Scalable Data Management

NoSQL Data Stores in Research and Practice

Felix Gessert, Norbert Ritter

{gessert,ritter}@informatik.uni-hamburg.de

May 17, ICDE 2016

Extended version of this tutorial:

slideshare.net/felixgessert

Outline

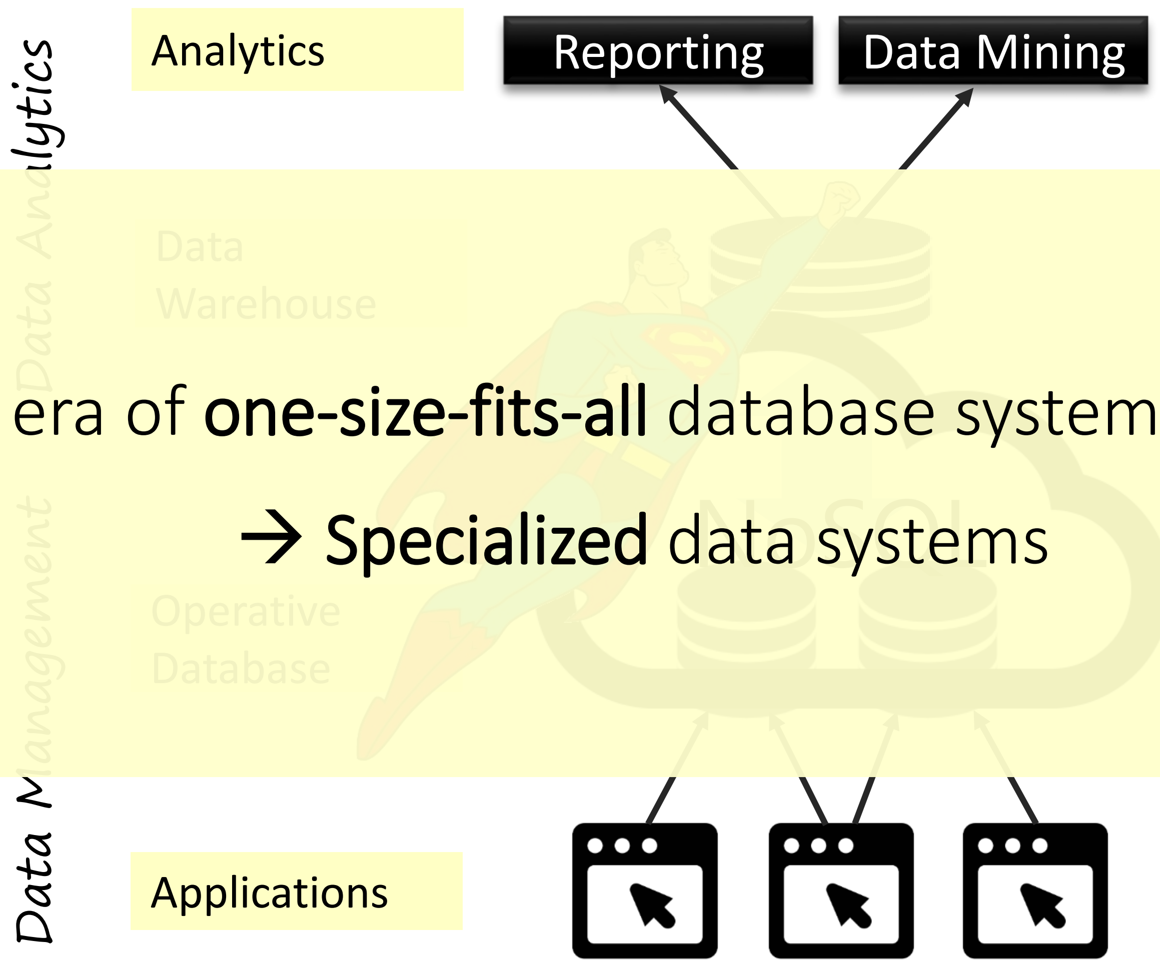
- NoSQL Foundations and Motivation
- The NoSQL Toolbox: Common Techniques
- NoSQL Systems
- Decision Guidance: NoSQL Decision Tree

- The Database Explosion
- NoSQL: Motivation and Origins
- The 4 Classes of NoSQL Databases:
  - Key-Value Stores
  - Wide-Column Stores
  - Document Stores
  - Graph Databases
- CAP Theorem

Introduction: What are NoSQL data stores?

Architecture

Typical Data Architecture:



The era of **one-size-fits-all** database systems is over

→ **Specialized** data systems

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

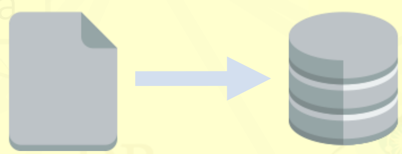
Hadoop-as-a-Service

Big Data Analytics

How to choose a database system?

Many Potential Candidates

Question in this tutorial:

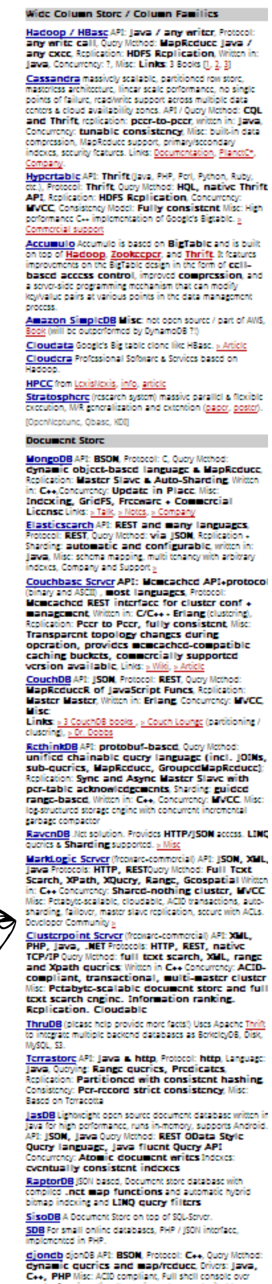
How to approach the  decision problem?

requirements database

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



Wide Column Store / Column Families

Hadoop / HBase API: **Java / any writer**, Protocol: **any write call**, Query Method: **MapReduce Java / any exec**, Replication: **HDFS Replication**, Written in: **Java**, Concurrency: ?, Misc: **Links**: 3 Books [1, 2, 3]

Cassandra massively scalable, partitioned row store, masterless architecture, linear scale performance, no single points of failure, read/write support across multiple data centers & cloud availability zones. API / Query Method: **CQL and Thrift**, replication: **peer-to-peer**, written in: **Java**, Concurrency: **tunable consistency**, Misc: built-in data compression, MapReduce support, primary/secondary indexes, security features. Links: Documentation, PlanetC*, Company.

Hypertable API: **Thrift** (Java, PHP, Perl, Python, Ruby, etc.), Protocol: **Thrift**, Query Method: **HQL, native Thrift API**, Replication: **HDFS Replication**, Concurrency: **MVCC**, Consistency Model: **Fully consistent** Misc: High performance C++ implementation of Google's Bigtable. » Commercial support

Accumulo Accumulo is based on **BigTable** and is built on top of **Hadoop**, **Zookeeper**, and **Thrift**. It features improvements on the BigTable design in the form of **cell-based access control**, improved **compression**, and a server-side programming mechanism that can modify key/value pairs at various points in the data management process.

Amazon SimpleDB **Misc**: not open source / part of AWS, Book (will be outperformed by DynamoDB ?!)

Cloudata Google's Big table clone like HBase. » Article

Cloudera Professional Software & Services based on Hadoop.

HPCC from LexisNexis, info, article

Stratosphere (research system) massive parallel & flexible execution, M/R generalization and extention (paper, poster).

[OpenNeptune, Qbase, KDI]

Document Store

MongoDB API: **BSON**, Protocol: C, Query Method: **dynamic object-based language & MapReduce**, Replication: **Master Slave & Auto-Sharding**, Written in: **C++**, Concurrency: **Update in Place**. Misc: **Indexing, GridFS, Freeware + Commercial License** Links: » Talk, » Notes, » Company

Elasticsearch API: **REST and many languages**, Protocol: **REST**, Query Method: **via JSON**, Replication + Sharding: **automatic and configurable**, written in: **Java**, Misc: schema mapping, multi tenancy with arbitrary indexes, Company and Support »

Couchbase Server **API: Memcached API+protocol** (binary and ASCII) , **most languages**, Protocol: **Memcached REST interface for cluster conf + management**, Written in: **C/C++ + Erlang** (clustering), Replication: **Peer to Peer, fully consistent**, Misc: **Transparent topology changes during operation, provides memcached-compatible caching buckets, commercially supported version available**, Links: » Wiki, » Article

CouchDB API: **JSON**, Protocol: **REST**, Query Method: **MapReduceR of JavaScript Funcs**, Replication: **Master Master**, Written in: **Erlang**, Concurrency: **MVCC**, **Misc**:
Links: » 3 CouchDB books , » Couch Lounge (partitioning / clusering), » Dr. Dobbs

RethinkDB API: **protobuf-based**, Query Method: **unified chainable query language (incl. JOINs, sub-queries, MapReduce, GroupedMapReduce)**; Replication: **Sync and Async Master Slave with per-table acknowledgements**, Sharding: **guided range-based**, Written in: **C++**, Concurrency: **MVCC**. Misc: log-structured storage engine with concurrent incremental garbage compactor

RavenDB .Net solution. Provides **HTTP/JSON** access. **LINQ** queries & **Sharding** supported. » Misc

MarkLogic Server (freeware+commercial) API: **JSON, XML, Java** Protocols: **HTTP, REST** Query Method: **Full Text Search, XPath, XQuery, Range, Geospatial** Written in: **C++** Concurrency: **Shared-nothing cluster, MVCC** Misc: Petabyte-scalable, cloudable, ACID transactions, auto-sharding, failover, master slave replication, secure with ACLs. Developer Community »

Clusterpoint Server (freeware+commercial) API: **XML, PHP, Java, .NET** Protocols: **HTTP, REST, native TCP/IP** Query Method: **full text search, XML, range and Xpath queries**; Written in **C++** Concurrency: **ACID-compliant, transactional, multi-master cluster** Misc: **Petabyte-scalable document store and full text search engine. Information ranking. Replication. Cloudable**

ThruDB (please help provide more facts!) Uses Apache Thrift to integrate multiple backend databases as BerkeleyDB, Disk, MySQL, S3.

Terrastore API: **Java & http**, Protocol: **http**, Language: **Java**, Querying: **Range queries, Predicates**, Replication: **Partitioned with consistent hashing**, Consistency: **Per-record strict consistency**, Misc: Based on Terracotta

JasDB Lightweight open source document database written in Java for high performance, runs in-memory, supports Android. API: **JSON, Java** Query Method: **REST OData Style Query language, Java fluent Query API** Concurrency: **Atomic document writes** Indexes: **eventually consistent indexes**

RaptorDB JSON based, Document store database with compiled **.net map functions** and automatic hybrid bitmap indexing and **LINQ query filters**

SisoDB A Document Store on top of SQL-Server.

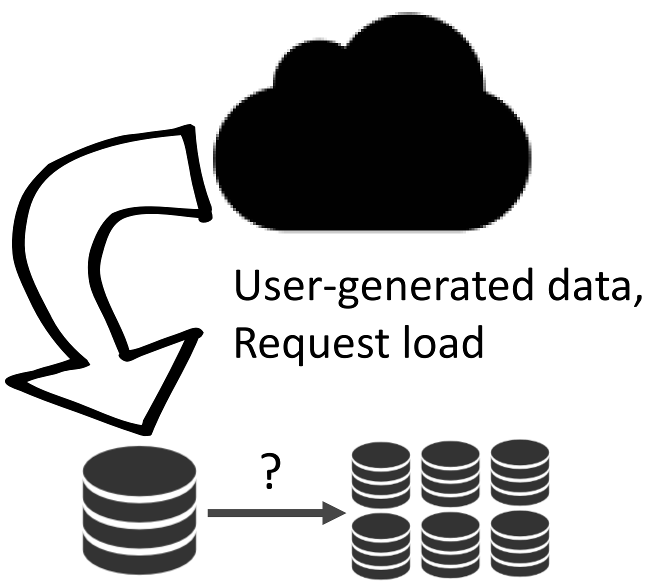
SDB For small online databases, PHP / JSON interface, implemented in PHP.

djondb djonDB API: **BSON**, Protocol: **C++**, Query Method: **dynamic queries and map/reduce**, Drivers: **Java, C++, PHP** Misc: ACID compliant, Full shell console over google v8 engine, djondb requirements are submited by users,

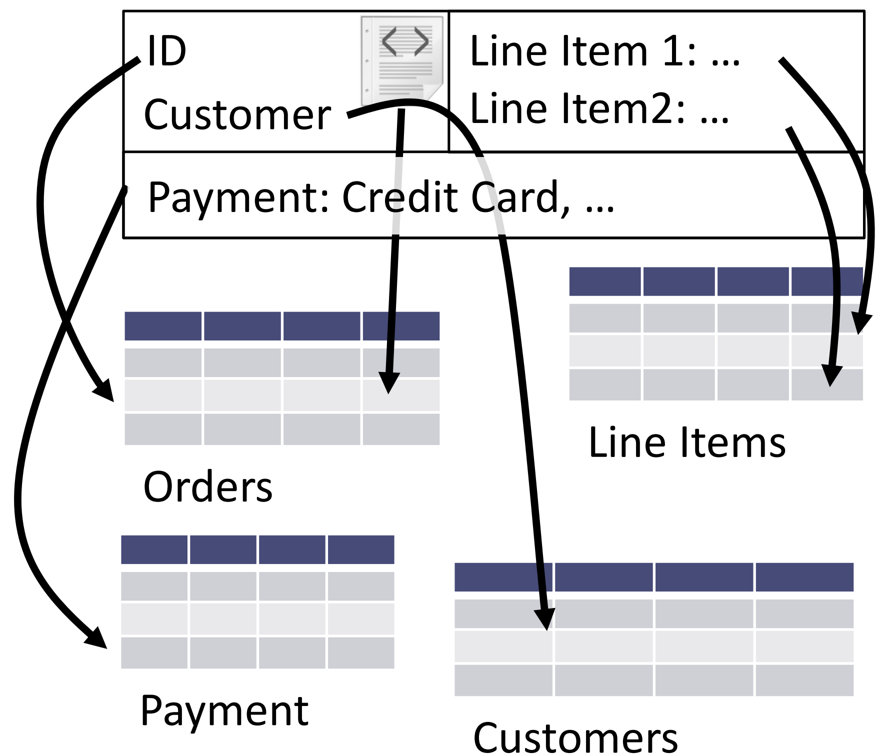
NoSQL Databases

- Two main motivations:

Scalability

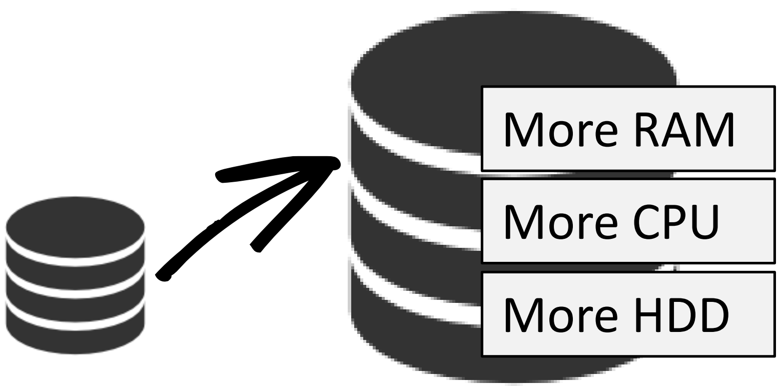


Impedance Mismatch

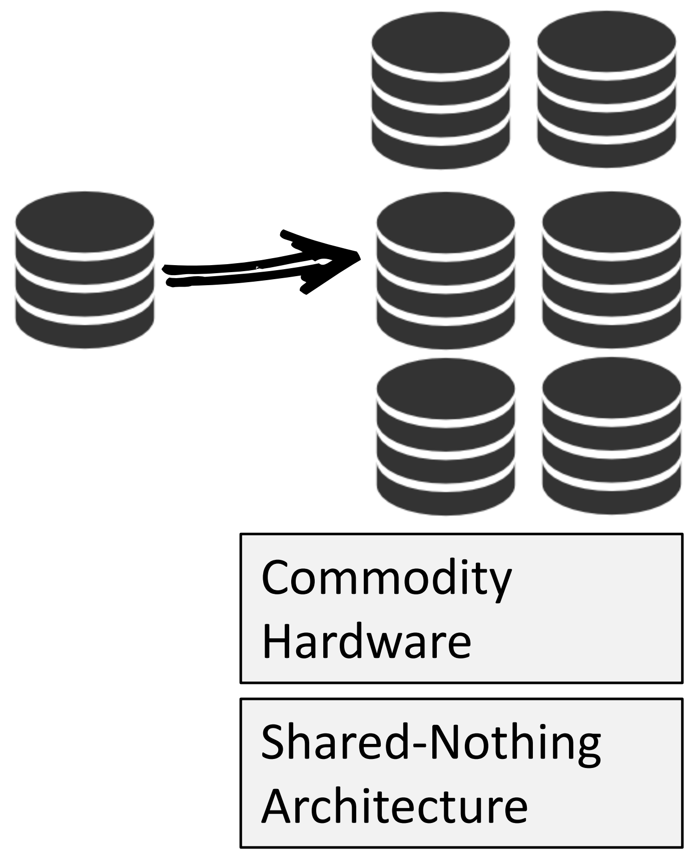


Scale-up vs Scale-out

Scale-Up (*vertical* scaling):

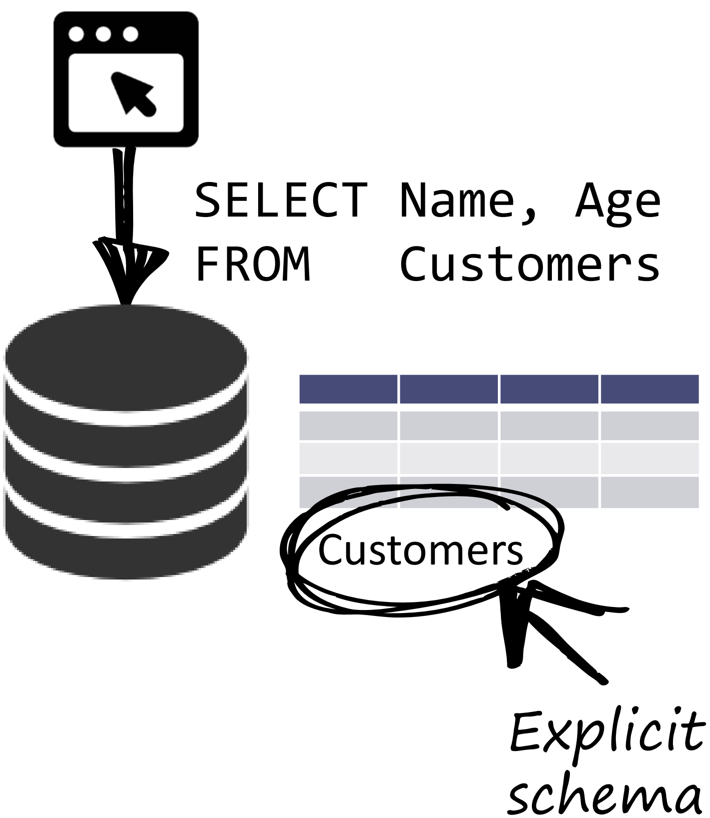


Scale-Out (*horizontal* scaling):

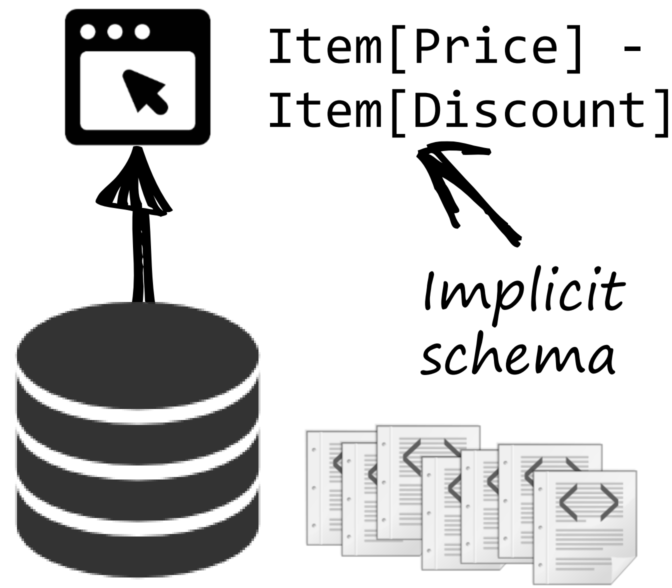


Schemafree Data Modeling

RDBMS:



NoSQL DB:



Open Source & Commodity Hardware

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)

NoSQL System Classification

- Two common criteria:

Data Model

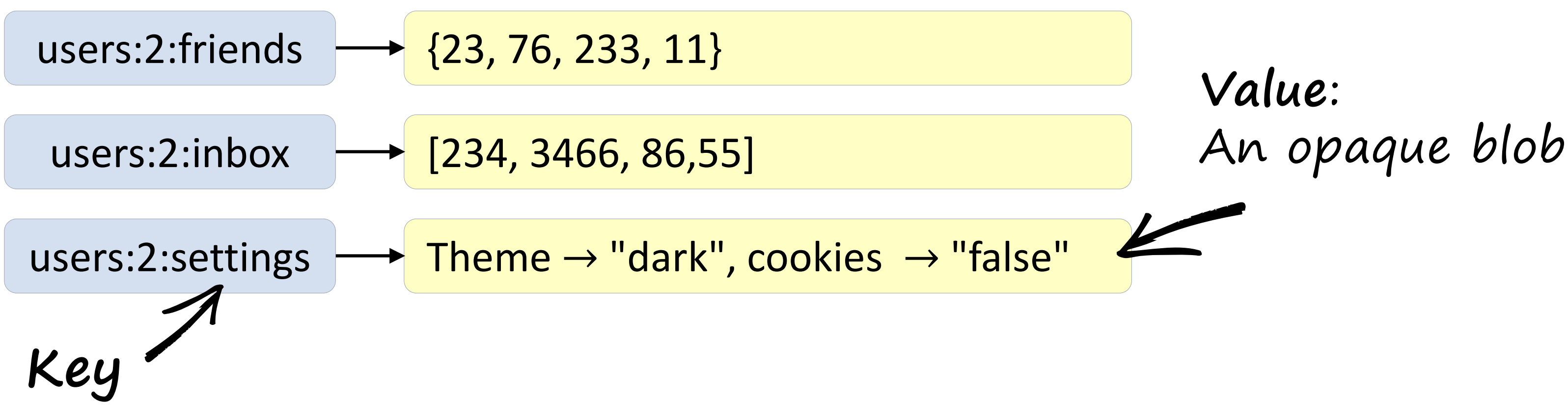
- Key-Value
- Wide-Column
- Document
- Graph

Consistency/Availability Trade-Off

- **AP**: Available & Partition Tolerant
- **CP**: Consistent & Partition Tolerant
- **CA**: Not Partition Tolerant

Key-Value Stores

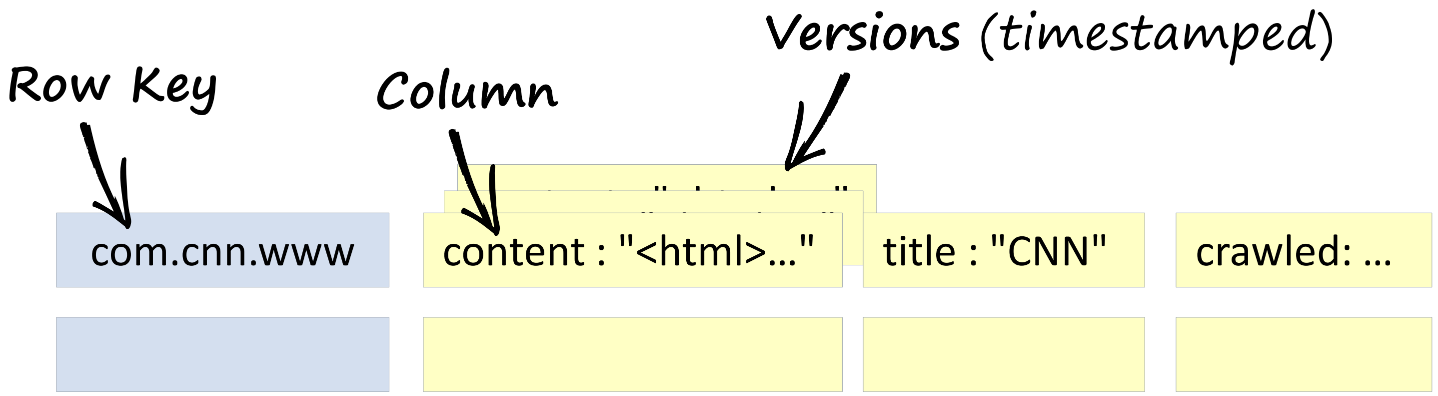
- **Data model:** (key) -> value
- **Interface**: CRUD (Create, Read, Update, Delete)



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

Wide-Column Stores

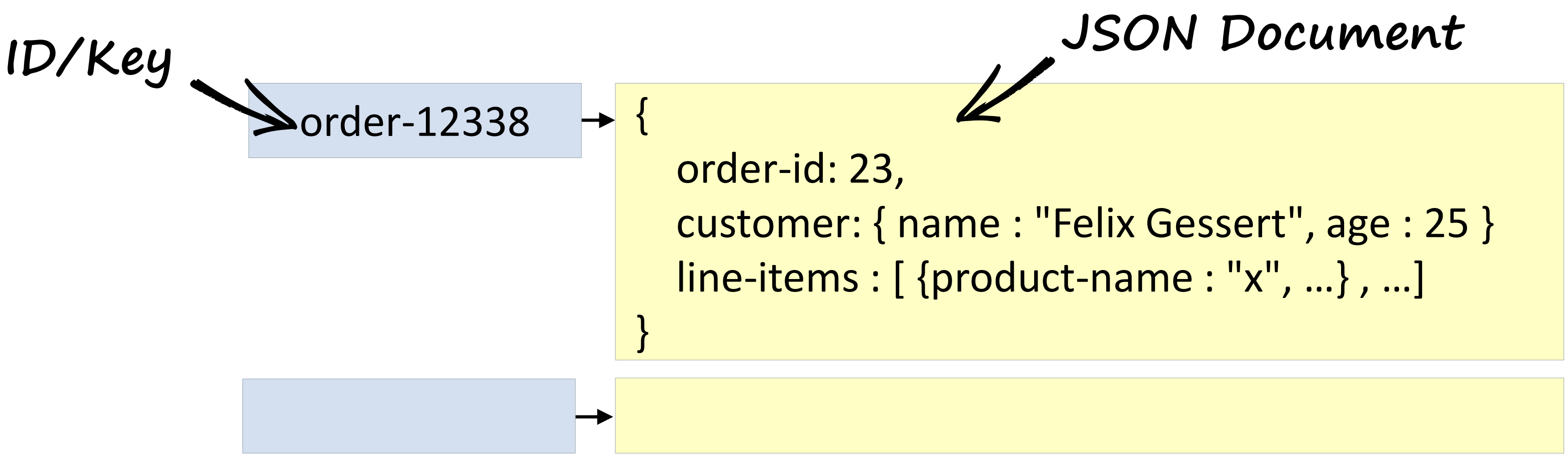
- **Data model:** (rowkey, column, timestamp) -> value
- **Interface**: CRUD, Scan



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

Document Stores

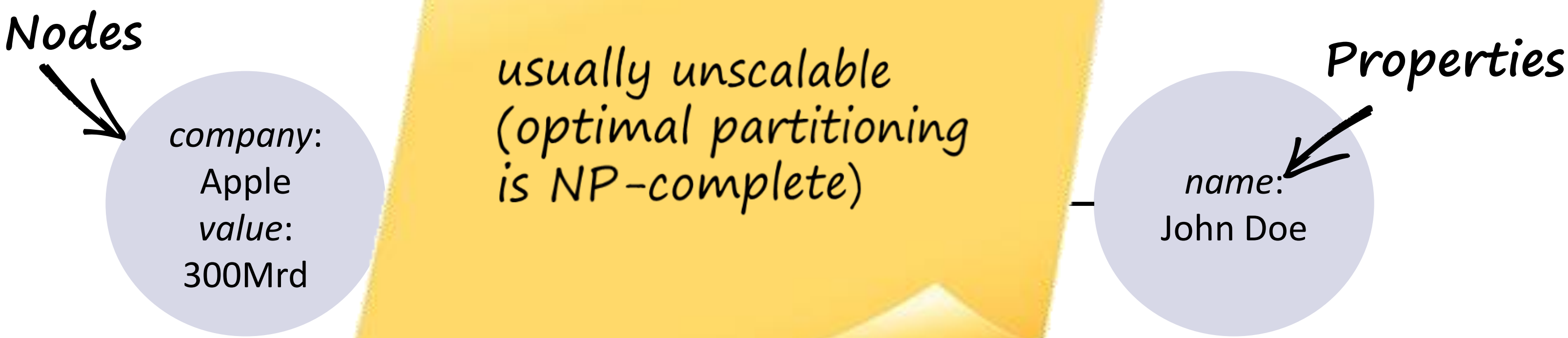
- **Data model:** (collection, key) -> document
- **Interface**: CRUD, Querys, Map-Reduce



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

Graph Databases

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



usually unscalable (optimal partitioning is NP-complete)

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

Soft NoSQL Systems

Not Covered Here

Search Platforms (Full Text Search):

- No persistence and consistency guarantees for OLTP
- *Examples*: ElasticSearch (AP), Solr (AP)

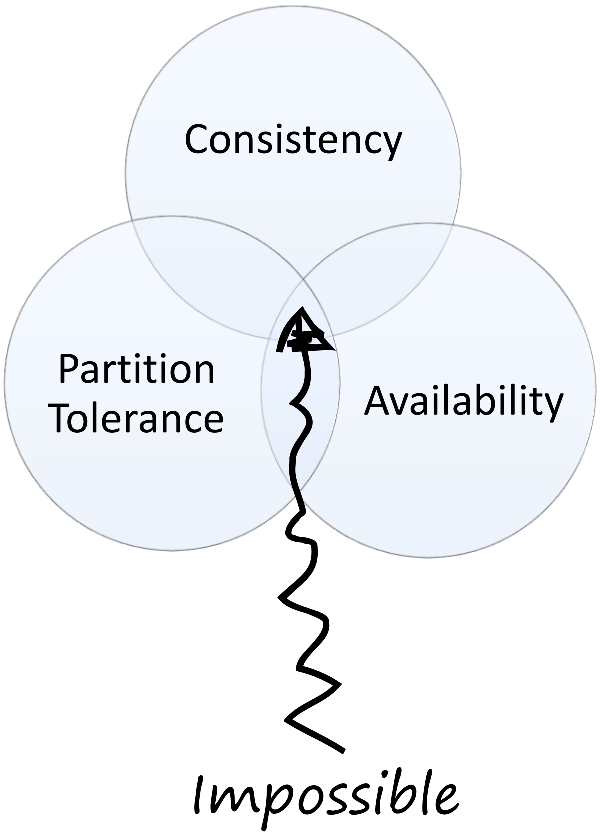
Object-Oriented Databases:

- Strong coupling of programming language and DB
- *Examples*: Versant (CA), db4o (CA), Objectivity (CA)

XML-Databases, RDF-Stores:

- Not scalable, data models not widely used in industry
- *Examples*: MarkLogic (CA), AllegroGraph (CA)

CAP-Theorem



Only 2 out of 3 properties are achievable at a time:

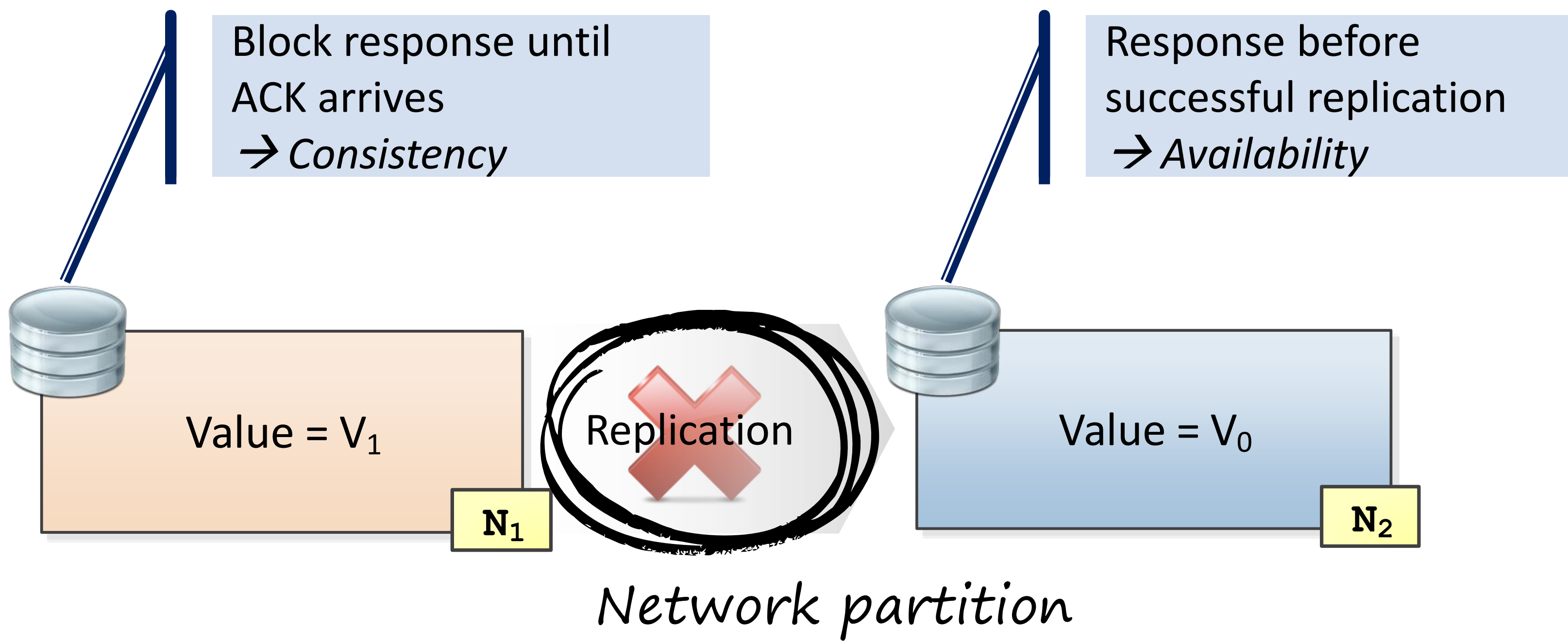
- **Consistency**: all clients have the same view on the data
- **Availability**: every request to a non-failed node most result in correct response
- **Partition tolerance**: the system has to continue working, even under arbitrary network partitions

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

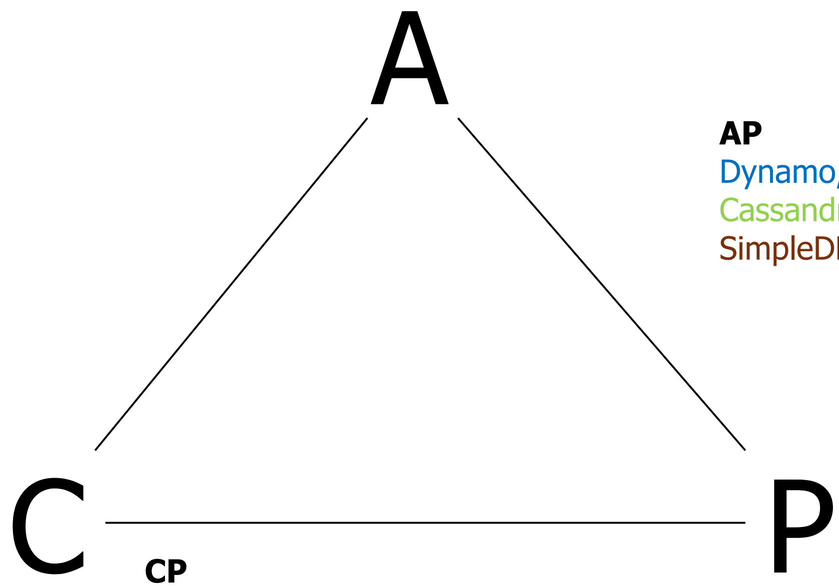
- **Problem**: when a network partition occurs, either consistency or availability have to be given up



NoSQL Triangle

Data models | Relational, Key-Value, Wide-Column, Document-Oriented

Every client can always read and write



CA
Oracle, MySQL, …

AP
Dynamo, Redis, Riak, Voldemort
Cassandra
SimpleDB,

All clients share the same view on the data

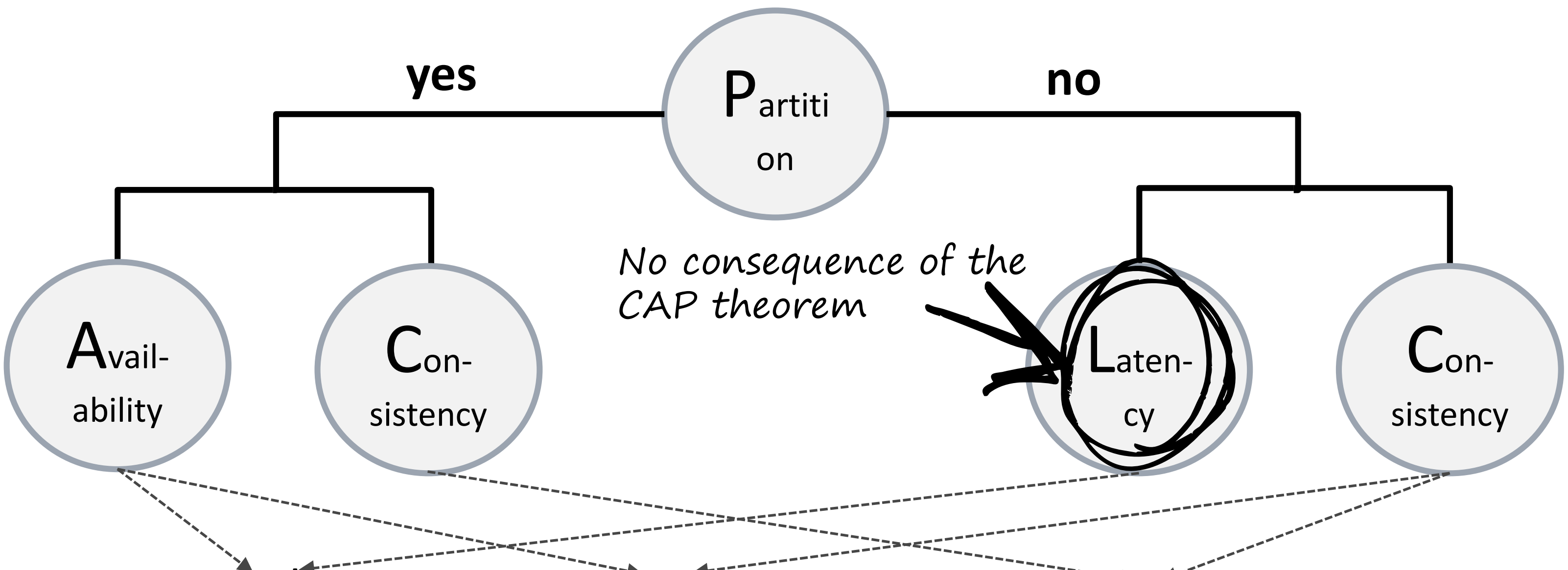
CP
Postgres, MySQL Cluster, Oracle RAC
BigTable, HBase, Accumulo, Azure Tables
MongoDB, RethinkDB

All nodes continue working under network partitions

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

PACELC – an alternative CAP formulation

- **Idea**: Classify systems according to their behavior during *network partitions*



AL - Dynamo-Style
Cassandra, Riak, etc.

AC - MongoDB

CC – Always Consistent
HBase, BigTable and ACID systems

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

Serializability

Not Highly Available Either

Global serializability and availability are incompatible:

Write A=1
Read B

Write B=1
Read A

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

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

- Some weaker isolation levels allow high availability:
  - RAMP Transactions (P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, und I. Stoica, „Scalable Atomic Visibility with RAMP Transactions“, SIGMOD 2014)

S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. ACM CSUR, 17(3):341–370, 1985.

Where CAP fits in

Negative Results in Distributed Computing

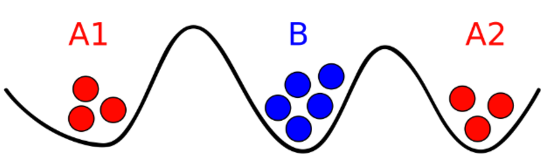
Asynchronous Network, _Unreliable_ Channel

Atomic Storage

Impossible:
CAP Theorem

Consensus

Impossible:
2 Generals Problem



Asynchronous Network, _Reliable_ Channel

Atomic Storage

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

Consensus

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

Lynch, Nancy A. *Distributed algorithms*. Morgan Kaufmann, 1996.

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

Data Models and **CAP** provide high-level classification.

But what about **fine-grained requirements**, e.g. query capabilites?

Outline

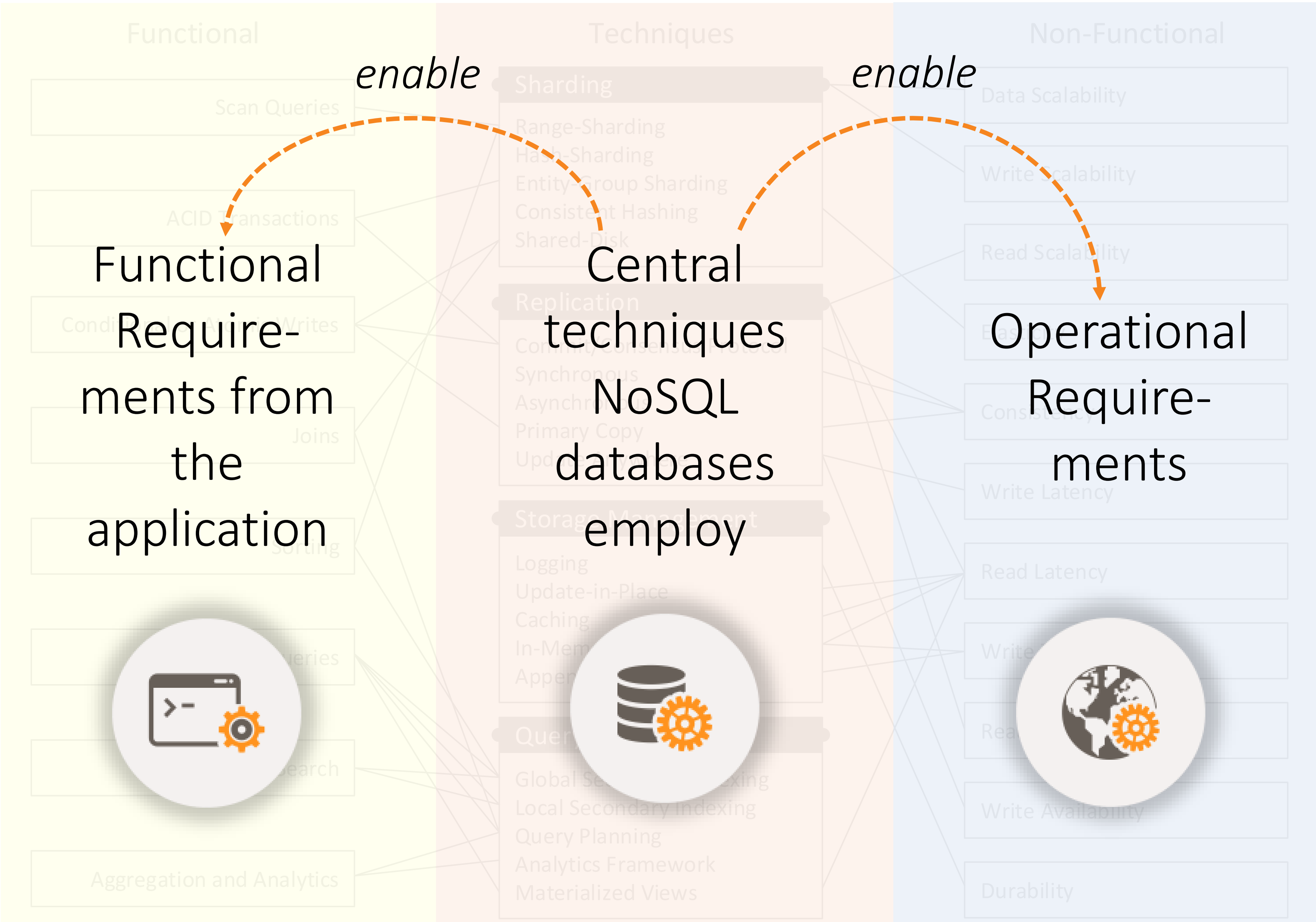
NoSQL Foundations and Motivation

The NoSQL Toolbox: Common Techniques

NoSQL Systems

Decision Guidance: NoSQL Decision Tree

- Techniques for Functional and Non-functional Requirements
  - Sharding
  - Replication
  - Storage Management
  - Query Processing


enable
enable
Functional Require-ments from the application
Central techniques NoSQL databases employ
Operational Require-ments

NoSQL Database Systems: A Survey and Decision Guidance

Felix Gessert, Wolfram Wingerath, Steffen Friedrich, and Norbert Ritter

Universität Hamburg, Germany
{gessert, wingerath, friedrich, ritter}@informatik.uni-hamburg.de

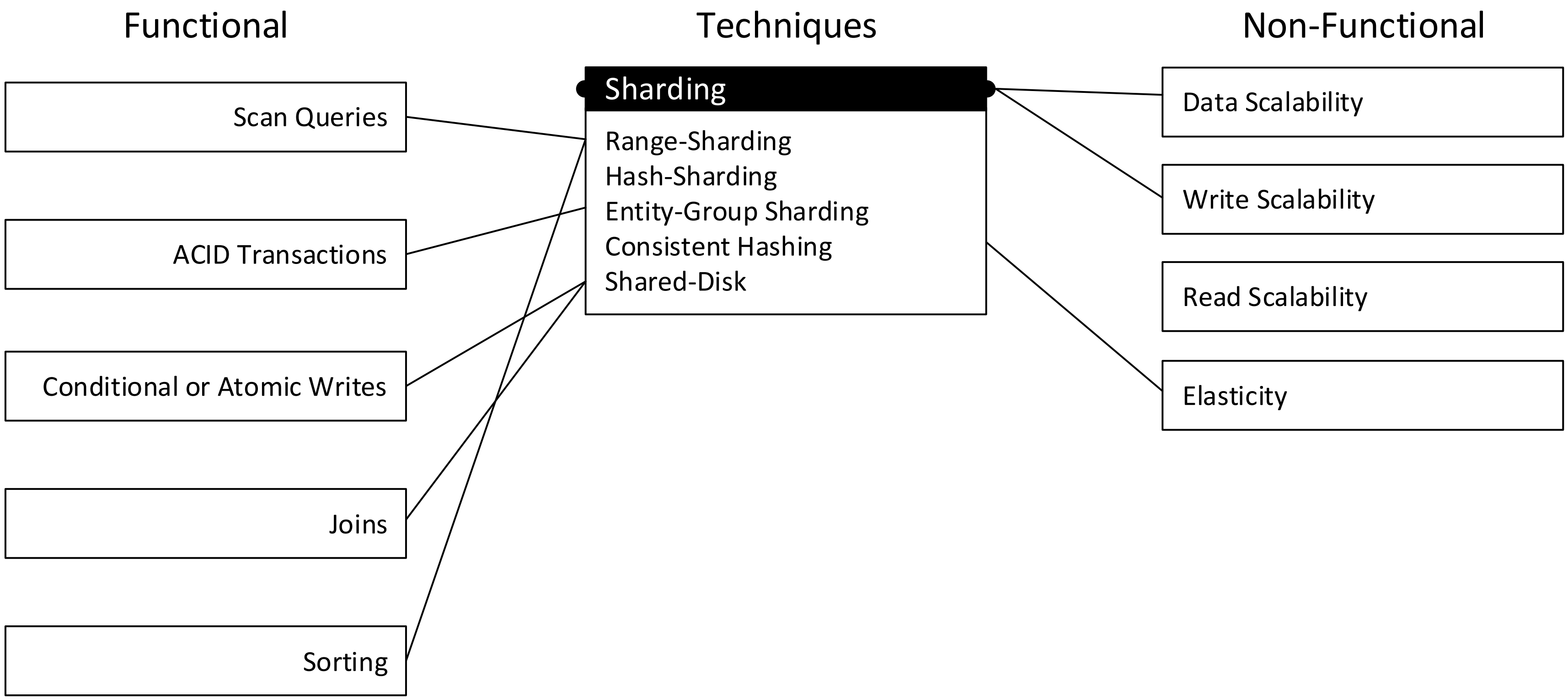
Abstract. Today, data is generated and consumed at unprecedented scale. This has lead to novel approaches for scalable data management subsumed under the term "NoSQL" database systems to handle the ever-increasing data volume and request loads. However, the heterogeneity and diversity of the numerous existing systems impede the well-informed selection of a data store appropriate for a given application context. Therefore, this article gives a top-down overview of the field: Instead of contrasting the implementation specifics of individual representatives, we propose a comparative classification model that relates functional and non-functional requirements to techniques and algorithms employed in NoSQL databases. This NoSQL Toolbox allows us to derive a simple decision tree to help practitioners and researchers filter potential system candidates based on central application requirements.

1 Introduction

Traditional relational database management systems (RDBMSs) provide powerful mechanisms to store and query structured data under strong consistency and transaction guarantees and have reached an unmatched level of reliability, stability and support through decades of development. In recent years, however, the amount of useful data in some application areas has become so vast that it cannot be stored or processed by traditional database solutions. User-generated content in social networks or data retrieved from large sensor networks are only two examples of this phenomenon commonly referred to as **Big Data** [35]. A class of novel data storage systems able to cope with Big Data are subsumed under the term **NoSQL databases**, many of which offer horizontal scalability and higher availability than relational databases by sacrificing querying capabilities and consistency guarantees. These trade-offs are pivotal for service-oriented computing and as-a-service models, since any stateful service can only be as scalable and fault-tolerant as its underlying data store.

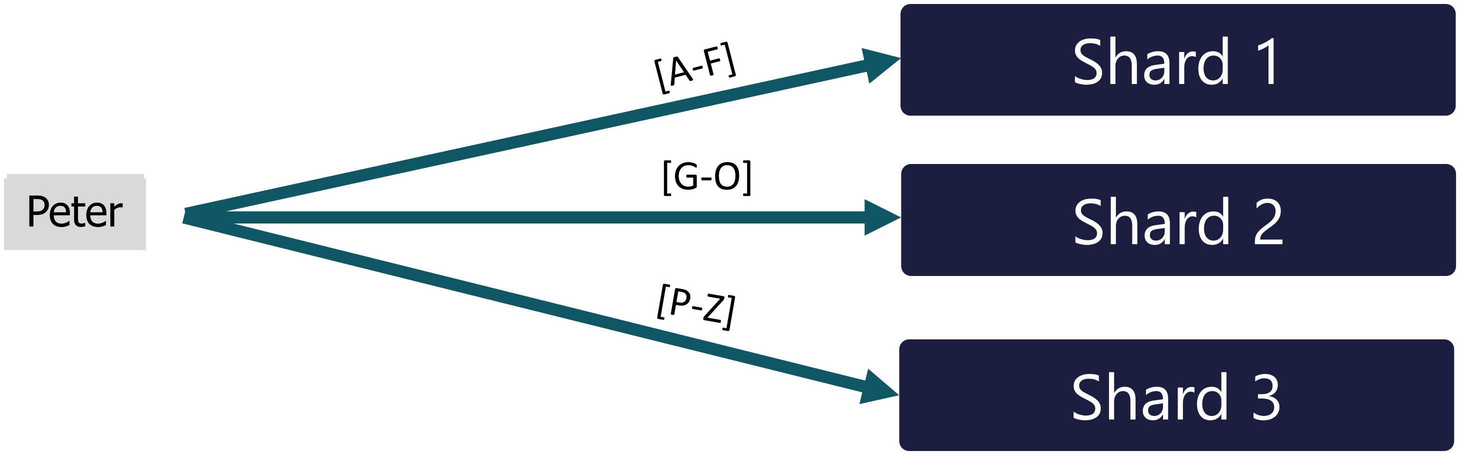
There are dozens of NoSQL database systems and it is hard to keep track of where they excel, where they fail or even where they differ, as implementation details change quickly and feature sets evolve over time. In this article, we therefore aim to provide an overview of the NoSQL landscape by discussing employed concepts rather than system specificities and explore the requirements typically posed to NoSQL database systems, the techniques used to fulfil these requirements and the trade-offs that have to be made in the process. Our focus lies on key-value, document and wide-column stores, since these NoSQL categories

http://www.baqend.com
/files/nosql-survey.pdf


Functional
Techniques
Non-Functional
Scan Queries
ACID Transactions
Conditional or Atomic Writes
Joins
Sorting
Sharding
Range-Sharding
Hash-Sharding
Entity-Group Sharding
Consistent Hashing
Shared-Disk
Data Scalability
Write Scalability
Read Scalability
Elasticity

Sharding (aka Partitioning, Fragmentation)

- Horizontal distribution of data over nodes



- **Partitioning strategies**: Hash-based vs. Range-based
- Difficulty: Multi-Shard-Operations (join, aggregation)

Sharding

Hash-based Sharding

- Hash of data values (e.g. key) d
- **Pro**: Even distribution
- **Contra**: No data locality

Range-based Sharding

- Assigns ranges defined over fie
- **Pro**: Enables *Range Scans* and *S*
- **Contra**: Repartitioning/balancin

Entity-Group Sharding

- Explicit data co-location for sin
- **Pro**: Enables *ACID Transactions*
- **Contra**: Partitioning not easily c

Implemented in

MongoDB, Riak, Redis, Cassandra, Azure Table, Dynamo

Implemented in

BigTable, HBase, DocumentDB Hypertable, MongoDB, RethinkDB, Espresso

Implemented in

G-Store, MegaStore, Relation Cloud, Cloud SQL Server

David J DeWitt and Jim N Gray: "Parallel database systems: The future of high performance database systems," Communications of the ACM, volume 35, number 6, pages 85–98, June 1992.

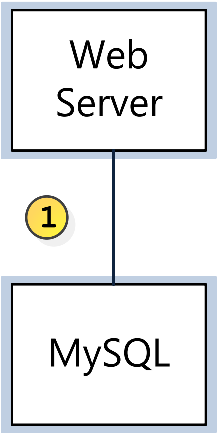
Problems of Application-Level Sharding

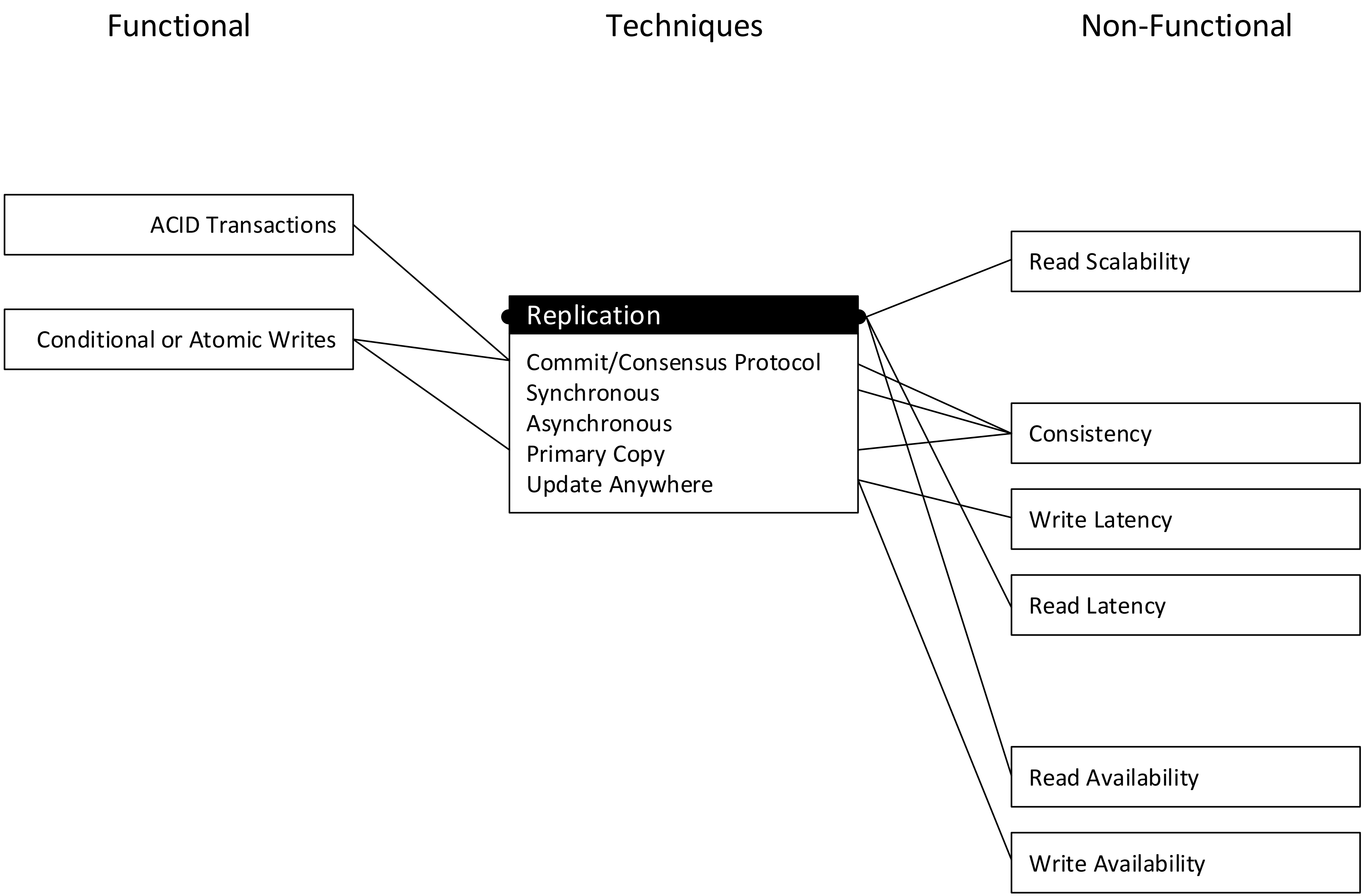
Example: **Tumblr**

- Caching
- Sharding from application

Moved towards:

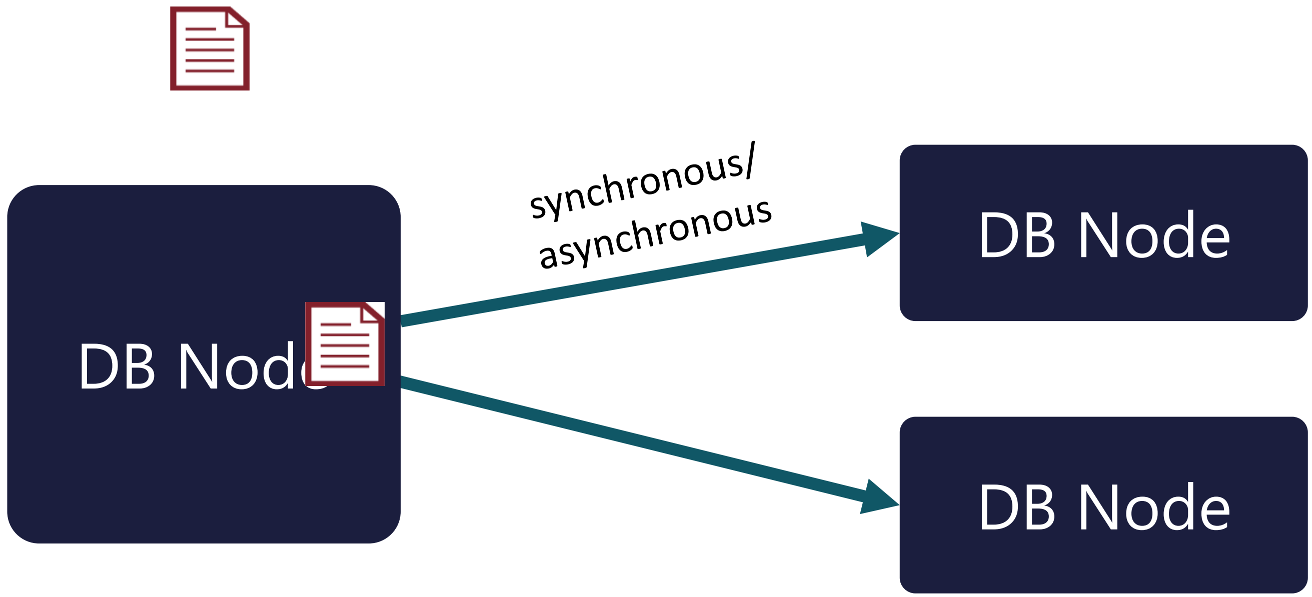
- Redis
- HBase




Functional
Techniques
Non-Functional
ACID Transactions
Conditional or Atomic Writes
Replication
Commit/Consensus Protocol
Synchronous
Asynchronous
Primary Copy
Update Anywhere
Read Scalability
Consistency
Write Latency
Read Latency
Read Availability
Write Availability

Replication

- Stores *N* copies of each data item



- **Consistency model**: synchronous vs asynchronous
- **Coordination**: Multi-Master, Master-Slave

Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer Science & Business Media (2011)

Replication: When

Asynchronous (lazy)

- Writes are acknowledged imm
- Performed through *log shippi*
- **Pro**: Fast writes, no coordinati
- **Contra**: Replica data potential

Synchronous (eager)

- The node accepting writes sync
  updates/transactions before a
- **Pro**: Consistent
- **Contra**: needs a commit proto
  unavaialable under certain network partitions

Implemented in

Dynamo , Riak, CouchDB, Redis, Cassandra, Voldemort, MongoDB, RethinkDB

Implemented in

BigTable, HBase, Accumulo, CouchBase, MongoDB, RethinkDB

Charron-Bost, B., Pedone, F., Schiper, A. (eds.): Replication: Theory and Practice, Lecture Notes in Computer Science, vol. 5959. Springer (2010)

Replication: Where

Master-Slave (*Primary Copy*)

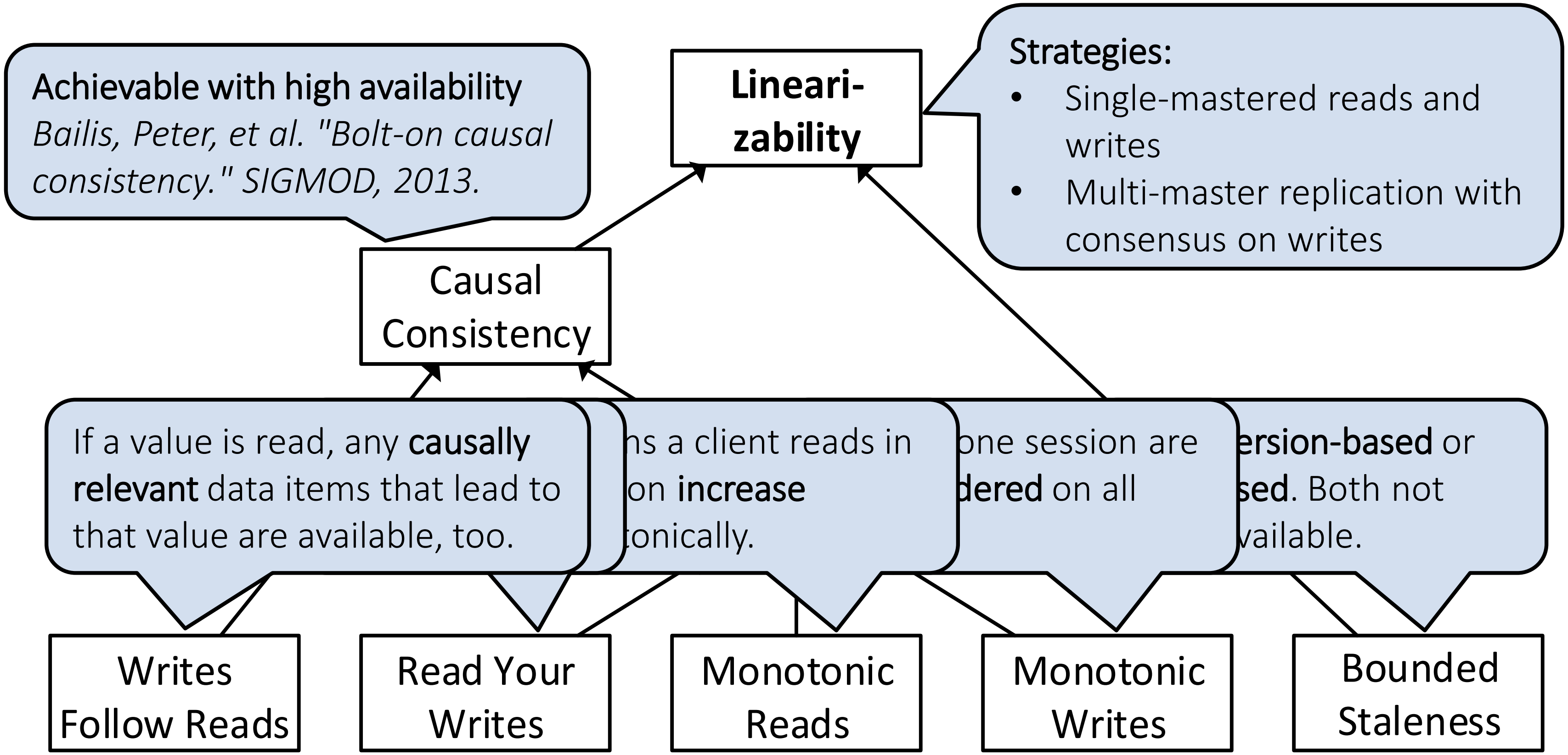
- Only a dedicated master is allowed to accept writes, slaves are read-replicas
- **Pro**: reads from the master are consistent
- **Contra**: master is a bottleneck and SPOF

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 coordination protocols (e.g. Paxos) or is inconsistent

Charron-Bost, B., Pedone, F., Schiper, A. (eds.): Replication: Theory and Practice, Lecture Notes in Computer Science, vol. 5959. Springer (2010)

Consistency Levels



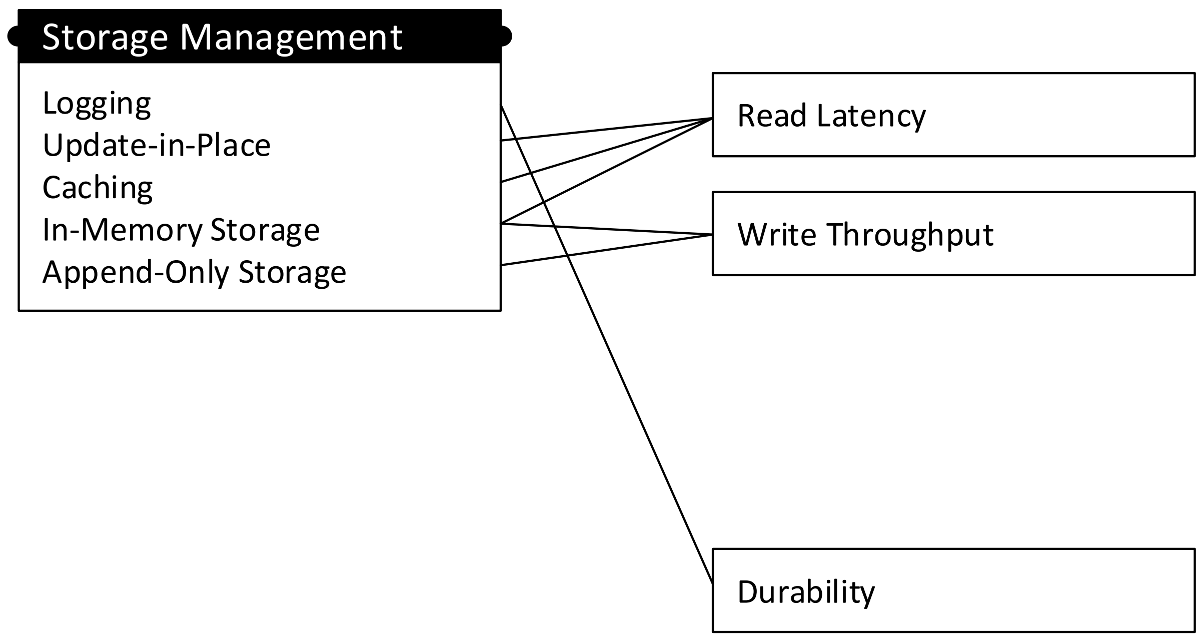
Viotti, Paolo, and Marko Vukolić. "Consistency in Non-Transactional Distributed Storage Systems." arXiv (2015).

Bailis, Peter, et al. "Highly available transactions: Virtues and limitations." Proceedings of the VLDB Endowment 7.3 (2013): 181-192.

Functional

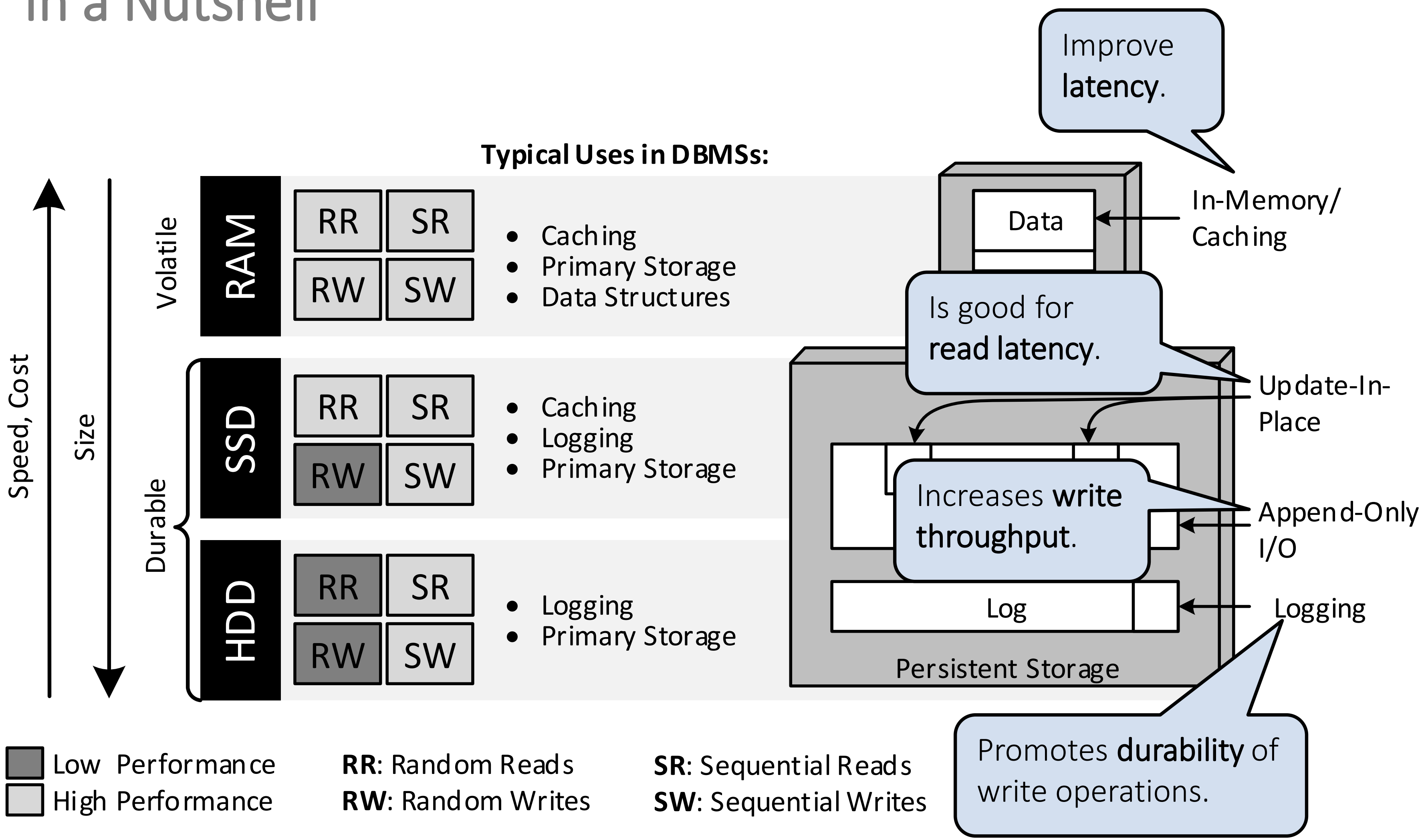
Techniques

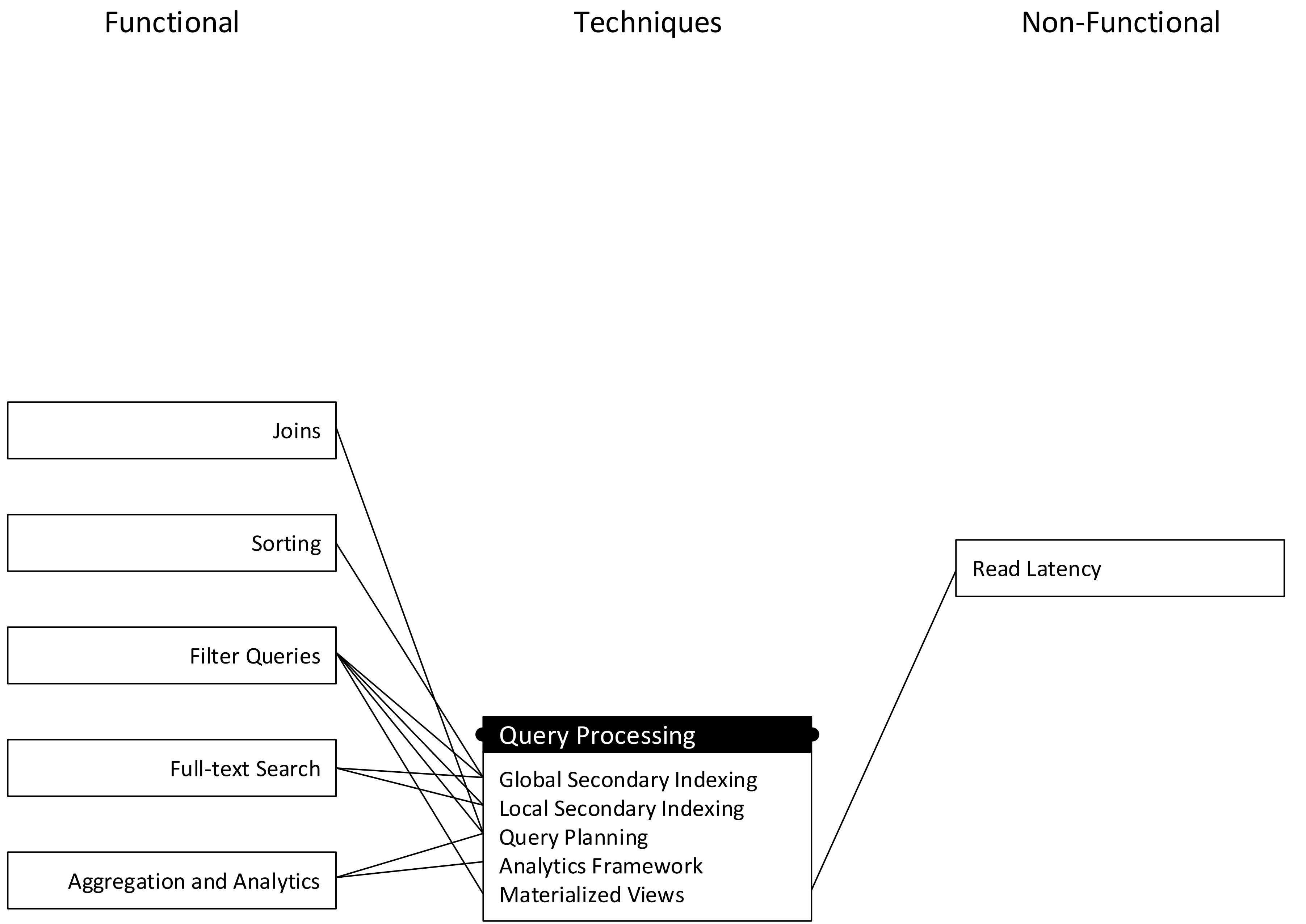
Non-Functional



NoSQL Storage Management

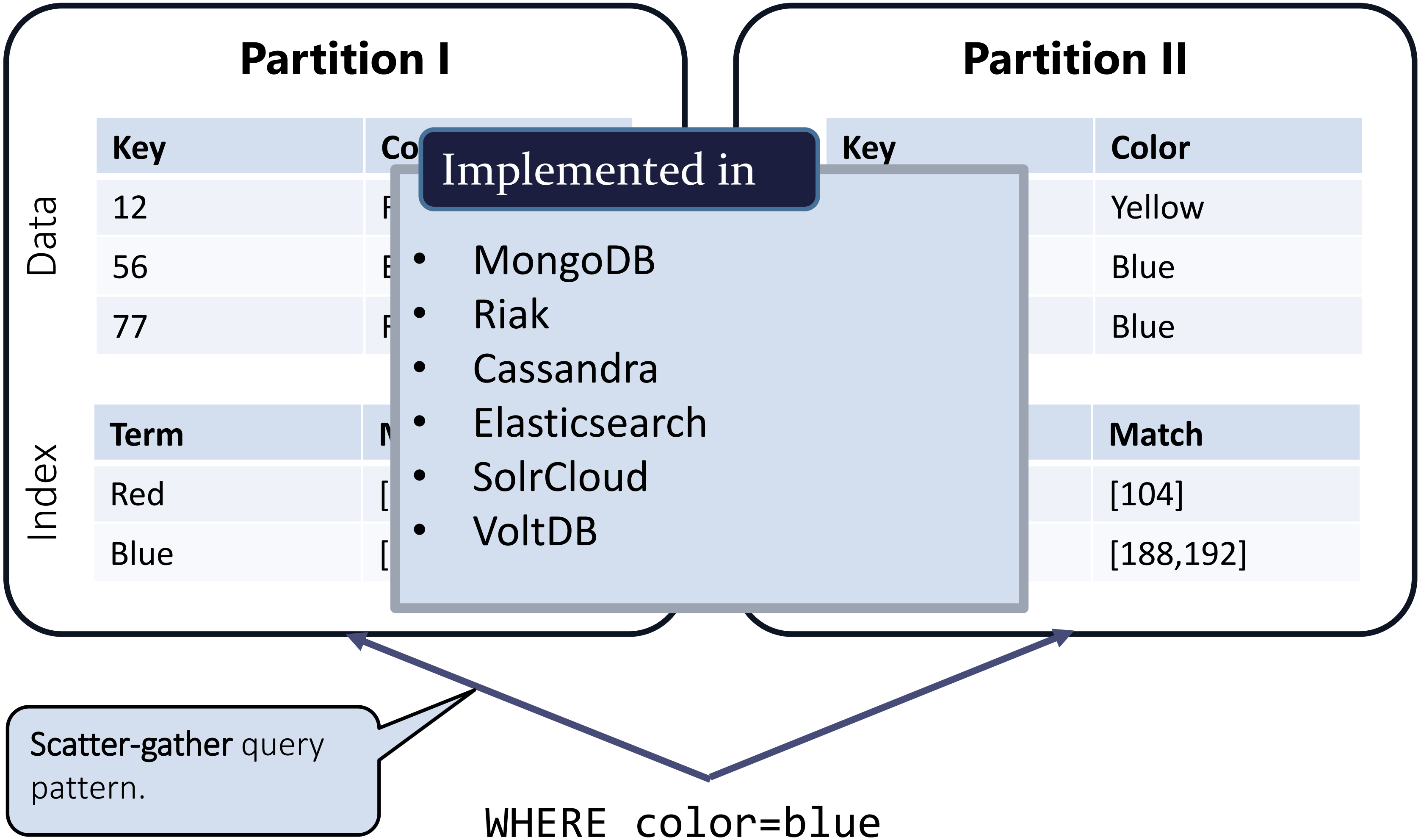
In a Nutshell




Functional
Techniques
Non-Functional
Joins
Sorting
Filter Queries
Full-text Search
Aggregation and Analytics
Query Processing
Global Secondary Indexing
Local Secondary Indexing
Query Planning
Analytics Framework
Materialized Views
Read Latency

Local Secondary Indexing

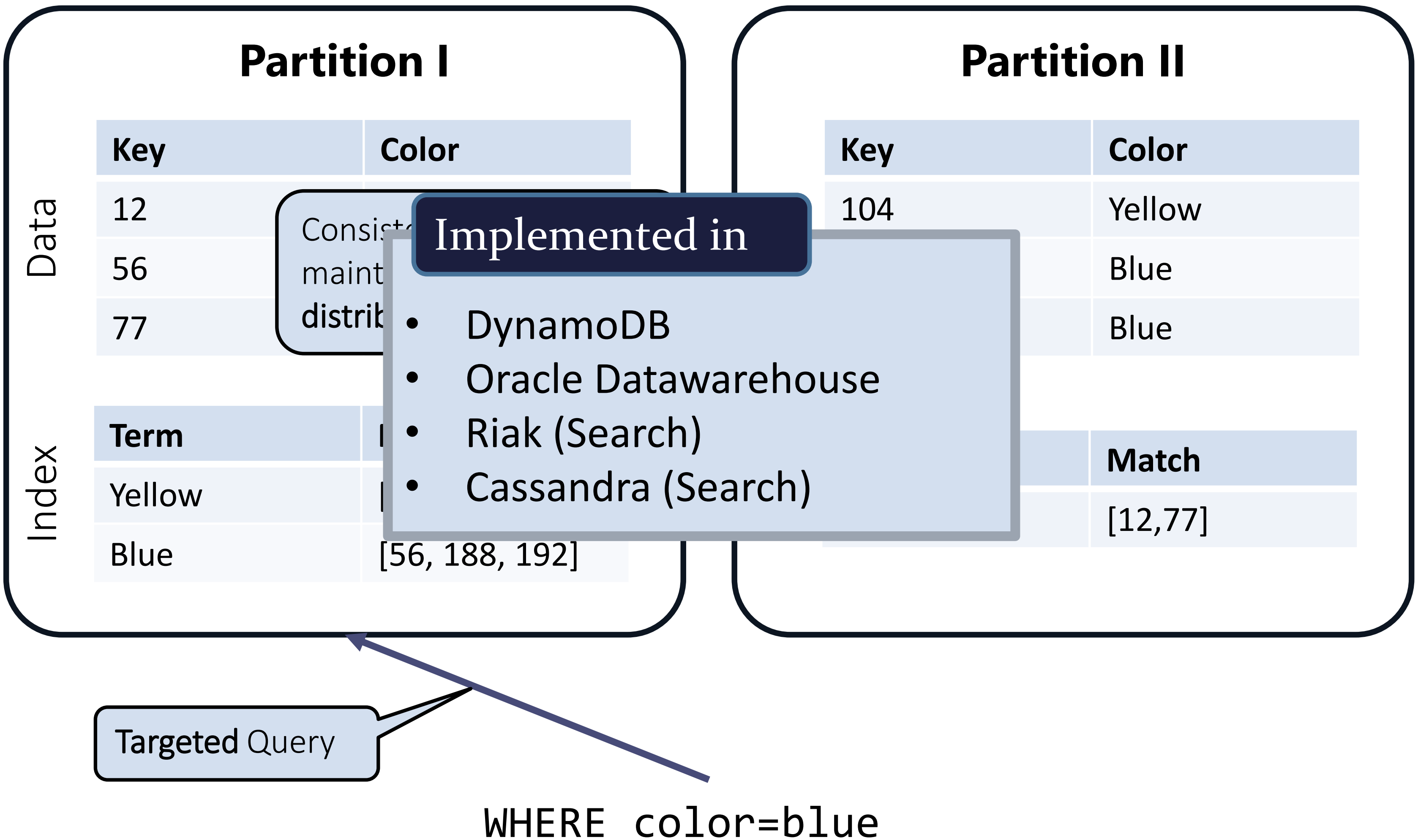
Partitioning By Document



Kleppmann, Martin. "Designing data-intensive applications." (2016).

Global Secondary Indexing

Partitioning By Term

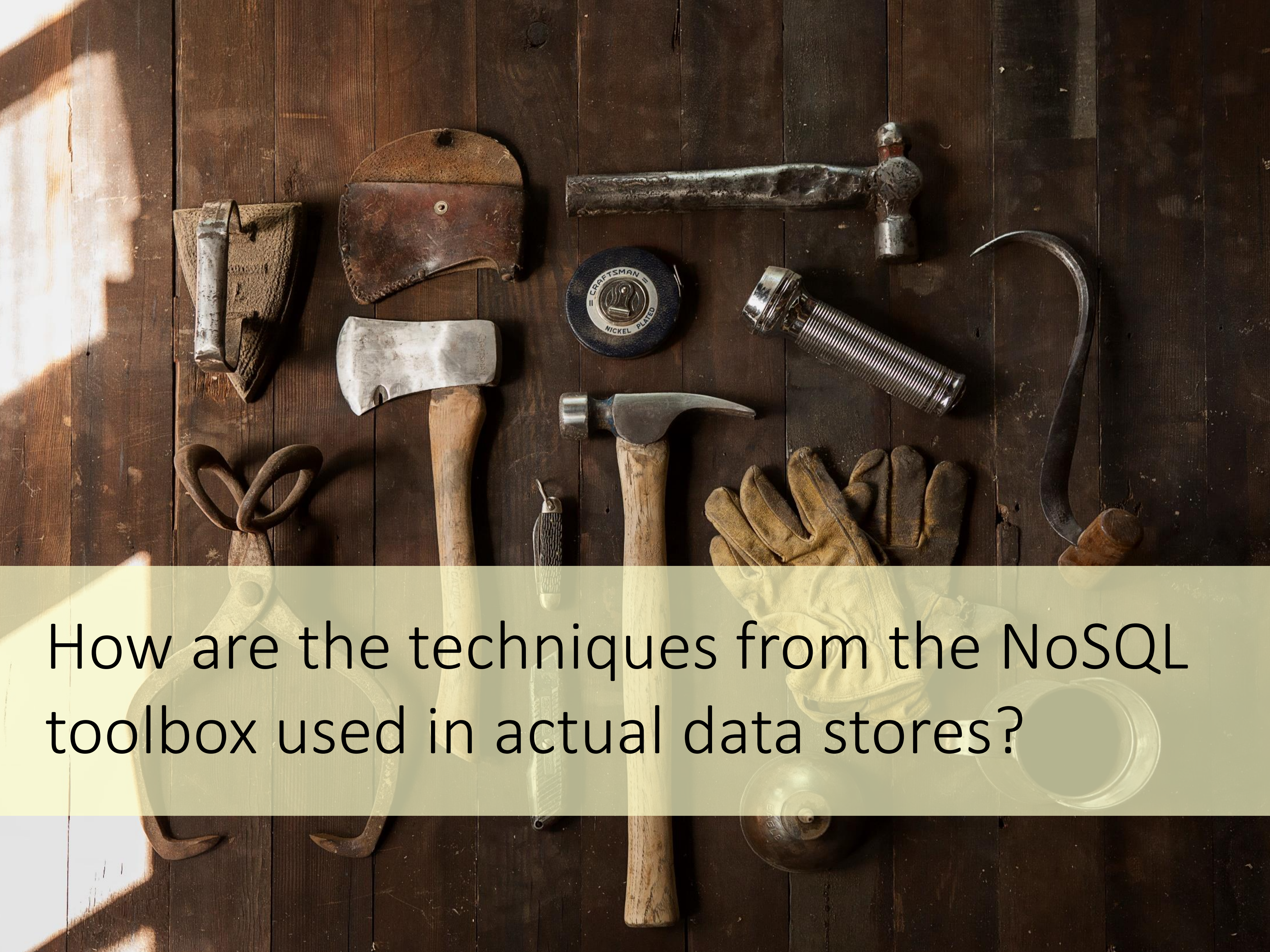


Kleppmann, Martin. "Designing data-intensive applications." (2016).

Query Processing Techniques

Summary

- **Local Secondary Indexing:** Fast writes, scatter-gather queries
- **Global Secondary Indexing:** Slow or inconsistent writes, fast queries
- **(Distributed) Query Planning**: scarce in NoSQL systems but increasing (e.g. left-outer equi-joins in MongoDB and θ-joins in RethinkDB)
- **Analytics Frameworks**: fallback for missing query capabilities
- **Materialized Views**: similar to global indexing



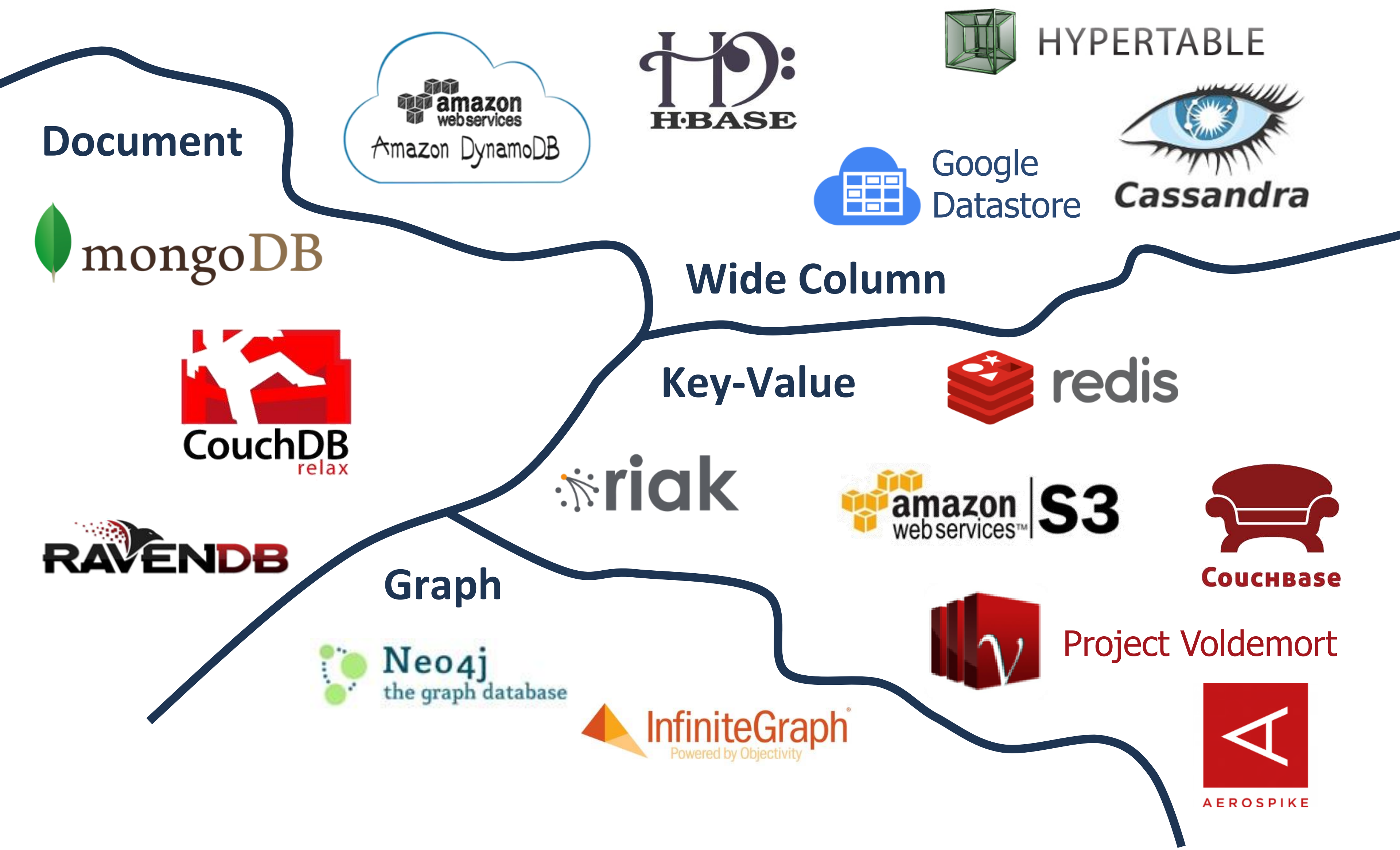
How are the techniques from the NoSQL toolbox used in actual data stores?

Outline

- NoSQL Foundations and Motivation
- The NoSQL Toolbox: Common Techniques
- NoSQL Systems
- Decision Guidance: NoSQL Decision Tree

- Overview & Popularity
- Core Systems:
  - Dynamo
  - BigTable
- Riak
- HBase
- Cassandra
- Redis
- MongoDB

NoSQL Landscape



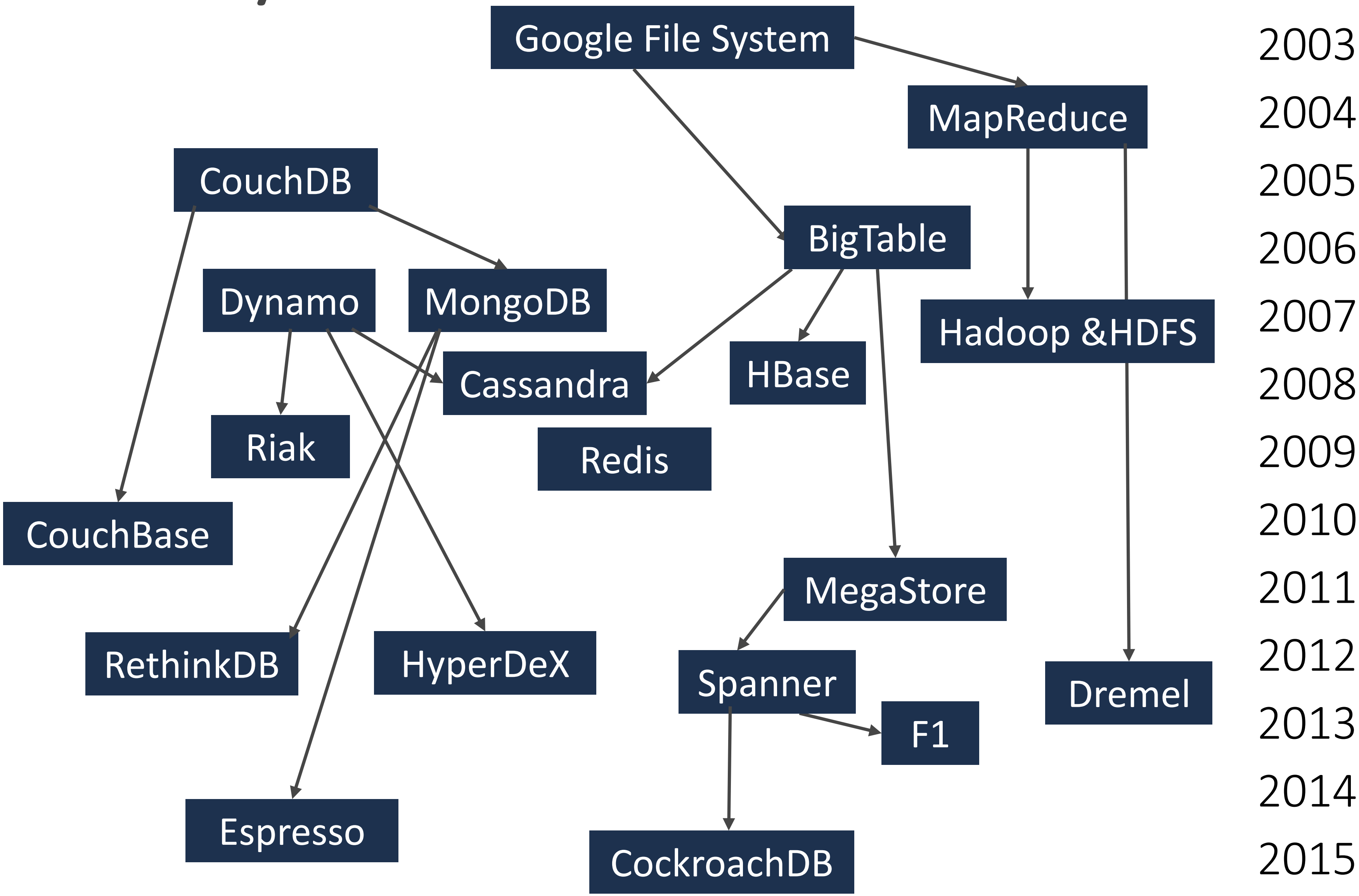
Popularity

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

#	System	Model	Score
1.	Oracle	Relational DBMS	1462.02
2.	MySQL	Relational DBMS	1371.83
3.	MS SQL Server	Relational DBMS	1142.82
4.	**MongoDB**	**Document store**	**320.22**
5.	PostgreSQL	Relational DBMS	307.61
6.	DB2	Relational DBMS	185.96
7.	**Cassandra**	**Wide column store**	**134.50**
8.	Microsoft Access	Relational DBMS	131.58
9.	**Redis**	**Key-value store**	**108.24**
10.	SQLite	Relational DBMS	107.26
11.	**Elasticsearch**	**Search engine**	**86.31**
12.	Teradata	Relational DBMS	73.74
13.	SAP Adaptive Server	Relational DBMS	71.48
14.	**Solr**	**Search engine**	**65.62**
15.	**HBase**	**Wide column store**	**51.84**
16.	Hive	Relational DBMS	47.51
17.	FileMaker	Relational DBMS	46.71
18.	**Splunk**	**Search engine**	**44.31**
19.	SAP HANA	Relational DBMS	41.37
20.	MariaDB	Relational DBMS	33.97
21.	**Neo4j**	**Graph DBMS**	**32.61**
22.	Informix	Relational DBMS	30.58
23.	**Memcached**	**Key-value store**	**27.90**
24.	**Couchbase**	**Document store**	**24.29**
25.	**Amazon DynamoDB**	**Multi-model**	**23.60**

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
RethinkDB
HyperDeX
Spanner
Dremel
F1
Espresso
CockroachDB
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015

NoSQL foundations

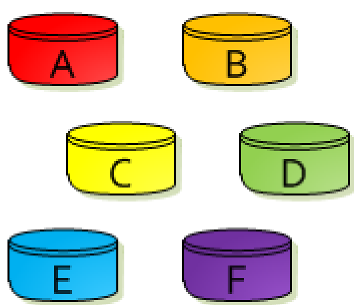
- **BigTable** (2006, Google)
  - Consistent, Partition Tolerant
  - Wide-Column data model
  - Master-based, fault-tolerant, large clusters (1.000+ Nodes), **HBase**, **Cassandra**, **HyperTable, Accumolo**
- **Dynamo** (2007, Amazon)
  - Available, Partition tolerant
  - Key-Value interface
  - Eventually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

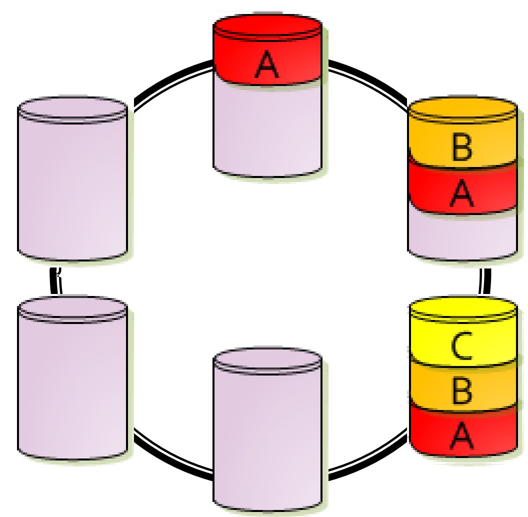
Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."

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 replicated **N** times

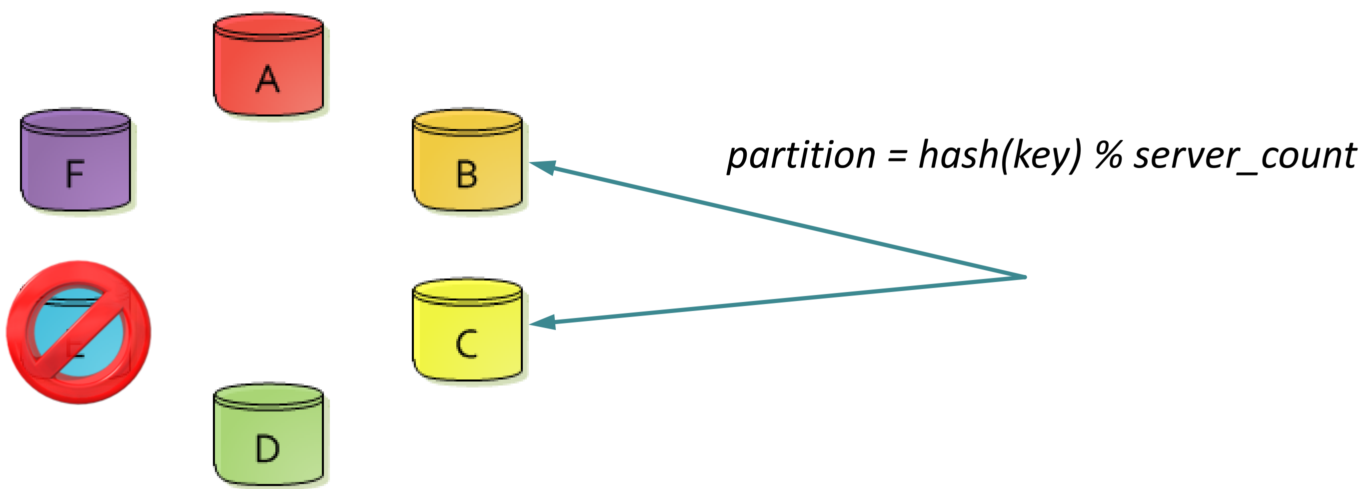




DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."

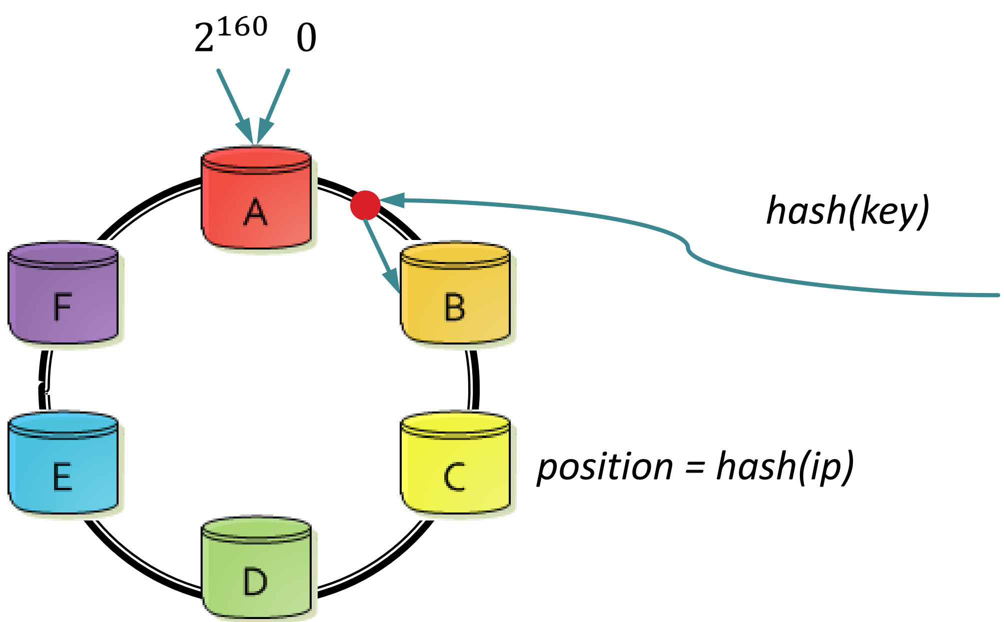
Consistent Hashing

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



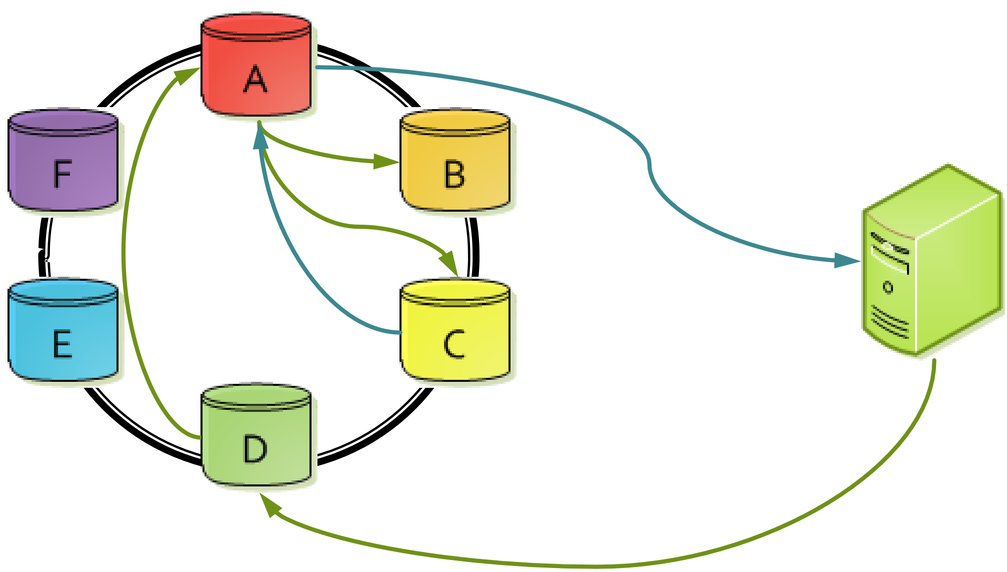
Consistent Hashing

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



Reading and Writing

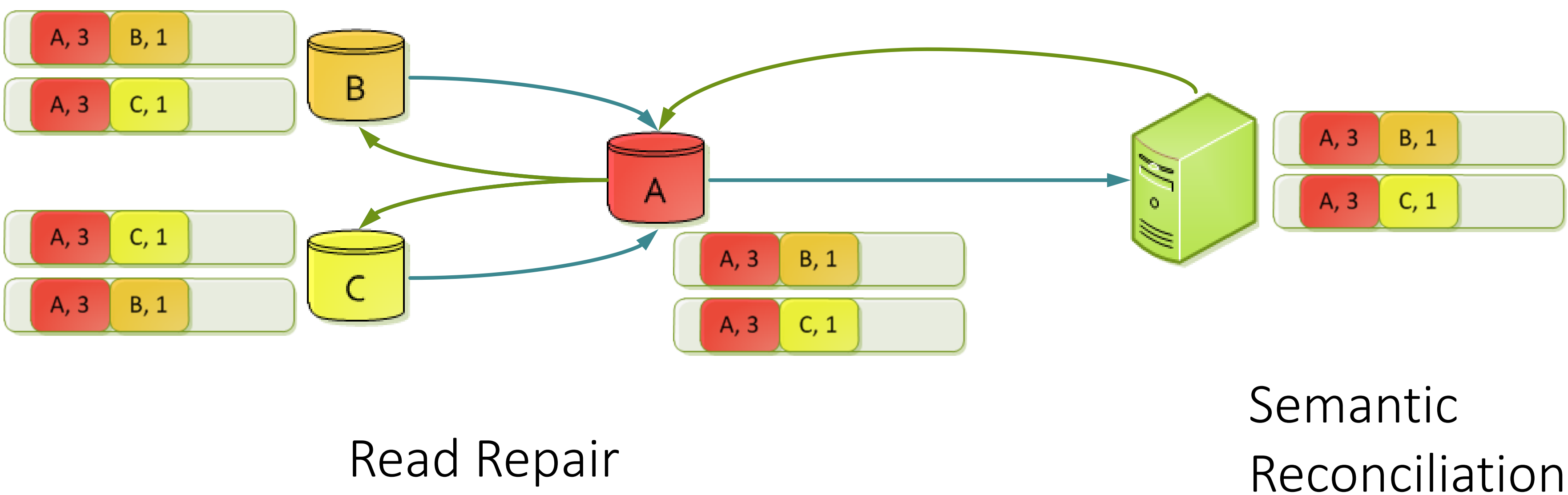
- An arbitrary node acts as a coordinator
- **N**: number of replicas
- **R**: number of nodes that need to confirm a read
- **W**: number of nodes that need to confirm a write



N=3
R=2
W=1

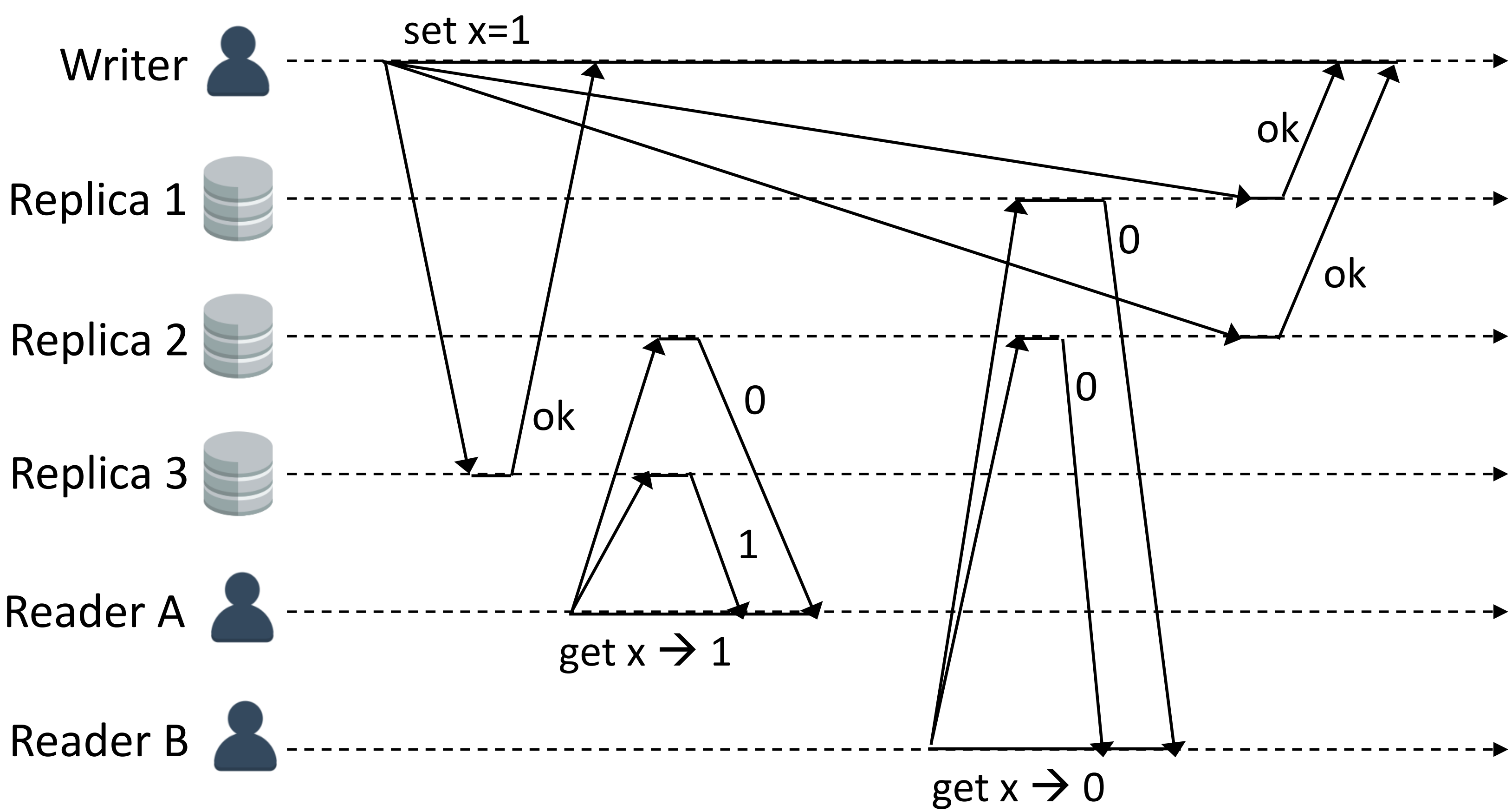
Versioning and Consistency

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



$R + W > N$ does not imply linearizability

- Consider the following execution:

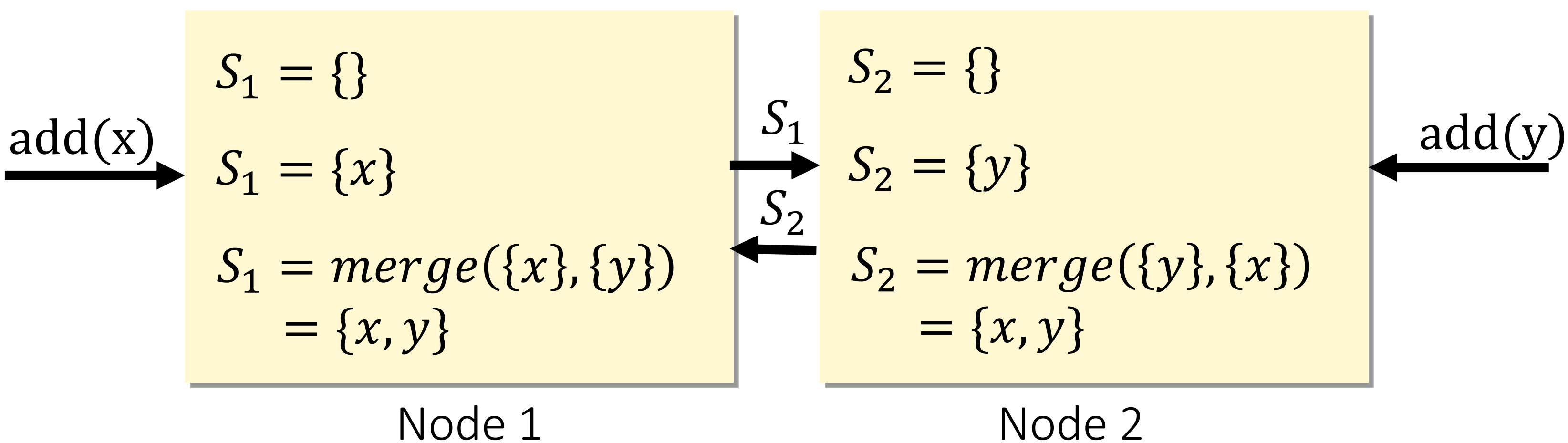


Kleppmann, Martin. "Designing data-intensive applications." (2016).

CRDTs

Convergent/Commutative Replicated Data Types

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

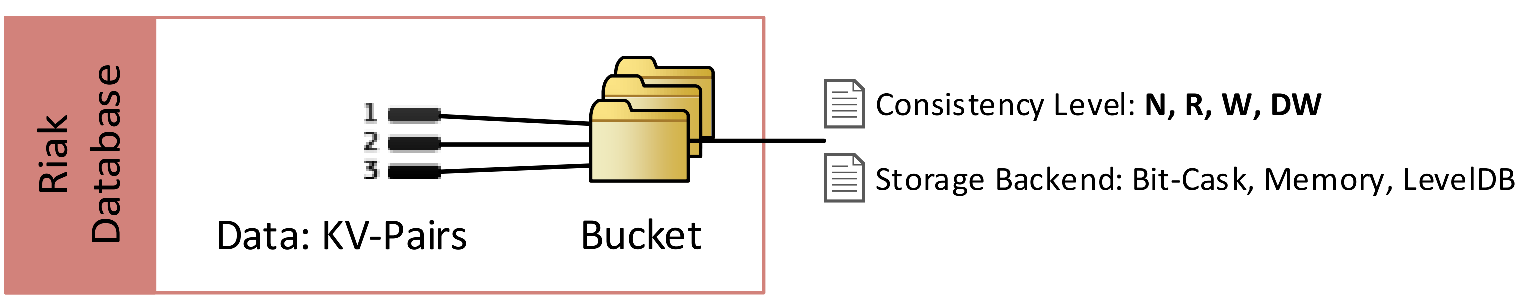


Marc Shapiro, Nuno Preguica, Carlos Baquero, and Marek Zawirski "Conflict-free Replicated Data Types"

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
  - **Riak CS:** S3-like file storage, **Riak TS**: time-series database



Summary: Dynamo and Riak

- Available and Partition-Tolerant
- **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
- **Vector Clocks**: concurrent modification can be detected, inconsistencies are healed by the application
- **API**: Create, Read, Update, Delete (CRUD) on key-value pairs
- **Riak**: Open-Source Implementation of the Dynamo paper

Dynamo and Riak

Classification

Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views

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 per node
- Redis Cluster adds sharding

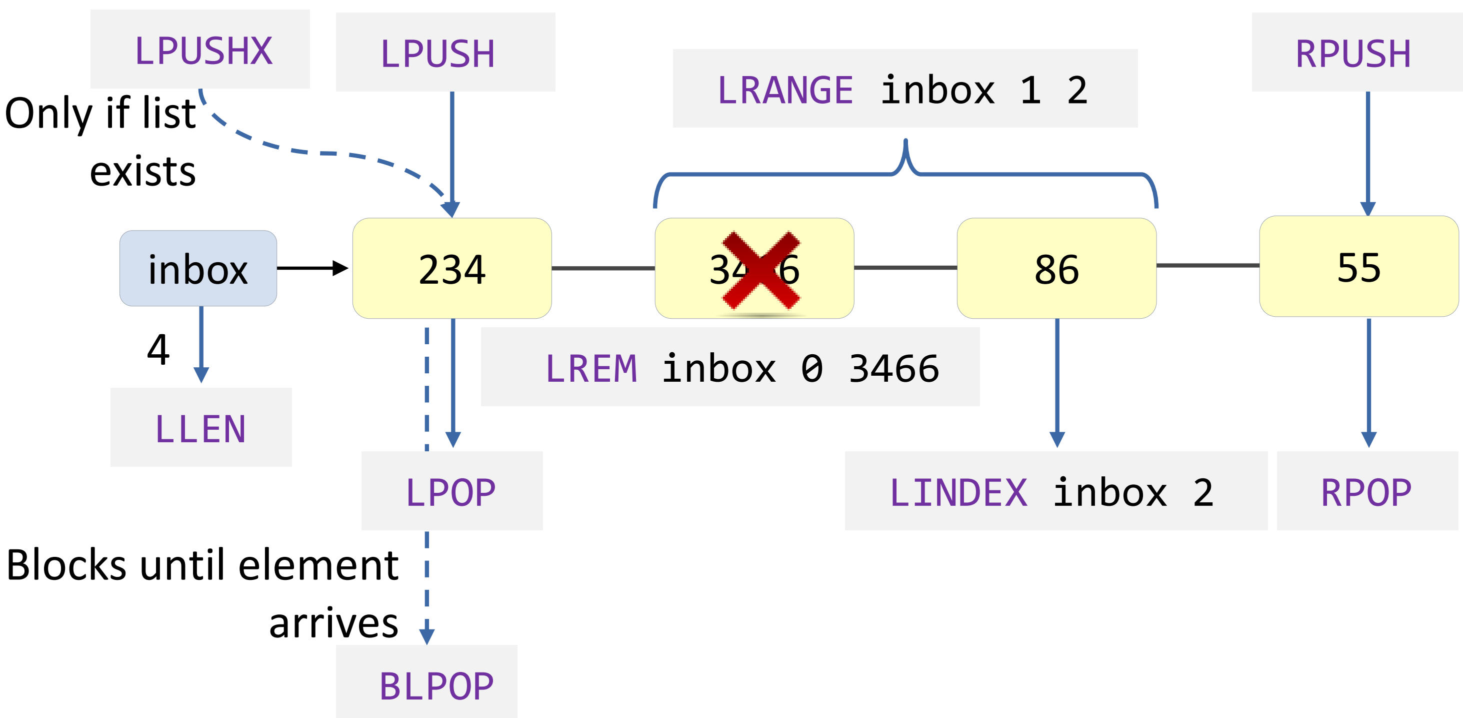
Data structures

- 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 : …"

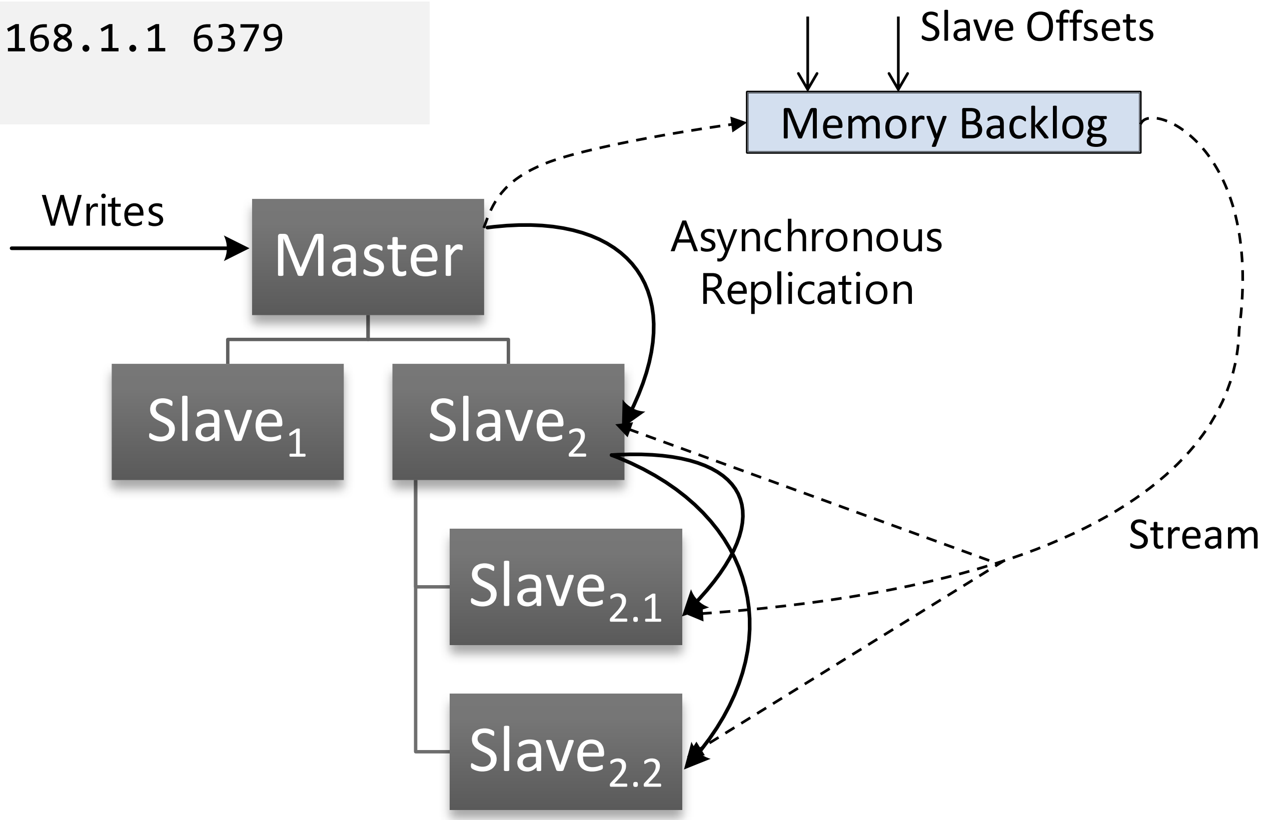
Example Redis Data Structure: lists

- (Linked) Lists:

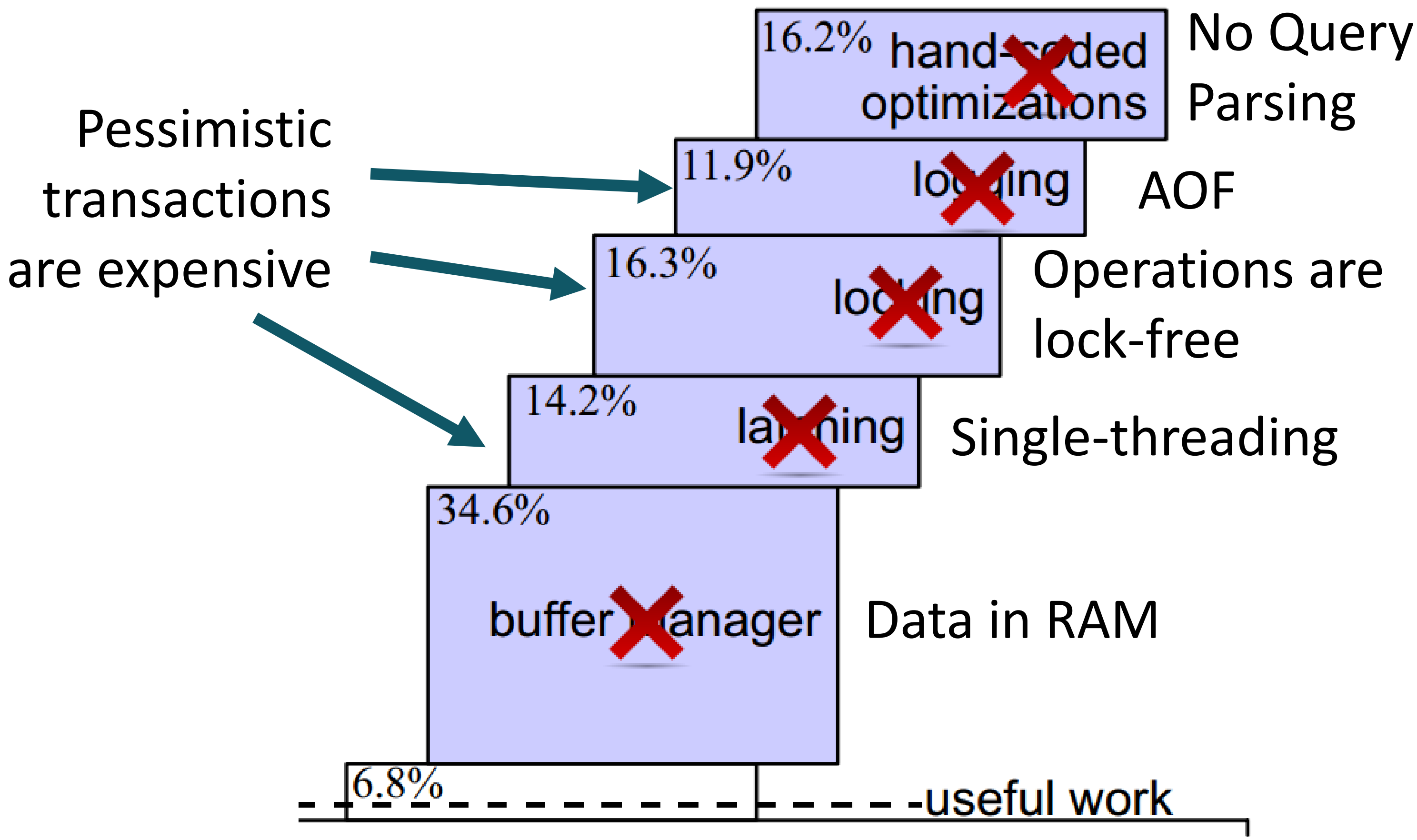


Master-Slave Replication

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



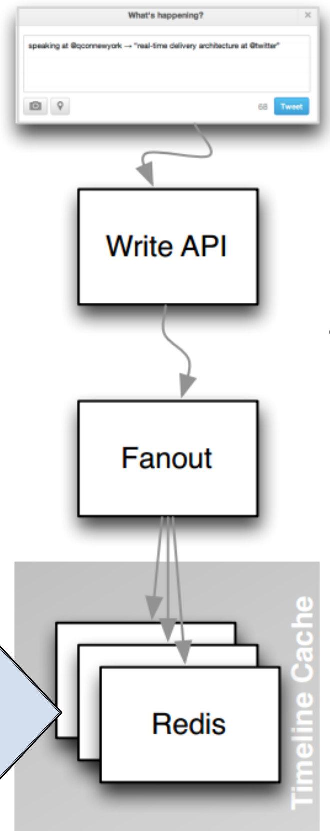
Why is Redis so fast?



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

Example Redis Use-Case: Twitter

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



>150 million users
~300k timeline querys/s

```
RPUSHX user_id tweet
```

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

Classification: Redis

Techniques

Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views

Google BigTable (CP)

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

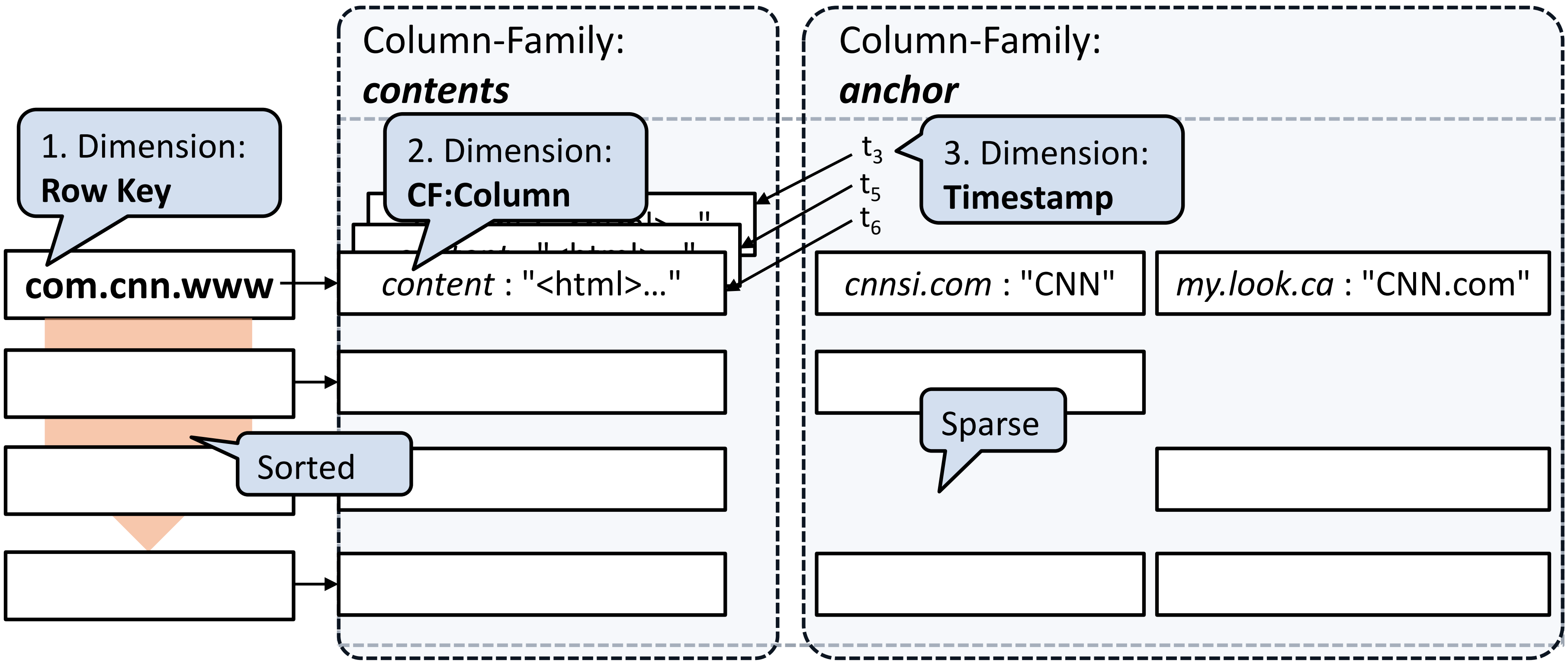
> A Bigtable is a sparse, distributed, persistent multidimensional sorted map.

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

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

Wide-Column Data Modelling

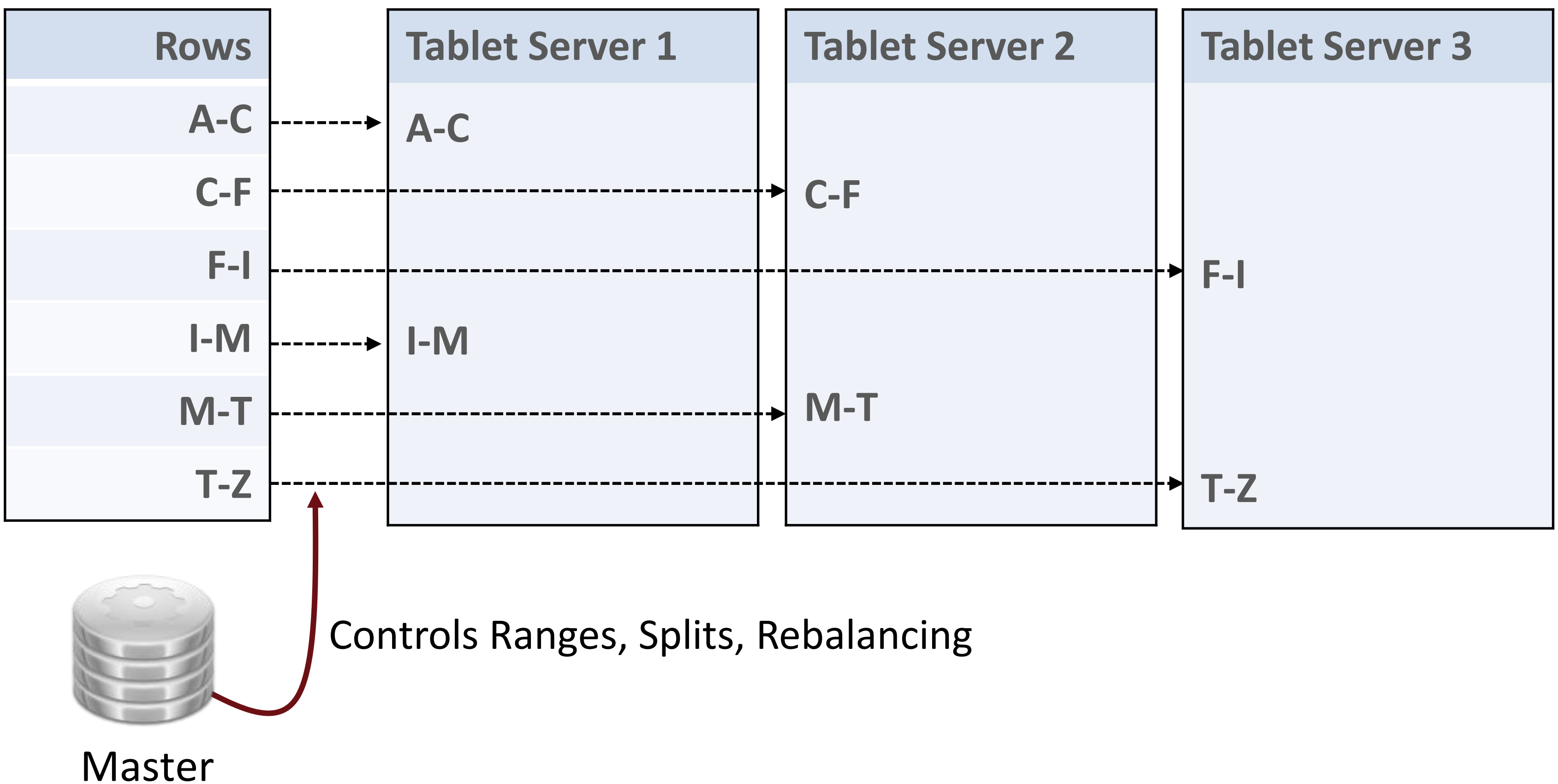
- Storage of crawled web-sites („Webtable“):



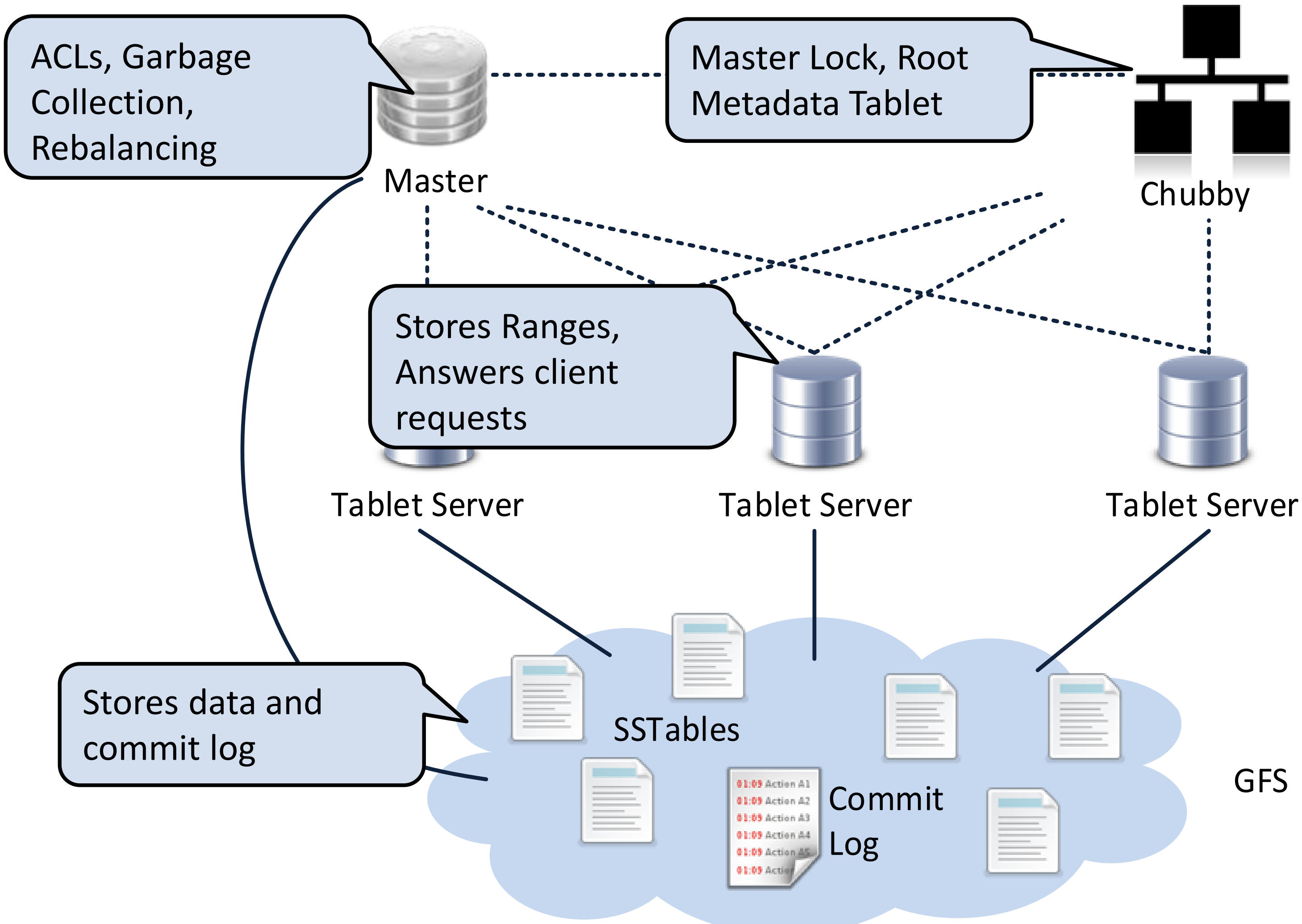
Range-based Sharding

BigTable Tablets

Tablet: Range partition of ordered records

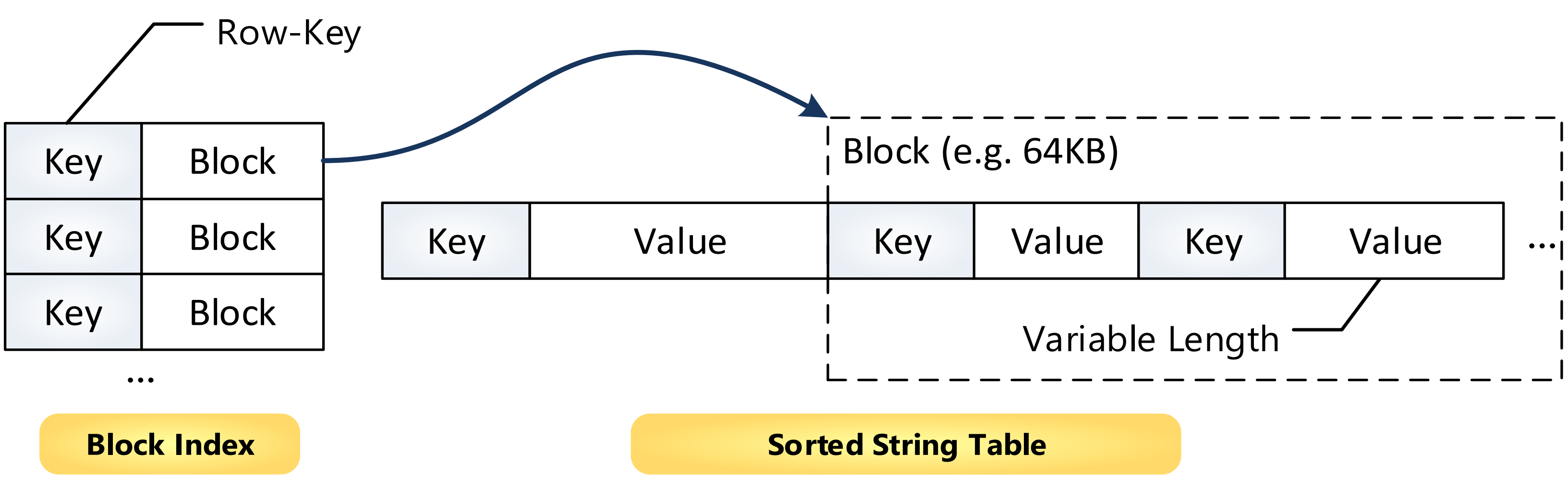


Architecture



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



Apache HBase (CP)

- 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

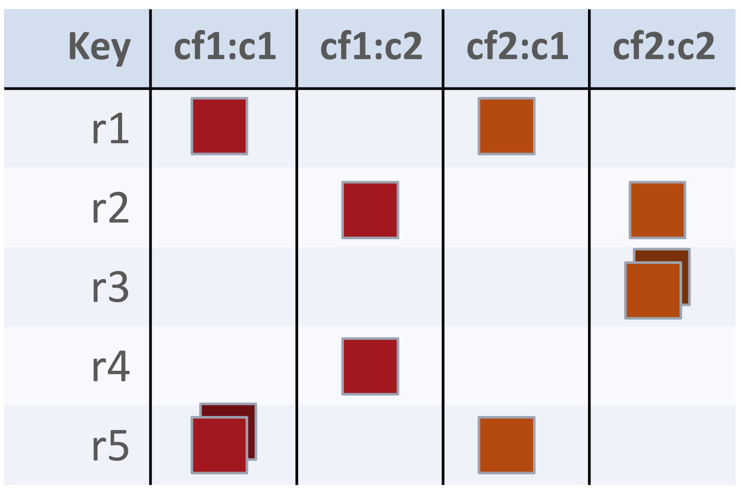
HBase
Model:
Wide-Column
License:
Apache 2
Written in:
Java

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.

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	

Atomic HBase **Counter**

CF:Daily

CF:Monthly

CF:All

TTL – automatic deletion of old rows

Lars George: “Advanced HBase Schema Design”

Summary: BigTable, HBase

- 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 BigTable implementation

Classification: HBase

Techniques

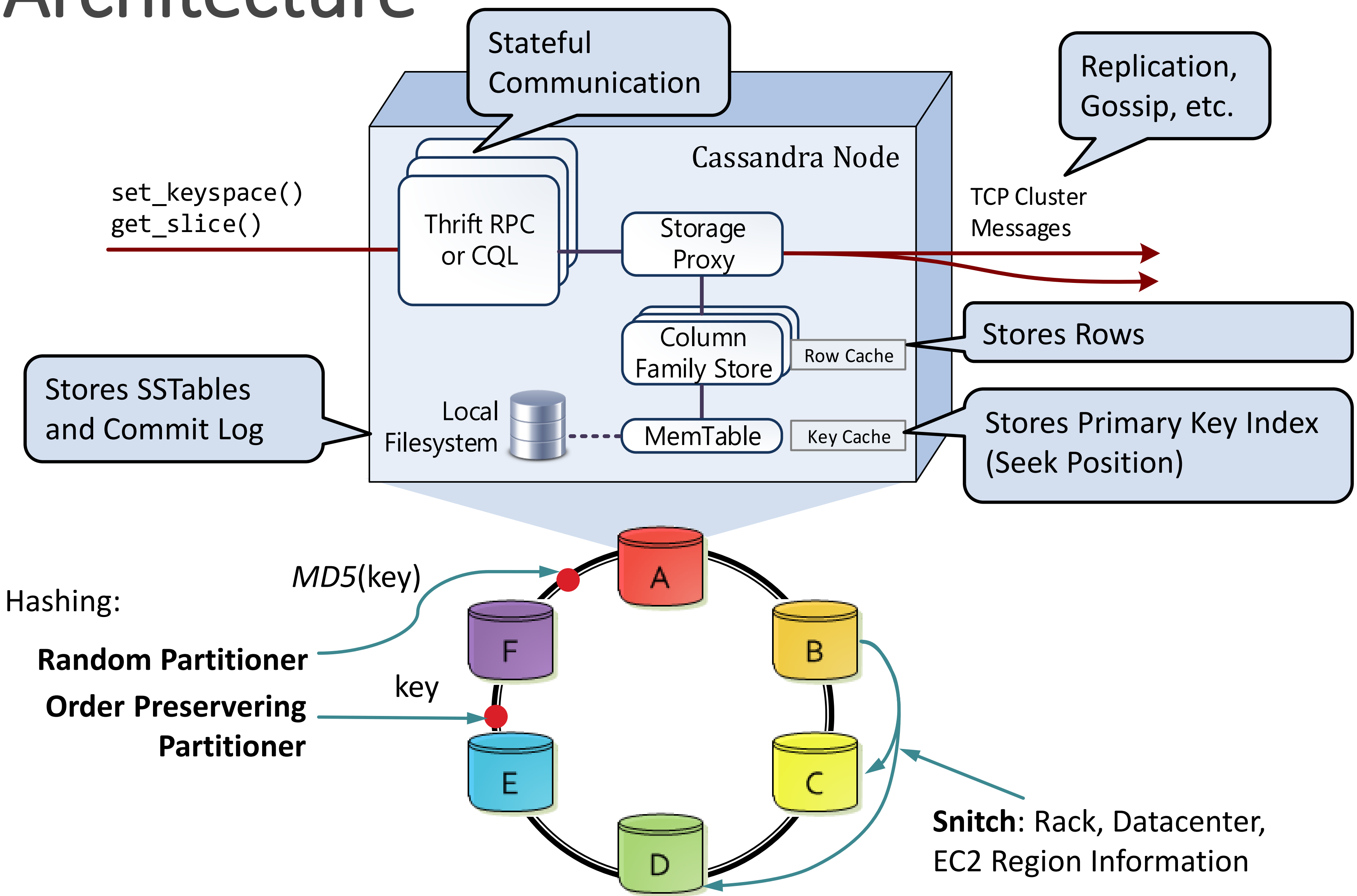
Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views

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



Classification: Cassandra

Techniques

Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views

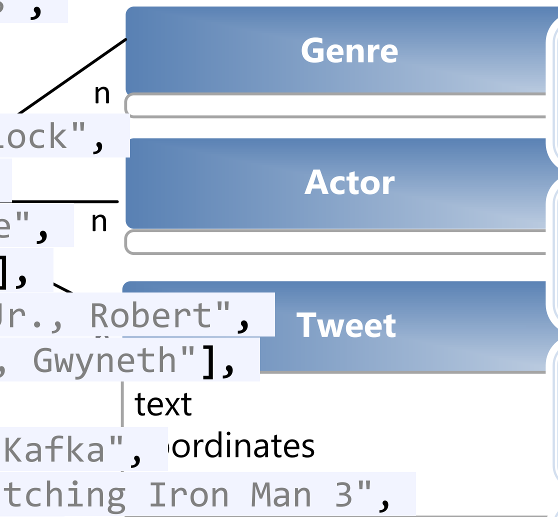
MongoDB (CP)

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

- 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*)
  - **Storage Engines:** memory-mapped files, in-memory, Log-structured merge trees (WiredTiger)

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

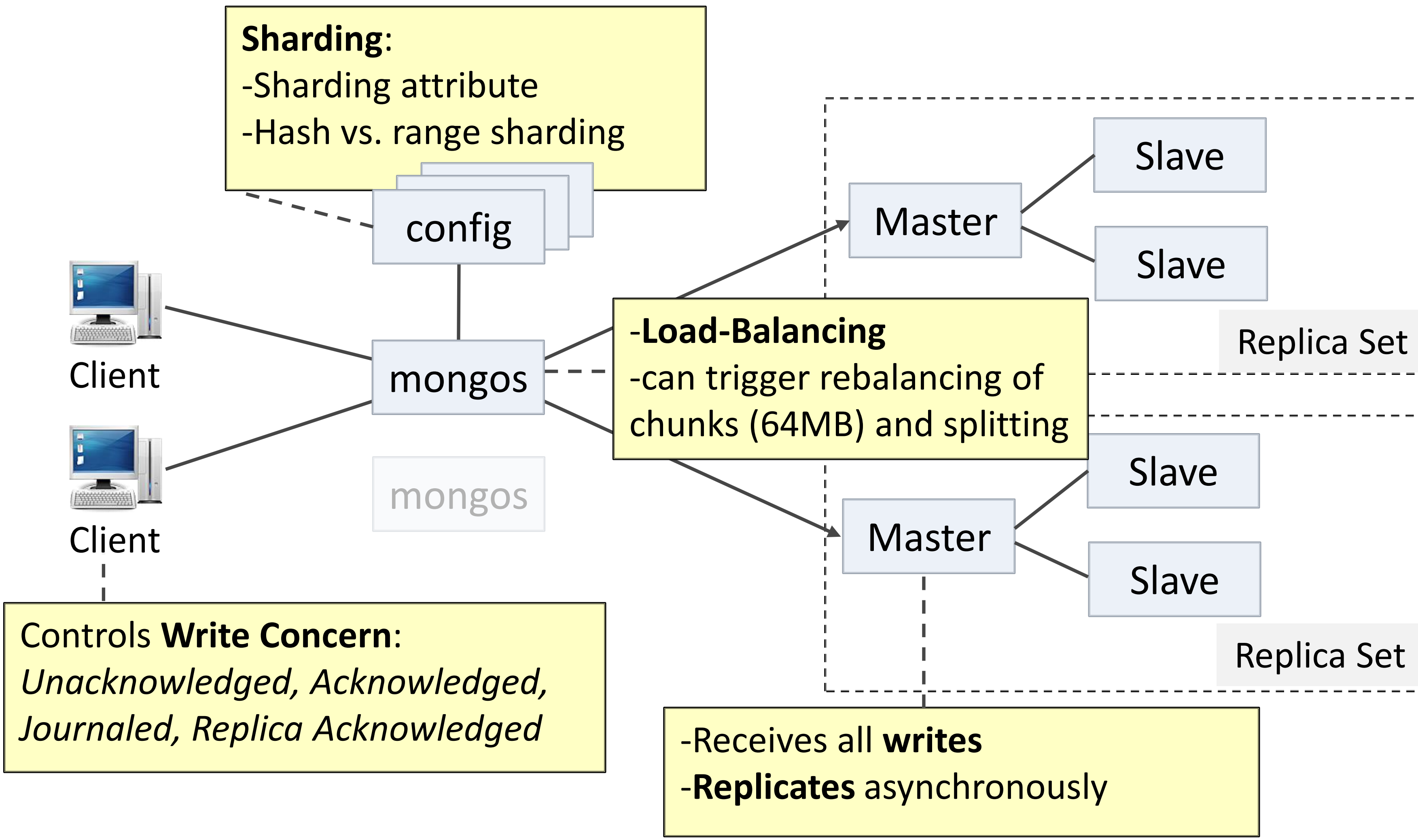
Nesting replaces 1:n and 1:1 relations

Schemafreeness: Attributes per document

Unit of atomicity: document

Principles

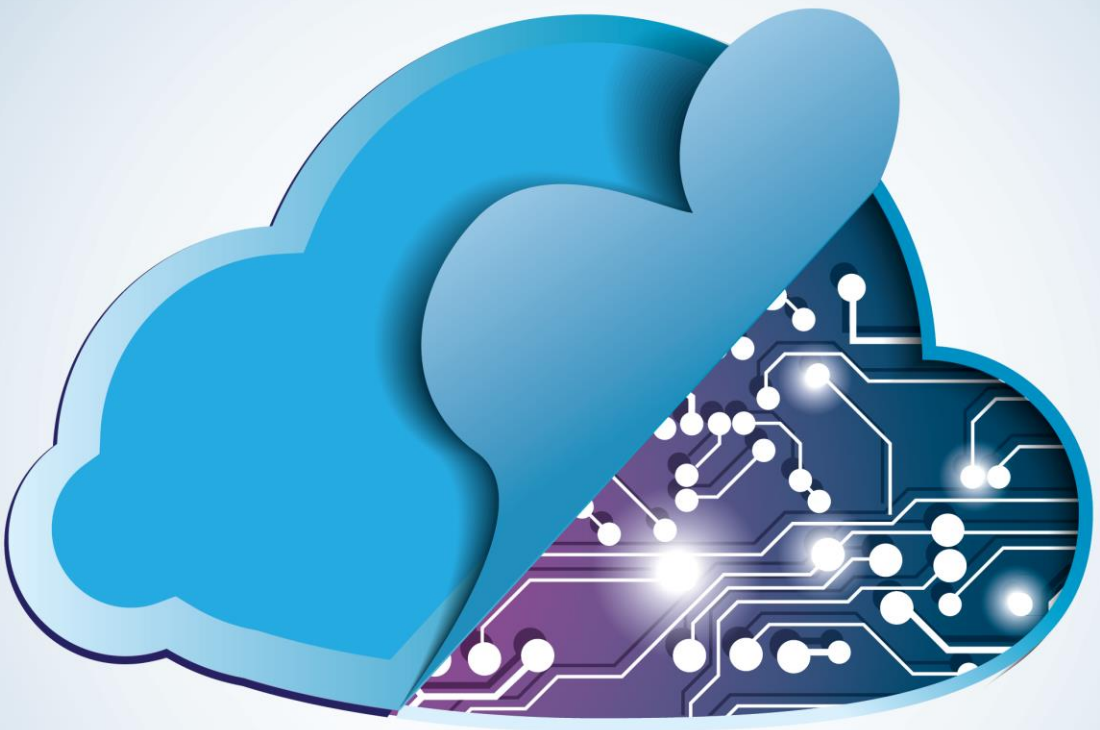
Sharding und Replication



Classification: MongoDB

Techniques

Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views



How can the choices for an appropriate system be narrowed down?

Outline

NoSQL Foundations and Motivation

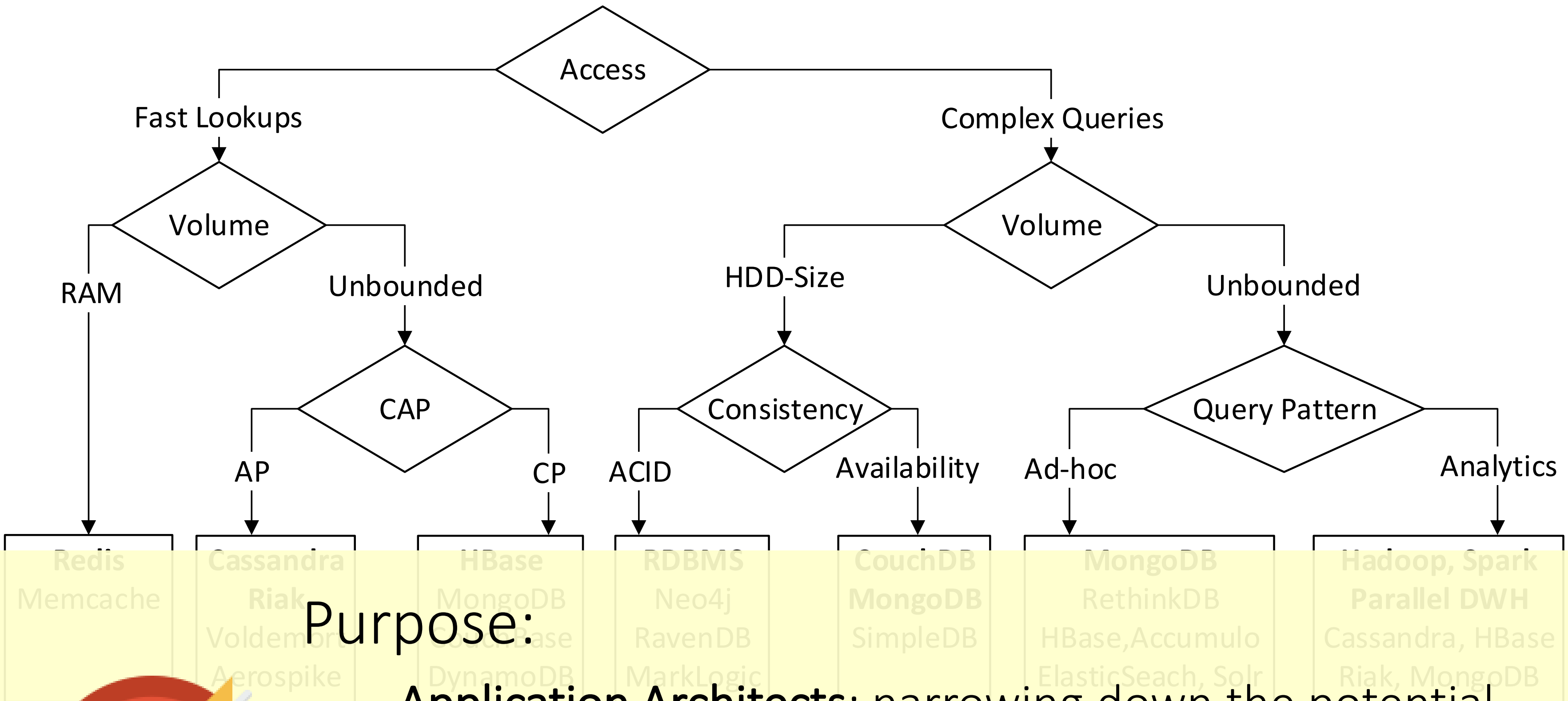
The NoSQL Toolbox: Common Techniques

NoSQL Systems

Decision Guidance: NoSQL Decision Tree

- Decision Tree
- Classification Summary
- Literature Reommendations

NoSQL Decision Tree



Purpose:

Application Architects: narrowing down the potential system candidates based on requirements

Database Vendors/Researchers: clear communication and design of system trade-offs

System Properties

According to the NoSQL Toolbox

- For fine-grained system selection:

	Functional Requirements							
	Scan Queries	ACID Transactions	Conditional Writes	Joins	Sorting	Filter Query	Full-Text Search	Analytics
Mongo	x		x		x	x	x	x
Redis	x	x	x					
HBase	x		x		x			x
Riak							x	x
Cassandra	x		x		x		x	x
MySQL	x	x	x	x	x	x	x	x

System Properties

According to the NoSQL Toolbox

- For fine-grained system selection:

	Non-functional Requirements										
	Data Scalability	**Write Scalability**	**Read Scalability**	**Elasticity**	**Consistency**	**Write Latency**	**Read Latency**	**Write Throughput**	**Read Availability**	**Write Availability**	**Durability**
Mongo	x	x	x		x	x	x		x		x
Redis			x		x	x	x	x	x		x
HBase	x	x	x	x	x	x		x			x
Riak	x	x	x	x		x	x	x	x	x	x
Cassandra	x	x	x	x		x		x	x	x	x
MySQL			x		x						x

System Properties

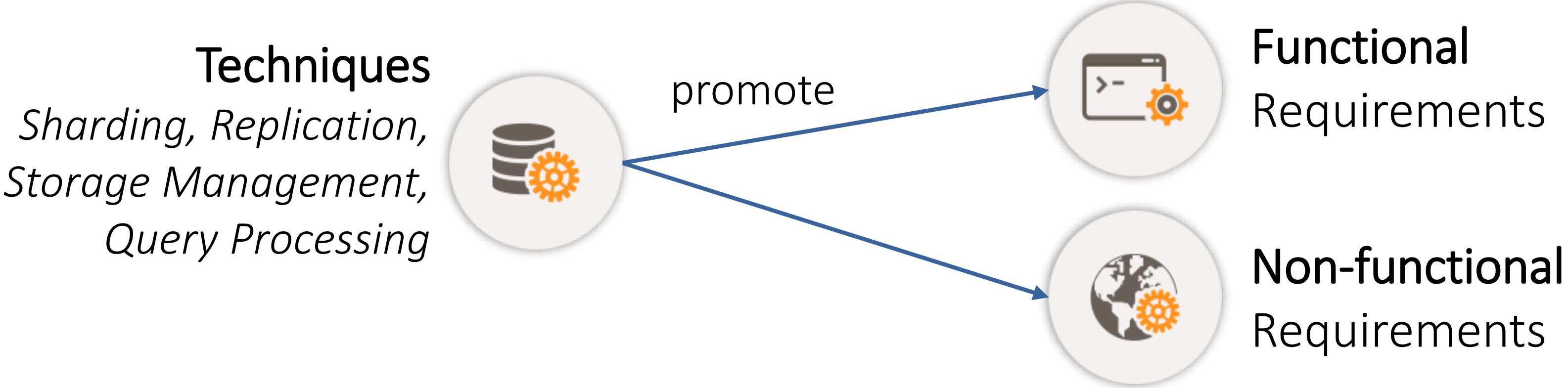
According to the NoSQL Toolbox

- For fine-grained system selection:

	Techniques																			
	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared-Disk	Transaction Protocol	Sync. Replication	Async. Replication	Primary Copy	Update Anywhere	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage	Global Indexing	Local Indexing	Query Planning	Analytics Framework	Materialized Views
Mongo	x	x					x	x	x		x		x	x	x		x	x	x	
Redis								x	x		x		x							
HBase	x						x		x		x		x		x					
Riak		x		x				x		x	x	x	x			x	x		x	
Cassandra		x		x				x		x	x		x		x	x	x			x
MySQL					x			x	x		x	x	x				x	x		

Summary

- High-Level NoSQL Categories:
  - Key-Value, Wide-Column, Docuement, Graph
  - Two out of {Consistent, Available, Partition Tolerant}
- The **NoSQL Toolbox**: systems use similar techniques that promote certain capabilities



- **Decision Tree**

Our NoSQL research at the University of Hamburg

Caching- and Database-as-a-Service Middleware for NoSQL databases

Orestes

BaQend

Build faster Apps faster.

Cloud Startup for Orestes as a Service

Orestes

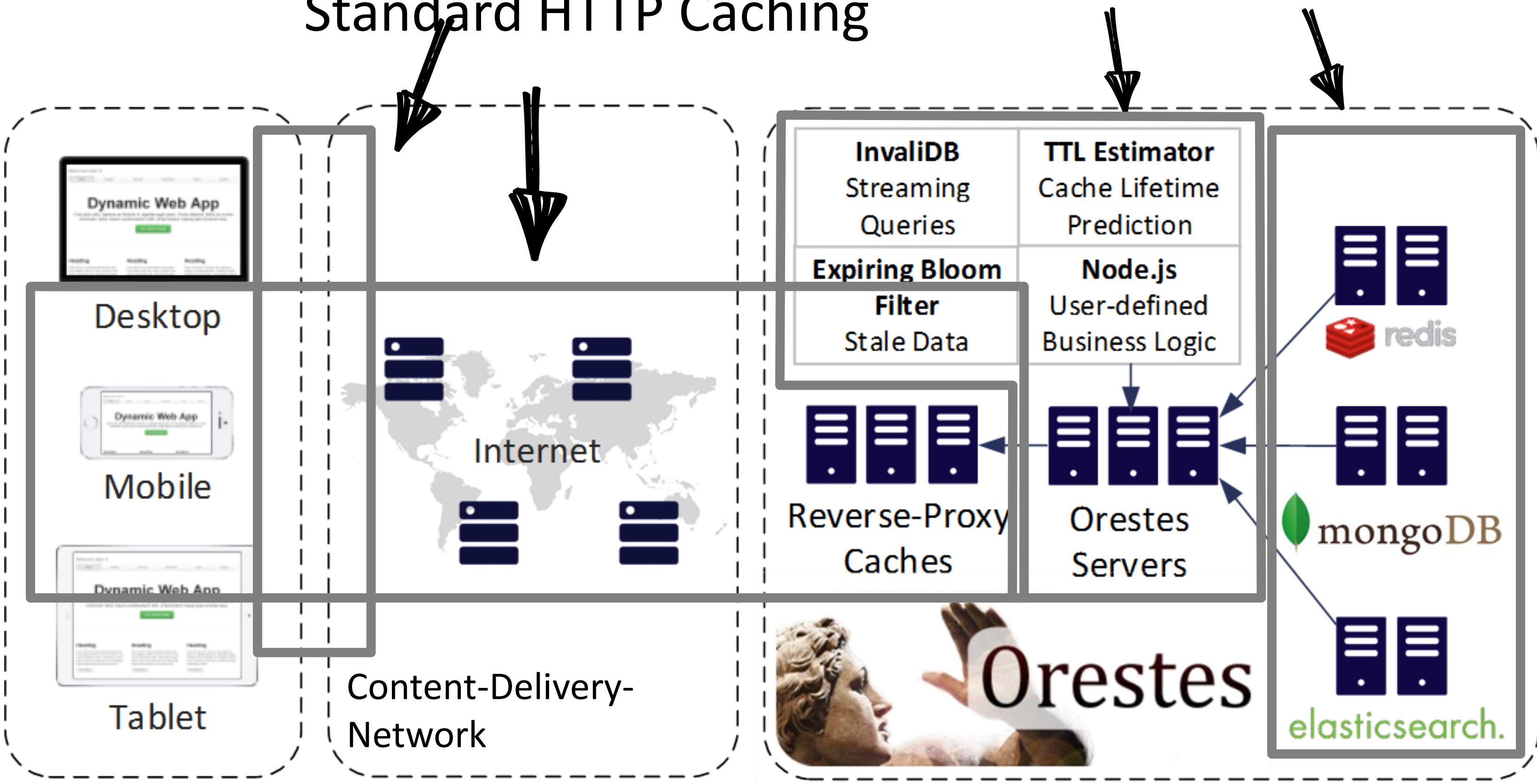
Components

Backend-as-a-Service Middleware:
Caching, Transactions, Schemas,
Invalidation Detection, …

Polyglot Persistence
Mediator

Unified REST API

Standard HTTP Caching



Orestes

As-a-Service

SCDM 2015

Home | Schedule | CFP | Submission | Contact

3rd Workshop on Scalable Cloud Data Management

Co-located with the IEEE BigData Conference.
Santa Clara, CA, October 29th 2015.
Starting at **8am in Ballroom C**.

Workshop Schedule

October 29, 2015

SCDM 2015

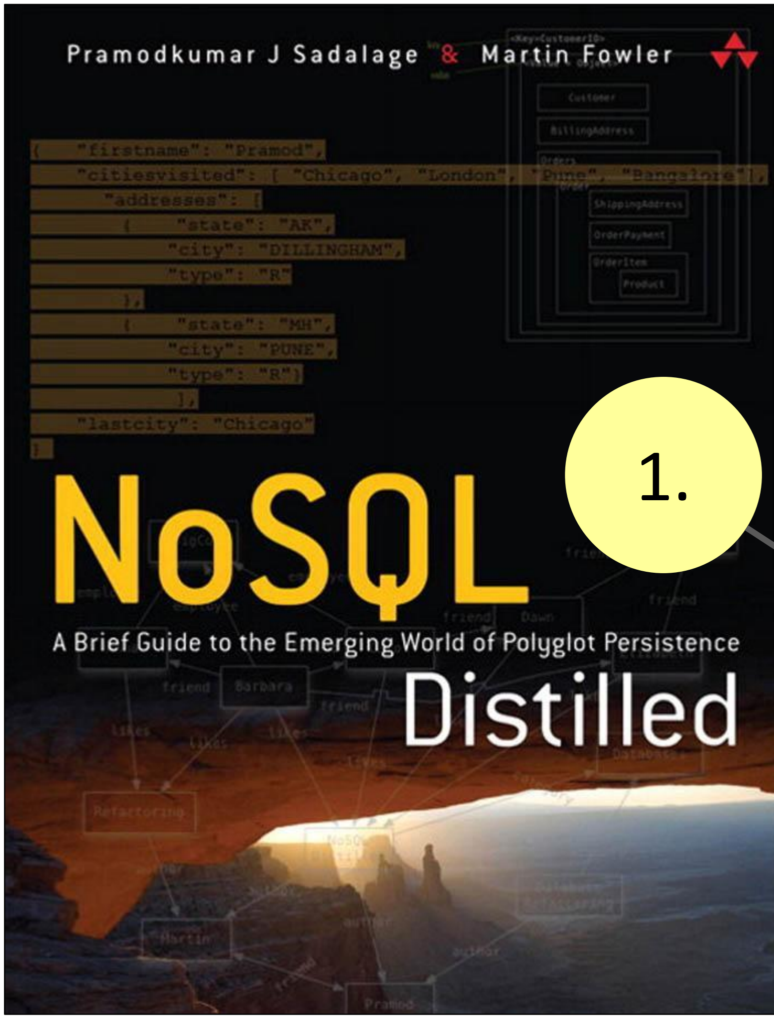
OCTOBER 5, 2015

SCDM 2015 Workshop in Santa Clara, CA. The preliminary schedule is online. SCDM starts on Oct 29, 8am (Ballroom C) with a keynote by Russel Sears on "Purity and the future of scalable storage".

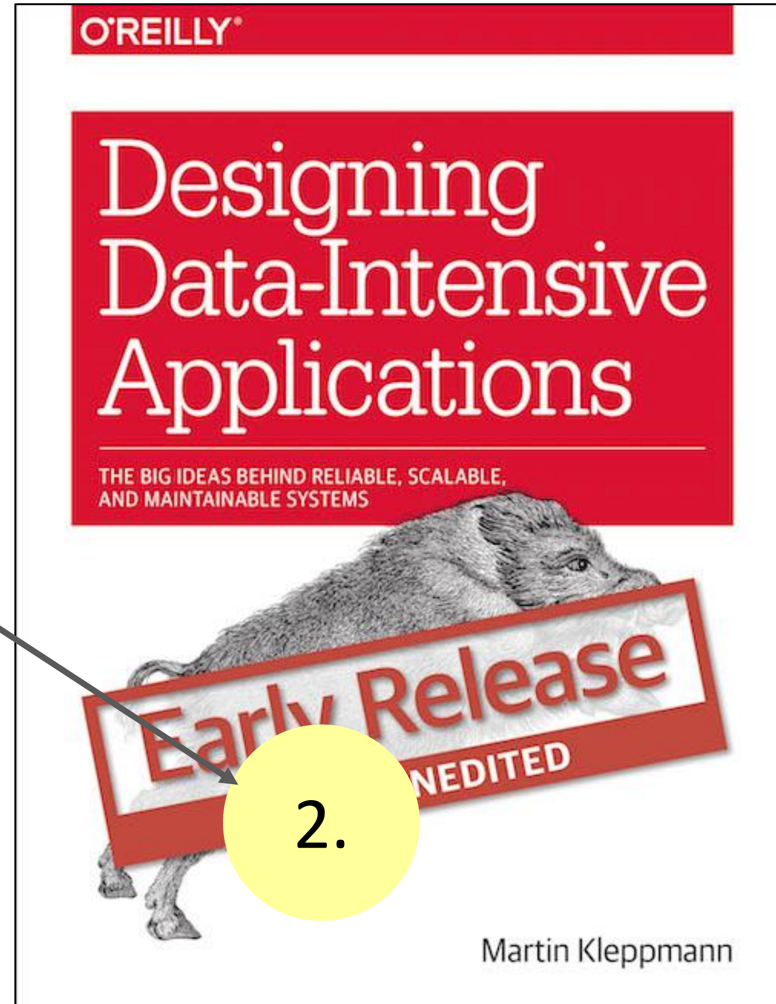
This year‘s SCDM will be announced soon

Literature Recommendations

Recommended Literature

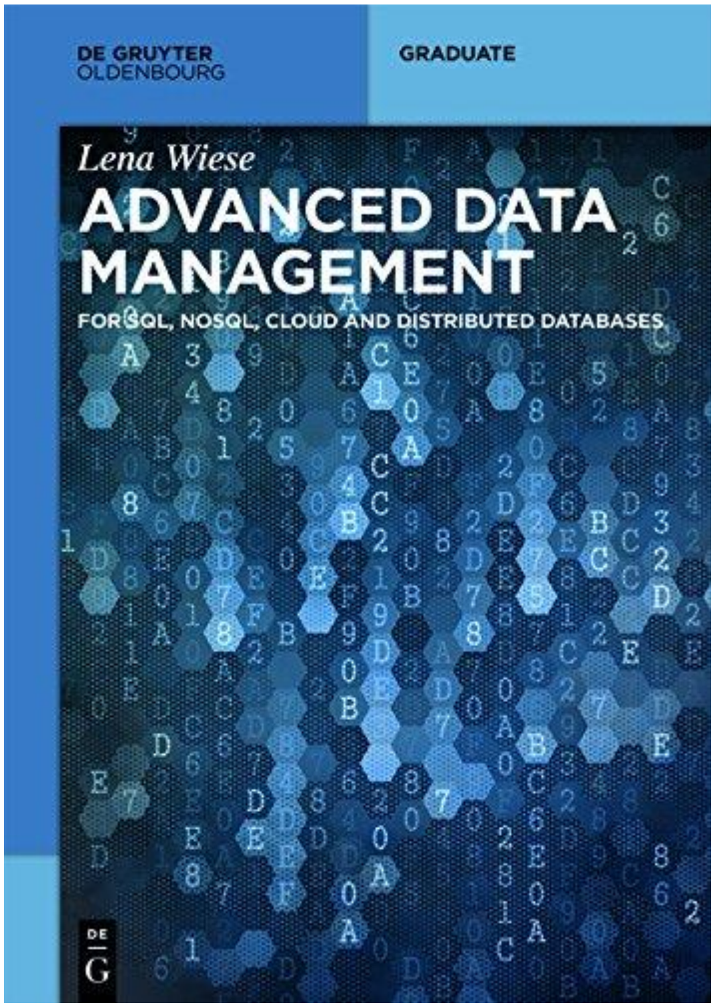


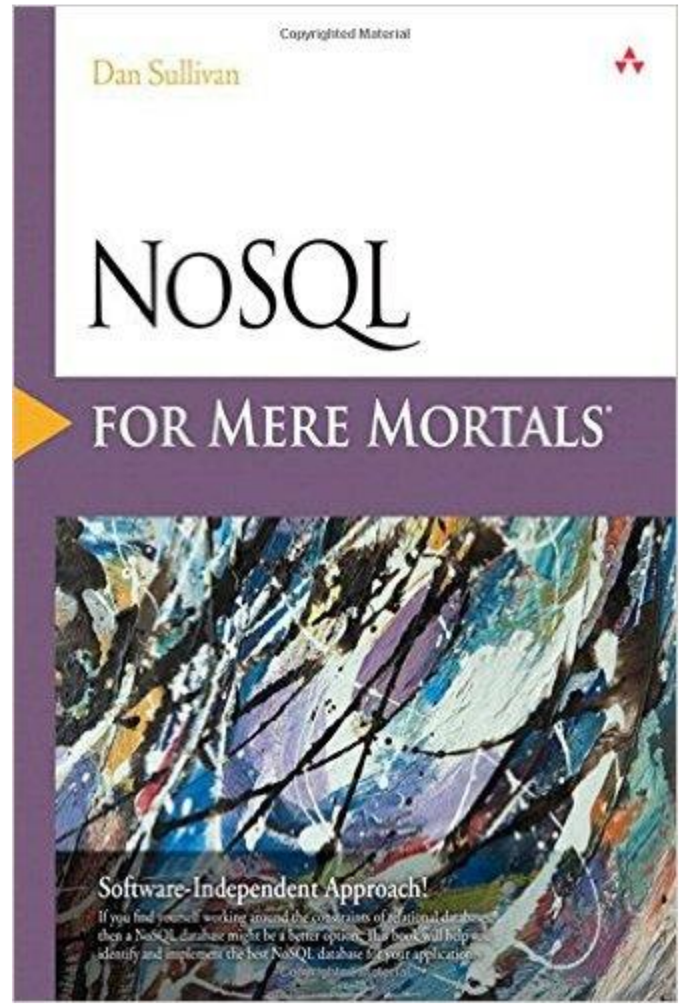
1.



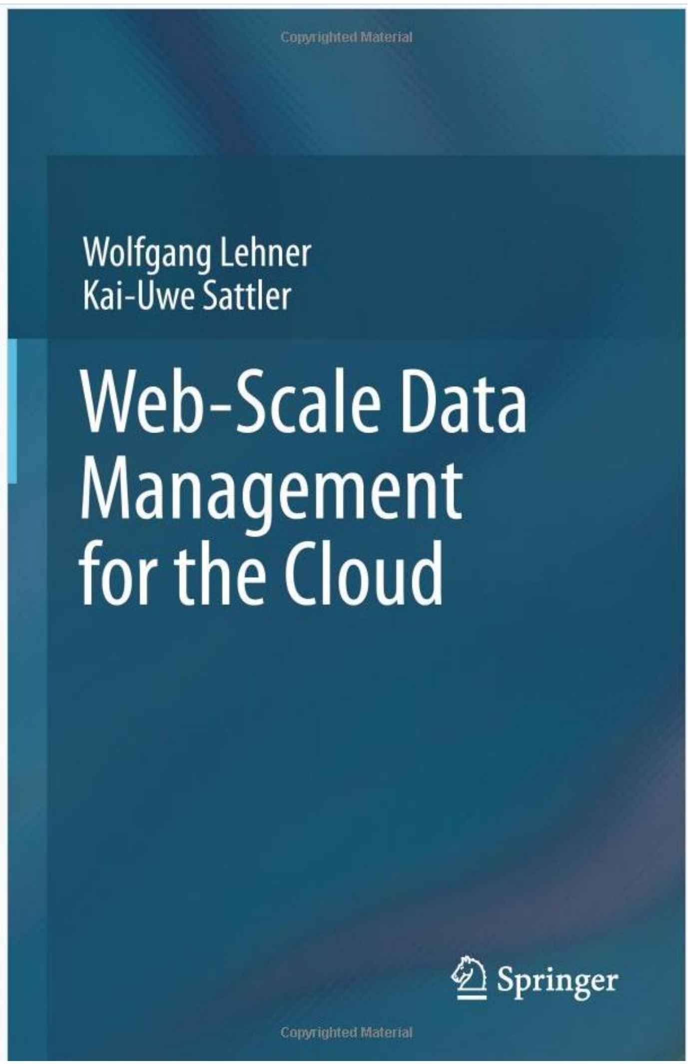
2.

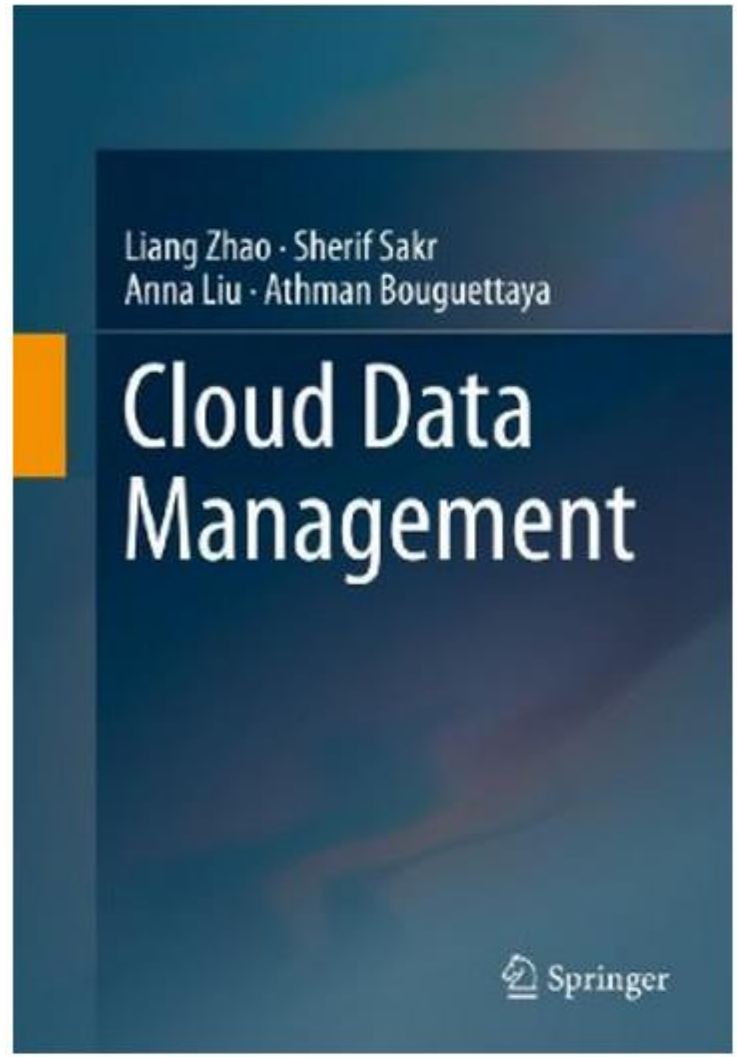
Recommended Literature





Recommended Literature: Cloud-DBs





Recommended Literature: Blogs

Martin Kleppmann

https://martin.kleppmann.com/

NoSQL Weekly

http://www.nosqlweekly.com/

DZone

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

BaQend

http://blog.baqend.com/

InfoQ

http://www.infoq.com/nosql/

High Scalability

http://highscalability.com/

Metadata

http://muratbuffalo.blogspot.de/

DB-ENGINES

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

Seminal NoSQL Papers

- Lamport, Leslie. **Paxos made simple**., SIGACT News, 2001
- S. Gilbert, et al., **Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services**, SIGACT News, 2002
- F. Chang, et al., **Bigtable: A Distributed Storage System For Structured Data**, OSDI, 2006
- G. DeCandia, et al., **Dynamo: Amazon's Highly Available Key-Value Store**, SOSP, 2007
- M. Stonebraker, el al., **The end of an architectural era: (it's time for a complete rewrite)**, VLDB, 2007
- B. Cooper, et al., **PNUTS: Yahoo!'s Hosted Data Serving Platform**, VLDB, 2008
- Werner Vogels, **Eventually Consistent**, ACM Queue, 2009
- B. Cooper, et al., **Benchmarking cloud serving systems with YCSB**., SOCC, 2010
- A. Lakshman**, Cassandra - A Decentralized Structured Storage System**, SIGOPS, 2010
- J. Baker, et al., **MegaStore: Providing Scalable, Highly Available Storage For Interactive Services**, CIDR, 2011
- M. Shapiro, et al.: **Conflict-free replicated data types**, Springer, 2011
- J.C. Corbett, et al., **Spanner: Google's Globally-Distributed Database**, OSDI, 2012
- Eric Brewer, **CAP Twelve Years Later: How the "Rules" Have Changed**, IEEE Computer, 2012
- J. Shute, et al., **F1: A Distributed SQL Database That Scales**, VLDB, 2013
- L. Qiao, et al., **On Brewing Fresh Espresso: Linkedin's Distributed Data Serving Platform**, SIGMOD, 2013
- N. Bronson, et al., **Tao: Facebook's Distributed Data Store For The Social Graph**, USENIX ATC, 2013
- P. Bailis, et al., **Scalable Atomic Visibility with RAMP Transactions**, SIGMOD 2014