NoSQL and Big Data Lecture

Felix Gessert, Norbert Ritter
fg@baqend.com
May 5, 2019

Universität Hamburg

BaQend

Outline

NoSQL Foundations and Motivation

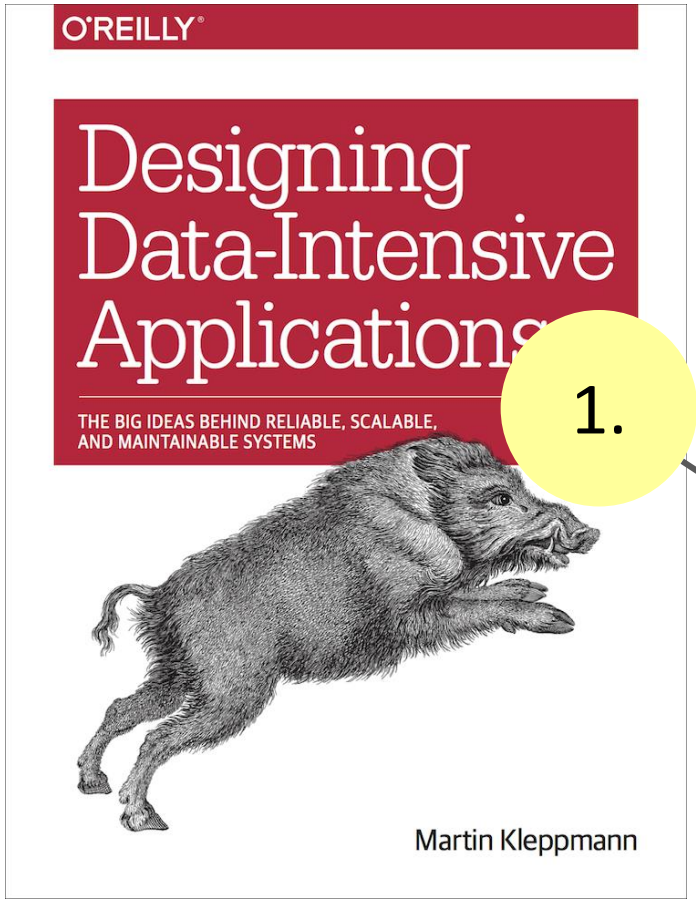
The NoSQL Toolbox: Common Techniques

NoSQL Systems & Decision Guidance

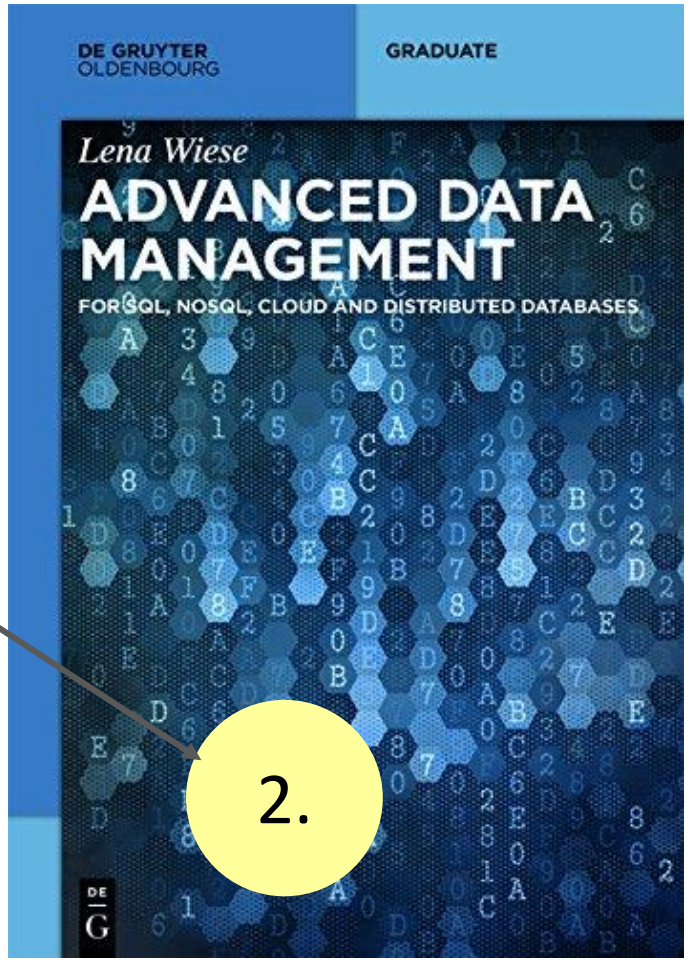
Cloud Databases

- Literature
- 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

Recommended Literature: NoSQL Books



1.



2.

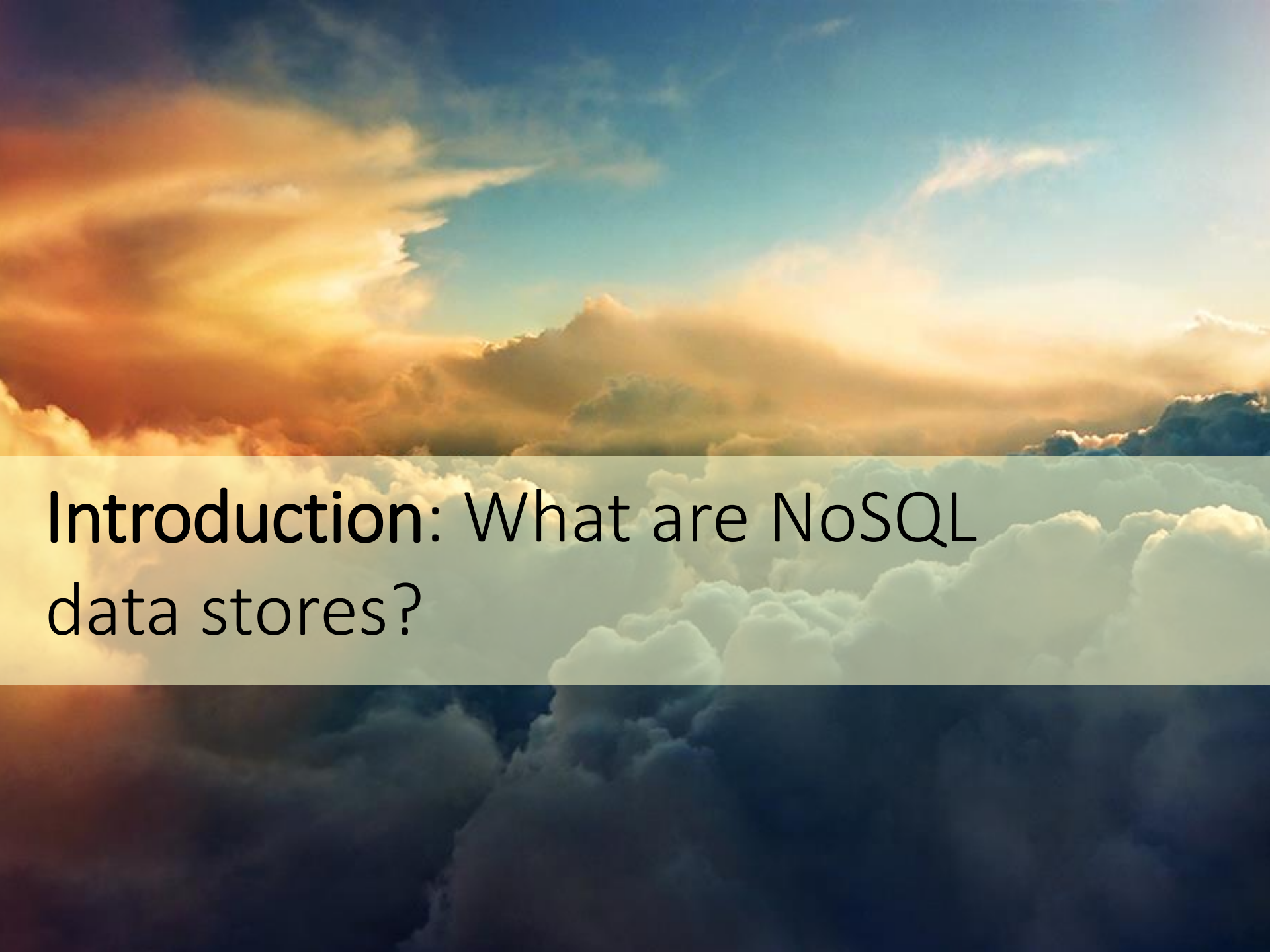
Recommended Literature: Blogs

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

http://blog.baqend.com/

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

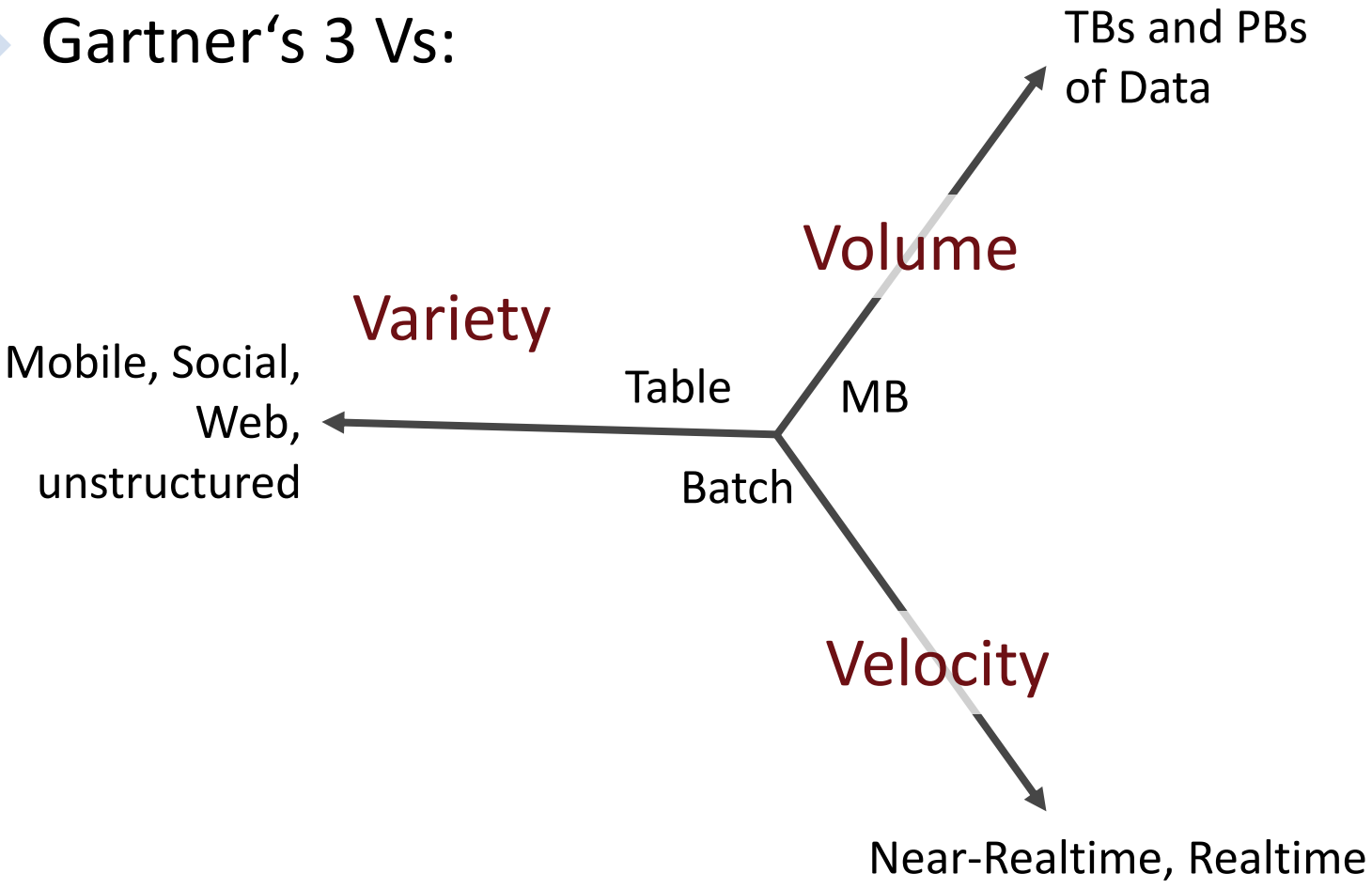
http://highscalability.com/



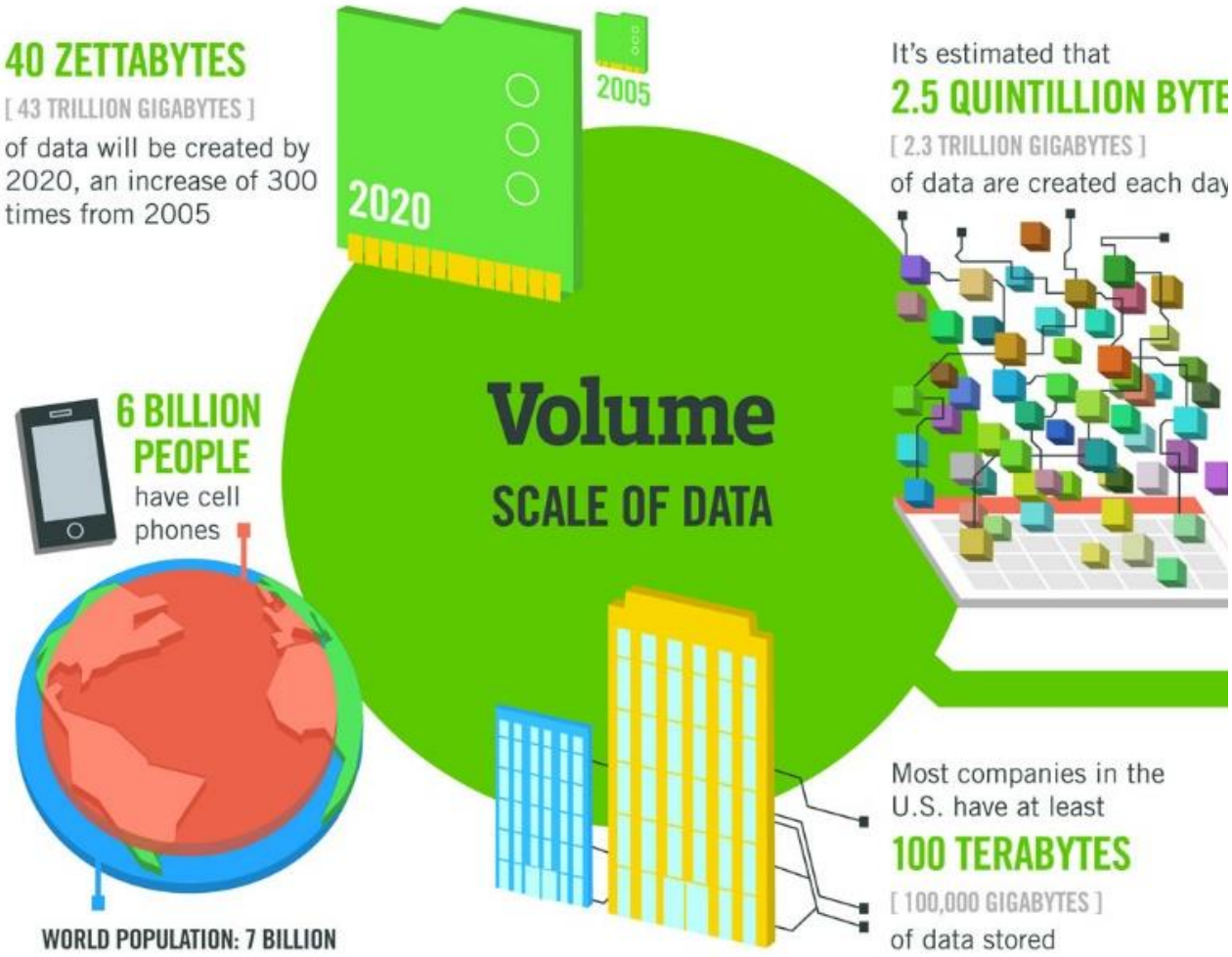
Introduction: What are NoSQL data stores?

Motivation: Big Data

- Gartner‘s 3 Vs:

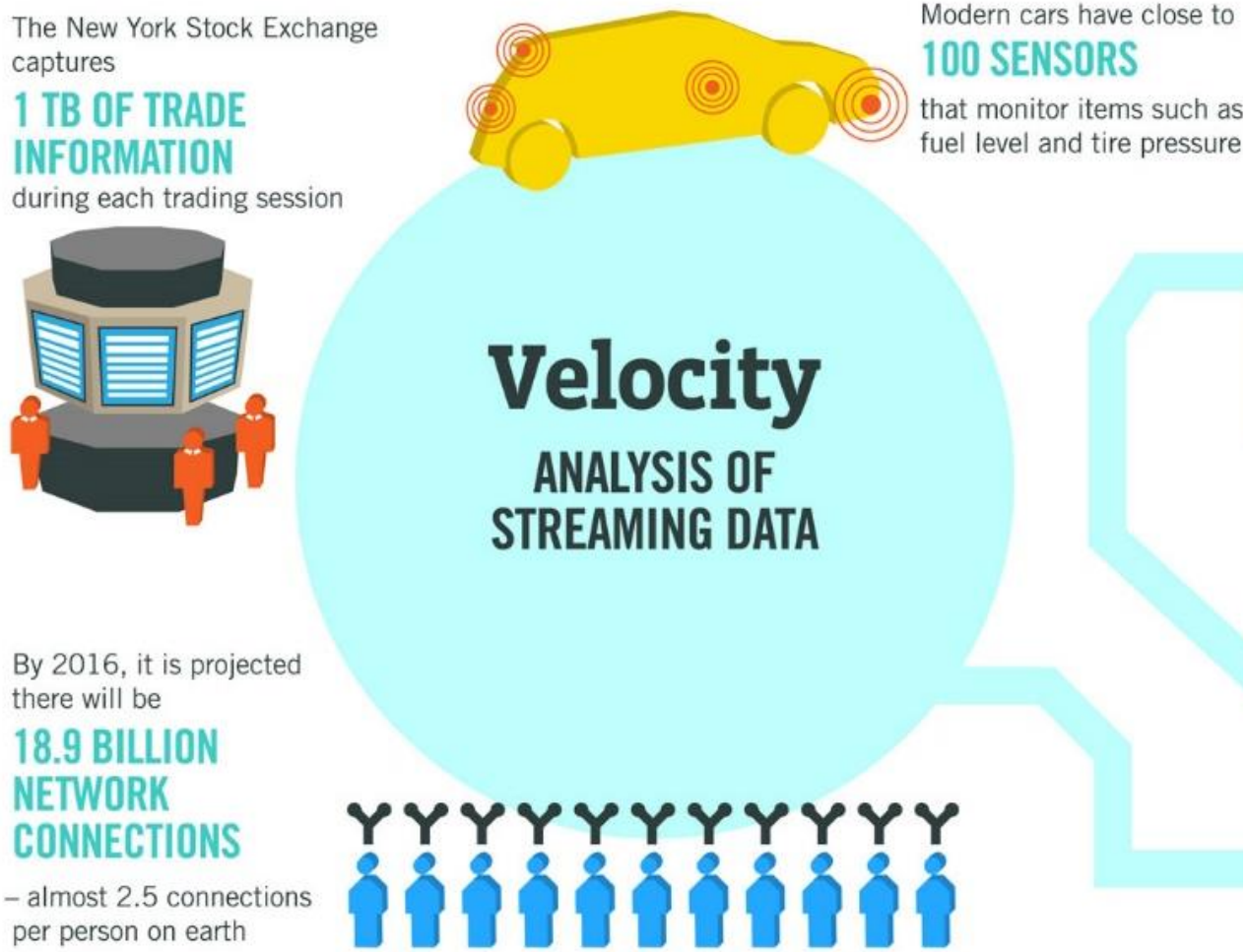


4Vs



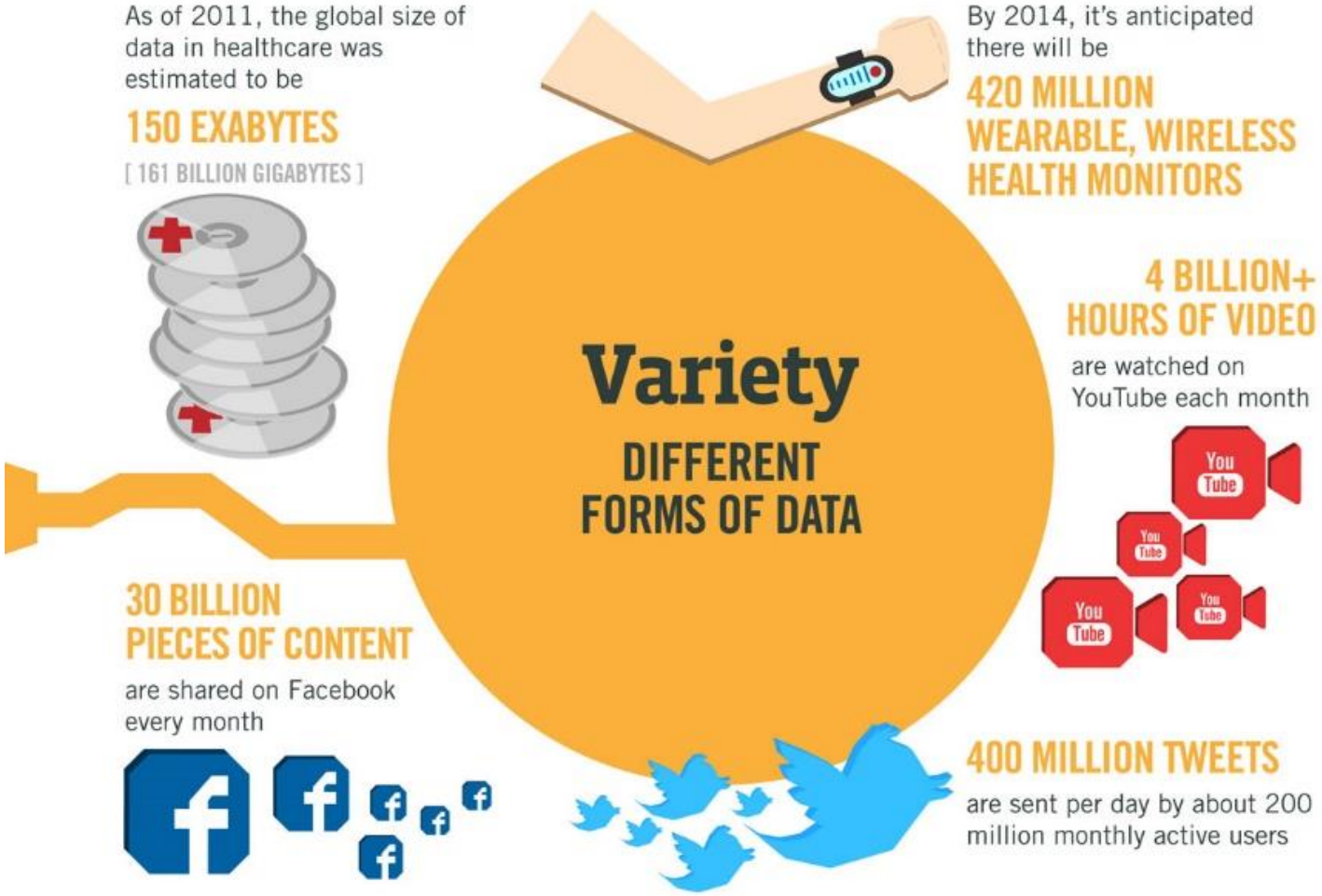
IBM Infographic (McKinsey, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS)

4Vs



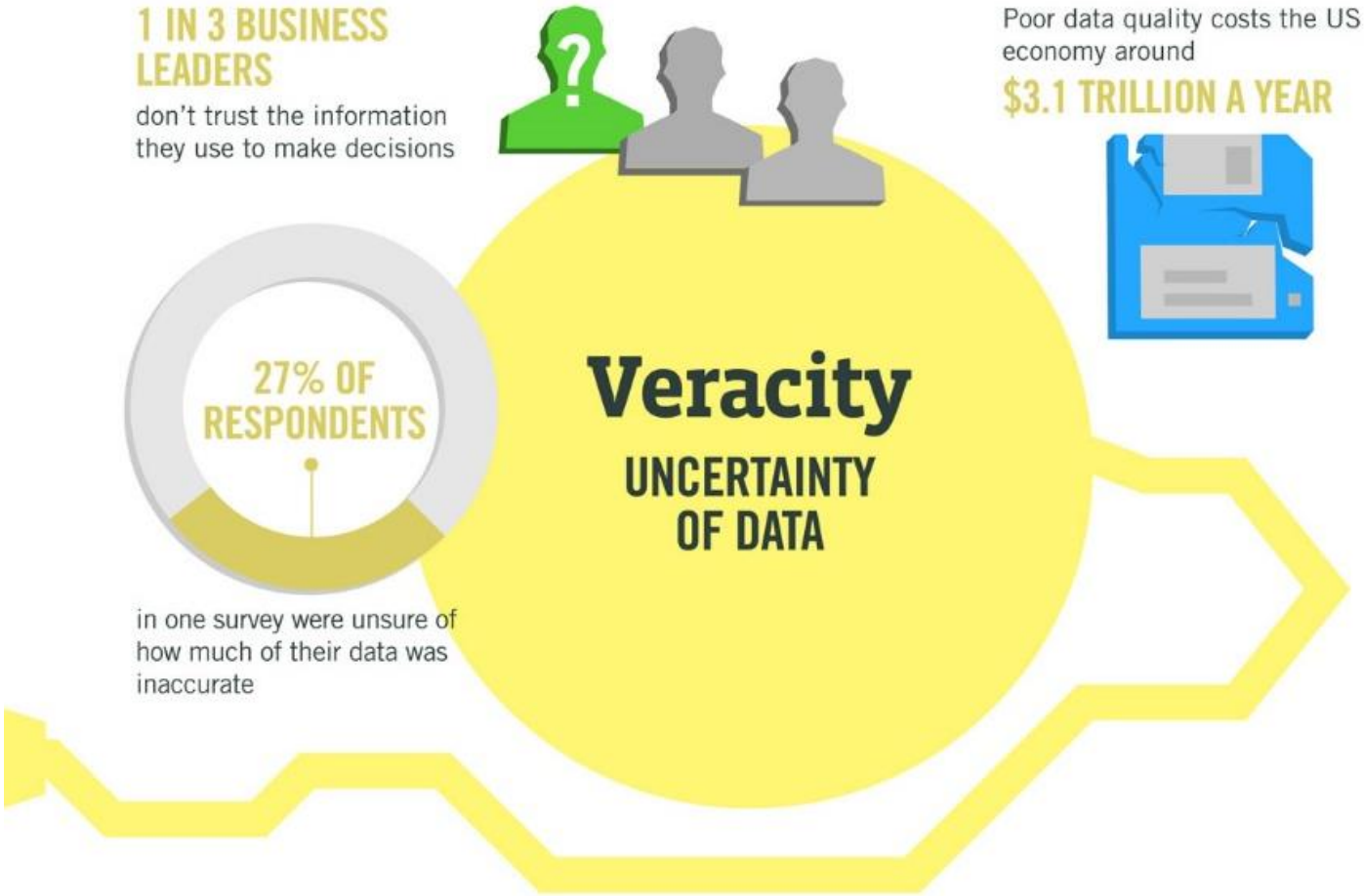
IBM Infographic (McKinsey, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS)

4Vs



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

4Vs



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

The Database Explosion

A database system for every use case?

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

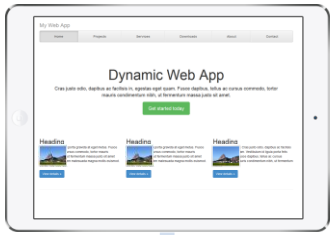
Counting & statistics

Wide-Column Store

Massive user-
generated content

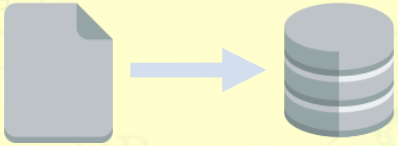
How to choose a database system?

Many Potential Candidates



Question in this lecture:

How to approach the



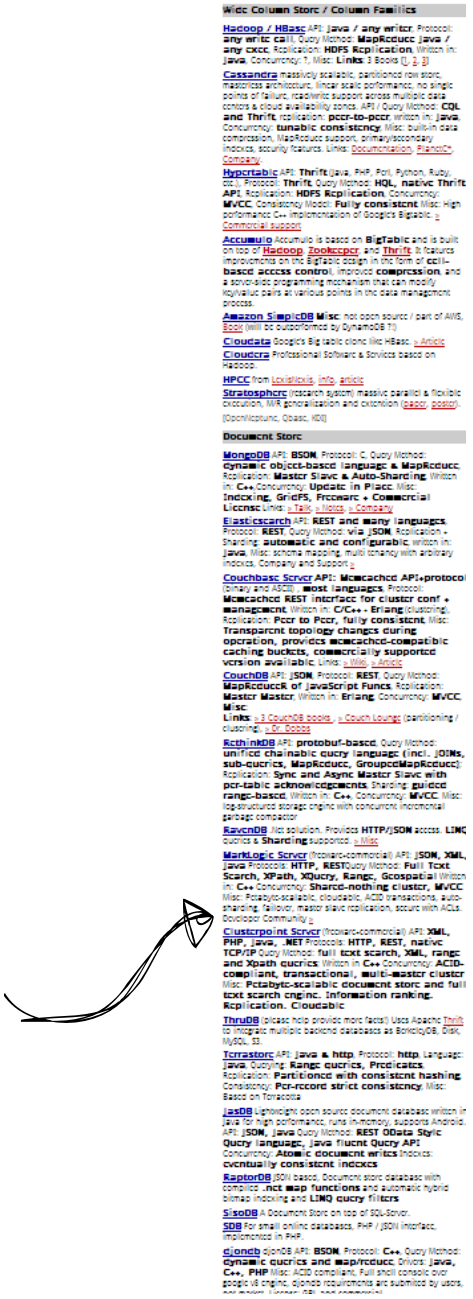
requirements database

decision problem?

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, readwrite 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: Presentation, Planet Cassandra

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 feaures 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, MR 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: MapReduce of JavaScript Funcs. Replication: Master Master. Written in: Erlang. Concurrency: MVCC. Misc: Links: 3 CouchDB books, Couch Lounge (partitioning / clustering), 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 Not solution. Provides HTTP/JSON access. LINQ queries & Sharding supported. Misc

MarkLogic Server (free/zero-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

Clustrpoint Server (free/zero-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 (private beta 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.

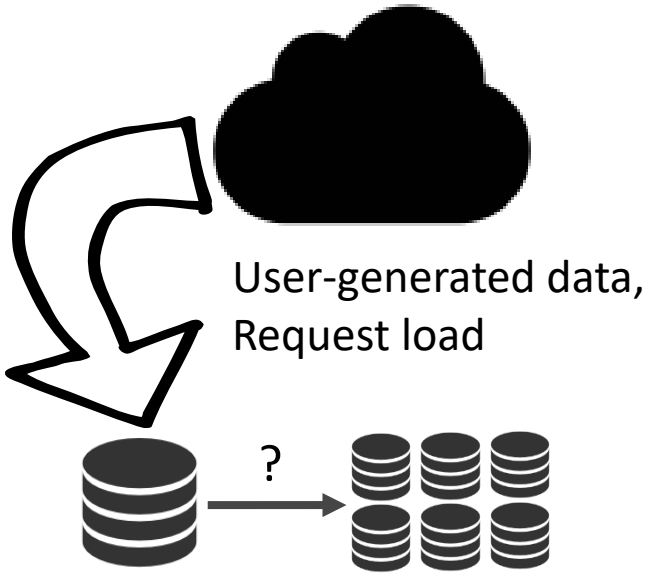
SDB For small online databases. PHP / JSON interface. Implemented in PHP.

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 equivalences are submited by users,

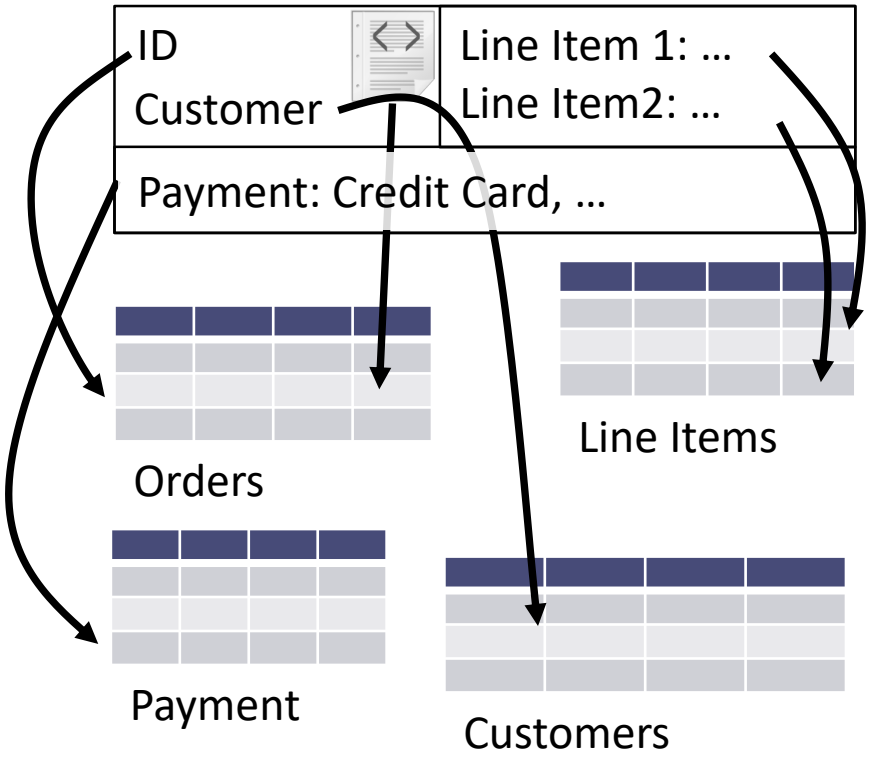
NoSQL Databases

- Two main motivations:

Scalability

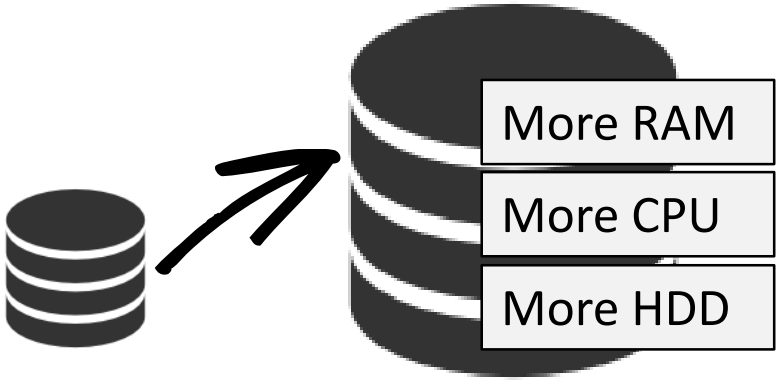


Impedance Mismatch

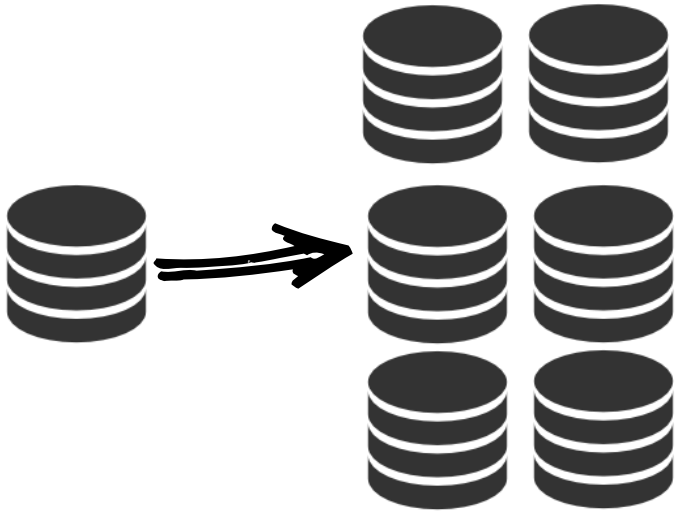


Scale-up vs Scale-out

Scale-Up (*vertical* scaling):



Scale-Out (*horizontal* scaling):

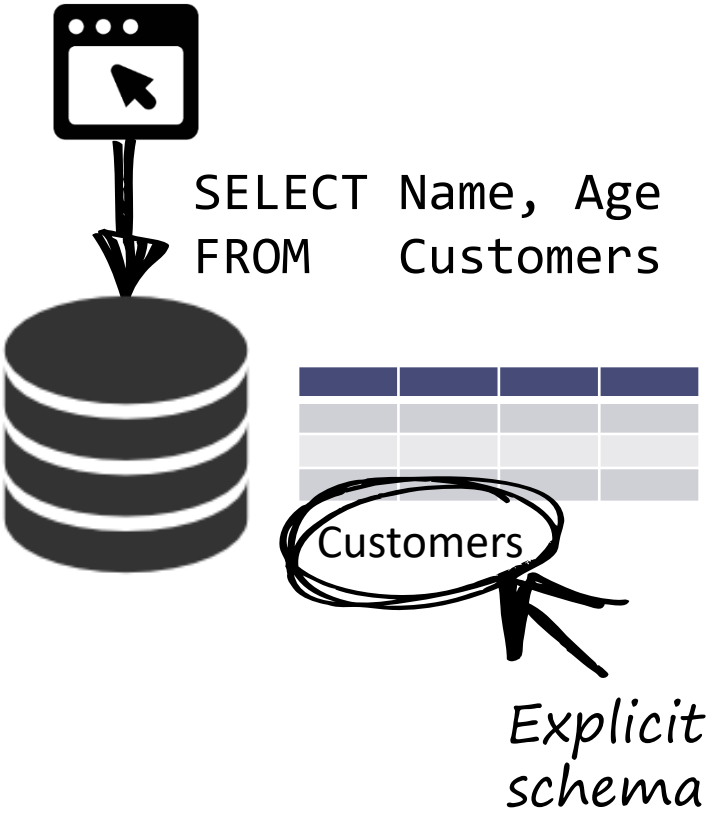


Same Hardware

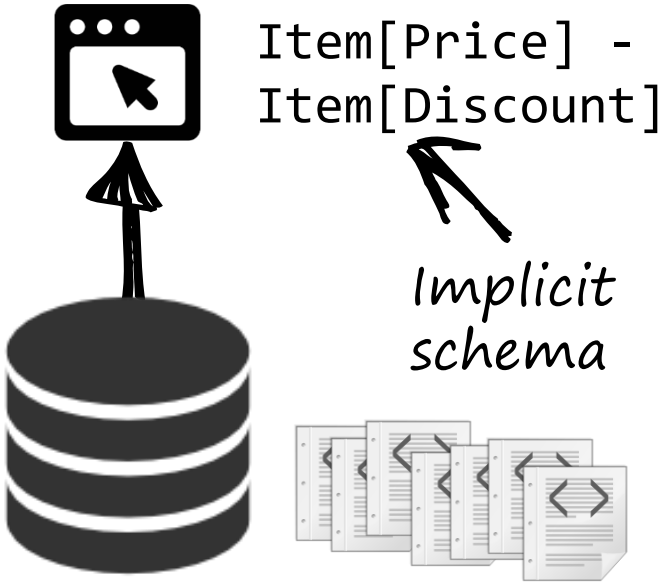
Connected by network

Schema-free Data Modeling

RDBMS:



NoSQL DB:



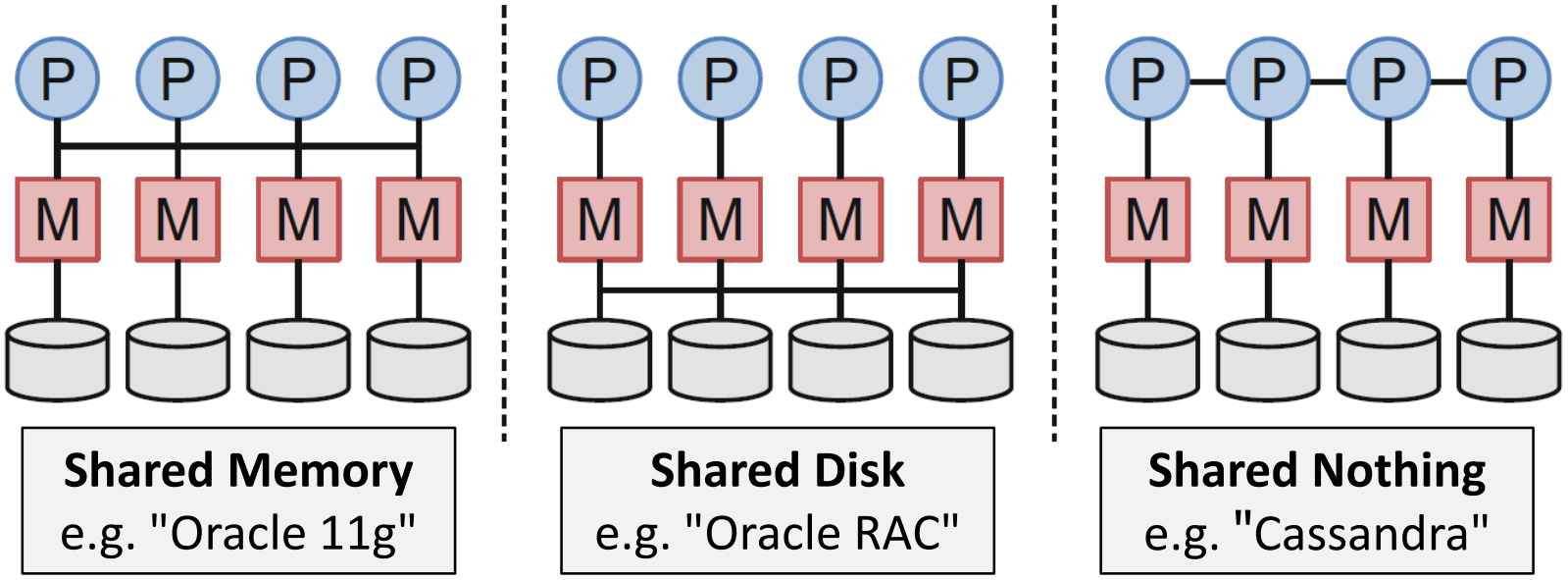
NoSQL Paradigm Shift

Open Source & Commodity Hardware

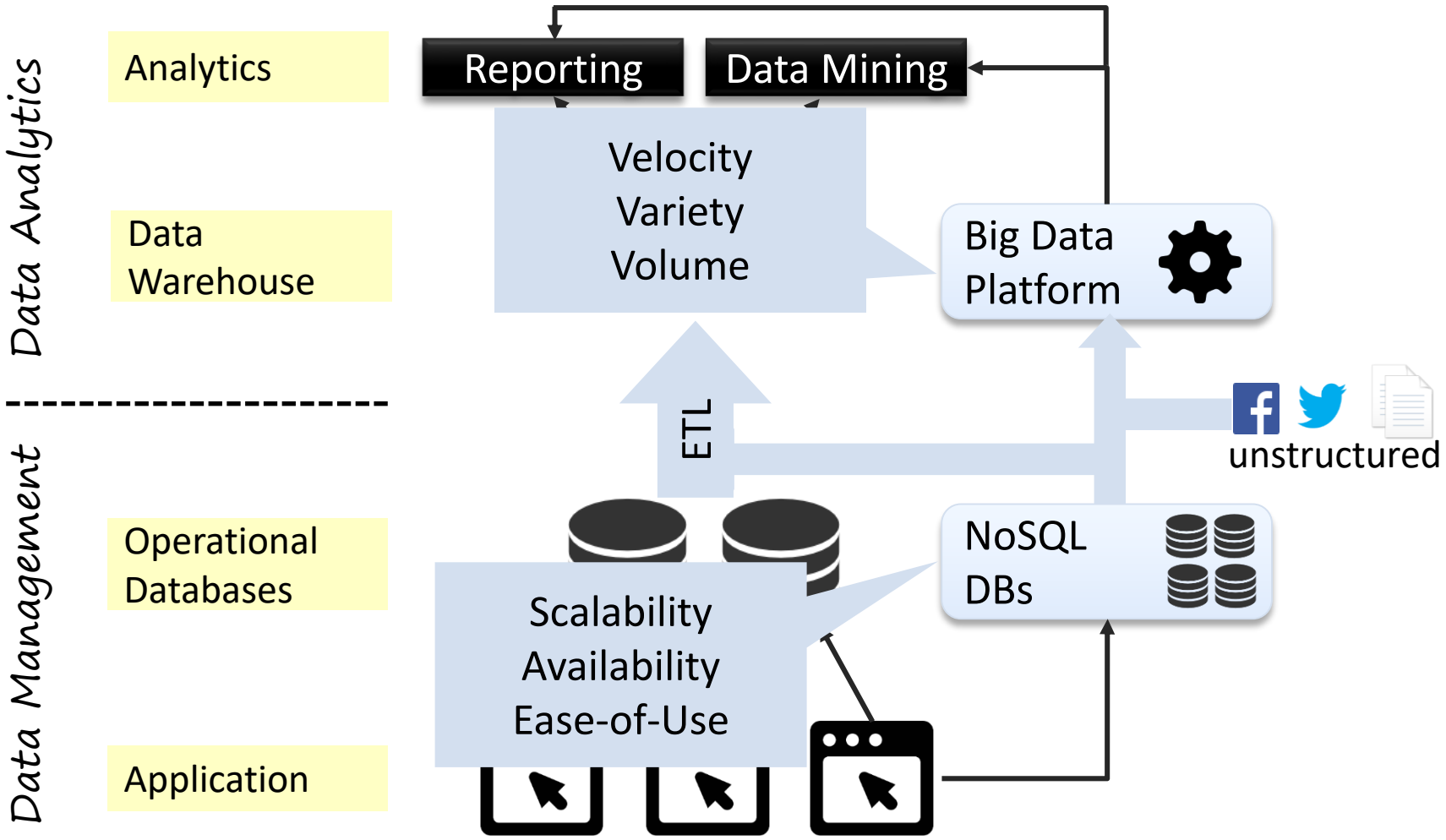
Commercial DBMS	Open-Source DBMS
Specialized DB hardware (Oracle Exadata, etc.)	Commodity hardware
Highly available network (Infiniband, Fabric Path, etc.)	Commodity network (Ethernet, etc.)
Highly Available Storage (SAN, RAID, etc.)	Commodity drives (standard HDDs, JBOD)

Paradigm Shift

- Shift towards **distributed computing architectures**

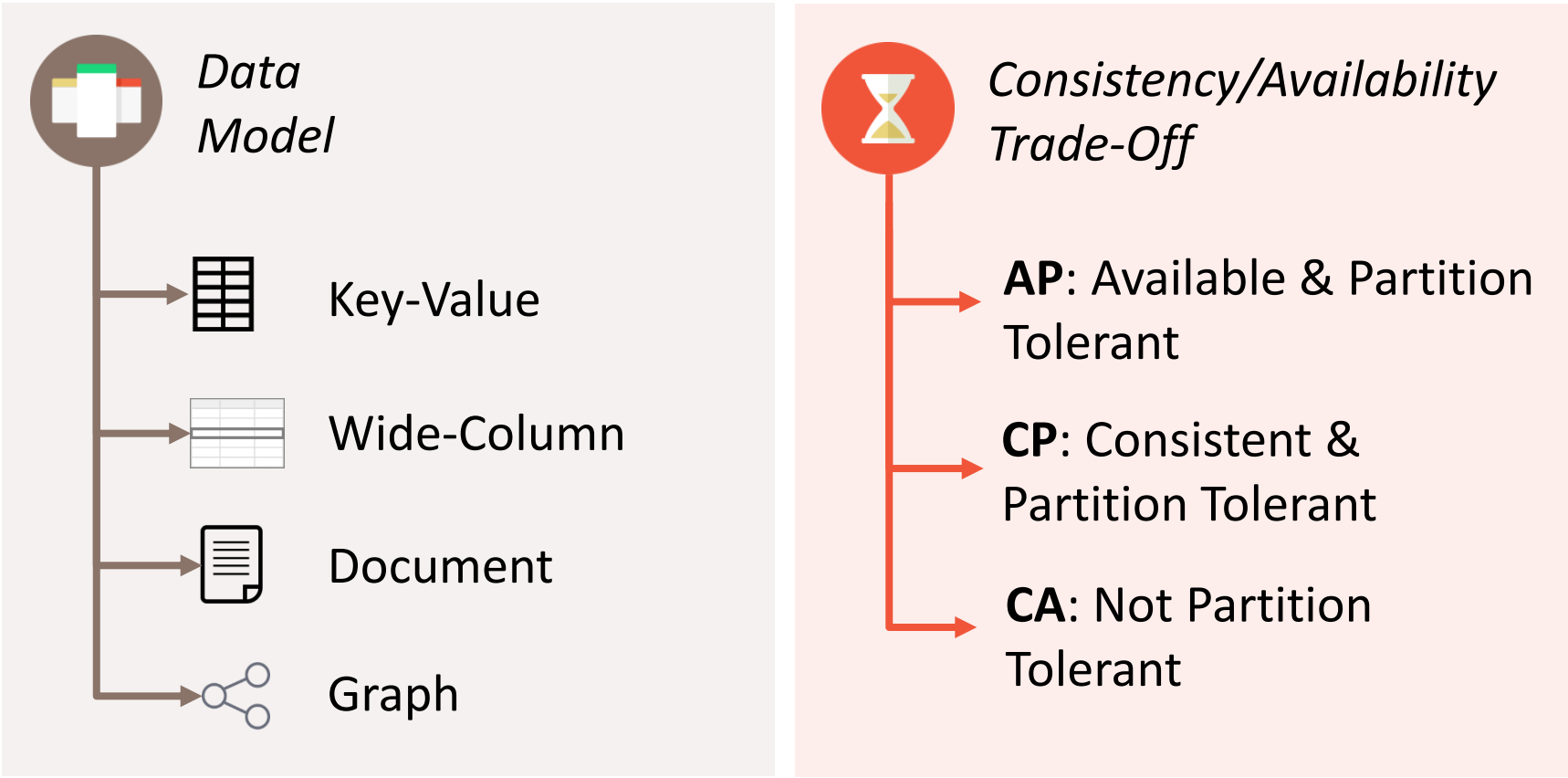


NoSQL and Big Data Architecture



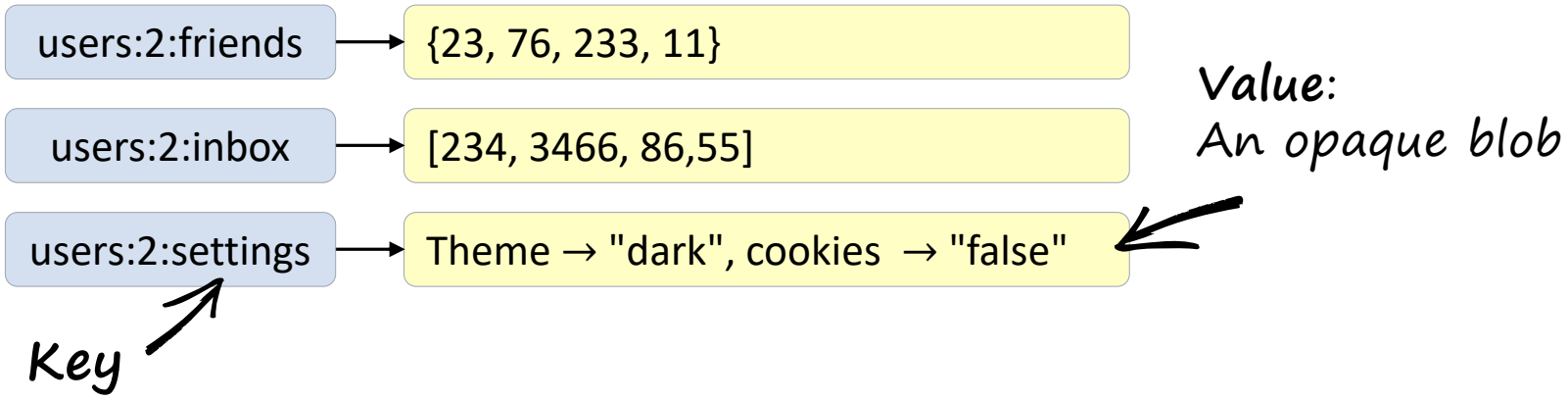
NoSQL System Classification

- Two common criteria:



Key-Value Stores

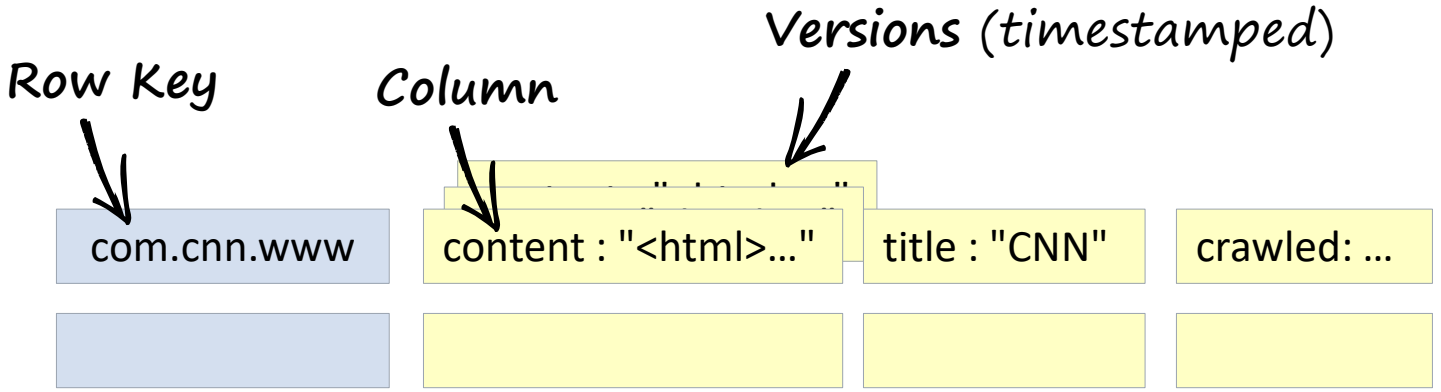
- **Data model:** (key) -> value
- **Interface**: CRUD (Create, Read, Update, Delete)



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

Wide-Column Stores

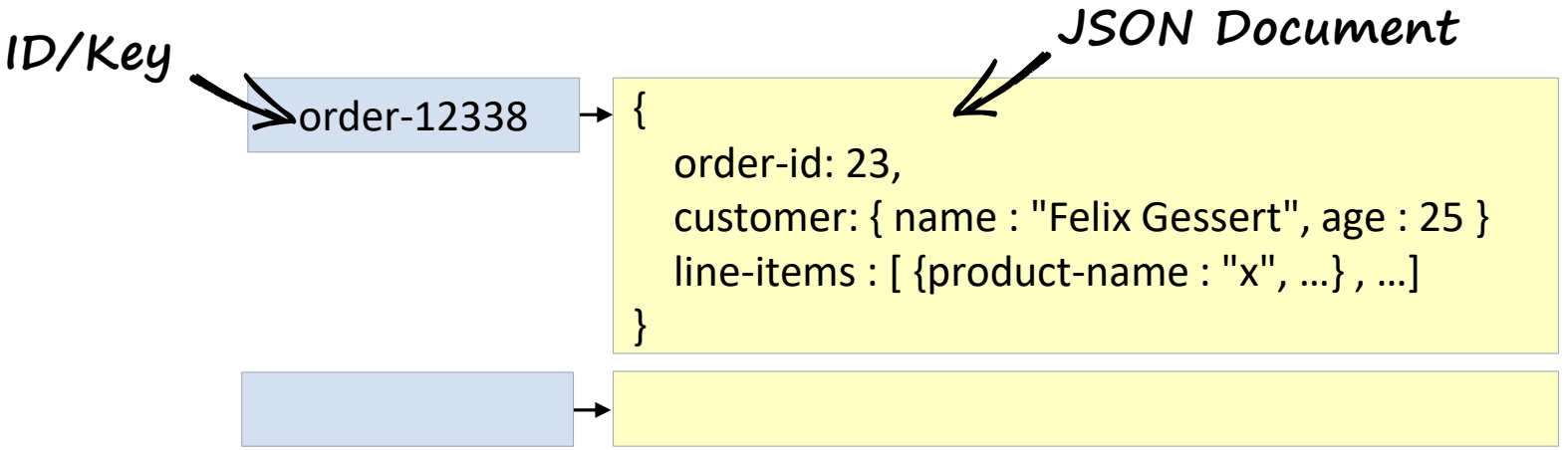
- **Data model:** (rowkey, column, timestamp) -> value
- **Interface**: CRUD, Scan



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

Document Stores

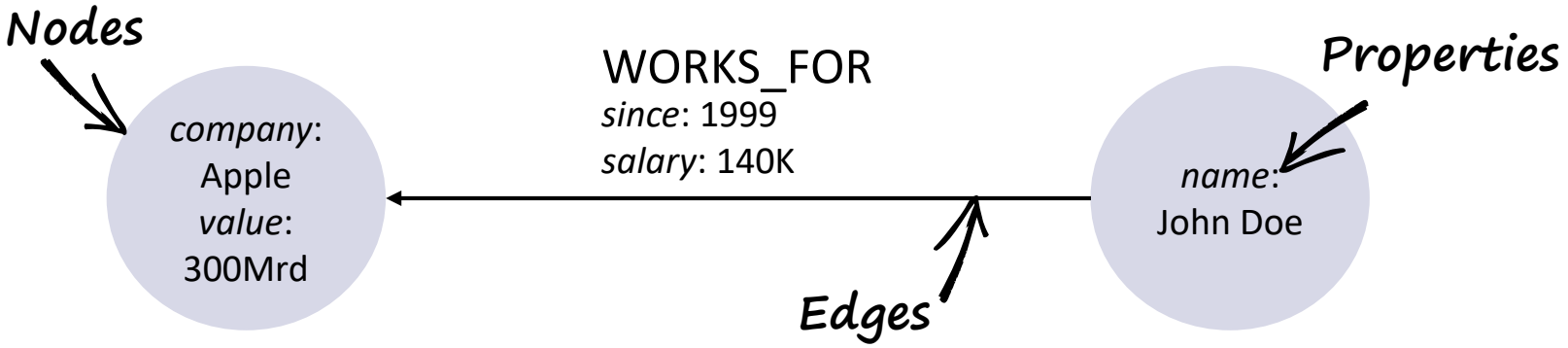
- **Data model:** (collection, key) -> document
- **Interface**: CRUD, Querys, Map-Reduce



- Examples: CouchDB (AP), RethinkDB (CP), MongoDB (CP)

Graph Databases

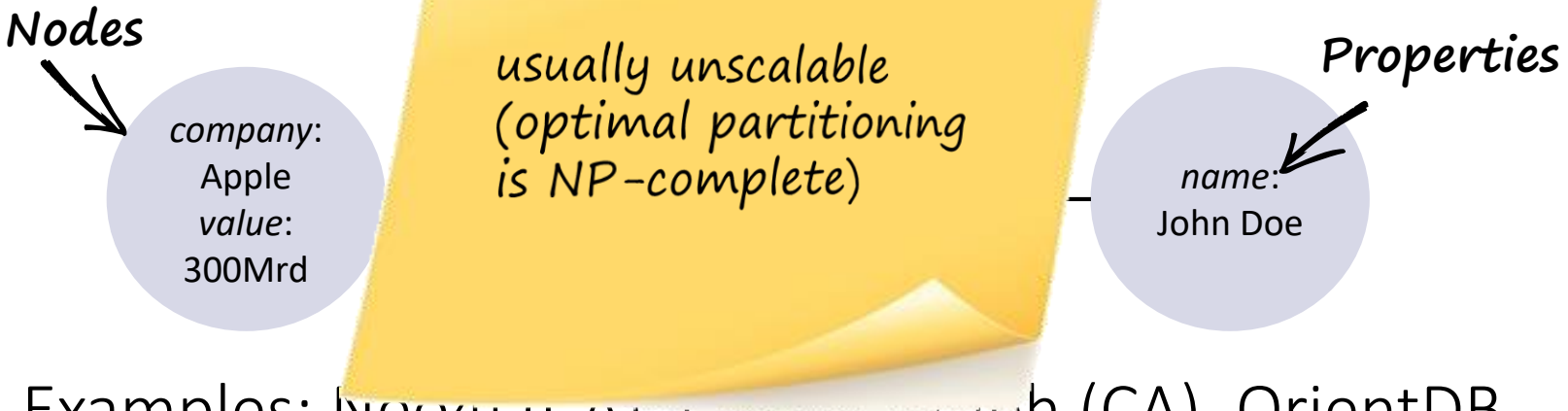
- **Data model:** G = (V, E): Graph-Property Modell
- **Interface**: Traversal algorithms, querys, transactions



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

Graph Databases

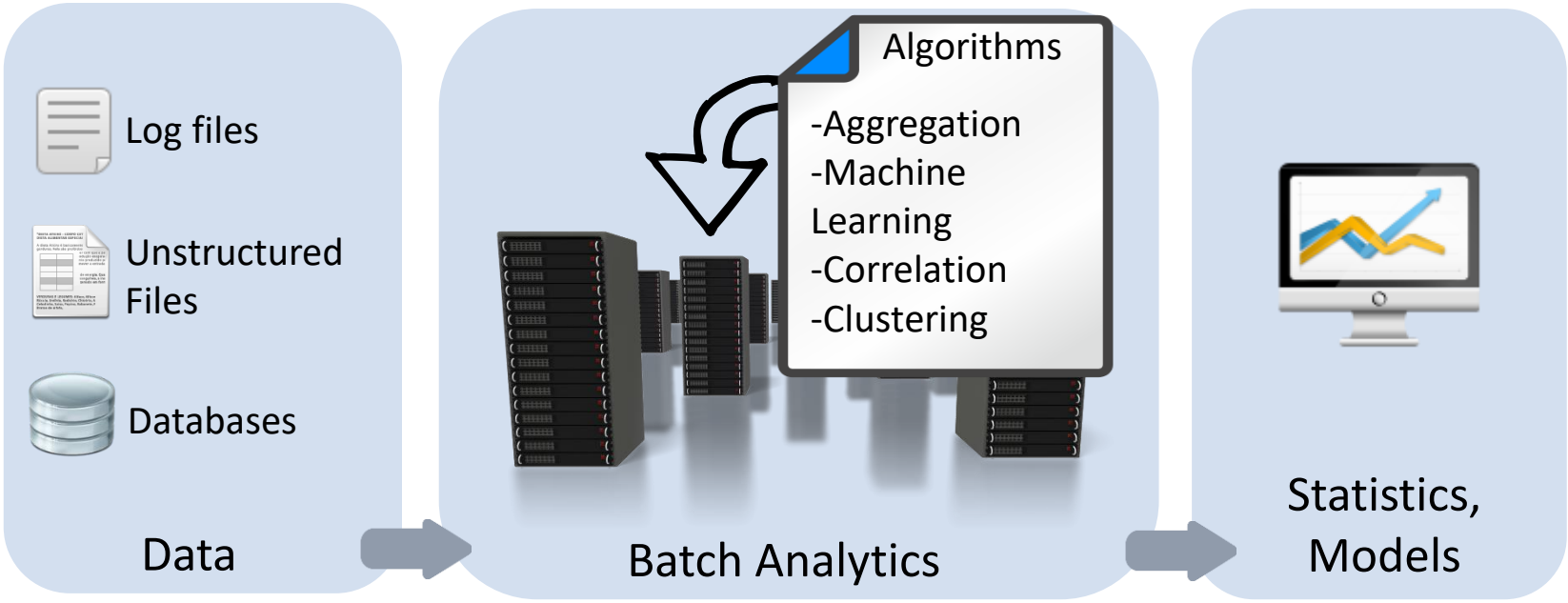
- **Data model:** G = (V, E): Graph-Property Modell
- **Interface**: Trave[illegible], transactions



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

Big Data Batch Processing

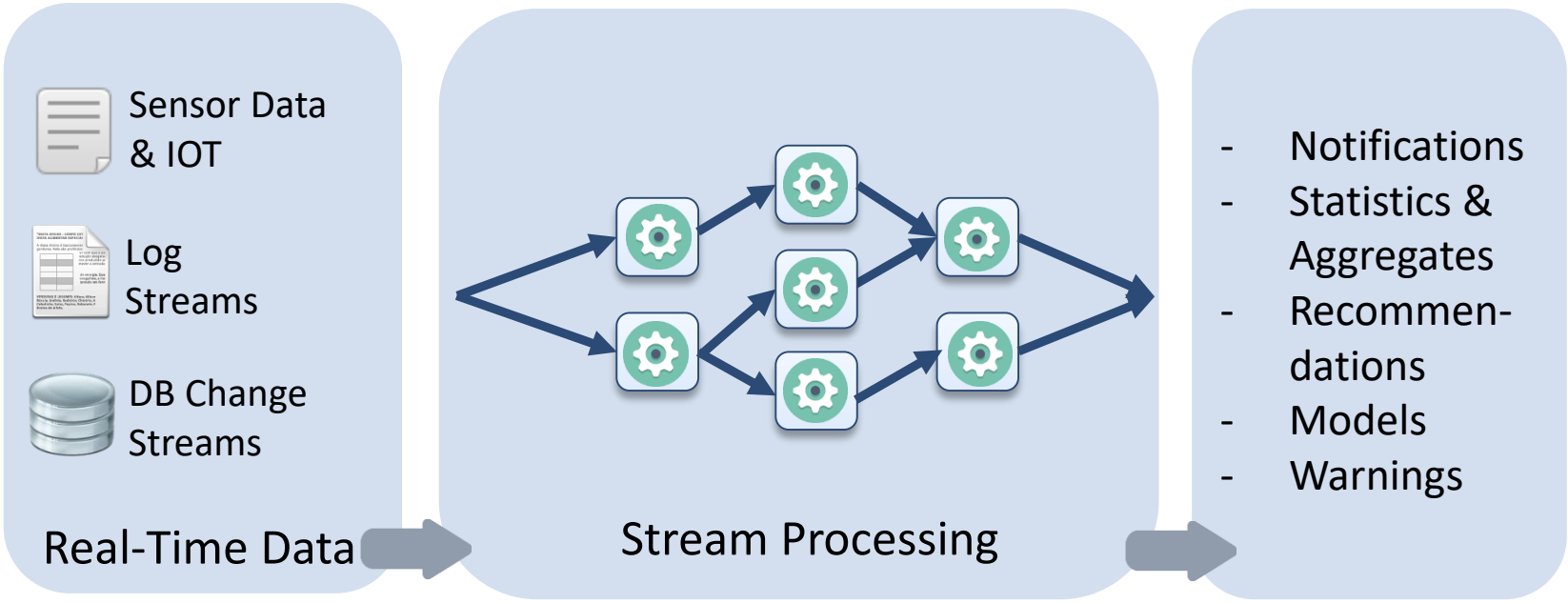
- **Data model:** arbitrary (frequently unstructured)
- Examples: Hadoop, Spark, Flink, DryadLink, Pregel



Big Data Stream Processing

Covered in Depth in the Last Part

- **Data model:** arbitrary
- Examples: Storm, Samza, Flink, Spark Streaming



Soft NoSQL Systems

Not Covered Here

Search Platforms (Full Text Search):

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

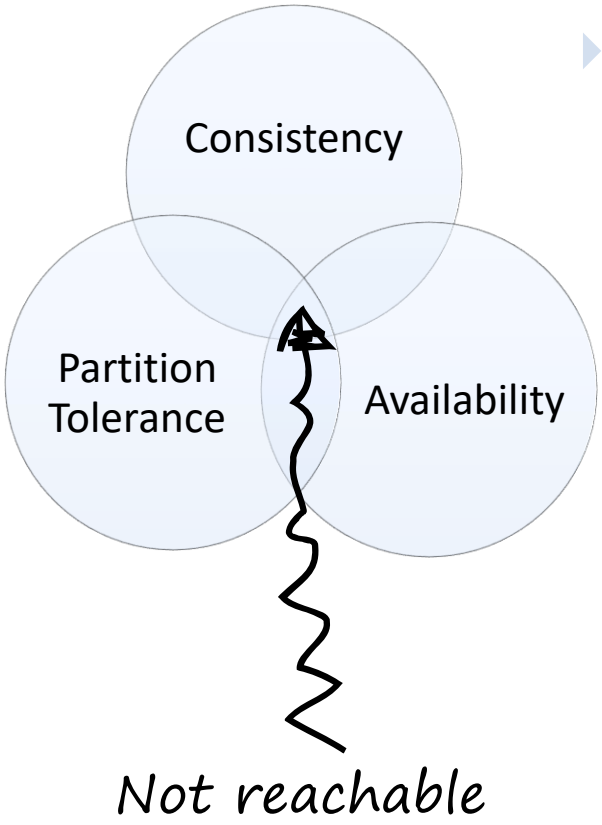
Object-Oriented Databases:

- 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



- 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

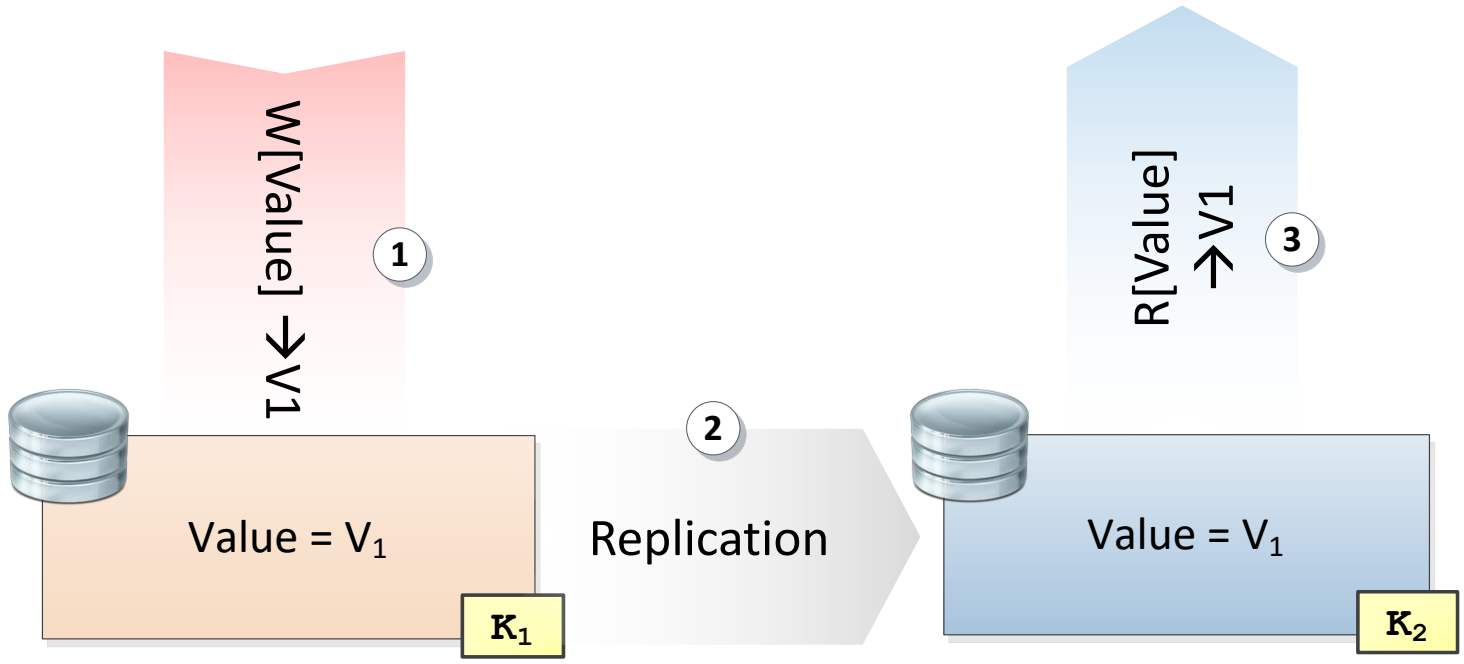
Gilbert, Lynch: Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services, SigAct News 2002

CAP-Theorem: simplified proof

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

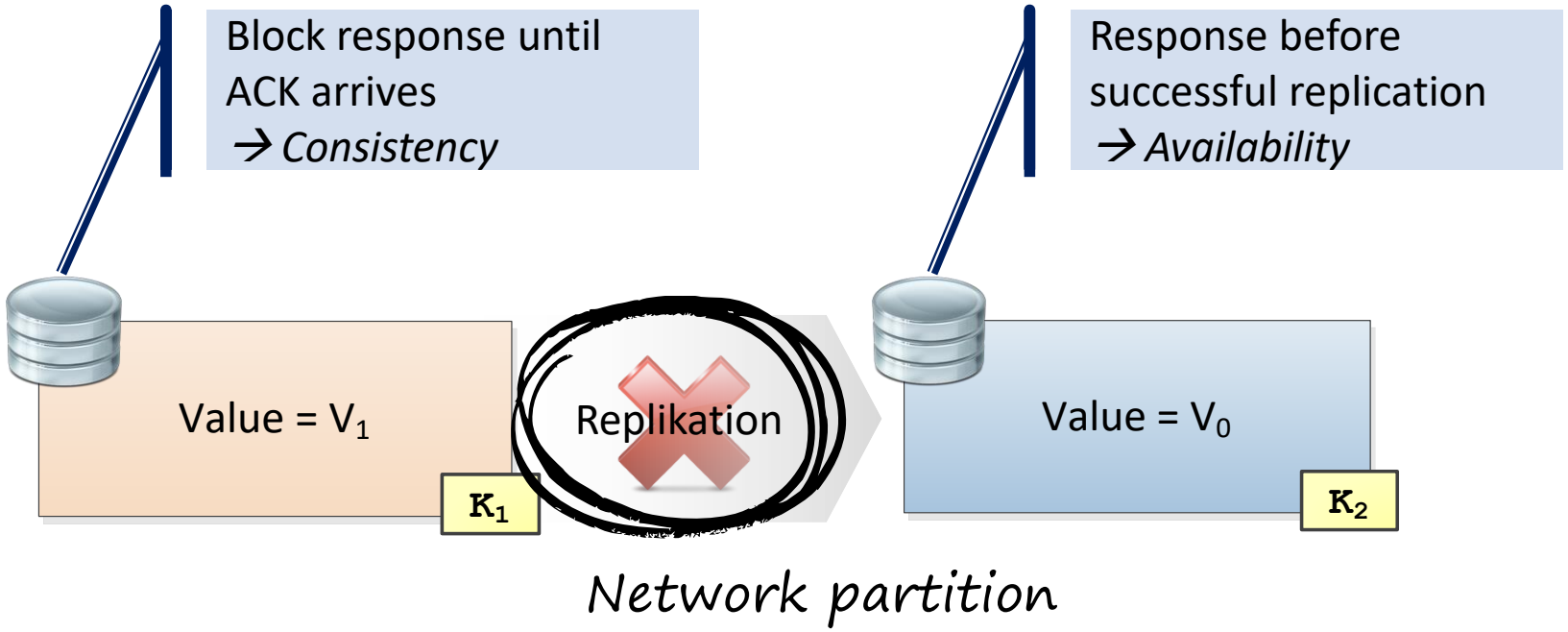
CAP-Theorem: simplified proof

- Failure-free reading and writing:

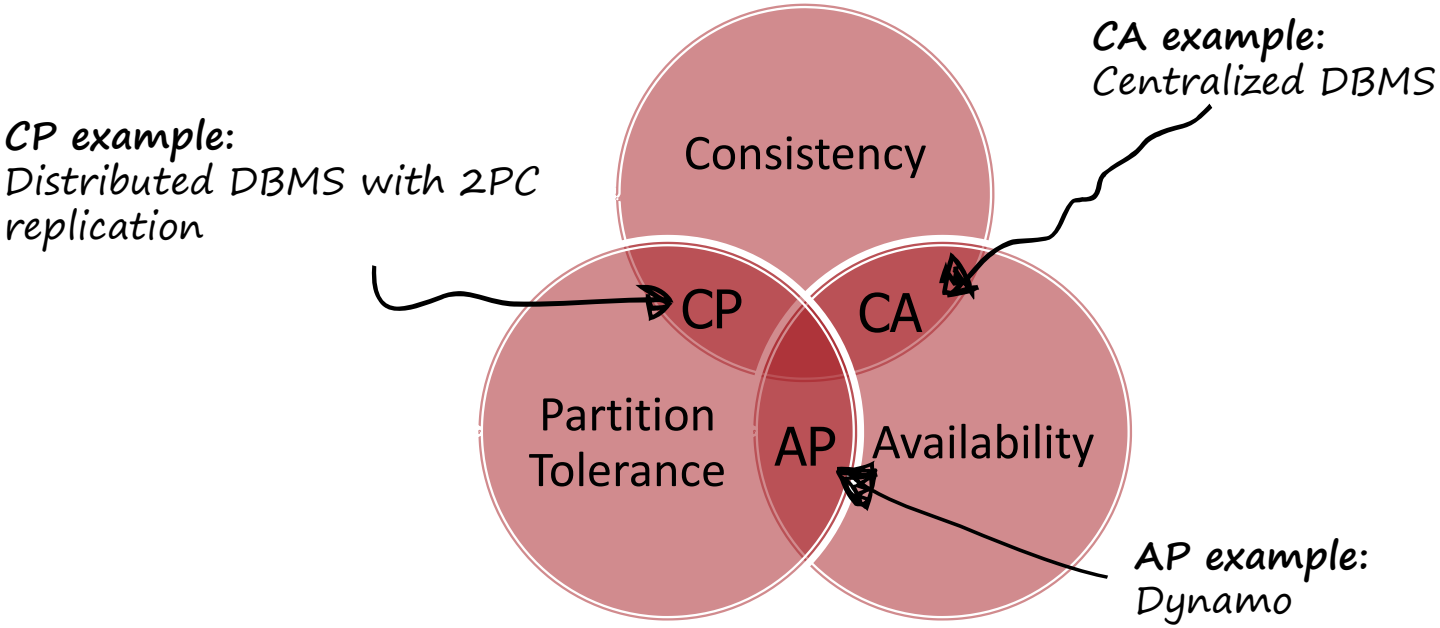


CAP-Theorem: simplified proof

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

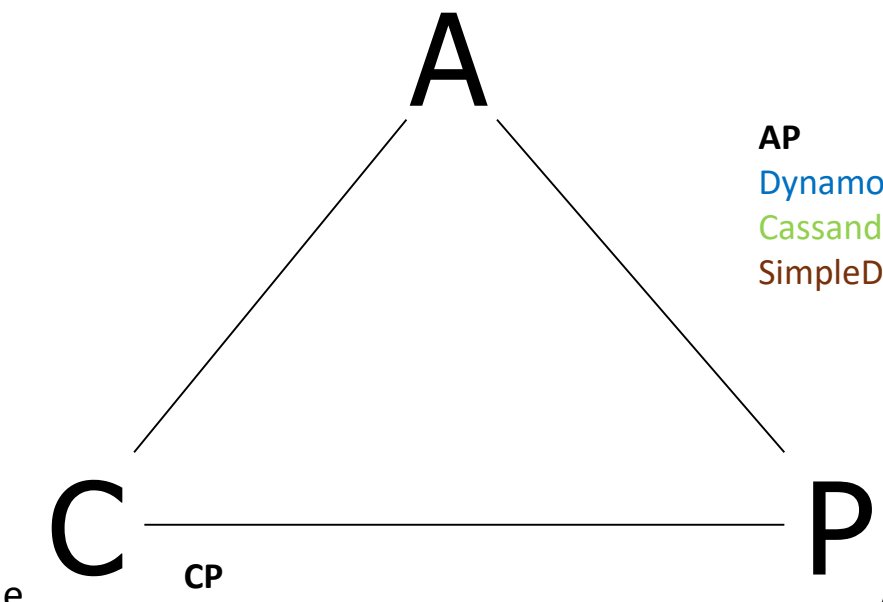


CAP-Theorem: Wrap-up



NoSQL Triangle

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



Every client can always read and write

CA
Oracle, MySQL, …

AP
Dynamo, Redis, 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, DocumentsDB

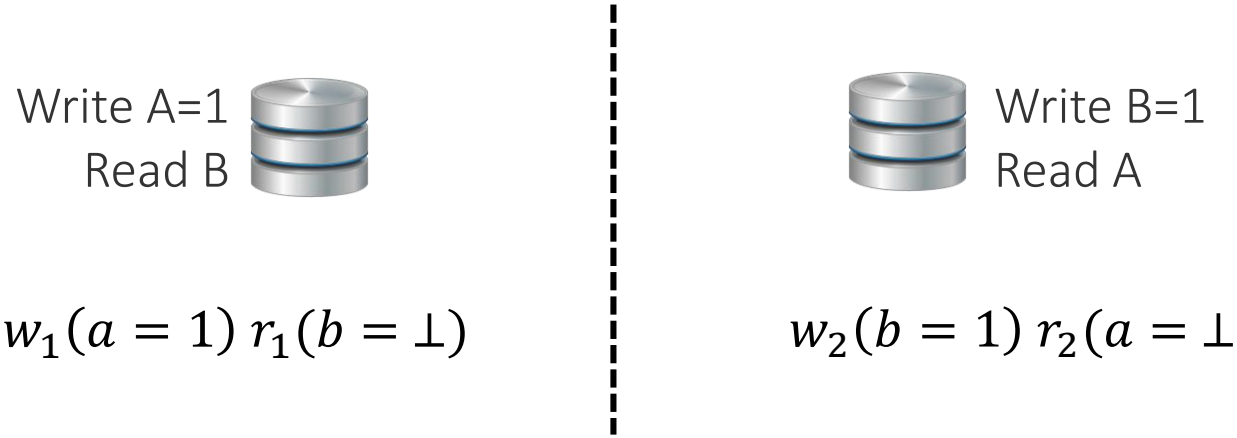
All nodes continue working under network partitions

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

Serializability

Not Highly Available Either

Global serializability and availability are incompatible:



- 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

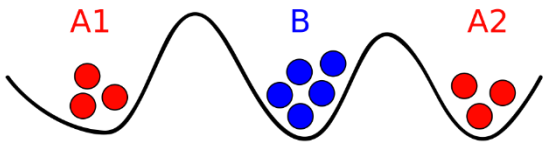
***Asynchronous* Network, *Unreliable* Channel**

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.

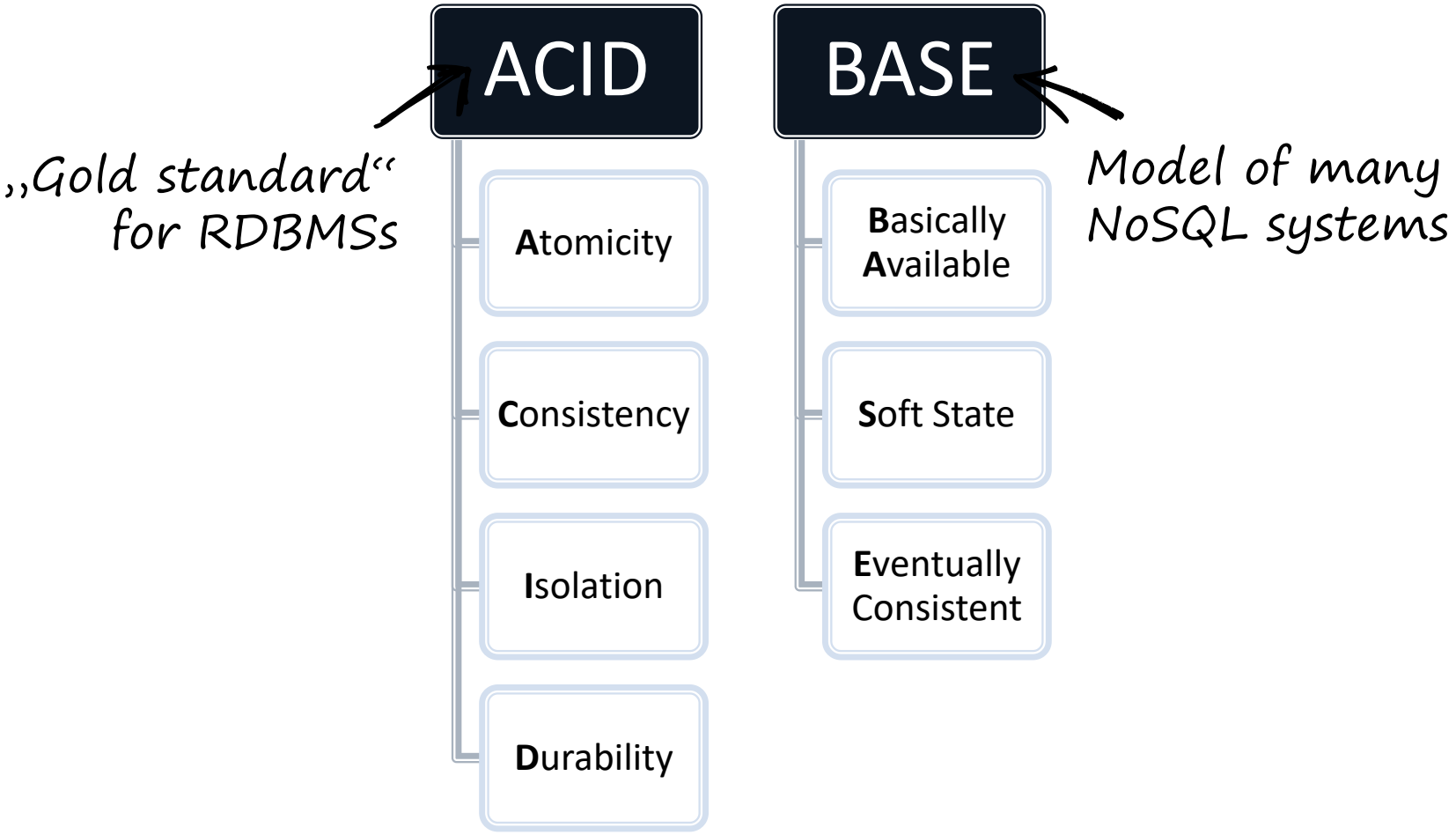
Consensus Algorithms

Even Harder Than CAP

- 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, Aerospike, Cassandra Lightweight Transactions)
  - **Raft** (e.g. RethinkDB, etcd service)
  - Zookeeper Atomic Broadcast (**ZAB**)

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

ACID vs BASE

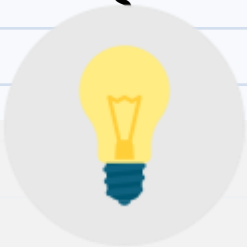


http://queue.acm.org/detail.cfm?id=1394128

Weaker guarantees in a database?!

Default Isolation Levels in RDBMSs

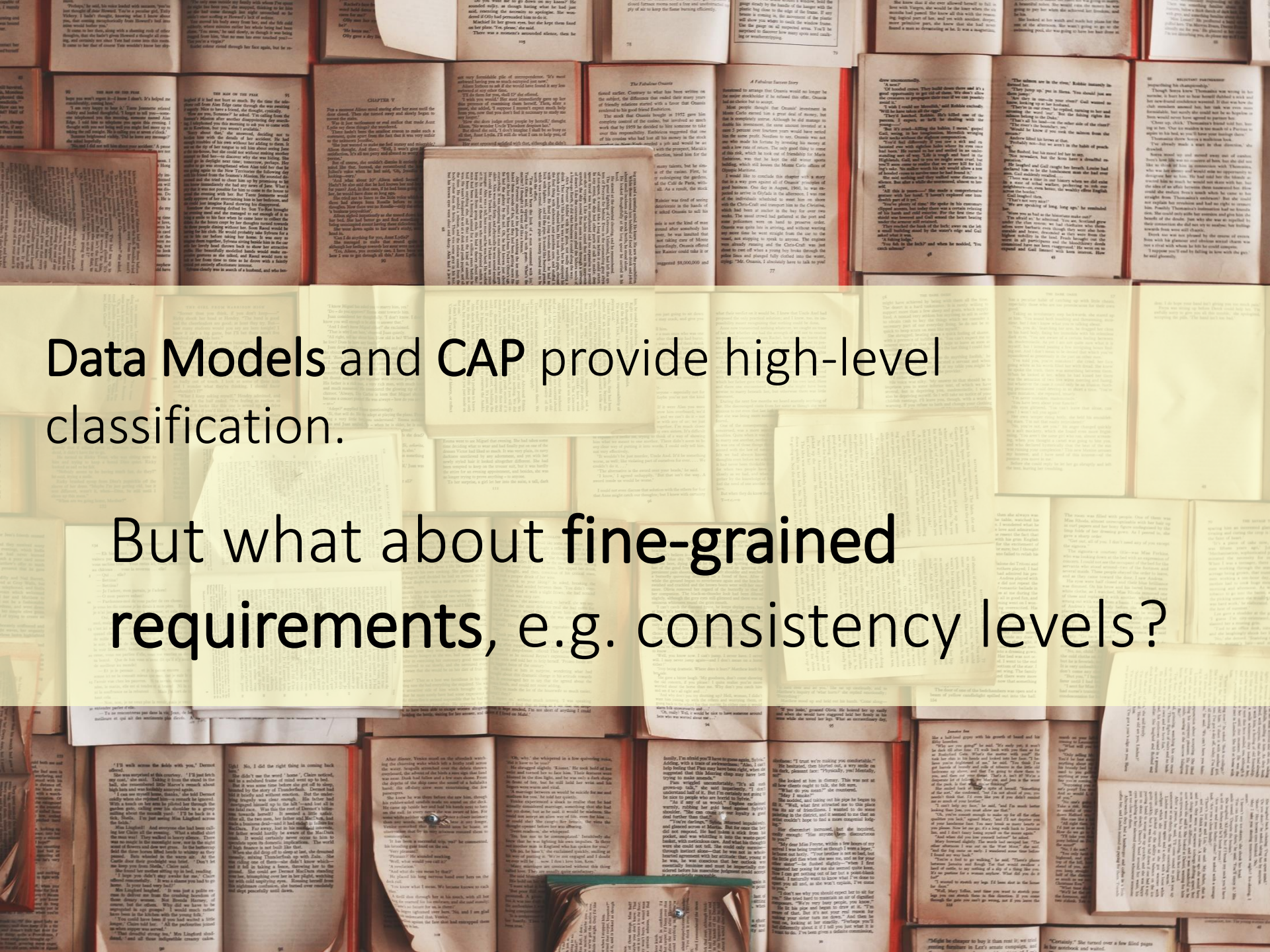
Database	Default Isolation	Maximum Isolation
Actian Ingres 10.0/10S	**S**	**S**
Aerospike		RC
Clustrix CLX 4100		?
Greenplum 4.1		S
IBM DB2 10 for z/OS		S
IBM Informix 11.50	Depends	RR
MySQL 5.6		S
MemSQL 1b	RC	RC
MS SQL Server 2012		S
NuoDB	CR	CR
Oracle 11g	RC	SI
Oracle Berkeley DB	S	S
Postgres 9.2.2	RC	**S**
SAP HANA	RC	SI
ScaleDB 1.02	RC	RC
VoltDB	**S**	**S**



Theorem:
Trade-offs are central to the choice of a database system.

RC: read committed, RR: repeatable read, S: serializability, SI: snapshot isolation, CS: cursor stability, CR: consistent read

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



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

But what about **fine-grained requirements**, e.g. consistency levels?

Outline

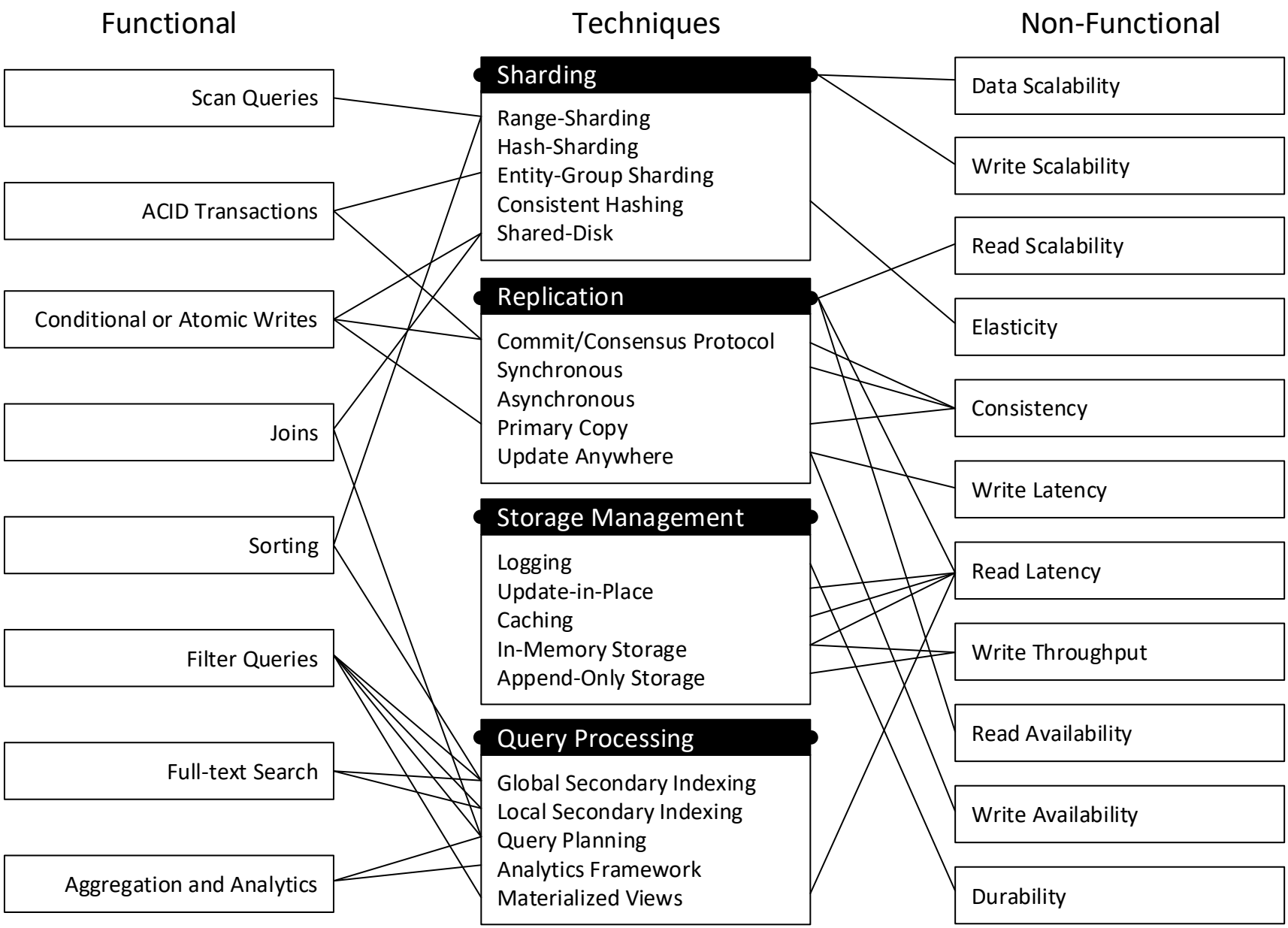
NoSQL Foundations and Motivation

The NoSQL Toolbox: Common Techniques

NoSQL Systems & Decision Guidance

Scalable Real-Time Databases and Processing

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


Functional
Techniques
Non-Functional
Scan Queries
ACID Transactions
Conditional or Atomic Writes
Joins
Sorting
Filter Queries
Full-text Search
Aggregation and Analytics
Sharding
Range-Sharding
Hash-Sharding
Entity-Group Sharding
Consistent Hashing
Shared-Disk
Replication
Commit/Consensus Protocol
Synchronous
Asynchronous
Primary Copy
Update Anywhere
Storage Management
Logging
Update-in-Place
Caching
In-Memory Storage
Append-Only Storage
Query Processing
Global Secondary Indexing
Local Secondary Indexing
Query Planning
Analytics Framework
Materialized Views
Data Scalability
Write Scalability
Read Scalability
Elasticity
Consistency
Write Latency
Read Latency
Write Throughput
Read Availability
Write Availability
Durability

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.

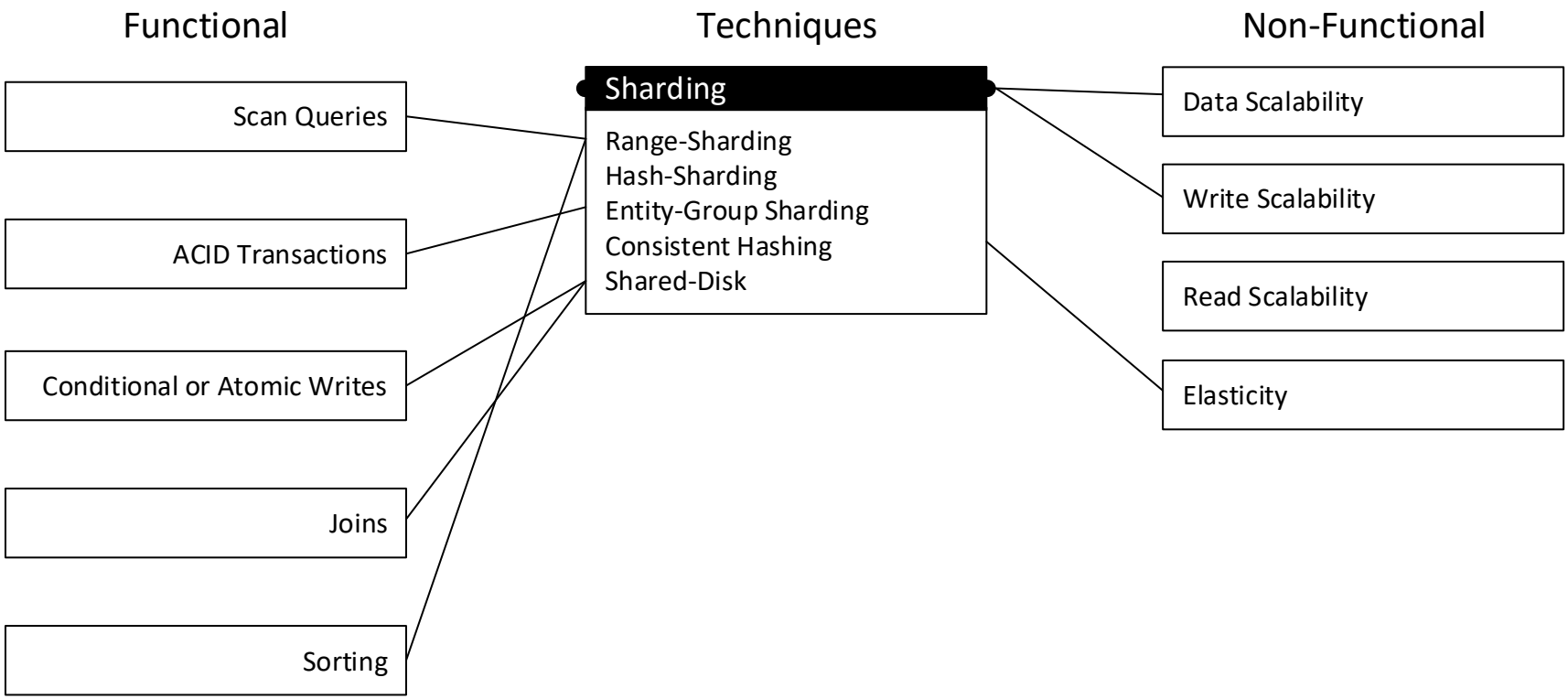
There are dozens of NoSQL database systems and it is hard to keep track of where they excel, where they fail or even where they differ, as implementation details change quickly and feature sets evolve over time. In this article, we therefore aim to provide an overview of the NoSQL landscape by discussing 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

NoSQL Databases: a Survey and Decision Guidance

Felix Gessert
Aug 15, 2016 · 26 min read

Together with our colleagues at the University of Hamburg, we—that is Felix Gessert, Wolfram Wingerath, Steffen Friedrich and Norbert Ritter—presented an overview over the NoSQL landscape at SummerSOC'16 last month. Here is the written gist. We give our best to convey the condensed NoSQL knowledge we gathered building Baqend.

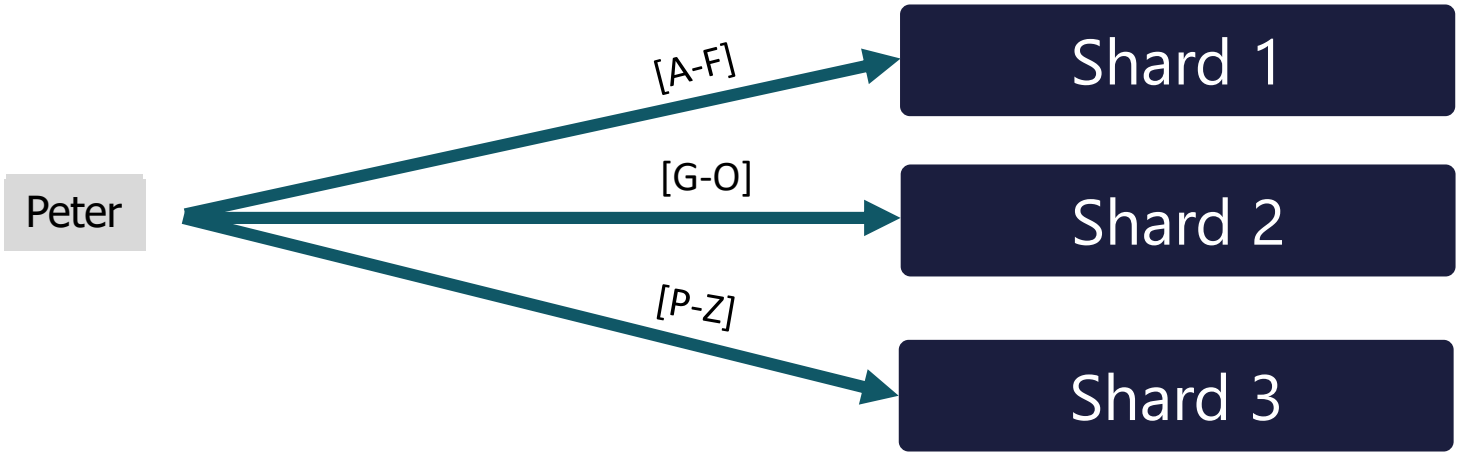
As a blog post: **blog.baqend.com**


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)

Scaling Storage and Throughput

- Horizontal distribution of data over nodes



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

Sharding

Approaches

Hash-based Sharding

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

Implemented in

Range-based Sharding

- Assigns ranges defined over fields (shard keys) to partitions
- **Pro**: Enables *Range Scans* and *Sorting*
- **Contra**: Repartitioning/balancing required

Entity-Group Sharding

- Explicit data co-location for single-node-transactions
- **Pro**: Enables *ACID Transactions*
- **Contra**: Partitioning not easily changable

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.

Sharding

Approaches

Hash-based Sharding

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

Implemented in

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

Range-based Sharding

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

Implemented in

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

Entity-Group Sharding

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

Implemented in

G-Store, MegaStore, Relational 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.

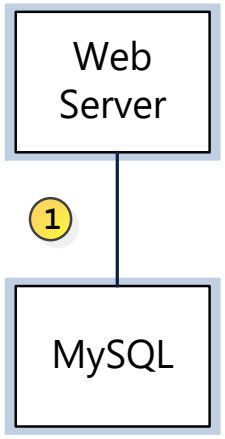
Problems of Application-Level Sharding

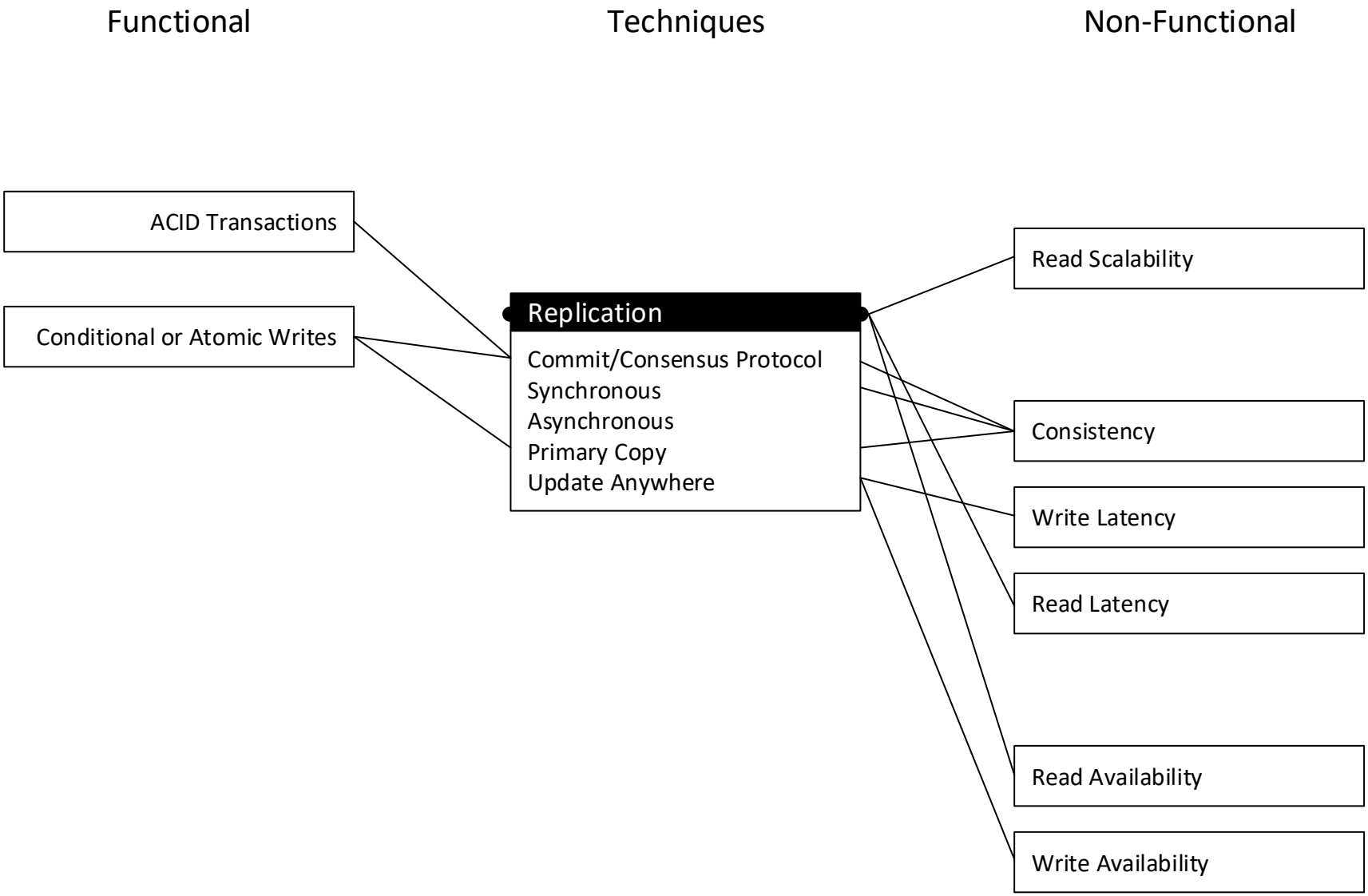
Example: **Tumblr**

- Caching
- Sharding from application

Moved towards:

- Redis
- HBase

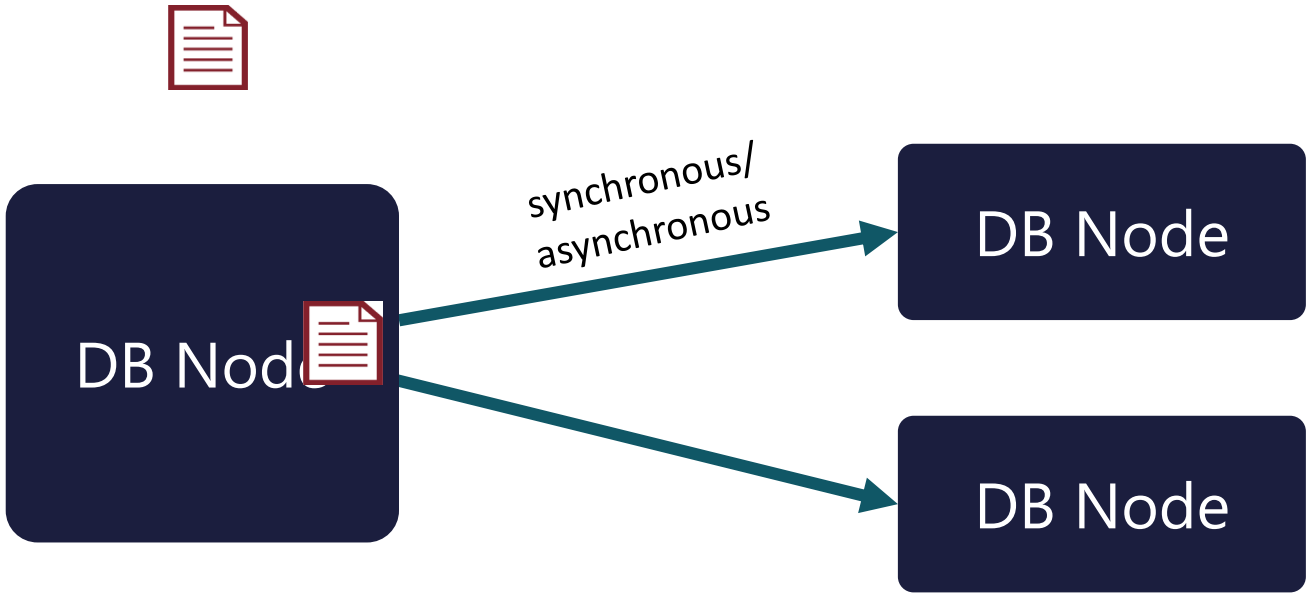



Functional
Techniques
Non-Functional
ACID Transactions
Conditional or Atomic Writes
Replication
Commit/Consensus Protocol
Synchronous
Asynchronous
Primary Copy
Update Anywhere
Read Scalability
Consistency
Write Latency
Read Latency
Read Availability
Write Availability

Replication

Read Scalability + Failure Tolerance

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

Synchronous (eager)

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

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

Replication: When

Asynchronous (lazy)

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

Implemented in

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

Synchronous (eager)

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

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*)

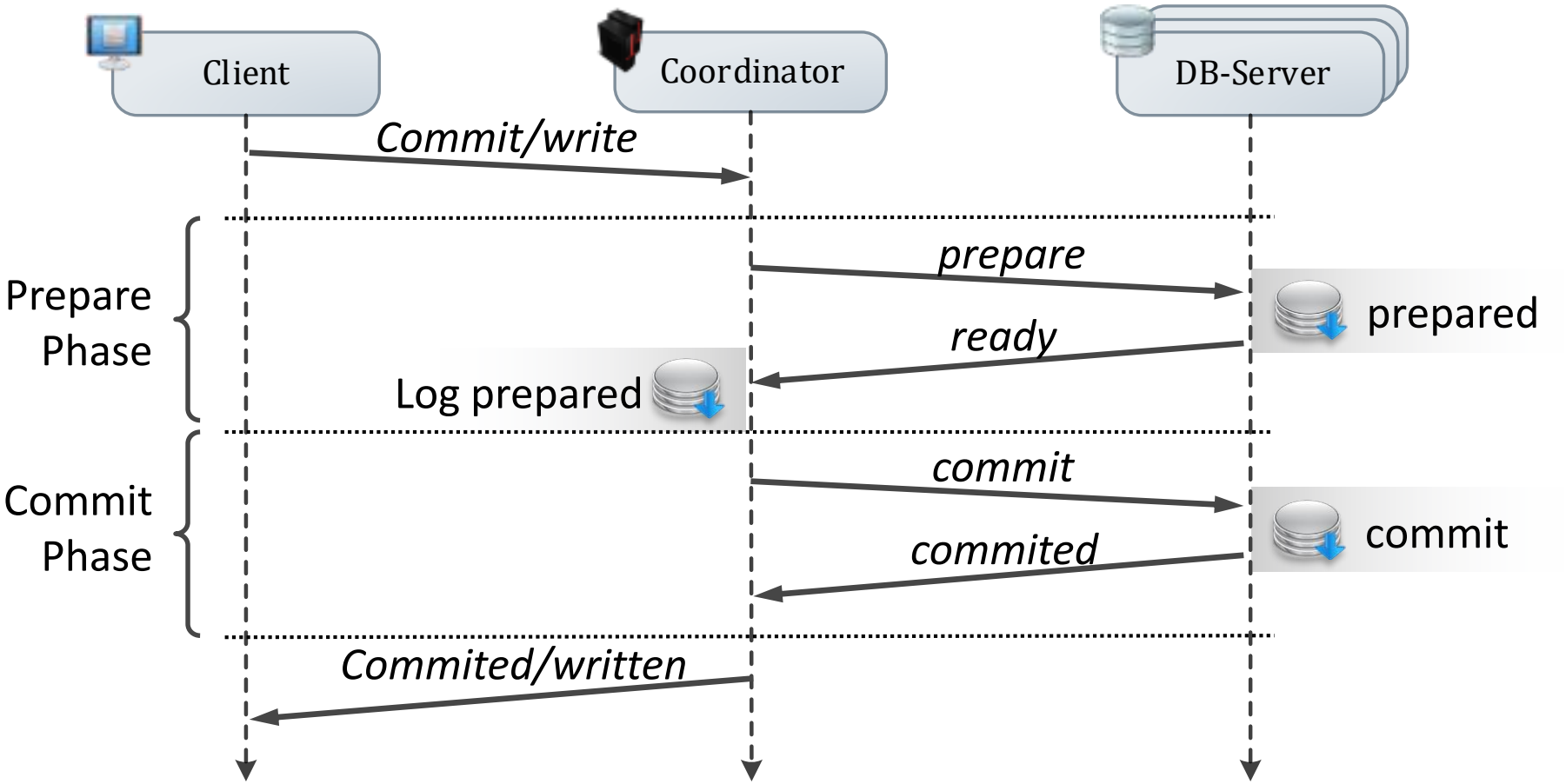
- Only a dedicated master is allowed to accept writes, slaves are read-replicas
- **Pro**: reads from the master are consistent
- **Contra**: master is a bottleneck and SPOF

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)

Synchronous Replication: 2PC



Error scenarios (timeouts)

In INITIAL:

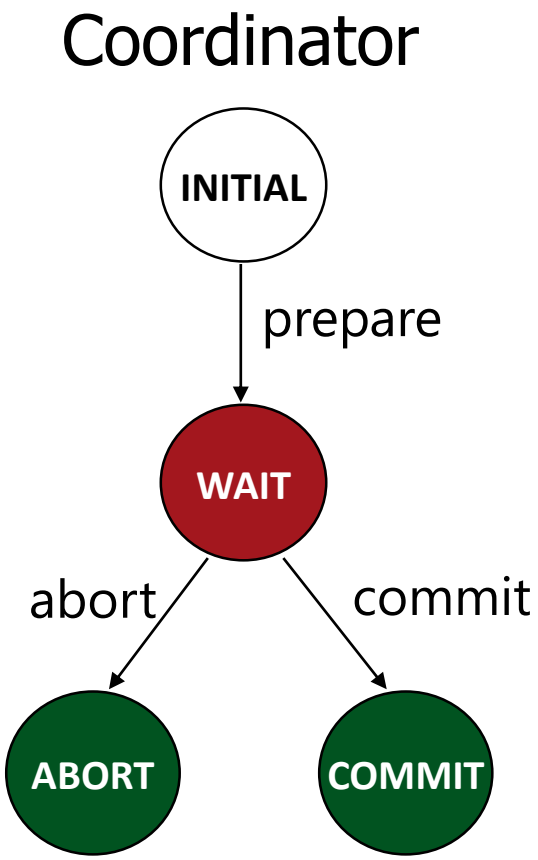
- No consequence

In WAIT:

- Abort

In ABORT oder COMMIT:

- Resend *commit/abort* and wait for all responses



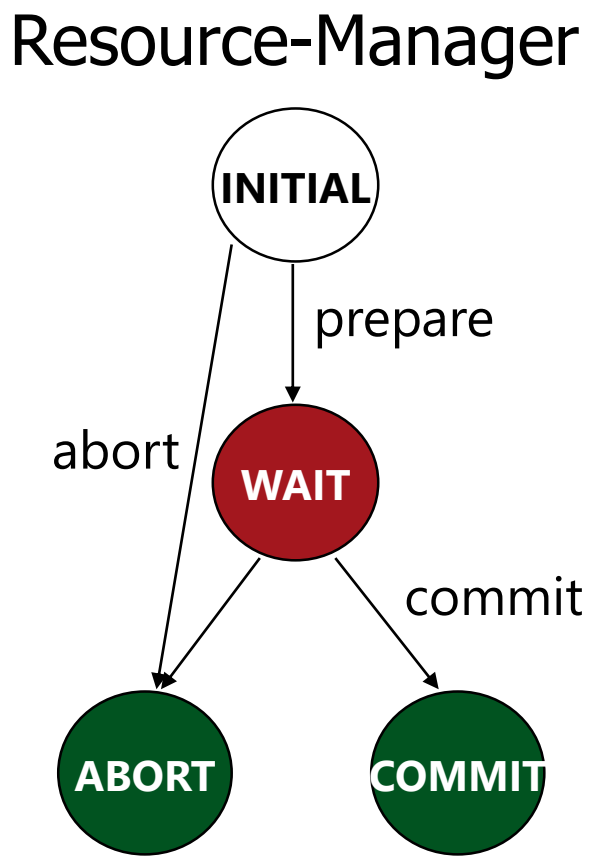
Error scenarios (timeouts)

In INITIAL:

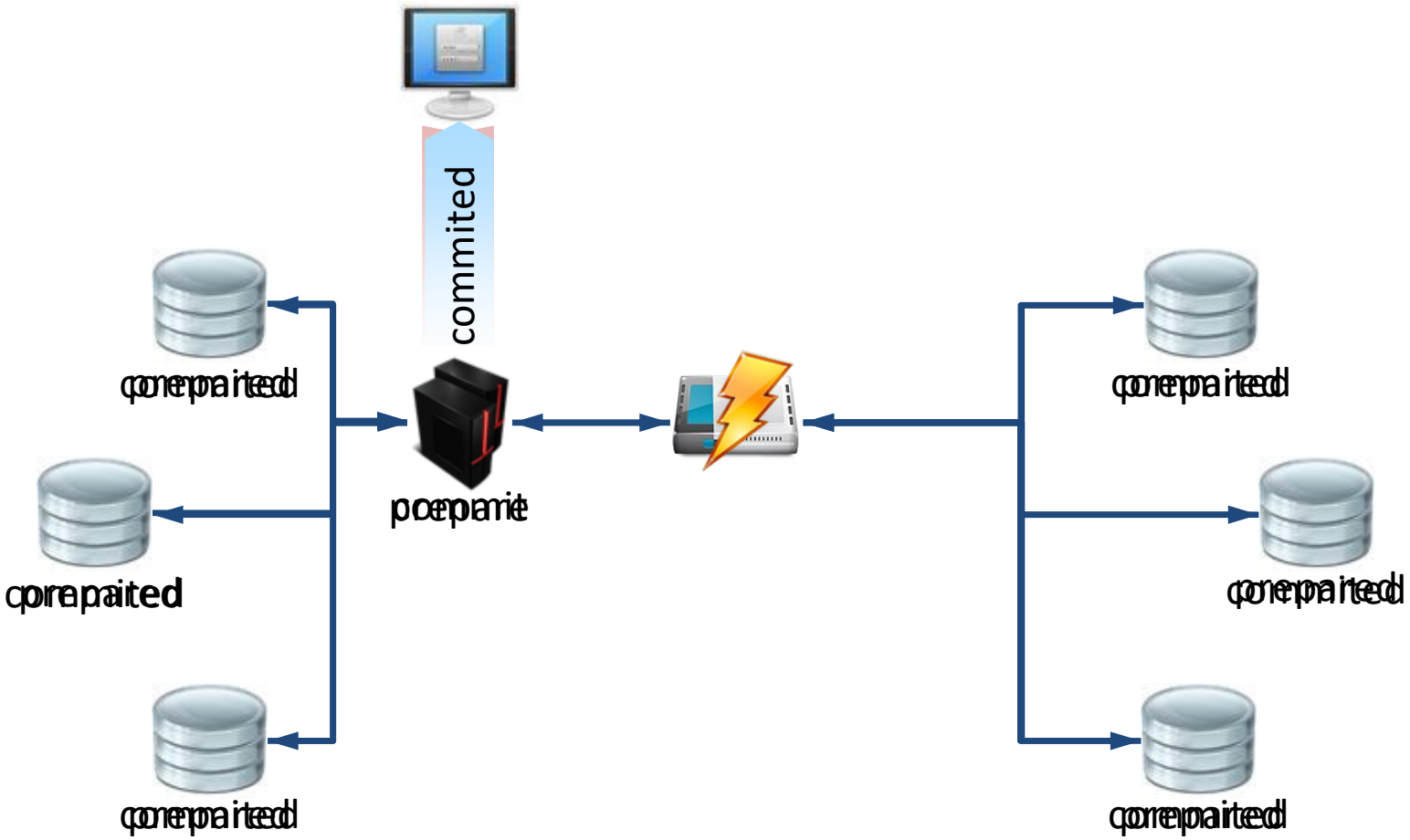
- Coordinator probably crashed → Abort („*Presumed-Abort-Protocol*“)

In WAIT:

- Wait for message from coordinator

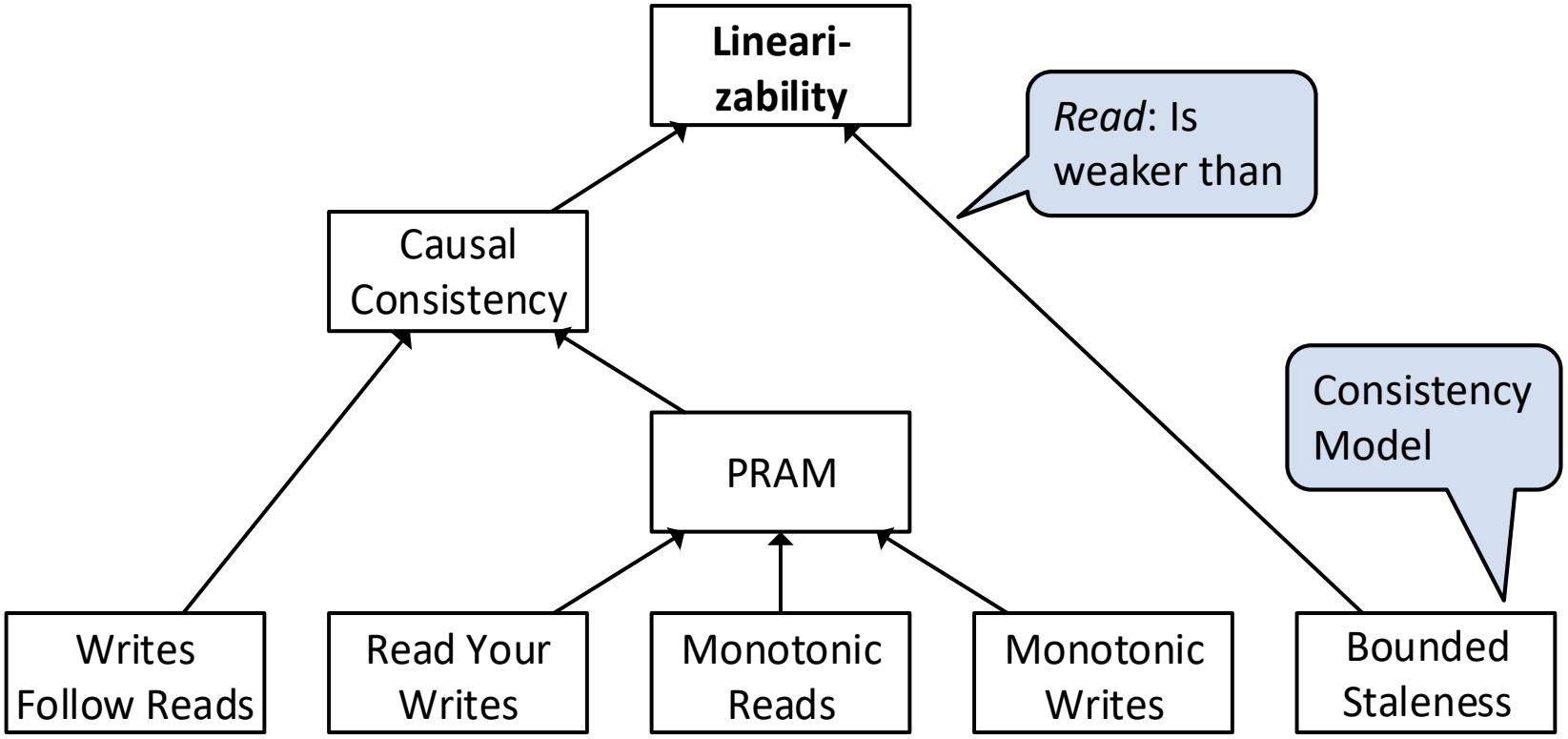


2PC is not *highly available*



NoSQL Consistency Models

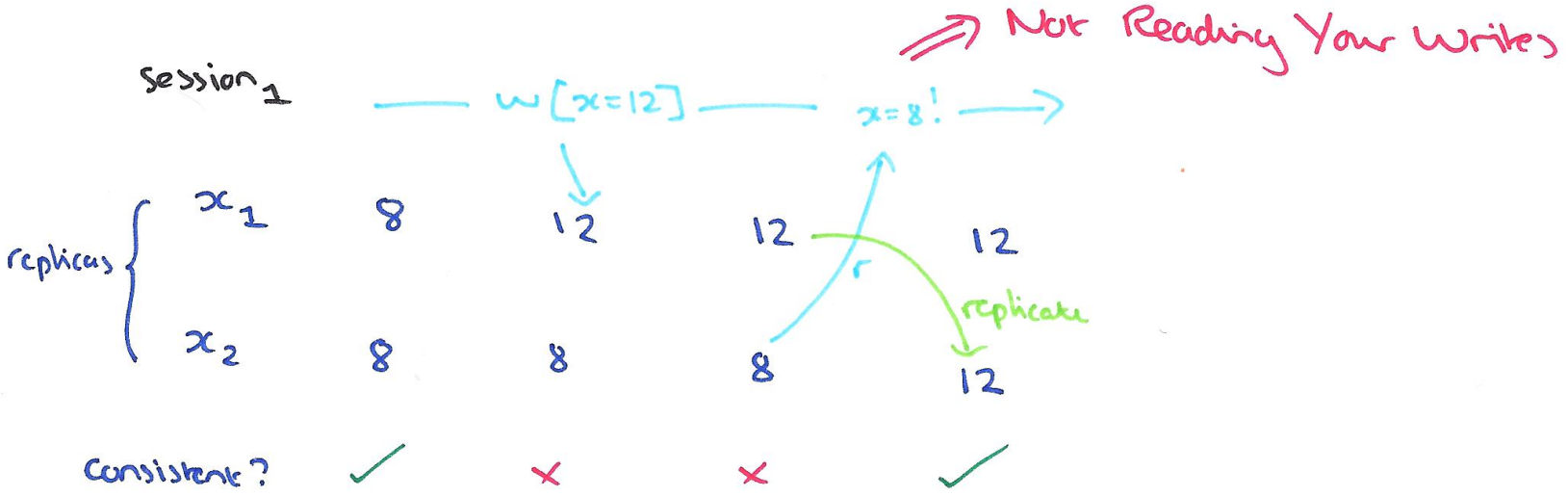
- Define models that relax strong consistency (=linearizability) in different aspects



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

Read Your Writes (RYW)

Definition: Once the user has written a value, subsequent reads will return this value (or newer versions if other writes occurred in between); the user will never see versions older than his last write.

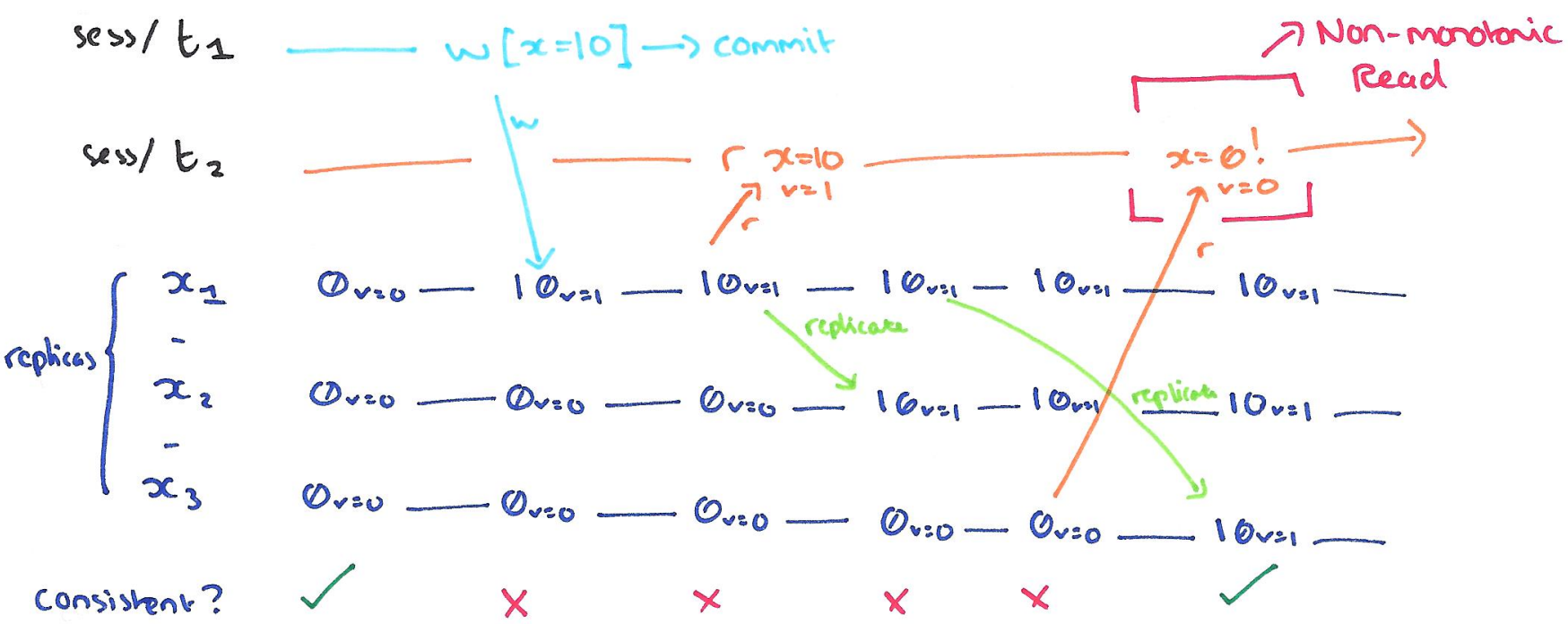


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

Monotonic Reads (MR)

Definition: Once a user has read a version of a data item on one replica server, it will never see an older version on any other replica server

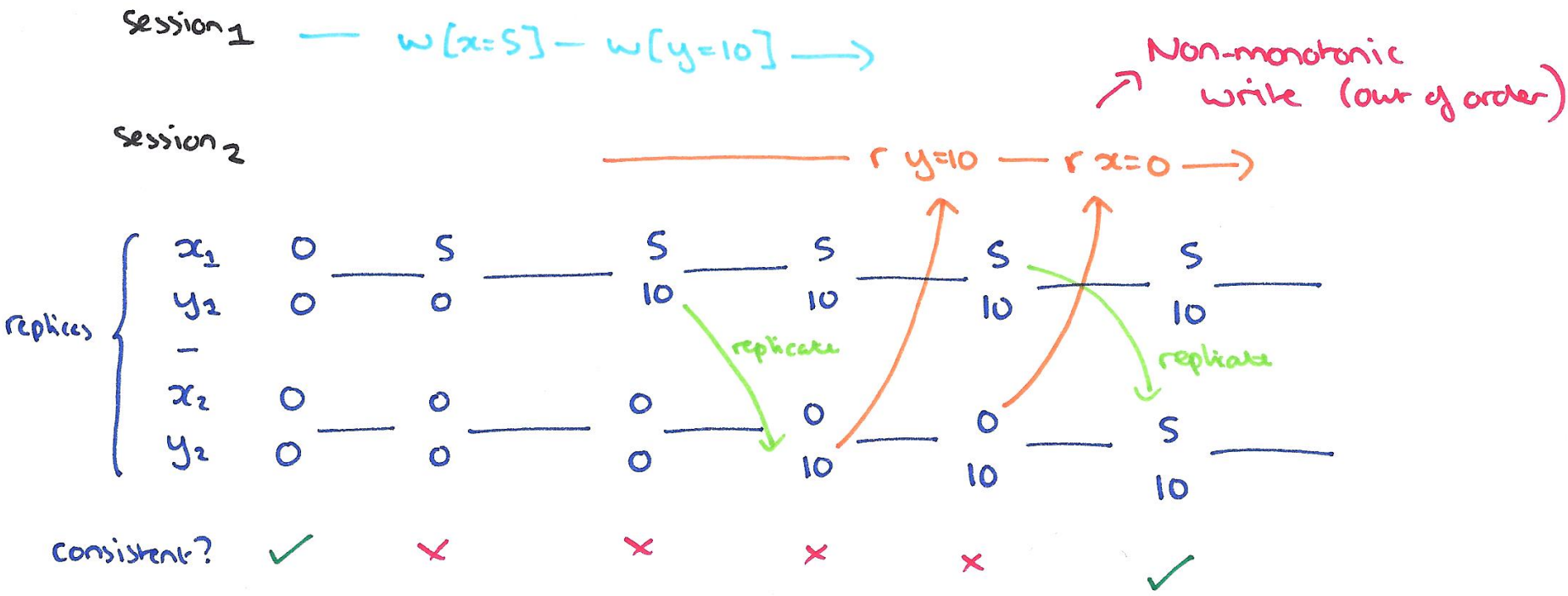


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

Montonic Writes (MW)

Definition: Once a user has written a new value for a data item in a session, any previous write has to be processed before the current one. I.e., the order of writes inside the session is strictly maintained.

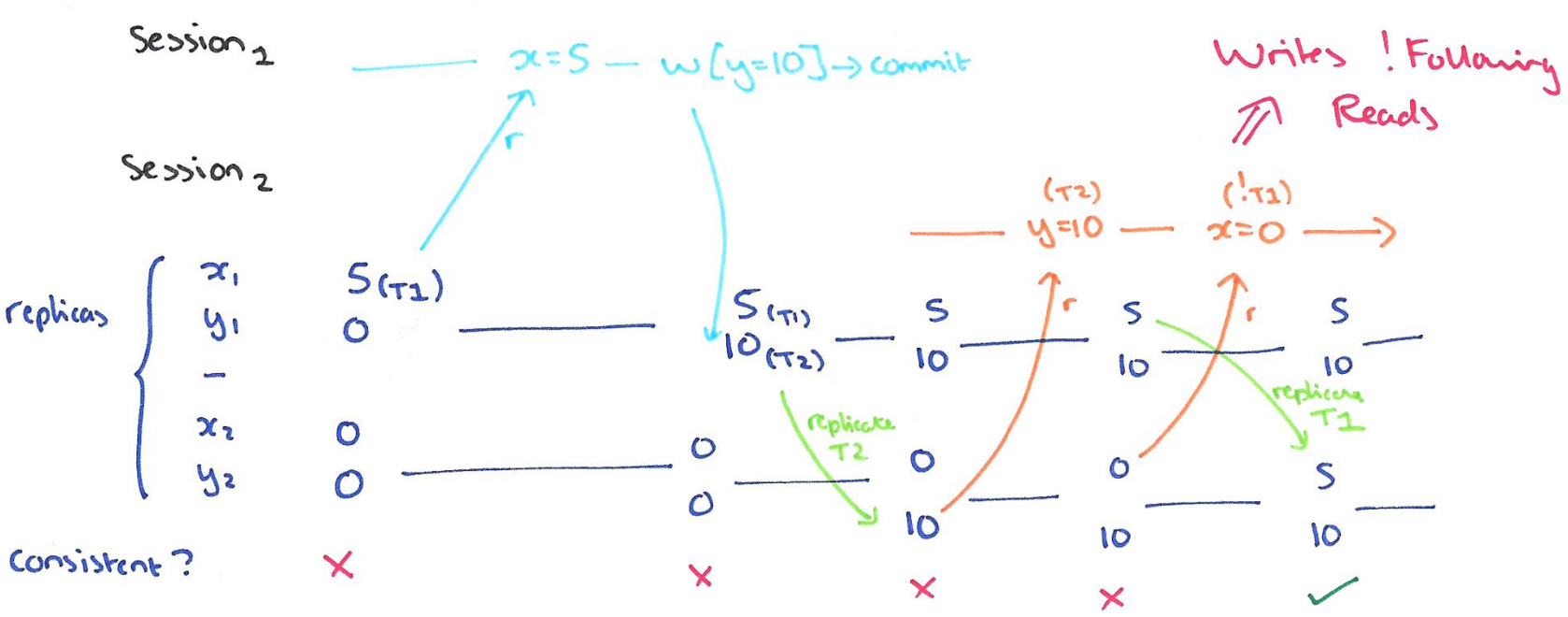


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

Writes Follow Reads (WFR)

Definition: When a user reads a value written in a session after that session already read some other items, the user must be able to see those *causally relevant* values too.



https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

Bounded Staleness

- Either **time-based**:

  t-Visibility (Δ-atomicity): the inconsistency window comprises at most t time units; that is, any value that is returned upon a read request was up to date t time units ago.

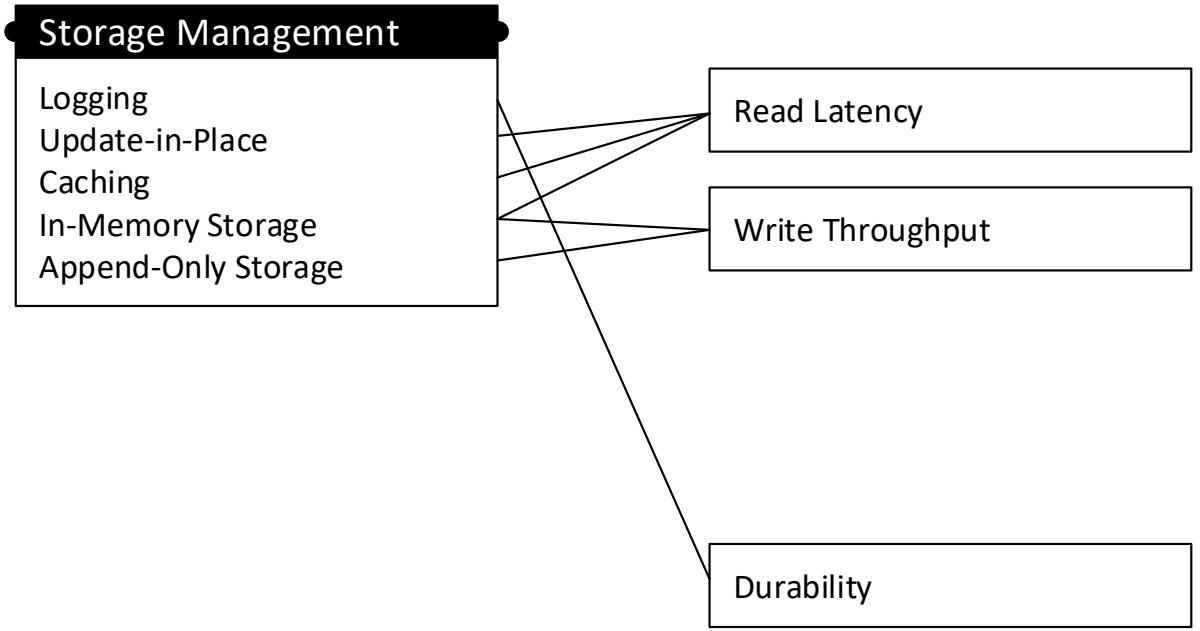
- Or **version-based:**

  k-Staleness: the inconsistency window comprises at most k versions; that is, lags at most k versions behind the most recent version.

- Both are *not* achievable with high availability

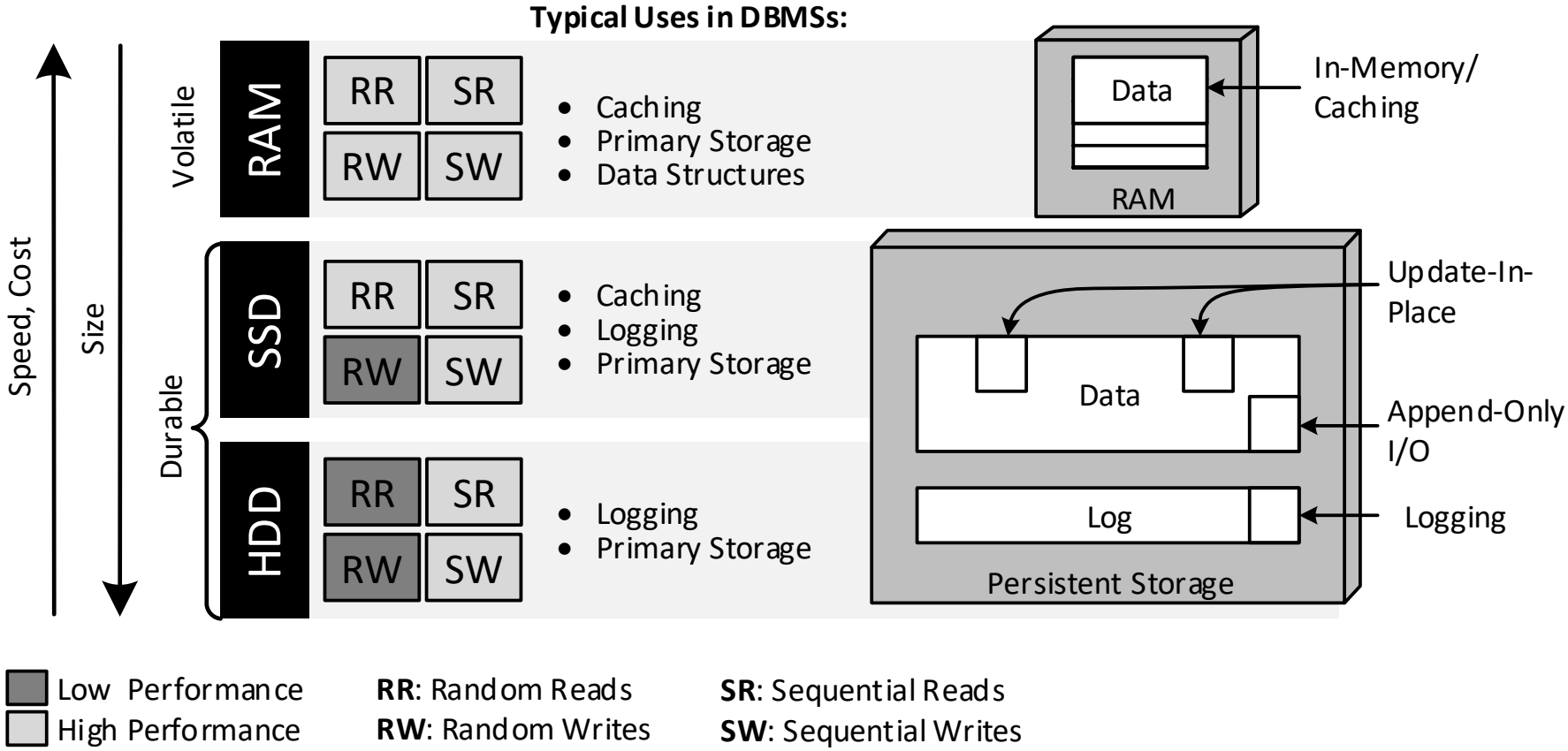
Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

Functional Techniques Non-Functional



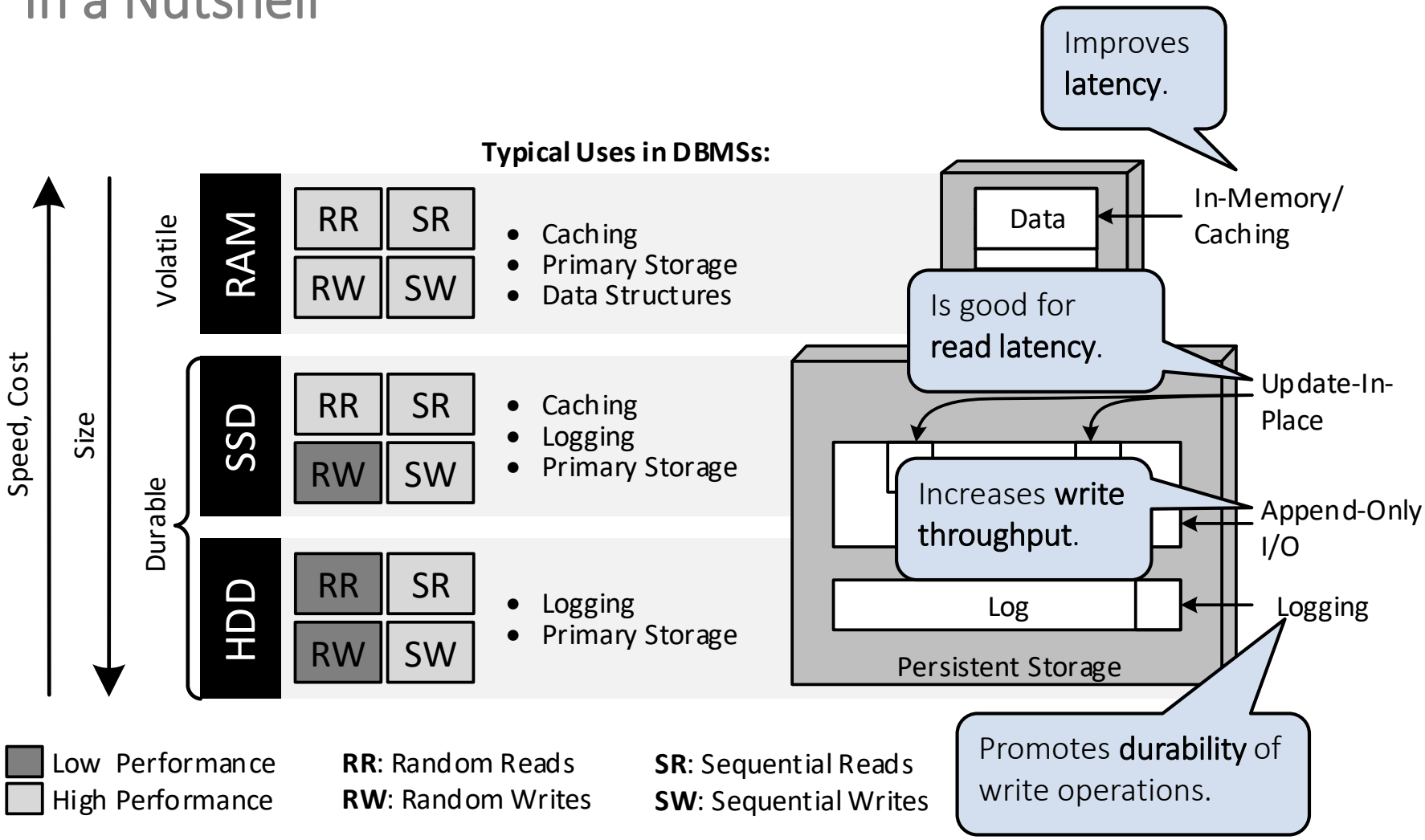
NoSQL Storage Management

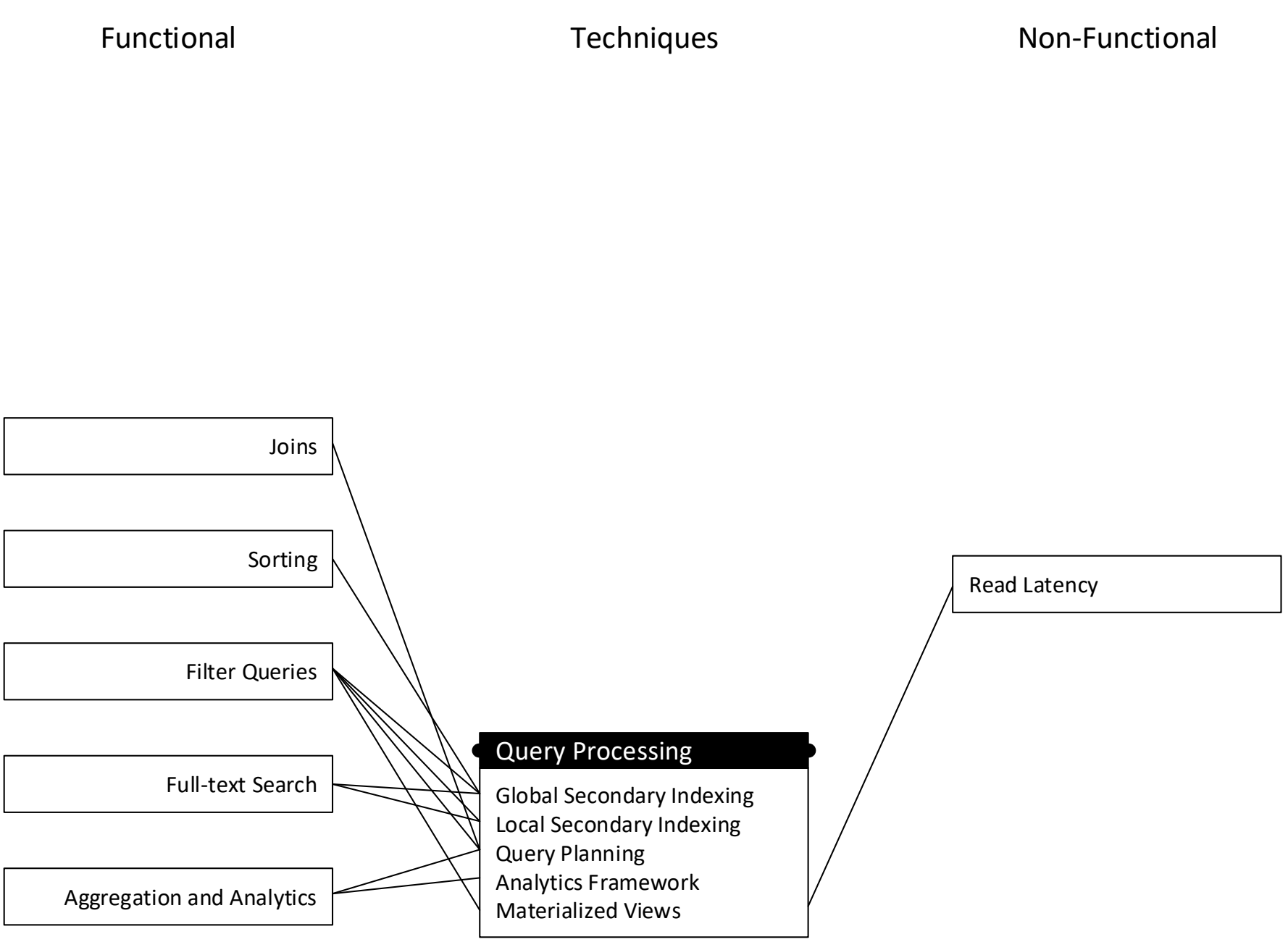
In a Nutshell



NoSQL Storage Management

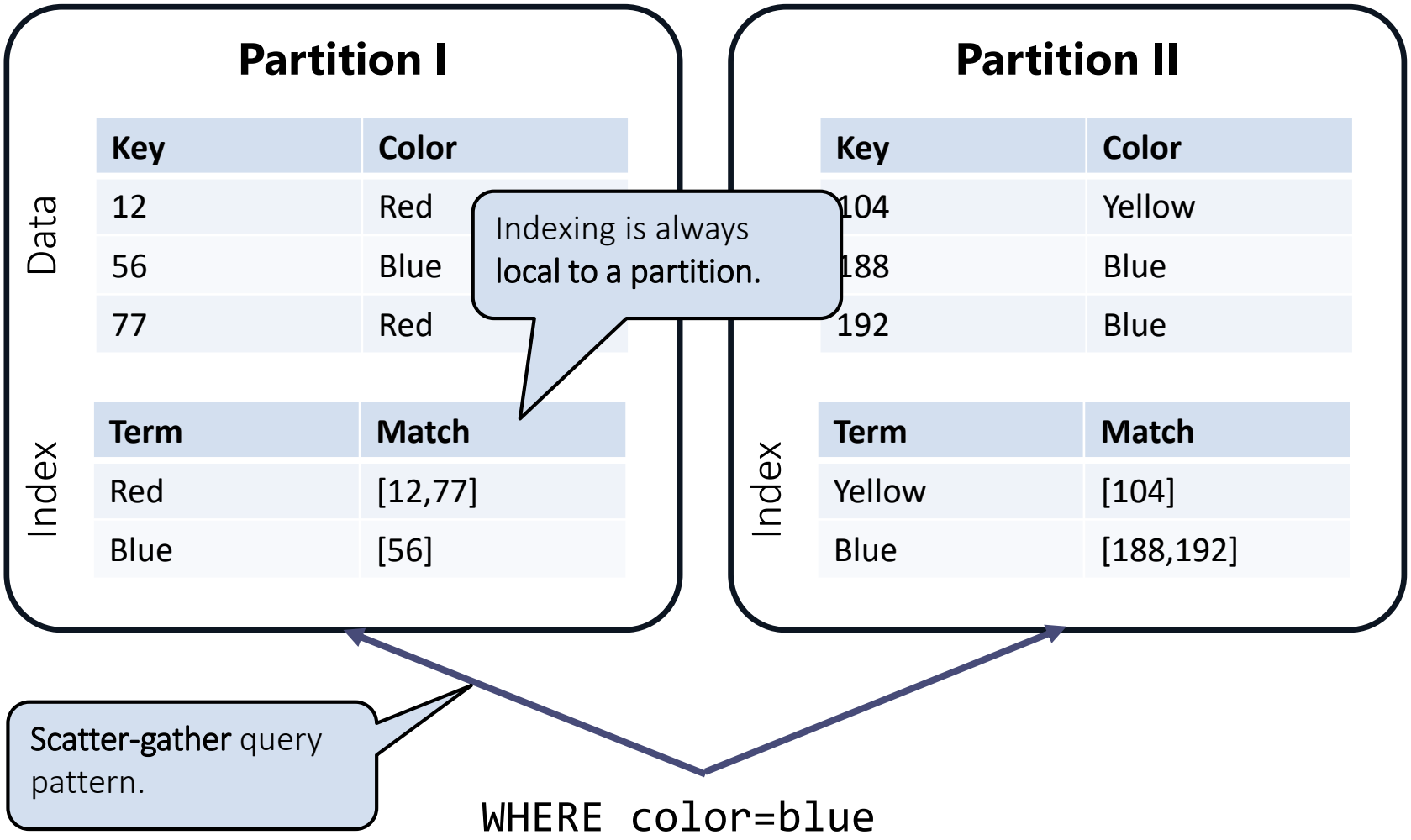
In a Nutshell




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

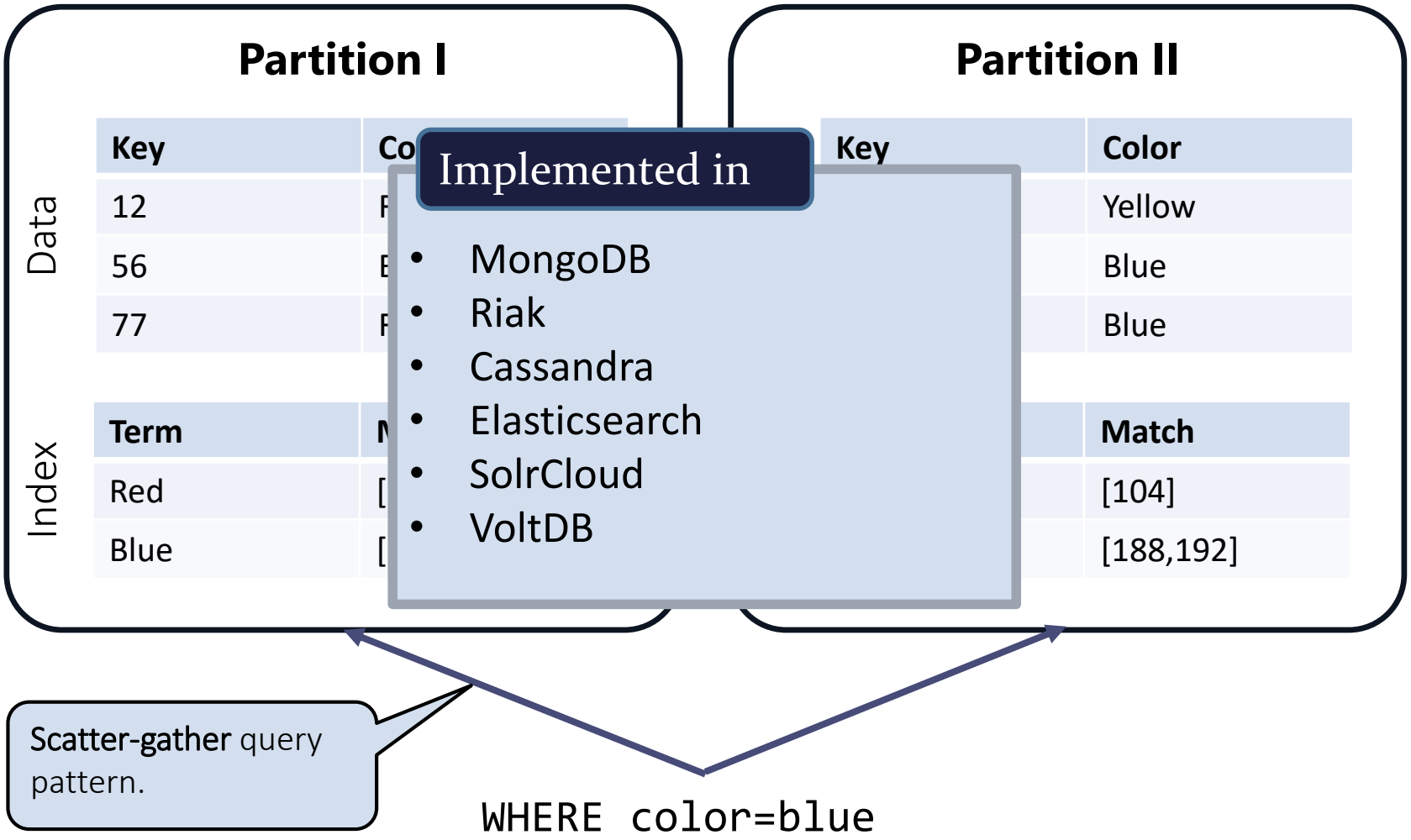
Partitioning By Data Item



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

Local Secondary Indexing

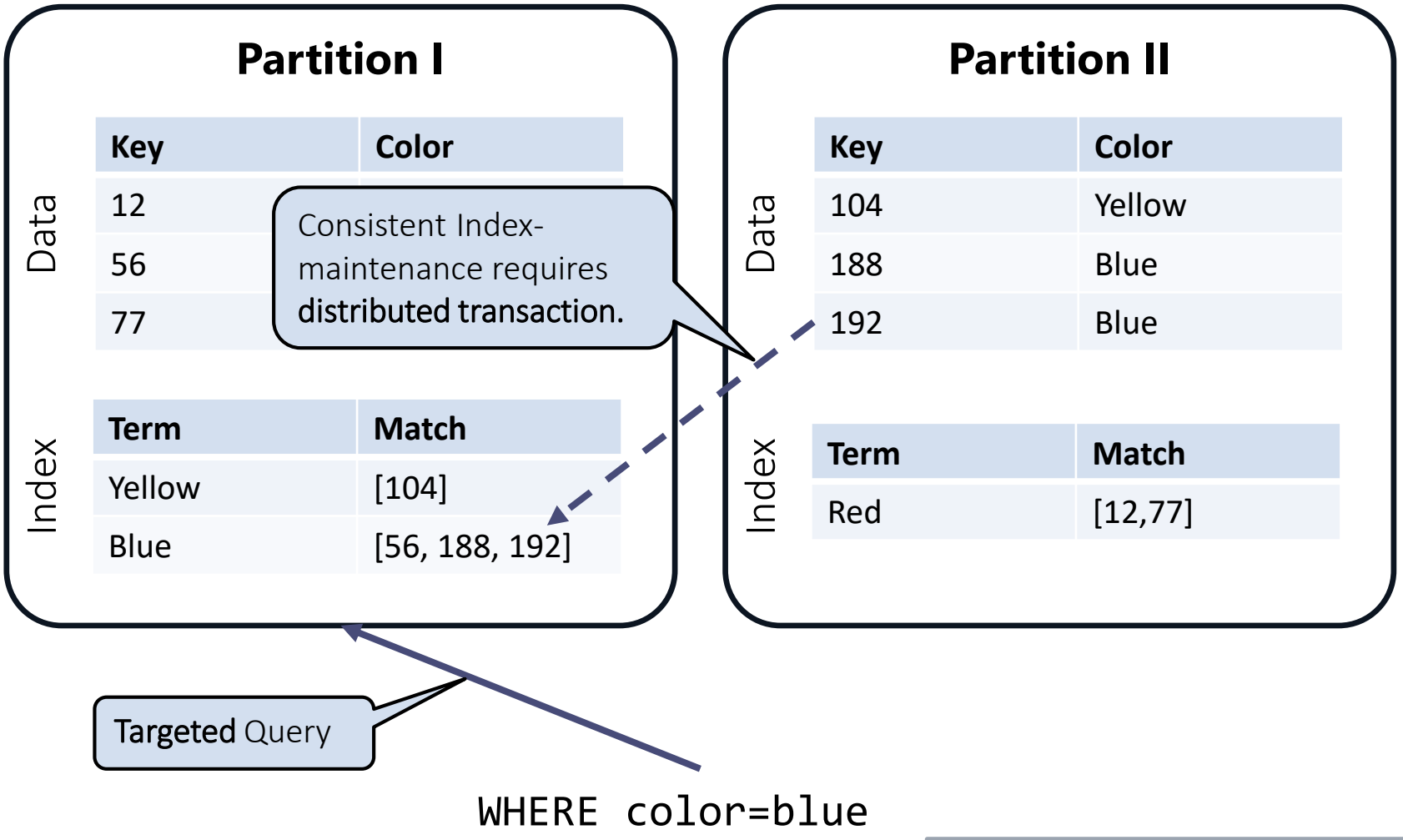
Partitioning By Data Item



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

Global Secondary Indexing

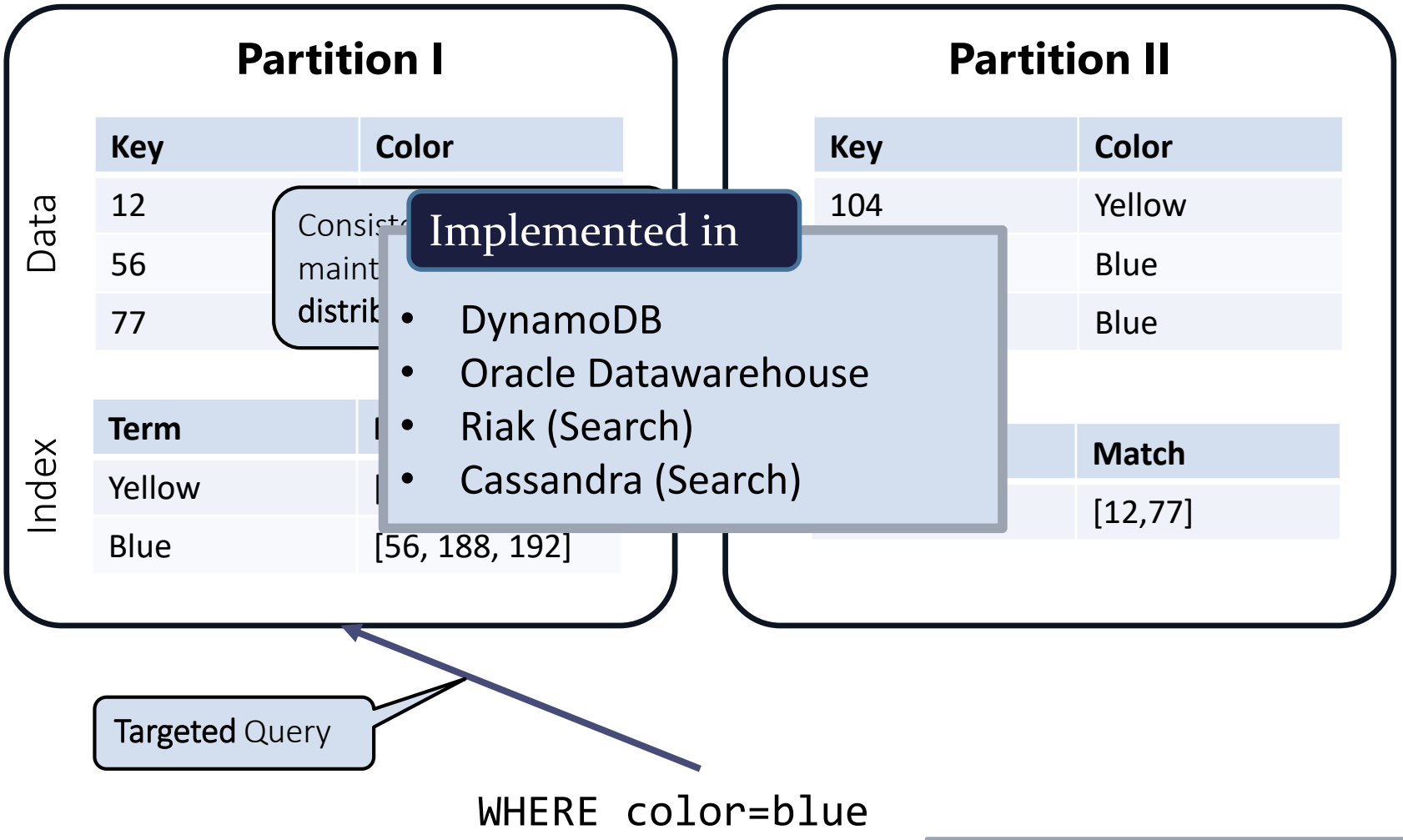
Partitioning By Query Term



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

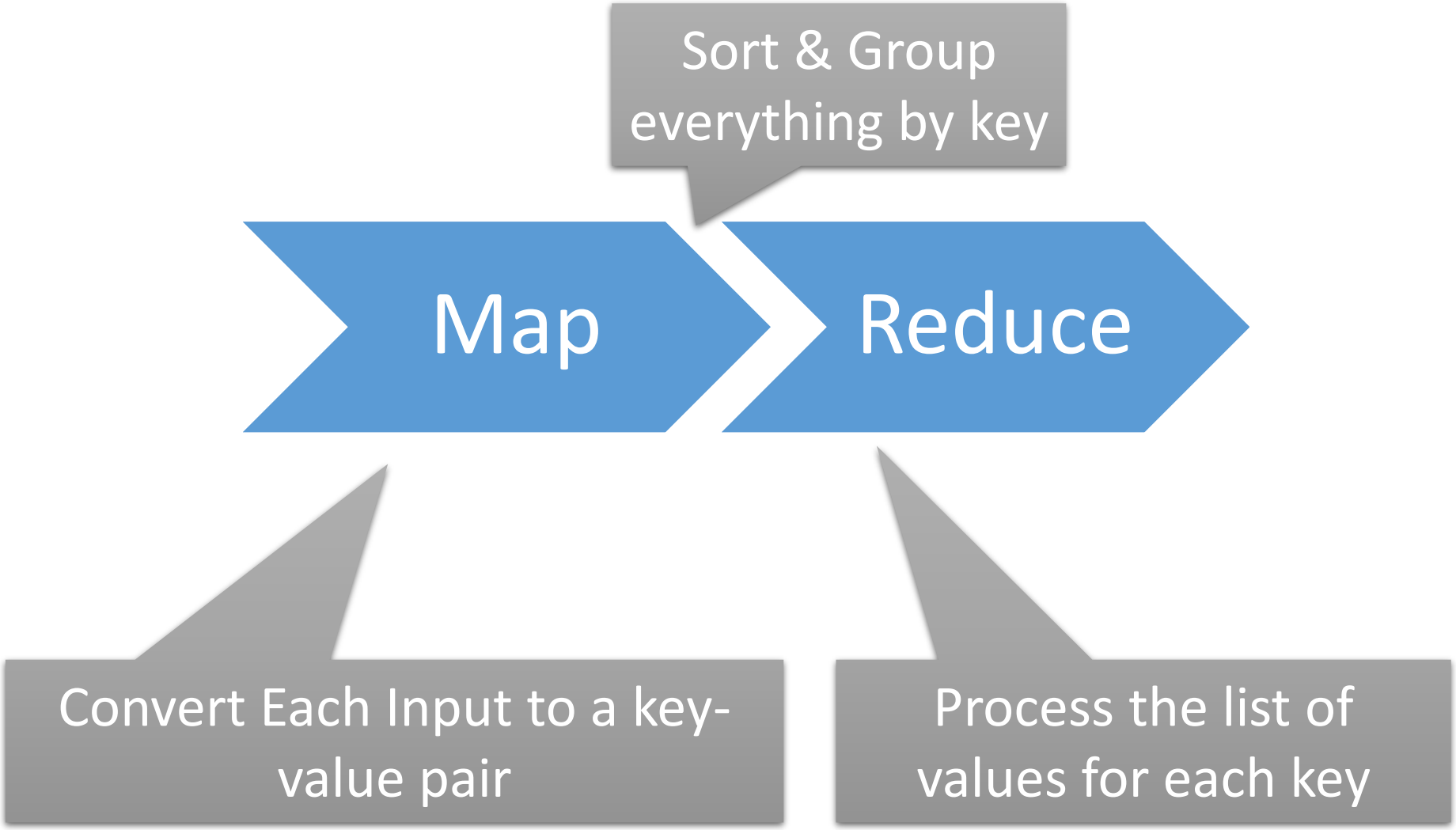
Global Secondary Indexing

Partitioning By Query Term



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

The MapReduce query concept



MapReduce: Word Count

```
map(zeilennr, text):
   for each word in text:
      emit(word, 1)

reduce(word, values):
   sum = 0
   for each v in values:
      sum = sum + v
   emit(word, sum)
```

1: ich bin ich

(ich, 1) (bin, 1) (ich, 1)

(ich, [1,1])

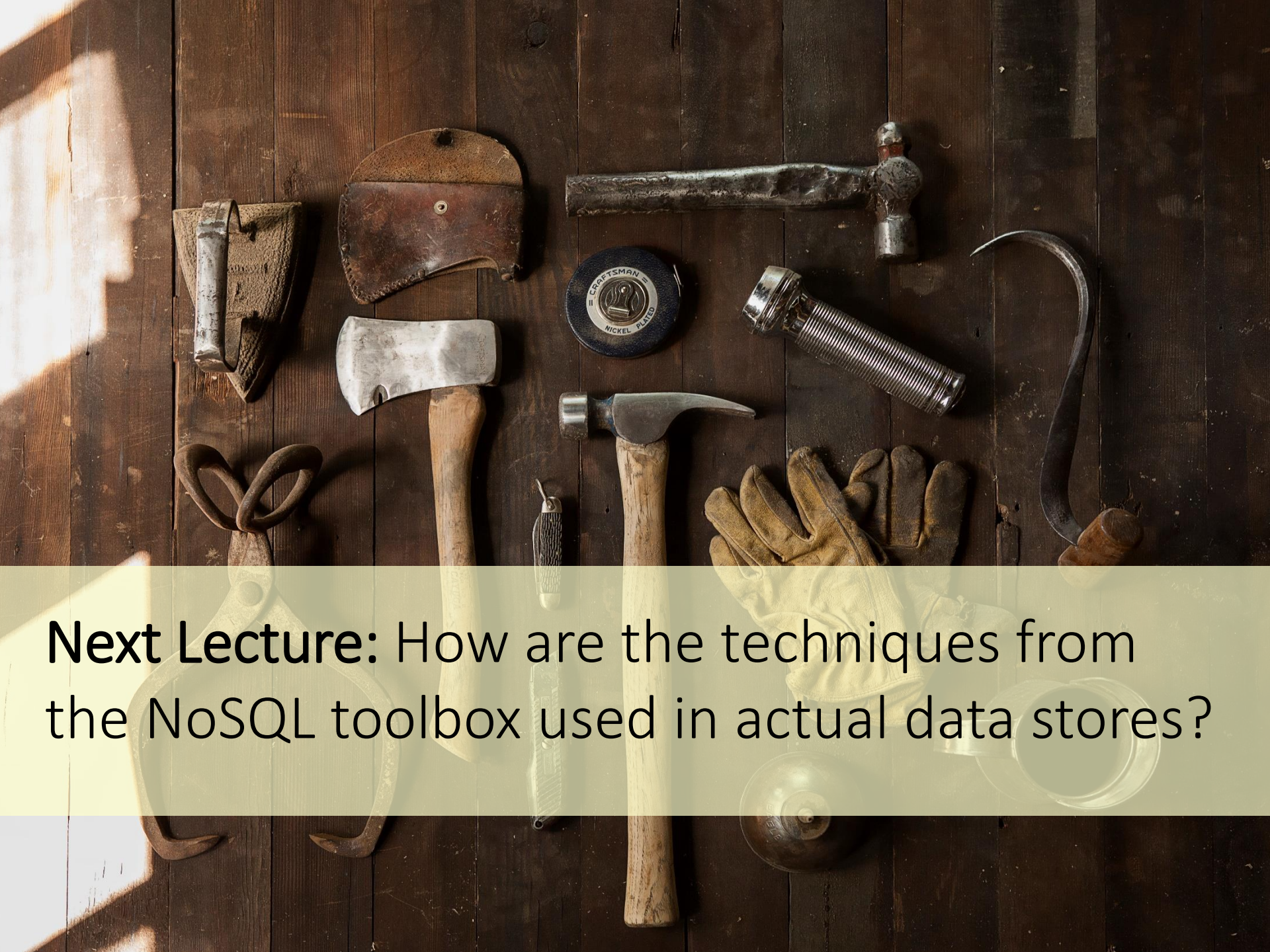
(bin, [1])

(ich, 2) (bin, 1)

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 (e.g. through MapReduce)
- **Materialized Views**: similar to global indexing



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

Summary

- High **data volumes**, **unstructured** sources and **new applications** triggered 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
- **4 NoSQL Data Model:** Key-Value Stores, Document Stores, Wide-Column Stores, Graph Databases
- **2-Phase-Commit** (2PC): popular protocol for atomic commitment; without availability guarantee
- **Consistency** can be relaxed in various ways

Outline

NoSQL Foundations and Motivation

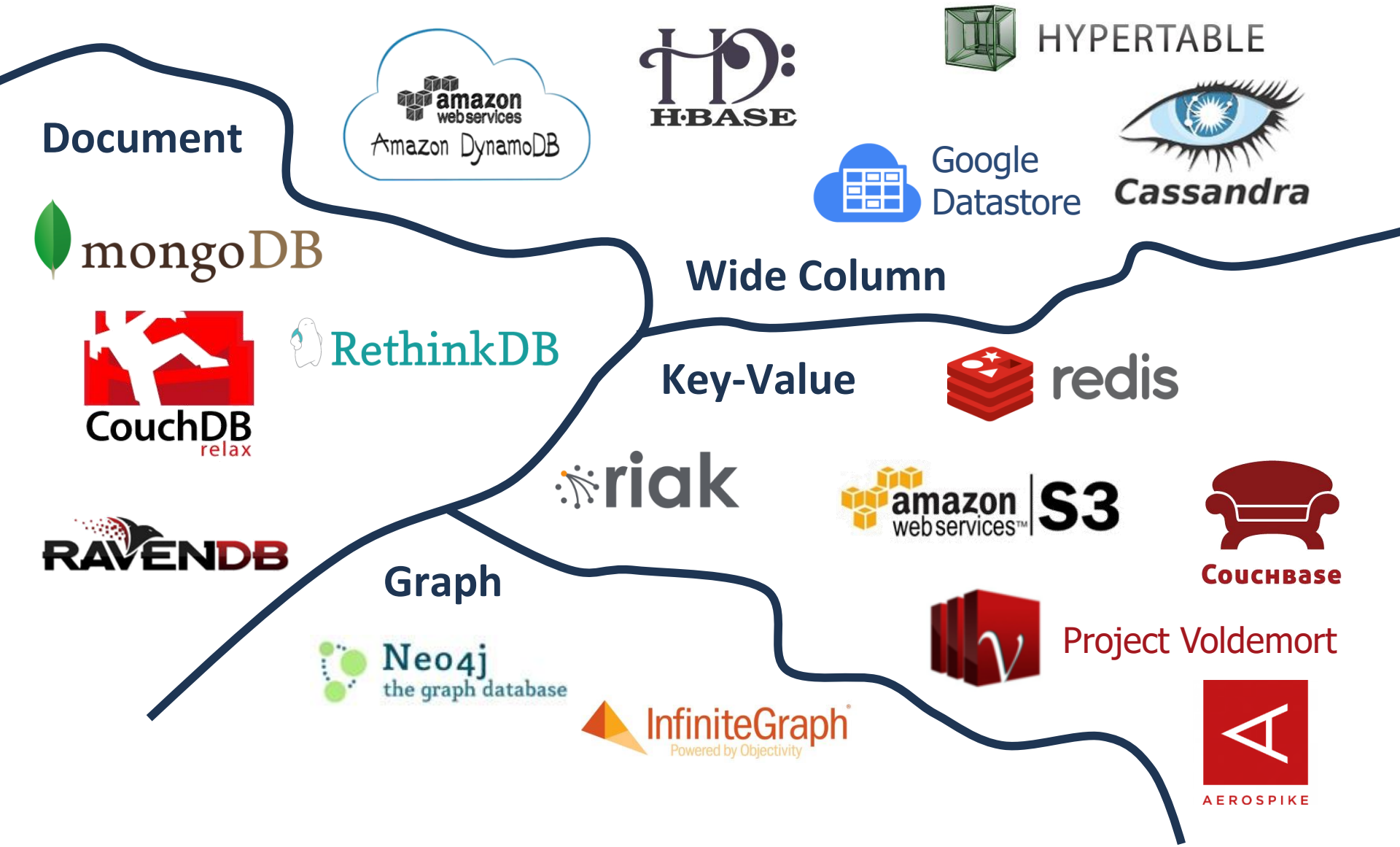
The NoSQL Toolbox: Common Techniques

NoSQL Systems & Decision Guidance

Scalable Real-Time Databases and Processing

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

NoSQL Landscape



Popularity (Feb 2019)

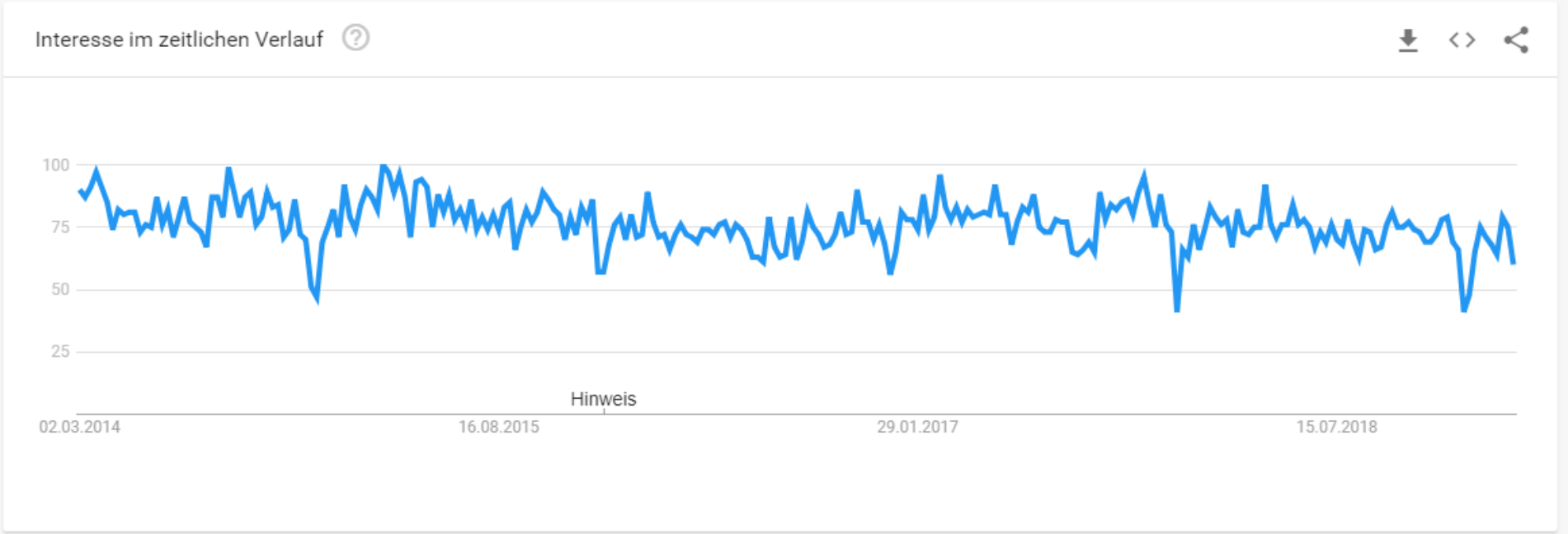
#	System	Model
1.	Oracle	Relational DBMS
2.	MySQL	Relational DBMS
3.	MS SQL Server	Relational DBMS
4.	PostgreSQL	Relational DBMS
5.	MongoDB	Document store
6.	DB2	Relational DBMS
7.	Microsoft Access	Relational
8.	Redis	Key-value store
9.	ElasticSearch	Search engine
10.	SQLite	Relational DBMS

11.	Cassandra	Wide column store
12.	MariaDB	Relational DBMS
13.	Splunk	Search engine
14.	Teradata	Search engine
15.	Hive	Relational
16.	Solr	Relational DBMS
17.	HBase	Relational DBMS
18.	FileMaker	Relational
19.	SAP Adaptive Server	Relational DBMS
20.	SAP HANA	Relational DBMS
21.	Amazon DynamoDB	Multi-model
22.	Neo4j	Graph DB
23.	Couchbase	Document store
24.	Memcached	Key-value store
25.	SQL Azure	Multi-model

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

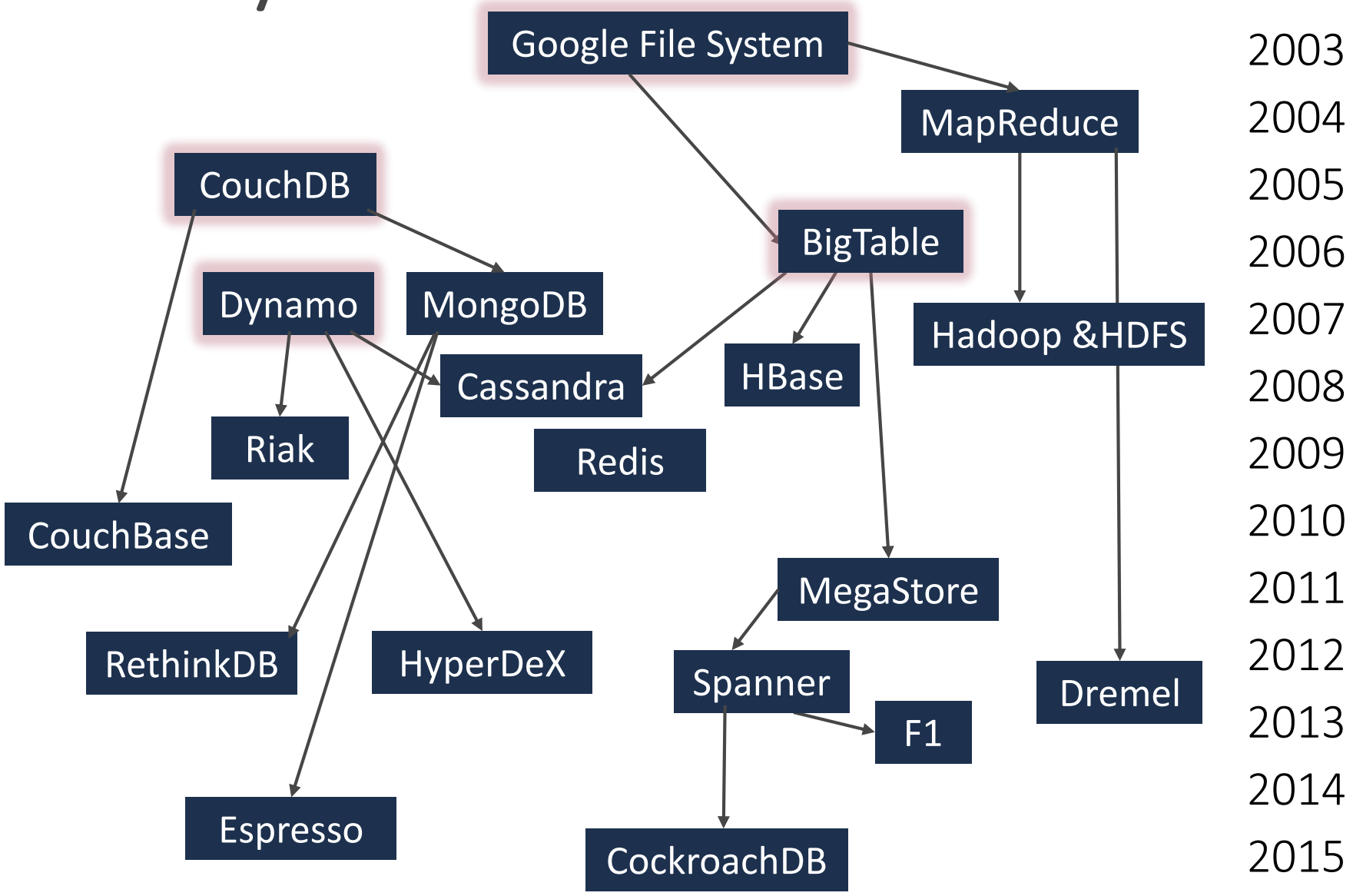
http://db-engines.com/de/ranking

NoSQL: Still a Thing in 2019



https://trends.google.com/trends/explore?date=today%205-y&q=nosql

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

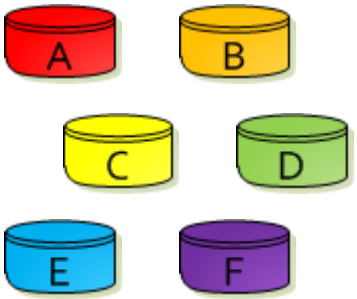
- **BigTable** (2006, Google)
  - **C**onsistent, **P**artition Tolerant
  - Wide-Column data model
  - Master-based, fault-tolerant, large clusters (1.000+ Nodes), **HBase**, **Cassandra**, **HyperTable, Accumolo**
- **Dynamo** (2007, Amazon)
  - **A**vailable, **P**artition tolerant
  - Key-Value interface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

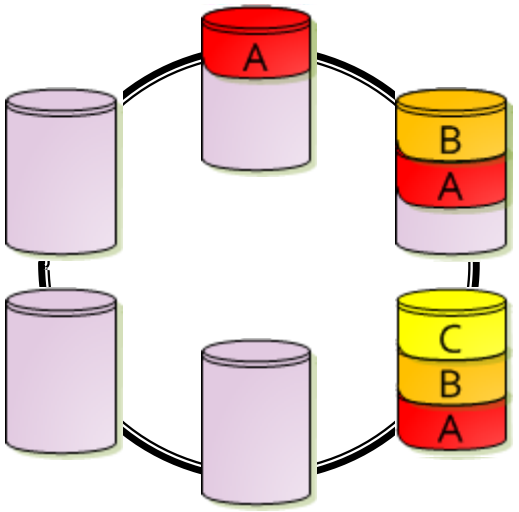
Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."

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

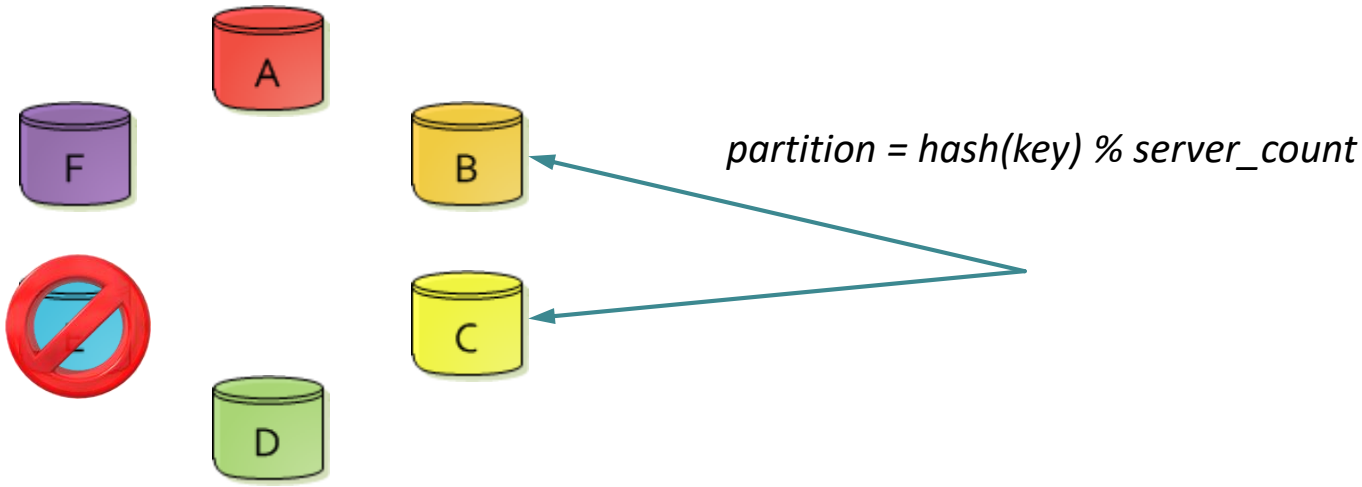




DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."

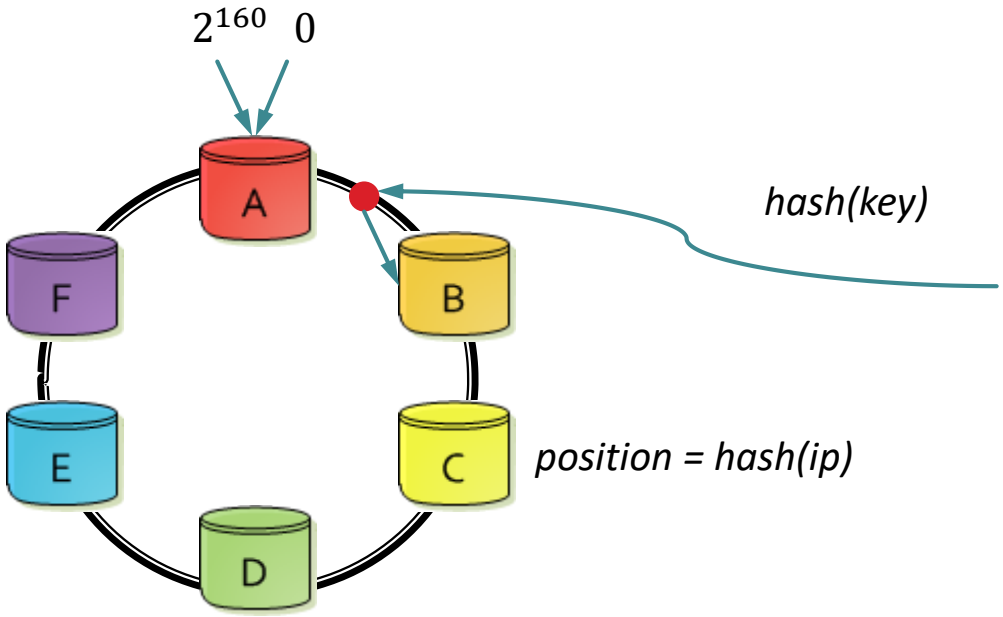
Consistent Hashing

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



Consistent Hashing

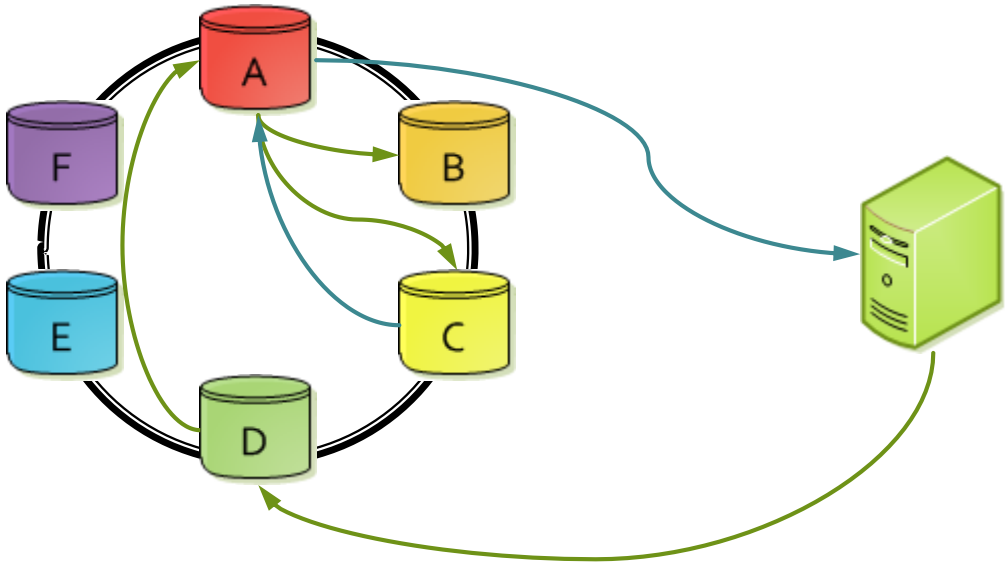
- Solution: **Consistent Hashing** – mapping of data to nodes is stable under topology changes



Reading

Parameters R, W, N

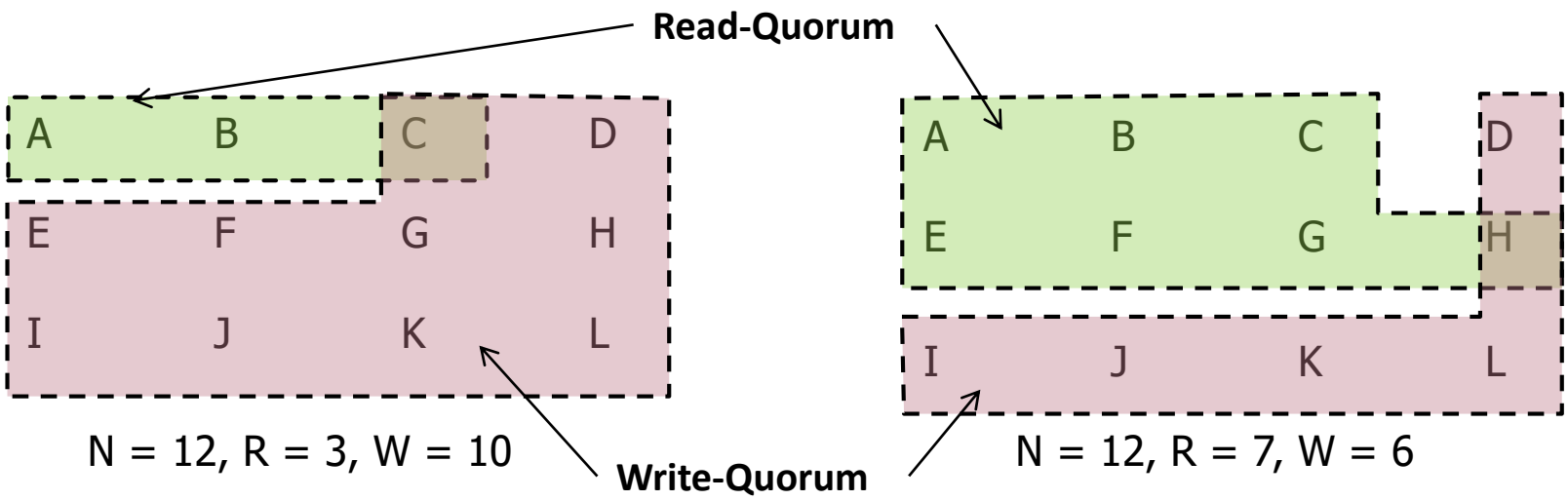
- An arbitrary node acts as a coordinator
- **N**: number of replicas
- **R**: number of nodes that need to confirm a read
- **W**: number of nodes that need to confirm a write



N=3
R=2
W=1

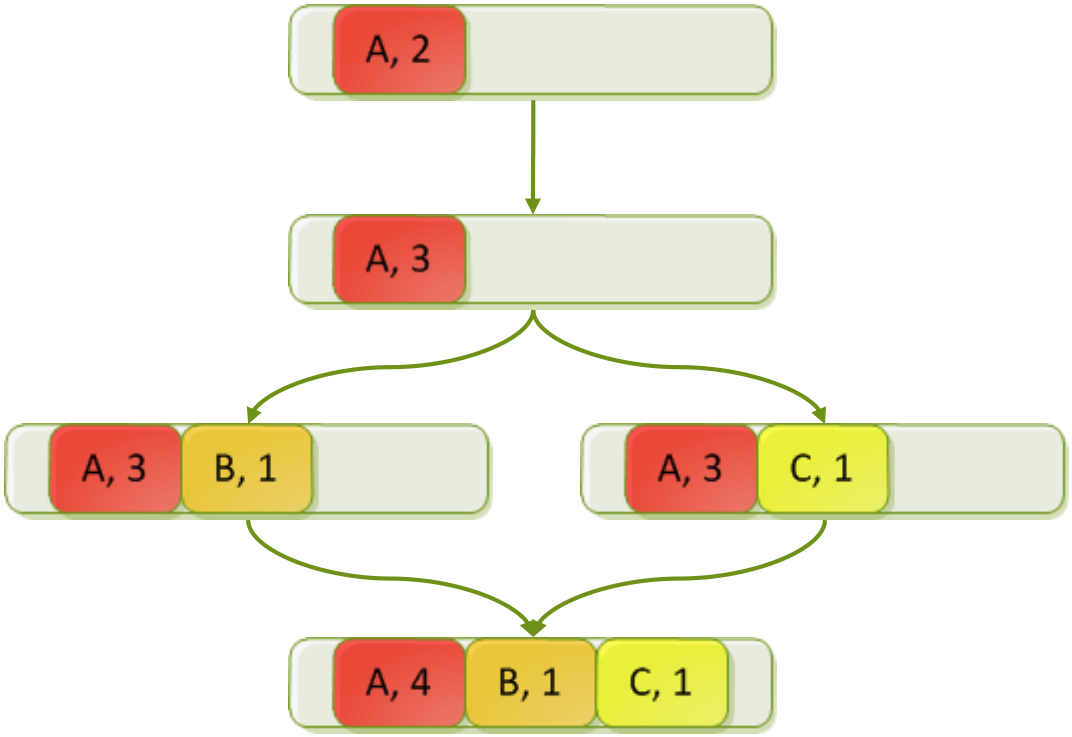
Quorums

- **N** (Replicas), **W** (Write Acks), **R** (Read Acks)
  - $R + W \leq N \Rightarrow$ No guarantee
  - $R + W > N \Rightarrow$ newest version included



Vector clocks

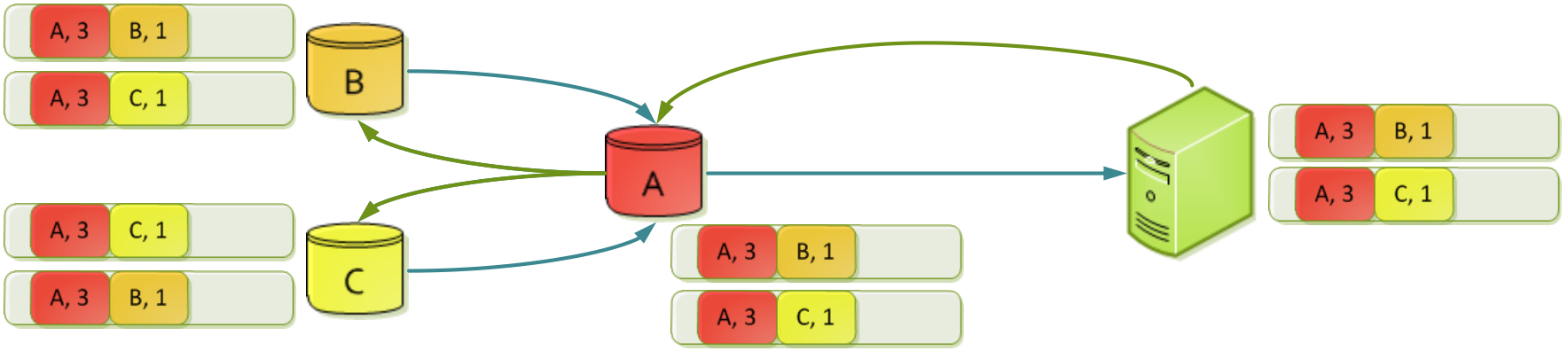
- Dynamo uses **Vector Clocks** for versioning



C. J. Fidge, Timestamps in message-passing systems that preserve the partial ordering (1988)

Versioning and Consistency

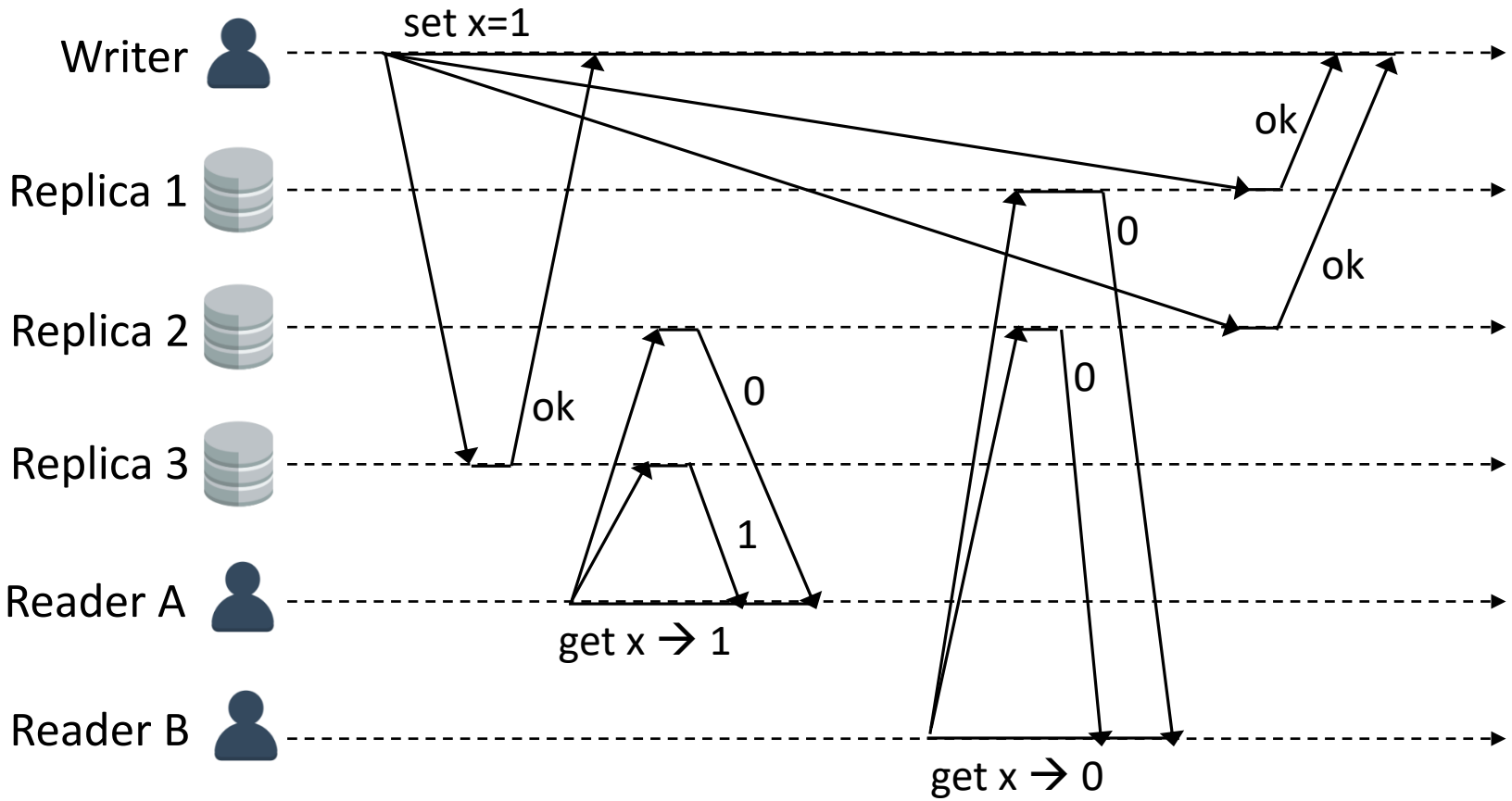
- $R + W \leq N \Rightarrow$ no consistency guarantee
- $R + W > N \Rightarrow$ newest acked value included in reads
- **Vector Clocks** used for versioning



Read Repair

$R + W > N$ does not imply linearizability

- Consider the following execution:

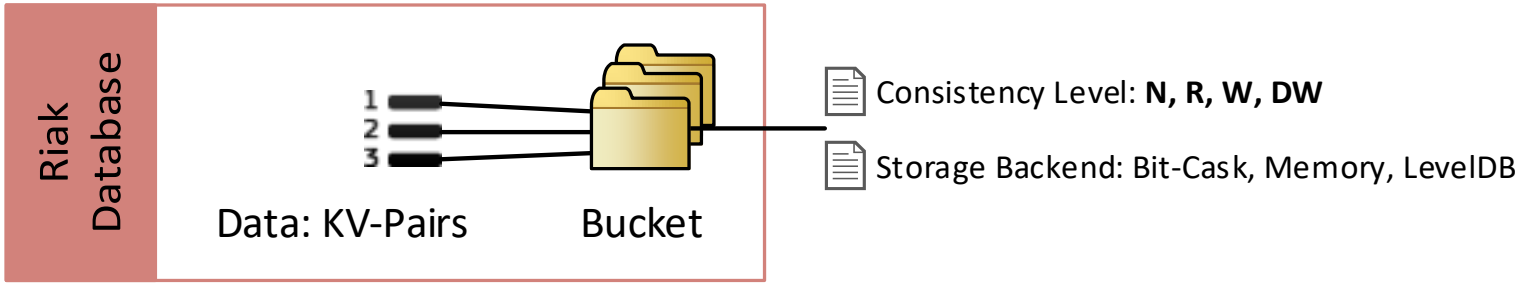


Kleppmann, Martin. "Designing data-intensive applications." (2016).

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
  - Option for **strongly consistent** buckets (experimental)
  - **Riak CS:** S3-like file storage, **Riak TS**: time-series database



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

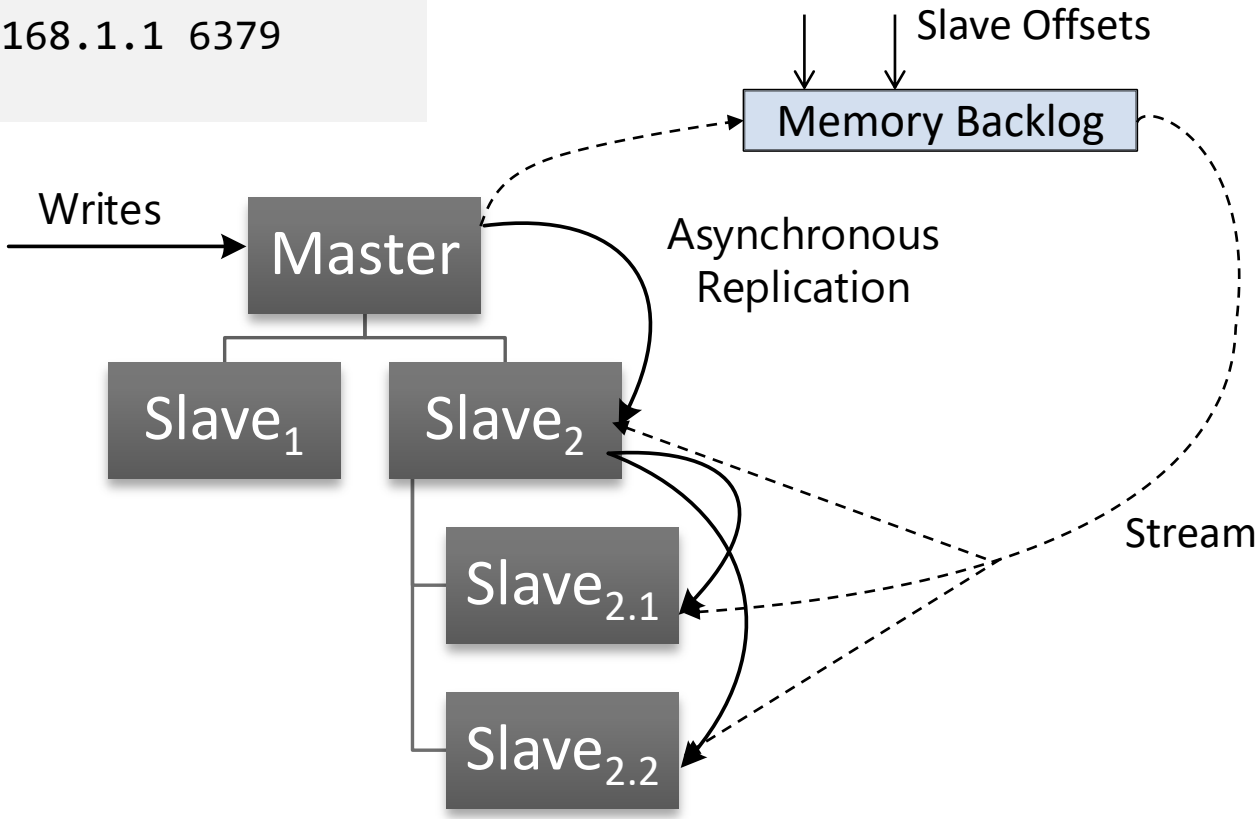
Redis (CA)

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

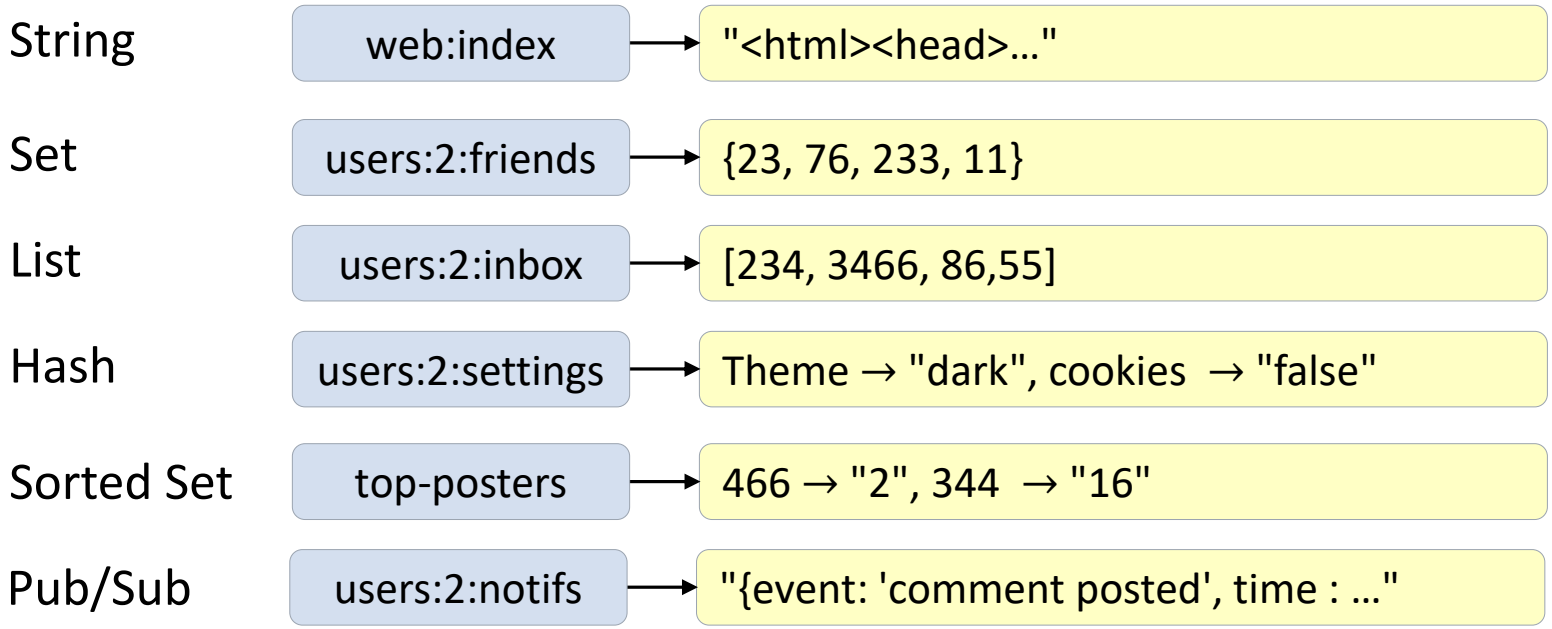
Master-Slave Replication

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

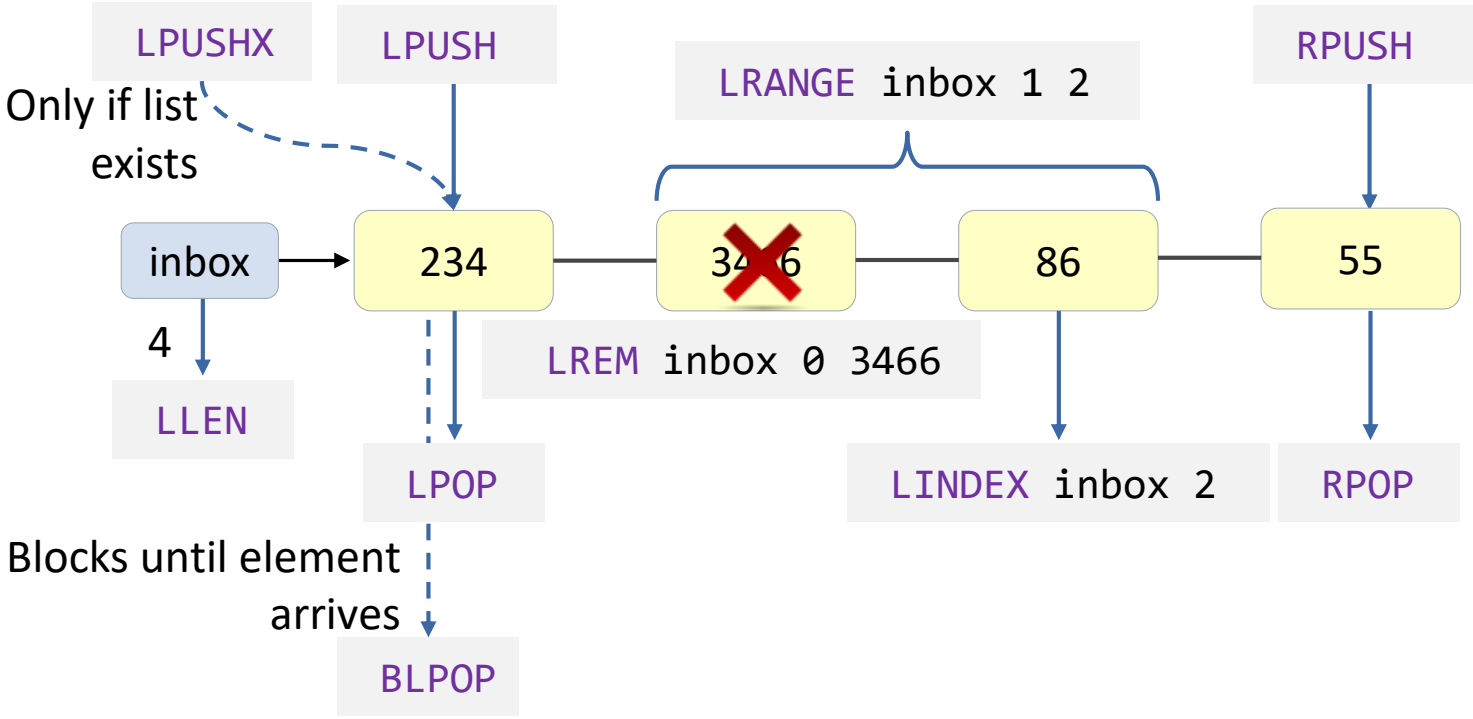


Data structures

- String, List, Set, Hash, Sorted Set

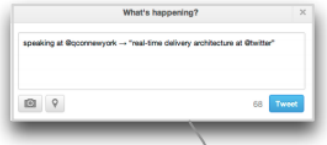


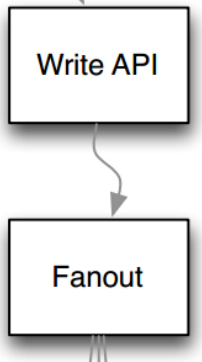
Example: List Data Structure

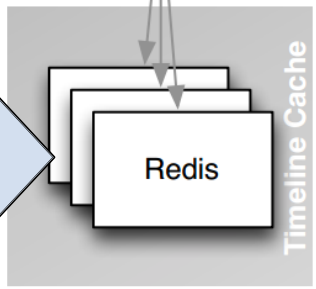


Example Redis Use-Case: Twitter

- 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

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

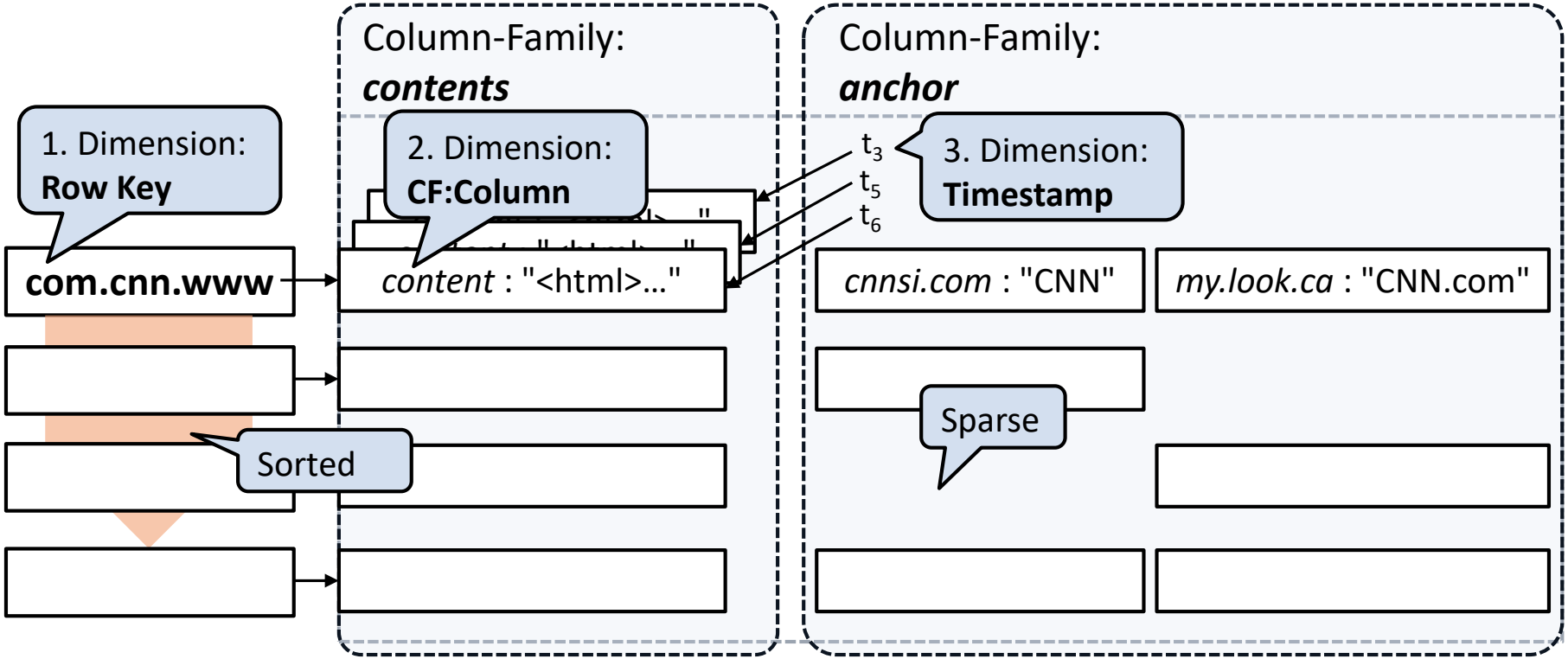
> A Bigtable is a sparse, distributed, persistent multidimensional sorted map.

- 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

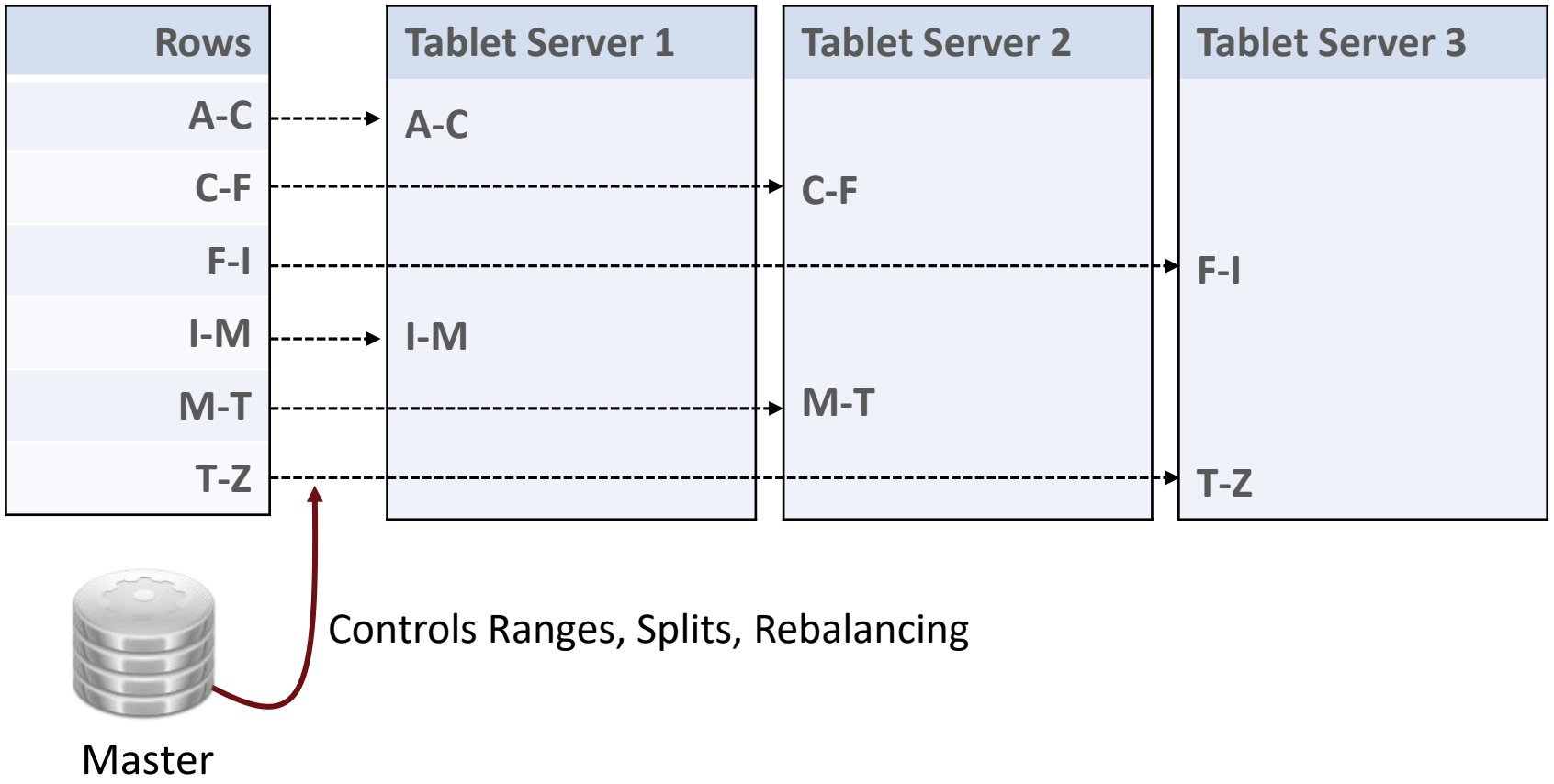
- Storage of crawled web-sites („Webtable“):



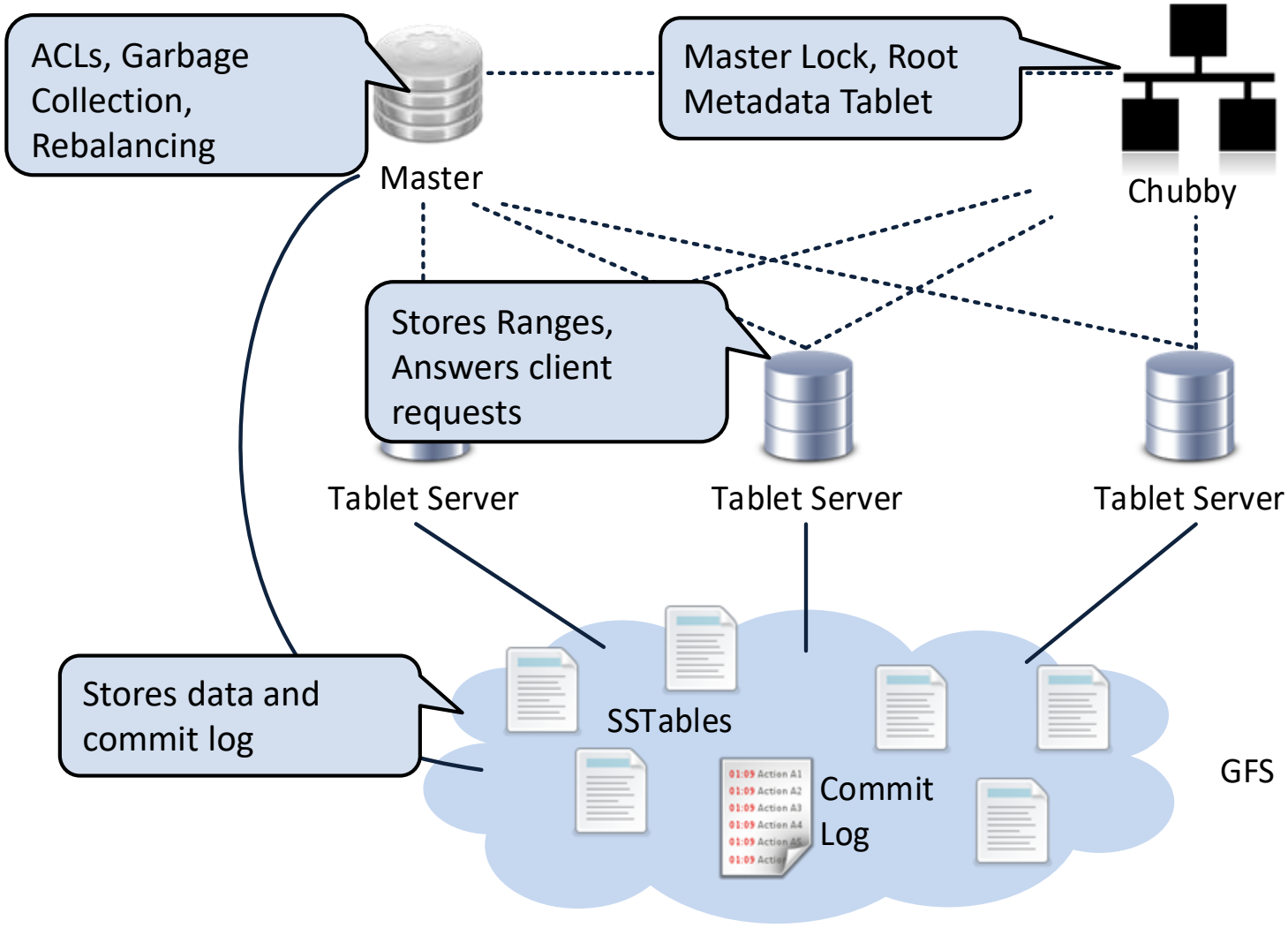
Range-based Sharding

BigTable Tablets

Tablet: Range partition of ordered records

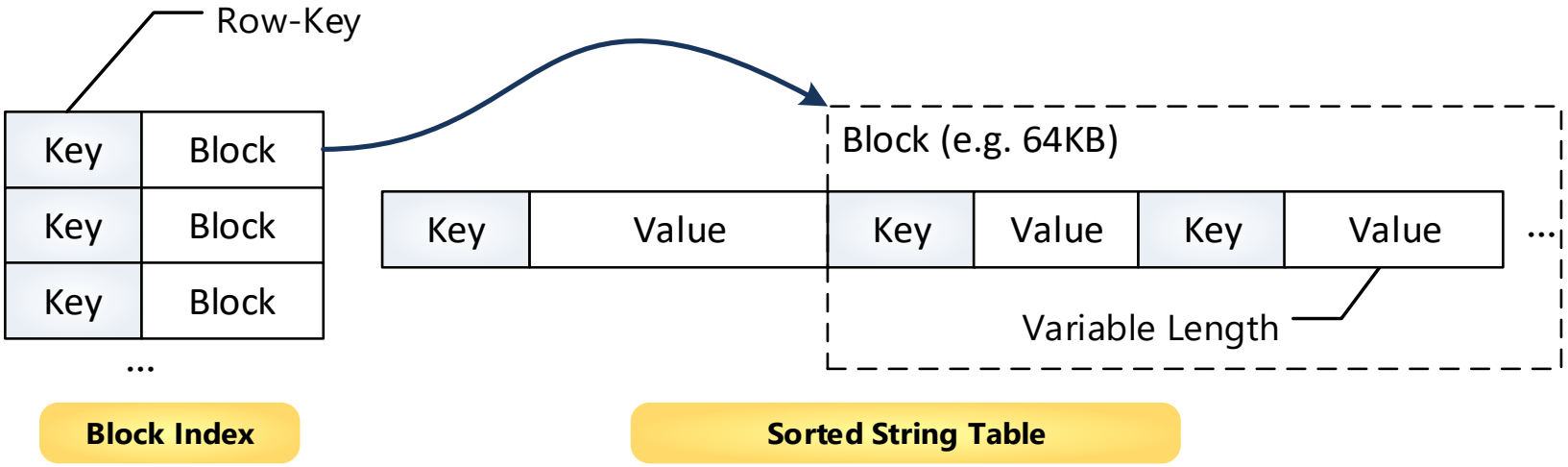


Architecture



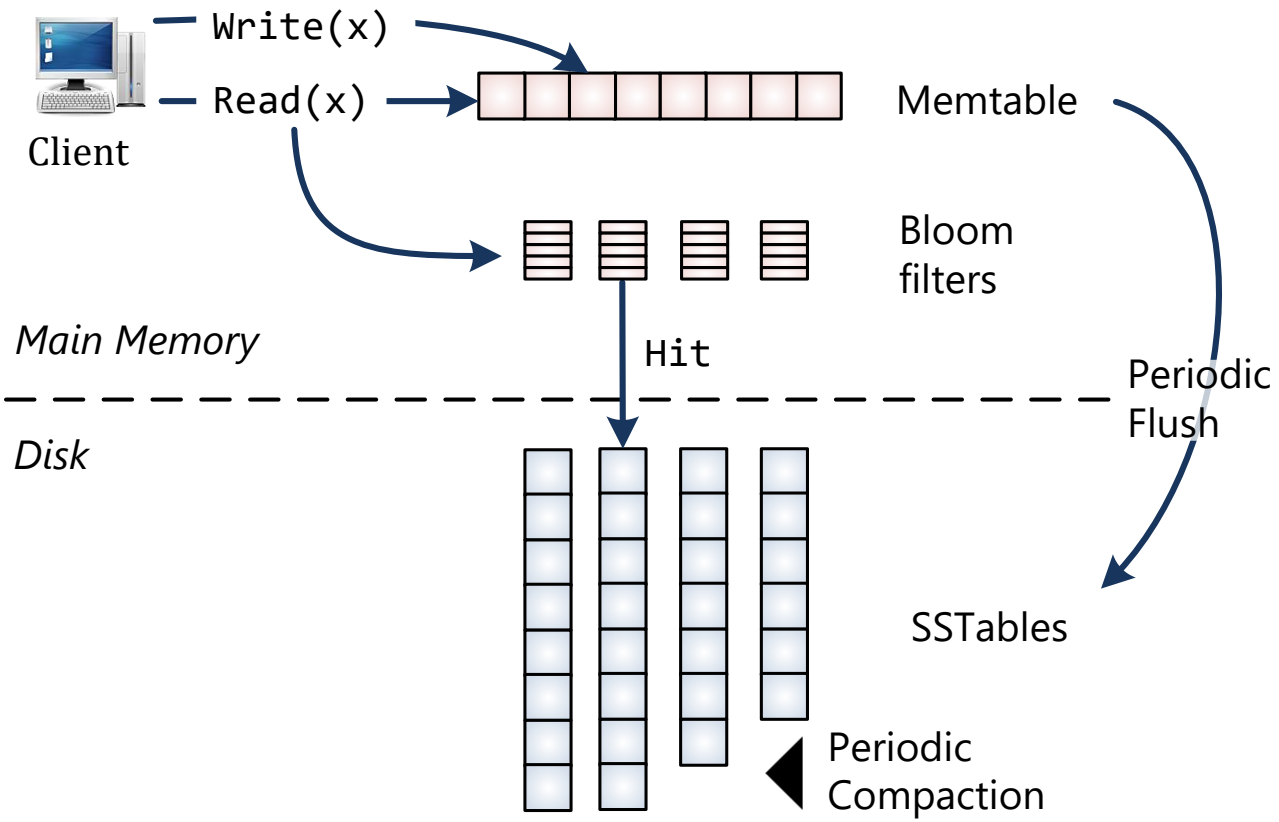
Storage: Sorted-String Tables

- **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



Storage: Optimization

- Writes: Buffered in-memory in **Memtable**
- SSTable disk access optimized by Bloom filters

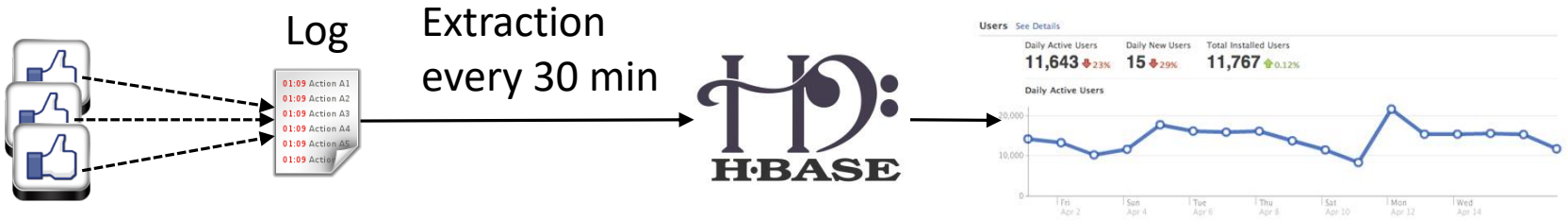


Apache HBase (CP)

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

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"

Classification: HBase

Techniques

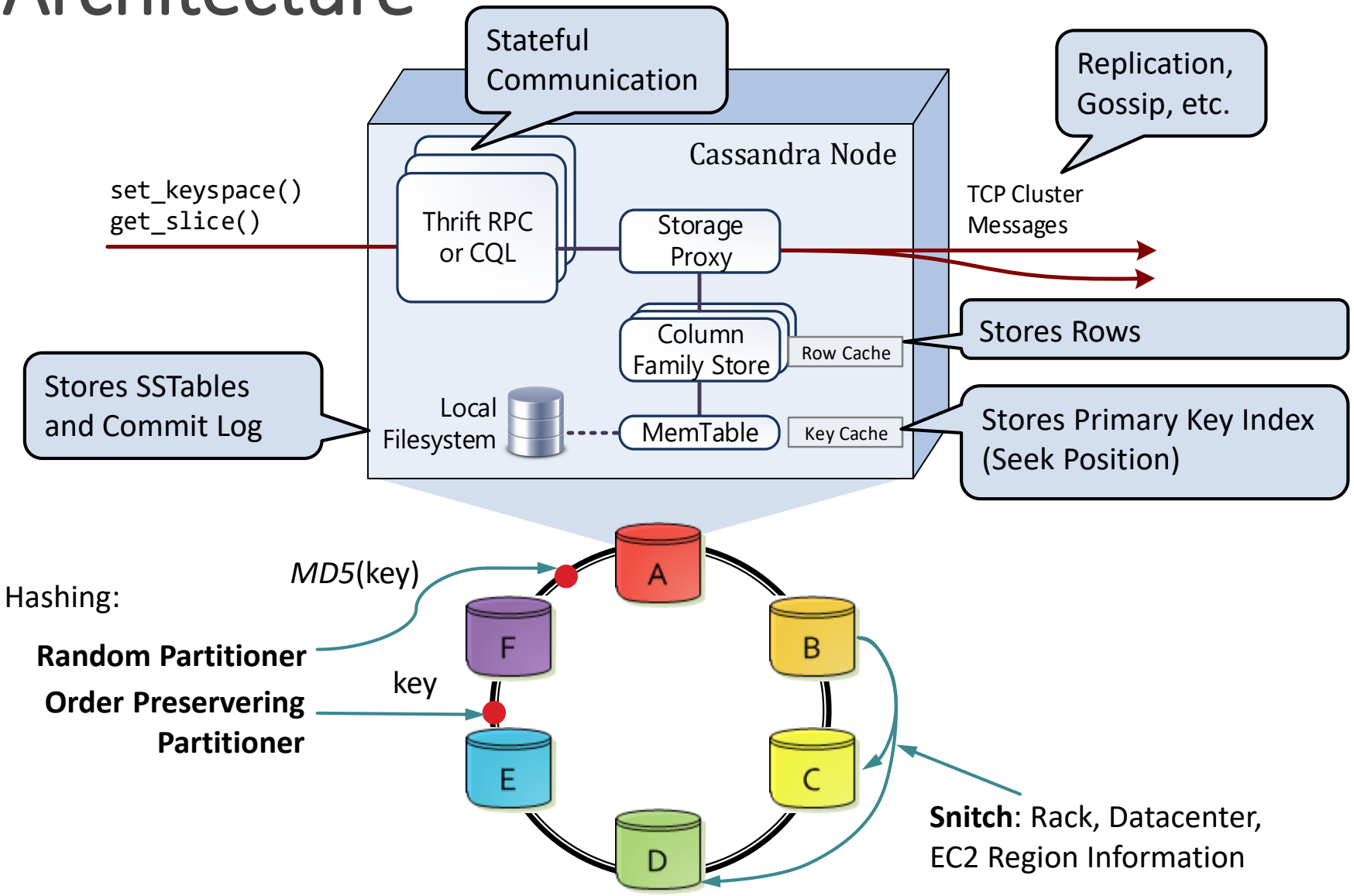
 **Sharding**	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
 **Replication**	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
 **Storage Management**	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
 **Query Processing**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

Architecture



Classification: Cassandra

Techniques

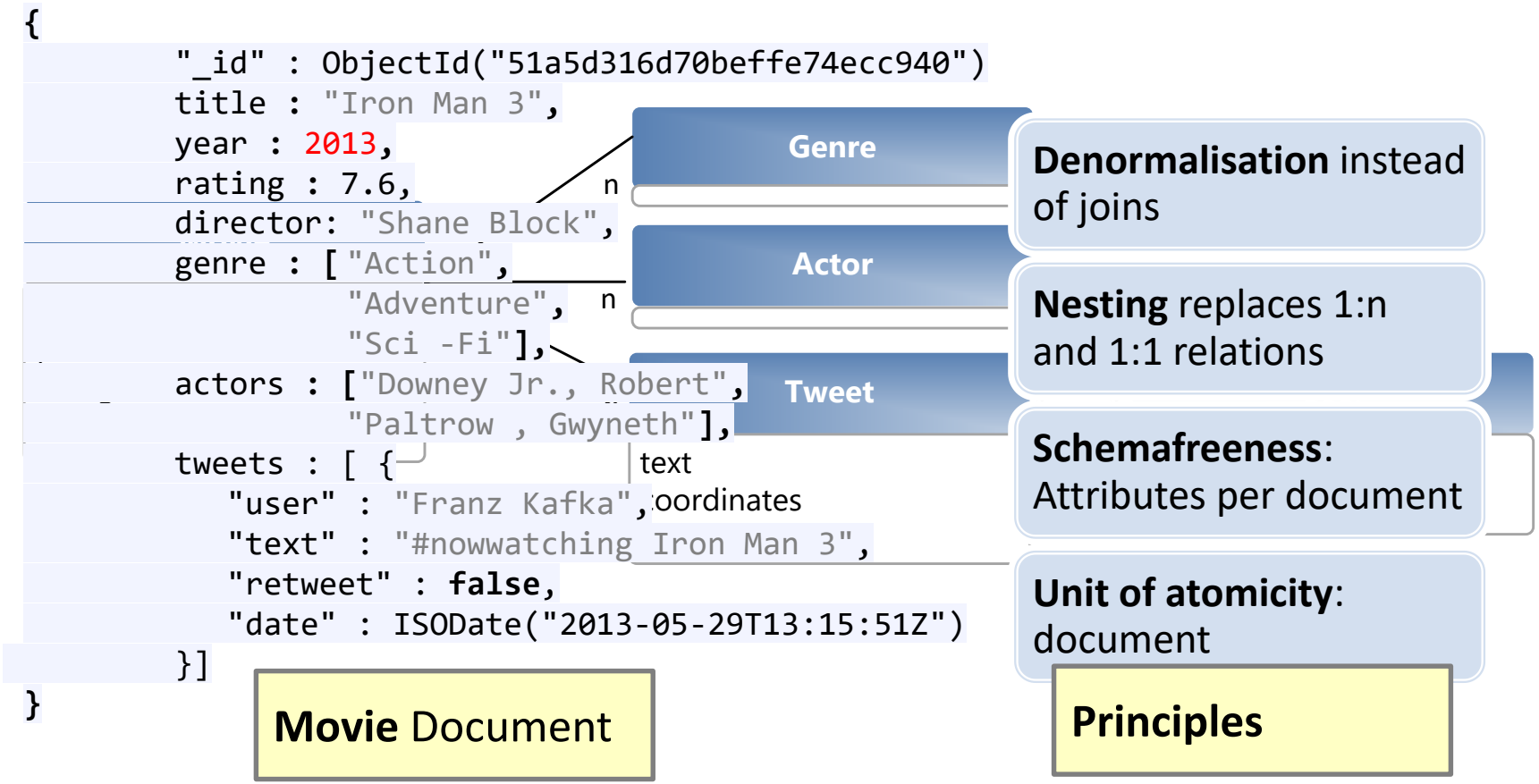
 **Sharding**	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
 **Replication**	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
 **Storage Management**	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
 **Query Processing**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

MongoDB (CP)

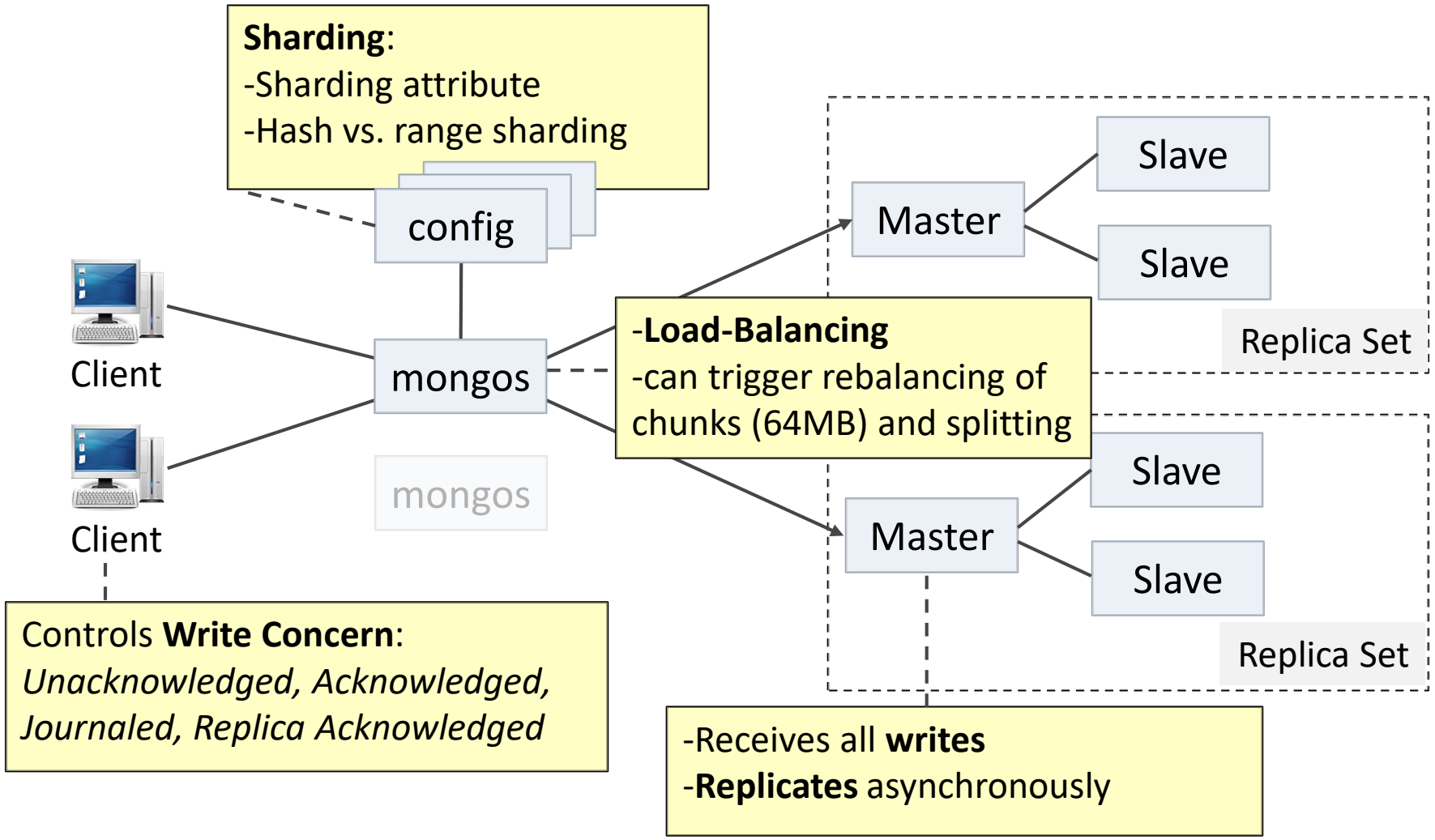
MongoDB
Model:
Document
License:
GNU AGPL 3.0
Written in:
C++

- From hu**mongo**us ≅ gigantic
- Schema-free document database with tunable consistency
- 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



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

Other Systems

Graph databases

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

Other Systems

Key-Value Stores

- **Aerospike** (SSD-optimized)
- **Voldemort** (Dynamo-style)
- **Memcache** (in-memory cache)
- **LevelDB** (embeddable, LSM-based)
- **RocksDB** (LevelDB-Fork with Transactions and Column Families)
- **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)

Other Systems

Document Stores

- **CouchDB** (Multi-Master, lazy synchronization)
- **CouchBase** (distributed Memcache, N1QL~SQL, MR-Views)
- **RavenDB** (single node, SI transactions)
- **RethinkDB** (distributed CP, MVCC, joins, aggregates, real-time)
- **MarkLogic** (XML, distributed 2PC-ACID)
- **ElasticSearch** (full-text search, scalable, unclear consistency)
- **Solr** (full-text search)
- **Azure DocumentDB** (cloud-only, ACID, WAS-based)

Other Systems

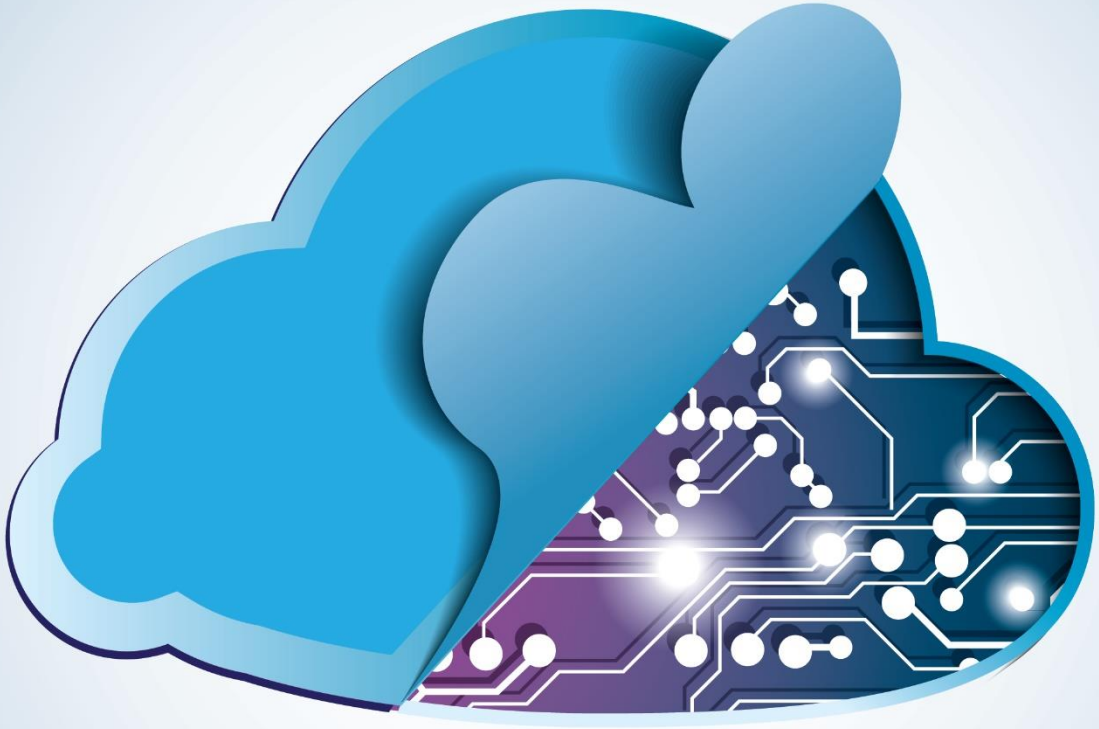
Wide-Column Stores

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

Other Systems

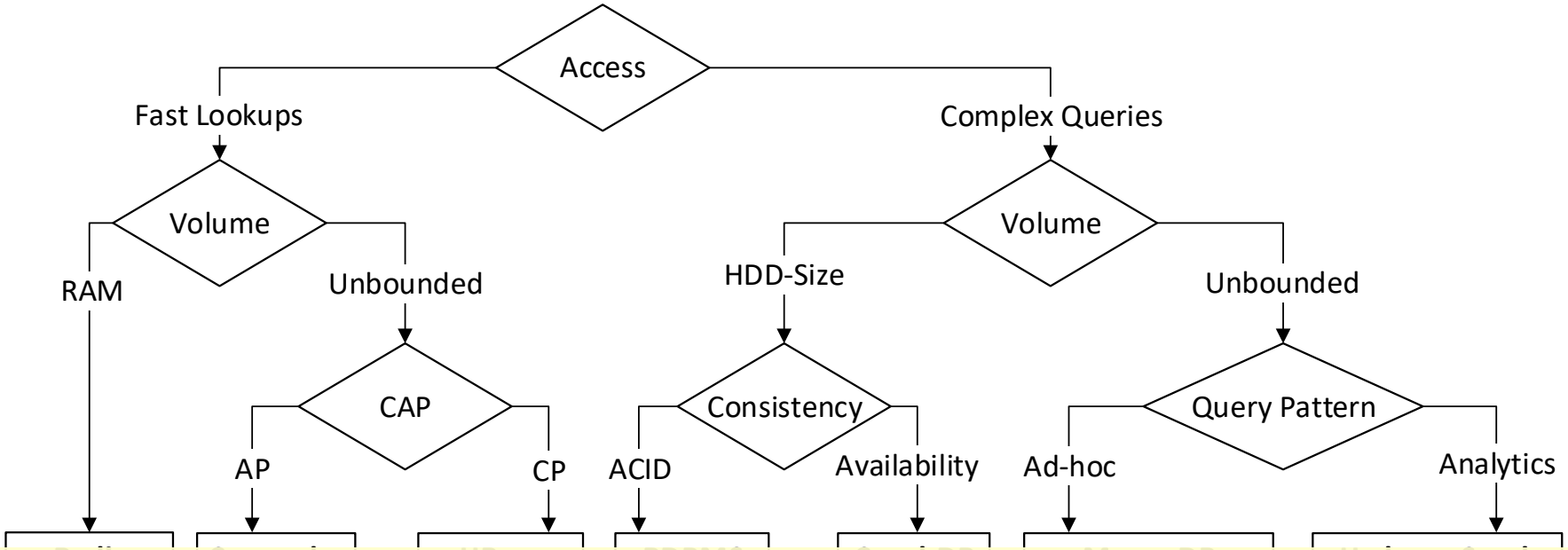
NewSQL Systems

- **CockroachDB** (Spanner-like, SQL, no joins, transactions)
- **Crate** (ElasticSearch-based, SQL, no transaction guarantees)
- **VoltDB** (HStore, ACID, in-memory, uses stored procedures)
- **Calvin** (log- & Paxos-based ACID transactions)
- **FaunaDB** (based on Calvin design, by Twitter engineers)
- **Google F1** (based on Spanner, SQL)
- **Google Cloud Spanner** (Improved F1 as a service)
- **Microsoft Cloud SQL Server** (distributed CP, MSSQL-comp.)
- **MySQL Cluster, Galera Cluster, Percona XtraDB Cluster** (distributed storage engine for MySQL)



How can the choices for an appro-priate system be narrowed down?

NoSQL Decision Tree



Purpose:

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

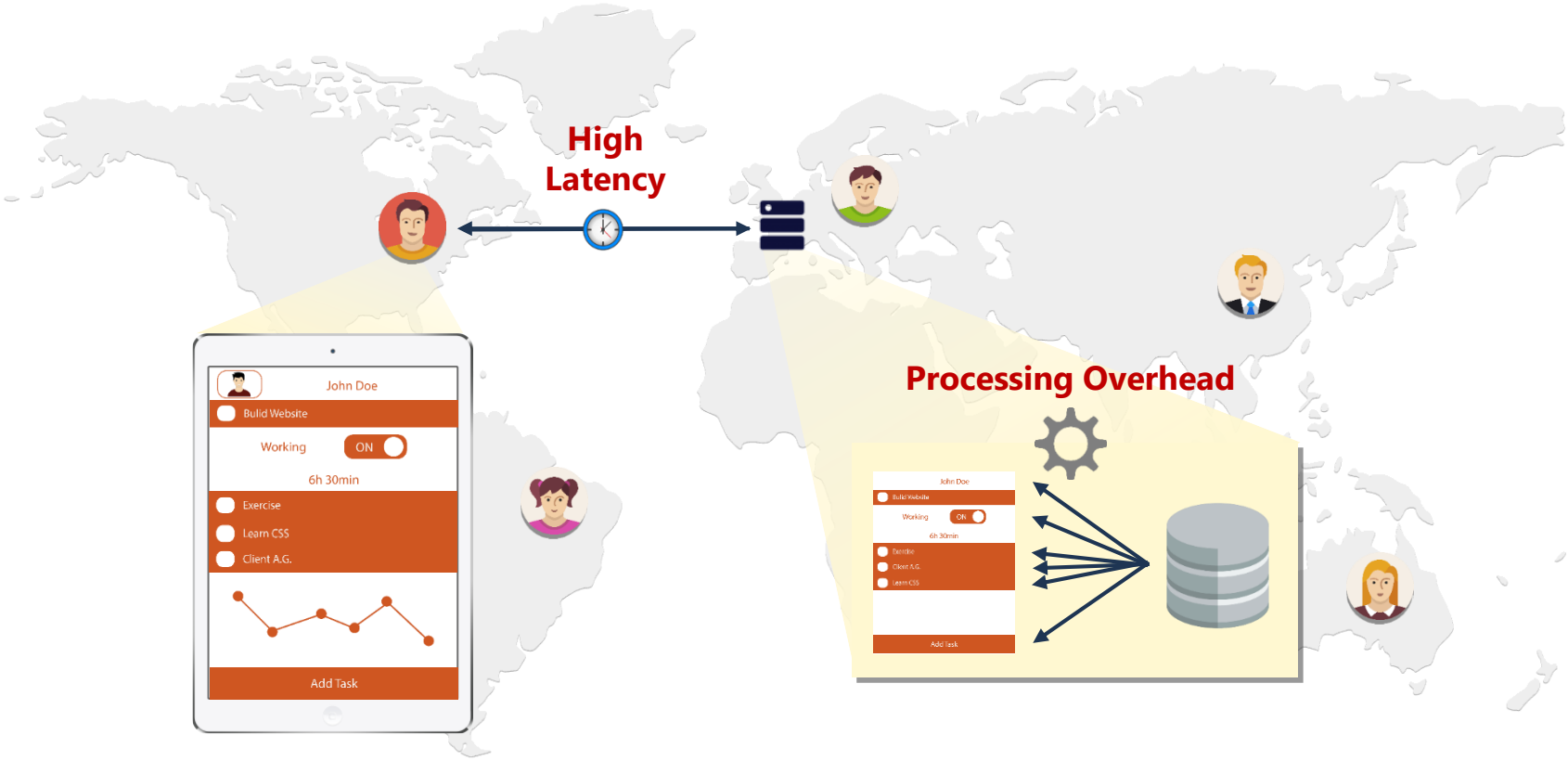
Database Vendors/Researchers: clear communication and design of system trade-offs

OUTLOOK

Baqend Research at the University of Hamburg

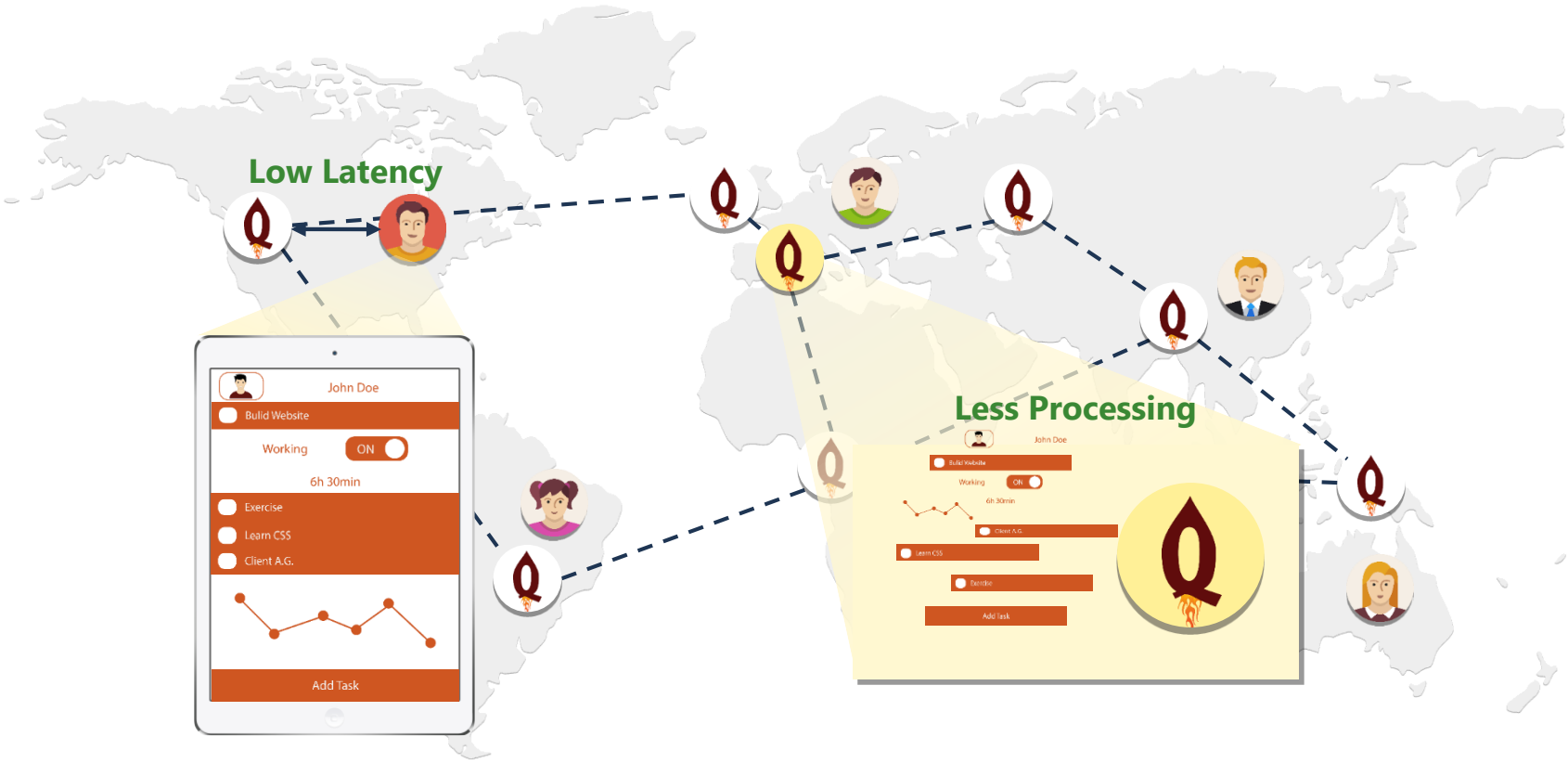
Problem: Slow Websites

Two Bottlenecks: Latency and Processing



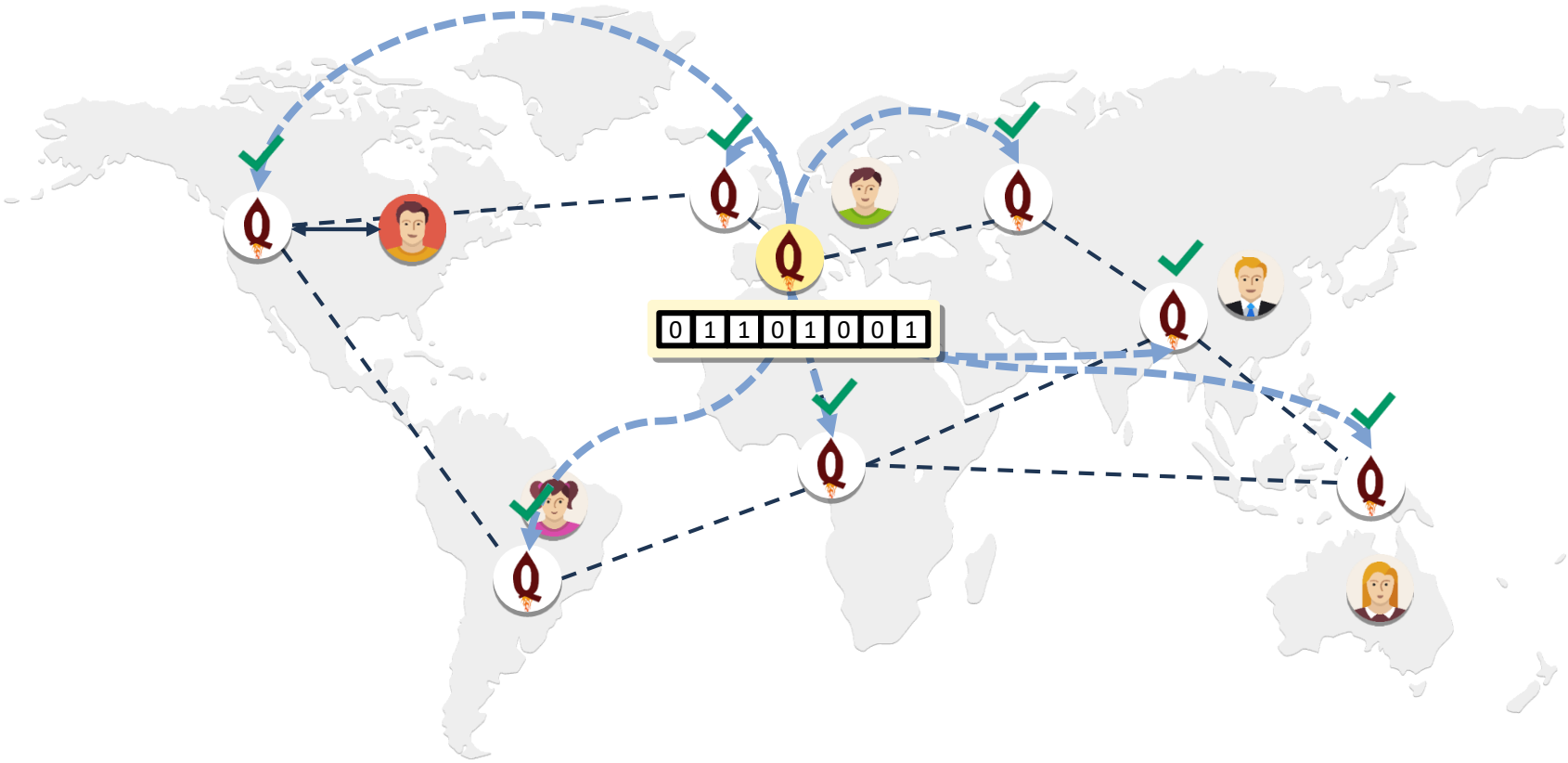
Solution: Global Caching

Fresh Data From Distributed Web Caches



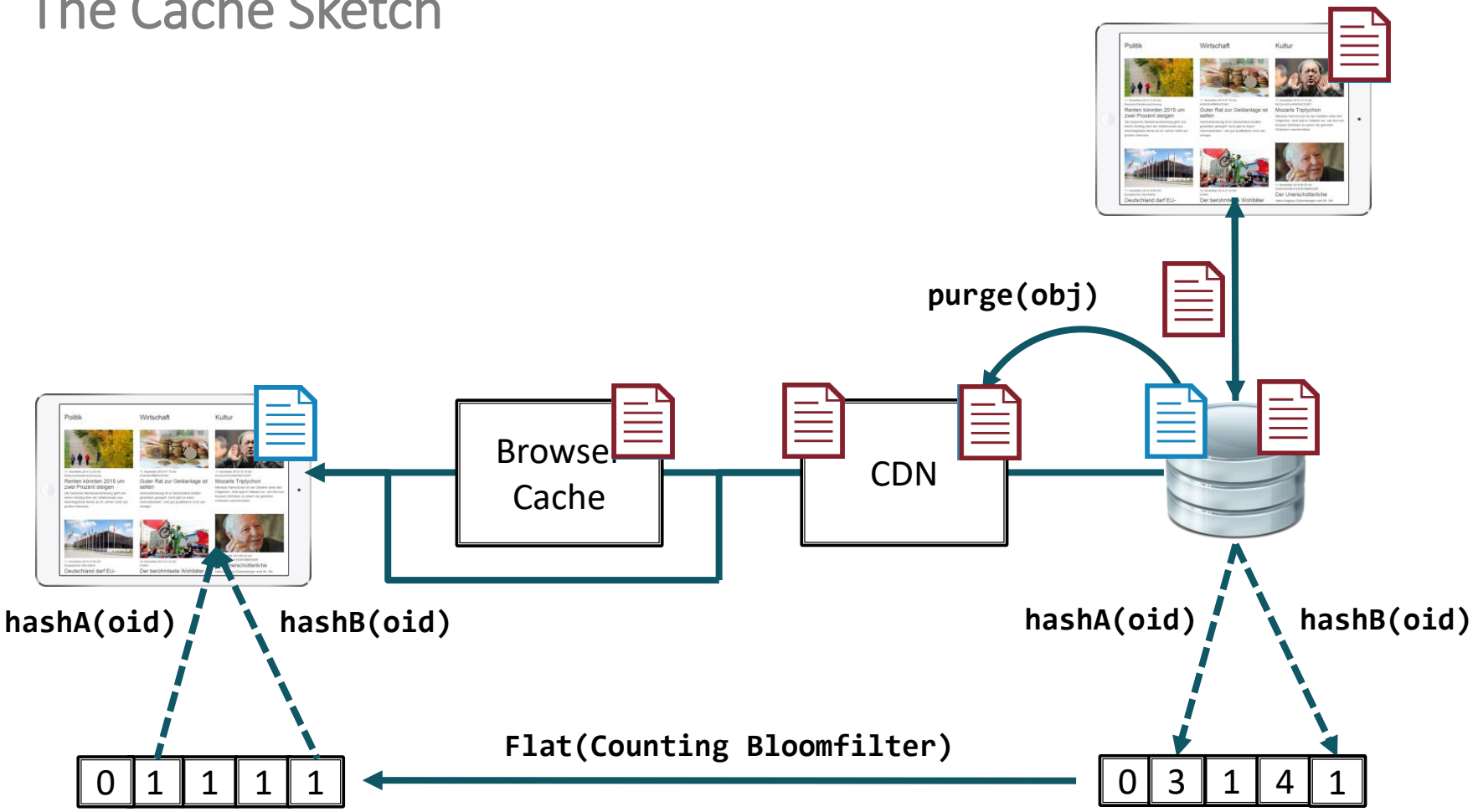
New Caching Algorithms

Solve Consistency Problem



Consistent Web Caching

The Cache Sketch



Baqend

Try It Out!

Platform

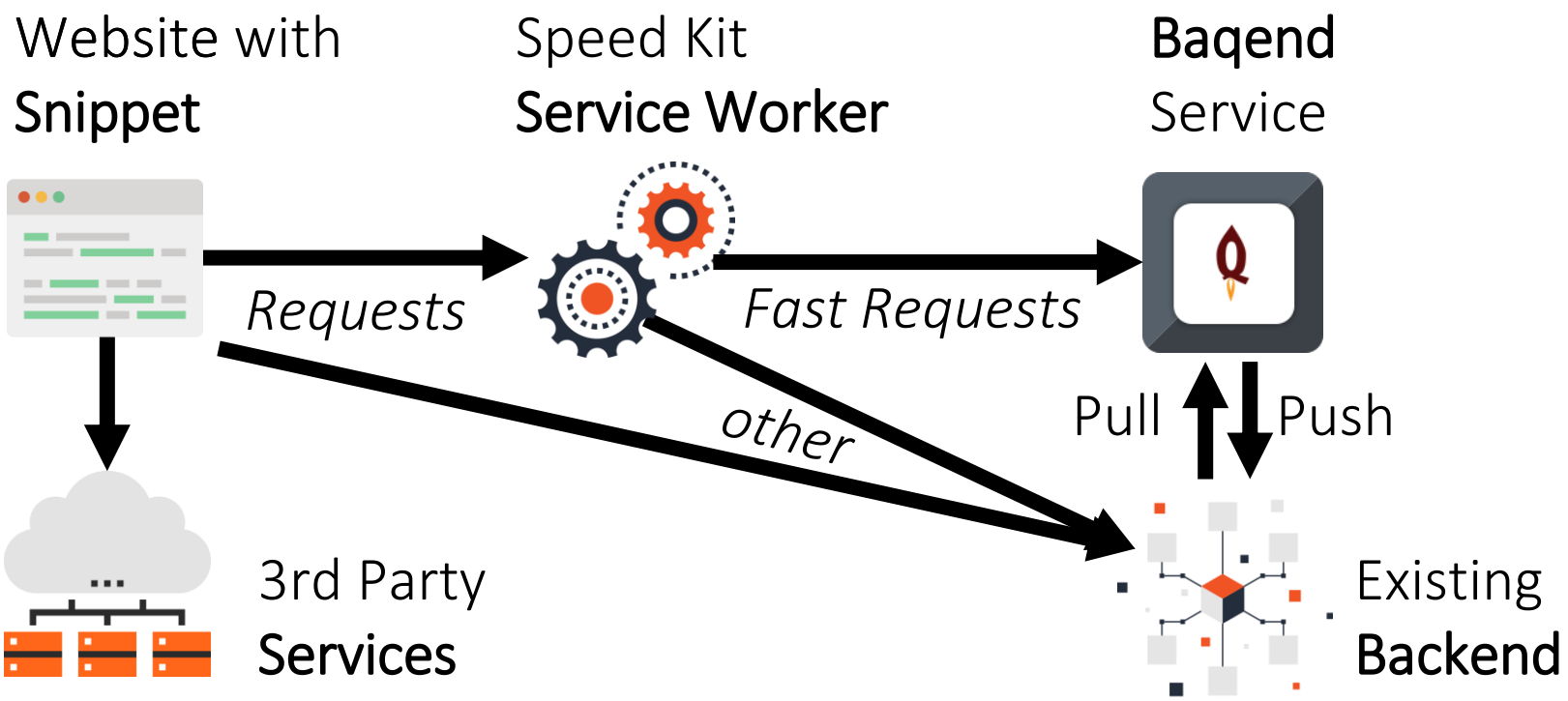
- Platform for building (Progressive) **Web Apps**
- **15x** Performance Edge
- Faster **Development**

Speed Kit

- Turns Existing Sites into **PWAs**
- **50-300% Faster** Loads
- **Offline** Mode

Speed Kit

Baqend Caching for Existing Websites



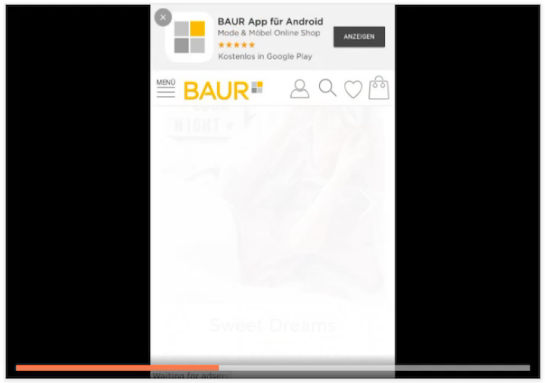
Speed Kit
Measure Your Page Speed!

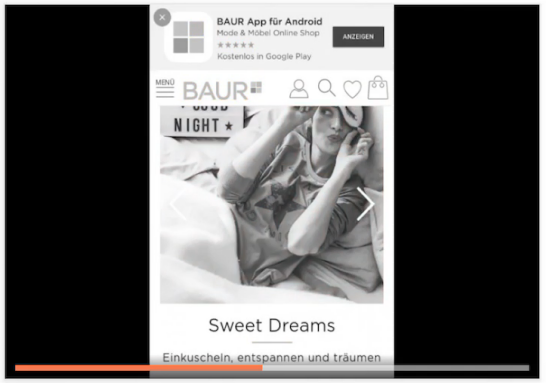
https://test.speed-kit.com/

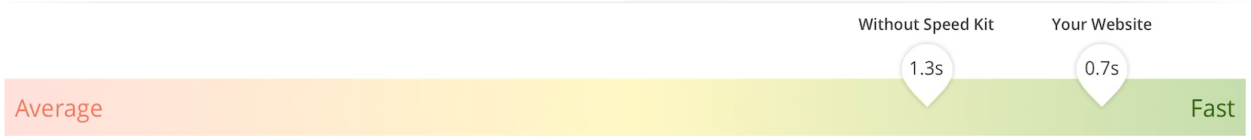
https://www.baur.de/

You are using Speed Kit 1.12.1

Without Speed Kit	1.9x Faster	Your Website (Speed Kit 1.12.1)
1283 ms		662 ms







Open Bachelor and Master Topics:

research.baqend.com

Offene Masterarbeiten

Providing Low Latency for PostgreSQL through the Orestes Caching Middleware

Orestes ist eine datenbankunabhängige Cloud-Middleware für (NoSQL) Datenbanken, die durch Caching extrem geringe Latenz erzielt. Orestes ist so konzipiert, dass durch die Implementierung bestimmter Schnittstellen (z.B. CRUD, Queries, Schemaänderungen) beliebige Datenbanken angebunden und durch eine *Unified REST API* einheitlich angesprochen werden können. Die Datenbank wird dabei automatisch angereichert um Fähigkeiten, die ihr sonst fehlen, u.a. gloabl verteiltes Web-Caching, semi-strukturierte Schemata, Zugriffskontrolle durch ACLs, etc. Ziel der Arbeit ist die Anbindung eines der Systeme DynamoDB, Cassandra, HBase oder PostgreSQL mit einer Studie darüber, welche Eigenschaften sich erzielen lassen und welche aufgrund der Datenbankarchitektur und seiner Schnittstellen nicht erreichbar sind.

Ziele:

- Analyse der Datenbankeigenschaften aus Sicht der verschiedenen Orestes-Schnittstellen für CRUD, Queries, Partial Updates etc.
- Problembeschreibung: Identifikation der fehlenden Fähigkeiten der Datenbank und Studie, welche sich durch Orestes automatisch erzielen lassen.
- **Konsistenzmodell**: Welche Auswirkung haben die Konistenzgarantien der Datenbank für Clients und wie spielen sie mit Caching zusammen?

Open Student Positions at Baqend:

baqend.com/jobs.html

Customer Success Manager

Help customers in leveraging Baqend technology.

Student Position: Download PDF

UX/Web Designer

Create user interfaces people love.

Full Time: Download PDF
Internship: Download PDF

JavaScript Engineer

High-performance, modern JavaScript development.

Full Time: Download PDF
Internship: Download PDF

DevOps Engineer

Build scalable and reliable cloud services.

Full Time: Download PDF
Internship: Download PDF

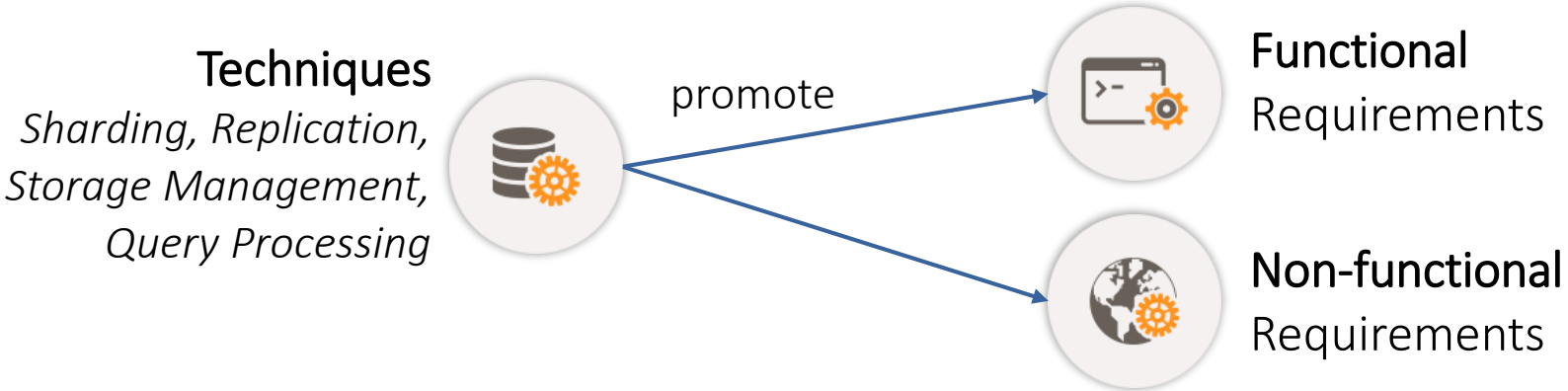
Growth Hacker

Versatile marketing for growing our startup.

Full Time: Download PDF
Internship: Download PDF

Summary (1/2)

- 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**

Summary (2/2)

- Variety of different NoSQL systems:
  - **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