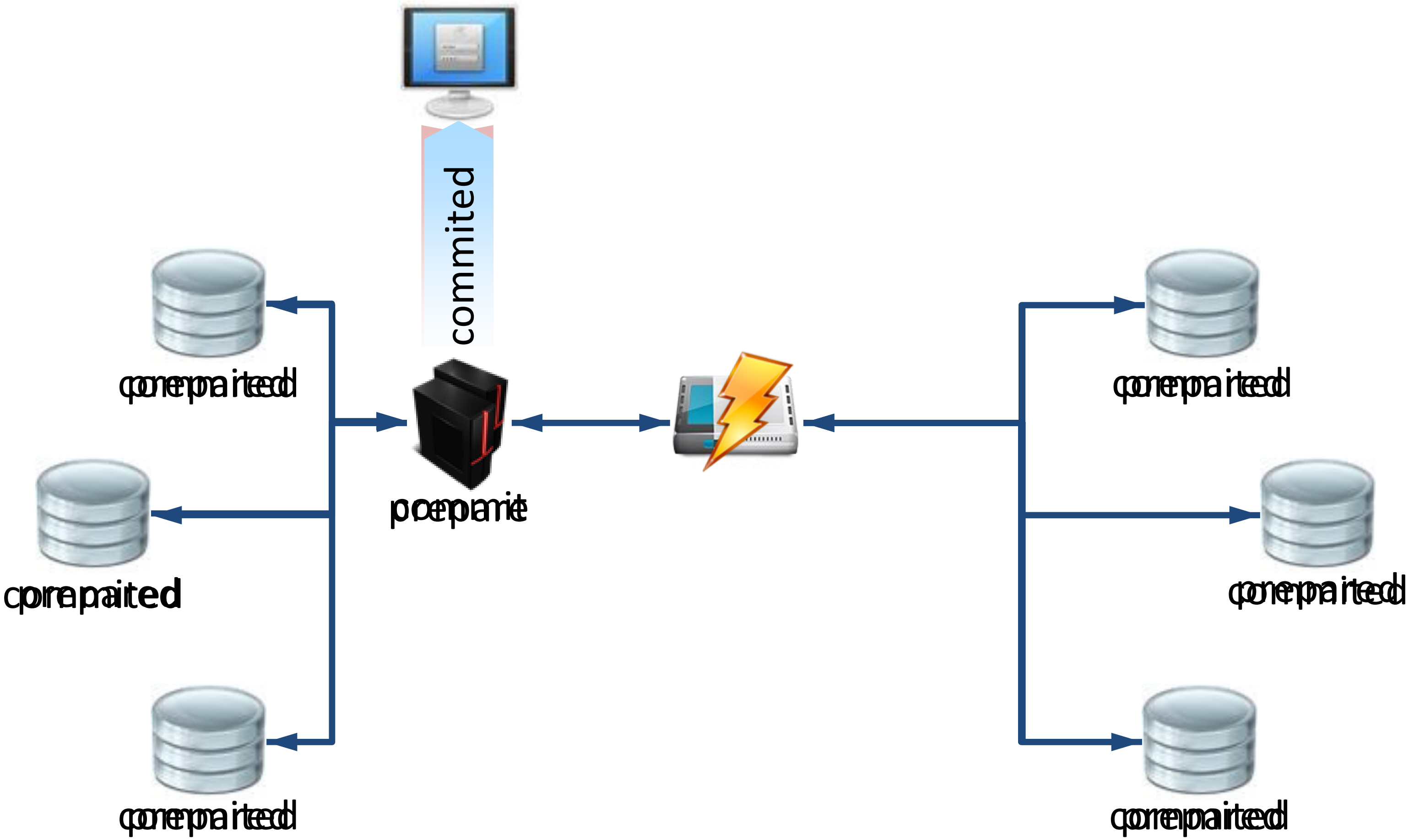
- Only a dedicated master is allowed to accept writes, slaves are read-replicas
- **Pro**: reads from the master are consistent
- **Contra**: master is a bottleneck and SPOF

Multi-Master (*Update anywhere*)

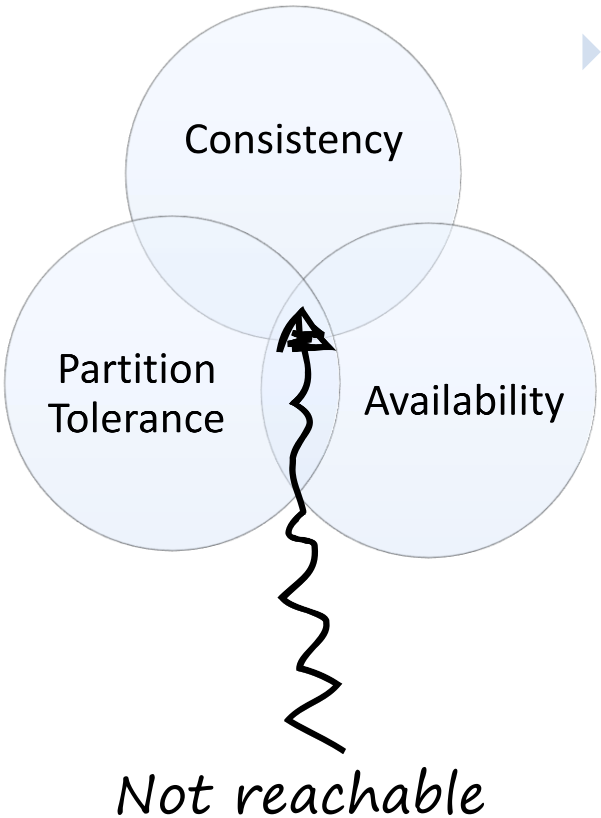
- The server node accepting the writes synchronously propagates the update or transaction before acknowledging
- **Pro**: fast and highly-available
- **Contra**: either needs complicated coordination protocols (e.g. Paxos) or is inconsistent

Synchronous Replication

Example: Two-Phase Commit is not partition-tolerant



CAP-Theorem



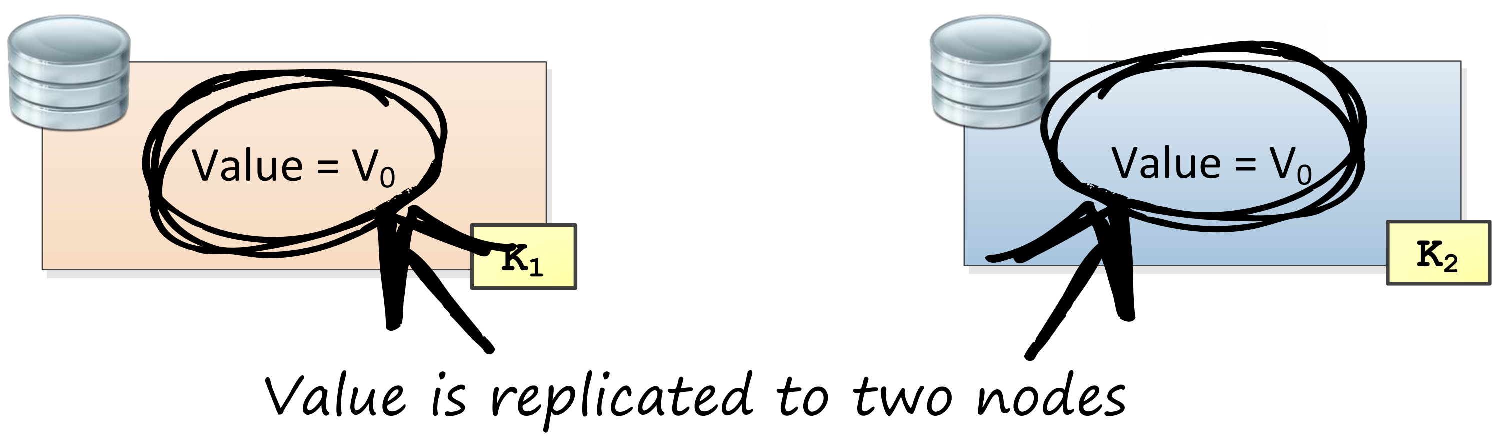
- Classifies distributed databases
- Only 2 out of 3 properties are achievable at a time:
  - **Consistency**: all clients have the same view on the data
  - **Availability**: every request to a non-failed node most result in correct response
  - **Partition tolerance**: the system has to continue working, even under arbitrary network partitions

Eric Brewer, ACM-PODC Keynote, Juli 2000

Gilbert, Lynch: Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services, SigAct News 2002

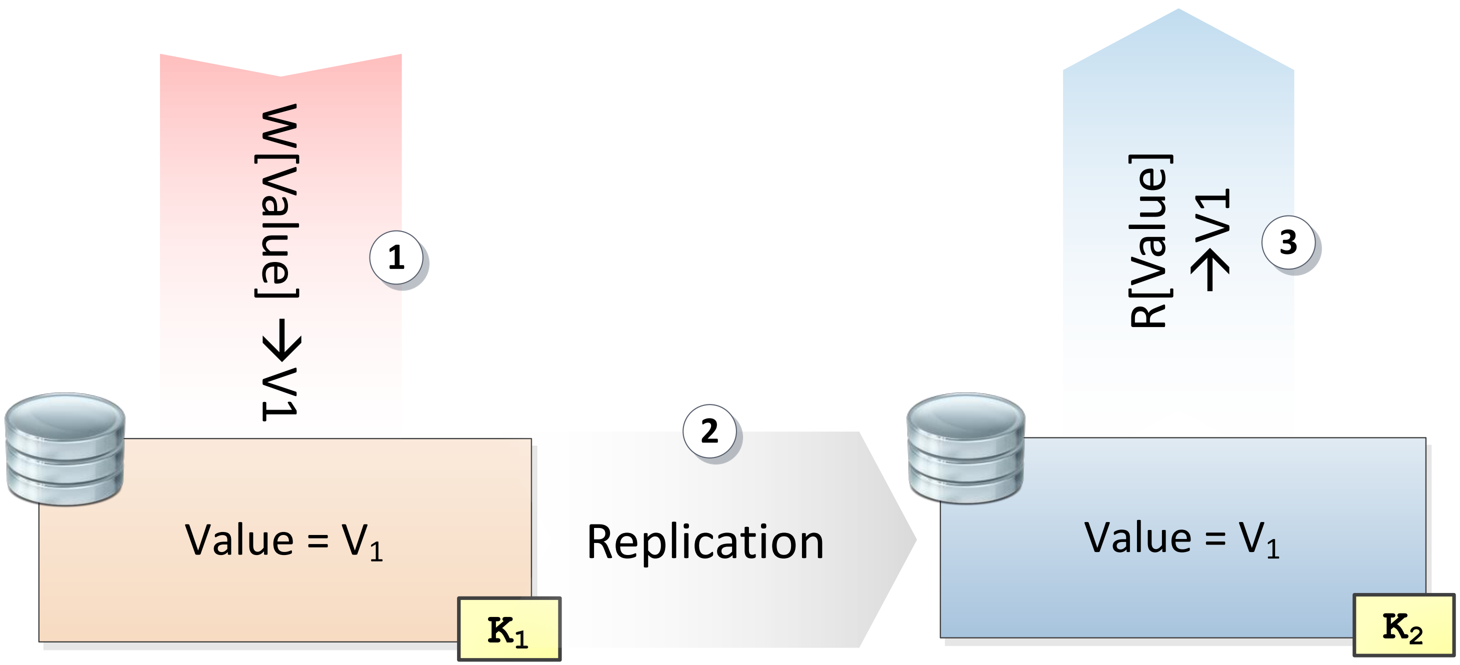
CAP-Theorem: simplified proof

- Intuition for the impossibility of simultaneously achieving **C**, **A** and **P** at once:



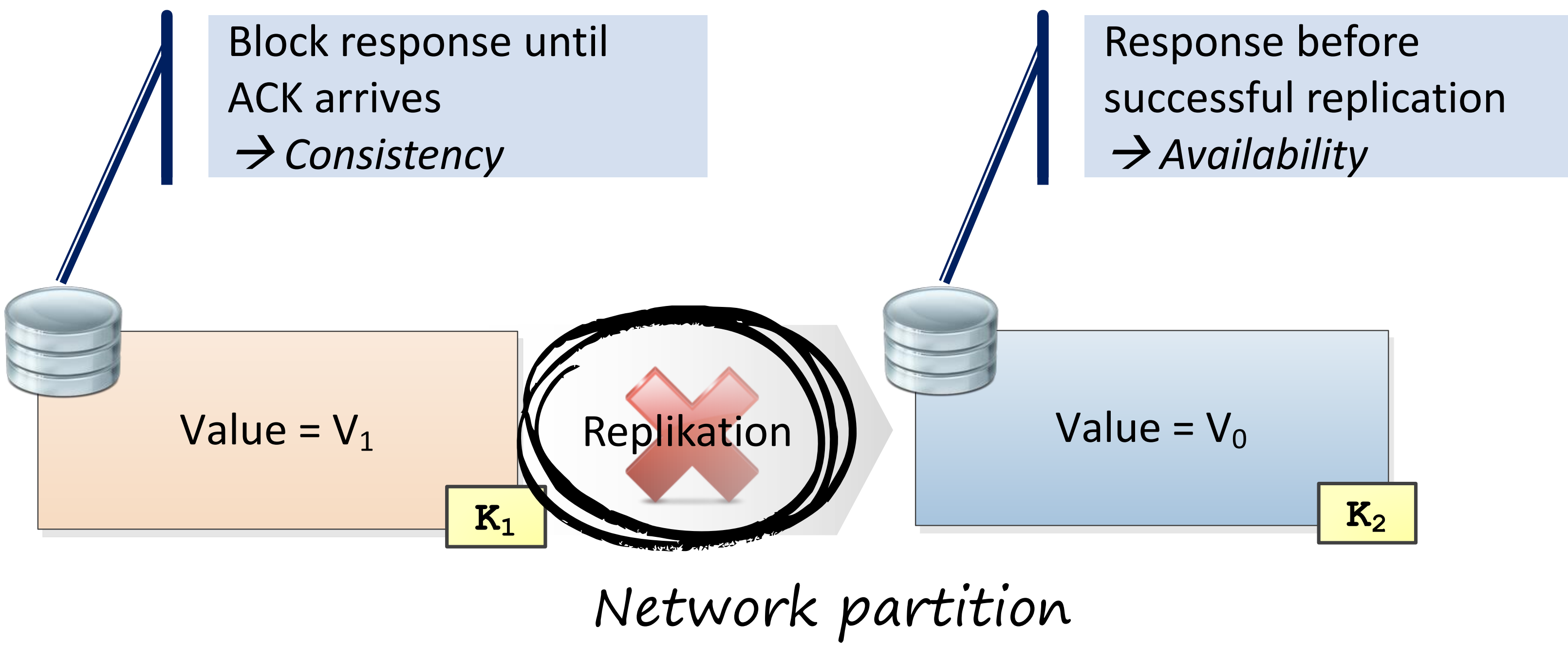
CAP-Theorem: simplified proof

- Failure-free reading and writing:

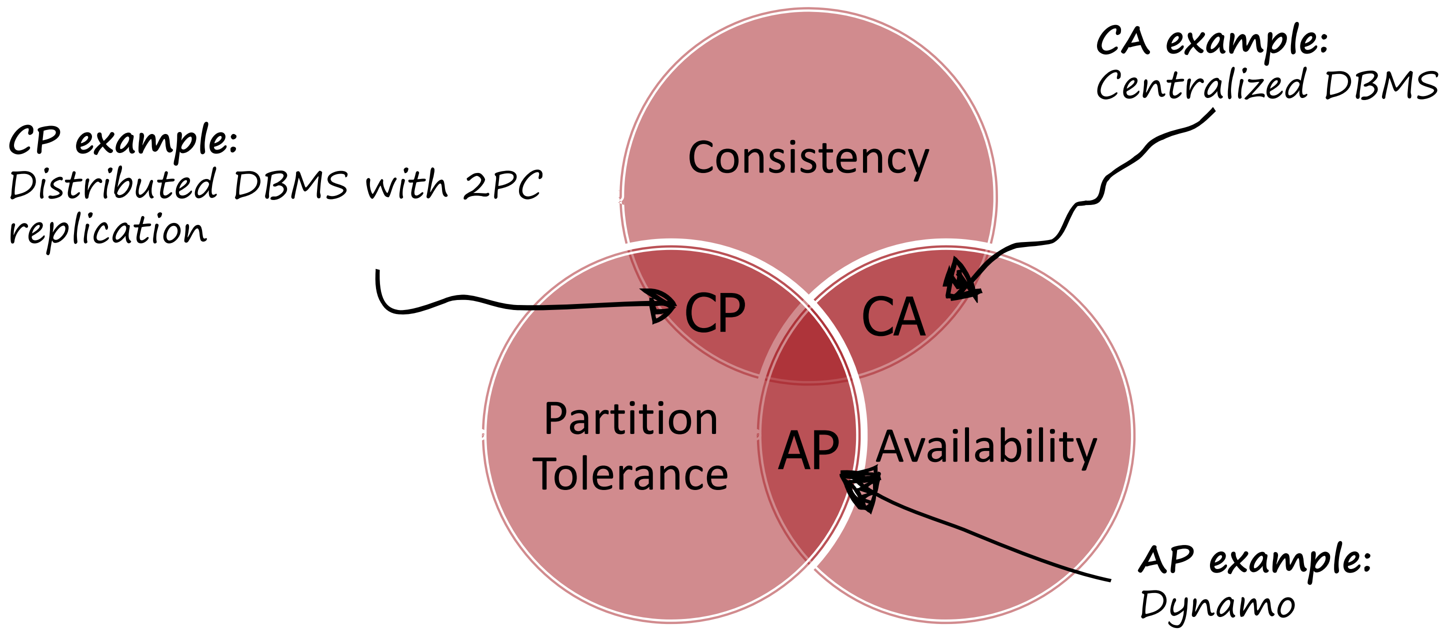


CAP-Theorem: simplified proof

- **Problem**: when a network partition occurs, either consistency or availability have to be given up



CAP-Theorem: Wrap-up

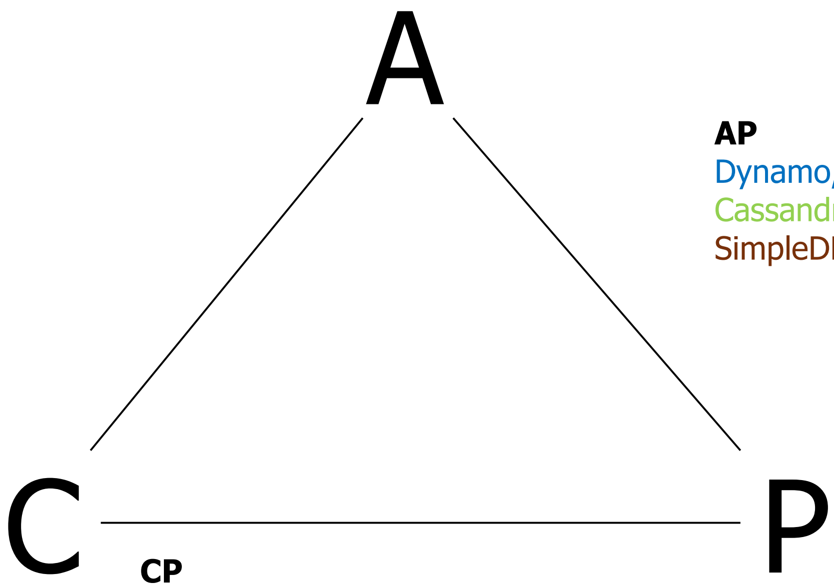


NoSQL triangle

Data models

Relational
Key-Value
Wide-Column
Document-Oriented

Every client can always read and write



CA
Oracle, MySQL, …

AP
Dynamo, Redis
Cassandra
SimpleDB, Riak

All clients share the same view on the data

CP
Postgres, MySQL Cluster, Oracle RAC
BigTable, HBase, Azure Table
MongoDB

All nodes continue working under network partitions

Nathan Hurst: Visual Guide to NoSQL Systems
http://blog.nahurst.com/visual-guide-to-nosql-systems

ACID vs BASE

„Gold standard“ for RDBMSs

ACID

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

BASE

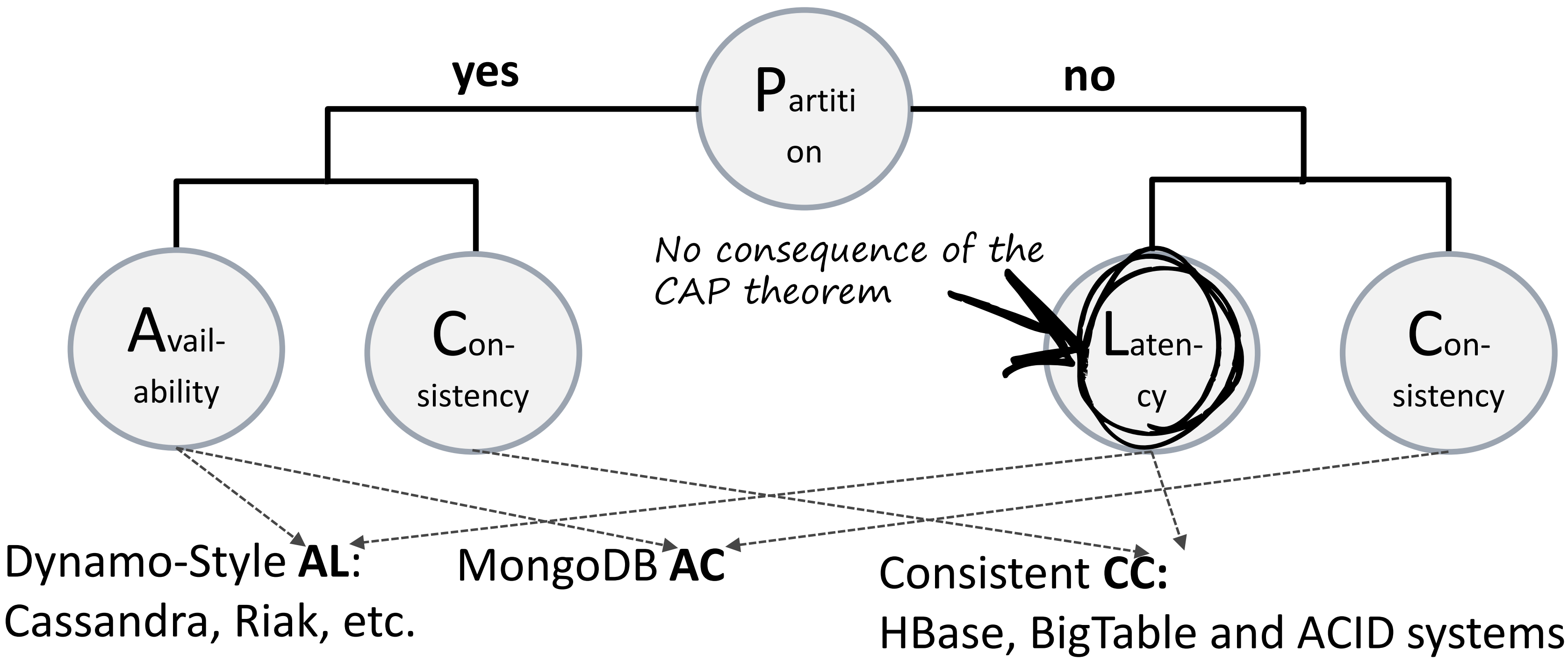
- **B**asically **A**vailable
- **S**oft State
- **E**ventually Consistent

Model of many NoSQL systems

http://queue.acm.org/detail.cfm?id=1394128

PACELC – an alternative CAP formulation

- **Idea**: Classify system according to their heaviour during network partitions



Abadi, Daniel. "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story."

Negative Results

In Distributed Computing

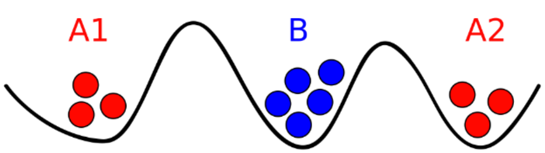
Asychronous Network, *Unreliable* Channel

Atomic Storage

Impossible:
CAP Theorem

Consensus

Impossible:
2 Generals Problem



Asychronous Network, *Reliable* Channel

Atomic Storage

Possible:
Attiya, Bar-Noy, Dolev (ABD) Algorithm

Consensus

Impossible:
Fisher **L**ynch **P**atterson (FLP) Theorem

Negative Results
Consensus Algorithms

- Consensus:
  - *Agreement*: No two processes can commit different decisions
  - *Validity (Non-triviality)*: If all initial values are same, nodes must commit that value
  - *Termination*: Nodes commit eventually
- No algorithm *guarantees* termination (FLP)
- Algorithms:
  - **Paxos** (e.g. Google Chubby, Spanner, Megastore, Cassandra Lightweight Transactions)
  - **Raft** (e.g. etcd service)
  - Zookeeper Atomic Broadcast (**ZAB**)

Negative Results
Correctness/Serializability

Distributed ACID and availability are incompatible:

Write A=1
Read B

Write B=1
Read A

$$w_1(a = 1)\, r_1(b = \perp) \qquad\qquad w_2(b = 1)\, r_2(a = \perp)$$

- Weaker isolations levels are possible:
  - RAMP Transactions (P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, und I. Stoica, „Scalable Atomic Visibility with RAMP Transactions“, SIGMOD 2014)
- **Consequence:** trade-offs are important

Typical Trade-offs

Read Performance	Write Performance
Latency	Durability
Synchronous replication	Asynchronous replication
Row-based	Column-based
Transactions	Availability
REST	RPC
Commodity servers	High-end hardware
Normalisation	Denormalisation
Schemas	Schemafreeness

Wrap-up: Foundations

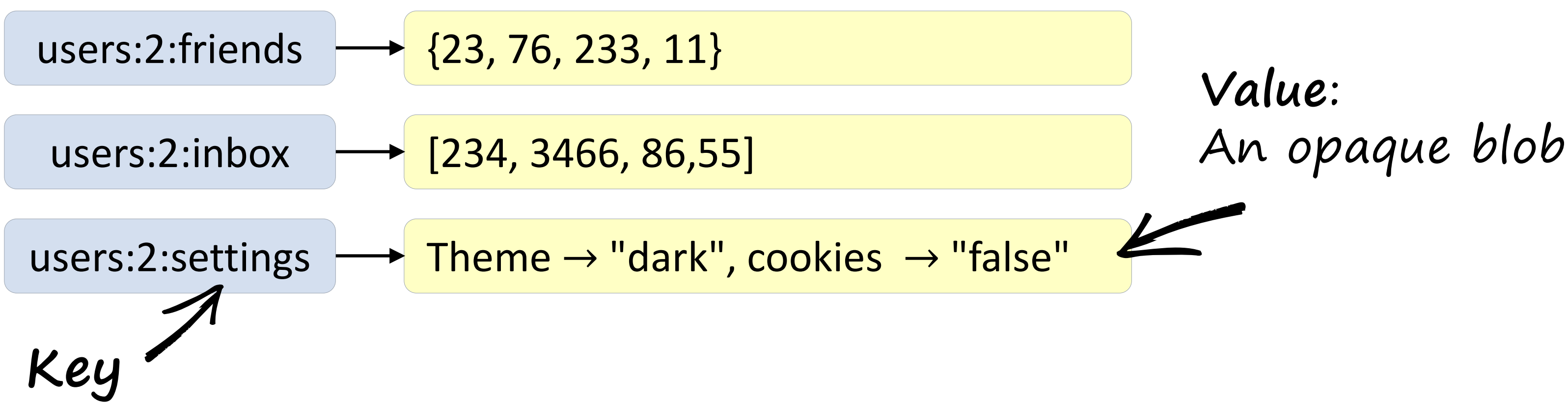
- High data volumes, unstructured sources and new kinds of applications triggered BigData and NoSQL technologies
- Shared Nothing architectures for **horizontal scalability**
  - **Replication** enables read scalability and fault tolerance
  - **Sharding** enables write scalability and data volume scalability
- **CAP Theorem**: Consistency, Availability and Partition Tolerance cannot be achieved at the same time
  - **BASE** (Basically available, soft-state, eventually consistent) paradigm as an alternative to ACID

Outline

- Foundations: Big Data, Scalability, Avaialbility
- The 4 Classes of NoSQL Databases
  - Key-Value stores
  - Wide-Column stores
  - Document stores
  - Graph databases
  - Other classes
- NoSQL Examples: concrete Architectures, Systems, APIs
- Cloud Databases

Key-Value Stores

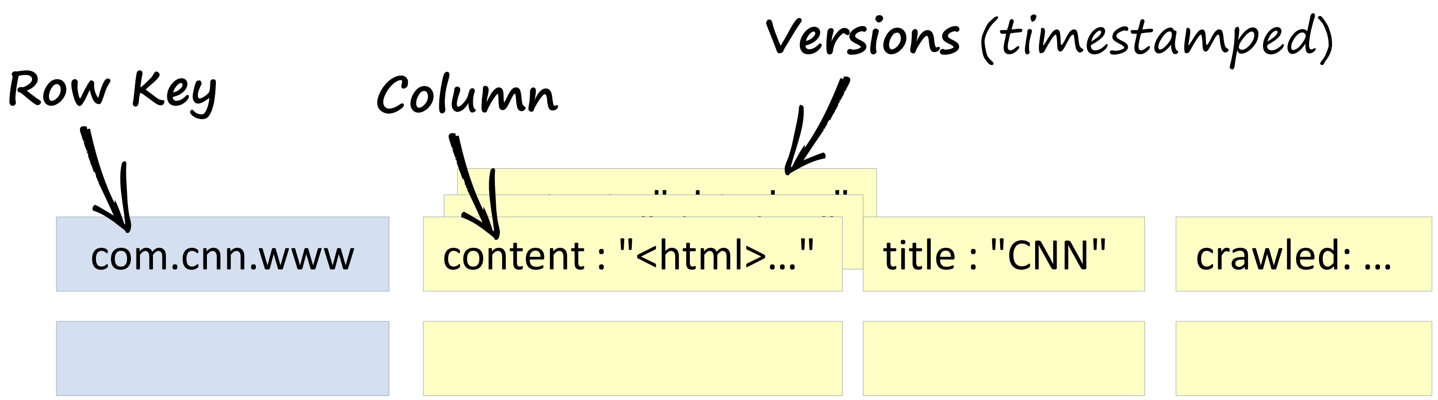
- **Data model:** (key) -> value
- **Interface**: CRUD (Create, Read, Update, Delete)



- Examples: Amazon Dynamo (AP), Riak (AP), Redis (CP)

Wide-Column Stores

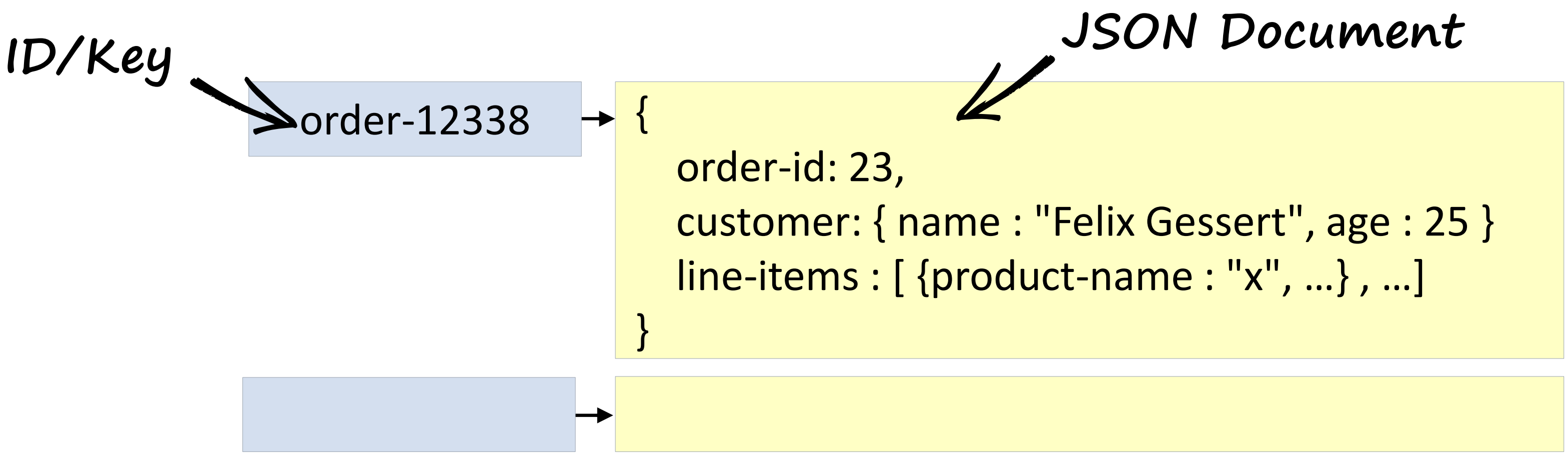
- **Data model:** (rowkey, column, timestamp) -> value
- **Interface**: CRUD, Scan



- Examples: Cassandra (AP), Google BigTable (CP), HBase (CP)

Document Stores

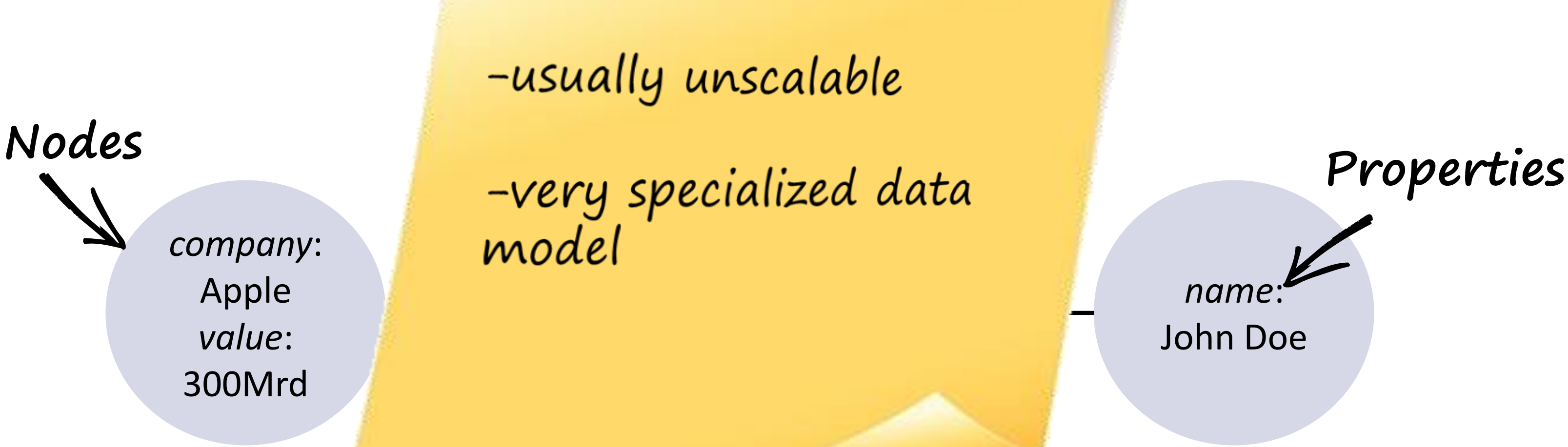
- **Data model:** (collection, key) -> document
- **Interface**: CRUD, Querys, Map-Reduce



- Examples: CouchDB (AP), Amazon SimpleDB (AP), MongoDB (CP)

Graph Databases

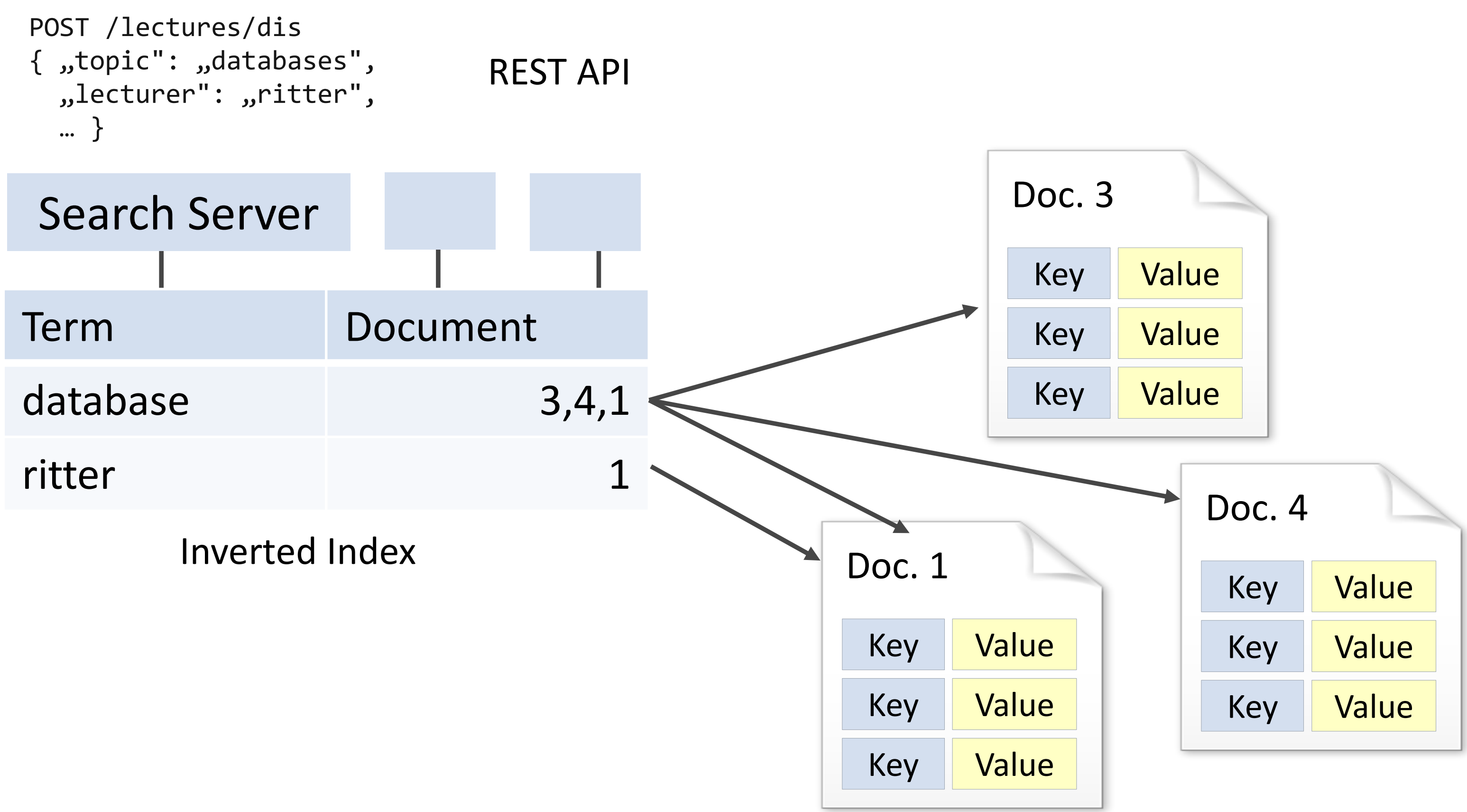
- **Data model:** G = (V, E): Graph-Property Modell
- **Interface**: Trave ... , transactions



- Examples: Neo4j (CA), InfiniteGraph (CA)

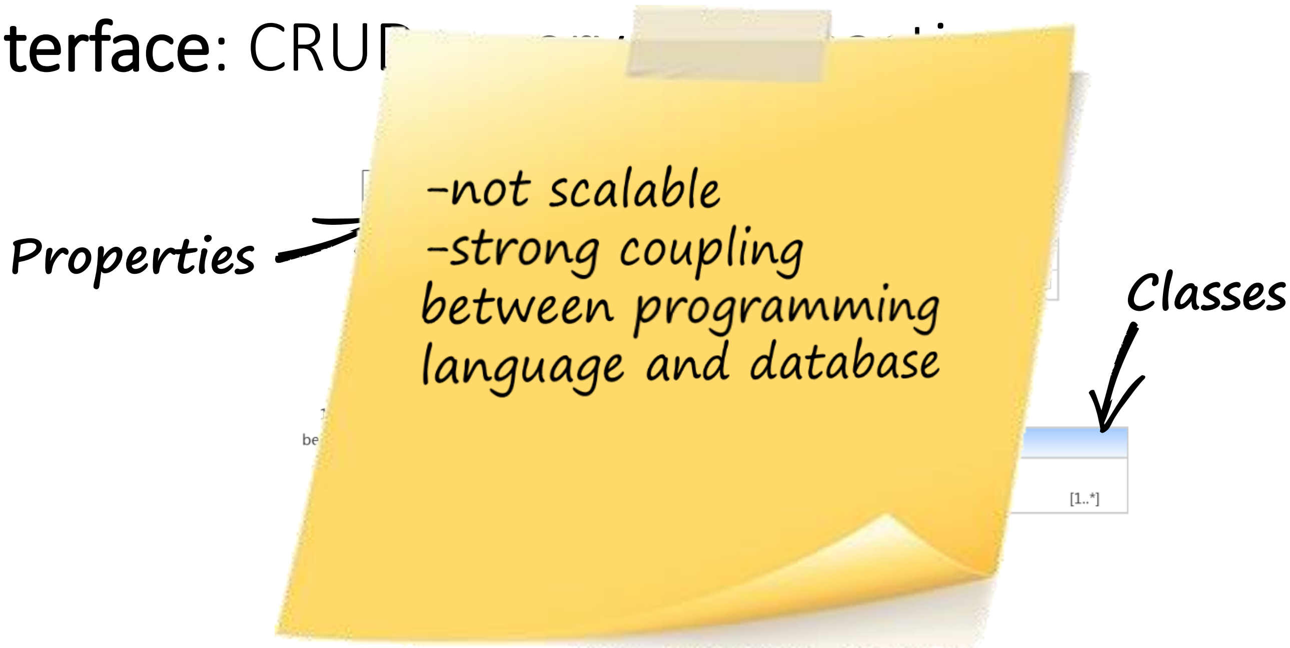
Search Platforms

- **Data model:** vectorspace model, docs + metadata
- Examples: Solr, ElasticSearch



Object-oriented Databases

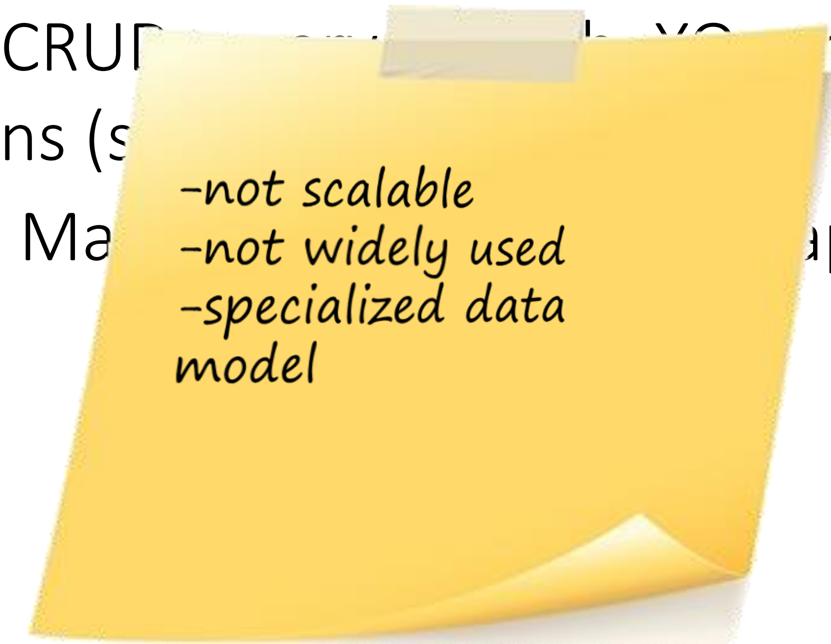
- **Data model:** Classes, objects, relations (references)
- **Interface**: CRUD



- Examples: Versant (CA), db4o (CA), Objectivity (CA)

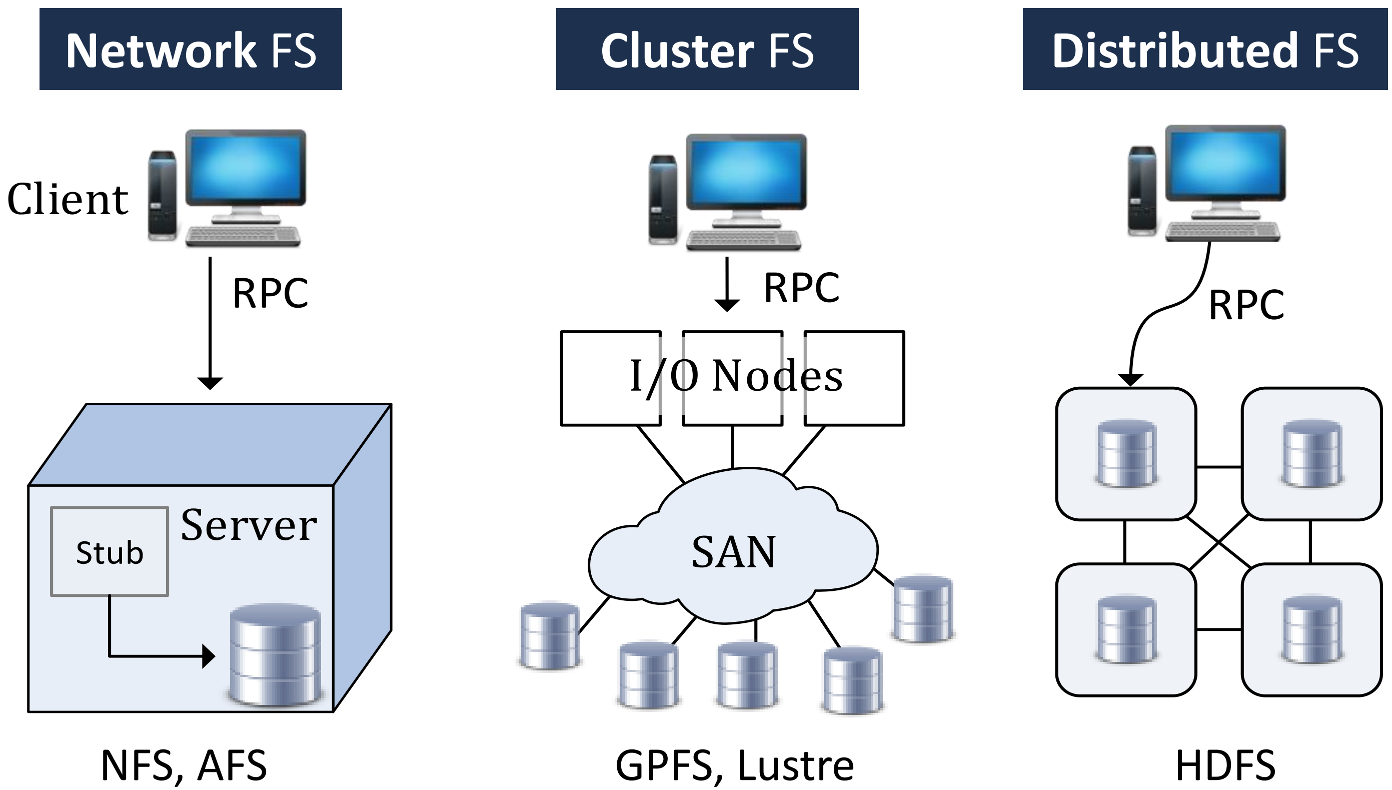
XML databases, RDF Stores

- **Data model:** XML, RDF
- **Interface**: CRU … rys, SPARQL), transactions (s…
- Examples: Ma … aph (CA)



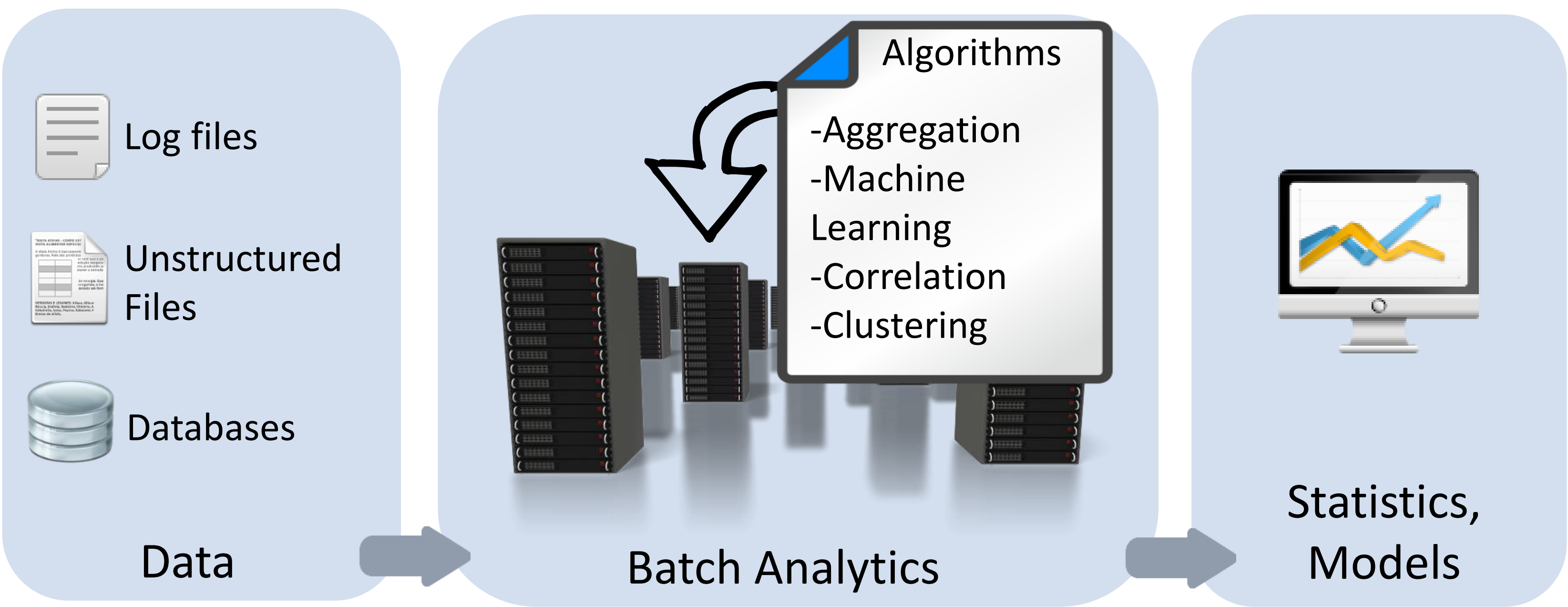
Distributed File System

- **Data model:** files + folders



Big Data Frameworks

- **Data model:** arbitrary (frequently unstructured)
- Examples: Hadoop, Spark, Flink, DryadLink, Pregel



Wrap-up

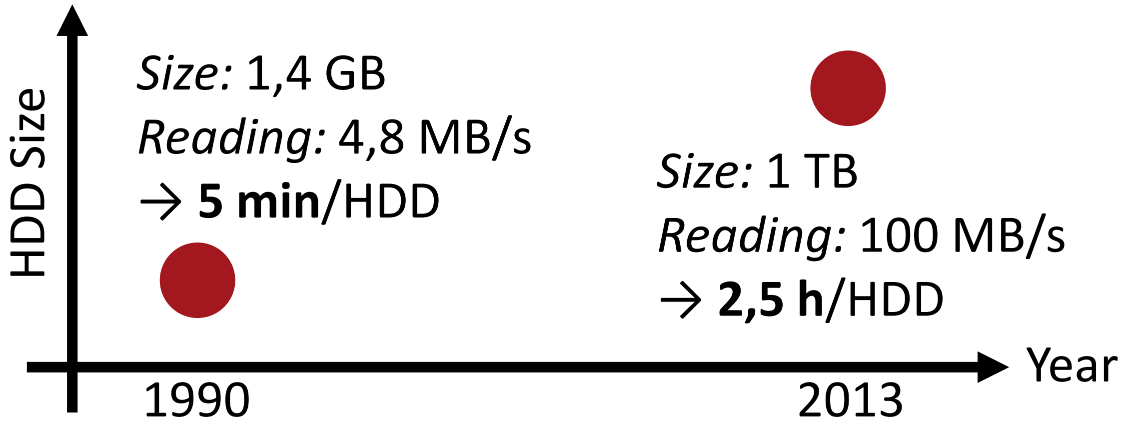
- 4 core NoSQL classes
  - **Key-Value** Stores: store opaque key-value pairs
  - **Document** Stores: store nested, rich, schema-free documents
  - **Wide-Column** Stores: extensible table data model
  - **Graph Databases**: graph-property-model (vertices and edges)
- Other NoSQL-related systems: Object-oriented databases, Search platforms, XML databases, Big Data Frameworks, Distributed File Systems
- **Lambda Architecture**: Pattern for integrating batch and realtime analytics in a scalable fashion using NoSQL technologies

Outline

- Foundations: Big Data, Scalability, Avaialbility
- The 4 Classes of NoSQL Databases
- NoSQL Examples: concrete Architectures, Systems, APIs
- Cloud Databases

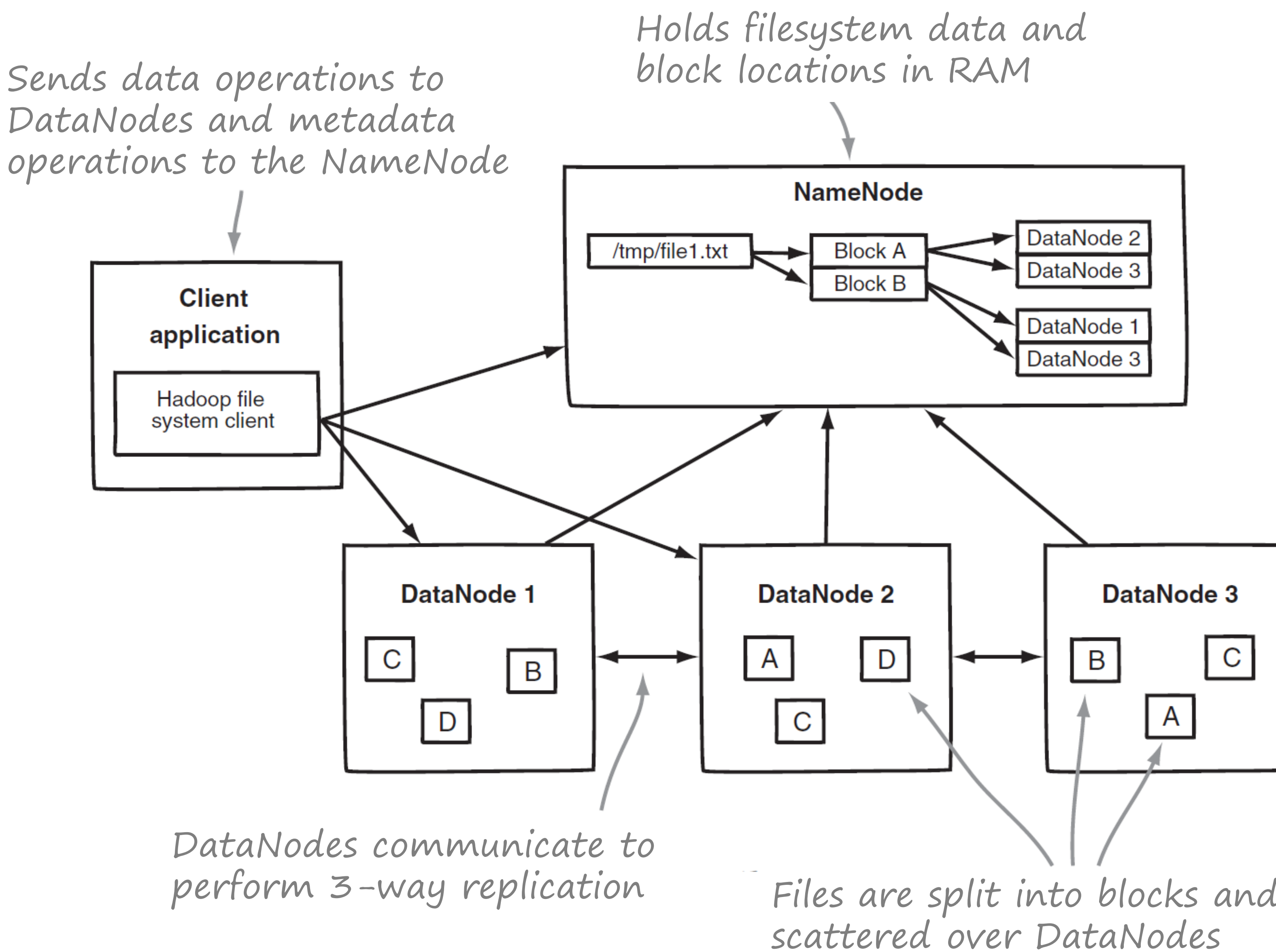
- MapReduce (Hadoop)
- Dynamo (Riak)
- BigTable (HBase)
- MongoDB
- Others

Hadoop Distributed FS (CP)



HDFS
Model:
File System
License:
Apache 2
Written in:
Java

- Modelled after: Googles GFS (2003)
- **Master-Slave** Replication
  - **Namenode**: Metadata (files + block locations)
  - **Datanodes**: Save file blocks (usually 64 MB)
- **Design goal**: Maximum Throughput and data locality for Map-Reduce



Holmes, Alex. Hadoop in Practice. Manning, 2012.

Hadoop

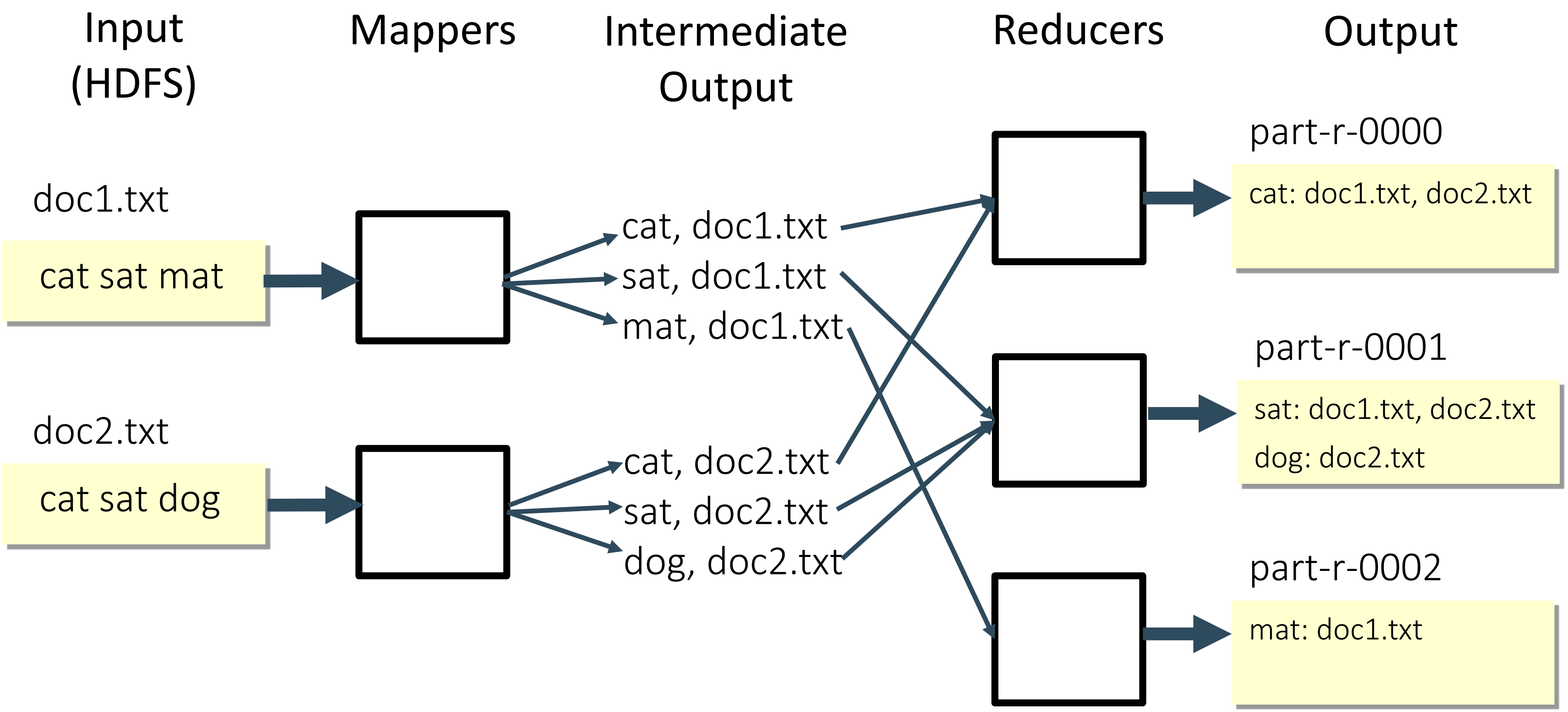
- For many synonymous to *Big Data Analytics*
- Large Ecosystem
- Creator: Doug Cutting (Lucene)
- Distributors: Cloudera, MapR, HortonWorks
- Gartner Prognosis: By 2015 65% of all complex analytic applications will be based on Hadoop
- Users: Facebook, Ebay, Amazon, IBM, Apple, Microsoft, NSA

Hadoop
Model:
Batch-Analytics Framework
License:
Apache 2
Written in:
Java

http://de.slideshare.net/cultureofperformance/gartner-predictions-for-hadoop-predictions

MapReduce: Example

Constructing a reverse-index

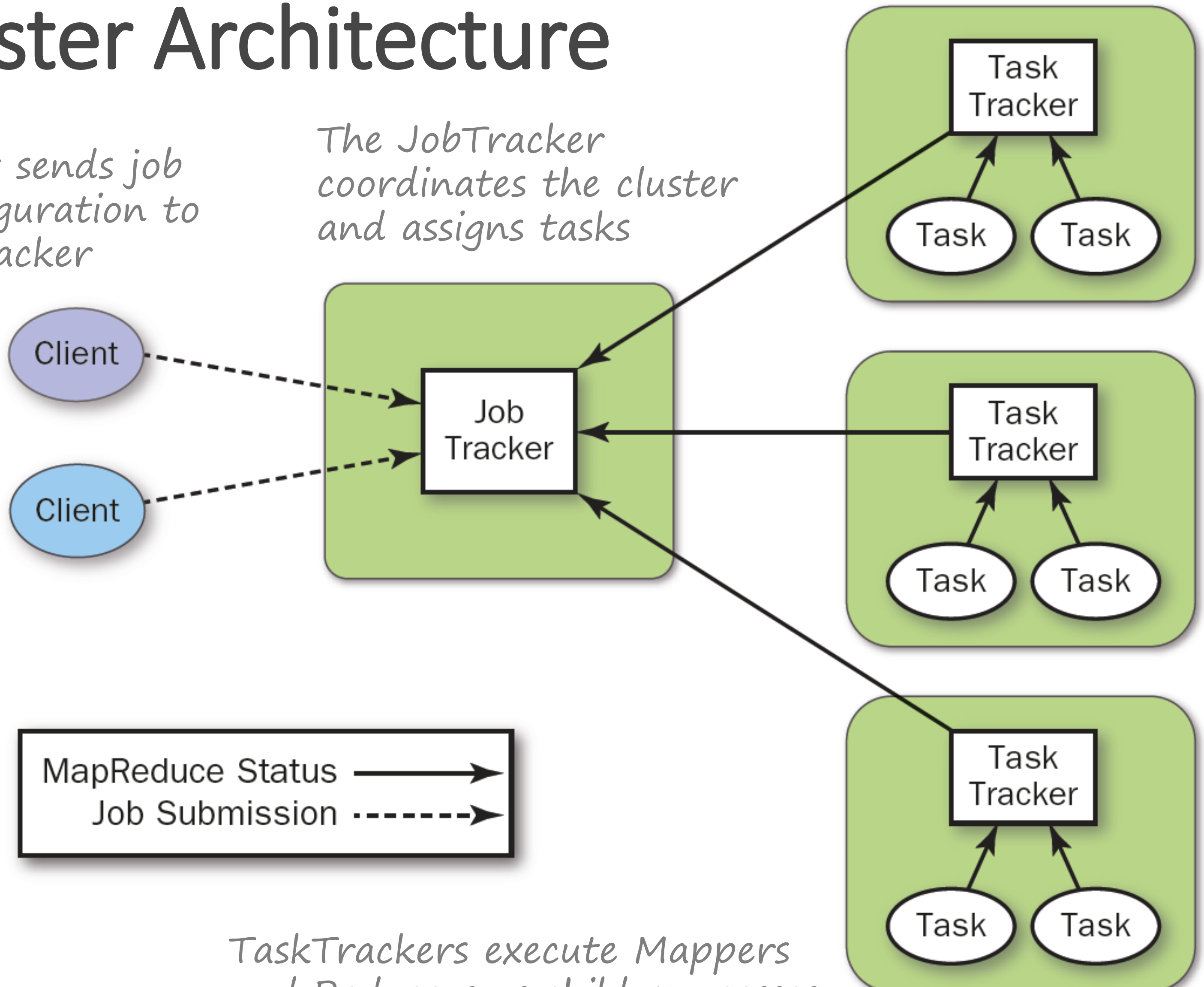


Holmes, Alex. Hadoop in Practice

Cluster Architecture

The client sends job and configuration to the Jobtracker

The JobTracker coordinates the cluster and assigns tasks

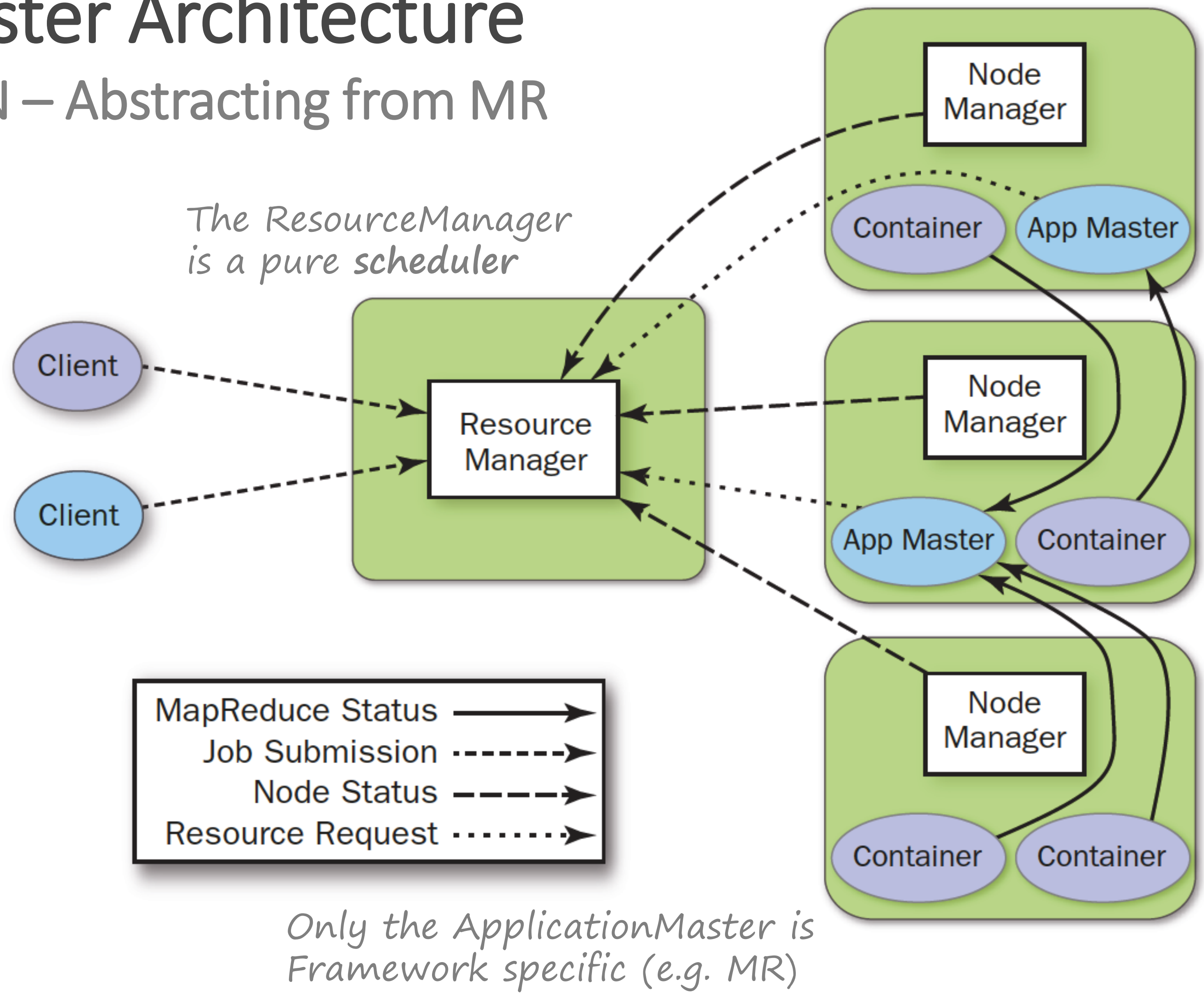


TaskTrackers execute Mappers and Reducers as child-processes

Arun Murthy “Apache Haddop YARN”

Cluster Architecture

YARN – Abstracting from MR



Arun Murthy "Apache Haddop YARN"

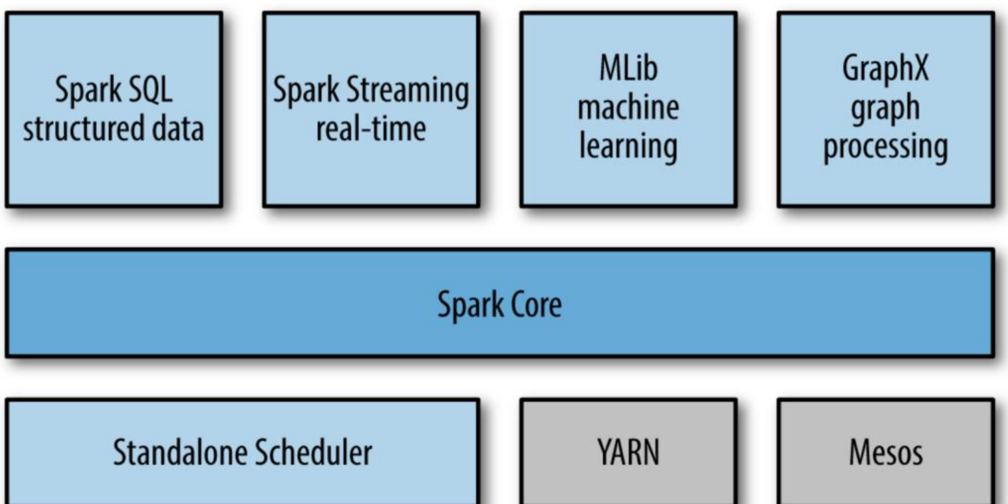
Summary: Hadoop Ecosystem

- **Hadoop**: Ecosystem for Big Data Analytics
- **Hadoop Distributed File System**: scalable, shared-nothing file system for thoughput-oriented workloads
- **Map-Reduce**: Paradigm for performing scalable distributed batch analysis
- Other Hadoop projects:
  - **Hive**: SQL(-dialect) compiled to YARN jobs (Facebook)
  - **Pig**: workflow-oriented scripting language (Yahoo)
  - **Mahout**: Machine-Learning algorithm library in Map-Reduce
  - **Flume**: Log-Collection and processing framework
  - **Whirr**: Hadoop provisioning for cloud environments
  - **Giraph**: Graph processing à la Google Pregel
  - **Drill**, **Presto, Impala**: SQL Engines

Spark

- „In-Memory“ Hadoop that does not suck for iterative processing (e.g. k-means)
- Resilient Distributed Datasets (**RDDs**): partitioned, in-memory set of records

Spark
Model:
Batch Processing Framework
License:
Apache 2
Written in:
Scala



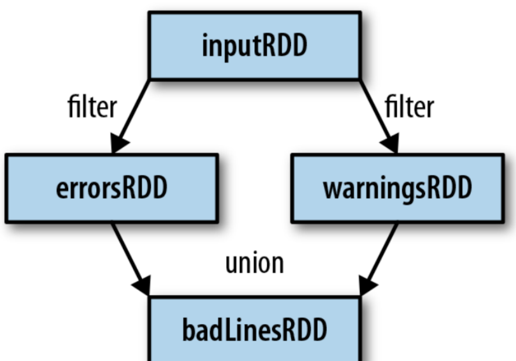
M. Zaharia, M. Chowdhury, T. Das, et al. „Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing“

Spark

Example RDD Evaluation

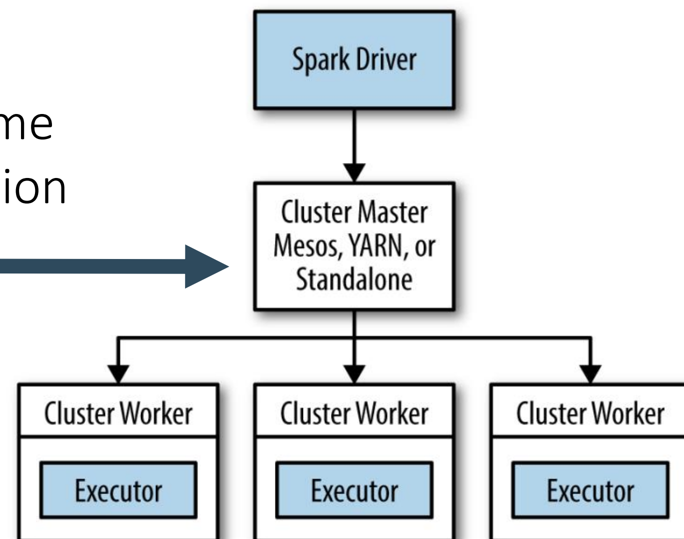
- **Transformations**: RDD → RDD
- **Actions**: Reports an operation

```
errors = sc.textFile("log.txt").filter(lambda x: "error" in x)
warnings = inputRDD.filter(lambda x: "warning" in x)
badLines = errorsRDD.union(warningsRDD).count()
```



RDD Lineage

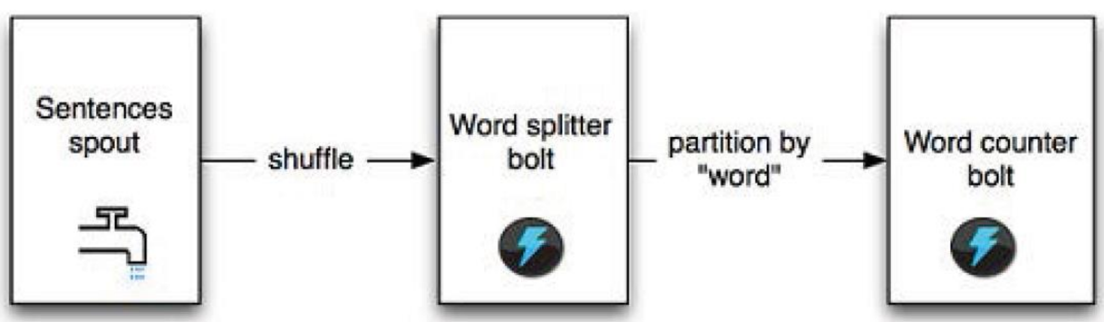
Runtime Execution



H. Karau et al. „*Learning Spark*"

Storm

- Distributed Stream Processing Framework
- Topology is a DAG of:
  - **Spouts**: Data Sources
  - **Bolts**: Data Processing Tasks
- Cluster:
  - **Nimbus** (Master) ↔ **Zookeeper** ↔ **Worker**



Storm
Model:
Stream Processing Framework
License:
Apache 2
Written in:
Java

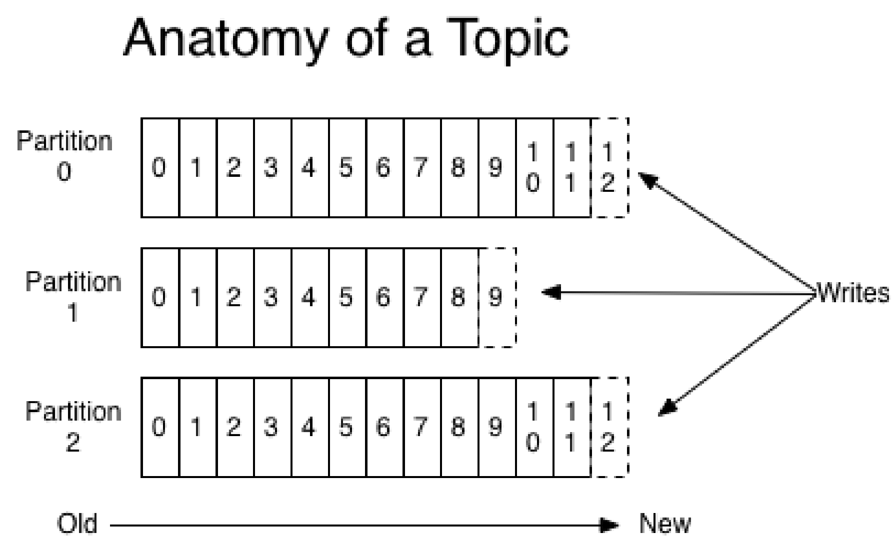
Nathan Marz „Big Data"

Kafka

- Scalable, Persistent Pub-Sub
- Log-Structured Storage
- **Guarantee**: At-least-once
- **Partitioning**:
  - By Topic/Partition
  - Producer-driven
    - Round-robin
    - Semantic
- **Replication**:
  - Master-Slave
  - Synchronous to majority

Apache Kafka

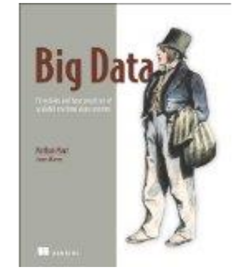
Kafka
Model:
Distributed Pub-Sub-System
License:
Apache 2
Written in:
Scala



J. Kreps, N. Narkhede, J. Rao, und others, „Kafka: A distributed messaging system for log processing"

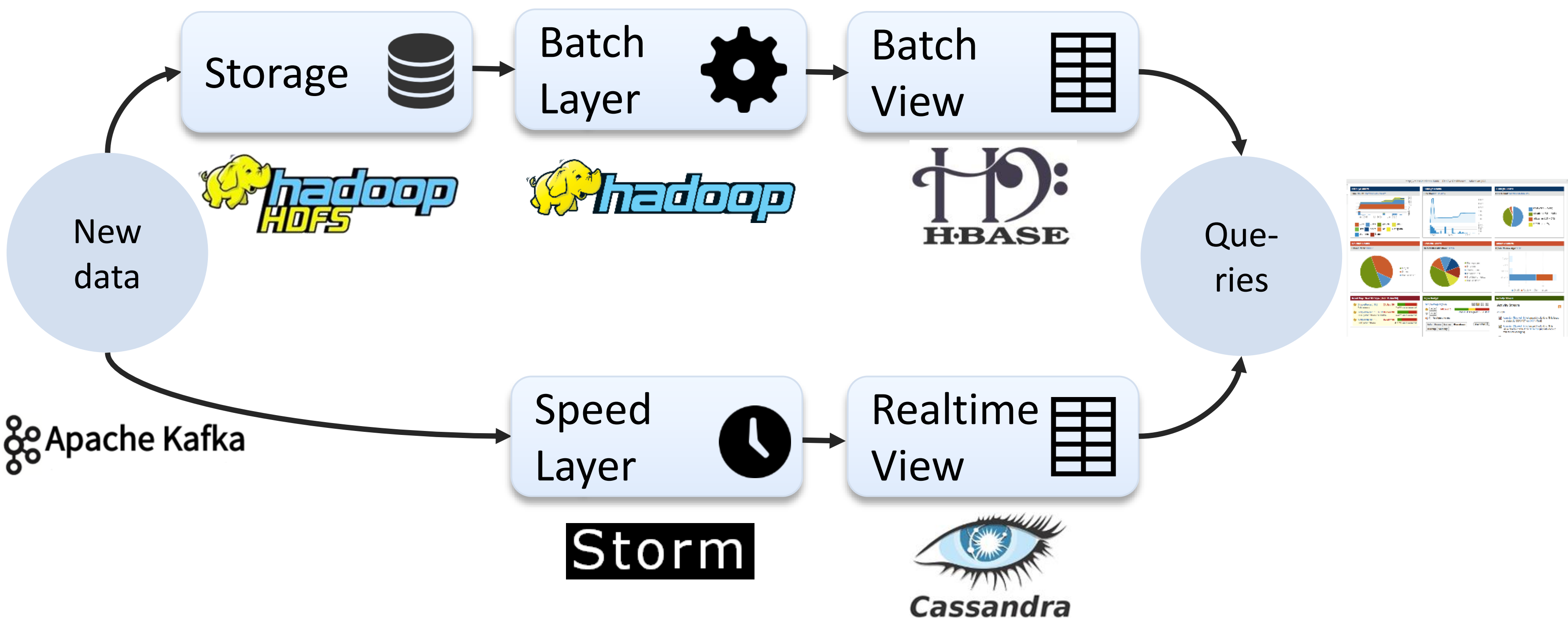
Lambda Architecture

An emerging Big Data Architecture

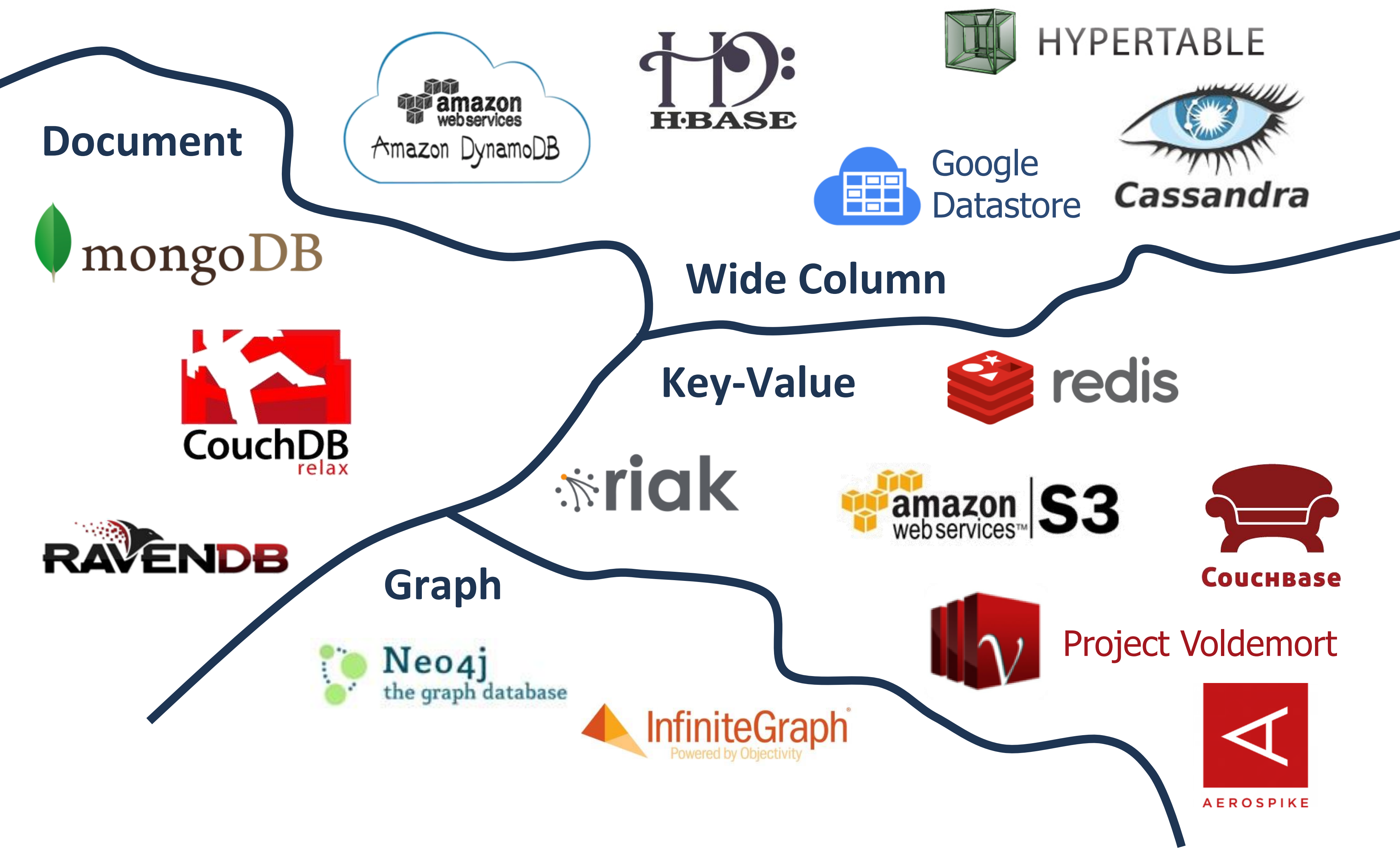


Given analytic function f:

$$result = f(all\ data) = f(old\ data) \odot f(lambda)$$



NoSQL landscape



Popularity

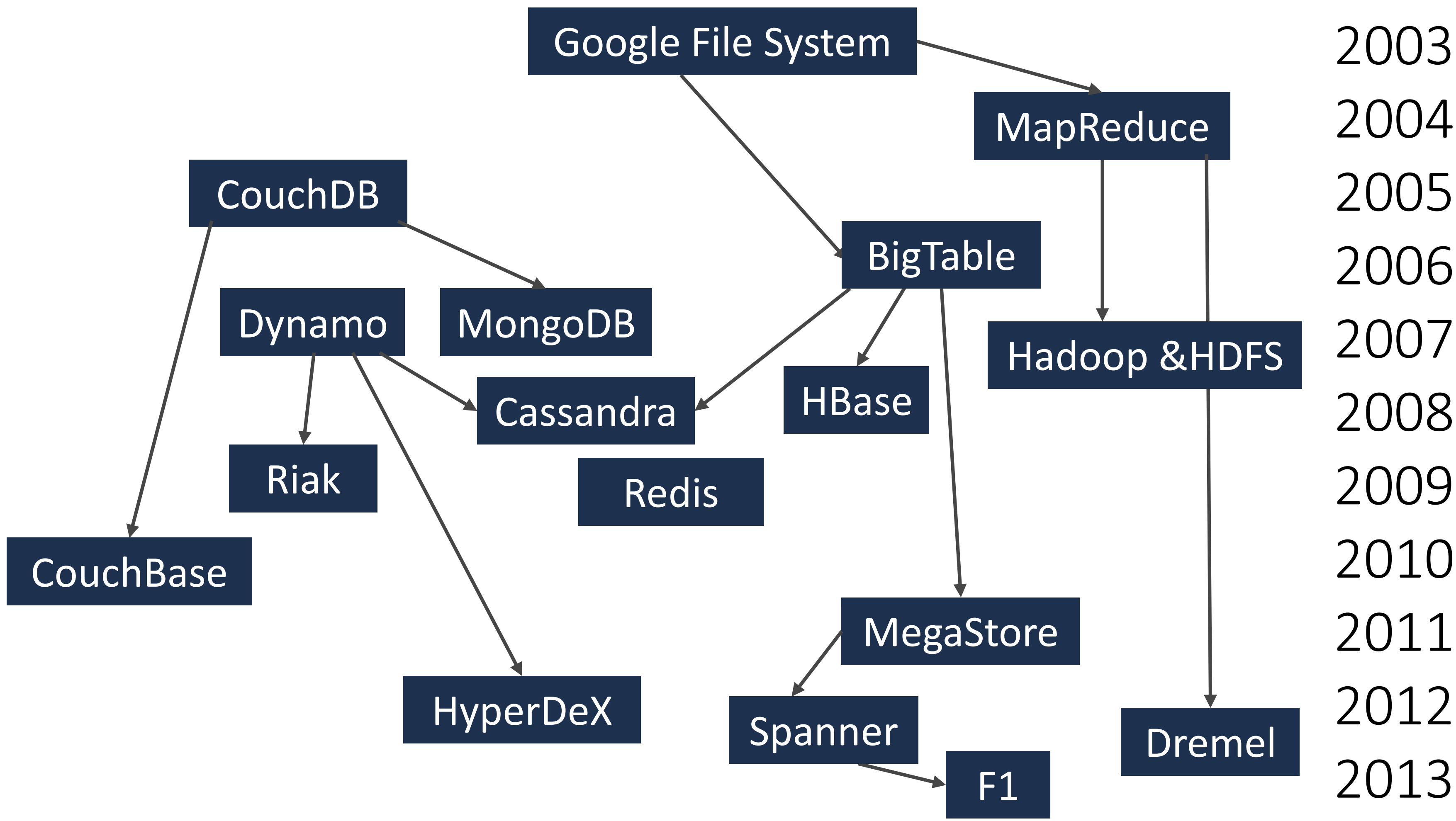
http://db-engines.com/de/ranking

Rang	DBMS	Modell	Punkte
1.	Oracle	Relational DBMS	1514,90
2.	MySQL	Relational DBMS	1334,94
3.	Microsoft SQL Server	Relational DBMS	1286,22
4.	PostgreSQL	Relational DBMS	199,39
5.	DB2	Relational DBMS	177,04
6.	Microsoft Access	Relational DBMS	149,66
7.	**MongoDB**	**Document Store**	**137,49**
8.	Sybase	Relational DBMS	88,41
9.	SQLite	Relational DBMS	87,81
10.	Teradata	Relational DBMS	51,11
11.	Solr	Suchmaschine	46,43
12.	**Cassandra**	**Wide Column Store**	**37,64**
13.	**Redis**	**Key-Value Store**	**34,22**

Rang	DBMS	Modell	Punkte
14.	**Memcached**	**Key-Value Store**	**30,73**
15.	**HBase**	**Wide Column Store**	**25,78**
16.	Informix	Relational DBMS	24,73
17.	Hive	Relational DBMS	22,16
18.	**CouchDB**	**Document Store**	**15,93**
19.	Firebird	Relational DBMS	14,55
20.	Netezza	Relational DBMS	11,44
21.	dBASE	Relational DBMS	10,44
22.	Elasticsearch	Suchmaschine	9,51
23.	Sphinx	Suchmaschine	9,02
24.	**Riak**	**Key-Value Store**	**8,99**
25.	**Neo4j**	**Graph DBMS**	**8,83**

Scoring: Google/Bing results, Google Trends, Stackoverflow, job offers, LinkedIn

History



Google File System
MapReduce
CouchDB
BigTable
Dynamo
MongoDB
Hadoop &HDFS
Cassandra
HBase
Riak
Redis
CouchBase
MegaStore
HyperDeX
Spanner
F1
Dremel
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

NoSQL foundations

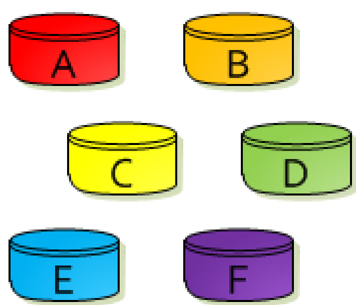
- **BigTable** (2006, Google)
  - Consistent, Partition Tolerant
  - Wide-Column data model
  - Master-based, fault-tolerant, large clusters (1.000+ Nodes), **HBase**, **Cassandra**, **HyperTable, Accumolo**
- **Dynamo** (2007, Amazon)
  - Available, Partition tolerant
  - Key-Value interface
  - Eventually Consistent, always writable, fault-tolerant
  - **Riak**, **Cassandra**, **Voldemort**, **DynamoDB**

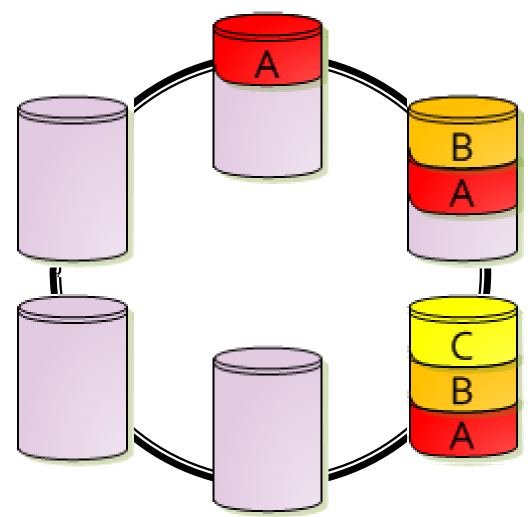
Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."

DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."

Dynamo (AP)

- Developed at Amazon (2007)
- Sharding of data over a ring of nodes
- Each node holds multiple partitions
- Each partition **N**-times replicated

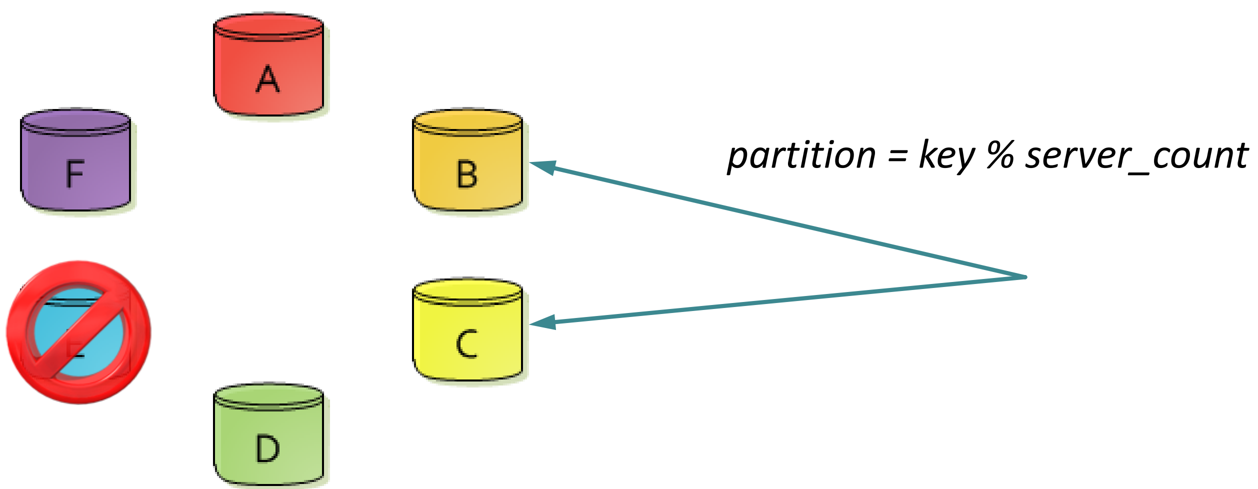




DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."

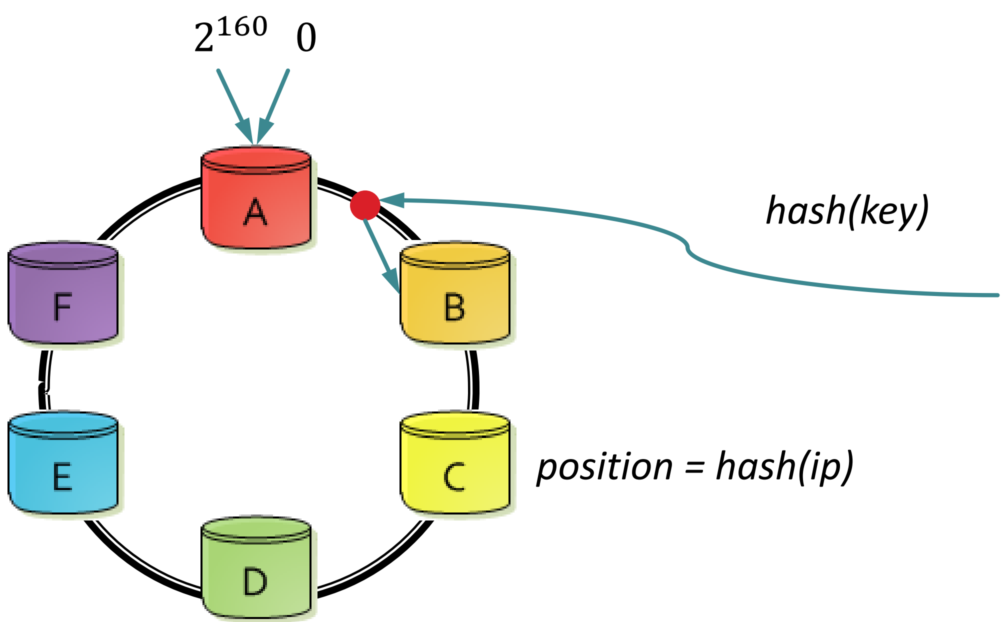
Consistent Hashing

- Naive approach: **Hash-partitioning** (e.g. in Memcache, Redis)



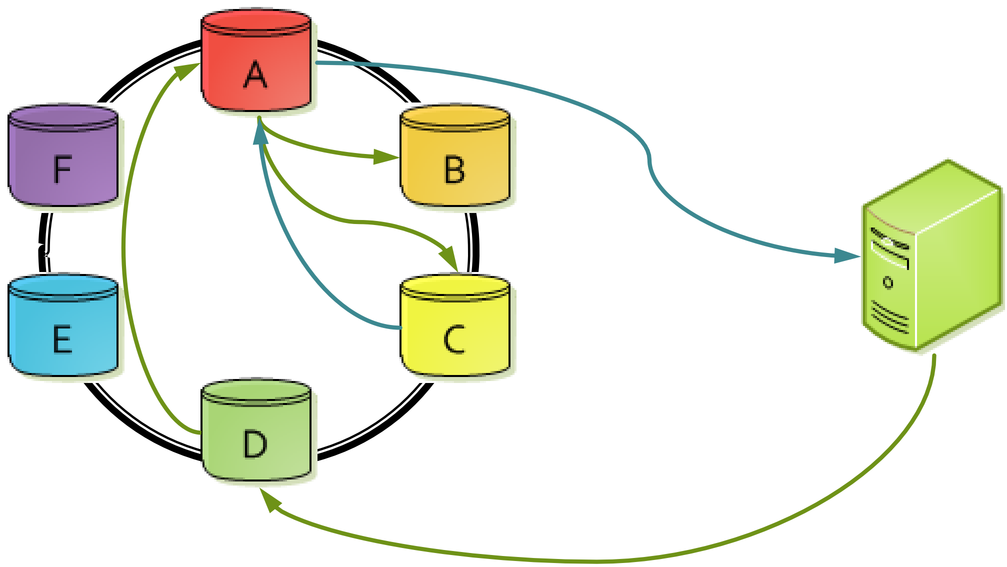
Consistent Hashing

- Solution: **Consistent Hashing** – mapping of data to nodes is stable under topology changes



Reading and Writing

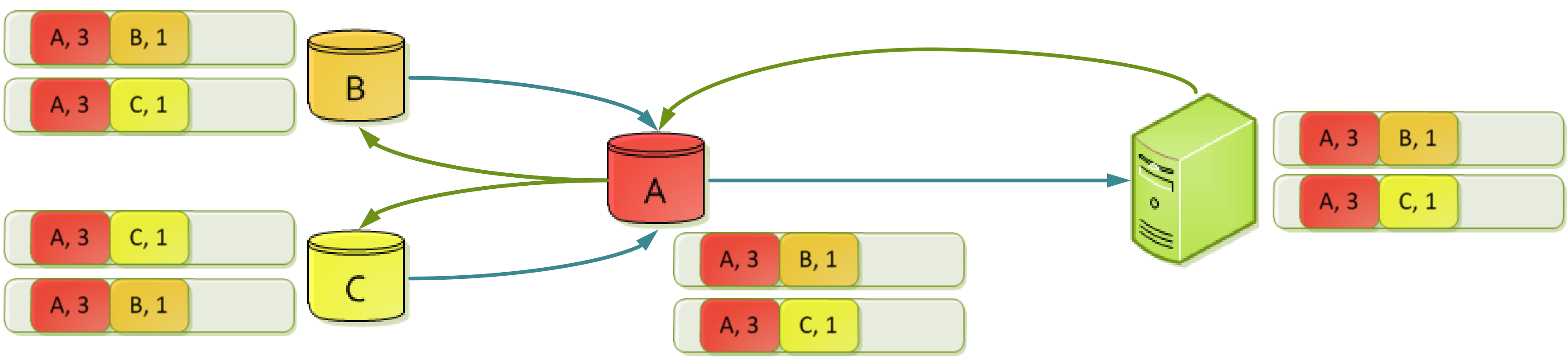
- An arbitrary node acts as a coordinator
- **N**: number of replicas
- **R**: number of nodes that need to confirm a read
- **W**: number of nodes that need to confirm a write



N=3
R=2
W=1

Versioning and Consistency

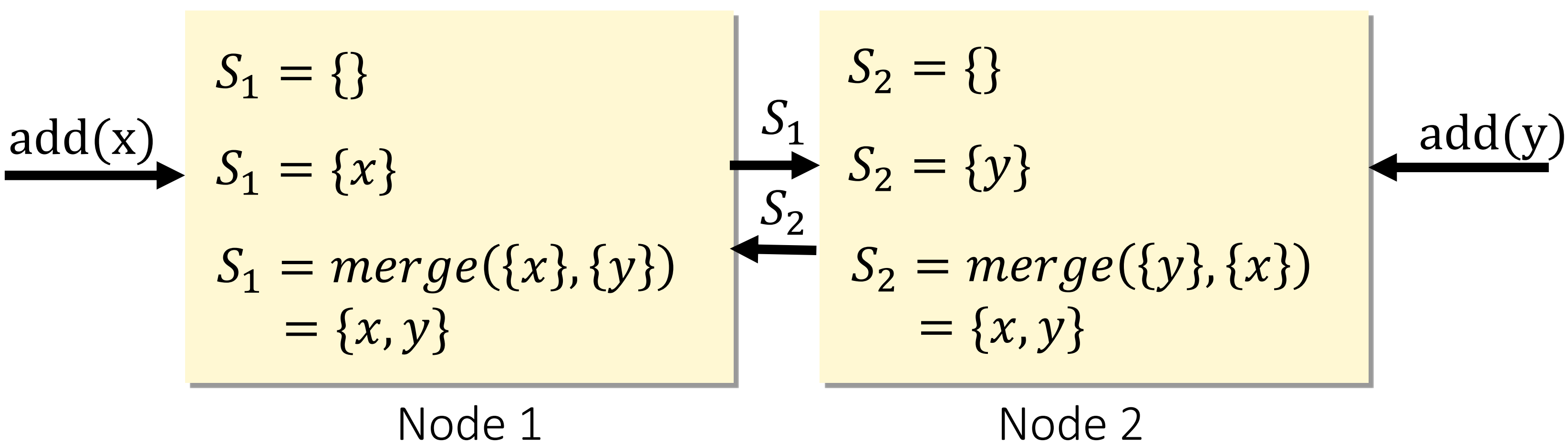
- $R + W \leq N \Rightarrow$ no consistency guarantee
- $R + W > N \Rightarrow$ newest value included in any read
- **Vector Clocks** used for versioning



CRDTs

Convergent/Commutative Replicated Data Types

- **Goal**: avoid manual conflict-resolution for
- **Approach**:
  - **State-based** – commutative, idempotent merge function
  - **Operation-based** – broadcasts of commutative upates
- Example: State-based Grow-only-Set (G-Set)



Marc Shapiro, Nuno Preguica, Carlos Baquero, and Marek Zawirski "Conflict-free Replicated Data Types"

Quorum

- Typical Configurations:

Performance (Cassandra Default)	N=3, R=1, W=1
Quorum, fast Writing:	N=3, R=3, W=1
Quorum, fast Reading	N=3, R=1, W=3
Trade-off (Riak Default)	N=3, R=2, W=2

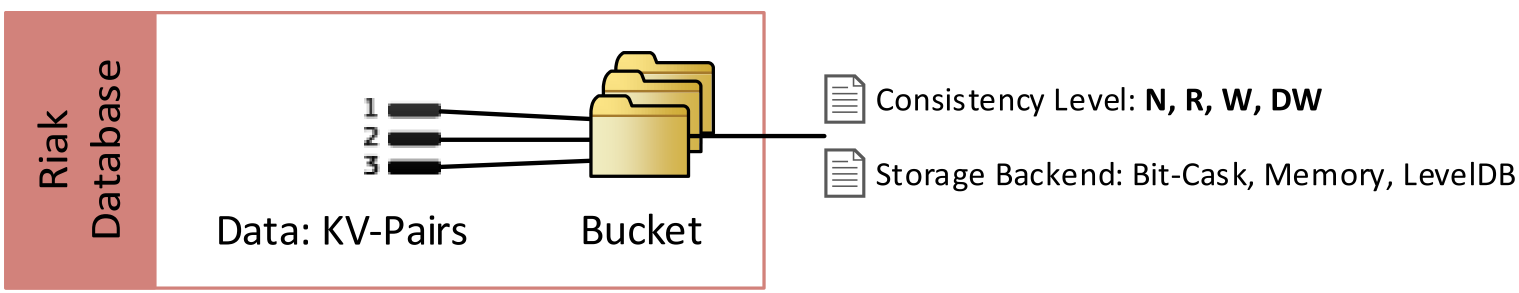
LinkedIn (SSDs):
$P(consistent) \geq 99.9\%$
nach $1.85\ ms$

P. Bailis, PBS Talk: http://www.bailis.org/talks/twitter-pbs.pdf

Riak (AP)

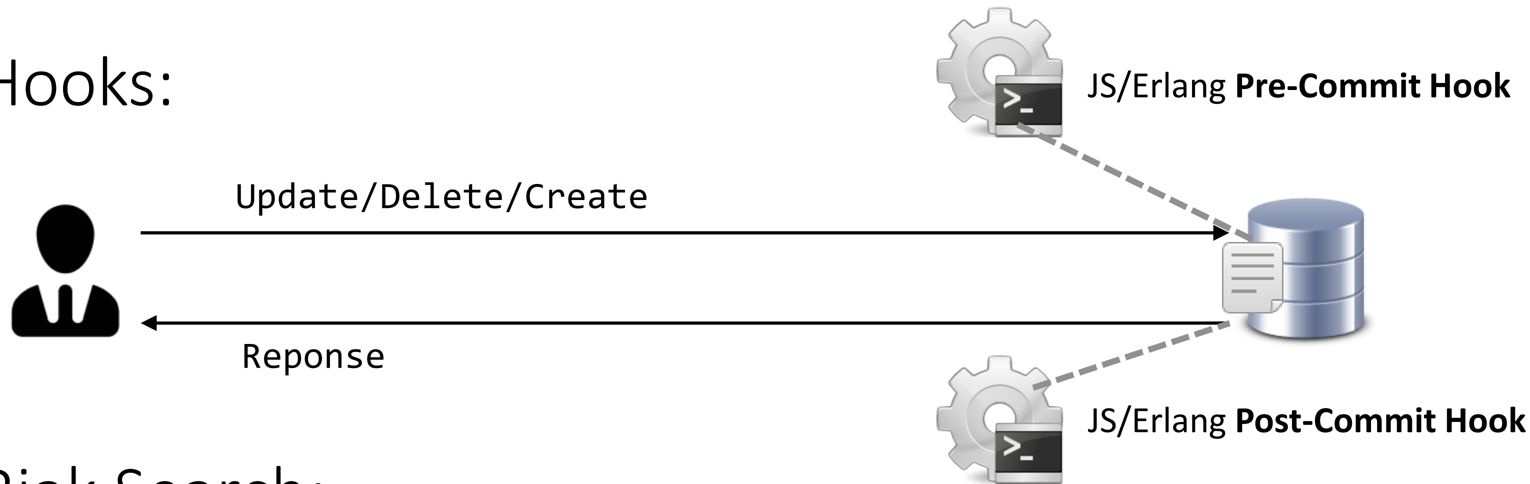
Riak
Model:
Key-Value
License:
Apache 2
Written in:
Erlang und C

- Open-Source Dynamo-Implementation
- Extends Dynamo:
  - Keys are grouped to **Buckets**
  - KV-pairs may have **metadata** and **links**
  - Map-Reduce support
  - Secondary Indices, Update Hooks, Solr Integration
  - **CRDTs**

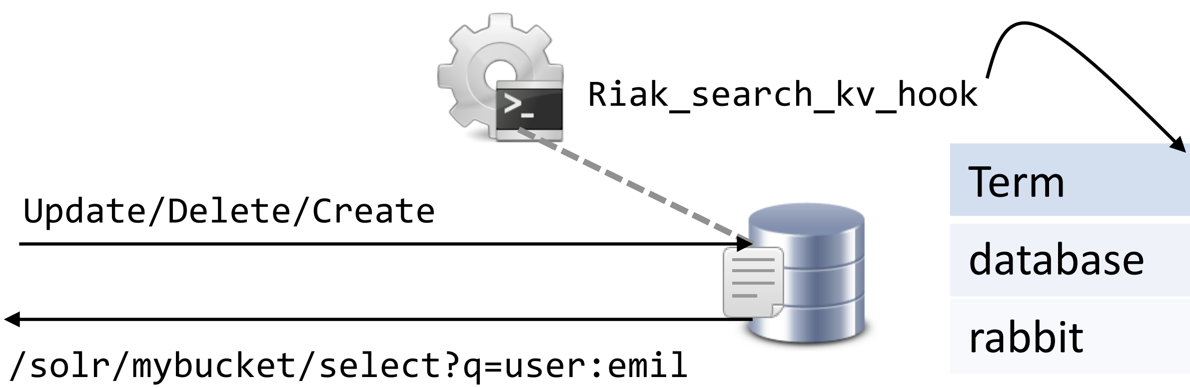


Hooks & Search

- Hooks:



- Riak Search:



Term	Dokument
database	3,4,1
rabbit	2

Search Index

Summary: Dynamo and Riak

- **Consistent Hashing**: hash-based distribution with stability under topology changes (e.g. machine failures)
- Parameters: **N** (Replicas), **R** (Read Acks), **W** (Write Acks)
  - N=3, R=W=1 → fast, potentially inconsistent
  - N=3, R=3, W=1 → slower reads, most recent object version contained
- Available and Partition-Tolerant
- **Vector Clocks**: concurrent modification can be detected, inconsistencies are healed through the application
- **API**: Create, Read, Update, Delete (CRUD) on key-value pairs
- **Riak**: Open-Source Implementation of the Dynamo paper

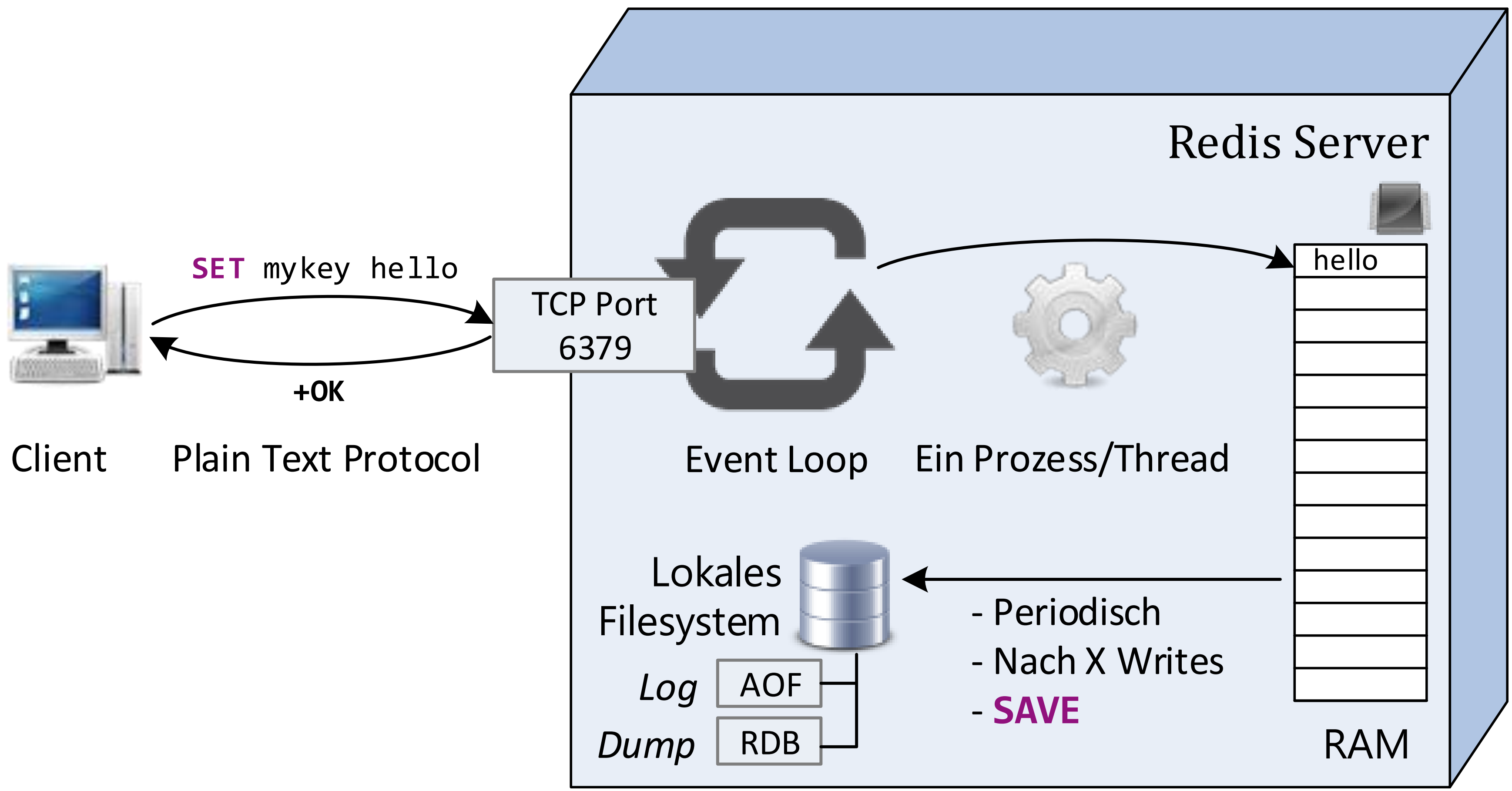
Redis (CA)

Redis
Model:
Key-Value
License:
BSD
Written in:
C

- **Re**mote **Di**ctionary **S**erver
- In-Memory Key-Value Store
- Asynchronous Master-Slave Replication
- Data model: rich data structures stored under key
- **Tunable persistence**: logging and snapshots
- Single-threaded event-loop design (similar to Node.js)
- Optimistic batch transactions (*Multi blocks*)
- Very high performance: >100k ops/sec on one machine
- Redis Cluster with sharding still under development

Redis Architektur

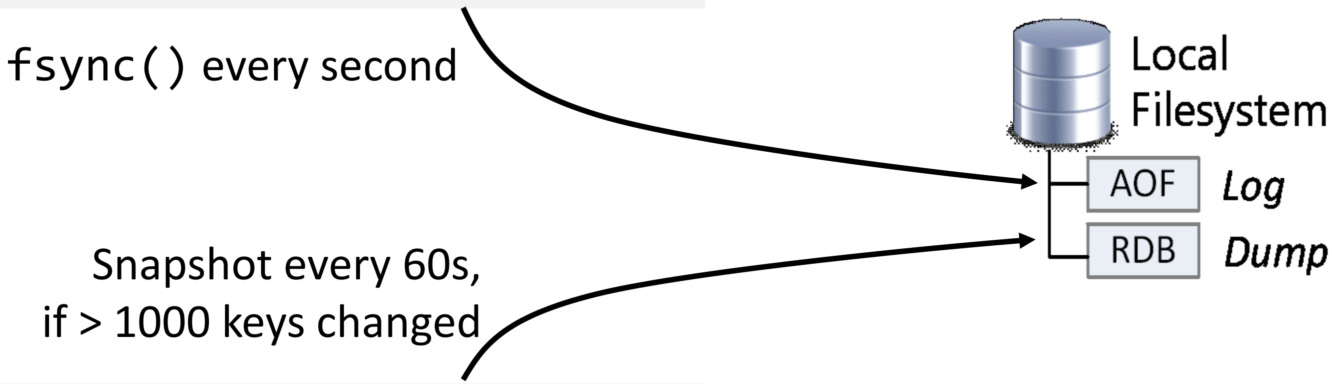
- Redis Codebase ≅ 20K LOC



Persistence

- Default: „Eventually Persistent“
- **AOF**: Append Only File (~Commitlog)
- **RDB**: Redis Database Snapshot

```
config set appendonly everysec
```




```
config set save 60 1000
```

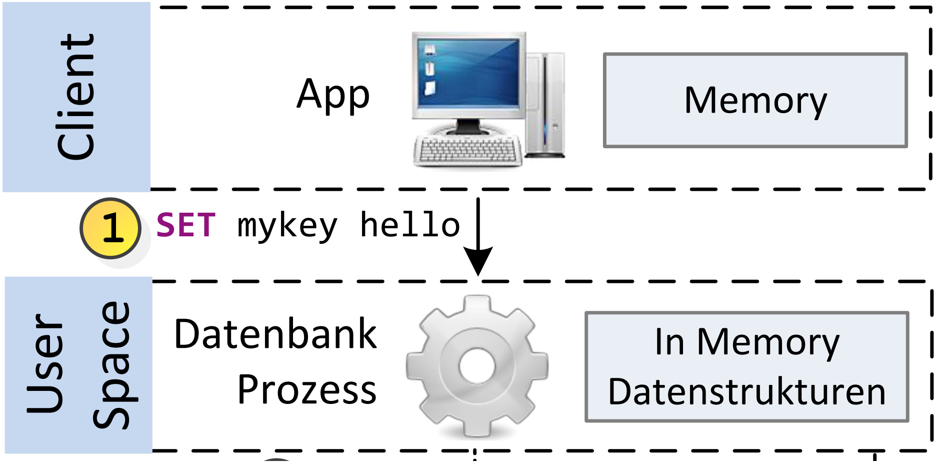
Persistenz

1. Resistenz gegen Client Crash

2. Resistenz gegen DB Prozess Crash

3. Resistenz gegen Hardware Crash bei *Write-Through*

4. Resistenz gegen Hardware Crash bei *Write-Back*

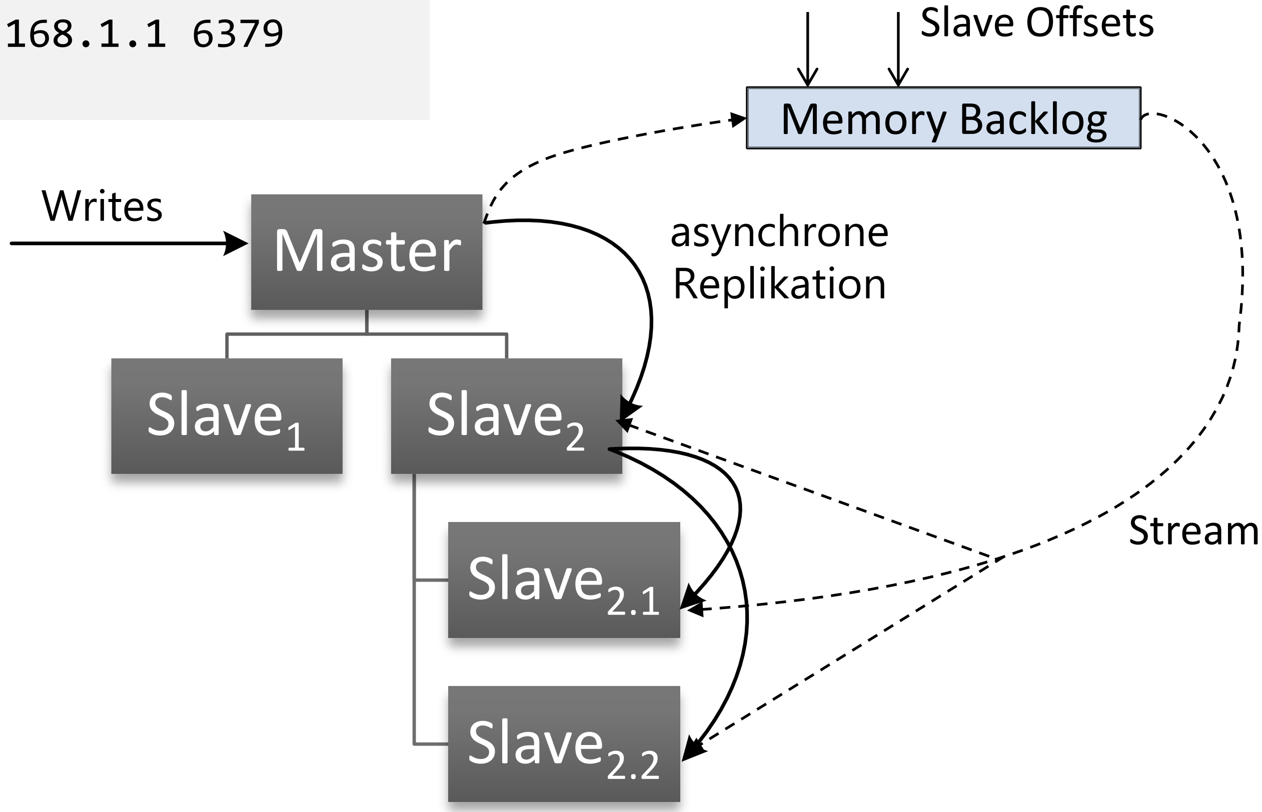


Persistenz: Redis vs RDBMS

PostgreSQL:	Redis:
> `synchronous_commit on`	> `appendfsync always`
Latenz > Disk Latenz, Group Commits, Langsam	
> `synchronous_commit off`	> `appendfsync everysec`
regelmäßiges `fsync()`, Datenverlust begrenzt	
> `fsync false`	> `appendfsync no`
Datenkorruption und –Verlust möglich	Datenverlust möglich, Korruption nicht
> `pg_dump`	> `save` oder `bgsave`

Replication

```
> SLAVEOF 192.168.1.1 6379
< +OK
```



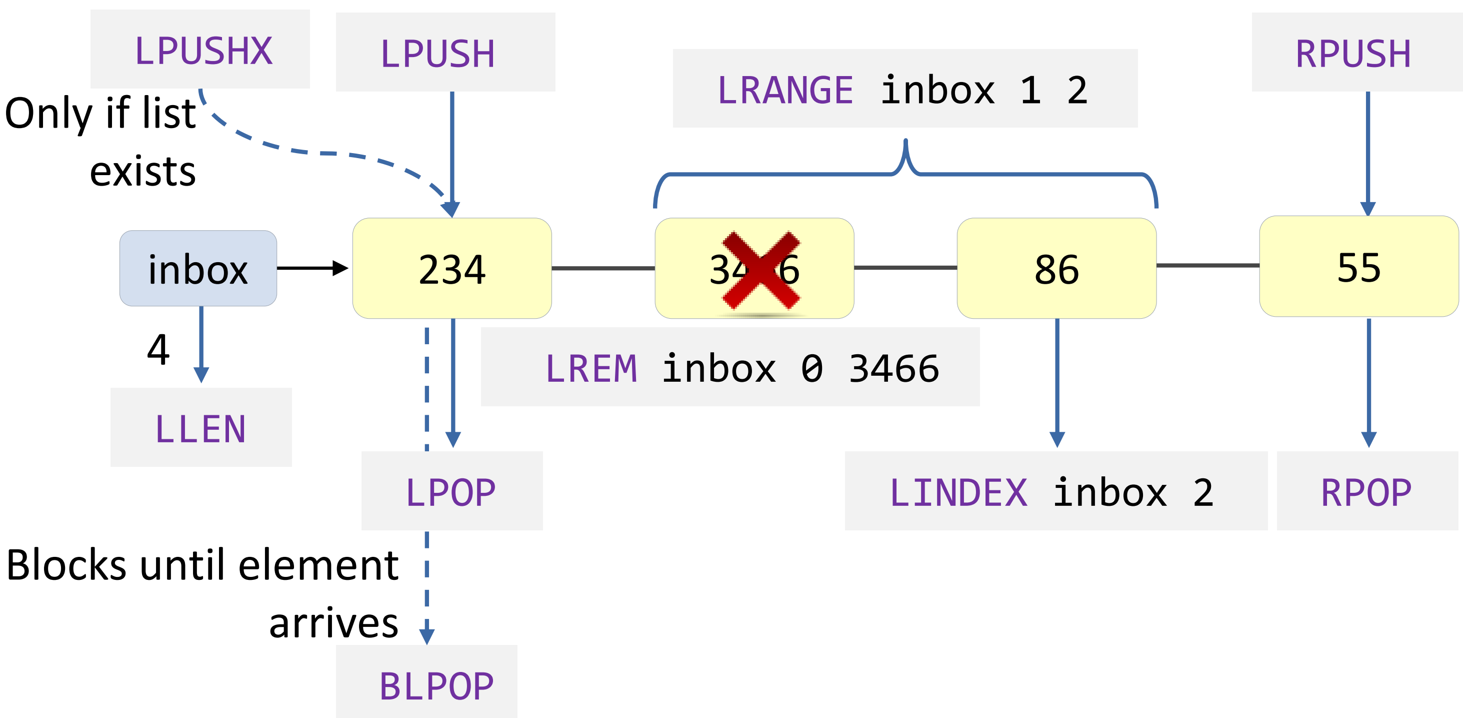
Data structures

- String, List, Set, Hash, Sorted Set

String	web:index	→ "<html><head>…"
Set	users:2:friends	→ {23, 76, 233, 11}
List	users:2:inbox	→ [234, 3466, 86,55]
Hash	users:2:settings	→ Theme → "dark", cookies → "false"
Sorted Set	top-posters	→ 466 → "2", 344 → "16"
Pub/Sub	users:2:notifs	→ "{event: 'comment posted', time : …"

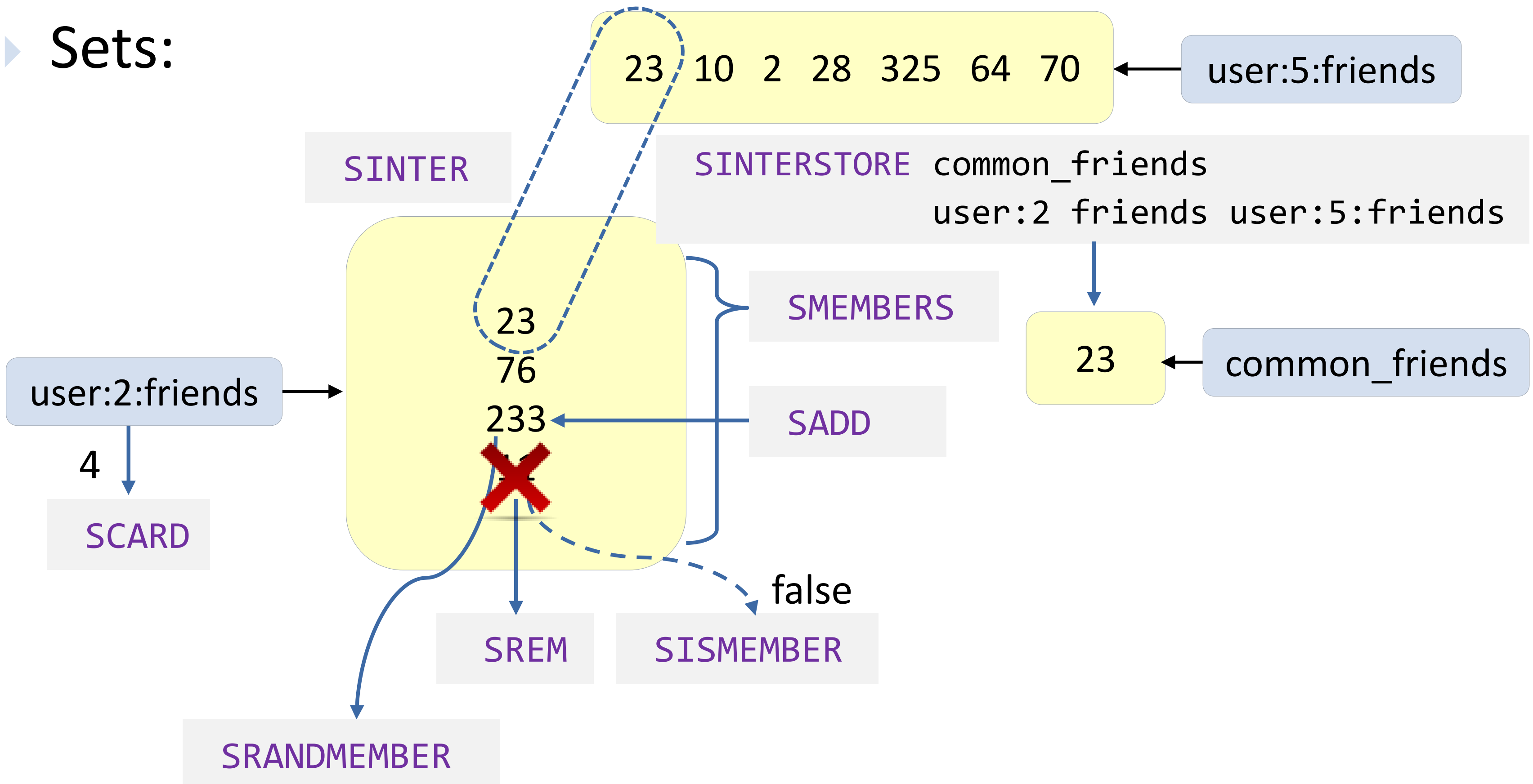
Data Structures

- (Linked) Lists:



Data Structures

- Sets:



Data Structures

- Pub/Sub: users:2:notifs → "{event: 'comment posted', time : …"

redis

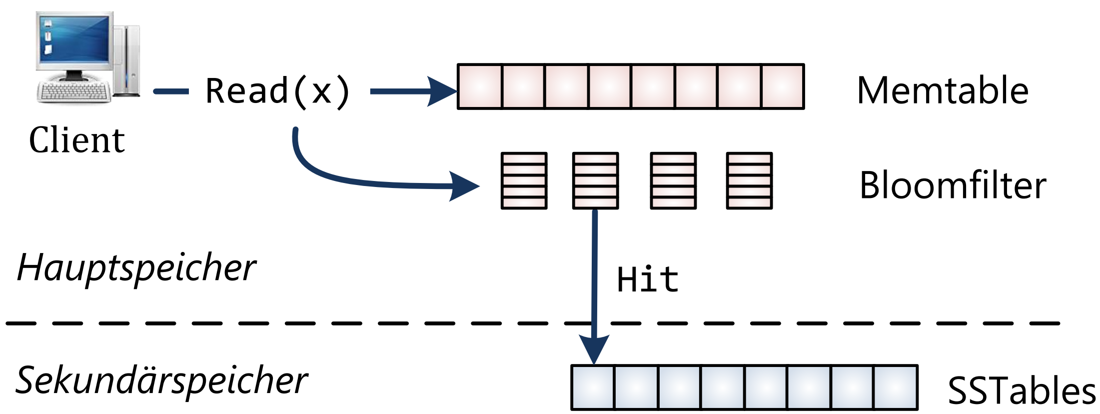
```
PUBLISH user:2:notifs
"{
   event: 'comment posted',
   time : …
}"
```

```
SUBSCRIBE user:2:notifs
```

```
{
   event: 'comment posted',
   time : …
}
```

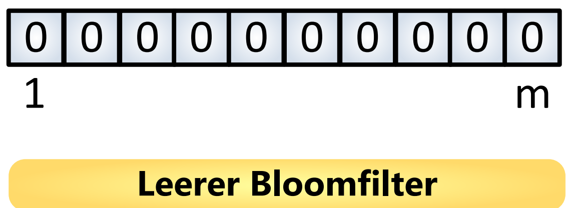
Beispiel: Bloomfilter

- Erinnerung:



- Set mit konstanter Größe
- Operationen:
  - `Add(element)` - $O(1)$
  - `Contains(element)` - $O(1)$
- False Positives möglich

Bloomfilter: Funktionsweise



Bloomfilter in Redis

- Bitvektor in Redis: String + SETBIT, GETBIT, BITOP

```
public void add(byte[] value) {
  for (int position : hash(value)) {
    jedis.setbit(name, position, true);
  }
}
```

Jedis: Redis Client für Java

SETBIT erzeugt oder vergrößert automatisch

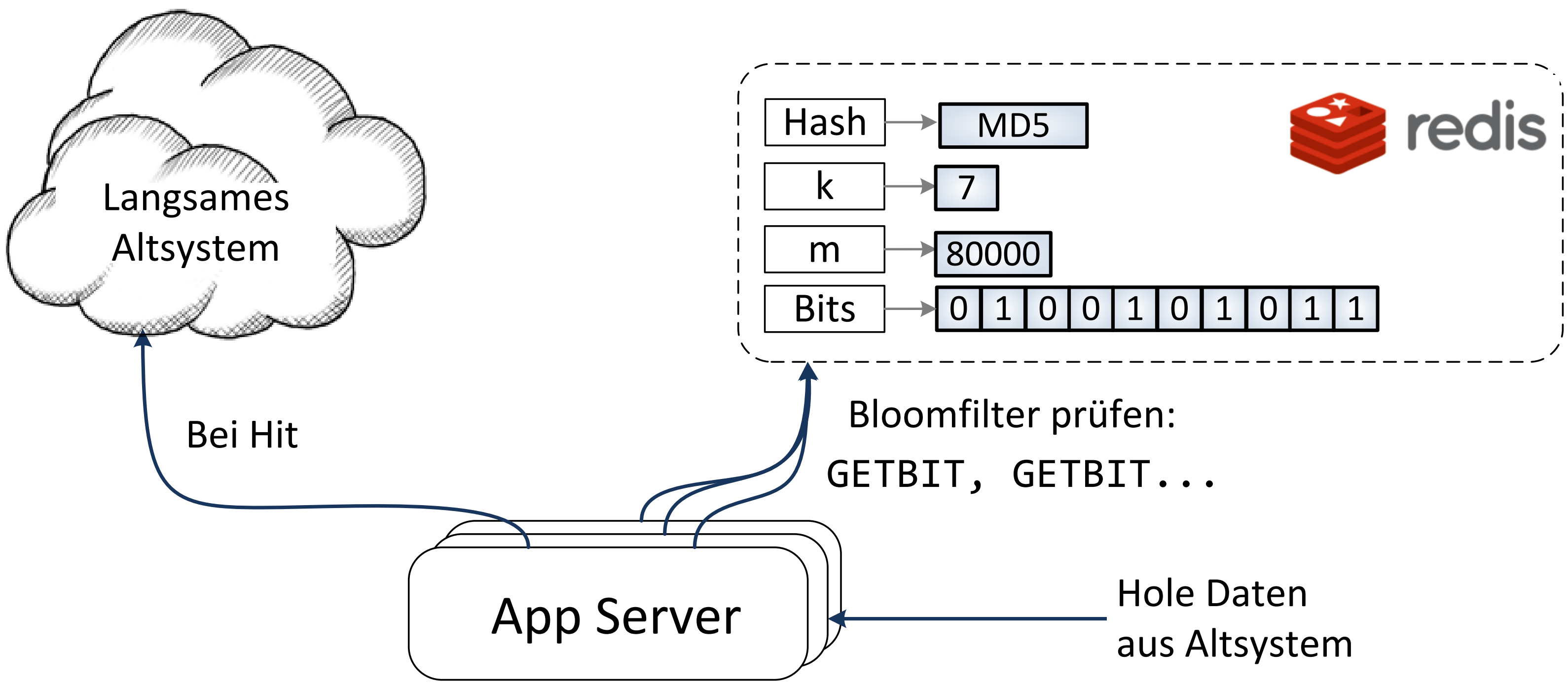
```
public void contains(byte[] value) {
  for (int position : hash(value))
     if (!jedis.getbit(name, position))
        return false;
  return true;
}
```

https://github.com/DivineTraube/Orestes-Bloomfilter

Redis für verteilte Systeme

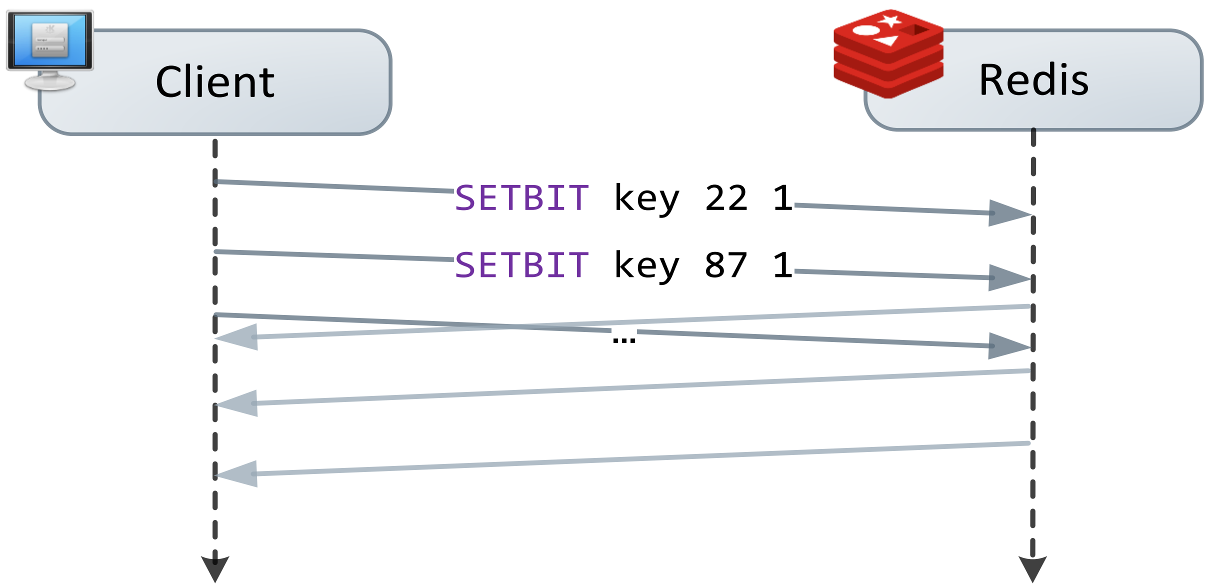
- Häufiges Pattern: verteiltes System mit gemeinsam genutzten Daten in Redis
- GETBIT und SETBIT sind idempotent und kommutativ

  → Keine Seiteneffekte bei Bloomfilterzugriff

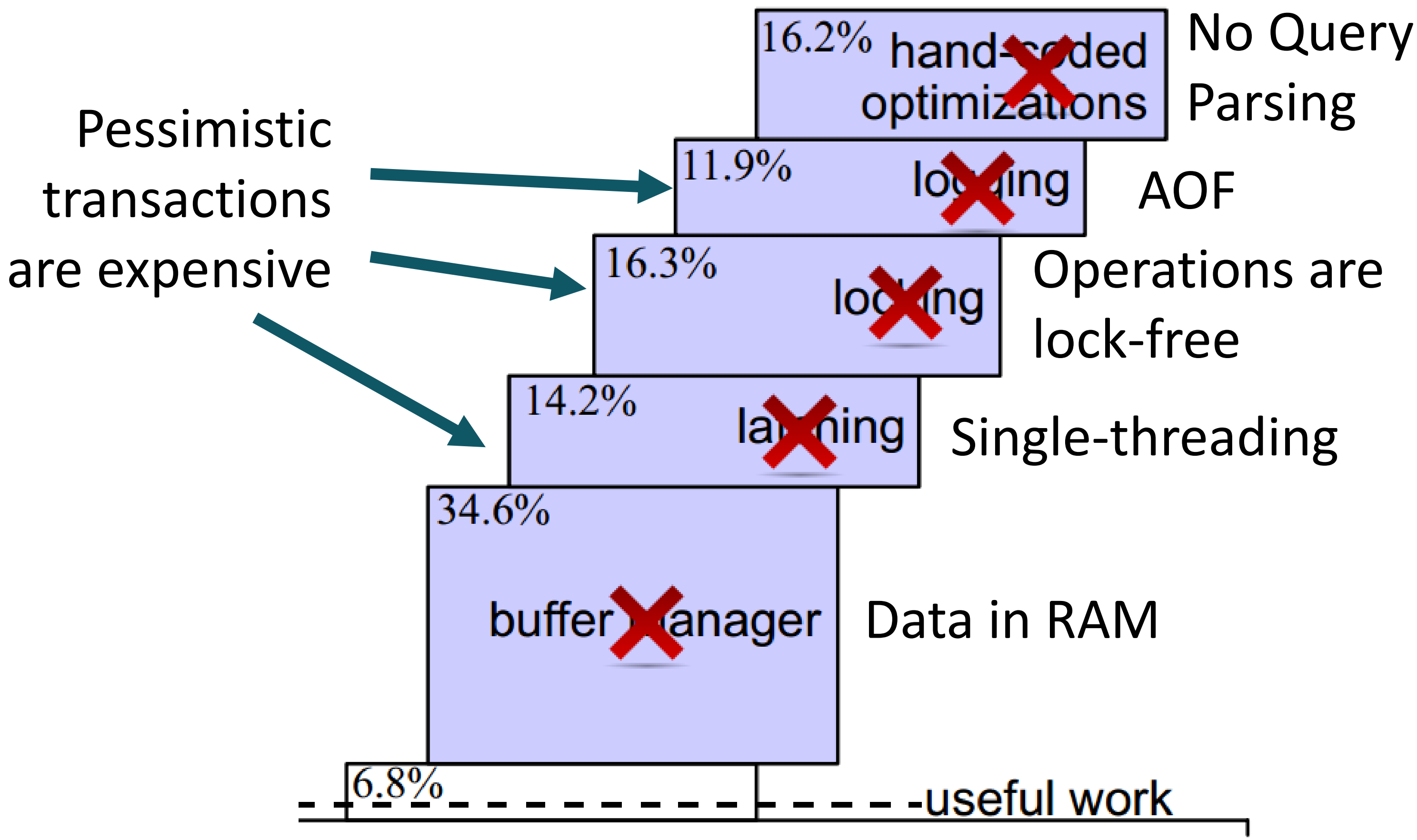


Pipelining

- Bei 7 Hashwerten: 7 Roundtrips
- **Lösung**: TCP Pipelining



Why is Redis so fast?

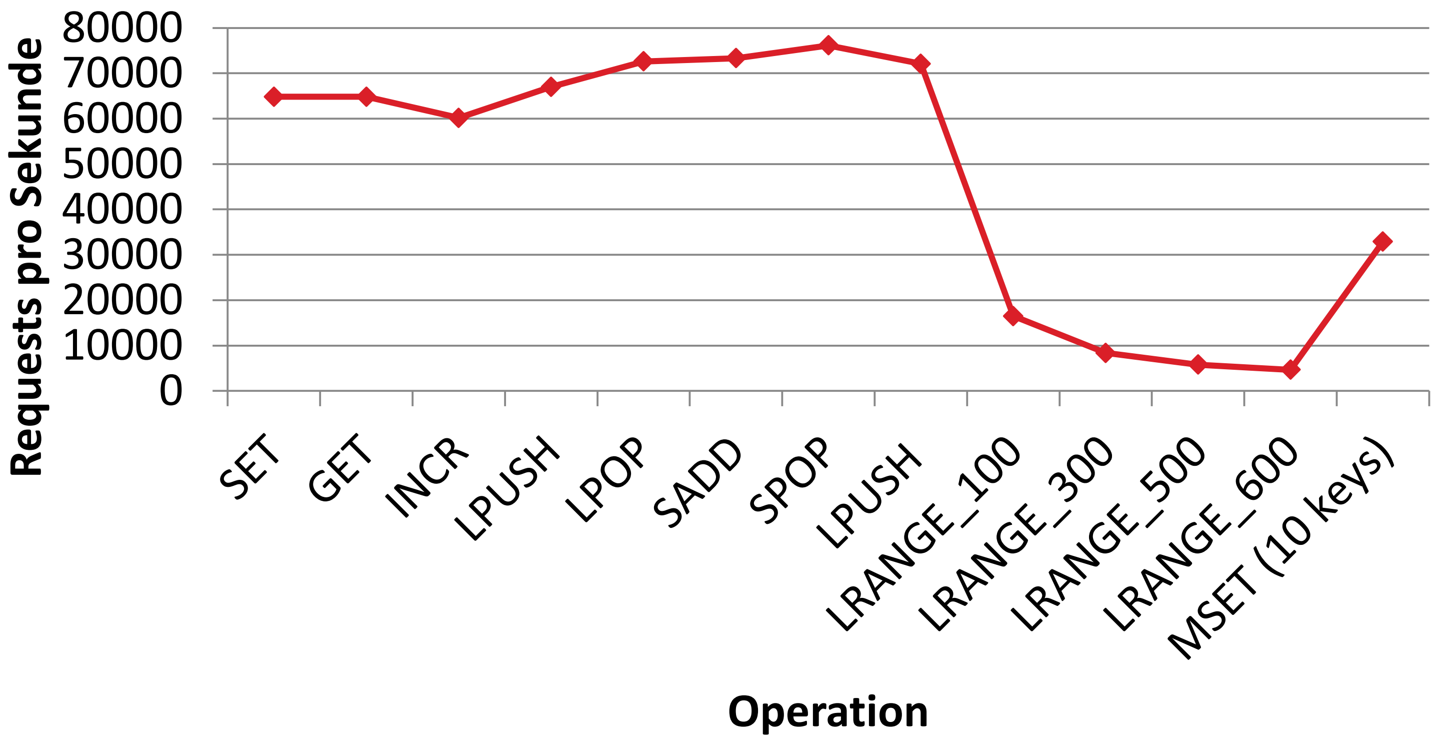


Harizopoulos, Stavros, Madden, Stonebraker "OLTP through the looking glass, and what we found there."

Performance

- Equivalent to Memcache

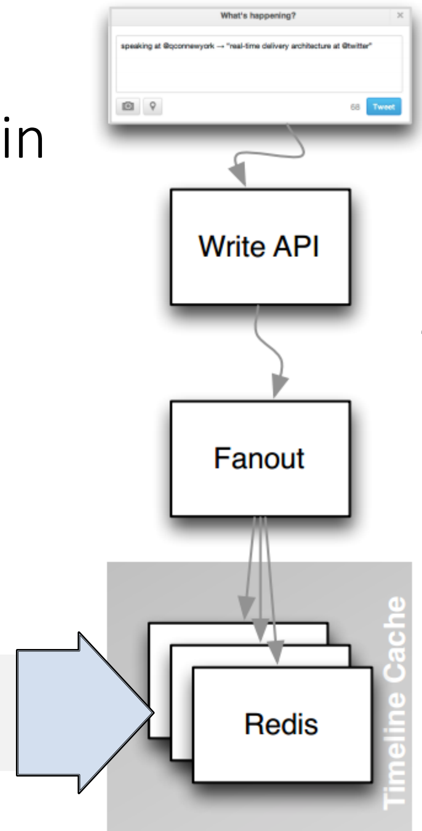
```
> redis-benchmark -n 100000 -c 50
```



Example: Twitter

- Per User: one materialized timeline in Redis
- Timeline = List
- Key: User ID

```
RPUSHX user_id tweet
```



>150 million users
~300k timeline querys/s

http://www.infoq.com/presentations/Real-Time-Delivery-Twitter

Google BigTable (CP)

- Published by Google in 2006
- Original purpose: storing the Google search index

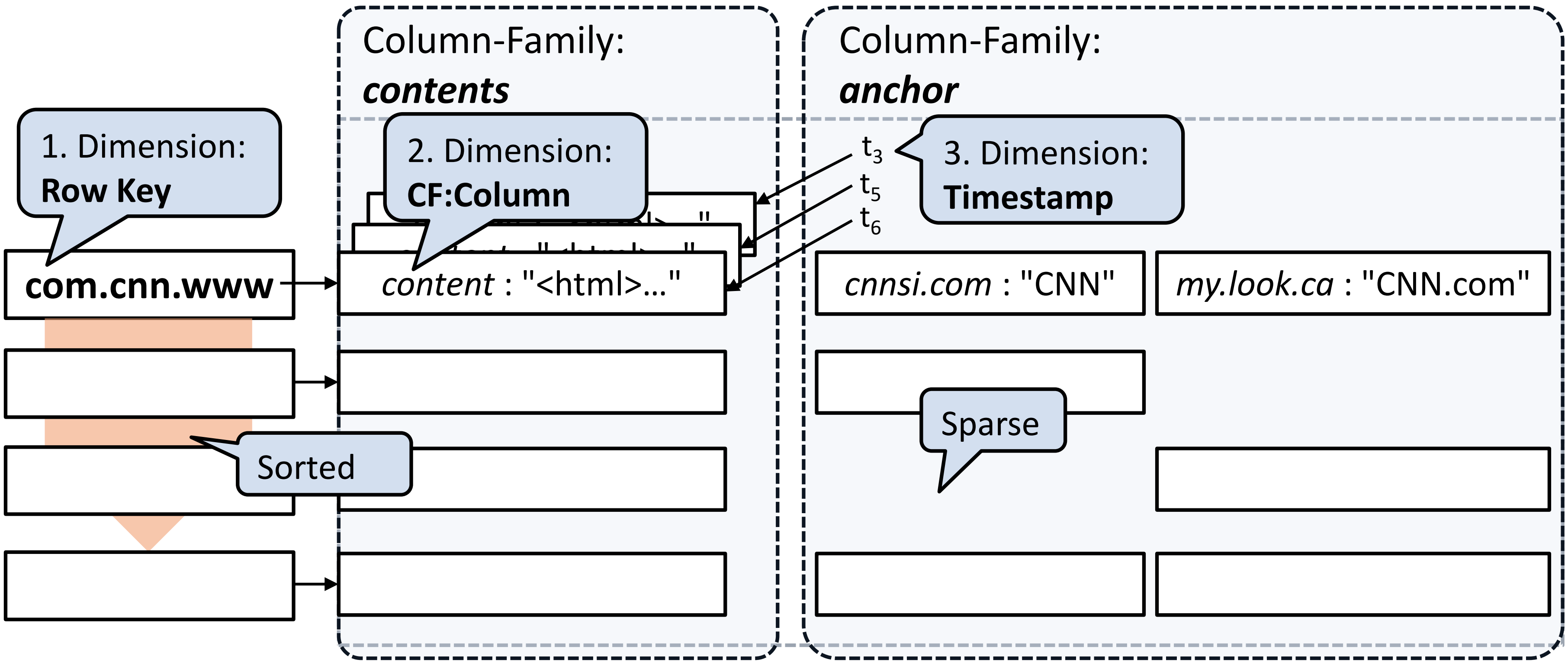
> A Bigtable is a sparse, distributed, persistent multidimensional sorted map.

- Data model also used in: **HBase**, **Cassandra**, **HyperTable, Accumolo**

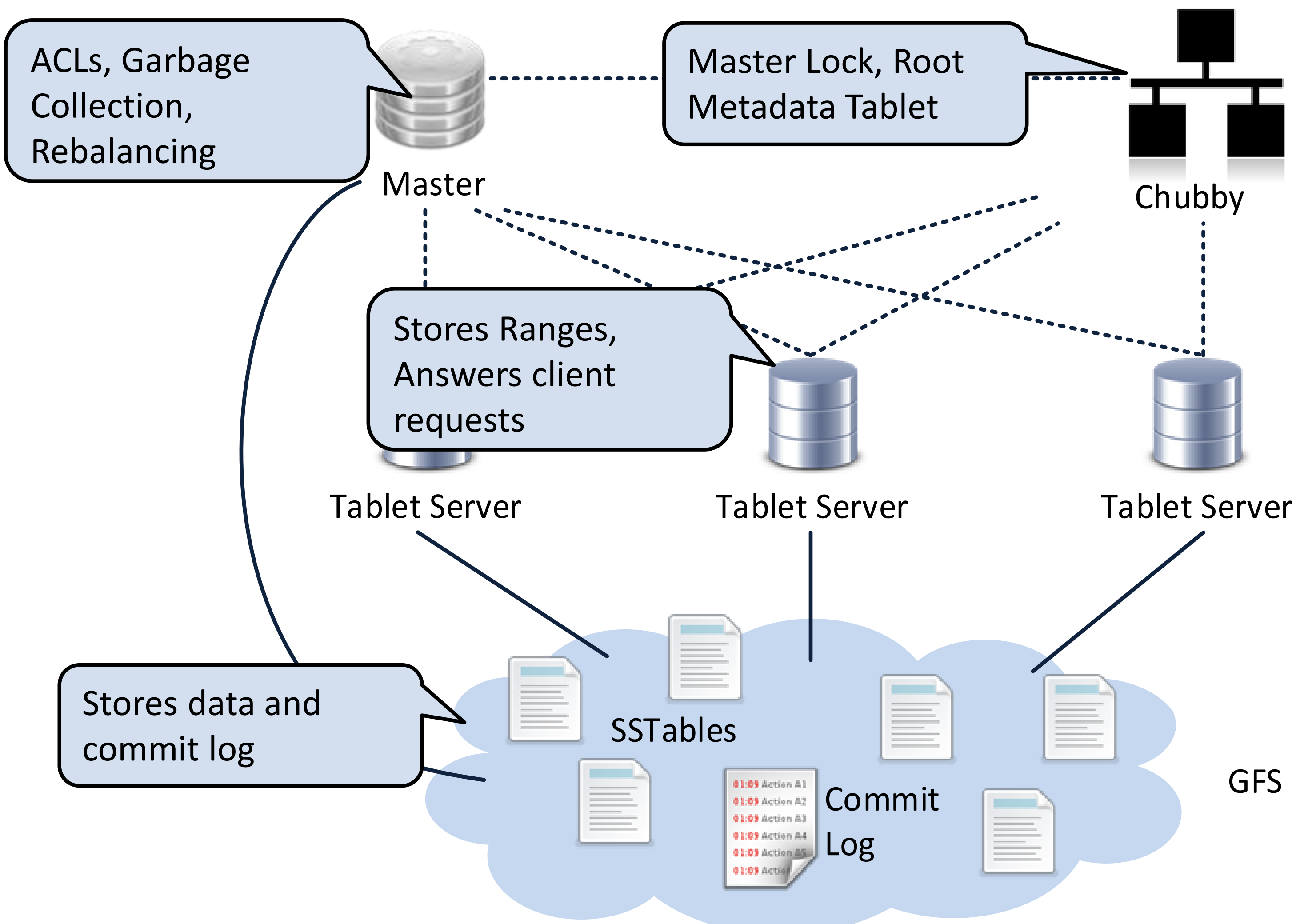
Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."

Wide-Column Data Modelling

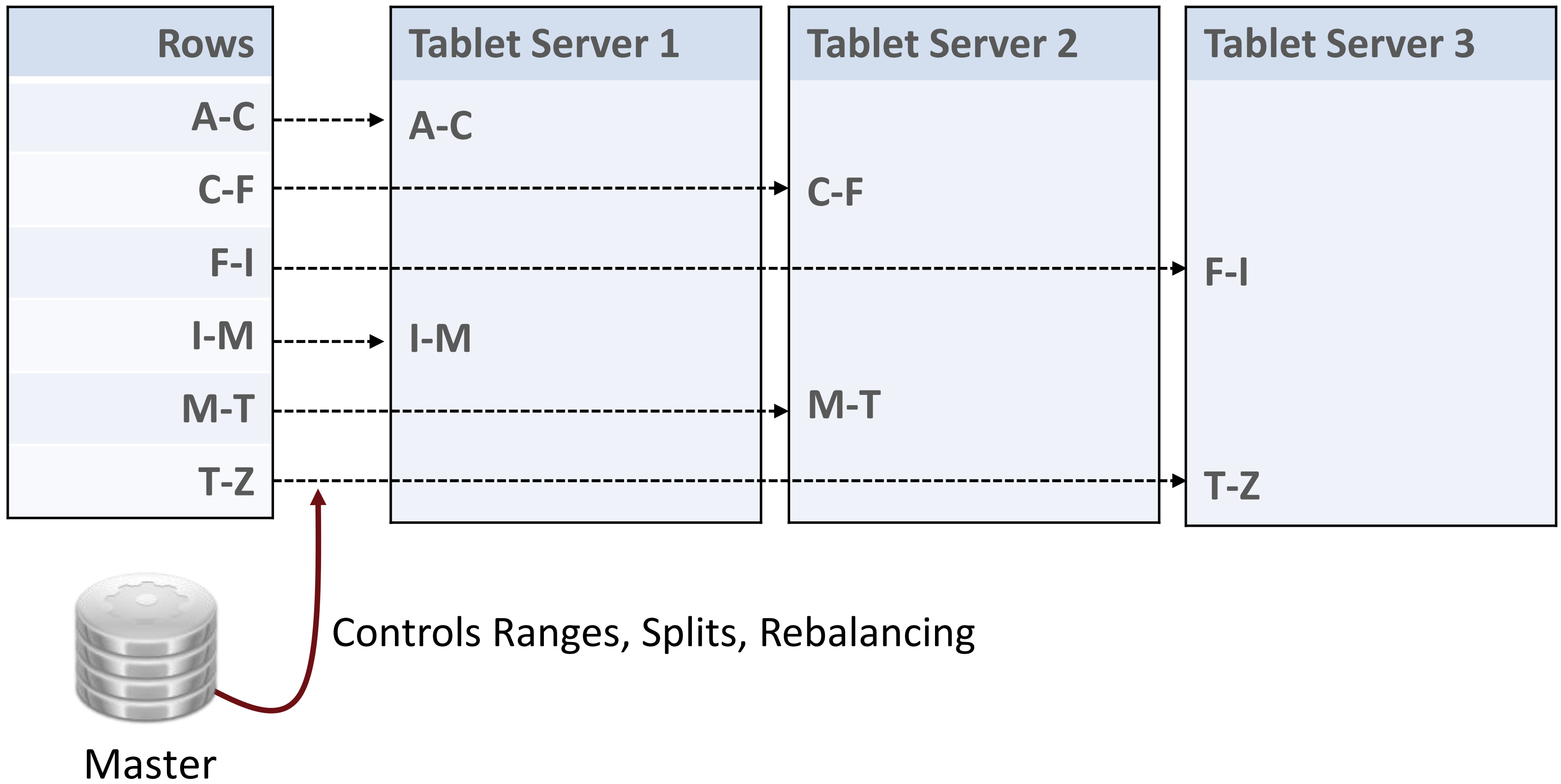
- Storage of crawled web-sites („Webtable“):



Architecture

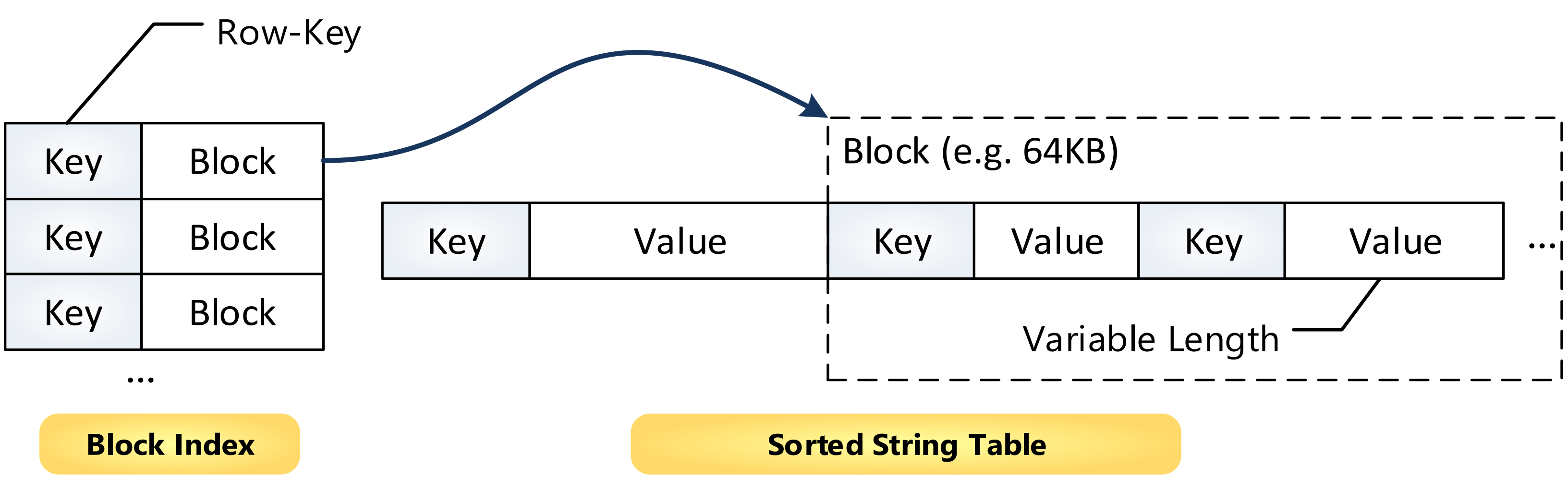


Range-based Sharding



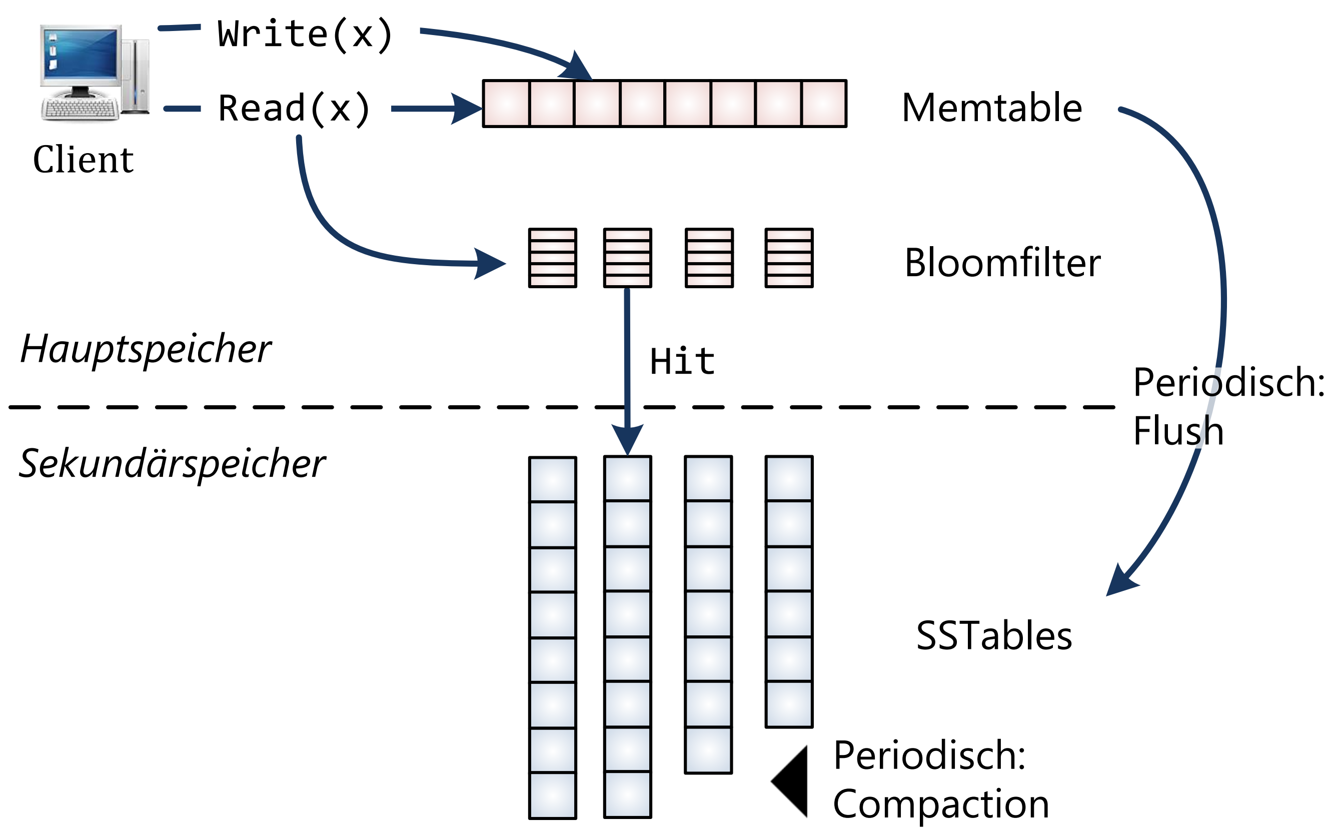
Storage: Sorted-String Tables

- **Goal**: Append-Only IO when writing (no disk seeks)
- Achieved through: **Log-Structured Merge Trees**
- **Writes** go to an in-memory *memtable* that is periodically persisted as an *SSTable* as well as a *commit log*
- **Reads** query memtable and all SSTables



Storage

- Writes: In-Memory in **Memtable**
- Periodisches Ausschreiben der Memtable als SSTable



Apache HBase (CP)

HBase
Model:
Wide-Column
License:
Apache 2
Written in:
Java

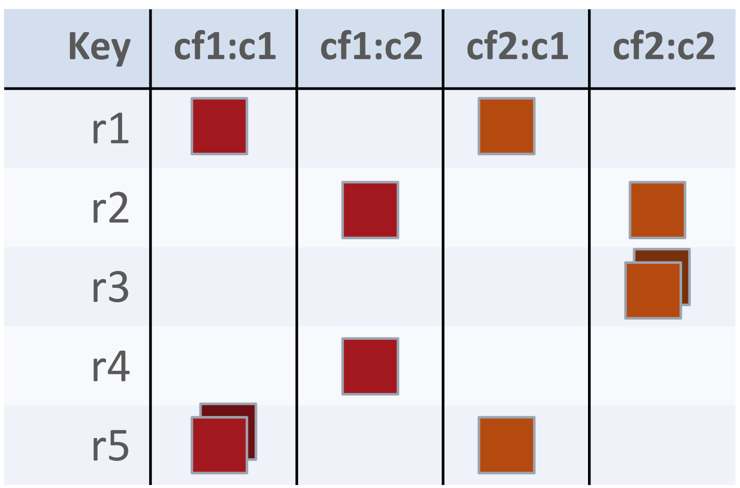
- Open-Source Implementation of BigTable
- Hadoop-Integration
  - Data source for Map-Reduce
  - Uses Zookeeper and HDFS
- Data modelling challenges: key design, tall vs wide
  - **Row Key**: only access key (no indices) → key design important
  - **Tall**: good for scans
  - **Wide**: good for gets, consistent (*single-row atomicity*)
- No typing: application handles serialization
- Interface: REST, Avro, Thrift

HBase Storage

- Logical to physical mapping:

In Value
In Key
In Column

Key Design – where to store data:
r2:cf2:c2:t1:<value>
r2-<value>:cf2:c2:t1:_
r2:cf2:c2<value>:t1:_




```
r1:cf2:c1:t1:<value>
r2:cf2:c2:t1:<value>
r3:cf2:c2:t2:<value>
r3:cf2:c2:t1:<value>
r5:cf2:c1:t1:<value>
```

HFile cf2

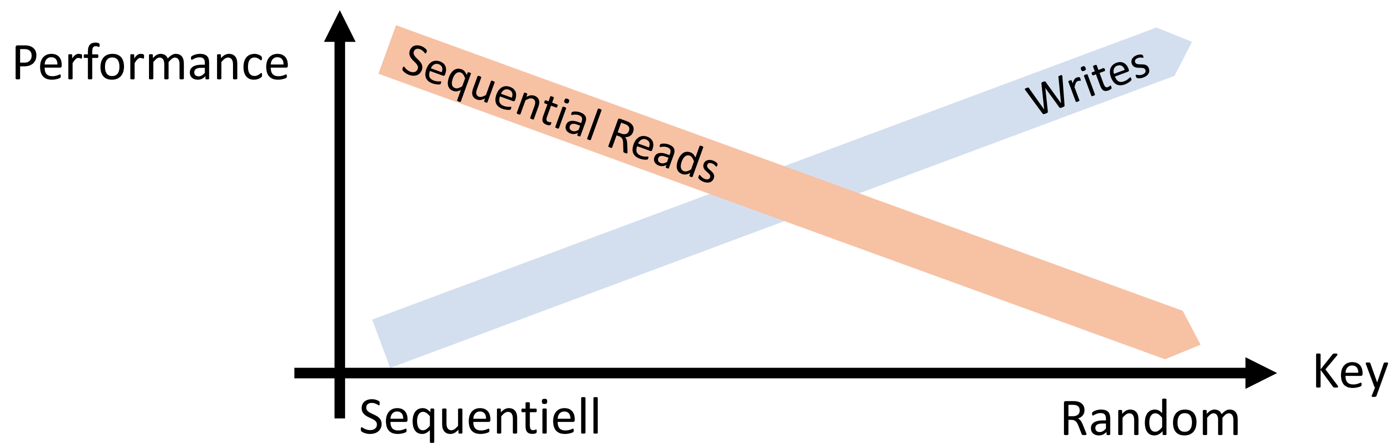
```
r1:cf1:c1:t1:<value>
r2:cf1:c2:t1:<value>
r3:cf1:c2:t1:<value>
r3:cf1:c1:t2:<value>
r5:cf1:c1:t1:<value>
```

HFile cf1

George, Lars. HBase: the definitive guide. 2011.

Schema Design

- Tall vs Wide Rows:
  - **Tall**: good for Scans
  - **Wide**: good for Gets
- Hotspots: Sequential Keys (z.B. Timestamp) dangerous



George, Lars. HBase: the definitive guide. 2011.

Schema: Messages

User ID	CF	Column	Timestamp	Message
12345	data	5fc38314-e290-ae5da5fc375d	1307097848	"Hi Lars, ..."
12345	data	725aae5f-d72e-f90f3f070419	1307099848	"Welcome, and ..."
12345	data	cc6775b3-f249-c6dd2b1a7467	1307101848	"To Whom It ..."
12345	data	dcbee495-6d5e-6ed48124632c	1307103848	"Hi, how are ..."

vs

User ID	CF	Column	Timestamp	Message
12345-5fc38314-e290-ae5da5fc375d	data		: 1307097848	"Hi Lars, ..."
12345-725aae5f-d72e-f90f3f070419	data		: 1307099848	"Welcome, and ..."
12345-cc6775b3-f249-c6dd2b1a7467	data		: 1307101848	"To Whom It ..."
12345-dcbee495-6d5e-6ed48124632c	data		: 1307103848	"Hi, how are ..."

Wide:
Atomicity
Scan over Inbox: **Get**

Tall:
Fast Message Access
Scan over Inbox: **Partial Key Scan**

http://2013.nosql-matters.org/cgn/wp-content/uploads/2013/05/HBase-Schema-Design-NoSQL-Matters-April-2013.pdf

Example: Facebook Insights

MD5(Reversed Domain) + Reversed Domain + URL-ID — Row Key

6PM Total	6PM Male	…	01.01 Total	01.01 Male	…	Total	Male	…
10	7		100	65			567	
CF:Daily			CF:Monthly			CF:All		

Atomic HBase **Counter**

TTL – automatic deletion of old rows

Lars George: "Advanced HBase Schema Design"

API: CRUD + Scan

Cloud Cluster aufsetzen:

```
> elastic-mapreduce --create --
hbase --num-instances 2 --instance-
type m1.large
```

```
> whirr launch-cluster --config
hbase.properties
```

Login, Clustergröße etc.

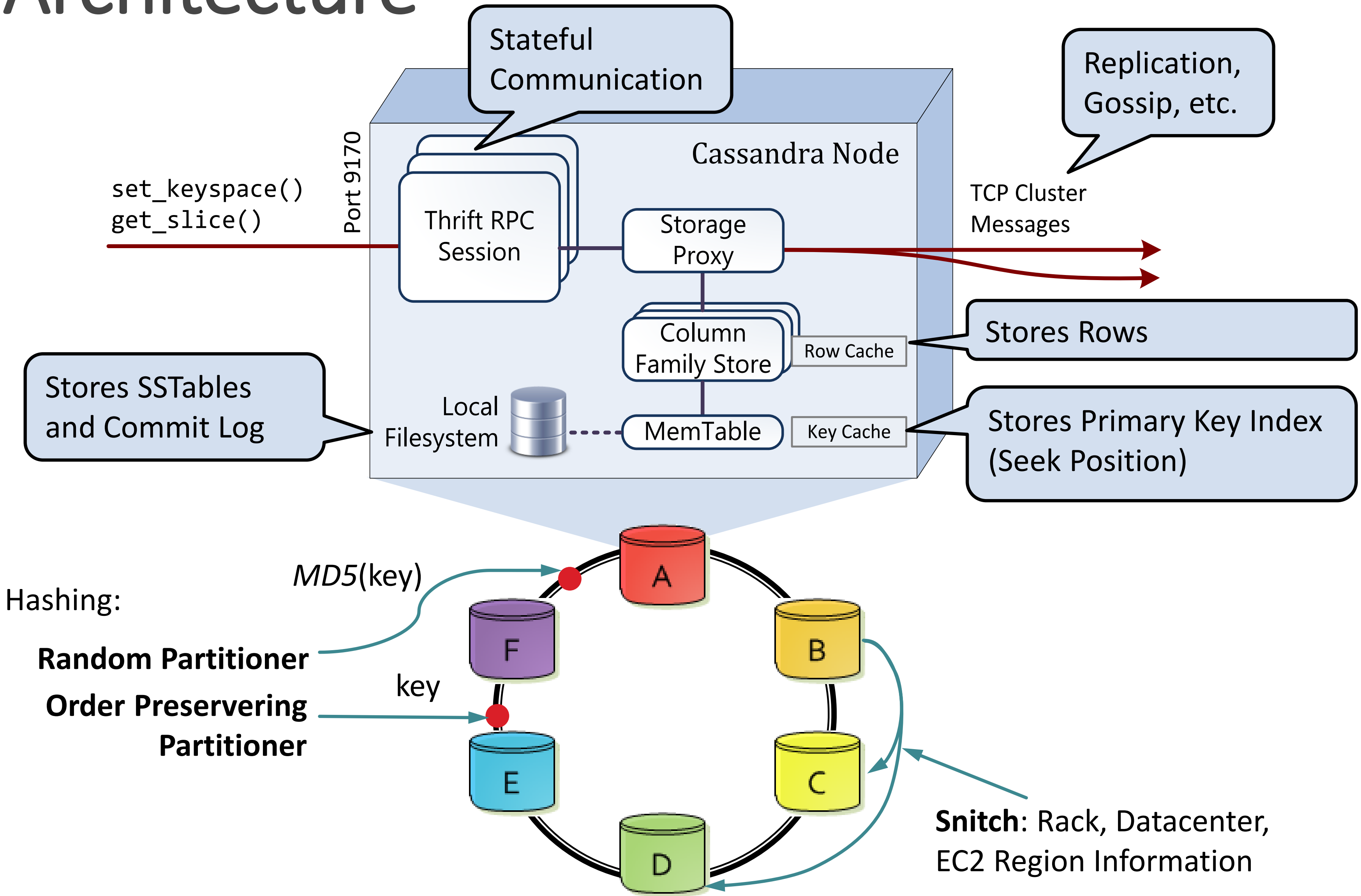
```
HTable table = ...
Get get = new Get("my-row");
get.addColumn(Bytes.toBytes("my-cf"), Bytes.toBytes("my-col"));
Result result = table.get(get);

table.delete(new Delete("my-row"));

Scan scan = new Scan();
scan.setStartRow( Bytes.toBytes("my-row-0"));
scan.setStopRow( Bytes.toBytes("my-row-101"));
ResultScanner scanner = table.getScanner(scan)
for(Result result : scanner) { }
```

Apache Cassandra (AP)

Cassandra
Model:
Wide-Column
License:
Apache 2
Written in:
Java

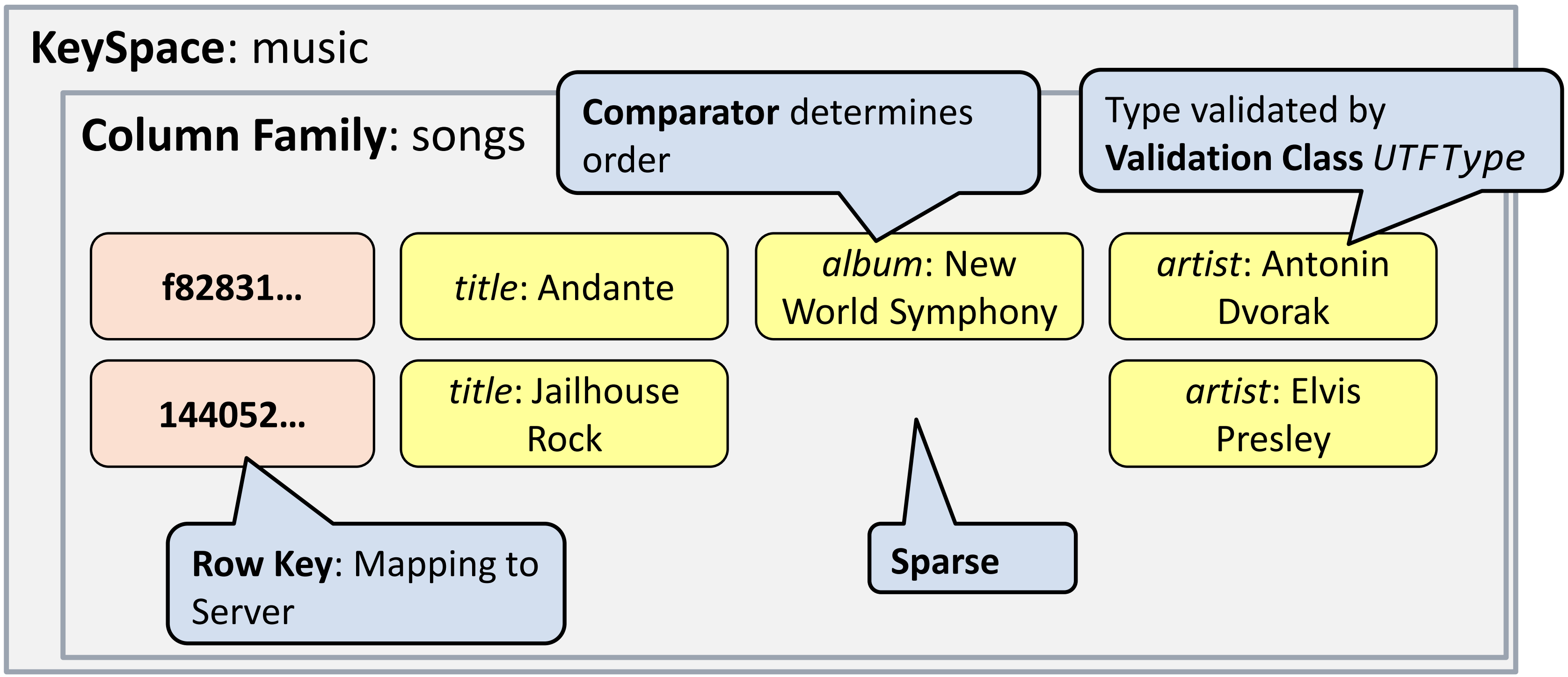
- Published 2007 by Facebook
- **Idea**:
  - BigTable‘s wide-column data model
  - Dynamo ring for replication and sharding
- Cassandra Query Language (CQL): SQL-like query- and DDL-language
- **Compound indices**: *partition key* (shard key) + *clustering key* (ordered per partition key) → Limited range queries
- **Secondary indices**: hidden table with mapping → queries with simple equality condition

Architecture



Storage Layer

- Uses BigTables Column Family Format



http://www.datastax.com/dev/blog/cql3-for-cassandra-experts

CQL Example

- Ermöglicht Scans trotz *Random Partitioner*

```
CREATE TABLE playlists (
  id uuid,
  song_order int,
  song_id uuid, ...
  PRIMARY KEY (id, song_order)
);
```

```
SELECT * FROM playlists
WHERE id = 23423
ORDER BY song_order DESC
LIMIT 50;
```

Partition Key

Clustering Columns:
sorted per node

id	song_order	song_id	artist
23423	1	64563	Elvis
23423	2	f9291	Elvis

Summary: BigTable, HBase, Cassandra

- Data model: $(rowkey, cf: column, timestamp) \rightarrow value$
- **API**: CRUD + Scan(*start-key*, *end-key*)
- Uses distributed file system (GFS/HDFS)
- Storage structure: **Memtable** (in-memory data structure) + **SSTable** (persistent; append-only-IO)
- **Schema design**: only primary key access → implicit schema (key design) needs to be carefully planned
- **HBase**: very literal open-source implementation BigTable
- **Cassandra**: combination of Dynamo and BigTable ideas

MongoDB (CP)

- From hu**mongo**us ≅ gigantic
- Tunable consistency
- Schema-free document database
- Allows complex queries and indexing
- **Sharding** (either range- or hash-based)
- **Replication** (either synchronous or asynchronous)
- Storage Management:
  - **Write-ahead logging** for redos (*journaling*)
  - **Memory-mapped** storage files, buffer management handled by operating system (paging)

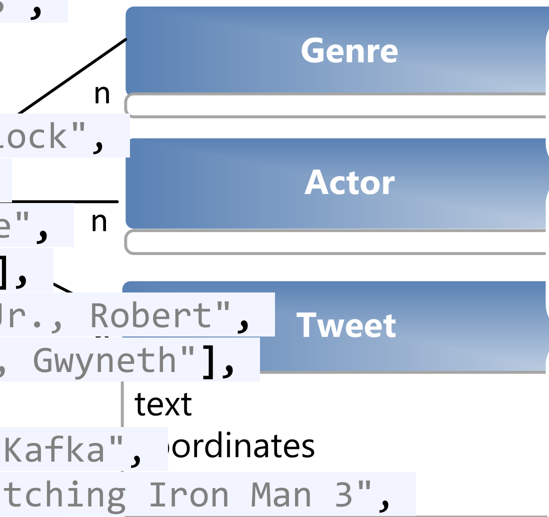
mongoDB

MongoDB
Model:
Document
License:
GNU AGPL 3.0
Written in:
C++

Data Modelling

```
{
        "_id" : ObjectId("51a5d316d70beffe74ecc940")
        title : "Iron Man 3",
        year : 2013,
        rating : 7.6,
        director: "Shane Block",
        genre : [ "Action",
                  "Adventure",
                  "Sci -Fi"],
        actors : ["Downey Jr., Robert",
                  "Paltrow , Gwyneth"],
        tweets : [ {
           "user" : "Franz Kafka",
           "text" : "#nowwatching Iron Man 3",
           "retweet" : false,
           "date" : ISODate("2013-05-29T13:15:51Z")
        }]
}
```

Movie Document



Denormalisation instead of joins

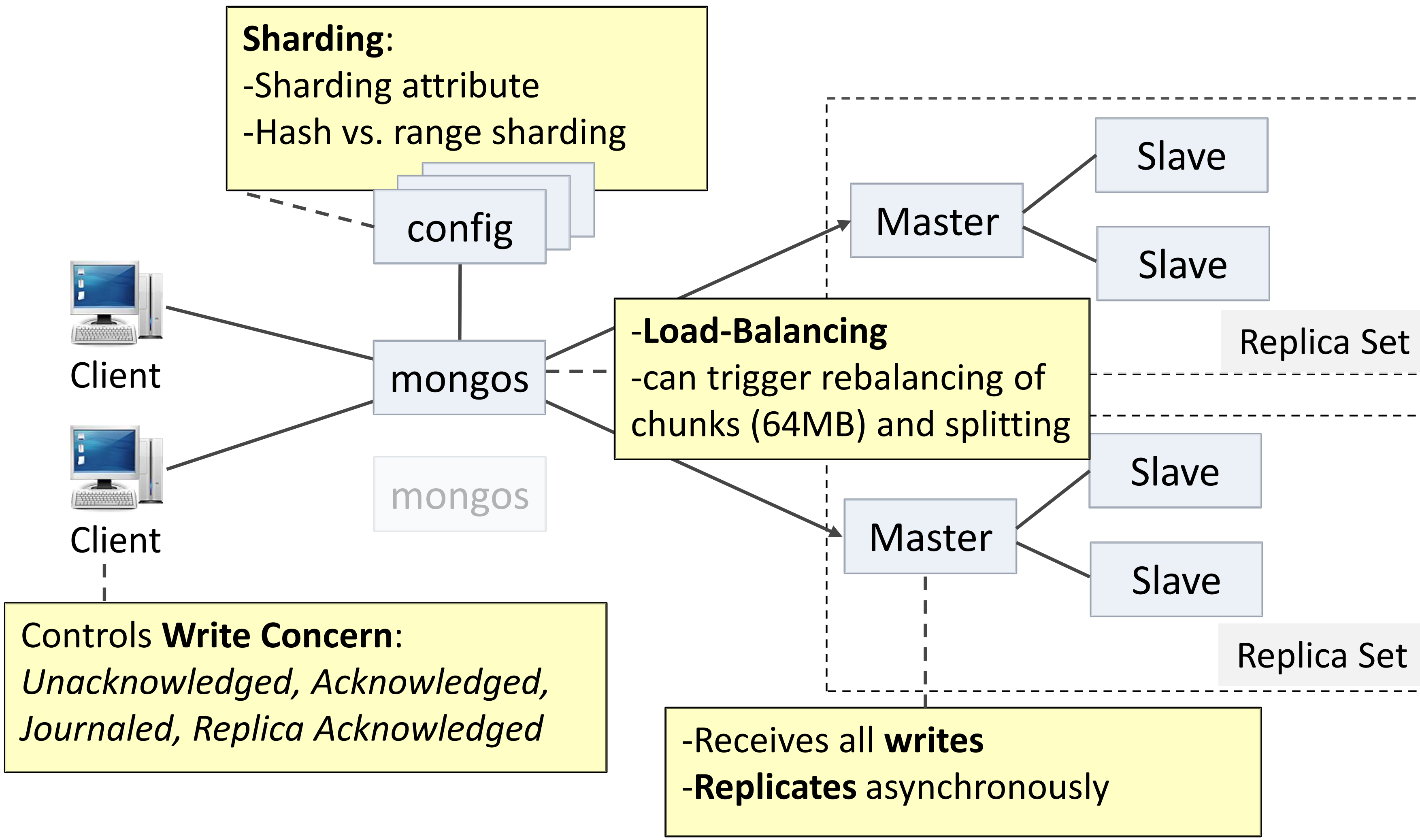
Nesting replaces 1:n and 1:1 relations

Schemafreeness: Attributes per document

Unit of atomicity: document

Principles

Sharding und Replication



MongoDB by Example

Movie mApp - DIS Task

DIS Website | MongoDB Tutorials

The Movie mApp

Unveiling the geographic patterns underlying tweets about movies.

Show Mongo at http://127.0.0.1:28017/

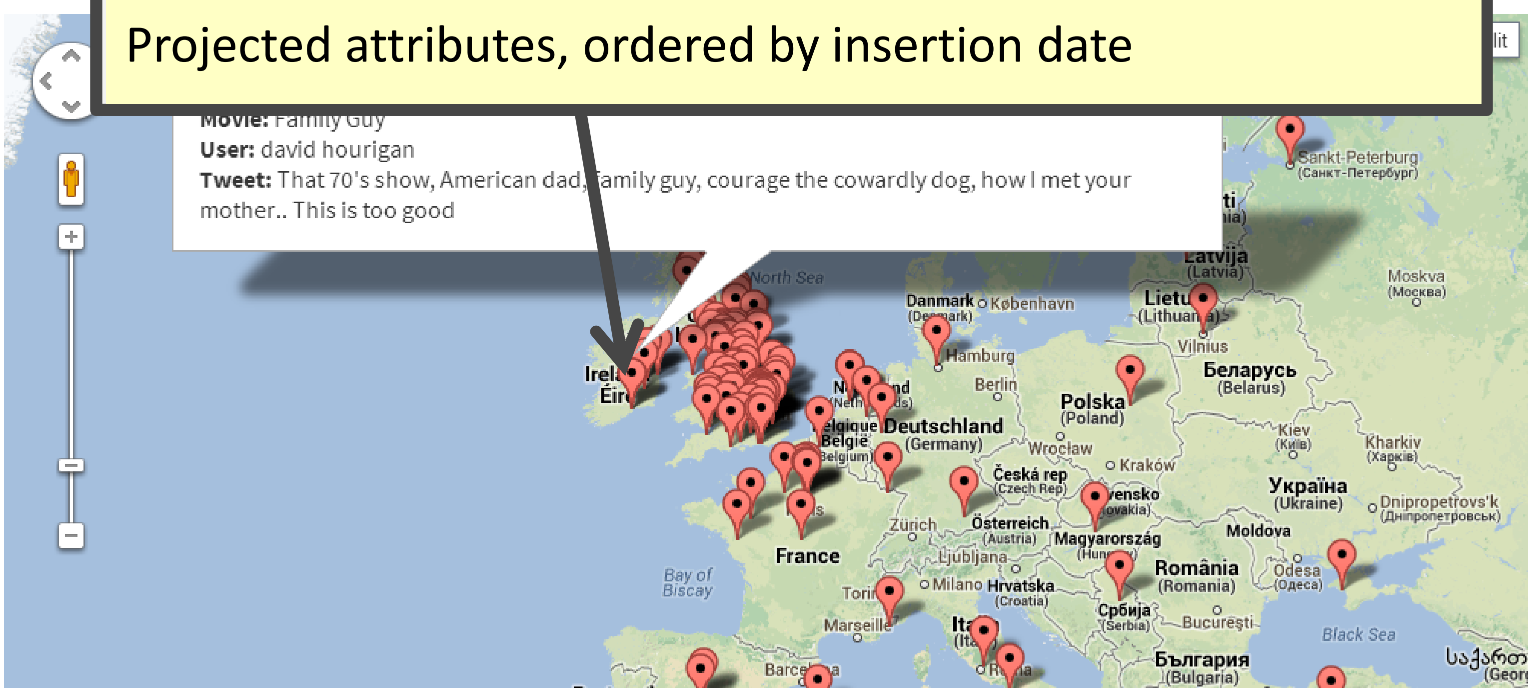
Map

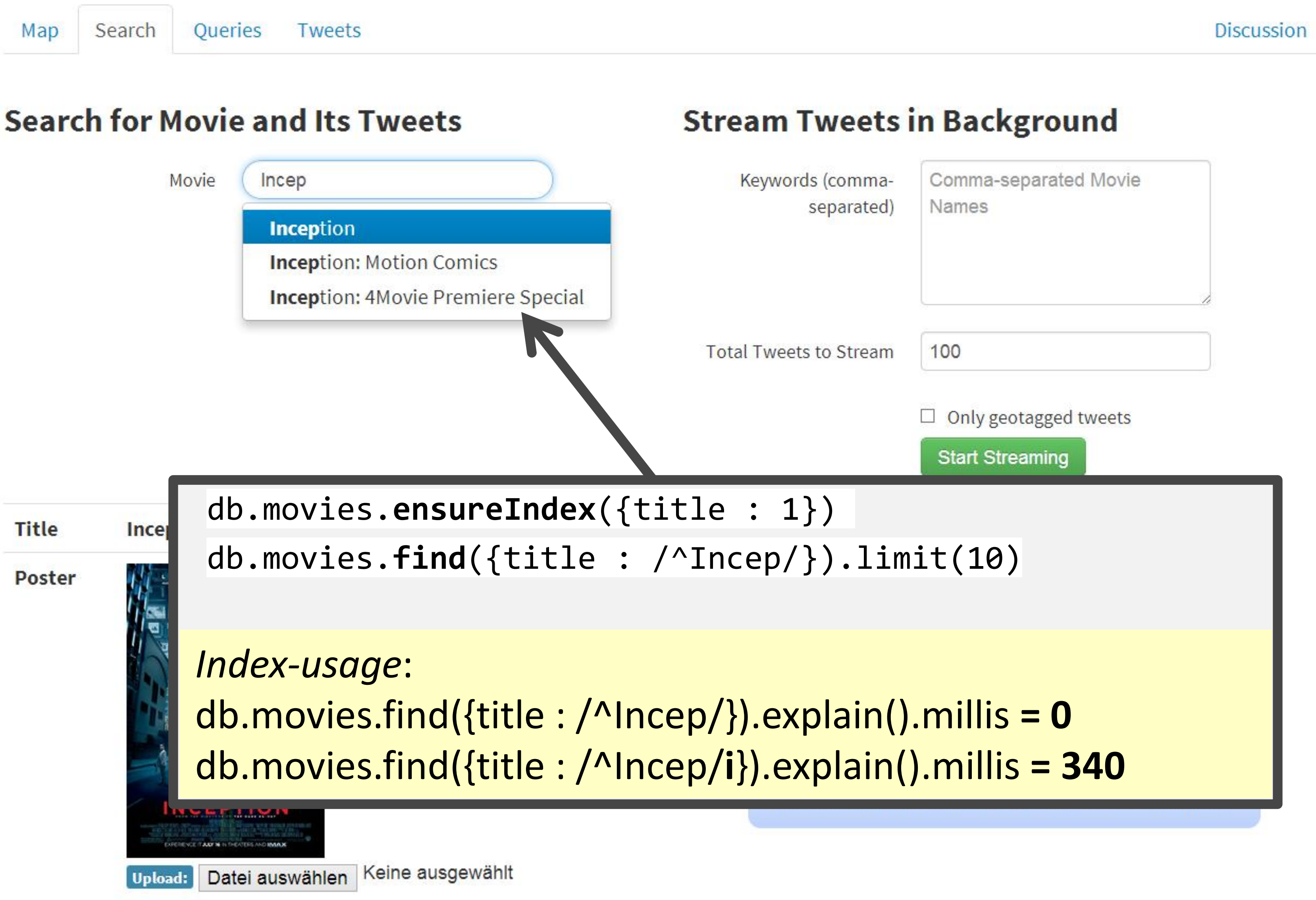
Geot

Note:

```
db.tweets.find({coordinates : {"$exists" : 1}},
  {text:1, movie:1, "user.name":1, coordinates:1})
  .sort({id:-1})
```

Projected attributes, ordered by insertion date



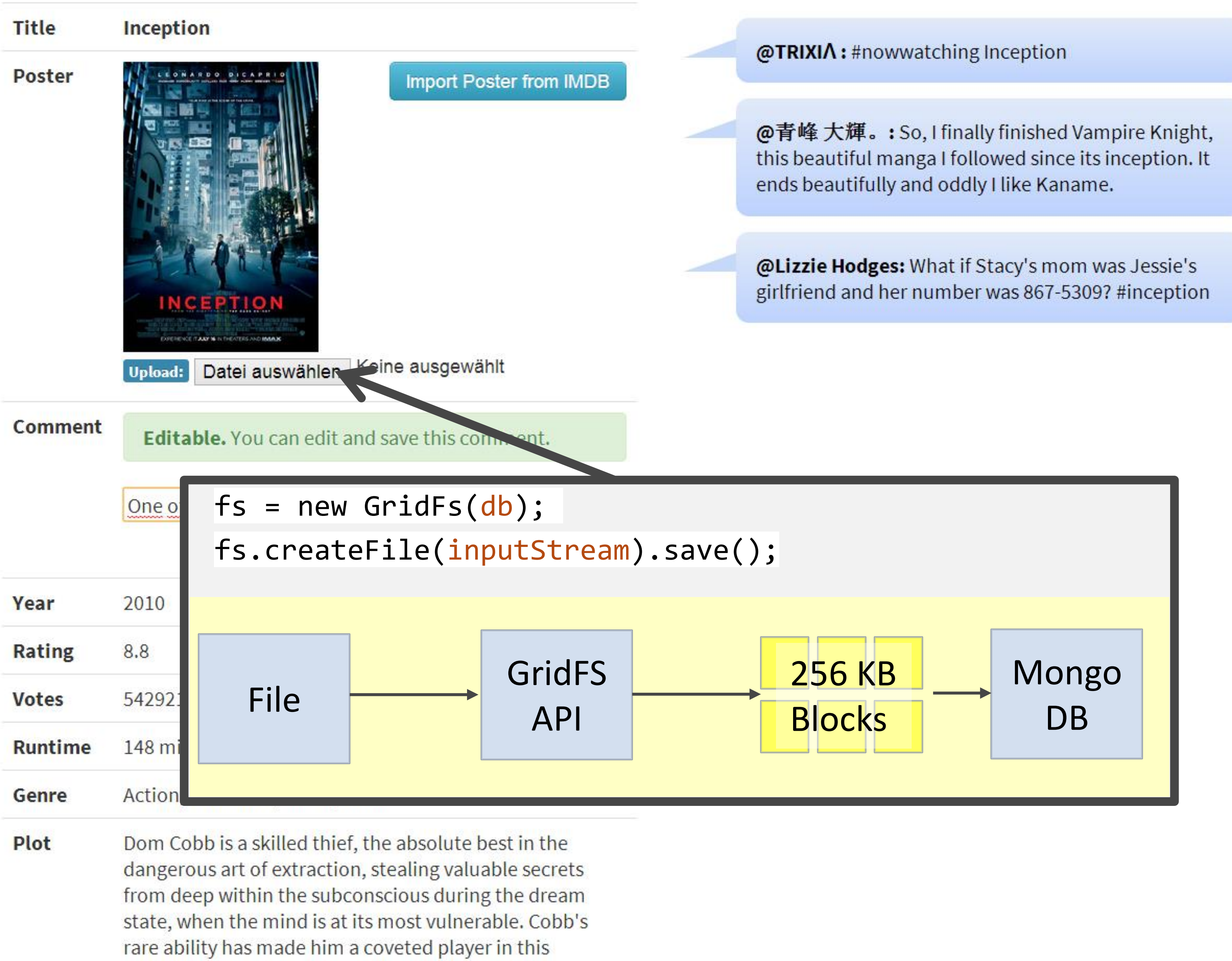



```
db.movies.ensureIndex({title : 1})
db.movies.find({title : /^Incep/}).limit(10)
```

Index-usage:

db.movies.find({title : /^Incep/}).explain().millis **= 0**

db.movies.find({title : /^Incep/**i**}).explain().millis **= 340**


Title Inception
Poster
INCEPTION
Import Poster from IMDB
Upload: Datei auswählen Keine ausgewählt
Comment
Editable. You can edit and save this comment.
One o
Year 2010
Rating 8.8
Votes 54292
Runtime 148 mi
Genre Action
Plot Dom Cobb is a skilled thief, the absolute best in the dangerous art of extraction, stealing valuable secrets from deep within the subconscious during the dream state, when the mind is at its most vulnerable. Cobb's rare ability has made him a coveted player in this
@TRIXIΛ : #nowwatching Inception
@青峰 大輝。: So, I finally finished Vampire Knight, this beautiful manga I followed since its inception. It ends beautifully and oddly I like Kaname.
@Lizzie Hodges: What if Stacy's mom was Jessie's girlfriend and her number was 867-5309? #inception
fs = new GridFs(db);
fs.createFile(inputStream).save();
File
GridFS API
256 KB Blocks
Mongo DB

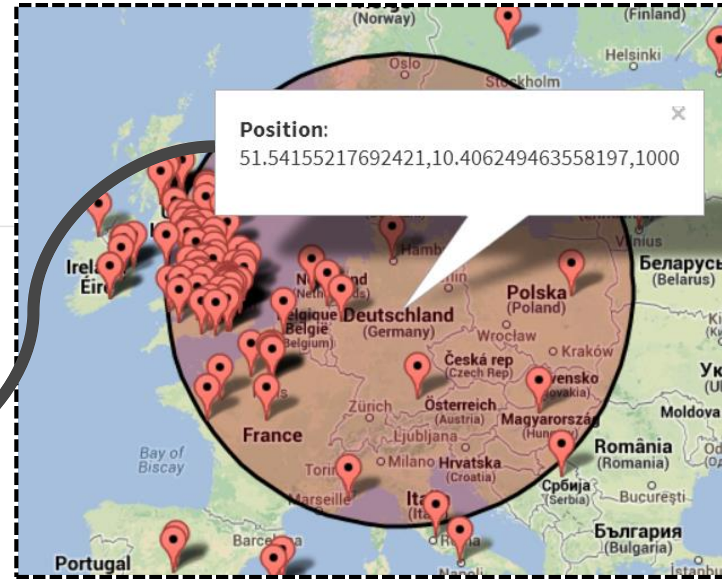
Map Search Queries Tweets

Query Tweets

Query: Get Tweets Near: lat,lng,radius-in-km [Go]

Parameter: 51.54155217692421,10.406249463558197,1000

Result Limit: 10



User	Tweet	Created at	Coordinates
MitchellyMonica			
J. Z.			
Party Hardy			
nadine stachowiak		2013	
PAIRSonnalites | DE	ES-News : http://t.co/WhBmNr6OM2: In vielen Staaten Afrikas gibt es	Wed May 29	13.46757813,52.5913198

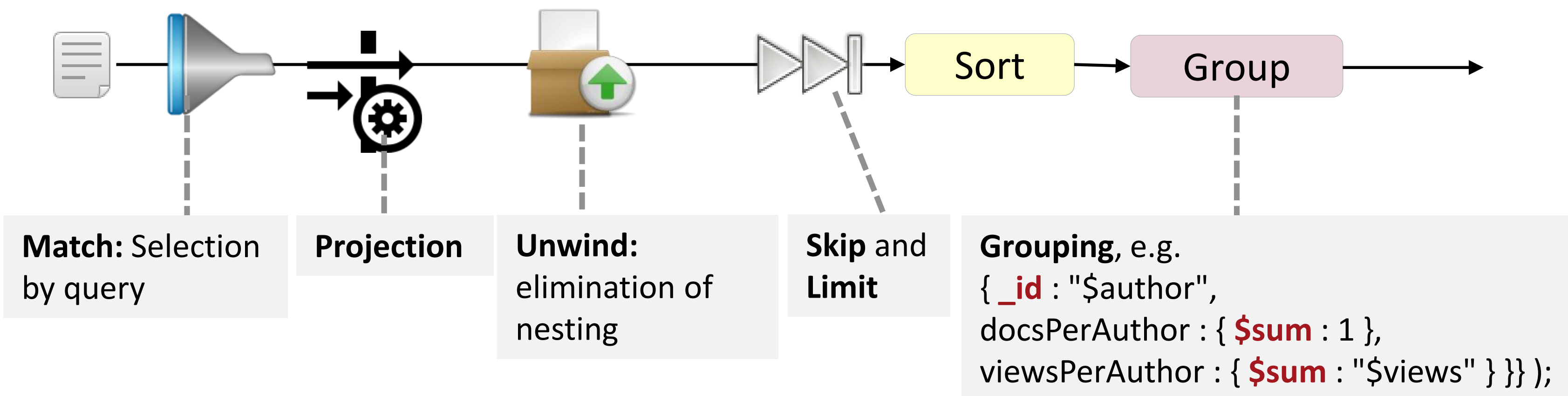
```
db.tweets.ensureIndex({coordinates : "2dsphere"})
db.tweets.find({"$near" : {"$geometry" : … }})
```

Geospatial Queries:

- **Distance**
- **Intersection**
- **Inclusion**

Analytic Capabilities

- Aggregation Pipeline Framework:

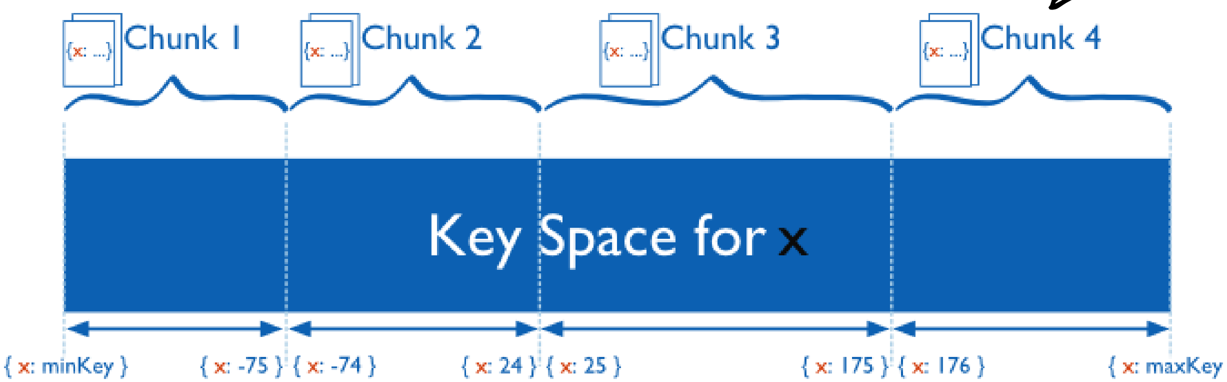


- Alternative: JavaScript MapReduce

Sharding

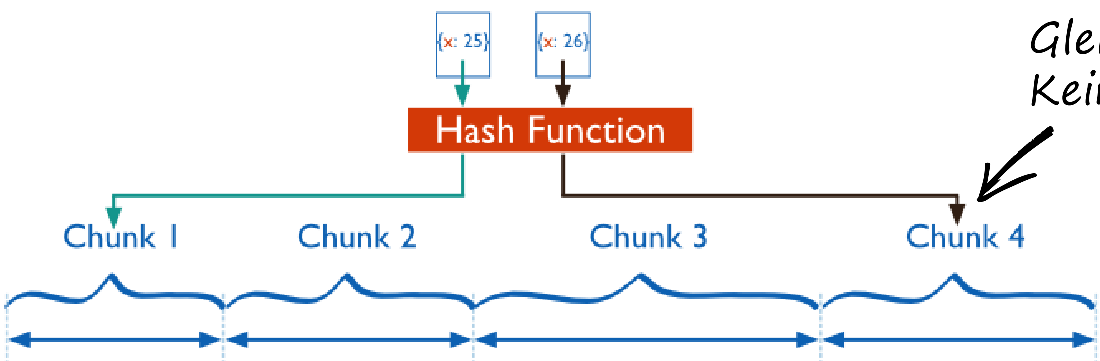
- Range-basiert:

Im Optimalfall nur ein
angefragter Server pro Query
Sonst: Scatter-and-gather



- Hash-basiert:

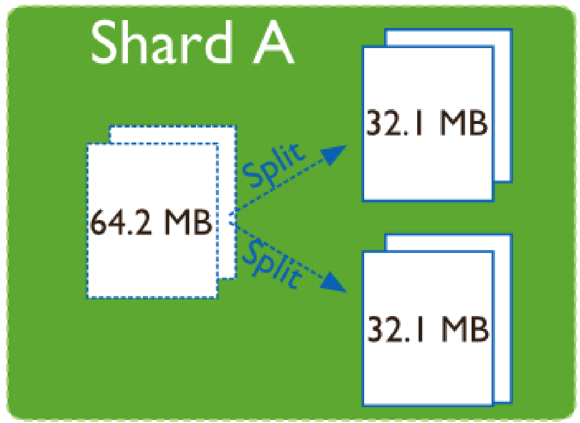
Gleichmäßige Verteilung
Keine Lokalität



docs.mongodb.org/manual/core/sharding-introduction/

Sharding

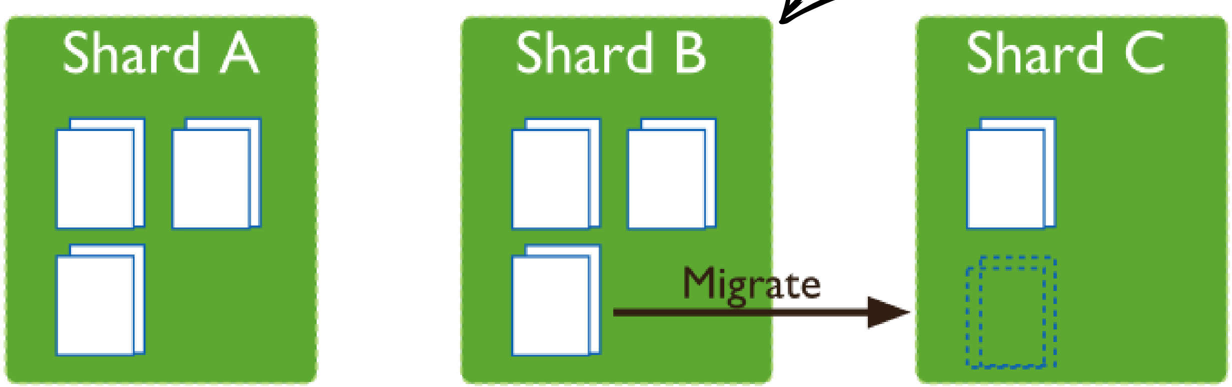
- Splitting:



Zu große Chunks in zwei Chunks teilen

- Migration:

Mongos Load Balancer triggert Rebalancierung



docs.mongodb.org/manual/core/sharding-introduction/

Other Systems

Graph databases

- **Neo4j** (ACID, replicated, Query-language)
- **HypergraphDB** (directed Hypergraph, BerkleyDB-based)
- **Titan** (distributed, Cassandra-based)
- **ArangoDB, OrientDB** („multi-model“, single-node)
- **SparkleDB** (RDF-Store, SPARQL)
- **InfinityDB** (embeddable)
- **InfiniteGraph** (distributed, low-level API, Objectivity-based)

Eric Redmond: “NoSQL is dead”

Other Systems

Key-Value Stores

- **Aerospike** (SSD-optimized)
- **Voldemort** (Dynamo-style)
- **Memcache** (in-memory cache)
- **LevelDB** (embeddable, LSM-based)
- **HyperDex** (Searchable, Hyperspace-Hashing, Transactions)
- **Oracle NoSQL database** (distributed frontend for BerkleyDB)
- **HazelCast** (in-memory data-grid based on Java Collections)
- **FoundationDB** (ACID through Paxos)

Eric Redmond: “NoSQL is dead”

Other Systems

Document Stores

- **CouchDB** (Multi-Master, lazy synchronization)
- **CouchBase** (persistent, distributed Memcache, MR-Views)
- **RavenDB** (single node, SI transactions)
- **RethinkDB** (distributed CP, MVCC, joins, aggregations)
- **MarkLogic** (XML, distributed 2PC-ACID)
- **ElasticSearch** (full-text search, scalable, unclear consistency)
- **Solr** (full-text search)
- **Azure DocumentDB** (cloud-only, ACID, WAS-based)

Eric Redmond: “NoSQL is dead”

Other Systems
Wide-Column Stores

- Accumolo (BigTable-style, cell-level security)
- HyperTable (BigTable-style, written in C++)

Eric Redmond: “NoSQL is dead”

Summary

- **HDFS and Hadoop**: Map-Reduce platform for batch analytics
- **Spark, Kafka, Storm**: In-Memory & Real-Time Analytics
- **Dynamo and Riak**: KV-store with consistent hashing
- **Redis**: replicated, in-memory KV-store
- **BigTable, HBase, Cassandra**: wide-column stores
- **MongoDB**: sharded and replicated document store

Outline

Foundations: Big Data, Scalability, Avaialbility

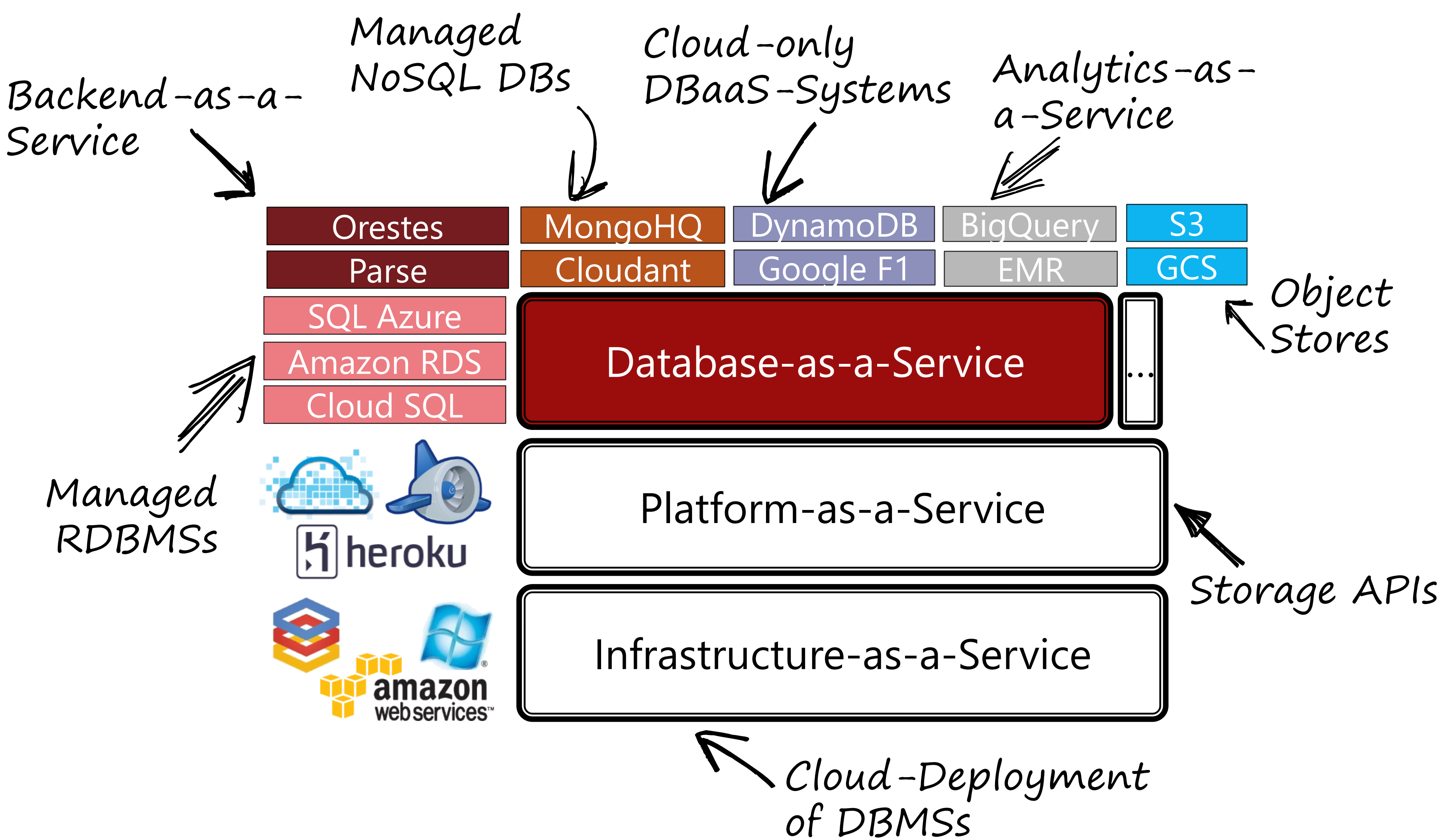
The 4 Classes of NoSQL Databases

NoSQL Examples: concrete Architectures, Systems, APIs

Cloud Databases

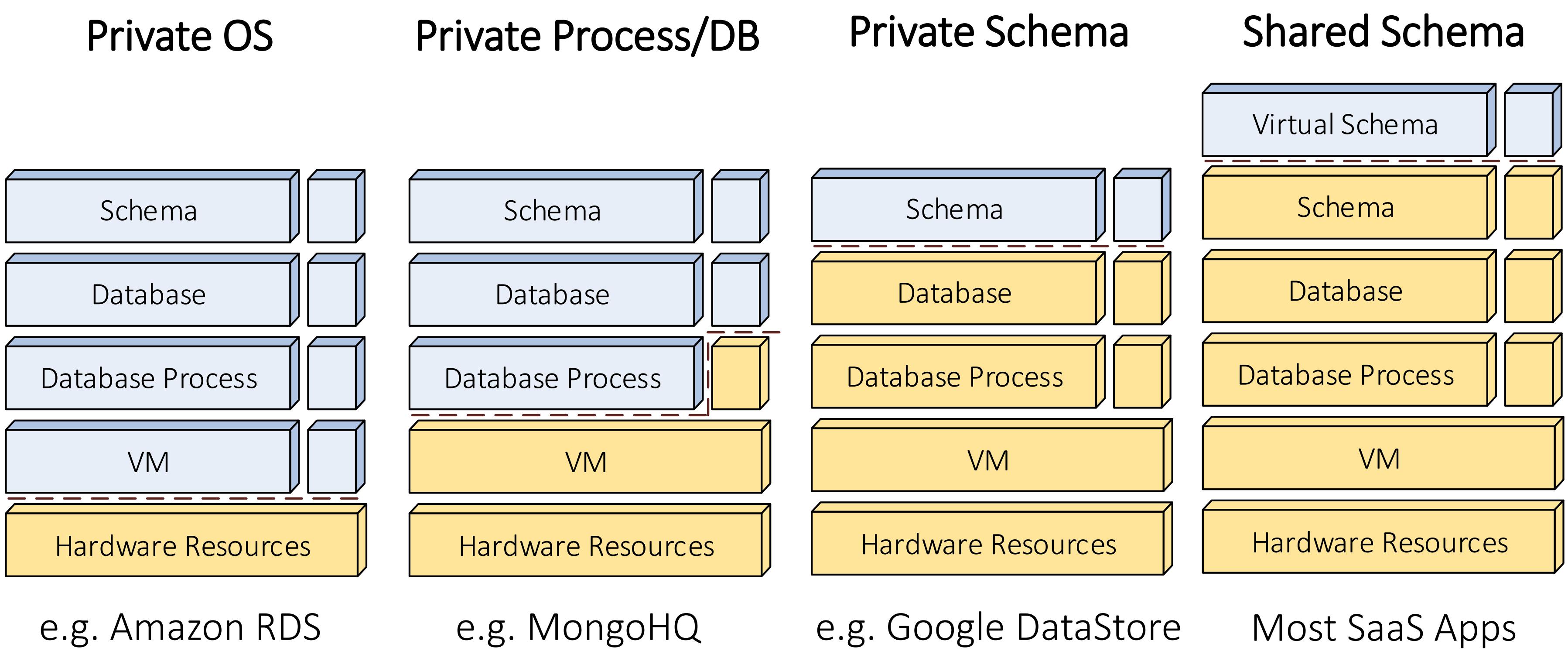
- Database-as-a-Service
- Backend-as-a-Service
- Research Challenges

Cloud Databases



Database-as-a-Service

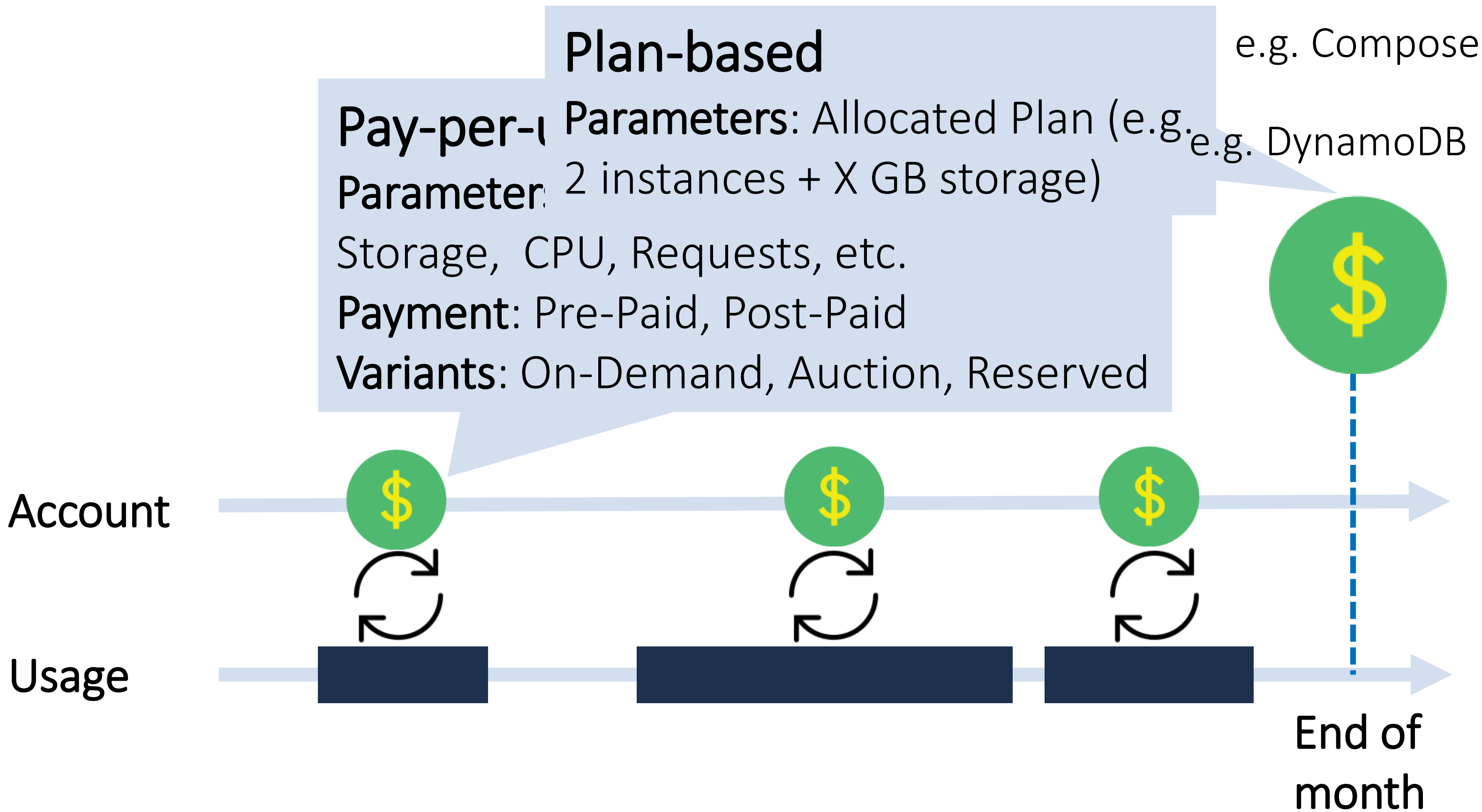
Multi-Tenancy



T. Kiefer, W. Lehner "Private table database virtualization for dbaas"
UCC, 2011

Database-as-a-Service

Pricing Models



Free Tier: free plan or free initial account credit

Requirements

Functional Requirements

- Scan-Querys
- Transactions
- Conditional Updates
- Joins
- Query by Example
- alytics

Non-Functional Requirements

- Scalability of Data Volume
- Write Scalability
- Read Scalability
- Elasticity
- Read-Availability
- Consistency
- Write-Availability
- Durability
- Read-Latency
- Write-Throughput
- Write-Latency

mongoDB aaS

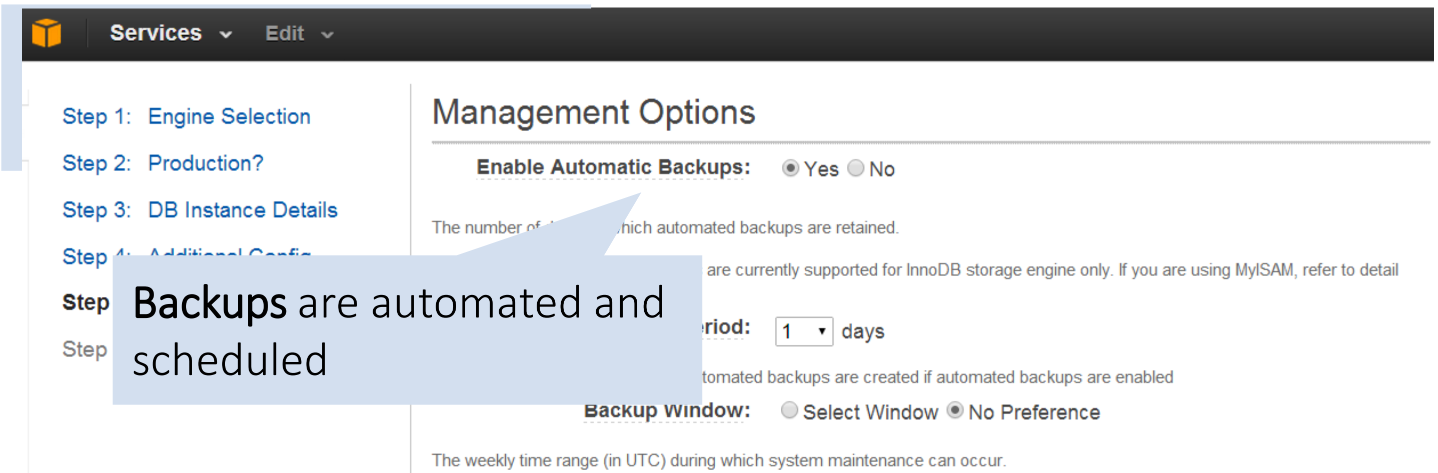
Questions to ask:

- Which requirements are met by the DB?
- Which are met by the provider? → SLAs

DBaaS Example

Amazon RDS

- Relational Database Service



Backups are automated and scheduled

Amazon RDS

RDS
Model:
Managed RDBMS
Pricing:
Instance + Volume + License
Underlying DB:
MySQL, Postgres, MSSQL, Oracle
API:
DB-specific

- Support for (asynchronous) Read Replicas
- **Administration**: Web-based or SDKs
- Only RDBMSs
- “Analytic Brother“ of RDS: RedShift (PDWH)

DBaaS Example

Azure Tables

REST API

No Index: Lookup only (!) by full table scan

Atomic "Entity-Group Batch Transaction" possible

Partition Key	Row Key *(sortiert)*	Timestamp *(autom.)*	Property1	P
intro.pdf	v1.1	14/6/2013	…	…
intro.pdf	v1.2	15/6/2013		…
p		11/6/2013	…	

Partition

Partition

Hash-distributed to parition servers

Sparse

- Similar to Amazon SimpleDB and DynamoDB

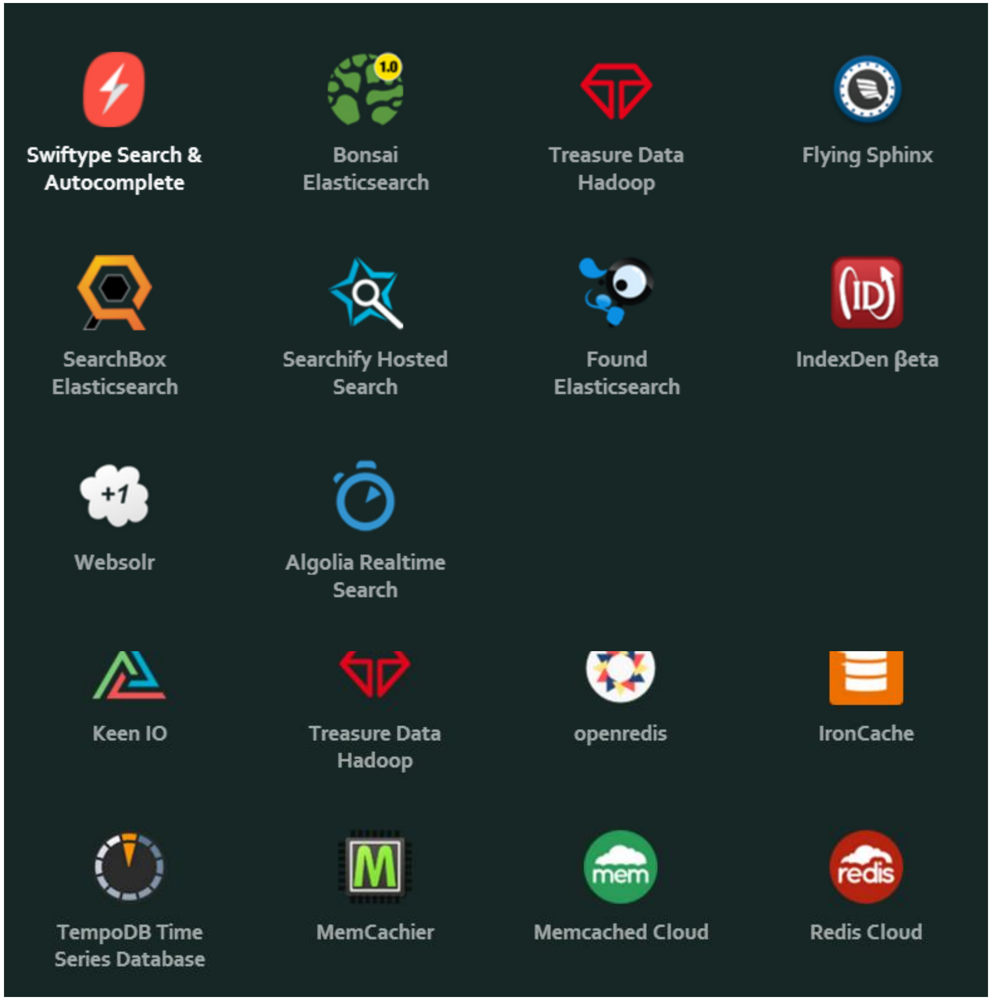
SimpleDB:
- Indexes all attributes
- Rich(er) queries
- Many Limits (size, RPS, etc.)

DynamoDB:
- Provisioned Throughput
- On SSDs („single digit latency“)
- Optional Indexes

DBaaS and PaaS Example

Heroku Addons

- Many Hosted NoSQL DbaaS Providers represented
- And Search



DBaaS and PaaS Example

Heroku Addons

Create Heroku App:

```
$ heroku create
```

Add Redis2Go Addon:

```
$ heroku addons:add redistogo
-----> Adding RedisToGo to fat-unicorn-1337... done, v18 (free)
```

Use Connection URL (environment variable):

```
uri = URI.parse(ENV["REDISTOGO_URL"])
REDIS = Redis.new(:url => ENV['REDISTOGO_URL'])
```

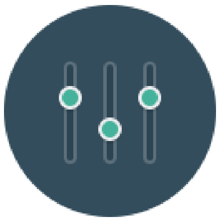
Redis2Go
Model:
Managed NoSQL
Pricing:
Plan-based
Underlying DB:
Redis
API:
Redis

- **Very simple**
- Only suited for small to medium applications (no SLAs, limited control)

Cloud-Deployed DB

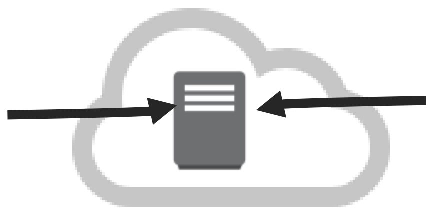
An alternative to DBaaS-Systems

- **Idea**: Run (mostly) unmodified DB on IaaS



- Method I: DIY

1. **Provision** VM(s)



2. **Install** DBMS (manual, script, Chef, Puppet)

- Method II: Deployment Tools

```
> whirr launch-cluster --config
hbase.properties
```

Login, cluster-size etc.

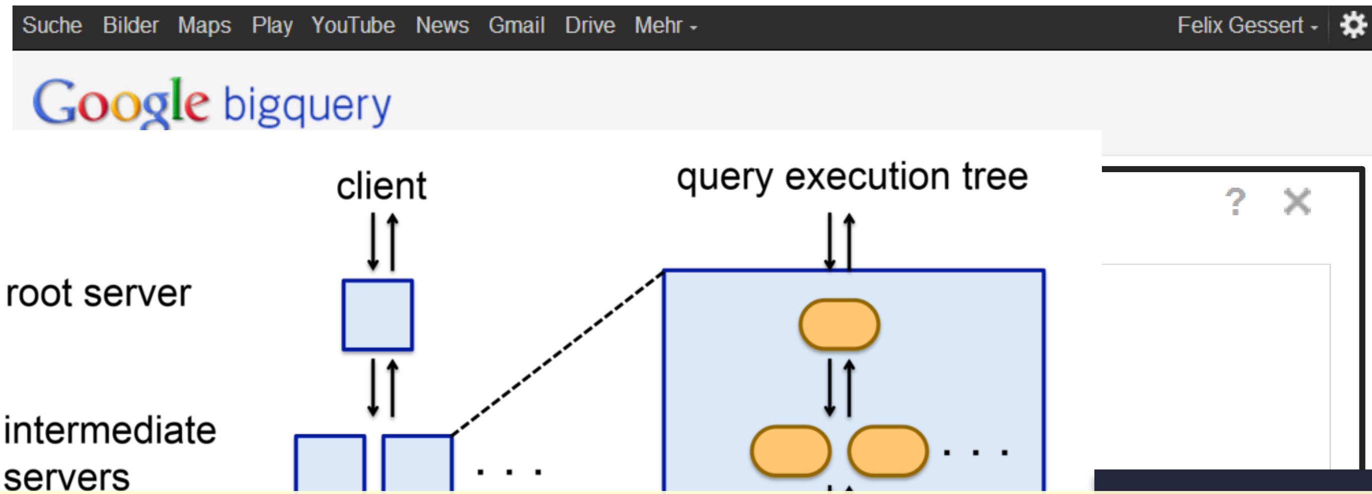
Amazon EC2

- Method III: Marketplaces

Google BigQuery

BigQuery
Model:
Analytics-aaS
Pricing:
Storage + GBs Processed
API:
REST

- **Idea**: Web-scale analysis of nested data



- **SLA**: 99.9% uptime / month
- Fundamentally different from relational DWHs and MapReduce
- Design copied by Apache Drill, Impala, Shark

Managed NoSQL services

Summary

	Model	CAP	Scans	Sec. Indices	Largest Cluster	Lear-ning	Lic.	DBaaS
HBase	Wide-Column	CP	Over Row Key	✗	~700	1/4	Apache	✗ (EMR)
MongoDB	Doc-ument	CP	yes	✓	>100 <500	4/4	GPL	mongoHQ
Riak	Key-Value	AP	✗	✓	~60	3/4	Apache	✗ (Softlayer)
Cassandra		AP	With	✓		2/4	Apache	instaclustr
Redis		CA	Lists etc.			4/4	BSD	Amazon ElastiCache

And there are many more:

- CouchDB (e.g. *Cloudant*)
- CouchBase (e.g. *KuroBase Beta*)
- ElasticSearch(e.g. *Bonsai*)
- Solr (e.g. *WebSolr*)
- …

Proprietary Database services

Summary

	Model	CAP	Scans	Sec. Indices	Queries	API	Scale-out	SLA
SimpleDB	Table-Store	CP	Yes (as queries)	Auto-matic	SQL-like (no joins, groups, …)	REST + SDKs	✗	✗
Dynamo-DB	Table-Store	CP	By range key / index	Local Sec. Global Sec.	Key+Cond. On Range Key(s)	REST + SDKs	Automatic over Prim. Key	✗
Azure Tables	Table-Store	CP	By range key	✗	Key+Cond. On Range Key	REST + SDKs	Automatic over Part. Key	99.9% uptime
AE/Cloud DataStore	Entity-Group	CP	Yes (as queries)	Auto-matic	Conjunct. of Eq. Predicates	REST/ SDK, JDO,JPA	Automatic over Entity Groups	✗
S3, Az. Blob, GCS	Blob-Store	AP	✗	✗	✗	REST + SDKs	Automatic over key	99.9% uptime (S3)

Research Challanges

Encrypted Databases

- Example: **CryptDB**
- **Idea:** Only decrypt as much

SQL-Proxy

Encrypts and decrypts,

Relational Cloud

DBaaS Architecture:

- Encrypted with **CryptDB**
- **Multi-Tenancy** through live migration
- Workload-aware **partitioning** (graph-based)

C. Curino, et al. "Relational cloud: A database-as-a-service for the cloud.", CIDR 2011

RDBMS

- Early approach
- Not adopted in practice, yet

Dream solution:

Full Homomorphic Encryption

Research Challanges

Transactions and Scalable Consistency

Dynamo	Eventual			
Yahoo PNuts	Timeline pe			
COPS	Causality			
MySQL (async)	Serializable			
Megastore	Serializable	Static Partition	2 RT	
Spann	Snapshot Isolation	Partition	2 RT	
MDCC	Read-Committed	Multi-Record	1 RT	-

Multi-Data Center Consistency

Idea:

- Multi-Data center commit protocol with single round-trip

Implementation:

- Optimistic Commit Protocol
- Fast, Generalized Multi-Paxos

Result: almost as fast as Dynamo-style

T. Kraska et al. "MDCC: Multi-data center consistency." EuroSys, 2013.

Currently no NoSQL DB implements consistent Multi-DC replication

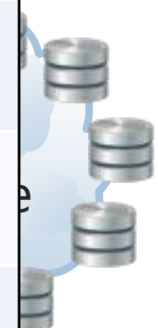
Research Challanges

NoSQL Benchmarking

- **YCSB** (**Y**ahoo **C**loud **S**erving **B**enchmark)

Read()

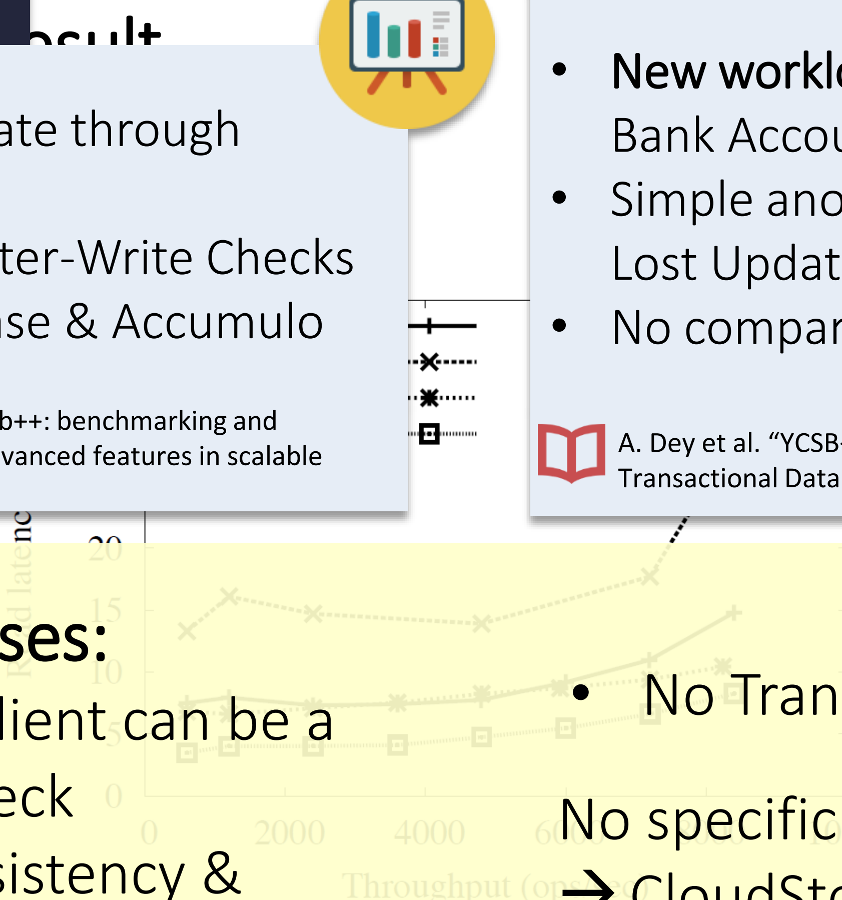
Workload	Operation Mix	Distribution	Example
A – Update Heavy	Read: 50% Update: 50%	Zipfian	Session Store
B – Read Heavy	Read: 95% Update: 5%	Zipfian	Photo Tagging
C – Read Only	Read: 100%	Zipfian	User Profile Cache
D – Read Latest	Read: 95% Insert: 5%	Latest	User Status Updates
E – Short Ranges	Scan: 95% Insert: 5%	Zipfian/ Uniform	Threaded Conversations



3. Popularity Distribution

Research Challanges

NoSQL Benchmarking



YCSB++

- Clients coordinate through Zookeeper
- Simple Read-After-Write Checks
- Evaluation: Hbase & Accumulo

S. Patil, M. Polte, et al.„Ycsb++: benchmarking and performance debugging advanced features in scalable table stores“, SOCC 2011

YCSB+T

- **New workload:** Transactional Bank Account
- Simple anomaly detection for Lost Updates
- No comparison of systems

A. Dey et al. “YCSB+T: Benchmarking Web-Scale Transactional Databases”, CloudDB 2014

Weaknesses:

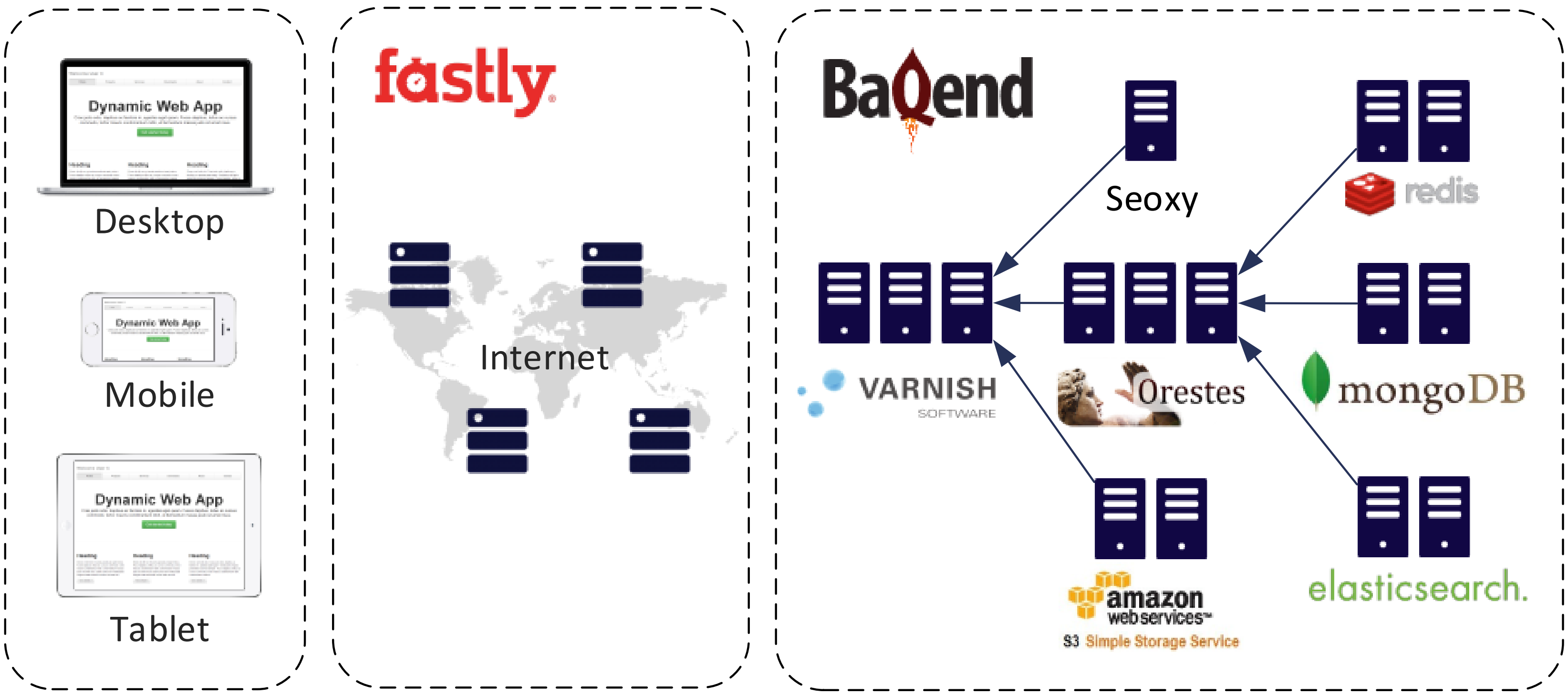
- Single client can be a bottleneck
- No consistency & availability measurement
- No Transaction Support

No specific application
→ CloudStone, CARE, TPC extensions?


BaQend
Build faster Apps faster.

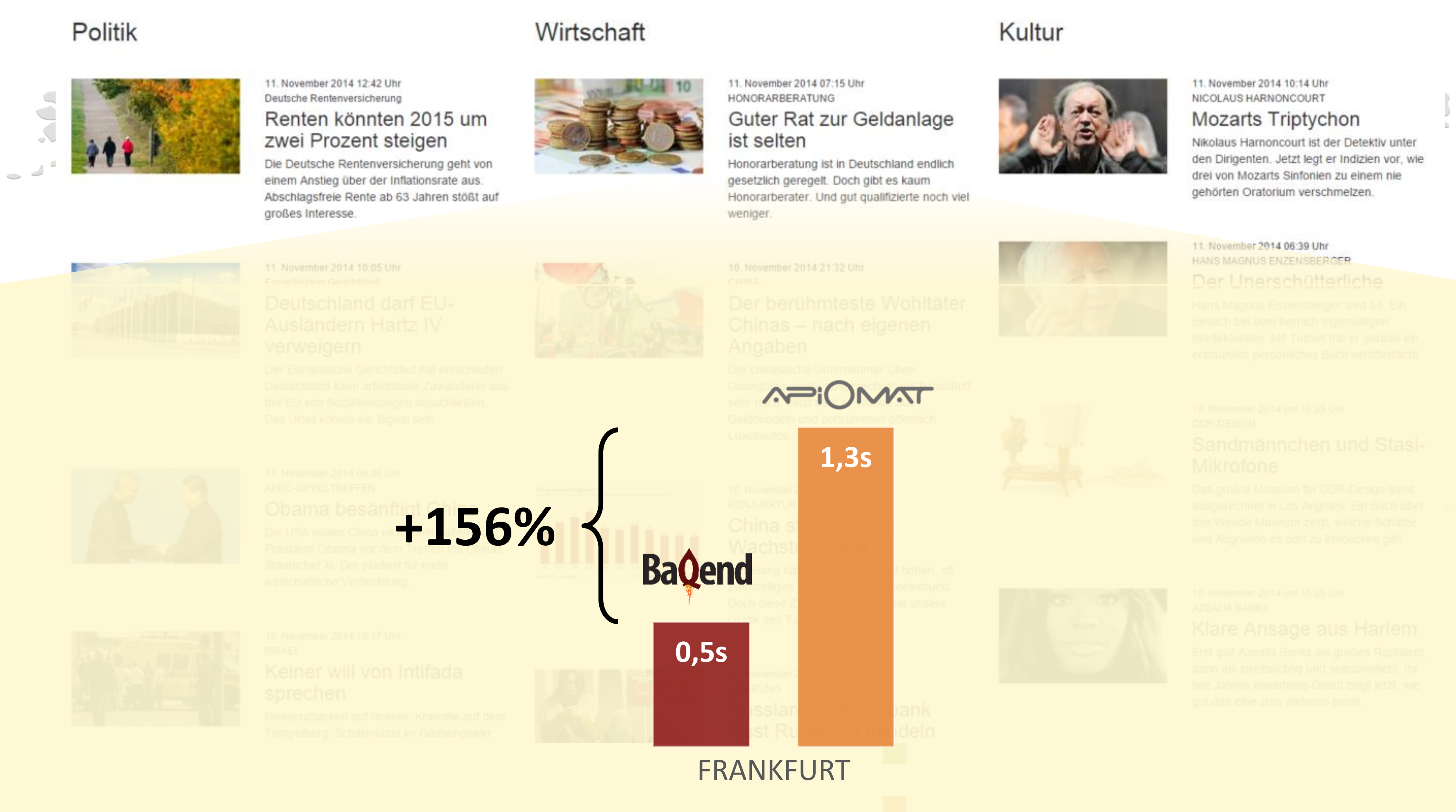
Baqend

- Orestes-as-a-Startup, funded since March 2014



Page-Load Times

Baqends Caching Advantage



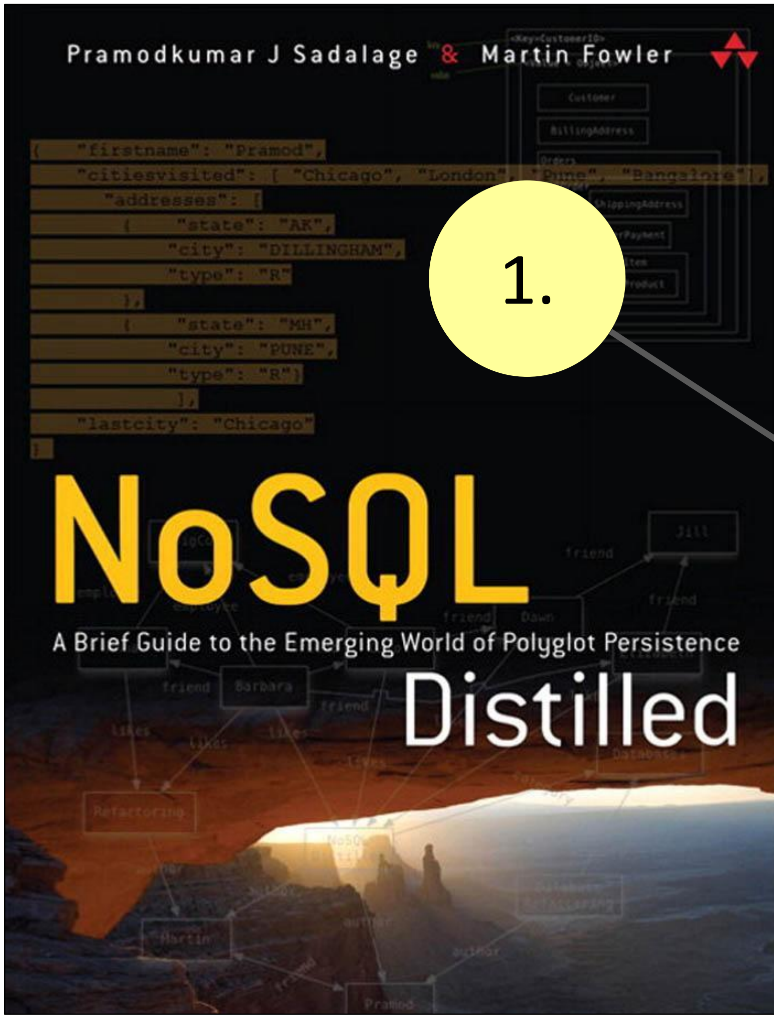
More at Baqend.com

and on **Wednesday 11:30, Hörsaal B**

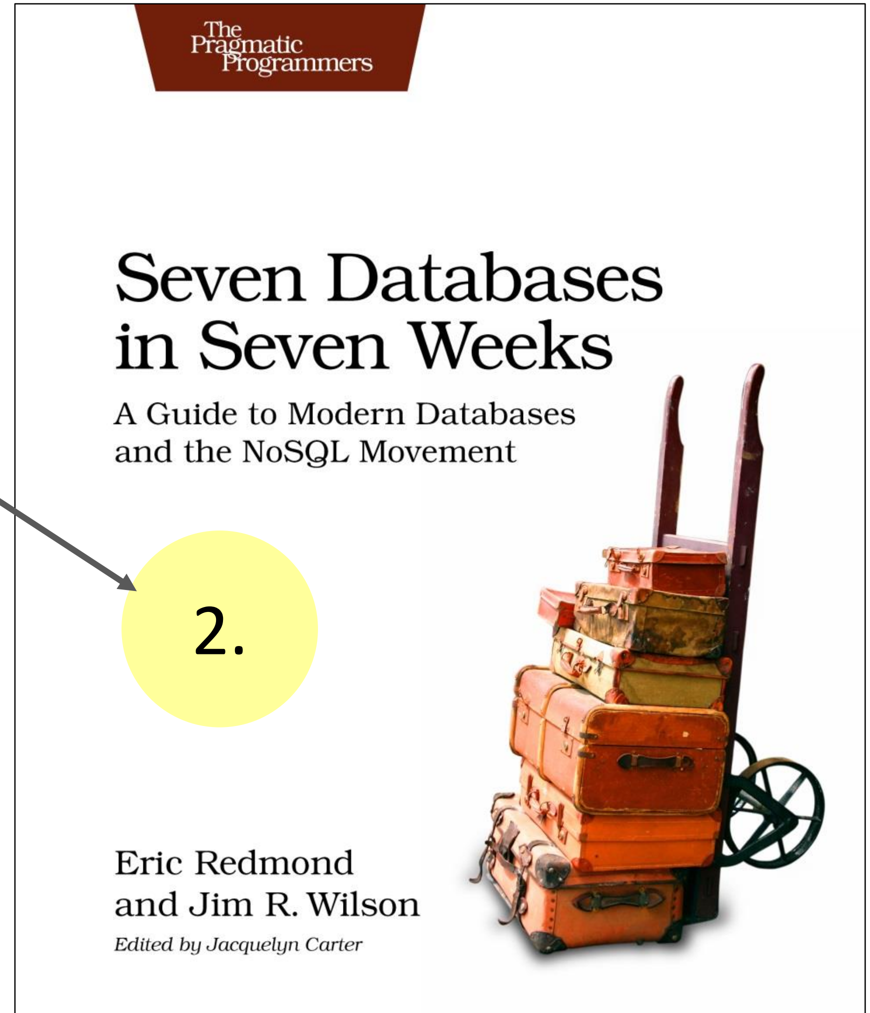
Public Beta starts at the
end of this month

Recommended Literature: NoSQL

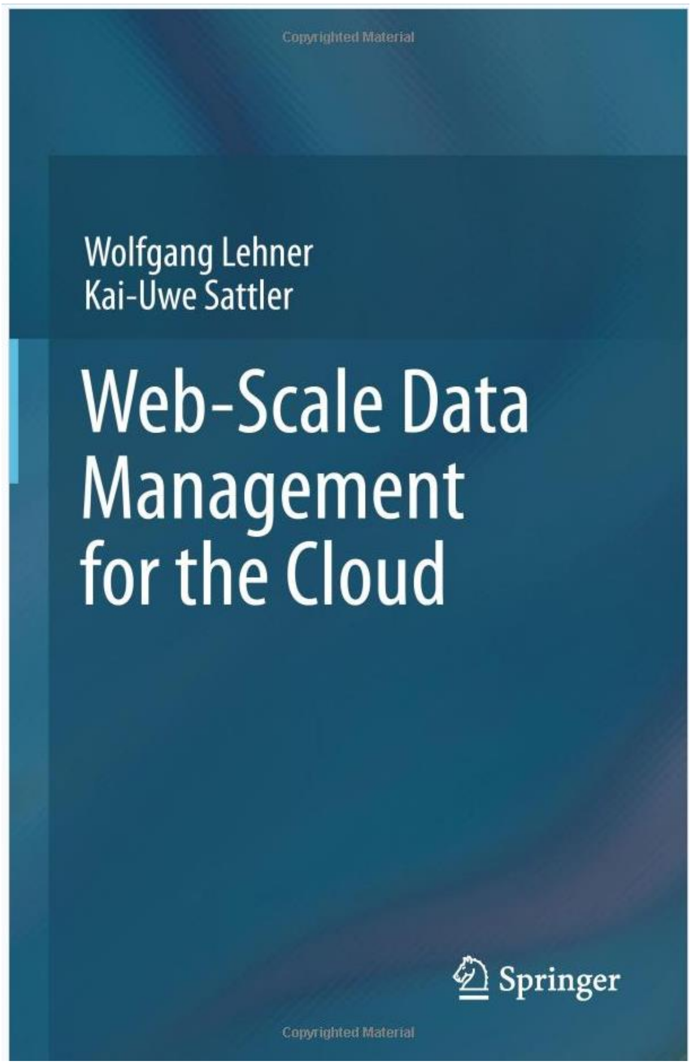
1.

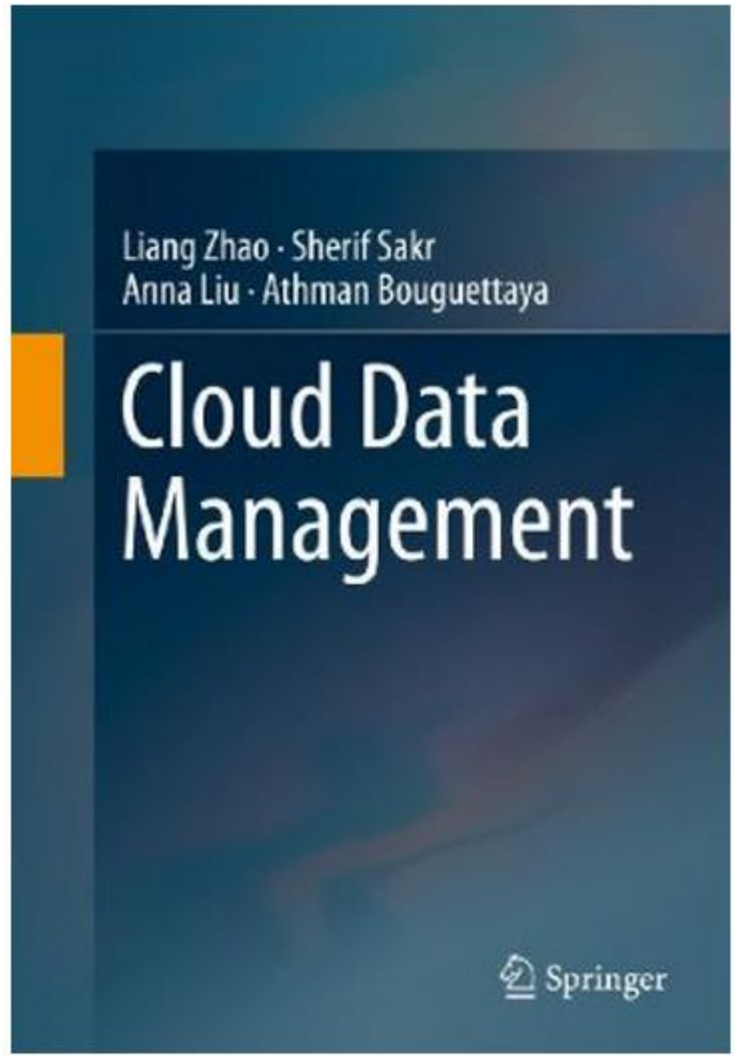


2.



Recommended Literature: Cloud DBs





Recommended Literature: Blogs

myNoSQL

http://nosql.mypopescu.com/

NoSQL Weekly

http://www.nosqlweekly.com/

DZone

http://www.dzone.com/mz/nosql

Hacking, Distributed

http://hackingdistributed.com/

DBMS2

http://www.dbms2.com/

High Scalability

http://highscalability.com/