NoSQL Database Systems:

A Survey and Decision Guidance

Felix Gessert, Wolfram Wingerath,
Steffen Friedrich, Norbert Ritter
gessert@informatik.uni-hamburg.de

June 28, SummerSOC 2016

Extended version of this talk (ICDE 2016 tutorial): slideshare.net/felixgessert

Outline

NoSQL Foundations and Motivation

The NoSQL Toolbox: Common Techniques

NoSQL Systems

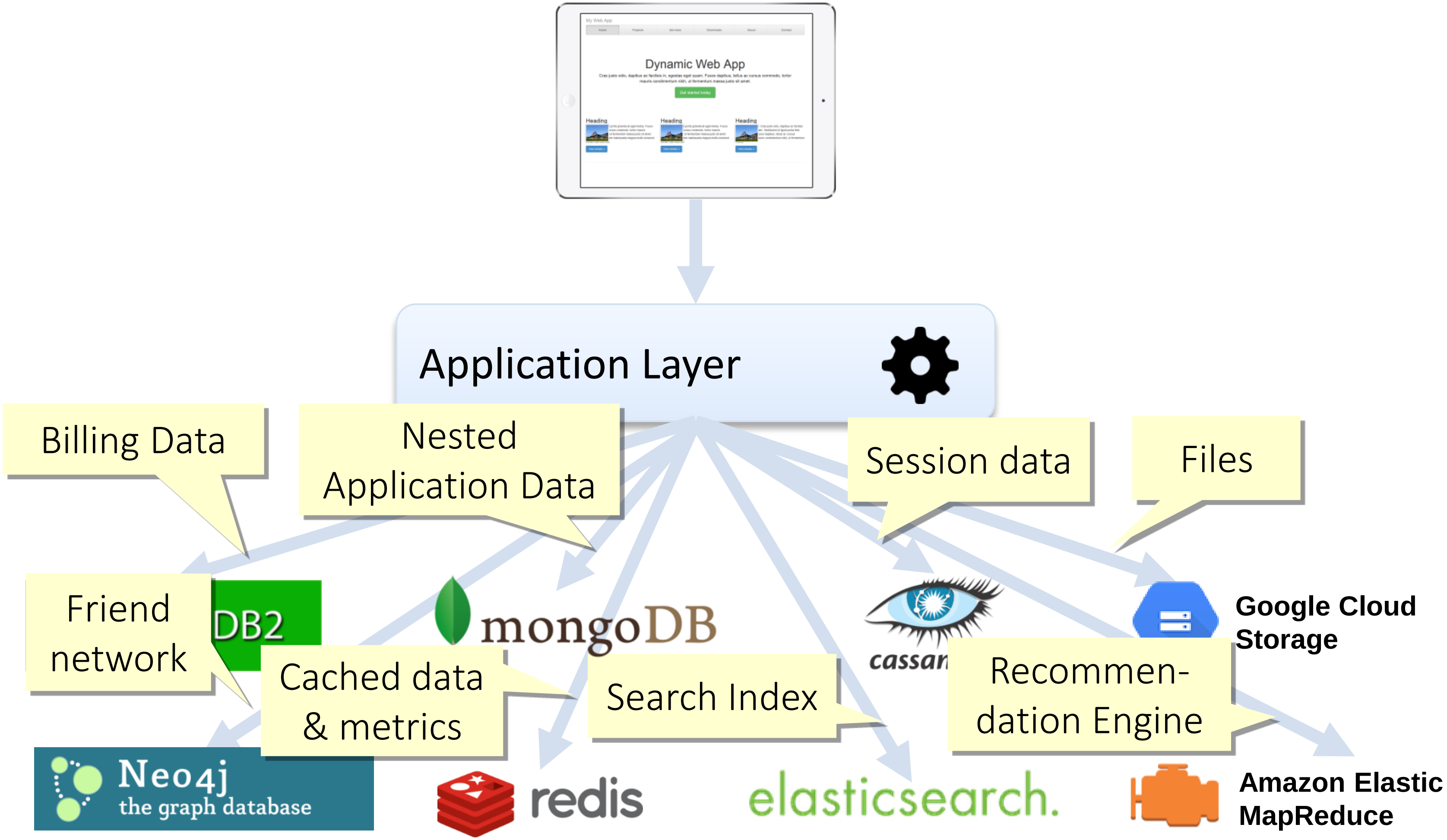
Decision Guidance: NoSQL Decision Tree

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

Introduction: Diversity of NoSQL data stores

How to choose a database system?

Many Potential Candidates



Application Layer

Billing Data

Nested Application Data

Session data

Files

Friend network

Cached data & metrics

Search Index

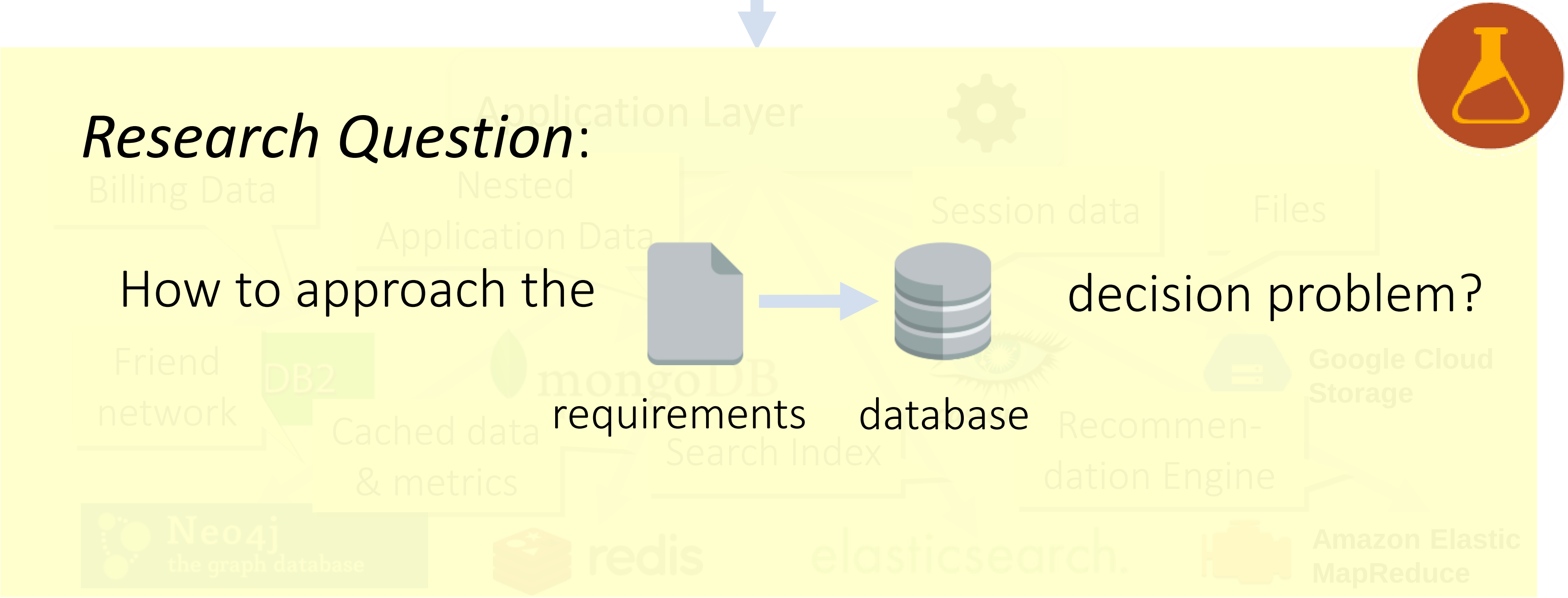
Recommendation Engine

How to choose a database system?

Many Potential Candidates

Dynamic Web App

Cras justo odio, dapibus ac facilisis in, egestas eget quam. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet.



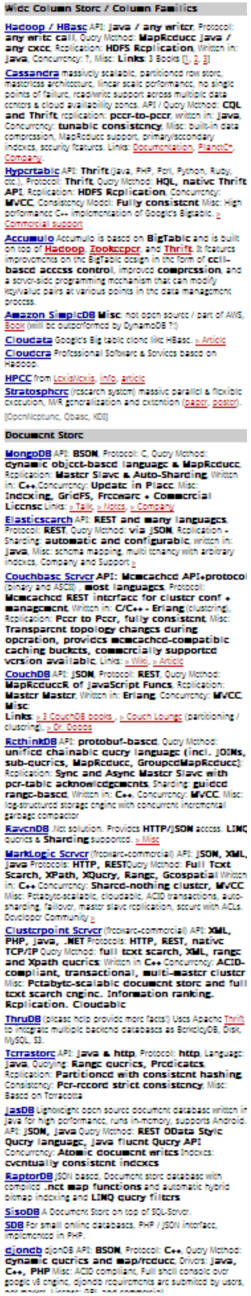
Research Question:

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, 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**

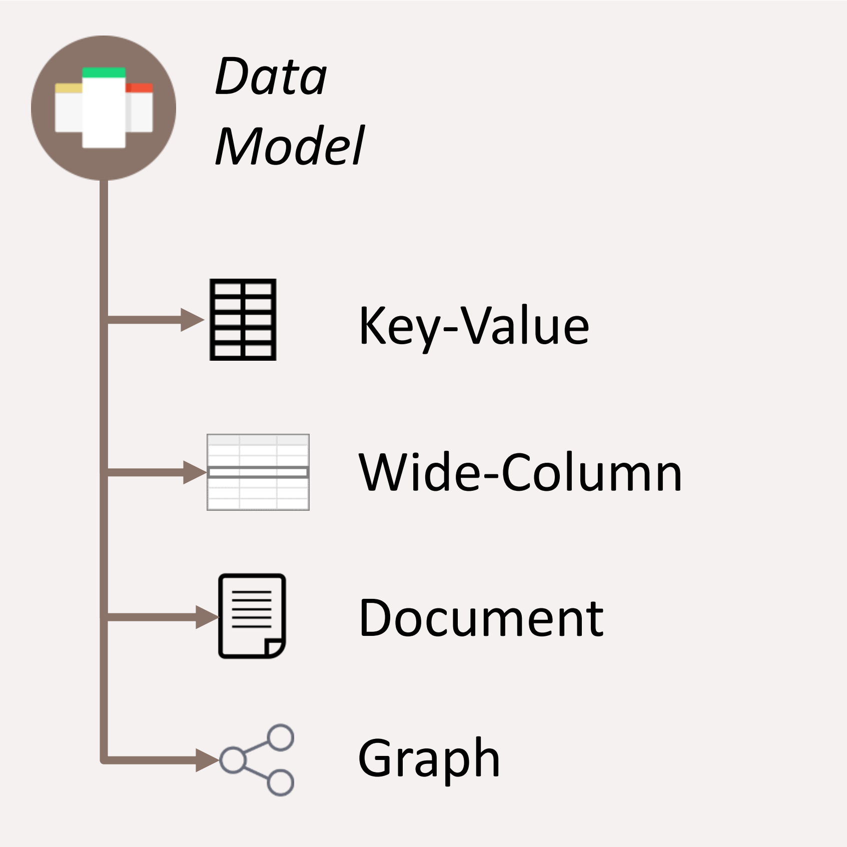
SisoDB A Document Store on top of SQL-Server.

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, not market. License: GPL and commercial

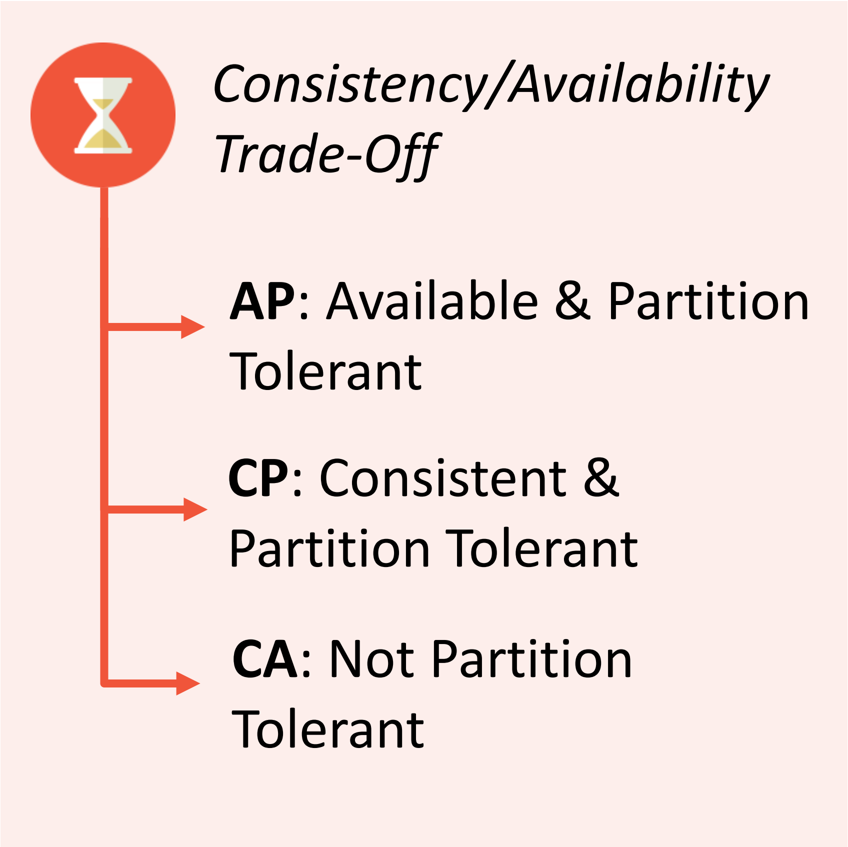
NoSQL System Classification

- Two common criteria:



Data Model

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

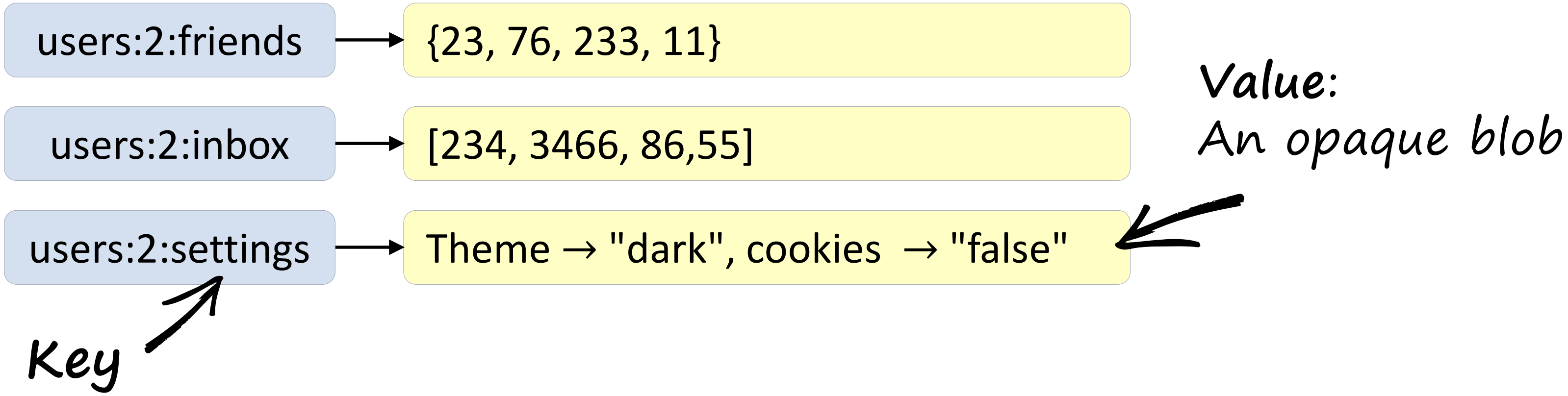


Consistency/Availability Trade-Off

- **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)



users:2:friends → {23, 76, 233, 11}

users:2:inbox → [234, 3466, 86,55]

users:2:settings → Theme → "dark", cookies → "false"

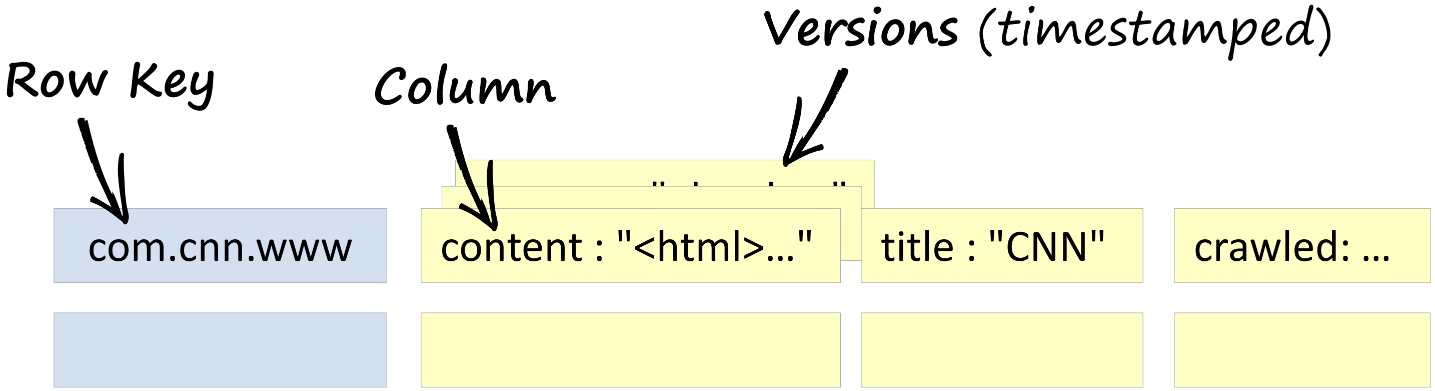
Value:
An opaque blob

Key

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

Wide-Column Stores

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



com.cnn.www

content : "<html>…"

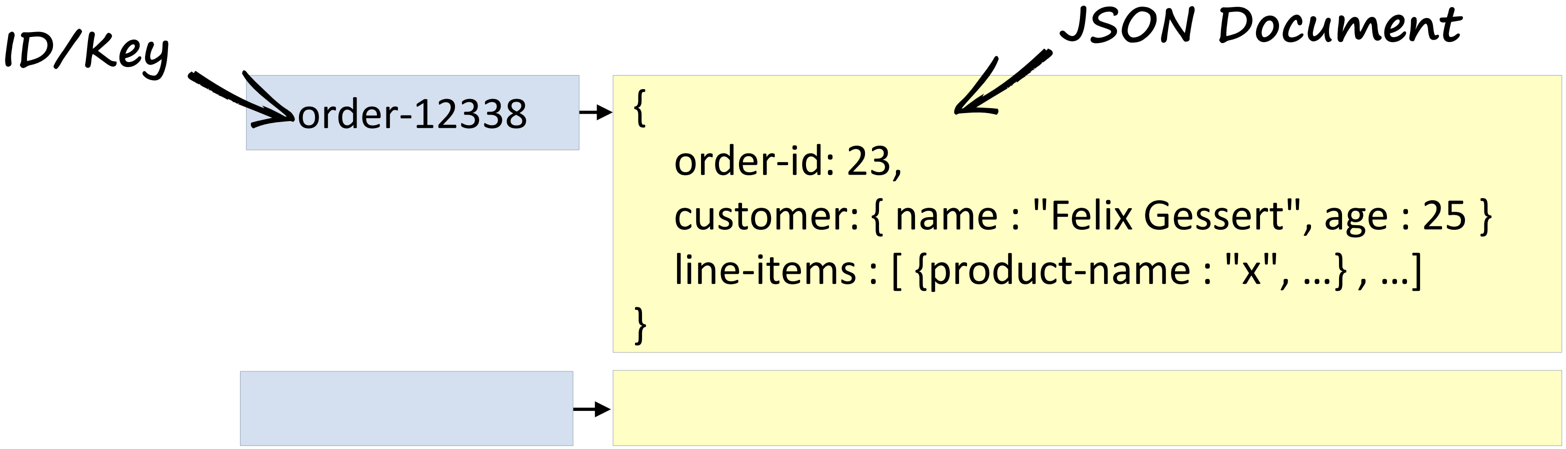
title : "CNN"

crawled: …

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

Document Stores

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



ID/Key

JSON Document

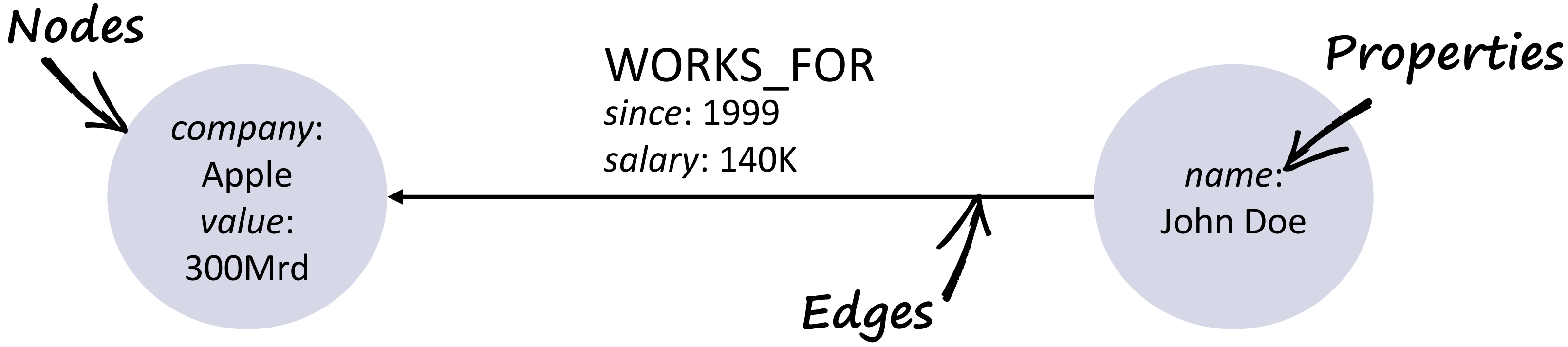
order-12338

```
{
   order-id: 23,
   customer: { name : "Felix Gessert", age : 25 }
   line-items : [ {product-name : "x", …} , …]
}
```

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

Graph Databases

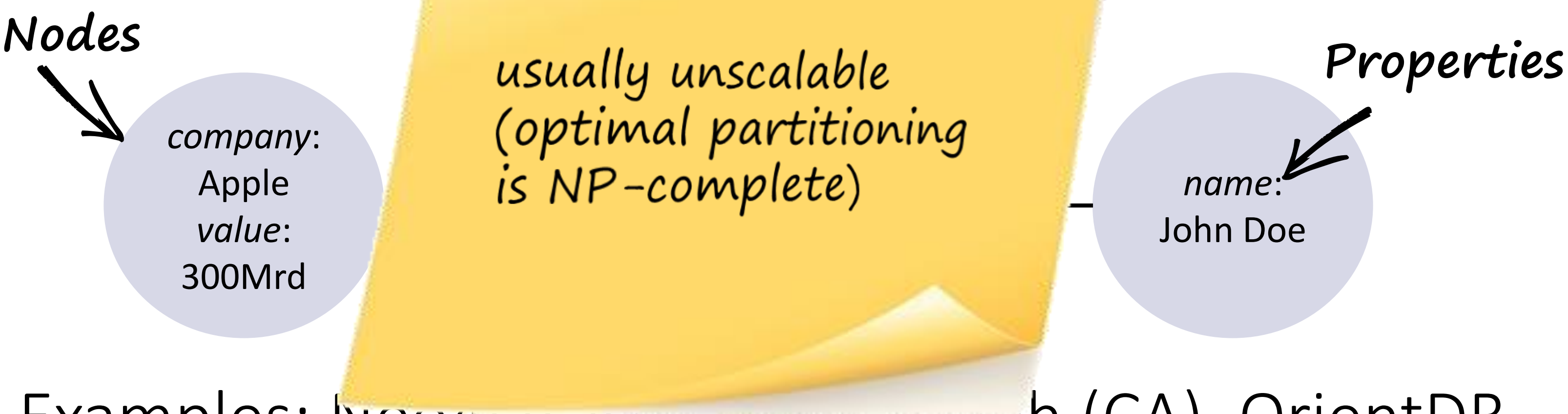
- **Data model:** G = (V, E): Graph-Property Modell
- **Interface**: Traversal algorithms, querys, transactions



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

Graph Databases

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

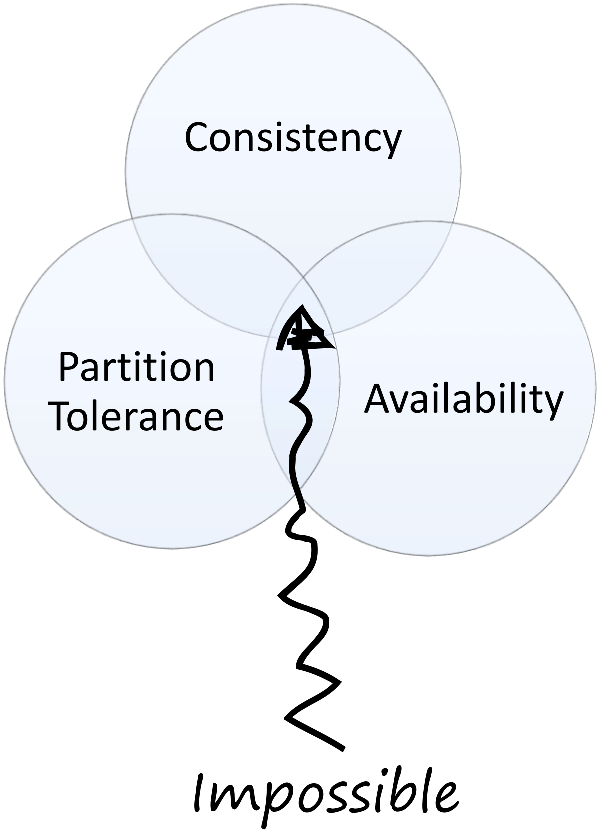
Object-Oriented Databases:

- 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

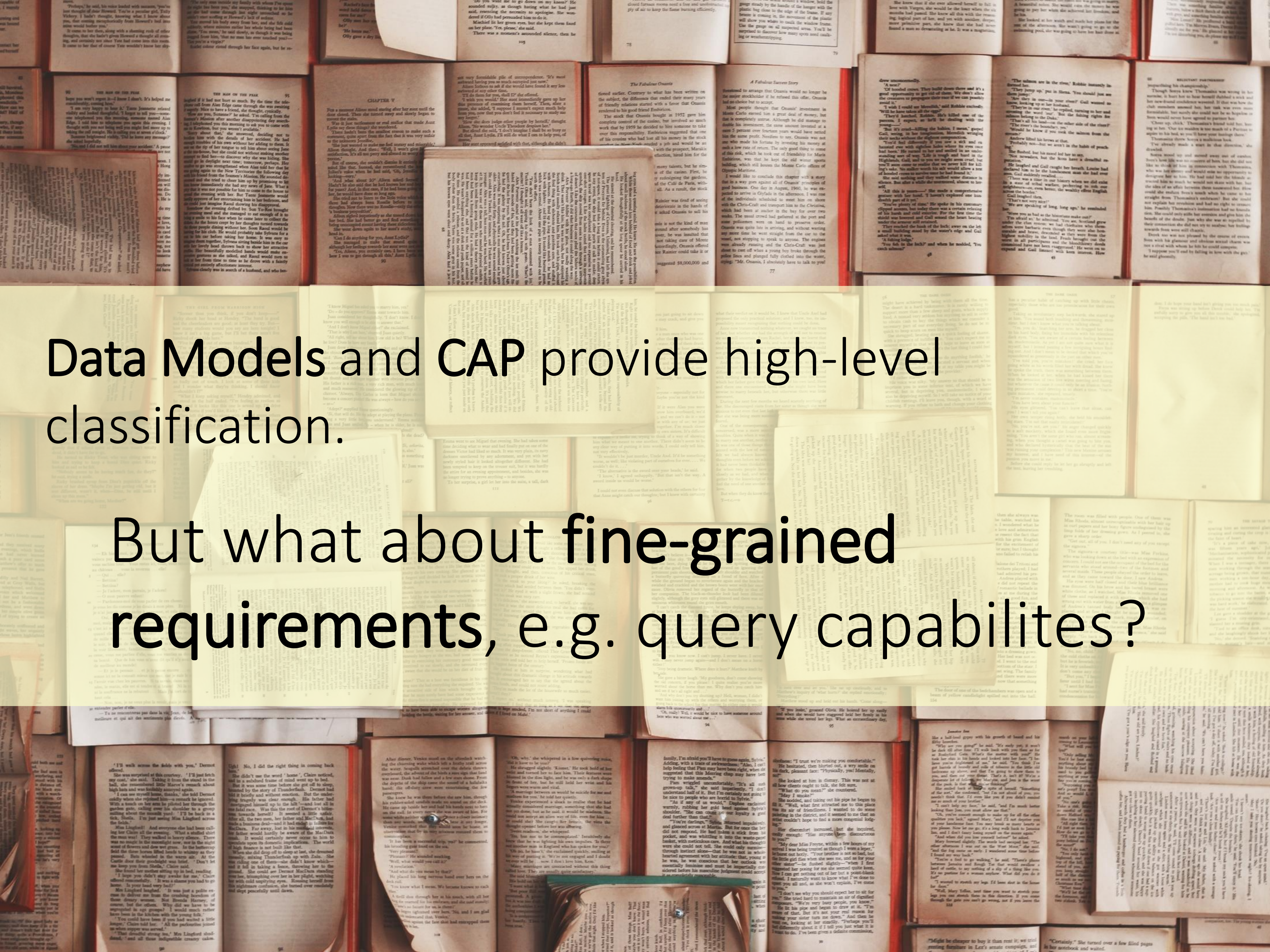


Only 2 out of 3 properties are achievable at a time:

- **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



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

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

Outline

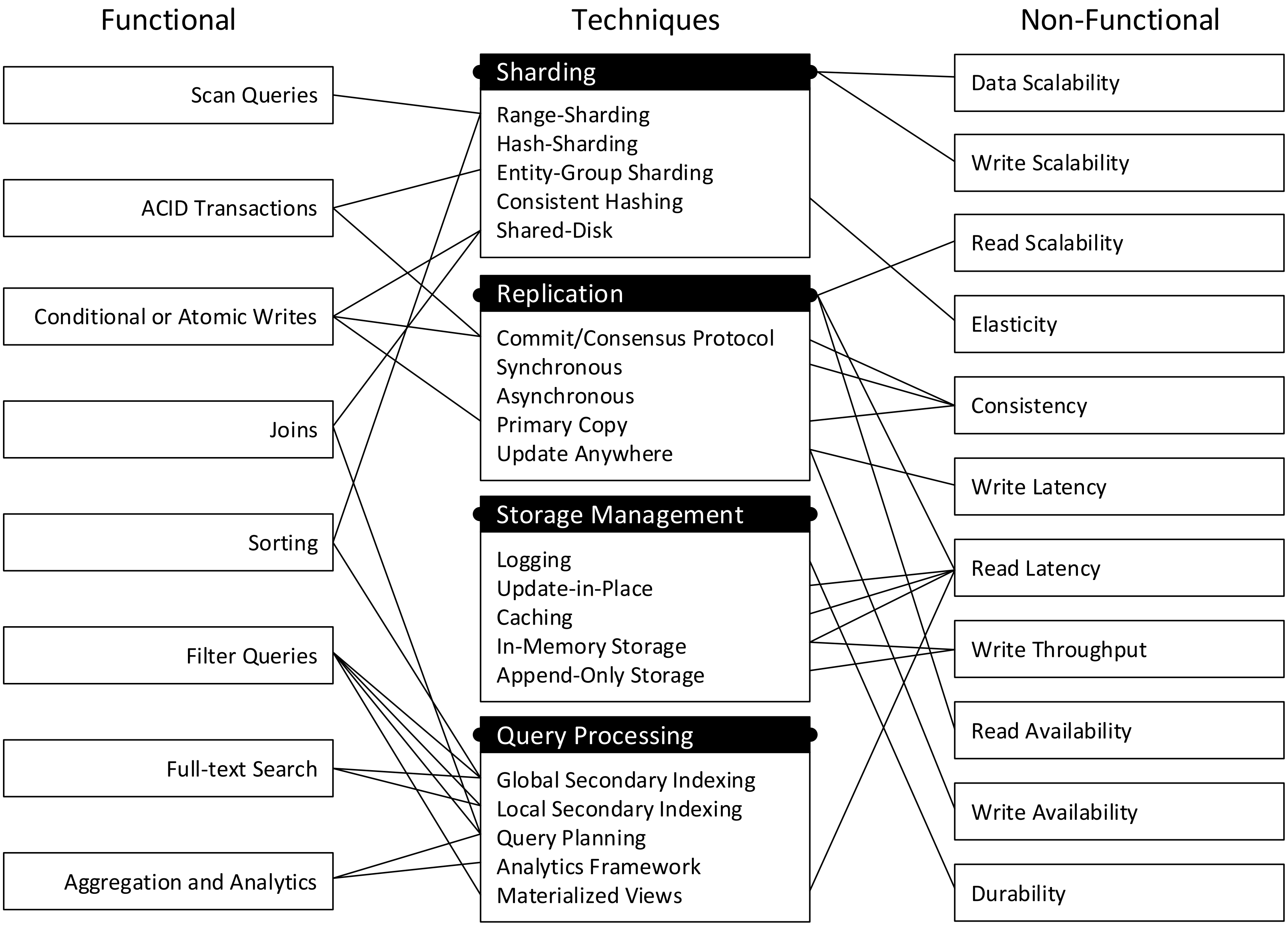
NoSQL Foundations and Motivation

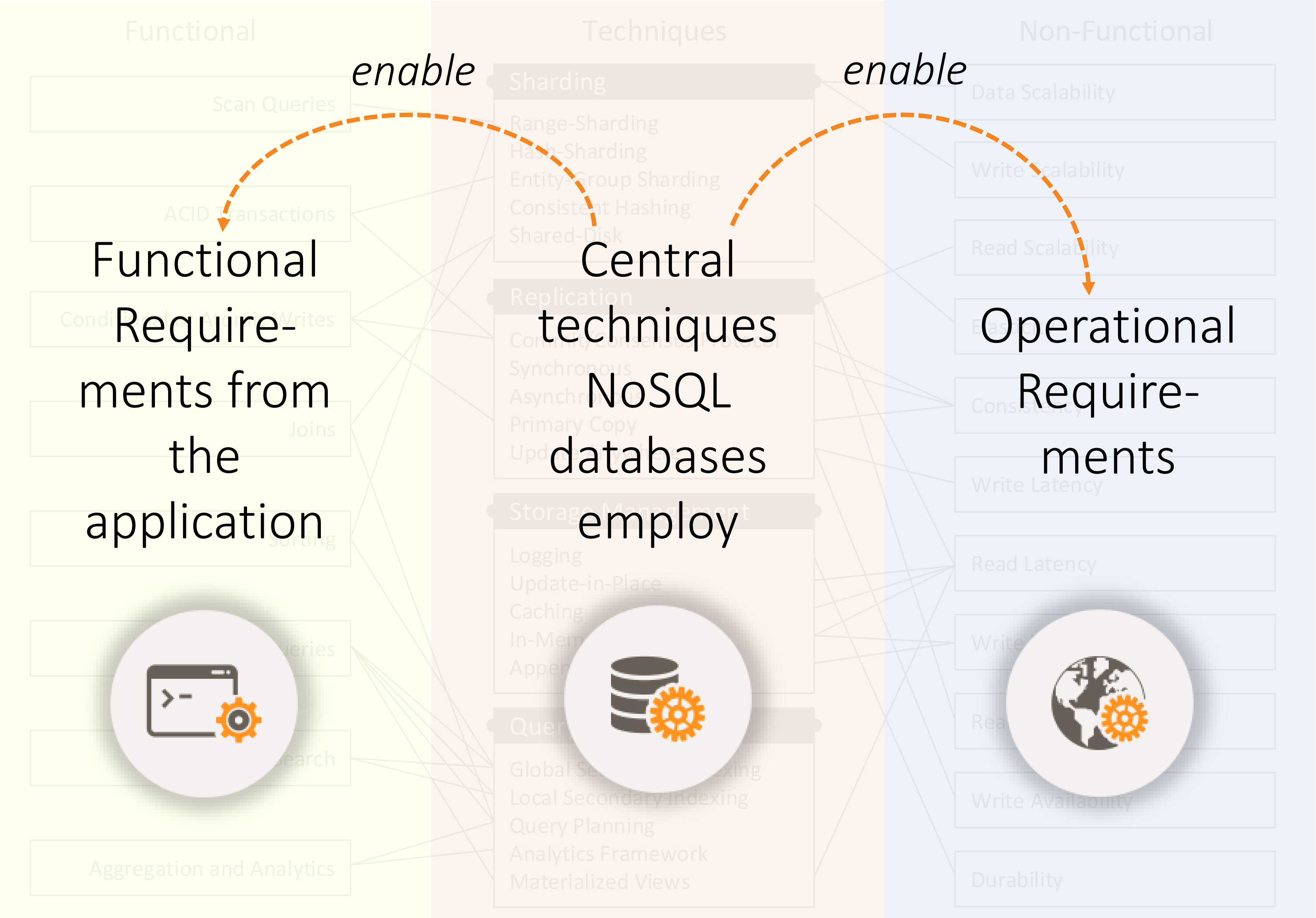
The NoSQL Toolbox: Common Techniques

NoSQL Systems

Decision Guidance: NoSQL Decision Tree

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


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



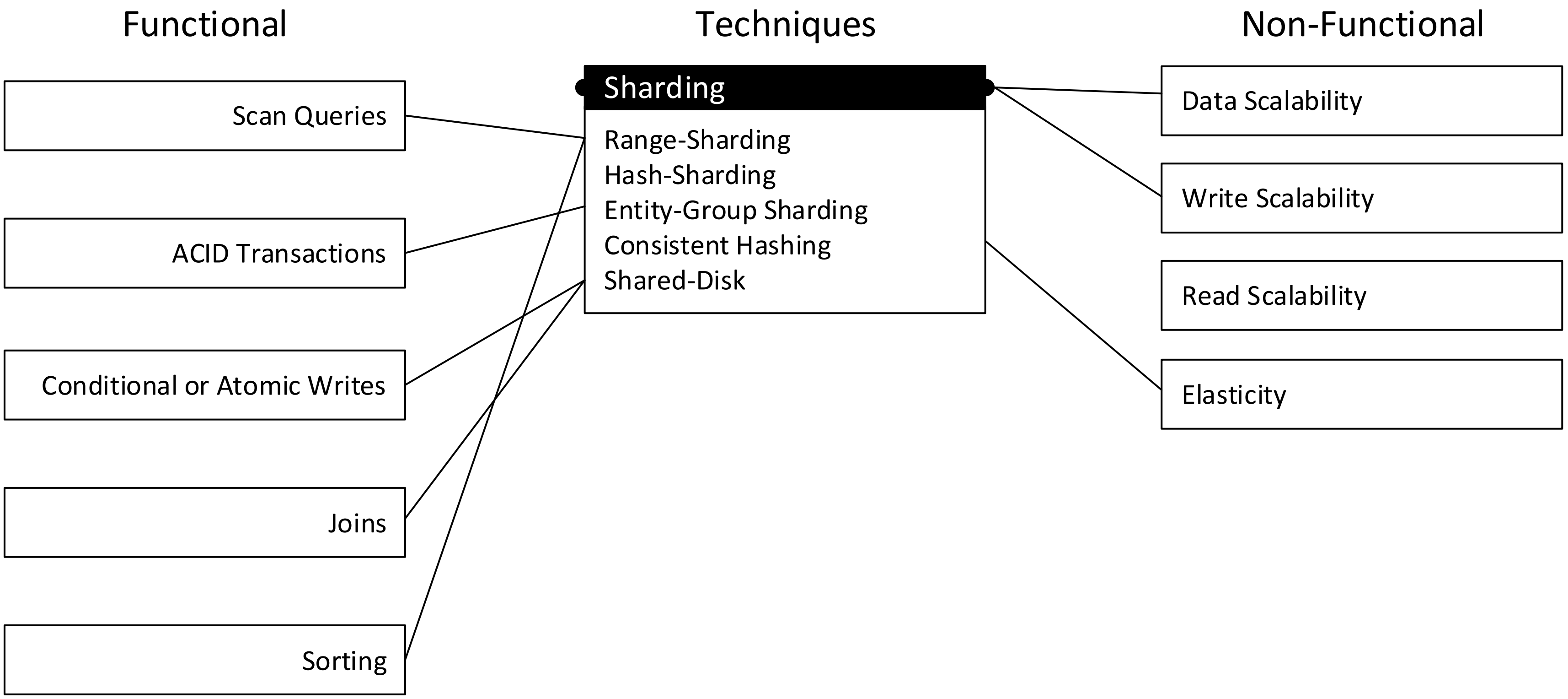
enable

enable

Functional Requirements from the application

Central techniques NoSQL databases employ

Operational Requirements



Functional

Scan Queries

ACID Transactions

Conditional or Atomic Writes

Joins

Sorting

Techniques

Sharding

Range-Sharding

Hash-Sharding

Entity-Group Sharding

Consistent Hashing

Shared-Disk

Non-Functional

Data Scalability

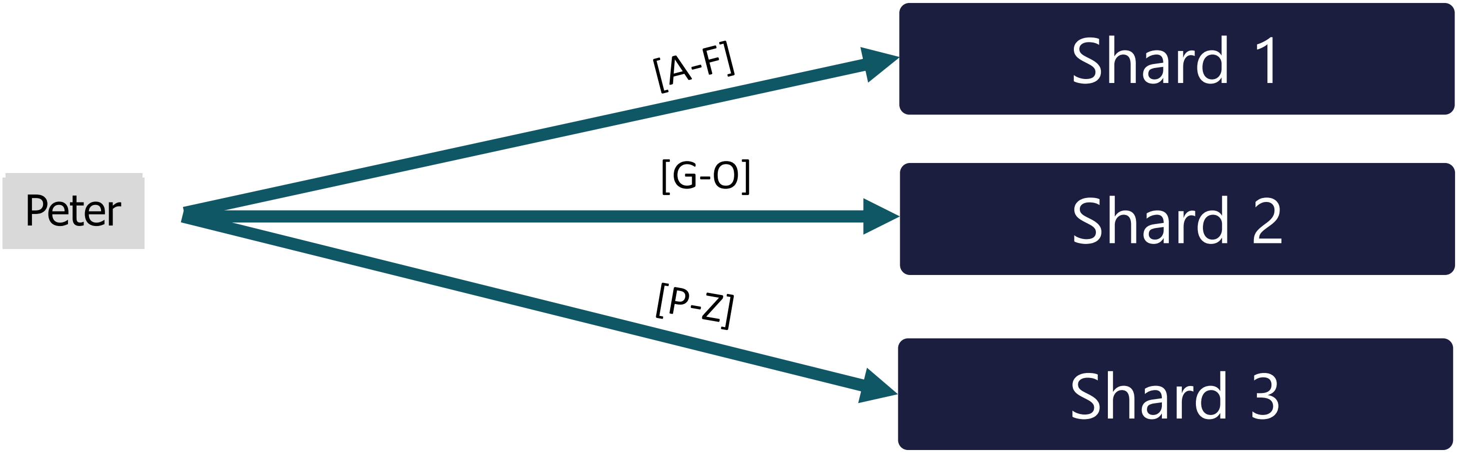
Write Scalability

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

Implemented in

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

Range-based Sharding

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

Implemented in

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

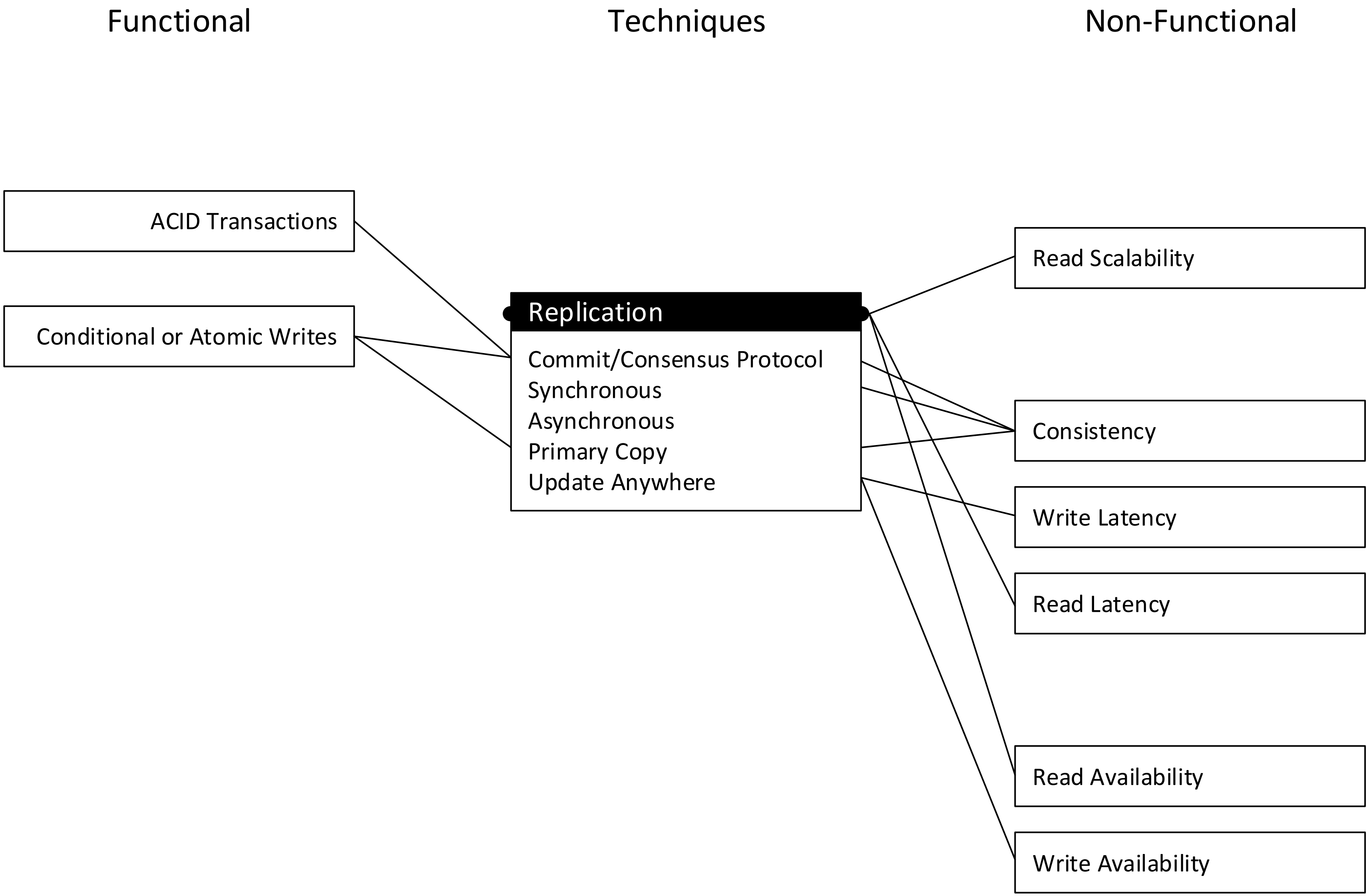
Entity-Group Sharding

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

Implemented in

G-Store, MegaStore, 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.



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

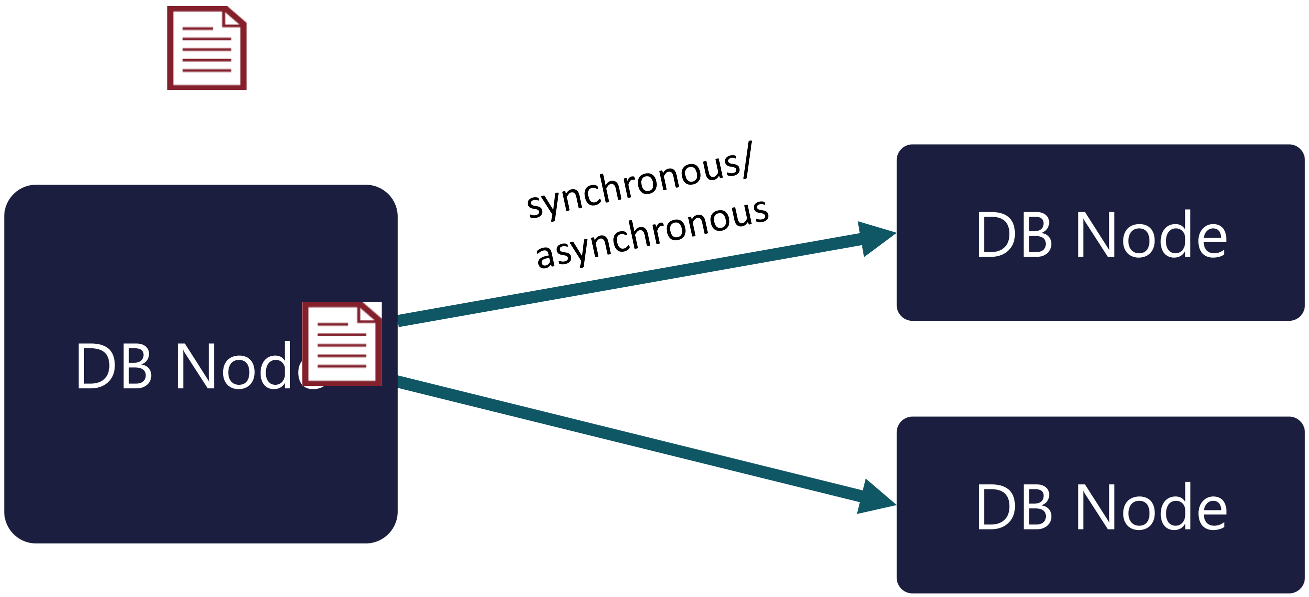
Read Latency

Read Availability

Write Availability

Replication

- Stores *N* copies of each data item



DB Node

synchronous/ asynchronous

DB Node

DB Node

- **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 sync 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*)

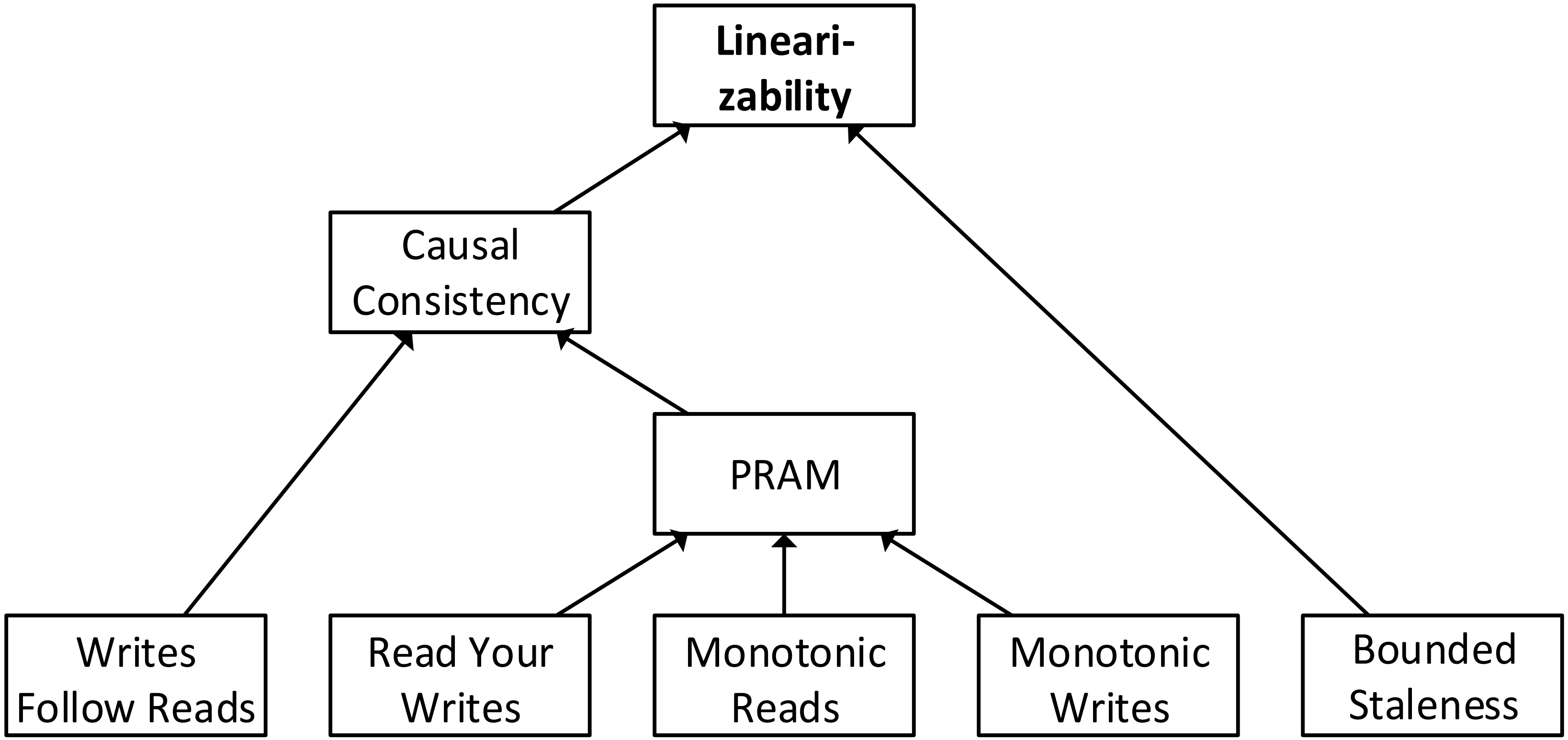
- Only a dedicated master is allowed to accept writes, slaves are read-replicas
- **Pro**: reads from the master are consistent
- **Contra**: master is a bottleneck and SPOF

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)

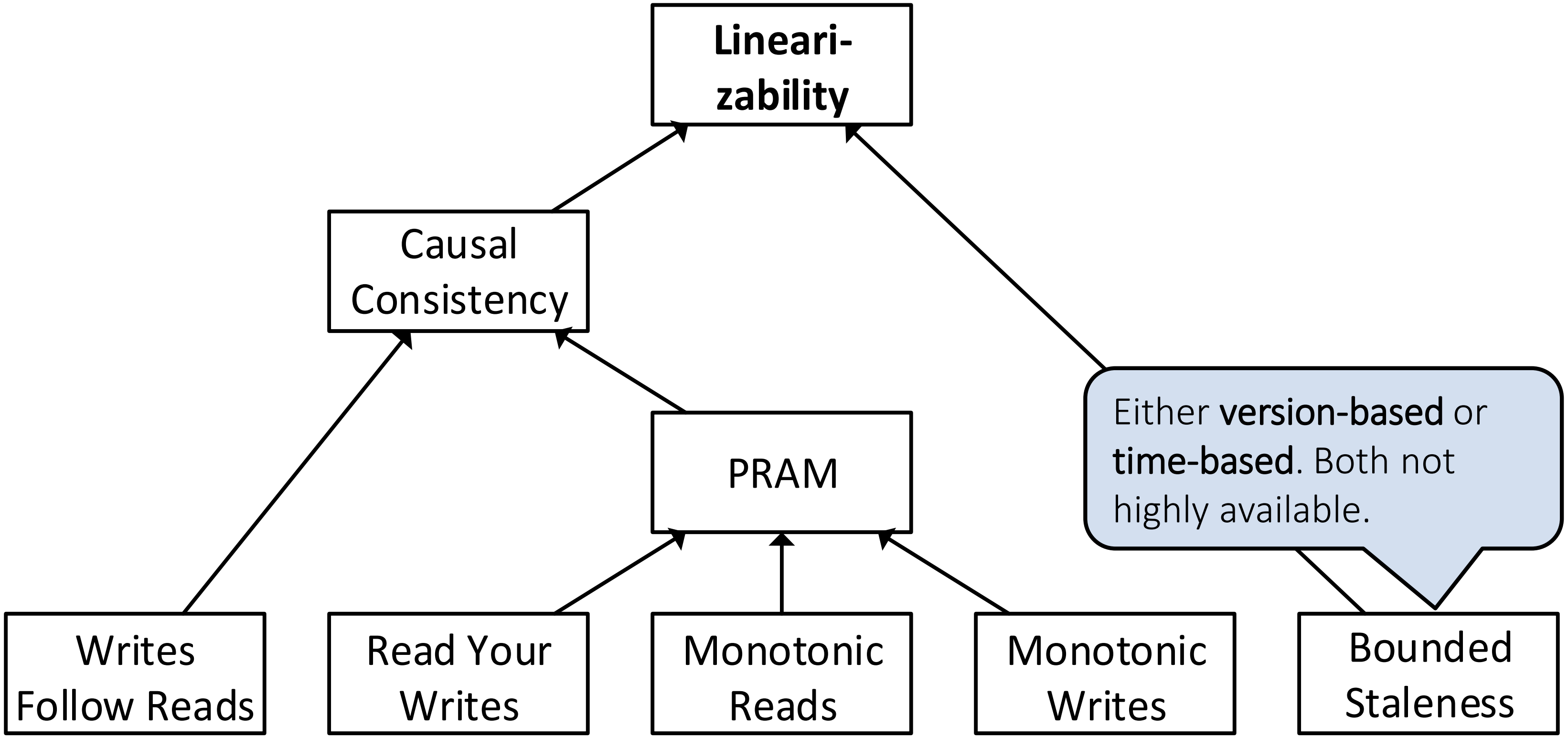
Consistency Levels



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.

Consistency Levels

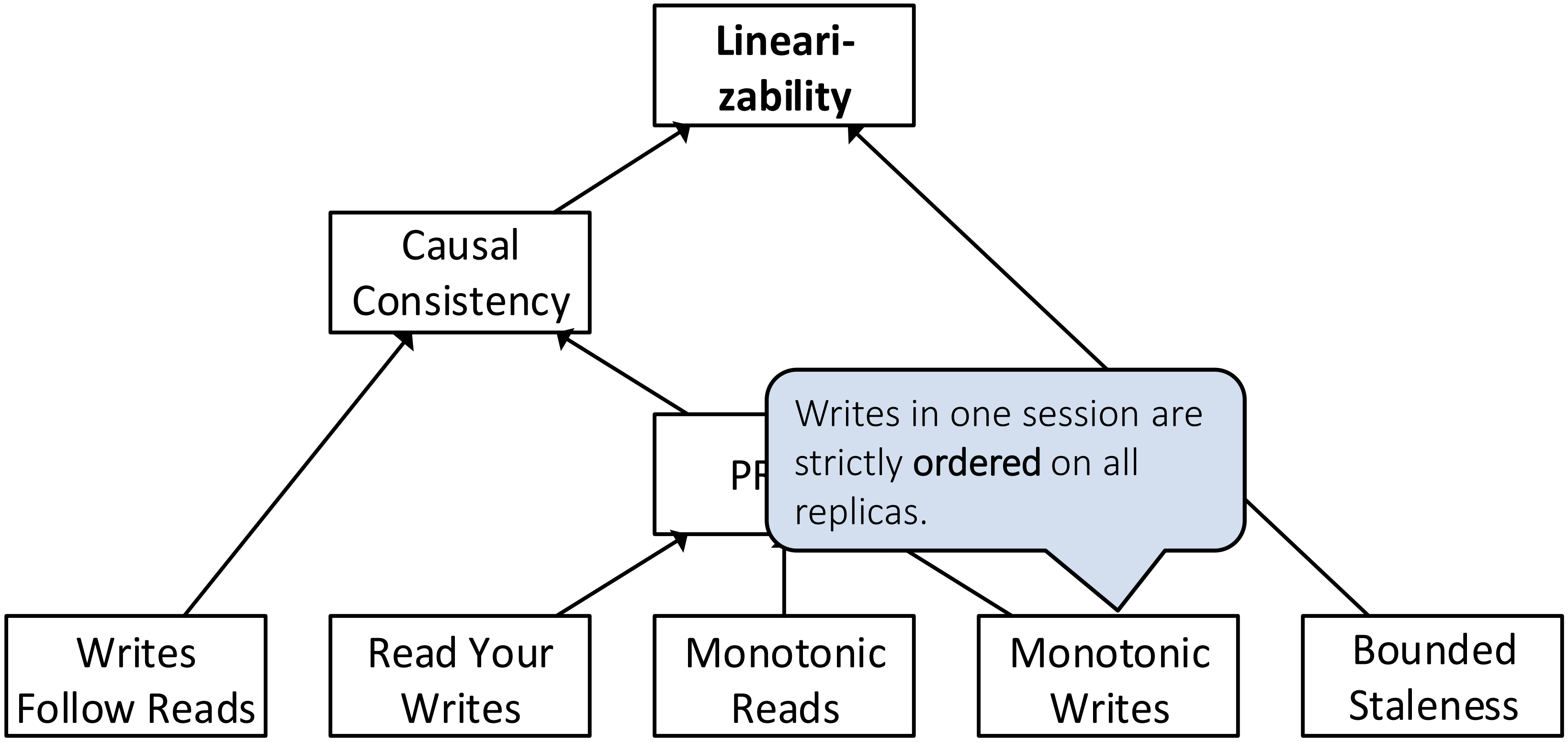


Either **version-based** or **time-based**. Both not highly available.

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.

Consistency Levels

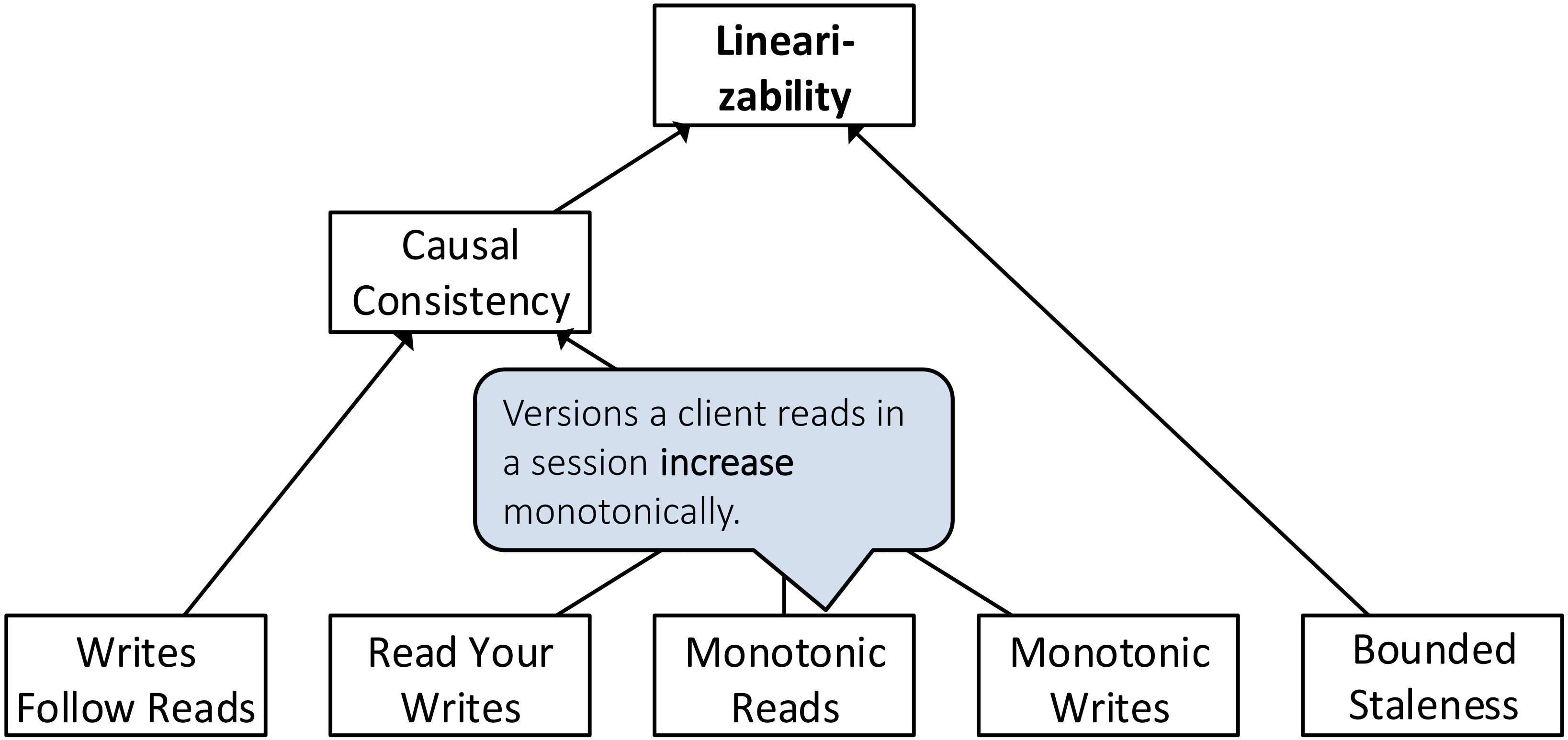


Writes in one session are strictly **ordered** on all replicas.

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.

Consistency Levels

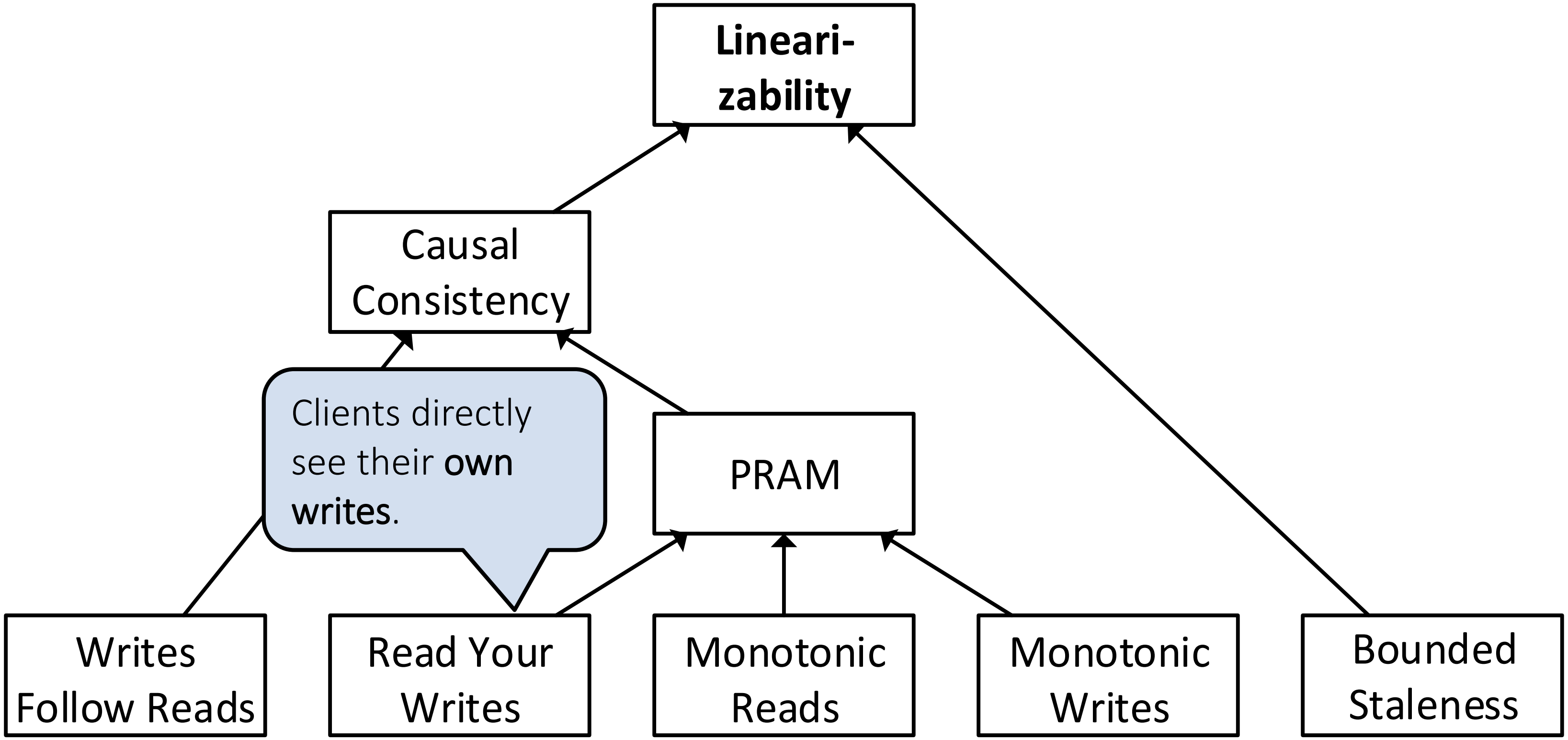


Versions a client reads in a session **increase** monotonically.

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.

Consistency Levels

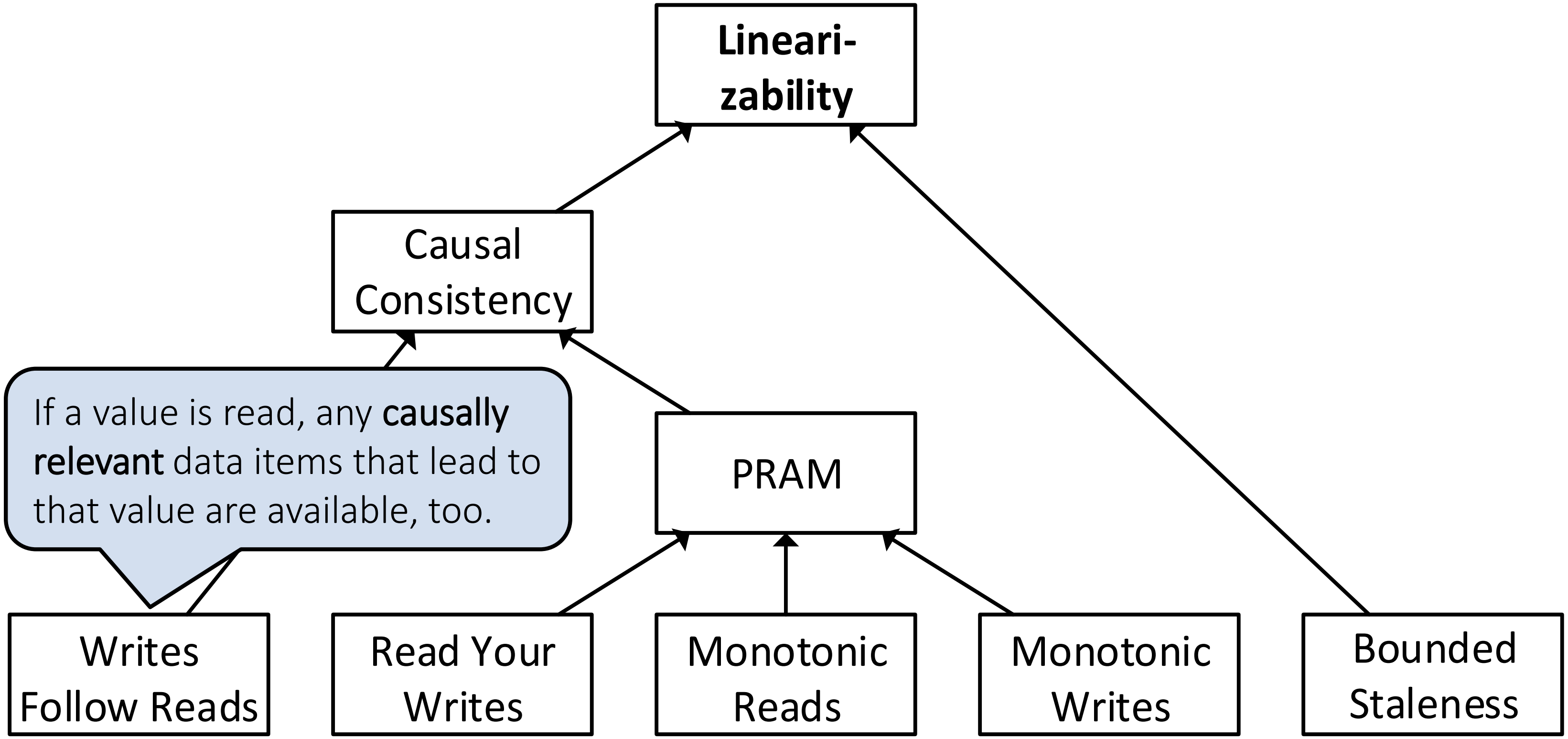


Clients directly see their **own writes**.

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.

Consistency Levels

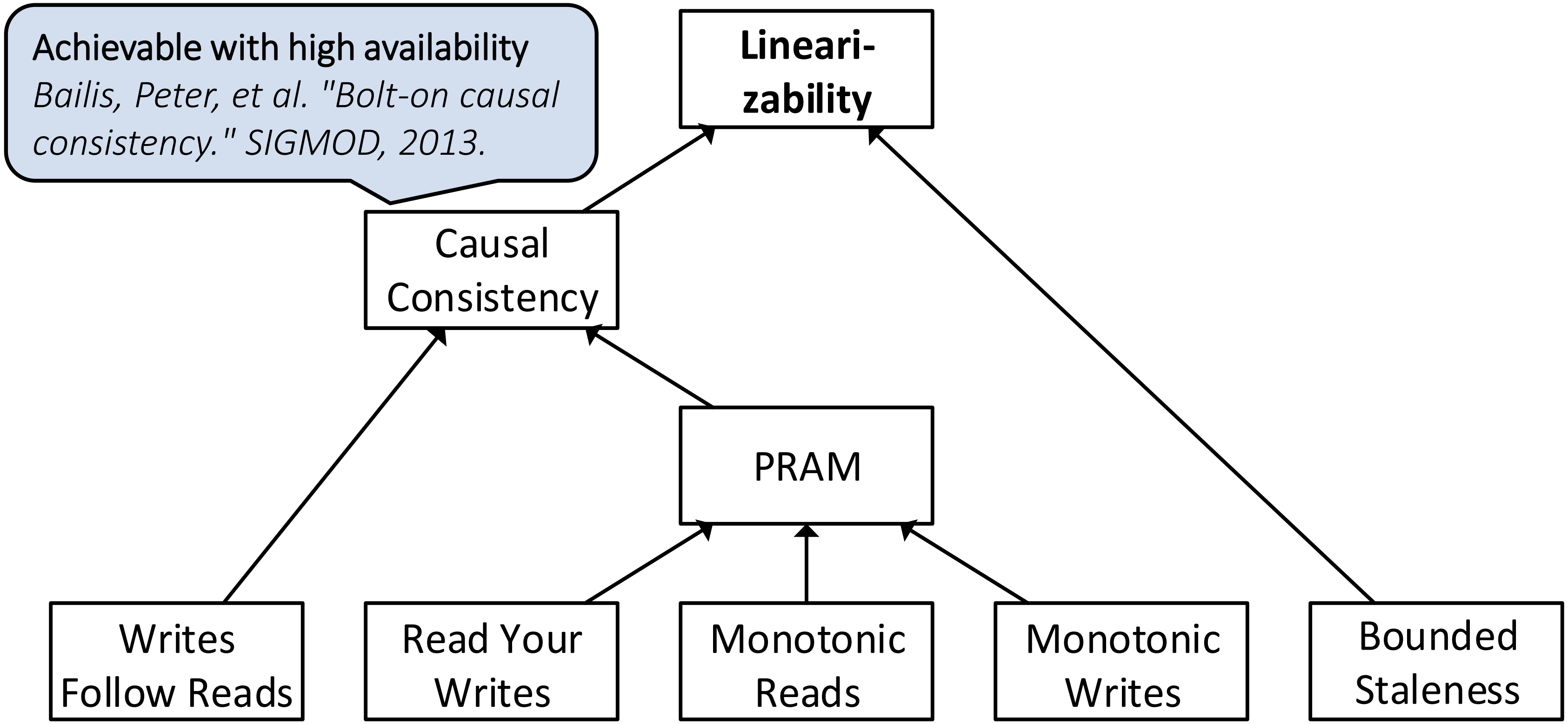


If a value is read, any **causally relevant** data items that lead to that value are available, too.

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.

Consistency Levels

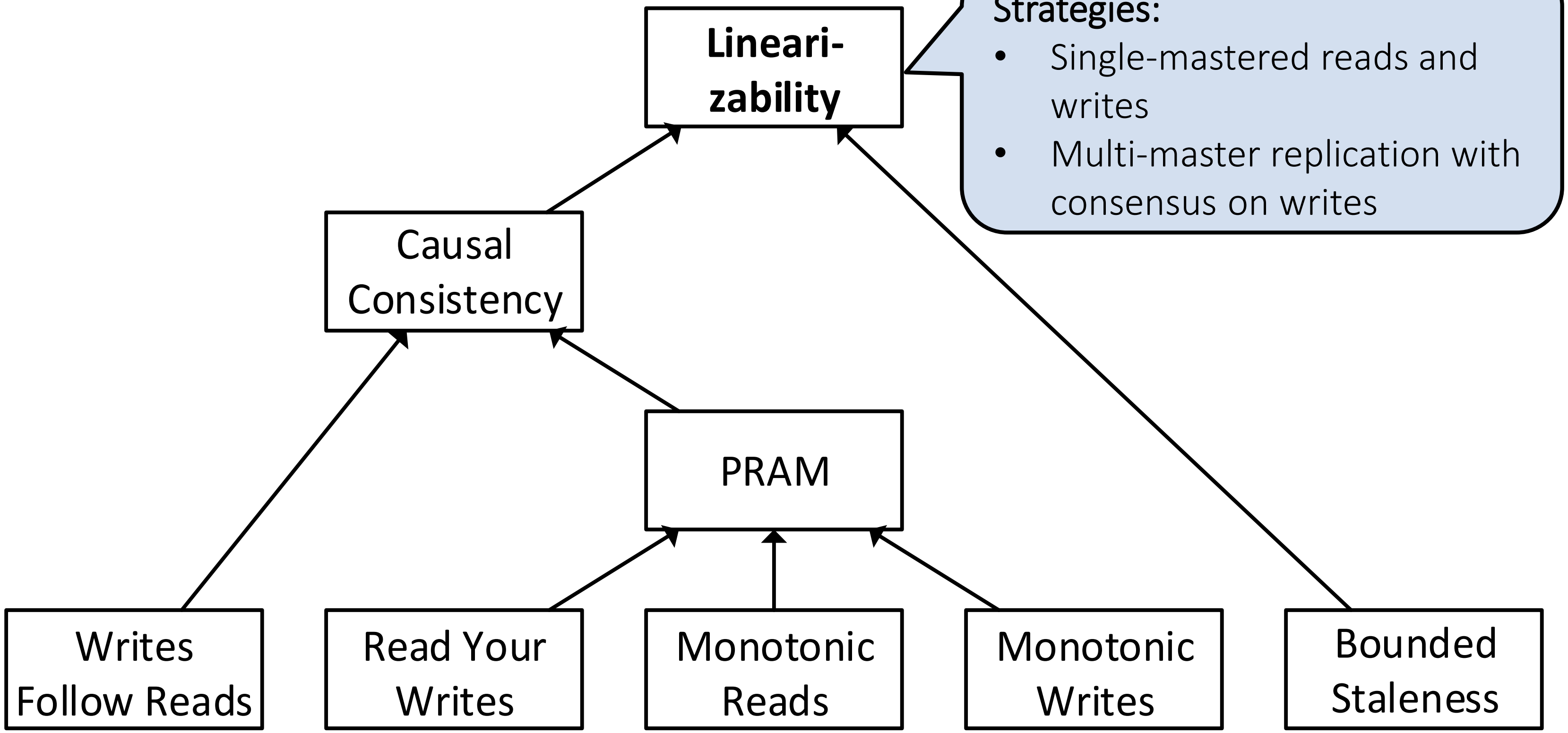


Achievable with high availability
Bailis, Peter, et al. "Bolt-on causal consistency." SIGMOD, 2013.

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.

Consistency Levels



Strategies:

- Single-mastered reads and writes
- Multi-master replication with consensus on writes

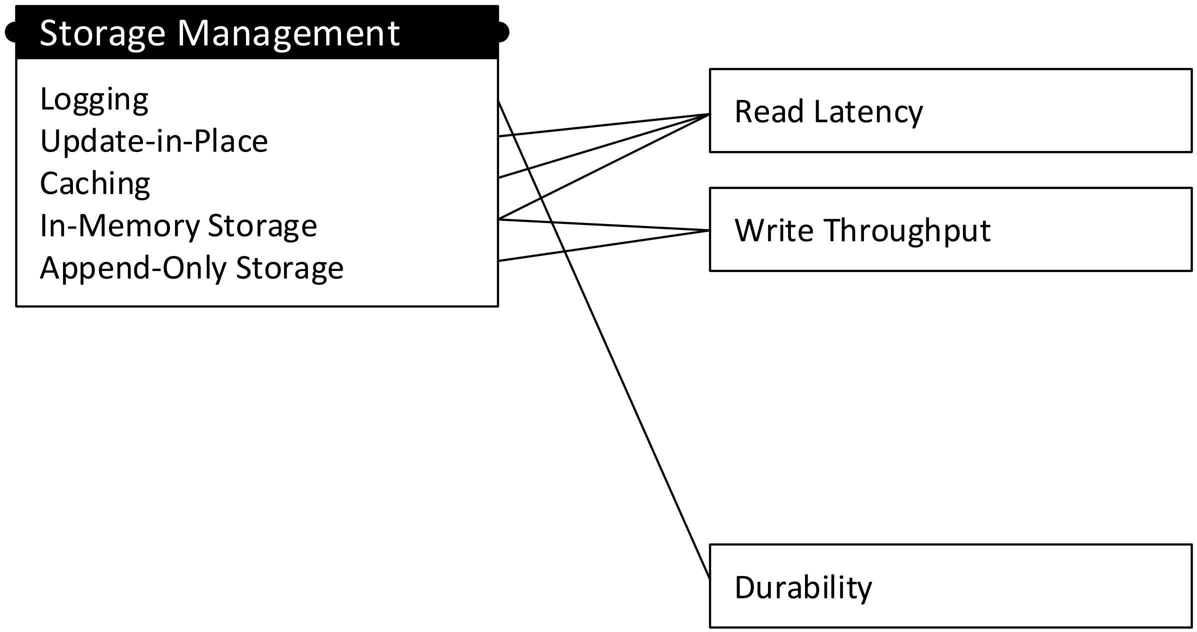
Viotti, Paolo, and Marko Vukolić. "Consistency in Non-Transactional Distributed Storage Systems." arXiv (2015).

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

Functional

Techniques

Non-Functional



Storage Management

Logging
Update-in-Place
Caching
In-Memory Storage
Append-Only Storage

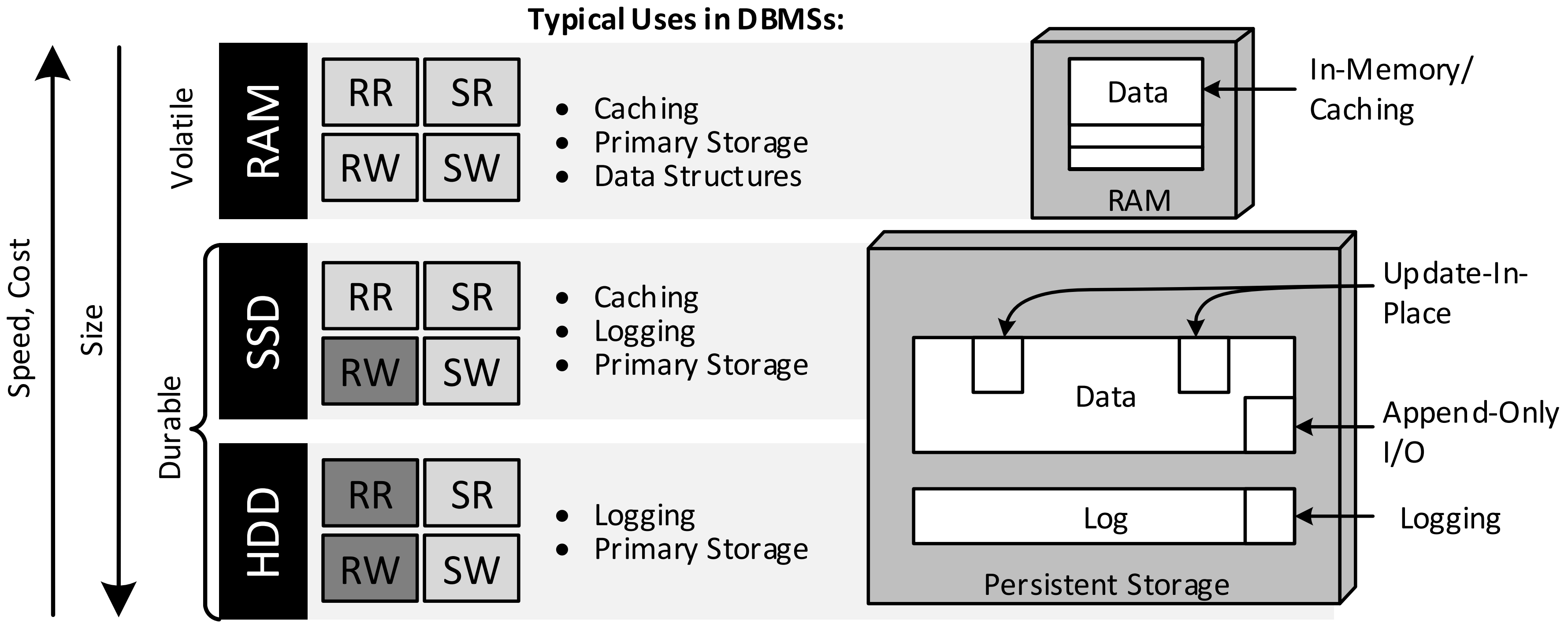
Read Latency

Write Throughput

Durability

NoSQL Storage Management

In a Nutshell



Typical Uses in DBMSs:

RAM
- Caching
- Primary Storage
- Data Structures

SSD
- Caching
- Logging
- Primary Storage

HDD
- Logging
- Primary Storage

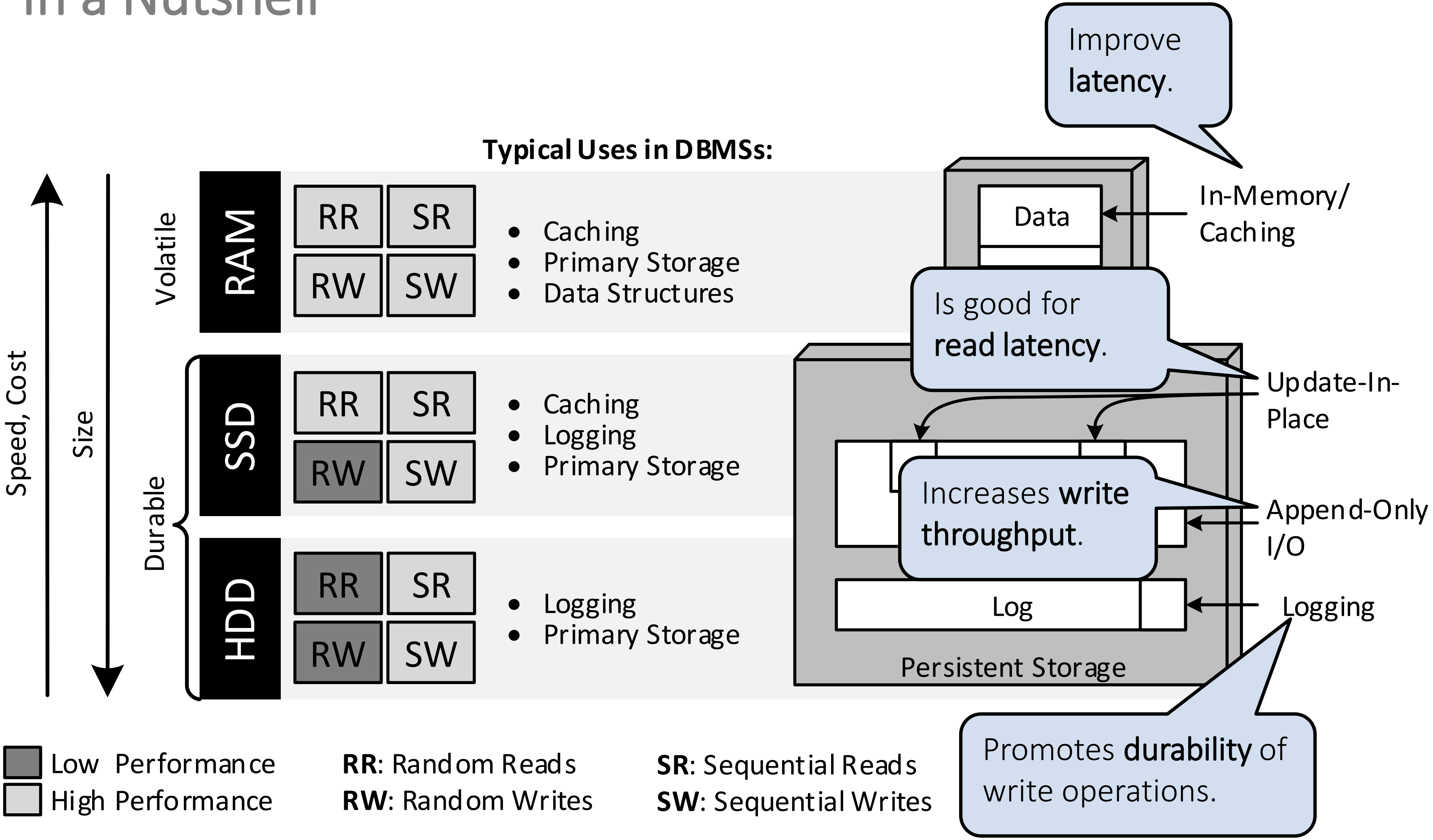
▨ Low Performance
□ High Performance

RR: Random Reads
RW: Random Writes

SR: Sequential Reads
SW: Sequential Writes

NoSQL Storage Management

In a Nutshell



Typical Uses in DBMSs:

Volatile	RAM	□ RR □ RW	□ SR □ SW	- Caching - Primary Storage - Data Structures
Durable	SSD	□ RR ■ RW	□ SR □ SW	- Caching - Logging - Primary Storage
Durable	HDD	■ RR ■ RW	□ SR □ SW	- Logging - Primary Storage

Improve **latency**.

Is good for **read latency**.

Increases **write throughput**.

Promotes **durability** of write operations.

■ Low Performance
□ High Performance

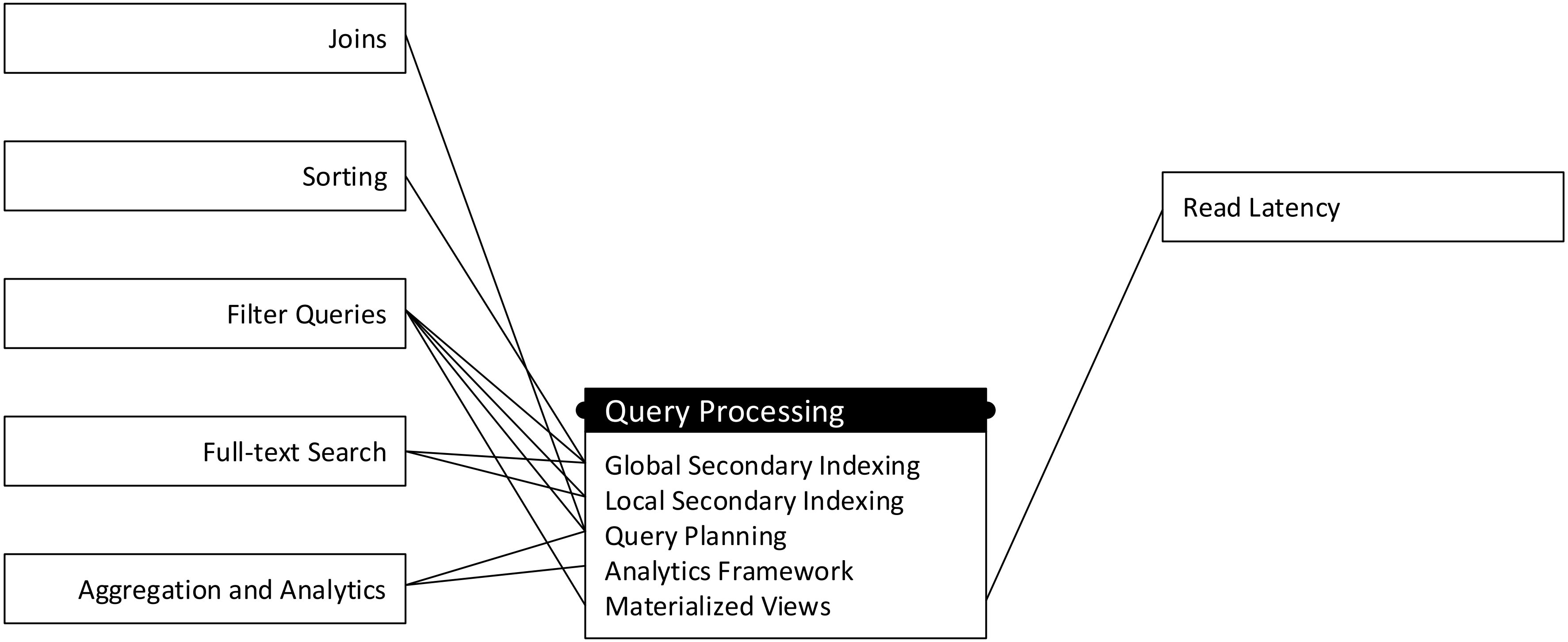
RR: Random Reads
RW: Random Writes

SR: Sequential Reads
SW: Sequential Writes

Functional

Techniques

Non-Functional



Joins

Sorting

Filter Queries

Full-text Search

Aggregation and Analytics

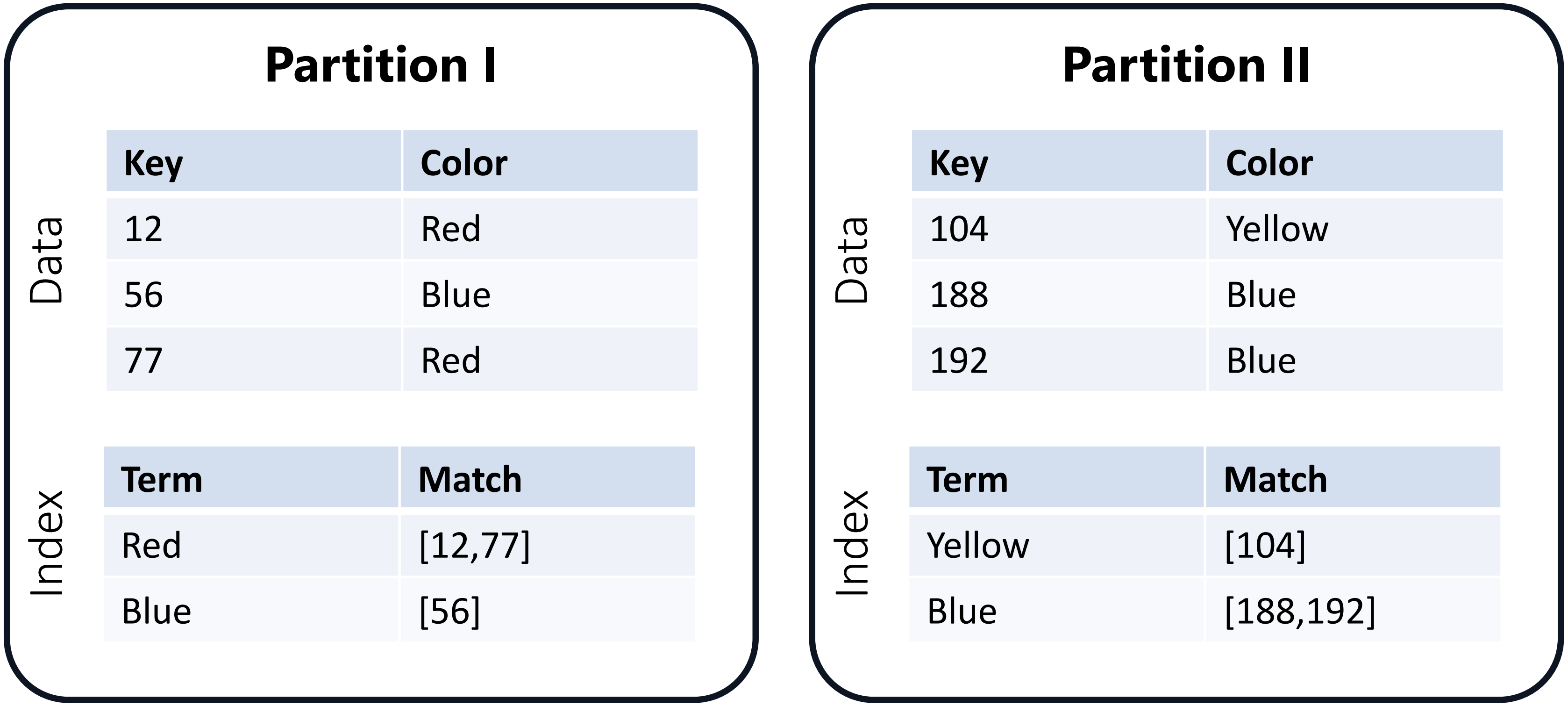
Query Processing

Global Secondary Indexing
Local Secondary Indexing
Query Planning
Analytics Framework
Materialized Views

Read Latency

Local Secondary Indexing

Partitioning By Document



Partition I

Data

Key	Color
12	Red
56	Blue
77	Red

Index

Term	Match
Red	[12,77]
Blue	[56]

Partition II

Data

Key	Color
104	Yellow
188	Blue
192	Blue

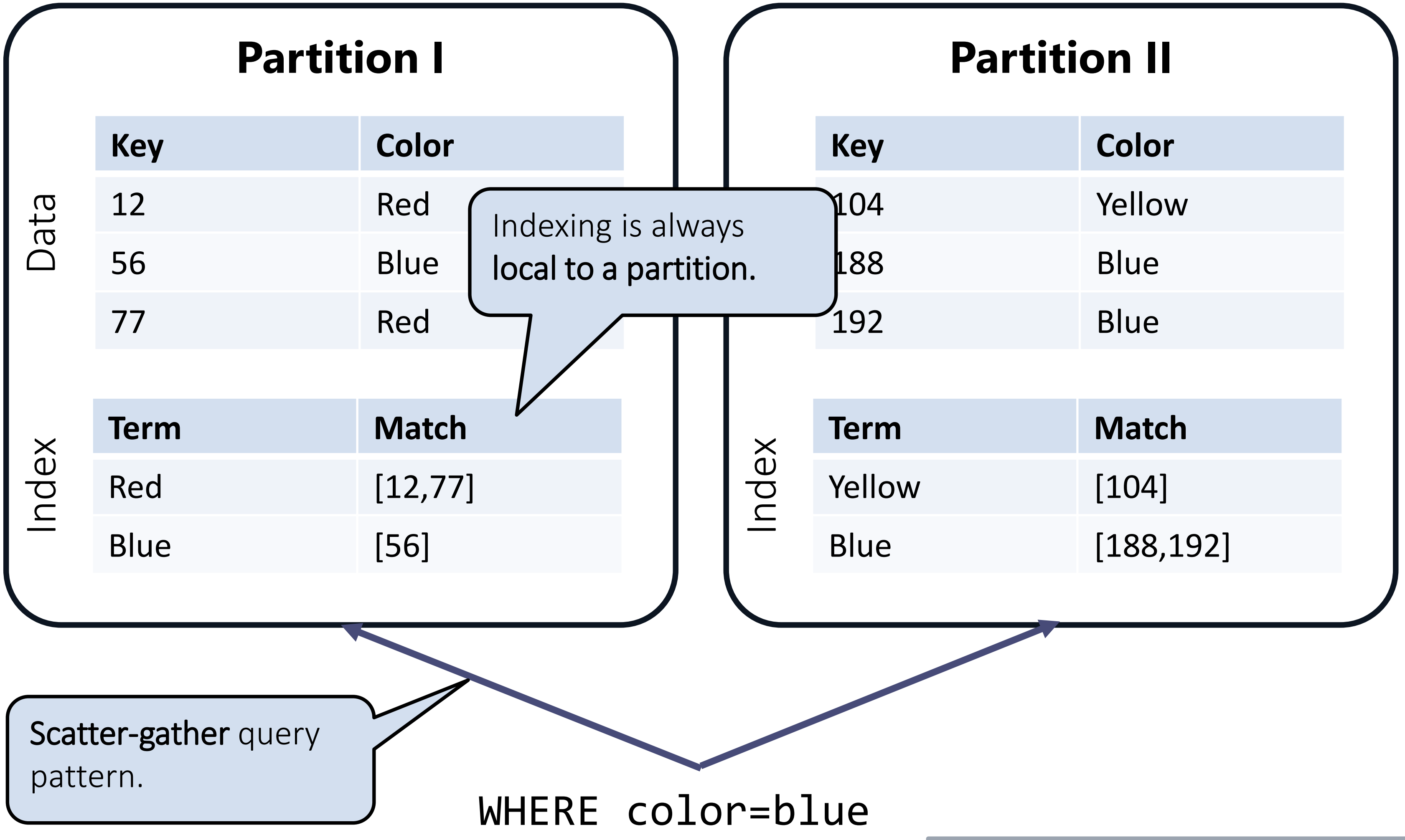
Index

Term	Match
Yellow	[104]
Blue	[188,192]

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

Local Secondary Indexing

Partitioning By Document



Partition I

Data

Key	**Color**
12	Red
56	Blue
77	Red

Index

Term	**Match**
Red	[12,77]
Blue	[56]

Partition II

Key	**Color**
104	Yellow
188	Blue
192	Blue

Index

Term	**Match**
Yellow	[104]
Blue	[188,192]

Indexing is always **local to a partition.**

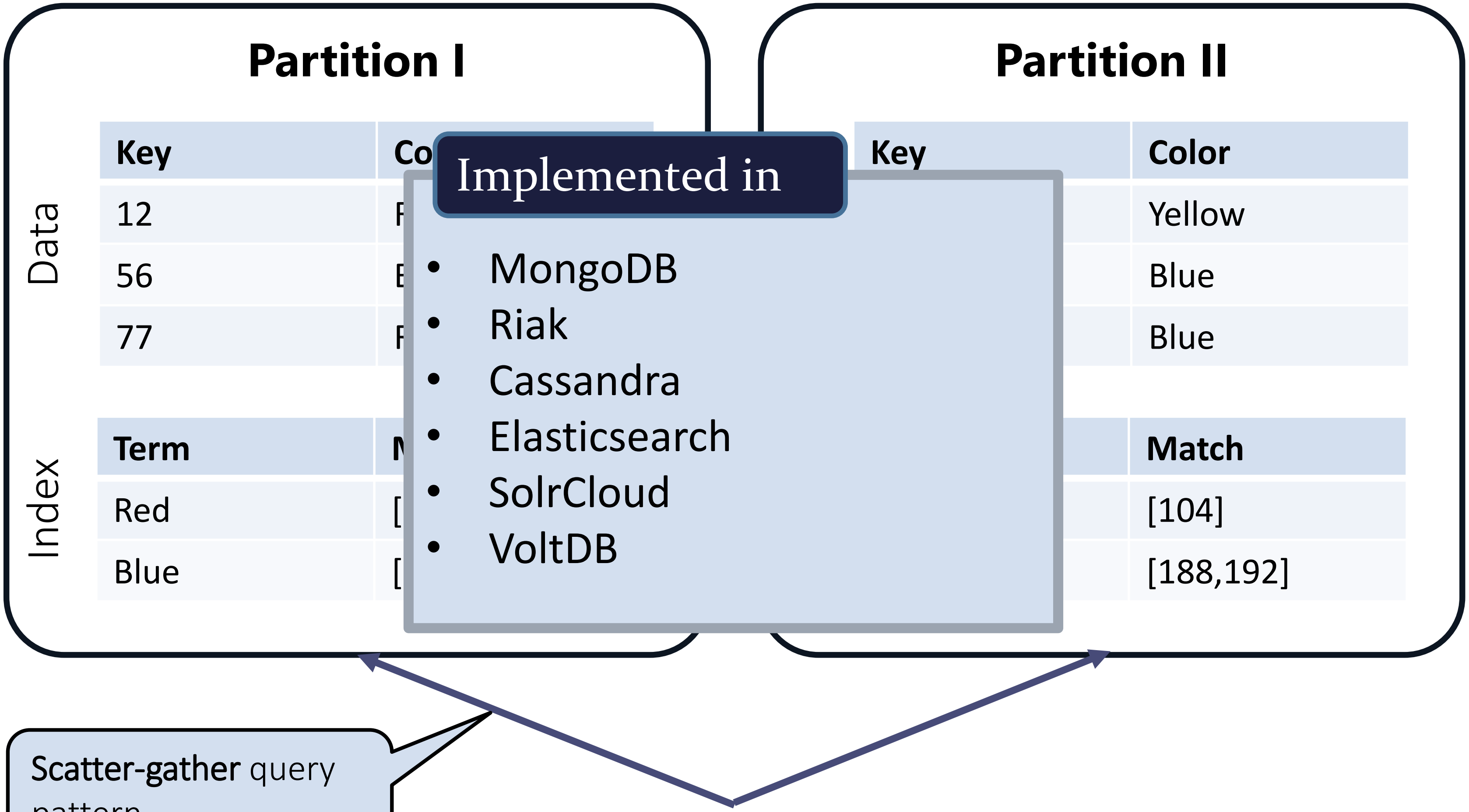
Scatter-gather query pattern.

```
WHERE color=blue
```

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

Local Secondary Indexing

Partitioning By Document



Partition I

Data

Key	**Co**
12	F
56	E
77	F

Index

Term	**M**
Red	[
Blue	[

Partition II

Key	**Color**
	Yellow
	Blue
	Blue

	Match
	[104]
	[188,192]

Implemented in

- MongoDB
- Riak
- Cassandra
- Elasticsearch
- SolrCloud
- VoltDB

Scatter-gather query pattern.

```
WHERE color=blue
```

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

Global Secondary Indexing

Partitioning By Term

Partition I

Data

Key	Color
12	Red
56	Blue
77	Red

Index

Term	Match
Yellow	[104]
Blue	[56, 188, 192]

Partition II

Data

Key	Color
104	Yellow
188	Blue
192	Blue

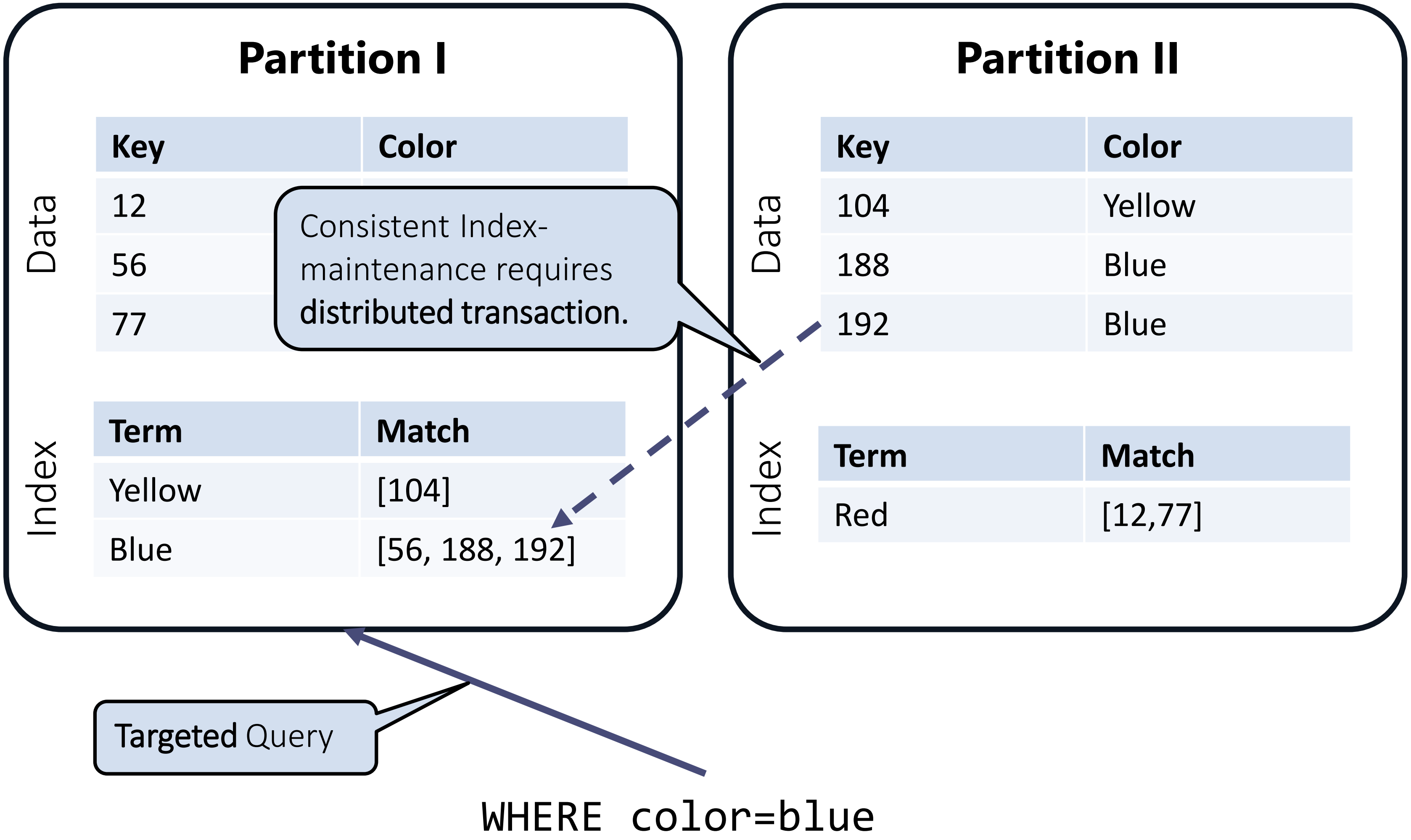
Index

Term	Match
Red	[12,77]

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

Global Secondary Indexing

Partitioning By Term



Partition I

Data

Key	Color
12	
56	
77	

Index

Term	Match
Yellow	[104]
Blue	[56, 188, 192]

Partition II

Data

Key	Color
104	Yellow
188	Blue
192	Blue

Index

Term	Match
Red	[12,77]

Consistent Index-maintenance requires **distributed transaction.**

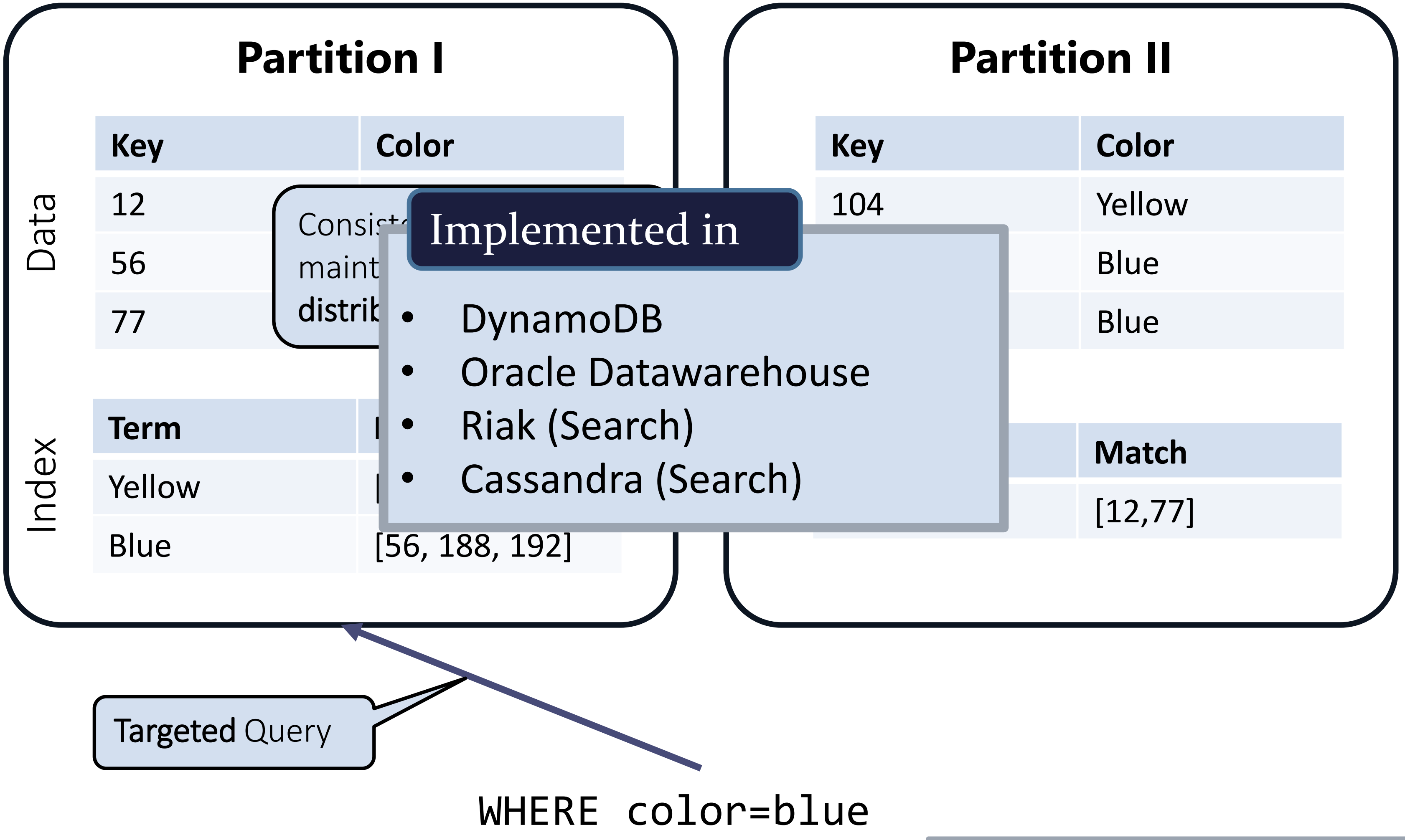
Targeted Query

```
WHERE color=blue
```

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

Global Secondary Indexing

Partitioning By Term



Partition I

Data

Key	**Color**
12	
56	
77	

Index

Term	**I**
Yellow	[
Blue	[56, 188, 192]

Consist
maint
distrib

Partition II

Key	**Color**
104	Yellow
	Blue
	Blue

	Match
	[12,77]

Implemented in

- DynamoDB
- Oracle Datawarehouse
- Riak (Search)
- Cassandra (Search)

Targeted Query

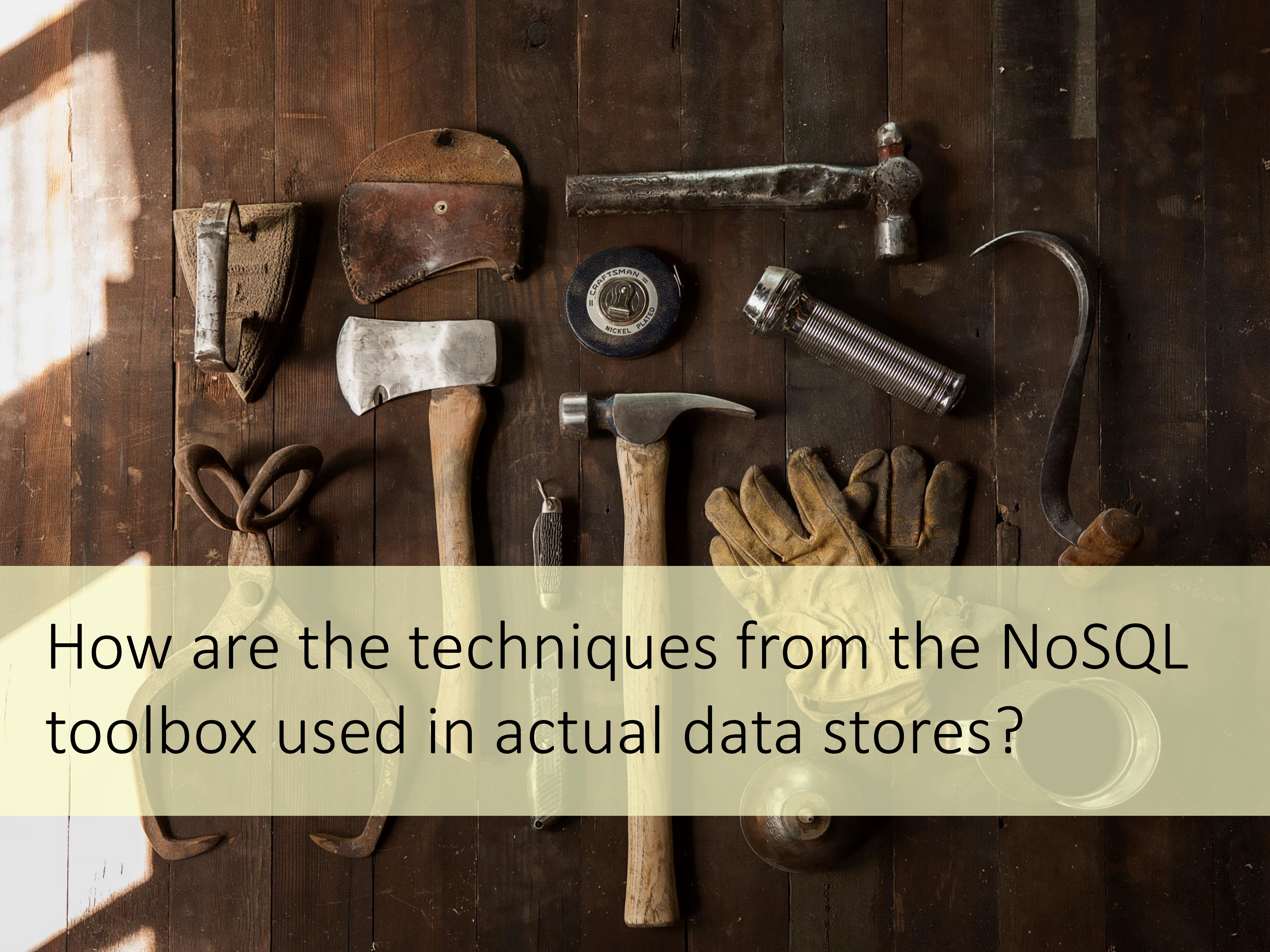
```
WHERE color=blue
```

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

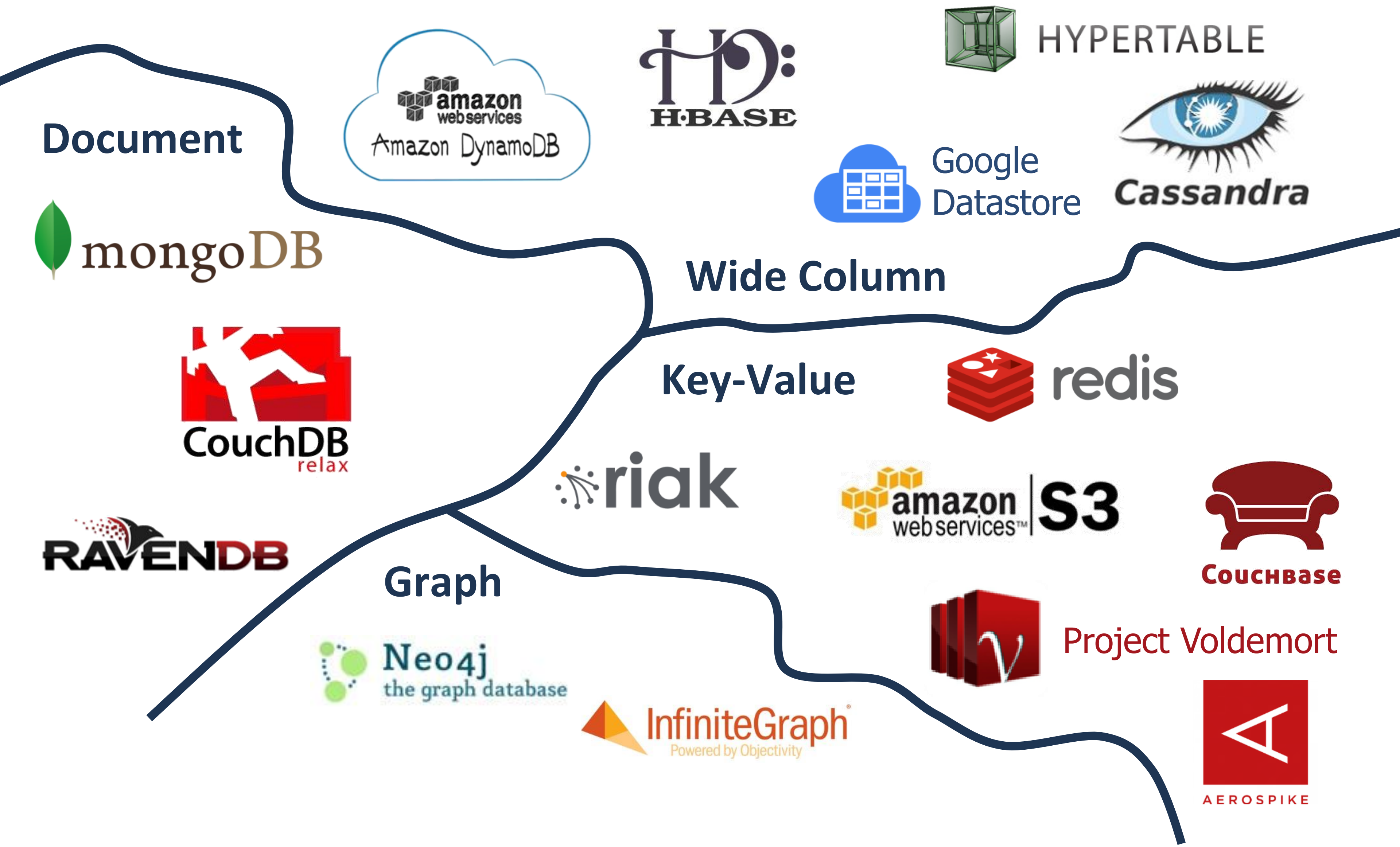
The NoSQL Toolbox: Common Techniques

NoSQL Systems

- Overview & Popularity
- Dynamo & Riak
- HBase
- Cassandra
- Redis
- MongoDB

Decision Guidance: NoSQL Decision Tree

NoSQL Landscape



Document

mongoDB

CouchDB relax

RAVENDB

amazon web services Amazon DynamoDB

Wide Column

H·BASE

HYPERTABLE

Google Datastore

Cassandra

Key-Value

redis

riak

amazon web services™ S3

Couchbase

Project Voldemort

AEROSPIKE

Graph

Neo4j the graph database

InfiniteGraph Powered by Objectivity

Popularity

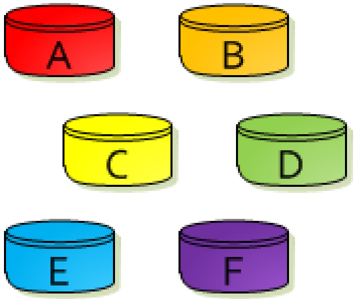
http://db-engines.com/de/ranking

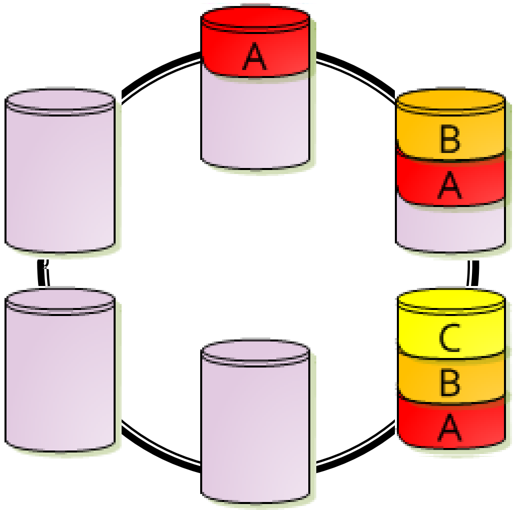
#	System	Model	Score
1.	Oracle	Relational DBMS	1462.02
2.	MySQL	Relational DBMS	1371.83
3.	MS SQL Server	Relational DBMS	1142.82
4.	**MongoDB**	**Document store**	**320.22**
5.	PostgreSQL	Relational DBMS	307.61
6.	DB2	Relational DBMS	185.96
7.	**Cassandra**	**Wide column store**	**134.50**
8.	Microsoft 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

Dynamo (AP)

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

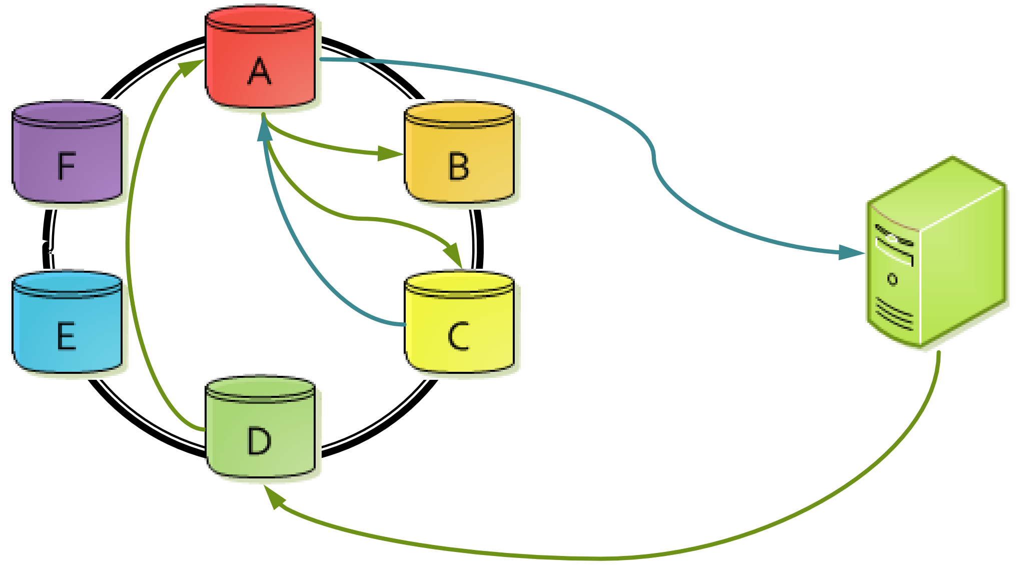




DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."

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

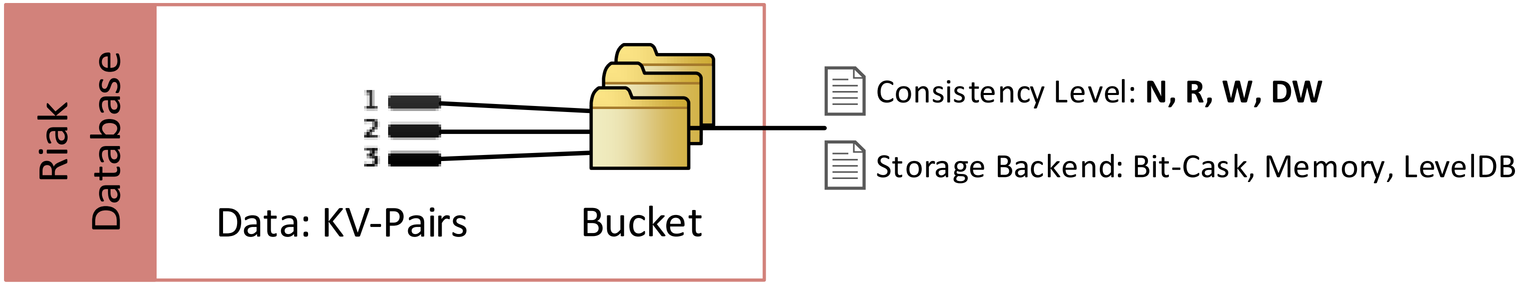


N=3
R=2
W=1

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



Riak Database

Data: KV-Pairs

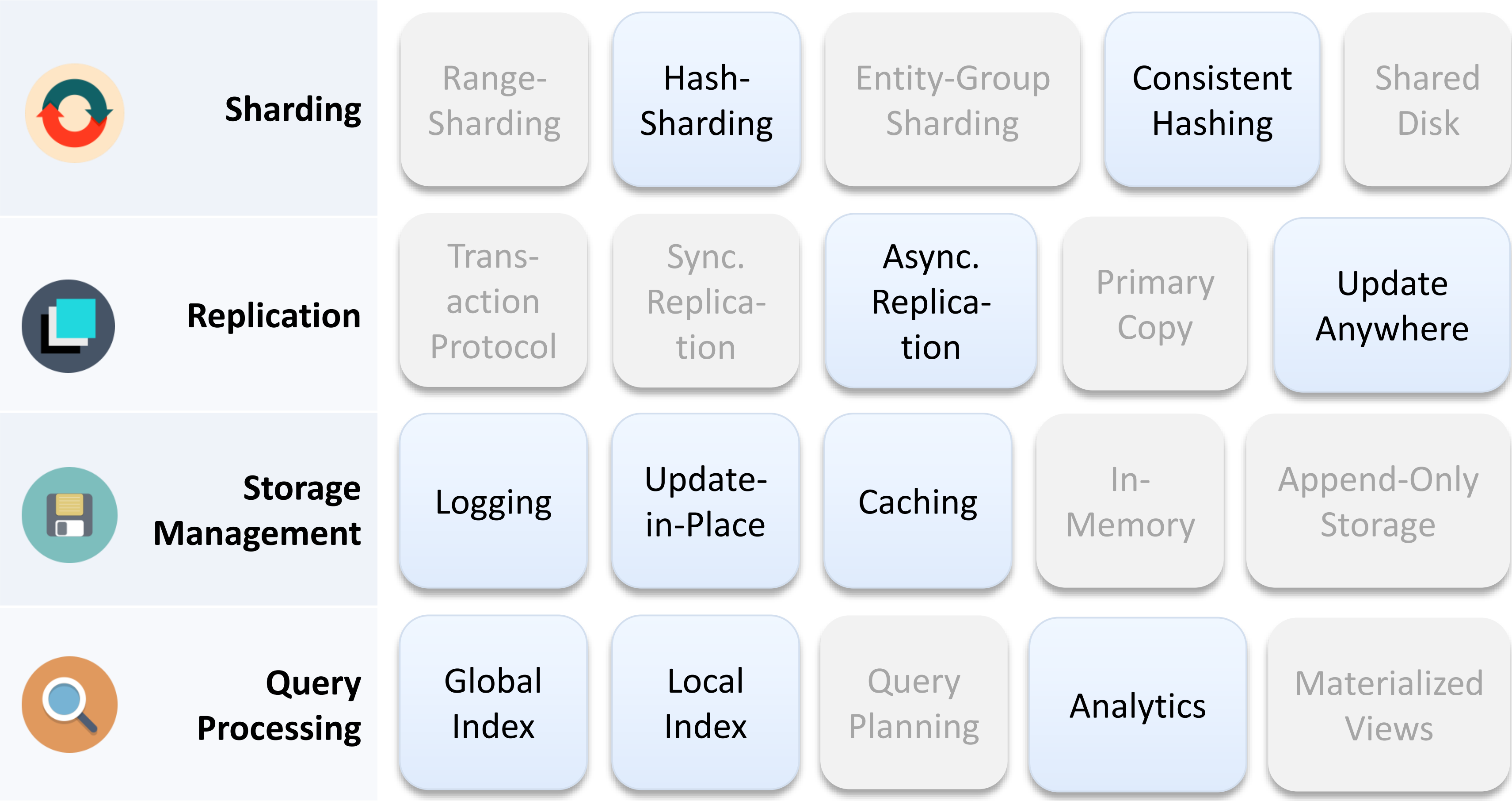
Bucket

Consistency Level: **N, R, W, DW**

Storage Backend: Bit-Cask, Memory, LevelDB

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

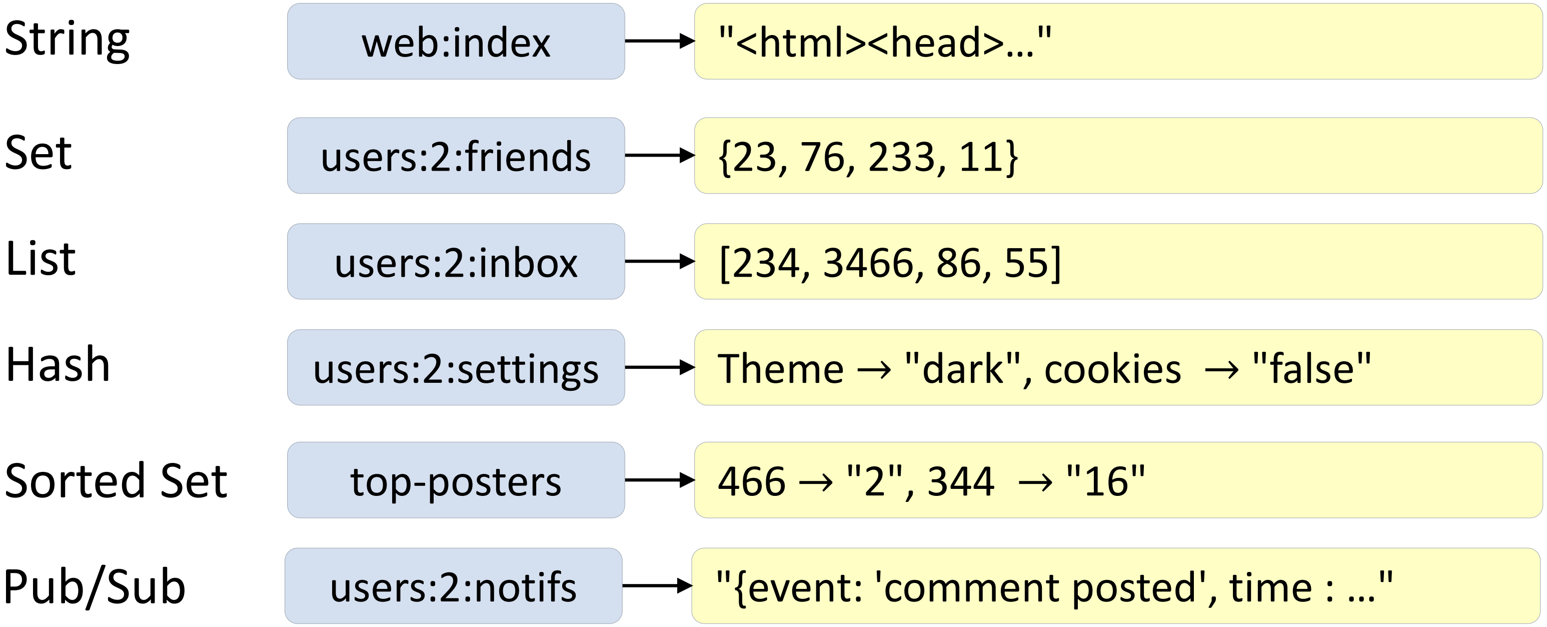
Redis (CA)

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

- **Re**mote **Di**ctionary **S**erver
- Rich Key-Value model
- Asynchronous Master-Slave Replication
- **Tunable persistence**: logging and snapshots
- Optimistic **batch transactions** (*Multi blocks*)
- Very high performance: >100k ops/sec per node
- Redis Cluster (sharding) still in the early stages

Redis 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 : …"

Classification: Redis

Techniques

Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Transaction Protocol	Sync. Replication	**Async. Replication**	**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

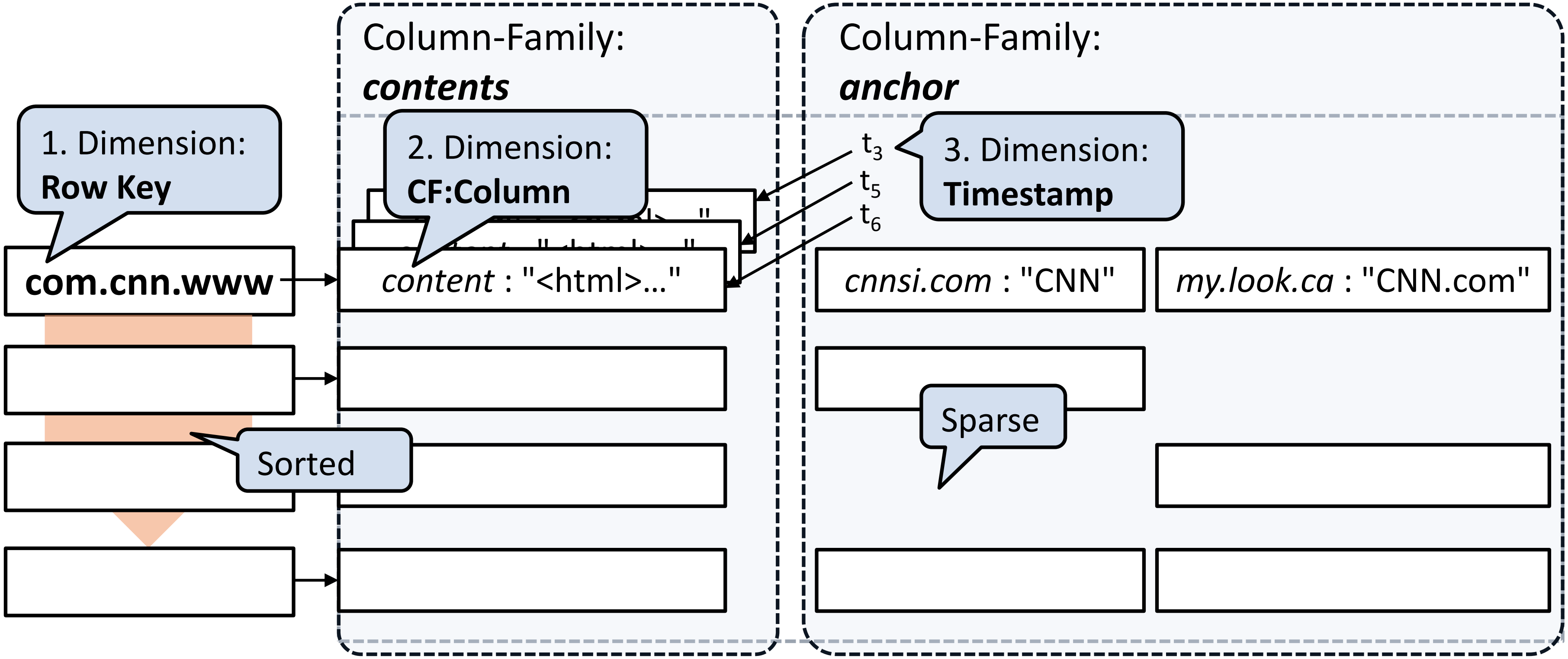
> A Bigtable is a sparse, distributed, persistent multidimensional sorted map.

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

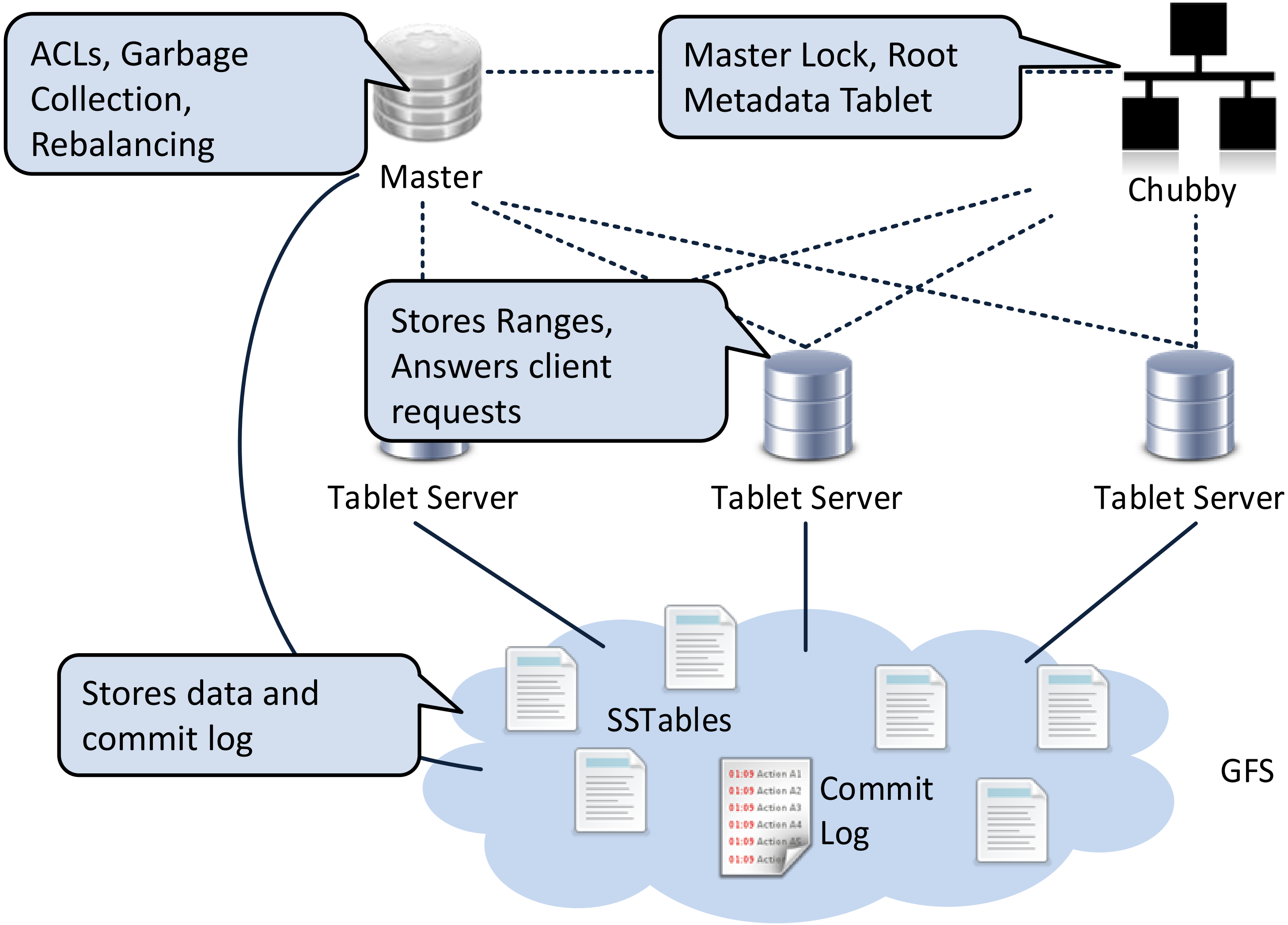
📖 Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."

Wide-Column Data Modelling

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



Architecture



ACLs, Garbage Collection, Rebalancing

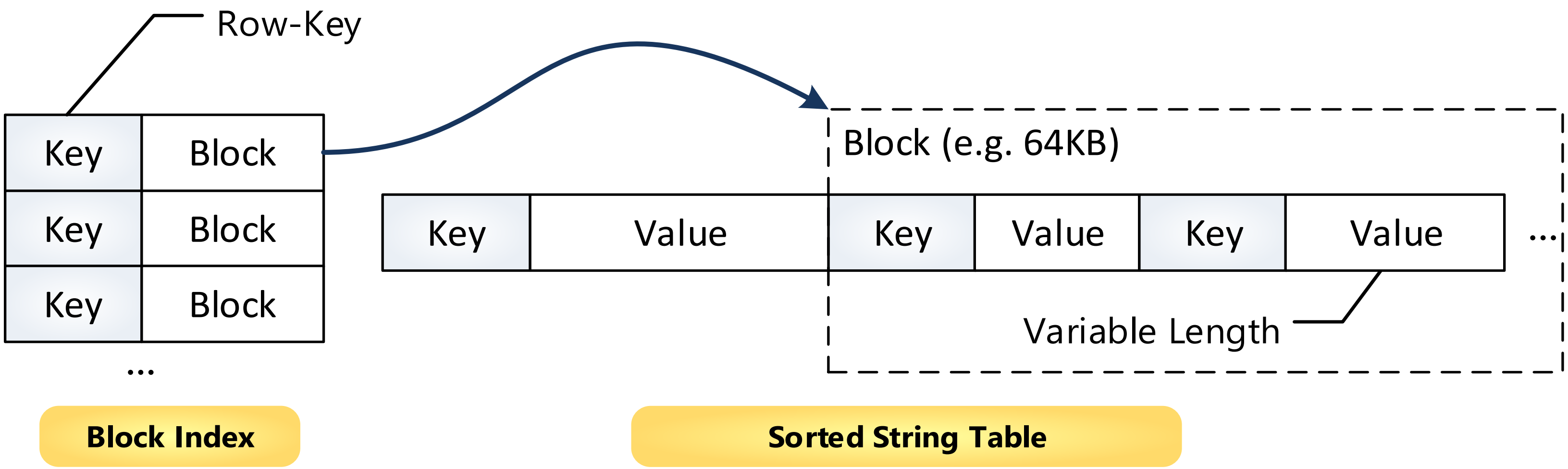
Master Lock, Root Metadata Tablet

Stores Ranges, Answers client requests

Stores data and commit log

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

- 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:_
```

Key	cf1:c1	cf1:c2	cf2:c1	cf2:c2
r1	🟥		🟧	
r2		🟥		🟧
r3				🟧🟧
r4		🟥		
r5	🟥🟥		🟧	

```
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>
```

File 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>
```

File cf1

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

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 challenge: 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*)
- Interface: REST, Avro, Thrift

Classification: HBase

Techniques

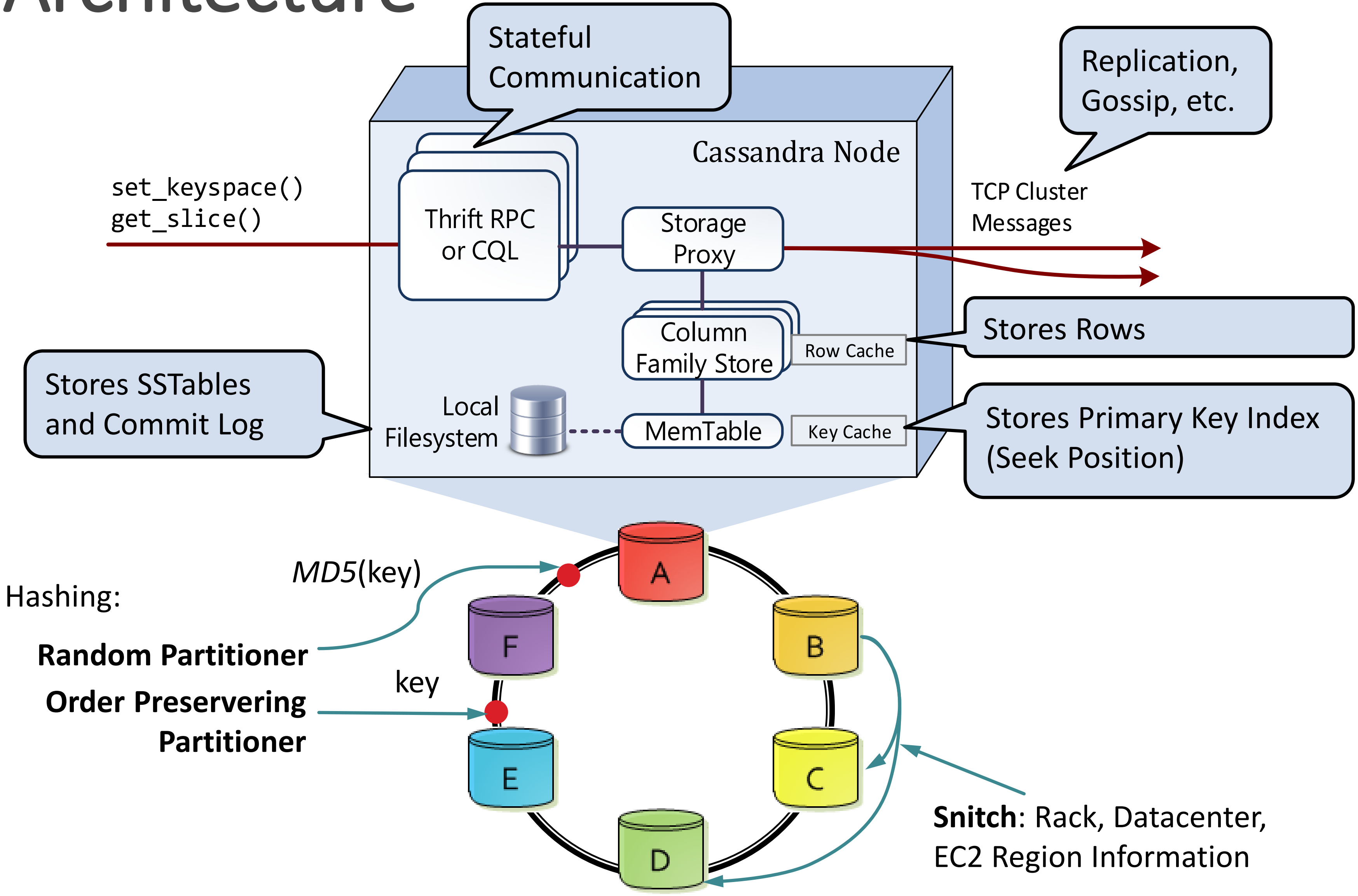
Sharding	**Range-Sharding**	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	**Sync. Replica-tion**	Async. Replica-tion	**Primary Copy**	Update Anywhere
Storage Management	**Logging**	Update-in-Place	**Caching**	In-Memory	**Append-Only Storage**
Query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

Apache Cassandra (AP)

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

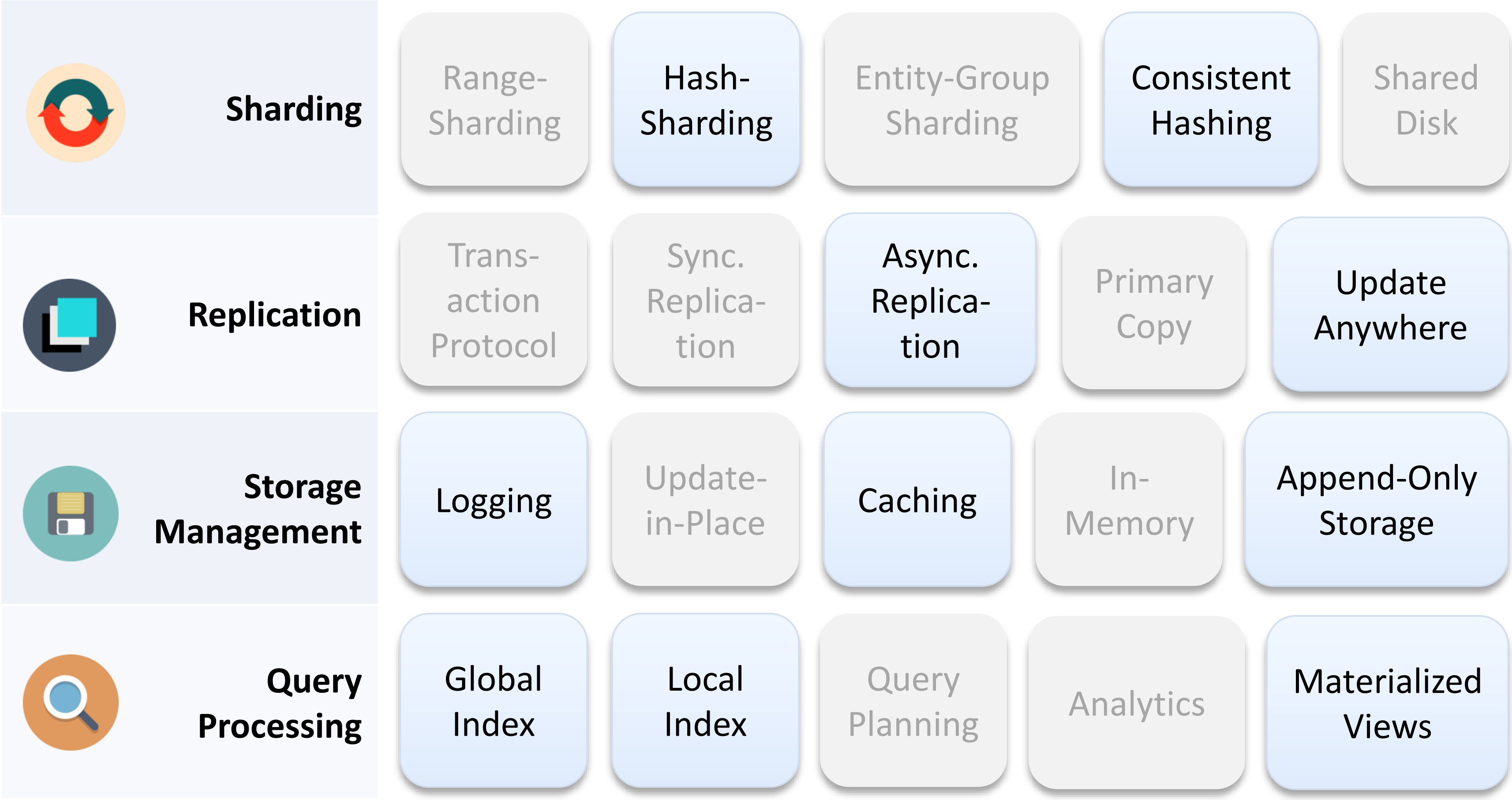
- Published 2007 by Facebook
- **Idea**:
  - BigTable‘s wide-column data model
  - Dynamo ring for replication and sharding
- Cassandra Query Language (CQL): SQL-like query- and DDL-language
- **Compound indices**: *partition key* (shard key) + *clustering key* (ordered per partition key) → Limited range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

Architecture



Classification: Cassandra

Techniques



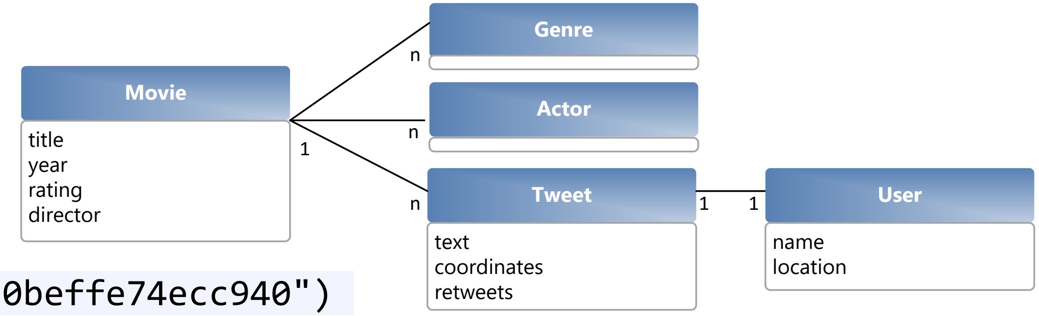
Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Transaction Protocol	Sync. Replication	Async. Replication	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	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

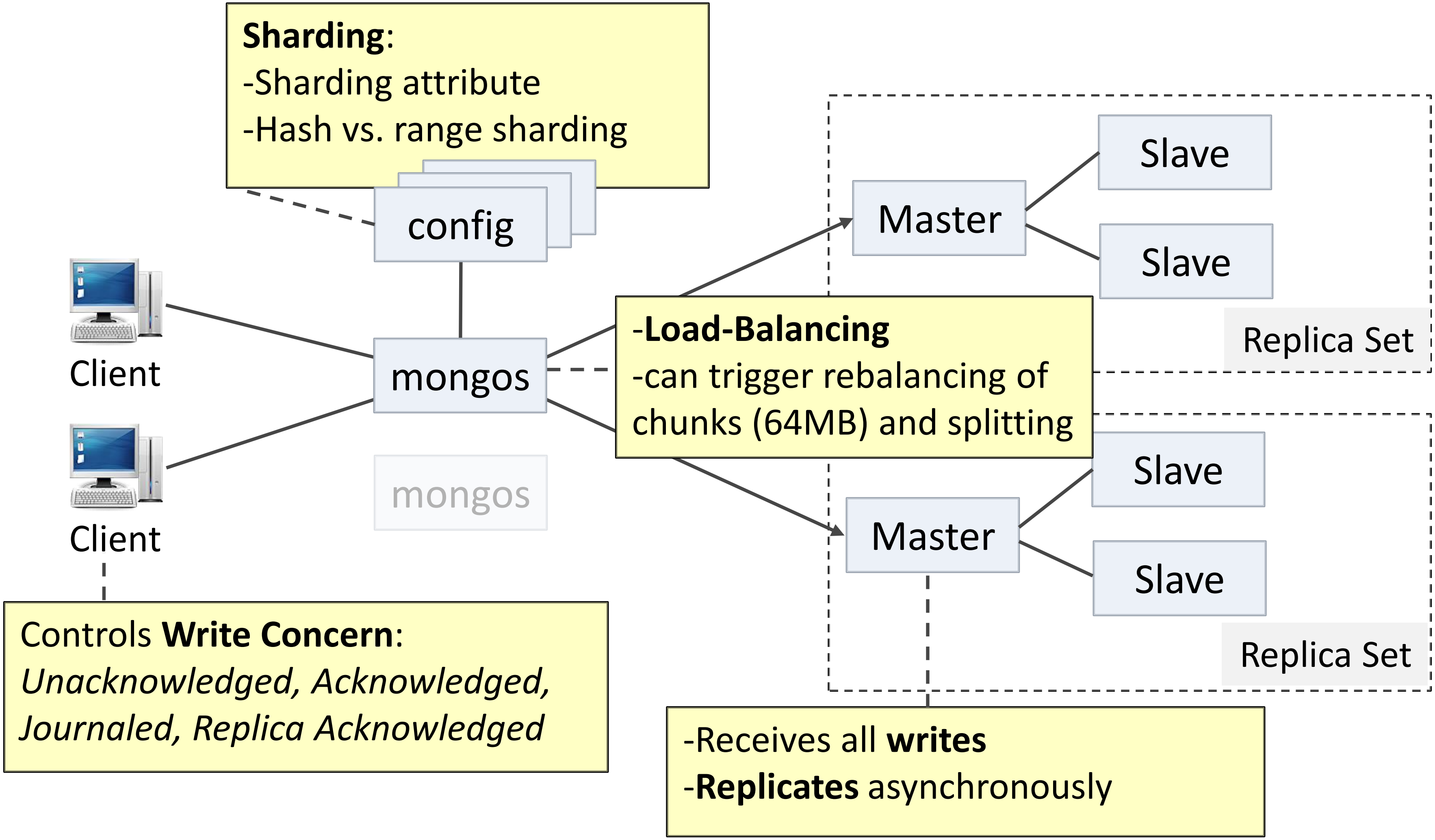
Nesting replaces 1:n and 1:1 relations

Schemafreeness: Attributes per document

Unit of atomicity: document

Principles

Sharding und Replication



Sharding:
-Sharding attribute
-Hash vs. range sharding

config

Client

Client

mongos

mongos

Master

Slave

Slave

Replica Set

-**Load-Balancing**
-can trigger rebalancing of chunks (64MB) and splitting

Master

Slave

Slave

Replica Set

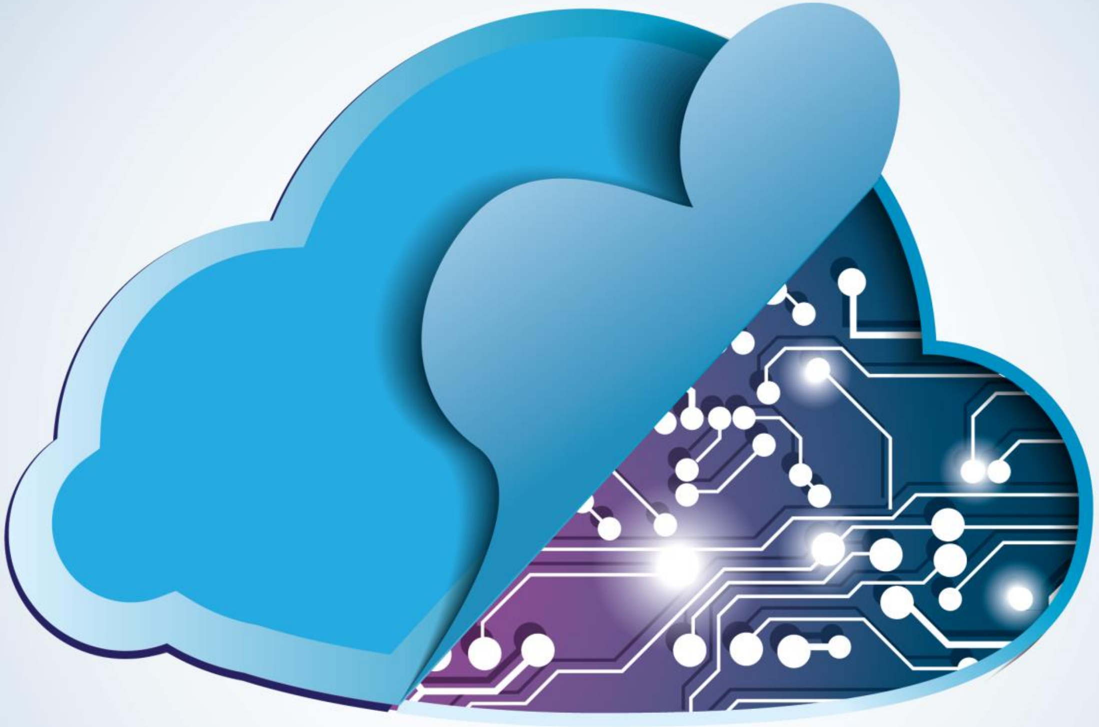
Controls **Write Concern**:
Unacknowledged, Acknowledged, Journaled, Replica Acknowledged

-Receives all **writes**
-**Replicates** asynchronously

Classification: MongoDB

Techniques

Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Transaction Protocol	Sync. Replication	Async. Replication	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

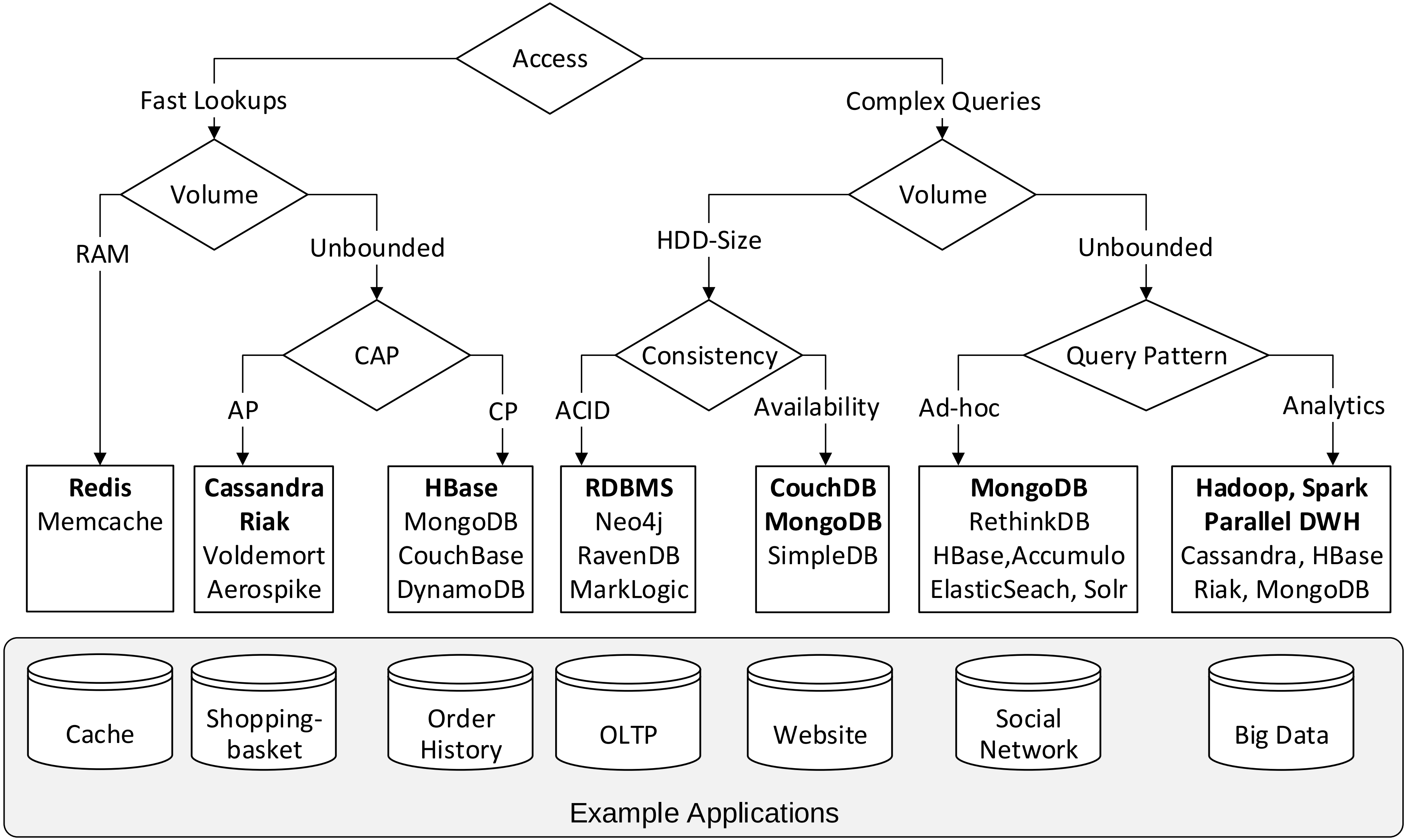
The NoSQL Toolbox: Common Techniques

NoSQL Systems

Decision Guidance: NoSQL Decision Tree

- Decision Tree
- Classification Summary
- Literature Reommendations

NoSQL Decision Tree



Access

Fast Lookups

Complex Queries

Volume

RAM

Unbounded

CAP

AP

CP

Volume

HDD-Size

Unbounded

Consistency

ACID

Availability

Query Pattern

Ad-hoc

Analytics

Redis
Memcache

Cassandra
Riak
Voldemort
Aerospike

HBase
MongoDB
CouchBase
DynamoDB

RDBMS
Neo4j
RavenDB
MarkLogic

CouchDB
MongoDB
SimpleDB

MongoDB
RethinkDB
HBase,Accumulo
ElasticSeach, Solr

Hadoop, Spark
Parallel DWH
Cassandra, HBase
Riak, MongoDB

Cache

Shopping-basket

Order History

OLTP

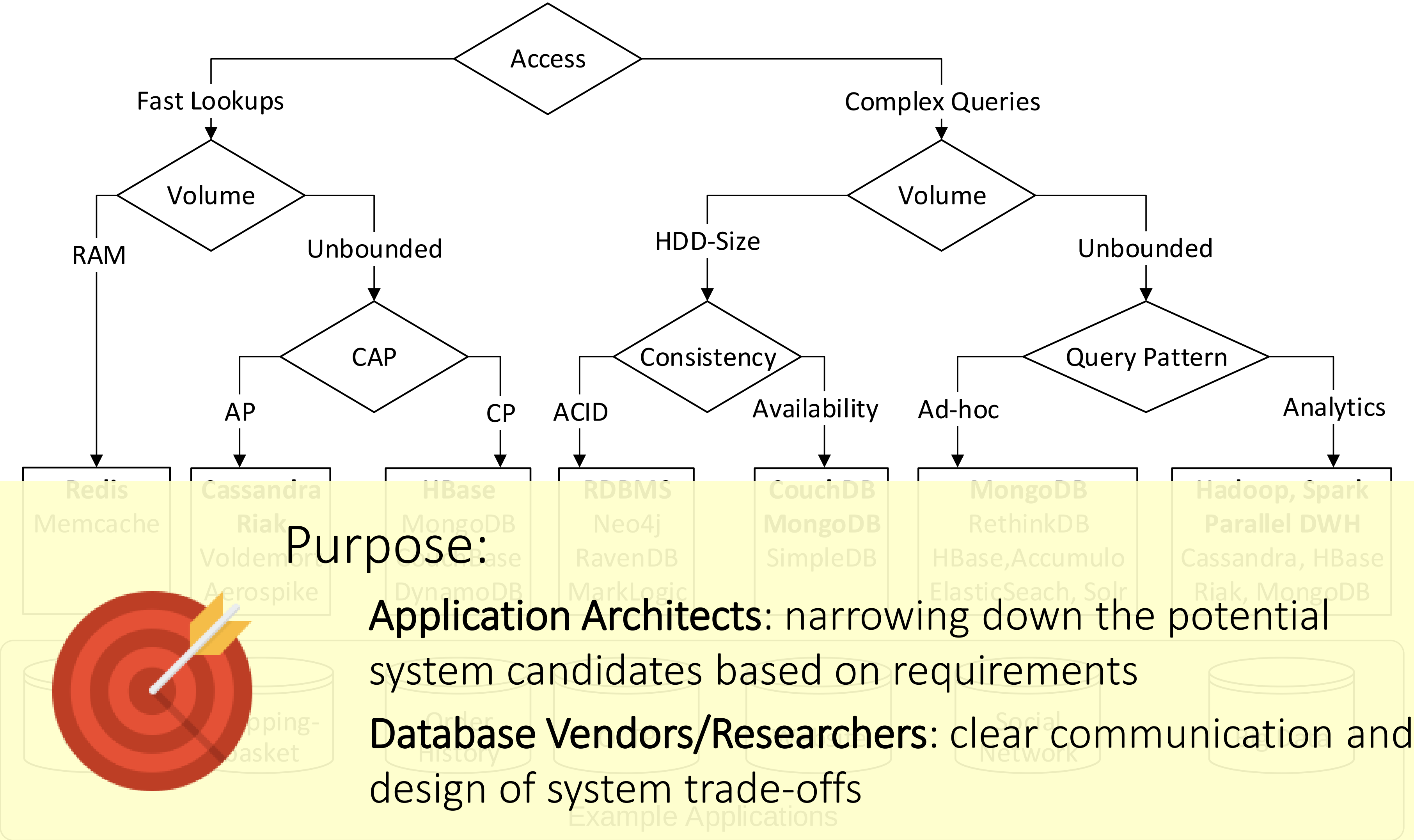
Website

Social Network

Big Data

Example Applications

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		

Future Work

Online Collaborative Decision Support

- Select **Requirements** in Web GUI:

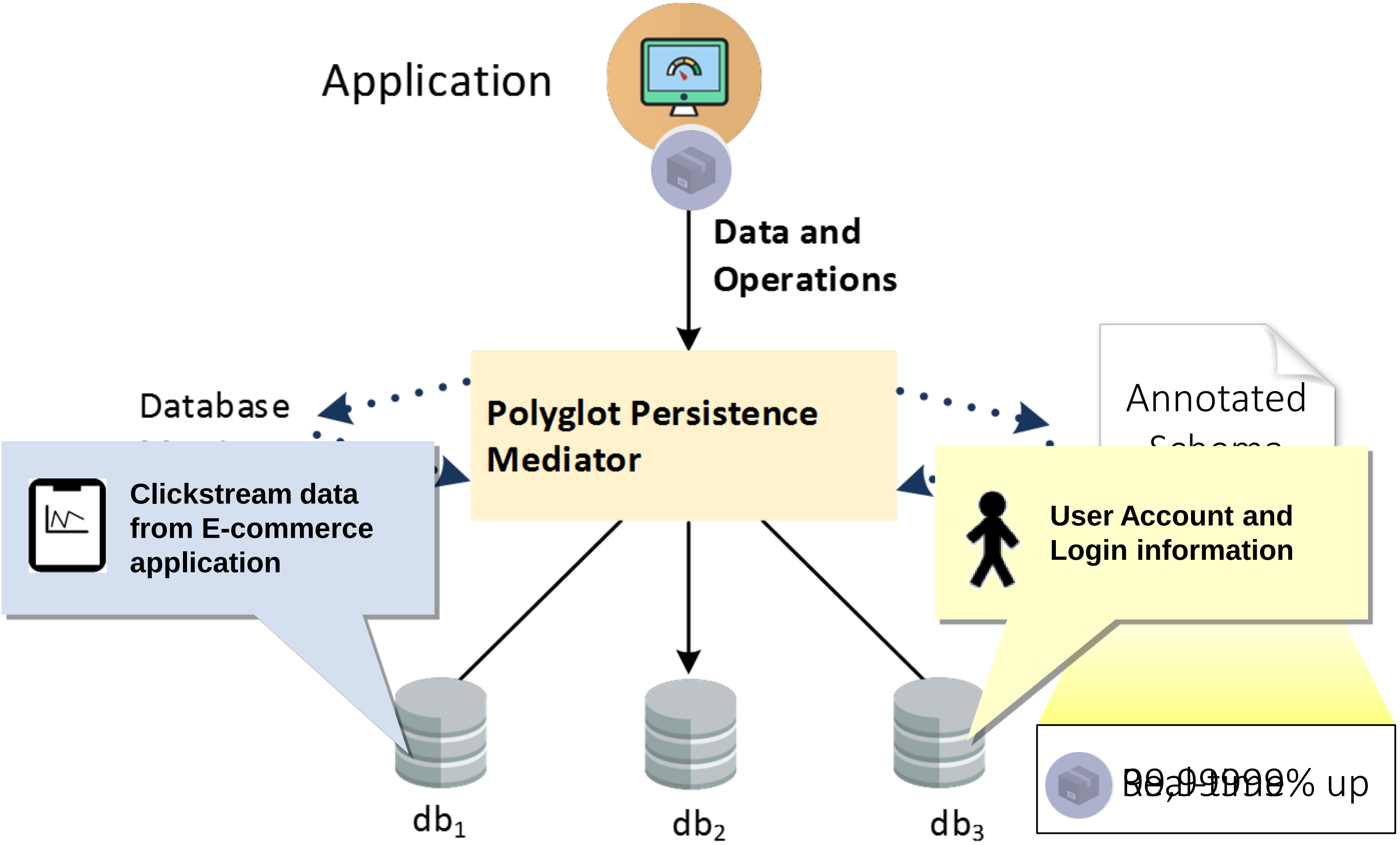
□ Read Scalability ☑ Conditional Writes ☑ Consistent

- System makes **suggestions** based on data from *practitioners*, *vendors* and *automated benchmarks*:

★		★	
4/5 4/5 3/5	redis	4/5 5/5 5/5	mongoDB

Future Work

Polyglot Persistence Mediator


Application
Data and Operations
Polyglot Persistence Mediator
Database
Annotated Schema
Clickstream data from E-commerce application
User Account and Login information
Real-time
99,99999% up
db₁
db₂
db₃

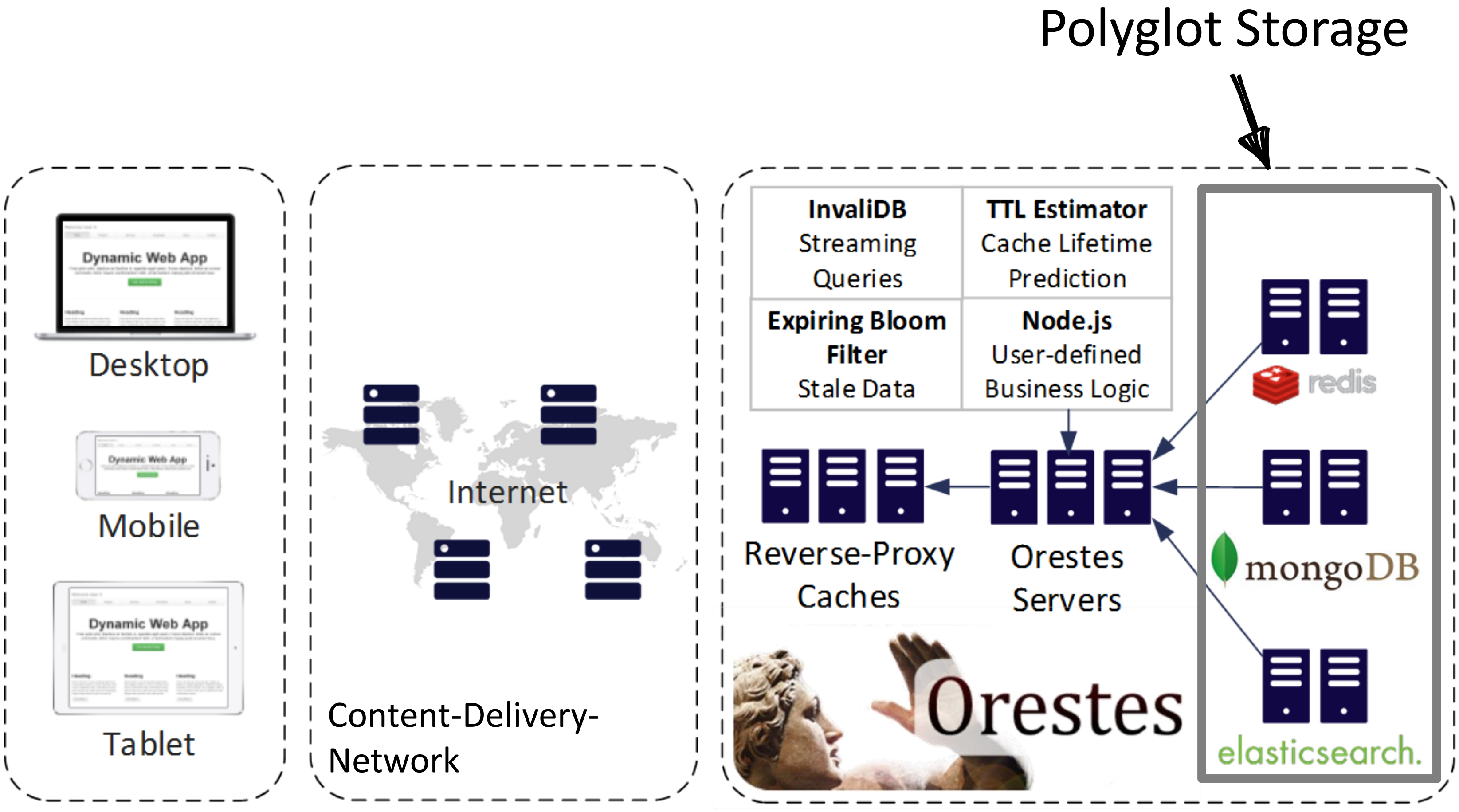
BaQend

Build faster Apps faster.

www.baqend.com

Approach: API as a Superset

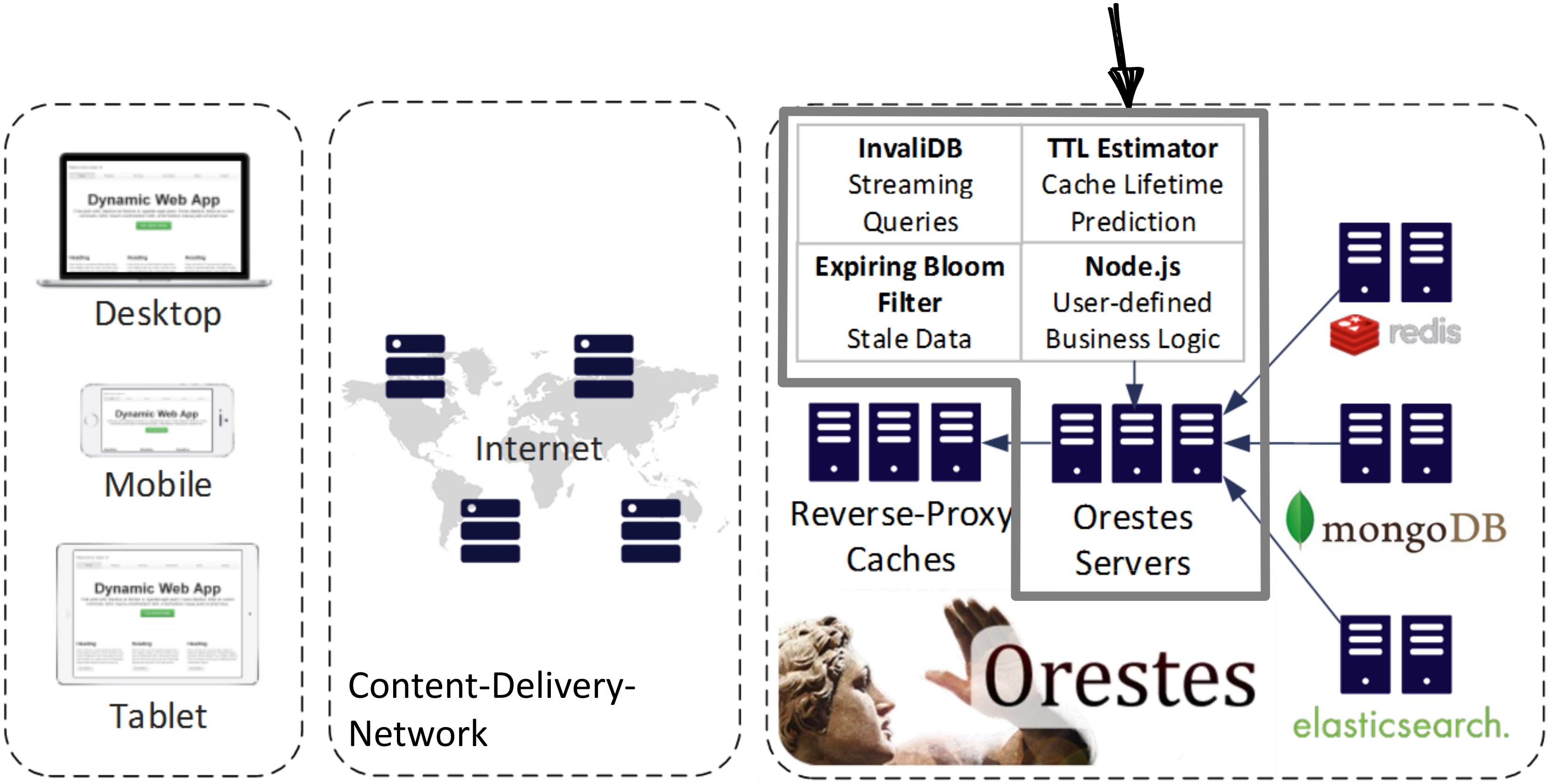
For Web-Apps and Mobile



Approach: API as a Superset

For Web-Apps and Mobile

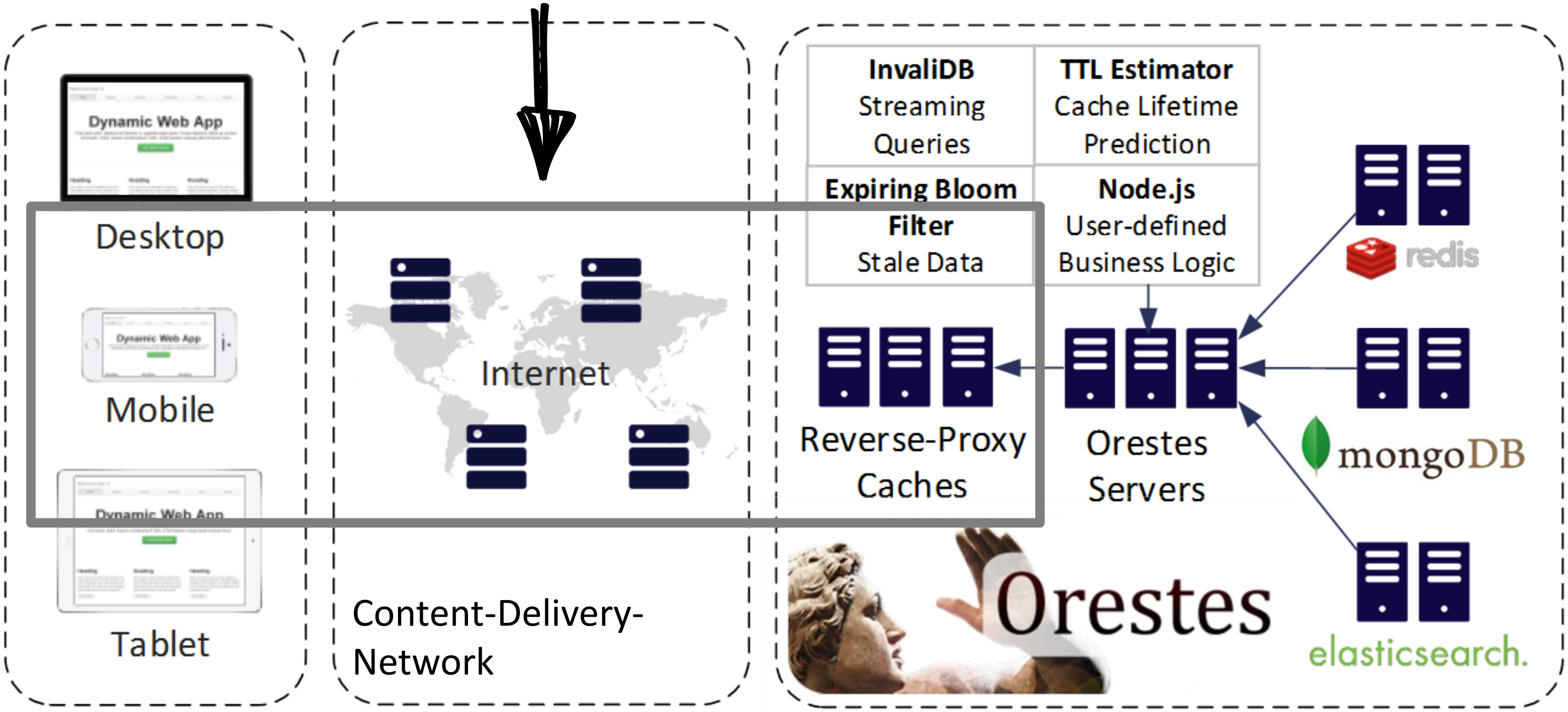
Backend-as-a-Service Middleware:
Caching, Transactions, Schemas,
Invalidation Detection, …



Approach: API as a Superset

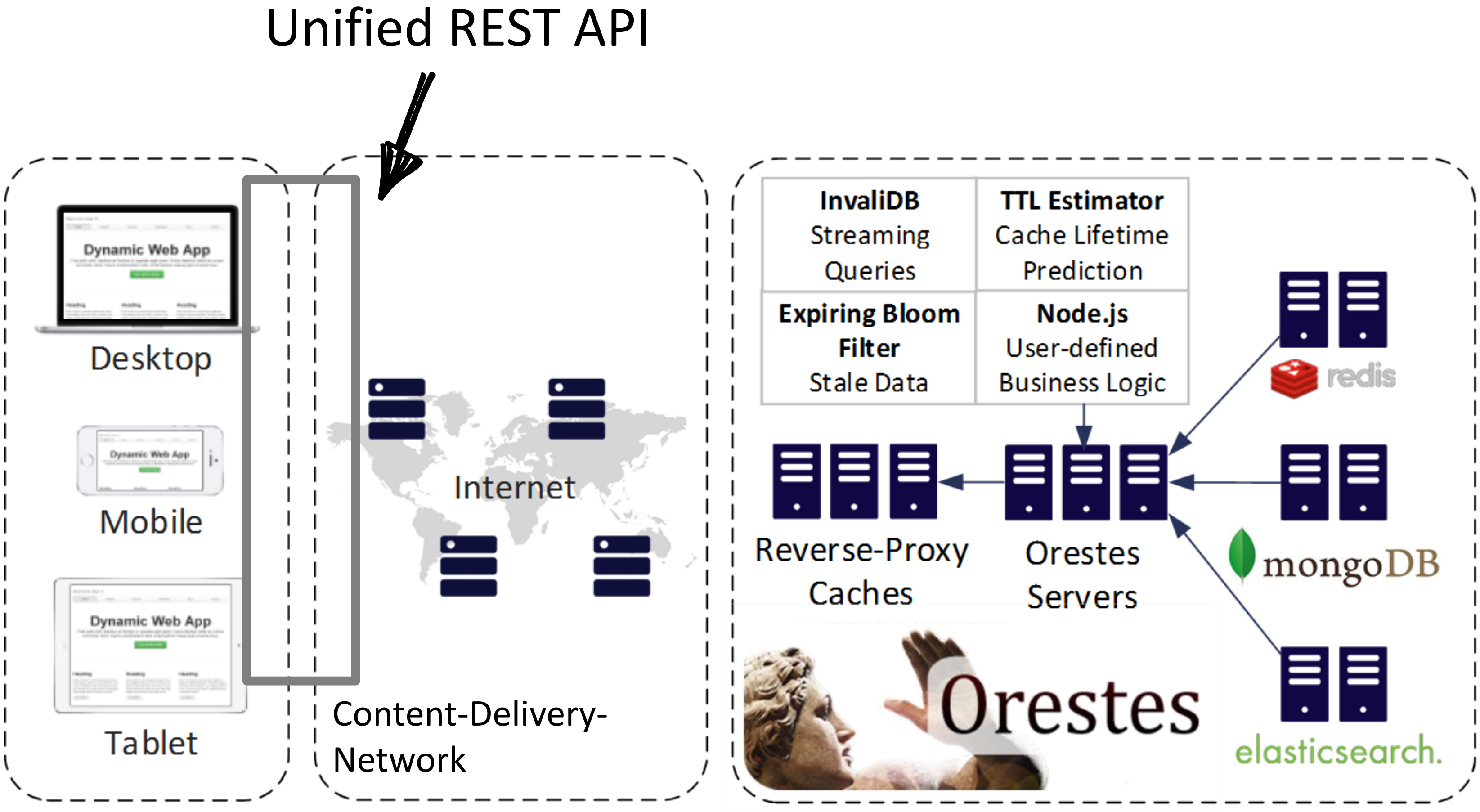
For Web-Apps and Mobile

Standard HTTP Caching



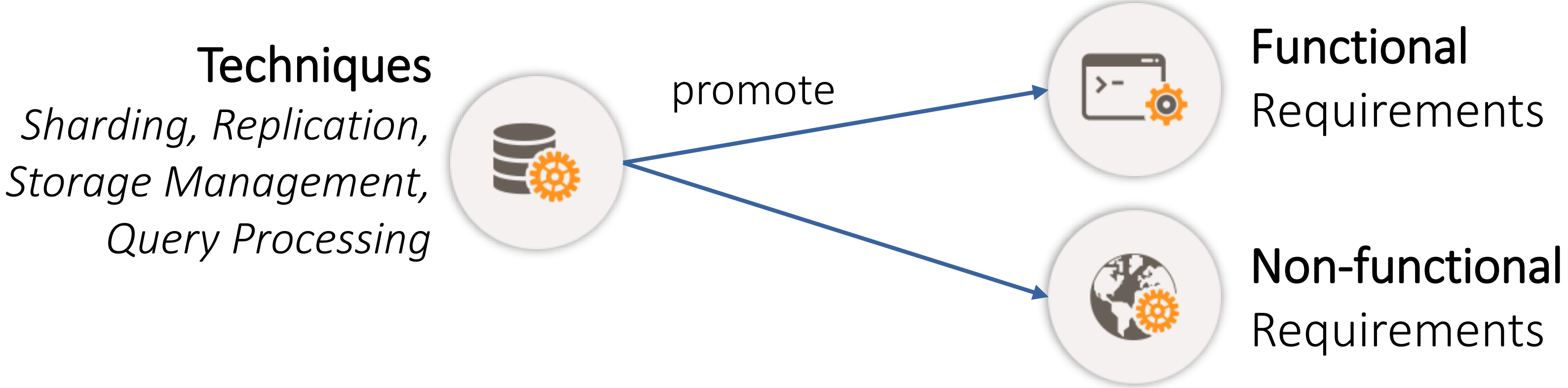
Approach: API as a Superset

For Web-Apps and Mobile



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



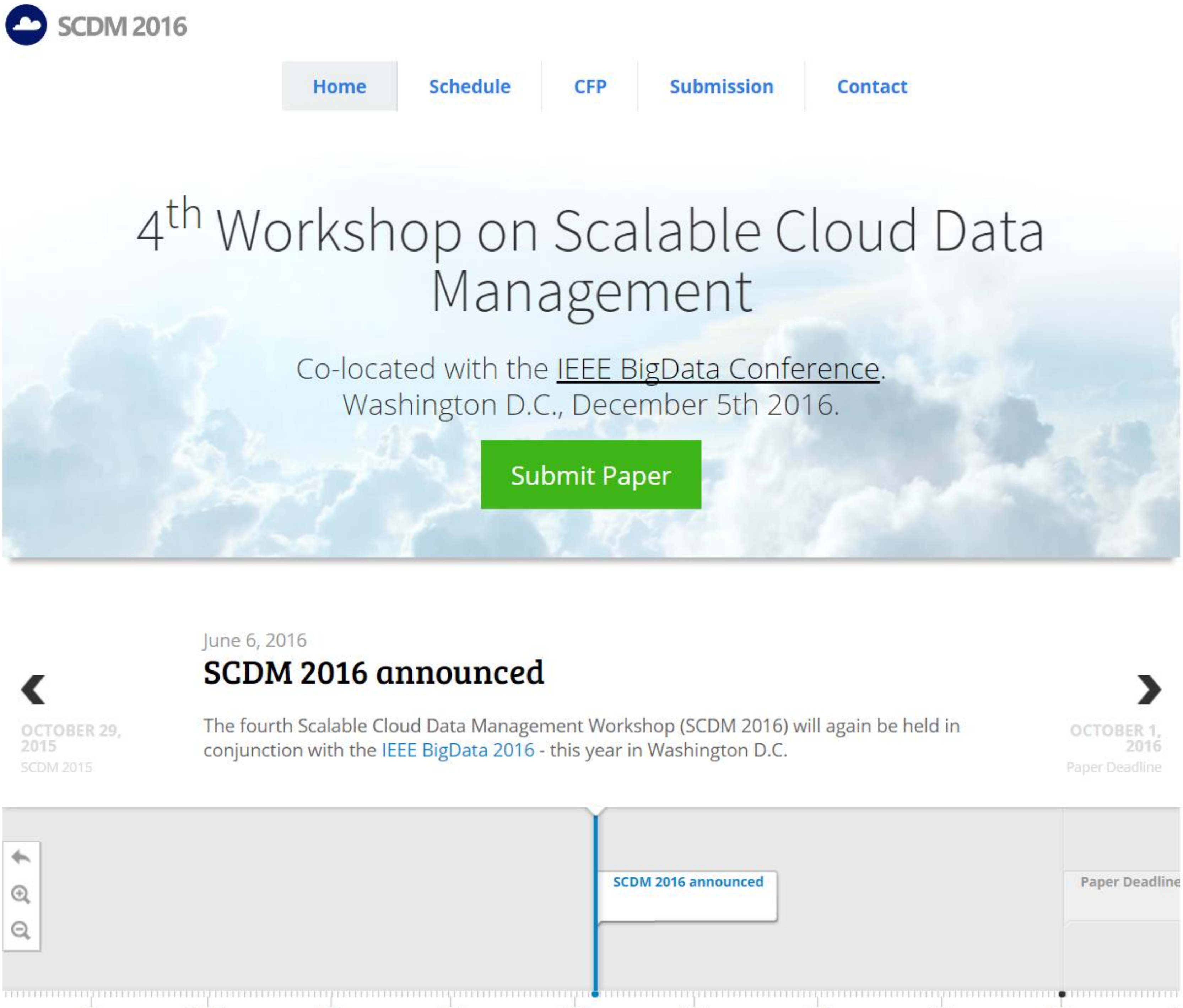
Techniques
Sharding, Replication, Storage Management, Query Processing

promote

Functional Requirements

Non-functional Requirements

- **Decision Tree**



4th Workshop on Scalable Cloud Data Management

Co-located with the IEEE BigData Conference.
Washington D.C., December 5th 2016.

June 6, 2016

SCDM 2016 announced

The fourth Scalable Cloud Data Management Workshop (SCDM 2016) will again be held in conjunction with the IEEE BigData 2016 - this year in Washington D.C.

Thank you!

gessert@informatik.uni-hamburg.de

www.baqend.com
www.scdm2016.com
vsis-www.informatik.uni-hamburg.de