Scalable Data Management

An In-Depth Tutorial on NoSQL Data Stores

Felix Gessert, Wolfram Wingerath, Norbert Ritter

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

7. März, BTW 2017, Stuttgart

Slides: slideshare.net/felixgessert

Article: medium.com/baqend-blog

Outline

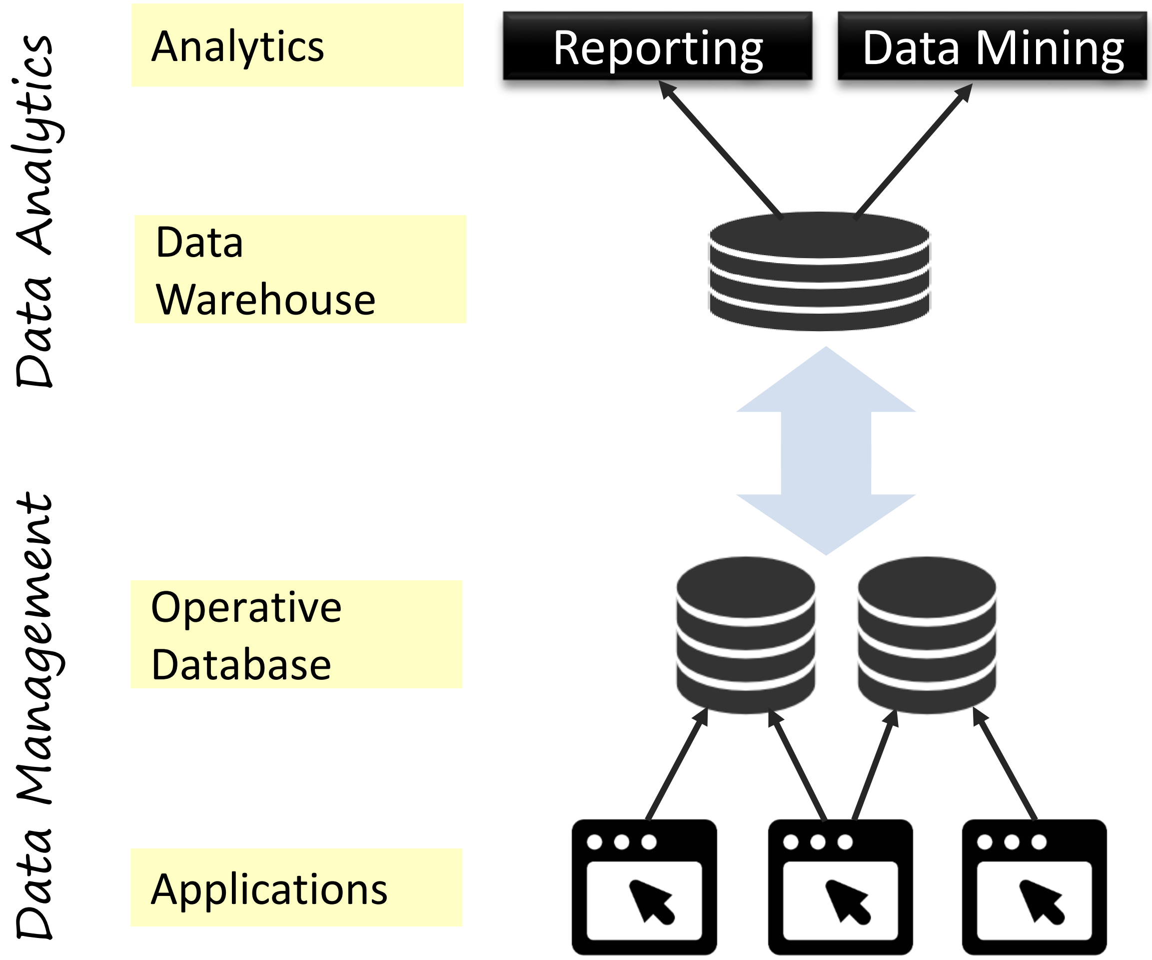
- NoSQL Foundations and Motivation
- The NoSQL Toolbox: Common Techniques
- NoSQL Systems & Decision Guidance
- Scalable Real-Time Databases and Processing

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

Introduction: What are NoSQL data stores?

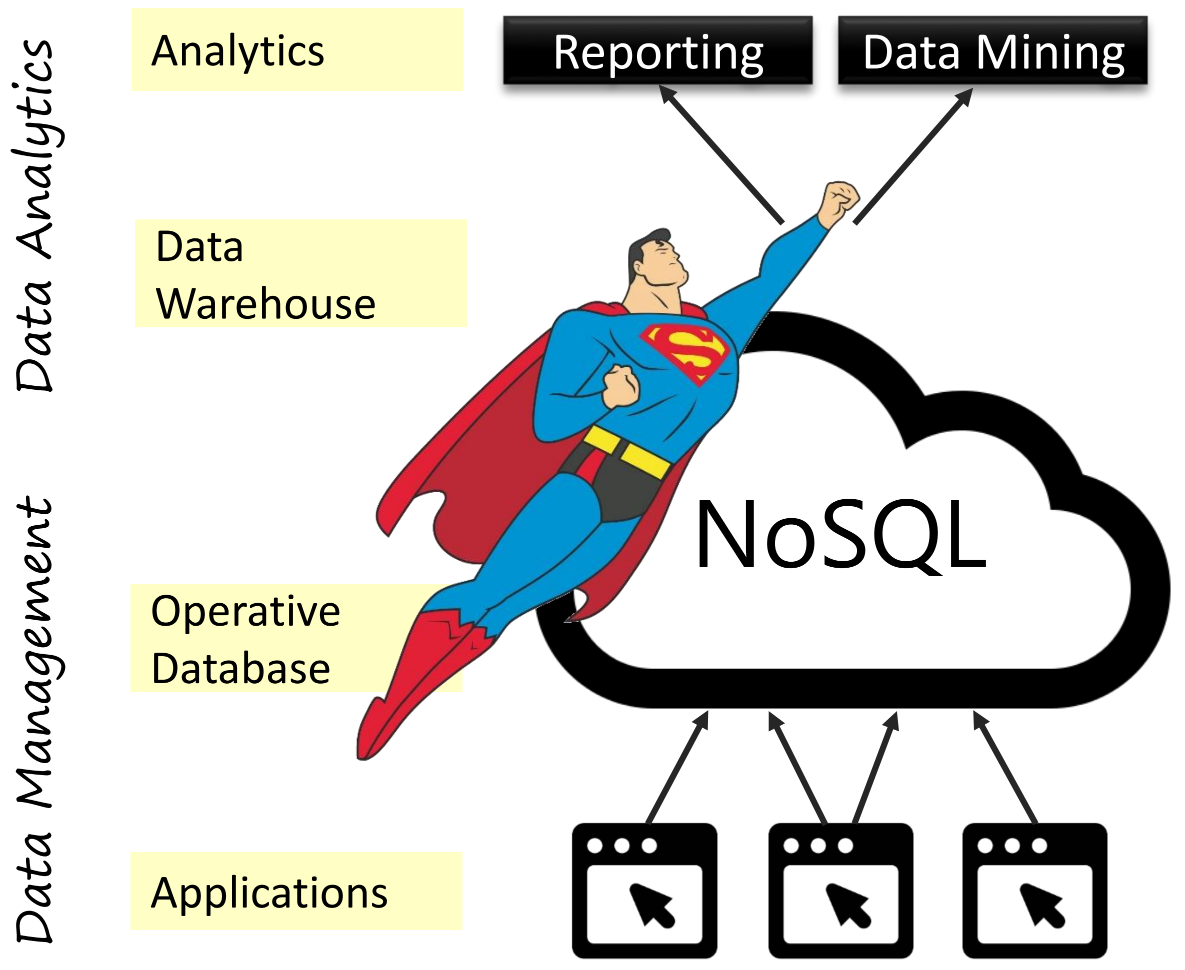
Architecture

Typical Data Architecture:



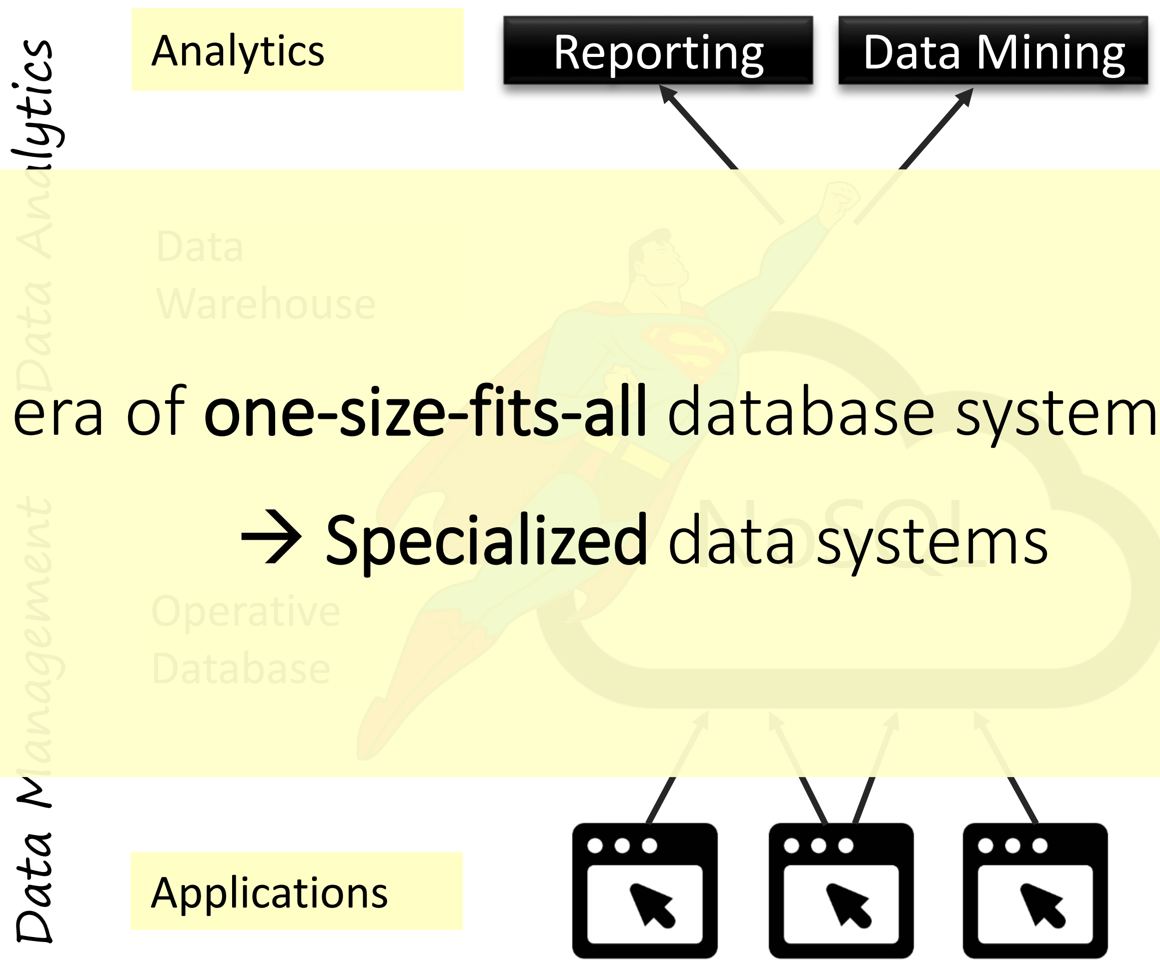
Architecture

Typical Data Architecture:



Architecture

Typical Data Architecture:



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

→ **Specialized** data systems

The Database Explosion

Sweetspots

RDBMS

General-purpose ACID transactions

Parallel DWH

Aggregations/OLAP for massive data amounts

NewSQL

High throughput relational OLTP

Wide-Column Store

Long scans over structured data

Document Store

Deeply nested data models

Key-Value Store

Large-scale session storage

Graph Database

Graph algorithms & queries

In-Memory KV-Store

Counting & statistics

Wide-Column Store

Massive user-generated content

The Database Explosion

Cloud-Database Sweetspots

Firebase

Realtime BaaS

Communication and collaboration

Amazon RDS

Managed RDBMS

General-purpose ACID transactions

Amazon ElastiCache

Managed Cache

Caching and transient storage

Azure Tables

Wide-Column Store

Very large tables

Amazon DynamoDB

Wide-Column Store

Massive user-generated content

Parse

Backend-as-a-Service

Small Websites and Apps

bonsai

Managed NoSQL

Full-Text Search

Google Cloud Storage

Object Store

Massive File Storage

Amazon Elastic MapReduce

Hadoop-as-a-Service

Big Data Analytics

How to choose a database system?

Many Potential Candidates

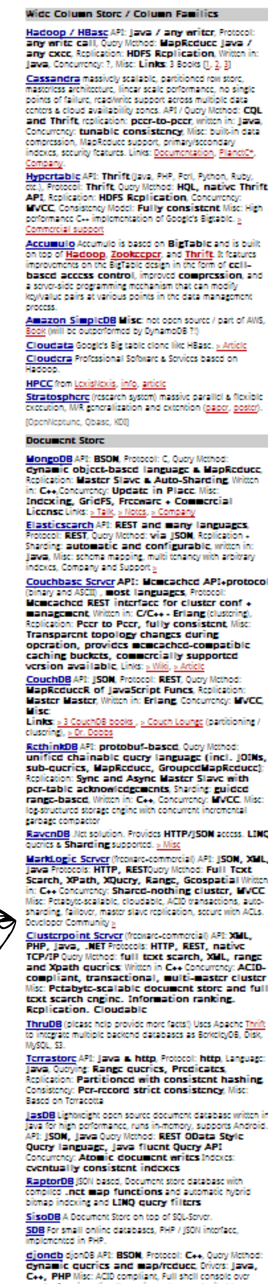
Question in this tutorial:

How to approach the 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

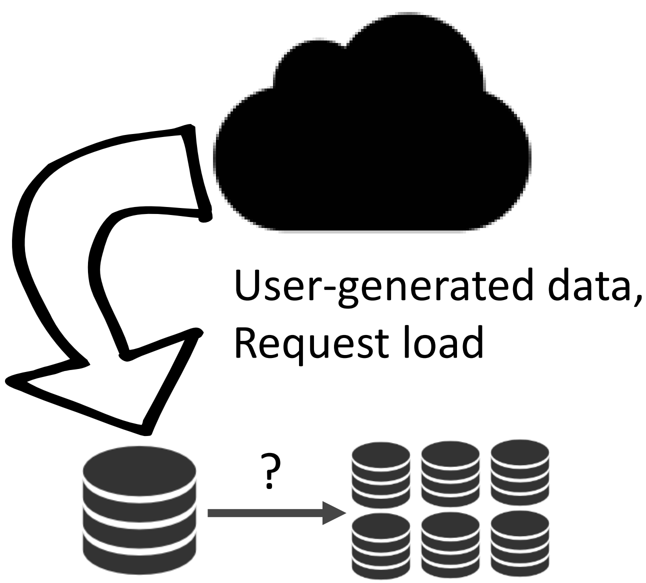
***NoSQL-Databases.org*:**
Current list has over 150 NoSQL systems


Wide Column Store / Column Families
Hadoop / HBase
Cassandra
Hypertable
Accumulo
Amazon SimpleDB
Cloudata
Cloudera
HPCC
Stratosphere
Document Store
MongoDB
Elasticsearch
Couchbase Server
CouchDB
RethinkDB
RavenDB
MarkLogic Server
Clusterpoint Server
ThruDB
Terrastore
JasDB
RaptorDB
SisoDB
SDB
djondb

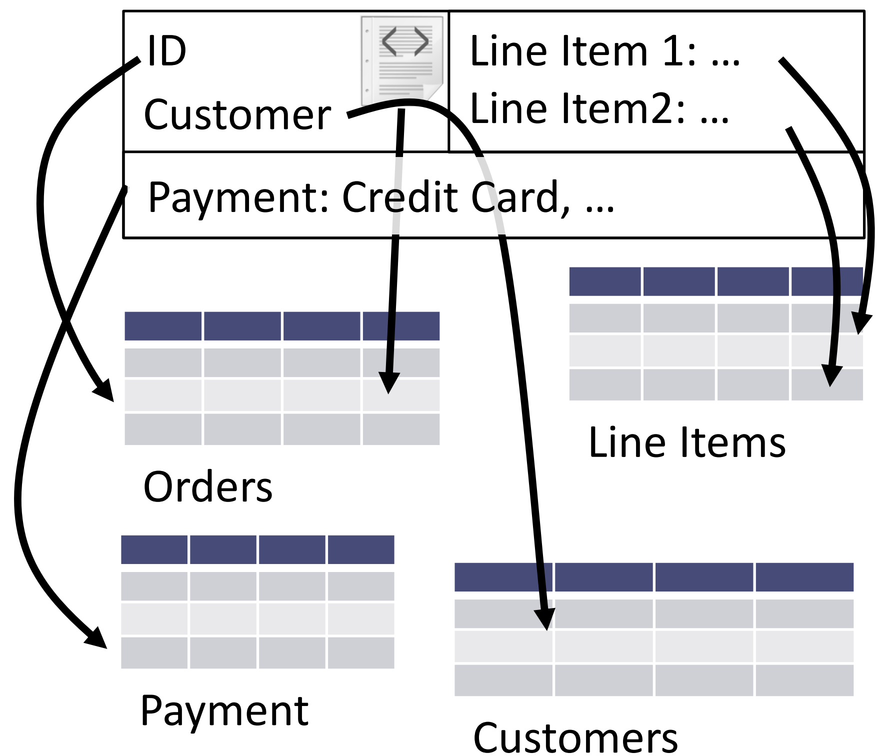
NoSQL Databases

- Two main motivations:

Scalability

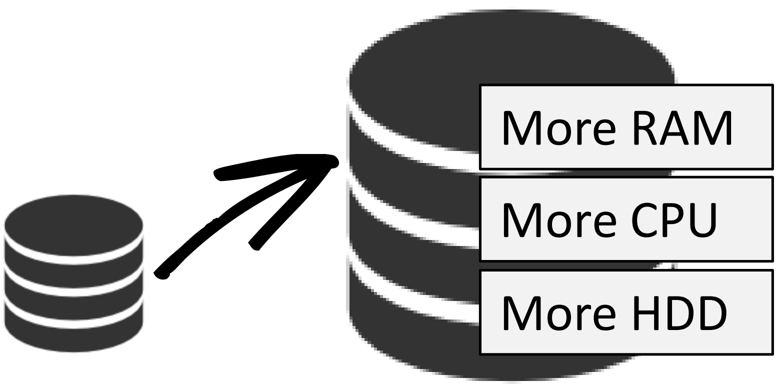


Impedance Mismatch

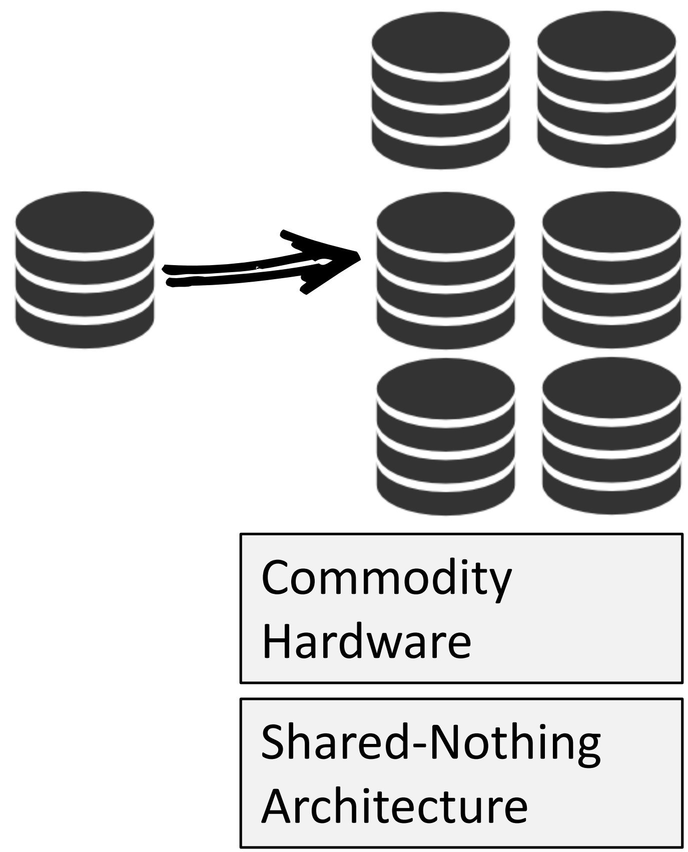


Scale-up vs Scale-out

Scale-Up (*vertical* scaling):

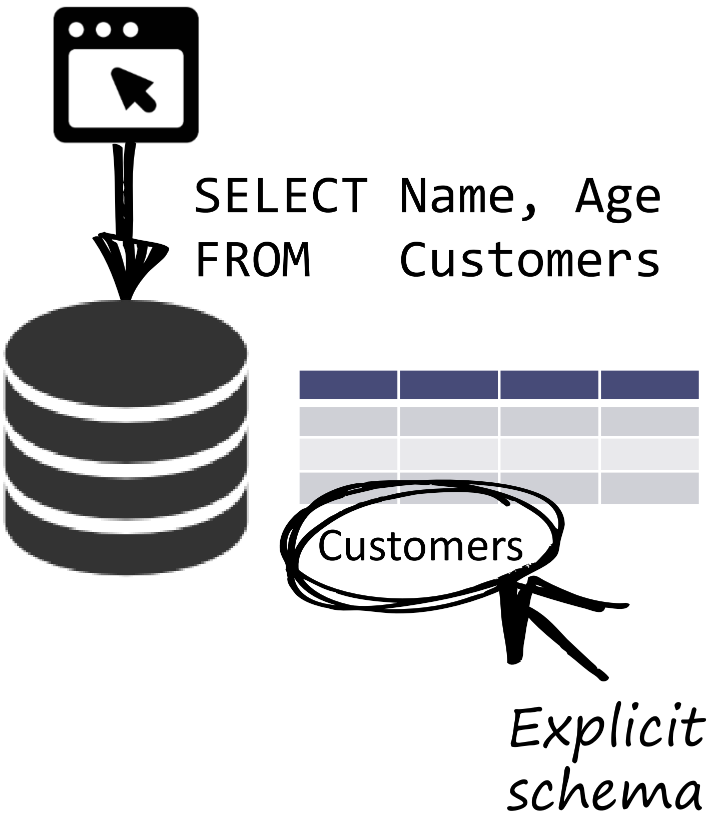


Scale-Out (*horizontal* scaling):

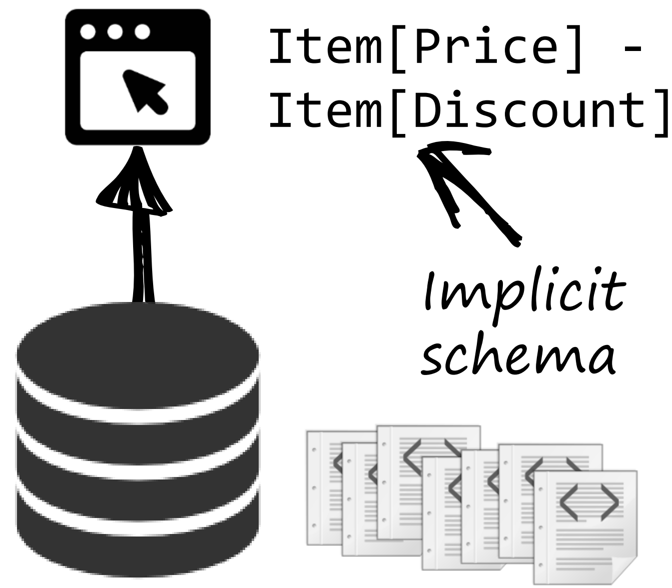


Schemafree Data Modeling

RDBMS:



NoSQL DB:



Big Data

The Analytic side of NoSQL

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

Sources

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

Analyst, Data Scientist,
Software Developer

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

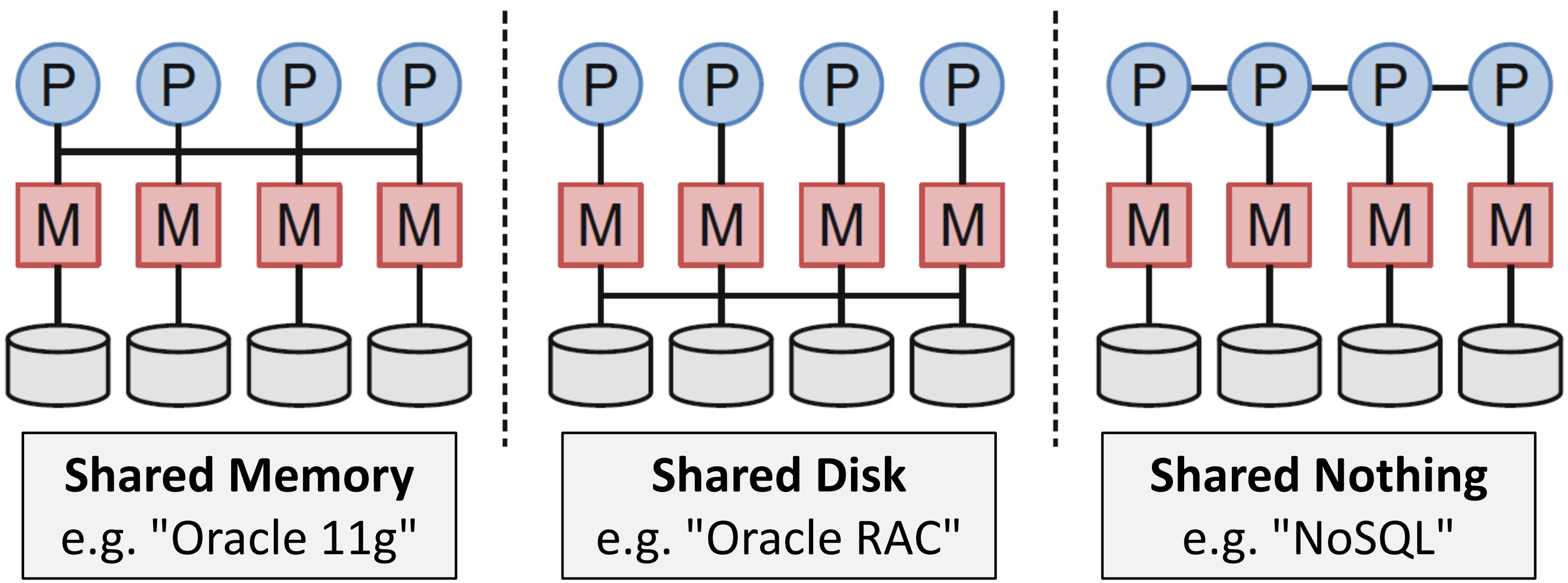
NoSQL Paradigm Shift

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)

NoSQL Paradigm Shift
Shared Nothing Architectures

Shift towards higher distribution & less coordination:



NoSQL System Classification

- Two common criteria:

Data Model

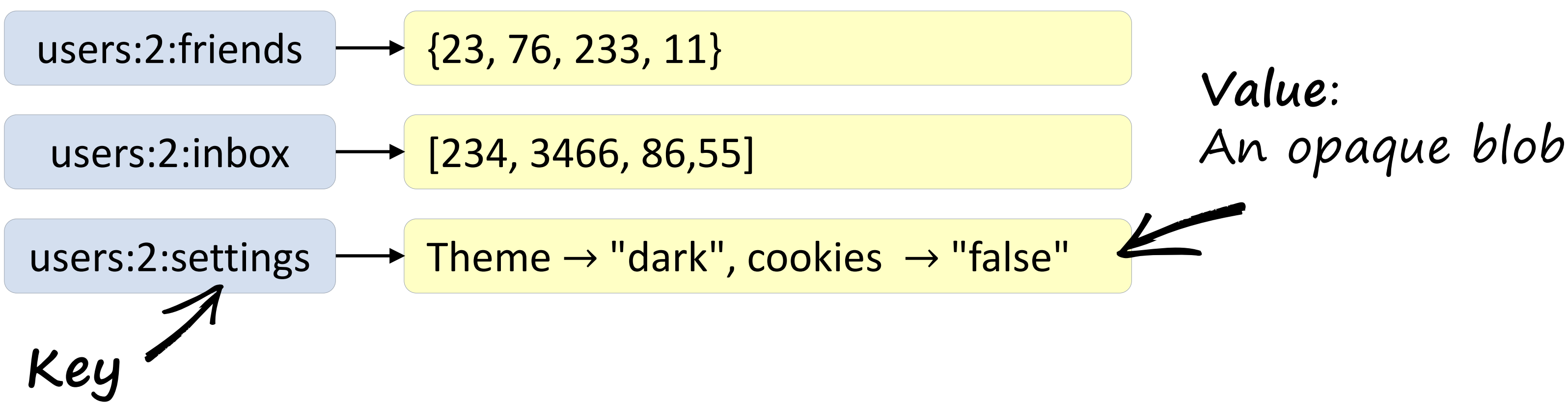
- Key-Value
- Wide-Column
- Document
- Graph

Consistency/Availability Trade-Off

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

Key-Value Stores

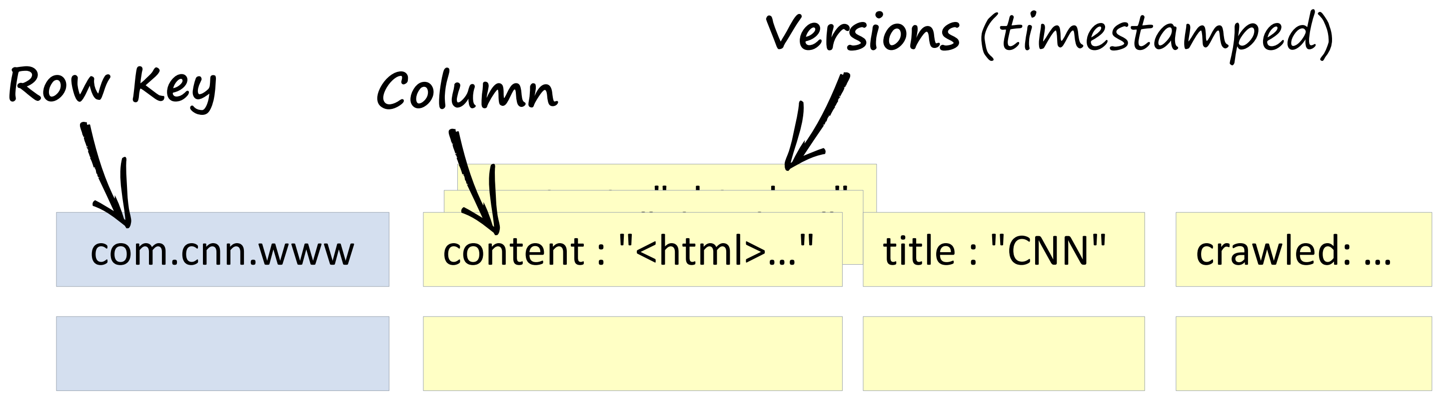
- **Data model:** (key) -> value
- **Interface**: CRUD (Create, Read, Update, Delete)



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

Wide-Column Stores

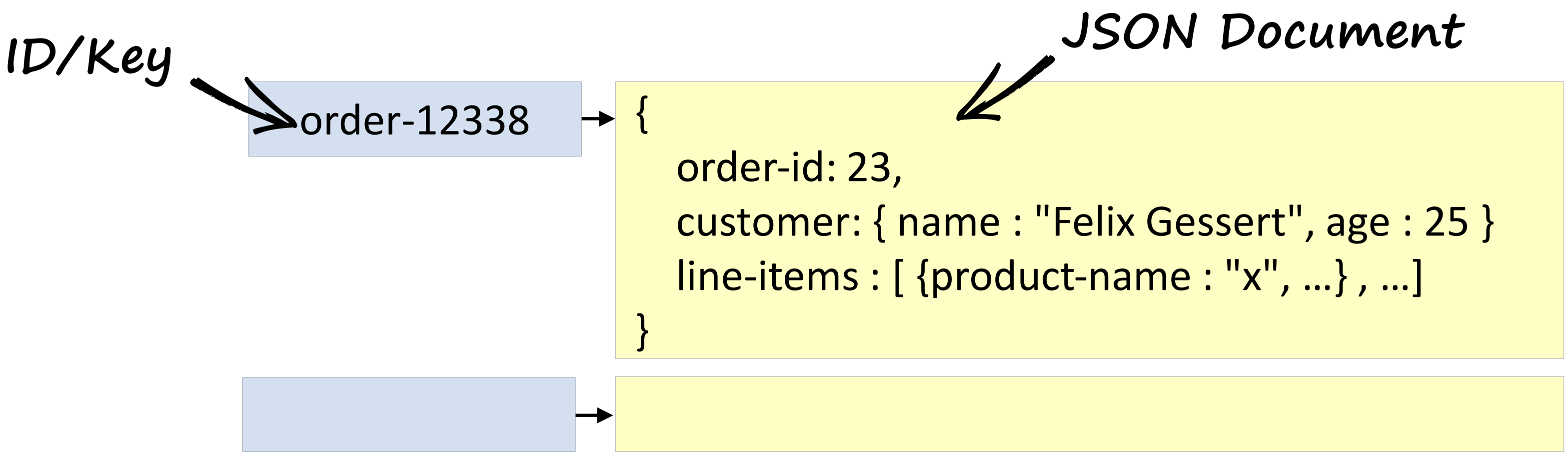
- **Data model:** (rowkey, column, timestamp) -> value
- **Interface**: CRUD, Scan



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

Document Stores

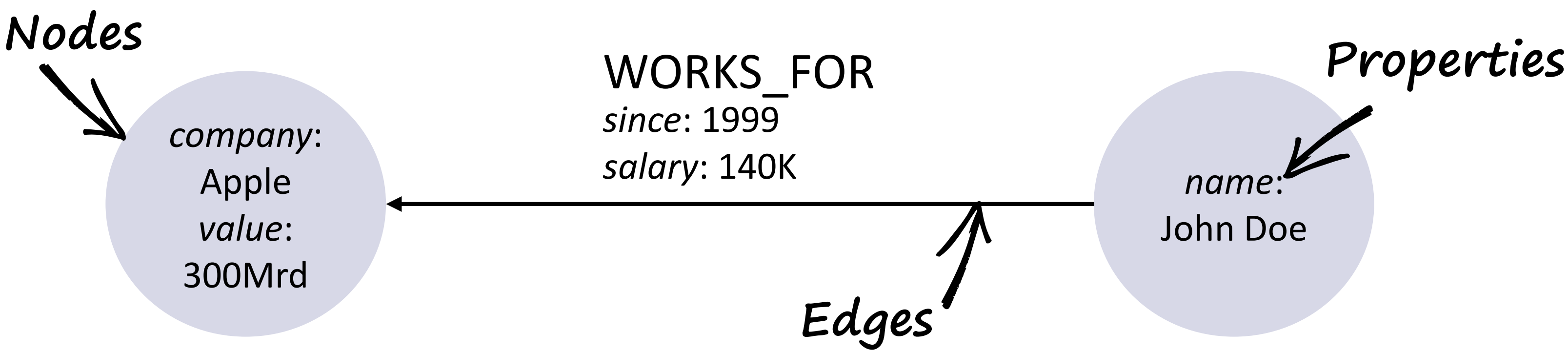
- **Data model:** (collection, key) -> document
- **Interface**: CRUD, Querys, Map-Reduce



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

Graph Databases

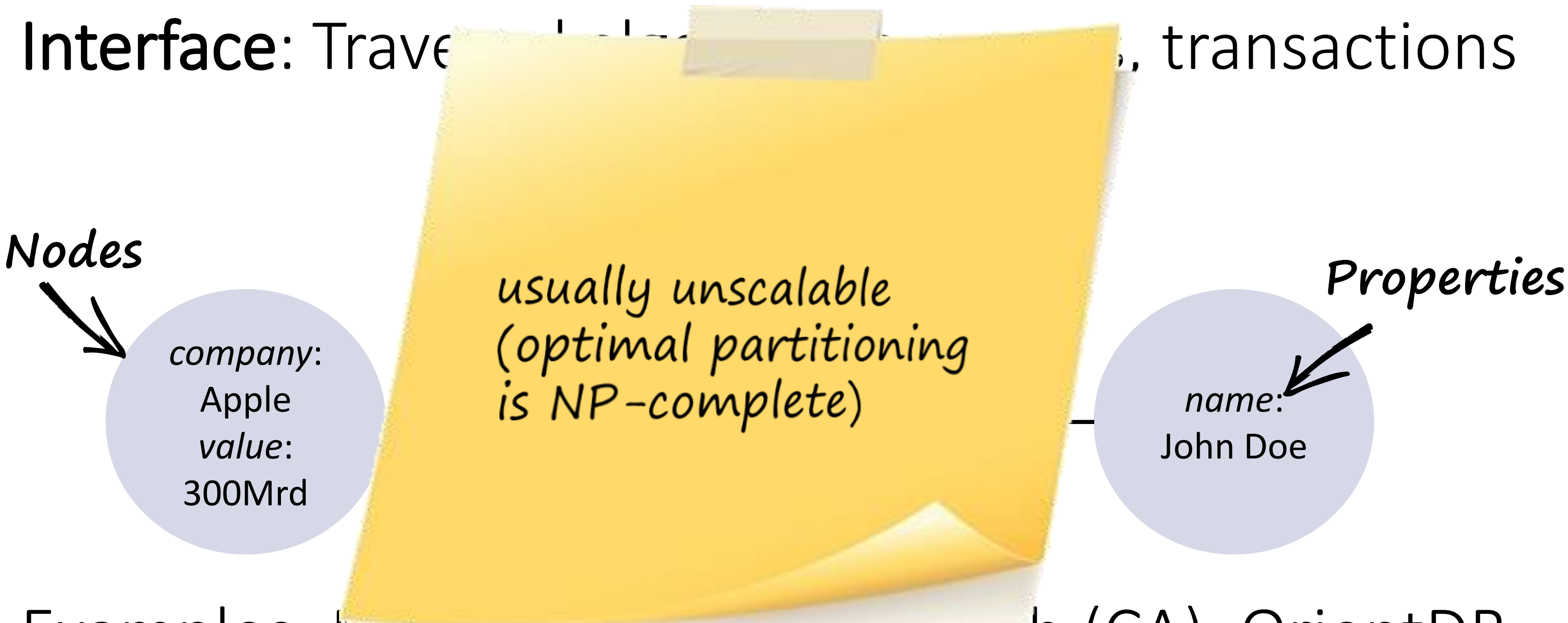
- **Data model:** G = (V, E): Graph-Property Modell
- **Interface**: Traversal algorithms, querys, transactions



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

Graph Databases

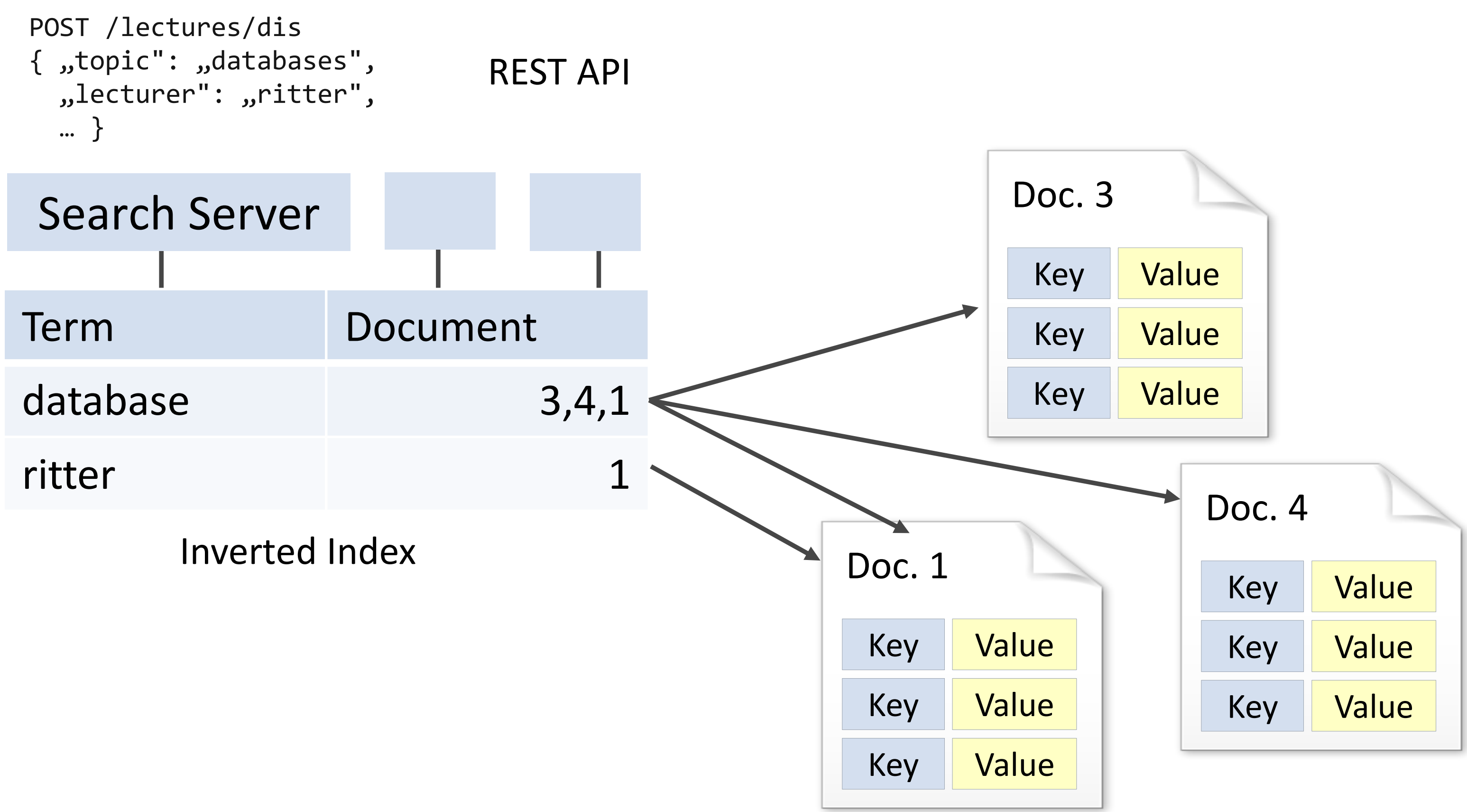
- **Data model:** G = (V, E): Graph-Property Modell
- **Interface**: Trave… , transactions



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

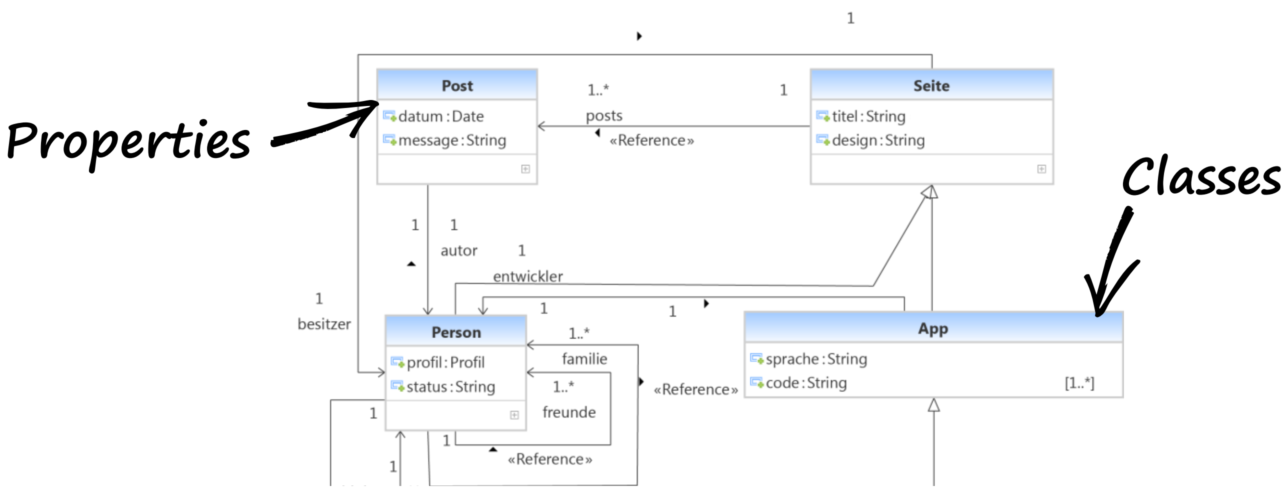
Search Platforms

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



Object-oriented Databases

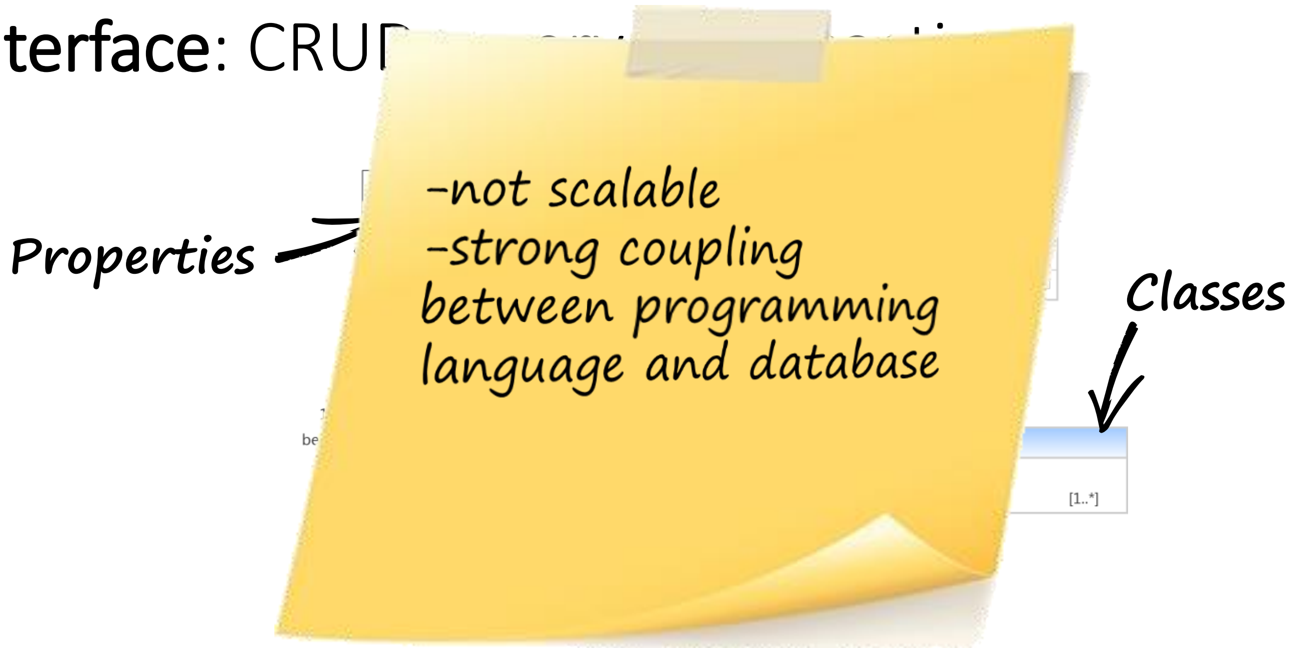
- **Data model:** Classes, objects, relations (references)
- **Interface**: CRUD, querys, transactions



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

Object-oriented Databases

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



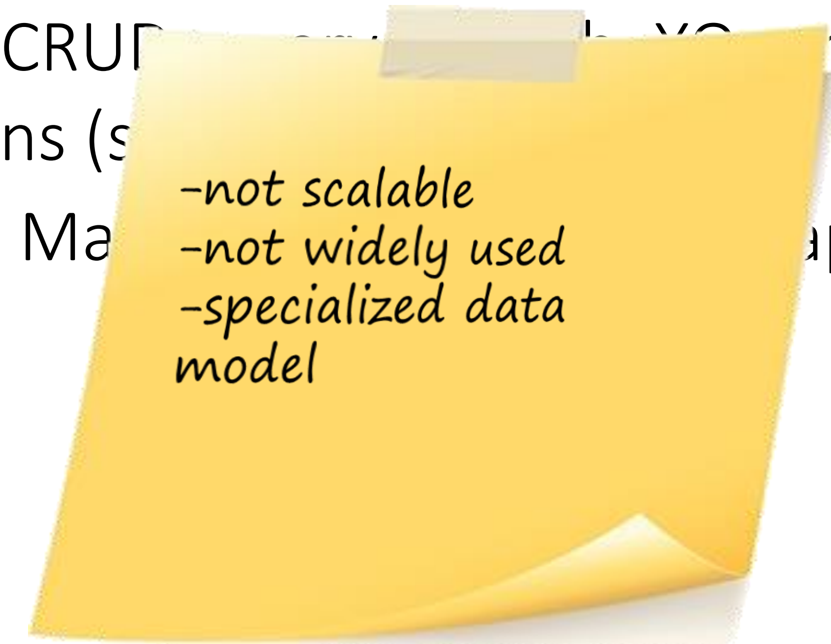
- Examples: Versant (CA), db4o (CA), Objectivity (CA)

XML databases, RDF Stores

- **Data model:** XML, RDF
- **Interface**: CRUD, querys (XPath, XQuerys, SPARQL), transactions (some)
- Examples: MarkLogic (CA), AllegroGraph (CA)

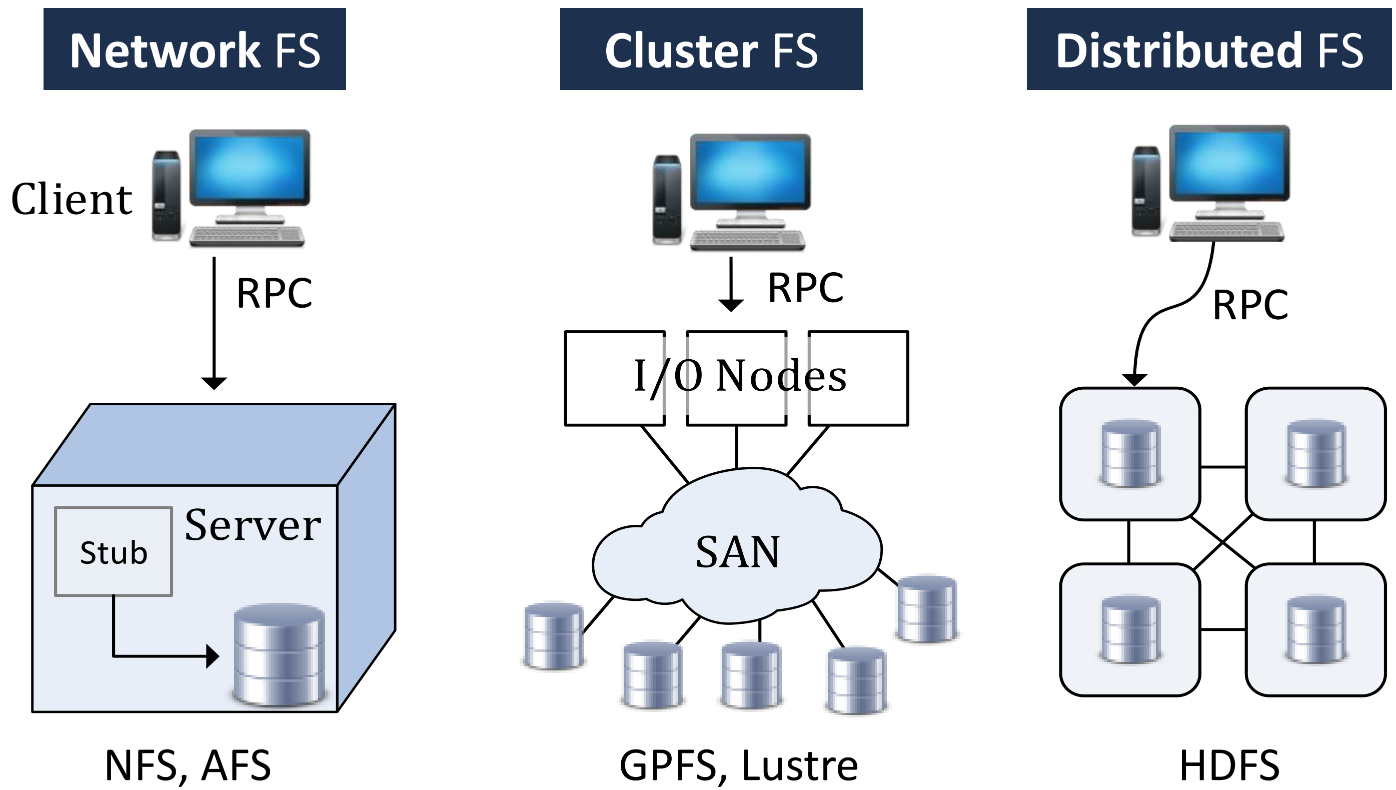
XML databases, RDF Stores

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



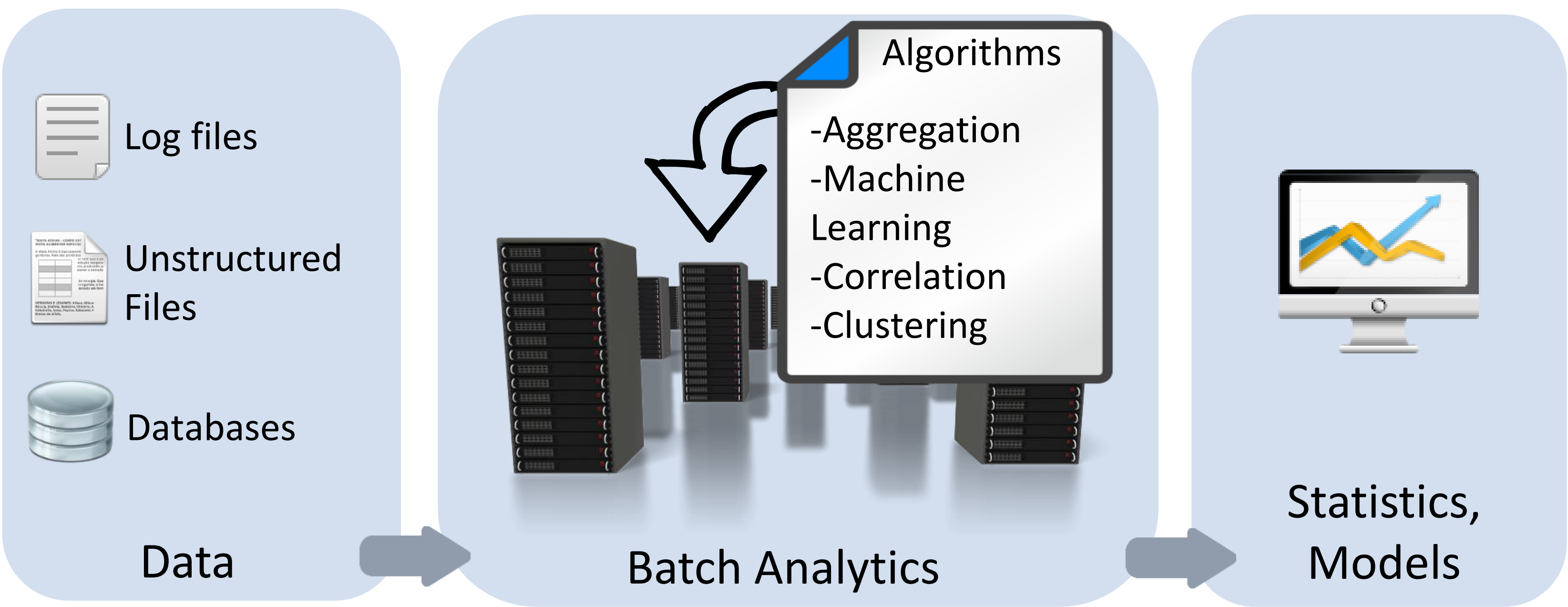
Distributed File System

- **Data model:** files + folders



Big Data Batch Processing

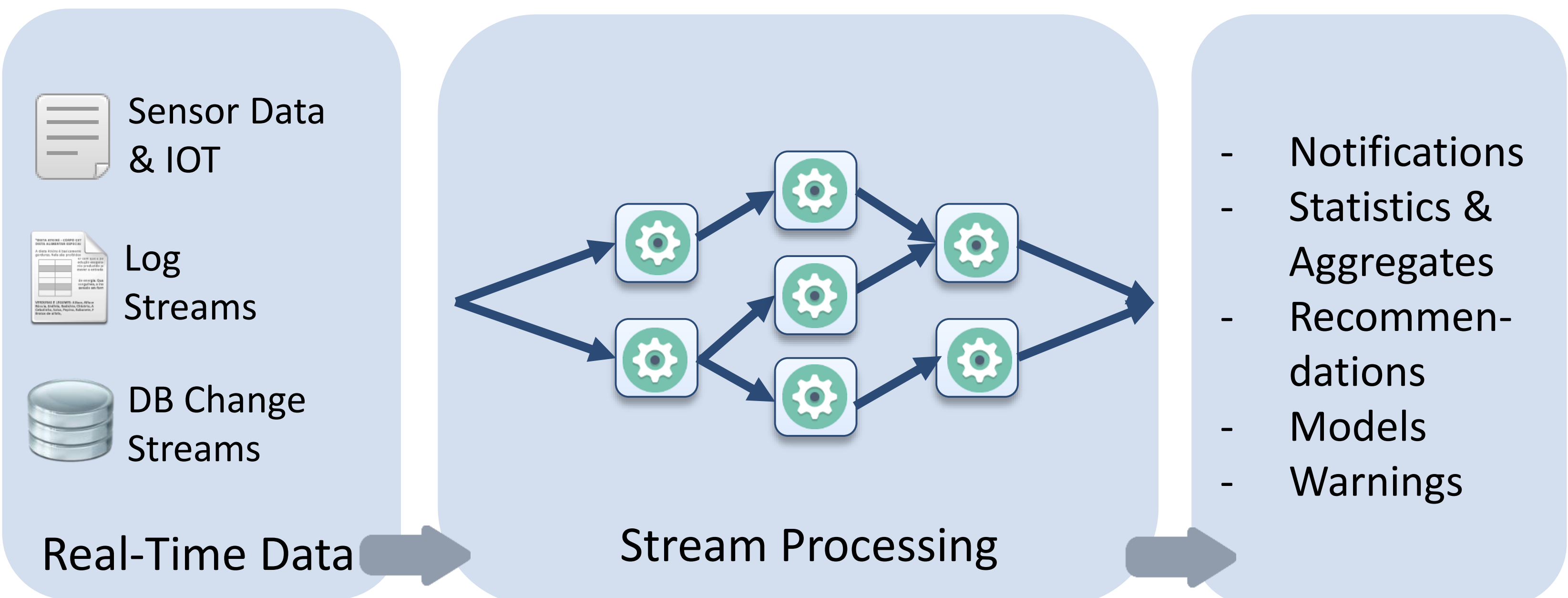
- **Data model:** arbitrary (frequently unstructured)
- Examples: Hadoop, Spark, Flink, DryadLink, Pregel



Big Data Stream Processing

Covered in Depth in the Last Part

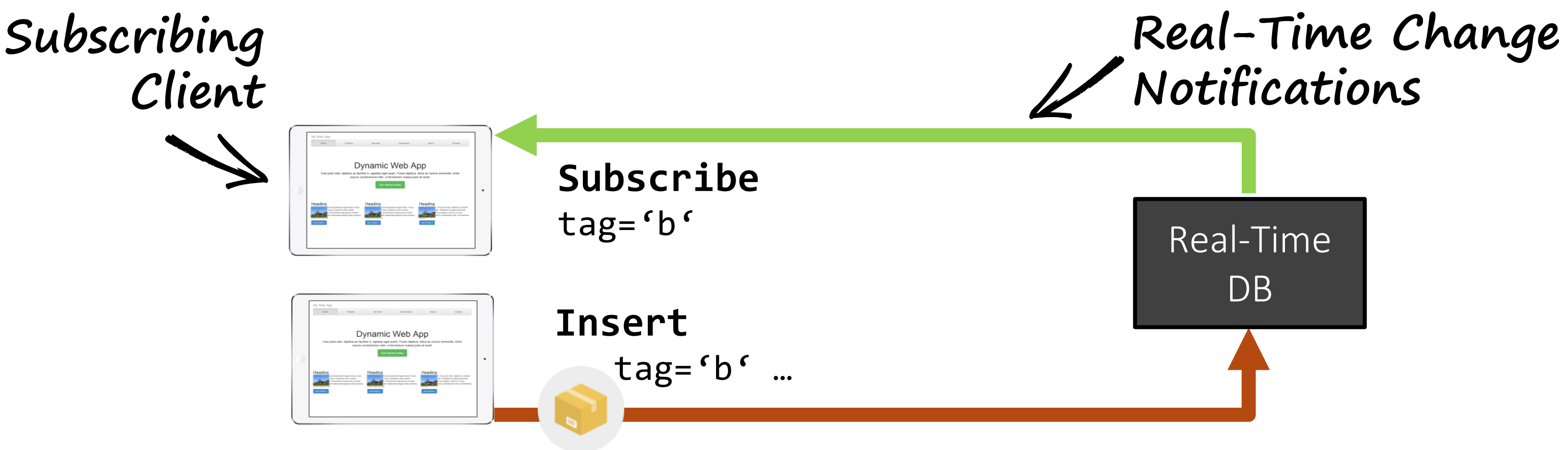
- **Data model:** arbitrary
- Examples: Storm, Samza, Flink, Spark Streaming



Real-Time Databases

Covered in Depth in the Last Part

- **Data model:** several data models possible
- **Interface**: CRUD, Querys + **Continuous Queries**



- Examples: Firebase (CP), Parse (CP), Meteor (CP), Lambda/Kappa Architecture

Soft NoSQL Systems

Not Covered Here

Search Platforms (Full Text Search):

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

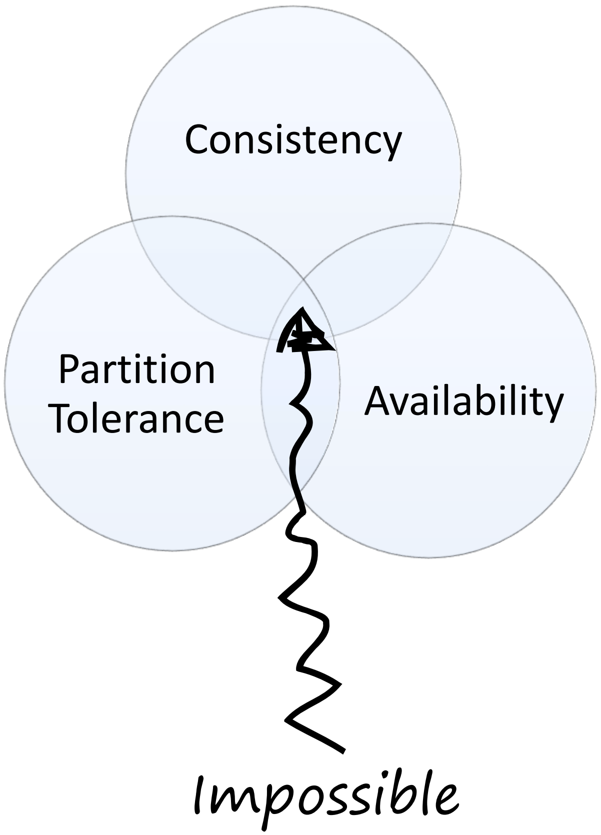
Object-Oriented Databases:

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

XML-Databases, RDF-Stores:

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

CAP-Theorem



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

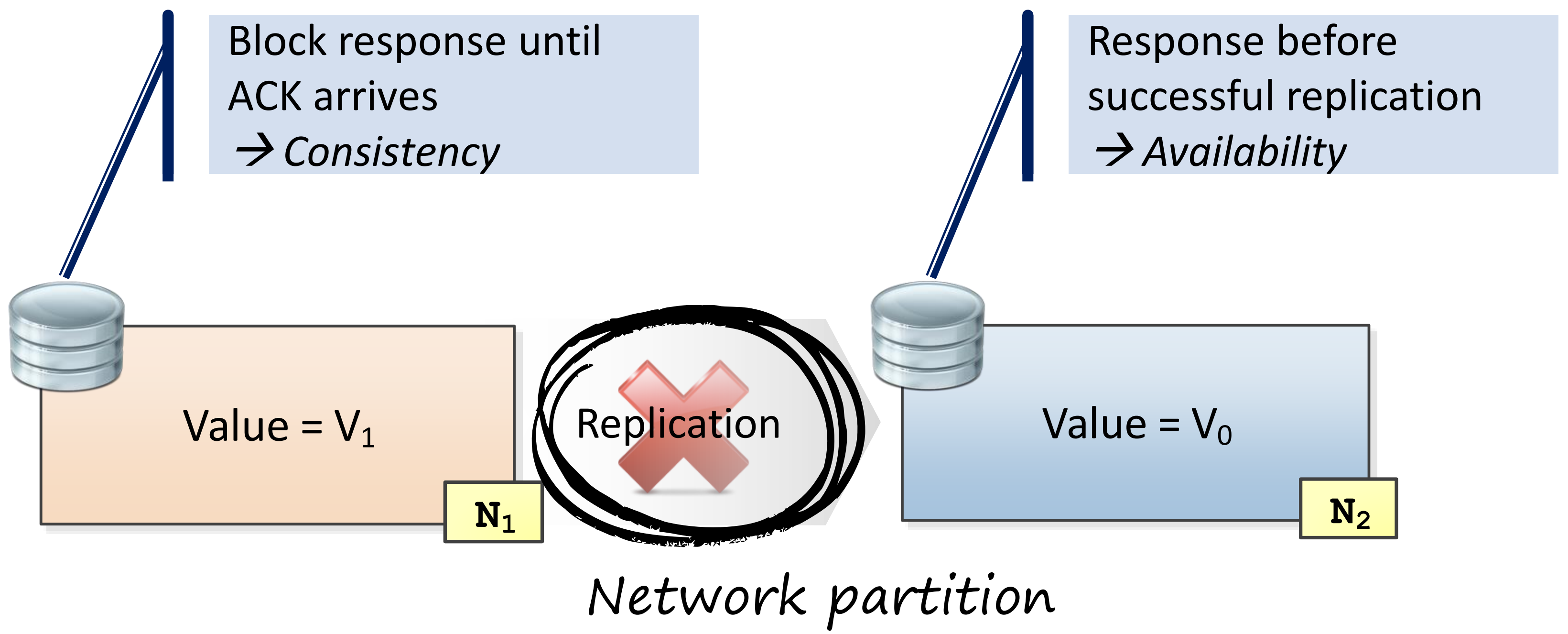
- **Consistency**: all clients have the same view on the data
- **Availability**: every request to a non-failed node most result in correct response
- **Partition tolerance**: the system has to continue working, even under arbitrary network partitions

Eric Brewer, ACM-PODC Keynote, Juli 2000

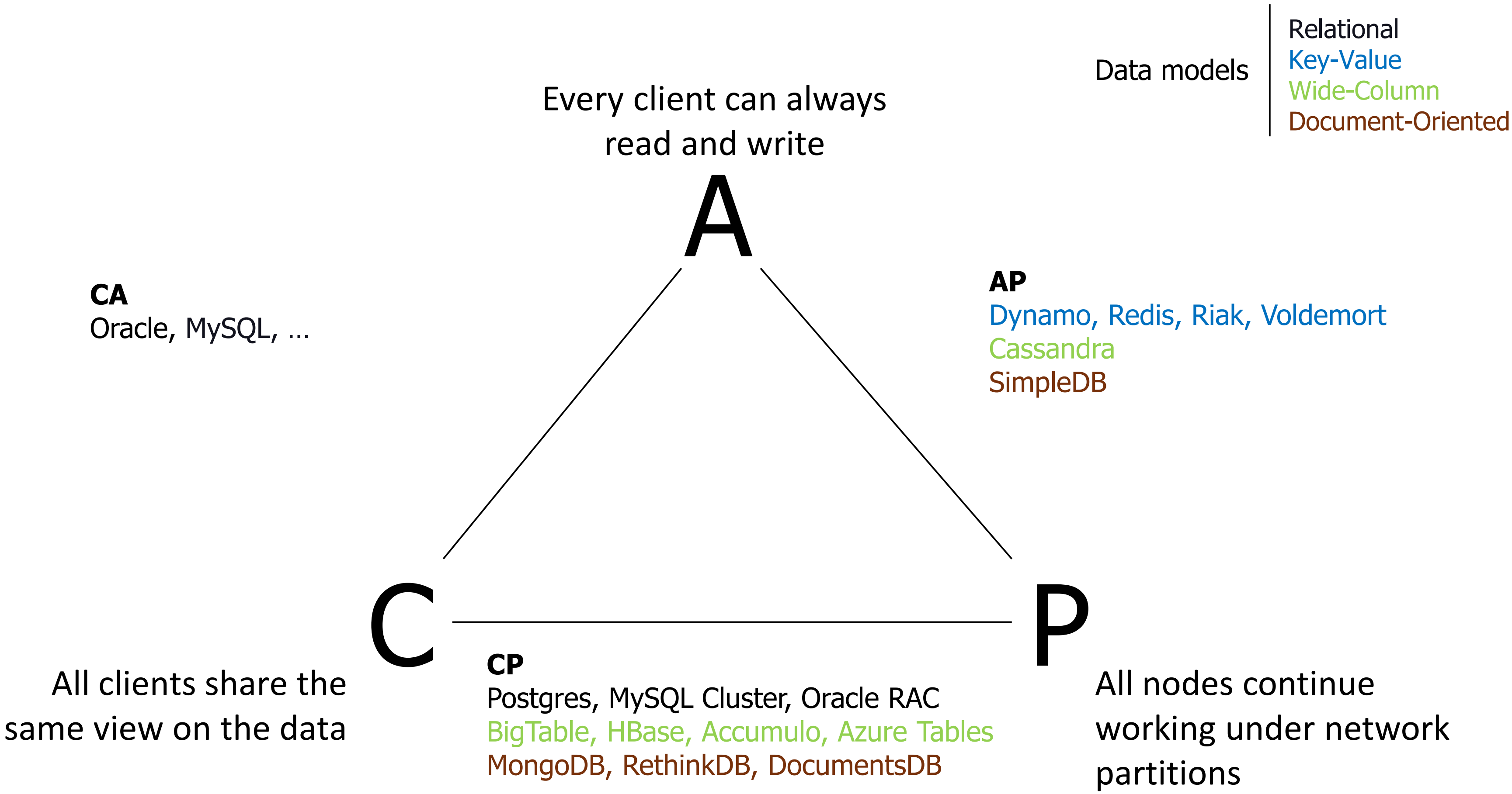
Gilbert, Lynch: Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services, SigAct News 2002

CAP-Theorem: simplified proof

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



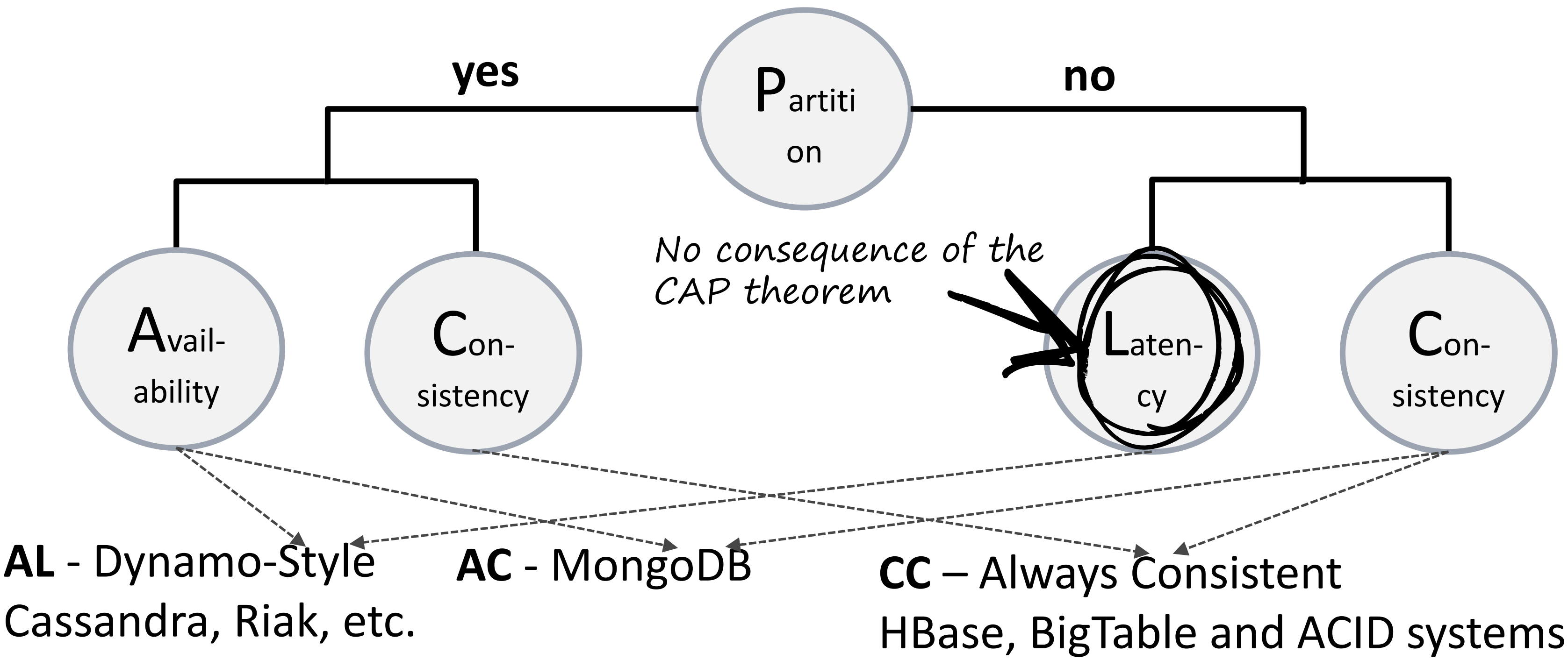
NoSQL Triangle



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

PACELC – an alternative CAP formulation

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



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

Serializability

Not Highly Available Either

Global serializability and availability are incompatible:

Write A=1
Read B

Write B=1
Read A

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

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

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

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

Impossibility Results

Consensus Algorithms

Safety Properties

- 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

Liveness Property

- 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.

Where CAP fits in

Negative Results in Distributed Computing

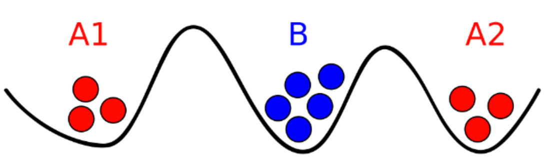
Asynchronous Network, _Unreliable_ Channel

Atomic Storage

Impossible:
CAP Theorem

Consensus

Impossible:
2 Generals Problem



Asynchronous Network, _Reliable_ Channel

Atomic Storage

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

Consensus

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

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

ACID vs BASE

„Gold standard“
for RDBMSs

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

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

Model of many
NoSQL systems

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

Weaker guarantees in a database?!

Default Isolation Levels in RDBMSs

Database	**Default Isolation**	**Maximum Isolation**
Actian Ingres 10.0/10S	**S**	**S**
Aerospike	RC	RC
Clustrix CLX 4100	RR	?
Greenplum 4.1	RC	**S**
IBM DB2 10 for z/OS	CS	**S**
IBM Informix 11.50	Depends	RR
MySQL 5.6	RR	**S**
MemSQL 1b	RC	RC
MS SQL Server 2012	RC	**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**

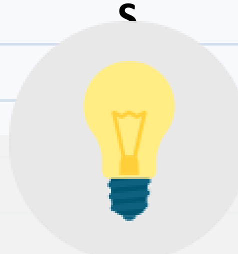
RC: read committed, RR: repeatable read, S: serializability,
SI: snapshot isolation, CS: cursor stability, CR: consistent read

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

Weaker guarantees in a database?!

Default Isolation Levels in RDBMSs

Database	Default Isolation	Maximum Isolation
Actian Ingres 10.0/10S	S	**S**
Aerospike	[illegible]	RC
Clustrix CLX 4100	[illegible]	?
Greenplum 4.1	[illegible]	S
IBM DB2 10 for z/OS	[illegible]	S
IBM Informix 11.50	Depends	RR
MySQL 5.6	[illegible]	S
MemSQL 1b	RC	RC
MS SQL Server 2012	RC	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 database systems.

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.

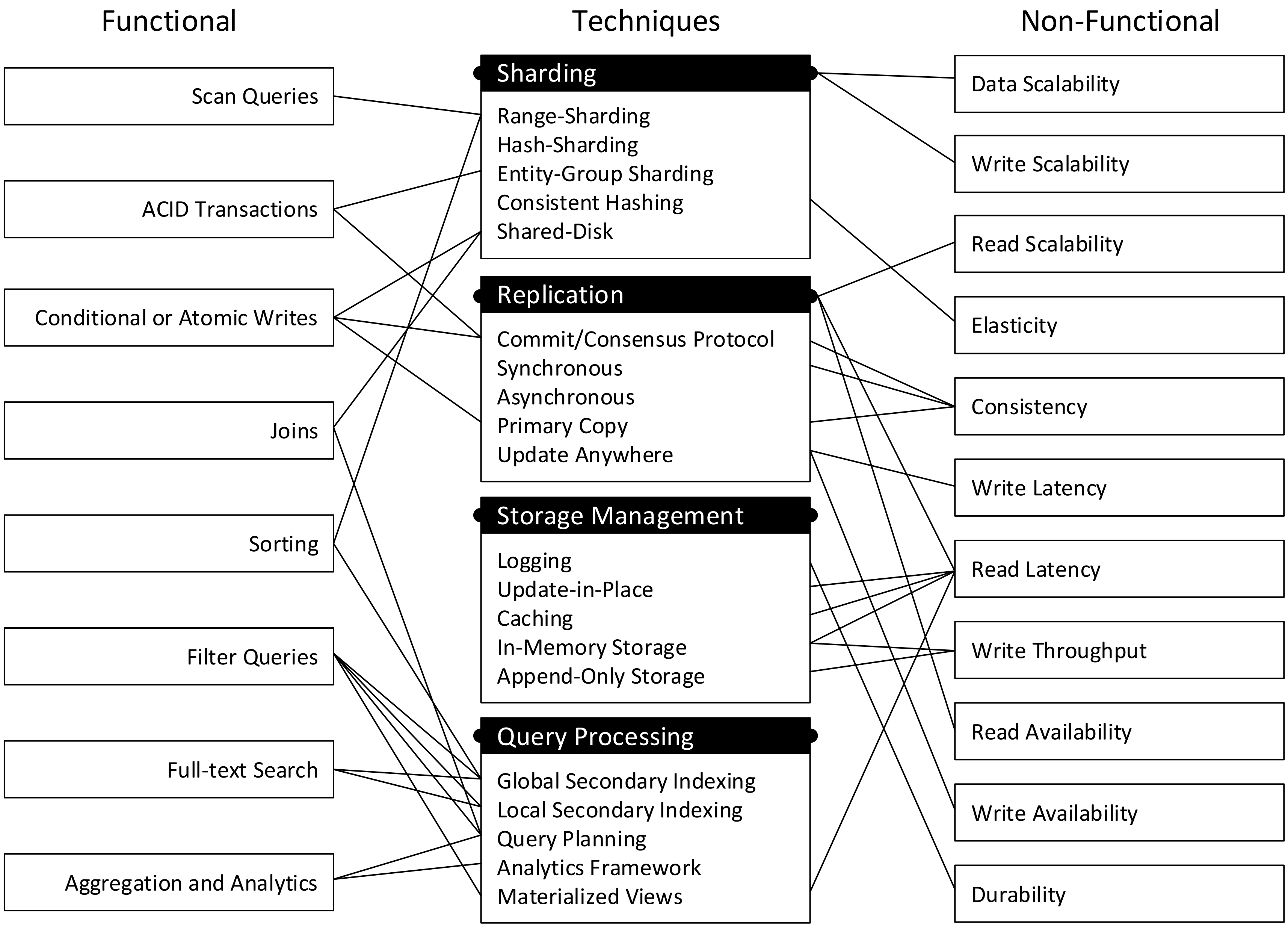
Data Models and **CAP** provide high-level classification.

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

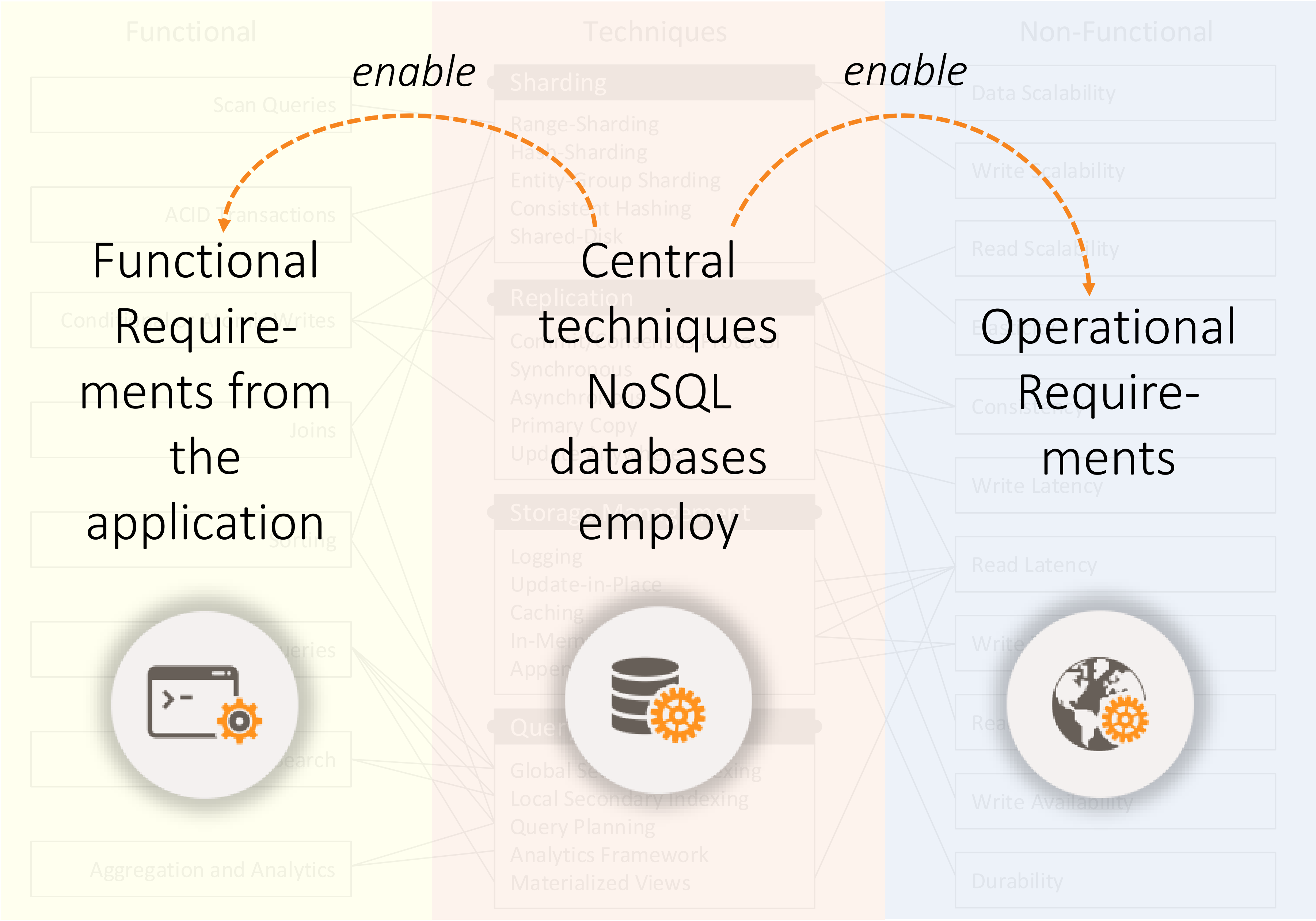
Outline

- 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

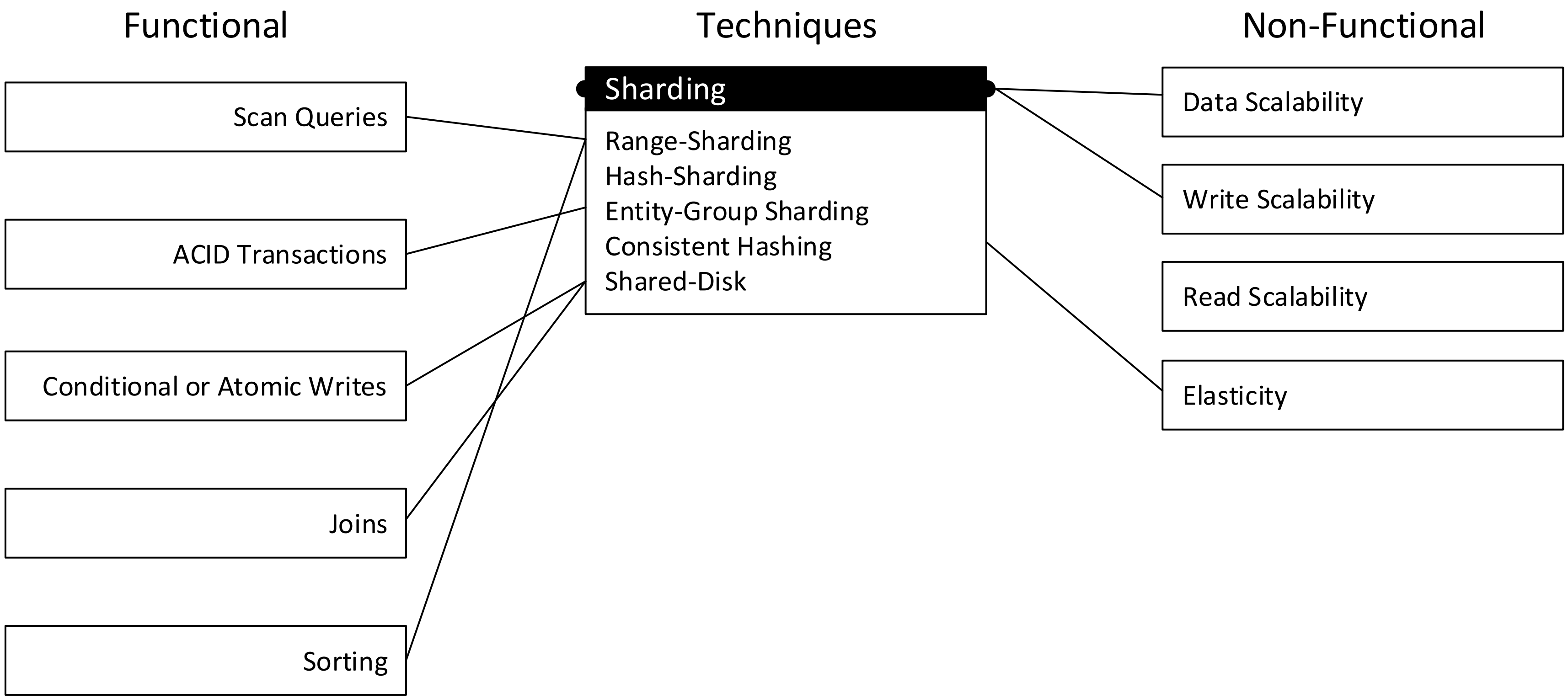
Abstract. Today, data is generated and consumed at unprecedented scale. This has lead to novel approaches for scalable data management subsumed under the term "NoSQL" database systems to handle the ever-increasing data volume and request loads. However, the heterogeneity and diversity of the numerous 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

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



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

Sharding

Approaches

Hash-based Sharding

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

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 *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.

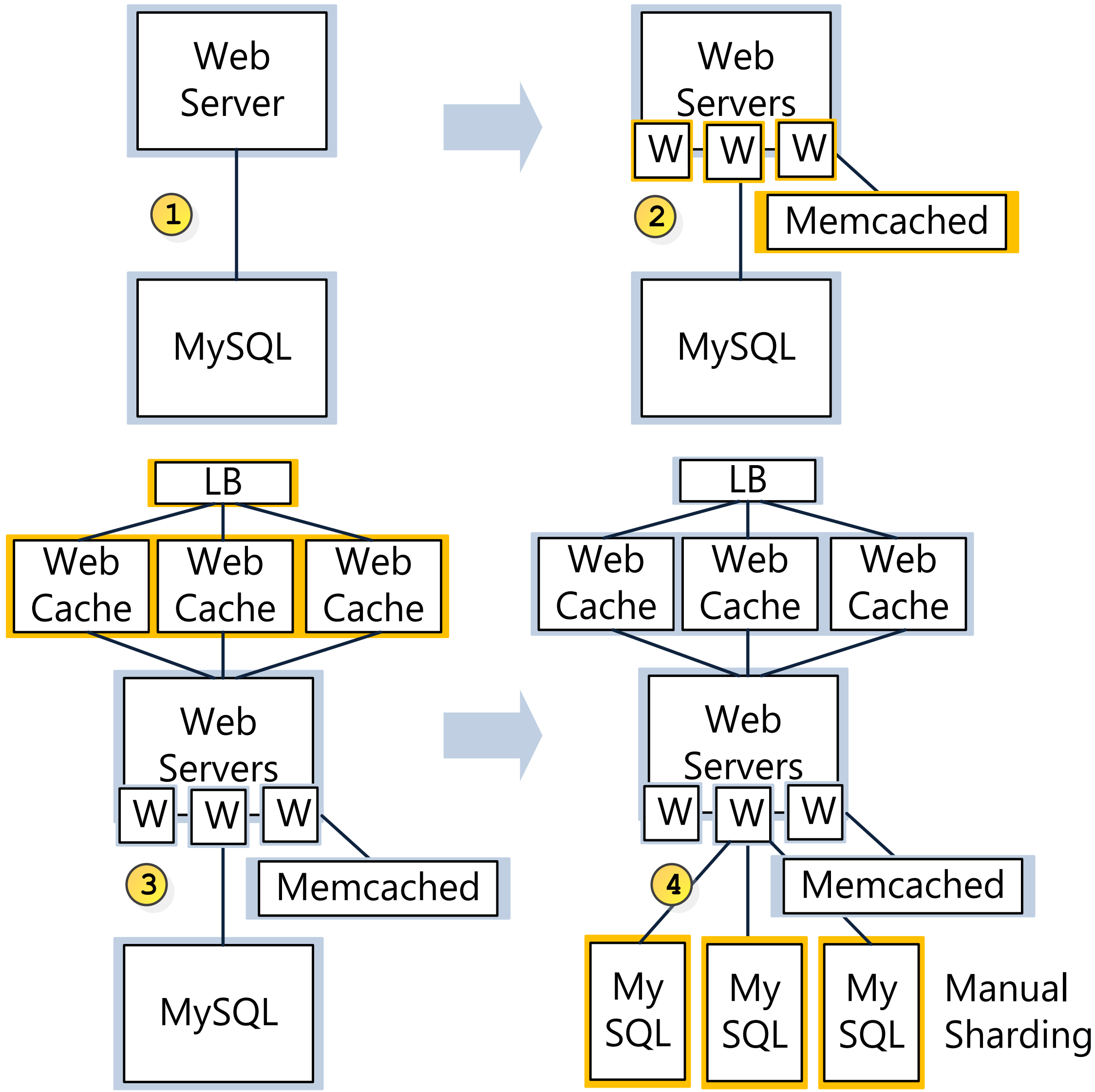
Problems of Application-Level Sharding

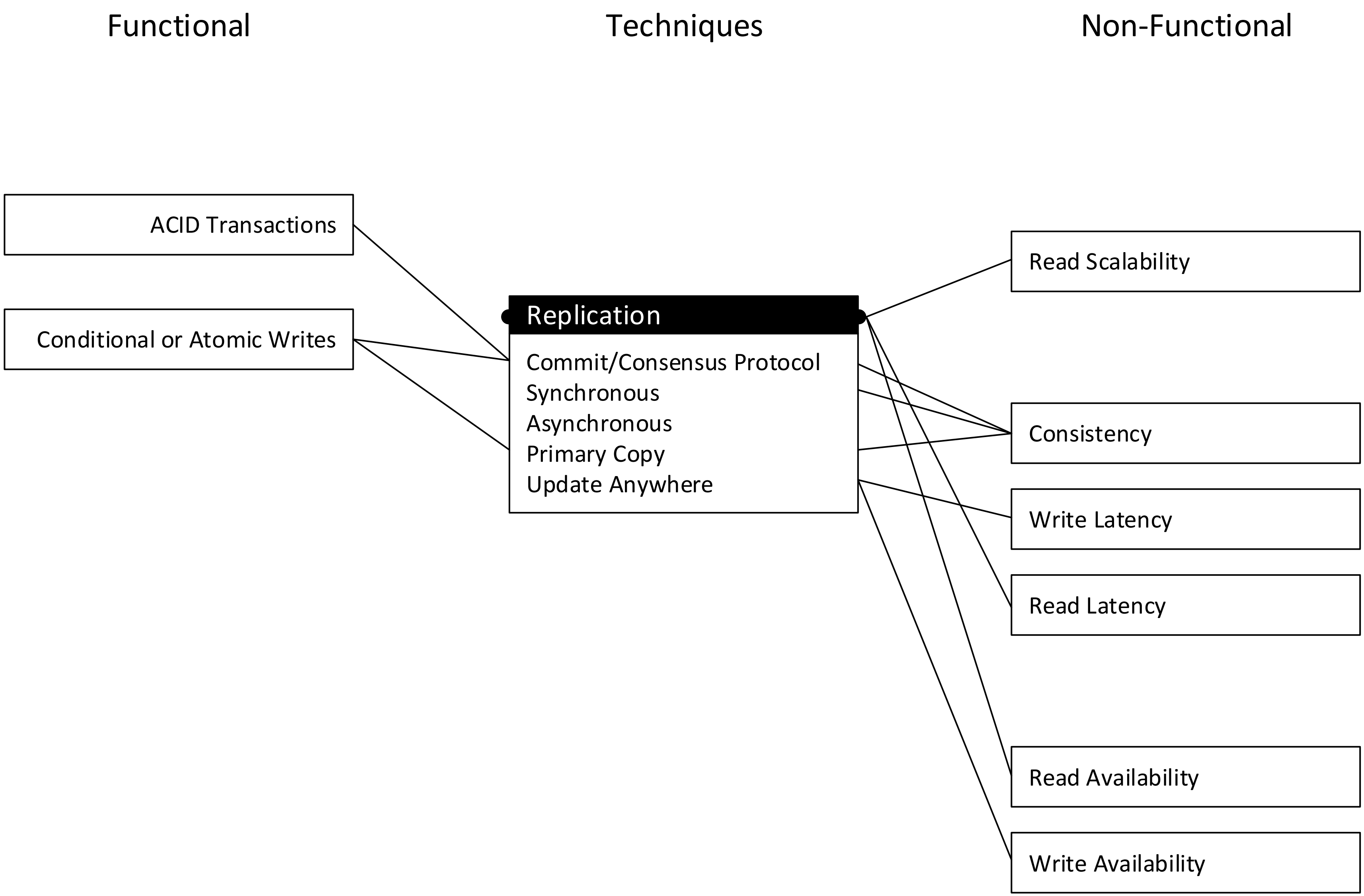
Example: **Tumblr**

- Caching
- Sharding from application

Moved towards:

- Redis
- HBase

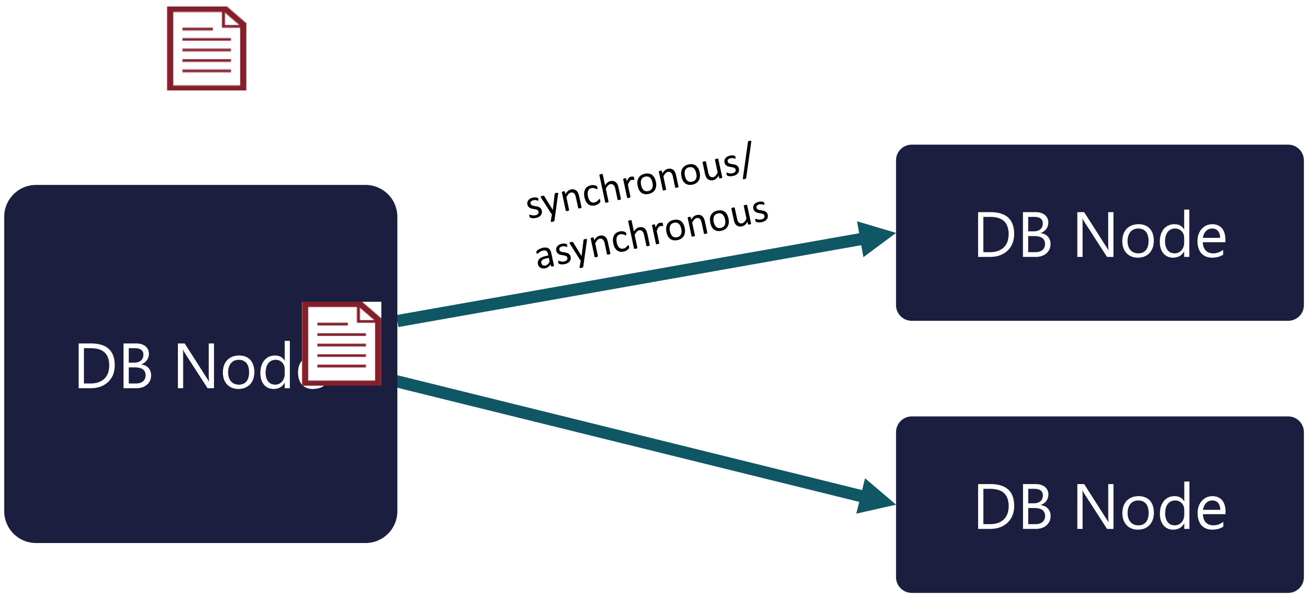



Functional
Techniques
Non-Functional
ACID Transactions
Conditional or Atomic Writes
Replication
Commit/Consensus Protocol
Synchronous
Asynchronous
Primary Copy
Update Anywhere
Read Scalability
Consistency
Write Latency
Read Latency
Read Availability
Write Availability

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

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

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

Implemented in

BigTable, HBase, Accumulo, CouchBase, MongoDB, RethinkDB

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

Replication: Where

Master-Slave (*Primary Copy*)

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

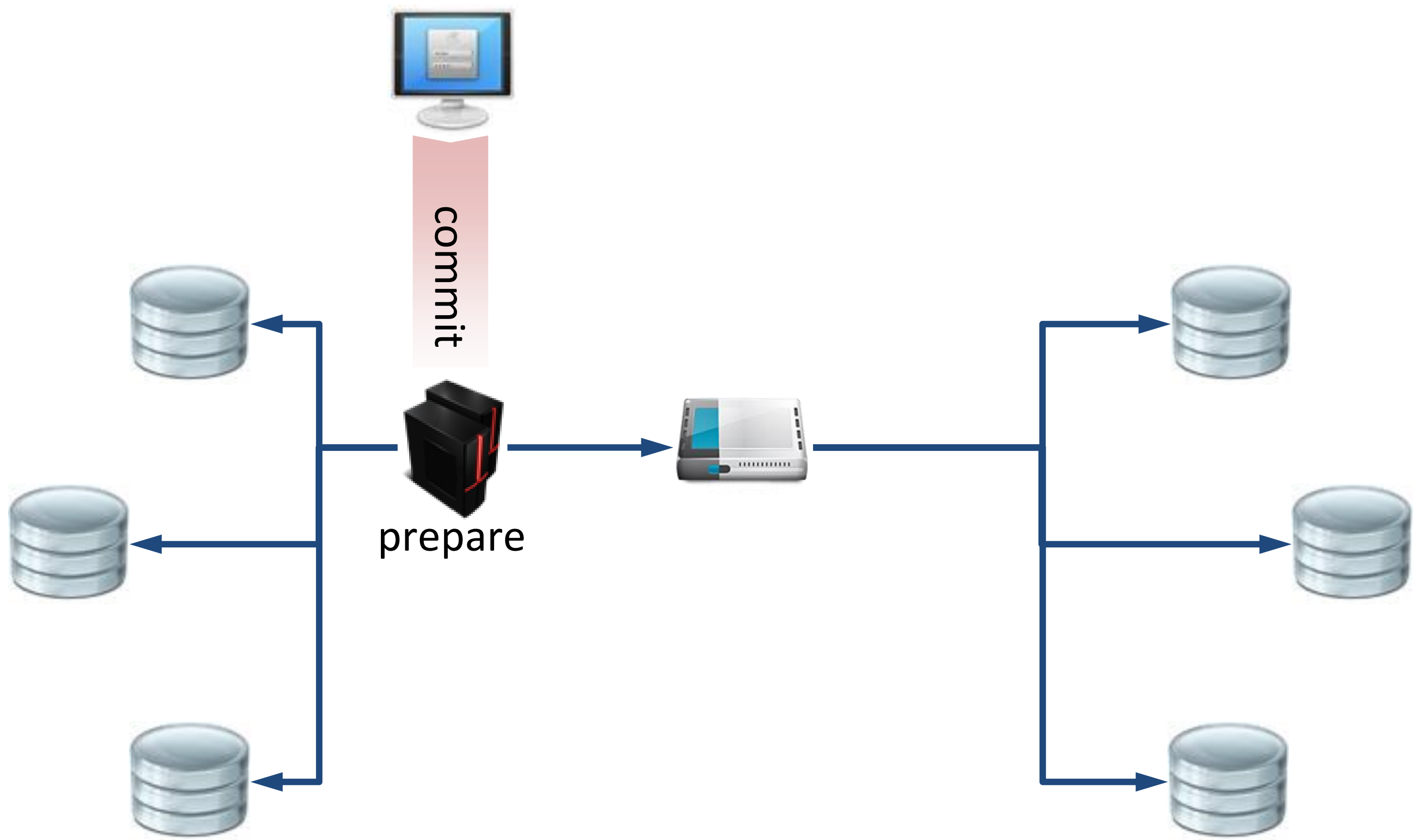
Multi-Master (*Update anywhere*)

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

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

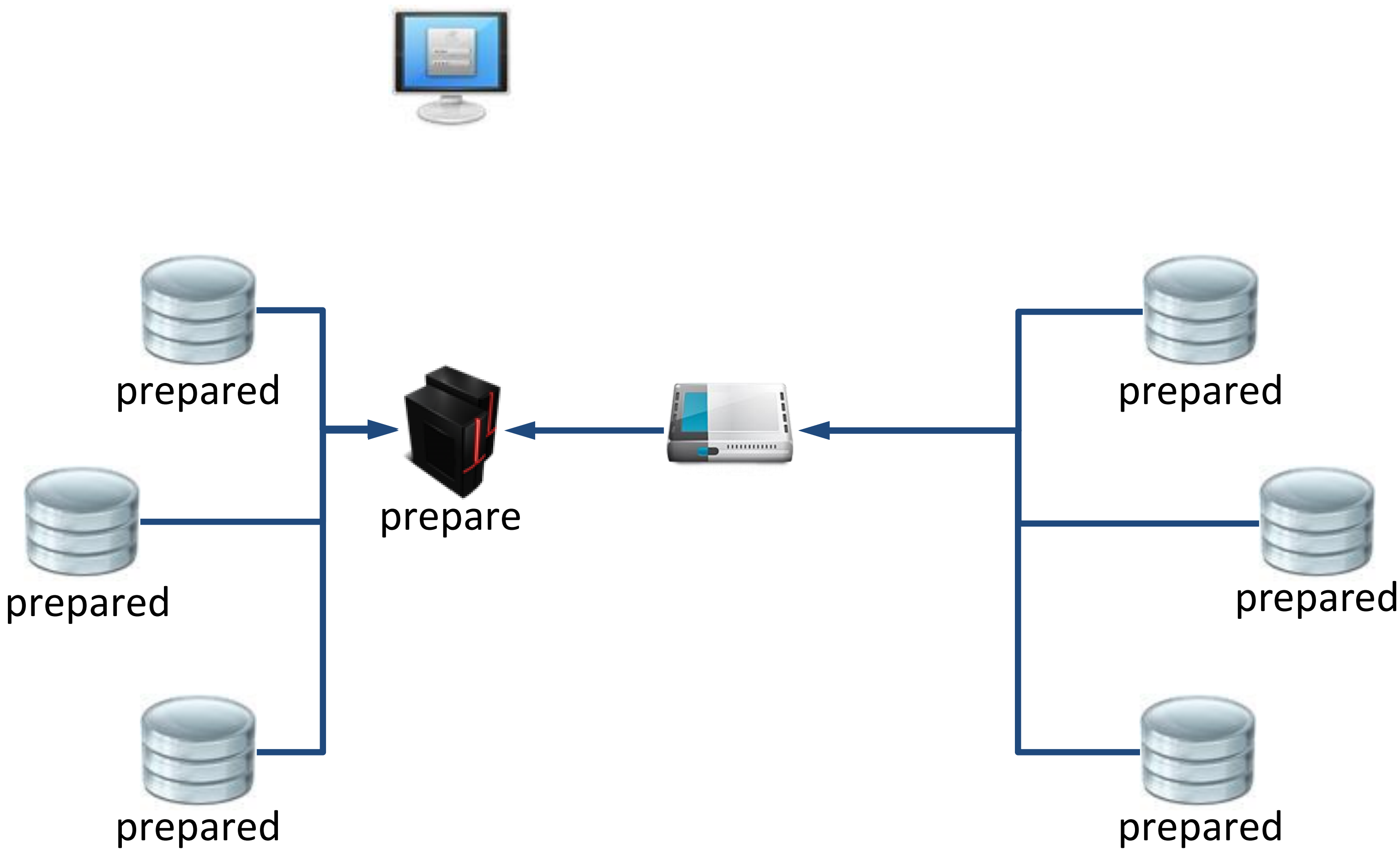
Synchronous Replication

Example: Two-Phase Commit is not partition-tolerant



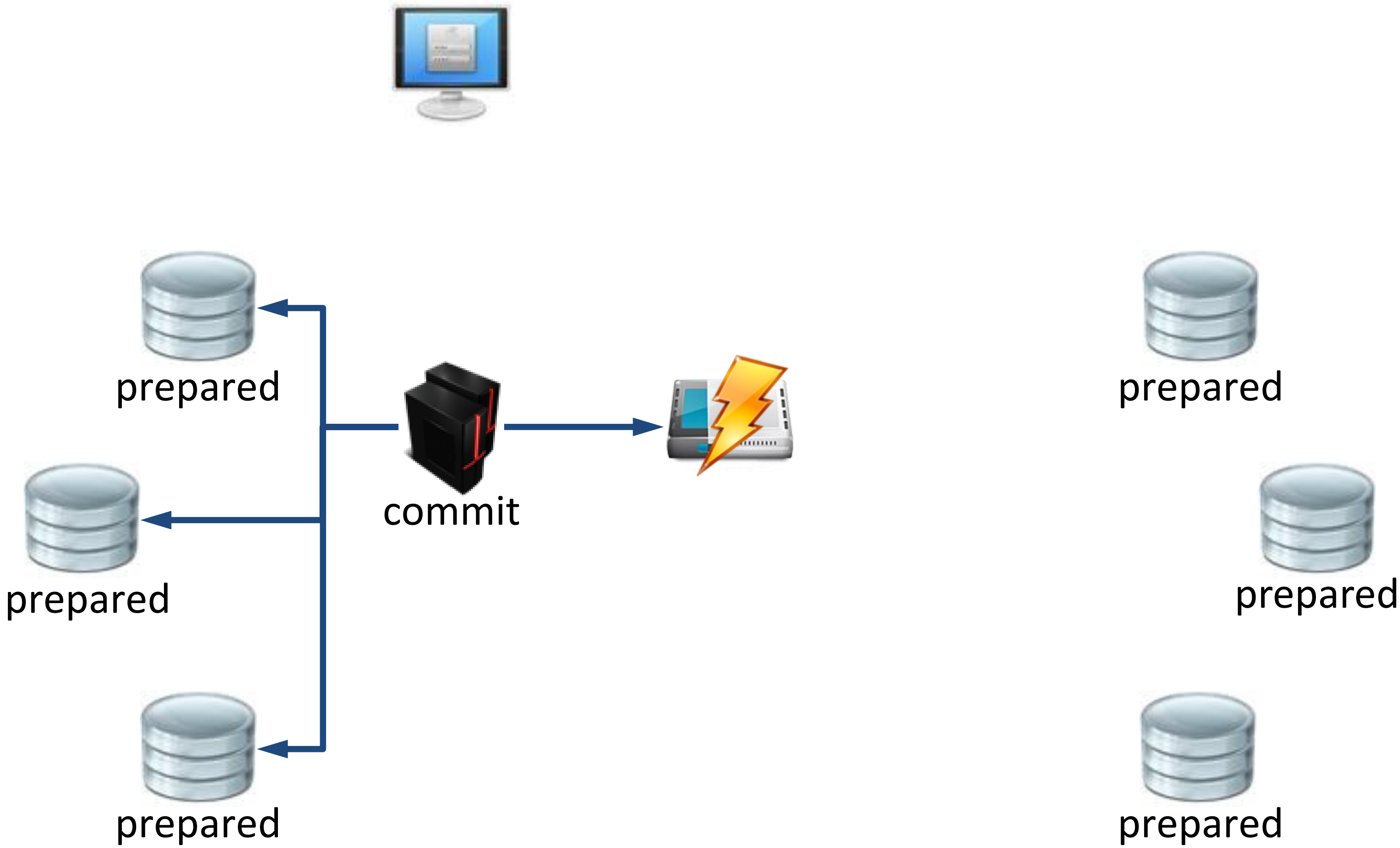
Synchronous Replication

Example: Two-Phase Commit is not partition-tolerant



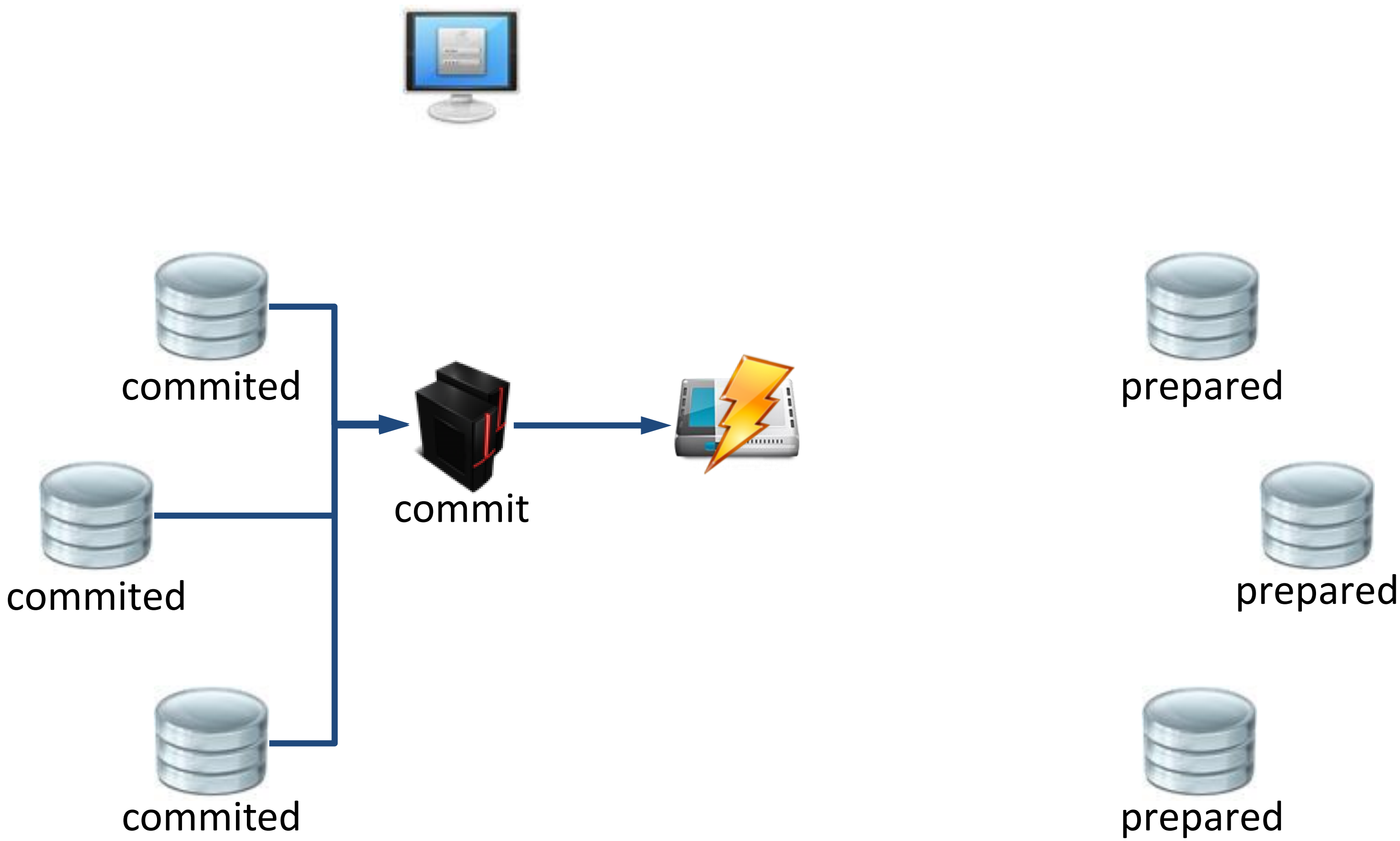
Synchronous Replication

Example: Two-Phase Commit is not partition-tolerant



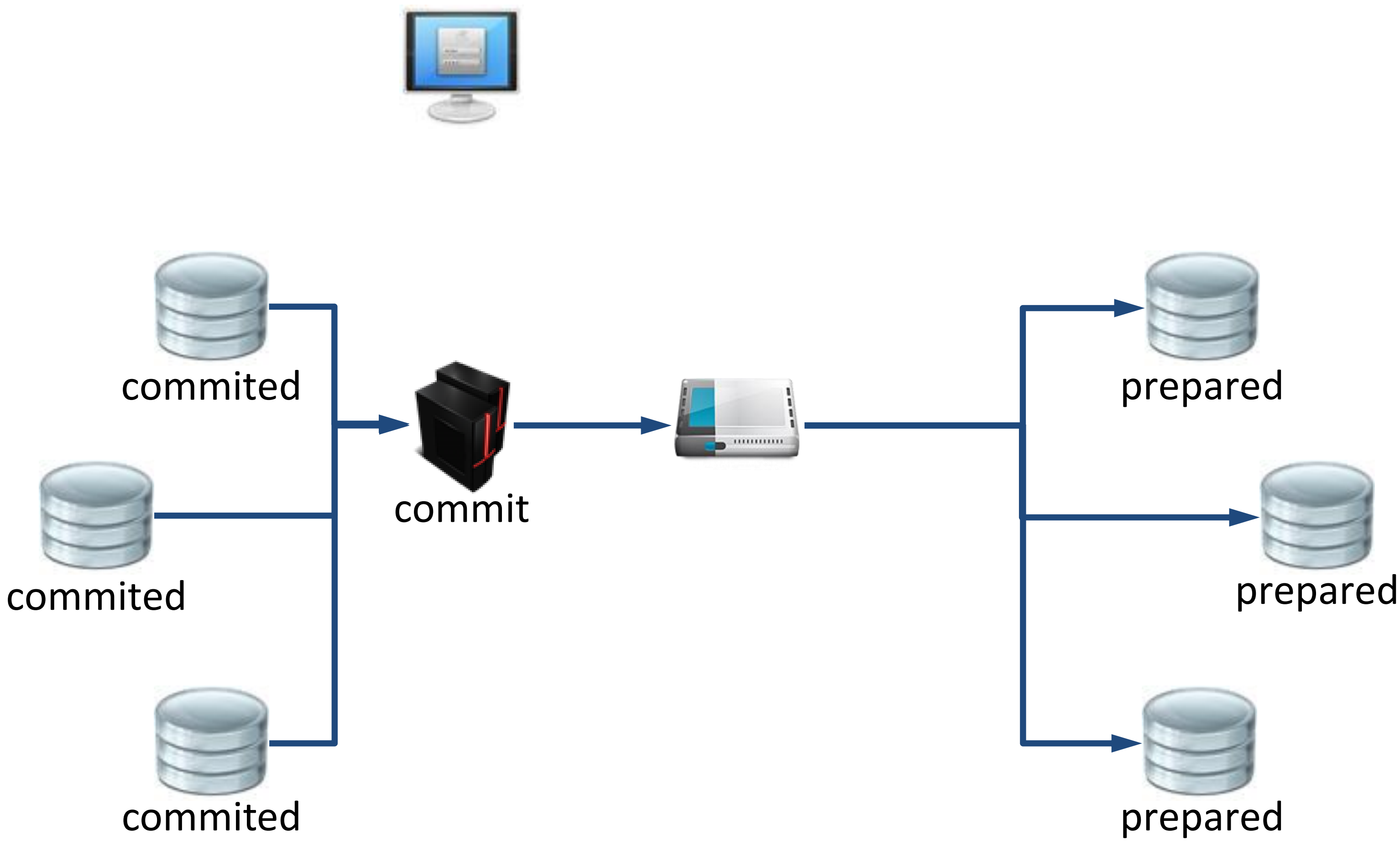
Synchronous Replication

Example: Two-Phase Commit is not partition-tolerant



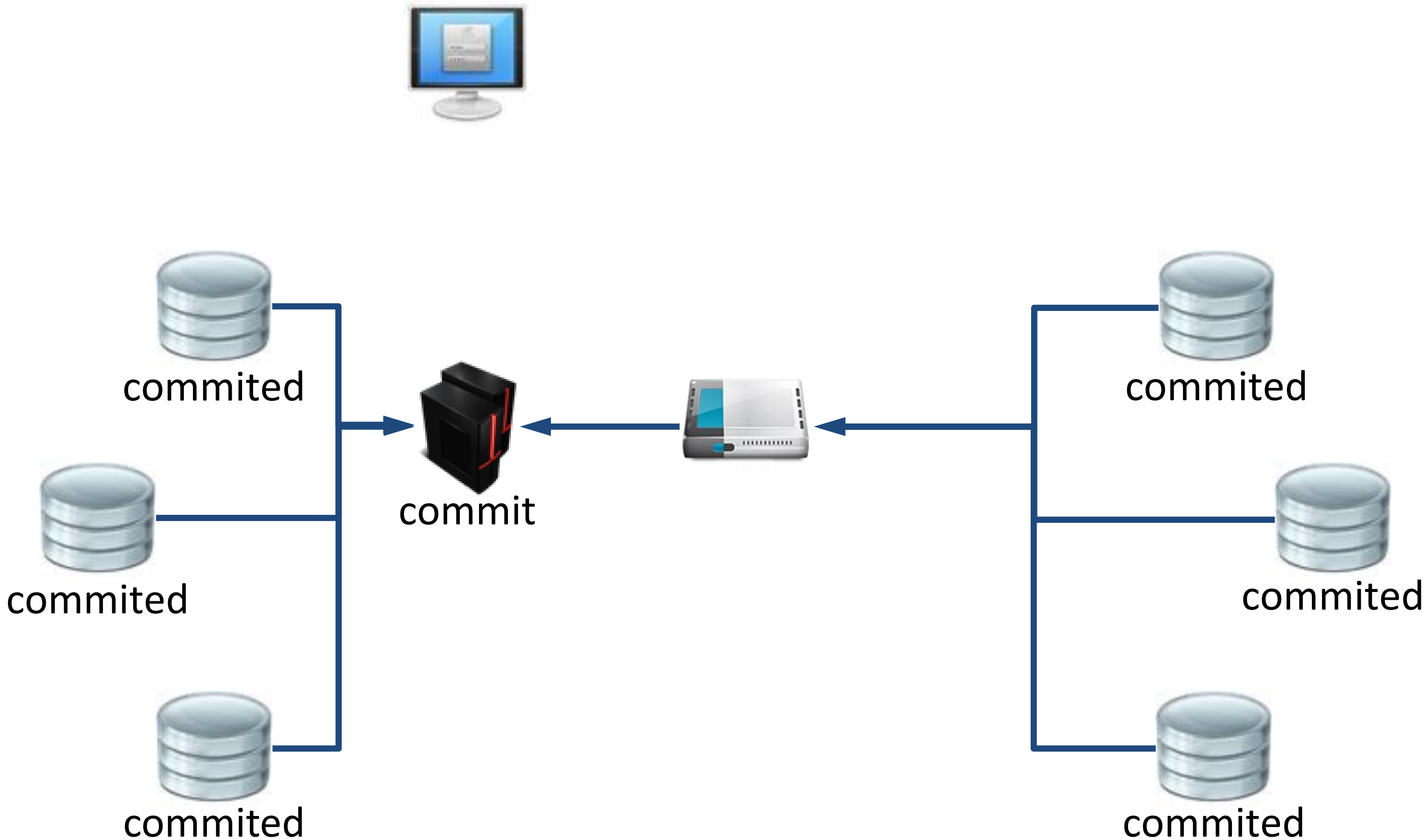
Synchronous Replication

Example: Two-Phase Commit is not partition-tolerant



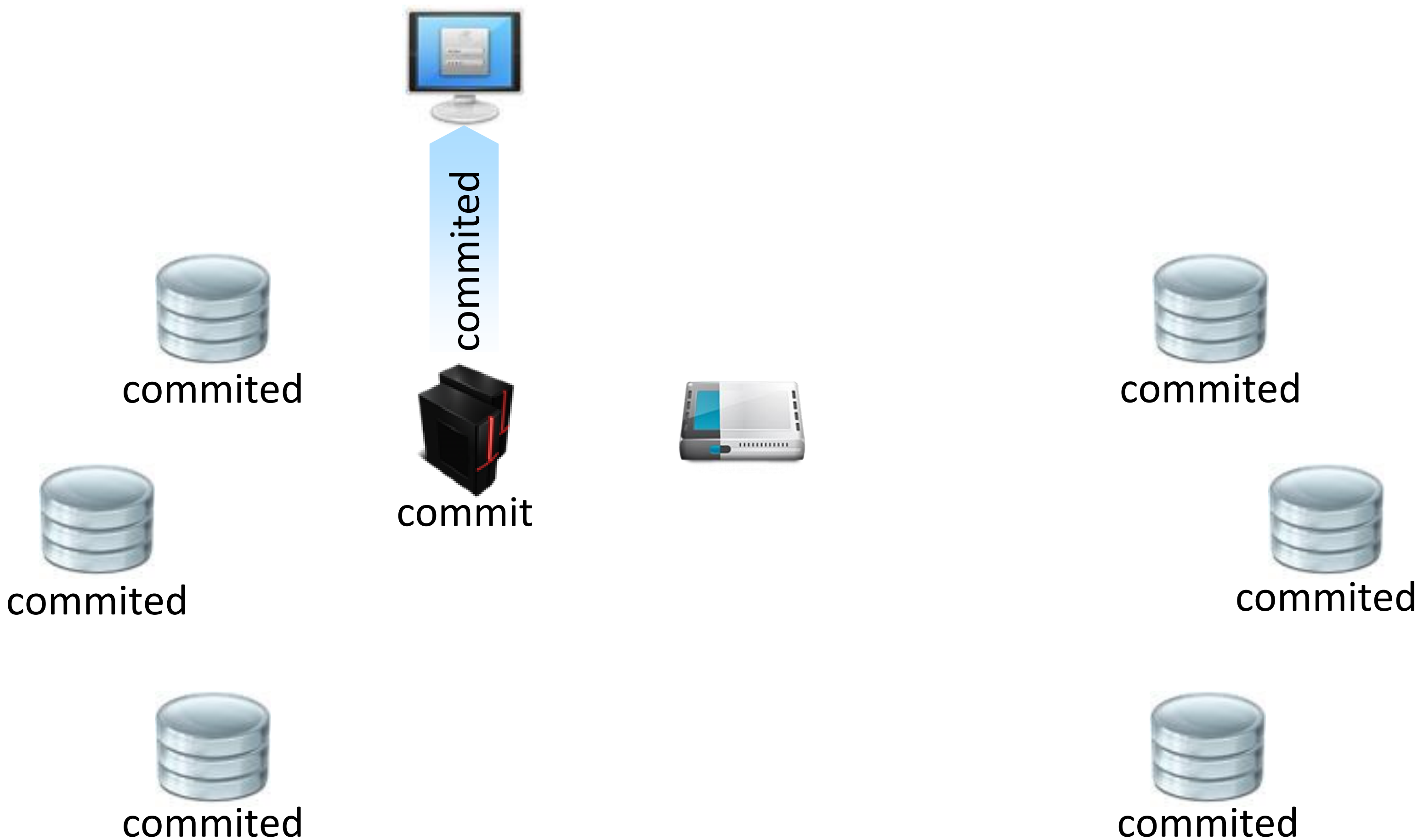
Synchronous Replication

Example: Two-Phase Commit is not partition-tolerant

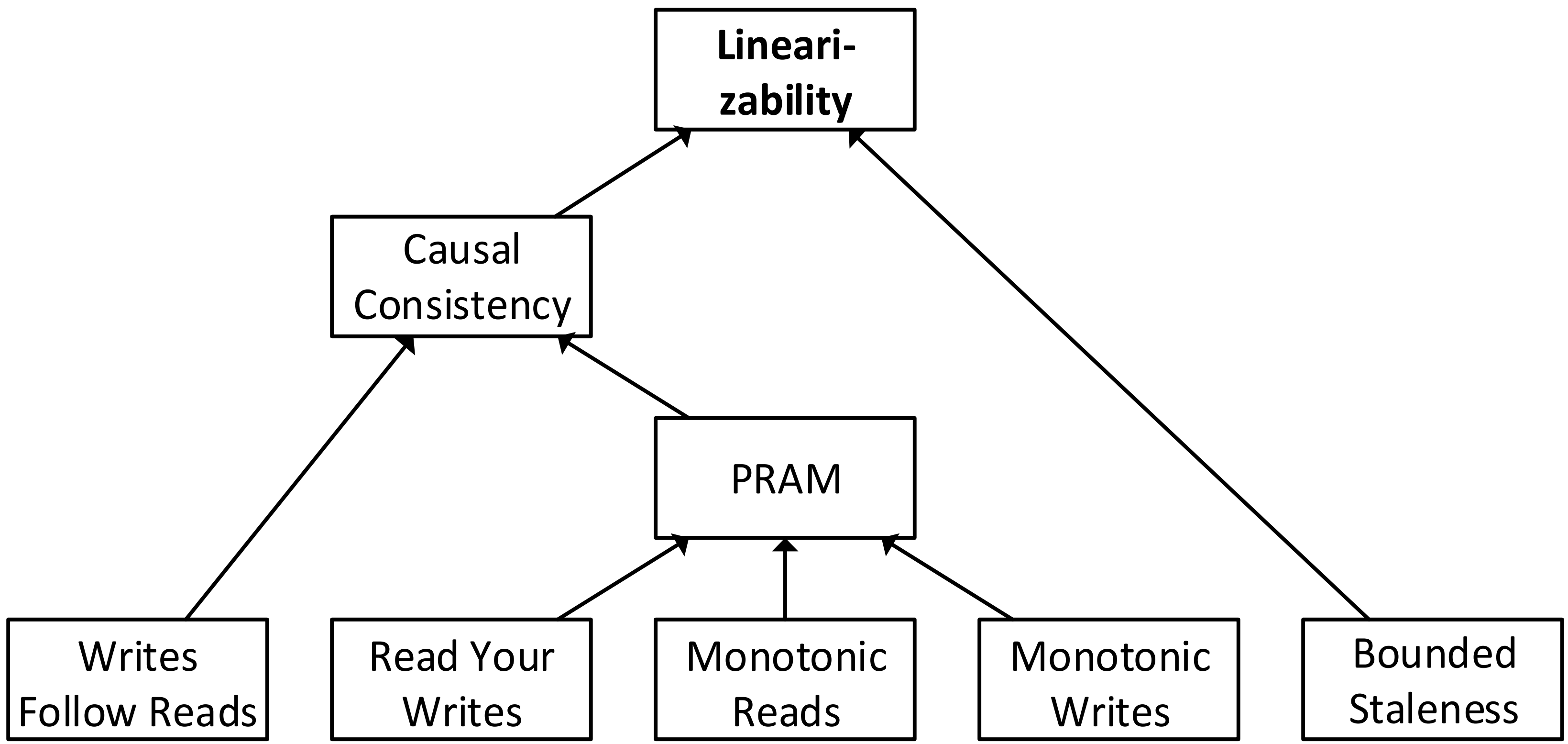


Synchronous Replication

Example: Two-Phase Commit is not partition-tolerant



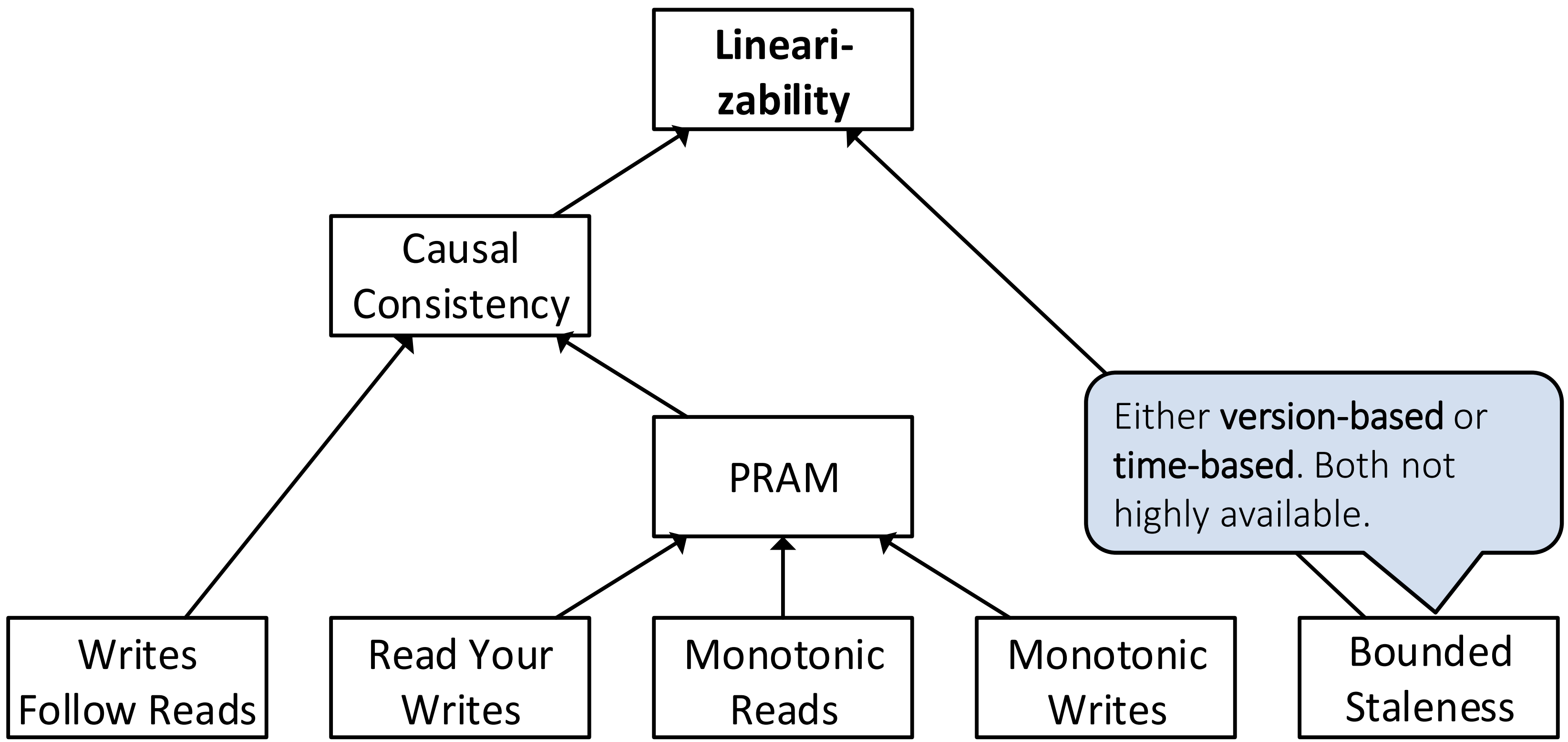
Consistency Levels



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.

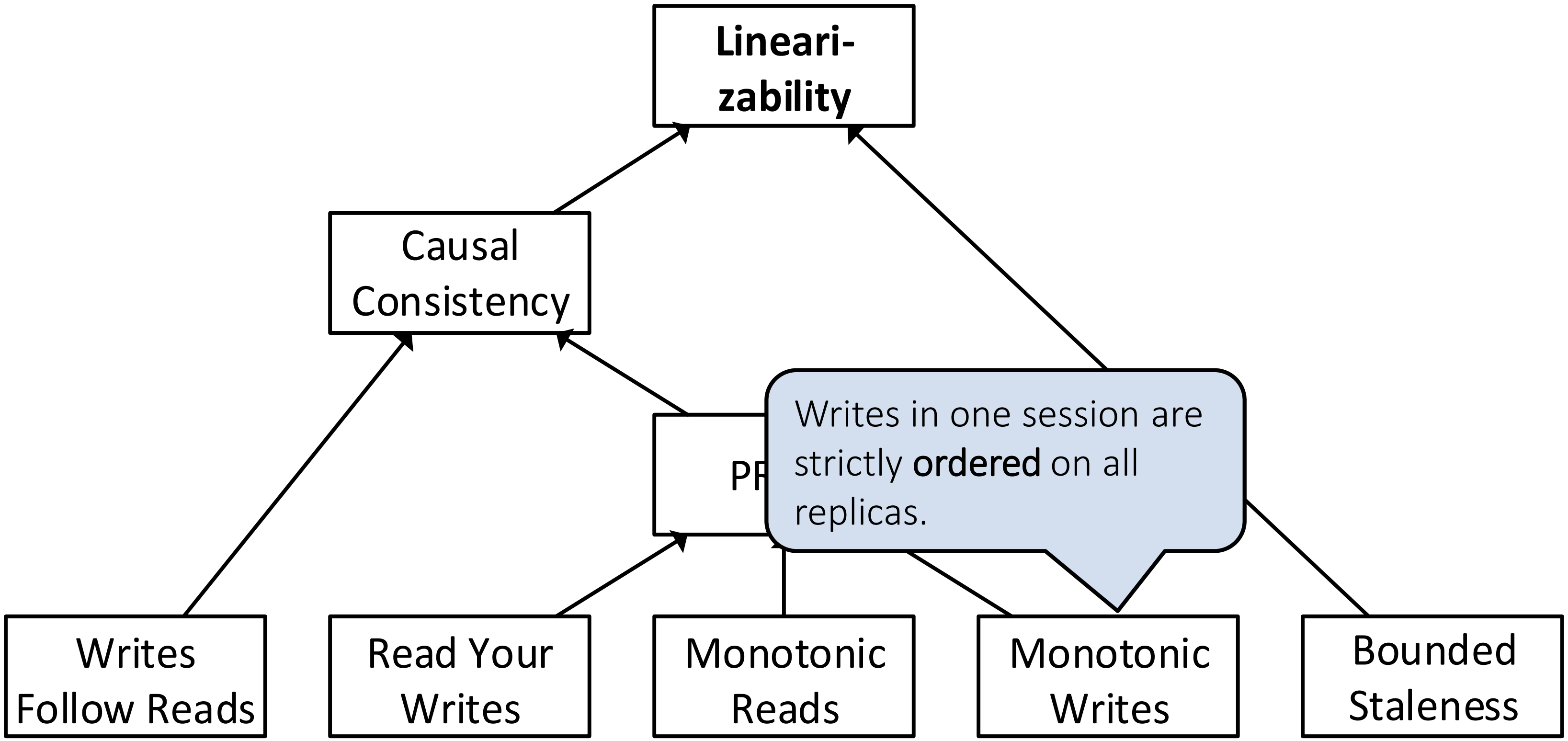
Consistency Levels



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.

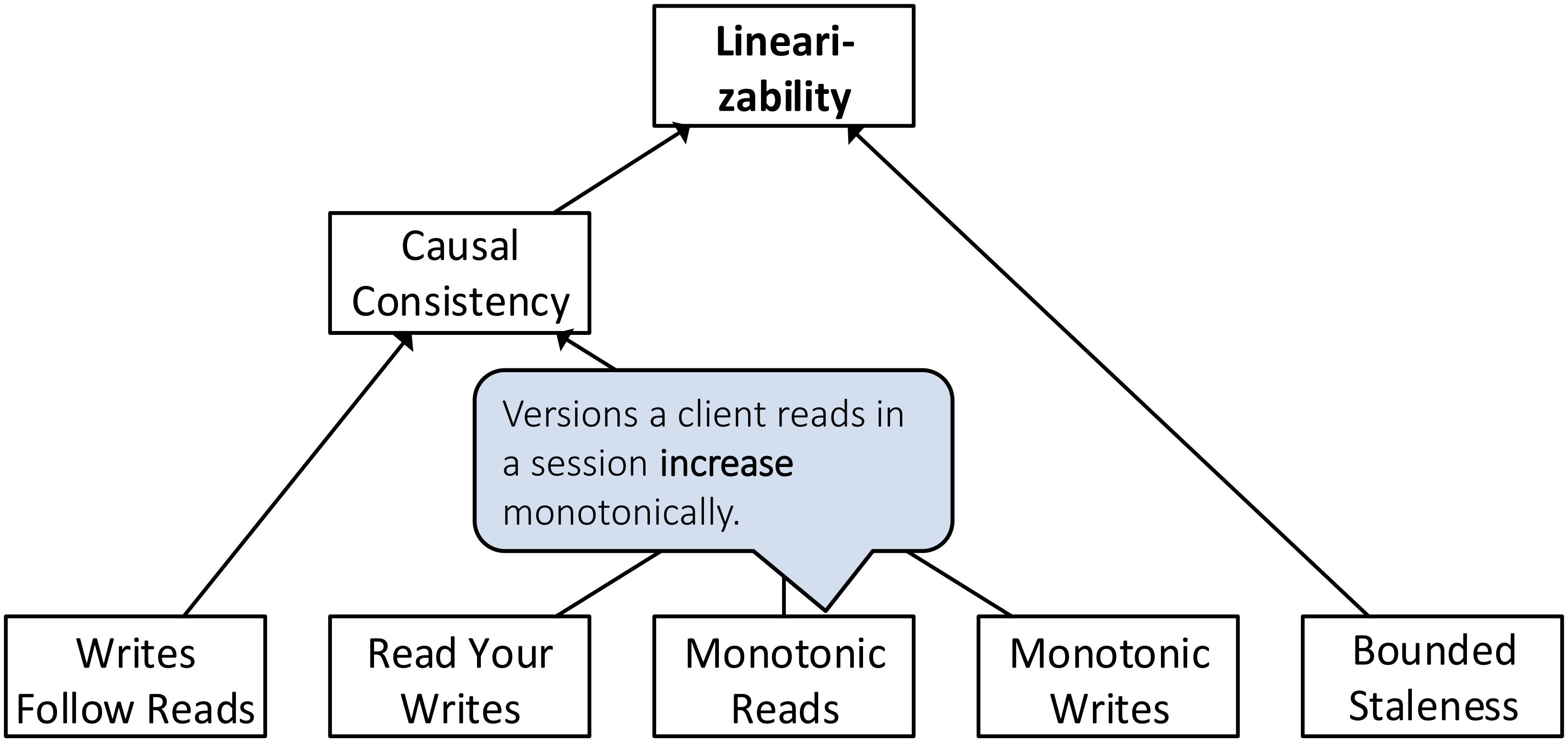
Consistency Levels



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.

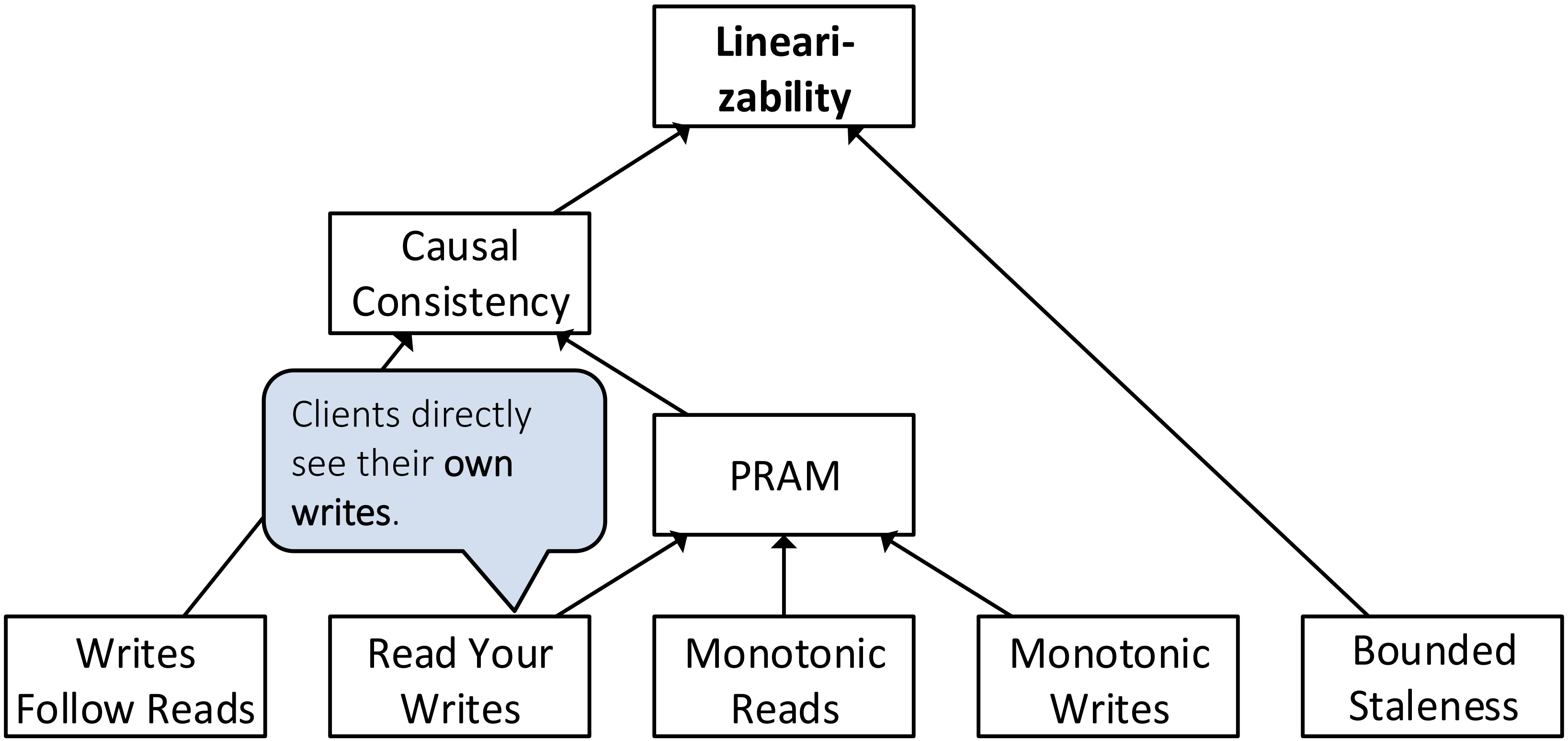
Consistency Levels



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.

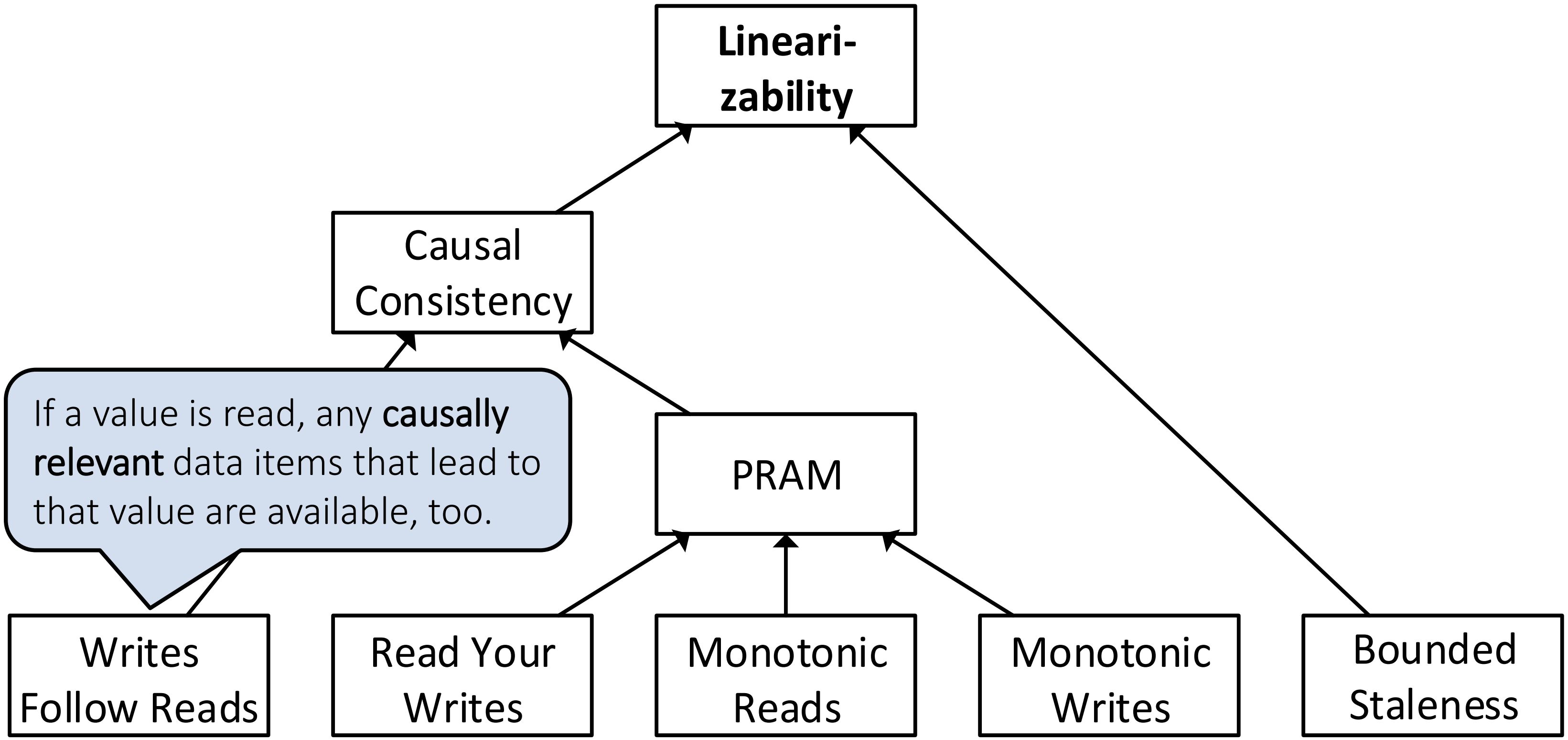
Consistency Levels



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.

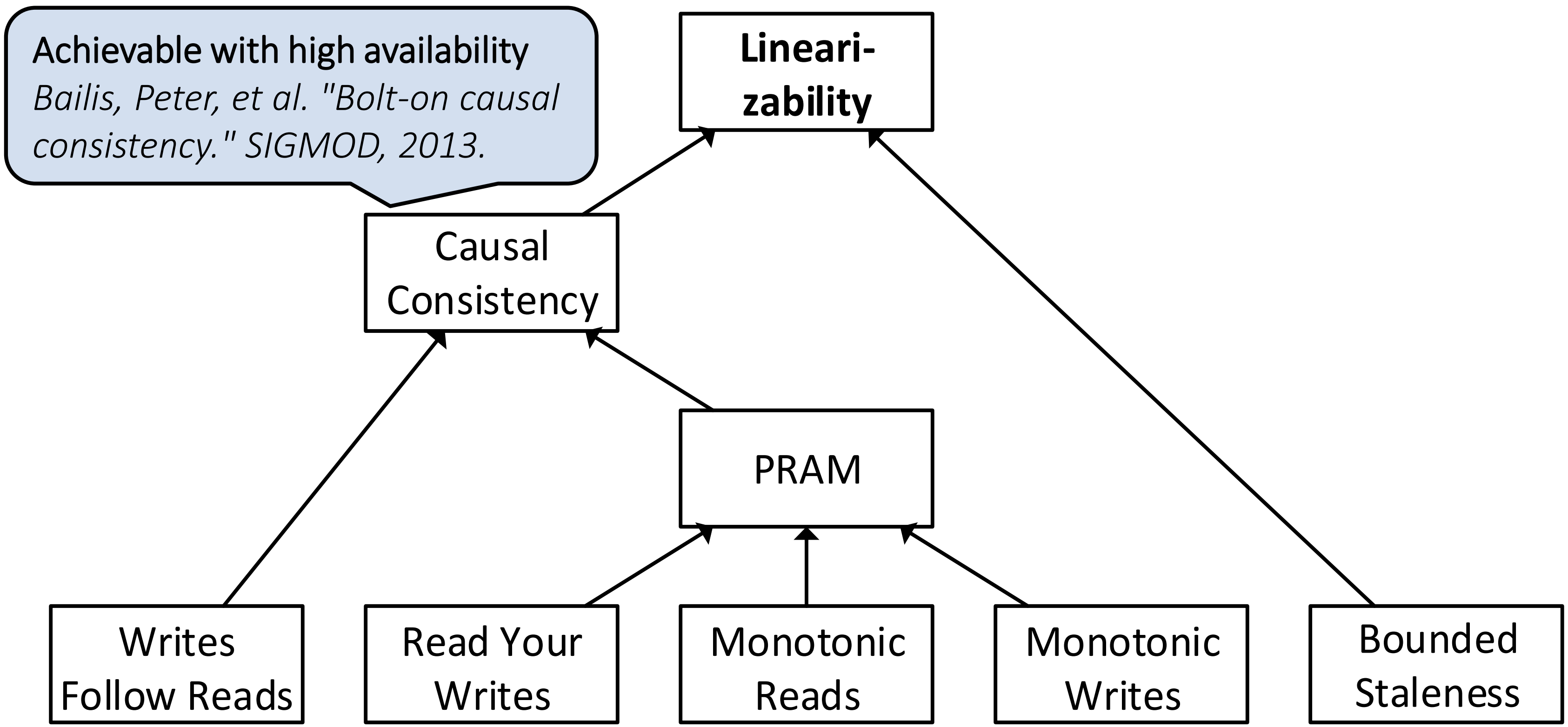
Consistency Levels



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.

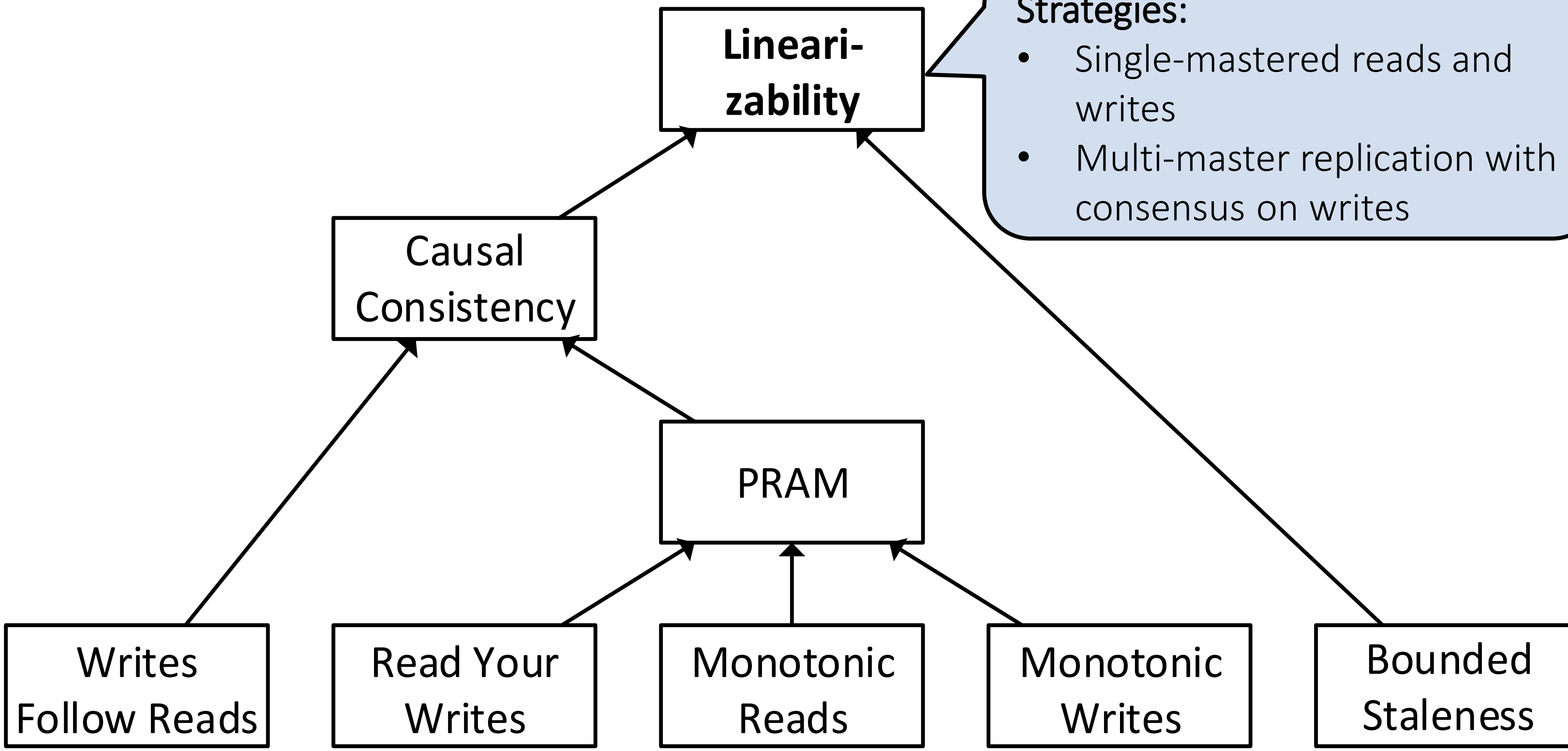
Consistency Levels



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



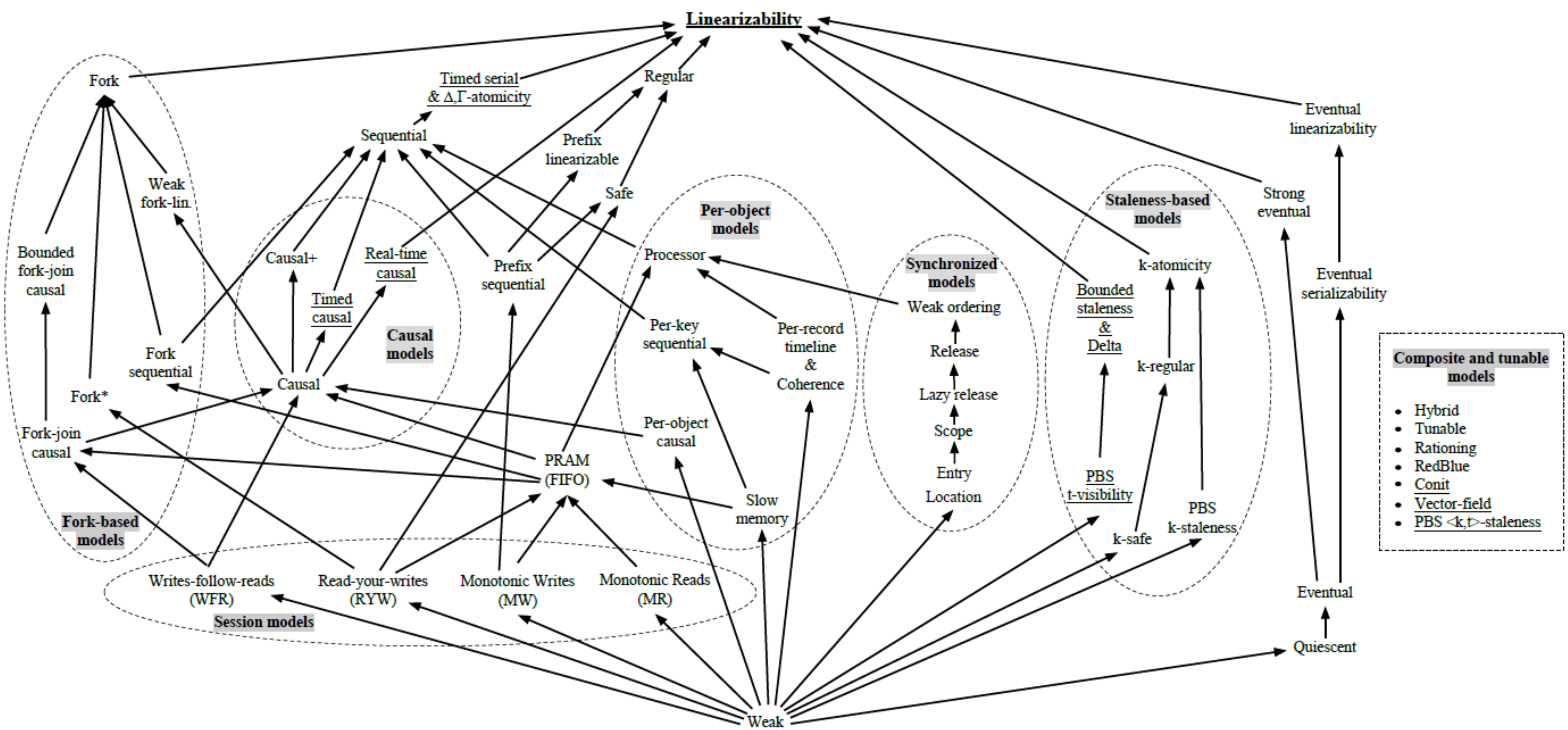
Strategies:

- 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.

Problem: Terminology

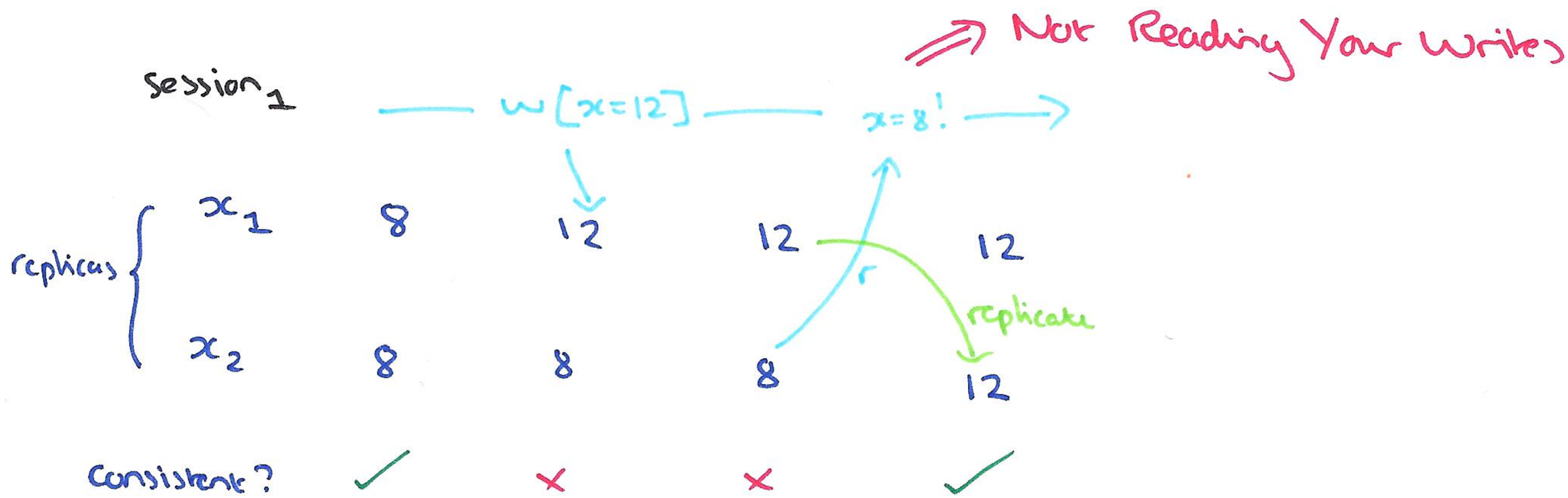


V., Paolo, and M. Vukolić. "Consistency in Non-Transactional Distributed Storage Systems." ACM CSUR (2016).

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.

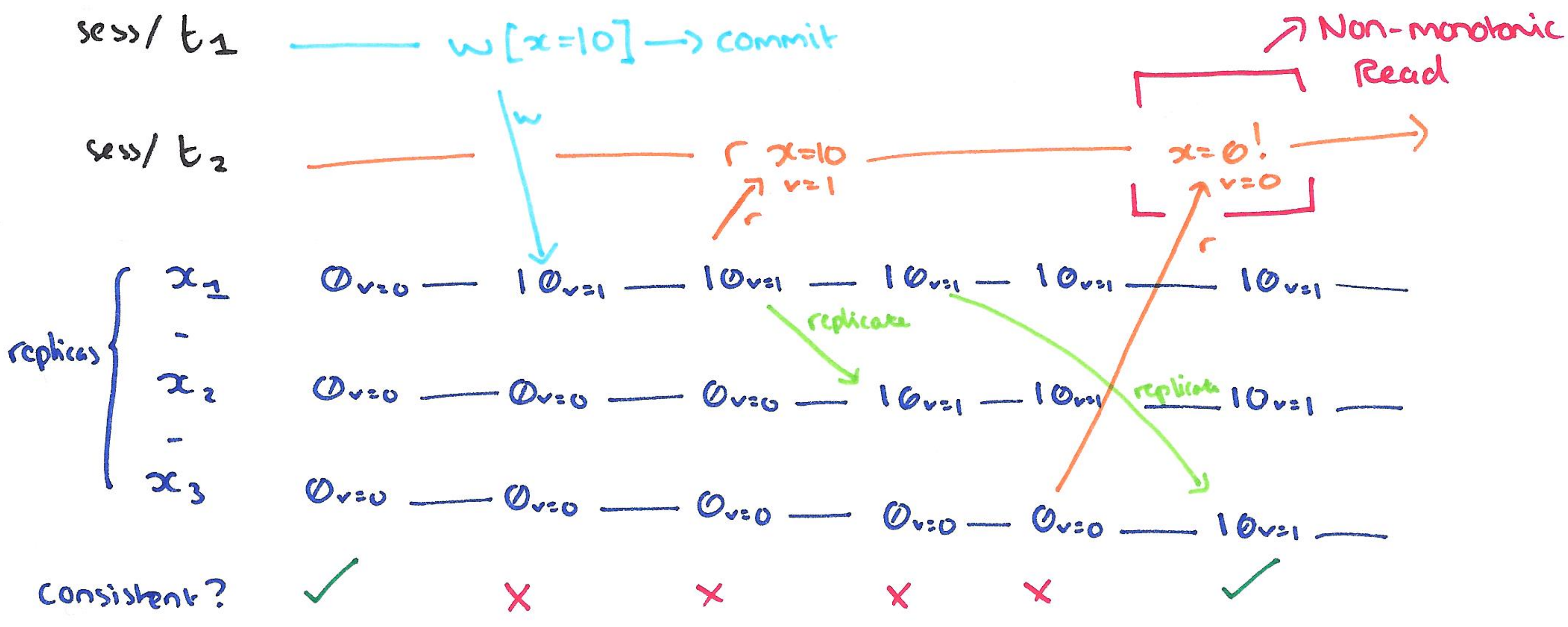


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

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

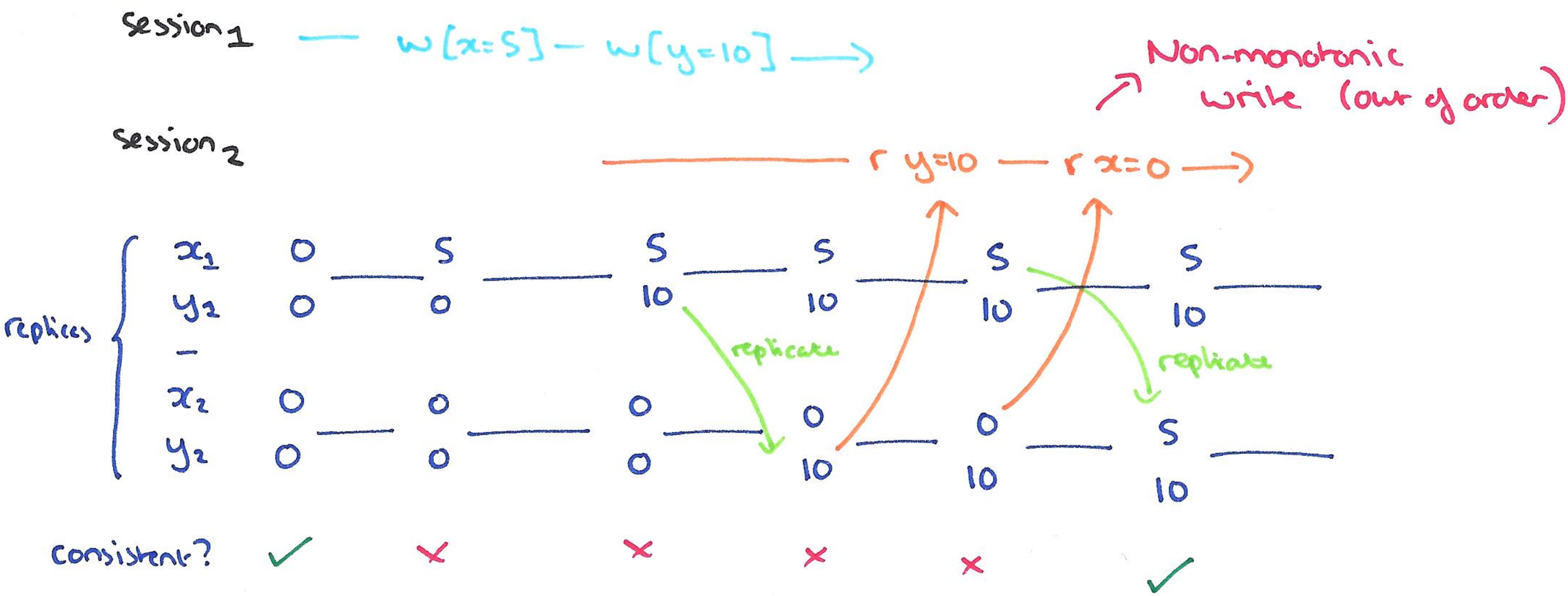


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

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.

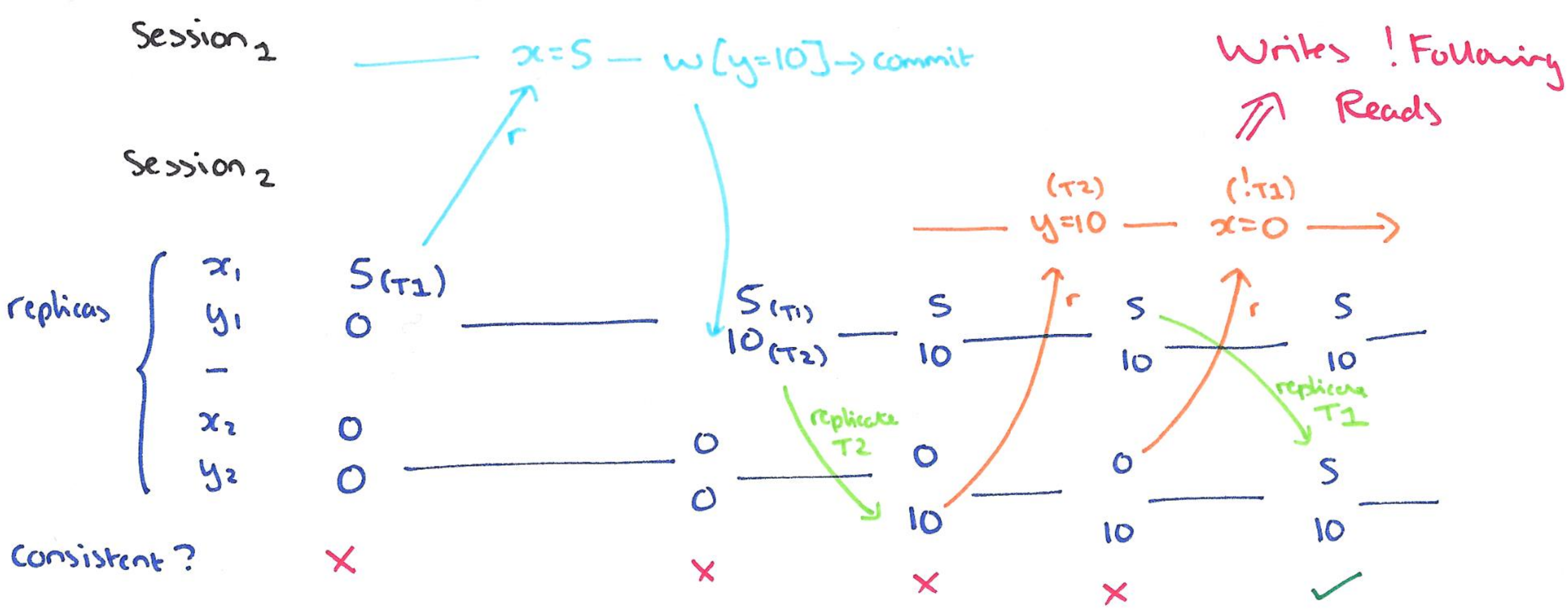


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

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.

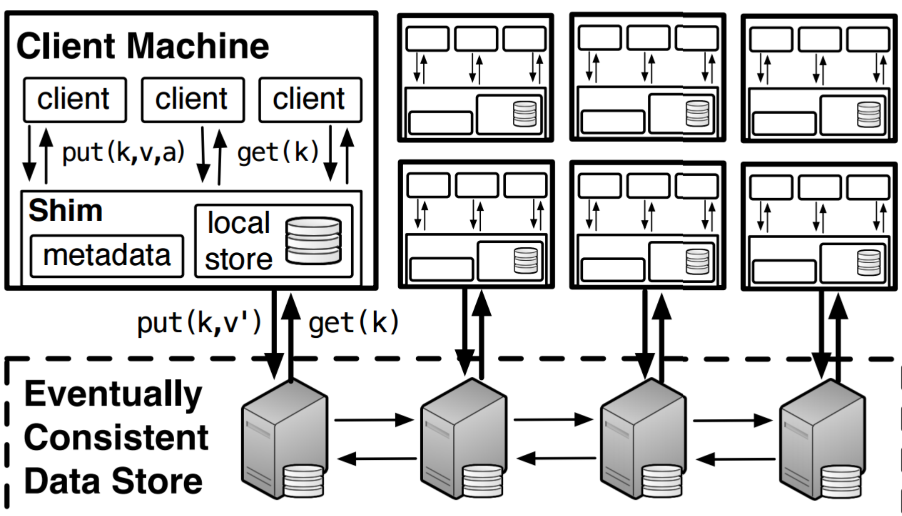


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

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

PRAM and Causal Consistency

- Combinations of previous session consistency guarantess
  - PRAM = MR + MW + RYW
  - Causal Consistency = PRAM + WFR
- All consistency level up to causal consistency can be guaranteed with **high availability**
- Example: Bolt-on causal consistency



Bailis, Peter, et al. "Bolt-on causal consistency." Proceedings of the 2013 ACM SIGMOD, 2013.

Bounded Staleness

- Either **time-based**:

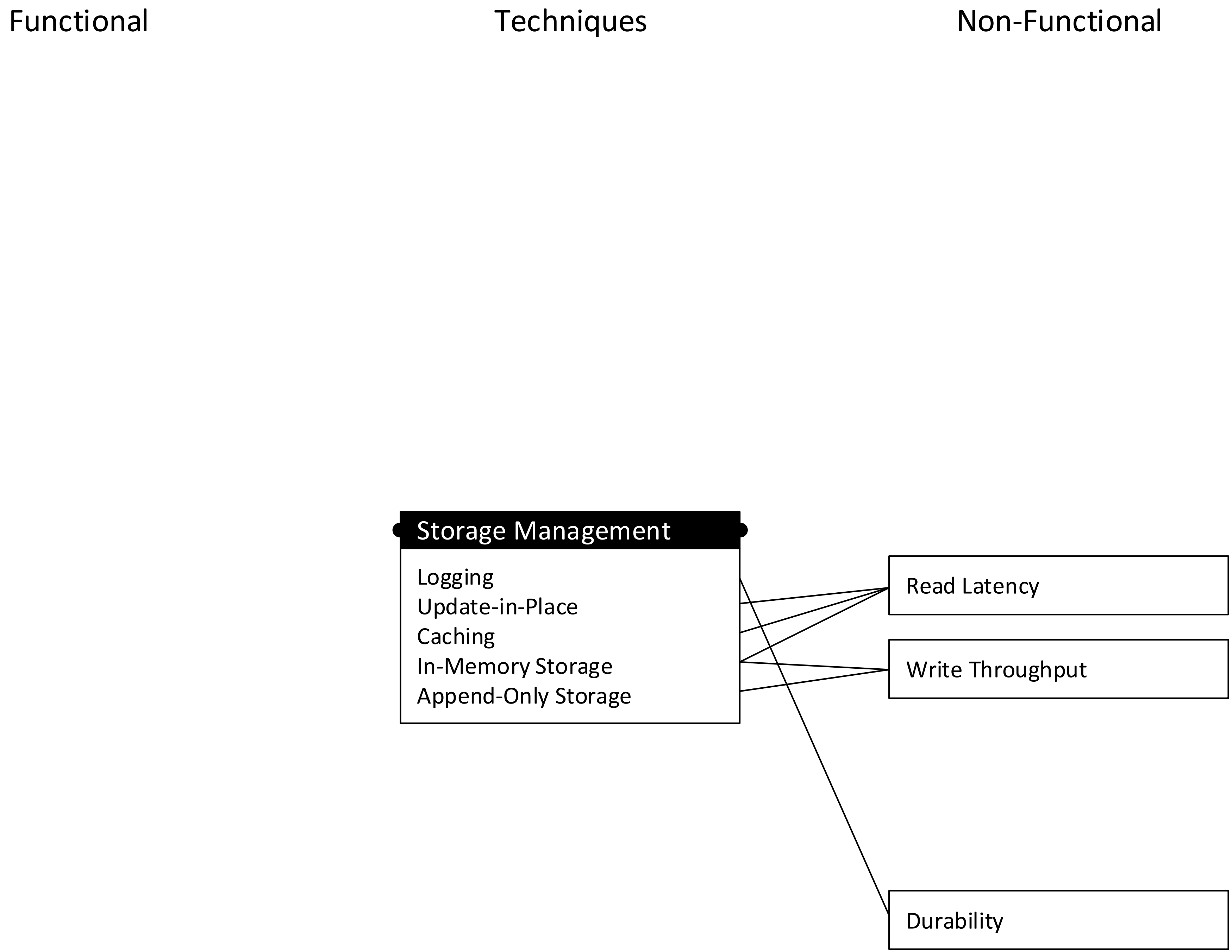
> **t-Visibility (Δ-atomicity):** the inconsistency window comprises at most t time units; that is, any value that is returned upon a read request was up to date t time units ago.

- 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.

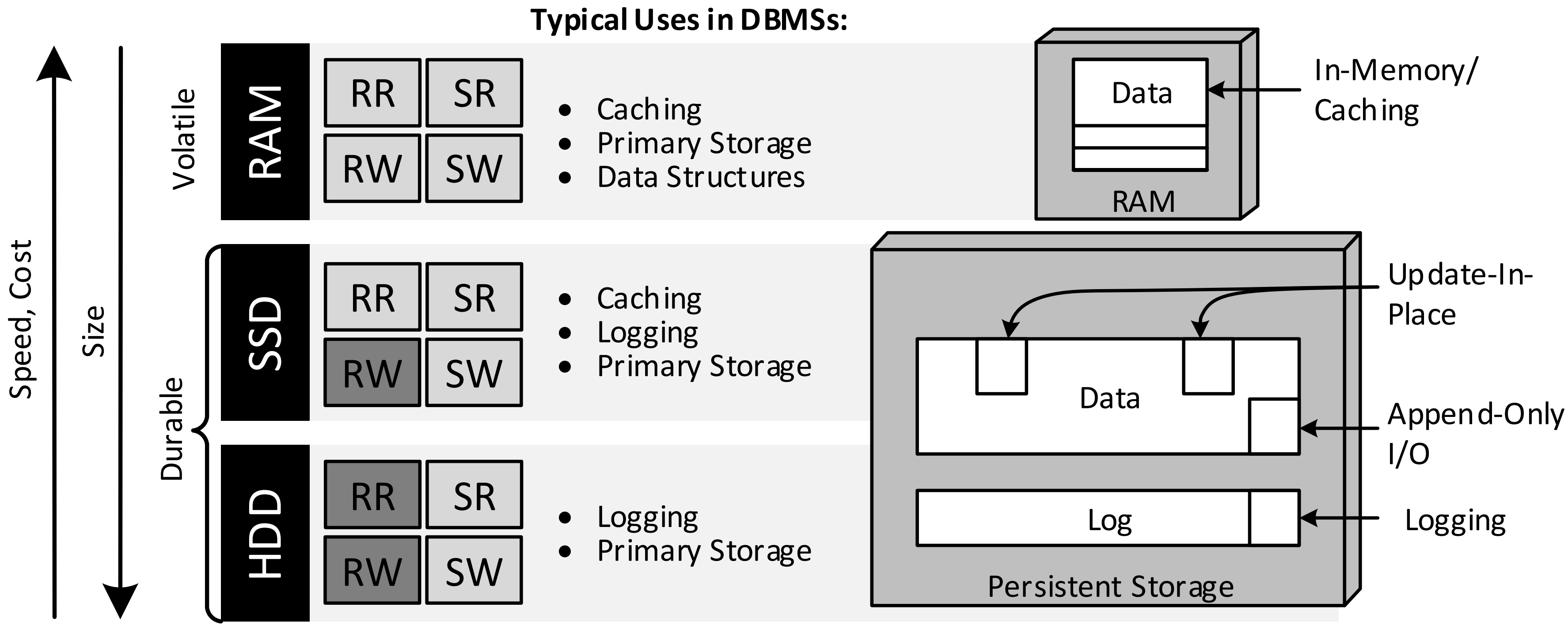
- Both are *not* achievable with high availability

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


Functional
Techniques
Non-Functional
Storage Management
Logging
Update-in-Place
Caching
In-Memory Storage
Append-Only Storage
Read Latency
Write Throughput
Durability

NoSQL Storage Management

In a Nutshell



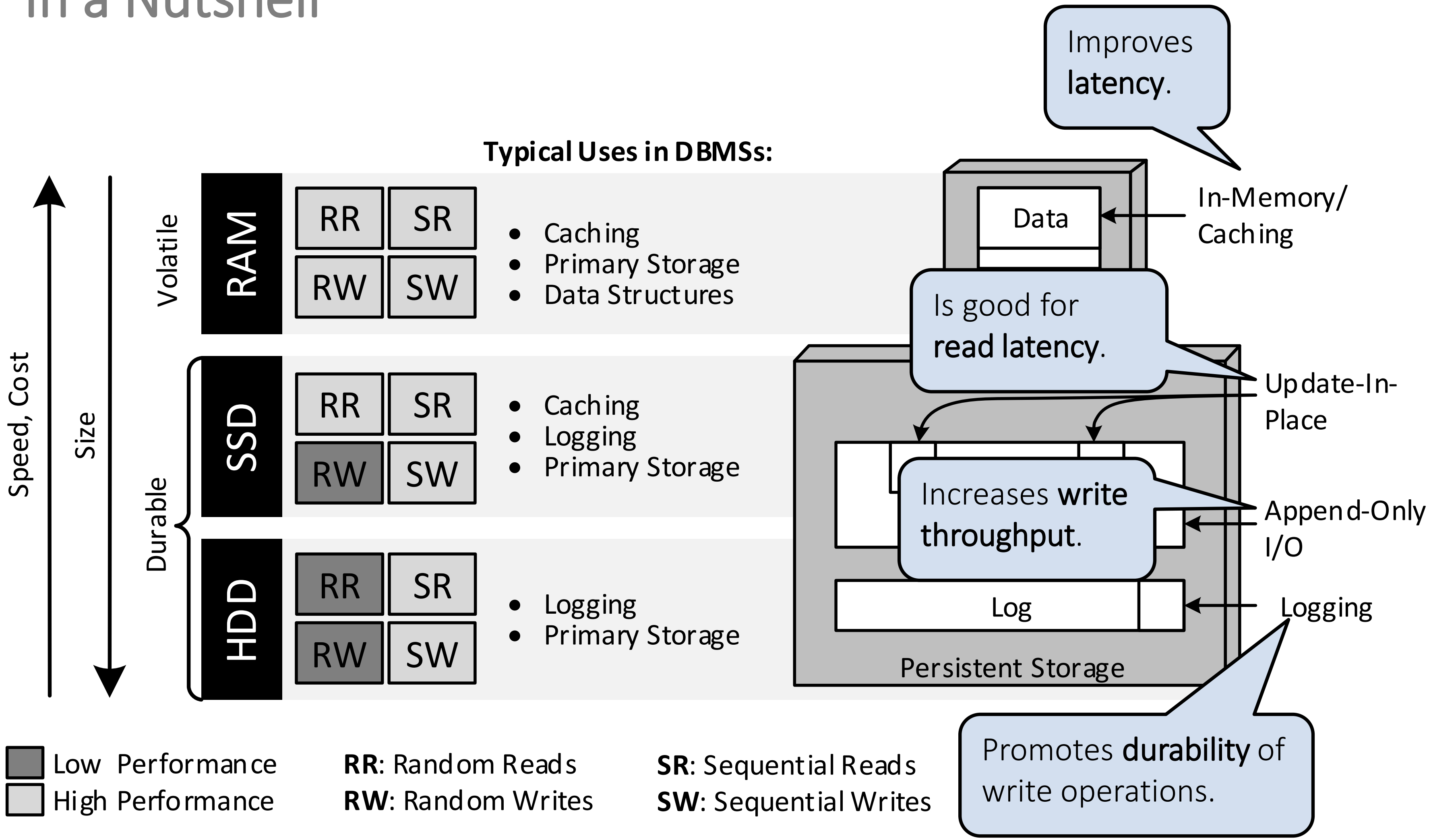
Low Performance
High Performance

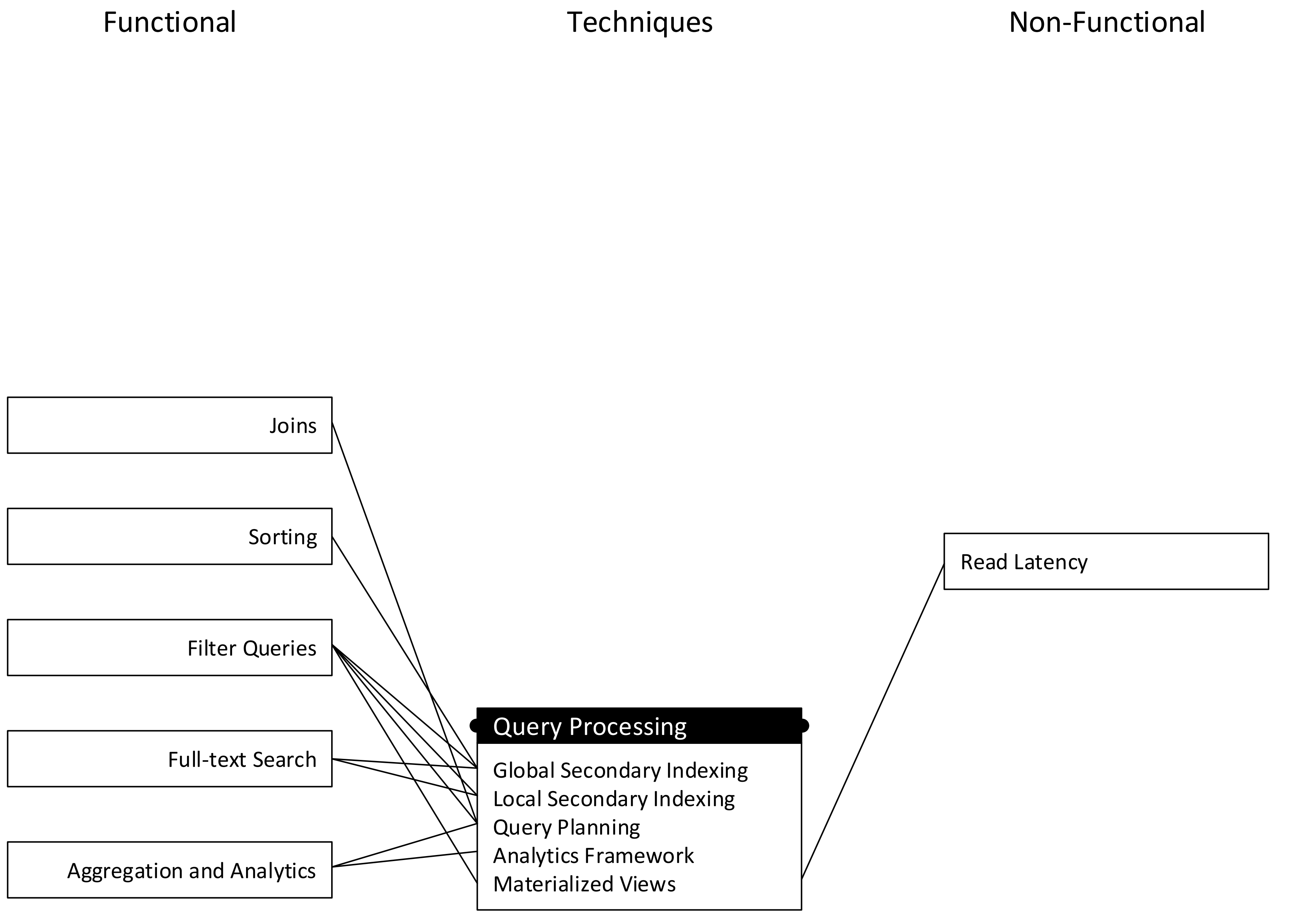
RR: Random Reads
RW: Random Writes

SR: Sequential Reads
SW: Sequential Writes

NoSQL Storage Management

In a Nutshell




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

Local Secondary Indexing

Partitioning By Document

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

Data

Key	Color
104	Yellow
188	Blue
192	Blue

Index

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

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

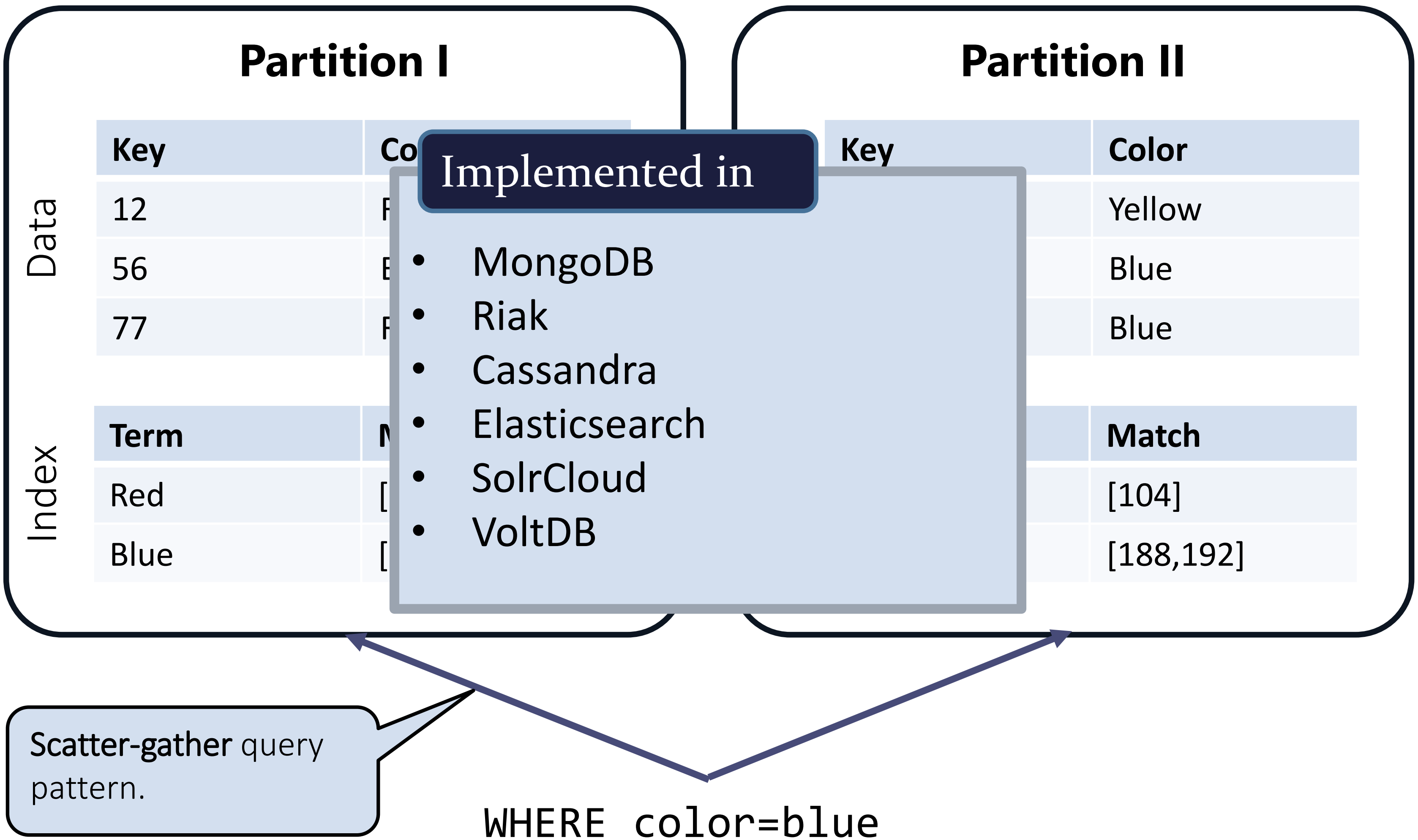
Scatter-gather query pattern.

`WHERE color=blue`

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

Local Secondary Indexing

Partitioning By Document



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	

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

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]

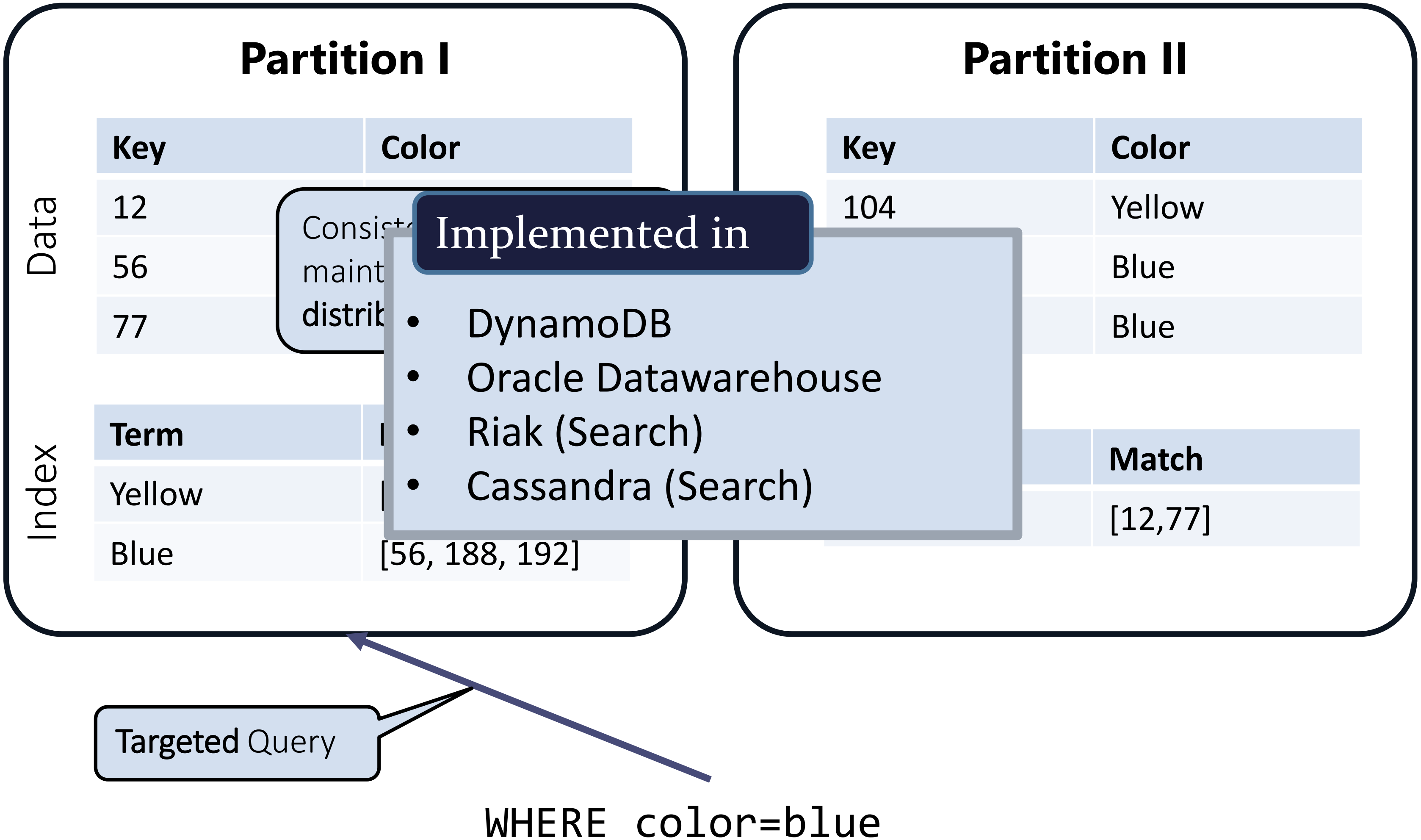
Targeted Query

```
WHERE color=blue
```

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

Global Secondary Indexing

Partitioning By Term

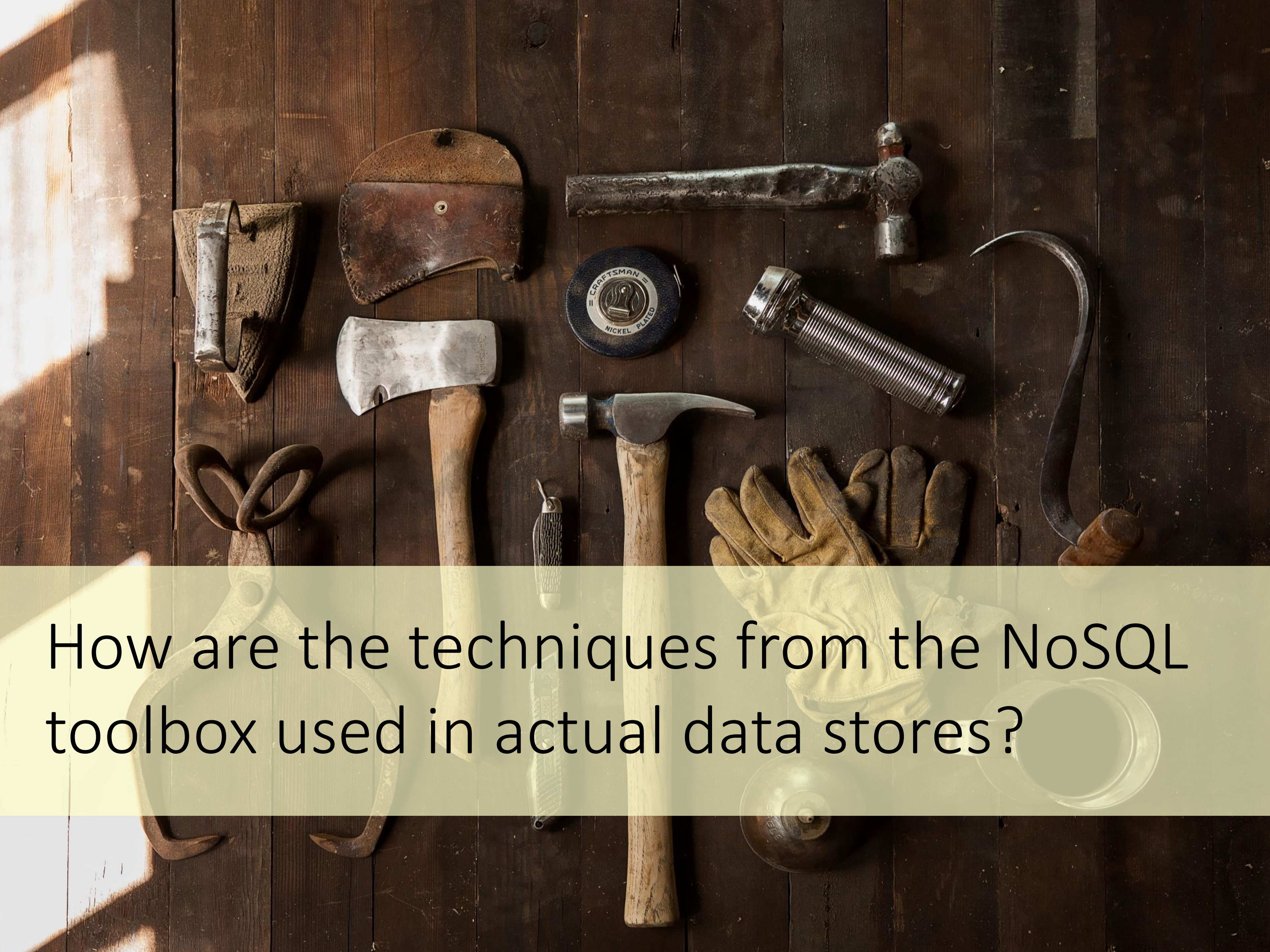


Kleppmann, Martin. "Designing data-intensive applications." (2016).

Query Processing Techniques

Summary

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



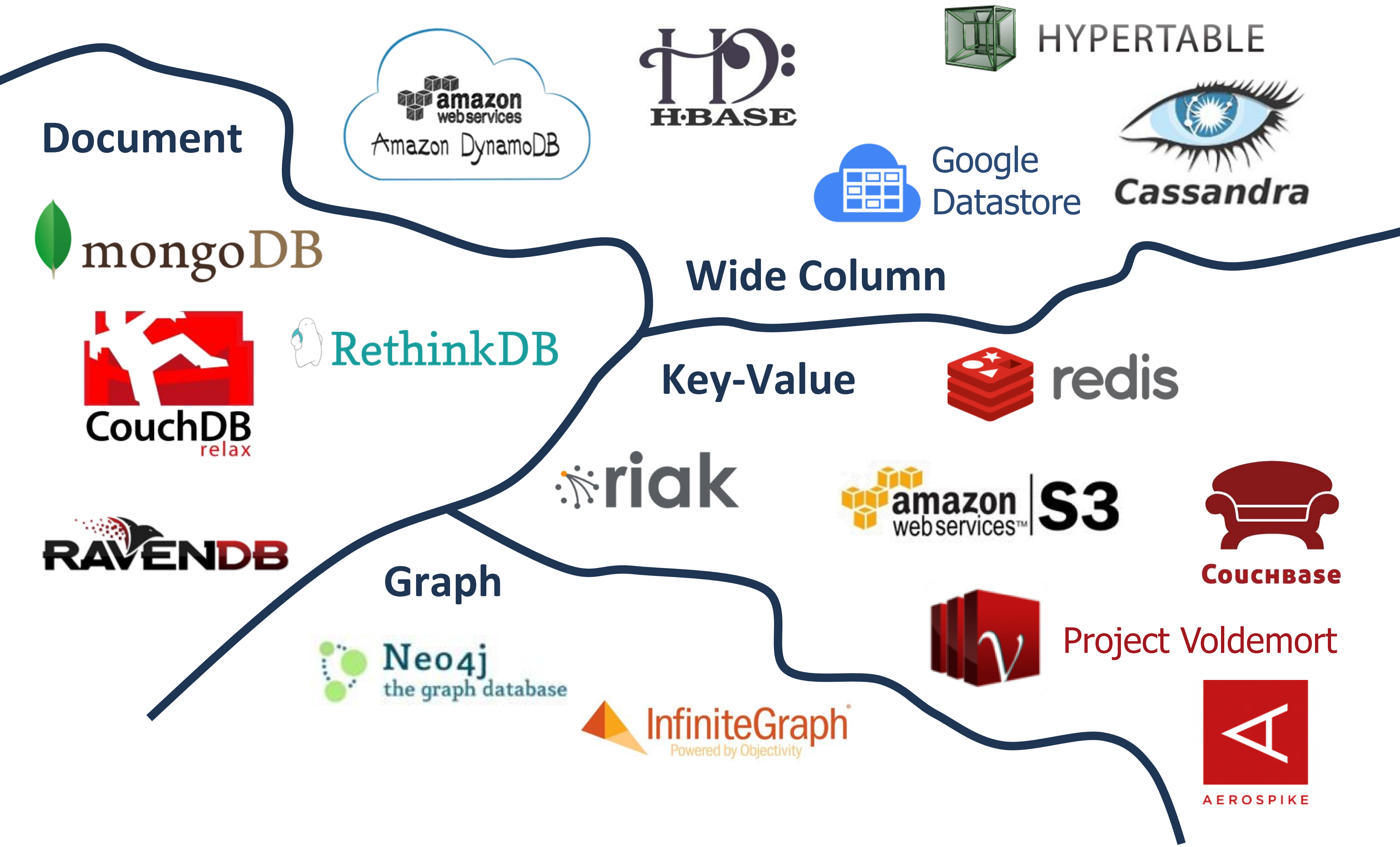
How are the techniques from the NoSQL toolbox used in actual data stores?

Outline

- NoSQL Foundations and Motivation
- The NoSQL Toolbox: Common Techniques
- NoSQL Systems & Decision Guidance
- Scalable Real-Time Databases and Processing

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

NoSQL Landscape



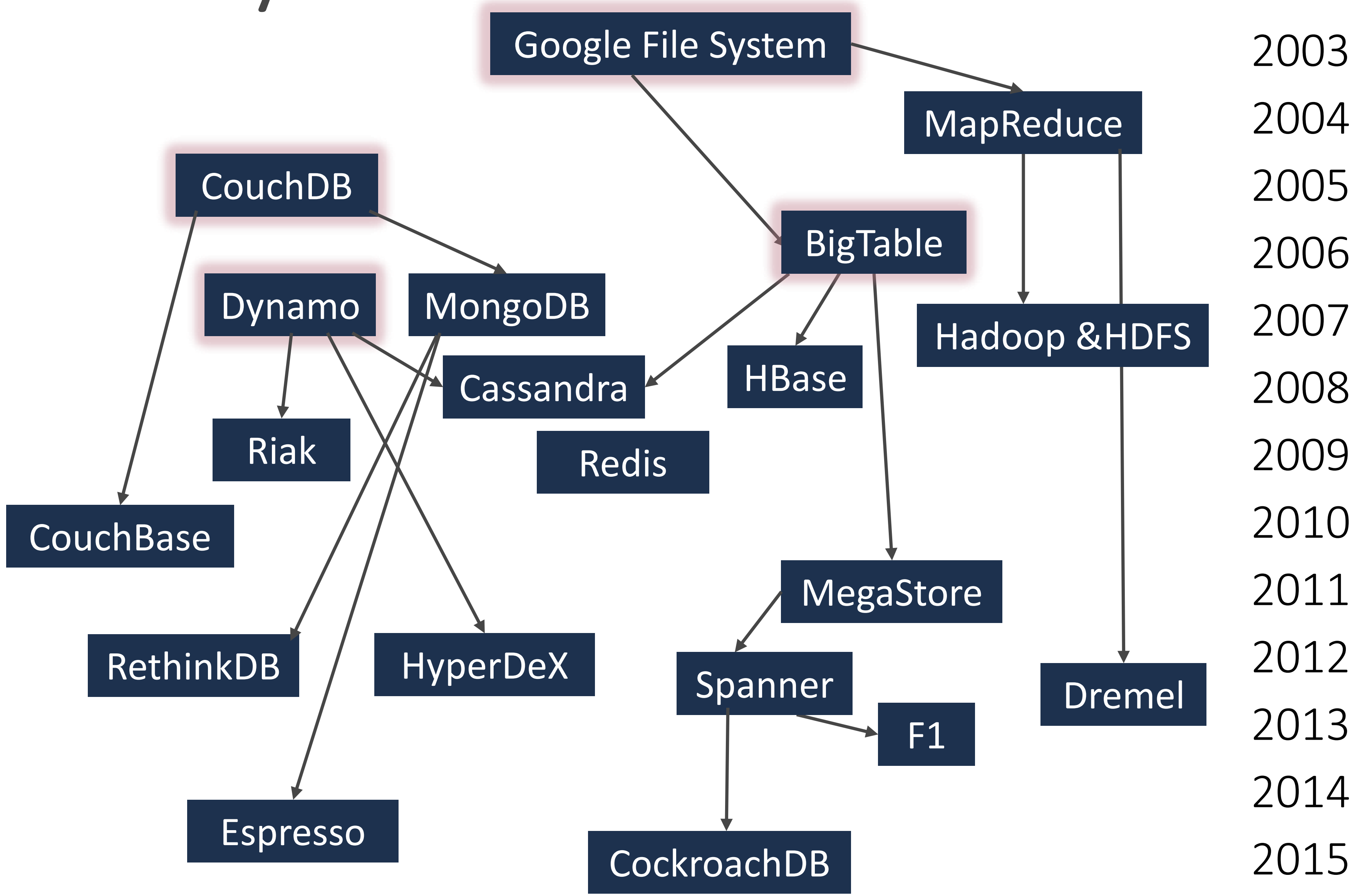
Popularity

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

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

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

History


Google File System
MapReduce
CouchDB
BigTable
Dynamo
MongoDB
Hadoop &HDFS
Cassandra
HBase
Riak
Redis
CouchBase
MegaStore
RethinkDB
HyperDeX
Spanner
Dremel
F1
Espresso
CockroachDB
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015

NoSQL foundations

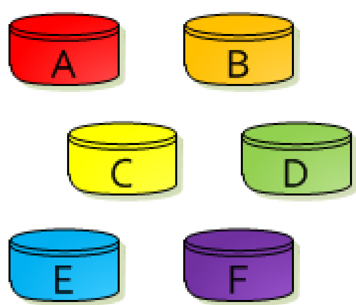
- **BigTable** (2006, Google)
  - Consistent, Partition Tolerant
  - Wide-Column data model
  - Master-based, fault-tolerant, large clusters (1.000+ Nodes), **HBase**, **Cassandra**, **HyperTable, Accumolo**
- **Dynamo** (2007, Amazon)
  - Available, Partition tolerant
  - Key-Value interface
  - Eventually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

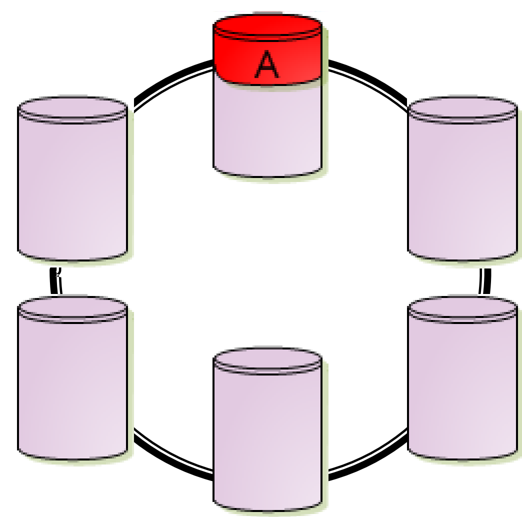
Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."

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

Dynamo (AP)

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

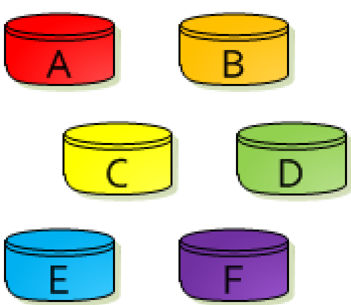


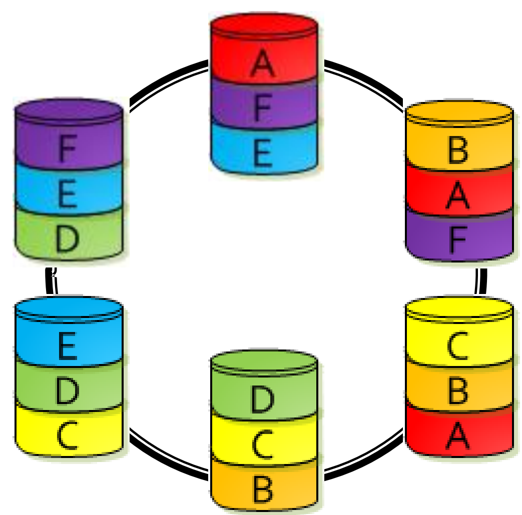


DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."

Dynamo (AP)

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

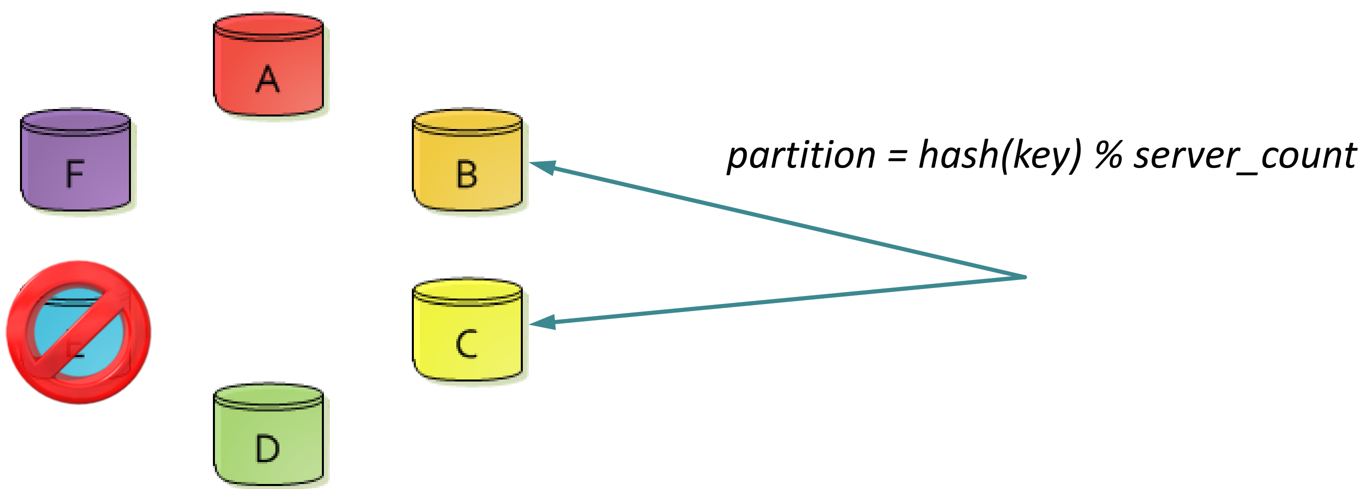




DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."

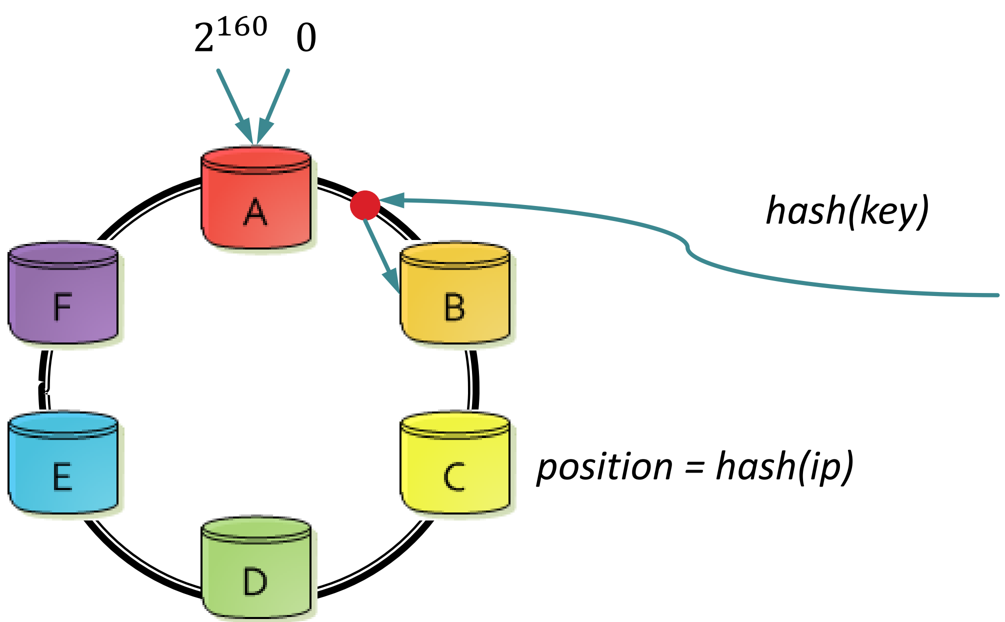
Consistent Hashing

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



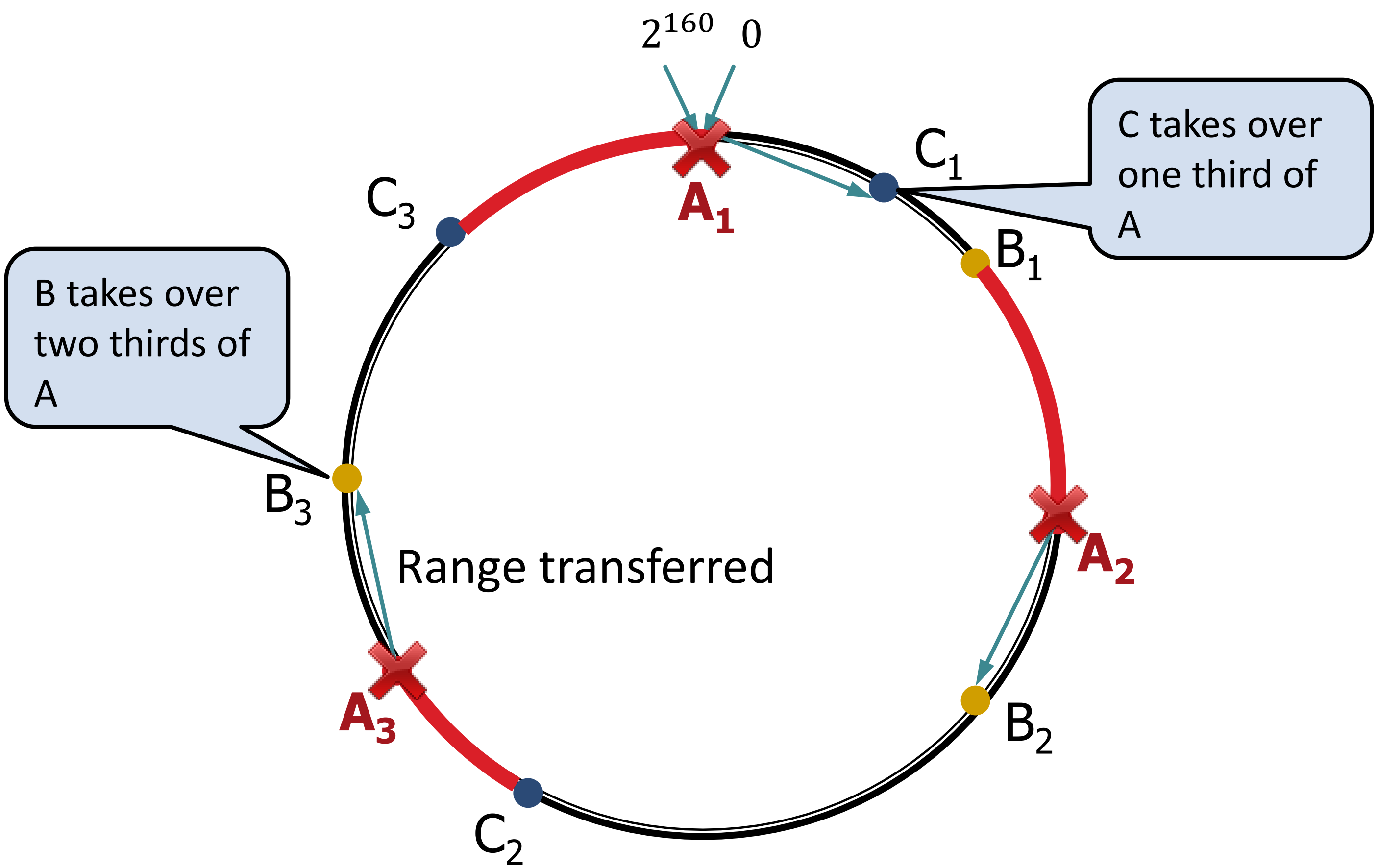
Consistent Hashing

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



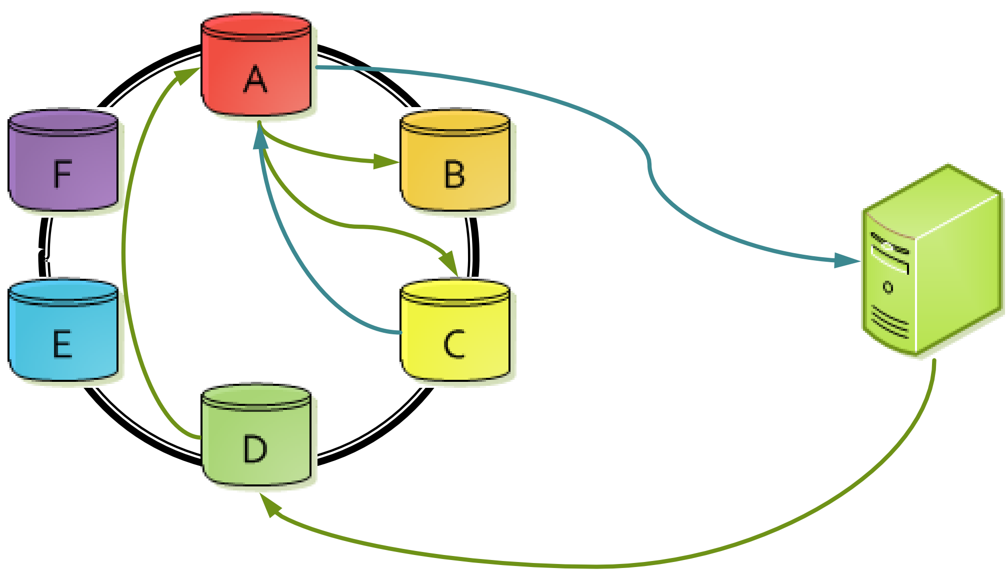
Consistent Hashing

- Extension: **Virtual Nodes** for Load Balancing



Reading
Parameters R, W, N

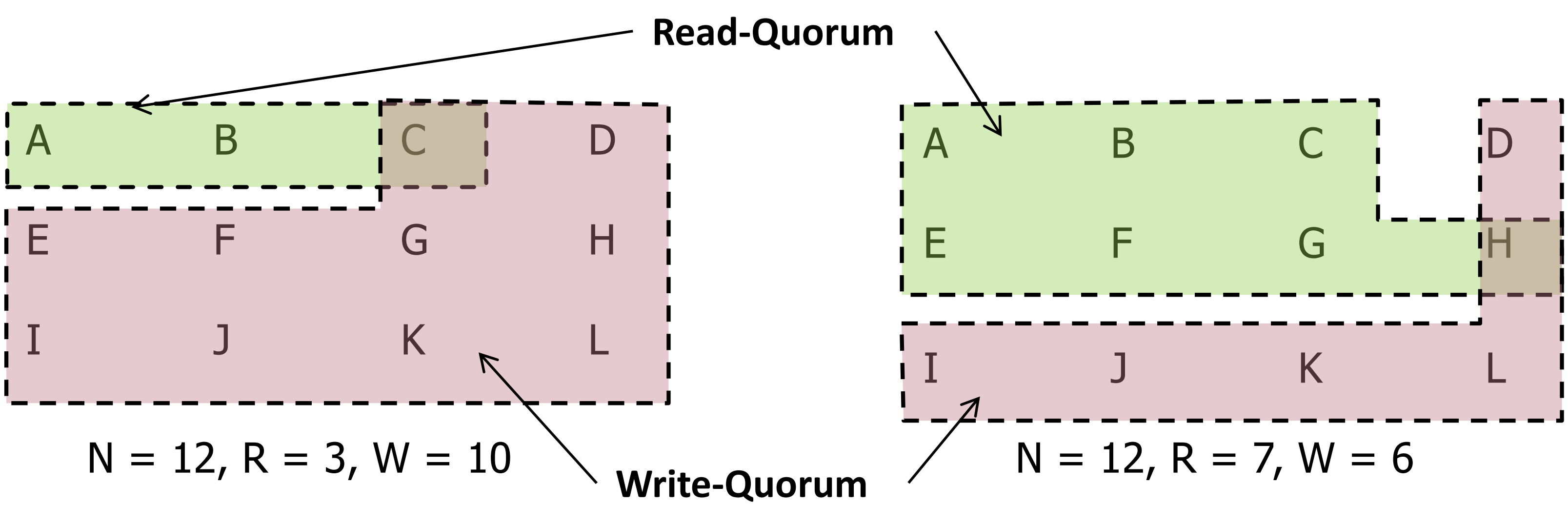
- An arbitrary node acts as a coordinator
- **N**: number of replicas
- **R**: number of nodes that need to confirm a read
- **W**: number of nodes that need to confirm a write



N=3
R=2
W=1

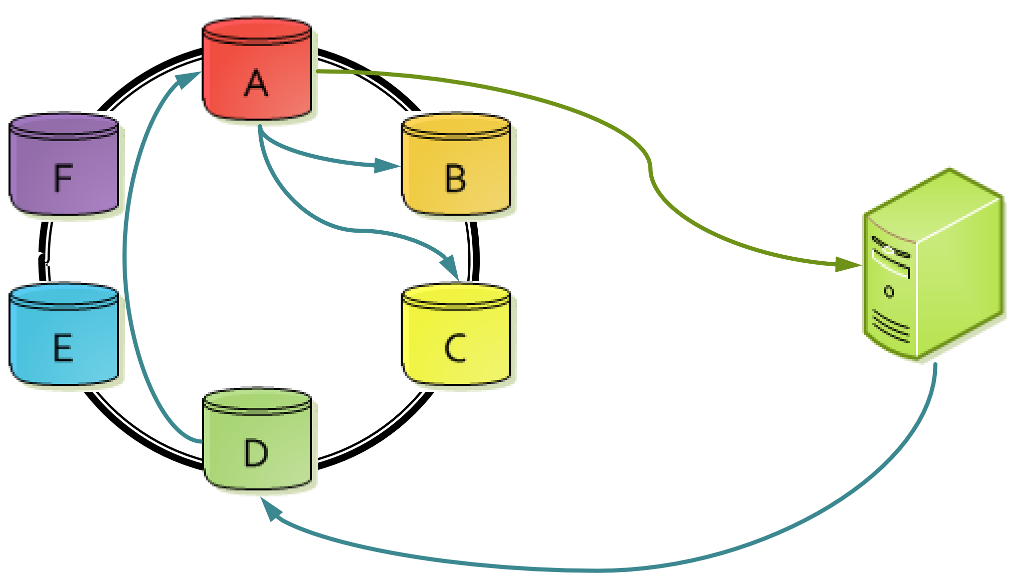
Quorums

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



Writing

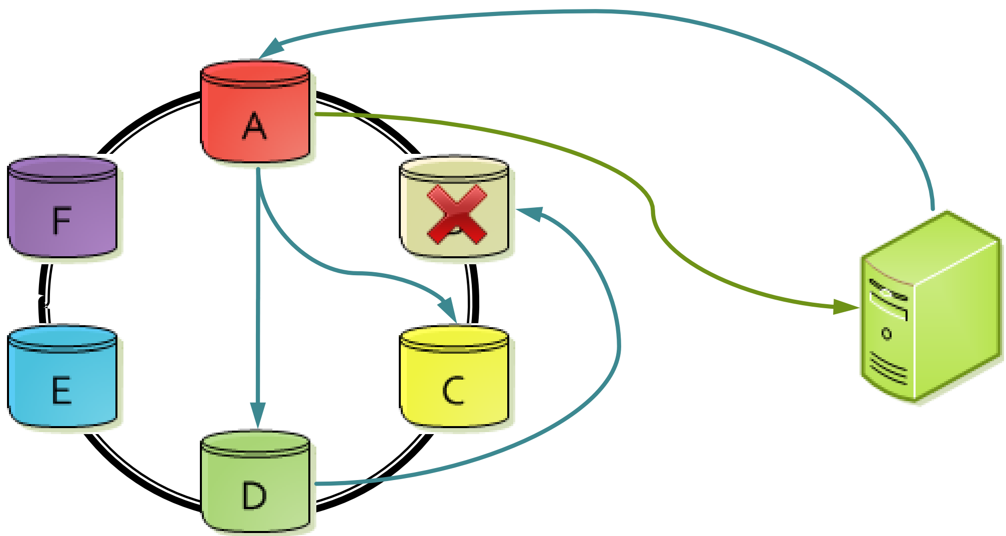
- **W** Servers have to acknowledge



N=3
R=2
W=1

Hinted Handoff

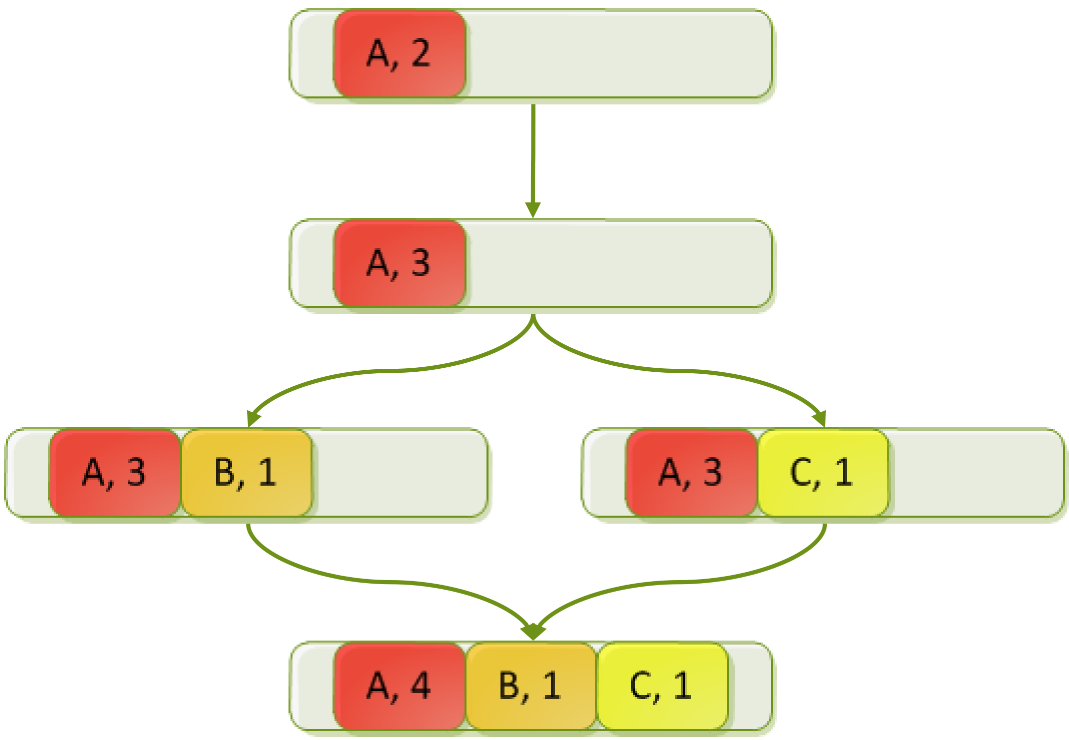
- Next node in the ring may take over, until original node is available again:



N=3
R=2
W=1

Vector clocks

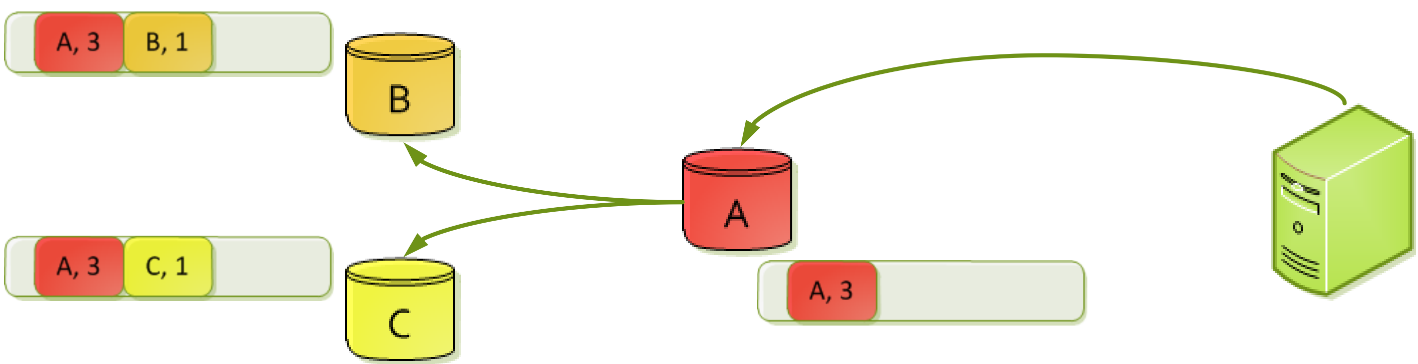
- Dynamo uses **Vector Clocks** for versioning



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

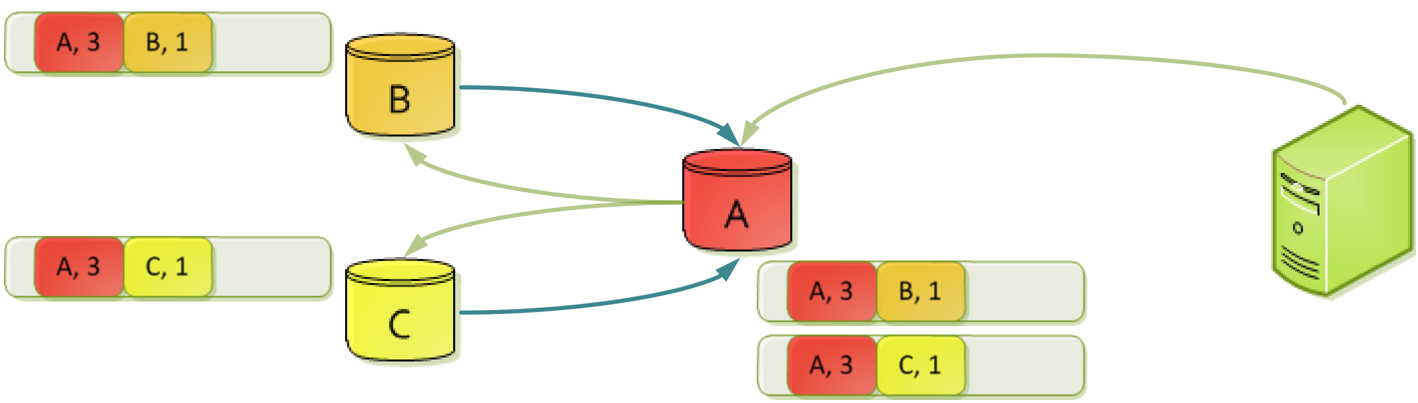
Versioning and Consistency

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



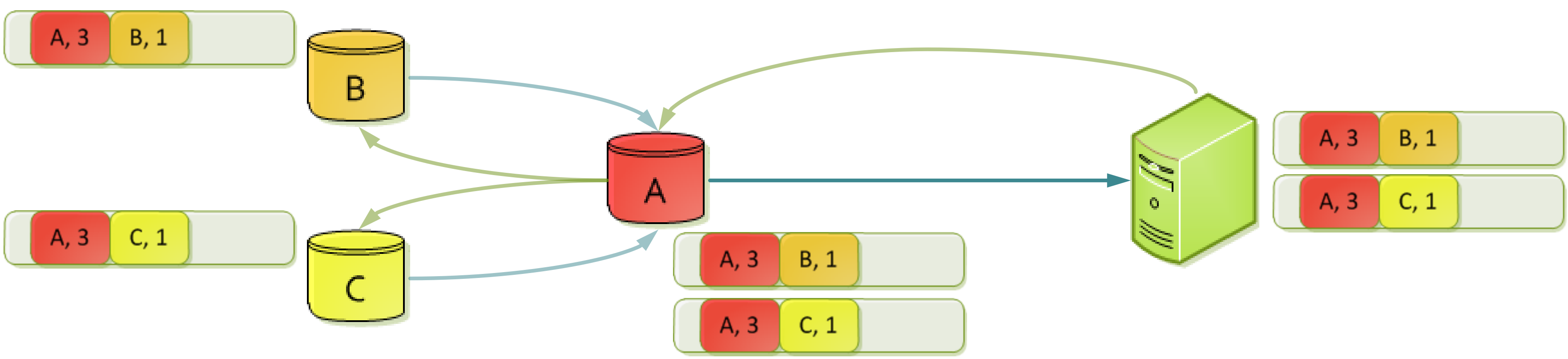
Versioning and Consistency

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



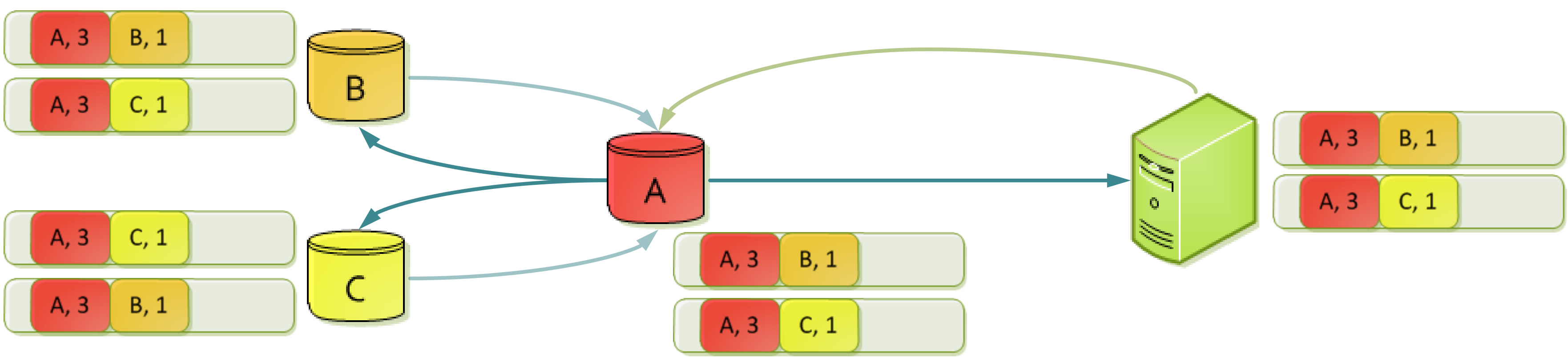
Versioning and Consistency

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



Versioning and Consistency

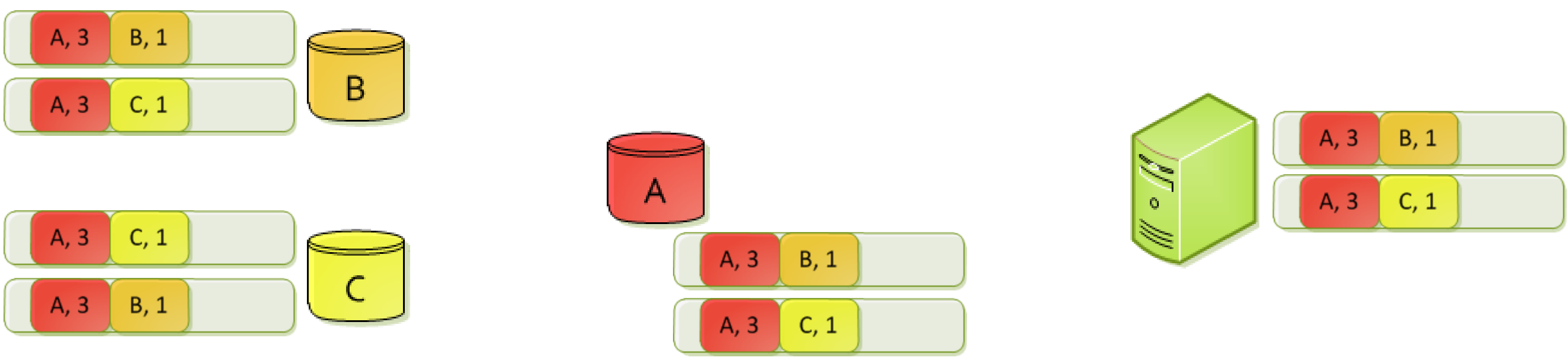
- $R + W \leq N \Rightarrow$ no consistency guarantee
- $R + W > N \Rightarrow$ newest acked value included in reads
- **Vector Clocks** used for versioning



Read Repair

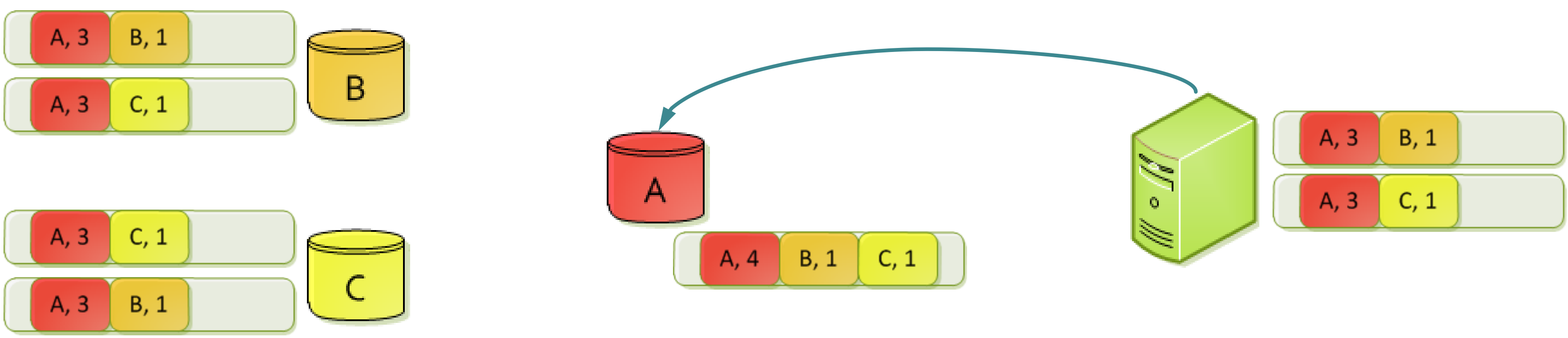
Conflict Resolution

- The application merges data when writing (*Semantic Reconciliation*)



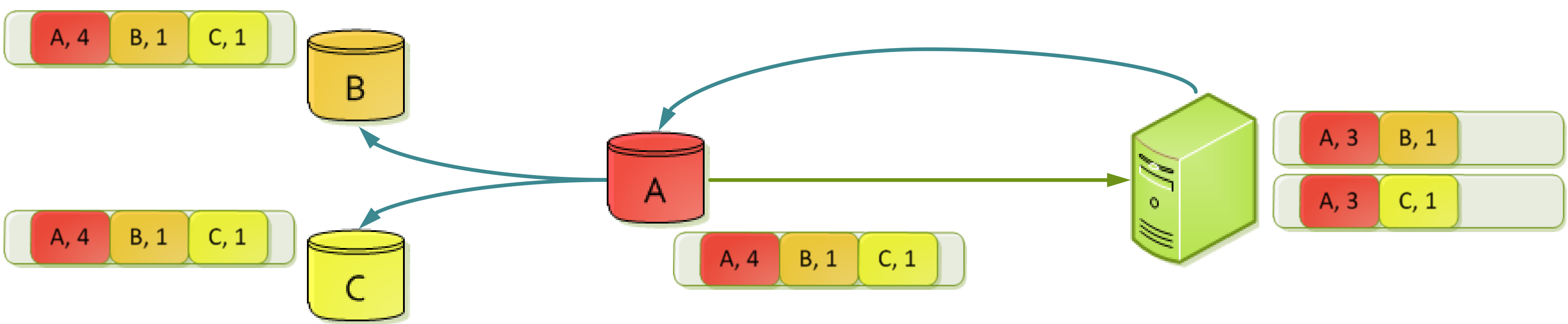
Conflict Resolution

- The application merges data when writing (*Semantic Reconciliation*)



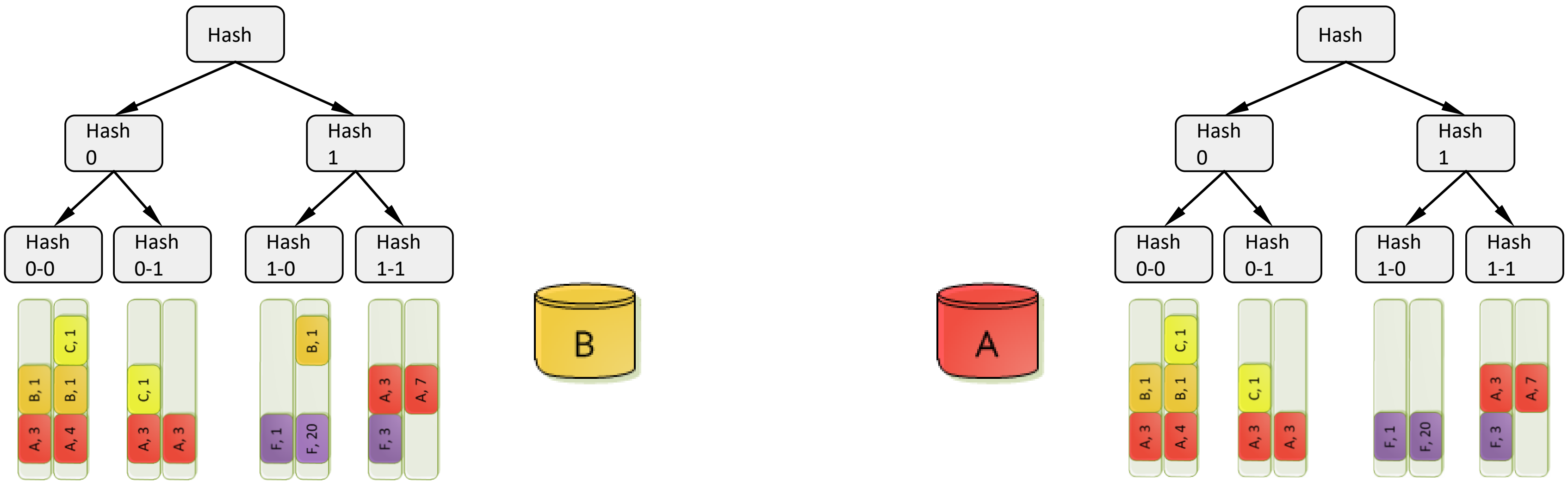
Conflict Resolution

- The application merges data when writing (*Semantic Reconciliation*)



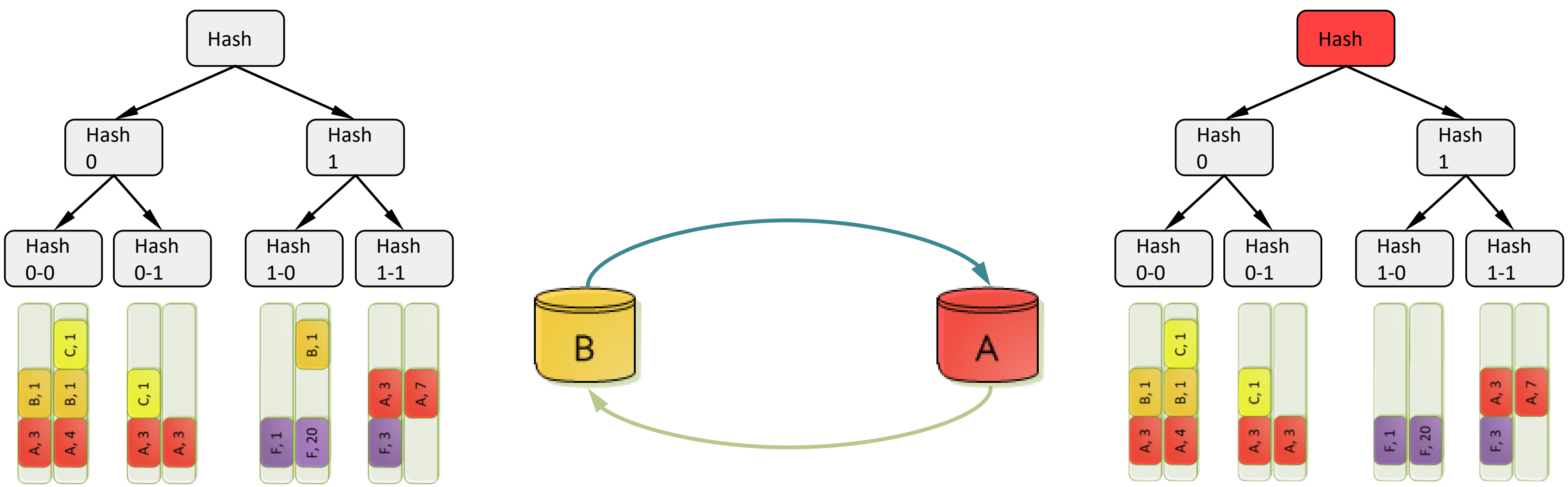
Merkle Trees: Anti-Entropy

- Every Second: Contact random server and compare



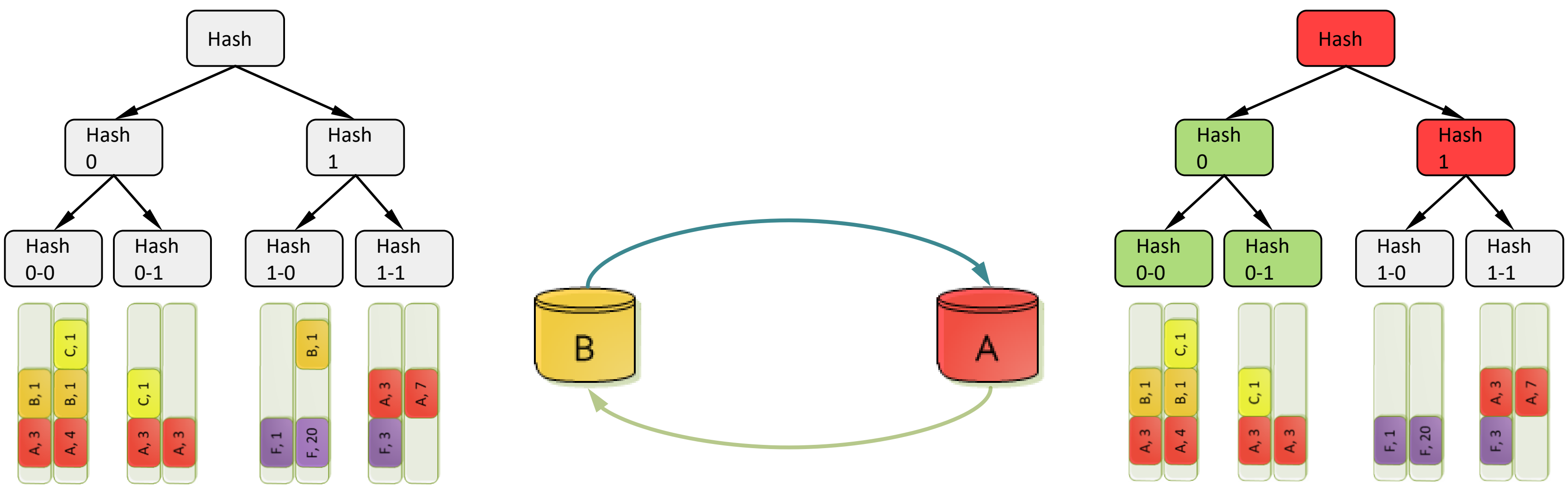
Merkle Trees: Anti-Entropy

- Every Second: Contact random server and compare



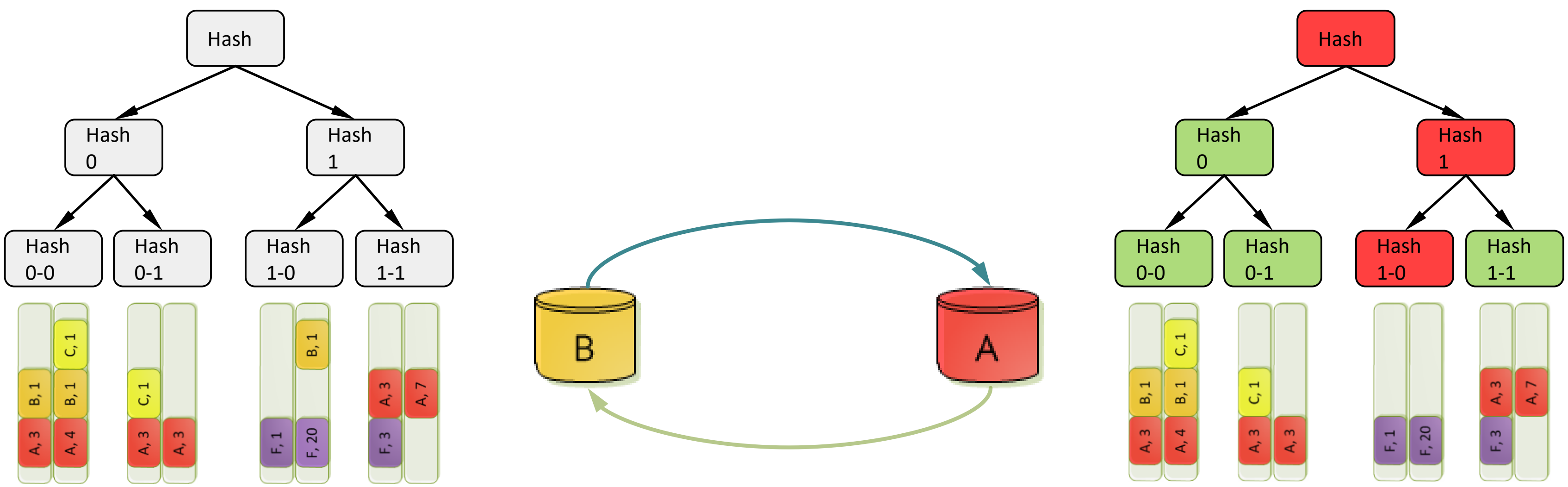
Merkle Trees: Anti-Entropy

- Every Second: Contact random server and compare



Merkle Trees: Anti-Entropy

- Every Second: Contact random server and compare



Quorum

- Typical Configurations:

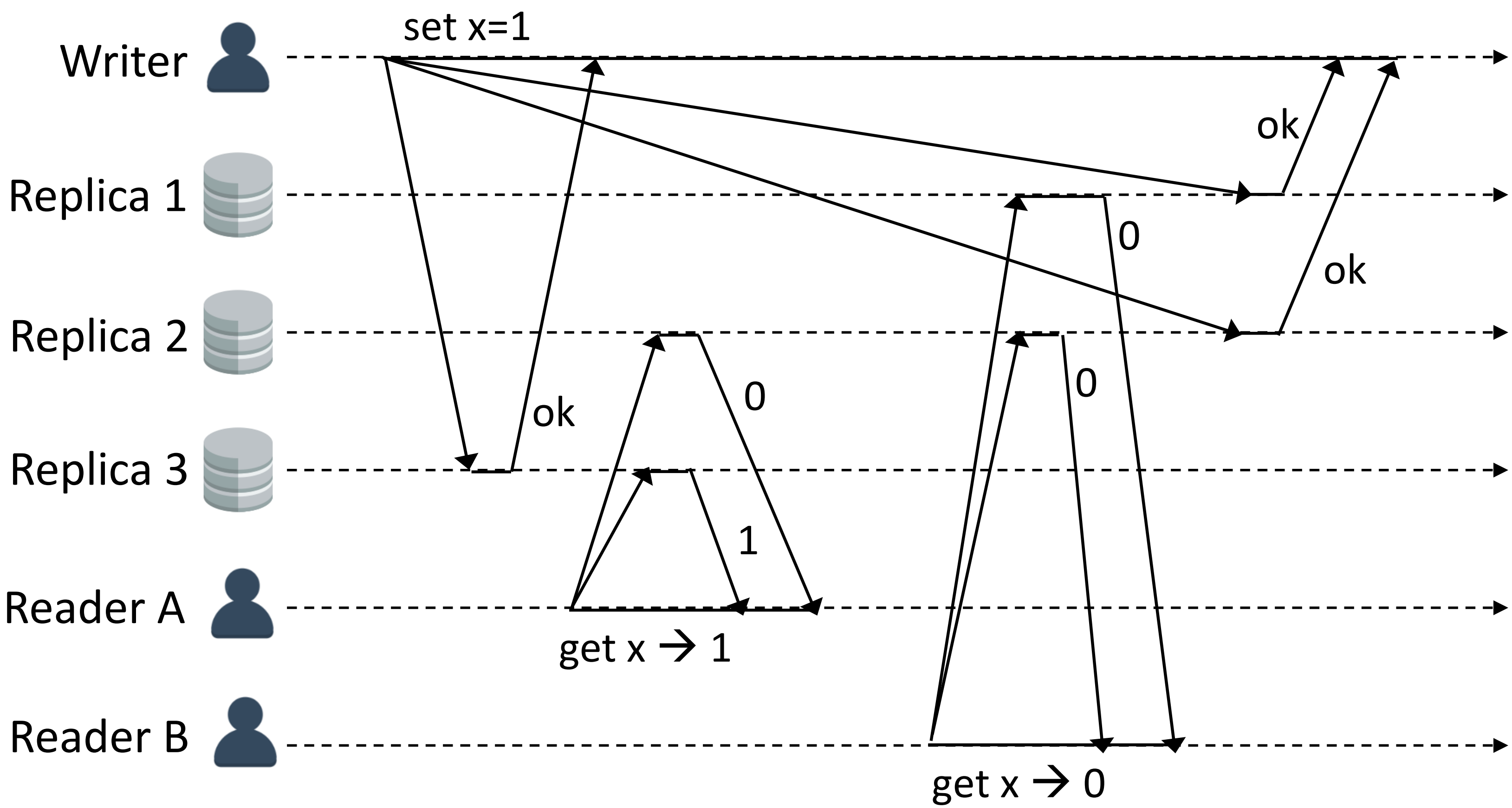
Performance (Cassandra Default)	N=3, R=1, W=1
Quorum, fast Writing:	N=3, R=3, W=1
Quorum, fast Reading	N=3, R=1, W=3
Trade-off (Riak Default)	N=3, R=2, W=2

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

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

$R + W > N$ does not imply linearizability

- Consider the following execution:

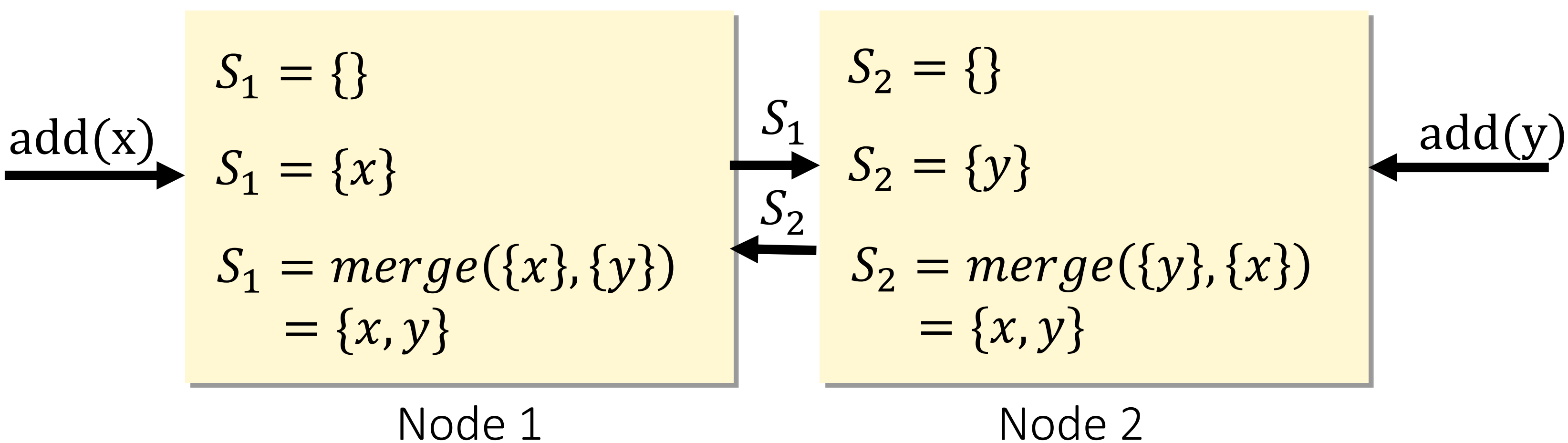


Kleppmann, Martin. "Designing data-intensive applications." (2016).

CRDTs

Convergent/Commutative Replicated Data Types

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

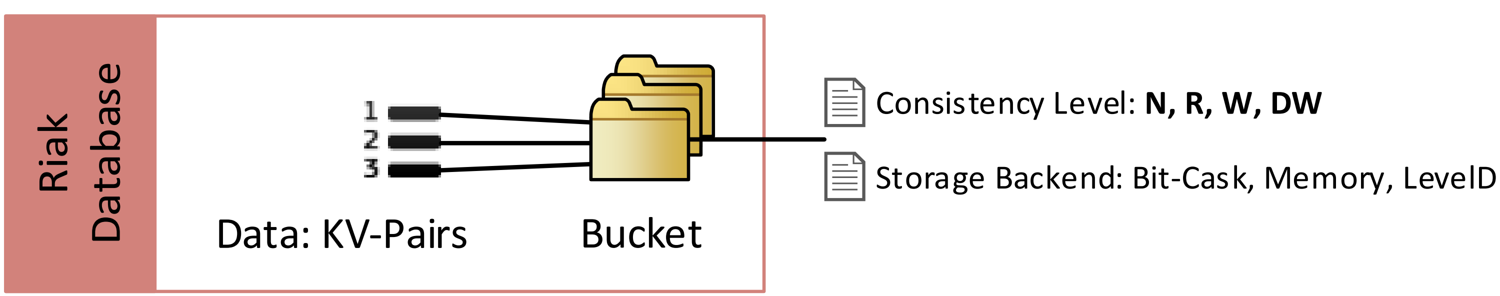


Marc Shapiro, Nuno Preguica, Carlos Baquero, and Marek Zawirski "Conflict-free Replicated Data Types"

Riak (AP)

- 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

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



Riak Data Types

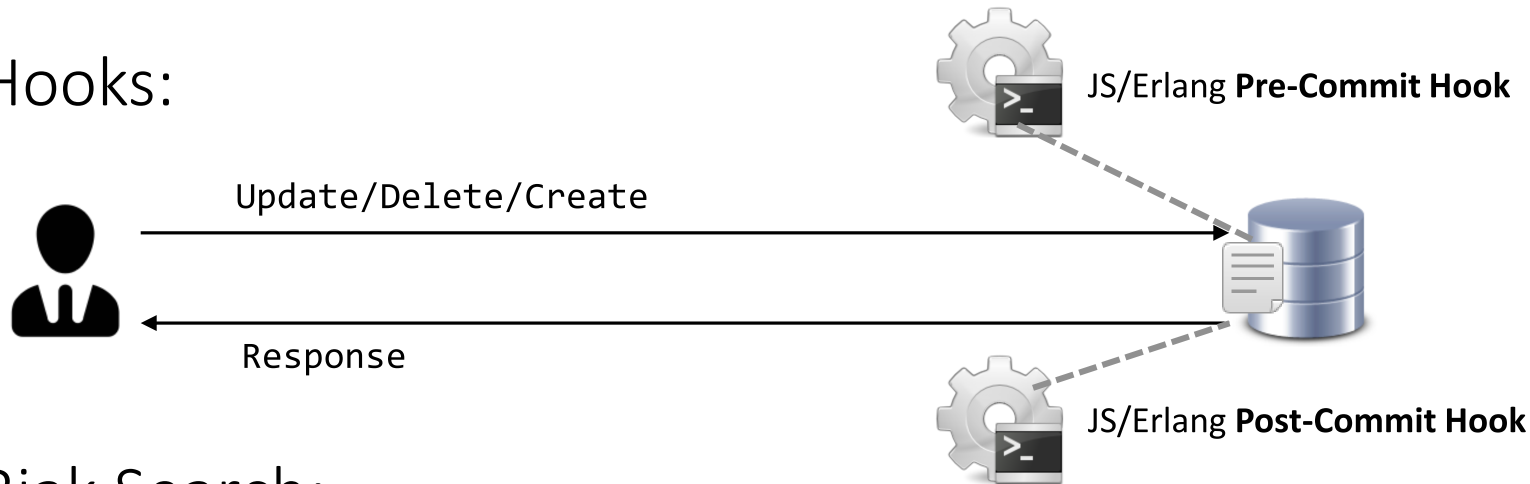
- Implemented as *state-based CRDTs*:

Data Type	Convergence rule
Flags	enable wins over disable
Registers	The most chronologically recent value wins, based on timestamps
Counters	Implemented as a PN-Counter, so all increments and decrements are eventually applied.
Sets	If an element is concurrently added and removed, the add will win
Maps	If a field is concurrently added or updated and removed, the add/update will win

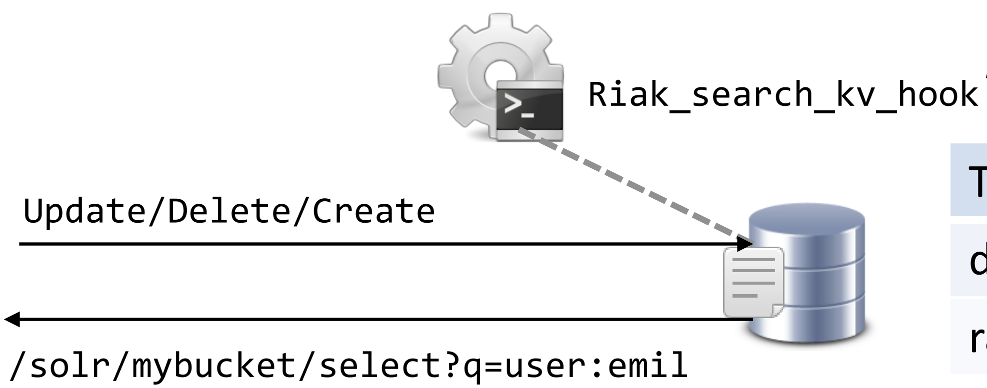
http://docs.basho.com/riak/kv/2.1.4/learn/concepts/crdts/

Hooks & Search

- Hooks:



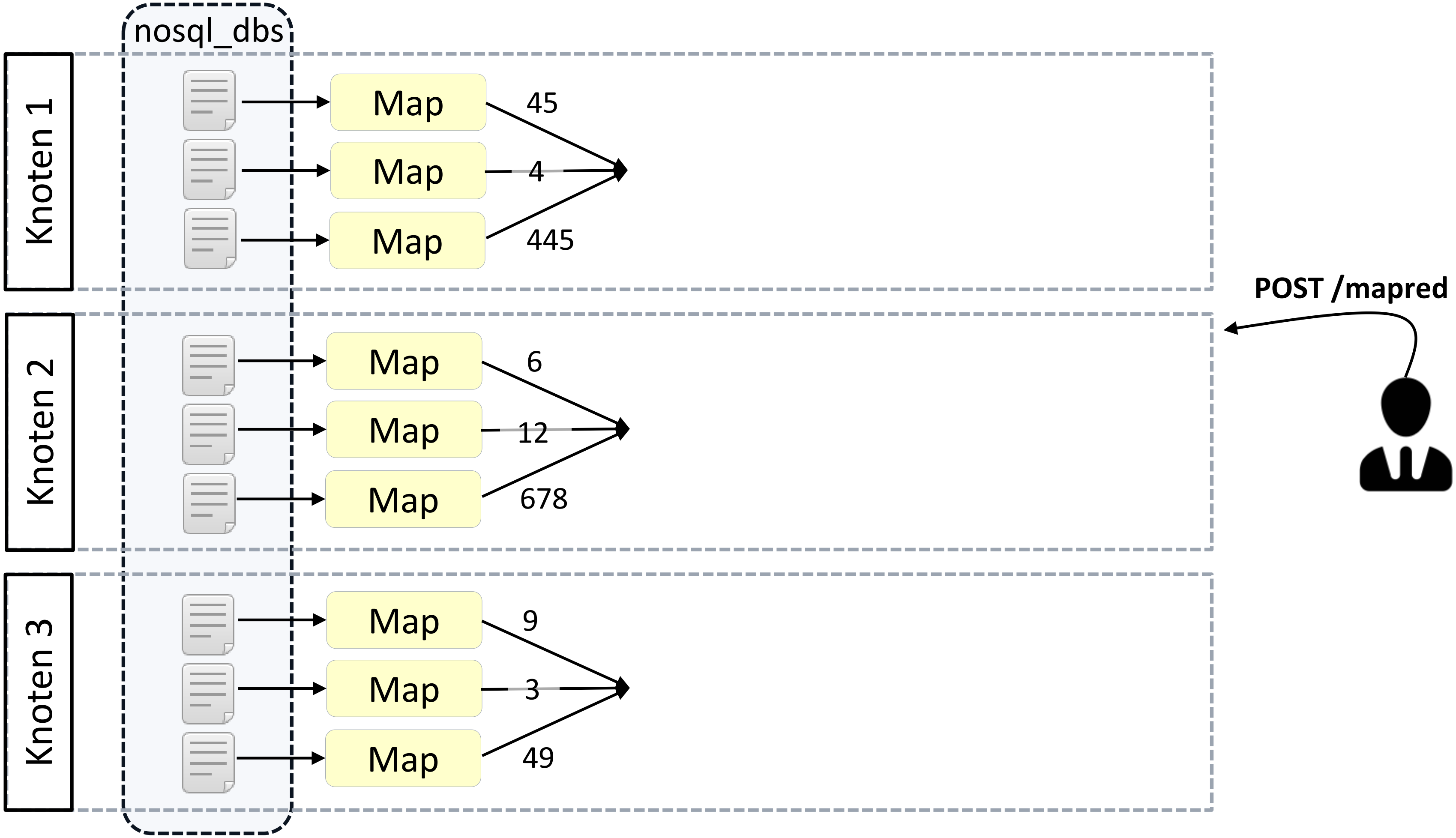
- Riak Search:



Term	Dokument
database	3,4,1
rabbit	2

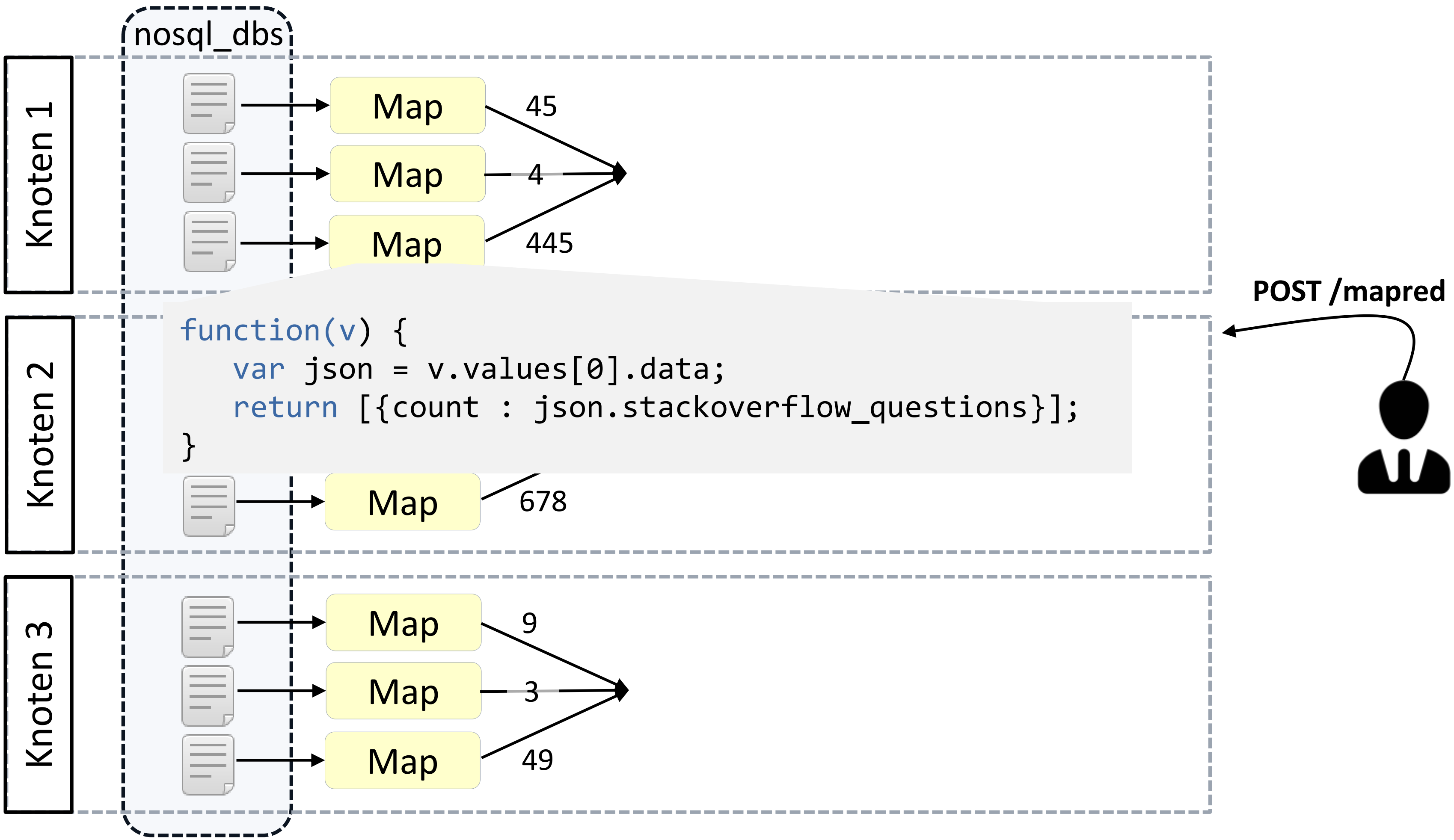
Search Index

Riak Map-Reduce



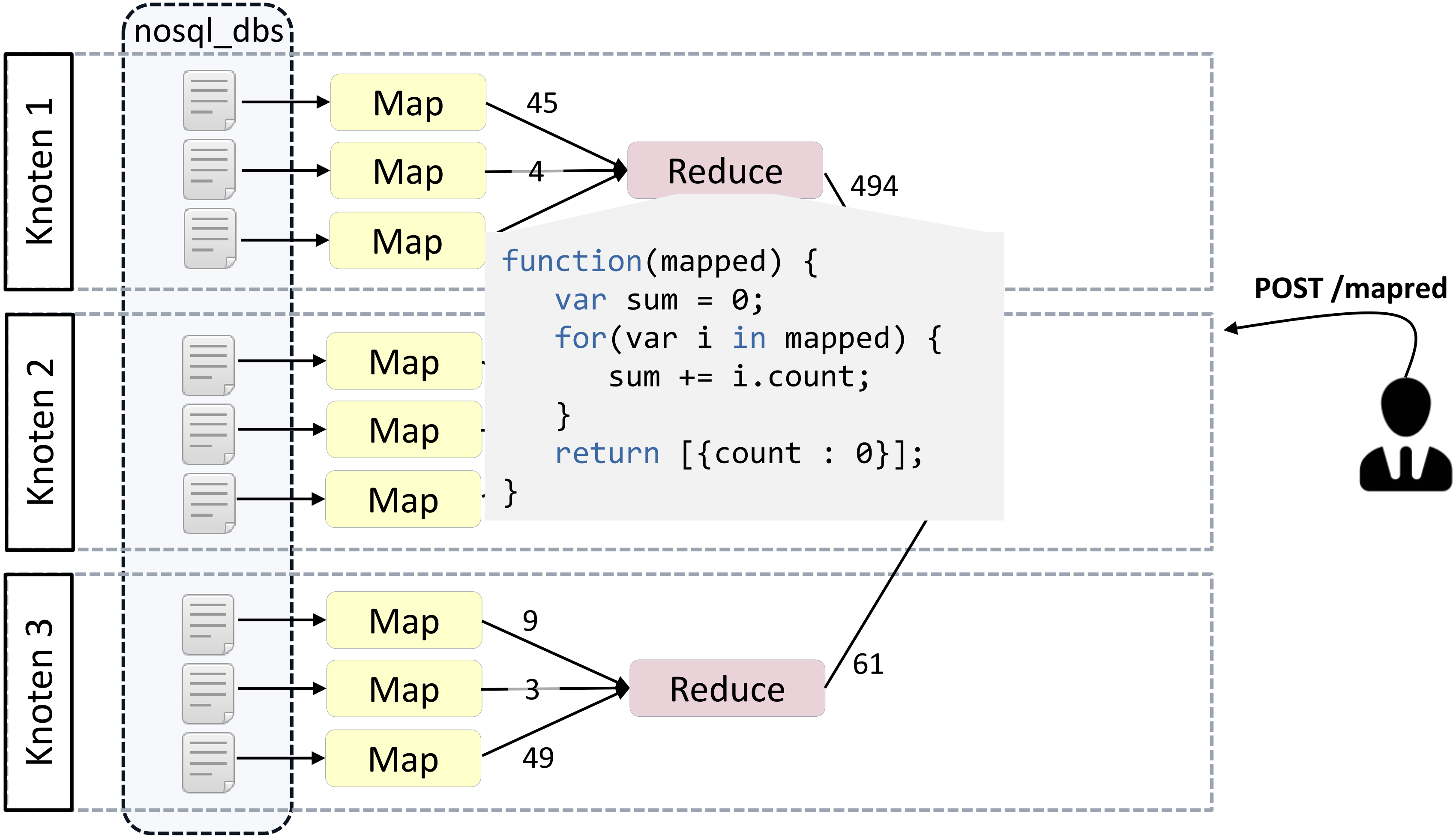
http://docs.basho.com/riak/latest/tutorials/querying/MapReduce/

Riak Map-Reduce



http://docs.basho.com/riak/latest/tutorials/querying/MapReduce/

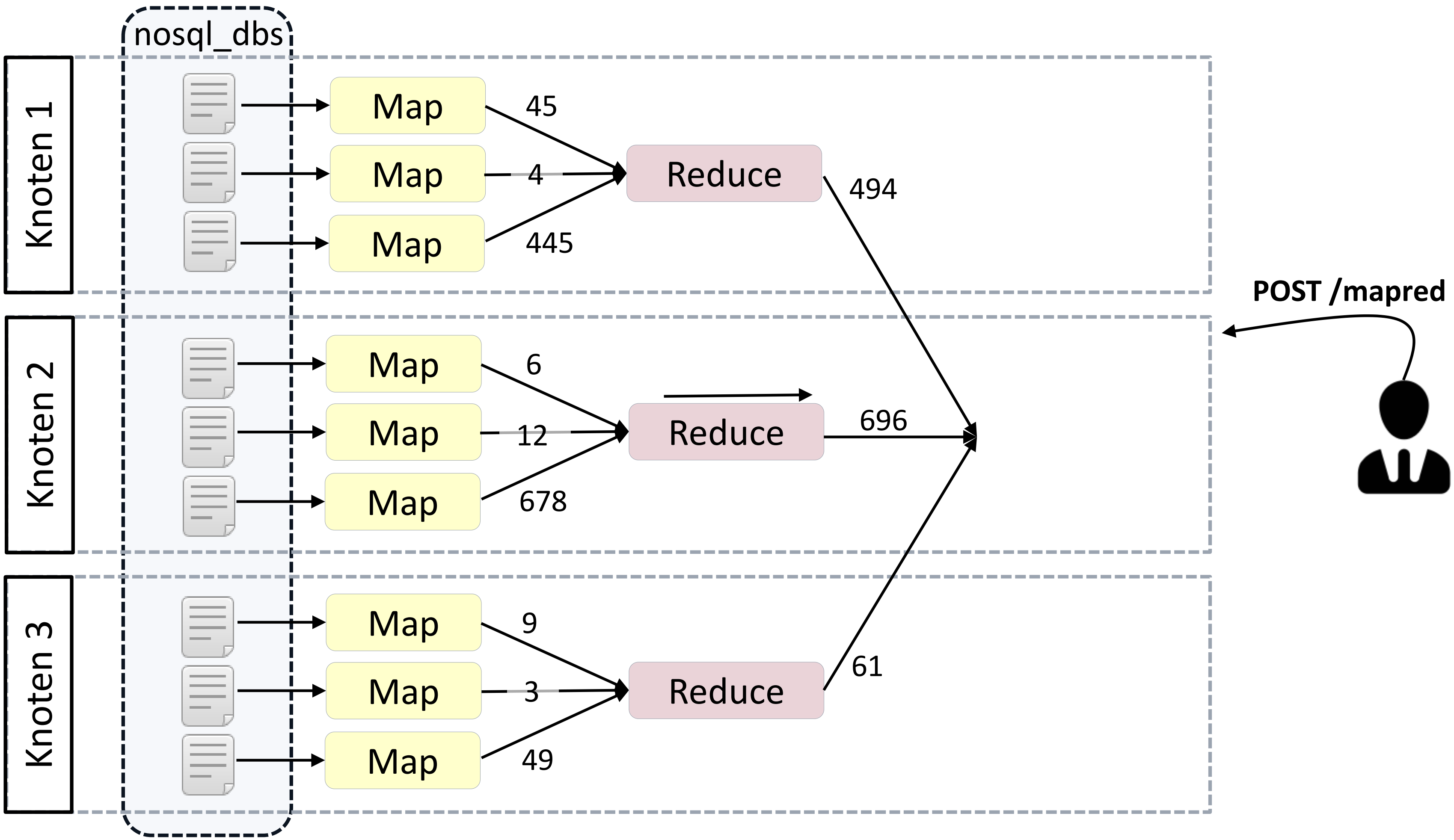
Riak Map-Reduce




```
function(mapped) {
   var sum = 0;
   for(var i in mapped) {
      sum += i.count;
   }
   return [{count : 0}];
}
```

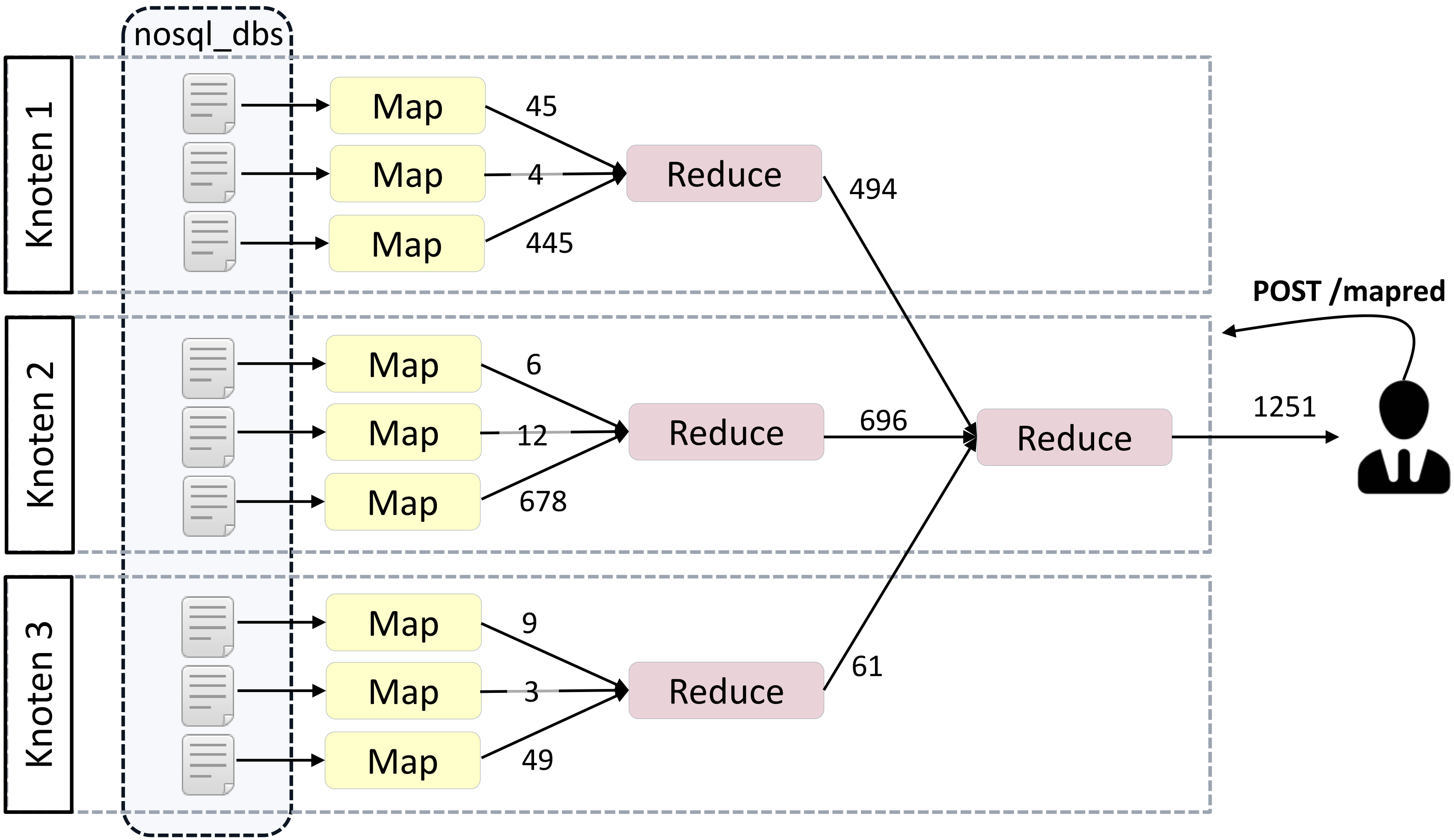
http://docs.basho.com/riak/latest/tutorials/querying/MapReduce/

Riak Map-Reduce



http://docs.basho.com/riak/latest/tutorials/querying/MapReduce/

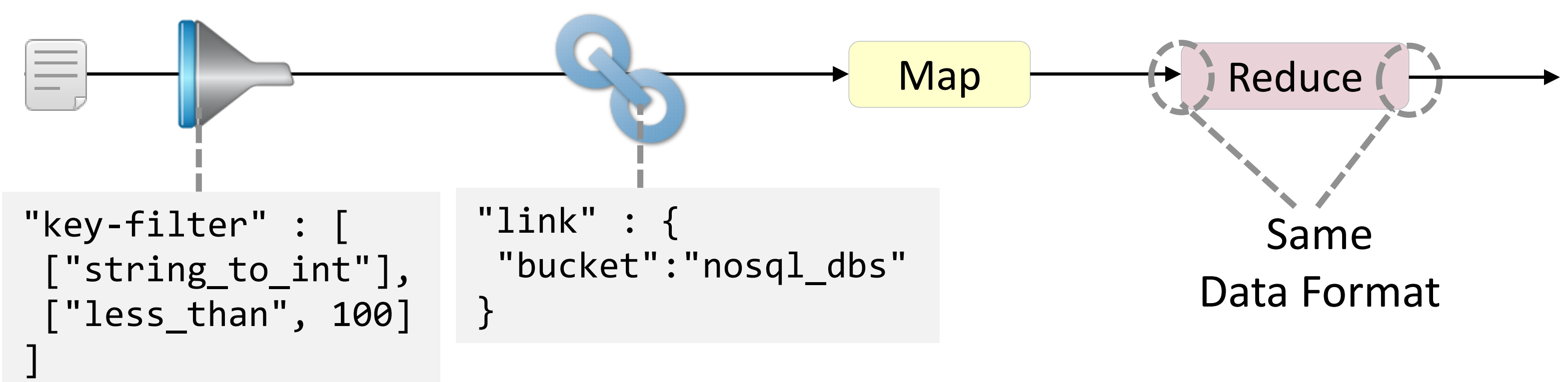
Riak Map-Reduce



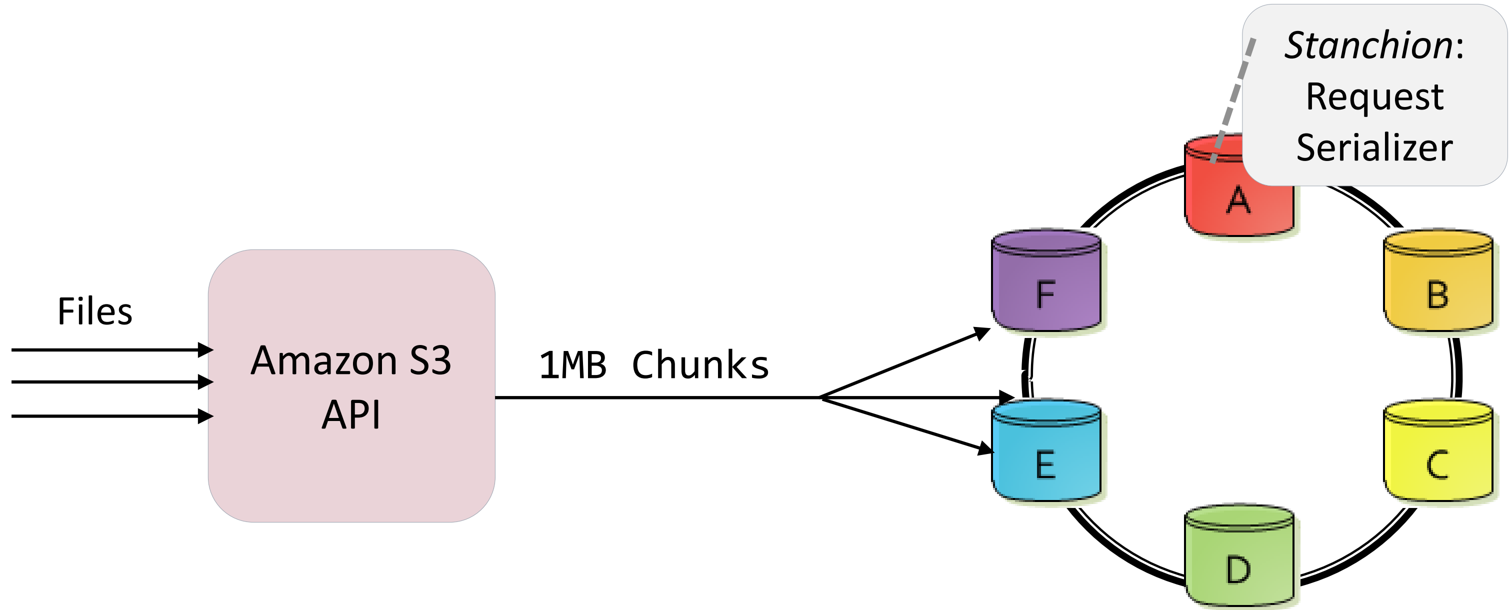
http://docs.basho.com/riak/latest/tutorials/querying/MapReduce/

Riak Map-Reduce

- JavaScript/Erlang, stored/ad-hoc
- Pattern: Chainable Reducers
- **Key-Filter**: Narrow down input
- **Link Phase**: Resolves links



Riak Cloud Storage



Summary: Dynamo and Riak

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

Dynamo and Riak

Classification

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

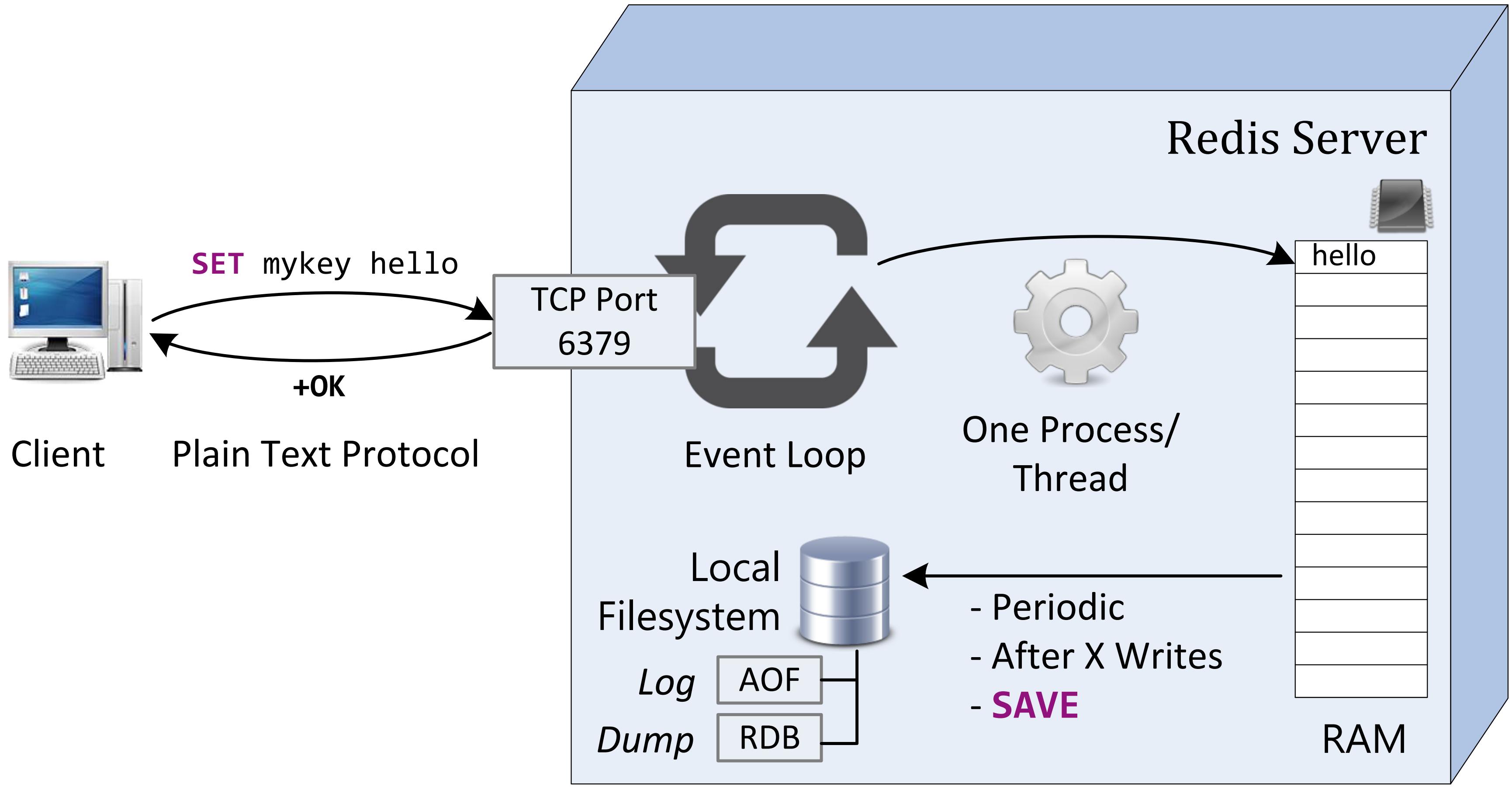
Redis (CA)

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

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

Redis Architecture

- Redis Codebase ≅ 20K LOC

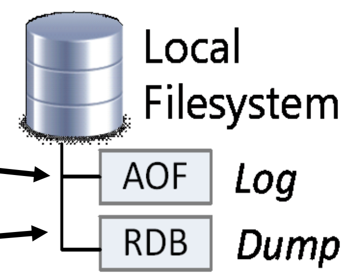


Persistence

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

```
config set appendonly everysec
```

`fsync()` every second



Snapshot every 60s,
if > 1000 keys changed

```
config set save 60 1000
```

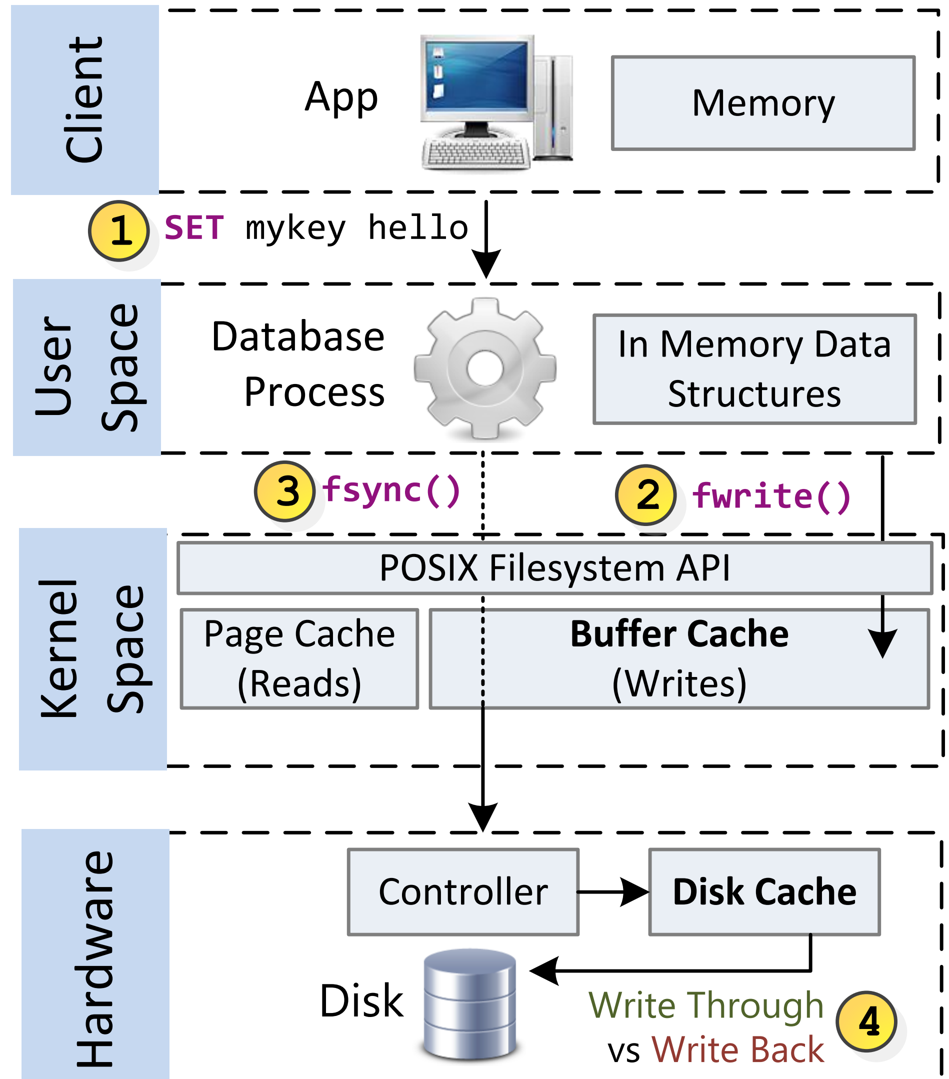
Persistence

1. Resistence to client crashes

2. Resistence to DB process crashes

3. Resistence to hardware crashes with *Write-Through*

4. Resistence to hardware crashes with *Write-Back*



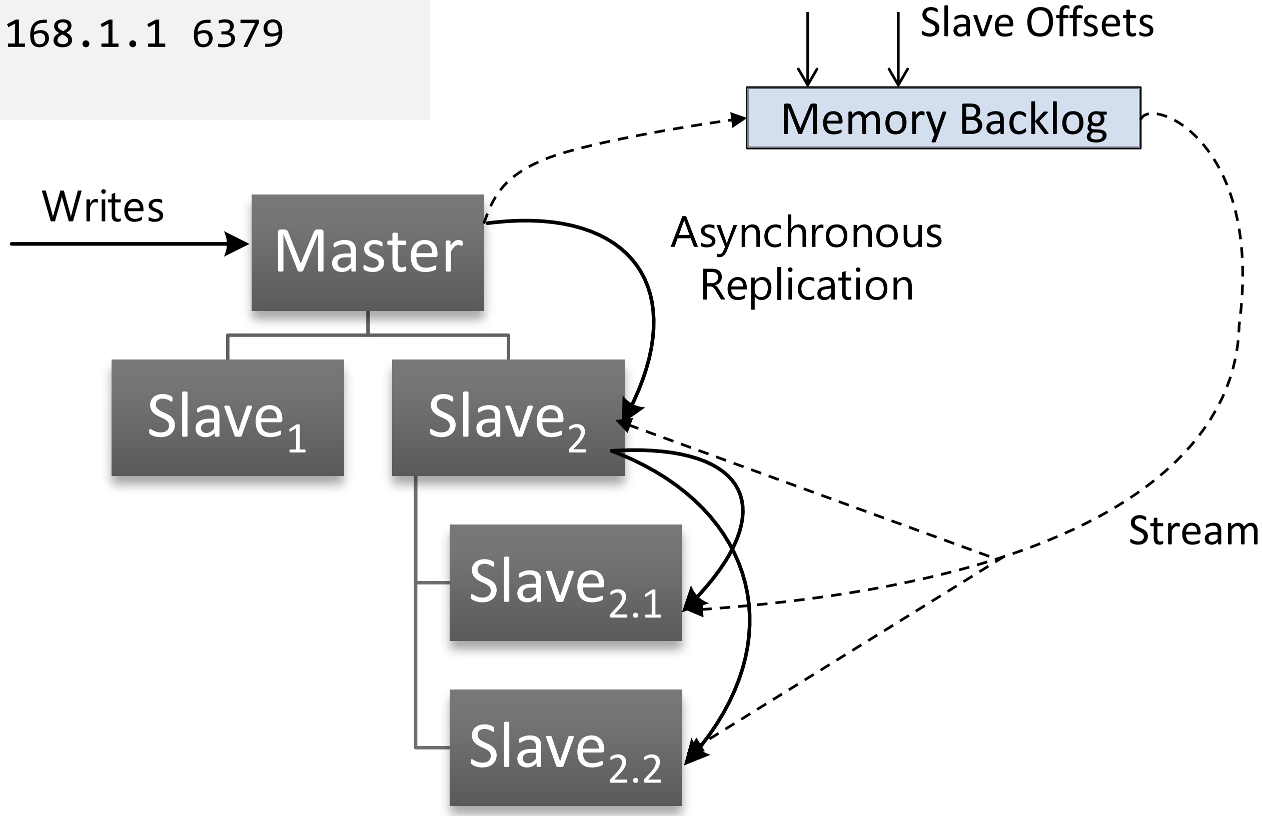
Persistence: Redis vs an RDBMS

- PostgreSQL:
- Redis:

PostgreSQL	Redis
> `synchronous_commit on`	> `appendfsync always`
Latency > Disk Latency, Group Commits, Slow	
> `synchronous_commit off`	> `appendfsync everysec`
periodic `fsync()`, data loss limited	
> `fsync false`	> `appendfsync no`
Data corruption and losspossible	Data loss possible, corruption prevented
> `pg_dump`	> `save` oder `bgsave`

Master-Slave Replication

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



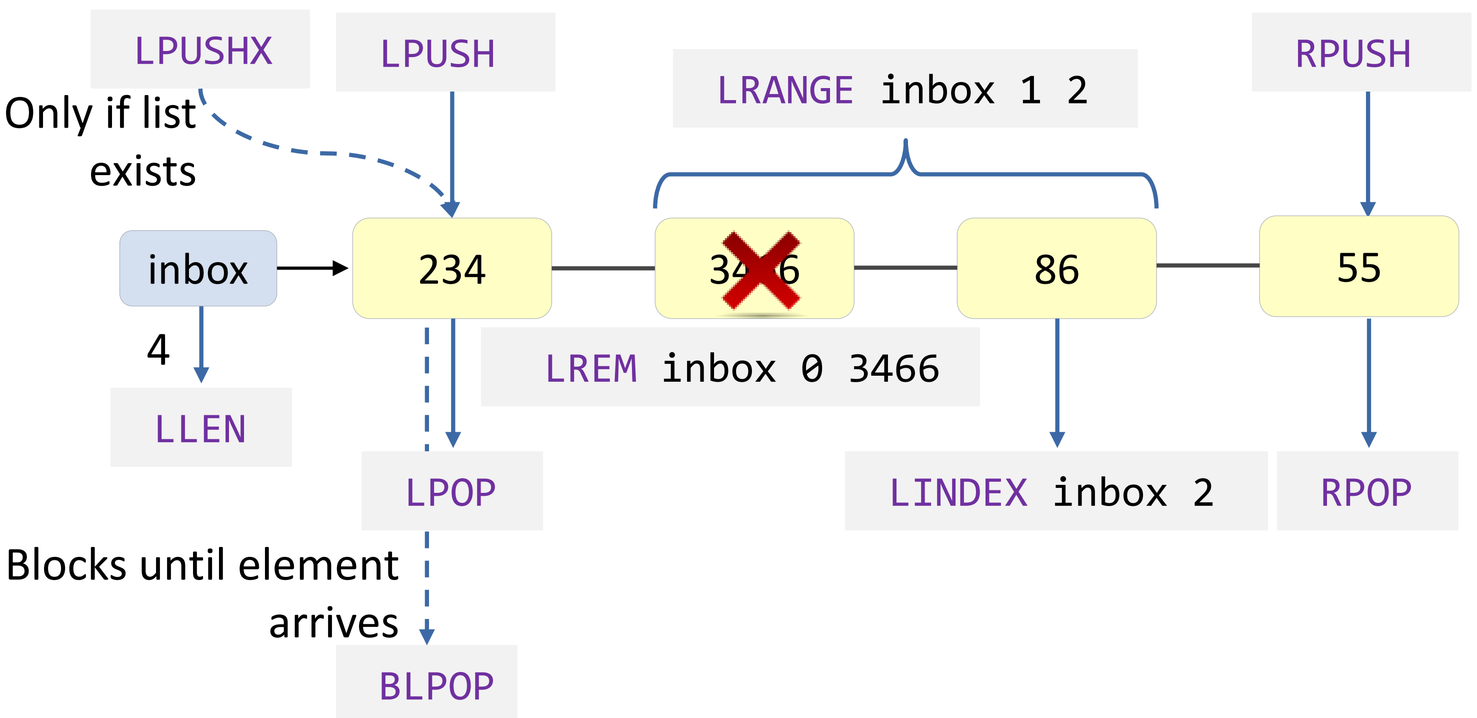
Data structures

- String, List, Set, Hash, Sorted Set

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

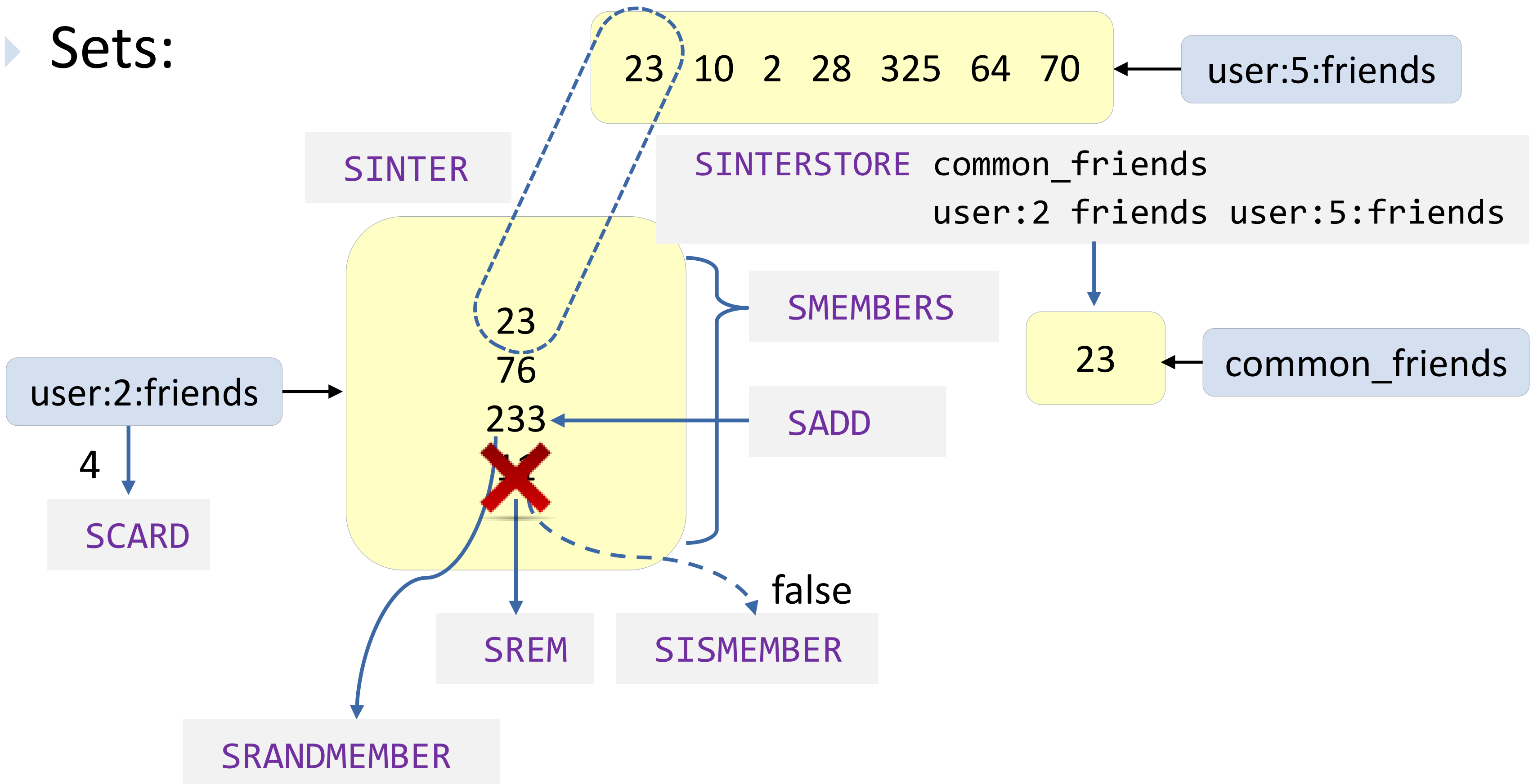
Data Structures

- (Linked) Lists:



Data Structures

- Sets:



Data Structures

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

redis

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

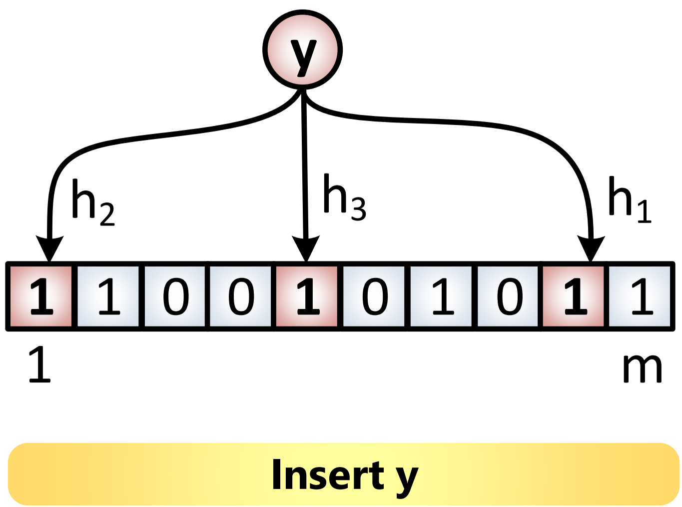
```
SUBSCRIBE user:2:notifs
```

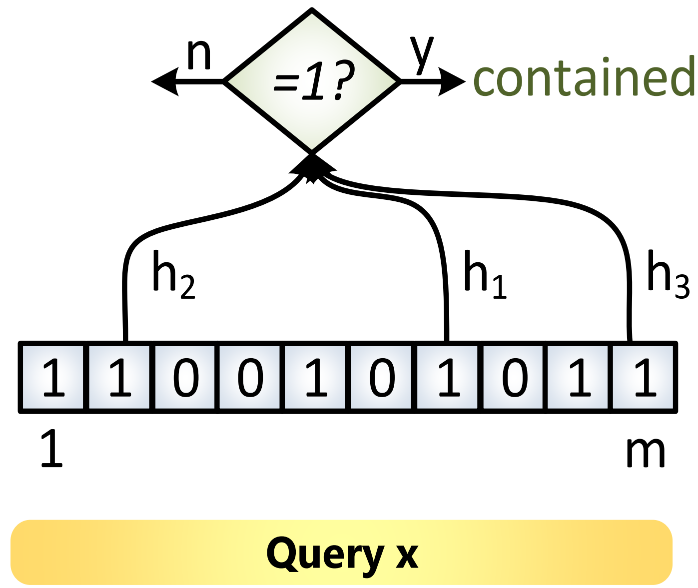
```
{
   event: 'comment posted',
   time : …
}
```

Example: Bloom filters

Compact Probabilistic Sets

- Bit array of length **m** and **k** independent hash functions
- **insert(obj)**: add to set
- **contains(obj)**: might give a false positive





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

Bloomfilters in Redis

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

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

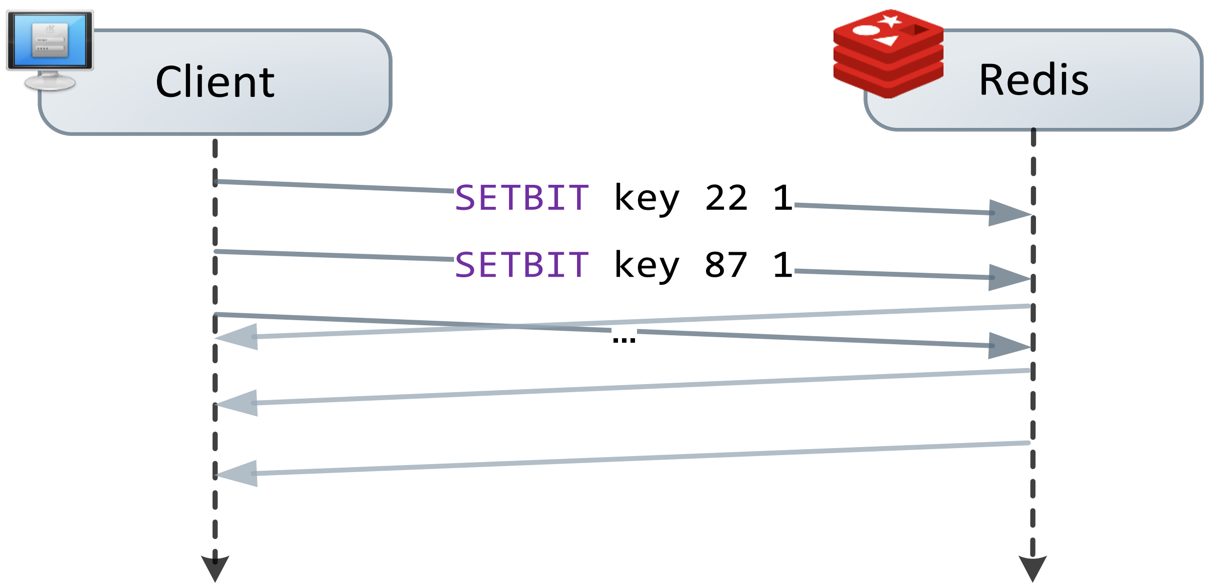
Jedis: Redis Client for Java

SETBIT creates and resizes automatically

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

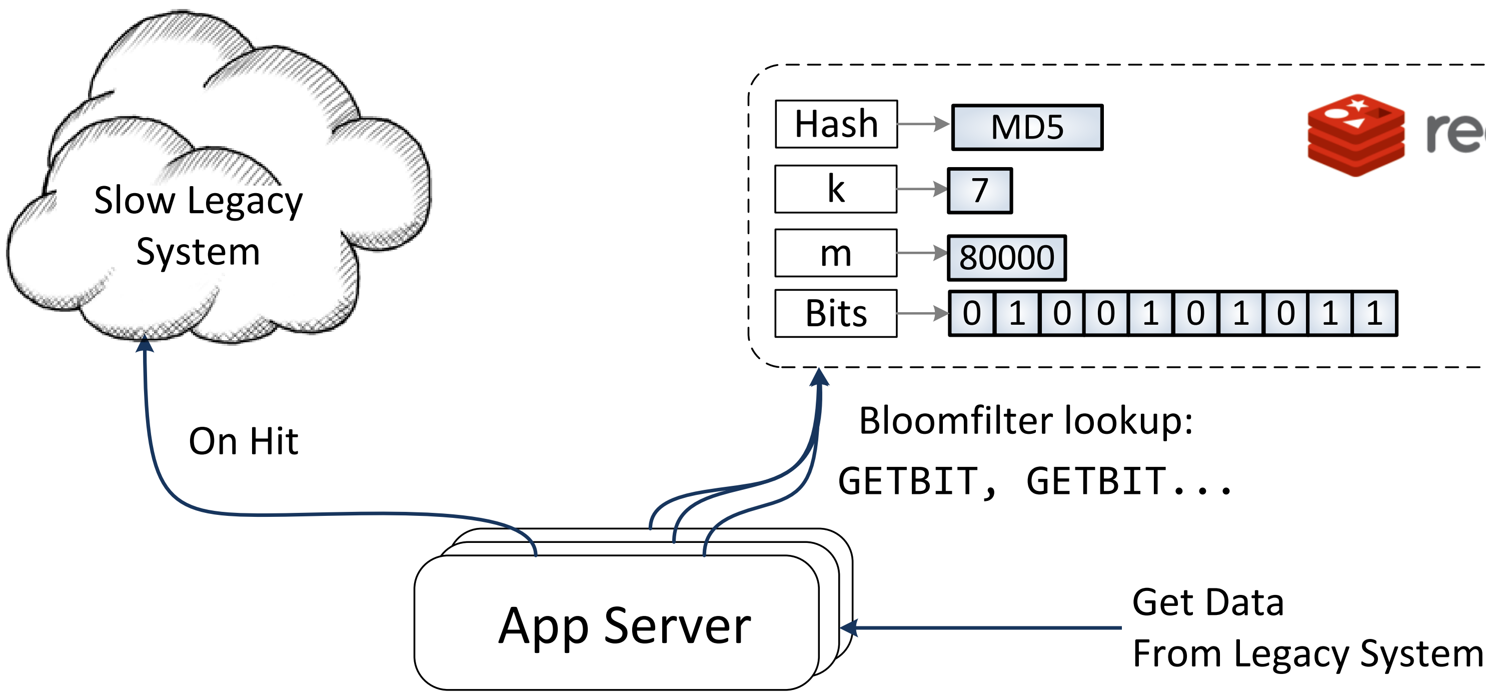
Pipelining

- If the Bloom filter uses 7 hashes: 7 roundtrips
- **Solution**: Redis Pipelining



Redis for distributed systems

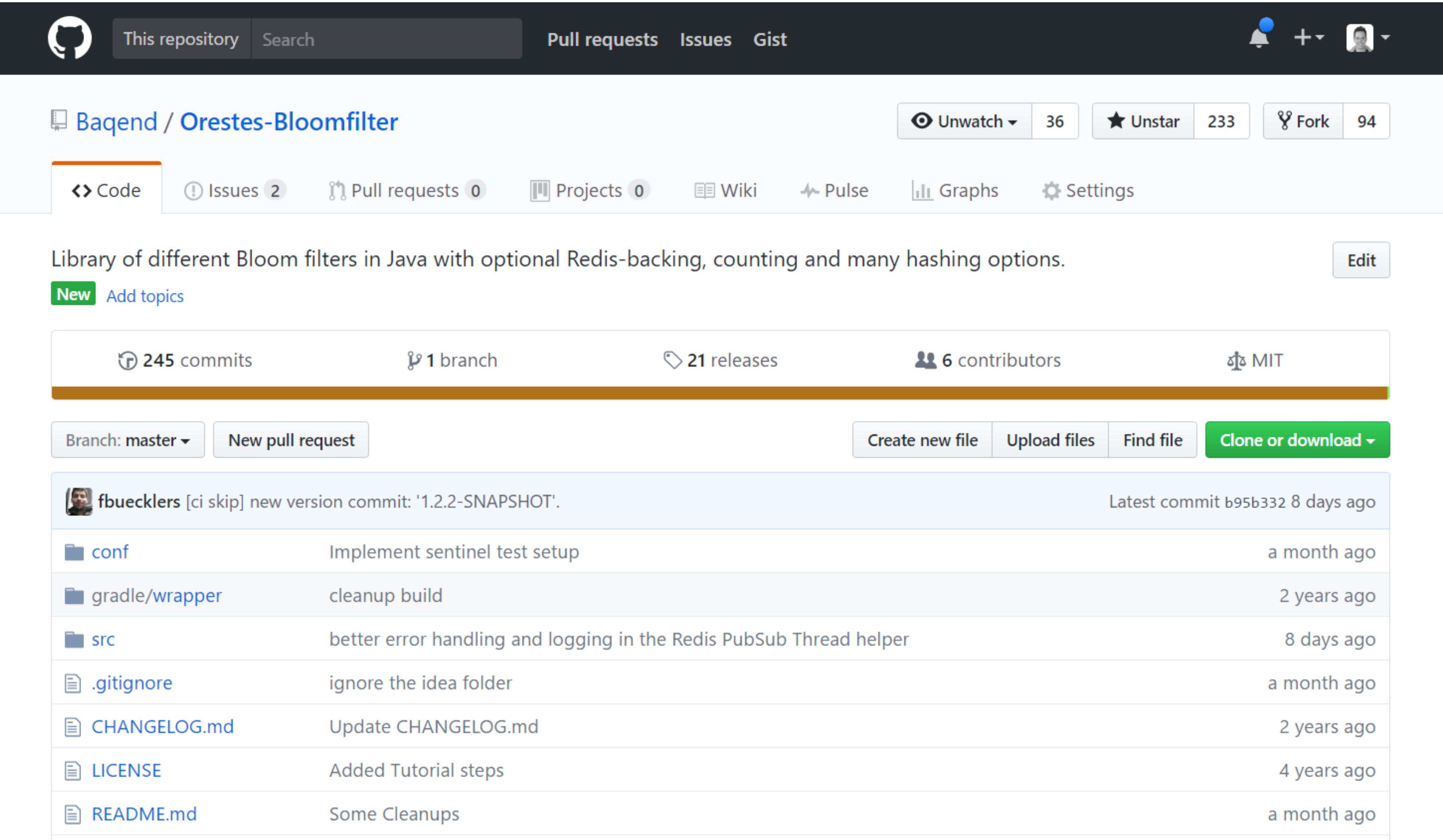
- Common Pattern: distributed system with **shared state** in Redis
- Example - Improve performance for legacy systems:



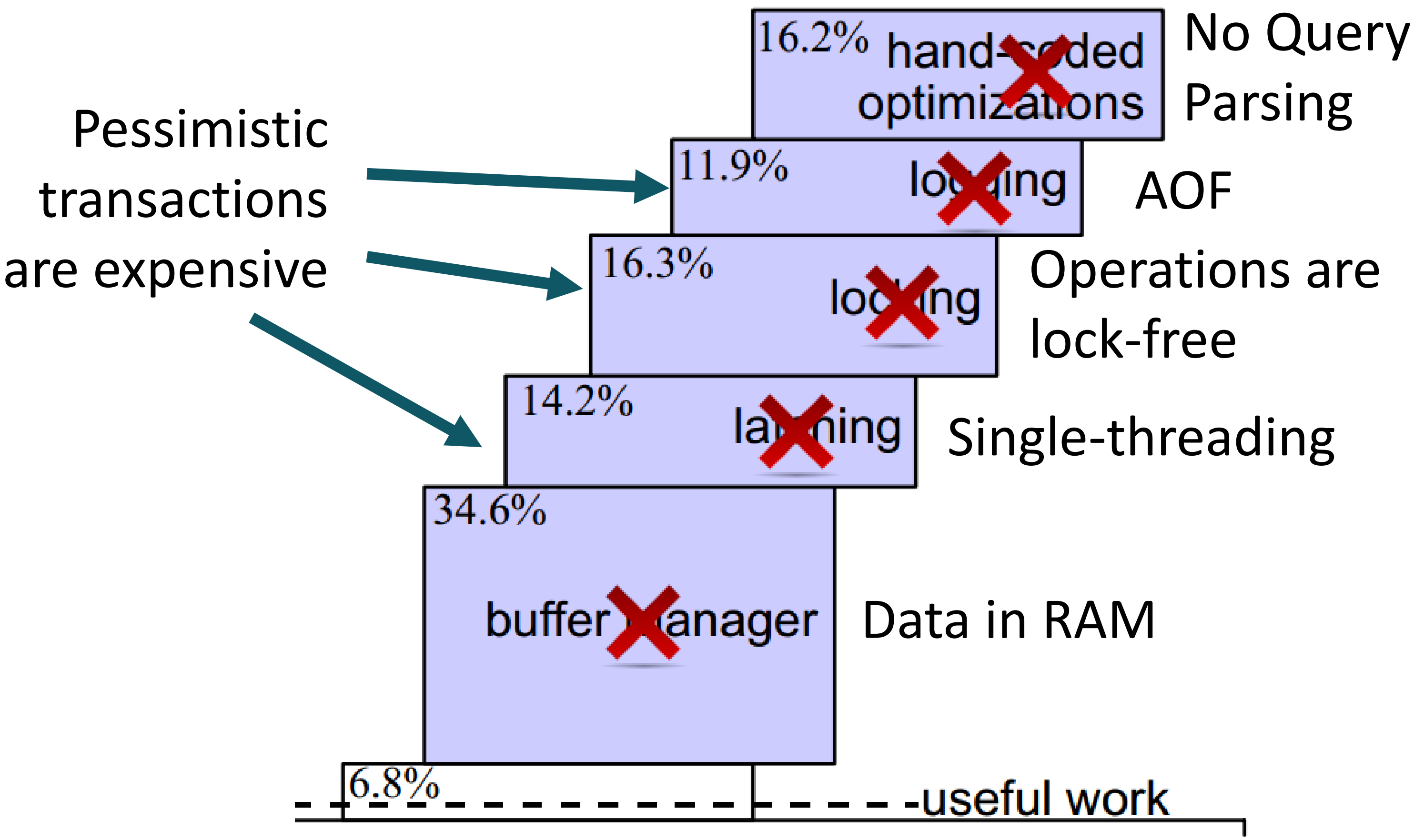
Redis Bloom filters

Open Source

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



Why is Redis so fast?



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

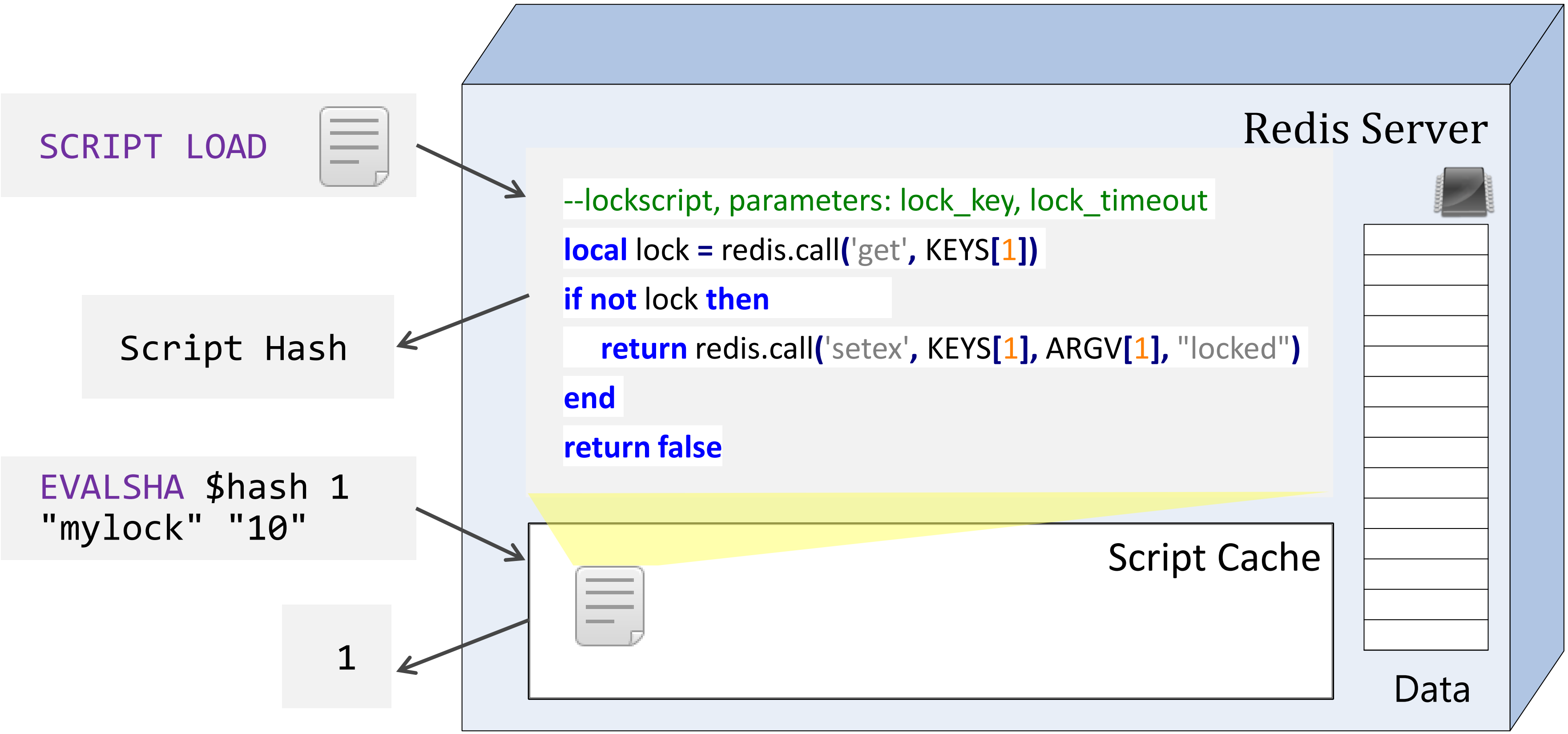
Optimistic Transactions

- MULTI: Atomic Batch Execution
- WATCH: Condition for MULTI Block

Only executed if bother keys are unchanged

```
WATCH users:2:followers, users:3:followers

MULTI

    SMEMBERS users:2:followers  →  Queued

    SMEMBERS users:3:followers  →  Queued

    INCR transactions  →  Queued

EXEC  →  Bulk reply with 3 results
```

Lua Scripting



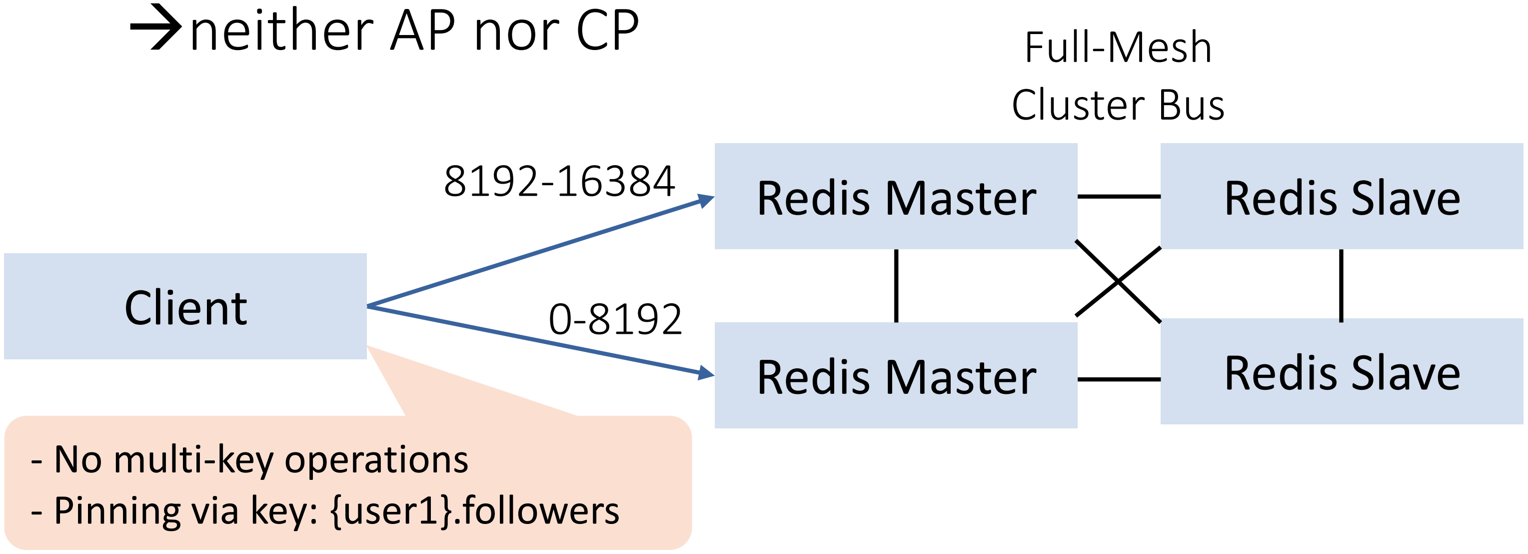
Ierusalimschy, Roberto. Programming in lua. 2006.

Redis Cluster

Work-in-Progress

- **Idea**: Client-driven hash-based sharing (CRC32, „hash slots“)
- **Asynchronous** replication with failover (variant of Raft‘s leader election)
  - **Consistency**: not guaranteed, last failover wins
  - **Availability**: only on the majority partition

    →neither AP nor CP

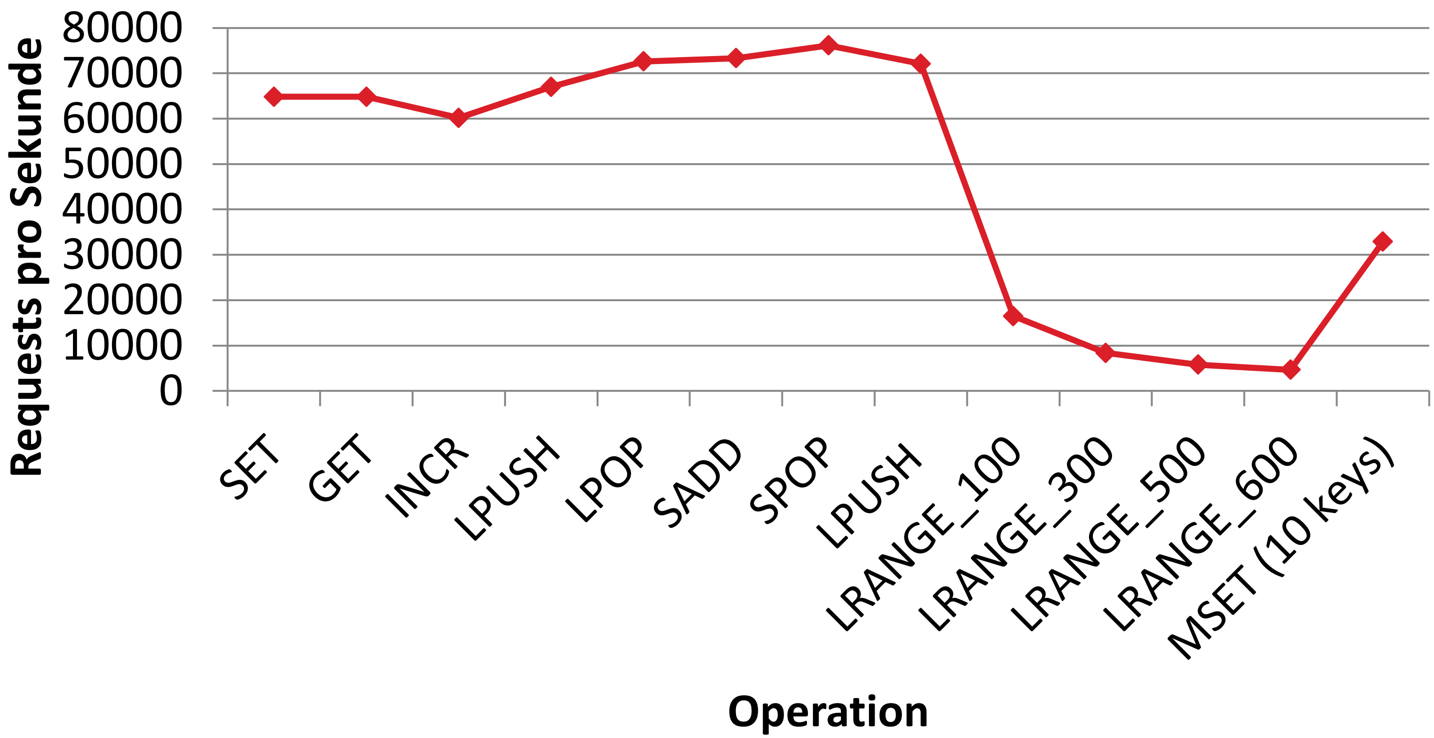


http://redis.io/topics/cluster-spec

Performance

- Comparable to Memcache

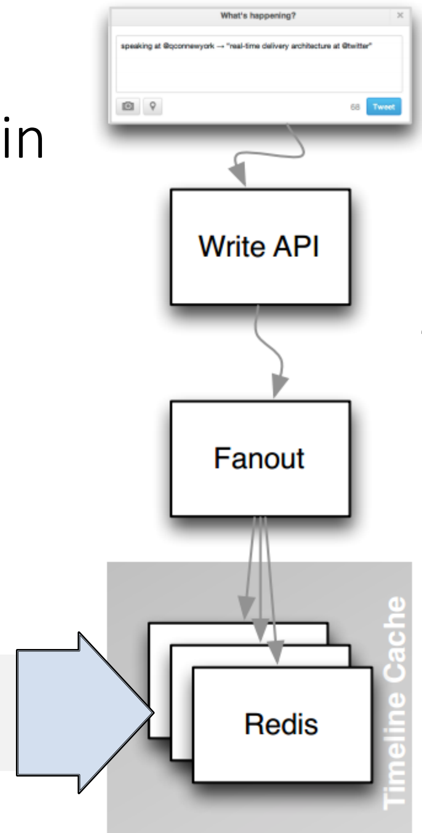
```
> redis-benchmark -n 100000 -c 50
```



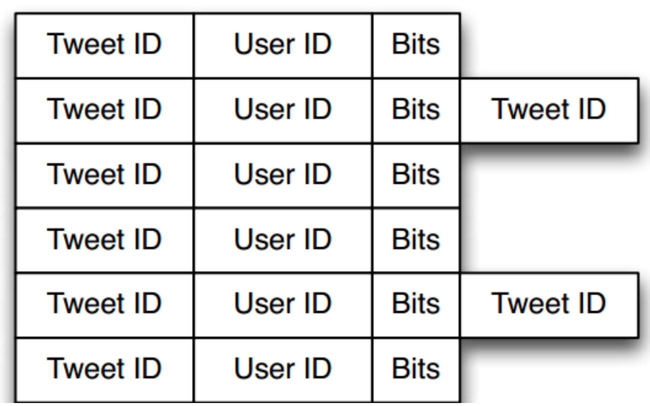
Example Redis Use-Case: Twitter

- 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

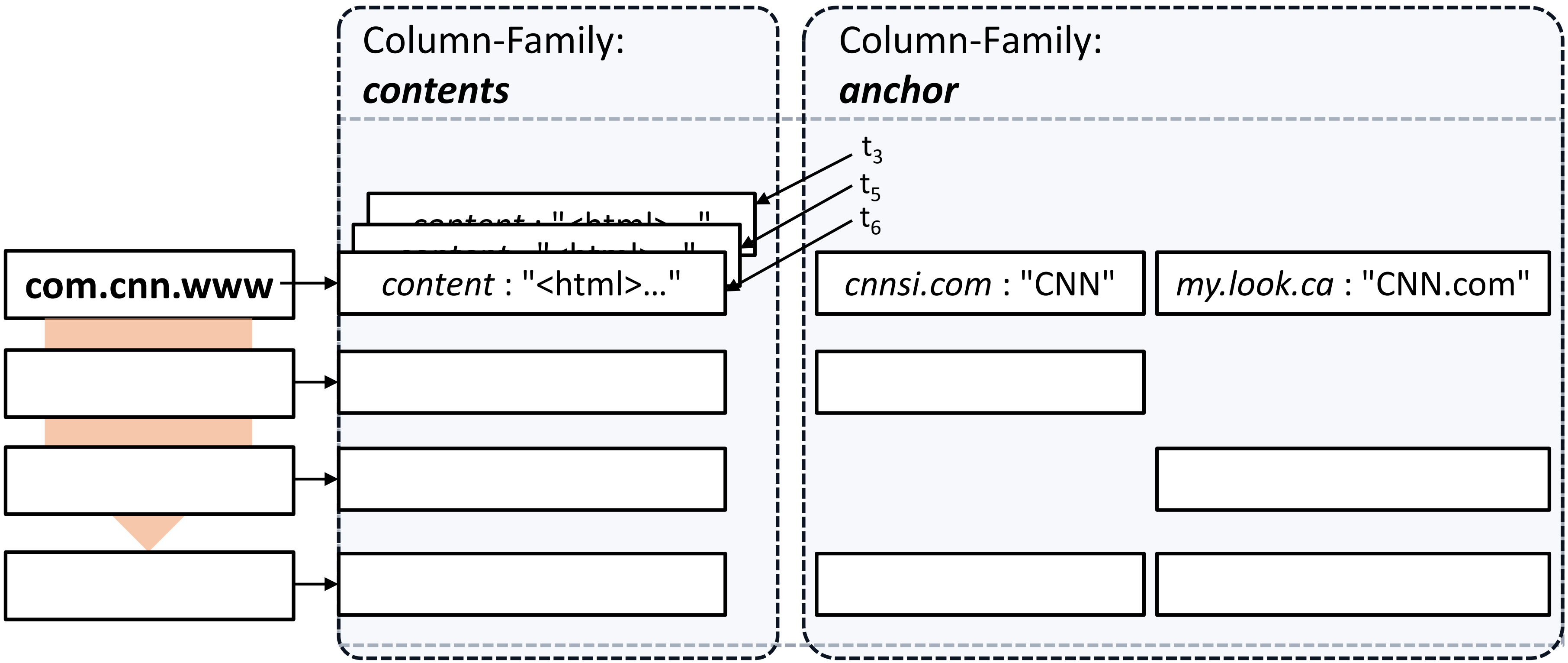
> A Bigtable is a sparse, distributed, persistent multidimensional sorted map.

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

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

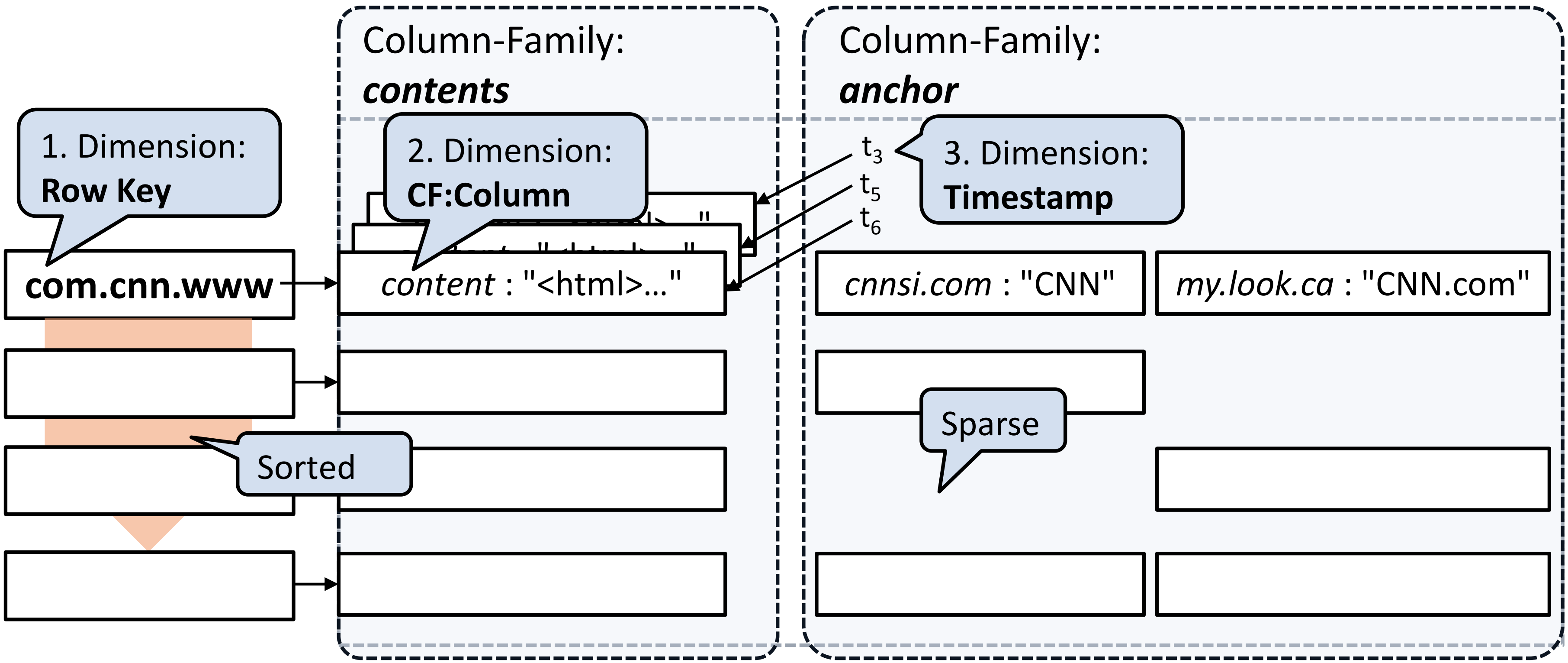
Wide-Column Data Modelling

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



Wide-Column Data Modelling

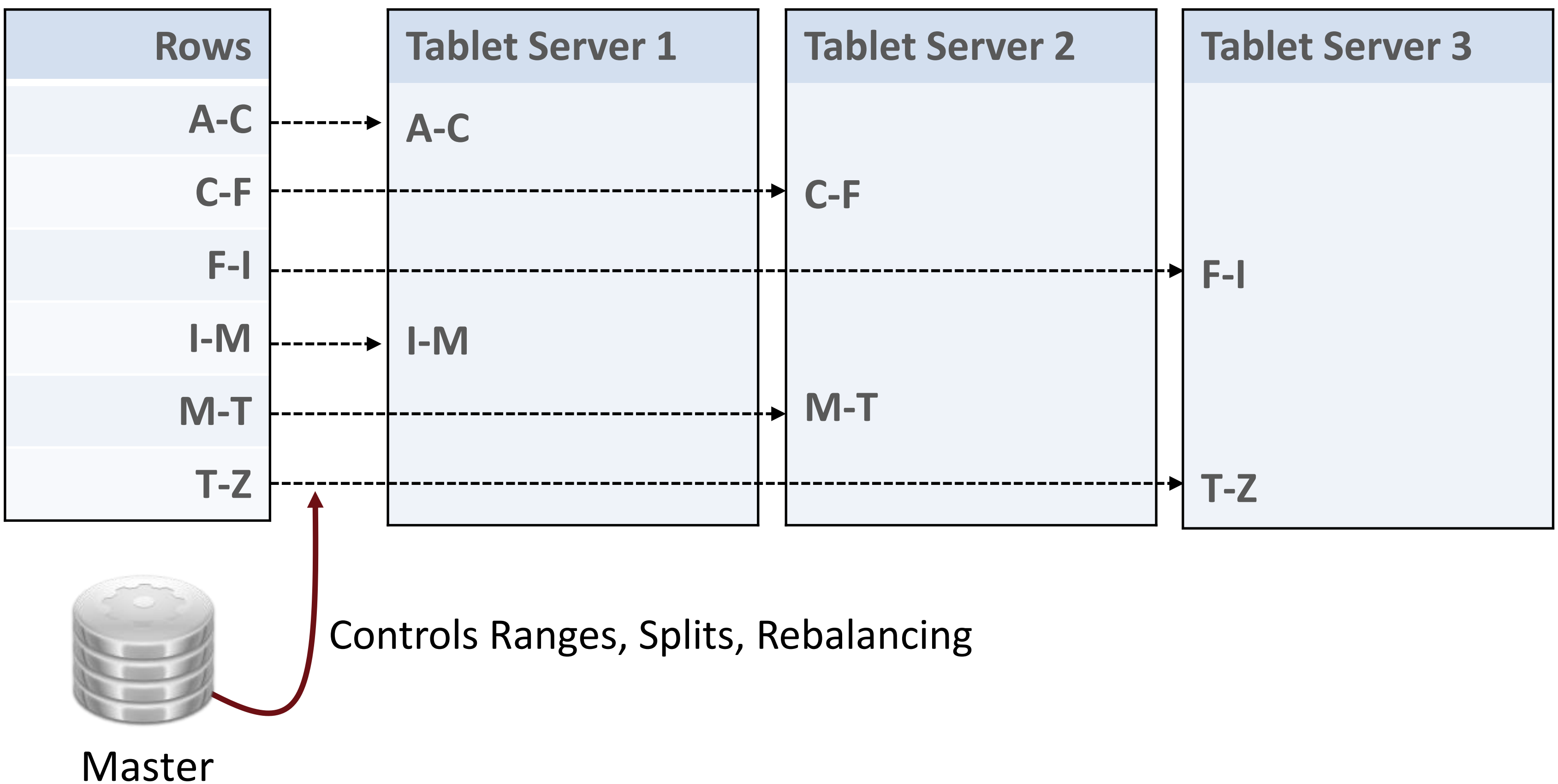
- Storage of crawled web-sites („Webtable“):



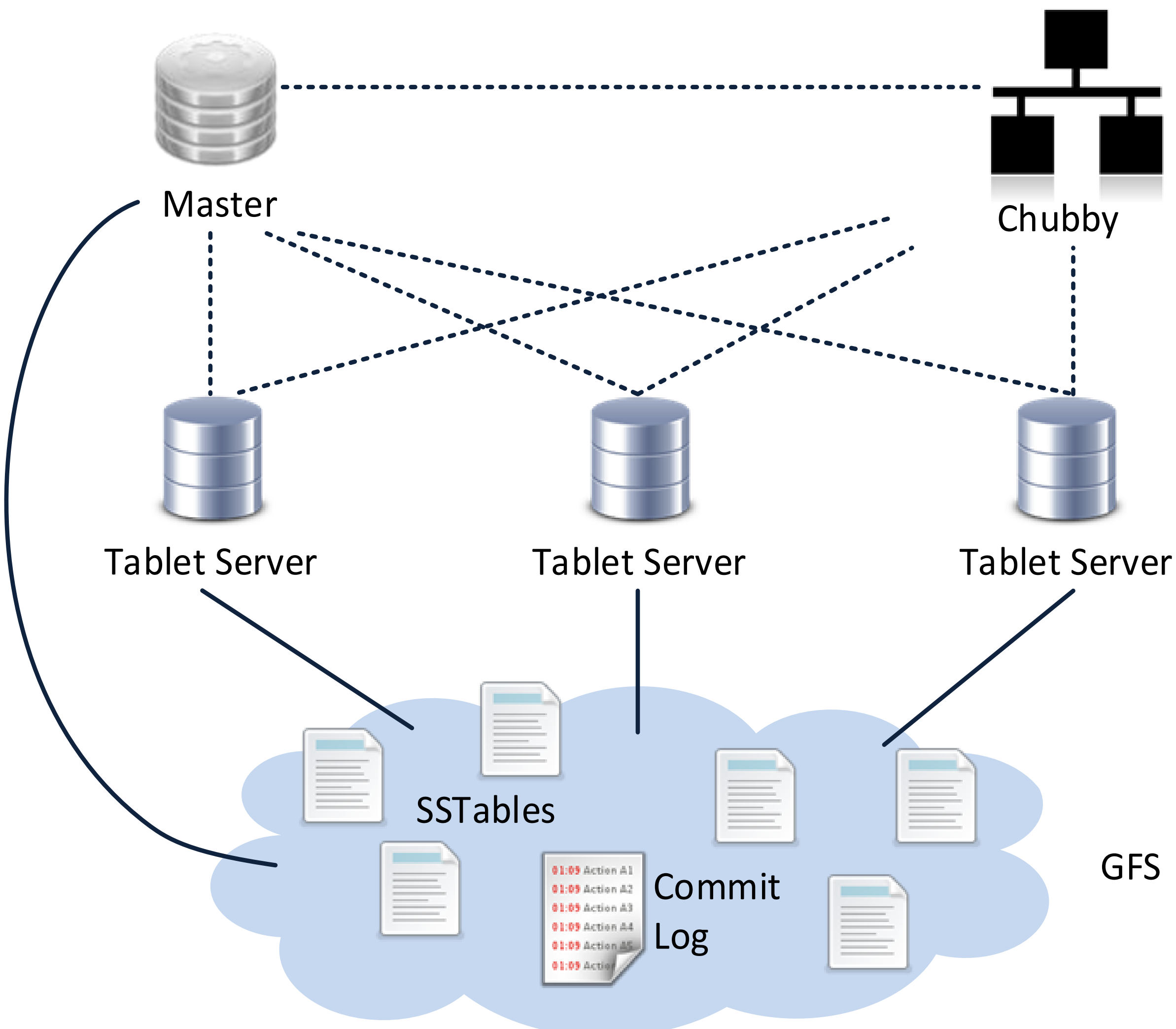
Range-based Sharding

BigTable Tablets

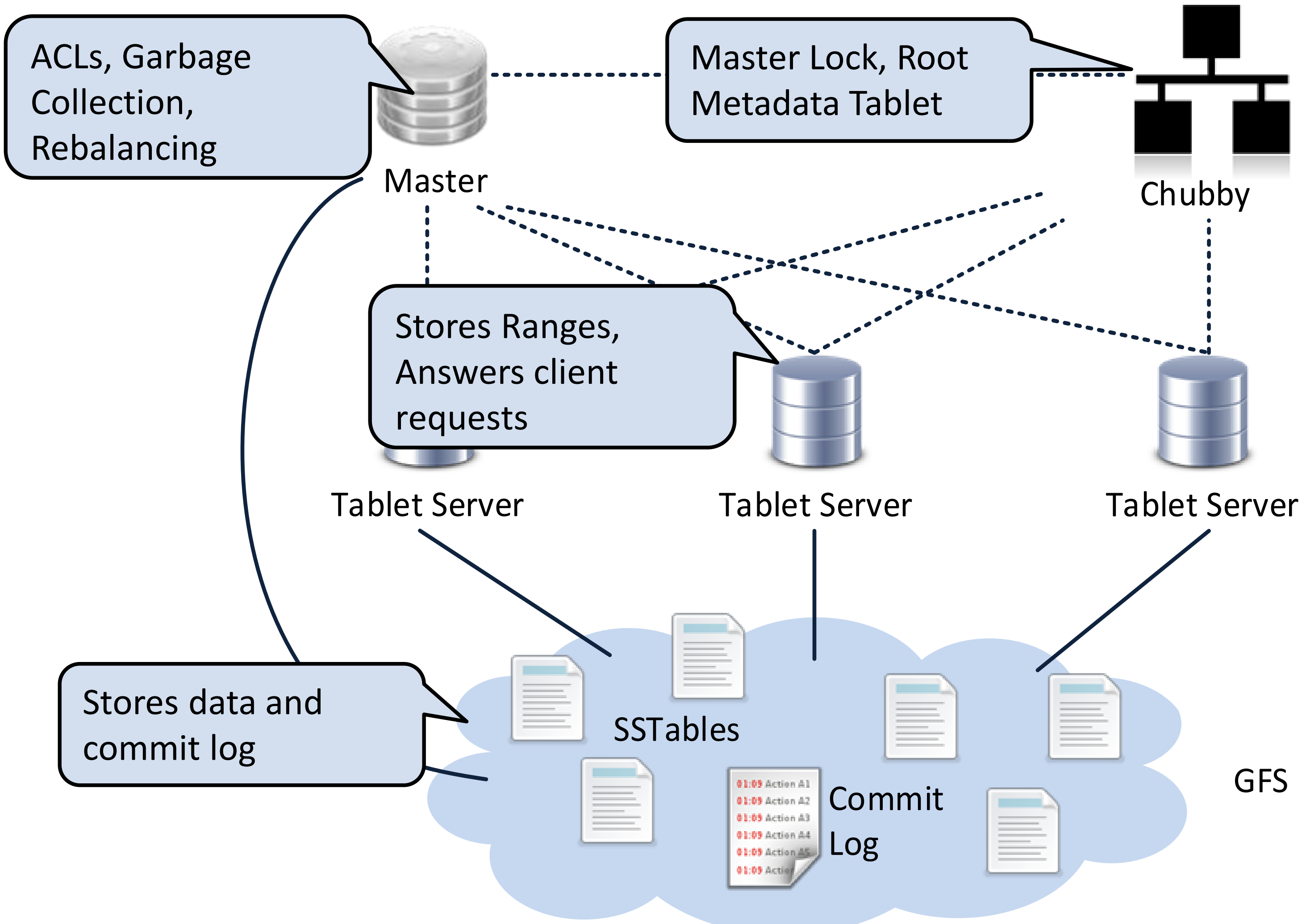
Tablet: Range partition of ordered records



Architecture

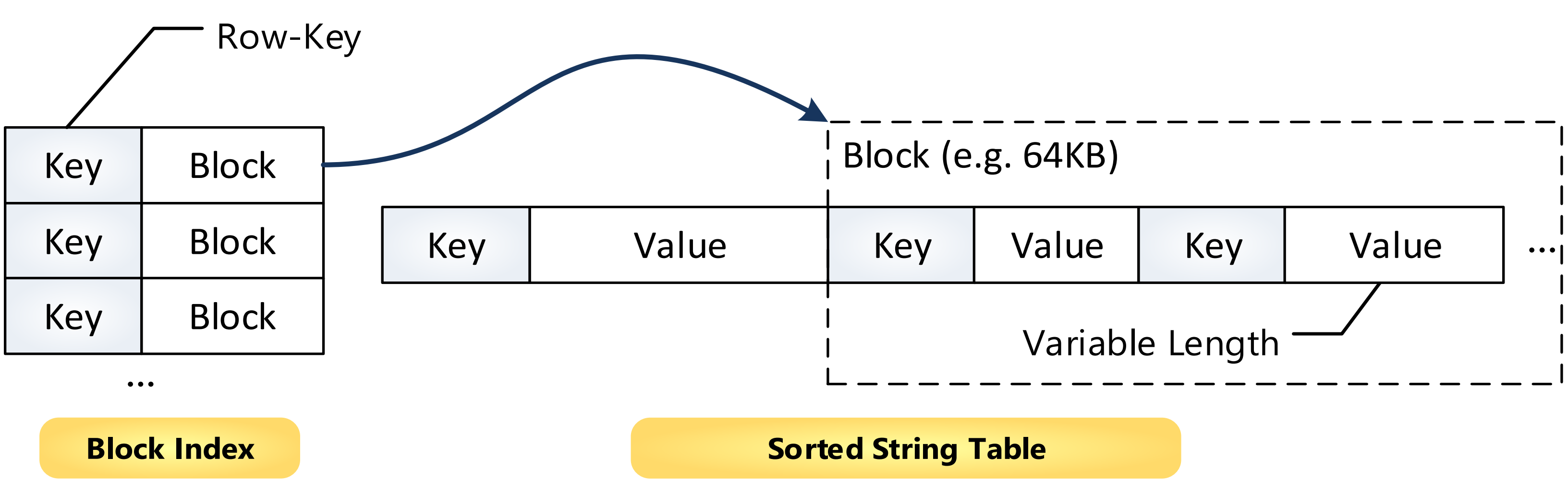


Architecture



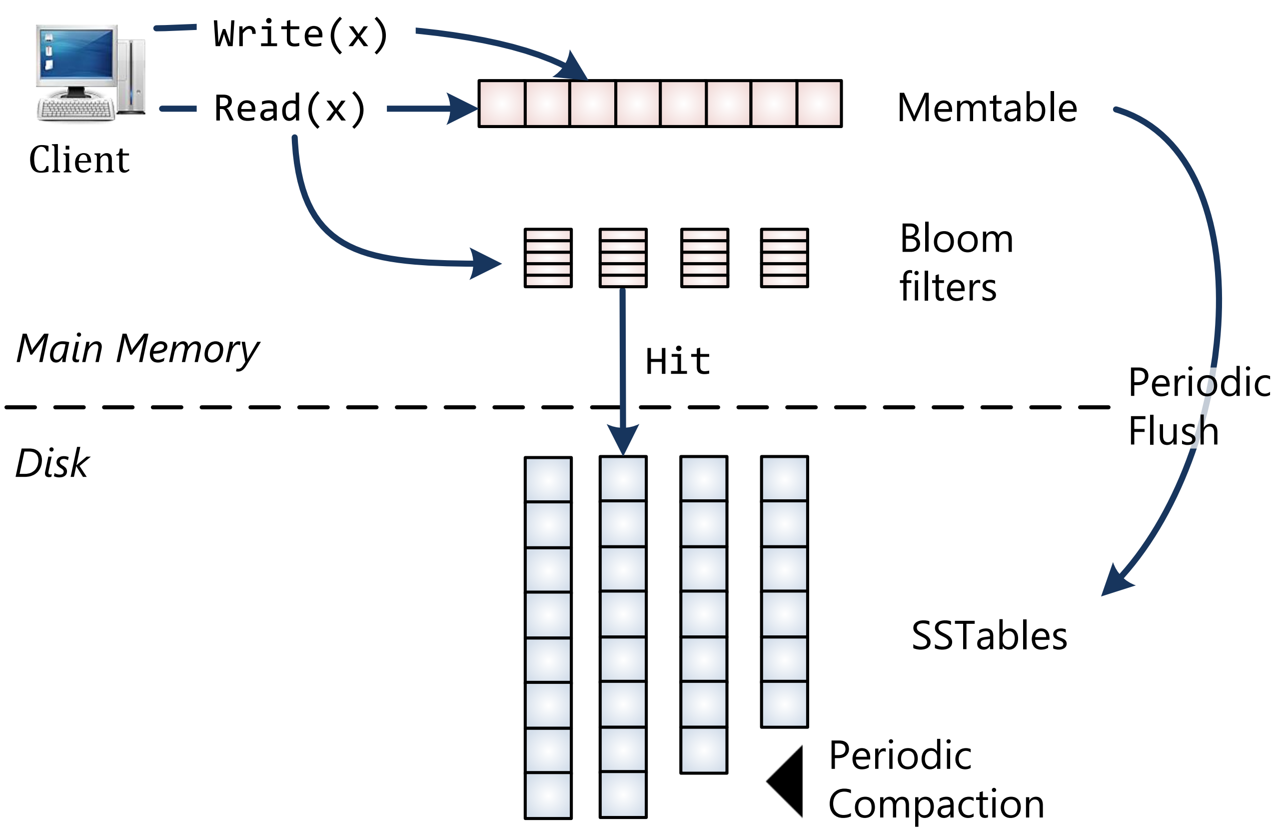
Storage: Sorted-String Tables

- **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: In-Memory in **Memtable**
- SSTable disk access optimized by Bloom filters



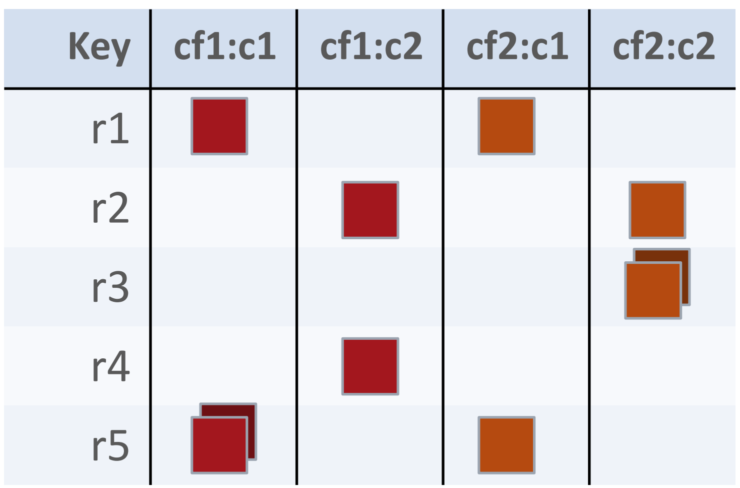
Apache HBase (CP)

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

HBase Storage

- Logical to physical mapping:



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

HBase Storage

- Logical to physical mapping:

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>
```

HFile cf2

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

HFile cf1

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

HBase Storage

- Logical to physical mapping:

In Value
In Key
In Column

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

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>
```

HFile cf2

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

HFile cf1

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

Example: Facebook Insights

MD5(Reversed Domain) + Reversed Domain + URL-ID

Row Key

6PM Total	6PM Male	…	01.01 Total	01.01 Male	…	Total	Male	…
10	7		100	65			567	

Atomic HBase **Counter**

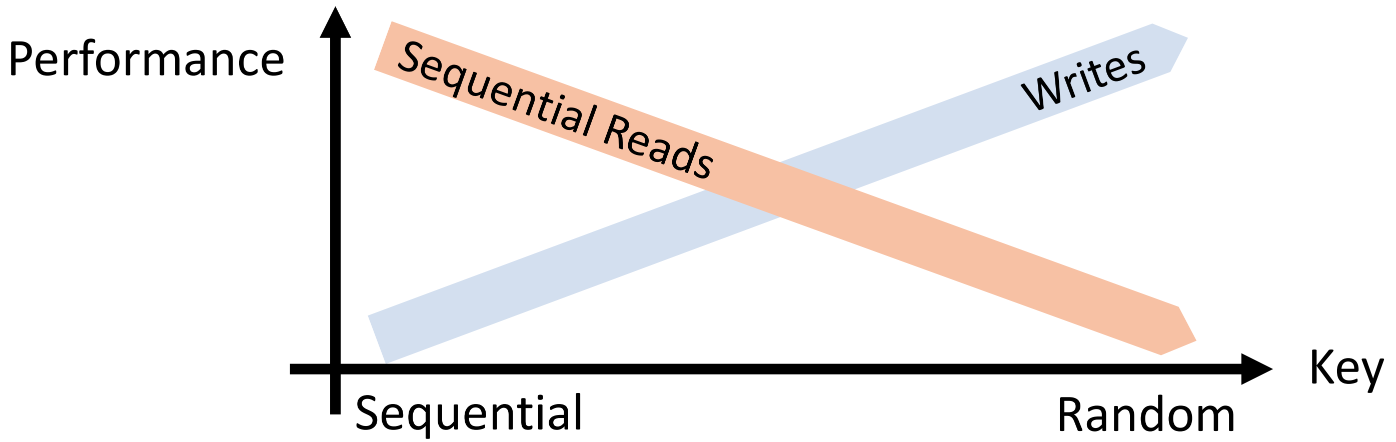
CF:Daily CF:Monthly CF:All

TTL – automatic deletion of old rows

Lars George: "Advanced HBase Schema Design"

Schema Design

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



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

Schema: Messages

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

vs

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

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

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

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

API: CRUD + Scan

Setup Cloud Cluster:

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

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

Login, cluster size, etc.

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

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

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

API: Features

- Row **Locks** (MVCC): `table.lockRow()`, `unlockRow()`
  - Problem: Timeouts, Deadlocks, Ressources
- **Conditional Updates**: `checkAndPut()`, `checkAndDelete()`
- CoProcessors - registriered Java-Classes for:
  - Observers (`prePut`, `postGet`, etc.)
  - Endpoints (Stored Procedures)
- HBase can be a Hadoop **Source**:

```
TableMapReduceUtil.initTableMapperJob(
  tableName,  //Table
  scan,  //Data input as a Scan
  MyMapper.class, ...  //usually a TableMapper<Text,Text> );
```

Summary: BigTable, HBase

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

Classification: HBase

Techniques

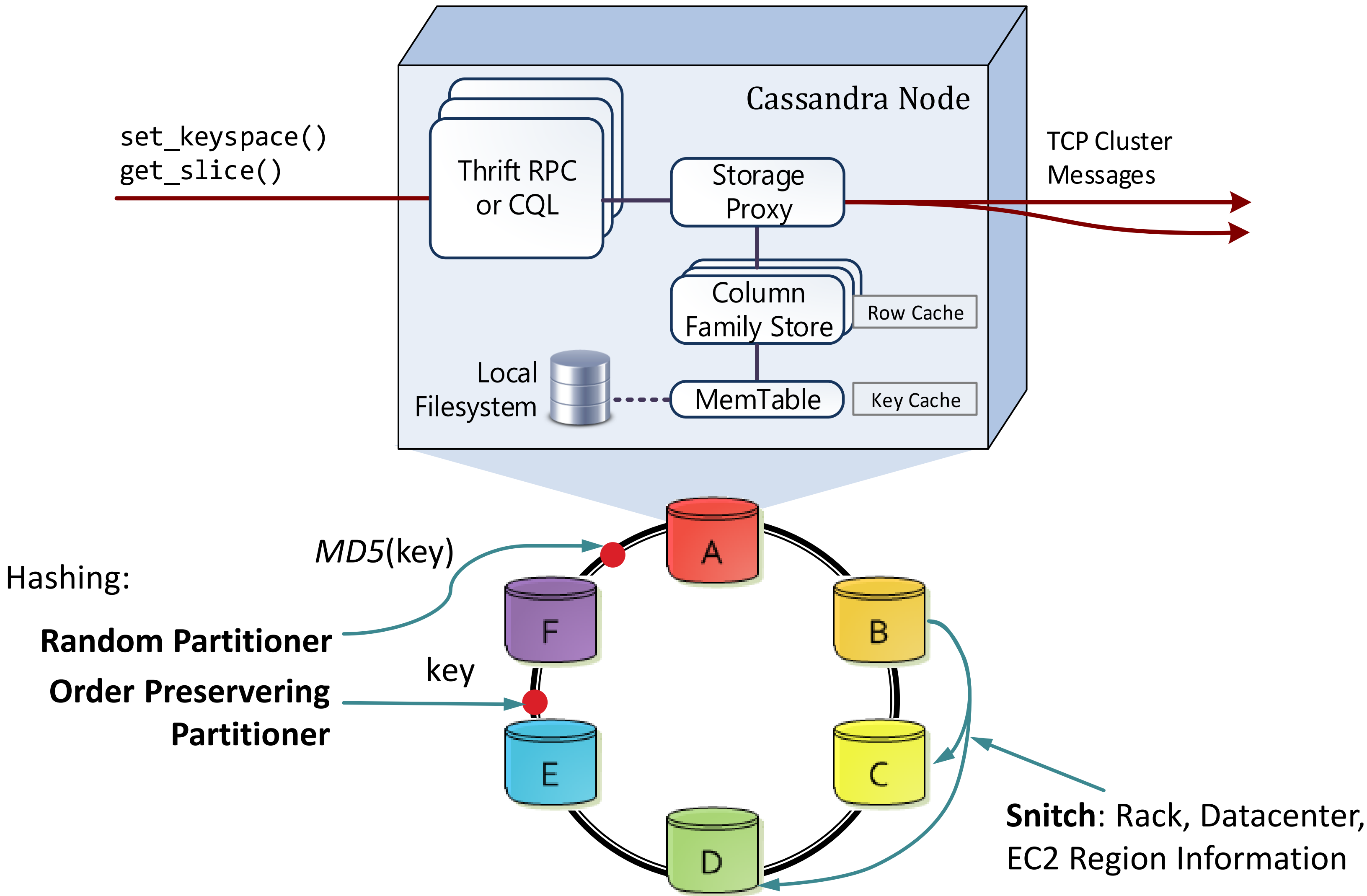
Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index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

Apache Cassandra (AP)

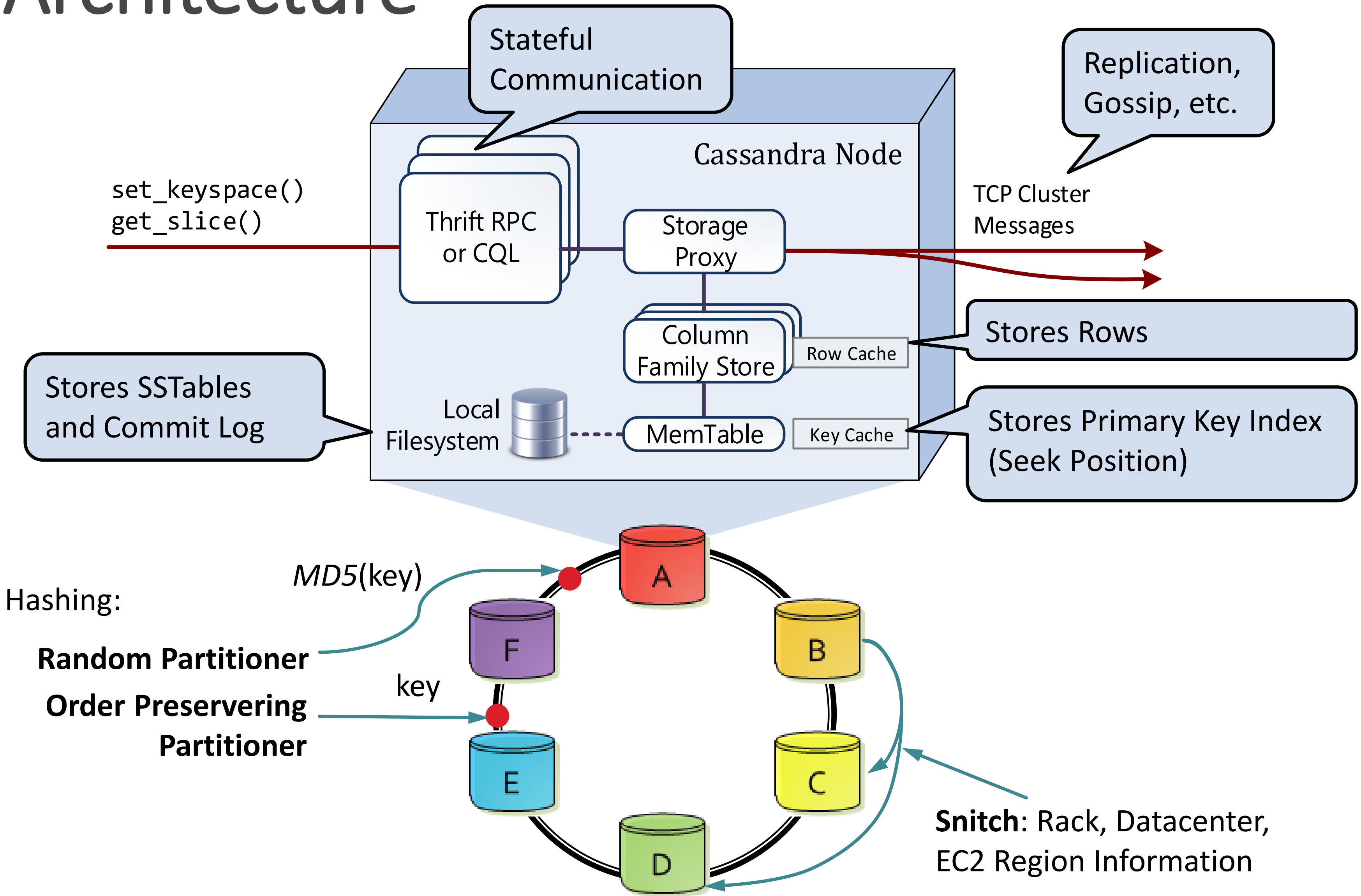
Cassandra
Model:
Wide-Column
License:
Apache 2
Written in:
Java

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

Architecture



Architecture



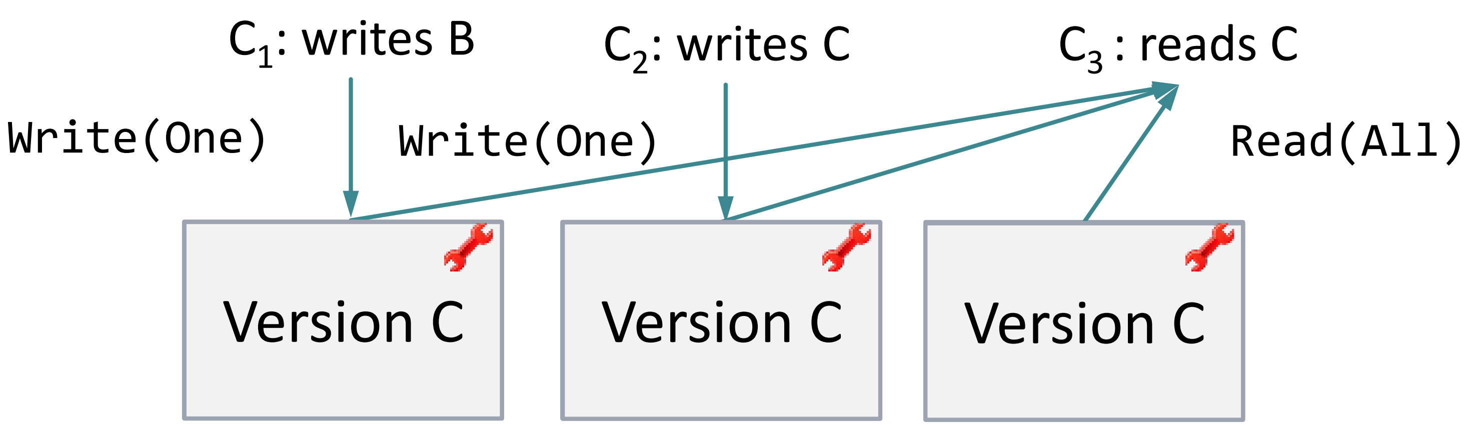
Consistency

- No Vector Clocks but **Last-Write-Wins**
  - ➔ Clock synchronisation required
- No Versionierung that keeps old cells

Write	**Read**
Any	-
One	One
Two	Two
Quorum	Quorum
Local_Quorum / Each_Quorum	Local_Quorum / Each_Quorum
All	All

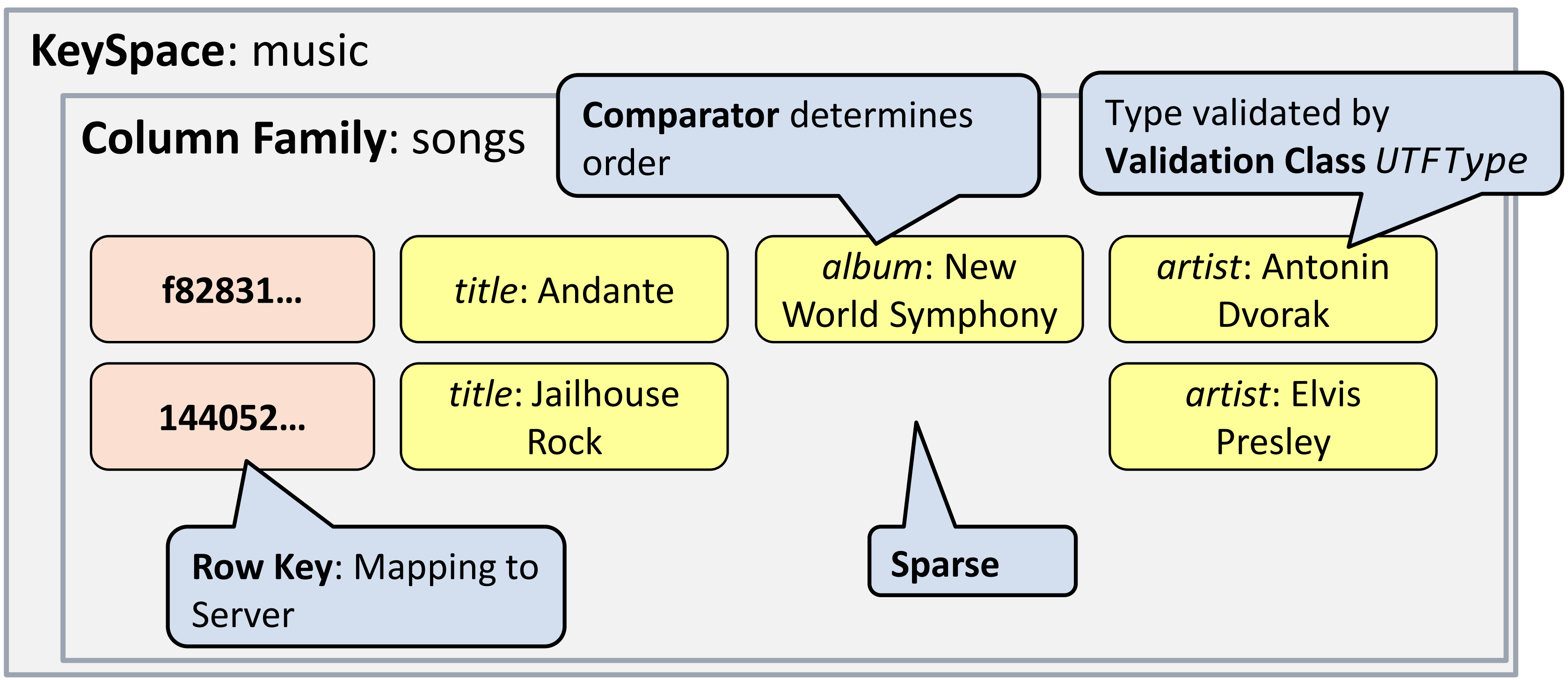
Consistency

- Coordinator chooses newest version and triggers *Read Repair*
- **Downside**: upon conflicts, changes are lost



Storage Layer

- Uses BigTables Column Family Format



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

CQL Example: Compound keys

- Enables Scans despite *Random Partitioner*

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

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

Partition Key

Clustering Columns:
sorted per node

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

Other Features

- **Distributed Counters** – prevent update anomalies
- **Full-text Search** (Solr) in Commercial Version
- **Column TTL** – automatic garbage collection
- **Secondary indices**: hidden table with mapping
  → queries with simple equality condition
- **Lightweight Transactions**: linearizable updates through a Paxos-like protocol

```
INSERT INTO USERS (login, email, name, login_count)
values ('jbellis', 'jbellis@datastax.com', 'Jonathan Ellis', 1)
IF NOT EXISTS
```

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
- 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), …

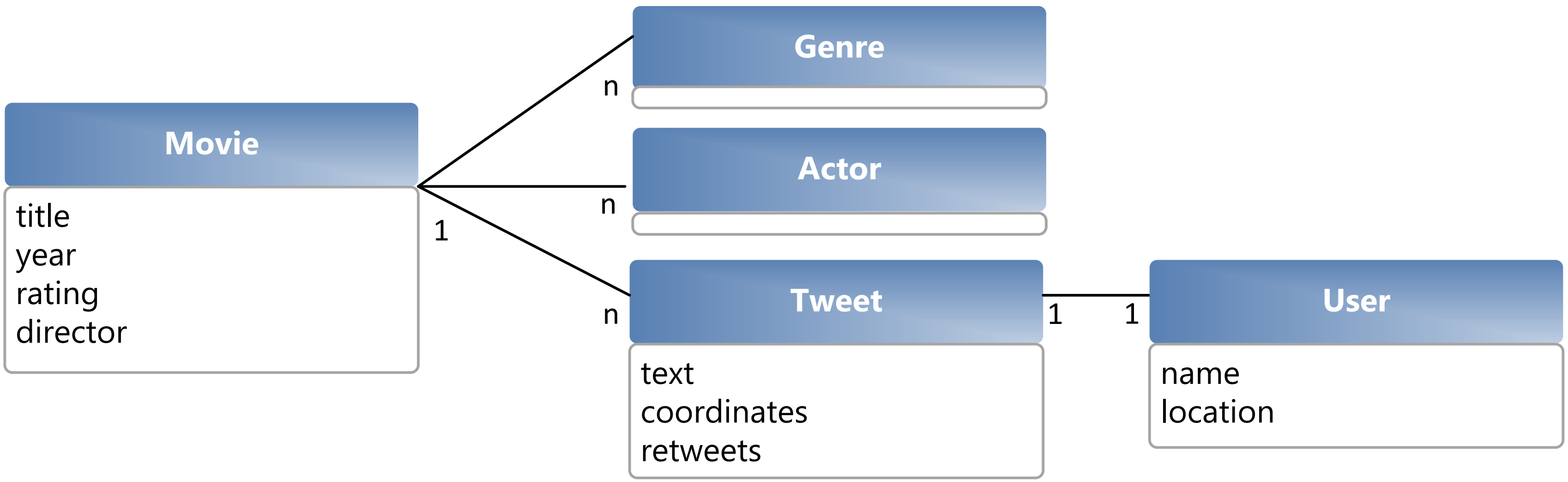
Basics

```
> mongod &
> mongo imdb
MongoDB shell version: 2.4.3
connecting to: imdb
> show collections
movies
tweets
> db.movies.findOne({title : "Iron Man 3"})
{
        title : "Iron Man 3",
        year : 2013 ,
        genre : [
                "Action",
                "Adventure",
                "Sci -Fi"],
        actors : [
                "Downey Jr., Robert",
                "Paltrow , Gwyneth",]
}
```

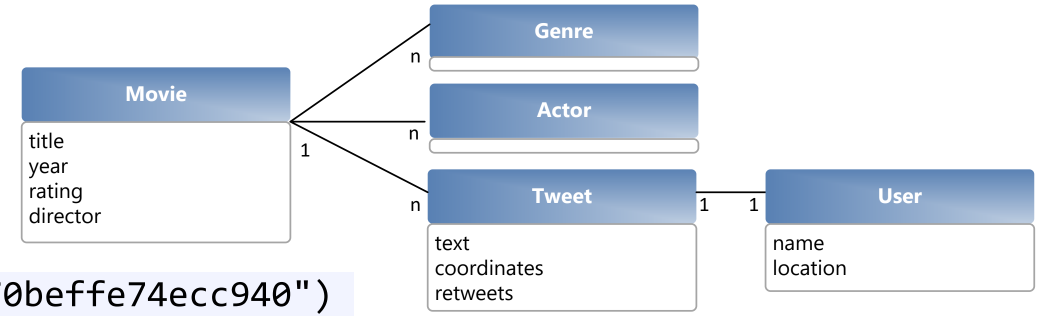
Properties

Arrays, Nesting allowed

Data Modelling



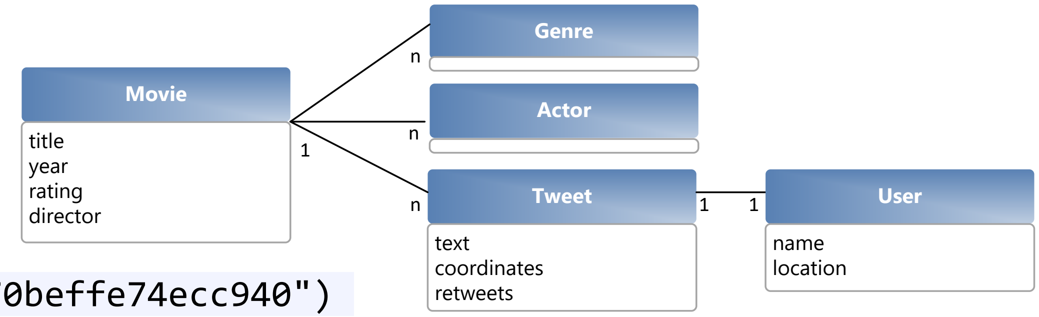
Data Modelling




```
{
        "_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

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 of joins

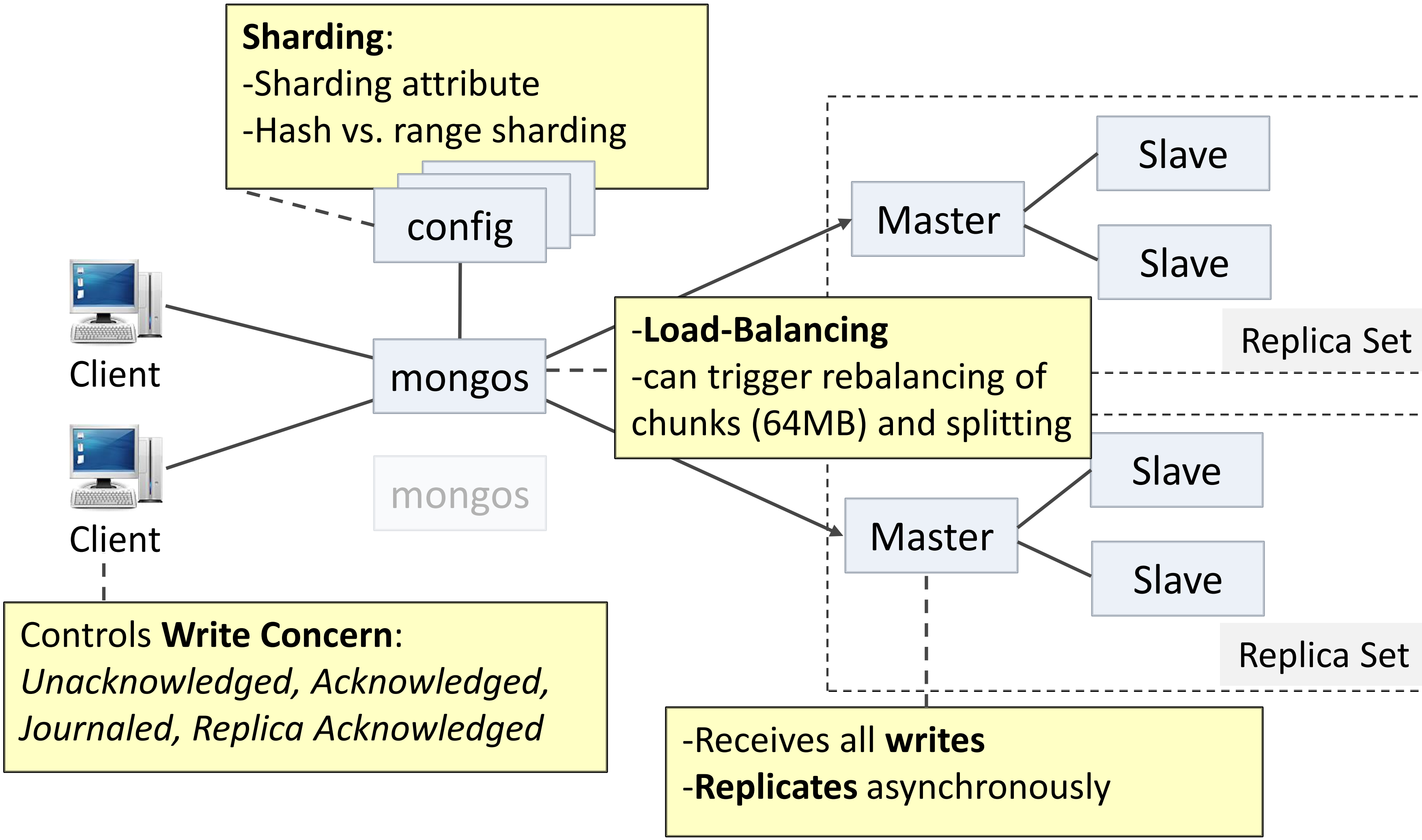
Nesting replaces 1:n and 1:1 relations

Schemafreeness: Attributes per document

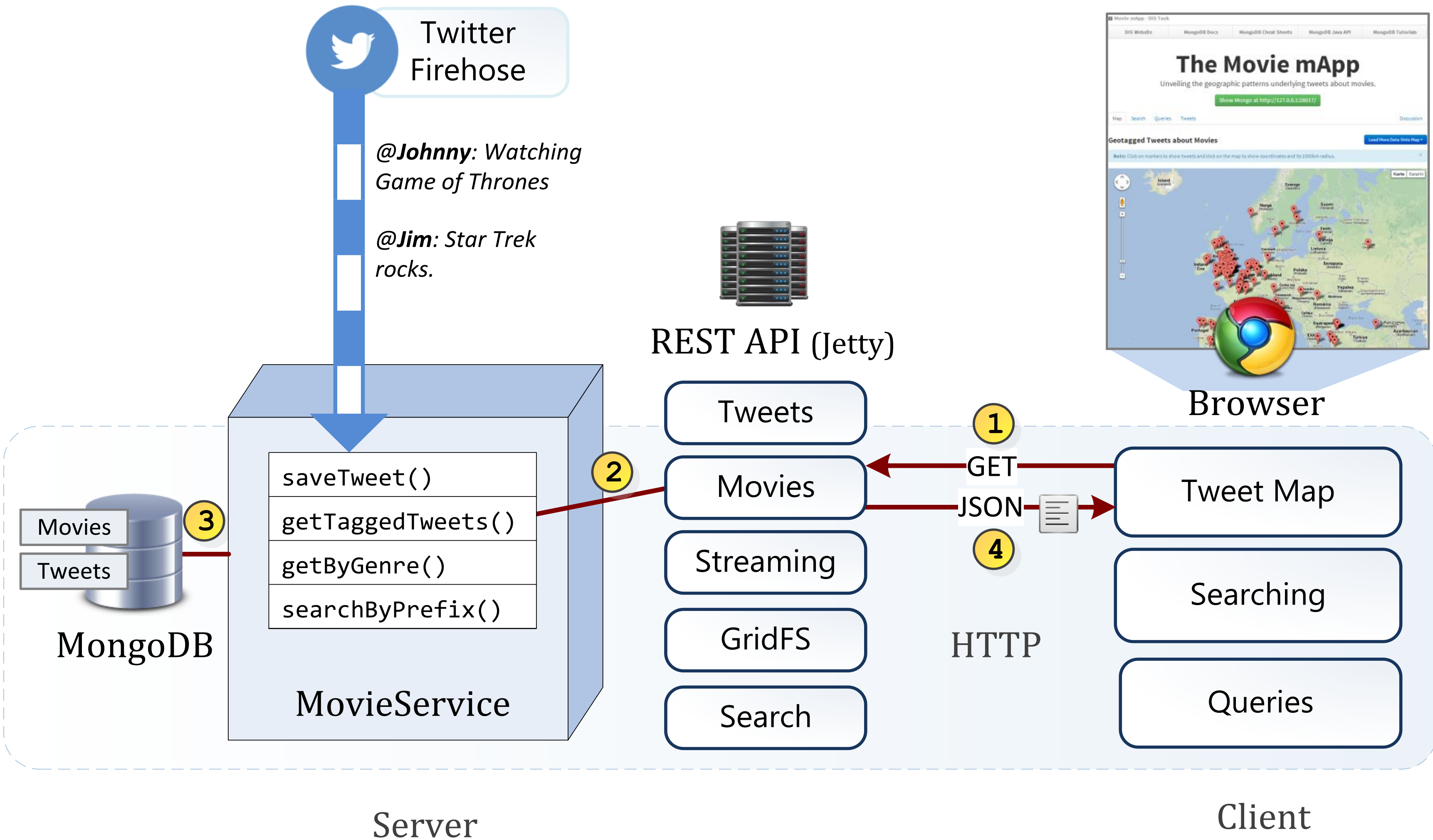
Unit of atomicity: document

Principles

Sharding und Replication



MongoDB Example App



MongoDB by Example

Movie mApp - DIS Task

DIS Website | MongoDB Tutorials

The Movie mApp

Unveiling the geographic patterns underlying tweets about movies.

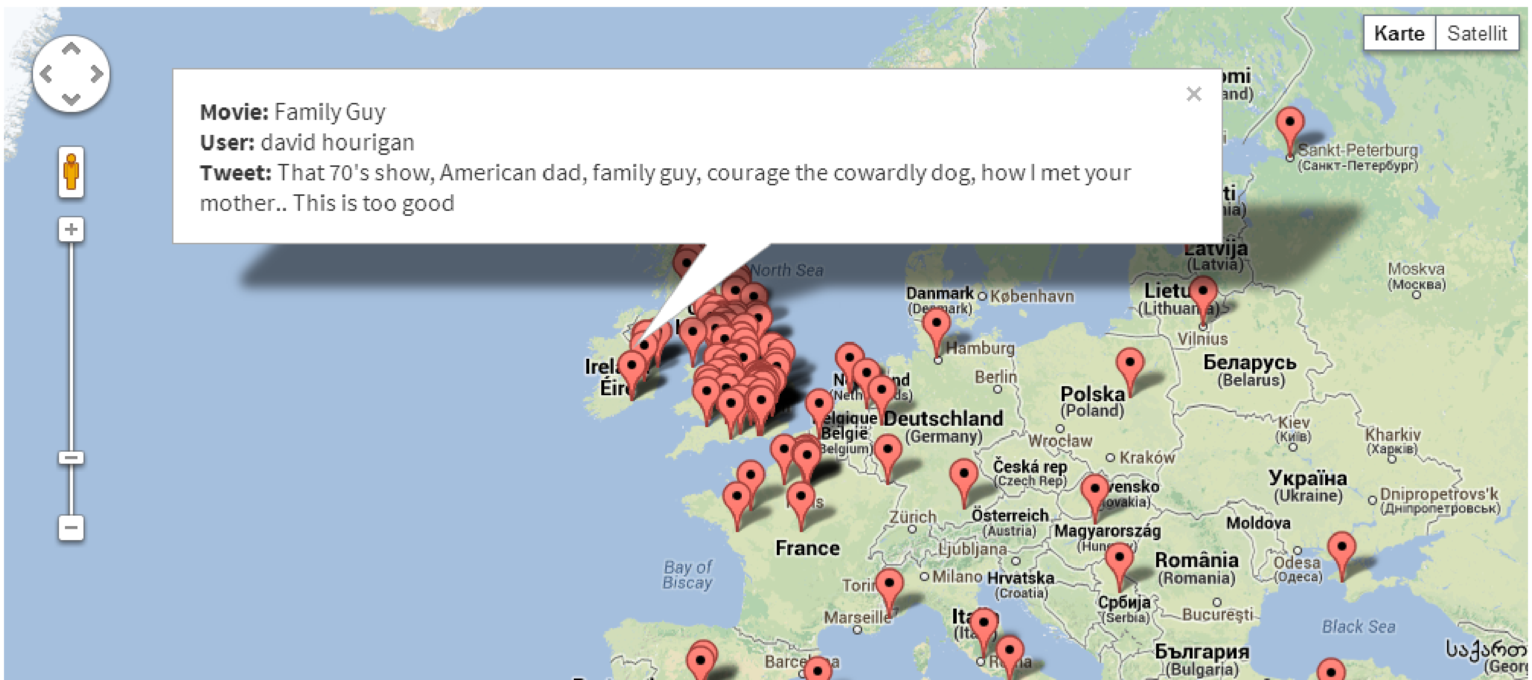
Show Mongo at http://127.0.0.1:28017/

Map | Search | Queries | Tweets | Discussion

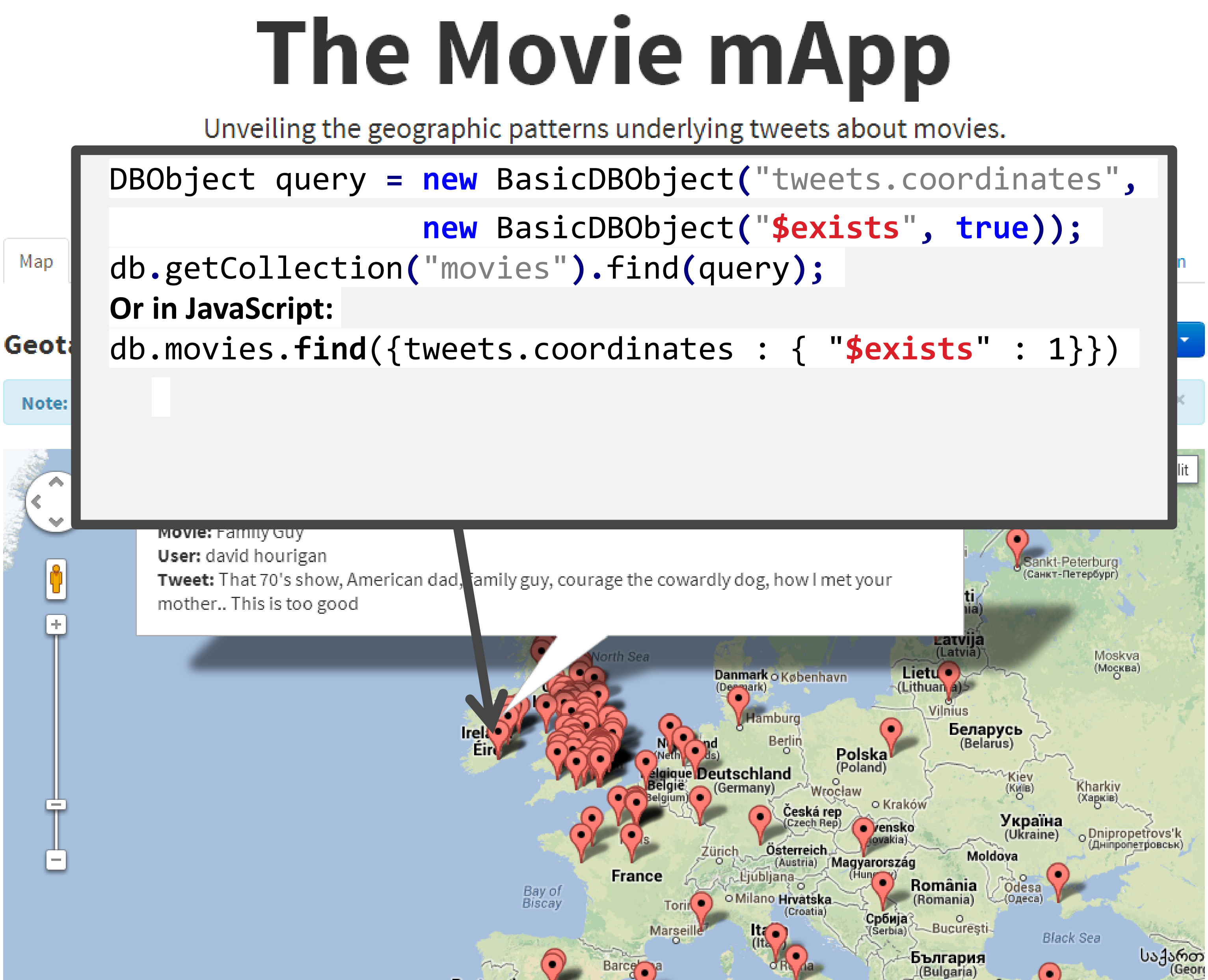
Geotagged Tweets about Movies

Load More Data Onto Map

Note: Click on markers to show tweets and click on the map to show coordinates and its 1000km radius.



MongoDB by Example




```java
DBObject query = new BasicDBObject("tweets.coordinates",
                  new BasicDBObject("$exists", true));
db.getCollection("movies").find(query);
```

Or in JavaScript:

```javascript
db.movies.find({tweets.coordinates : { "$exists" : 1}})
```

MongoDB by Example

Movie mApp - DIS Task

DIS Website | MongoDB Docs | MongoDB Cheat Sheet | MongoDB Java API | MongoDB Tutorials

The Movie mApp

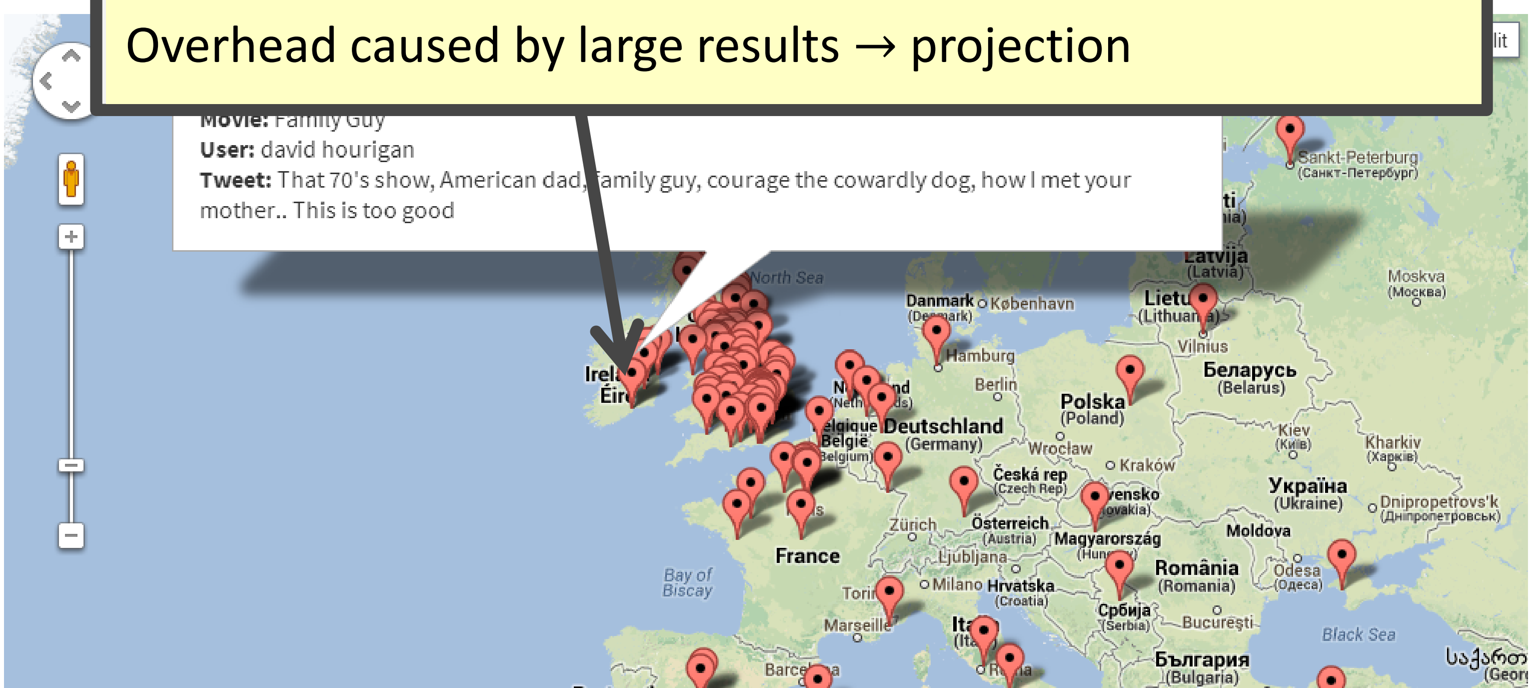
Unveiling the geographic patterns underlying tweets about movies.

```
DBObject query = new BasicDBObject("tweets.coordinates",
                 new BasicDBObject("$exists", true));
db.getCollection("movies").find(query);
```

Or in JavaScript:

```
db.movies.find({tweets.coordinates : { "$exists" : 1}})
```

Overhead caused by large results → projection



Movie mApp - DIS Task

DIS Website | MongoDB Docs | MongoDB Cheat Sheets | MongoDB Java API | MongoDB Tutorials

The Movie mApp

Unveiling the geographic patterns underlying tweets about movies.

Show Mongo at http://127.0.0.1:28017/

Map

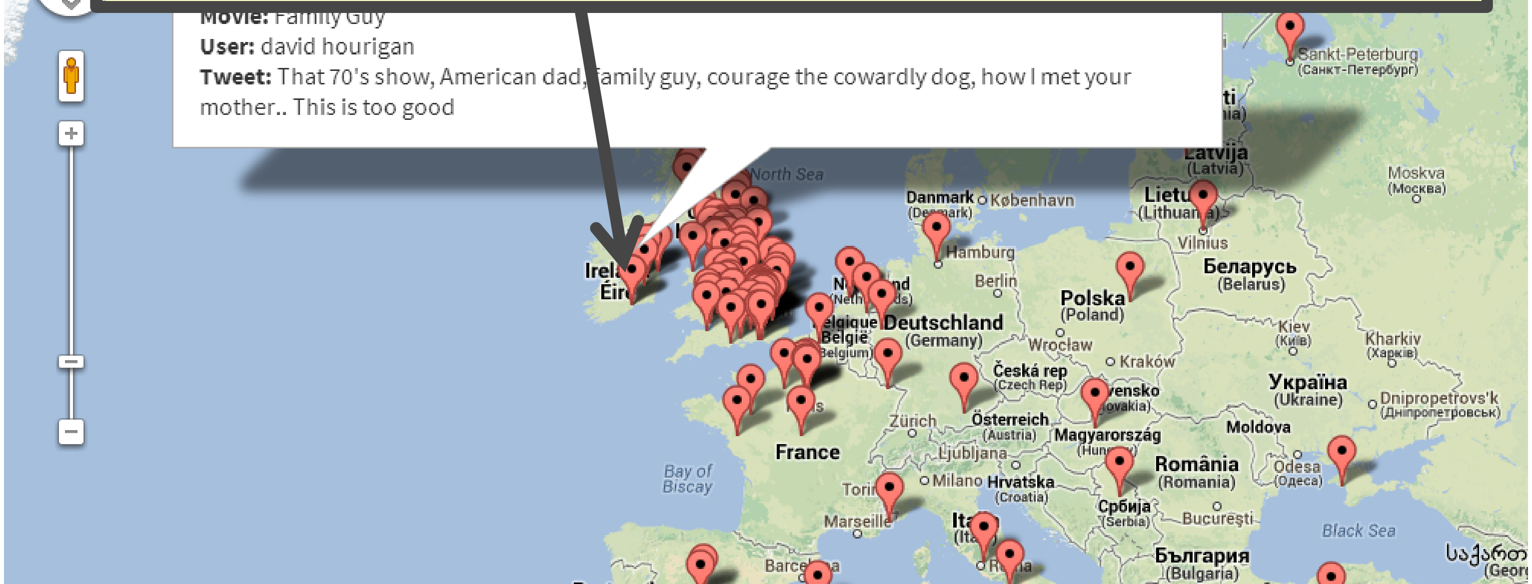
Geot

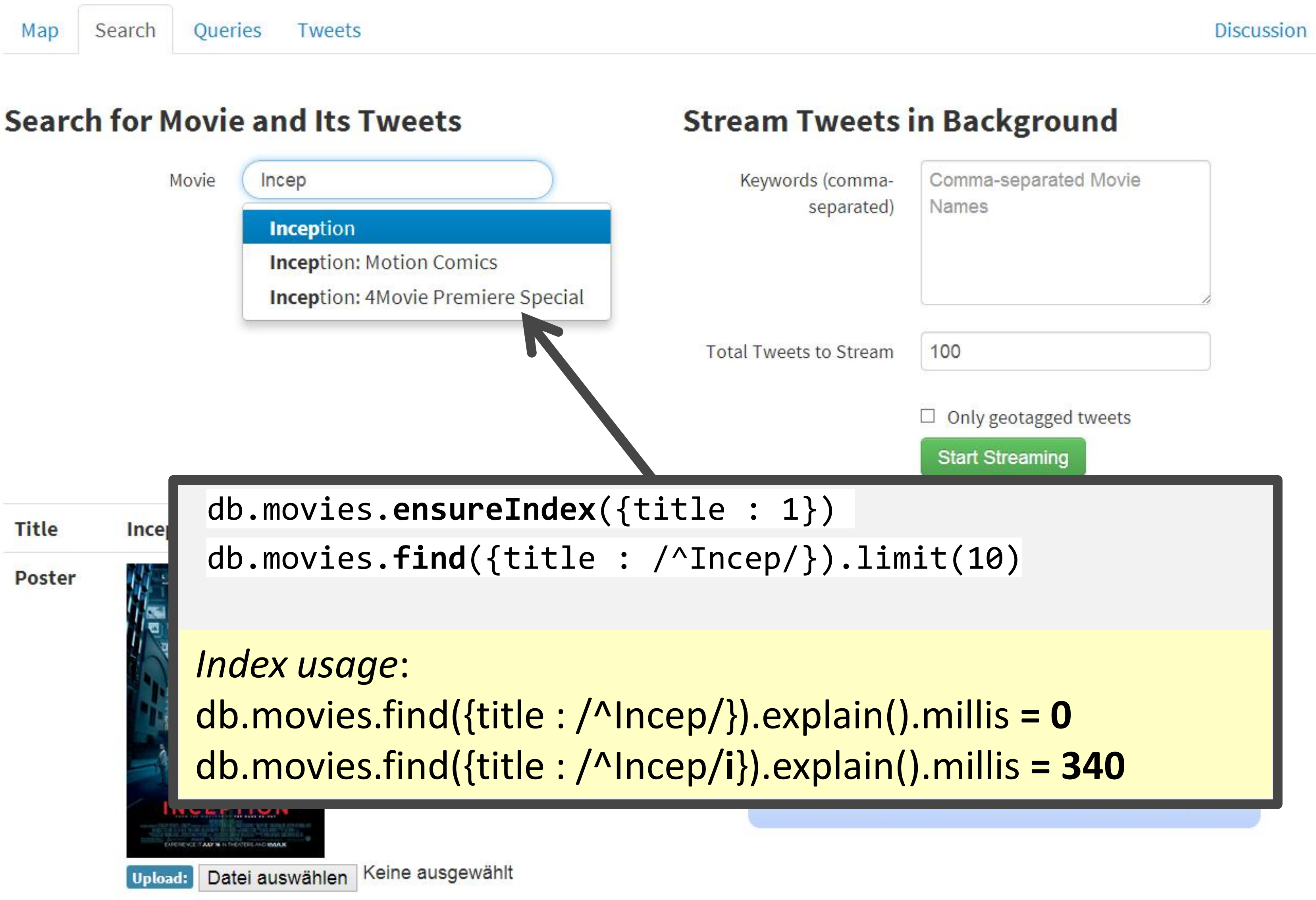
Note:

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

Projected attributes, ordered by insertion date

Movie: Family Guy
User: david hourigan
Tweet: That 70's show, American dad, family guy, courage the cowardly dog, how I met your mother.. This is too good



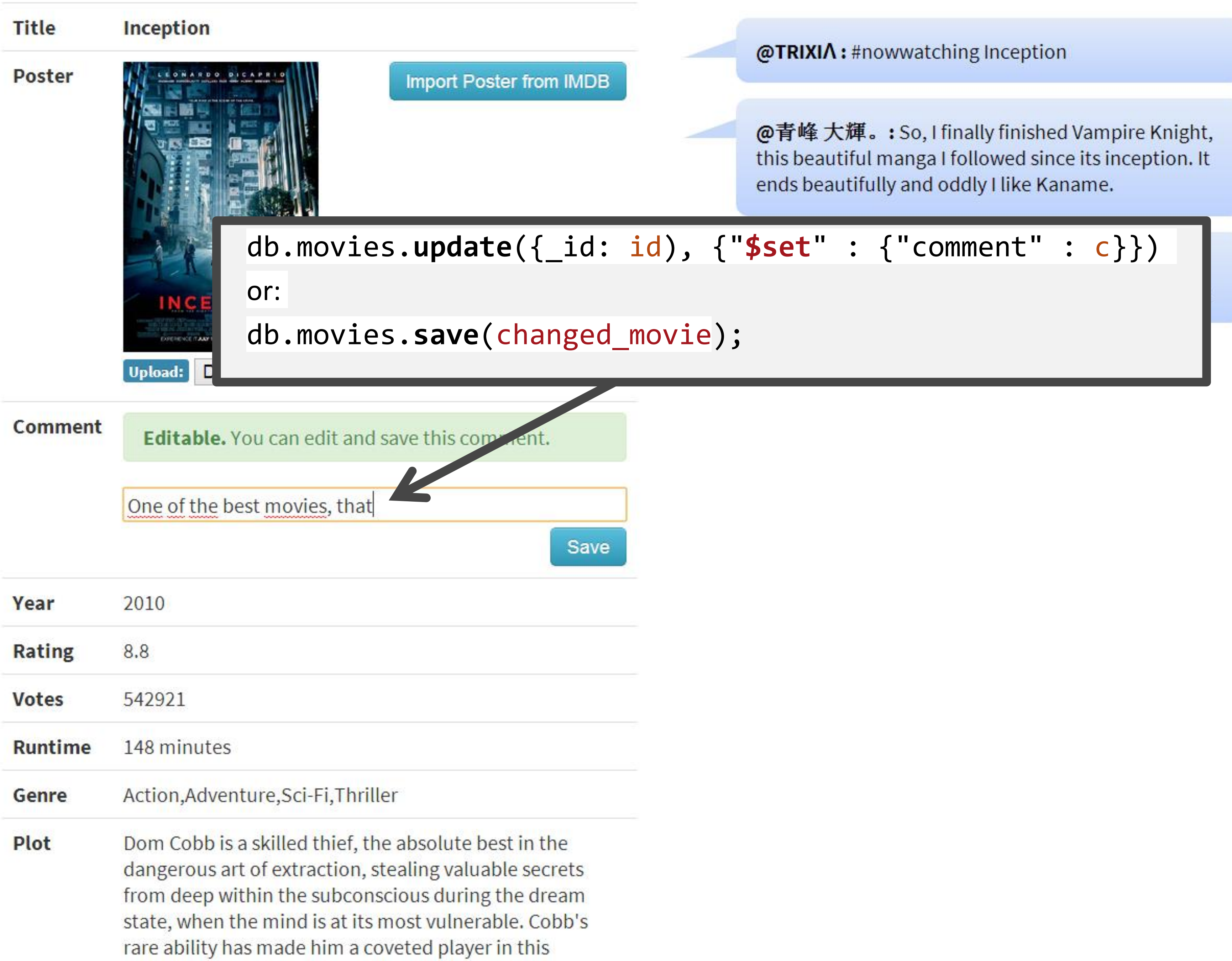


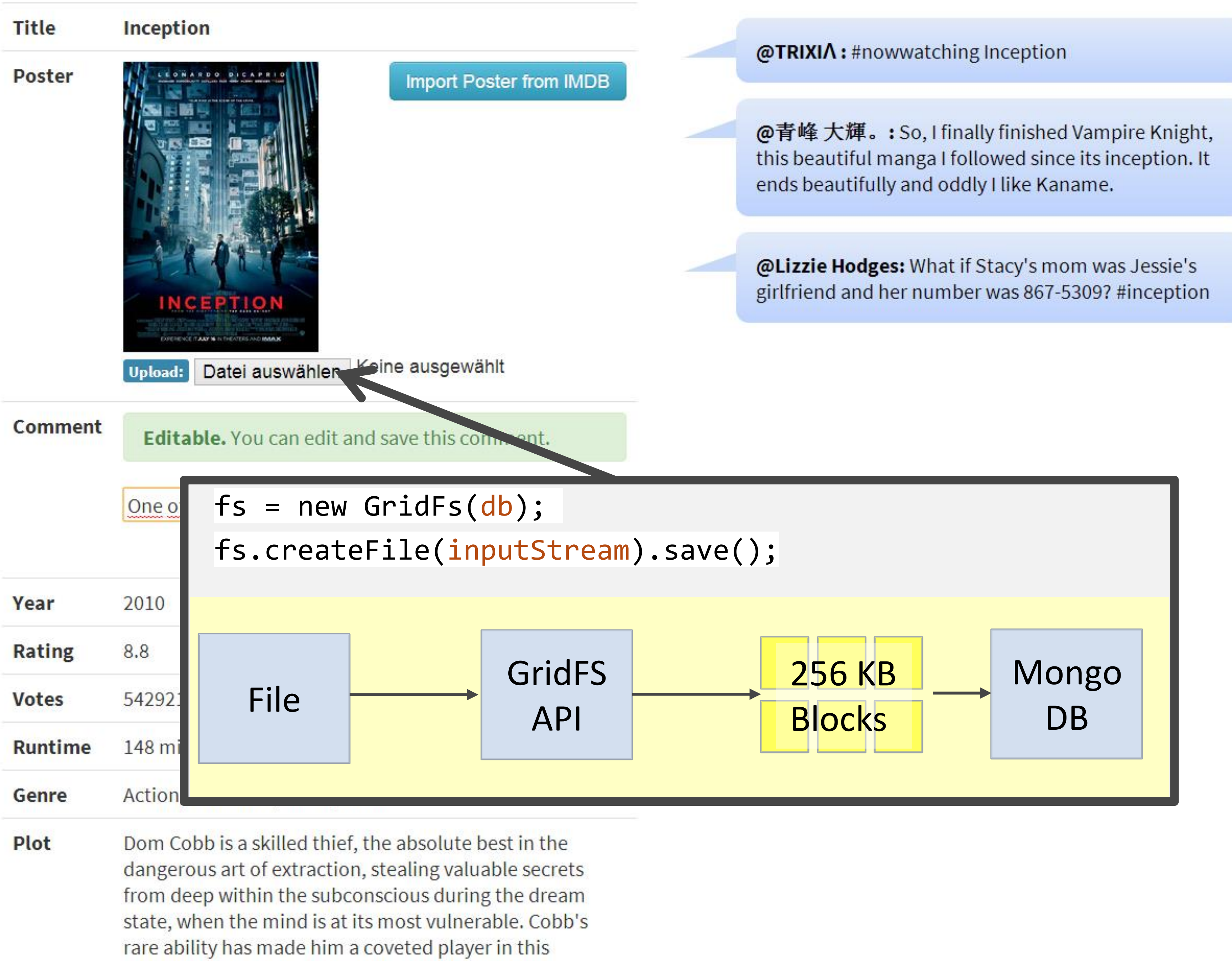

```
db.movies.ensureIndex({title : 1})
db.movies.find({title : /^Incep/}).limit(10)
```

Index usage:

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

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


Title
Inception
Poster
Import Poster from IMDB
Upload:
@TRIXIΛ : #nowwatching Inception
@青峰 大輝。: So, I finally finished Vampire Knight, this beautiful manga I followed since its inception. It ends beautifully and oddly I like Kaname.
db.movies.update({_id: id), {"$set" : {"comment" : c}})
or:
db.movies.save(changed_movie);
Comment
Editable. You can edit and save this comment.
One of the best movies, that
Save
Year
2010
Rating
8.8
Votes
542921
Runtime
148 minutes
Genre
Action,Adventure,Sci-Fi,Thriller
Plot
Dom Cobb is a skilled thief, the absolute best in the dangerous art of extraction, stealing valuable secrets from deep within the subconscious during the dream state, when the mind is at its most vulnerable. Cobb's rare ability has made him a coveted player in this


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

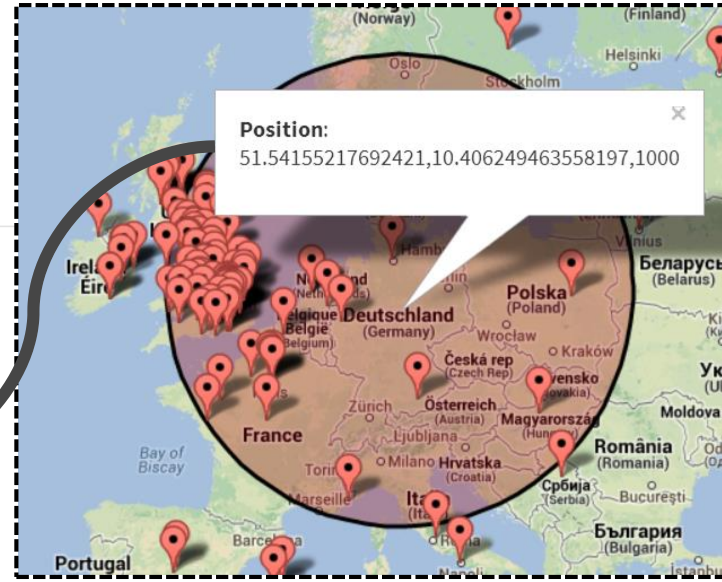
Map | Search | Queries | Tweets

Query Tweets

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

Parameter: 51.54155217692421,10.406249463558197,1000

Result Limit: 10




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

Geospatial Queries:

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

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

Map Search Queries Tweets Discussion

Query Tweets

Query: Indexed Fulltext Search on Tweets [Go]

Parameter: StAr trek

Result Limit: 100

Show 25 search results per page

Filter search results:

```
db.tweets.runCommand( "text", { search: "StAr trek" } )
```

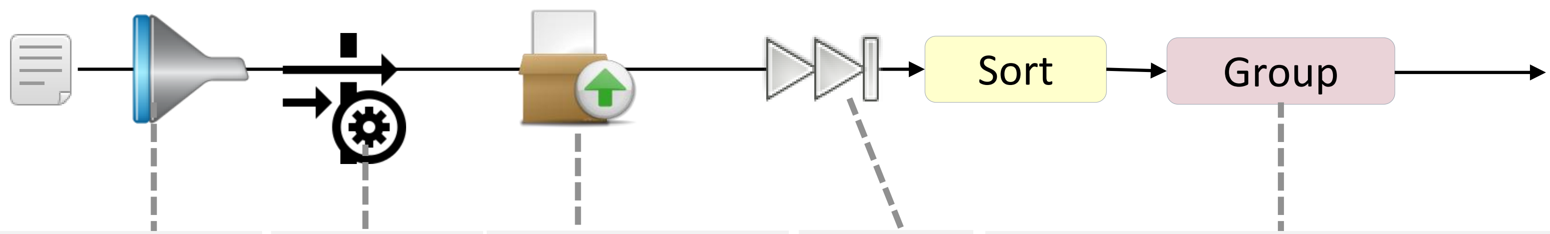
Full-text Search:

- **Tokenization, Stop Words**
- **Stemming**
- **Scoring**

User	Tweet	Created at	Coordinates
manwonman			
Mia Clrss Hrnndz ♡			
A N G G I_			
Stefany Ezra Elvina		2013	
Vanessa Yung	Star Trek into Darkness☐	Wed May 29 19:21:06 +0000 2013	-2.986771,53.404051
tam wilson	Finally getting to see Star Trek! (at @DCADundee Contemporary Arts for Star Trek Into Darkness 3D) http://t.co/0ojg4KMBL5	Wed May 29 18:48:56 +0000	-2.97489166,56.45753477

Analytic Capabilities

- Aggregation Pipeline Framework:



Match: Selection by query

Projection

Unwind: elimination of nesting

Skip and **Limit**

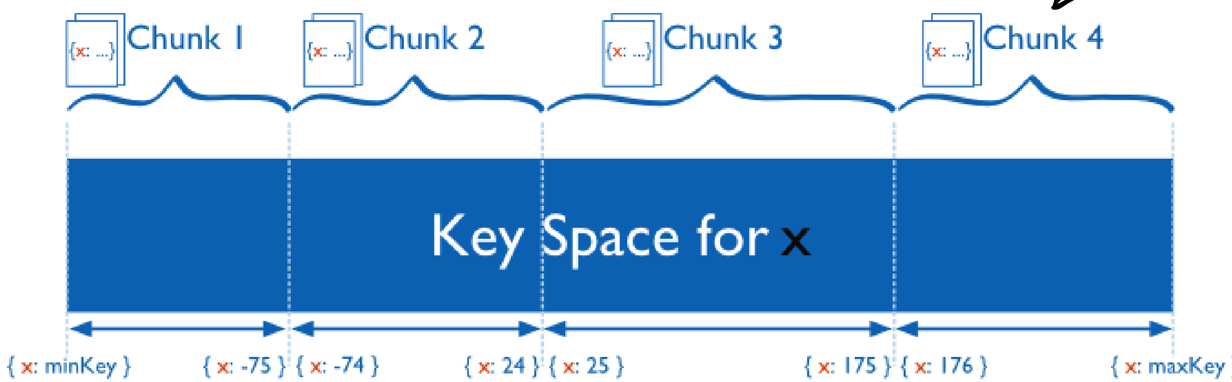
Grouping, e.g.
{ _id : "$author",
docsPerAuthor : { $sum : 1 },
viewsPerAuthor : { $sum : "$views" } }});

- Alternative: JavaScript MapReduce

Sharding

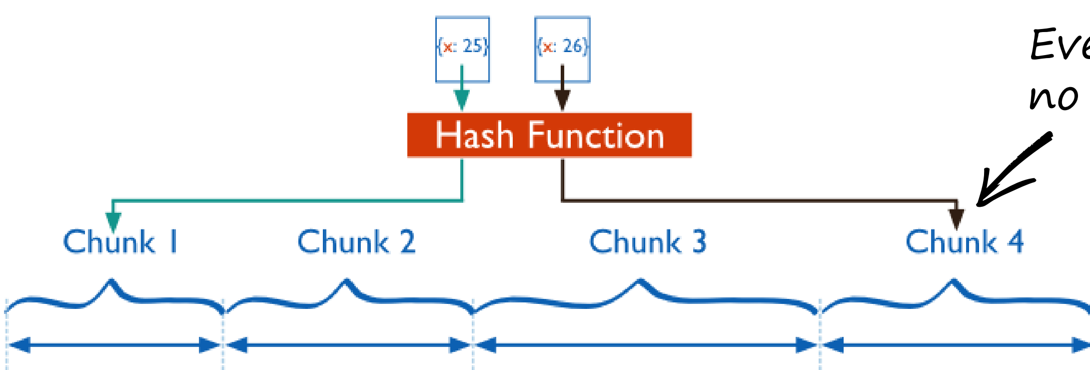
- Range-based:

In the optimal case only one shard asked per query, else: Scatter-and-gather



- Hash-based:

Even distribution, no locality

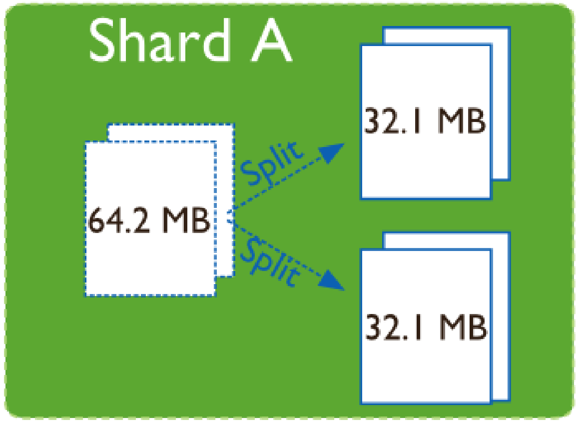


docs.mongodb.org/manual/core/sharding-introduction/

Sharding

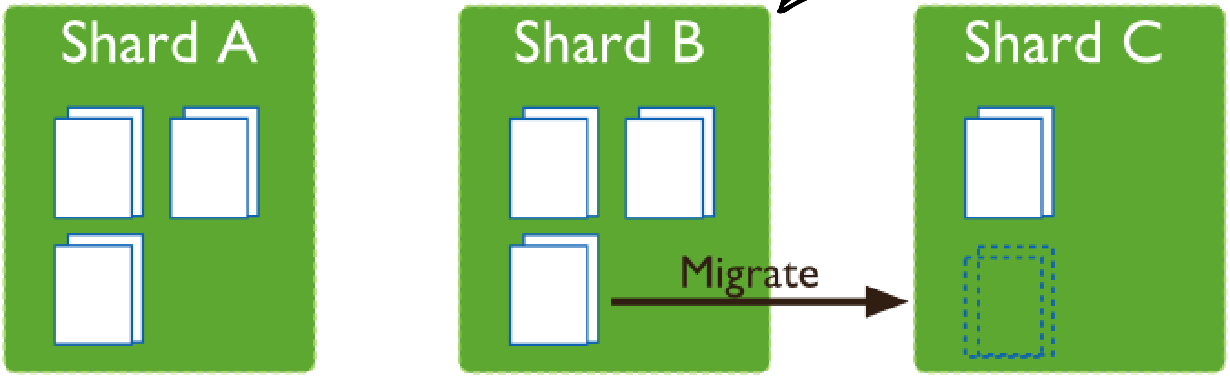
- Splitting:

Split chunks that are too large



- Migration:

Mongos Load Balancer triggers rebalancing



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

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) 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)
- **Microsoft Cloud SQL Server** (distributed CP, MSSQL-comp.)
- **MySQL Cluster, Galera Cluster, Percona XtraDB Cluster** (distributed storage engine for MySQL)

Open Research Questions

For Scalable Data Management

- **Service-Level Agreements**
  - How can SLAs be guaranteed in a virtualized, multi-tenant cloud environment?
- **Consistency**
  - Which consistency guarantees can be provided in a geo-replicated system without sacrificing availability?
- **Performance & Latency**
  - How can a database deliver low latency in face of distributed storage and application tiers?
- **Transactions**
  - Can ACID transactions be aligned with NoSQL and scalability?

Distributed Transactions

ACID and Serializability

Definition: A transaction is a sequence of operations transforming the database from one consistent state to another.

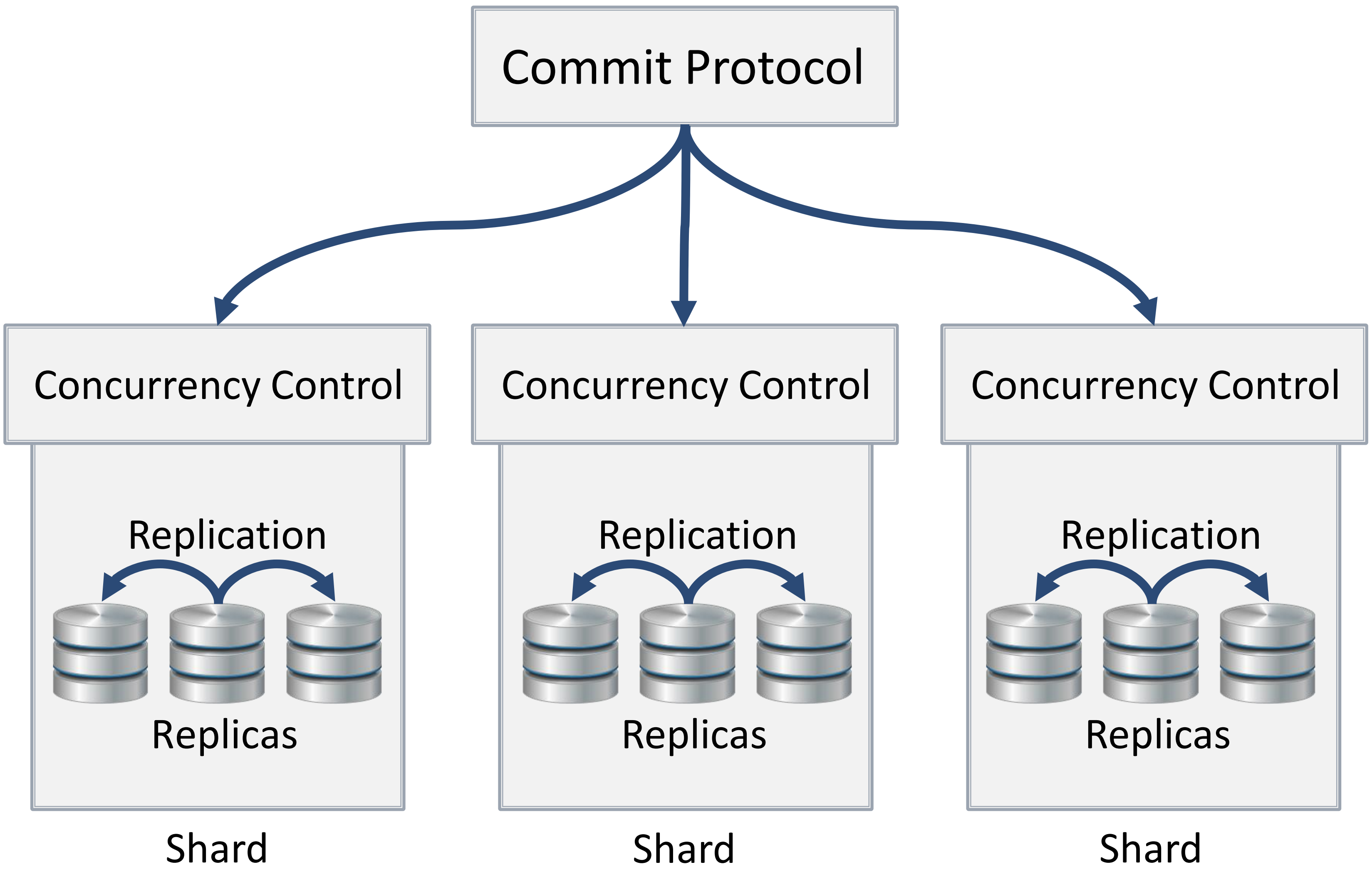
- Atomicity
- Consistency
- Isolation
- Durability

Isolation Levels:

1. Serializability
2. Snapshot Isolation
3. Read-Committed
4. Read-Atomic
5. …

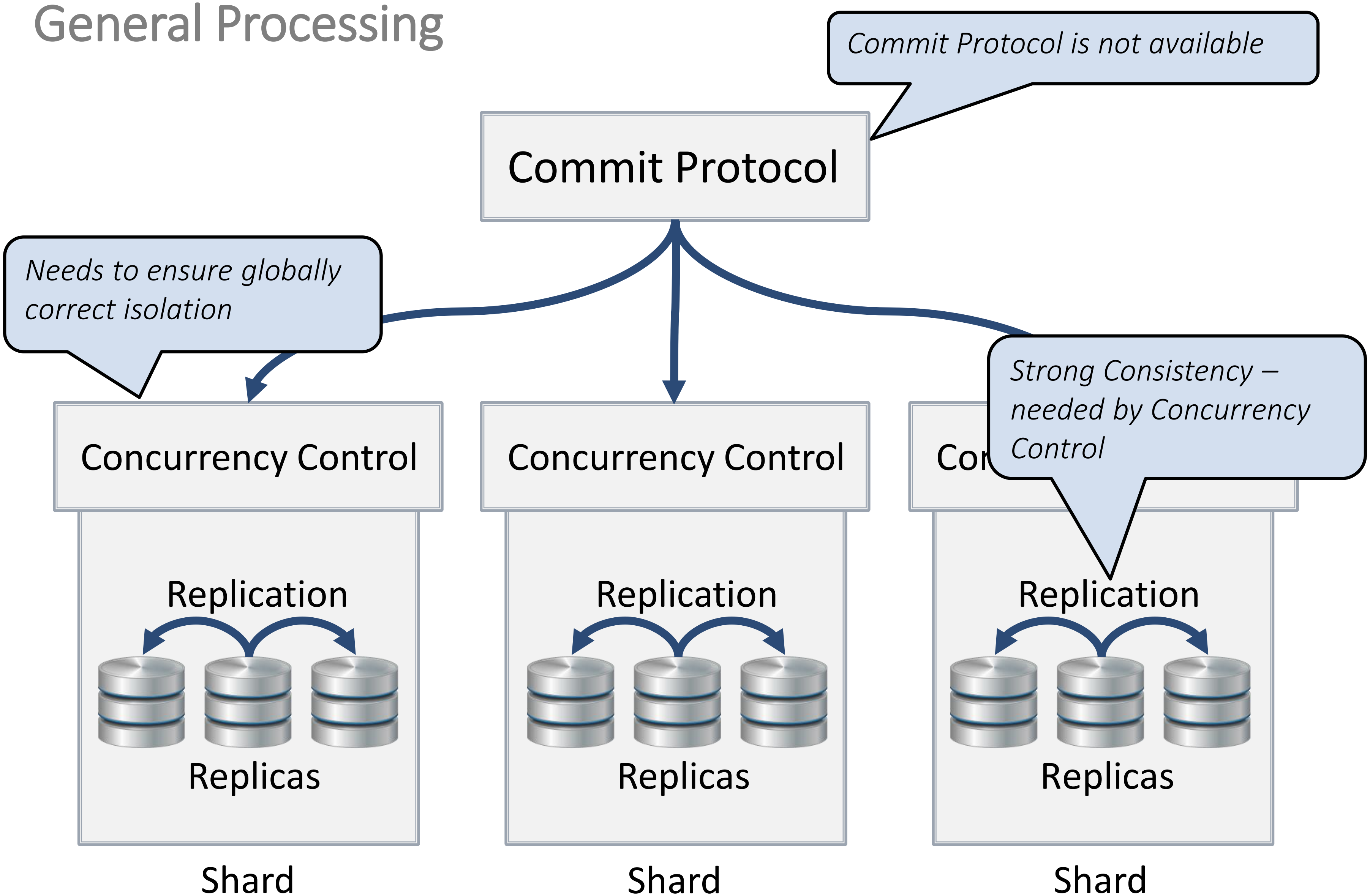
Distributed Transactions

General Processing



Distributed Transactions

General Processing



Distributed Transactions

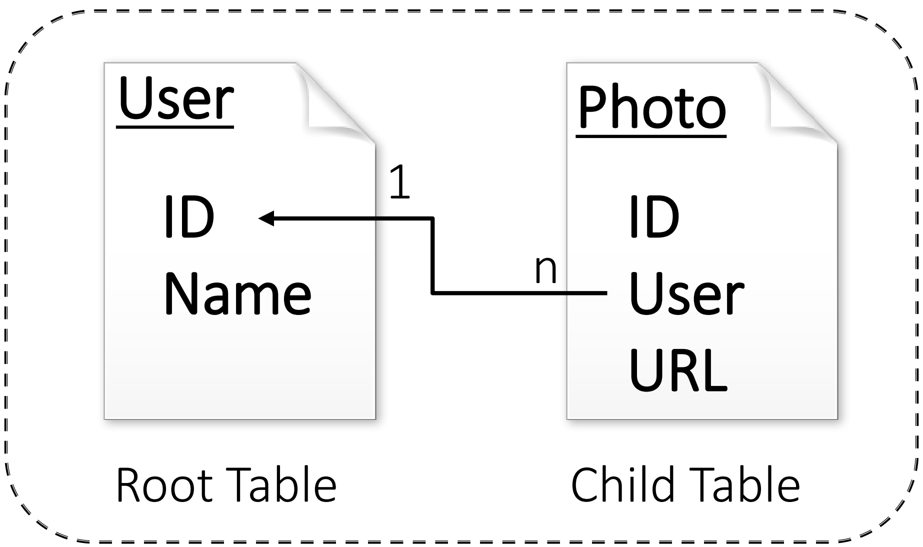
In NoSQL Systems – An Overview

System	Concurrency Control	Isolation	Granularity	Commit Protocol
Megastore	OCC	SR	Entity Group	Local
G-Store	OCC	SR	Entity Group	Local
ElasTras	PCC	SR	Entity Group	Local
Cloud SQL Server	PCC	SR	Entity Group	Local
Spanner / F1	PCC / OCC	SR / SI	Multi-Shard	2PC
Percolator	OCC	SI	Multi-Shard	2PC
MDCC	OCC	RC	Multi-Shard	Custom – 2PC like
CloudTPS	TO	SR	Multi-Shard	2PC
Cherry Garcia	OCC	SI	Multi-Shard	Client Coordinated
Omid	MVCC	SI	Multi-Shard	Local
FaRMville	OCC	SR	Multi-Shard	Local
H-Store/VoltDB	Deterministic CC	SR	Multi-Shard	2PC
Calvin	Deterministic CC	SR	Multi-Shard	Custom
RAMP	Custom	Read-Atomic	Multi-Shard	Custom

Distributed Transactions

Megastore

- Synchronous Paxos-based replication
- Fine-grained partitions (entity groups)
- Based on BigTable
- Local commit protocol, optmisistic concurrency control



EG: User + n Photos

- Unit of ACID **transactions**/ consistency
- Local commit protocol, optimistic concurrency control

Distributed Transactions

Megastore

Spanner

Idea:
- Auto-sharded Entity Groups
- Paxos-replication per shard

Transactions:
- **Multi-shard** transactions
- **SI** using **TrueTime** API (GPA and atomic clocks)
- **SR** based on **2PL** and **2PC**
- Core of **F1** powering ad business

J. Corbett et al. "Spanner: Google's globally distributed database." TOCS 2013

Percolator

Idea:
- Indexing and transactions based on BigTable

Implementation:
- Metadata columns to coordinate transactions
- Client-coordinated 2PC
- Used for search index (not OLTP)

Peng, Daniel, and Frank Dabek. "Large-scale Incremental Processing Using Distributed Transactions and Notifications." OSDI 2010.

URL

Root Table

Child Table

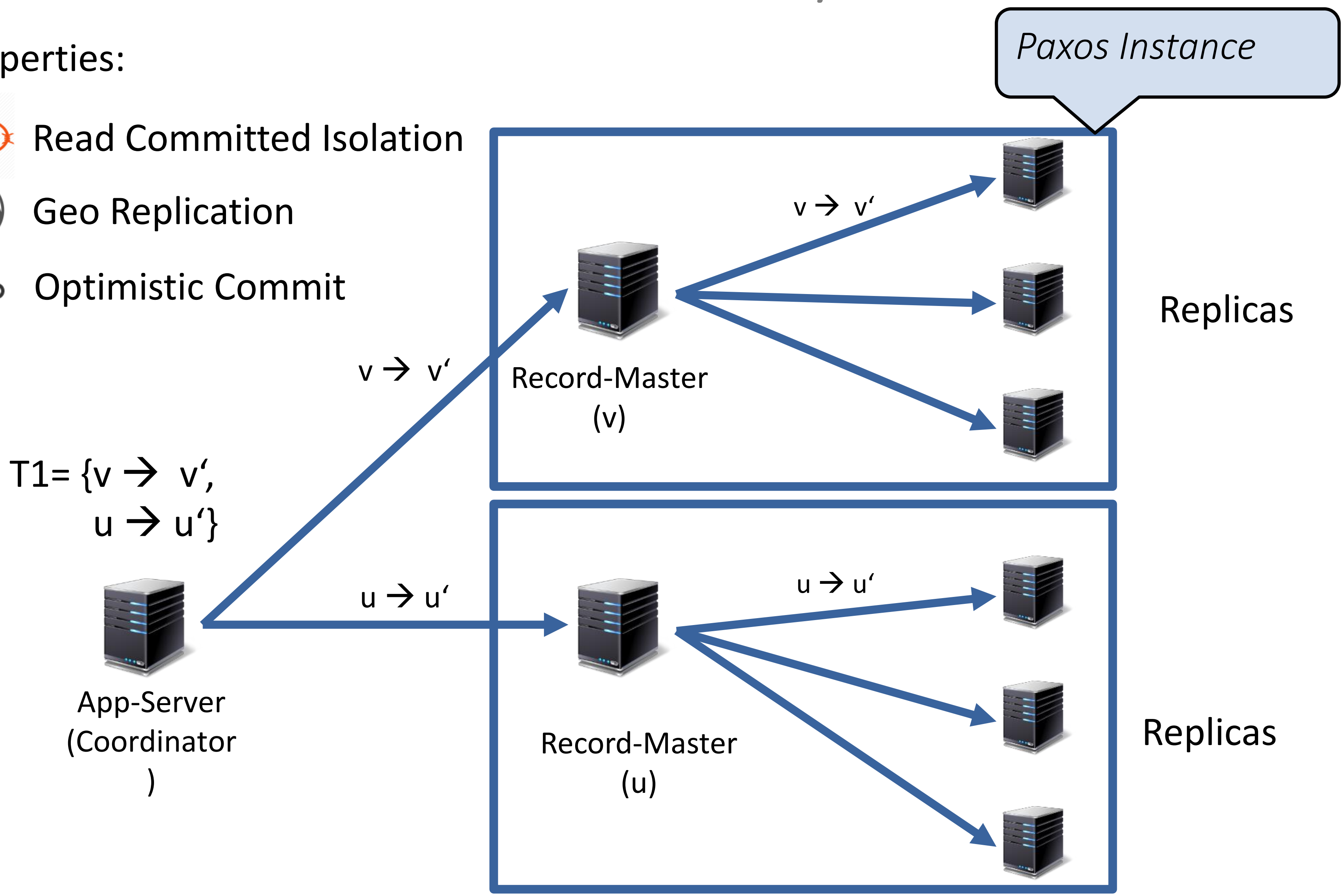
optimistic concurrency control

Distributed Transactions

MDCC – Multi Datacenter Concurrency Control

Properties:

- Read Committed Isolation
- Geo Replication
- Optimistic Commit



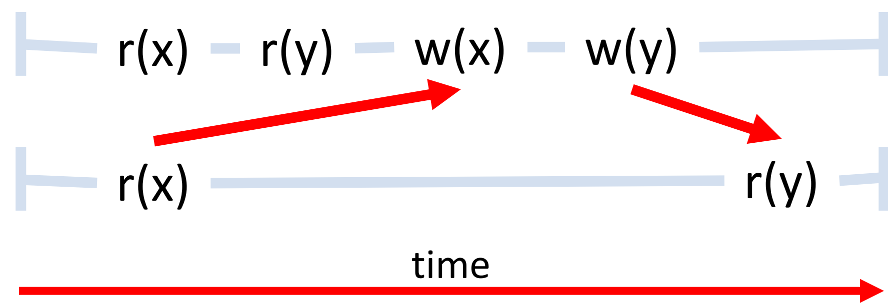
Distributed Transactions

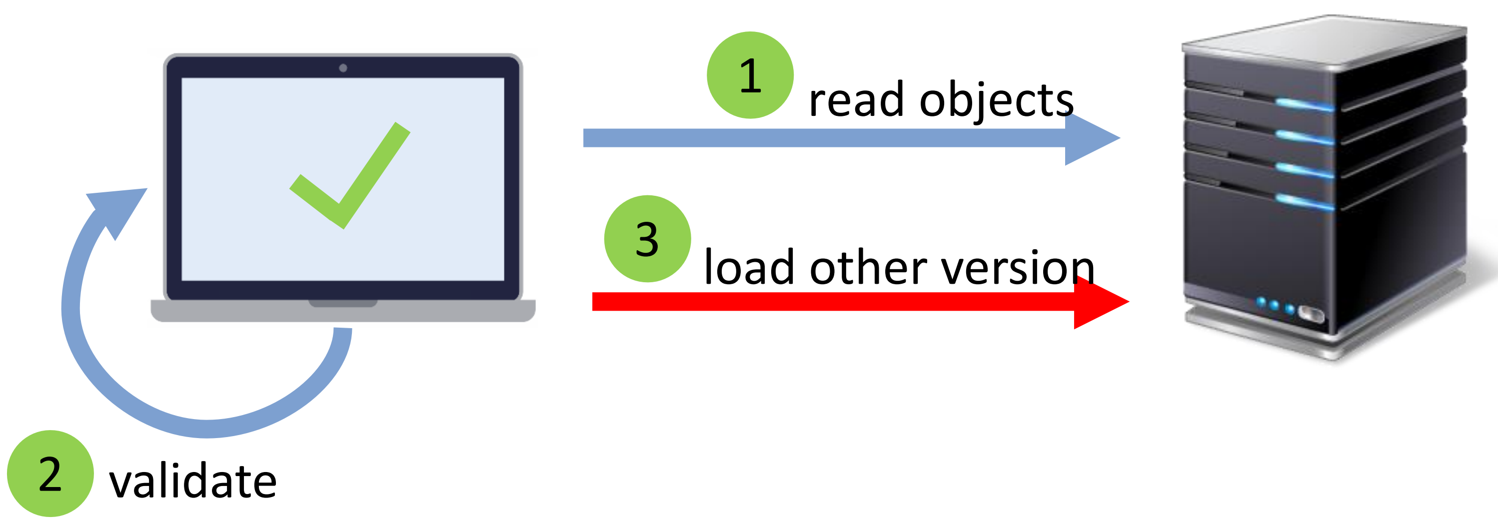
RAMP – Read Atomic Multi Partition Transactions

Properties:

- Read Atomic Isolation
- Synchronization Independence
- Partition Independence
- Guaranteed Commit

Fractured Read

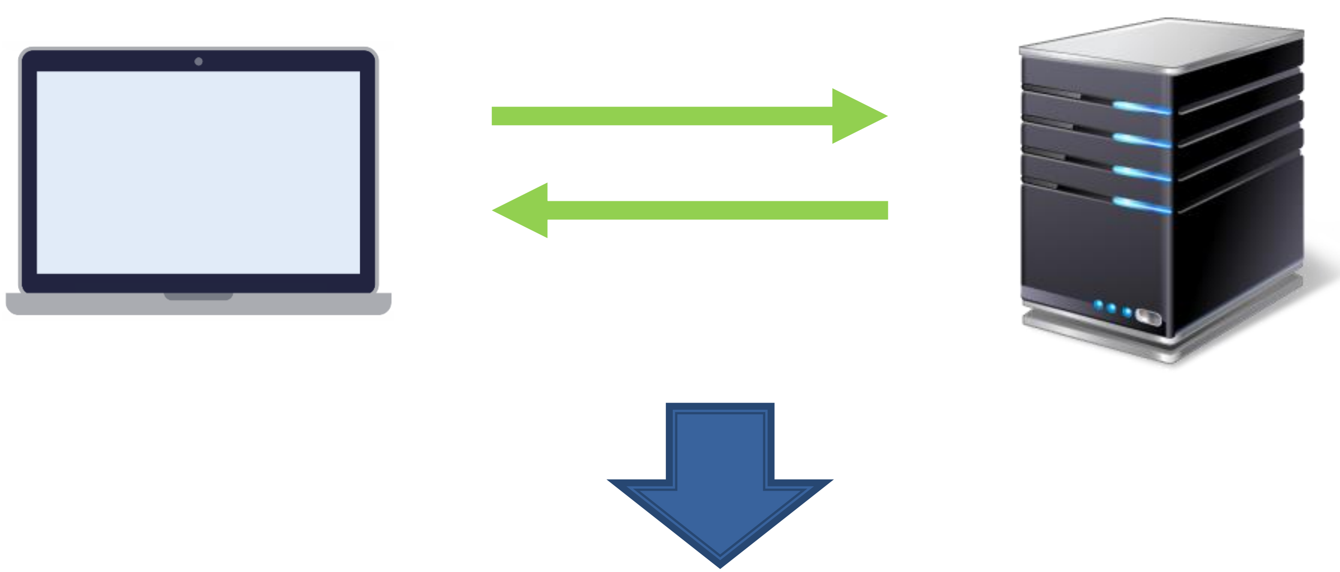




Distributed Transactions in the Cloud

The Latency Problem

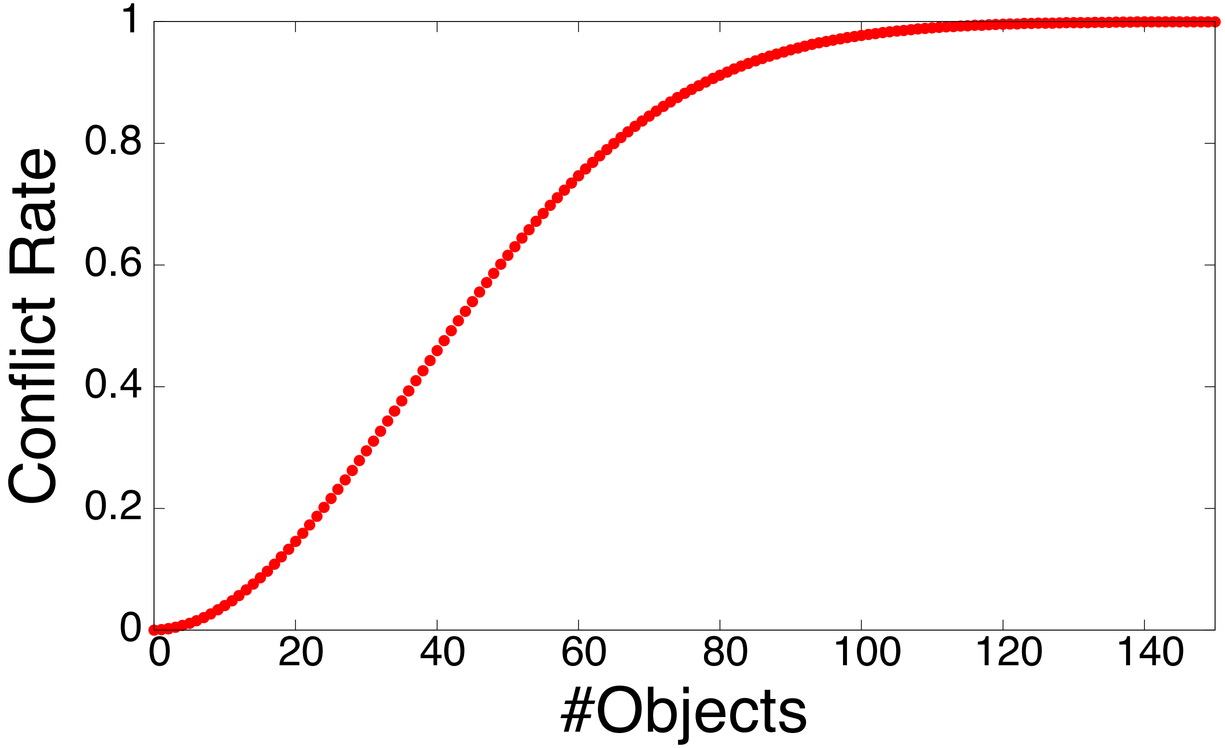
Interactive Transactions:



Optimistic Concurrency Control

Optimistic Concurrency Control

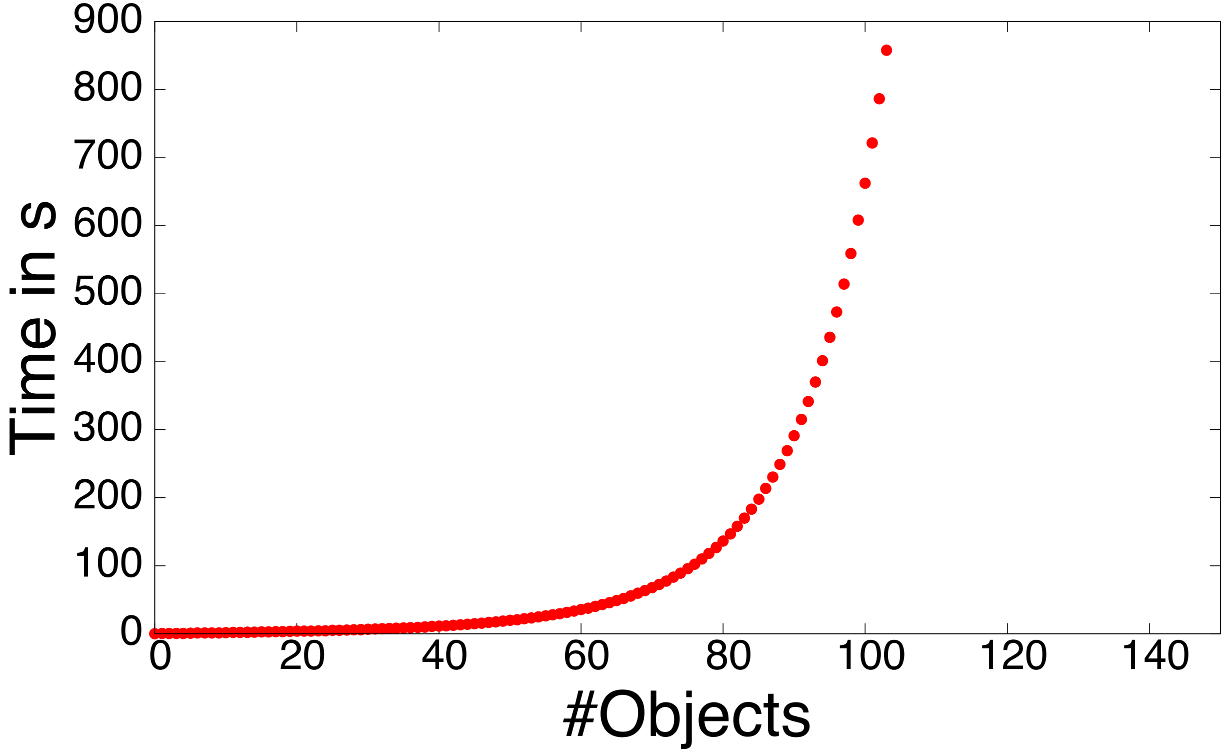
The Abort Rate Problem



- 10.000 objects
- 20 writes per second
- 95% reads

Optimistic Concurrency Control

The Abort Rate Problem

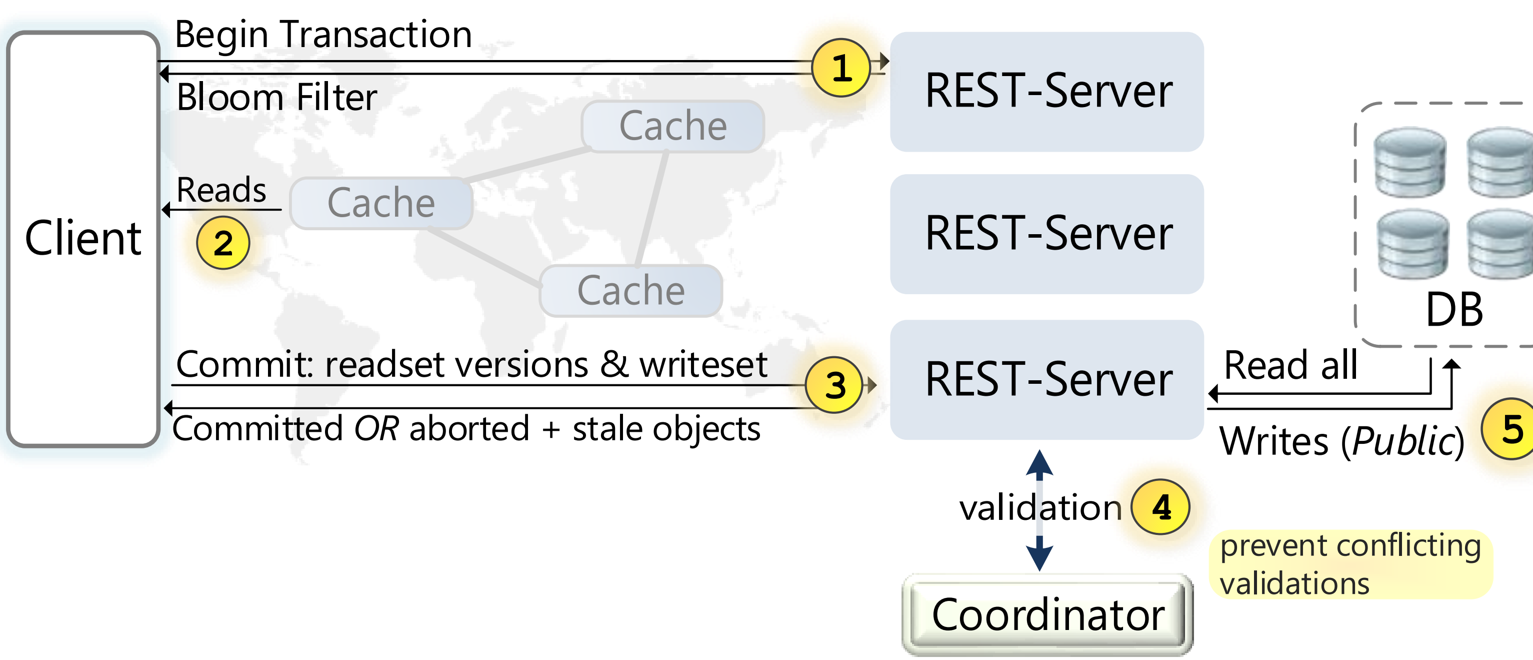


- 10.000 objects
- 20 writes per second
- 95% reads

Distributed Cache-Aware Transaction

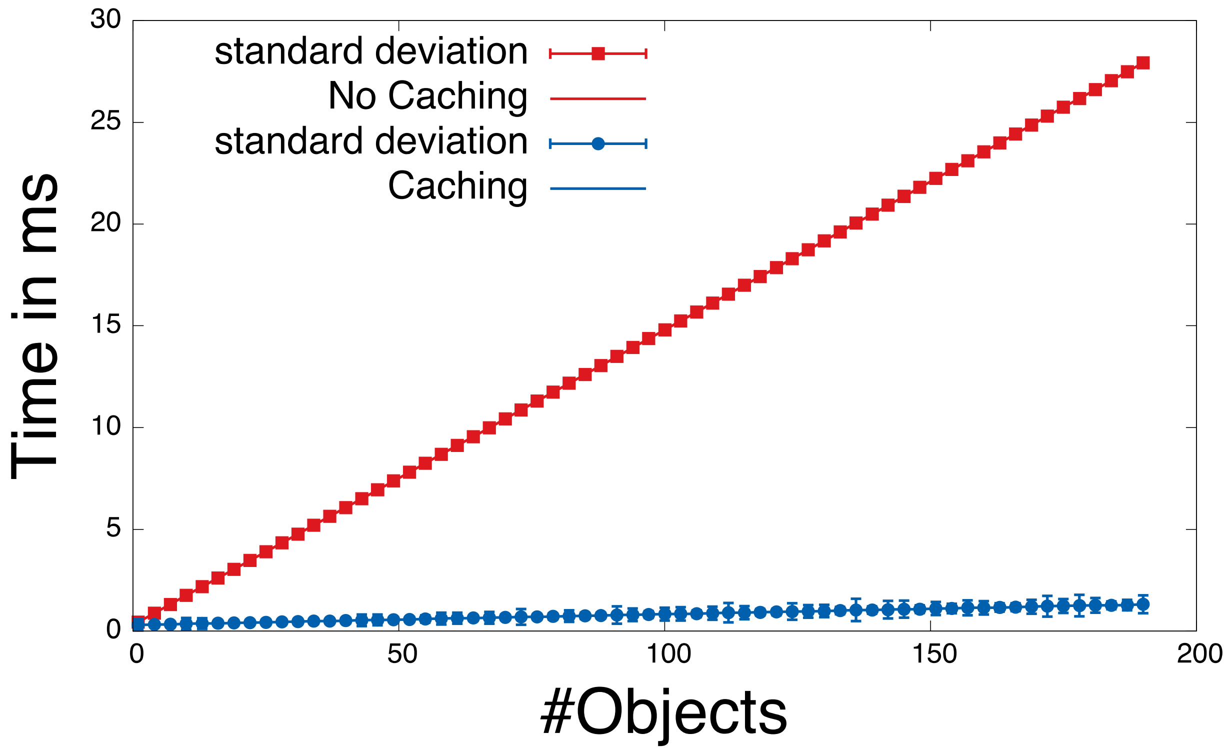
Scalable ACID Transactions

- Solution: **Conflict-Avoidant Optimistic Transactions**
  - **Cached reads → Shorter transaction duration → less aborts**
  - Bloom Filter to identify outdated cache entries



Distributed Cache-Aware Transaction

Speed Evaluation

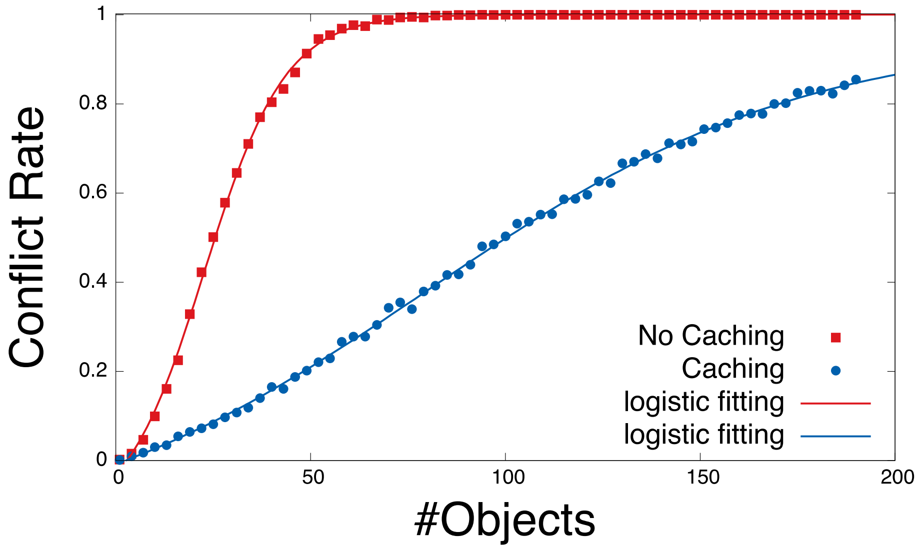


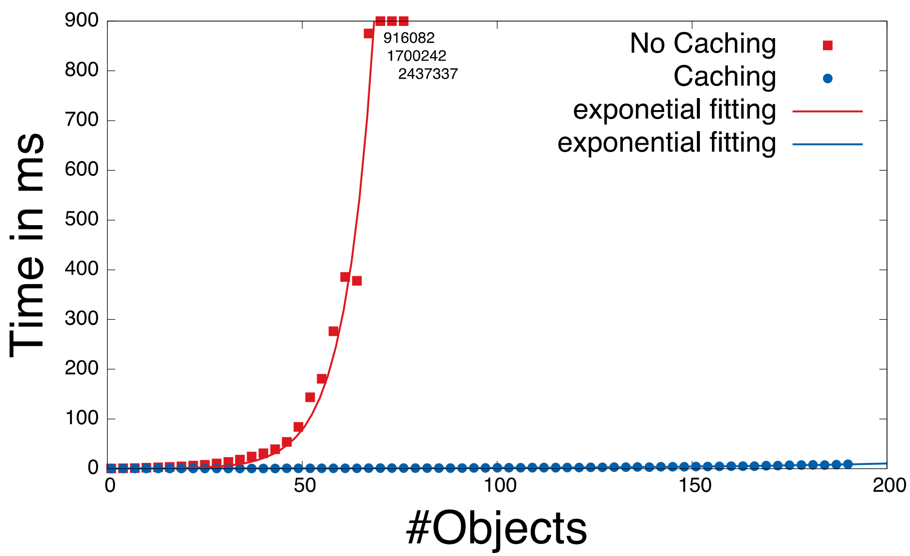
- 10.000 objects
- 20 writes per second
- 95% reads

➢ **16 times** speedup

Distributed Cache-Aware Transaction

Abort Rate Evaluation

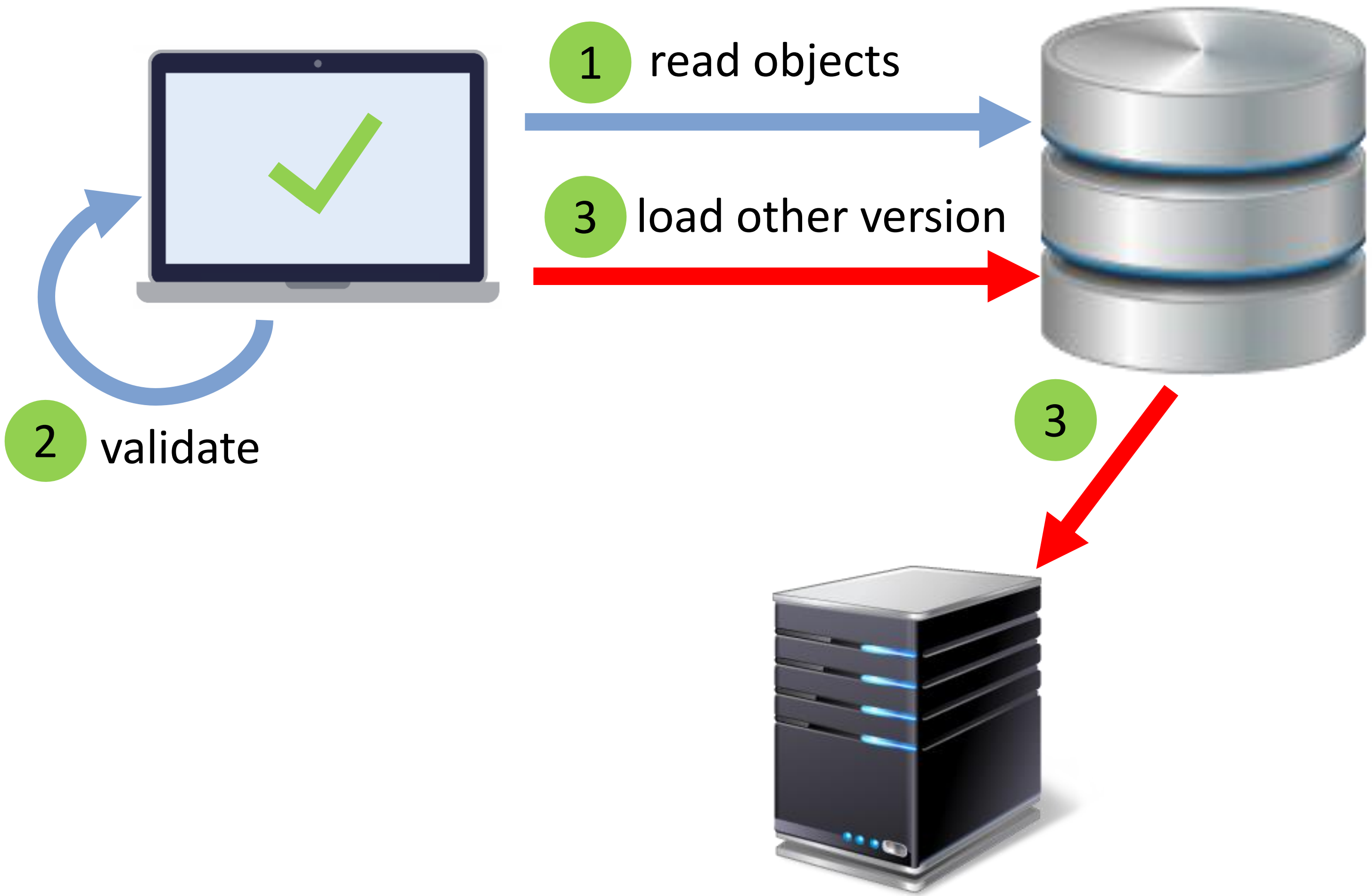




- 10.000 objects
- 20 writes per second
- 95% reads
- ➢ 16 times speedup
- ➢ Significantly less aborts
- ➢ Highly reduced runtime of retried transactions

Distributed Cache-Aware Transaction

Combined with RAMP Transactions



Selected Research Challanges

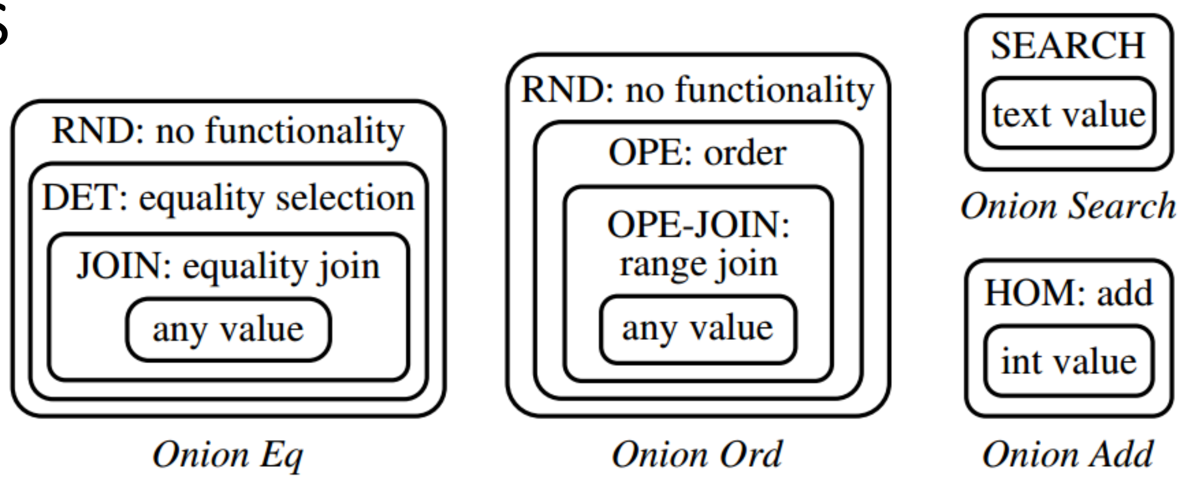
Encrypted Databases

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

SQL-Proxy

Encrypts and decrypts, rewrites queries

RDBMS



Selected Research Challanges

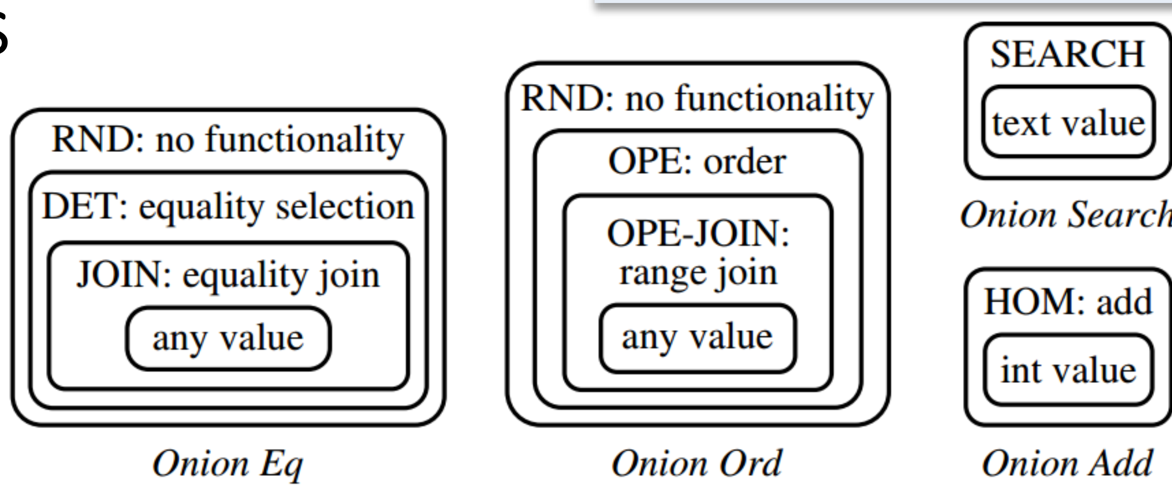
Encrypted Databases

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

SQL-Proxy

Encrypts and decrypts,

RDBMS



Relational Cloud

DBaaS Architecture:

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

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

Selected Research Challanges

Encrypted Databases

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

SQL-Proxy

Encrypts and decrypts,

RDBMS

Relational Cloud

DBaaS Architecture:

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

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

- Early approach
- Not adopted in practice, yet

Dream solution:

Full Homomorphic Encryption

Research Challanges

Transactions and Scalable Consistency

	Consistency	Transactional Unit	Commit Latency	Data Loss?
Dynamo	Eventual	None	1 RT	-
Yahoo PNuts	Timeline per key	Single Key	1 RT	possible
COPS	Causality	Multi-Record	1 RT	possible
MySQL (async)	Serializable	Static Partition	1 RT	possible
Megastore	Serializable	Static Partition	2 RT	-
Spanner/F1	Snapshot Isolation	Partition	2 RT	-
MDCC	Read-Commited	Multi-Record	1 RT	-

Research Challanges

Transactions and Scalable Consistency

	Consisten		mit	Data
Dynamo	Eventual			
Yahoo PNuts	Timeline pe			
COPS	Causality			
MySQL (async)	Serializable			
Megastore	Serializable			
Spanner/F1	Snapshot Isolation	Partition	2 RT	-
MDCC	Read-Commited	Multi-Record	1 RT	-

Google's F1

Idea:

- Consistent multi-data center replication with SQL and ACID transaction

Implementation:

- Hierarchical schema (Protobuf)
- Spanner + Indexing + Lazy Schema Updates
- Optimistic and Pessimistic Transactions

Shute, Jeff, et al. "F1: A distributed SQL database that scales." Proceedings of the VLDB 2013.

Research Challanges

Transactions and Scalable Consistency

	Consisten	mit	Data
Dynamo	Eventual		
Yahoo PNuts	Timeline pe		
COPS	Causality		
MySQL (async)	Serializable		

Google's F1

Idea:

- Consistent multi-data center replication with SQL and ACID transaction

Implementation:

- Hierarchical schema (Protobuf)
- Spanner + Indexing + Lazy Schema Updates
- Optimistic and Pessimistic Transactions

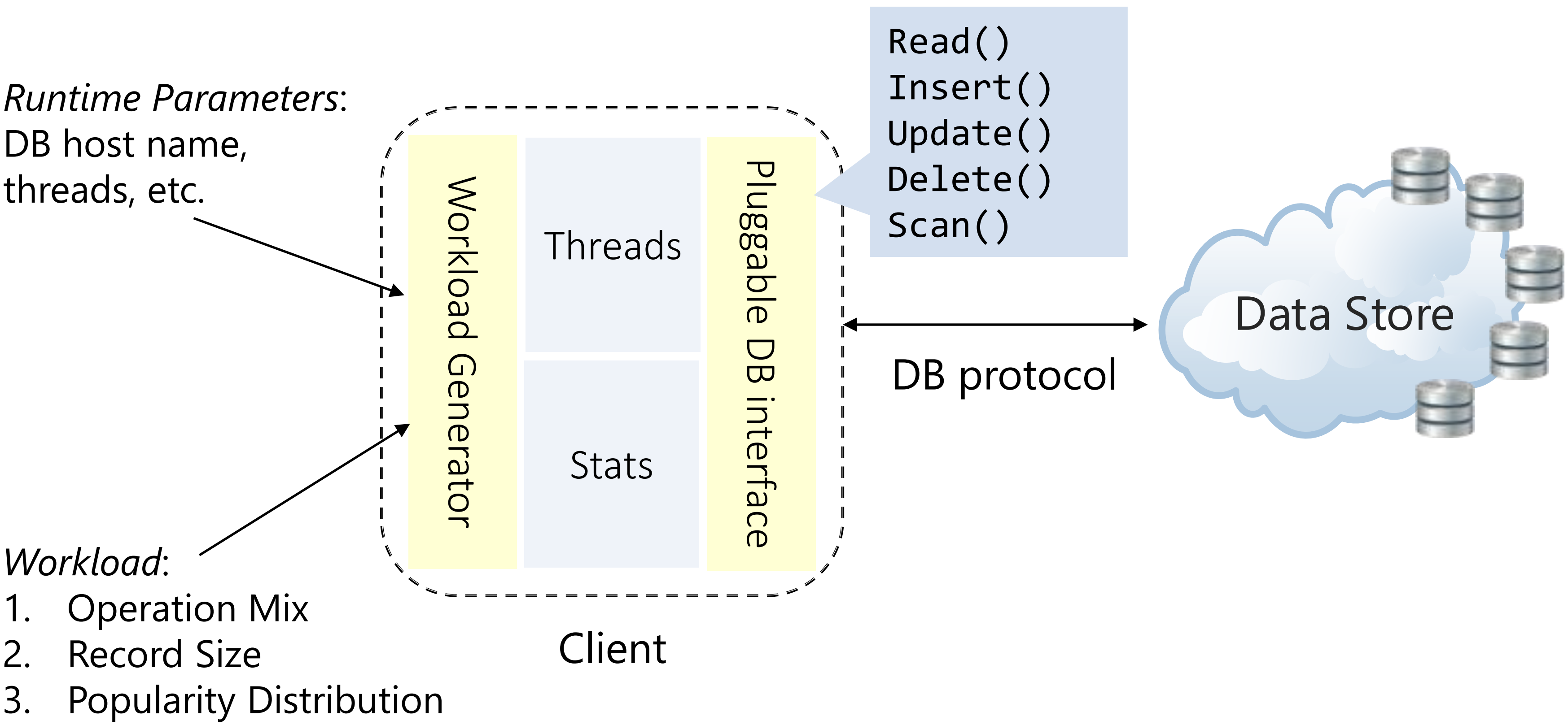
Shute, Jeff, et al. "F1: A distributed SQL database that scales." Proceedings of the VLDB 2013.

Currently very few NoSQL DBs implement consistent Multi-DC replication

Research Challanges

NoSQL Benchmarking

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



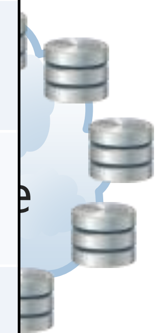
Research Challanges

NoSQL Benchmarking

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

Read()

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



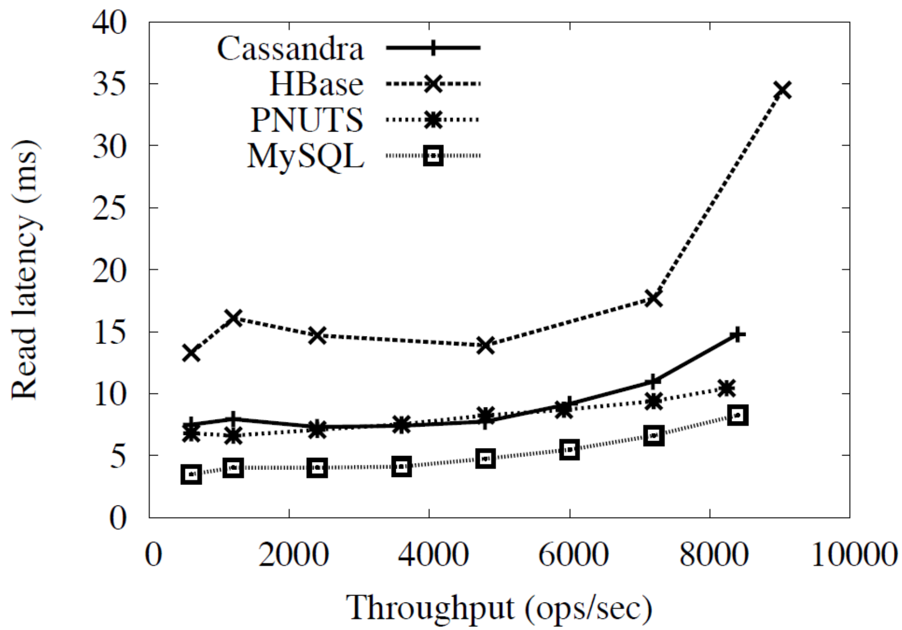
3. Popularity Distribution

Research Challanges

NoSQL Benchmarking

- Example Result

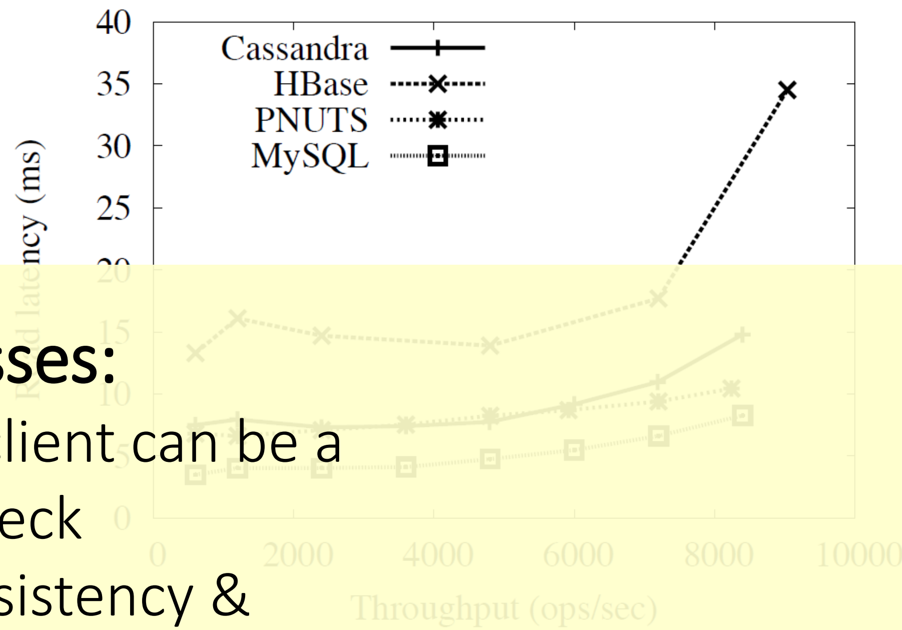
 (Read Heavy):



Research Challanges
NoSQL Benchmarking

- **Example Result**

(Read Heavy):



Weaknesses:

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

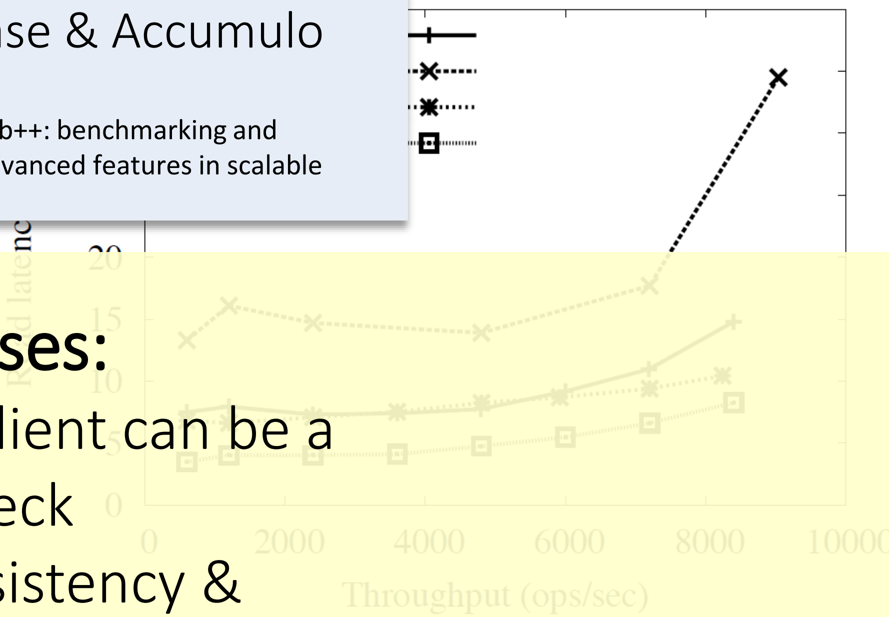
Research Challanges

NoSQL Benchmarking

YCSB++

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

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



Weaknesses:

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

Research Challanges

NoSQL Benchmarking

YCSB++

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

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

YCSB+T

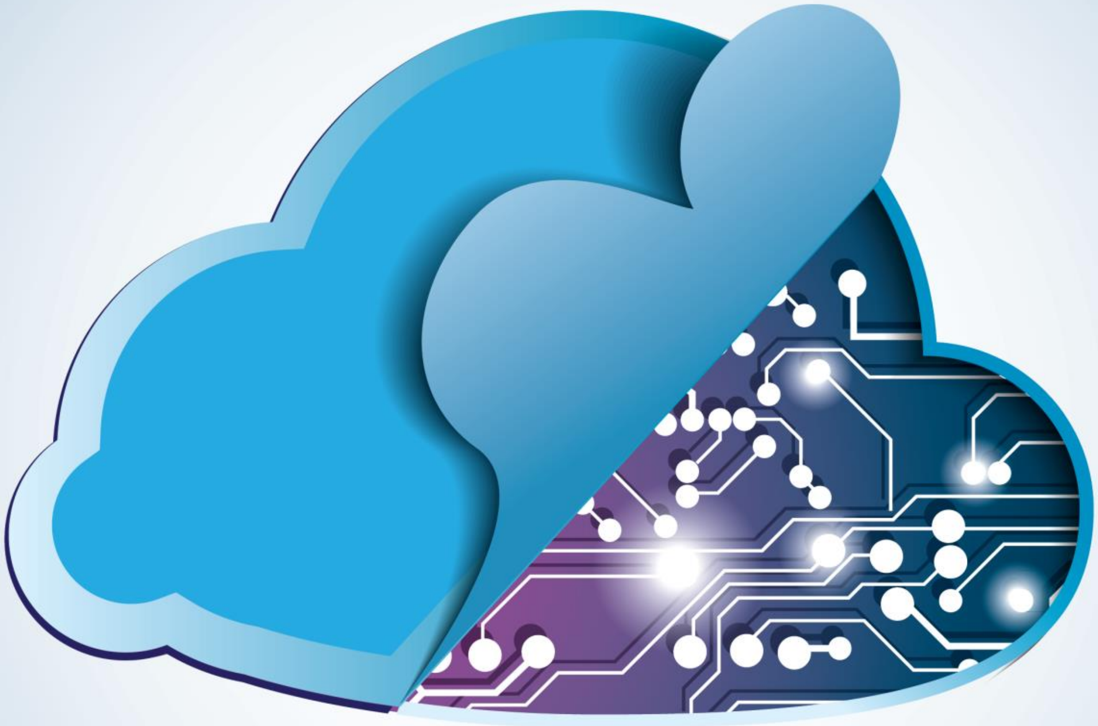
- **New workload:** Transactional Bank Account
- Simple anomaly detection for Lost Updates
- No comparison of systems

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

Weaknesses:

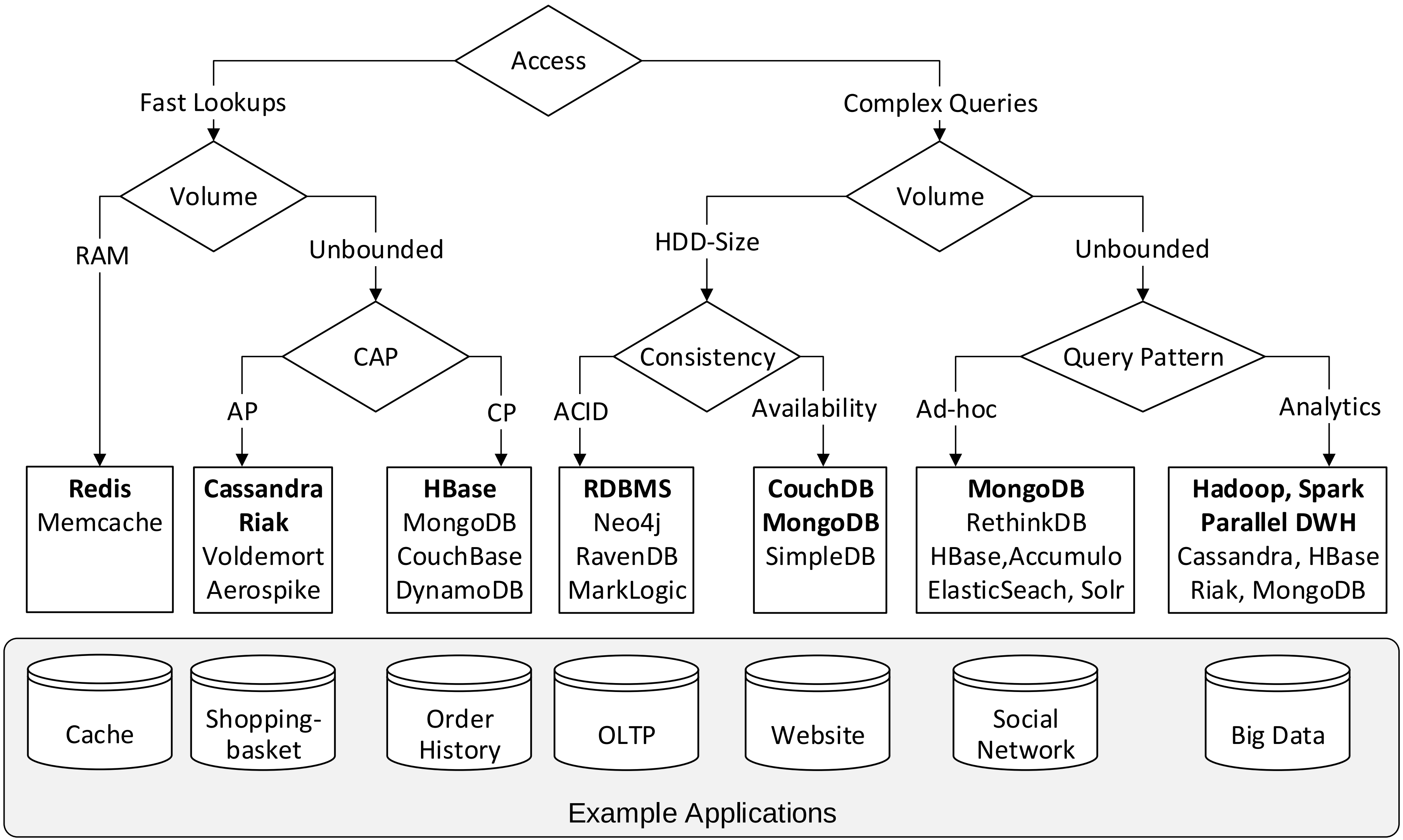
- Single client can be a bottleneck
- No consistency & availability measurement
- No Transaction Support

No specific application
→ CloudStone, CARE, TPC extensions?

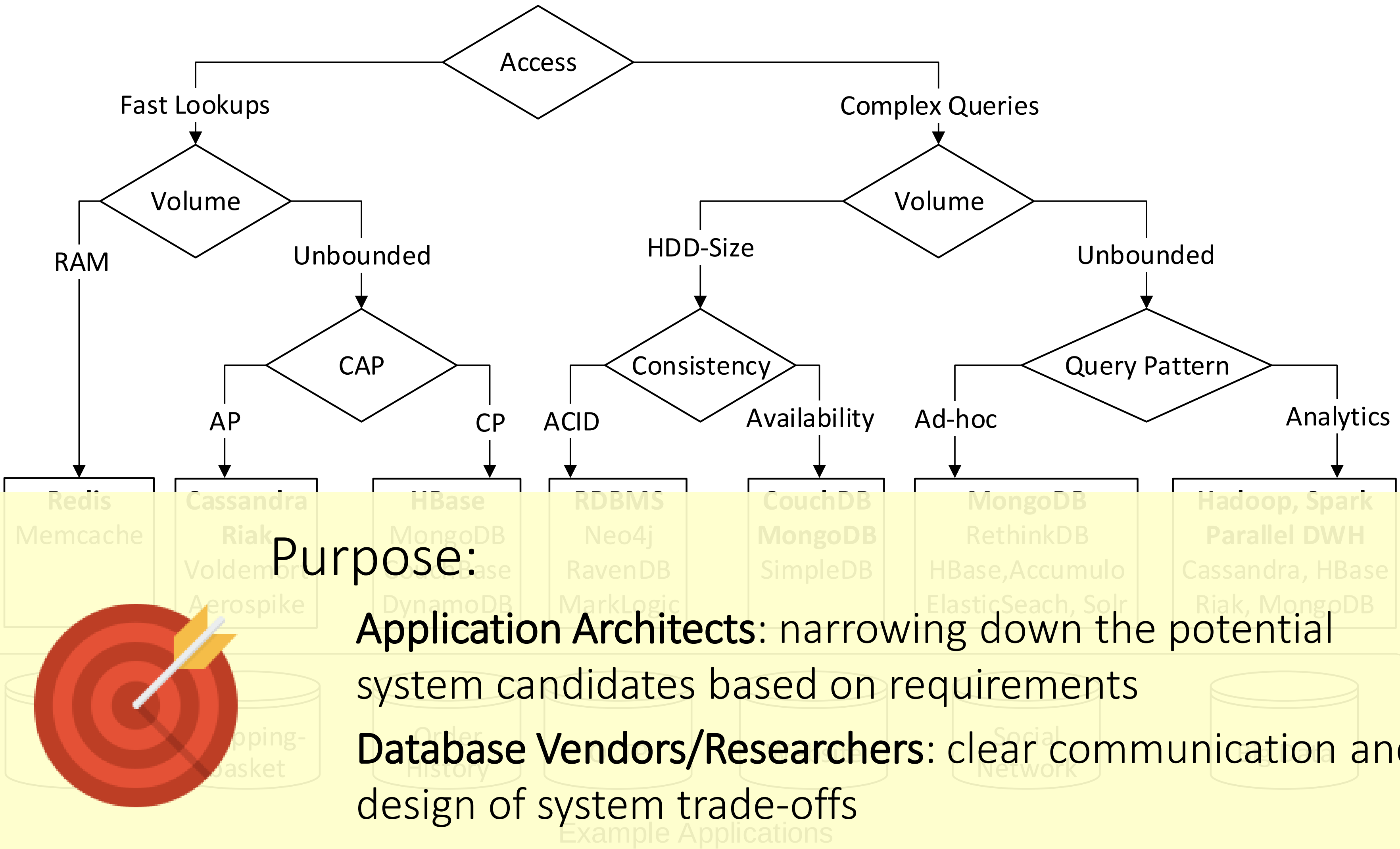


How can the choices for an appro-priate system be narrowed down?

NoSQL Decision Tree



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

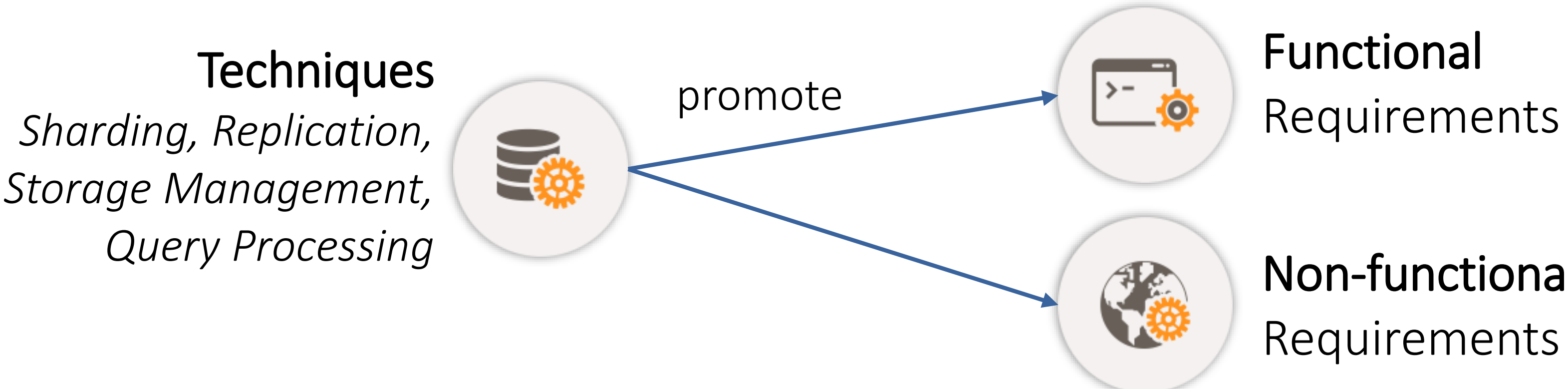
- System makes **suggestions** based on data from *practitioners*, *vendors* and *automated benchmarks*:

★ 4/5 4/5 3/5 redis

★ 4/5 5/5 5/5 mongoDB

Summary

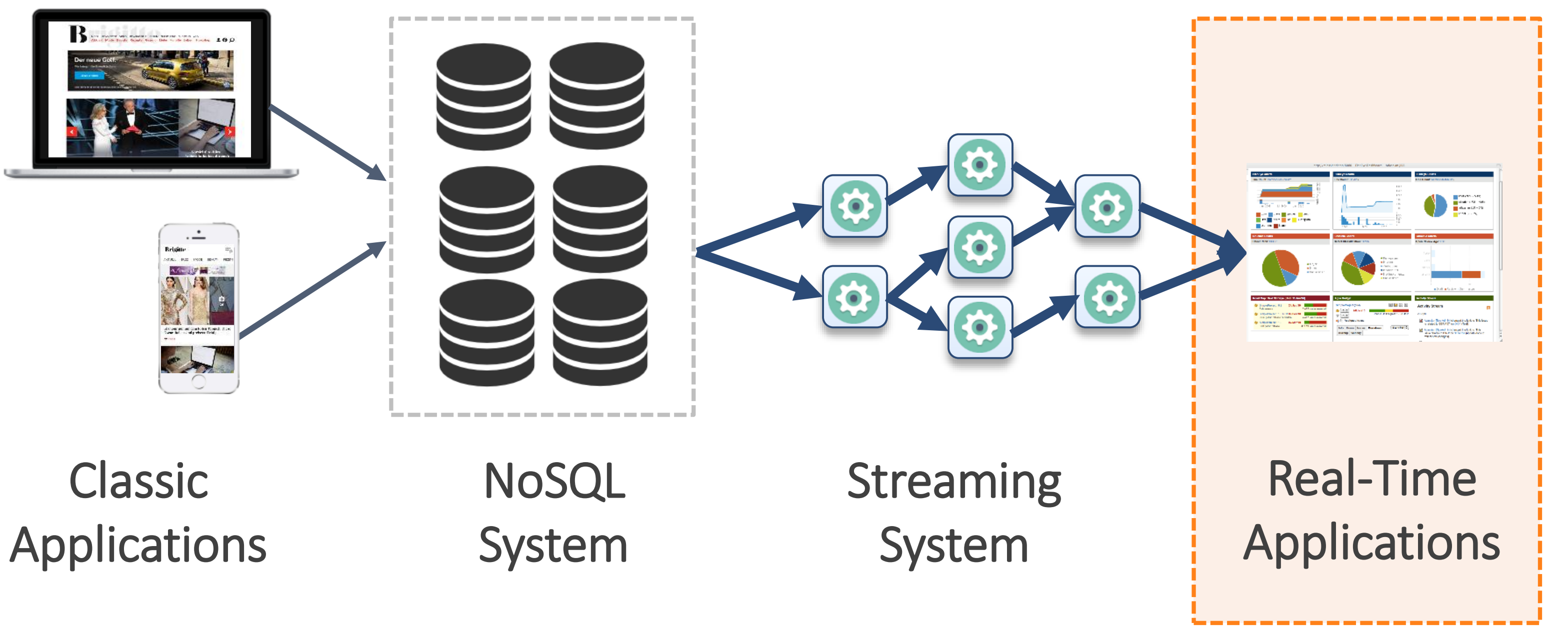
- High-Level NoSQL Categories:
  - Key-Value, Wide-Column, Docuement, Graph
  - Two out of {Consistent, Available, Partition Tolerant}
- The **NoSQL Toolbox**: systems use similar techniques that promote certain capabilities



- **Decision Tree**

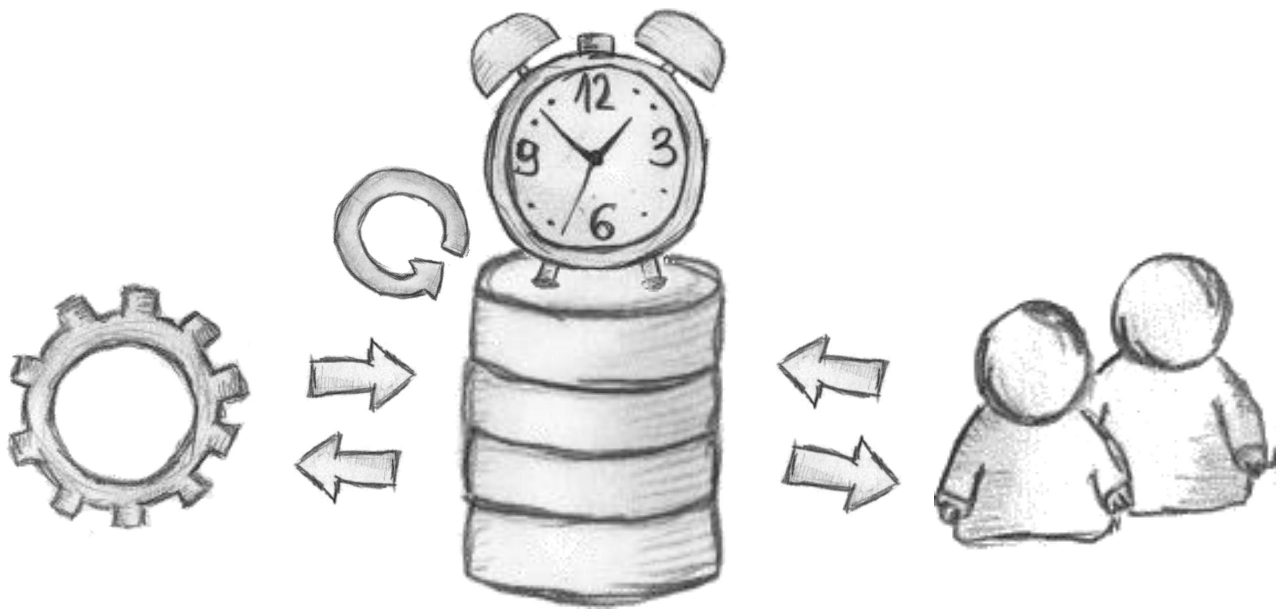
Summary

- Current NoSQL systems very good at scaling:
  - Data storage
  - Simple retrieval
- But how to handle real-time queries?



Real-Time Data Management

in Research and Industry



Wolfram Wingerath
wingerath@informatik.uni-hamburg.de
March 7th, 2017, Stuttgart

About me

Wolfram Wingerath

- *PhD student at the University of Hamburg, Information Systems group*
- Researching distributed data management:

NoSQL database systems

Scalable stream processing

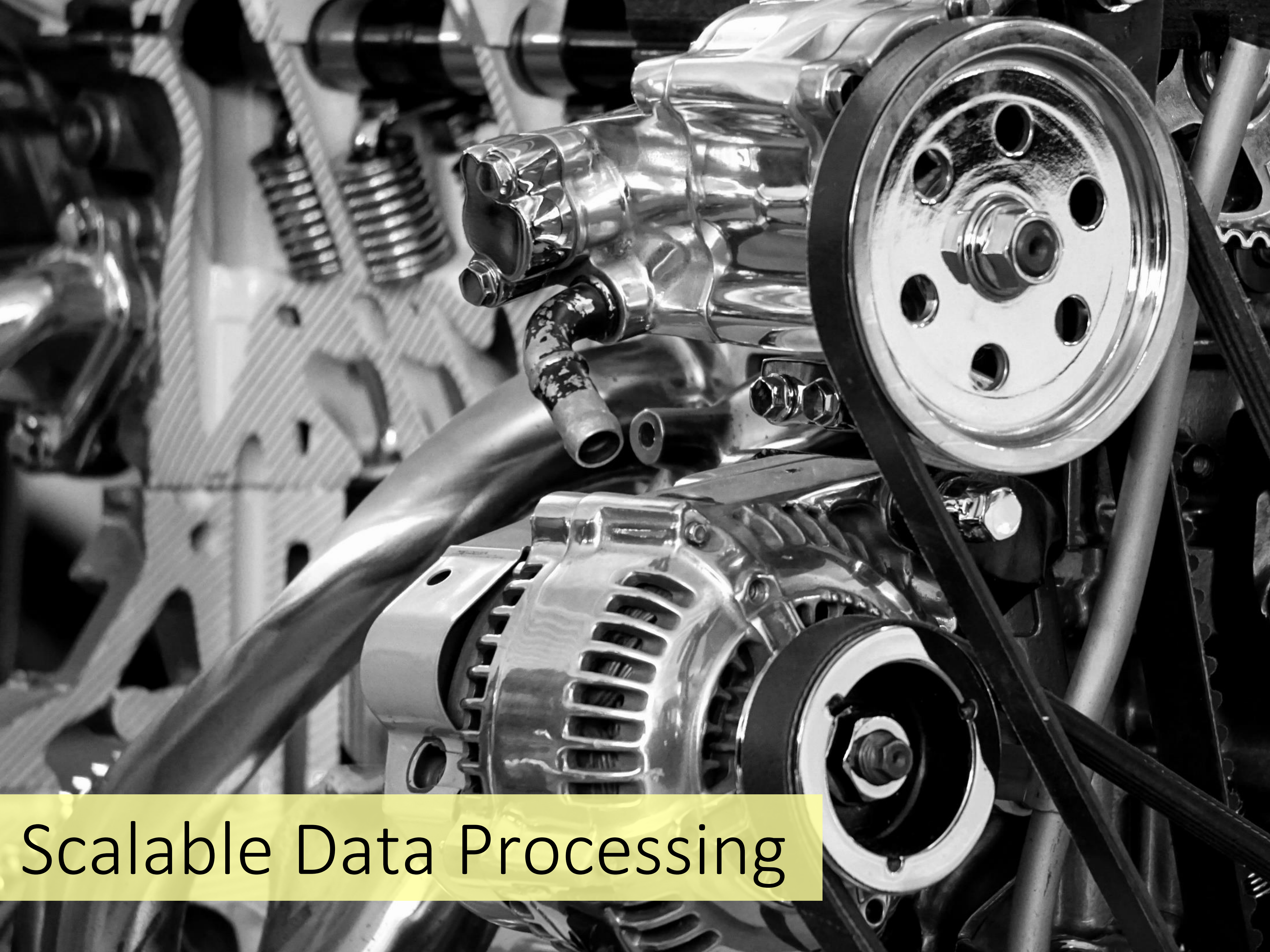
Scalable real-time queries

NoSQL benchmarking

Outline

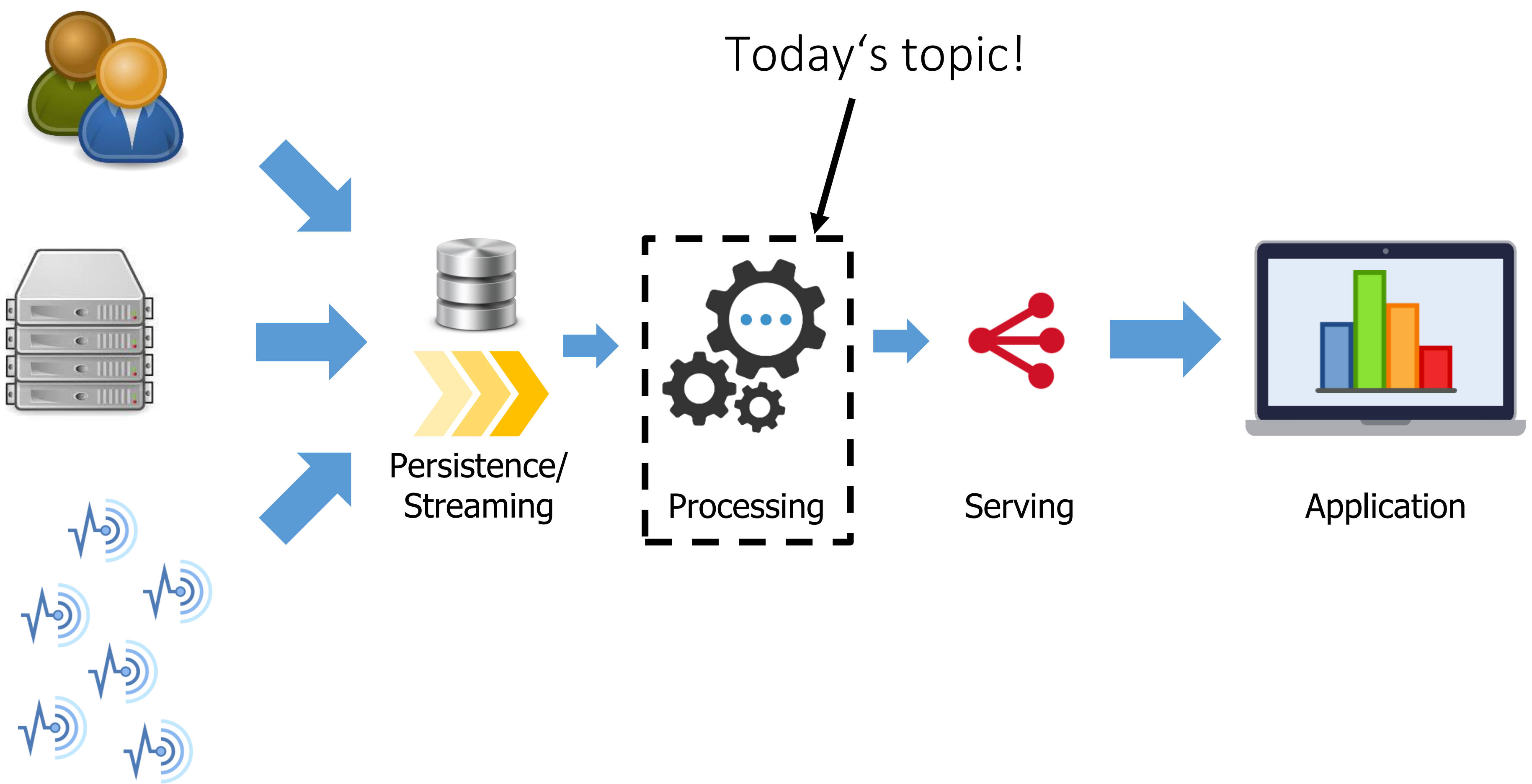
- Scalable Data Processing: Big Data in Motion
- Stream Processors: Side-by-Side Comparison
- Real-Time Databases: Push-Based Data Access
- Current Research: Opt-In Push-Based Access

- Data Processing Pipelines
- Why Data Processing Frameworks?
- Overview: Processing Landscape
- Batch Processing
- Stream Processing
- Lambda Architecture
- Kappa Architecture
- Wrap-Up



Scalable Data Processing

A Data Processing Pipeline

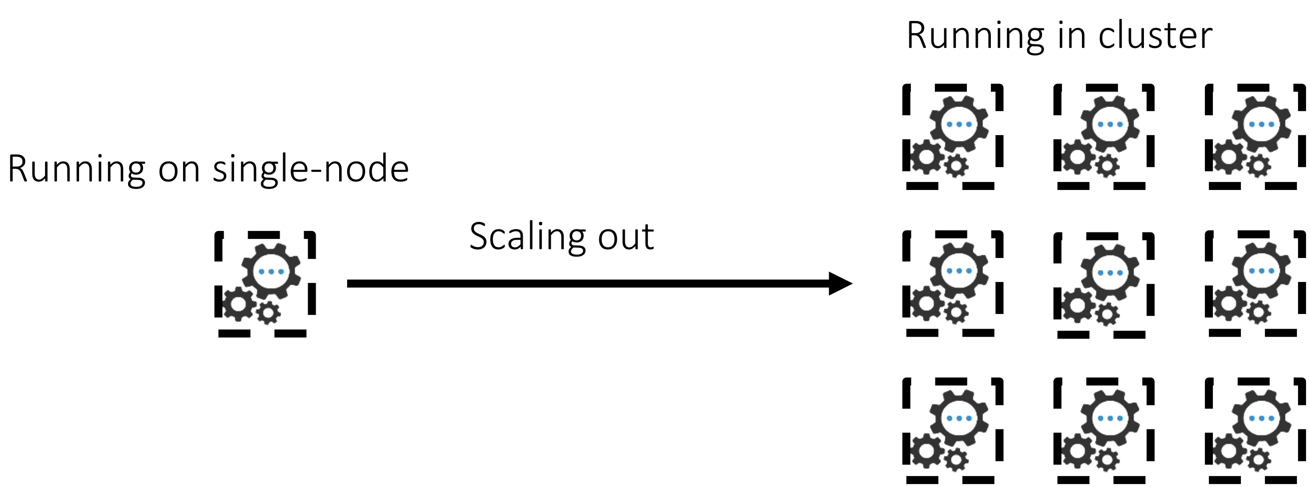


Data Processing Frameworks

Scale-Out Made Feasible

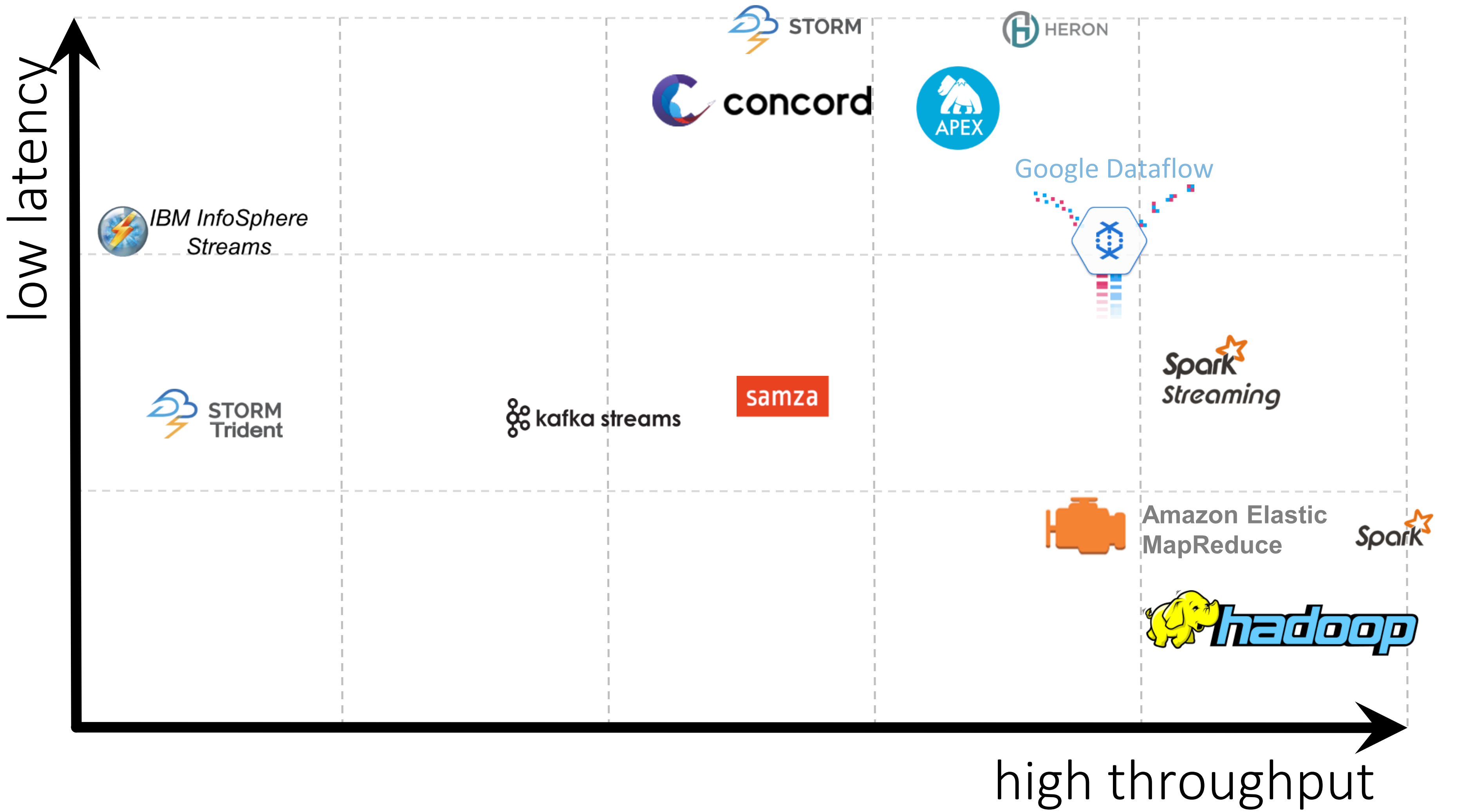
Data processing frameworks **hide some complexities of scaling**, e.g.:

- **Deployment**: code distribution, starting/stopping work
- **Monitoring**: health checks, application stats
- **Scheduling**: assigning work to machines, rebalancing
- **Fault-tolerance**: restarting failed workers, rescheduling failed work



Big Data Processing Frameworks

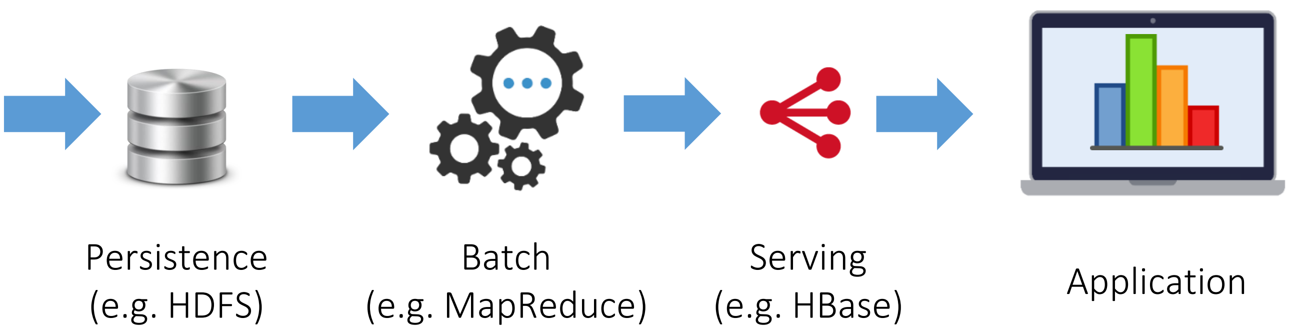
What are your options?



Batch Processing
„Volume“

- Cost-effective
- Efficient
- **Easy to reason about**: operating on complete data

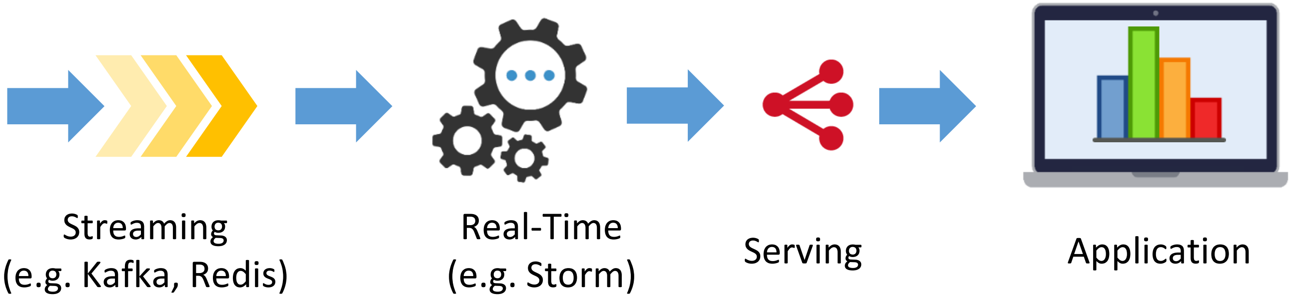
But:
- High latency: jobs periodically (e.g. during night times)



Stream Processing

„Velocity“

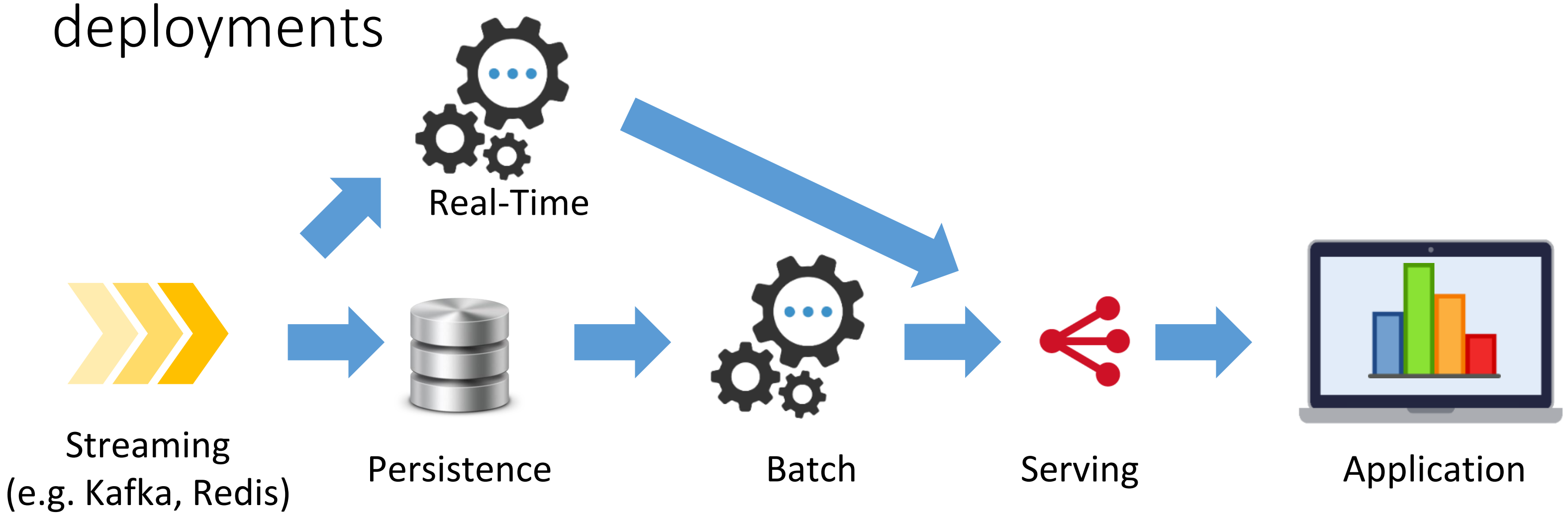
- **Low end-to-end latency**
- Challenges:
  - **Long-running jobs**: no downtime allowed
  - **Asynchronism**: data may arrive delayed or out-of-order
  - **Incomplete input**: algorithms operate on partial data
  - More: fault-tolerance, state management, guarantees, …



Lambda Architecture

$\text{Batch}(D_{old}) + \text{Stream}(D_{\Delta now}) \approx \text{Batch}(D_{all})$

- **Fast** output (real-time)
- **Data retention + reprocessing** (batch)
  → **„eventually accurate"** merged views of real-time and batch layer
  Typical setups: Hadoop + Storm (→ Summingbird), Spark, Flink
- High complexity: synchronizing 2 code bases, managing 2 deployments



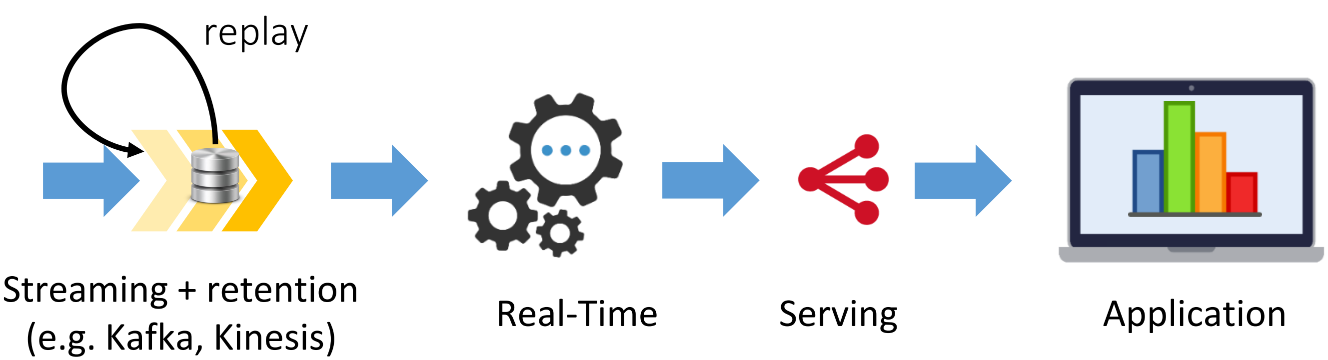
Nathan Marz, *How to beat the CAP theorem* (2011)
http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html

Kappa Architecture

$\text{Stream}(D_{all}) = \text{Batch}(D_{all})$

Simpler than Lambda Architecture

- **Data retention** for relevant portion of history
- Reasons to forgo Kappa:
  - **Legacy batch system** that is not easily migrated
  - **Special tools** only available for a particular batch processor
  - **Purely incremental** algorithms



Jay Kreps, *Questioning the Lambda Architecture* (2014)
https://www.oreilly.com/ideas/questioning-the-lambda-architecture

Wrap-up: Data Processing

- **Processing frameworks abstract from scaling issues**
- Two paradigms:
  - **Batch processing**:
    - easy to reason about
    - extremely efficient
    - Huge input-output latency
  - **Stream processing**:
    - Quick results
    - purely incremental
    - potentially complex to handle
- **Lambda Architecture**: batch + stream processing
- **Kappa Architecture**: stream-only processing

Outline

- Scalable Data Processing: Big Data in Motion
- Stream Processors: Side-by-Side Comparison
- Real-Time Databases: Push-Based Data Access
- Current Research: Opt-In Push-Based Access

- Processing Models: Stream ↔ Batch
- Stream Processing Frameworks:
  - Storm
  - Trident
  - Samza
  - Flink
  - Other Systems
- Side-By-Side Comparison
- Discussion

Stream Processors

Processing Models

Batch vs. Micro-Batch vs. Stream

stream	micro-batch		batch
Flink			hadoop
STORM	STORM Trident	Spark Streaming	Spark
samza			Amazon Elastic MapReduce

low latency ⟷ high throughput

Storm

Overview:

- **„Hadoop of real-time"**: abstract programming model (cf. MapReduce)
- **First** production-ready, well-adopted stream processing framework
- **Compatible**: native Java API, Thrift-compatible, distributed RPC
- **Low-level** interface: no primitives for joins or aggregations
- **Native stream processor**: end-to-end latency < 50 ms feasible
- **Many big users**: Twitter, Yahoo!, Spotify, Baidu, Alibaba, ...

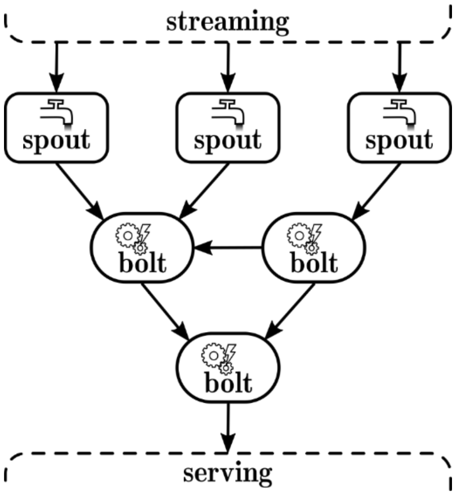
History:

- 2010: start of development at BackType (acquired by twitter)
- 2011: open-sourced
- 2014: Apache top-level project

Dataflow

Directed Acyclic Graphs (DAG):

- **Spouts**: pull data into the topology
- **Bolts**: do the processing, emit data
- Asynchronous
- Lineage can be tracked for each tuple
  → At-least-once delivery roughly
  doubles messaging overhead



Parallelism

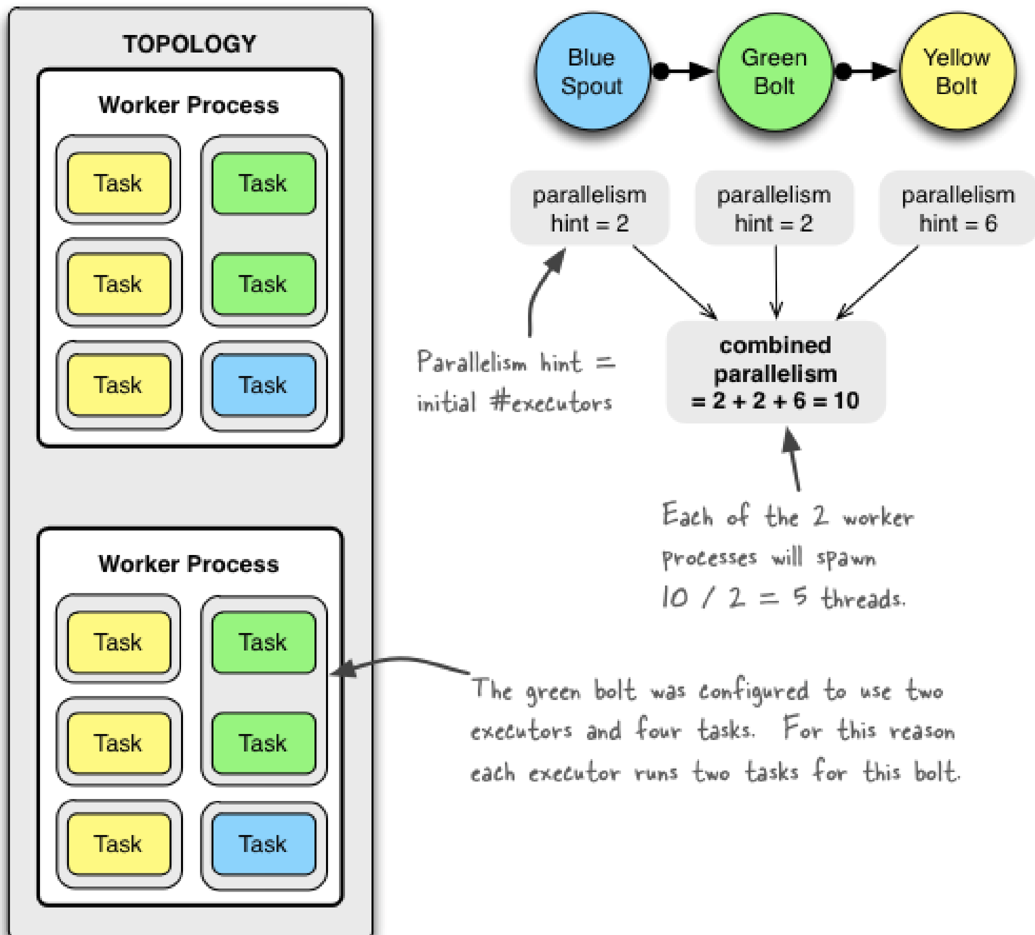
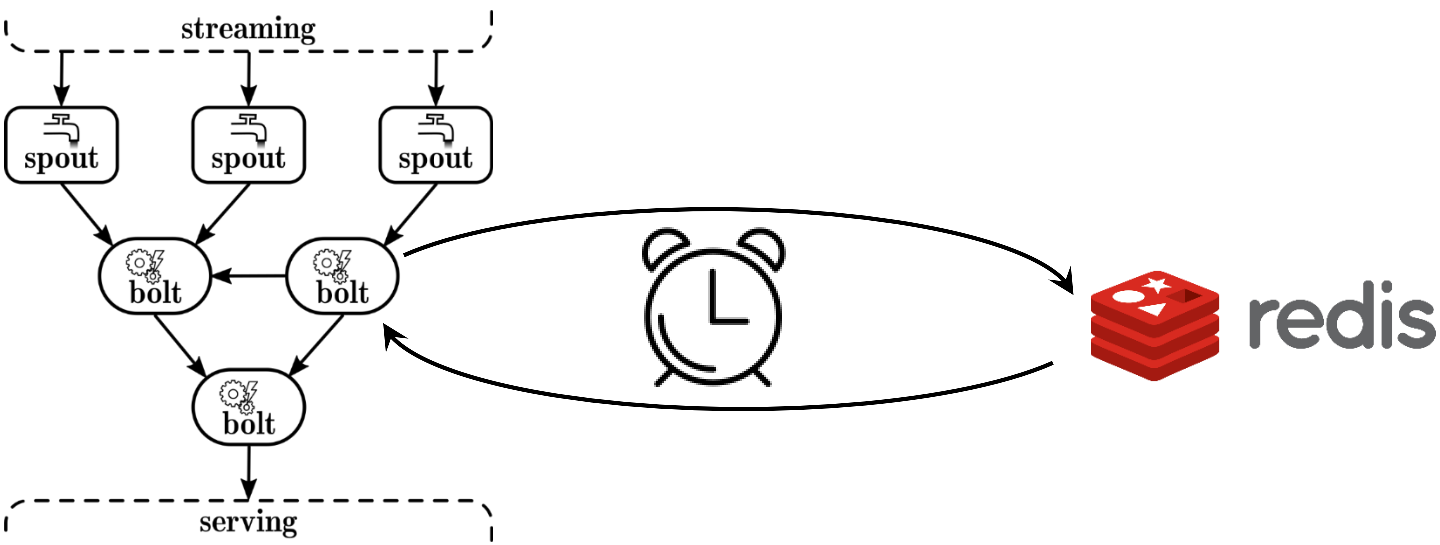


Illustration taken from:
http://storm.apache.org/releases/1.0.1/Understanding-the-parallelism-of-a-Storm-topology.html (2017-02-19)

State Management
Recover State on Failure

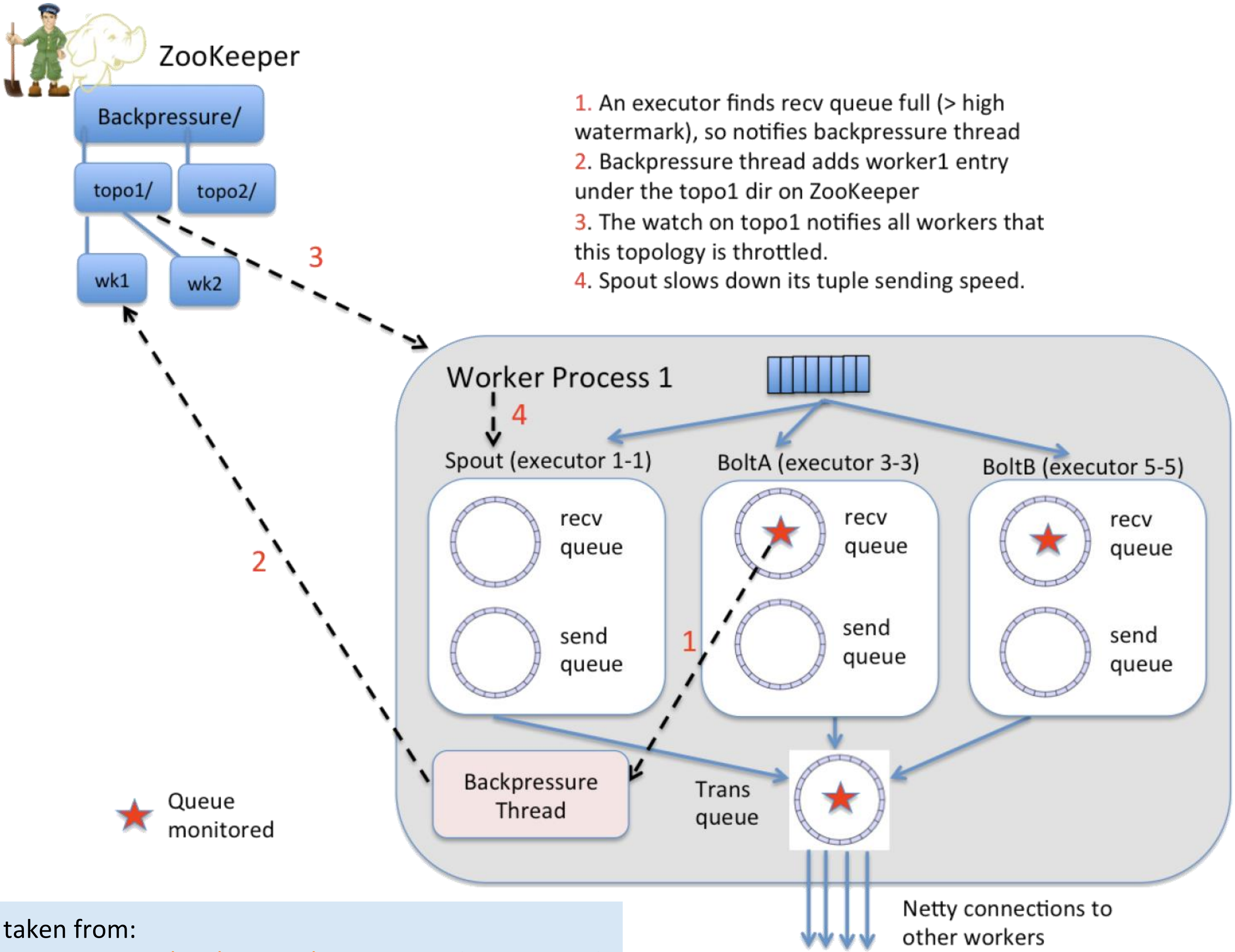
- **In-memory or Redis**-backed reliable state
- *Synchronous state communication* on the critical path

→ infeasible for large state



Back Pressure

Flow Control Through Watermarks



1. An executor finds recv queue full (> high watermark), so notifies backpressure thread
2. Backpressure thread adds worker1 entry under the topo1 dir on ZooKeeper
3. The watch on topo1 notifies all workers that this topology is throttled.
4. Spout slows down its tuple sending speed.

Illustration taken from: https://issues.apache.org/jira/browse/STORM-886 (2017-02-21)

Back Pressure
Throttling Ingestion on Overload

1. too many tuples → 2. tuples time out and fail → 3. tuples get replayed → 1. too many tuples

!

Approach: monitoring bolts' inbound buffer

1. Exceeding **high watermark** → throttle!
2. Falling below **low watermark** → full power!

Trident

Stateful Stream Joining on Storm

STORM Trident

Overview:

- Abstraction layer on top of Storm
- Released in 2012 (Storm 0.8.0)
- **Micro-batching**
- **New features**:
  - Stateful exactly-once processing
  - High-level API: aggregations & joins
  - Strong ordering

Trident

Exactly-Once Delivery Configs

Can *block* the topology
when failed batch cannot be replayed

Spout \ State	Non-transactional	Transactional	Opaque transactional
Non-transactional	No	No	No
Transactional	No	Yes	Yes
Opaque transactional	No	No	Yes

Does not scale:

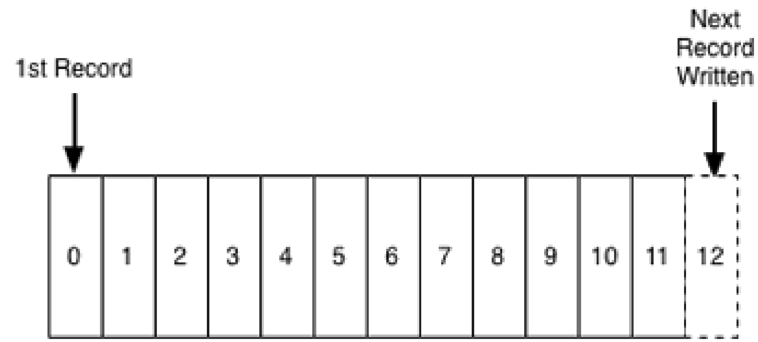
- Requires before- *and* after-images
- Batches are written in order

Illustration taken from:
http://storm.apache.org/releases/1.0.2/Trident-state.html (2017-02-26)

Samza

Overview:

- Co-developed with **Kafka**
  → **Kappa Architecture**
- **Simple**: only single-step jobs
- Local state
- Native stream processor: low latency
- **Users**: LinkedIn, Uber, Netflix, TripAdvisor, Optimizely, ...



History:

- Developed at **LinkedIn**
- 2013: open-source (Apache Incubator)
- 2015: Apache top-level project

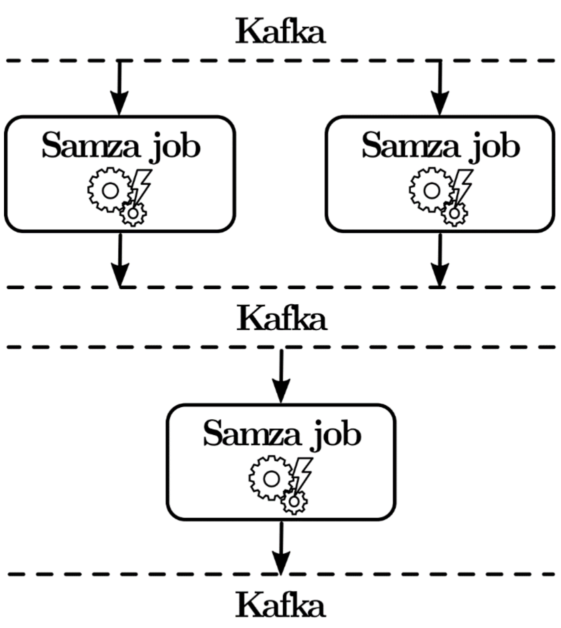
Illustration taken from: Jay Kreps, *Questioning the Lambda Architecture* (2014) https://www.oreilly.com/ideas/questioning-the-lambda-architecture (2017-03-02)

Dataflow
Simple By Design

samza

- **Job**: a single processing step (≈ Storm bolt)
  → Robust
  → But: complex applications require several jobs
- **Task**: a job instance (determines job parallelism)
- **Message**: a single data item

- **Output is always persisted** in Kafka
  → Jobs can easily share data
  → Buffering (no back pressure!)
  → But: Increased latency
- **Ordering** within partitions
- Task = Kafka partitions: not-elastic on purpose



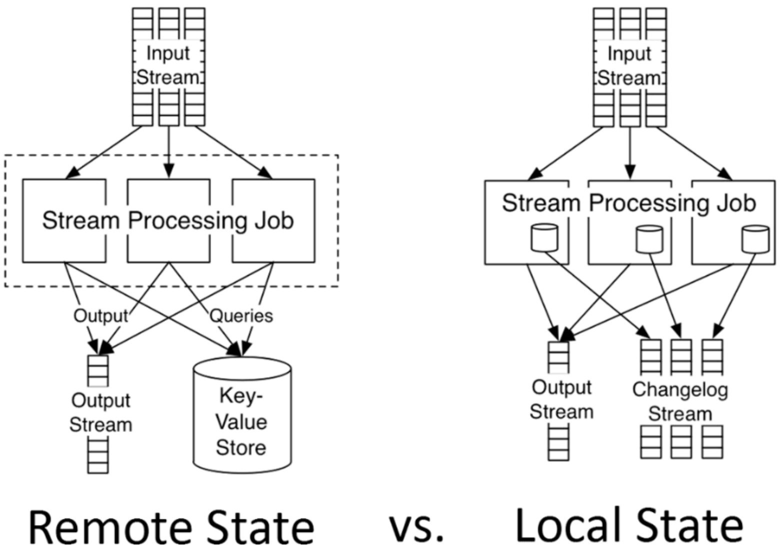
Martin Kleppmann, *Turning the database inside-out with Apache Samza* (2015)
https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/ (2017-02-23)

Samza

Local State

Advantages of local state:

- **Buffering**
  → No back pressure
  → At-least-once delivery
  → Straightforward recovery
  (see next slide)
- **Fast lookups**



Illustrations taken from: Jay Kreps, *Why local state is a fundamental primitive in stream processing* (2014) https://www.oreilly.com/ideas/why-local-state-is-a-fundamental-primitive-in-stream-processing (2017-02-26)

Dataflow

Example: Enriching a Clickstream

Example: the *enriched clickstream* is available to every team within the organization

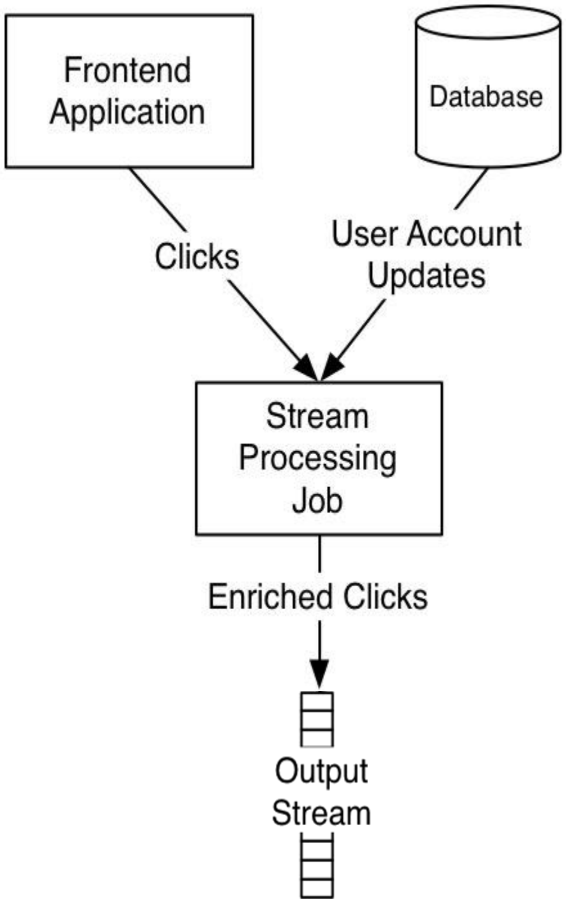


Illustration taken from: Jay Kreps, *Why local state is a fundamental primitive in stream processing* (2014) https://www.oreilly.com/ideas/why-local-state-is-a-fundamental-primitive-in-stream-processing (2017-02-26)

State Management

Straightforward Recovery

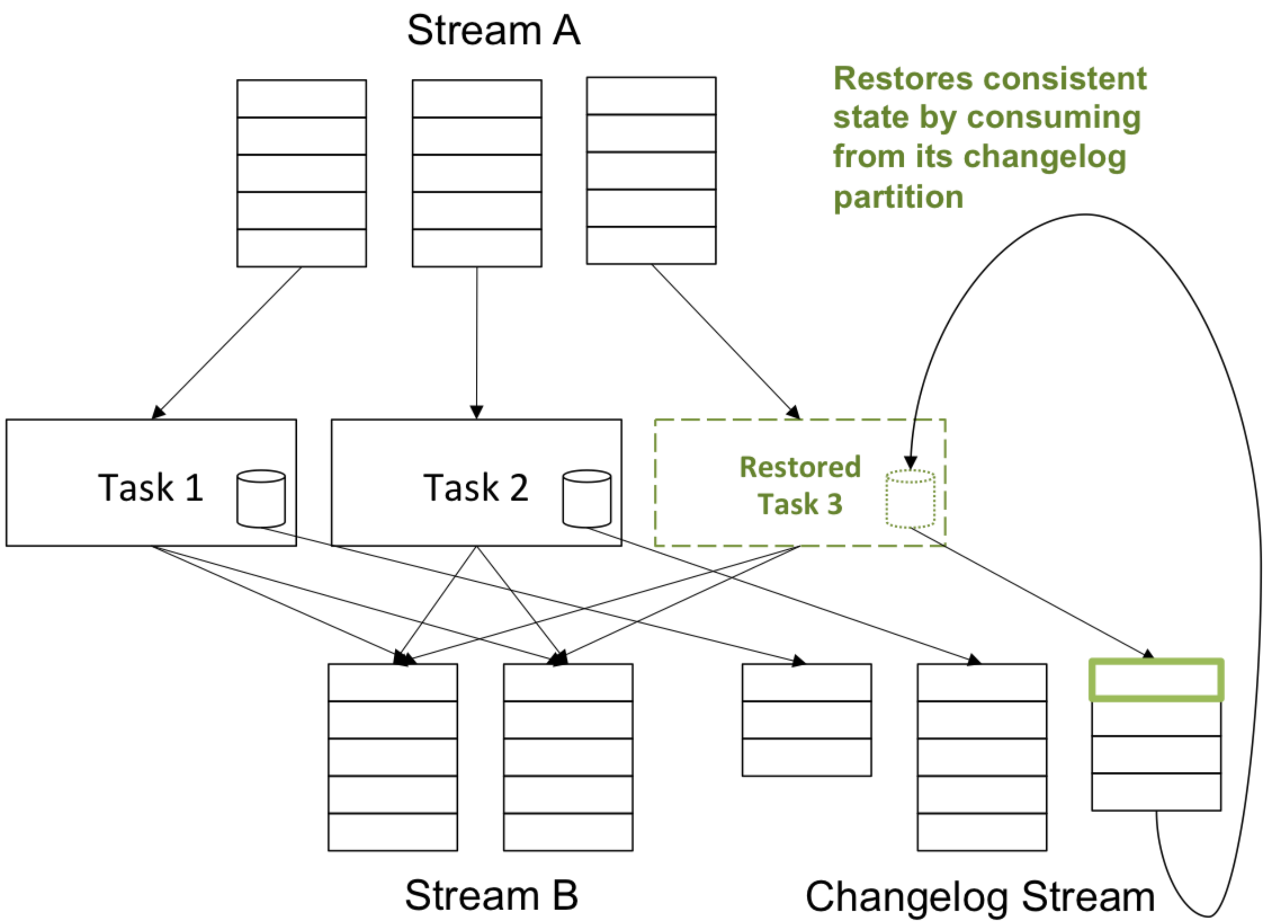


Illustration taken from: Navina Ramesh, *Apache Samza, LinkedIn's Framework for Stream Processing* (2015)
https://thenewstack.io/apache-samza-linkedins-framework-for-stream-processing (2017-02-26)

Spark

Spark

- **„MapReduce successor“**: batch, no unnecessary writes, faster scheduling
- **High-level API**: immutable collections (RDDs) as core abstraction
- **Many libraries**
  - Spark Core: batch processing
  - Spark SQL: distributed SQL
  - Spark MLlib: machine learning
  - Spark GraphX: graph processing
  - **Spark Streaming**: stream processing
- Huge community: 1000+ contributors in 2015
- **Many big users**: Amazon, eBay, Yahoo!, IBM, Baidu, ...

History:

- 2009: Spark is developed at UC Berkeley
- 2010: Spark is open-sourced
- 2014: Spark becomes Apache top-level project

Spark Streaming

Spark

- **High-level API**: DStreams as core abstraction (~Java 8 Streams)
- **Micro-Batching**: latency on the order of seconds
- **Rich feature set**: statefulness, exactly-once processing, elasticity

History:

- 2011: start of development
- 2013: Spark Streaming becomes part of Spark Core

Spark Streaming

Core Abstraction: DStream

Resilient Distributed Data set (RDD):

- **Immutable** collection
- **Deterministic** operations
- **Lineage** tracking:
  → state can be reproduced
  → periodic checkpoints to reduce recovery time

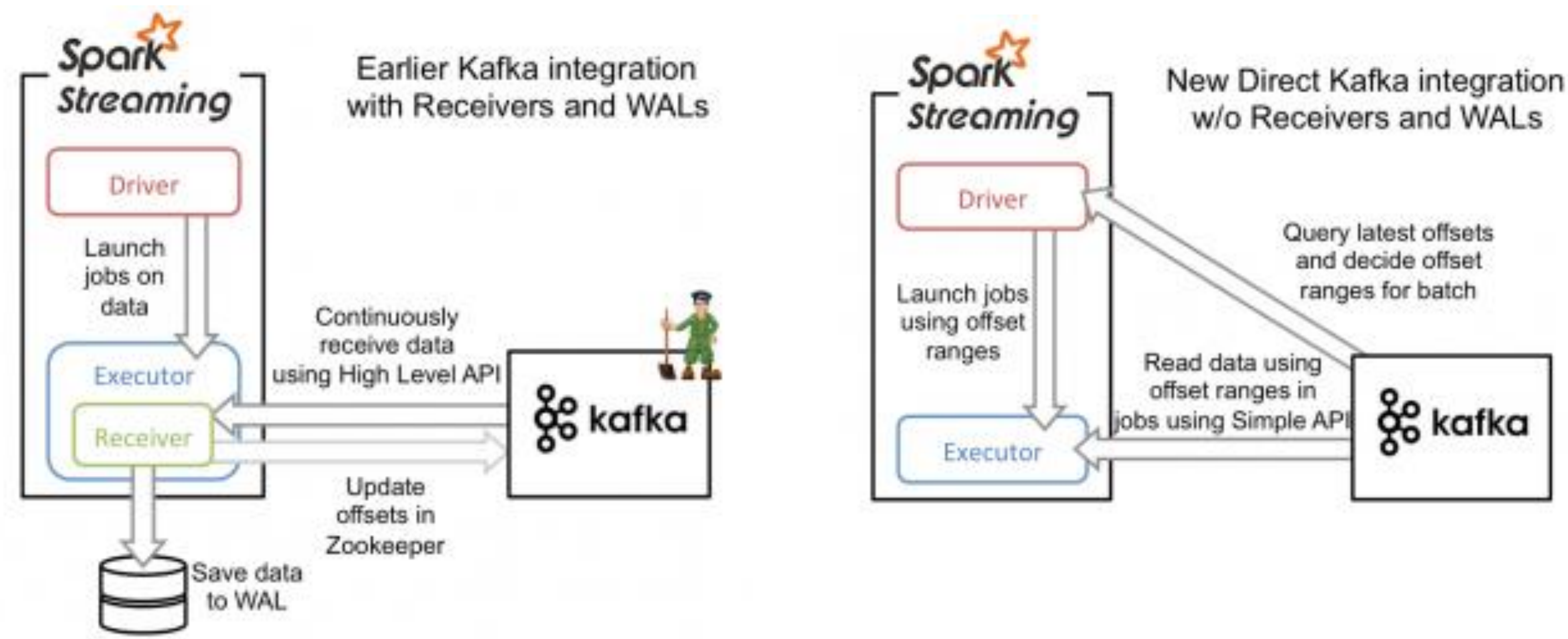
DStream: Discretized RDD

- **RDDs are processed in order**: no ordering for data within an RDD
- RDD Scheduling ~50 ms → latency <100ms infeasible

Illustration taken from:
http://spark.apache.org/docs/latest/streaming-programming-guide.html#overview (2017-02-26)

Spark Streaming

Fault-Tolerance: Receivers & WAL



Illustrations taken from:
https://databricks.com/blog/2015/03/30/improvements-to-kafka-integration-of-spark-streaming.html (2017-02-26)

Flink

Overview:

- **Native stream processor:** Latency <100ms feasible
- **Abstract API** for stream and batch processing, stateful, exactly-once delivery
- **Many libraries**:
  - Table and SQL: distributed and streaming SQL
  - CEP: complex event processing
  - Machine Learning
  - Gelly: graph processing
  - Storm Compatibility: adapter to run Storm topologies
- **Users**: Alibaba, Ericsson, Otto Group, ResearchGate, Zalando…

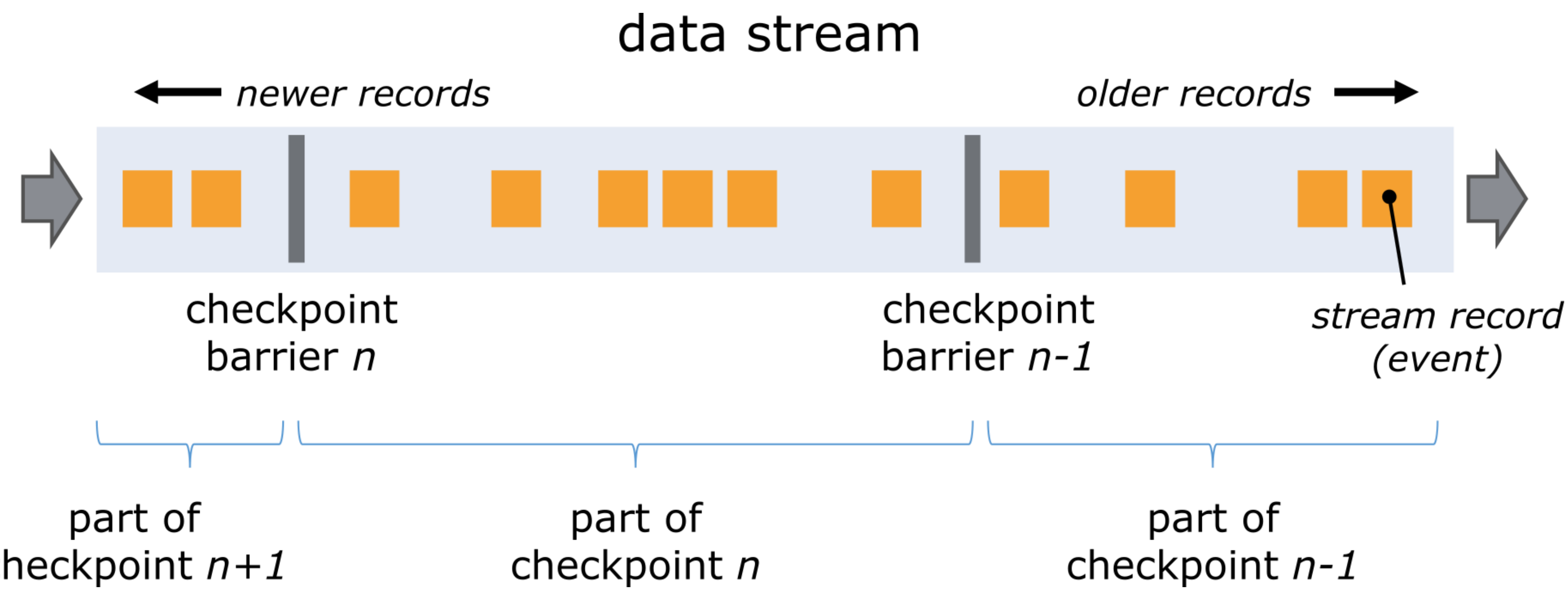
History:

- 2010: start of project **Stratosphere** at TU Berlin, HU Berlin, and HPI Potsdam
- 2014: Apache Incubator, project renamed to Flink
- 2015: Apache top-level project

Highlight: State Management

Distributed Snapshots

Flink



- **Ordering** within stream partitions
- Periodic **checkpointing**
- **Recovery** procedure:

1. *reset state* to last checkpoint
2. *replay data* from last checkpoint

Illustration taken from: https://ci.apache.org/projects/flink/flink-docs-release-1.2/internals/stream_checkpointing.html (2017-02-26)

State Management

Checkpointing (1/4)

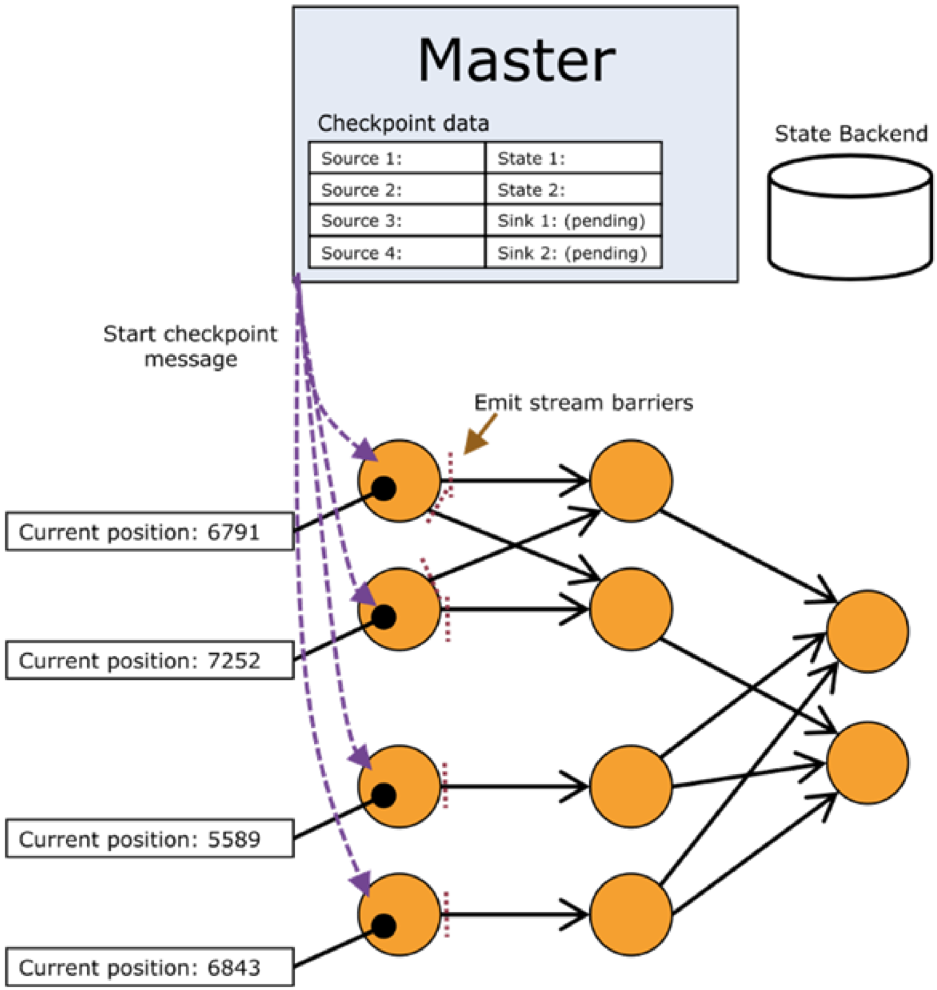


Illustration taken from: Robert Metzger, *Architecture of Flink's Streaming Runtime* (ApacheCon EU 2015) https://www.slideshare.net/robertmetzger1/architecture-of-flinks-streaming-runtime-apachecon-eu-2015 (2017-02-27)

State Management

Checkpointing (2/4)

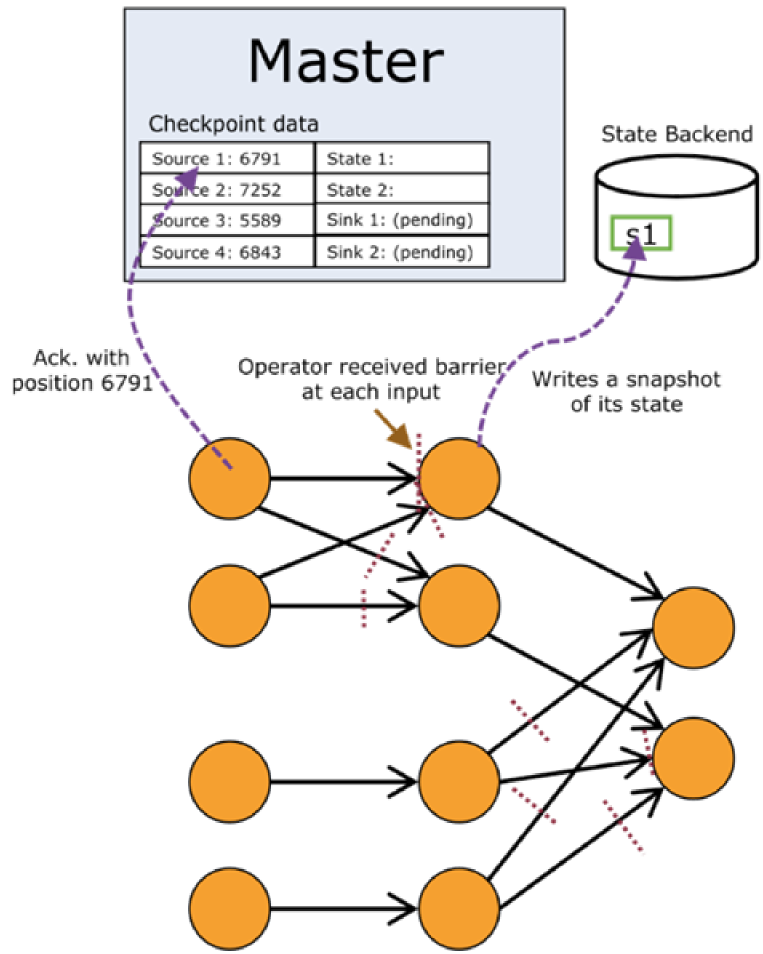


Illustration taken from: Robert Metzger, *Architecture of Flink's Streaming Runtime* (ApacheCon EU 2015) https://www.slideshare.net/robertmetzger1/architecture-of-flinks-streaming-runtime-apachecon-eu-2015 (2017-02-27)

State Management
Checkpointing (3/4)

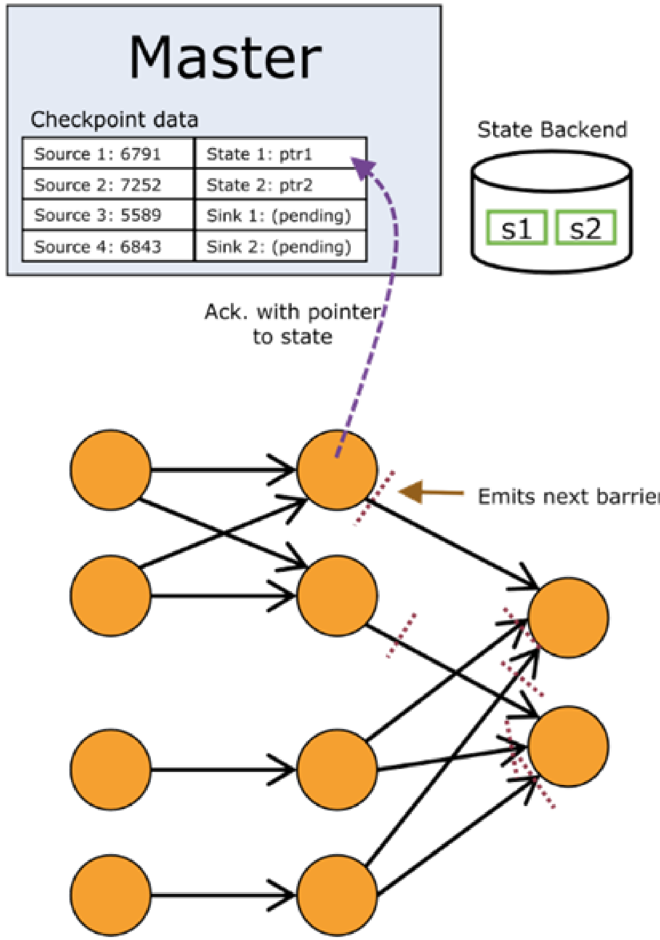


Illustration taken from: Robert Metzger, *Architecture of Flink's Streaming Runtime* (ApacheCon EU 2015) https://www.slideshare.net/robertmetzger1/architecture-of-flinks-streaming-runtime-apachecon-eu-2015 (2017-02-27)

State Management

Checkpointing (4/4)

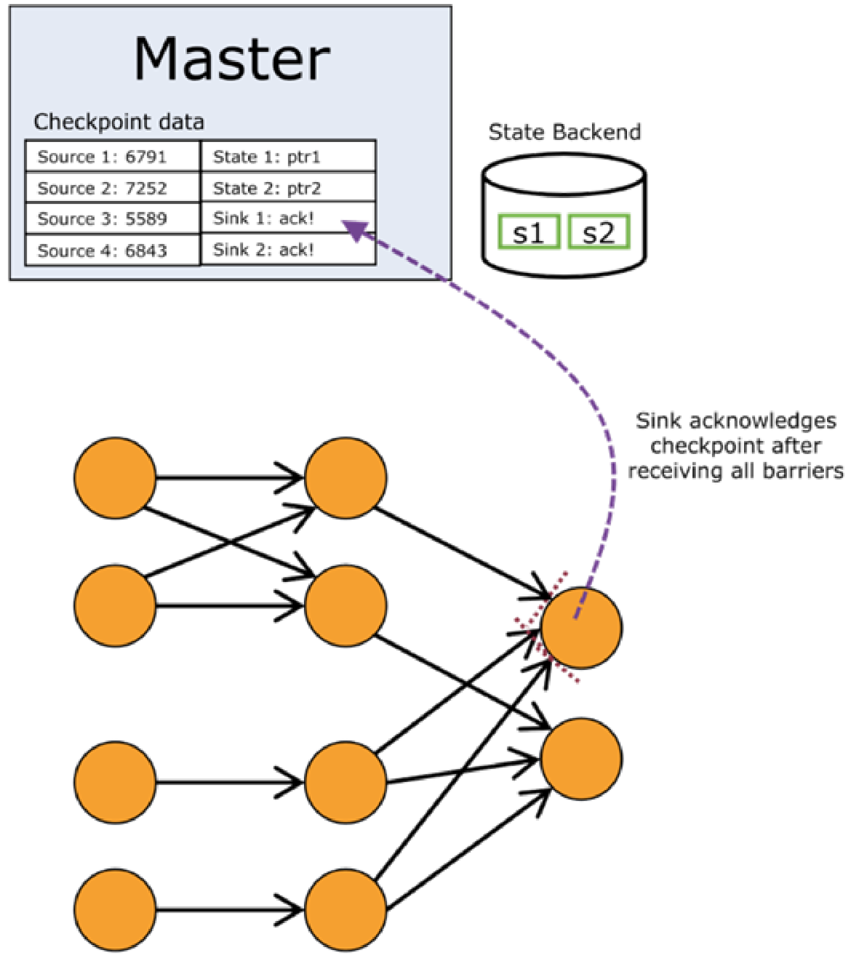


Illustration taken from: Robert Metzger, *Architecture of Flink's Streaming Runtime* (ApacheCon EU 2015)
https://www.slideshare.net/robertmetzger1/architecture-of-flinks-streaming-runtime-apachecon-eu-2015 (2017-02-27)

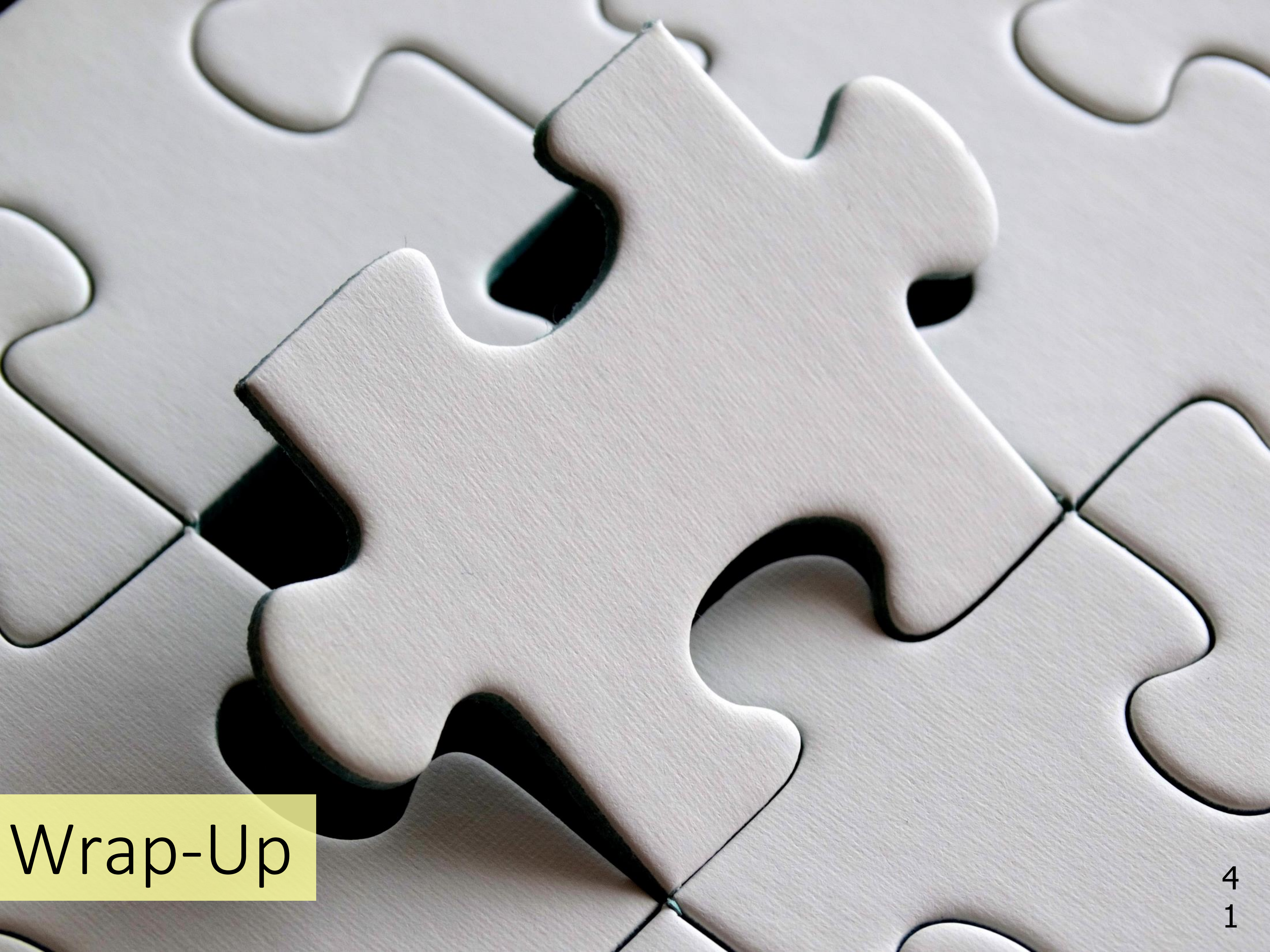
Other Systems

- **Heron**: open-source, Storm successor
- **Apex**: stream and batch process so with many libraries
  Dataflow: Fully managed cloud service for batch and stream processing, proprietary
- **Beam**: open-source runtime-agnostic API for Dataflow programming model; runs on Flink, Spark and others
- **KafkaStreams**: integrated with Kafka, open-source
- **IBM Infosphere Streams**: proprietary, managed, bundled with IDE

- **And even more**: Kinesis, Gearpump, MillWheel, Muppet, S4, Photon, ...

Direct Comparison

	Storm	Trident	Samza	Spark Streaming	Flink (streaming)
Strictest Guarantee	at-least-once	exactly-once	at-least-once	exactly-once	exactly-once
Achievable Latency	≪100 ms	<100 ms	<100 ms	<1 second	<100 ms
State Management	○ (small state)	○ (small state)	✓	✓	✓
Processing Model	one-at-a-time	micro-batch	one-at-a-time	micro-batch	one-at-a-time
Backpressure	✓	✓	not required (buffering)	✓	✓
Ordering	✗	between batches	within partitions	between batches	within partitions
Elasticity	✓	✓	✗	✓	✗



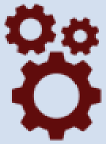
Wrap-Up

Wrap-up

- **Push-based data access**
  - Natural for many applications
  - Hard to implement on top of traditional (pull-based) databases
- **Real-time databases**
  - Natively push-based
  - Challenges: scalability, fault-tolerance, semantics, rewrite vs. upgrade, …
- **Scalable Stream Processing**
  - Stream vs. Micro-Batch (vs. Batch)
  - Lambda & Kappa Architecture
  - Vast feature space, many frameworks
- **InvaliDB**
  - A linearly scalable design for add-on push-based queries
  - Database-independent
  - Real-time updates for powerful queries: filter, sorting, joins, aggregations

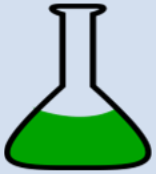
Outline

Scalable Data Processing:
Big Data in Motion



Stream Processors:
Side-by-Side Comparison

Real-Time Databases:
Push-Based Data Access



Current Research:
Opt-In Push-Based Access

- Pull-Based vs Push-Based Data Access
- DBMS vs. RT DB vs. DSMS vs. Stream Processing
- Popular Push-Based DBs:
  - Firebase
  - Meteor
  - RethinkDB
  - Parse
  - Others
- Discussion

Real-Time Databases

Traditional Databases

No Request? No Data!

Query maintenance: periodic polling

→ Inefficient

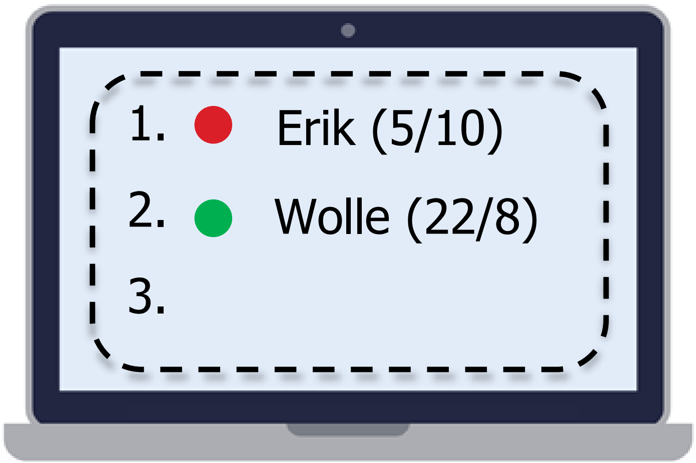
→ Slow

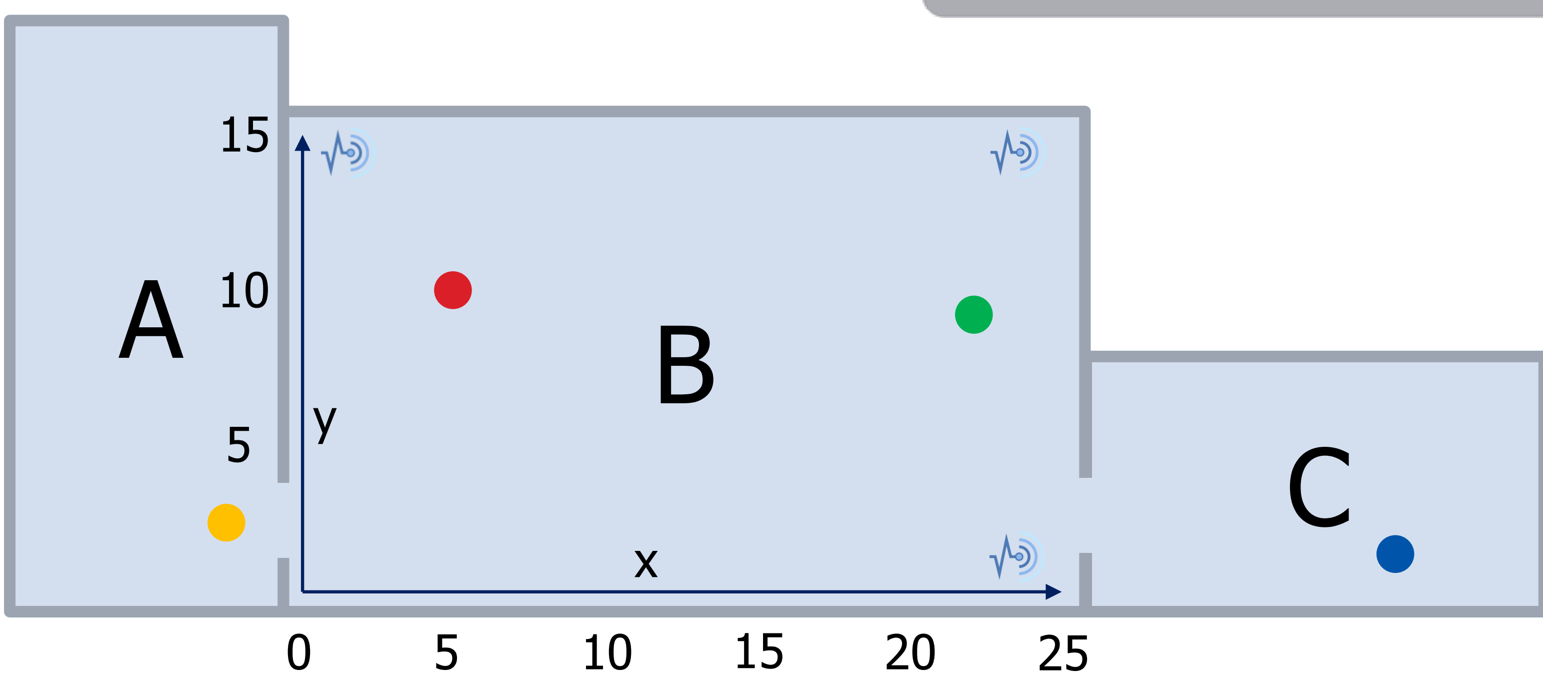
Ideal: Push-Based Data Access

Self-Maintaining Results

Find people in Room B:

```
db.User.find()
     .equal('room','B')
     .ascending('name')
     .limit(3)
     .streamResult()
```





Popular Real-Time Databases

Firebase

Firebase

Overview:

- **Real-time state synchronization** across devices
- **Simplistic data model:** nested hierarchy of lists and objects
- **Simplistic queries**: mostly navigation/filtering
- **Fully managed**, proprietary
- **App SDK** for App development, mobile-first
- **Google services integration**: analytics, hosting, authorization, ...

History:

- 2011: chat service startup Envolve is founded
  → was often used for cross-device state synchronization
  → state synchronization is separated (Firebase)
- 2012: Firebase is founded
- 2013: Firebase is acquired by Google

Firebase

Real-Time State Synchronization

Firebase

- **Tree data model**: application state ~JSON object
- **Subtree synching**: push notifications for specific keys only
  → Flat structure for fine granularity

→ Limited expressiveness!

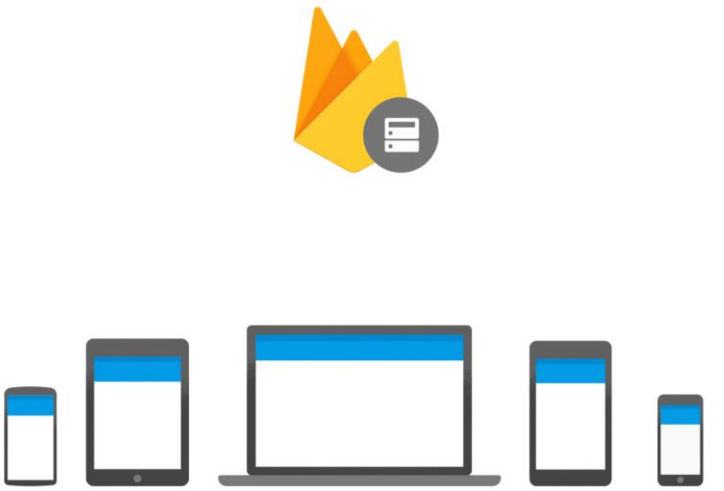


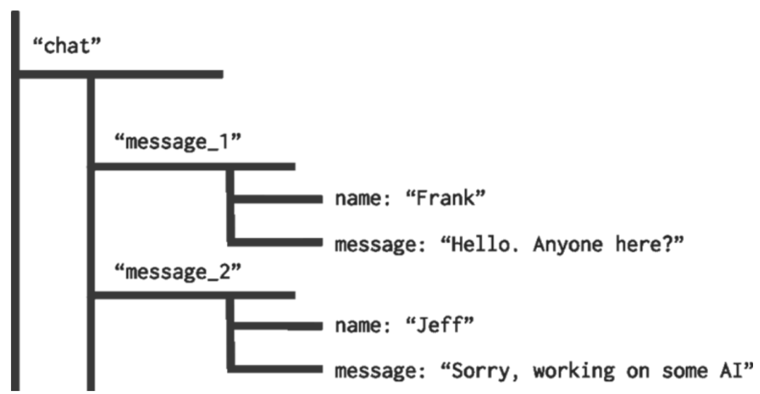
Illustration taken from: Frank van Puffelen, *Have you met the Realtime Database? (2016)* https://firebase.googleblog.com/2016/07/have-you-met-realtime-database.html (2017-02-27)

Firebase

Query Processing in the Client

- Push notifications for **specific keys** only
  - Order by a **single attribute**
  - Apply a **single filter** on that attribute

- Non-trivial query processing in client
  → does not scale!



Jacob Wenger, on the Firebase Google Group *(2015)*
https://groups.google.com/forum/#!topic/firebase-talk/d-XjaBVL2Ko (2017-02-27)

Illustration taken from: Frank van Puffelen, *Have you met the Realtime Database? (2016)*
https://firebase.googleblog.com/2016/07/have-you-met-realtime-database.html (2017-02-27)

Meteor

METEOR

Overview:

- **JavaScript Framework** for interactive apps and websites
  - **MongoDB** under the hood
  - **Real-time** result updates, full MongoDB expressiveness
- **Open-source**: MIT license
- **Managed service**: Galaxy (Platform-as-a-Service)

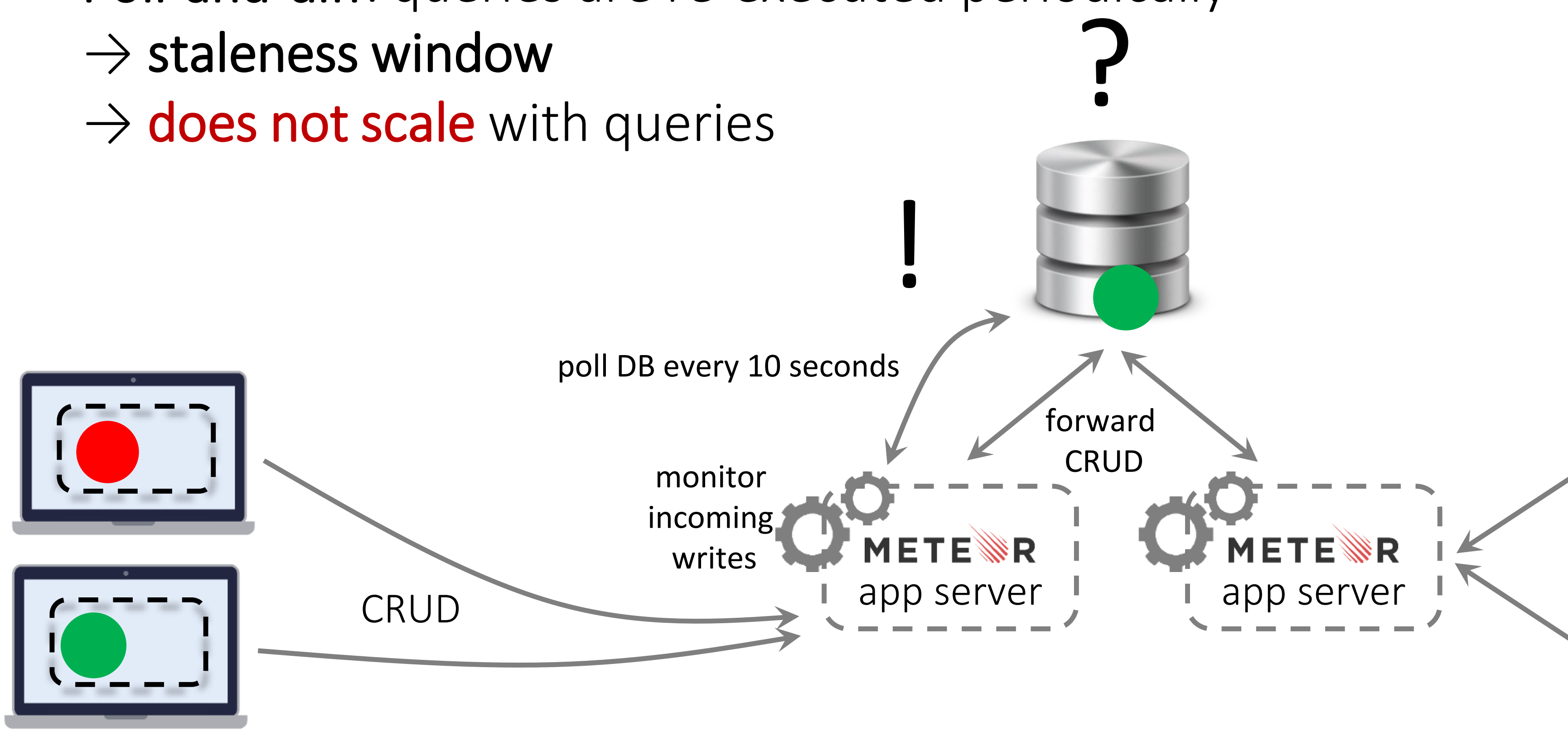
History:

- 2011: *Skybreak* is announced
- 2012: Skybreak is renamed to Meteor
- 2015: Managed hosting service Galaxy is announced

Live Queries

Poll-and-Diff

- **Change monitoring**: app servers detect relevant changes
  → *incomplete* in multi-server deployment
- **Poll-and-diff**: queries are re-executed periodically
  → **staleness window**
  → **does not scale** with queries



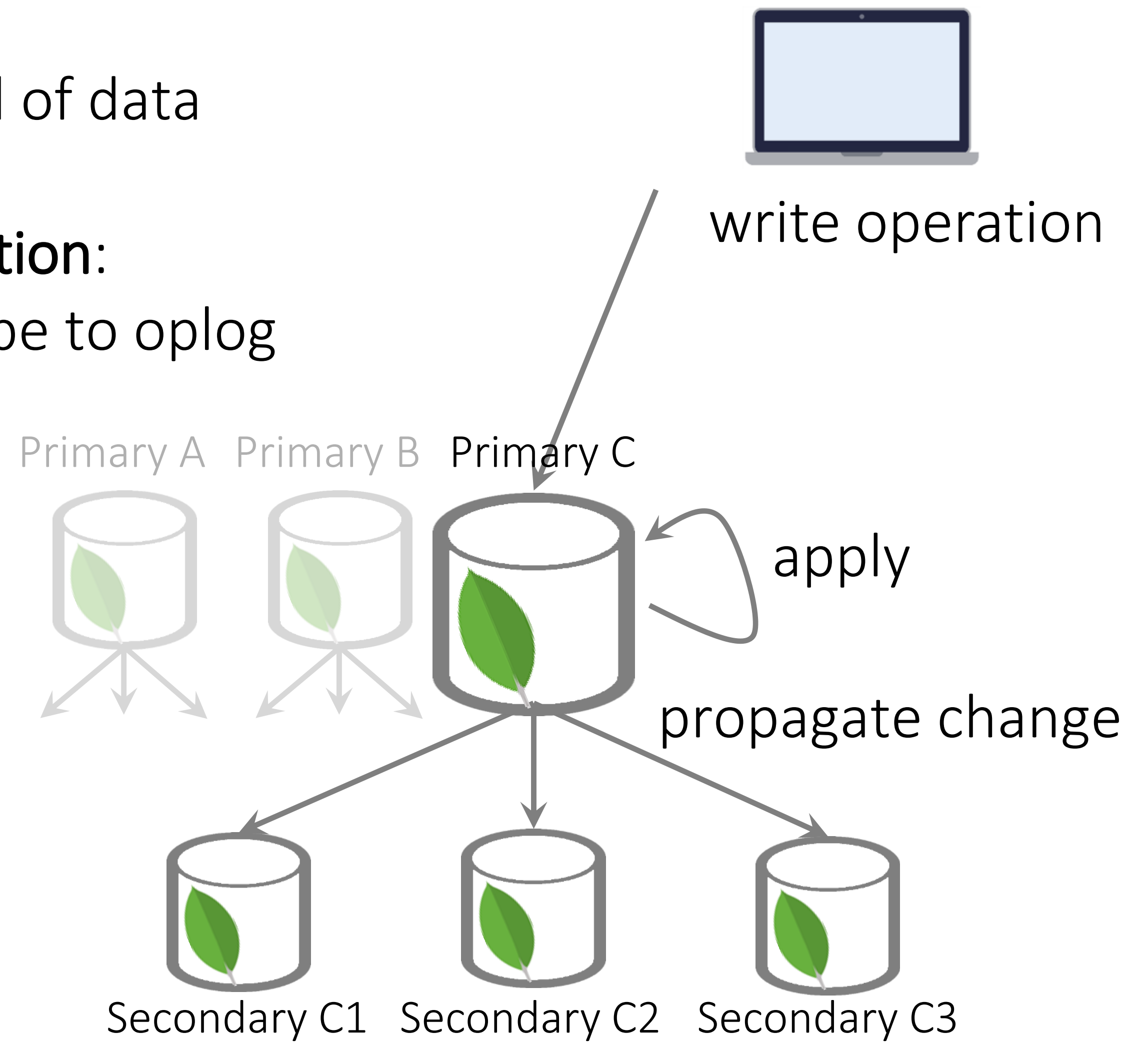
Oplog Tailing

Basics: MongoDB Replication

METEOR

- **Oplog**: rolling record of data modifications
- **Master-slave replication**: Secondaries subscribe to oplog

mongoDB cluster
(3 shards)

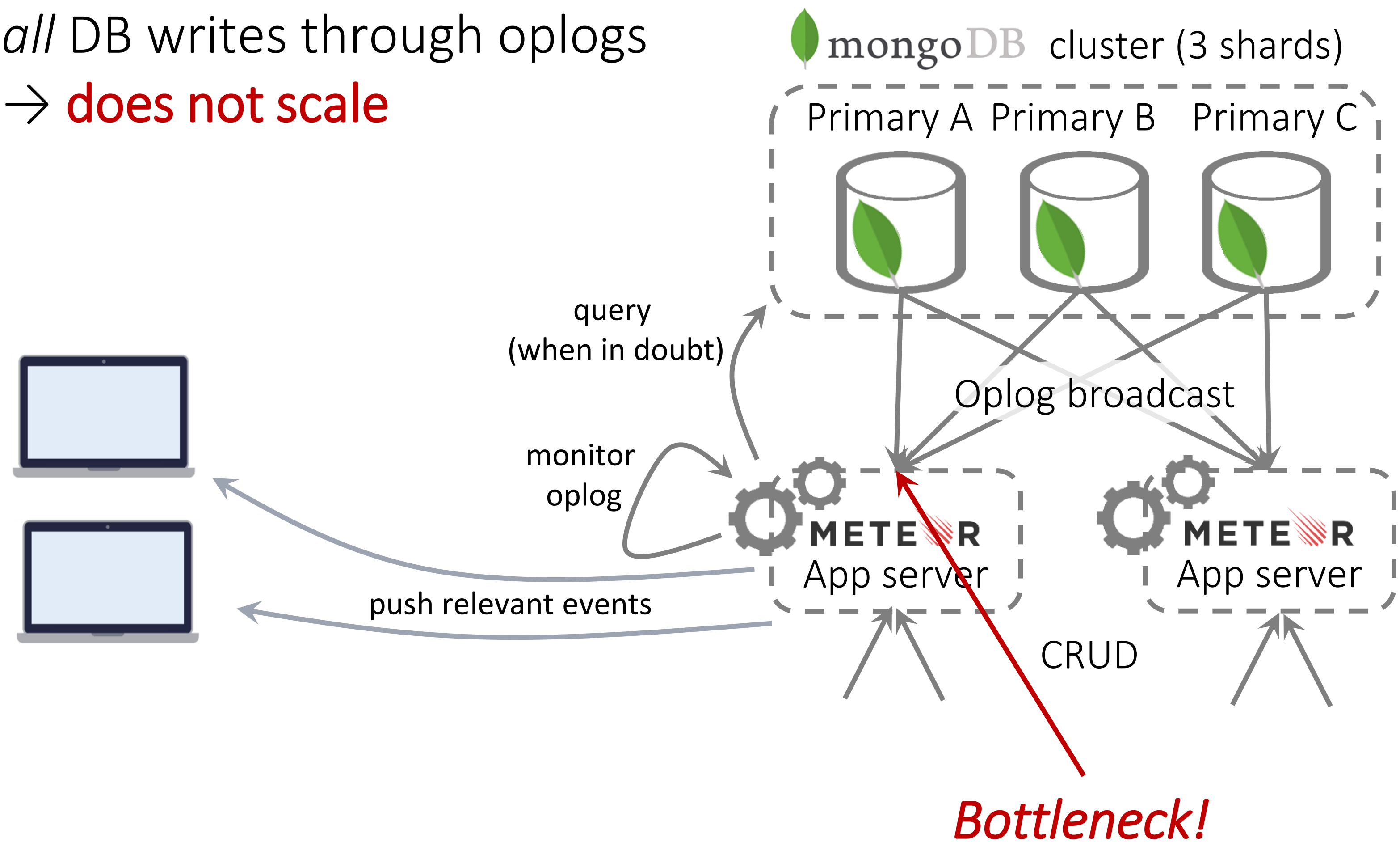


Oplog Tailing

Tapping into the Oplog

METEOR

- *Every* Meteor server receives *all* DB writes through oplogs → does not scale



Oplog Tailing

Oplog Info is Incomplete

METEOR

What game does Bobby play?

→ if baccarat, he takes first place!
→ if something else, nothing changes!

Partial update from oplog:

```
{ name: „Bobby“, score: 500 } // game: ???
```

Baccarat players sorted by high-core

METEOR

```
1. { name: „Joy“, game: „baccarat“, score: 100 }
2. { name: „Tim“, game: „baccarat“, score: 90 }
3. { name: „Lee“, game: „baccarat“, score: 80 }
```

RethinkDB

Overview:

- **„MongoDB done right"**: comparable queries and data model, but also:
  - **Push-based queries** (filters only)
  - **Joins** (non-streaming)
  - **Strong consistency**: linearizability
- **JavaScript SDK** (*Horizon*): open-source, as managed service
- **Open-source**: Apache 2.0 license

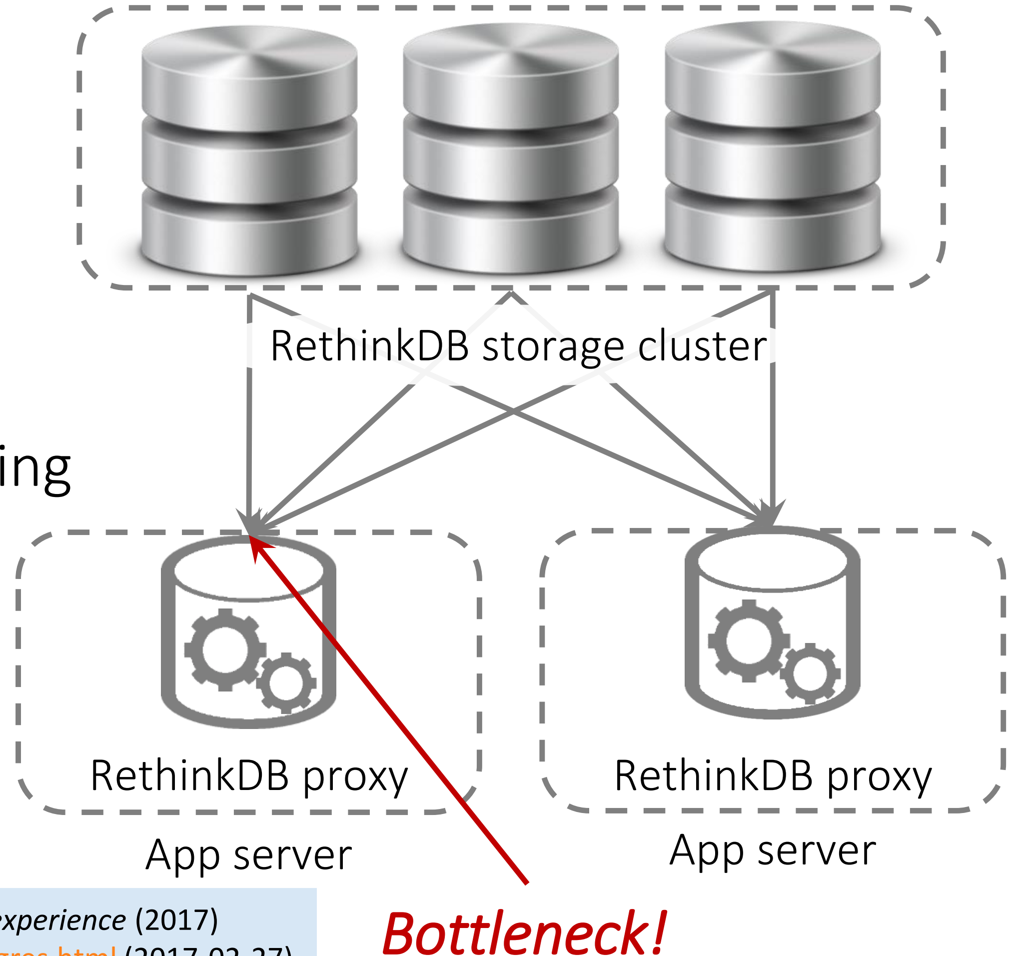
History:

- 2009: RethinkDB is founded
- 2012: RethinkDB is open-sourced under AGPL
- 2016, May: first official release of Horizon (JavaScript SDK)
- 2016, October: RethinkDB announces shutdown
- 2017: RethinkDB is relicensed under Apache 2.0

RethinkDB
Changefeed Architecture

RethinkDB

- Range-sharded data
- **RethinkDB proxy**: support node without data
    - Client communication
    - Request routing
    - Real-time query matching
- *Every* proxy receives *all* database writes
  → **does not scale**



William Stein, *RethinkDB versus PostgreSQL: my personal experience* (2017)
http://blog.sagemath.com/2017/02/09/rethinkdb-vs-postgres.html (2017-02-27)

Daniel Mewes, *Comment on GitHub issue #962: Consider adding more docs on RethinkDB Proxy* (2016)
https://github.com/rethinkdb/docs/issues/962 (2017-02-27)

Parse

Parse

Overview:

- **Backend-as-a-Service** for mobile apps
  - **MongoDB:** largest deployment world-wide
  - **Easy development**: great docs, push notifications, authentication, …
  - **Real-time** updates for most MongoDB queries
- **Open-source**: BSD license
- **Managed service**: discontinued

History:

- 2011: Parse is founded
- 2013: Parse is acquired by Facebook
- 2015: more than 500,000 mobile apps reported on Parse
- 2016, January: Parse shutdown is announced
- 2016, March: **Live Queries** are announced
- 2017: Parse shutdown is finalized

Parse

LiveQuery Architecture

- **LiveQuery Server**: no data, real-time query matching
- *Every* LiveQuery Server receives *all* database writes
  → does not scale

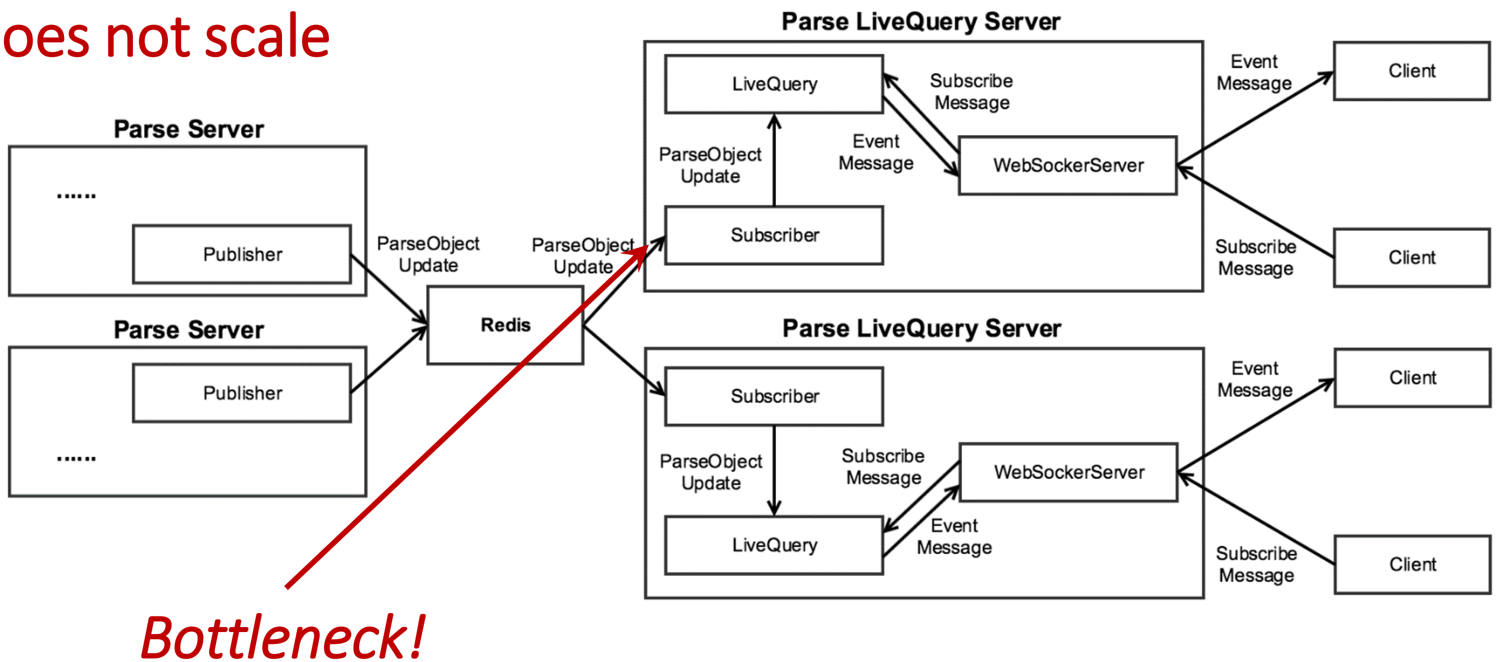


Illustration taken from:
http://parseplatform.github.io/docs/parse-server/guide/#live-queries (2017-02-22)

Comparison by Real-Time Query

Why Complexity Matters

	matching conditions	ordering	Firebase	Meteor	RethinkDB	Parse
Todos	created by „Bob“	ordered by deadline	✓	✓	✓	✗
Todos	created by „Bob“ AND with status equal to „active“		✗	✓	✓	✓
Todos	with „work“ in the name		✗	✓	✓	✓
		ordered by deadline	✗	✓	✓	✗
Todos	with „work“ in the name AND status of „active“	ordered by deadline AND then by the creator‘s name	✗	✓	✓	✗

Quick Comparison

DBMS vs. RT DB vs. DSMS vs. Stream Processing

	Database Management	**Real-Time Databases**	**Data Stream Management**	**Stream Processing**
Data	persistent collections		persistent/ephemeral streams	
Processing	one-time	one-time + continuous	continuous	
Access	random	random + sequential	sequential	
Streams	structured			structured, unstructured
	ORACLE, PostgreSQL, MySQL, IBM DB2	Firebase, METEOR, RethinkDB, Parse	PIPELINEDB, EsperTech, sqlstream, influxdata	STORM, samza, Flink, Spark Streaming

Discussion
Common Issues

Every database with real-time features suffers from several of these problems:

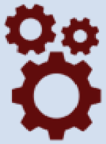
- *Expressiveness*:
  - Queries
  - Data model
  - Legacy support
- *Performance*:
  - Latency & throughput
  - Scalability
- *Robustness*:
  - Fault-tolerance, handling malicious behavior etc.
  - Separation of concerns:

    → **Availability**:
    will a crashing real-time subsystem take down primary data storage?

    → **Consistency**:
    can real-time be scaled out independently from primary storage?

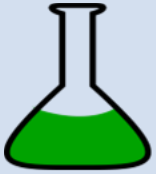
Outline

Scalable Data Processing:
Big Data in Motion



Stream Processors:
Side-by-Side Comparison

Real-Time Databases:
Push-Based Data Access



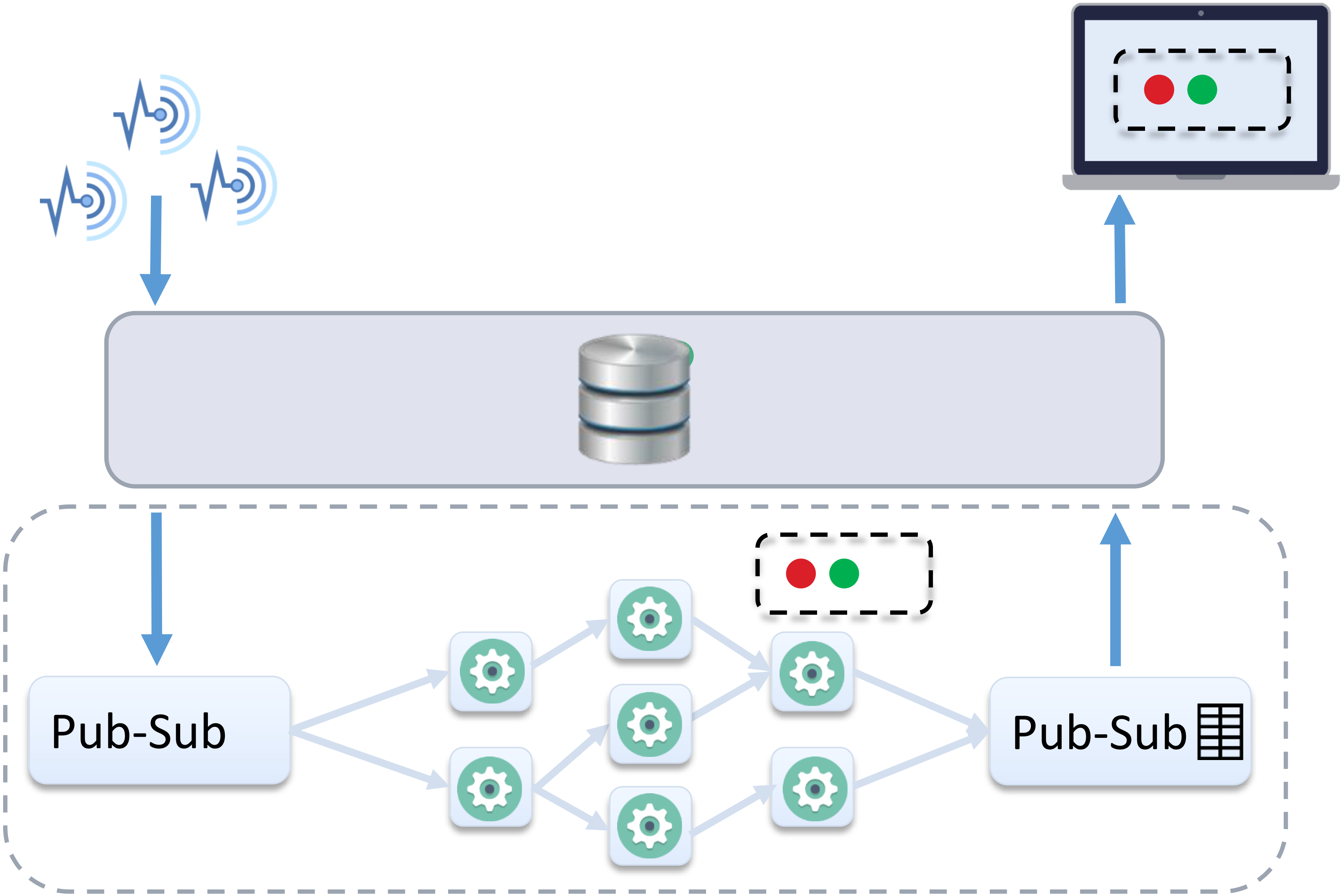
Current Research:
Opt-In Push-Based Access

- InvaliDB:
  Opt-In Real-Time Queries
- Distributed Query Matching
- Staged Query Processing
- Performance Evaluation
- Wrap-Up

Current Research

InvaliDB

External Query Maintenance



InvaliDB
Change Notifications

```
SELECT *
FROM posts
WHERE title LIKE "%NoSQL%"
ORDER BY year DESC
```

```
{ title: "SQL",
  year: 2016 }
```

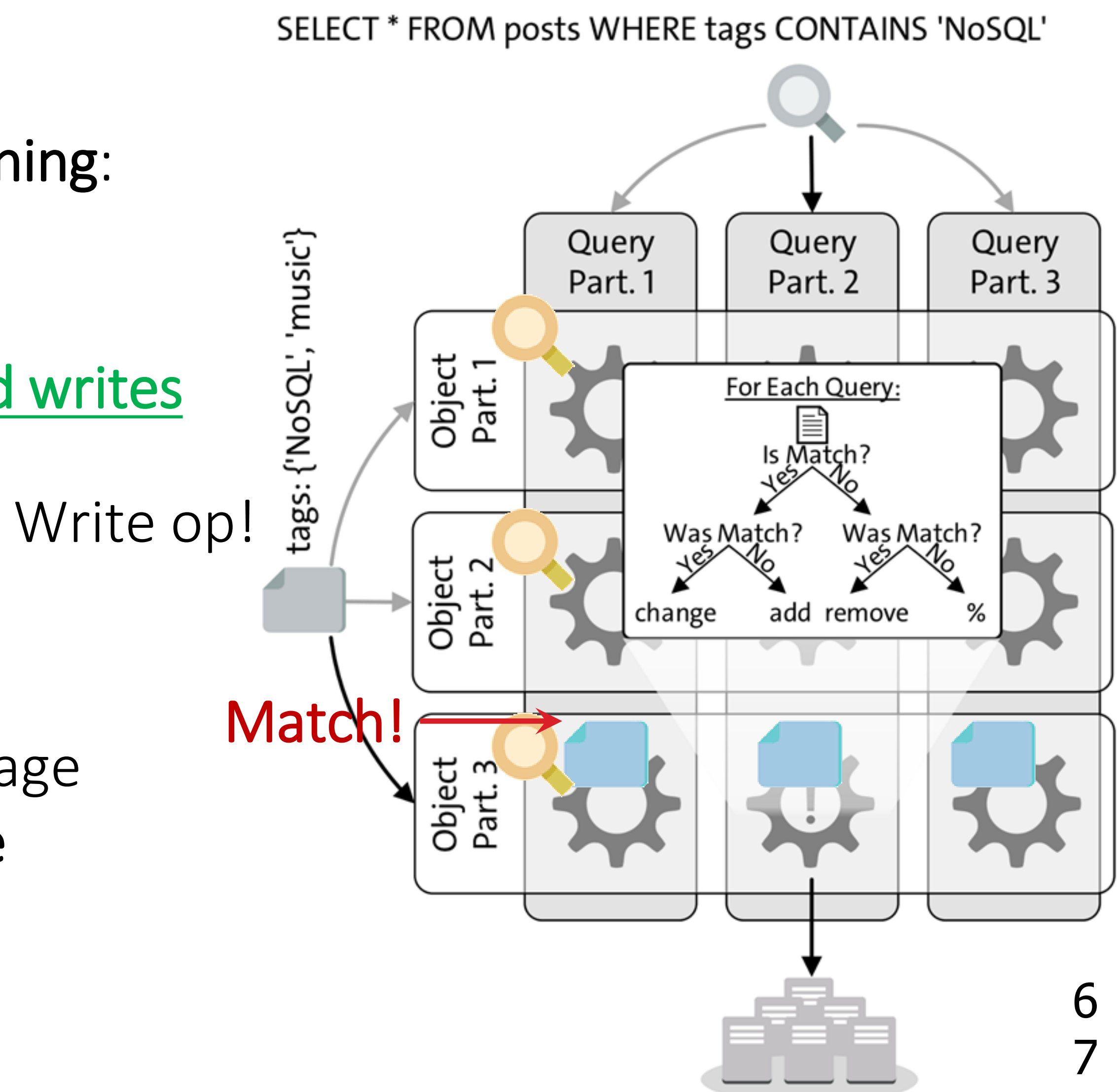
add changeIndex change remove

InvaliDB
Filter Queries: Distributed Query Matching

Two-dimensional partitioning:
- *by Query*
- *by Object*

→ scales with queries and writes

Implementation:
- Apache Storm
- Topology in Java
- MongoDB query language
- **Pluggable query engine**

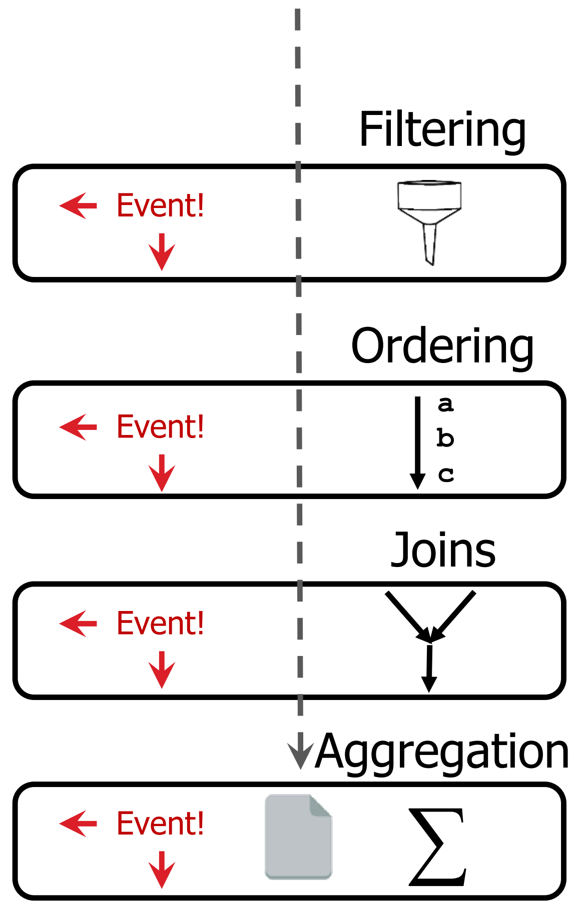


InvaliDB

Staged Real-Time Query Processing

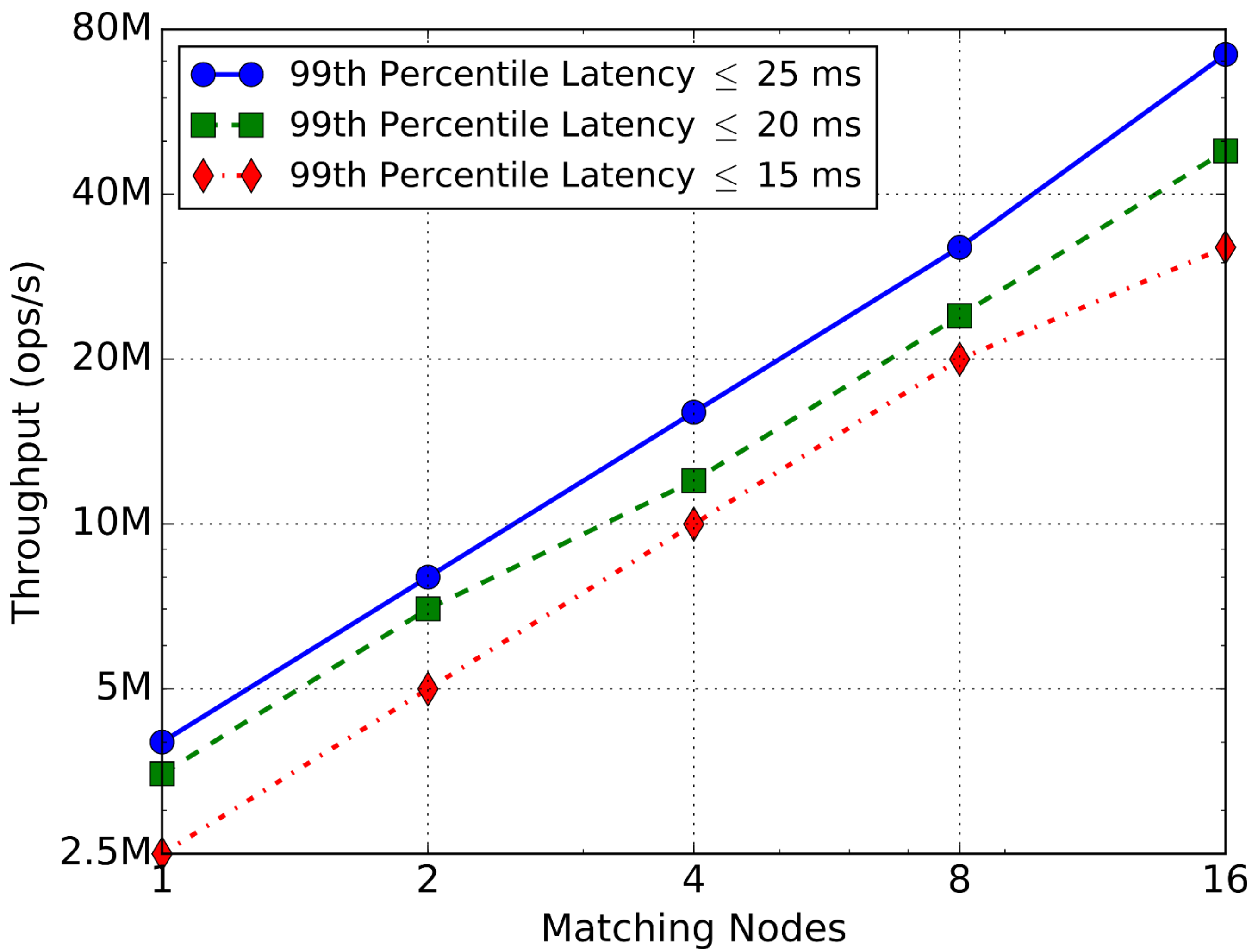
Change notifications go through up to 4 query processing stages:

1. **Filter queries**: track matching status
   → *before*- and after-images
2. **Sorted queries**: maintain result order
3. **Joins**: combine maintained results
4. **Aggregations**: maintain aggregations



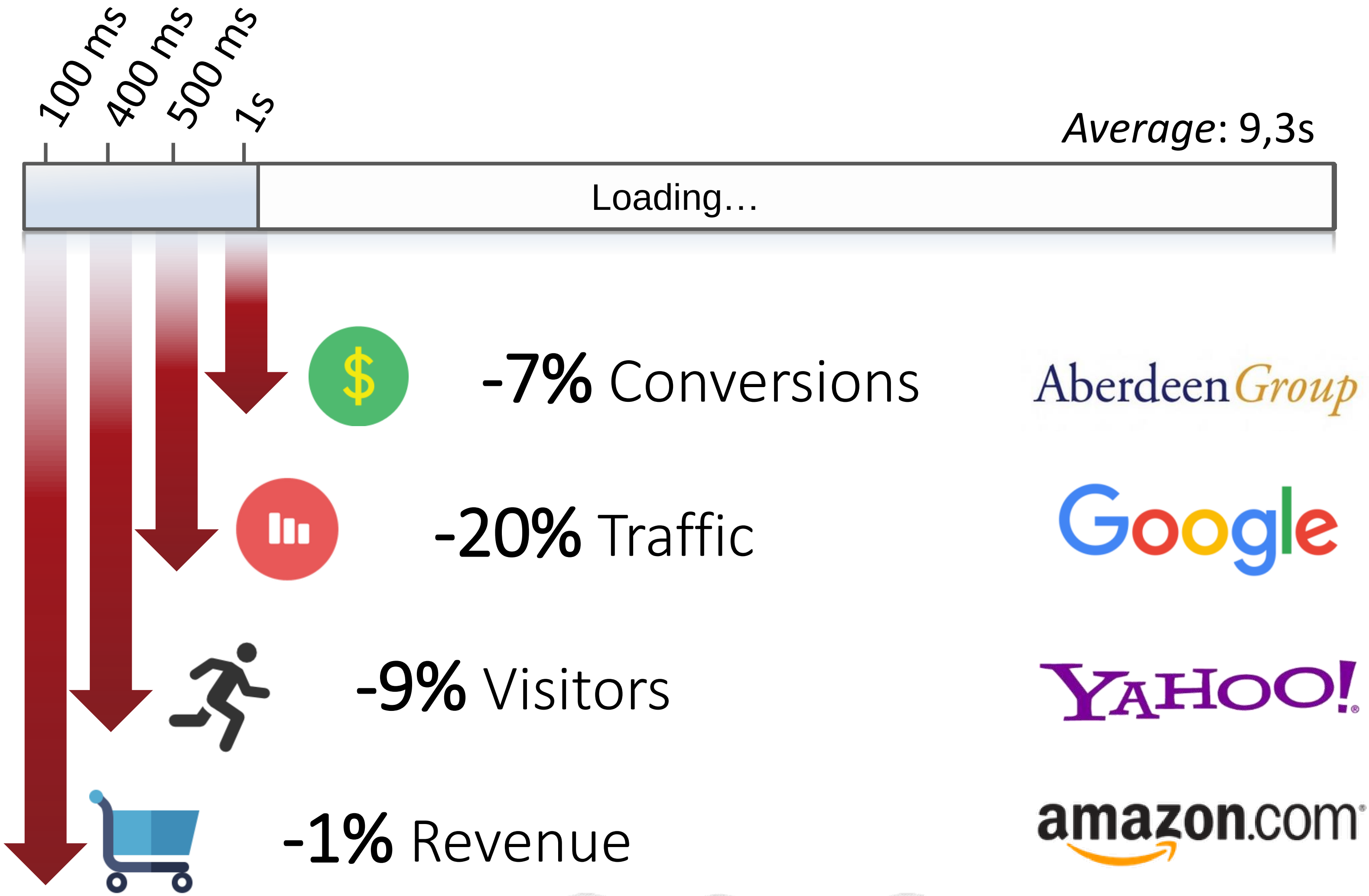
InvaliDB

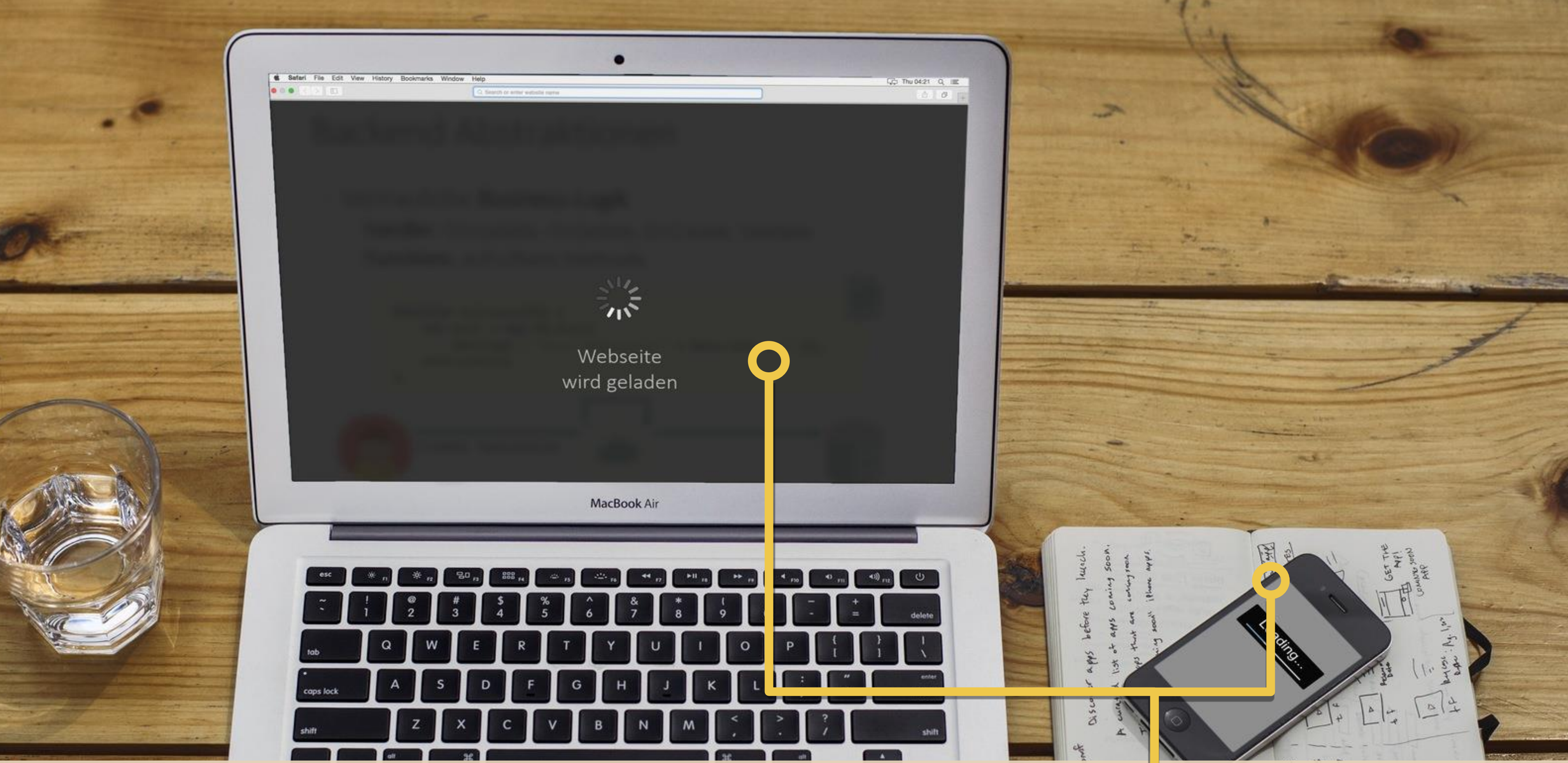
Low Latency + Linear Scalability



Our NoSQL research at the University of Hamburg

The Latency Problem



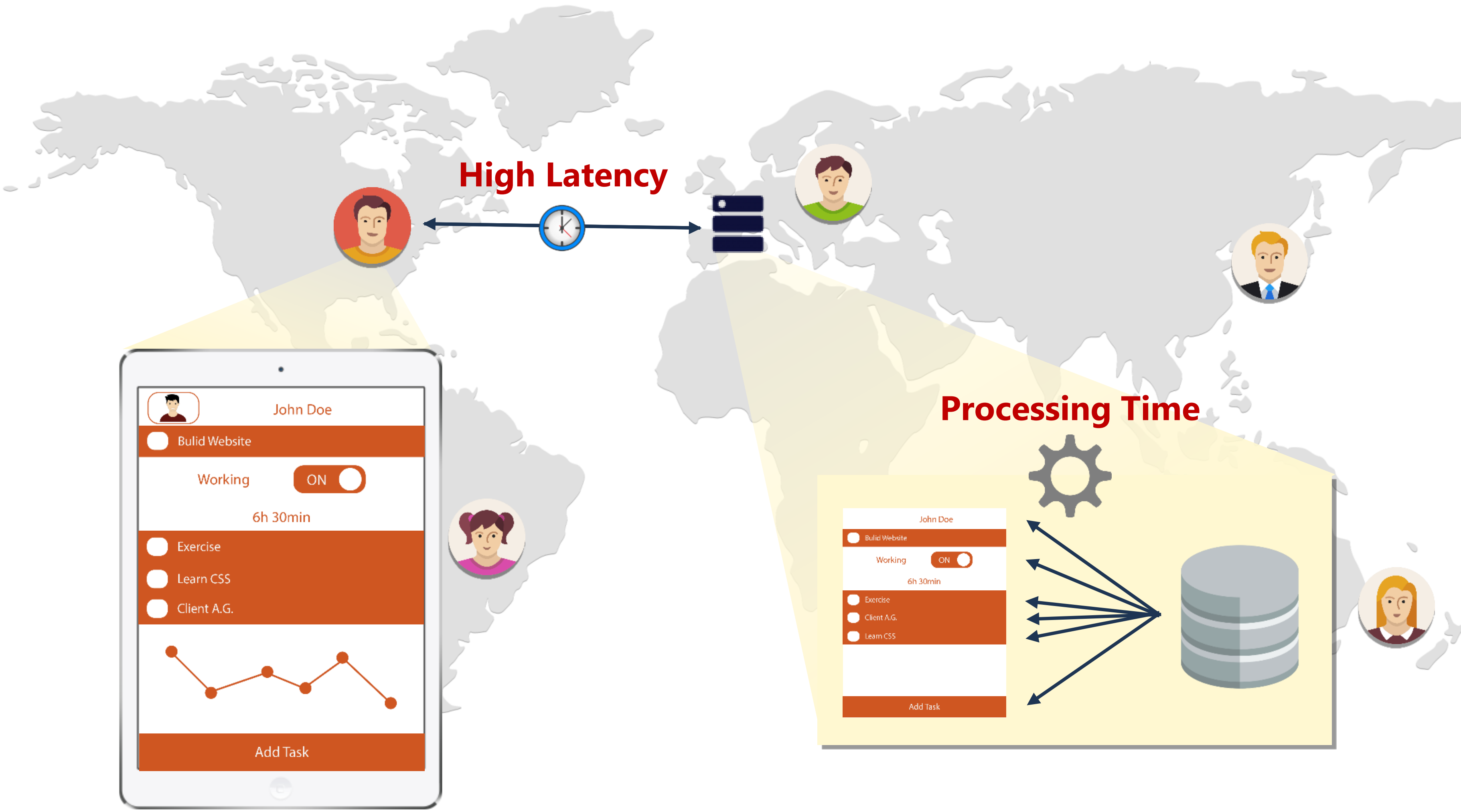


If perceived speed is such an important factor

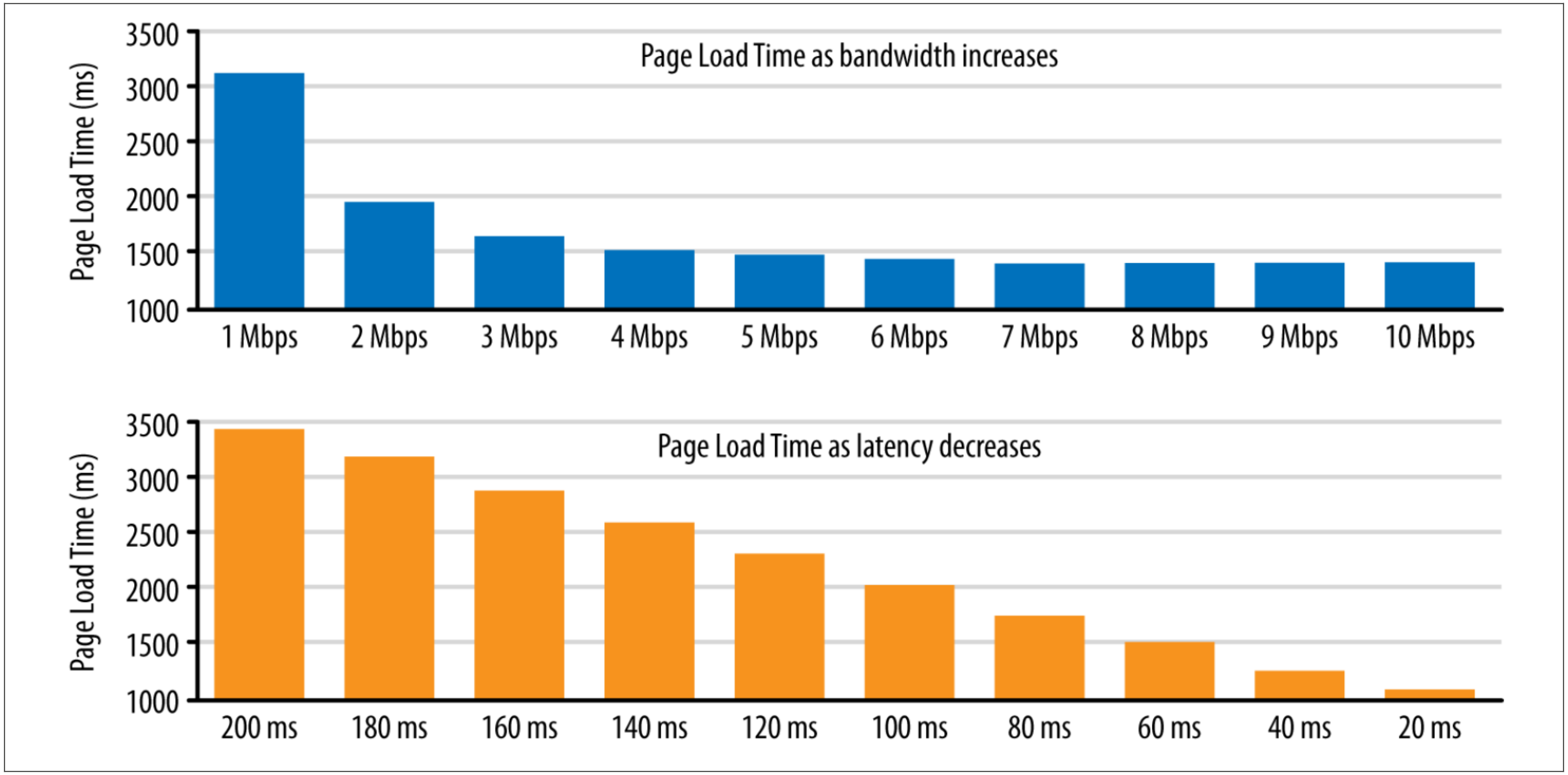
...what causes slow page load times?

State of the Art

Two bottlenecks: latency und processing

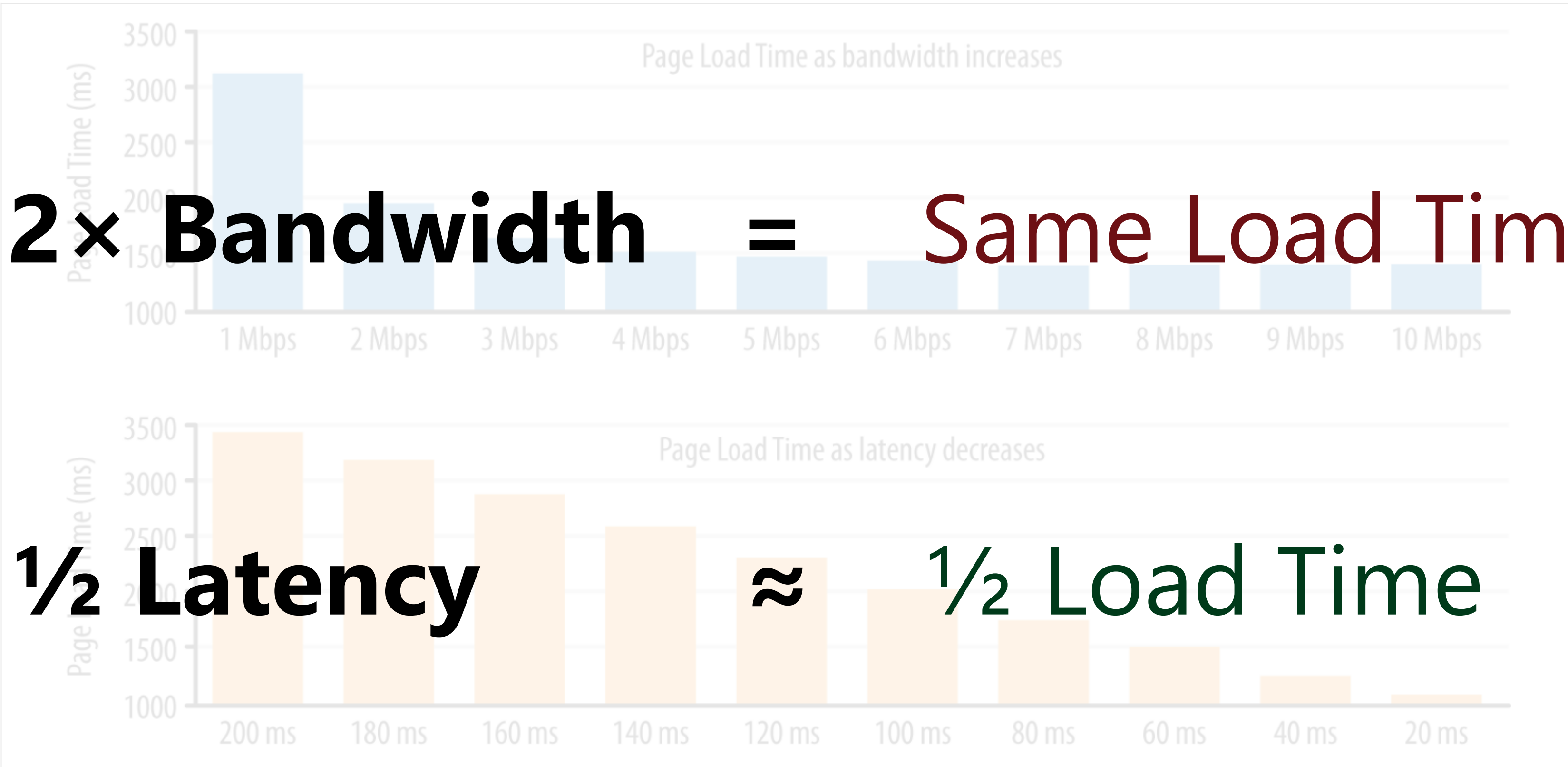


Network Latency: Impact



I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

Network Latency: Impact



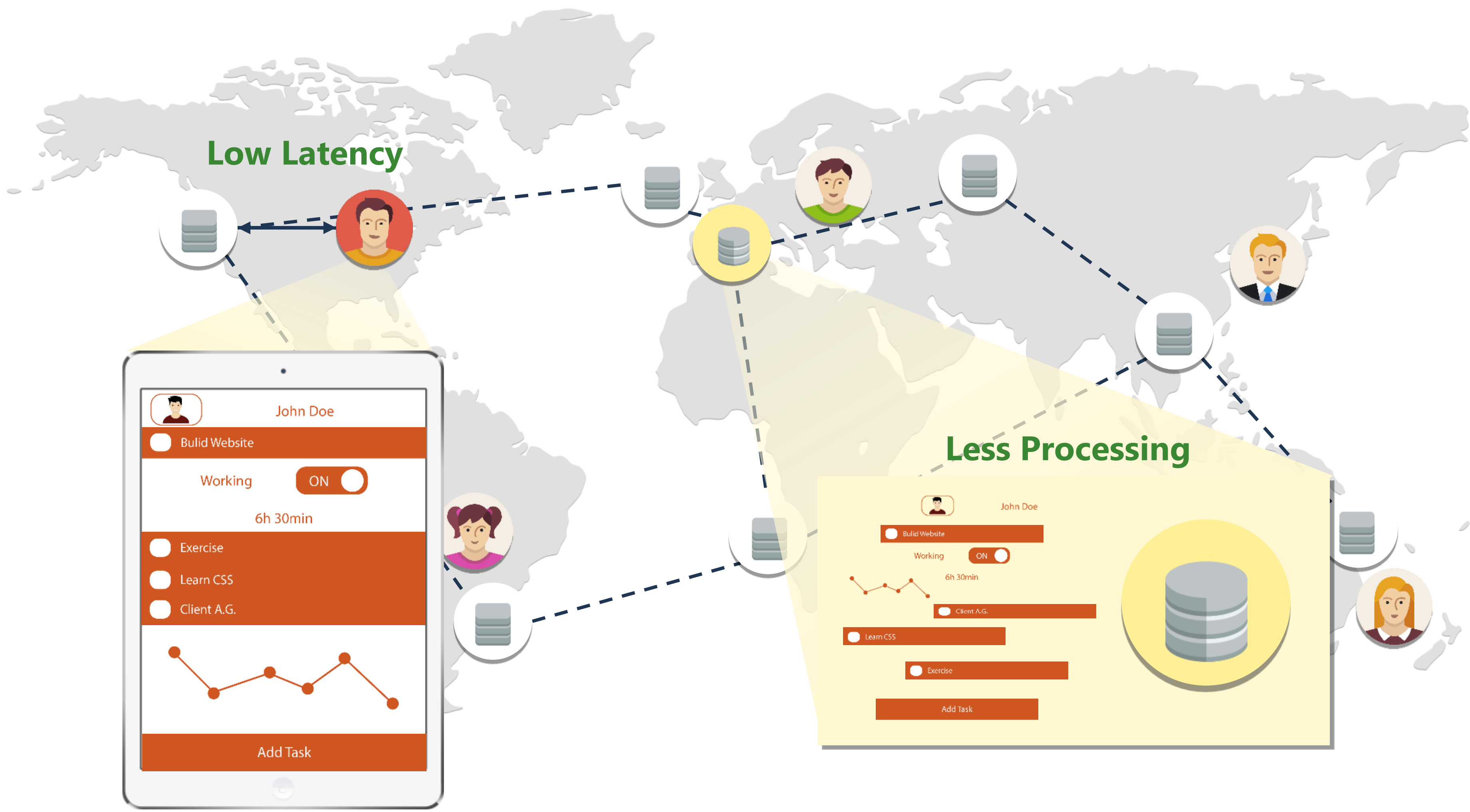
2× Bandwidth = Same Load Time

½ Latency ≈ ½ Load Time

I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

Our Low-Latency Vision

Data is served by ubiquitous web-caches

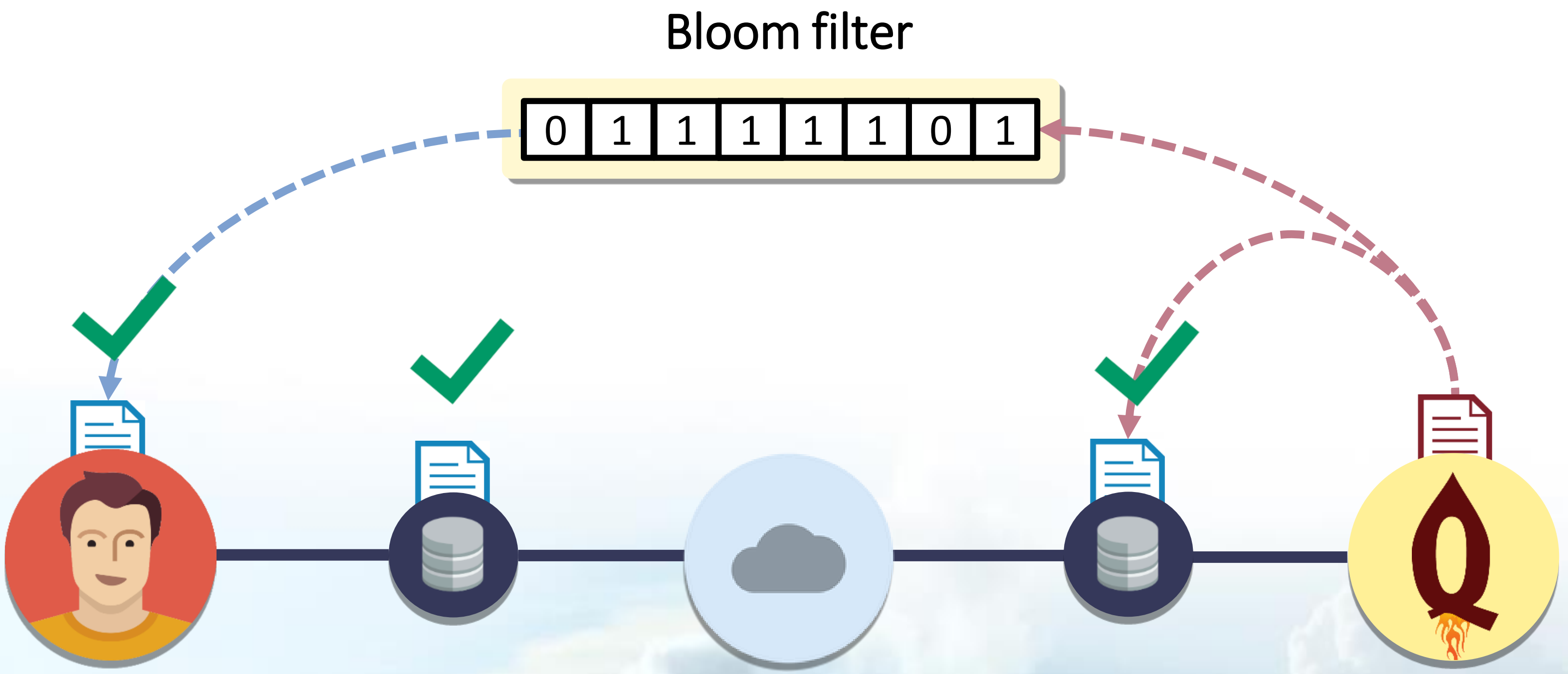


Innovation

Solution: Proactively Revalidate Data

5 Years
Research & Development

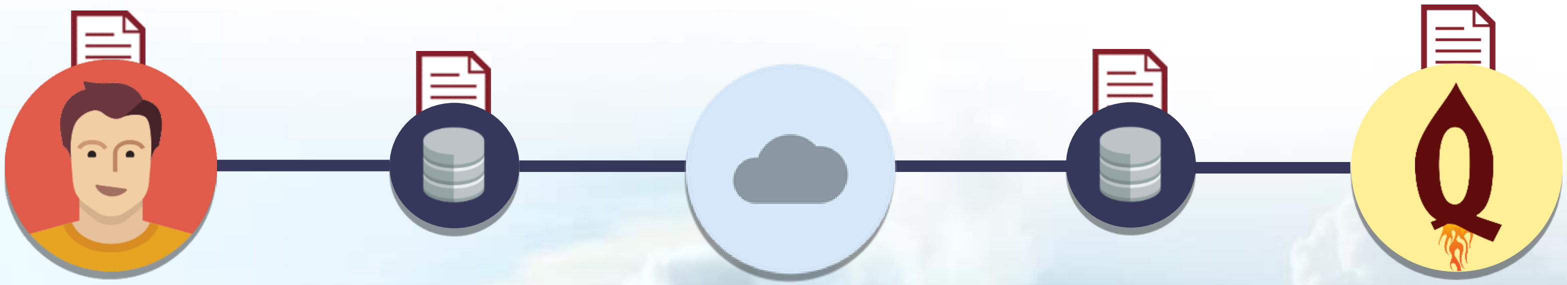
New Algorithms
Solve Consistency Problem



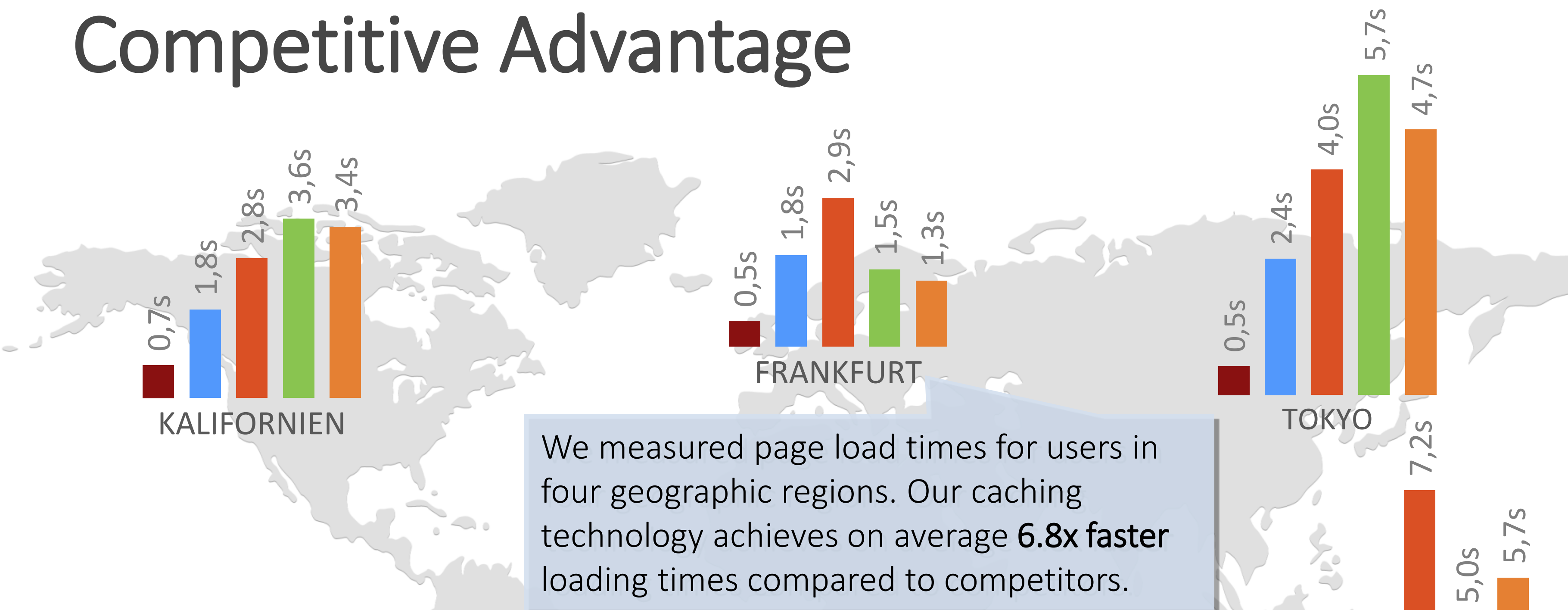
Innovation

Solution: Proactively Revalidate Data

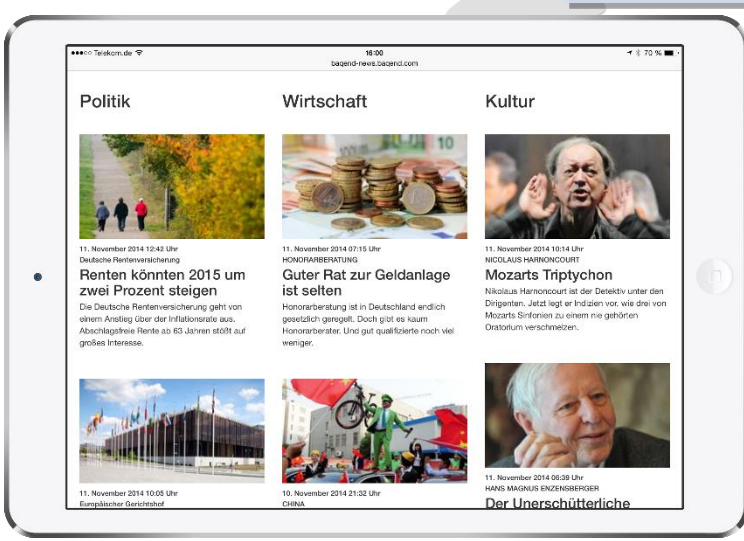
- F. Gessert, F. Bücklers, und N. Ritter, „ORESTES: a Scalable Database-as-a-Service Architecture for Low Latency“, in *CloudDB 2014*, 2014.
- F. Gessert, S. Friedrich, W. Wingerath, M. Schaarschmidt, und N. Ritter, „Towards a Scalable and Unified REST API for Cloud Data Stores“, in *44. Jahrestagung der GI*, Bd. 232, S. 723–734.
- F. Gessert und F. Bücklers, „ORESTES: ein System für horizontal skalierbaren Zugriff auf Cloud-Datenbanken“, in Informatiktage 2013, 2013.
- F. Gessert, M. Schaarschmidt, W. Wingerath, S. Friedrich, und N. Ritter, „The Cache Sketch: Revisiting Expiration-based Caching in the Age of Cloud Data Management“, in BTW 2015.
- F. Gessert und F. Bücklers, *Performanz- und Reaktivitätssteigerung von OODBMS vermittels der Web-Caching-Hierarchie*. Bachelorarbeit, 2010.
- F. Gessert und F. Bücklers, *Kohärentes Web-Caching von Datenbankobjekten im Cloud Computing*. Masterarbeit 2012.
- M. Schaarschmidt, F. Gessert, und N. Ritter, „Towards Automated Polyglot Persistence“, in BTW 2015.
- W. Wingerath, S. Friedrich, und F. Gessert, „Who Watches the Watchmen? On the Lack of Validation in NoSQL Benchmarking“, in BTW 2015.
- S. Friedrich, W. Wingerath, F. Gessert, und N. Ritter, „NoSQL OLTP Benchmarking: A Survey“, in *44. Jahrestagung der Gesellschaft für Informatik*, 2014, Bd. 232, S. 693–704.
- F. Gessert, „Skalierbare NoSQL- und Cloud-Datenbanken in Forschung und Praxis“, BTW 2015



Competitive Advantage



We measured page load times for users in four geographic regions. Our caching technology achieves on average **6.8x faster** loading times compared to competitors.

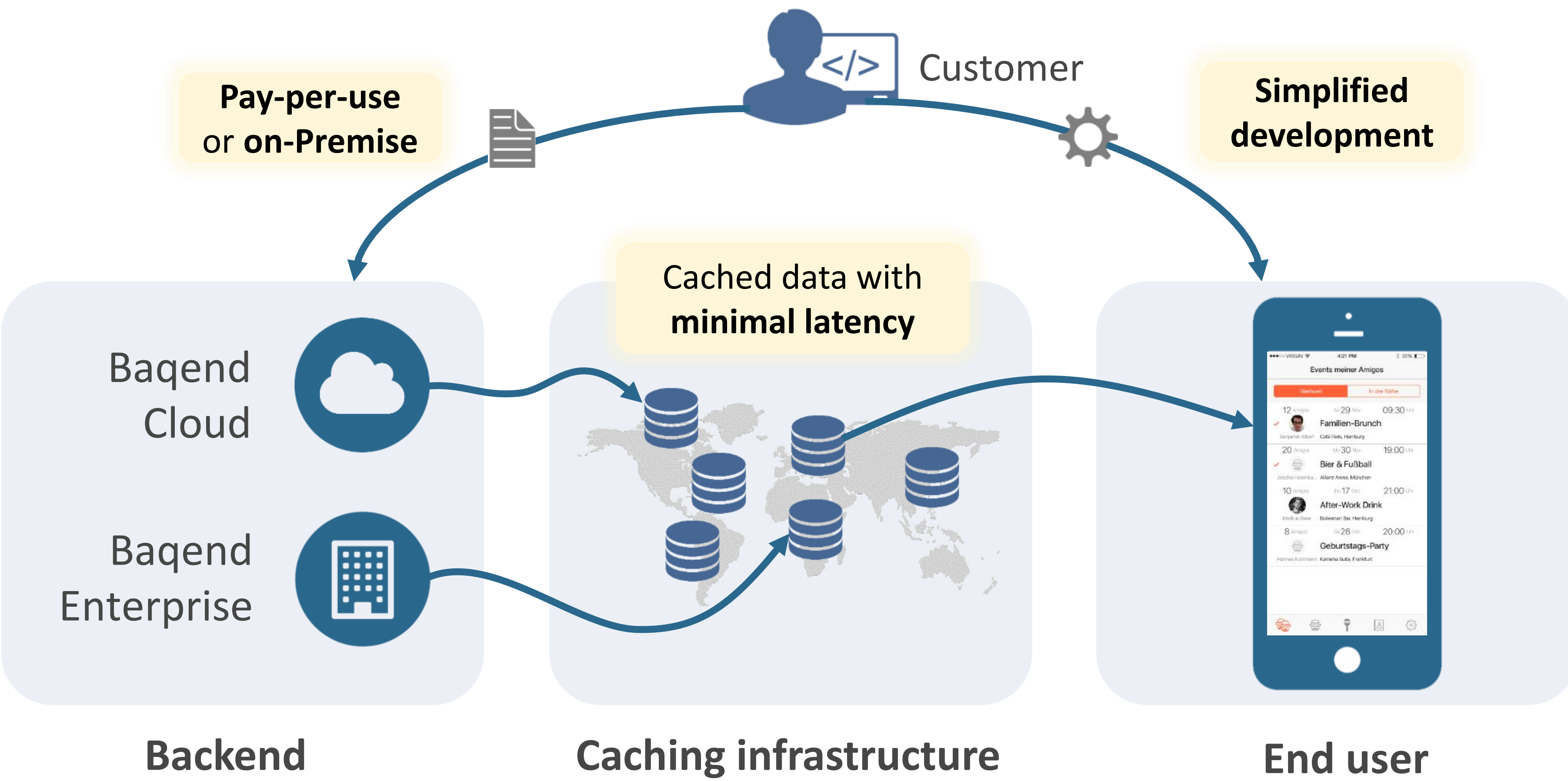


BaQend

Other BaaS providers

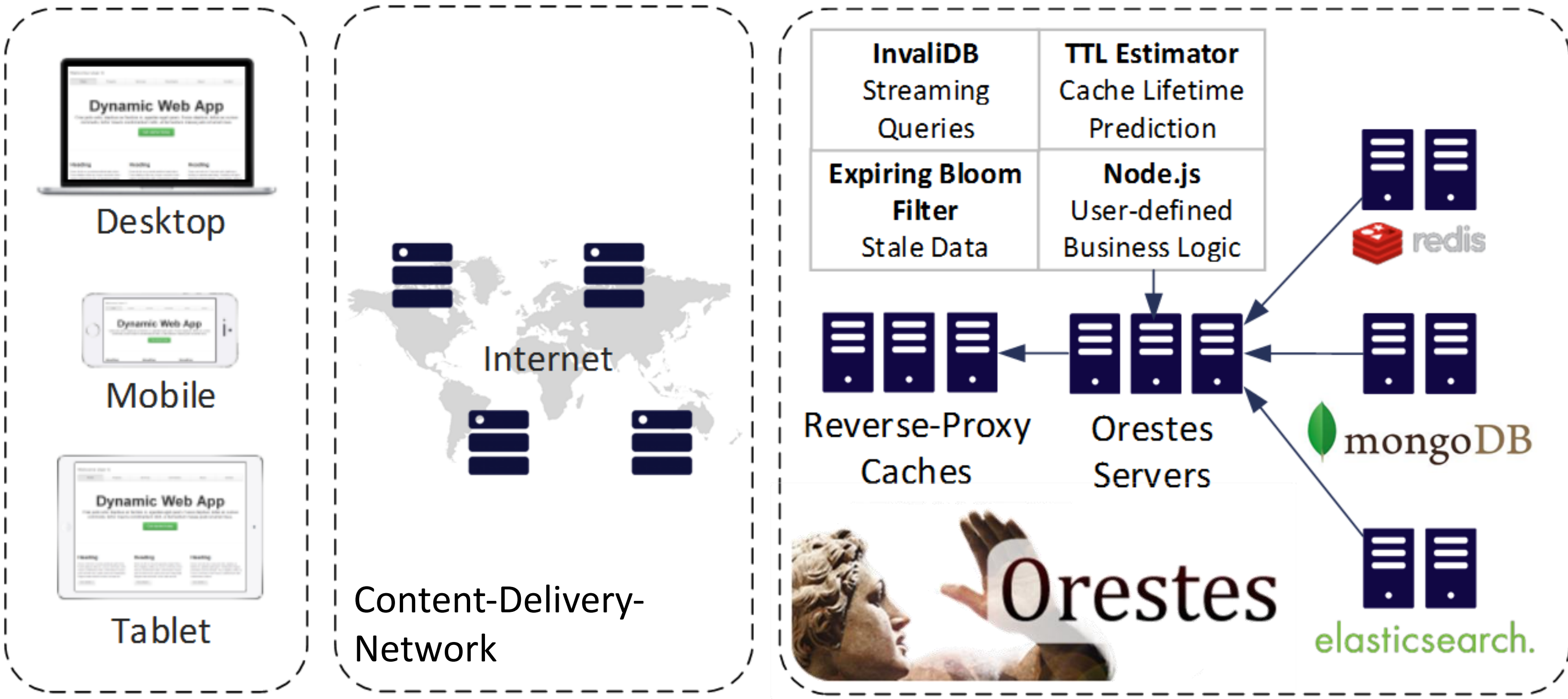
Business Model

Backend-as-a-Service



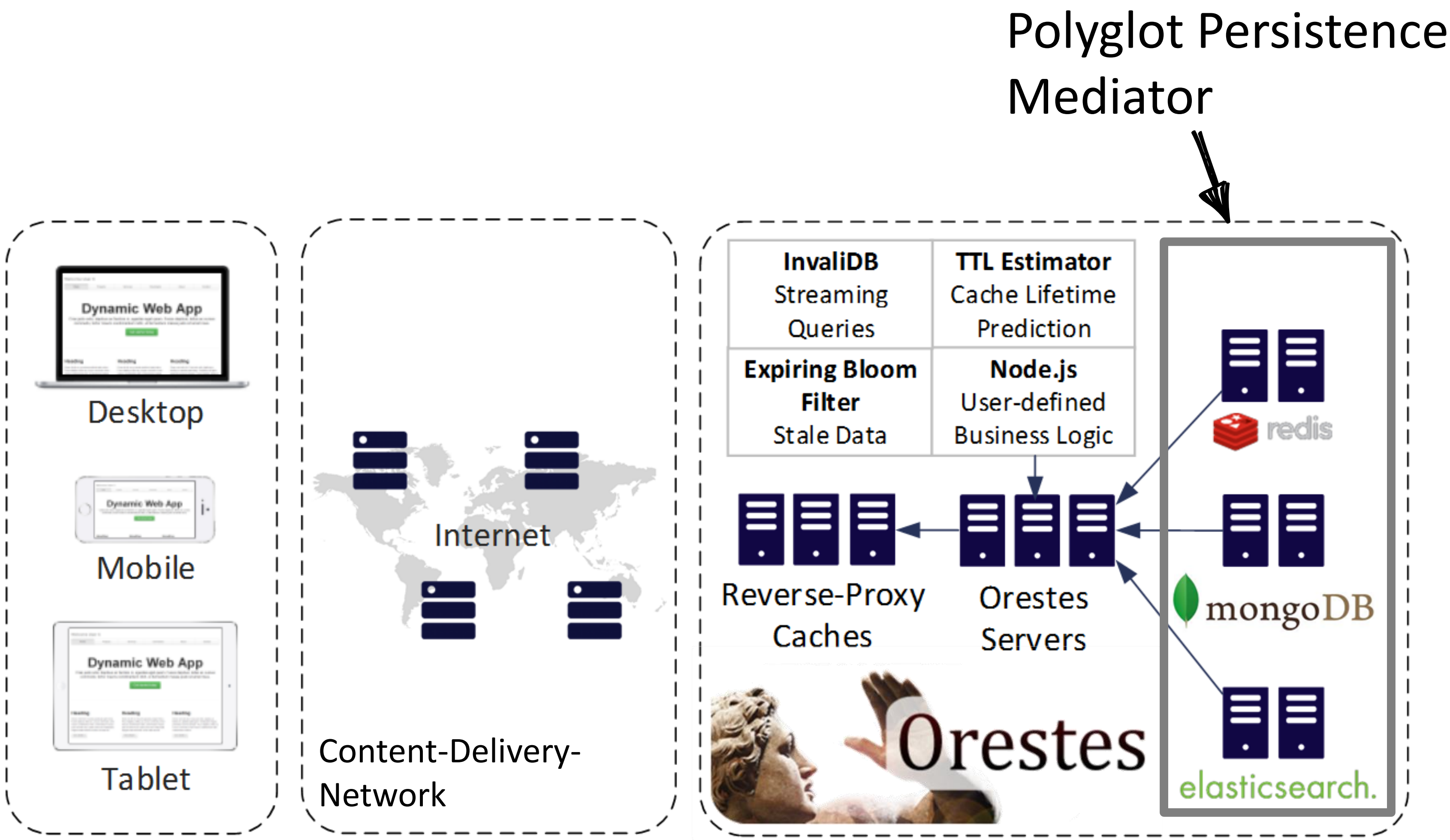
Orestes

Components



Orestes

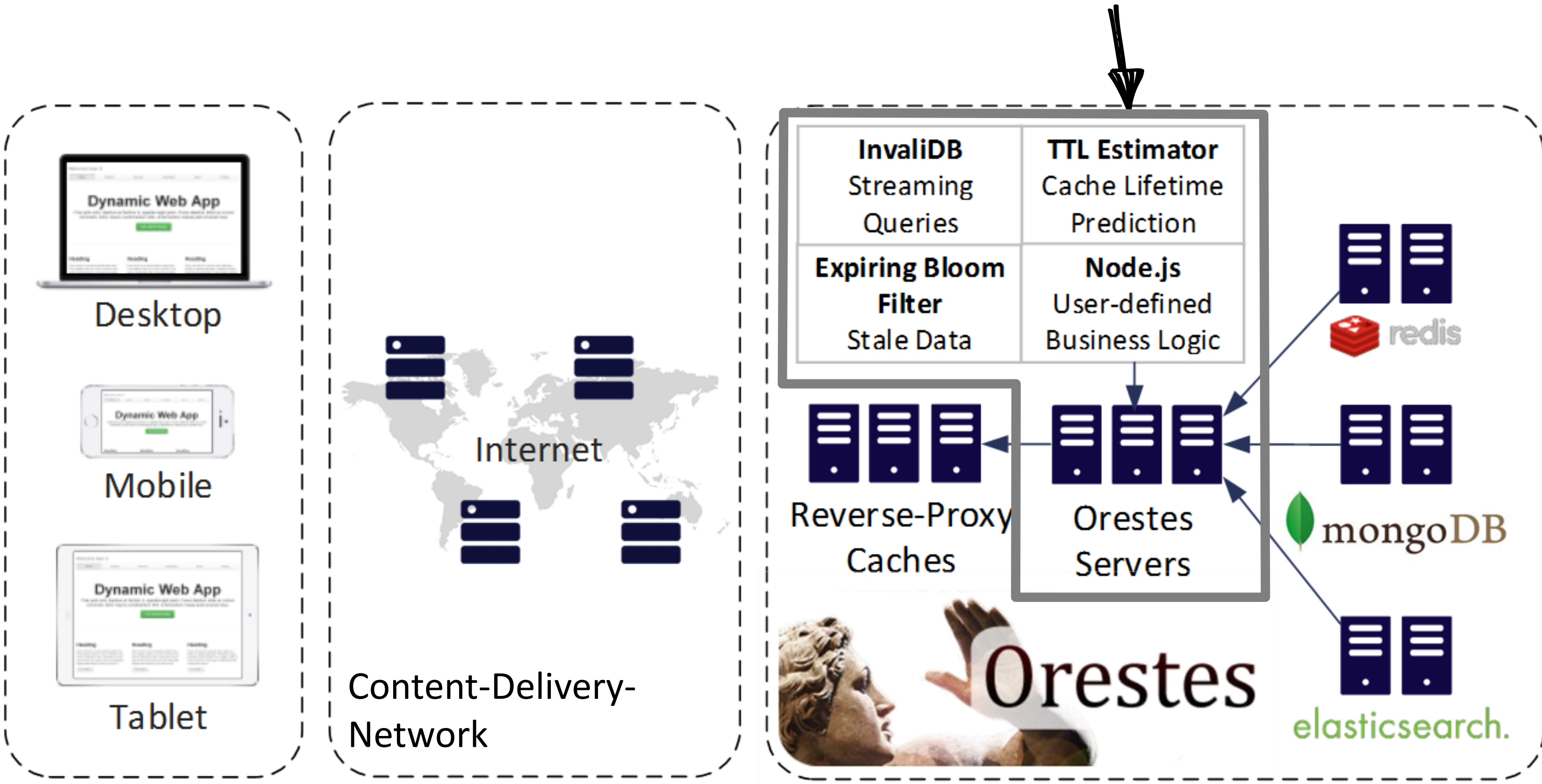
Components



Orestes

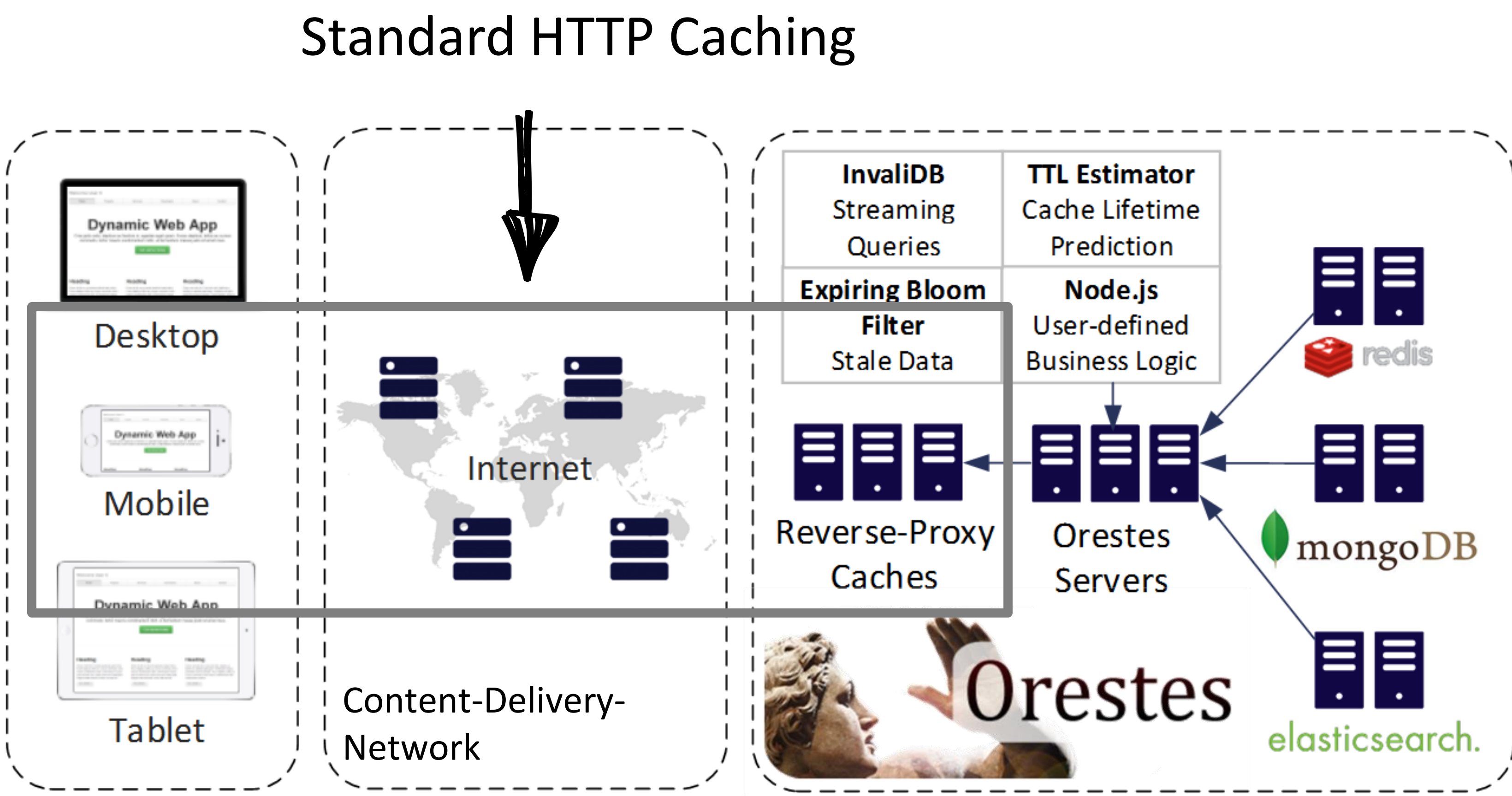
Components

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



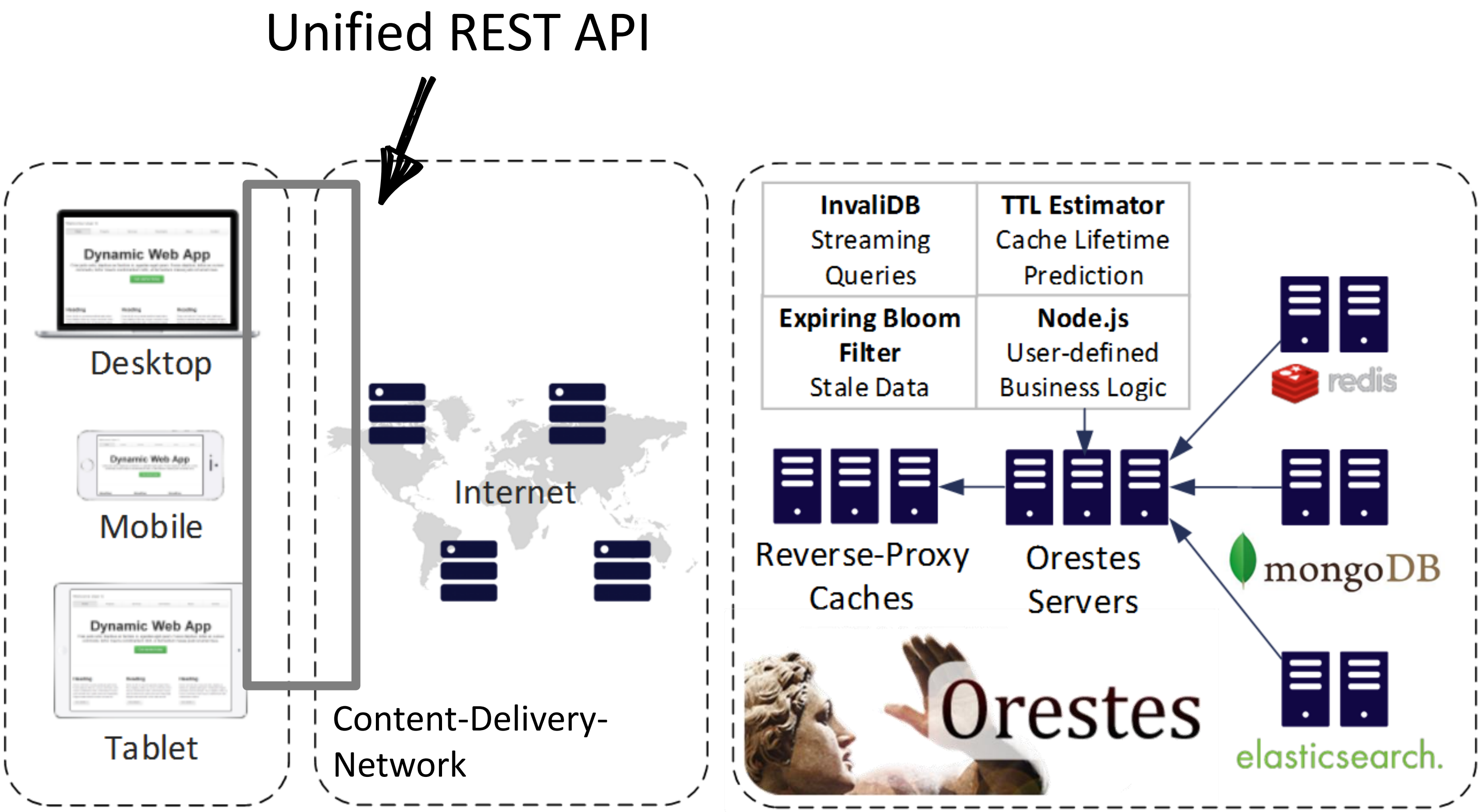
Orestes

Components



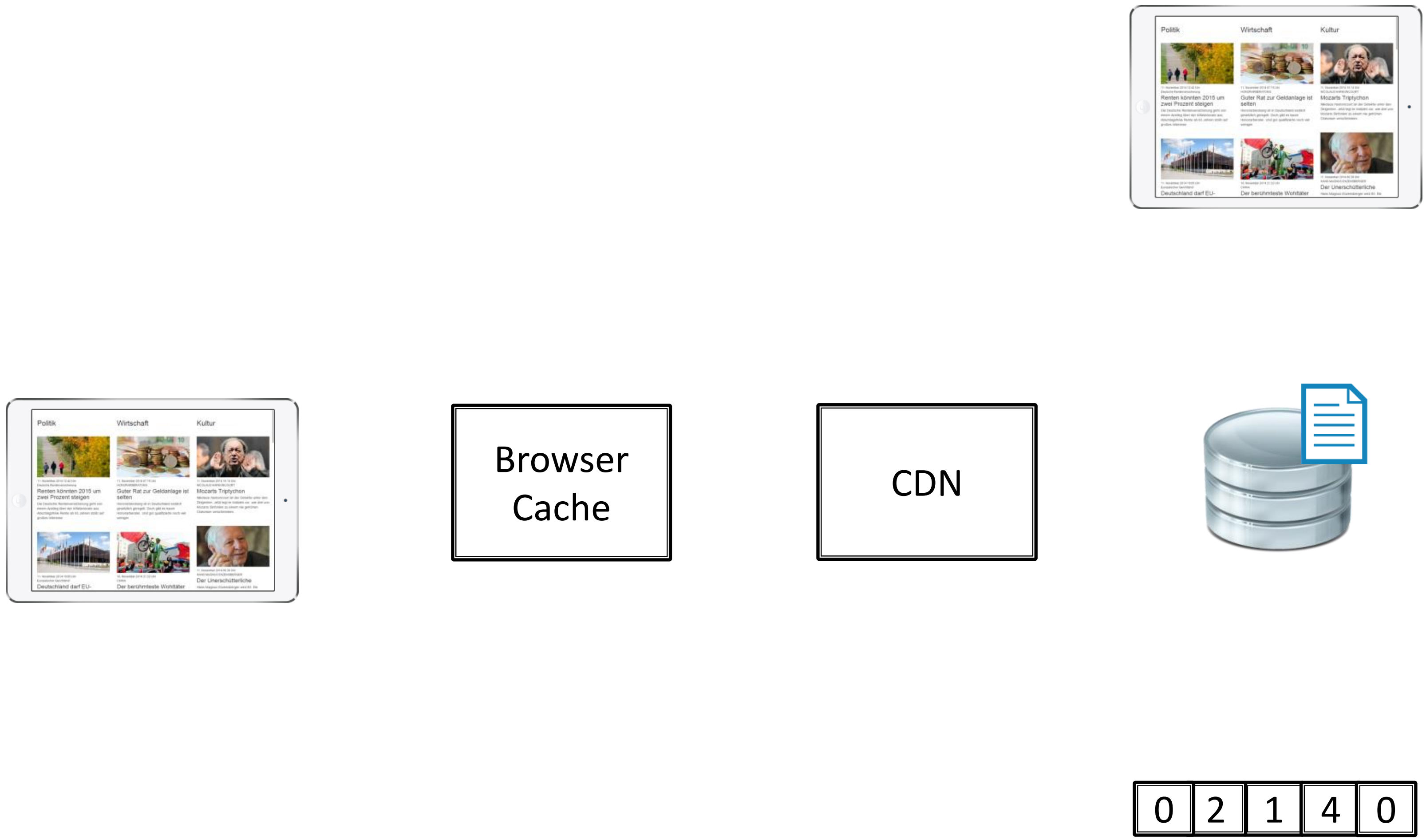
Orestes

Components



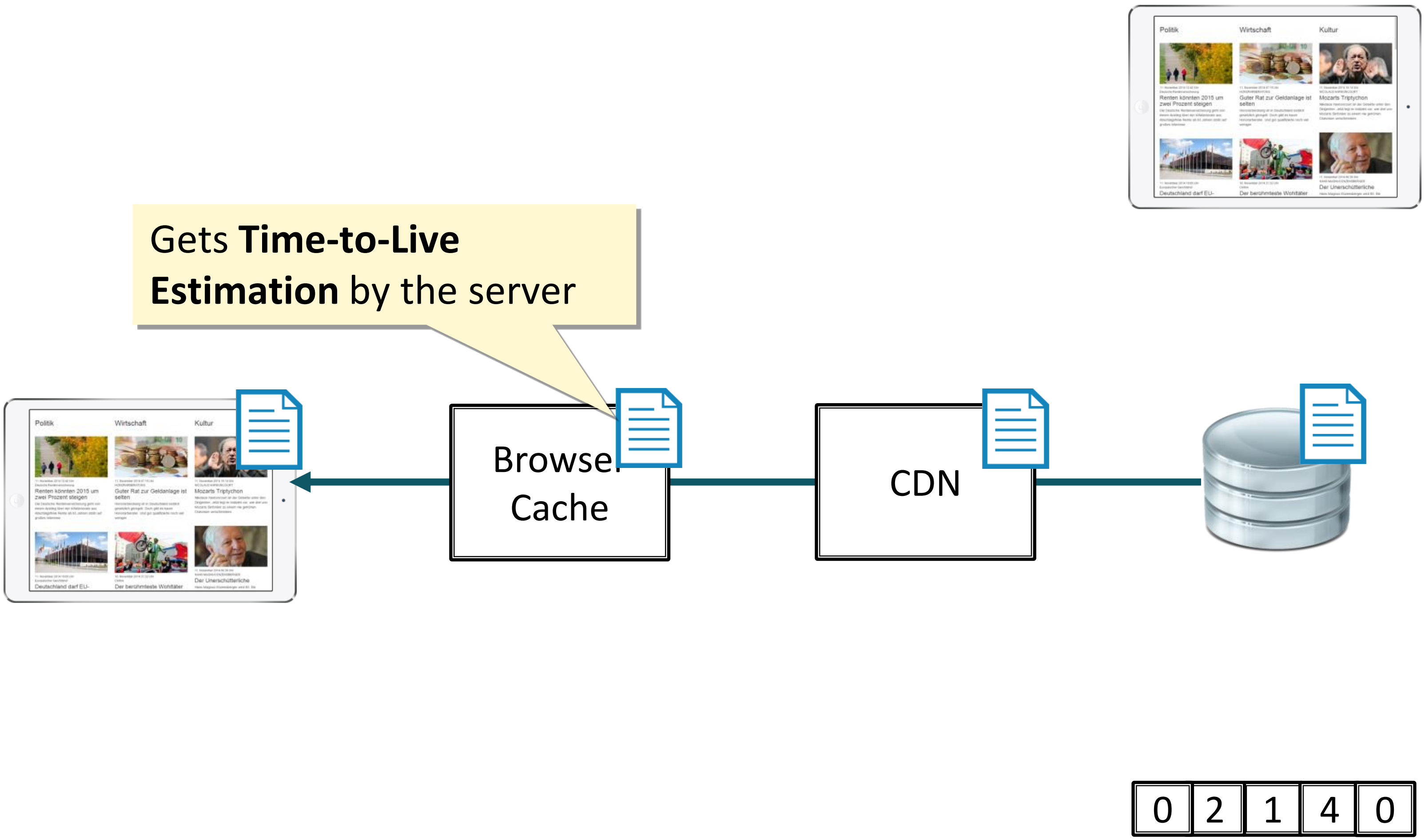
Bloom filters for Caching

End-to-End Example



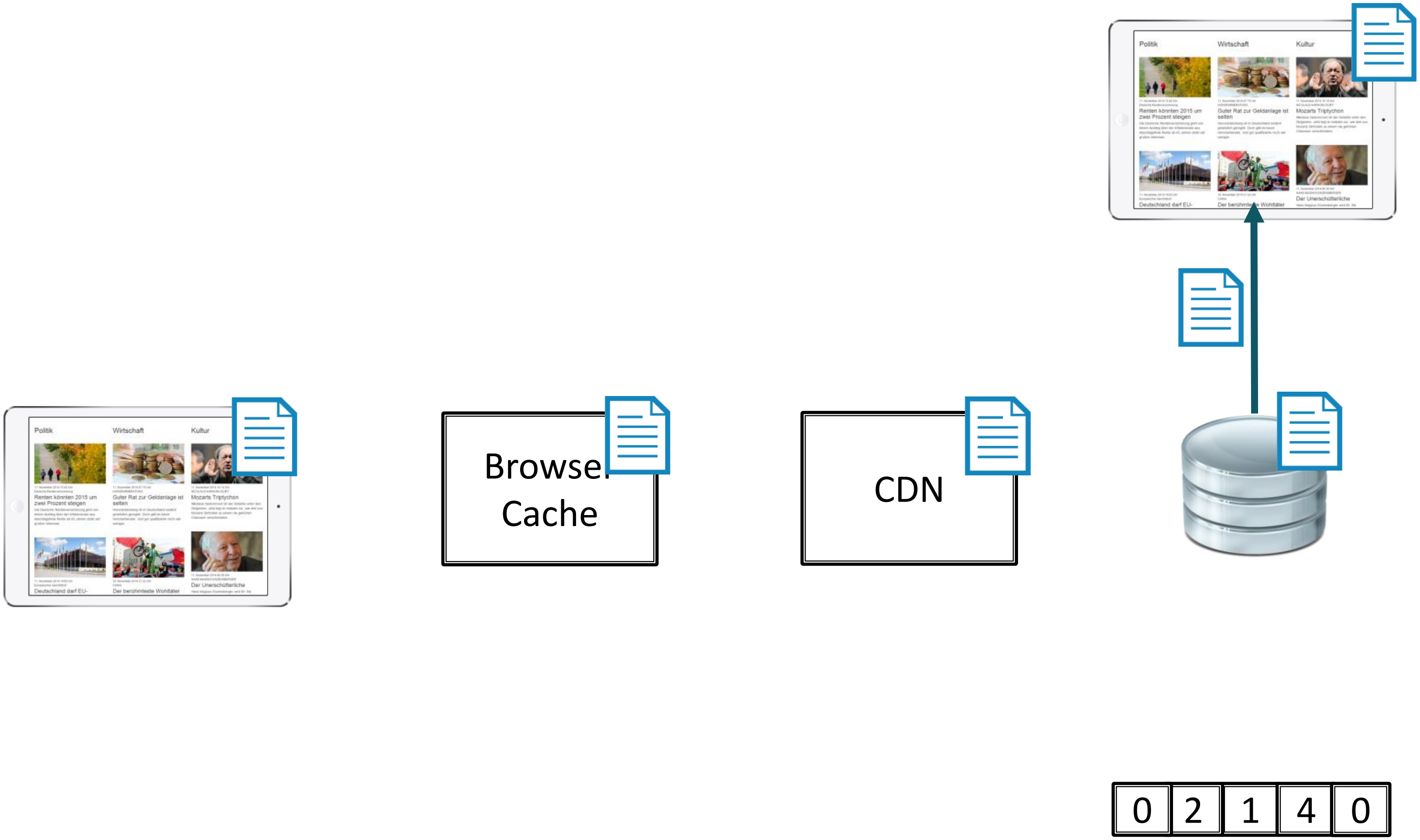
Bloom filters for Caching

End-to-End Example



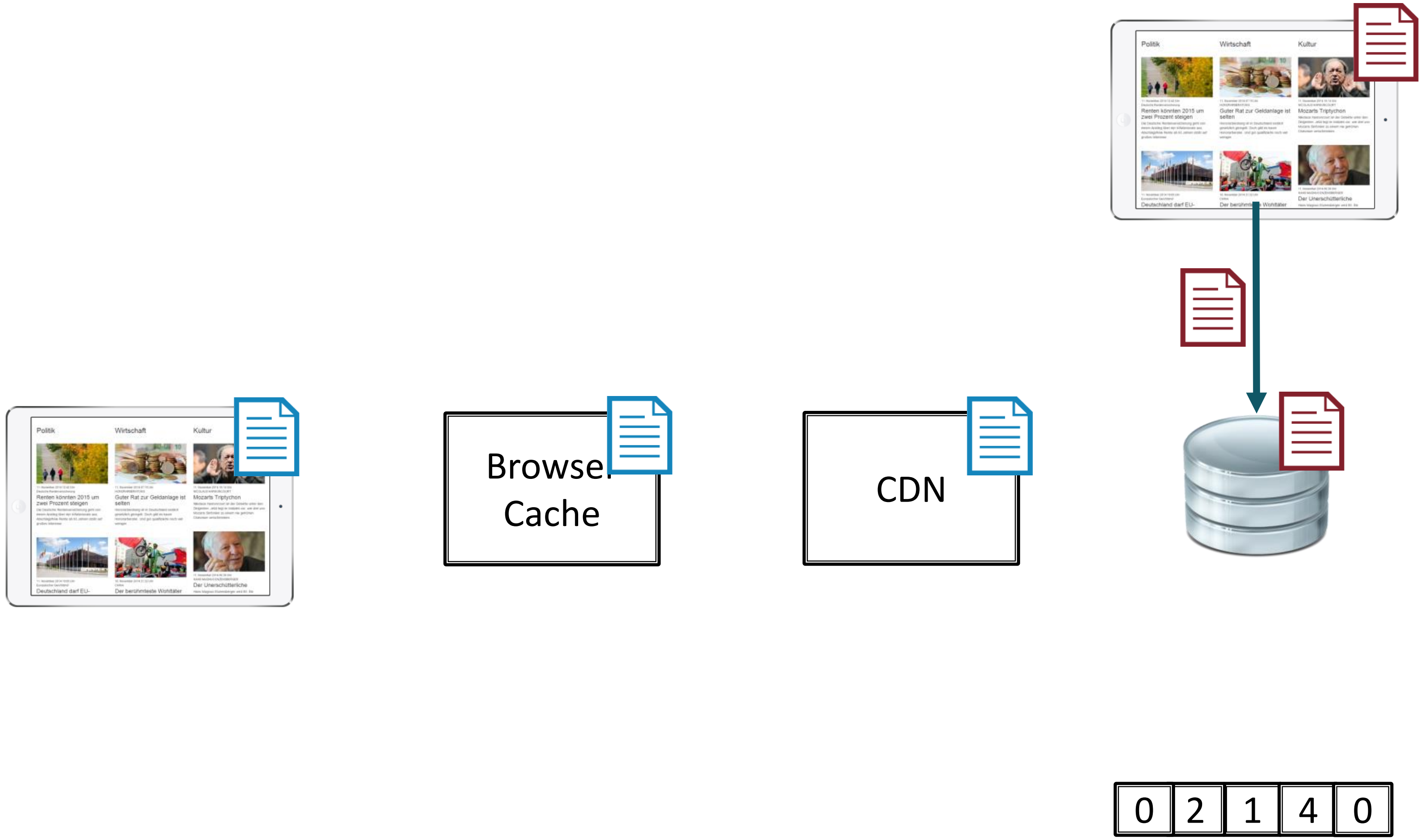
Bloom filters for Caching

End-to-End Example



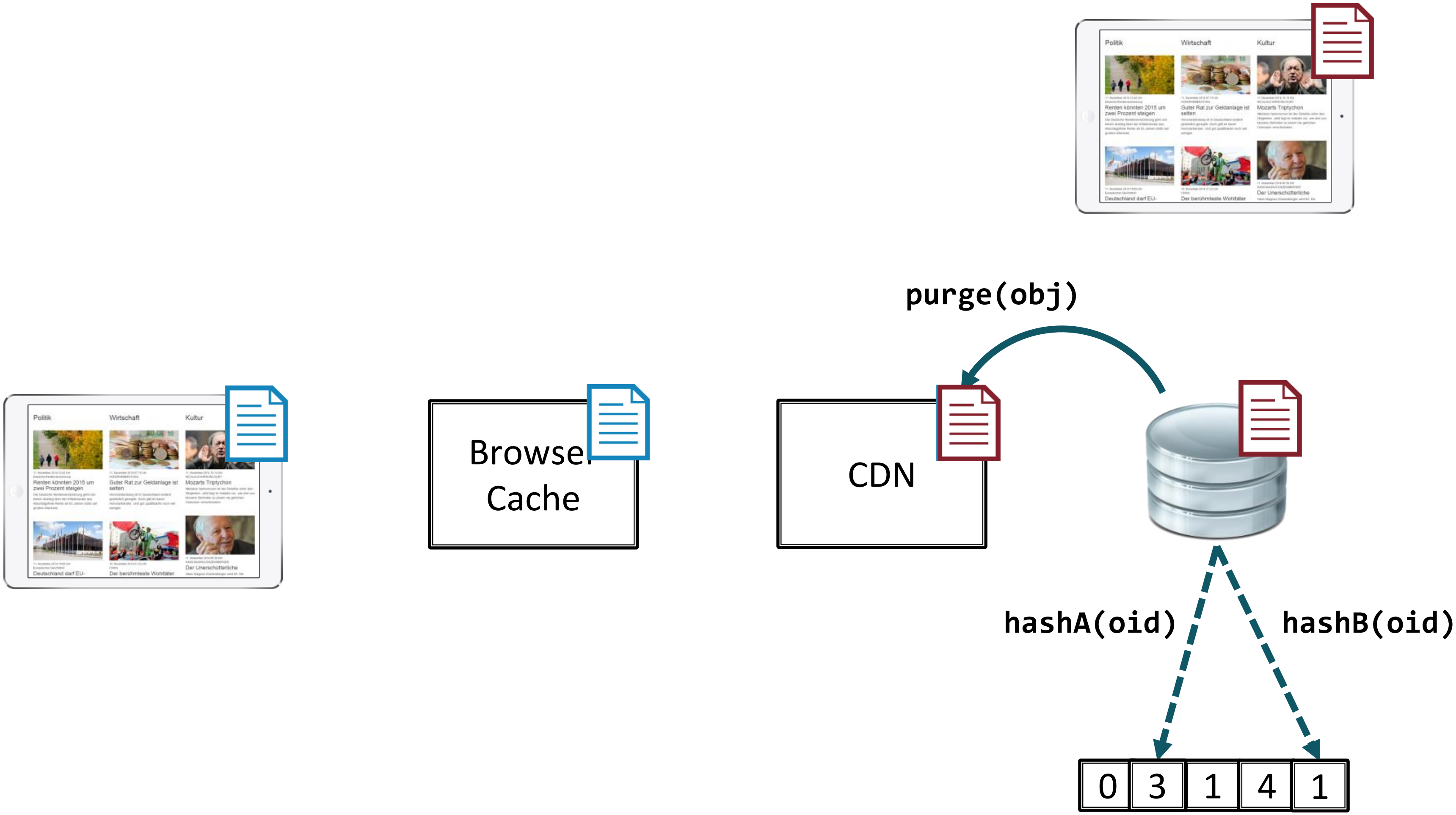
Bloom filters for Caching

End-to-End Example



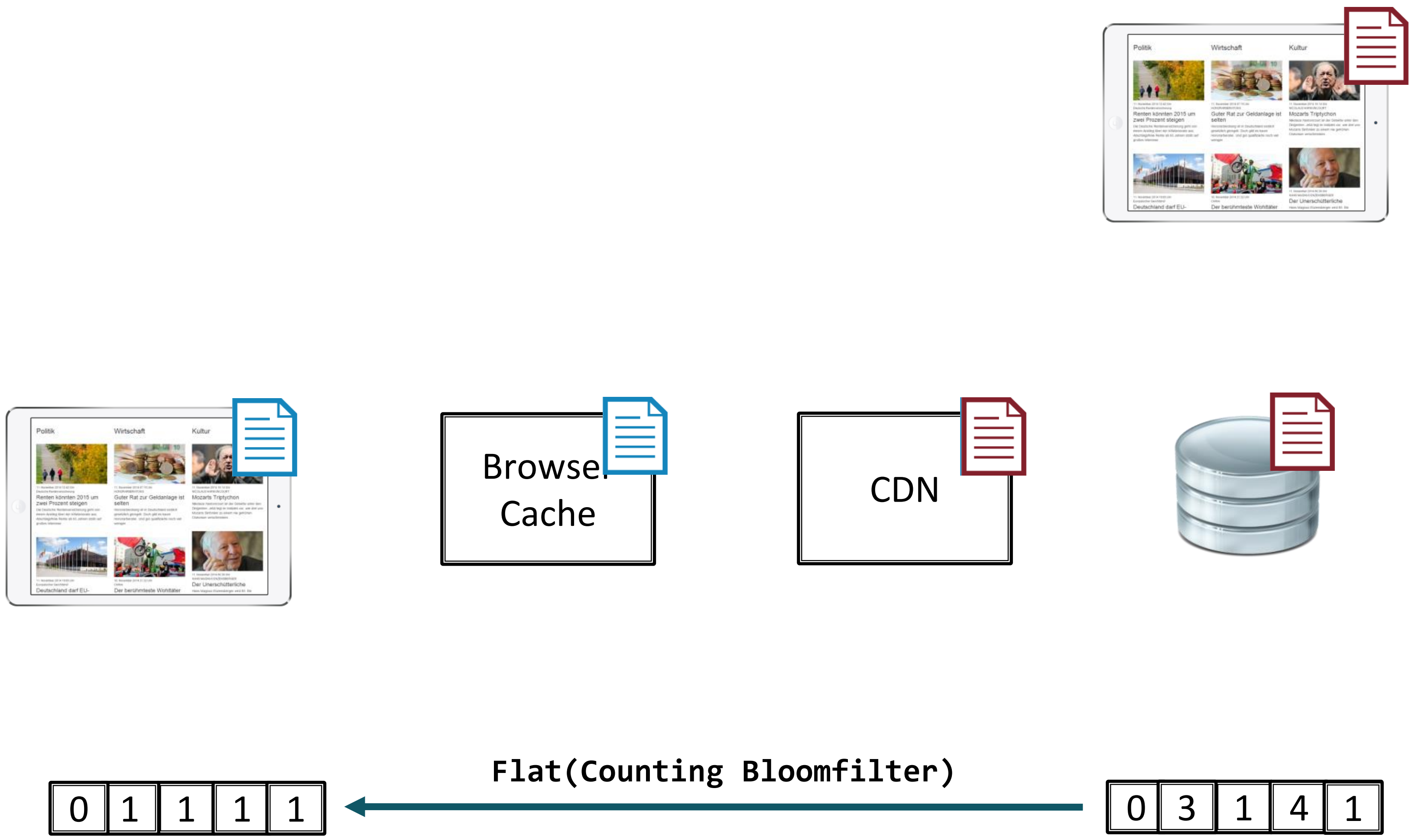
Bloom filters for Caching

End-to-End Example



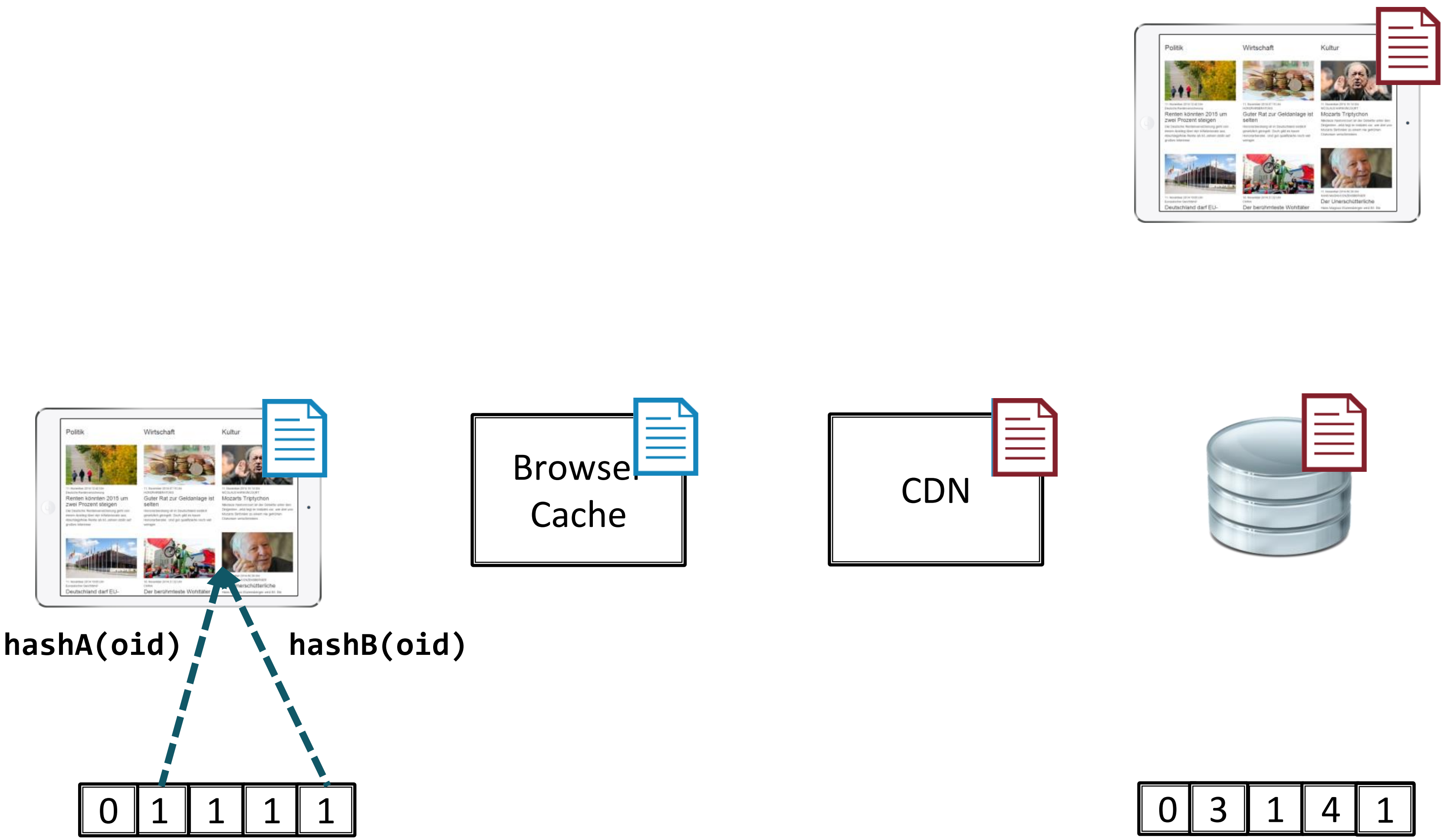
Bloom filters for Caching

End-to-End Example



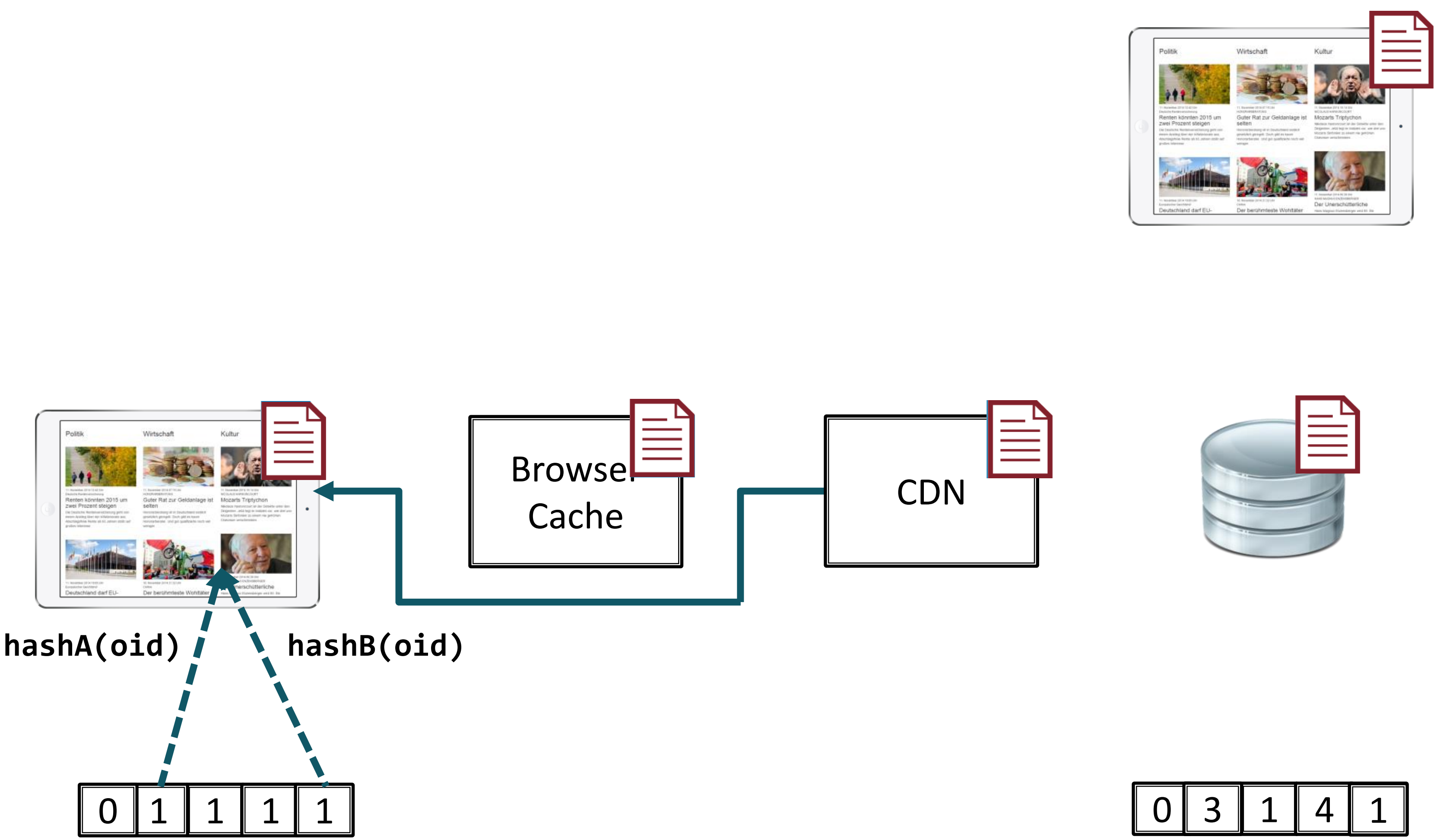
Bloom filters for Caching

End-to-End Example



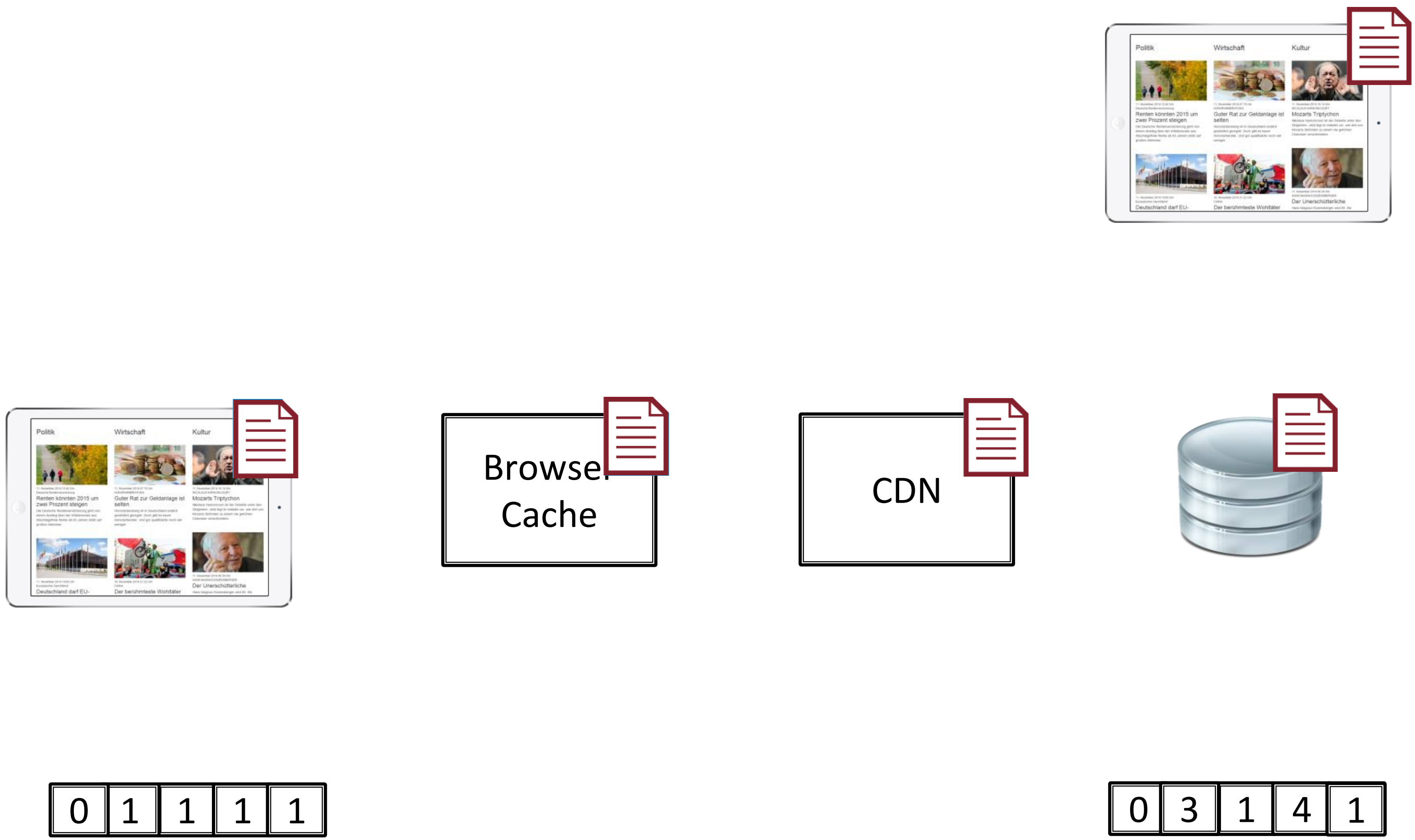
Bloom filters for Caching

End-to-End Example



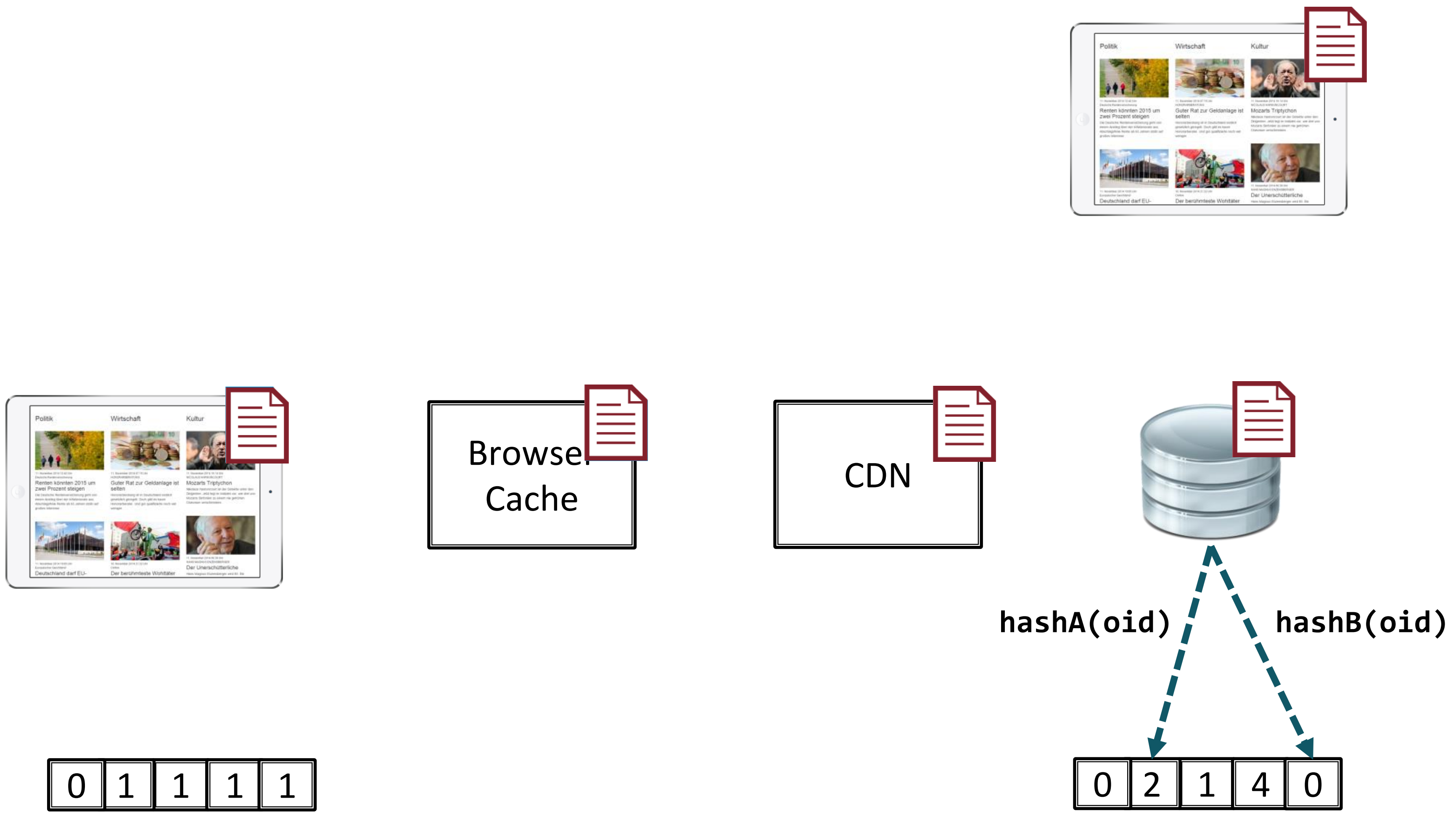
Bloom filters for Caching

End-to-End Example



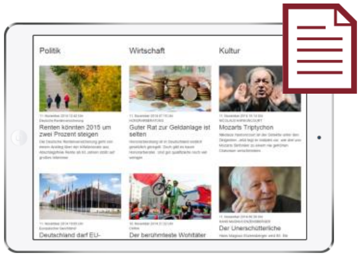
Bloom filters for Caching

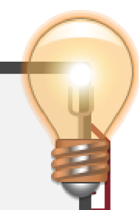
End-to-End Example



Bloom filters for Caching

End-to-End Example





False-Positive Rate: $f \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$

Hash-Functions: $k = \left\lceil \ln(2) \cdot \left(\frac{n}{m}\right) \right\rceil$

With 20.000 distinct updates and 5% error rate: **11 Kbyte**

Consistency Guarantees: Δ-Atomicity, Read-Your-Writes, Monotonic Reads, Monotonic Writes, Causal Consistency

hashB(oid)

0	1	1	1	1

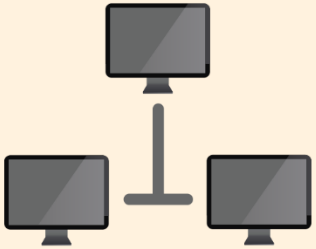
0	2	1	4	0

Baqend: Core Features

>250%

Faster Loads

Automatic Scaling



Faster Development

#1 Users are less annoyed and less annoying.

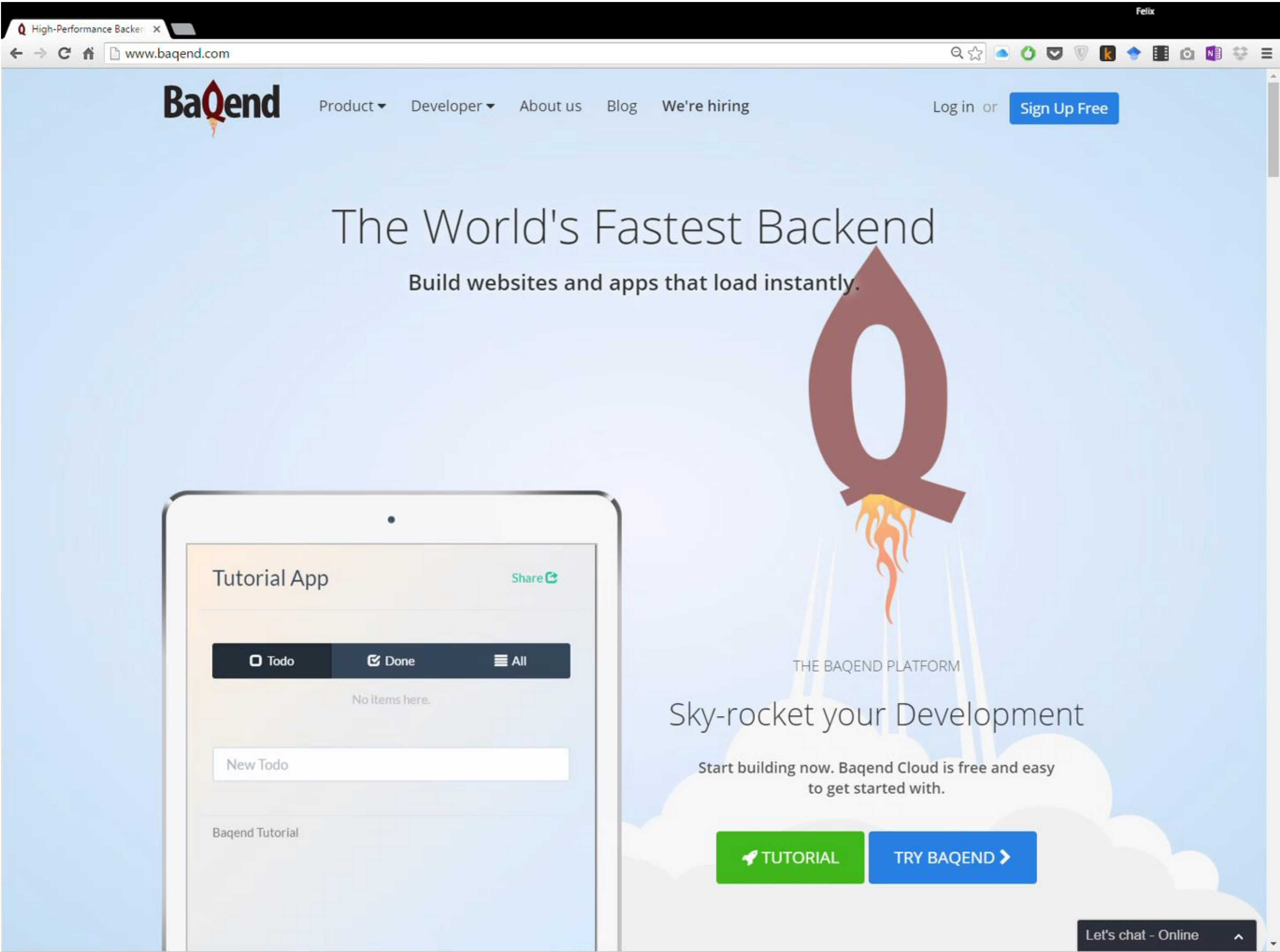
#2 The admin does not look as grim and angry as usual.

#3 The nerds have time to catch some fresh air.

Sources

http://de.slideshare.net/felixgessert/talk-cache-sketches-using-bloom-filters-and-web-caching-against-slow-load-times

http://www.baqend.com/paper/clouddb.pdf



Felix
High-Performance Backen
www.baqend.com
BaQend
Product
Developer
About us
Blog
We're hiring
Log in or
Sign Up Free
The World's Fastest Backend
Build websites and apps that load instantly.
Tutorial App
Share
Todo
Done
All
No items here.
New Todo
Baqend Tutorial
THE BAQEND PLATFORM
Sky-rocket your Development
Start building now. Baqend Cloud is free and easy to get started with.
TUTORIAL
TRY BAQEND
Let's chat - Online

Want to try Baqend?

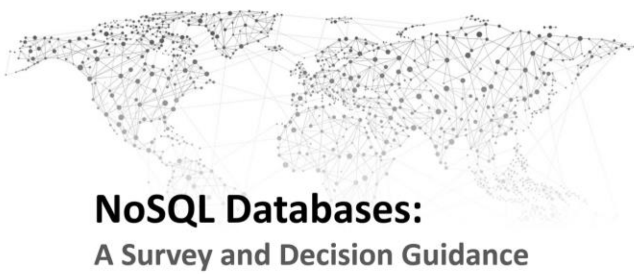
- Free **Baqend Cloud** Instance at baqend.com
- Download **Community Edition**

Literature Recommendations

Recommended Literature

NoSQL Databases: a Survey and Decision Guidance

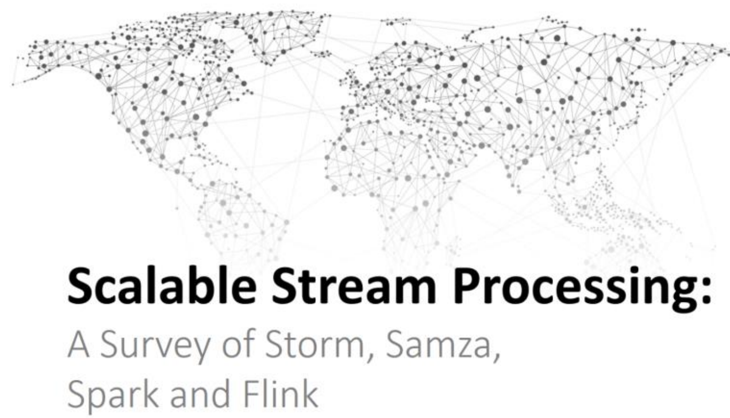
Together with our colleagues at the University of Hamburg, we—that is Felix Gessert, Wolfram Wingerath, Steffen Friedrich and Norbert Ritter—presented an overview over the NoSQL landscape at SummerSOC'16 last month. Here is the written gist. We give our best to convey the condensed NoSQL knowledge we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



TL;DR

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.

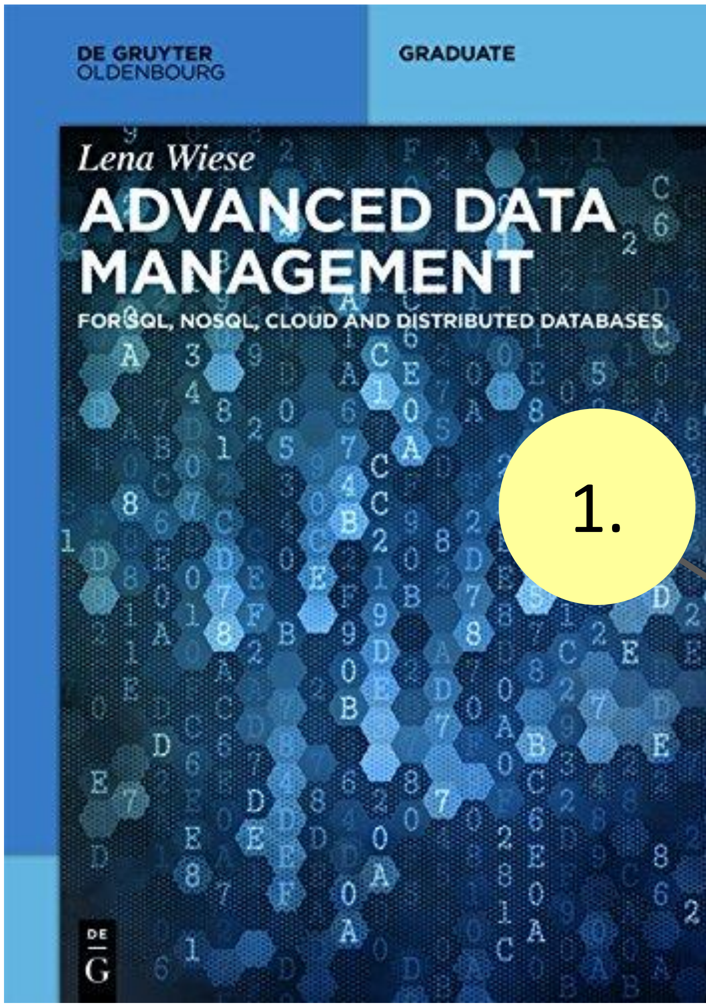
Scalable Stream Processing: A Survey of Storm, Samza, Spark and Flink

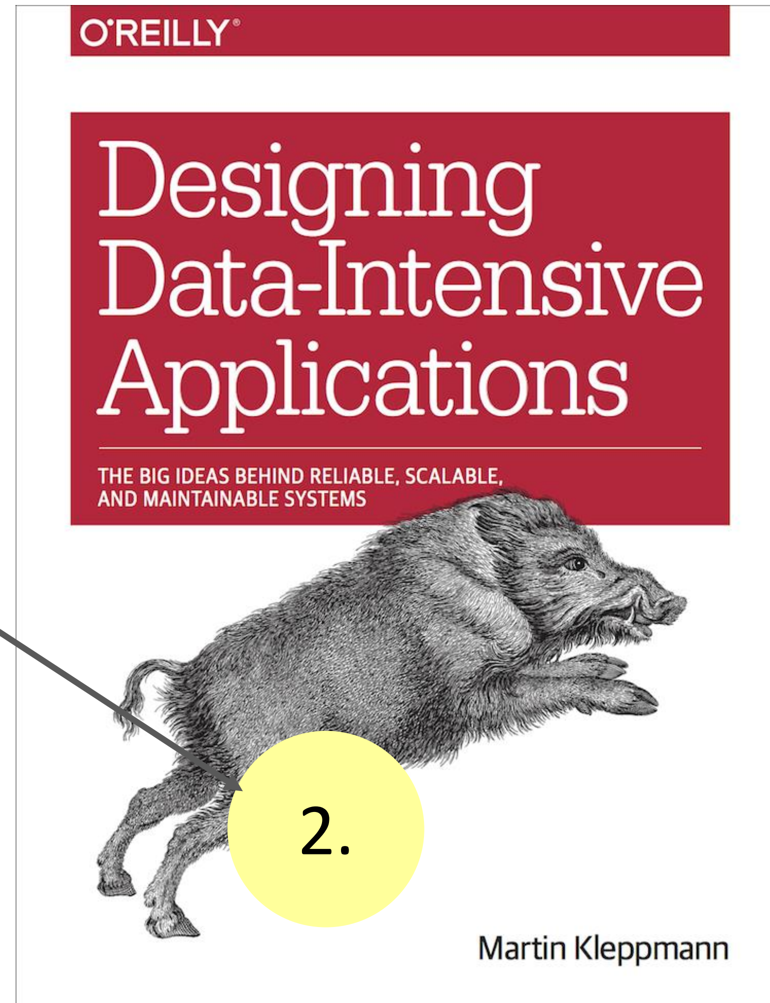


With this article, we would like to share our insights on real-time data processing we gained building Baqend. This is an updated version of our most recent stream processor survey which is another cooperation with the University of Hamburg (authors: Wolfram Wingerath, Felix Gessert, Steffen Friedrich and Norbert Ritter). As you may or may not have been aware of, a lot of stream processing is going on behind the curtains at Baqend. In our quest to provide the lowest-possible latency, we have built a system to enable **query caching** and **real-time notifications** (similar to *changefeeds* in RethinkDB/Horizon) and hence learned a lot about the competition in the field of stream processors.

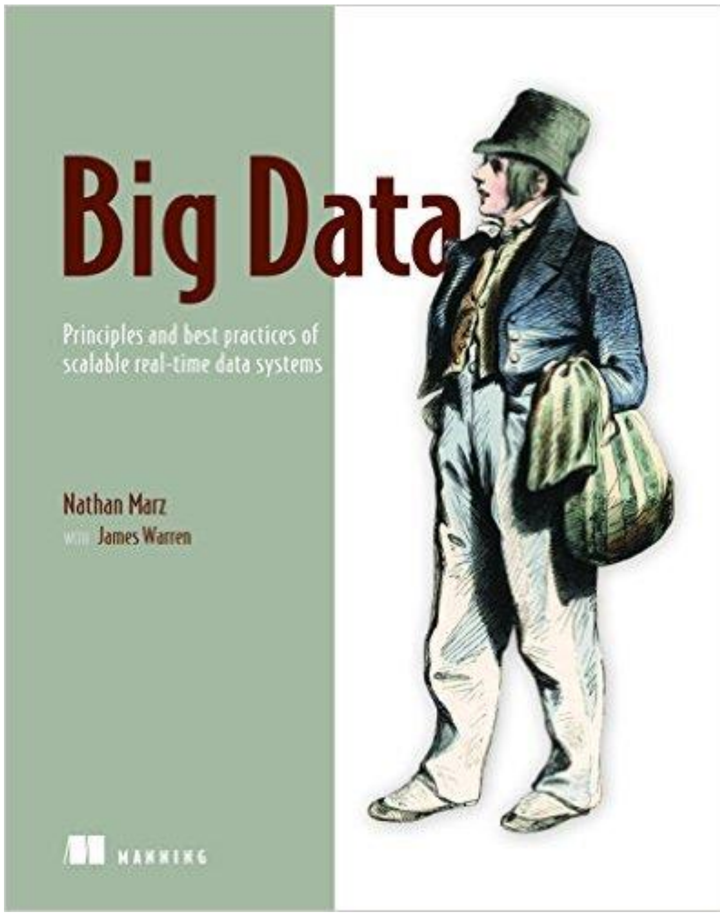
Read them at blog.baqend.com!

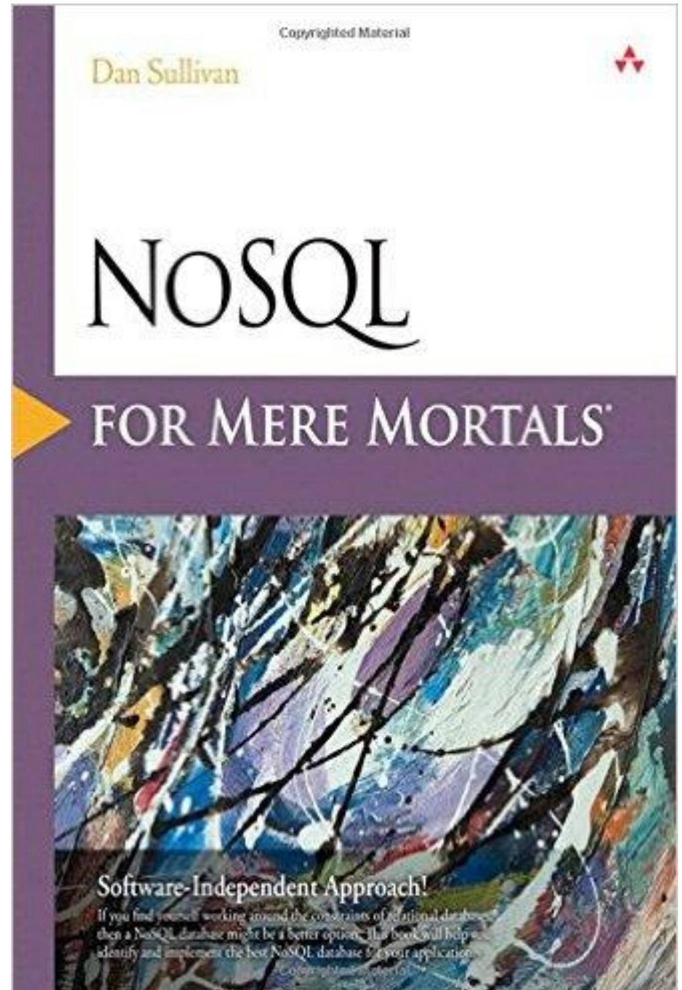
Recommended Literature



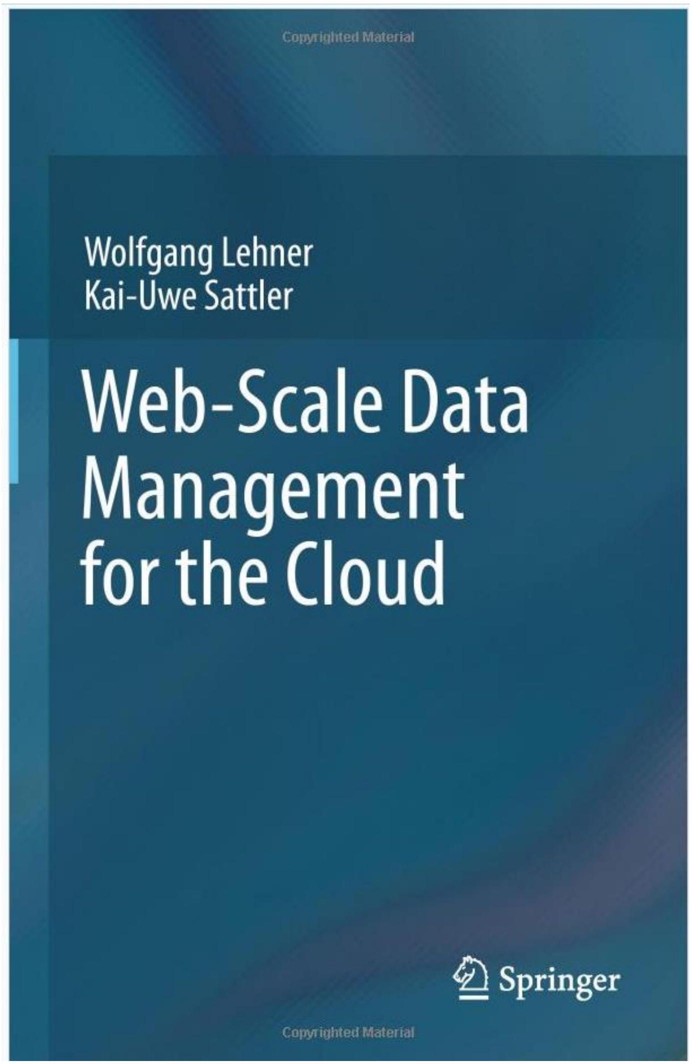


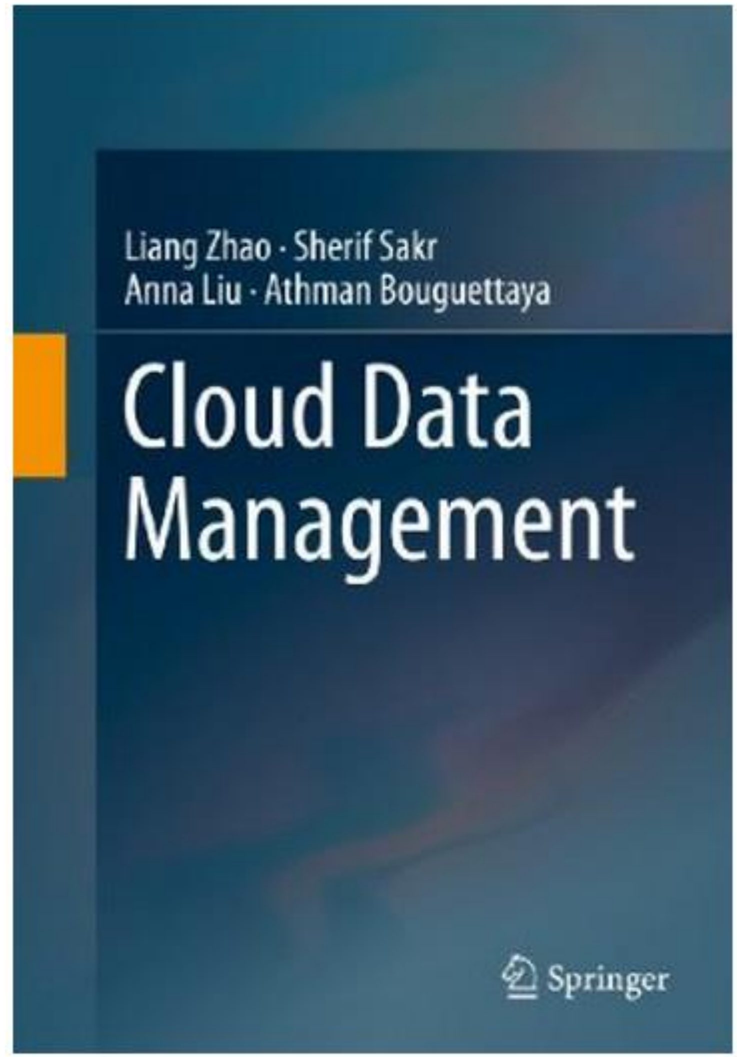
Recommended Literature





Recommended Literature: Cloud-DBs





Recommended Literature: Blogs

BaQend

http://medium.baqend.com/

DZone InfoQ

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

Aphyr

https://aphyr.com/

Metadata

http://muratbuffalo.blogspot.de/

NoSQL Weekly

http://www.nosqlweekly.com/

Martin Kleppmann

https://martin.kleppmann.com/

High Scalability

http://highscalability.com/

DB-ENGINES

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

Seminal NoSQL Papers

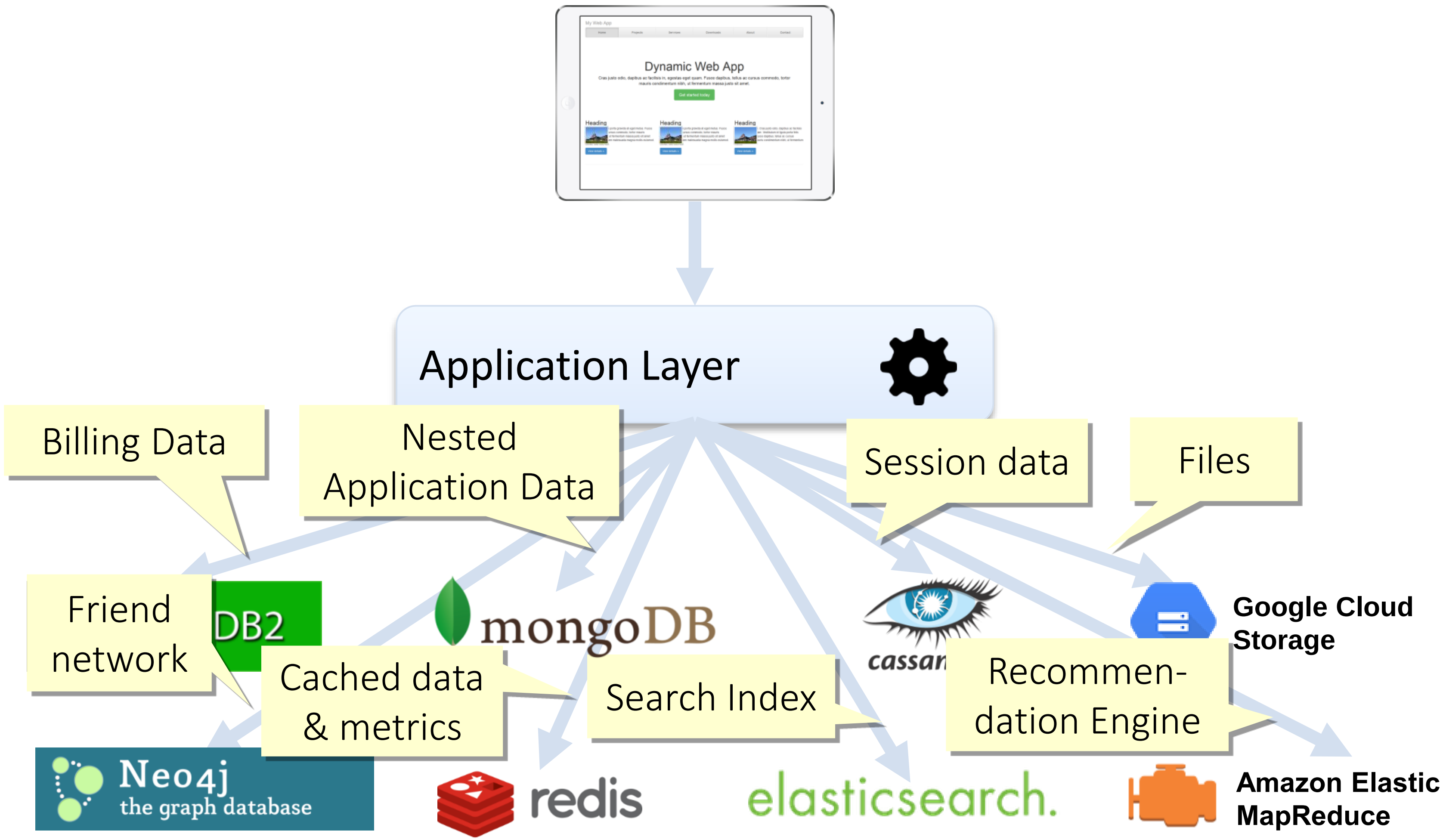
- Lamport, Leslie. **Paxos made simple.**, SIGACT News, 2001
- S. Gilbert, et al., **Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services**, SIGACT News, 2002
- F. Chang, et al., **Bigtable: A Distributed Storage System For Structured Data**, OSDI, 2006
-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

Thank you – questions?

Norbert Ritter, Felix Gessert, Wolfram Wingerath
{ritter,gessert,wingerath}@informatik.uni-hamburg.de

Polyglot Persistence

Current best practice

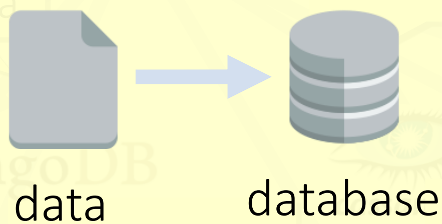


Polyglot Persistence

Current best practice

Research Question:

Can we automate the

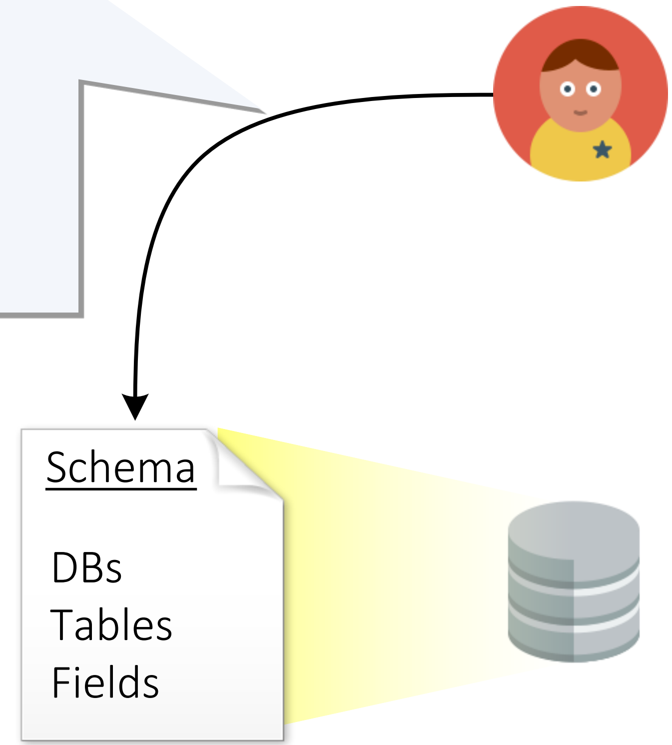


mapping problem?

Vision

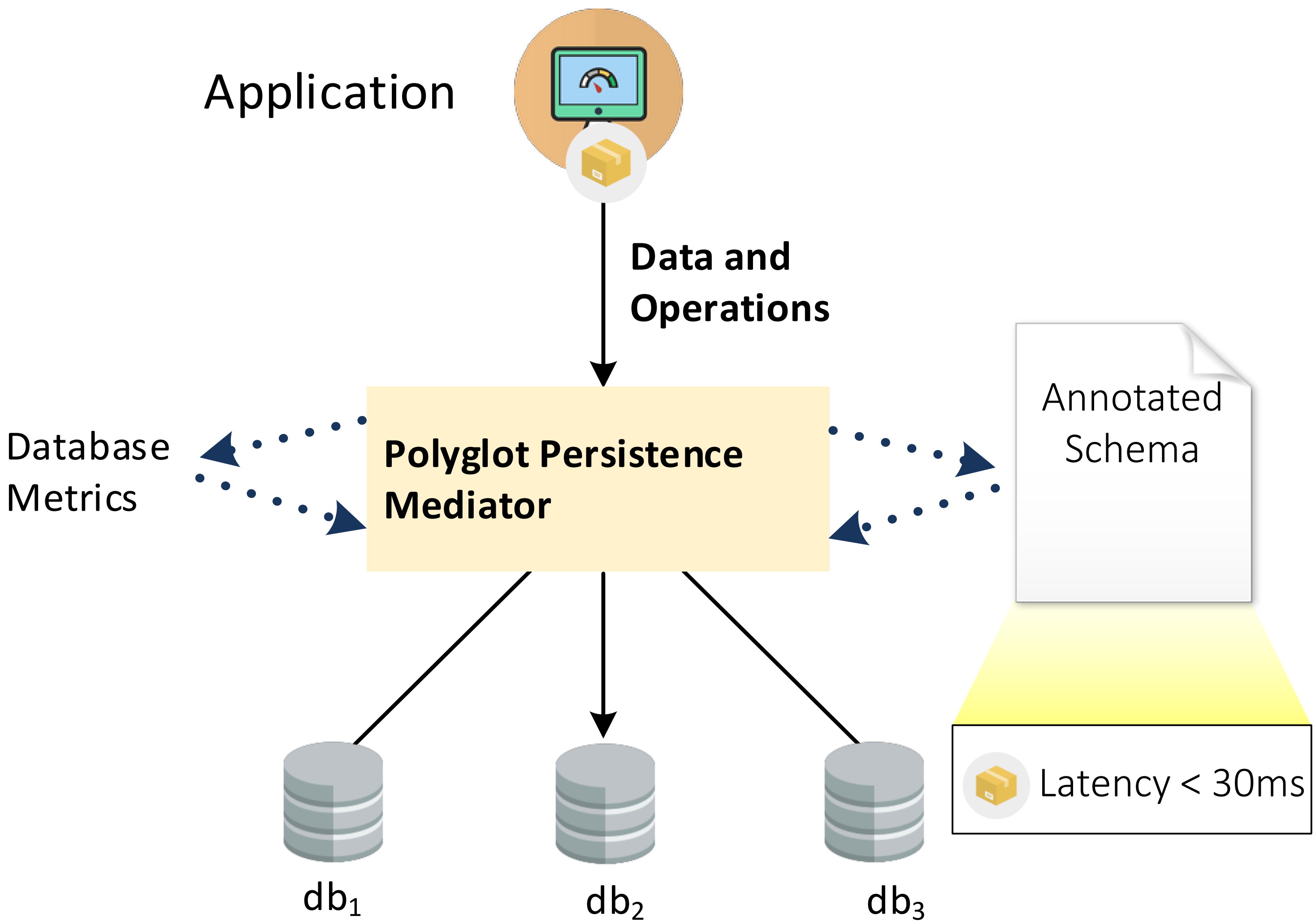
Schemas can be annotated with requirements

- Write Throughput > 10,000 RPS
- Read Availability > 99.9999%
- Scans = true
- Full-Text-Search = true
- Monotonic Read = true



Vision

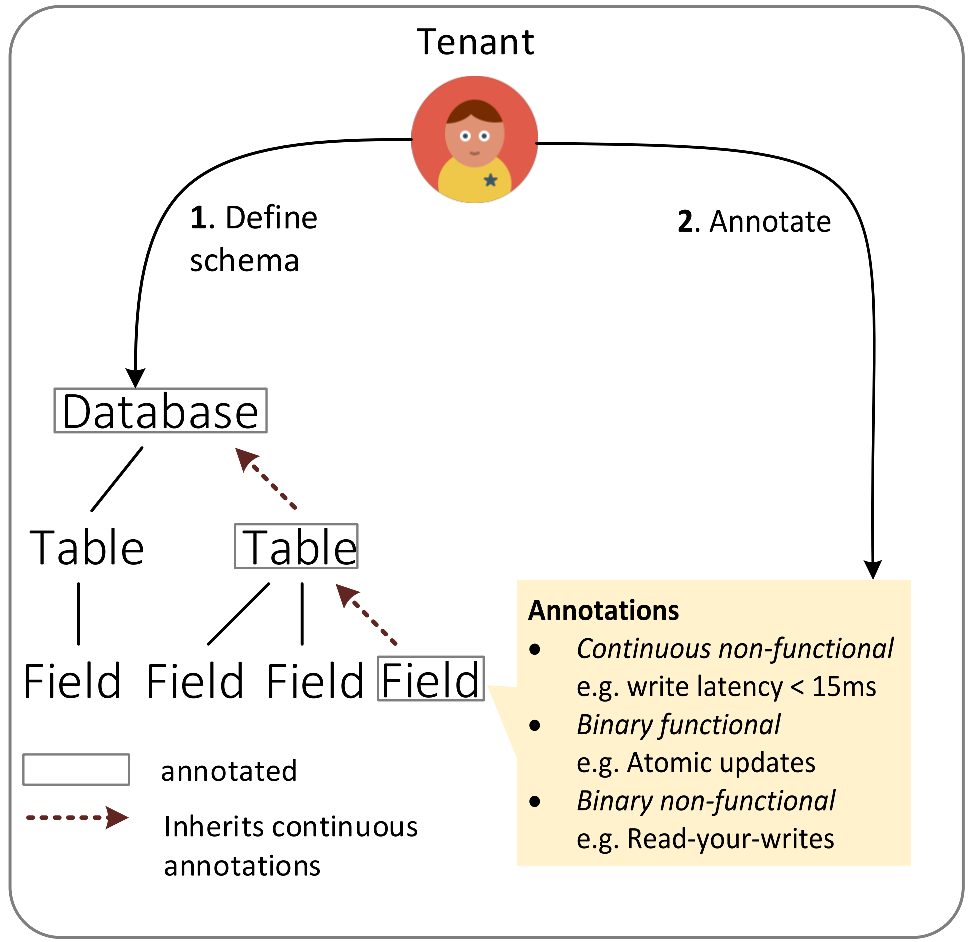
The Polyglot Persistence Mediator chooses the database



Step I - Requirements

Expressing the application's needs

- Tenant annotates schema with his requirements

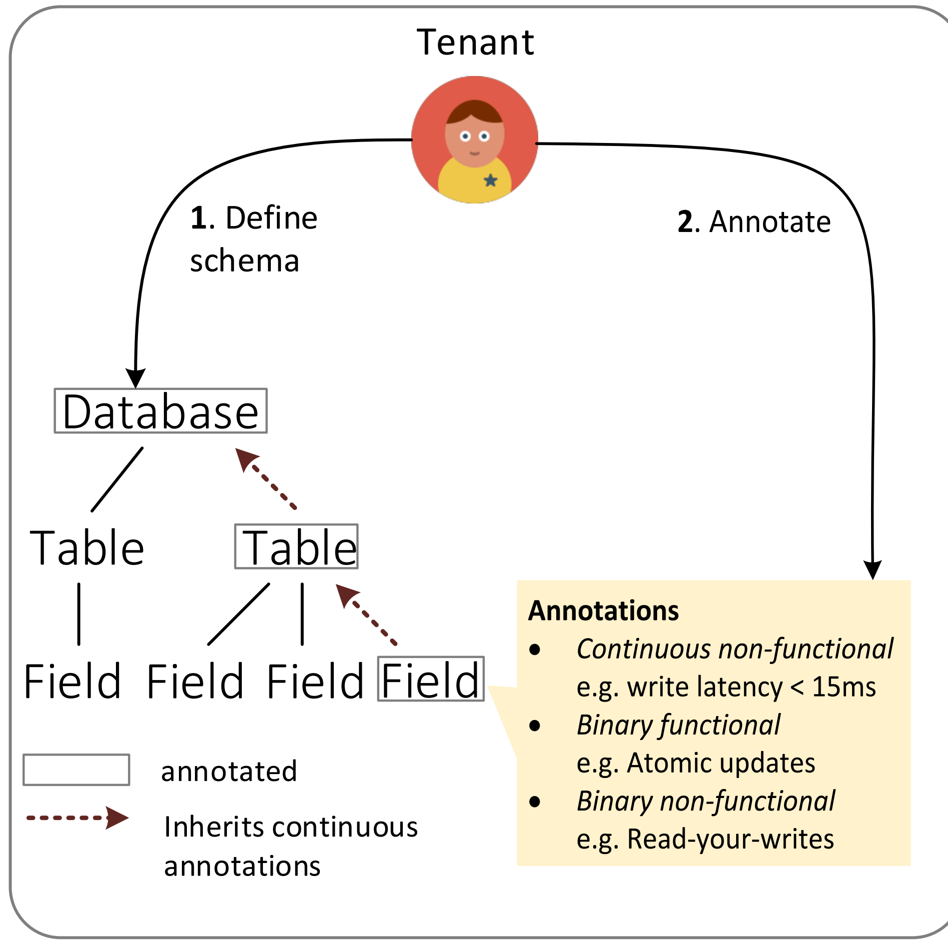


1 Requirements

Step I - Requirements

Expressing the application's needs

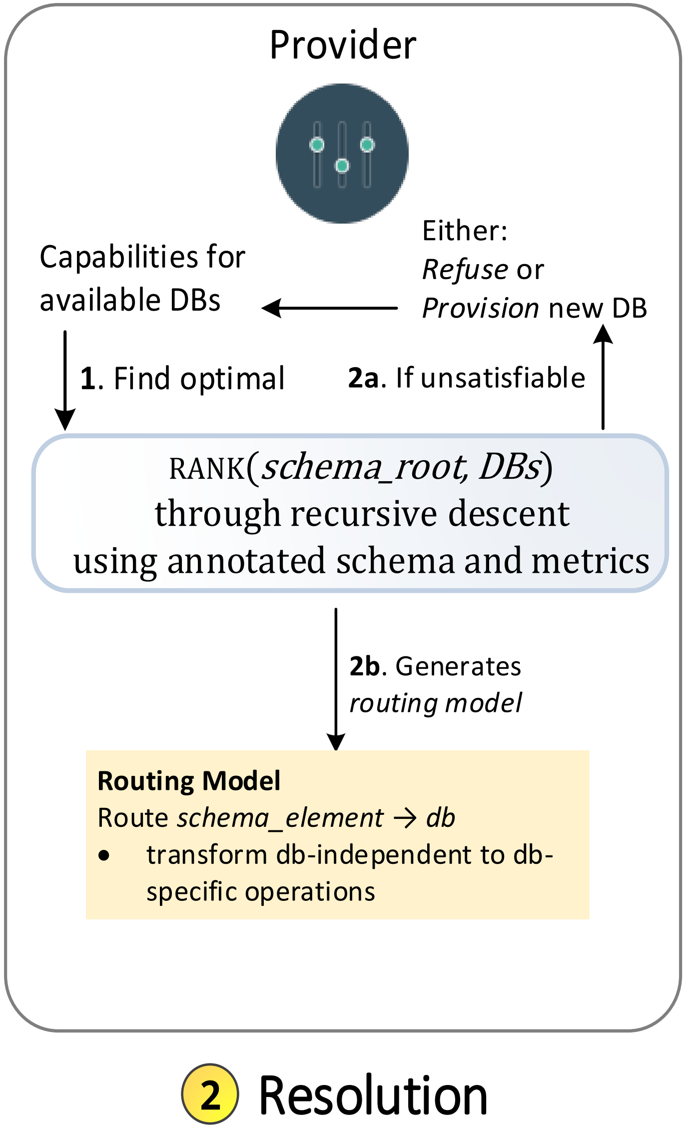
Annotation	Type	Annotated at
Read Availability	Continuous	*
Write Availability	Continuous	*
Read Latency	Continuous	*
Write Latency	Continuous	*
Write Throughput	Continuous	*
Data Vol. Scalability	Non-Functional	Field/Class/DB
Write Scalability	Non-Functional	Field/Class/DB
Read Scalabilty	Non-Functional	Field/Class/DB
Elasticity	Non-Functional	Field/Class/DB
Durability	Non-Functional	Field/Class/DB
Replicated	Non-Functional	Field/Class/DB
Linearizability	Non-Functional	Field/Class
Read-your-Writes	Non-Functional	Field/Class
Causal Consistency	Non-Functional	Field/Class
Writes follow reads	Non-Functional	Field/Class
Monotonic Read	Non-Functional	Field/Class
Monotonic Write	Non-Functional	Field/Class
Scans	Functional	Field
Sorting	Functional	Field
Range Queries	Functional	Field
Point Lookups	Functional	Field
ACID Transactions	Functional	Class/DB
Conditional Updates	Functional	Field
Joins	Functional	Class/DB
Analytics Integration	Functional	Field/Class/DB
Fulltext Search	Functional	Field
Atomic Updates	Functional	Field/Class



1 Requirements

Step II - Resolution
Finding the best database

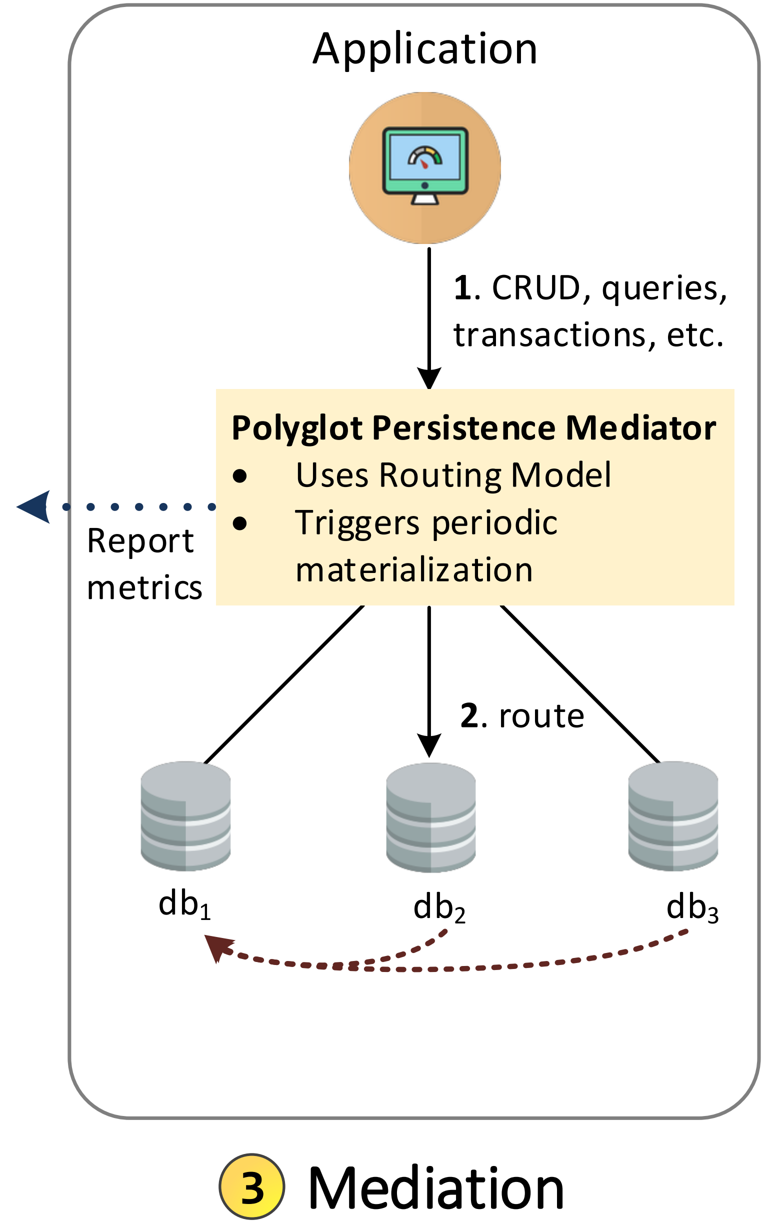
- The Provider resolves the requirements
- RANK: scores available database systems
- **Routing Model**: defines the optimal mapping from schema elements to databases



Step III - Mediation

Routing data and operations

- The PPM routes data
- **Operation Rewriting:** translates from abstract to database-specific operations
- **Runtime Metrics**: Latency, availability, etc. are reported to the resolver
- **Primary Database Option**: All data periodically gets materialized to designated database



Evaluation: News Article

Prototype of Polyglot Persistence Mediator in ORESTES

Scenario: news articles with impression counts
Objectives: low-latency top-k queries, high-throughput counts, article-queries

Evaluation: News Article

Prototype built on ORESTES

Scenario: news articles with impression counts
Objectives: low-latency top-k queries, high-throughput counts, article-queries

Counter updates kill performance

Evaluation: News Article

Prototype built on ORESTES

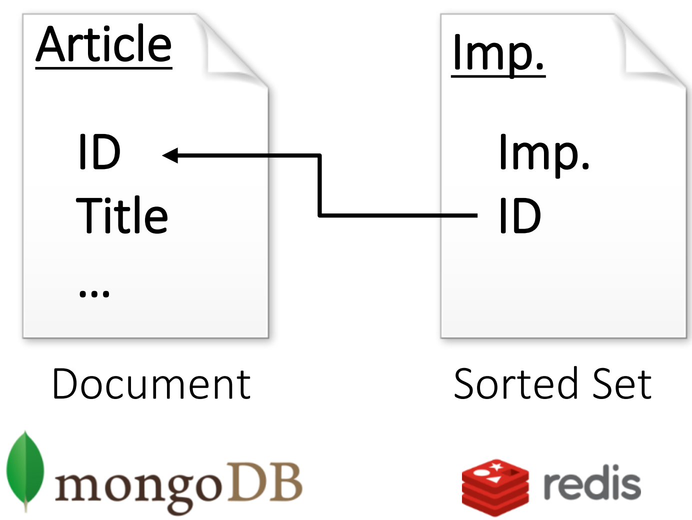
Scenario: news articles with impression counts
Objectives: low-latency top-k queries, high-throughput counts, article-queries

No powerful queries

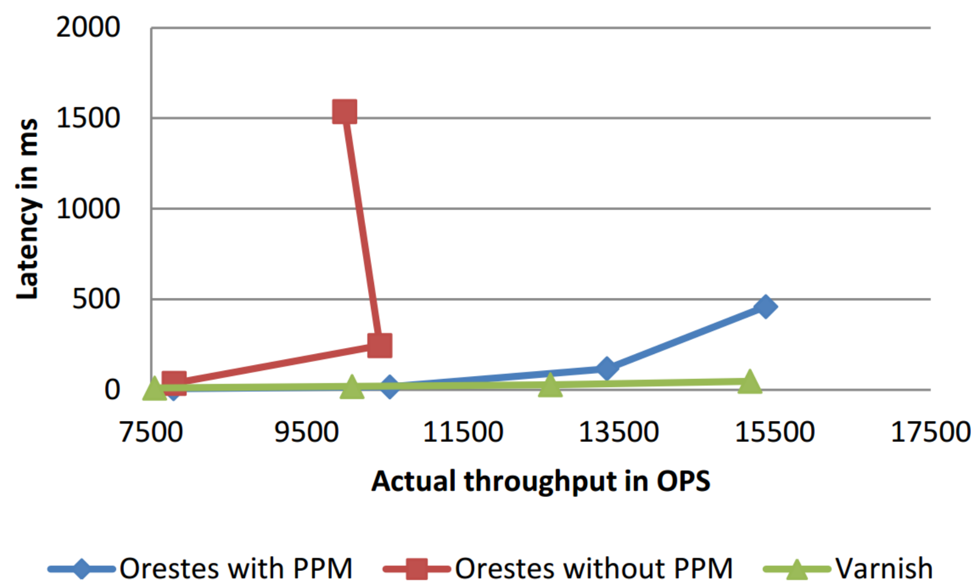
Evaluation: News Article

Prototype built on ORESTES

Scenario: news articles with impression counts
Objectives: low-latency top-k queries, high-throughput counts, article-queries

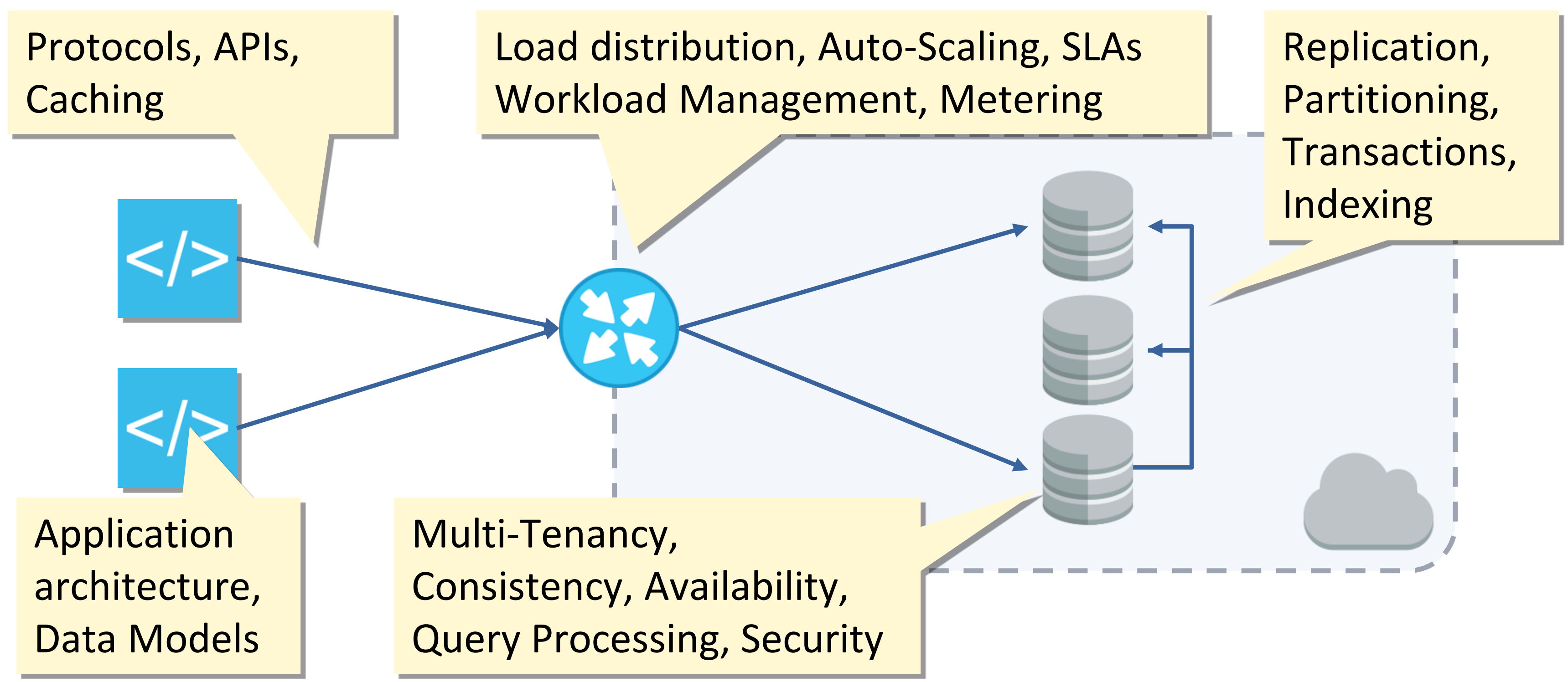


Found Resolution

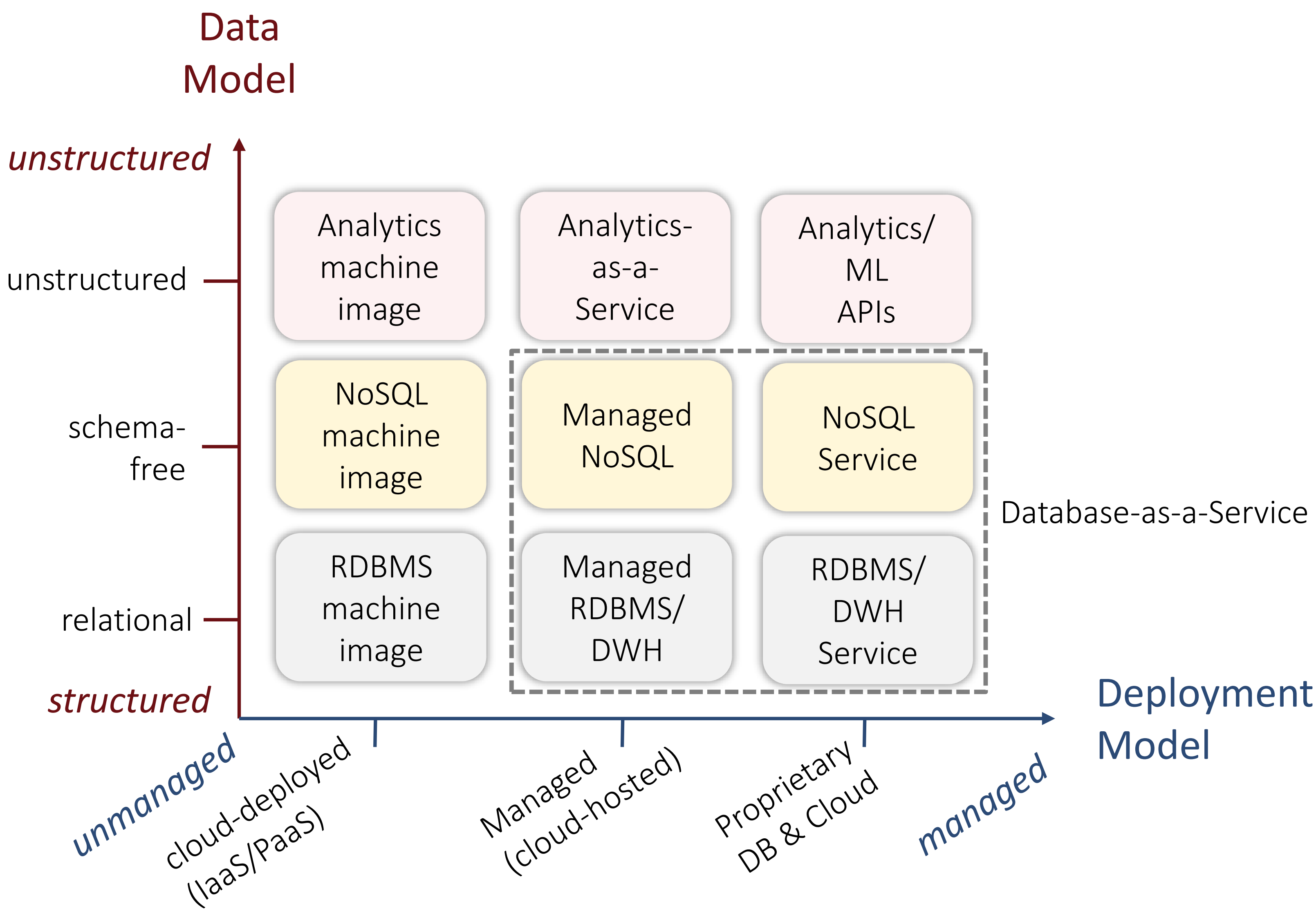


Cloud Data Management

- New field tackling the *design*, *implementation*, *evaluation* and *application implications* of **database systems in cloud environments**:

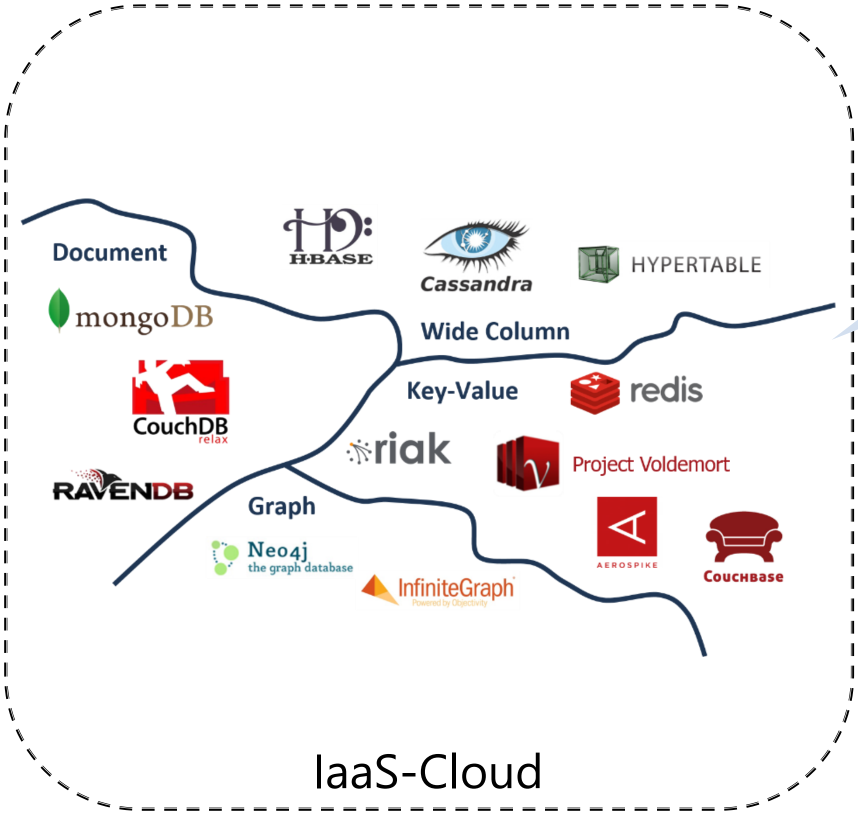


Cloud-Database Models



Cloud-Deployed Database

Database-image provisioned in IaaS/PaaS-cloud



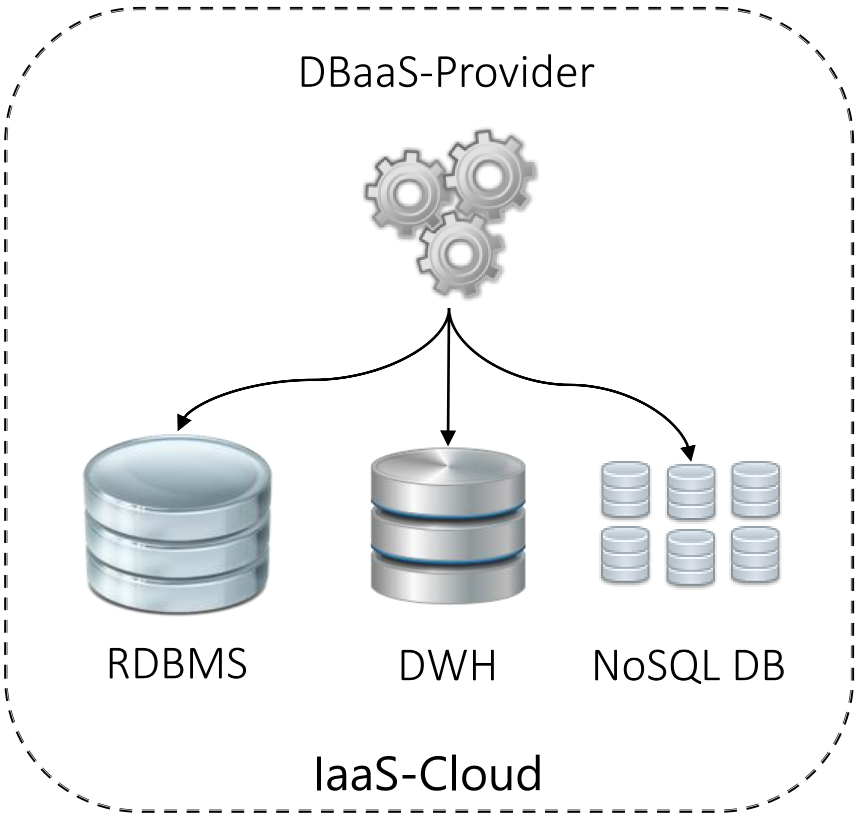
IaaS/PaaS deployment of database system

Does not solve:

Provisioning, Backups, Security, Scaling, Elasticity, Performance Tuning, Failover, Replication, ...

Managed RDBMS/DWH/NoSQL DB

Cloud-hosted database



RDBMS

- Amazon RDS
- SQL Azure
- Clustrix
- EDB ENTERPRISEDB
- Google Cloud SQL
- Heroku Postgres

NoSQL DB

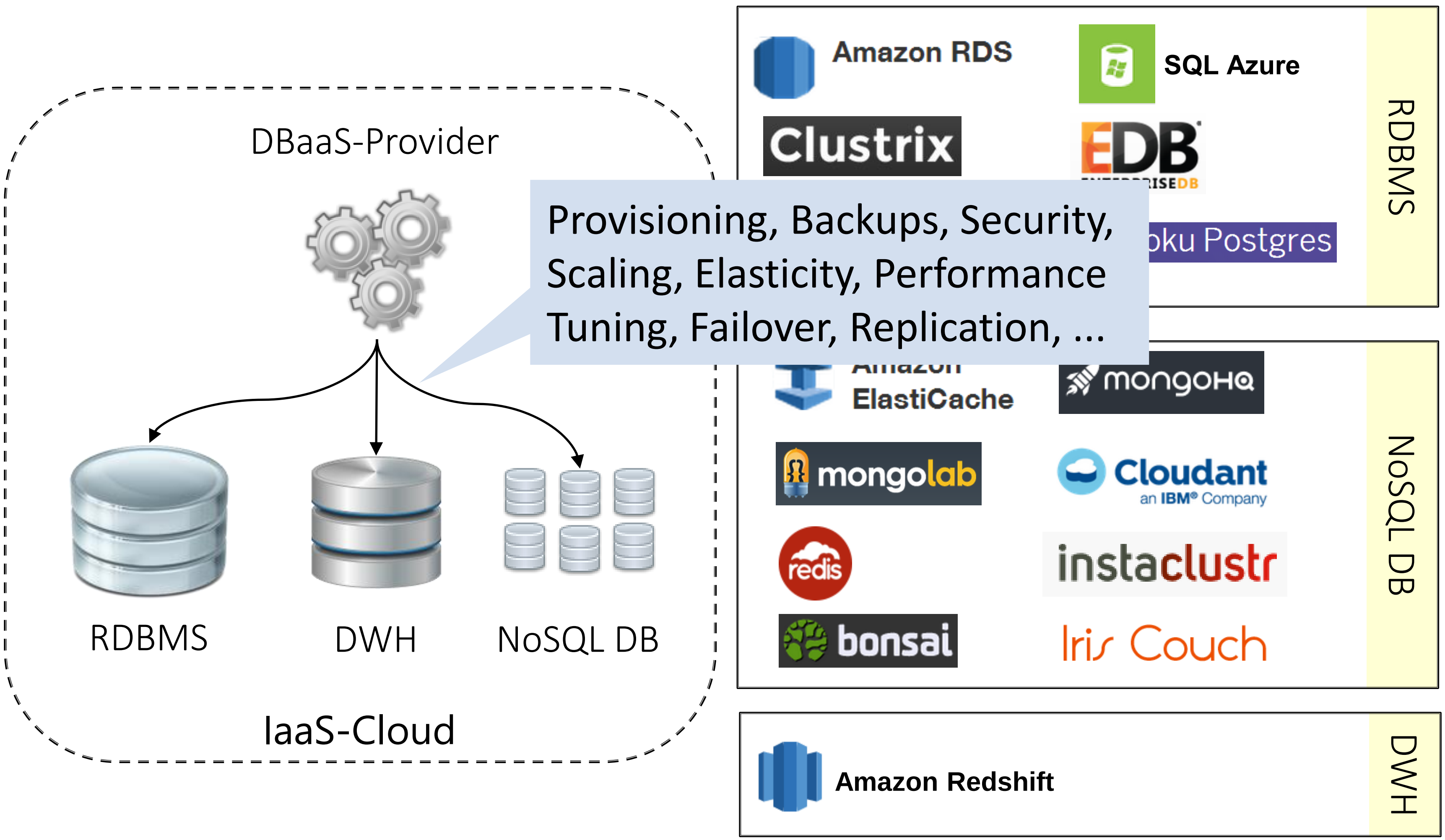
- Amazon ElastiCache
- mongoHQ
- mongolab
- Cloudant an IBM® Company
- redis
- instaclustr
- bonsai
- Iris Couch

DWH

- Amazon Redshift

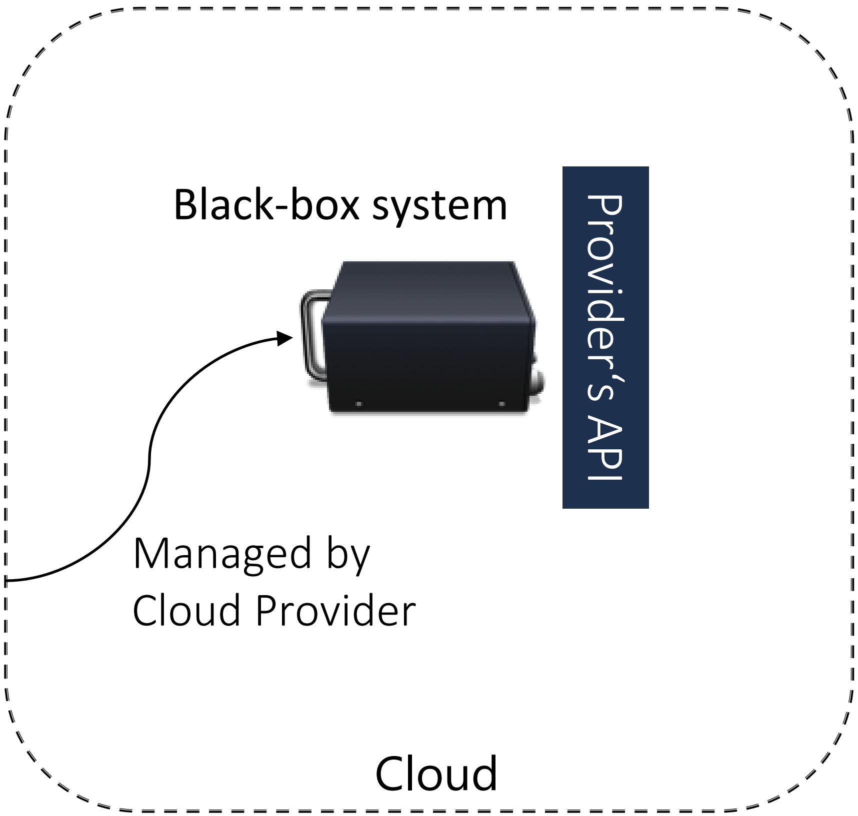
Managed RDBMS/DWH/NoSQL DB

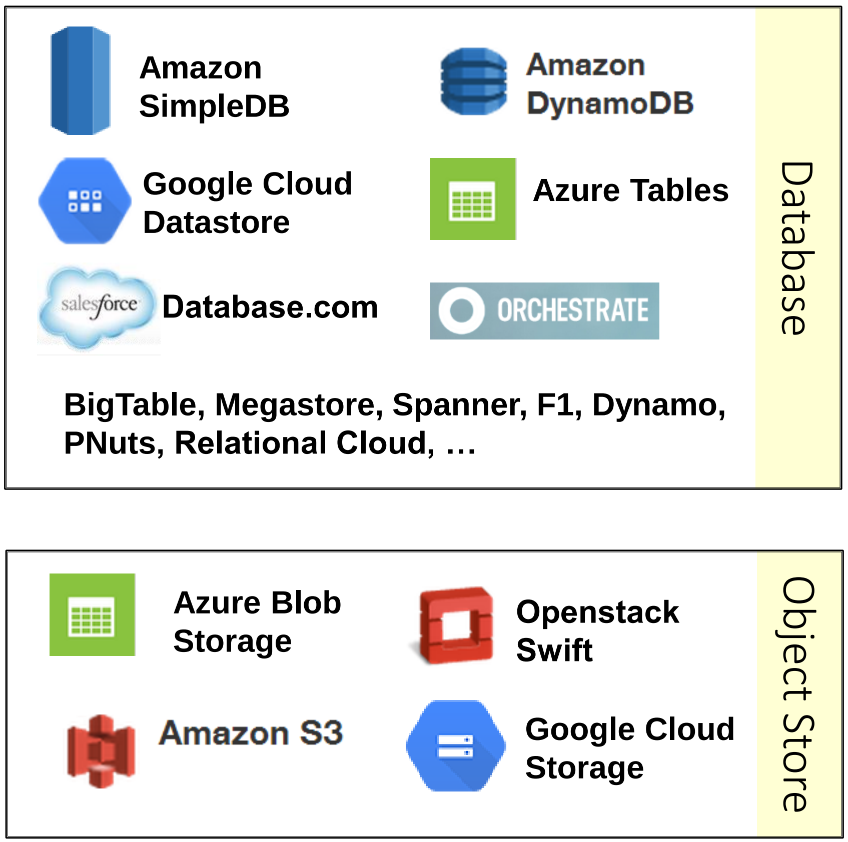
Cloud-hosted database



Proprietary Cloud Database

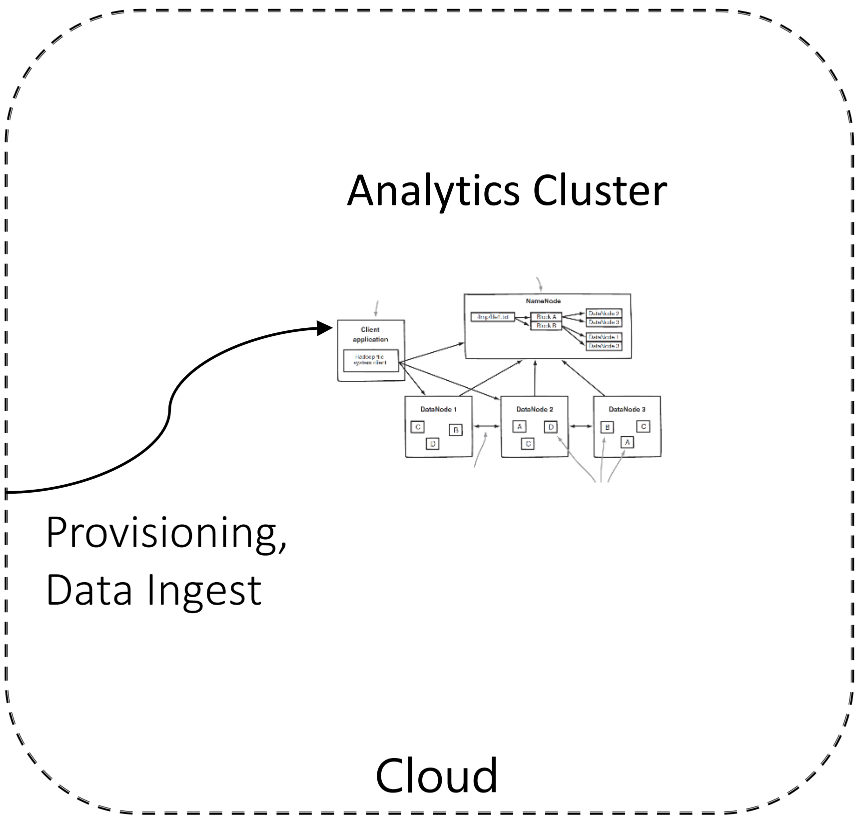
Designed for and deployed in vendor-specific cloud environment





Analytics-as-a-Service

Analytic frameworks and machine learning with service APIs



Analytics

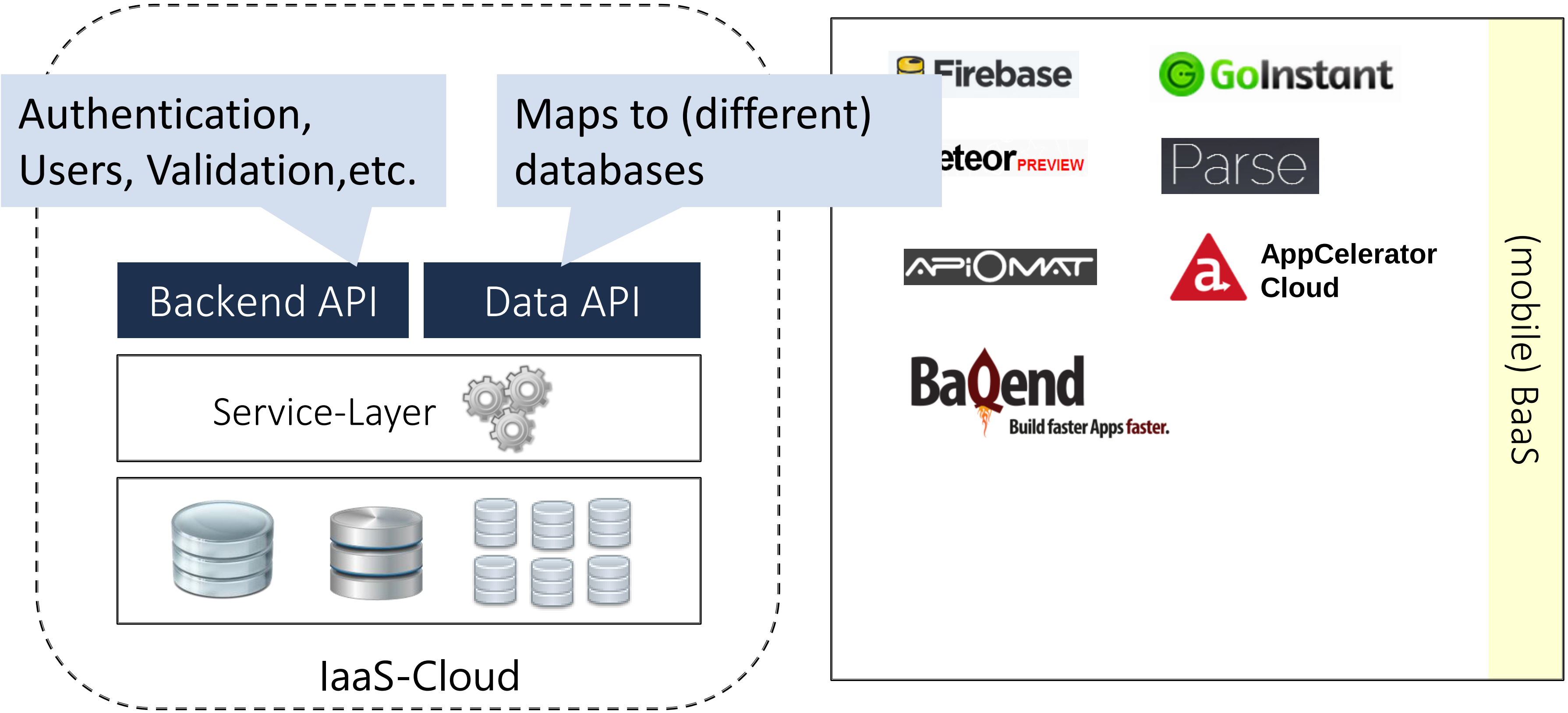
- **Amazon Elastic MapReduce**
- **Azure HDInsight**

ML

- **Google BigQuery**
- **Google Prediction API**

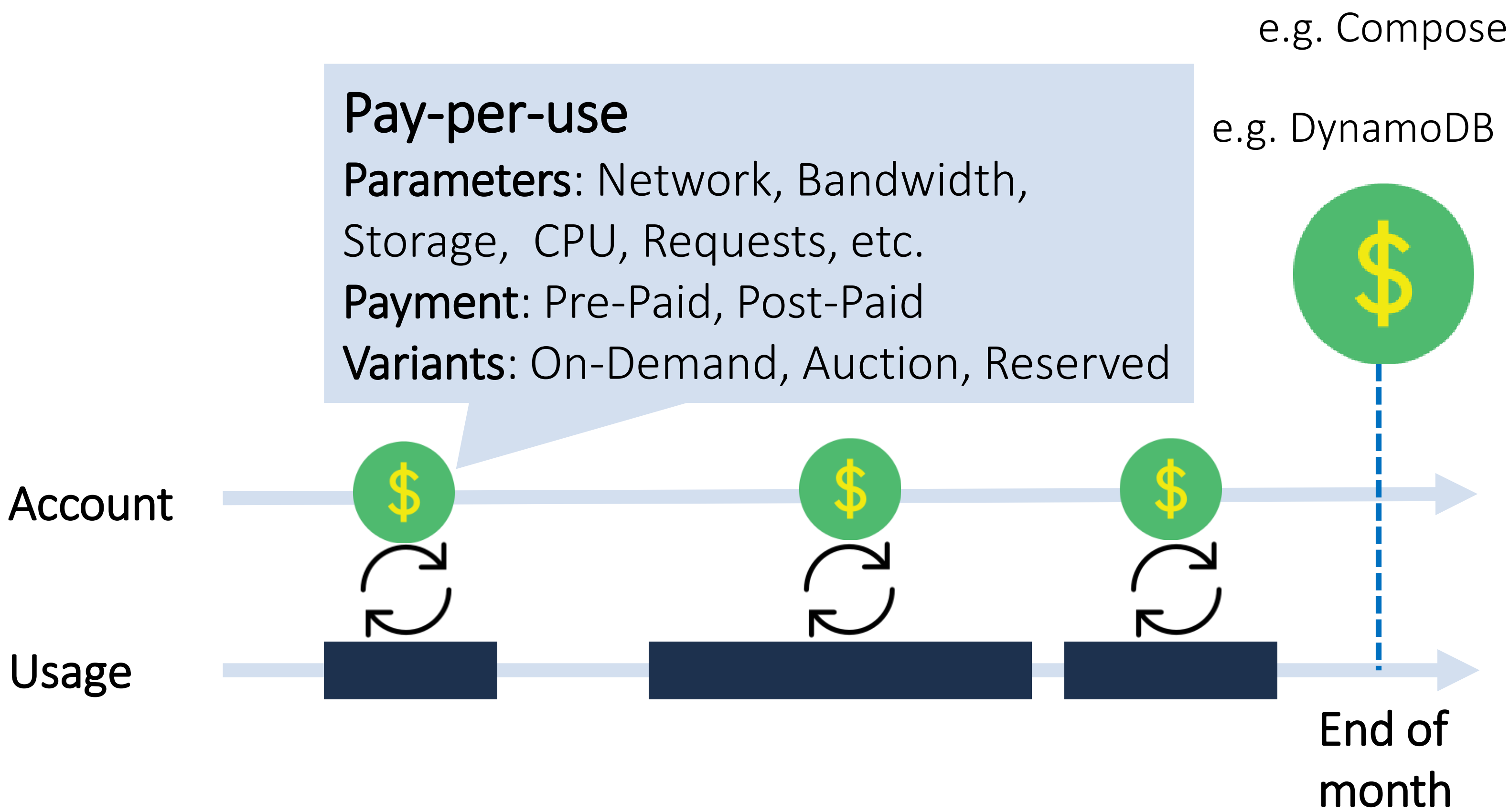
Backend-as-a-Service

DBaaS with embedded custom and predefined application logic



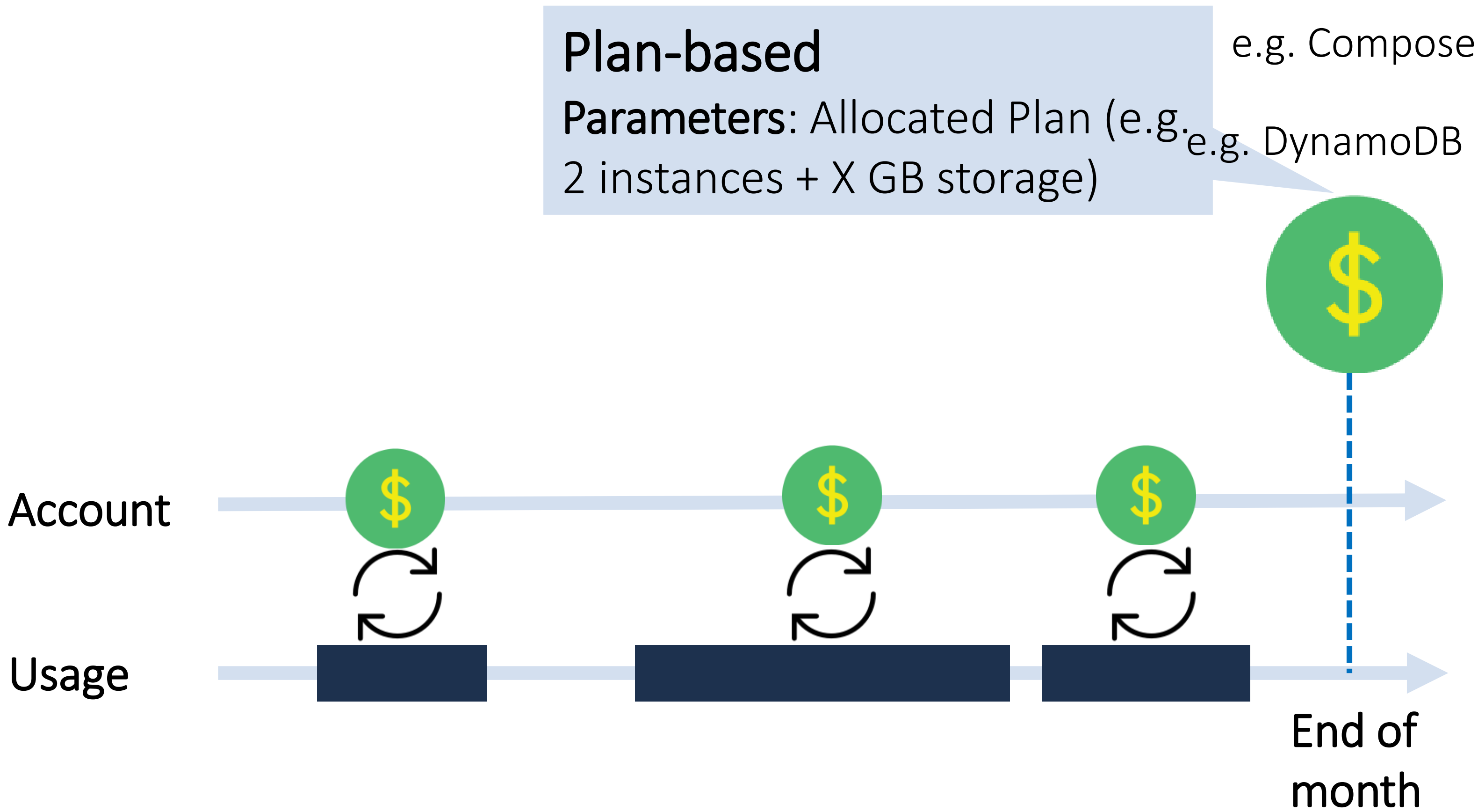
Pricing Models

Pay-per-use and plan-based



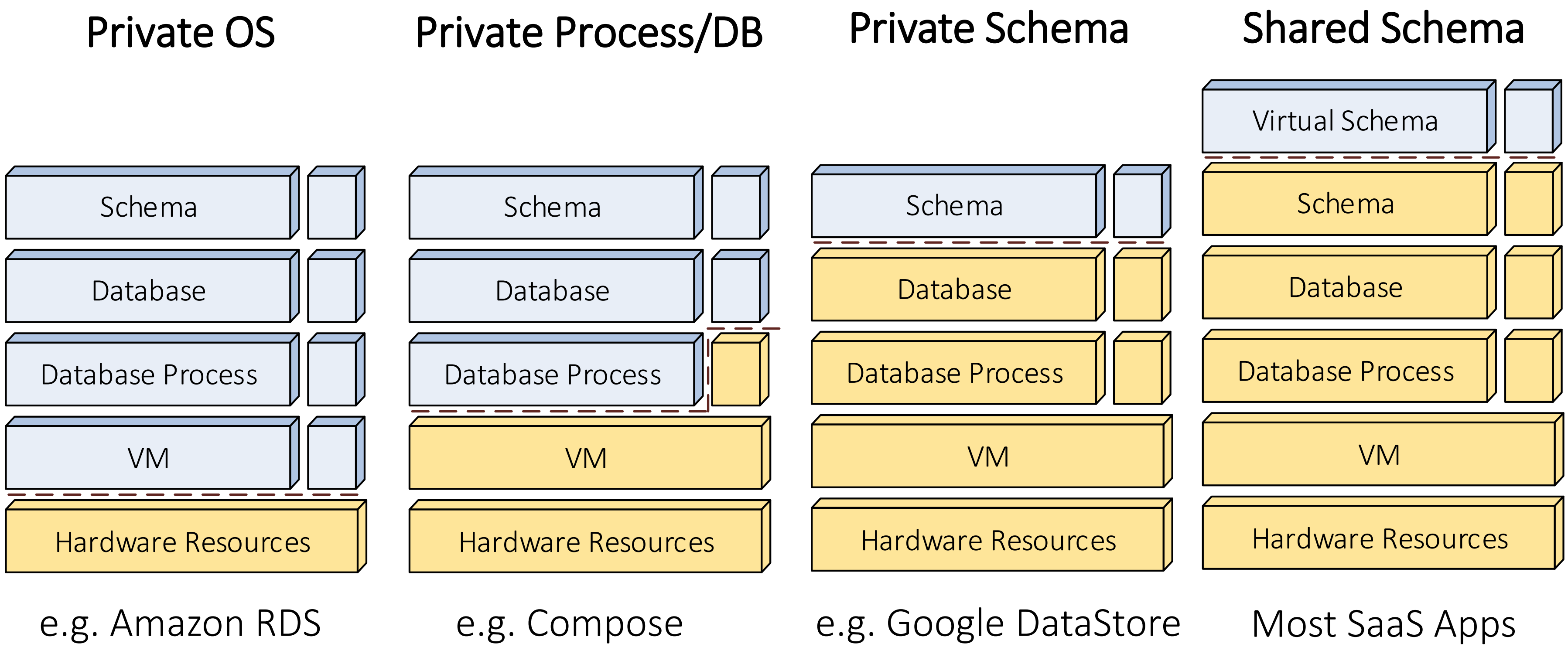
Pricing Models

Pay-per-use and plan-based



Database-as-a-Service

Approaches to Multi-Tenancy



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

Multi-Tenancy: Trade-Offs

	App. indep.	Ressource Util.	Isolation	Maintenance, Provisioning
Private OS	✔	☆☆☆	★★★	★☆☆
Private Process/DB	✔	★☆☆	★★★	★☆☆
Private Schema	✔	★★☆	★★☆	★★★
Shared Schema	✘	★★★	★☆☆	★★★

W. Lehner, U. Sattler "Web-scale Data Management for the Cloud"
Springer, 2013

Authentication & Authorization

Checking Permissions and Indentity

Internal Schemes	External Identity Provider	Federated Identity (Single Sign On)
e.g. Amazon IAM	e.g. OpenID	e.g. SAML

User-based Access Control	Role-based Access Control	Policies
e.g. Amazon S3 ACLs	e.g. Amazon IAM	e.g. XACML

Service Level Agreements (SLAs)

Specification of Application/Tenant Requirements

SLA

<u>Technical Part</u>

1. SLO
2. SLO
3. SLO

<u>Legal Part</u>

1. Fees
2. Penalties

Service Level Objectives:

- Availability
- Durability
- Consistency/Staleness
- Query Response Time

Service Level Agreements

Expressing application requirements

Functional Service Level Objectives

- Guarantee a „feature“
- Determined by database system
- *Examples*: transactions, join

Non-Functional Service Level Objectives

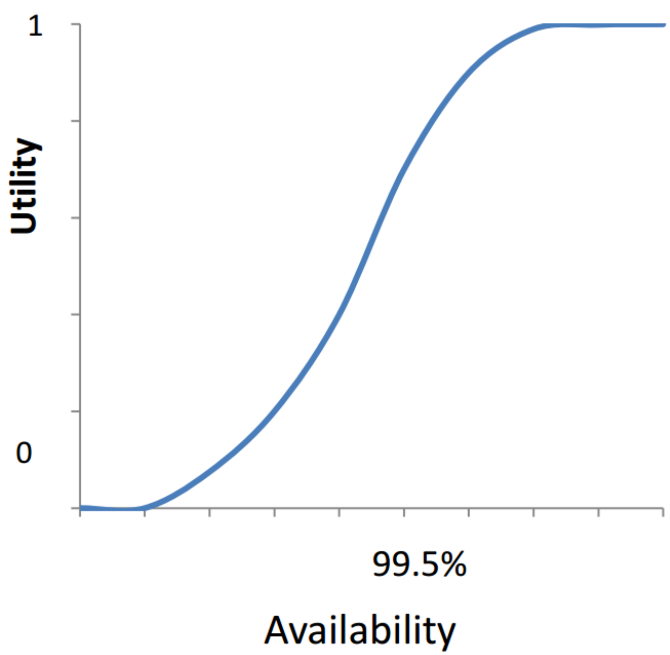
- Guarantee a certain *quality of service* (QoS)
- Determined by database system and service provider
- *Examples*:
  - **Continuous**: response time (latency), throughput
  - **Binary**: Elasticity, Read-your-writes

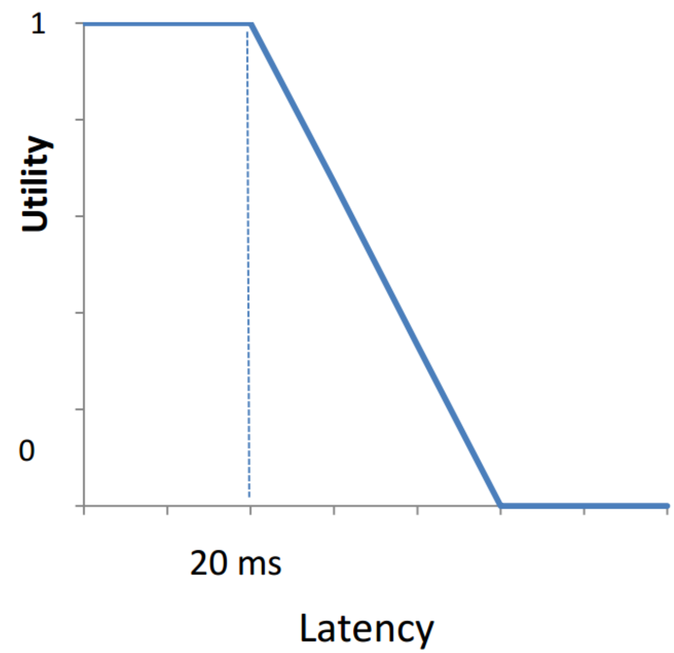
Service Level Objectives

Making SLOs measurable through utilities

Utility expresses „value“ of a continuous non-functional requirement:

$$f_{utility}(metric) \rightarrow [0,1]$$

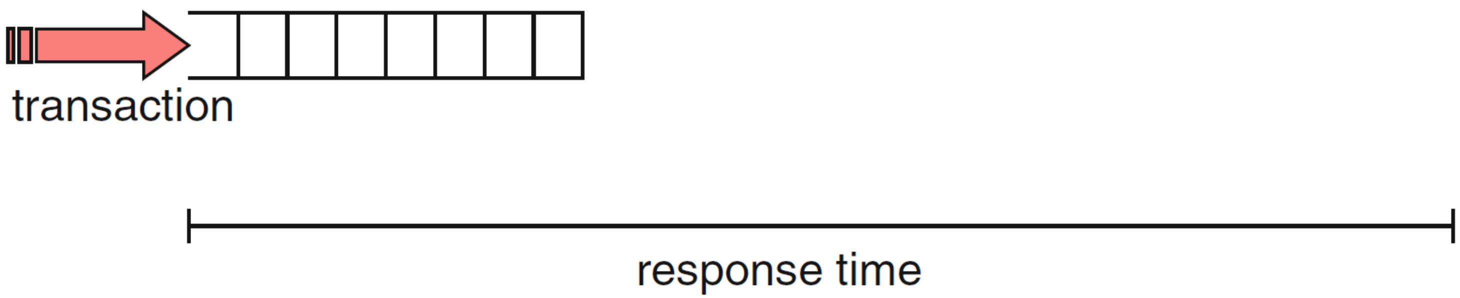




Workload Management

Guaranteeing SLAs

Typical approach:

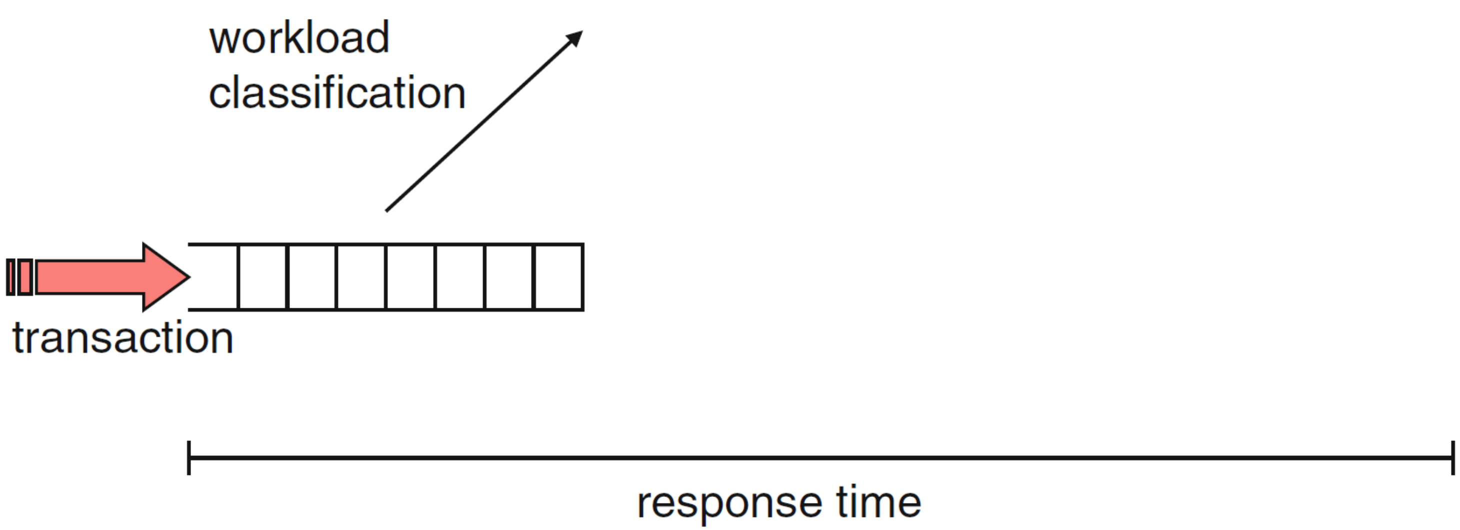


W. Lehner, U. Sattler “Web-scale Data Management for the Cloud”
Springer, 2013

Workload Management

Guaranteeing SLAs

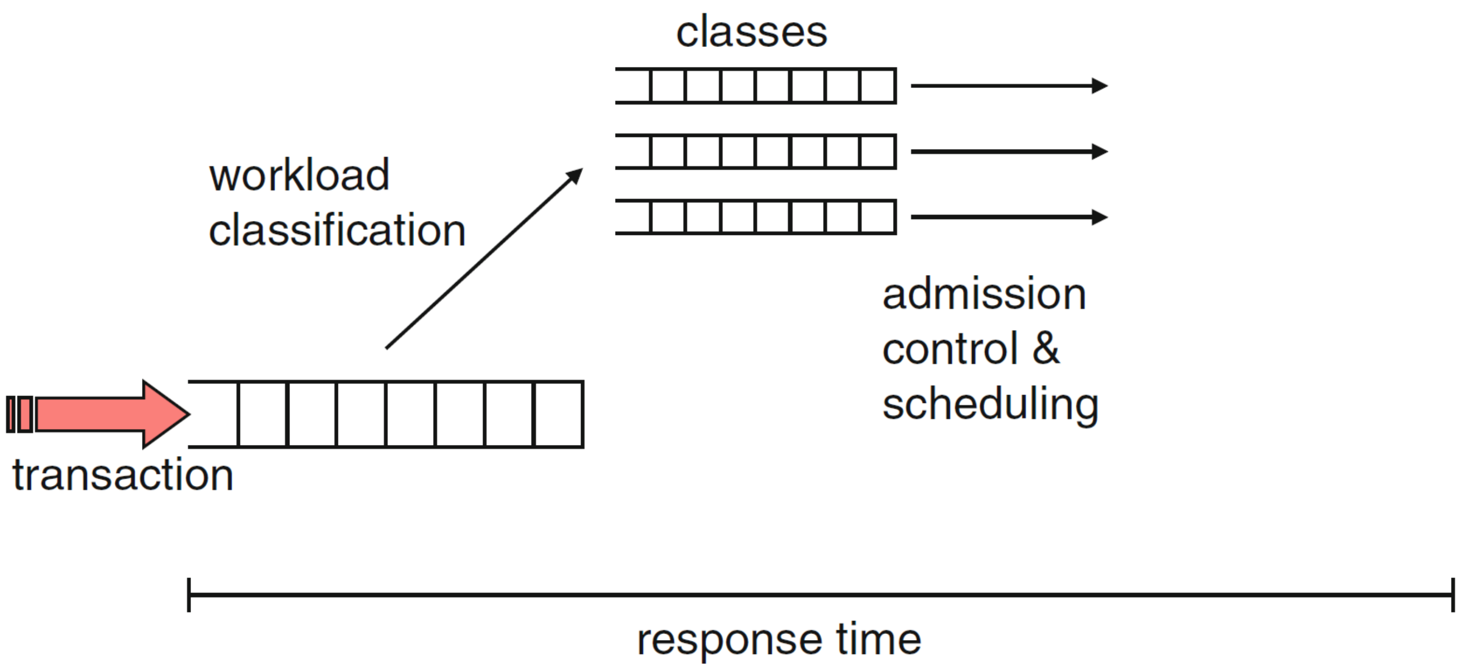
Typical approach:



W. Lehner, U. Sattler "Web-scale Data Management for the Cloud"
Springer, 2013

Workload Management
Guaranteeing SLAs

Typical approach:



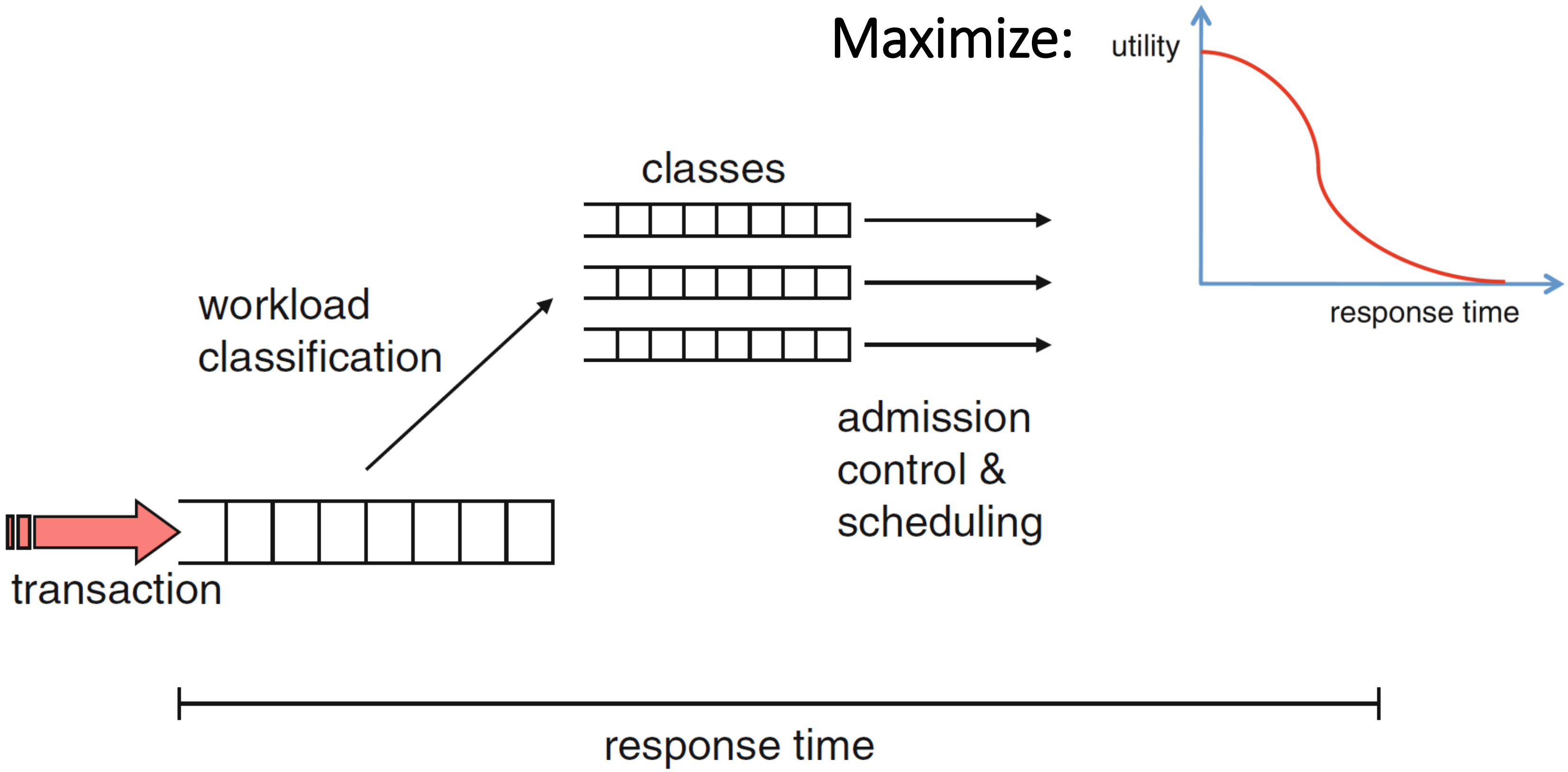
W. Lehner, U. Sattler "Web-scale Data Management for the Cloud" Springer, 2013

Workload Management

Guaranteeing SLAs

Typical approach:

Maximize:

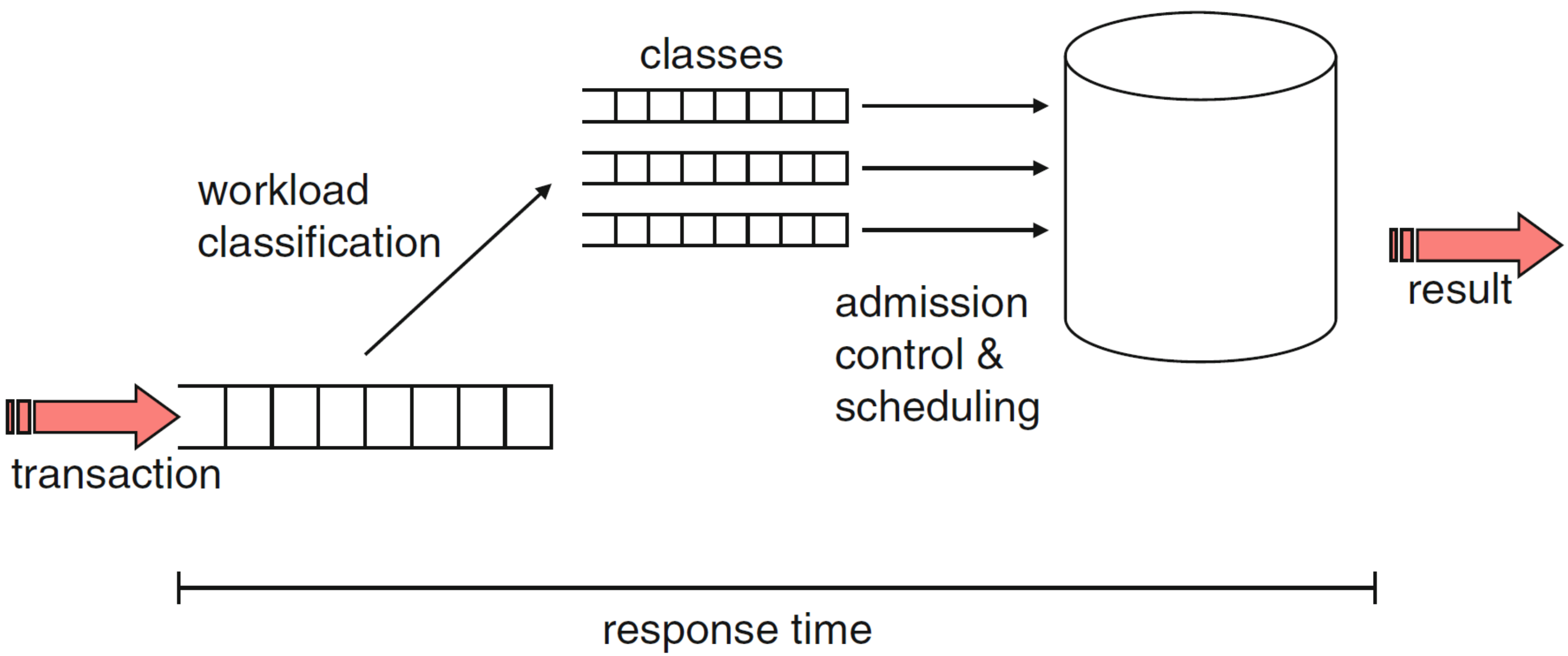


W. Lehner, U. Sattler "Web-scale Data Management for the Cloud"
Springer, 2013

Workload Management

Guaranteeing SLAs

Typical approach:

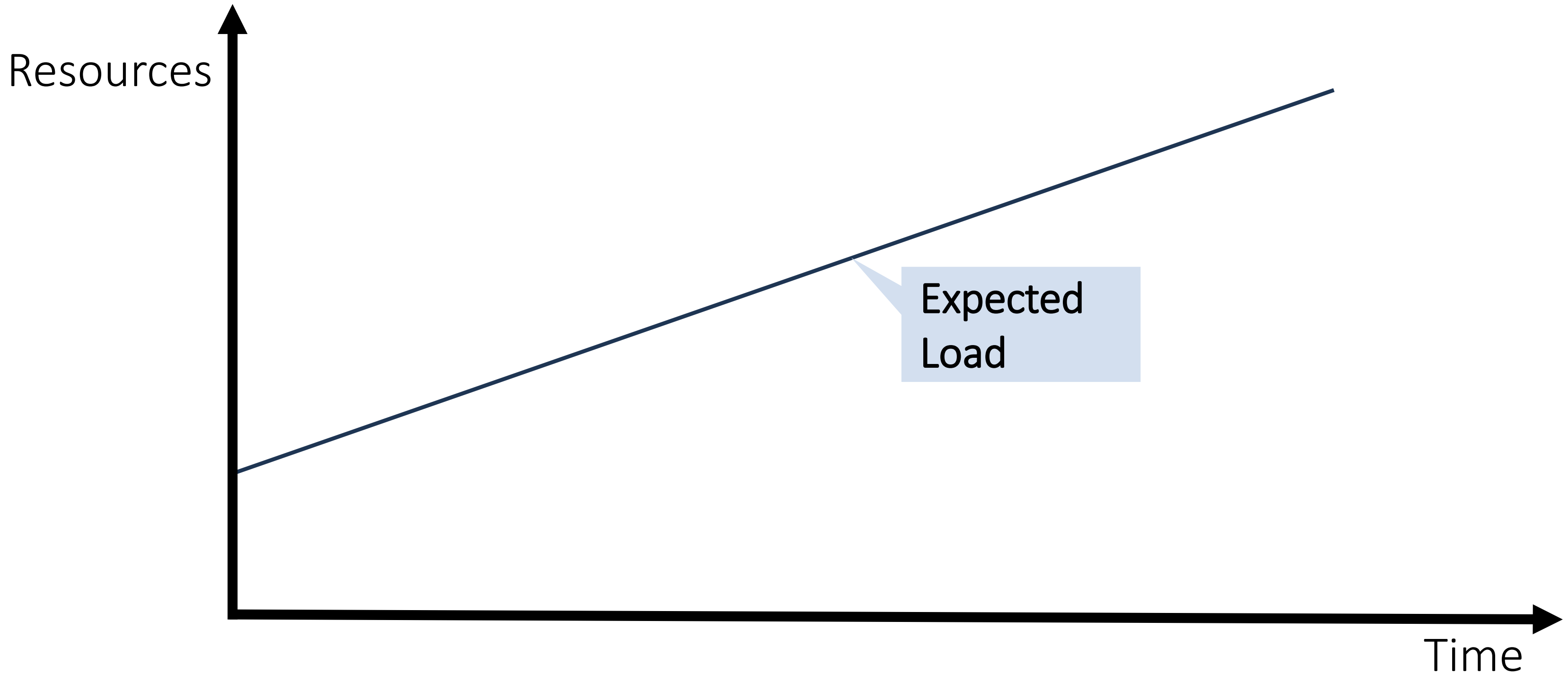


W. Lehner, U. Sattler "Web-scale Data Management for the Cloud"
Springer, 2013

Resource & Capacity Planning

From a DBaaS provider‘s perspective

Goal: minimize penalty and resource costs

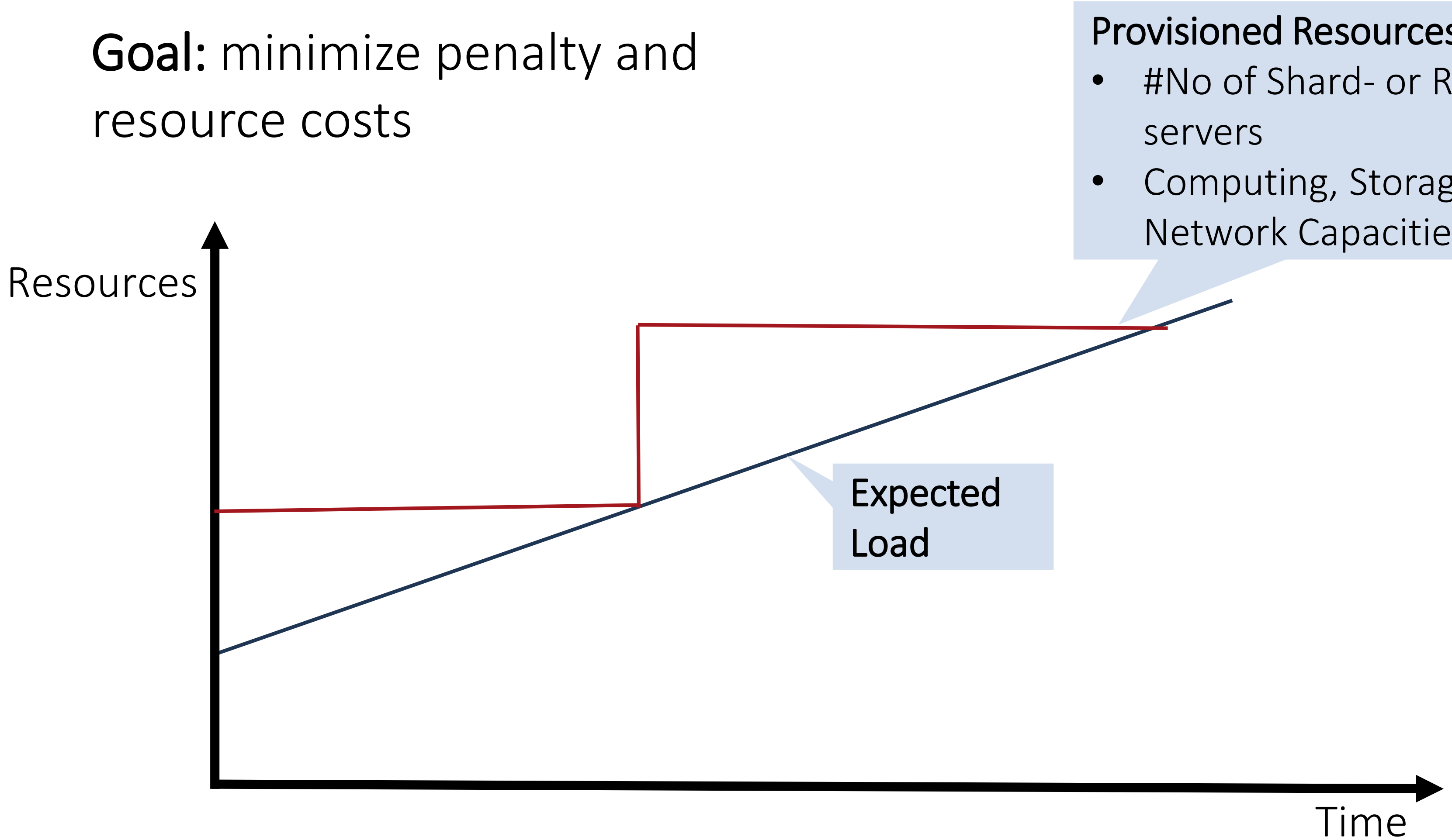


T. Lorido-Botran, J. Miguel-Alonso et al.: “Auto-scaling Techniques for Elastic Applications in Cloud Environments”. Technical Report, 2013

Resource & Capacity Planning

From a DBaaS provider's perspective

Goal: minimize penalty and resource costs

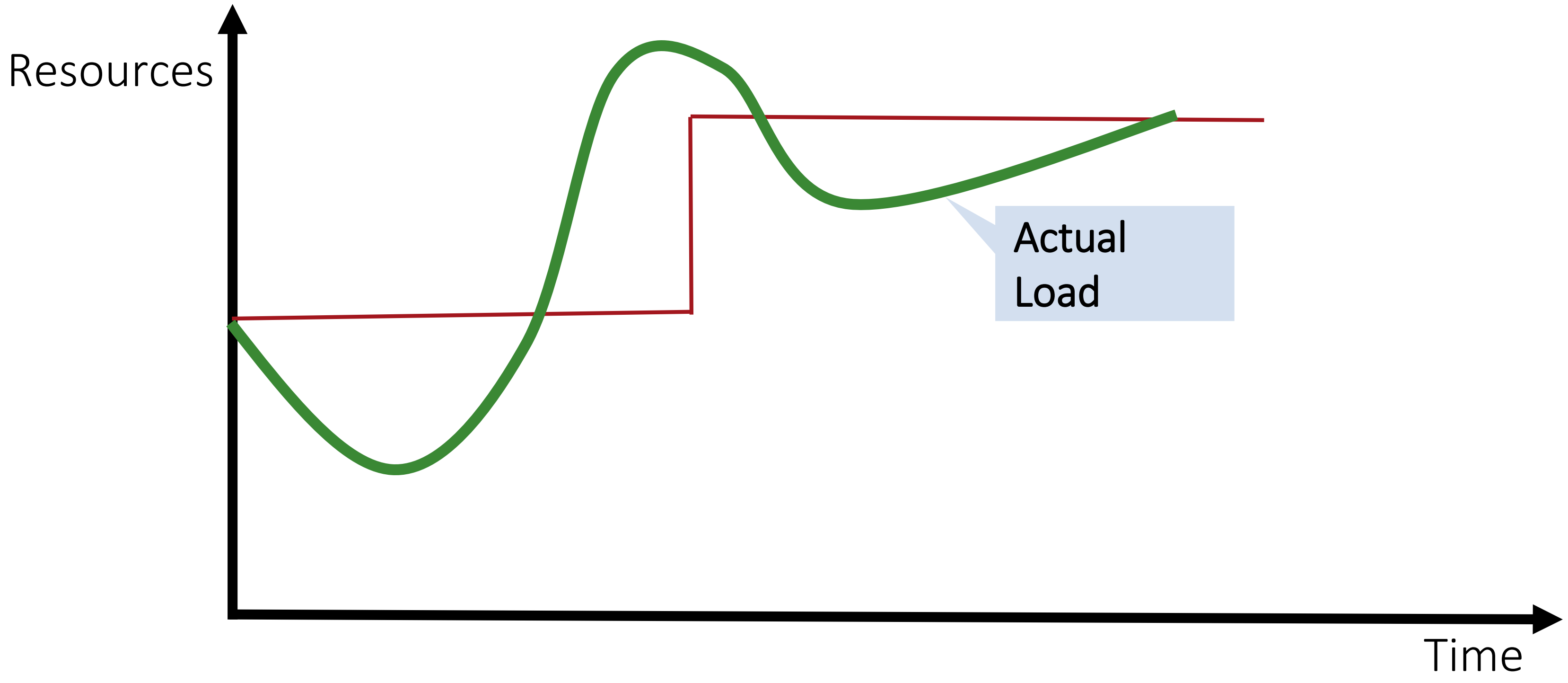


T. Lorido-Botran, J. Miguel-Alonso et al.: "Auto-scaling Techniques for Elastic Applications in Cloud Environments". Technical Report, 2013

Resource & Capacity Planning

From a DBaaS provider‘s perspective

Goal: minimize penalty and resource costs

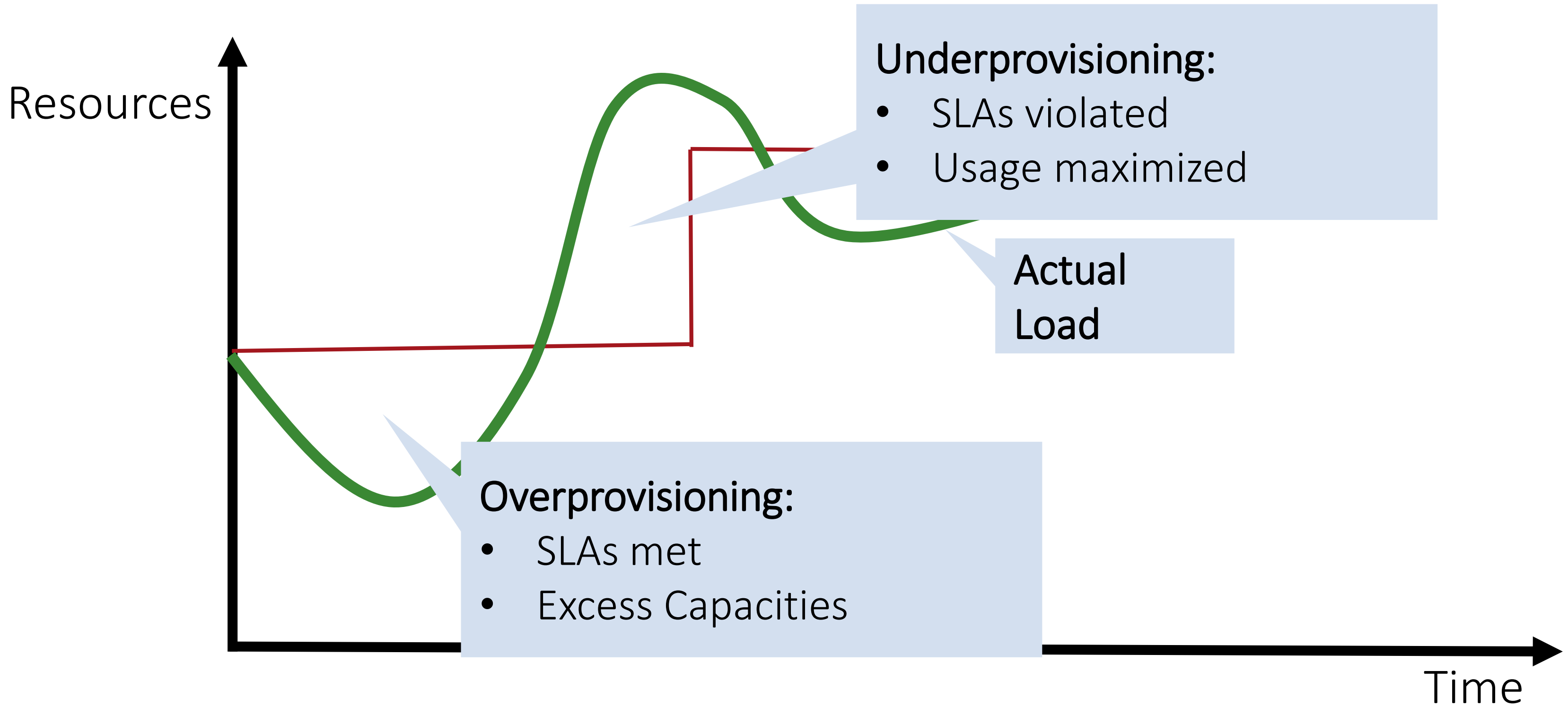


T. Lorido-Botran, J. Miguel-Alonso et al.: “Auto-scaling Techniques for Elastic Applications in Cloud Environments”. Technical Report, 2013

Resource & Capacity Planning

From a DBaaS provider's perspective

Goal: minimize penalty and resource costs



T. Lorido-Botran, J. Miguel-Alonso et al.: "Auto-scaling Techniques for Elastic Applications in Cloud Environments". Technical Report, 2013

SLAs in the wild

Most DBaaS systems offer no SLAs, or only a a simple uptime guarantee

	Model	CAP	SLAs
SimpleDB	Table-Store (*NoSQL Service*)	CP	✗
Dynamo-DB	Table-Store (*NoSQL Service*)	CP	✗
Azure Tables	Table-Store (*NoSQL Service*)	CP	99.9% uptime
AE/Cloud DataStore	Entity-Group Store (*NoSQL Service*)	CP	✗
S3, Az. Blob, GCS	Object-Store (*NoSQL Service*)	AP	99.9% uptime (S3)

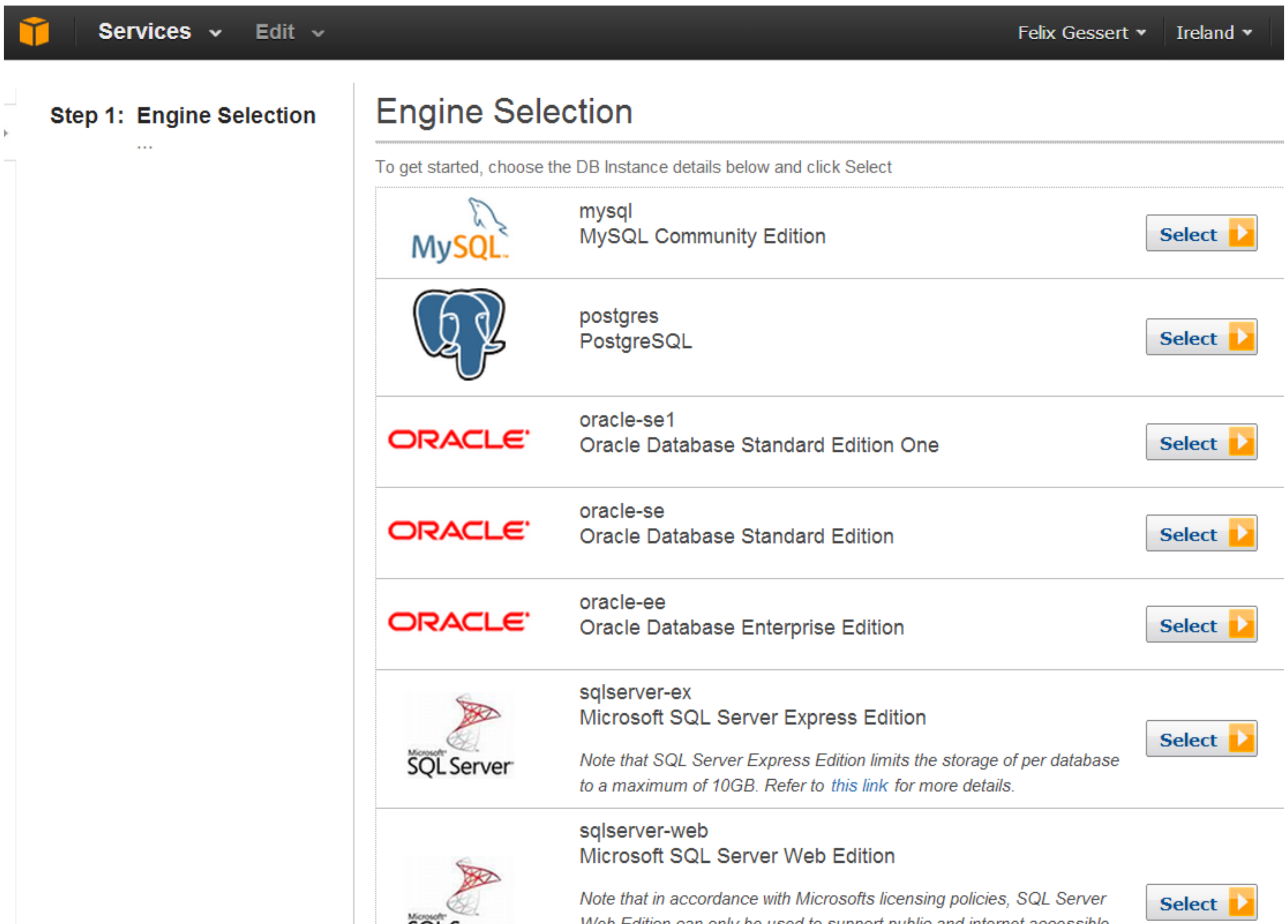
Open Research Questions
in Cloud Data Management

- Service-Level Agreements
  - How can SLAs be guaranteed in a virtualized, multi-tenant cloud environment?
- Consistency
  - Which consistency guarantees can be provided in a geo-replicated system without sacrificing availability?
- Performance & Latency
  - How can a DBaaS deliver low latency in face of distributed storage and application tiers?
- Transactions
  - Can ACID transactions be aligned with NoSQL and scalability?

DBaaS Example

Amazon RDS

- Relational Database Service



Amazon RDS

RDS

Model:

Managed RDBMS

Pricing:

Instance + Volume + License

Underlying DB:

MySQL, Postgres, MSSQL, Oracle

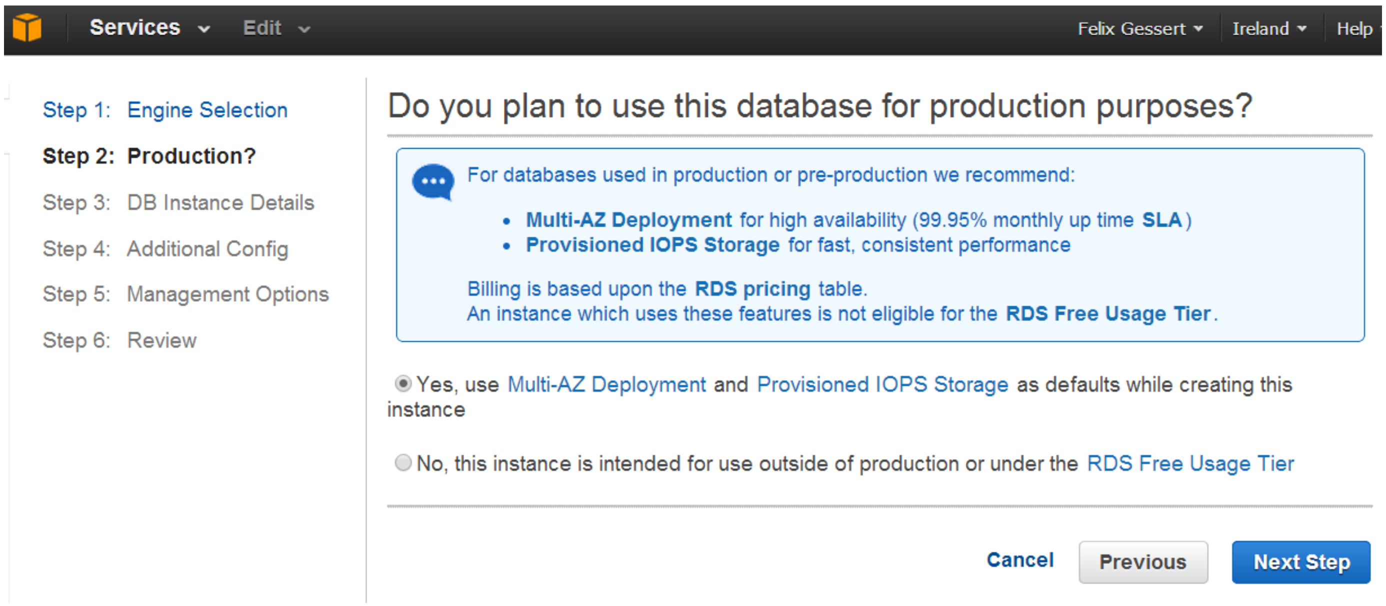
API:

DB-specific

DBaaS Example

Amazon RDS

- Relational Database Service



Amazon RDS

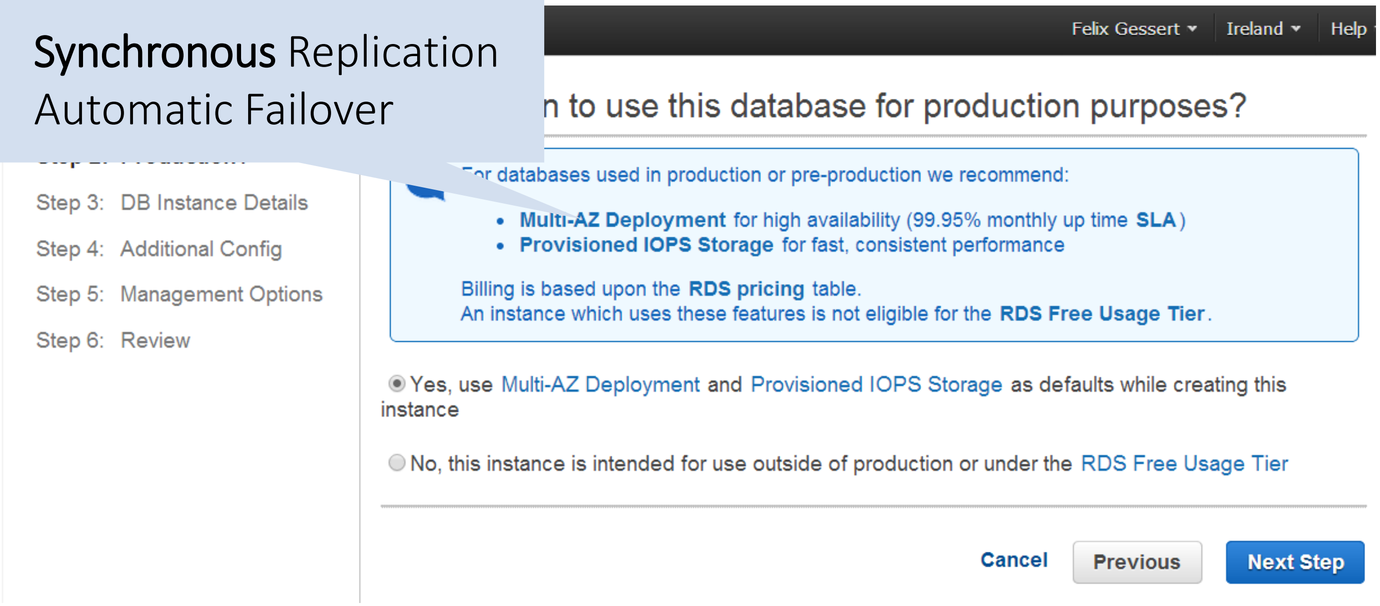
RDS
Model:
Managed RDBMS
Pricing:
Instance + Volume + License
Underlying DB:
MySQL, Postgres, MSSQL, Oracle
API:
DB-specific

DBaaS Example

Amazon RDS

- **R**elational **D**atabase **S**ervice

- **Synchronous** Replication
- Automatic Failover



Amazon RDS

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

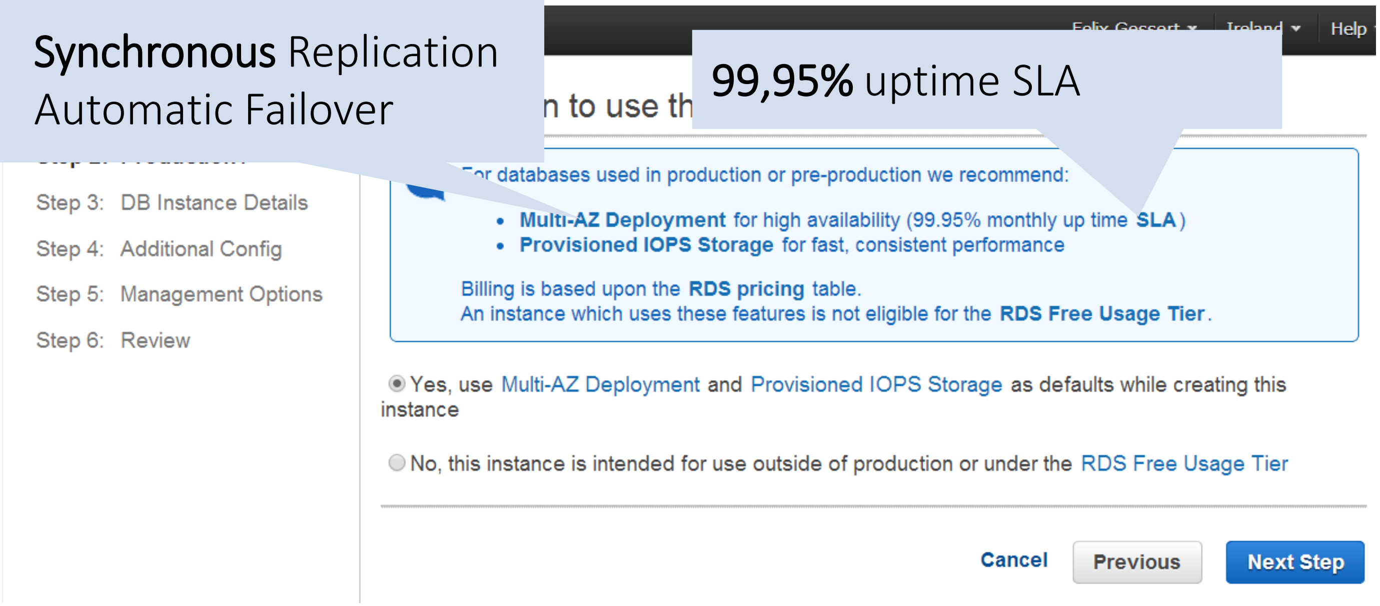
DBaaS Example

Amazon RDS

- Relational **D**atabase **S**ervice

- **Synchronous** Replication
- Automatic Failover

99,95% uptime SLA



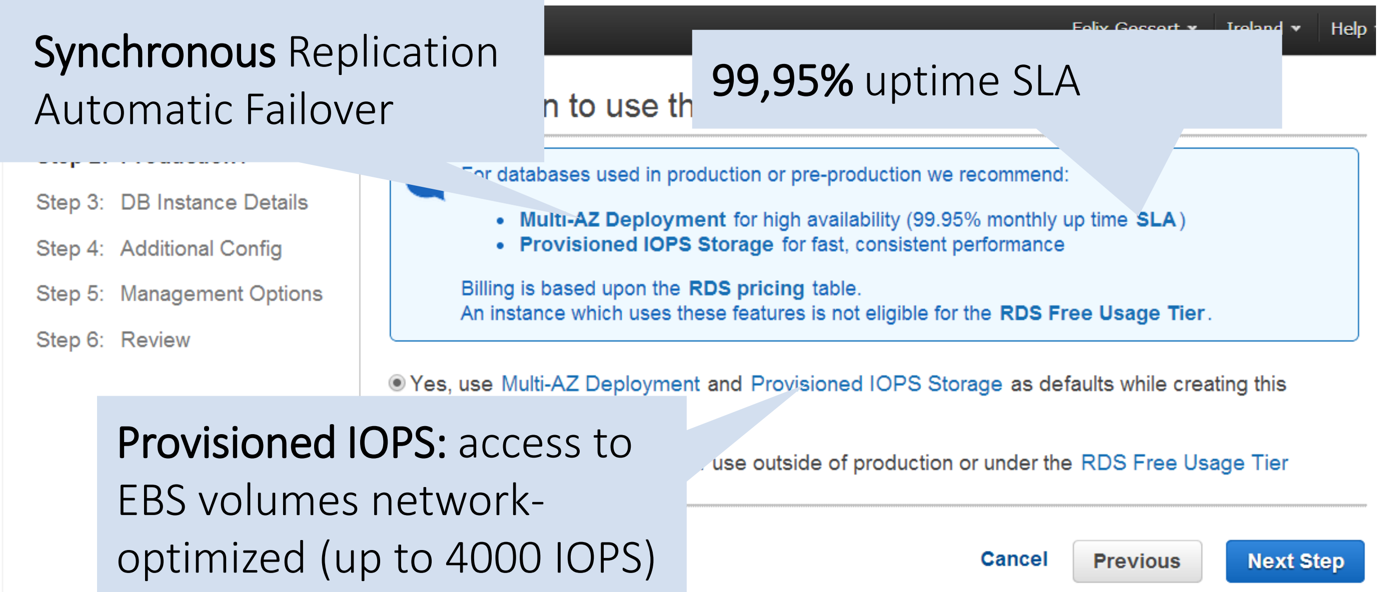
Amazon RDS

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

DBaaS Example

Amazon RDS

- **R**elational **D**atabase **S**ervice



- **Synchronous** Replication
- Automatic Failover

99,95% uptime SLA

Provisioned IOPS: access to EBS volumes network-optimized (up to 4000 IOPS)

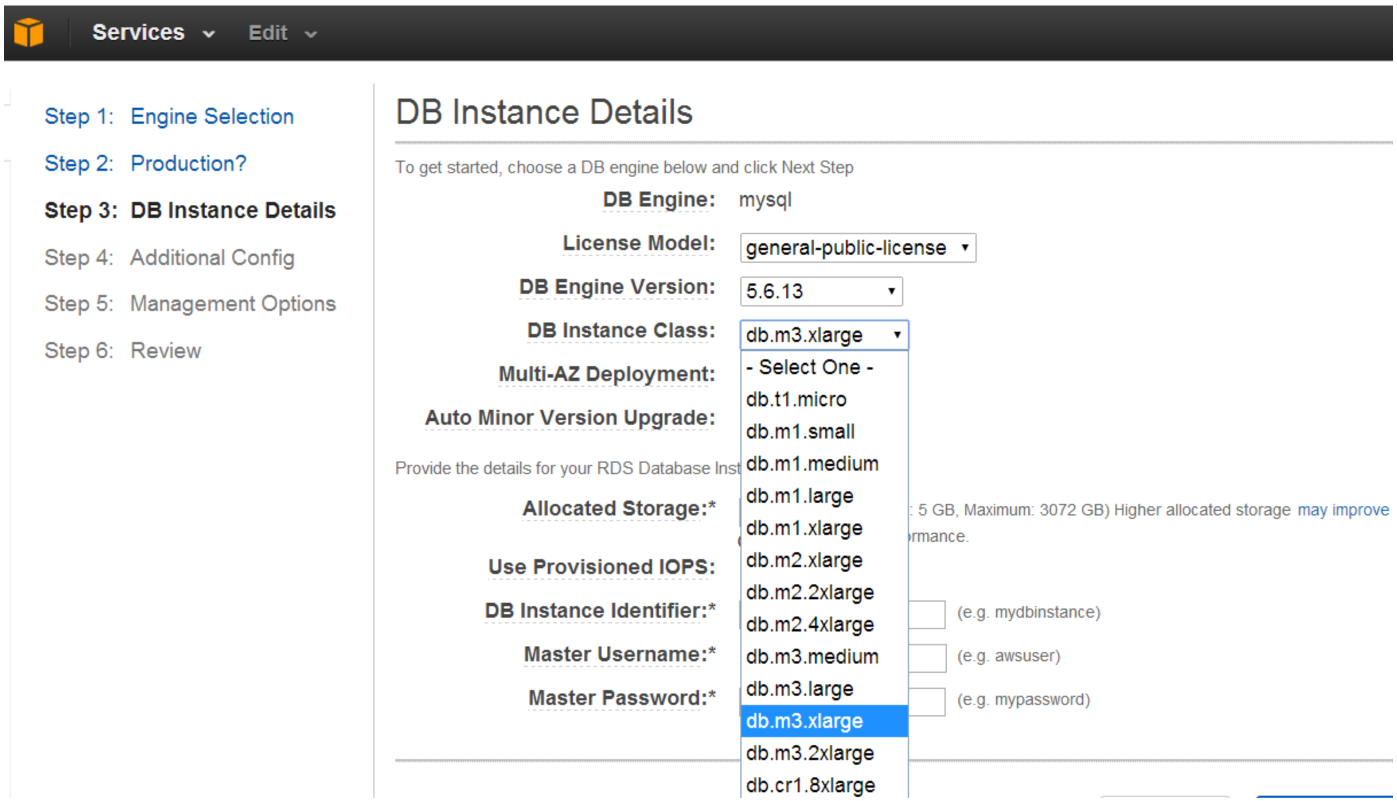
Amazon RDS

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

DBaaS Example

Amazon RDS

- Relational Database Service



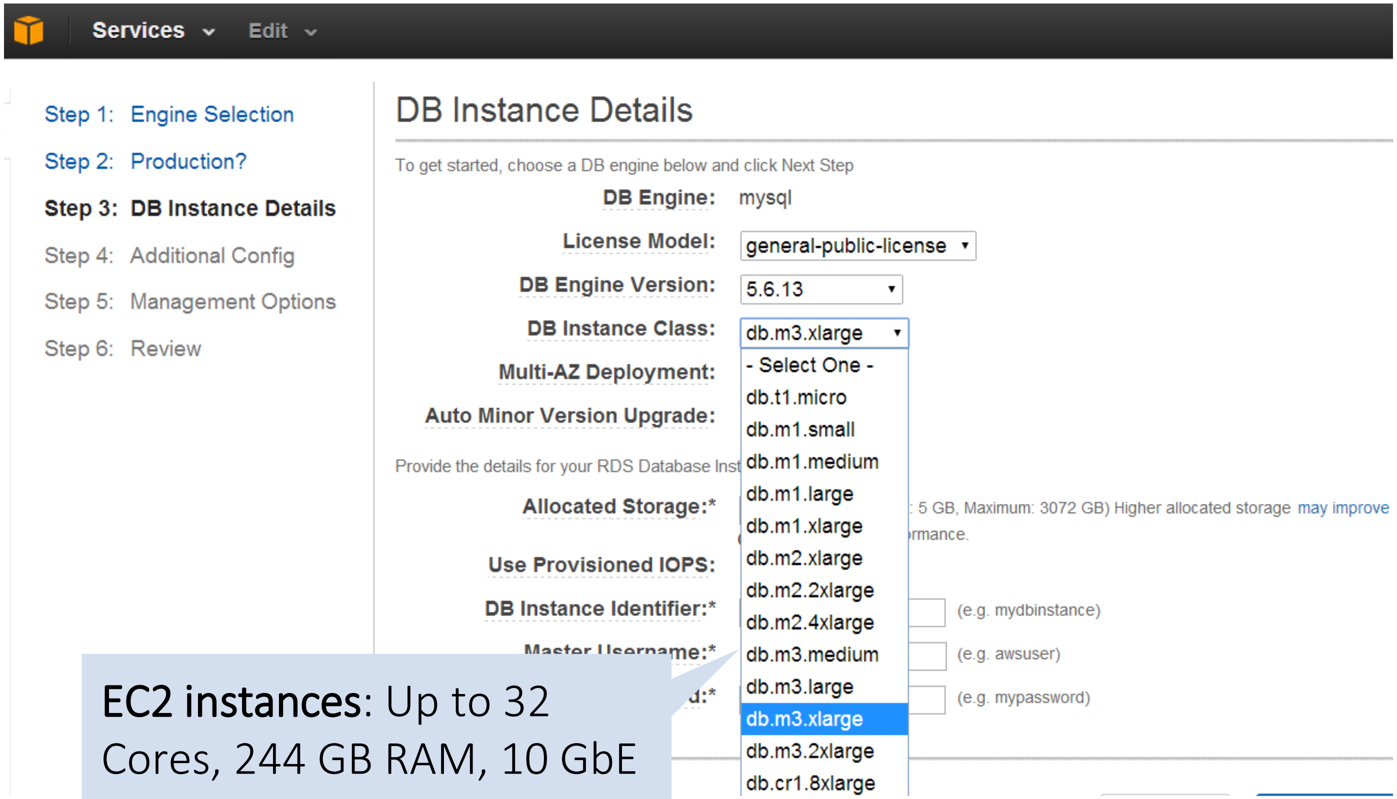
Amazon RDS

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

DBaaS Example

Amazon RDS

- Relational **D**atabase **S**ervice



EC2 instances: Up to 32 Cores, 244 GB RAM, 10 GbE

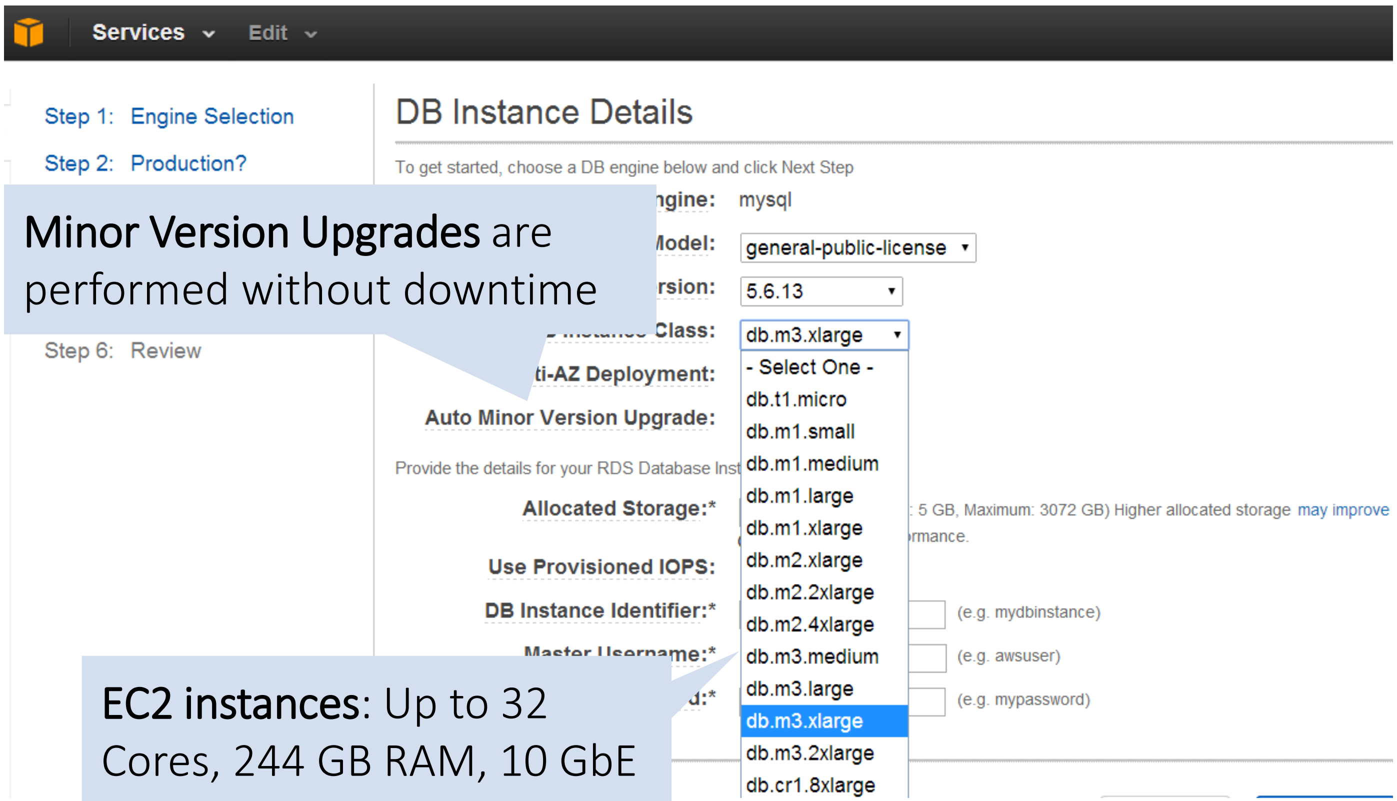
Amazon RDS

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

DBaaS Example

Amazon RDS

- Relational Database Service



Minor Version Upgrades are performed without downtime

EC2 instances: Up to 32 Cores, 244 GB RAM, 10 GbE

Amazon RDS

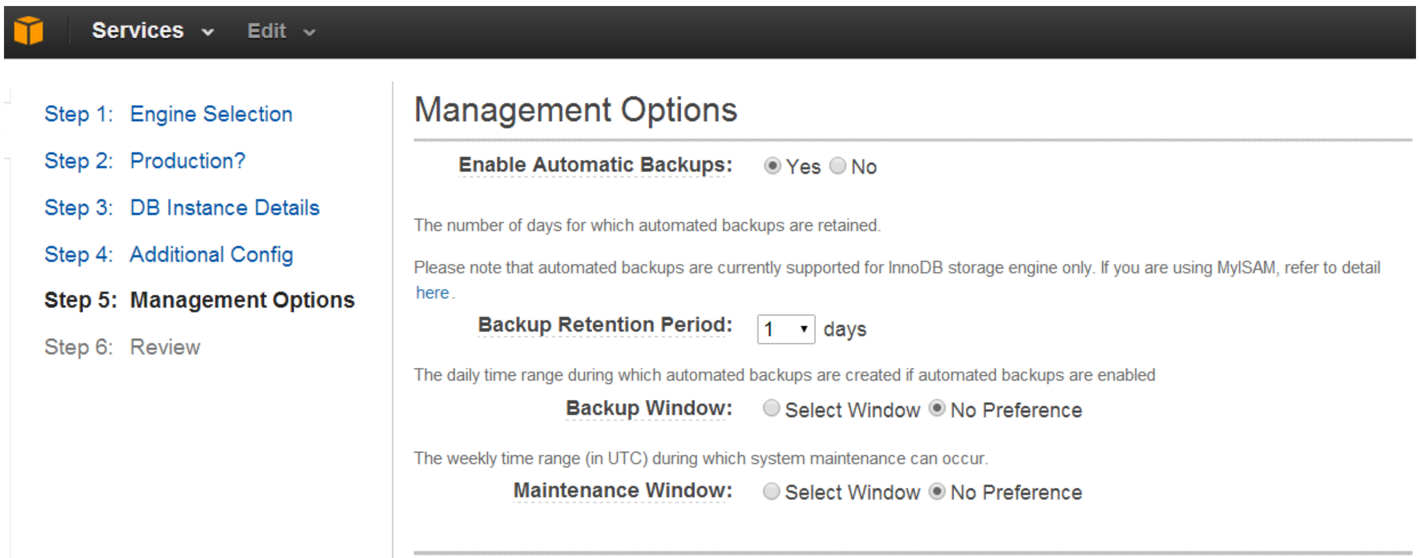
RDS
Model:
Managed RDBMS
Pricing:
Instance + Volume + License
Underlying DB:
MySQL, Postgres, MSSQL, Oracle
API:
DB-specific

Amazon RDS

DBaaS Example

Amazon RDS

- **R**elational **D**atabase **S**ervice



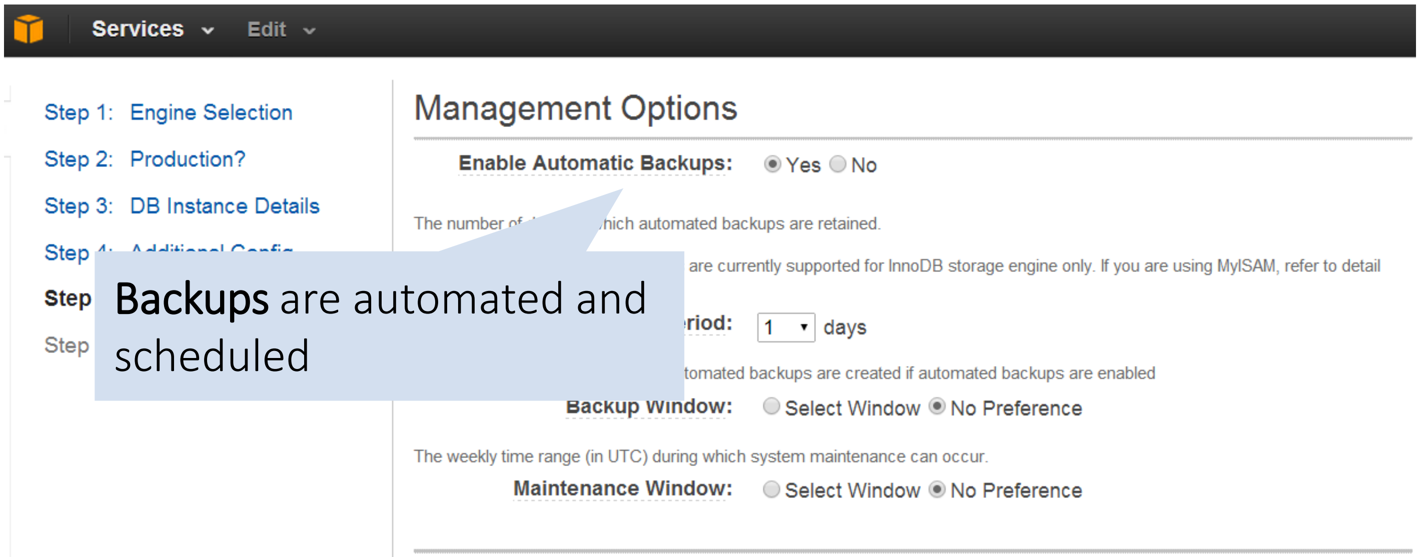
RDS
Model:
Managed RDBMS
Pricing:
Instance + Volume + License
Underlying DB:
MySQL, Postgres, MSSQL, Oracle
API:
DB-specific

Amazon RDS

DBaaS Example

Amazon RDS

- **R**elational **D**atabase **S**ervice



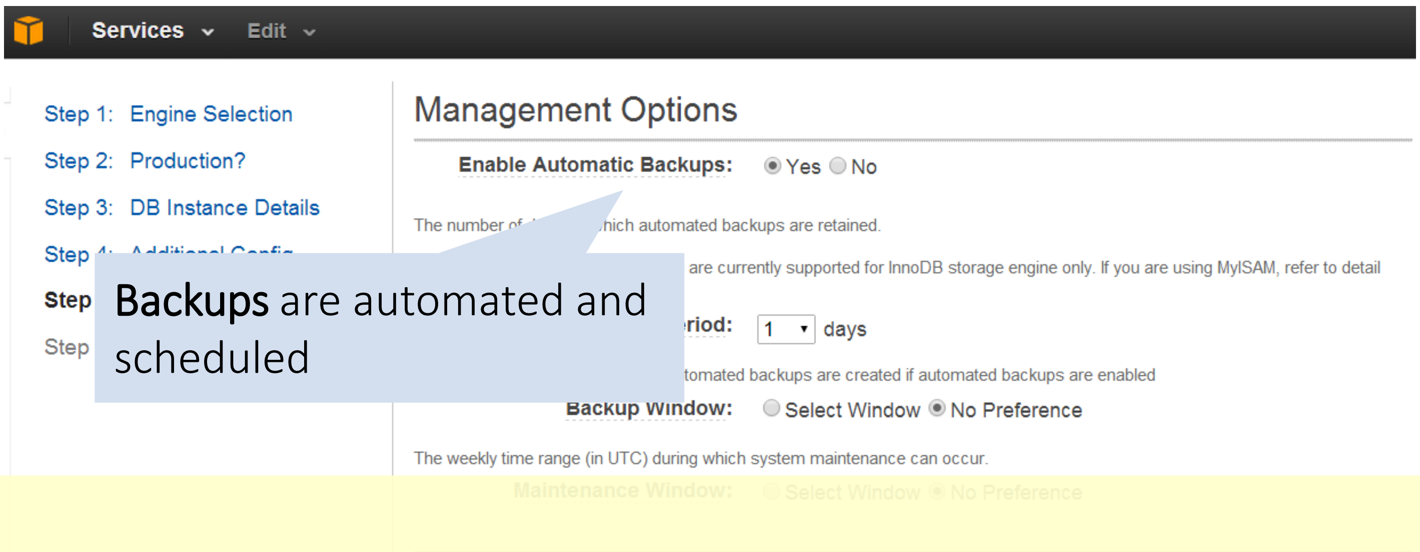
Backups are automated and scheduled

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

DBaaS Example

Amazon RDS

- **R**elational **D**atabase **S**ervice



Backups are automated and scheduled

Amazon RDS

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

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

DBaaS Example

Azure Tables

REST API

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

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

Atomic "Entity-Group Batch Transaction" possible

Partition

Partition

Hash-distributed to parition servers

Sparse

- Similar to Amazon SimpleDB and DynamoDB

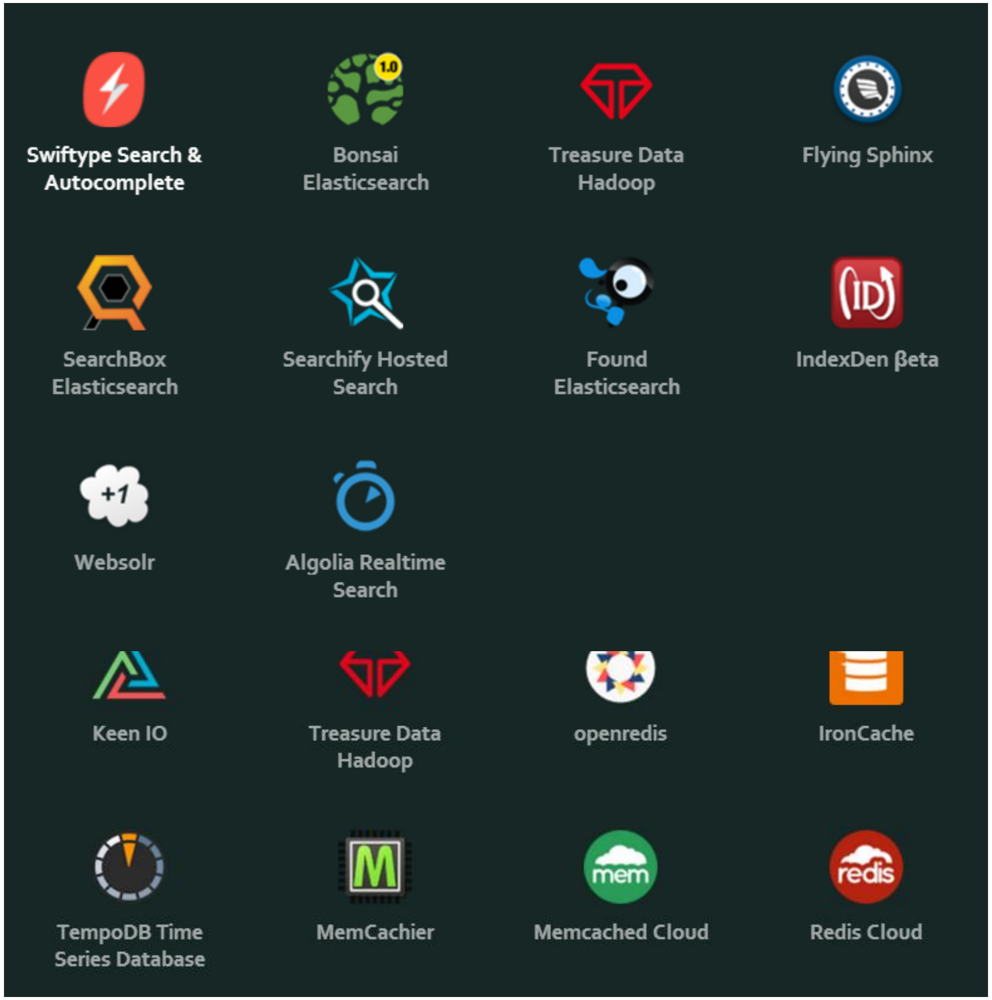
SimpleDB:
- Indexes all attributes
- Rich(er) queries
- Many Limits (size, RPS, etc.)

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

DBaaS and PaaS Example

Heroku Addons

- Many Hosted NoSQL DbaaS Providers represented
- And Search



DBaaS and PaaS Example

Heroku Addons

Create Heroku App:

```
$ heroku create
```

Add Redis2Go Addon:

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

Use Connection URL (environment variable):

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

Deploy:

```
$ git push heroku master
```

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

DBaaS and PaaS Example

Heroku Addons

Create Heroku App:

```
$ heroku create
```

Add Redis2Go Addon:

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

Use Connection URL (environment variable):

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

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

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

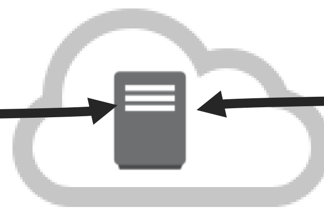
Cloud-Deployed DB

An alternative to DBaaS-Systems

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

- Method I: DIY

1. **Provision** VM(s)



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

- Method II: Deployment Tools

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

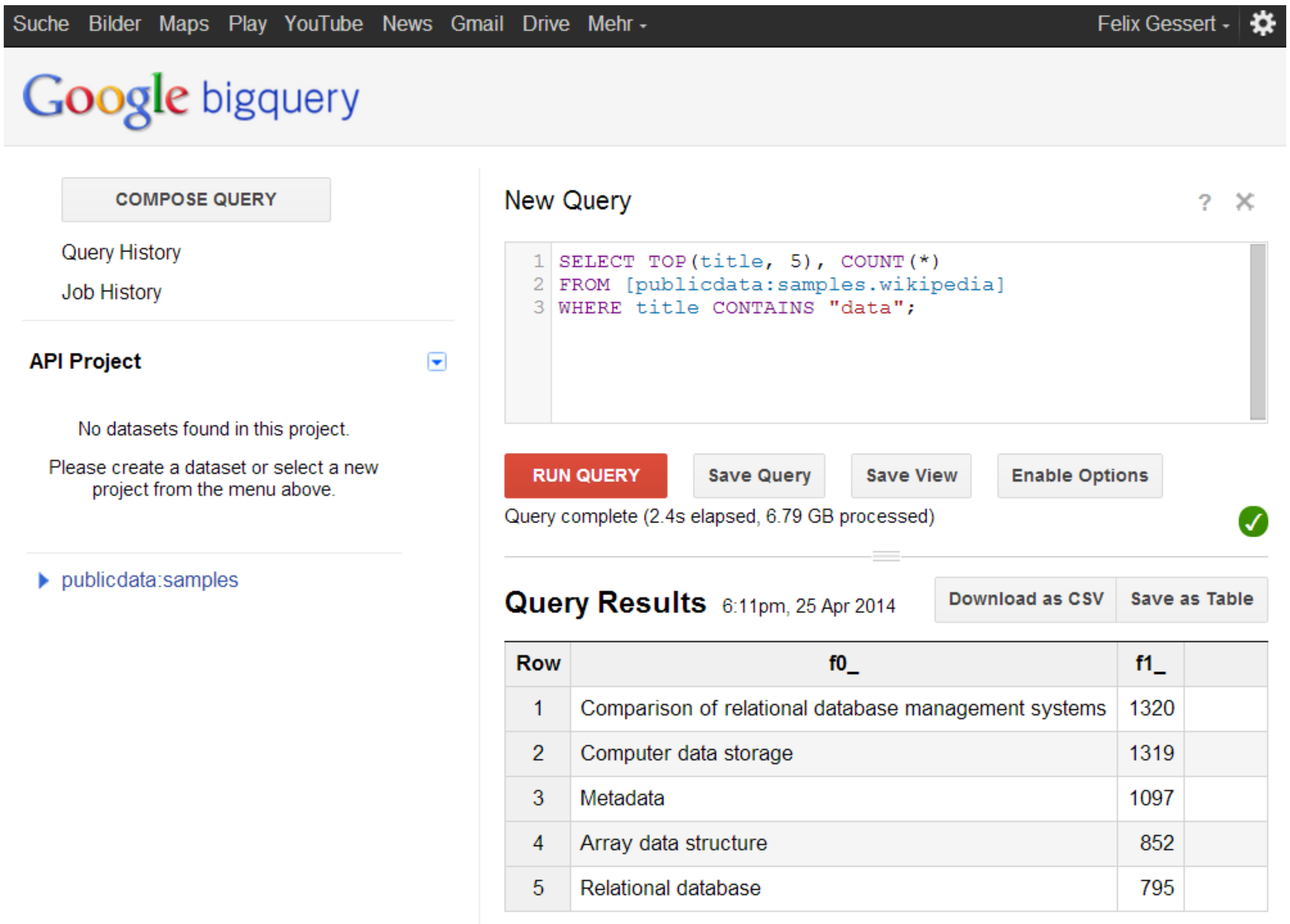
Login, cluster-size etc.

Amazon EC2

- Method III: Marketplaces

Google BigQuery

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

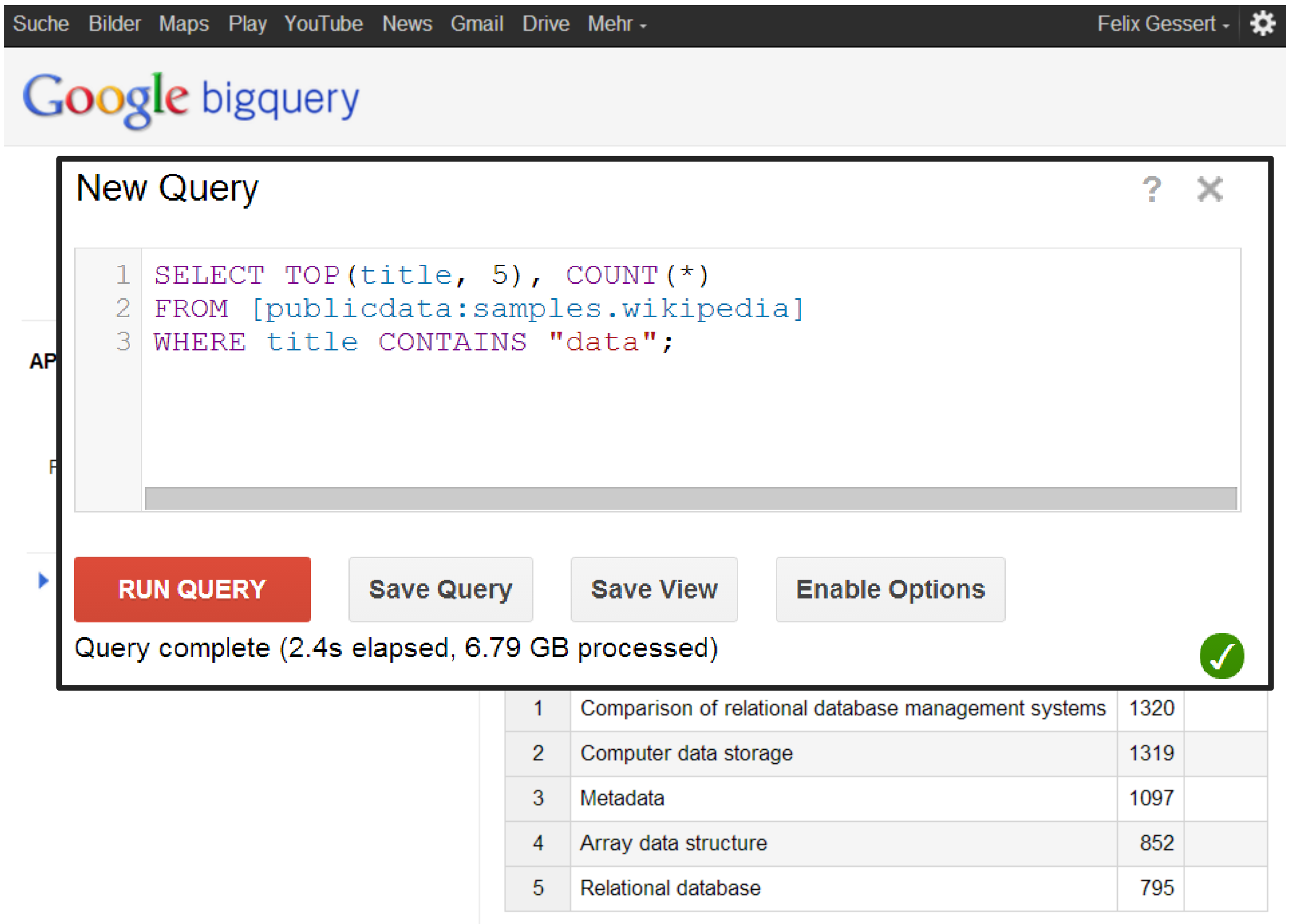


Google BigQuery

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

Google BigQuery

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

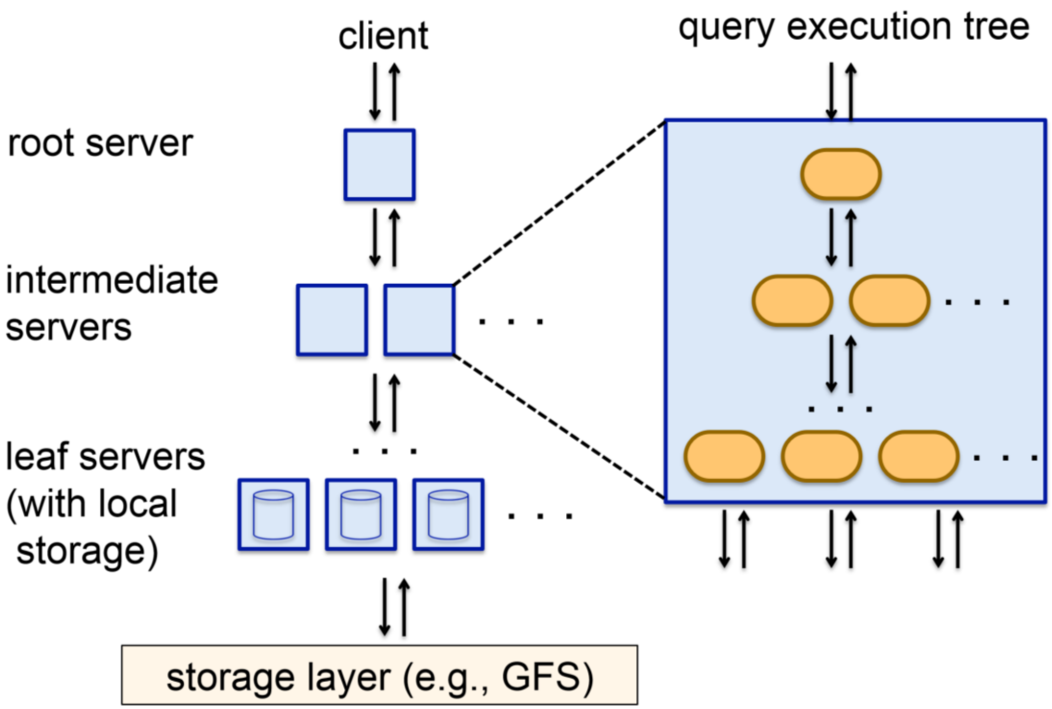


Google BigQuery

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

Google BigQuery

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



Google BigQuery

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

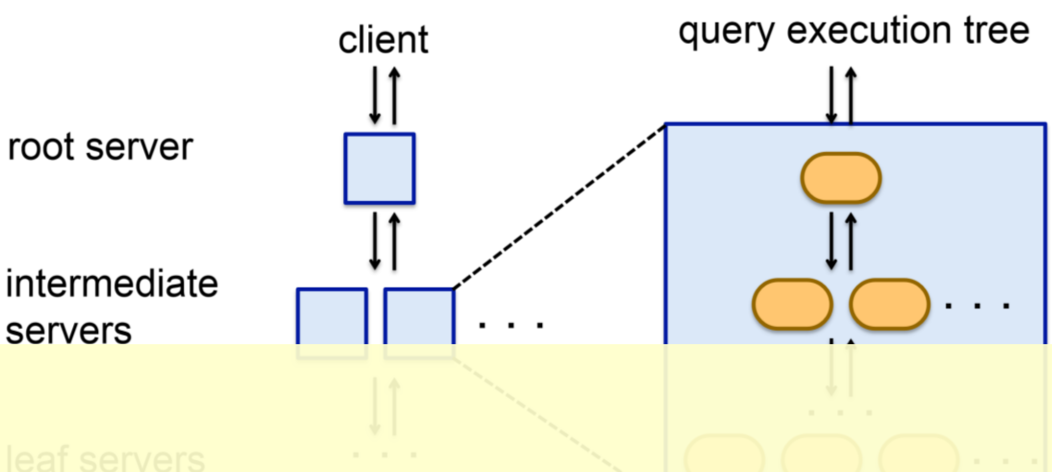
Dremel

Idea:
Multi-Level execution tree on nested columnar data format (≥100 nodes)

Melnik et al. “Dremel: Interactive analysis of web-scale datasets”, VLDB 2010

Google BigQuery

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



Google BigQuery

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

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

Managed NoSQL services

Summary

	Model	CAP	Scans	Sec. Indices	Largest Cluster	Lear-ning	Lic.	DBaaS
HBase	Wide-Column	CP	Over Row Key	✗	~700	1/4	Apache	✗ (EMR)
MongoDB	Doc-ument	CP	yes	✓	>100 <500	4/4	GPL	mongoHQ
Riak	Key-Value	AP	✗	✓	~60	3/4	Apache	✗ (Softlayer)
Cassandra	Wide-Column	AP	With Comp. Index	✓	>300 <1000	2/4	Apache	instaclustr
Redis	Key-Value	CA	Through Lists, etc.	manual	N/A	4/4	BSD	Amazon ElastiCache

Managed NoSQL services

Summary

	Model	CAP	Scans	Sec. Indices	Largest Cluster	Lear-ning	Lic.	DBaaS
HBase	Wide-Column	CP	Over Row Key	✗	~700	1/4	Apache	✗ (EMR)
MongoDB	Doc-ument	CP	yes	✓	>100 <500	4/4	GPL	mongoHQ
Riak	Key-Value	AP	✗	✓	~60	3/4	Apache	✗ (Softlayer)
Cassandra	[illegible]	AP	With [illegible]	✓	>300 [illegible]	2/4	Apache	instaclustr
Redis	[illegible]	CA	Lists etc.			4/4	BSD	Amazon ElastiCache

And there are many more:

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

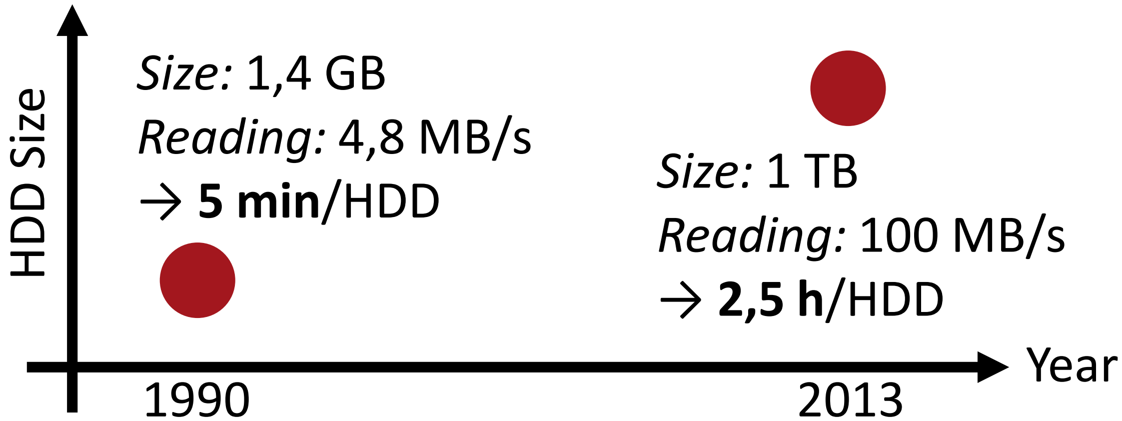
Proprietary Database services

Summary

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

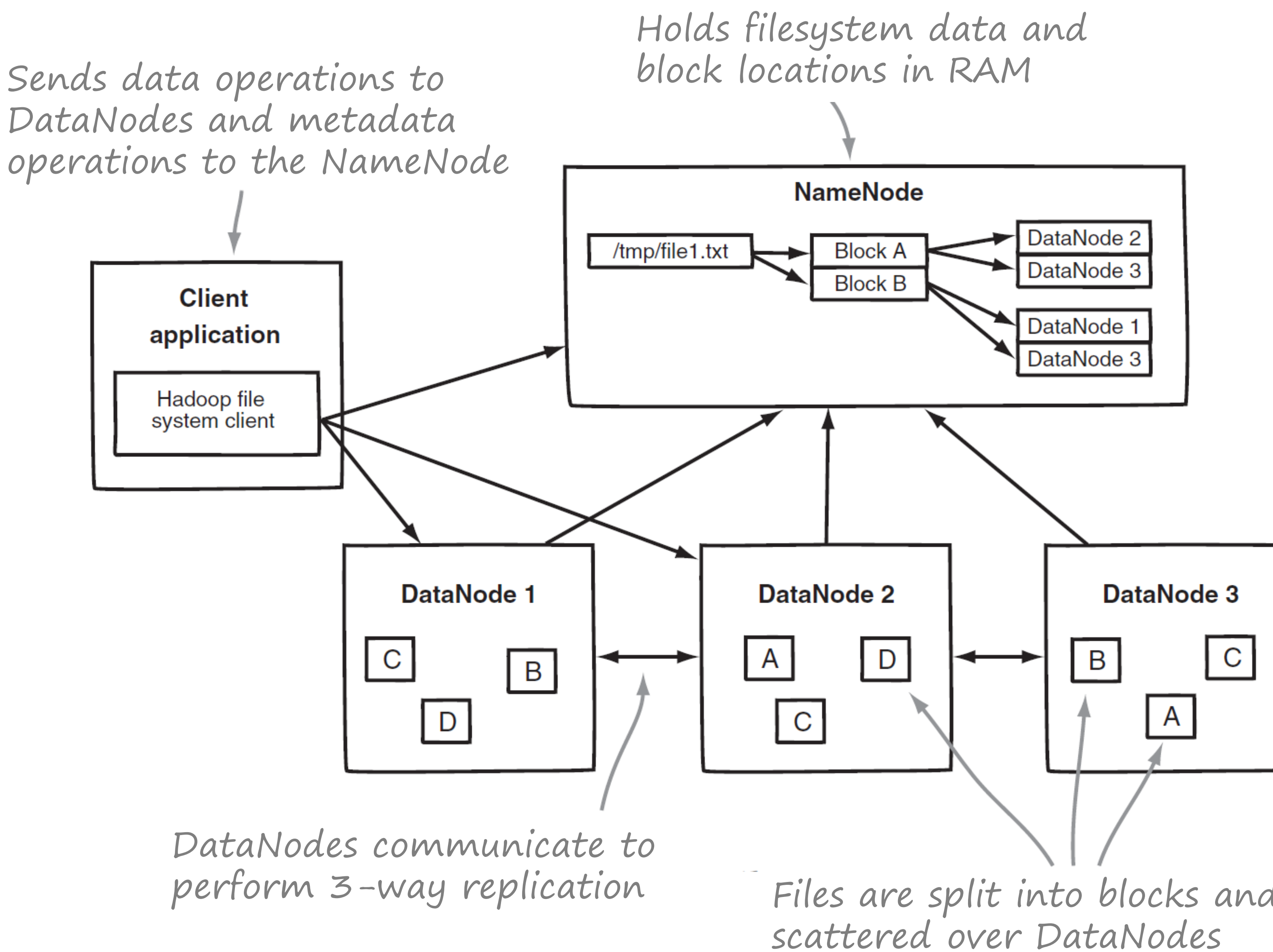
Big Data Frameworks

Hadoop Distributed FS (CP)



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

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



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

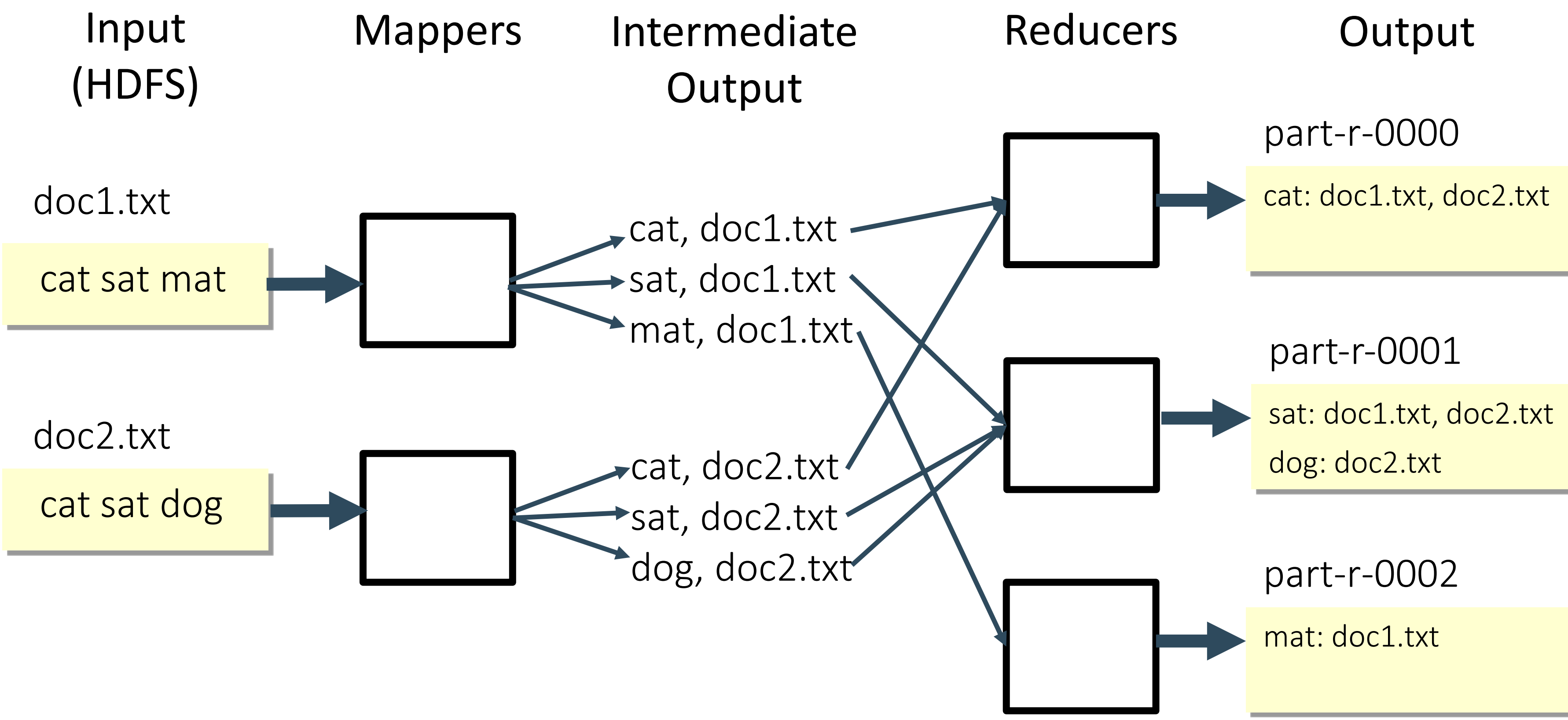
Hadoop

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

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

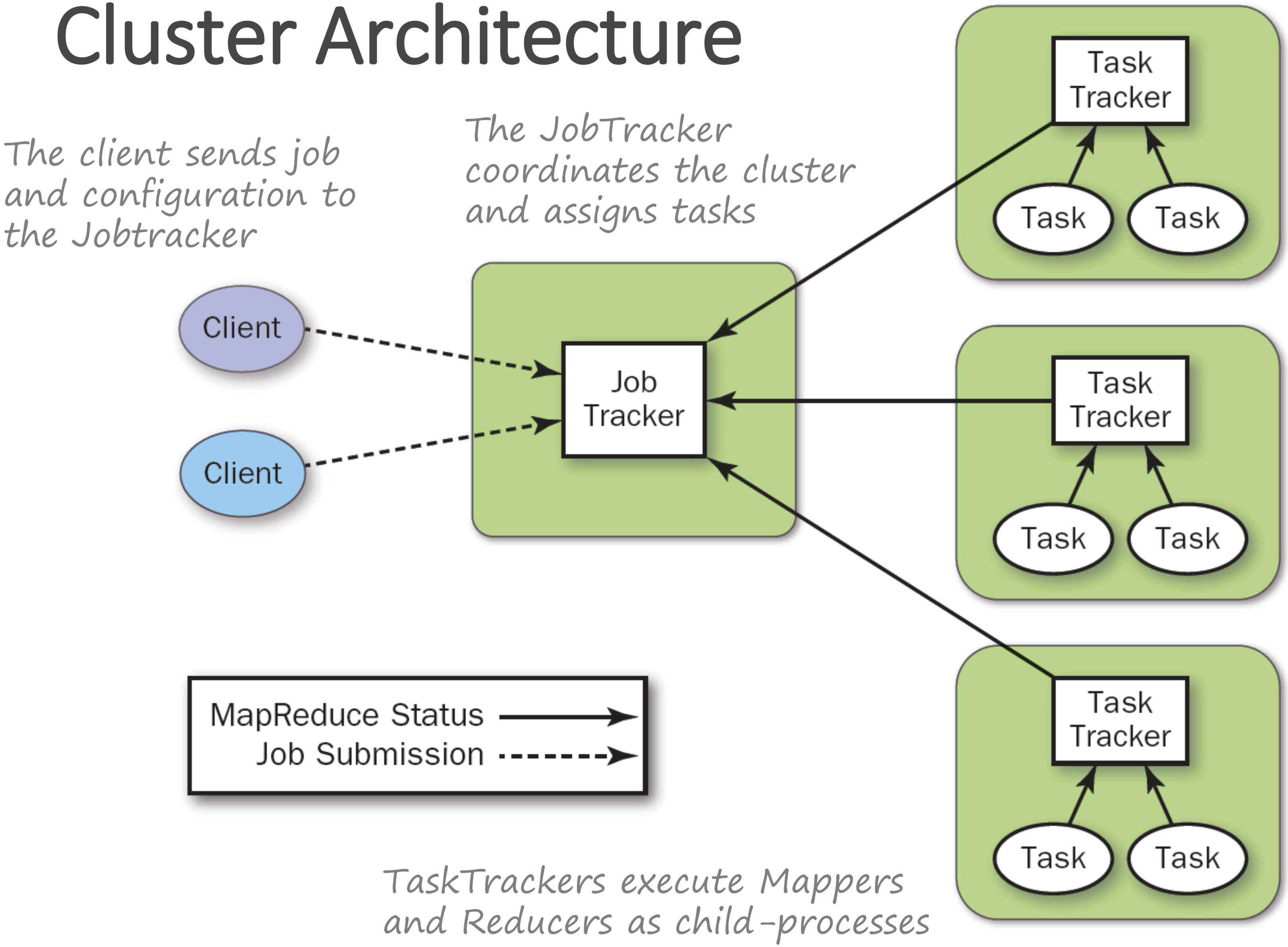
http://de.slideshare.net/cultureofperformance/gartner-predictions-for-hadoop-predictions

MapReduce: Example
Constructing a reverse-index



Holmes, Alex. Hadoop in Practice

Cluster Architecture

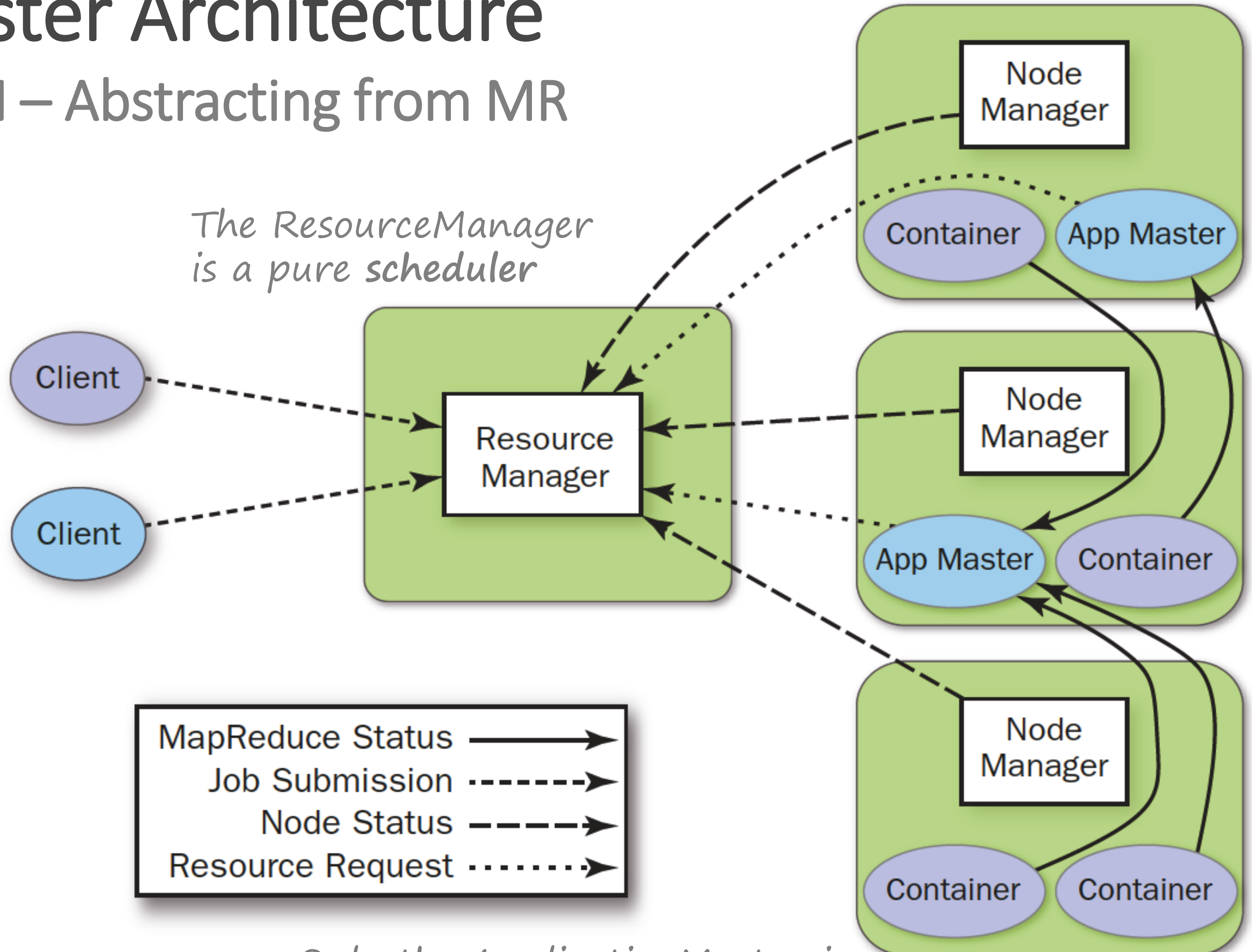


Arun Murthy “Apache Haddop YARN”

Cluster Architecture

YARN – Abstracting from MR

*The ResourceManager is a pure **scheduler***



Only the ApplicationMaster is Framework specific (e.g. MR)

Arun Murthy "Apache Haddop YARN"

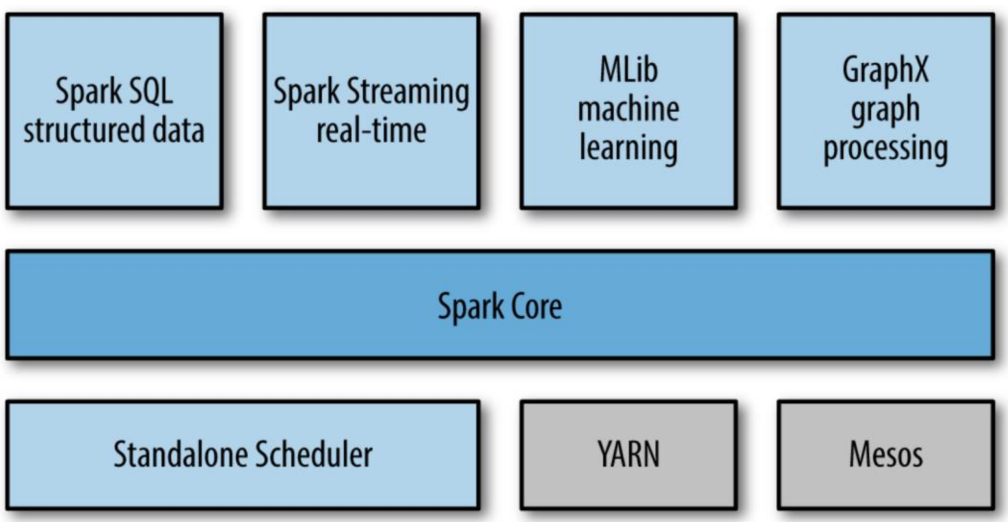
Summary: Hadoop Ecosystem

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

Spark

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

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



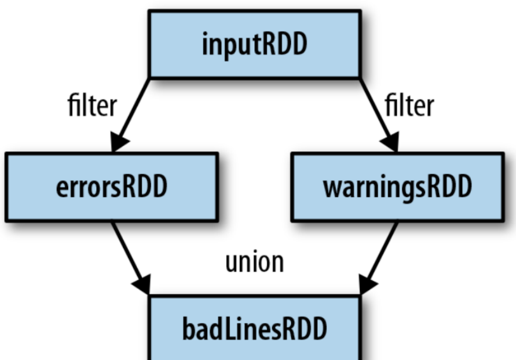
M. Zaharia, M. Chowdhury, T. Das, et al. „Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing"

Spark

Example RDD Evaluation

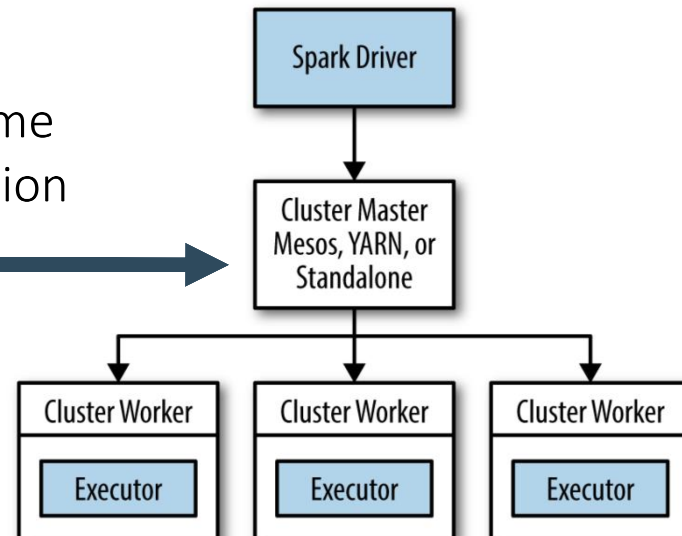
- **Transformations**: RDD → RDD
- **Actions**: Reports an operation

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



RDD Lineage

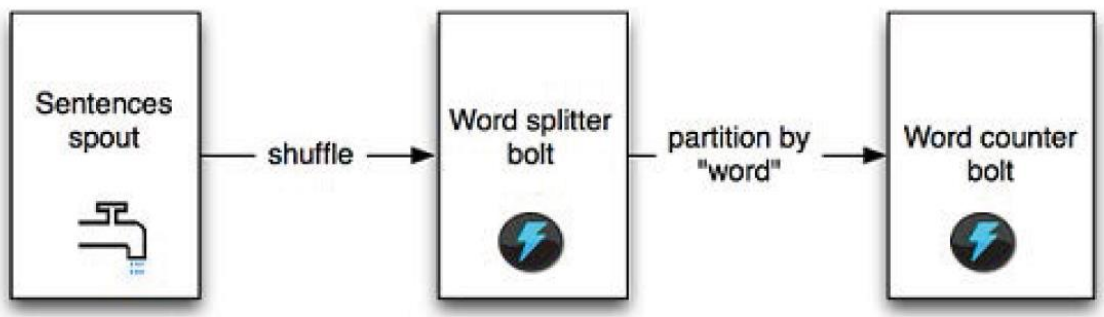
Runtime Execution



H. Karau et al. „*Learning Spark*"

Storm

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



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

Nathan Marz „Big Data"

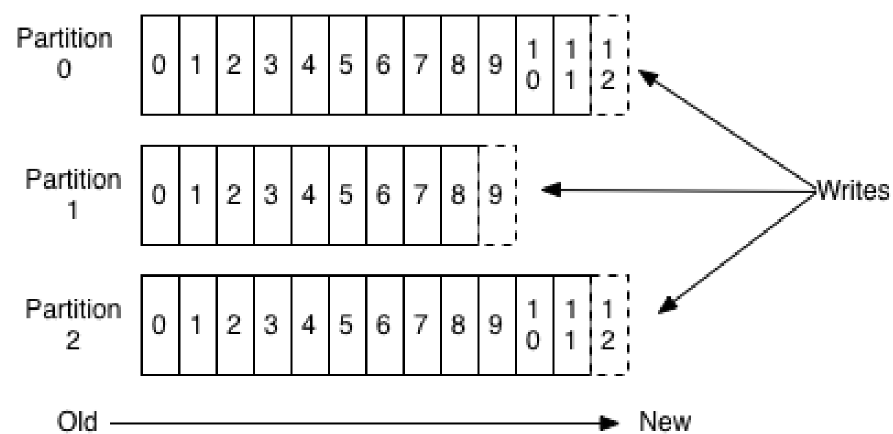
Kafka

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

Apache Kafka

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

Anatomy of a Topic



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