



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](https://www.slideshare.net/felixgessert)

Article: medium.com/baqend-blog

Outline



NoSQL Foundations and Motivation



The NoSQL Toolbox: Common Techniques

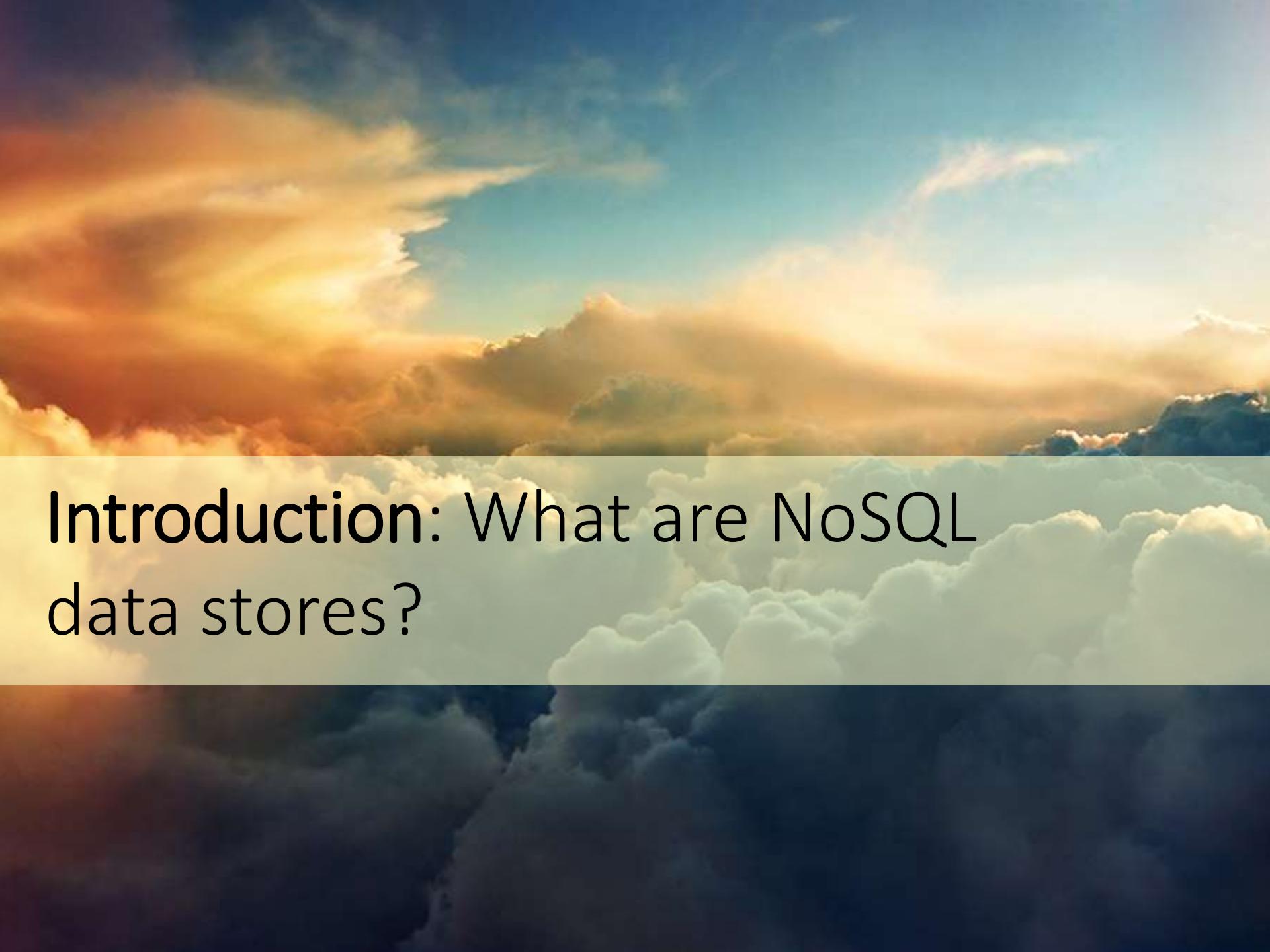


NoSQL Systems & Decision Guidance



Scalable Real-Time Databases and Processing

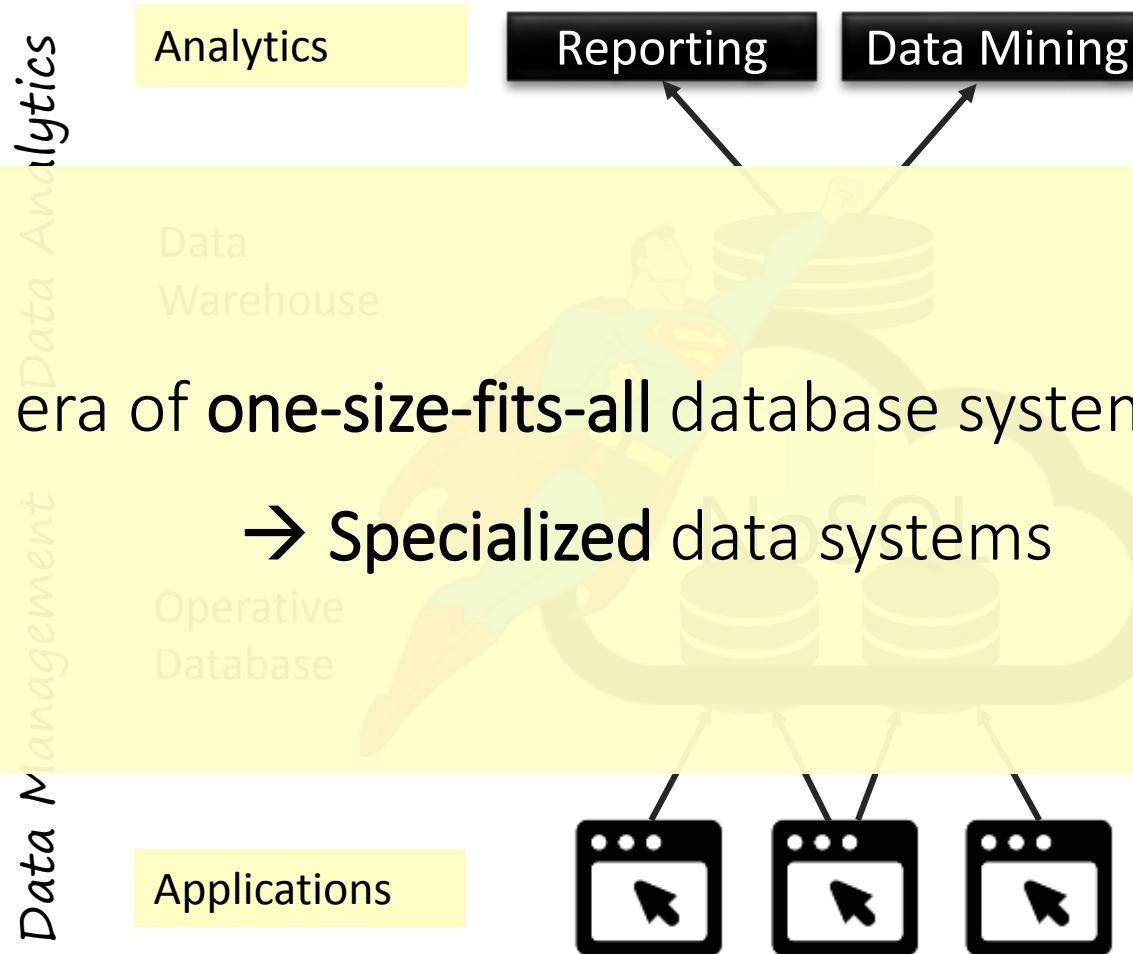
- The Database Explosion
- NoSQL: Motivation and Origins
- The 4 Classes of NoSQL Databases:
 - Key-Value Stores
 - Wide-Column Stores
 - Document Stores
 - Graph Databases
- CAP Theorem

The background of the slide features a wide-angle photograph of a sky at either dawn or dusk. The upper portion of the image is filled with wispy, orange and yellow clouds, while the lower portion shows more dense, white and grey cumulus clouds against a darker blue sky.

Introduction: What are NoSQL data stores?

Architecture

Typical Data Architecture:



The Database Explosion

Sweetspots



RDBMS

General-purpose
ACID transactions



Wide-Column Store

Long scans over
structured data



Graph Database

Graph algorithms
& queries



Parallel DWH

Aggregations/OLAP for
massive data amounts



Document Store

Deeply nested
data models



In-Memory KV-Store

Counting & statistics



NewSQL

High throughput
relational OLTP



Key-Value Store

Large-scale
session storage



Wide-Column Store

Massive user-
generated content

The Database Explosion

Cloud-Database Sweetspots



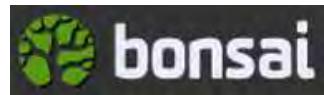
Realtime BaaS

Communication and collaboration



Wide-Column Store

Very large tables



Managed NoSQL

Full-Text Search



Amazon RDS

Managed RDBMS

General-purpose ACID transactions



**Amazon
DynamoDB**

Wide-Column Store

Massive user-generated content



Google Cloud Storage

Object Store

Massive File Storage



**Amazon
ElastiCache**

Managed Cache

Caching and transient storage



Backend-as-a-Service

Small Websites and Apps



**Amazon Elastic
MapReduce**

Hadoop-as-a-Service

Big Data Analytics

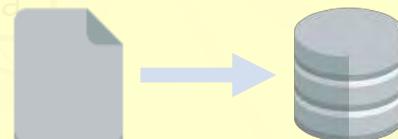
How to choose a database system?

Many Potential Candidates



Question in this tutorial:

How to approach the decision problem?

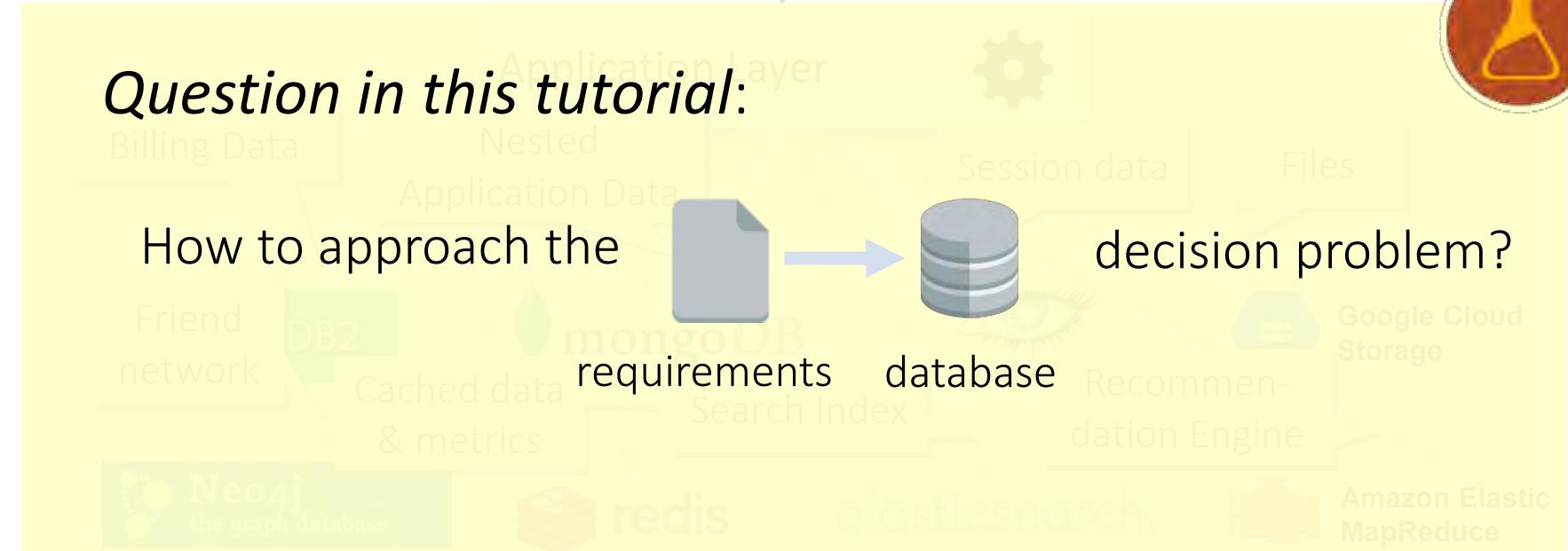


requirements database

Friend network



Cached data & metrics



NoSQL Databases

- ▶ „NoSQL“ term coined in 2009
 - ▶ Interpretation: „Not Only 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



Wicc Column Store / Column Families

Hadoop / HBase API: Java / any writer, Protocol: any write call, Query Method: MapReduce Java / any exec, Replication: HDFS Replication, Written in: Java, Concurrency: 1, Misc: Links: 3 Books [U-2-3] [Cassandra](#) massively scalable, partitioned row store.

masterless architecture, linear scale performance, no single points of failure, read/write support across multiple data centers & cloud availability zones, API / Query Method, CQL and Thrift, replication: p2c-to-p2c written in: java, Concurrency: tunable consistency, Miss: built-in data compression, MapReduce support, primary/secondary indexes, security features, Links: Documentation, Plugins, Company.

HDFS HDFS is a distributed file system designed for large-scale distributed processing. It is highly reliable, fault-tolerant, and provides high performance. It is built on top of the MapReduce framework.

Apache Accumulo Accumulo is based on BigTable and is built on top of **Hadoop**, **Zookeeper**, and **Thrift**. It features integrated column-oriented storage, row-level security, cell-based access control, mergeable compression, and fast scan operations.

Amazon SimpleDB Misc: not open source / part of AWS.

- Cloud** Google Big Data clone like HBase, [Apache](#)
- Cloudera** Professional Software & Services based on Hadoop.
- HPC** from [LexisNexis](#), [Info](#), [article](#)
- Stratosphere** (research system) massive parallel & flexible execution, MR generalization and extension ([paper](#), [poster](#)).
[OpenComputing, Qbase, HDI]

Document Store

General Object-based language & MapReduce;
Replication: Master Slave & Auto-Sharding; written in C/C++ concurrency; Update in Place; Misc;
Indexing: GridFS; Freewarc + Commercial
License Links: [Talk](#), [Notes](#), [Comics](#)

Elasticsearch API: REST and many languages.
Protocol: REST, Query Method: via JSON. Replication - Sharding: automatic and configurable; written in Java; Misc: schema mapping, multi tenancy with arbitrary indices. Company and Support: [Elastic](#)

Couchbase Server API: Memcached API+protocol

[binary and ASCII], **most languages**, **Protocol:** **Memcached REST interface for cluster conf + management**, **Which in:** C/C++ - Erlang (clustering), **Replication:** P2P to P2P, **fully consistent** **Misc:** **Transparent topology changes during operation, provides memcached-compatible options, hyphenated names in the parameter**

[CouchDB API](#): JSON Protocol: REST, Query Method: MapReduce, or JavaScript Functions: Application:

Master Master, Written in: Erlang, Concurrency: MVCC,
Misc:
Links: [» 3 CouchDB books](#), [» Couch Lounge](#) (partitioning /
sharding), [» Flickr](#)

RethinkDB API: protobuf-based, Query Method:
unified chainable query language (incl. JOINs,
sub-queries, MapReduce, GroupedMapReduce)

Replication, Sync and Async Master Slave with per-table acknowledgements, Sharding, guided range-based. Written in: C++, Concurrency: MVCC, Misc: In-memory storage system with concurrent incremental

RavenDB .Net solution. Provides **HTTP/JSON** access. **LINQ** queries & **Sharding** supported. > [Mac](#)

MarkLogic Server (Incorporate commercial) API: JSON, XML, Java Protocols: HTTP, RESTQuery Method: Full Text Search, XPath, XQuery, Range, Geospatial Written

In. C++ Concurrency Shared-nothing cluster, MVCC
Misc. Petabyte-scalable, cloudable, ACID transactions, auto-sharding, failover, master-slave replication, secure with ACLs, Developer Community

Clusterpoint Server (non-real-time) API: XML, PHP, Java, .NET Protocols: HTTP, REST, native TCP/IP Query Model: full text search, XML, range and XPath queries Written in C++ Concurrency: ACID-compliant, multi-threaded, distributed, sharded, clustered Misc. Petabyte-scalable document store and full text search engine, Information ranking.

[ThruDB](#) (please help provide more facts!) Uses Apache Thrift to integrate multiple backend databases as BerkleyDB, Disk, Cloudable.

MySQL, S3.
Terrastore API: java & http, **Protocol:** http, **Language:** java, **Caching:** Range queries, Predicates, **Replication:** Partitioned with consistent hashing, **Consistency:** Per-record strict consistency, Miss:

[JasDB](#) Lightweight open source document database written in Java for high performance, runs in-memory, supports Android.

API: JSON, Java Query Method: REST OData Style
Query language: Java fluent Query API
Concurrency: Atomic document writes Indexes:
Eventually consistent indexes

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

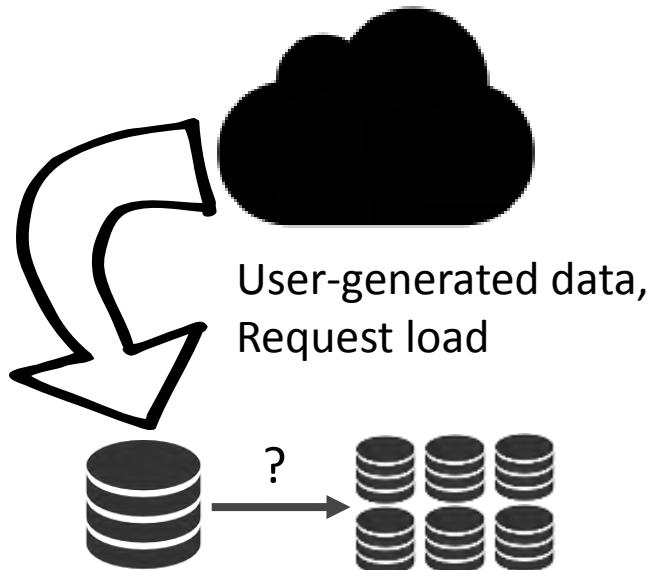
[SisoDB](#) A Document Store on top of SQL-Server.
[SDB](#) For small online databases, PHP / JSON interface.

cjondb **cjonDB API: BSON, Protocol: C++, Query Method:**

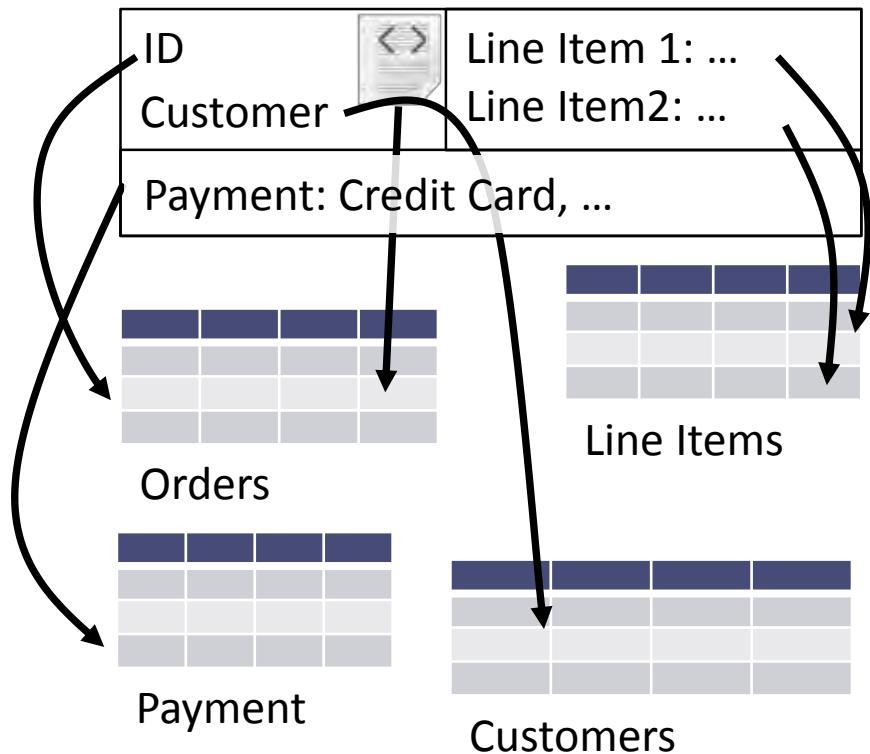
NoSQL Databases

- ▶ Two main motivations:

Scalability

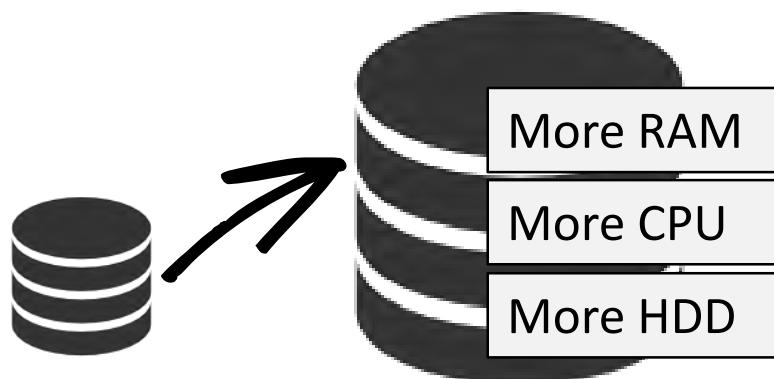


Impedance Mismatch

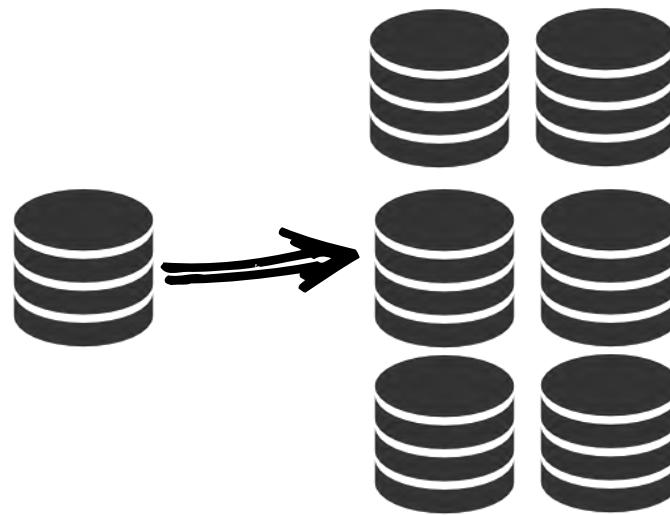


Scale-up vs Scale-out

Scale-Up (*vertical scaling*):



Scale-Out (*horizontal scaling*):

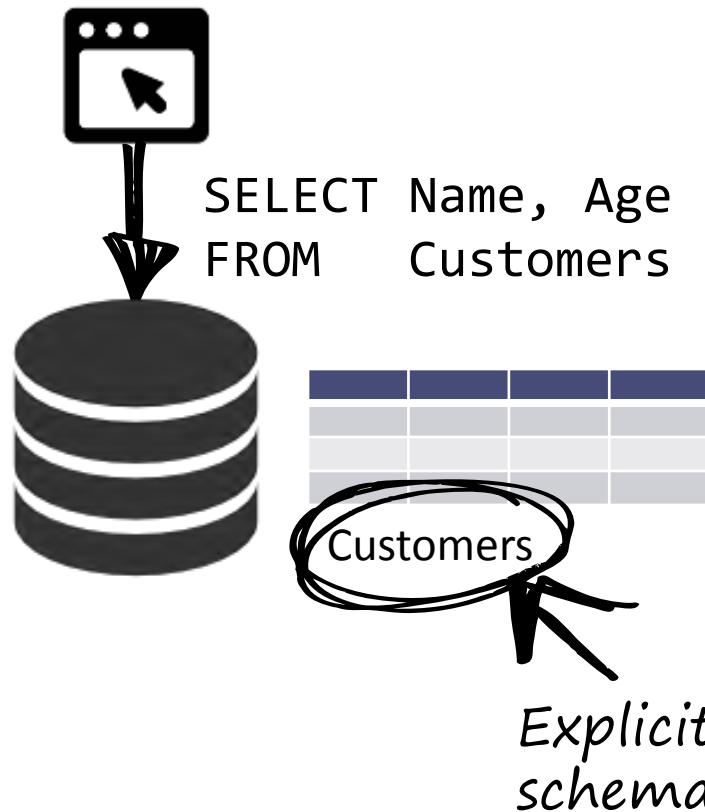


Commodity Hardware

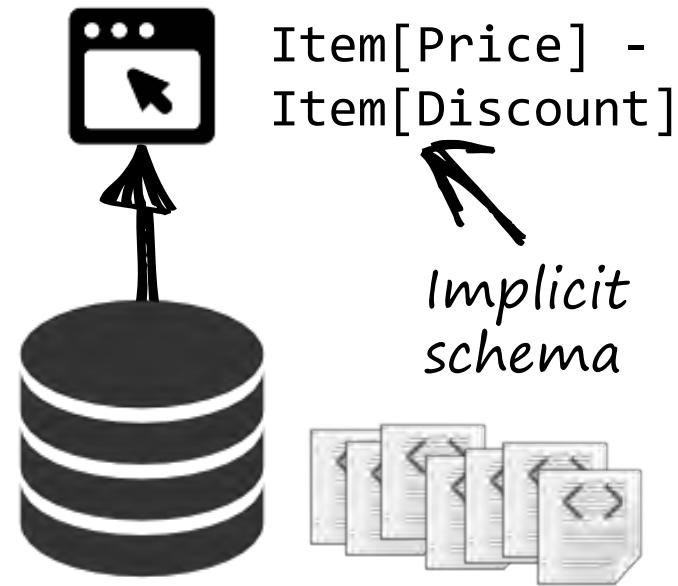
Shared-Nothing Architecture

Schemafree Data Modeling

RDBMS:



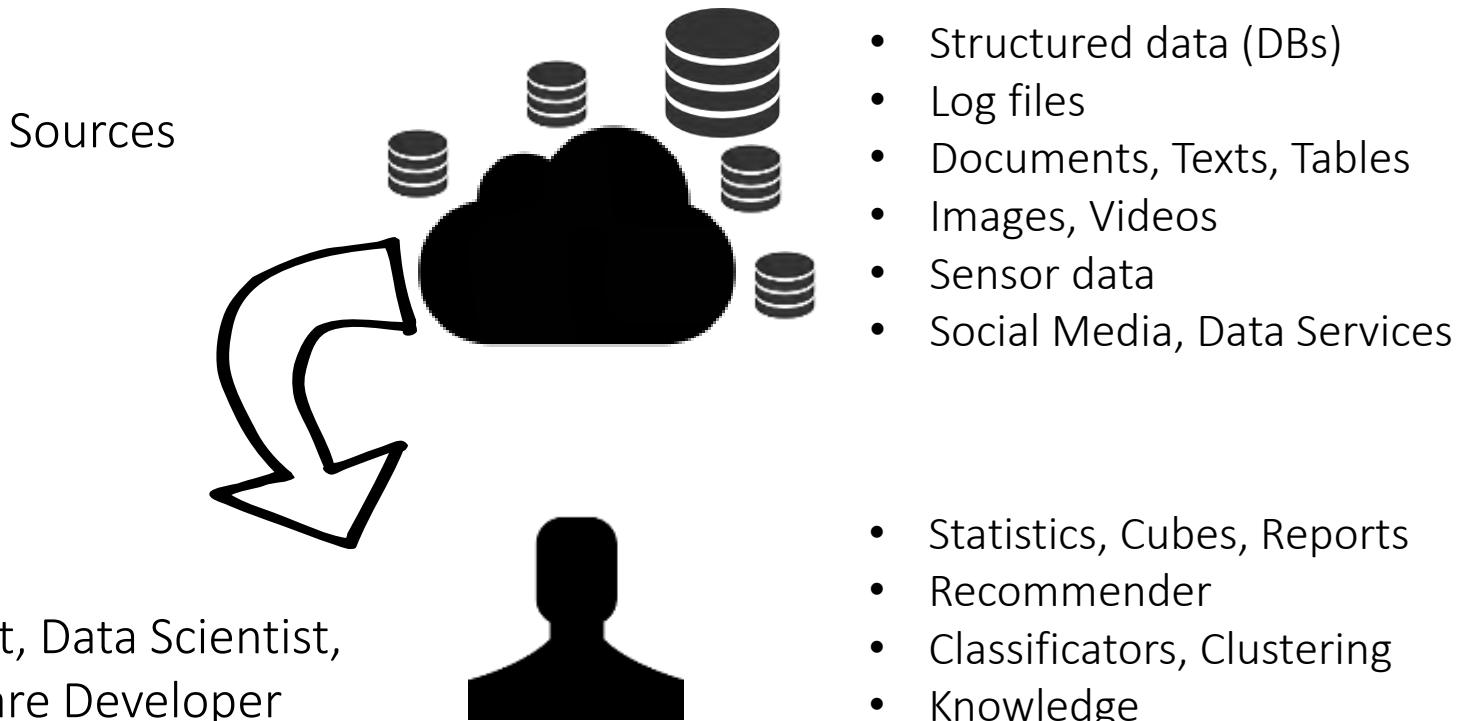
NoSQL DB:



Big Data

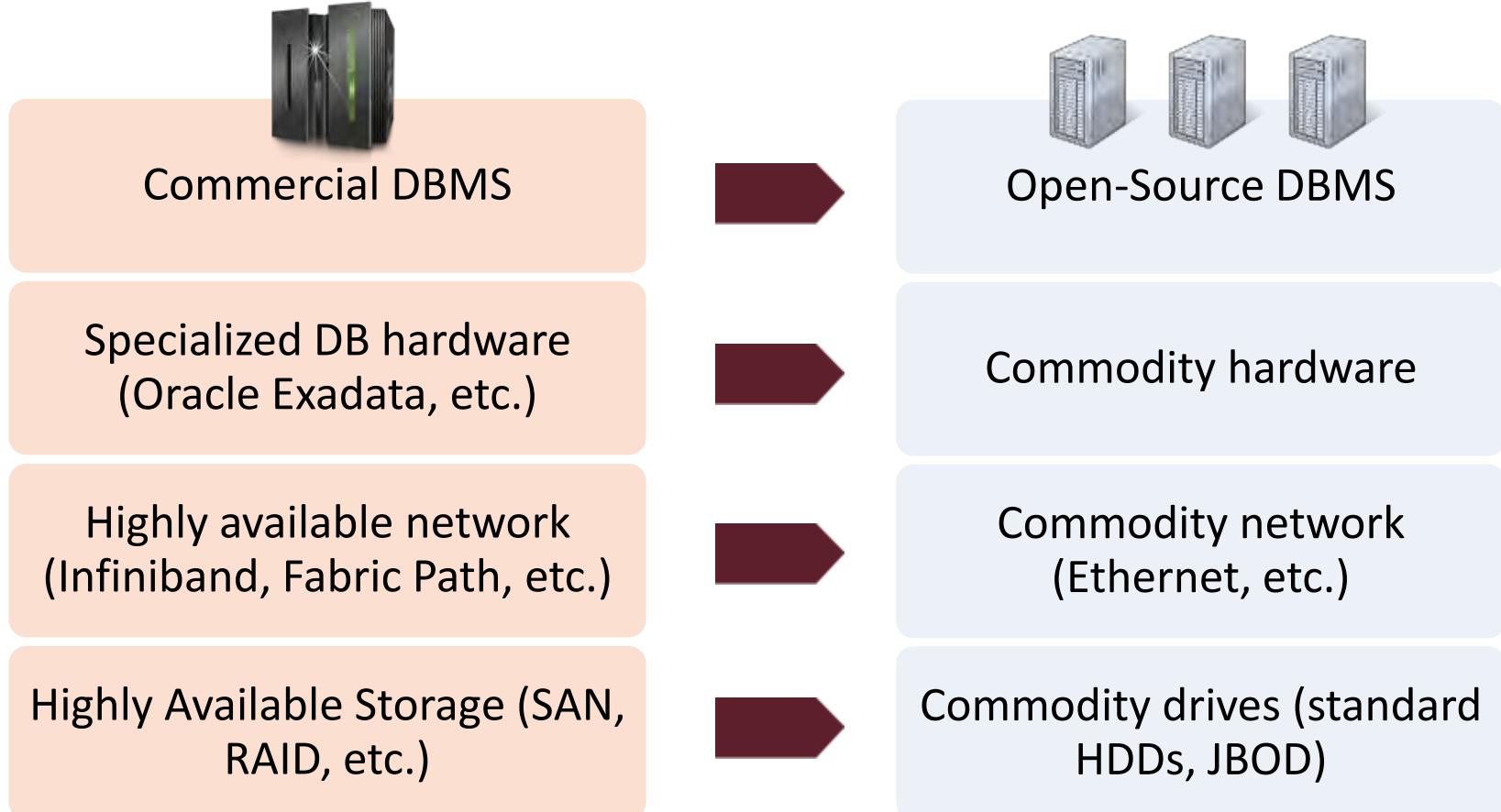
The Analytic side of NoSQL

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



NoSQL Paradigm Shift

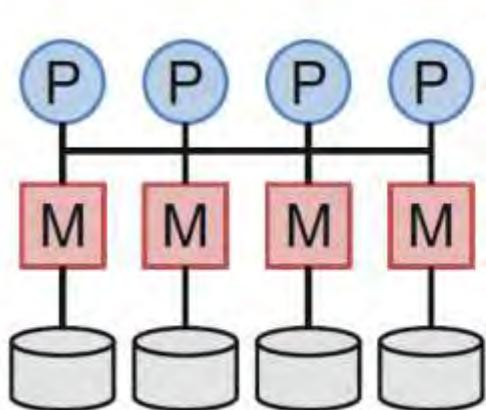
Open Source & Commodity Hardware



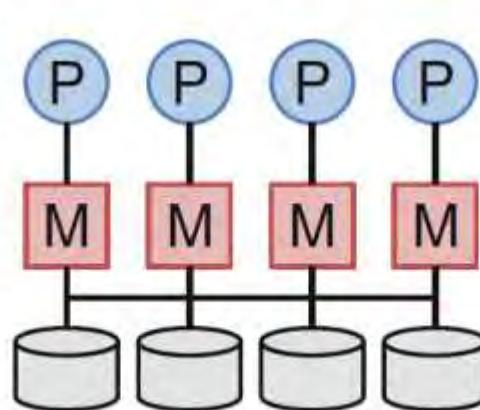
NoSQL Paradigm Shift

Shared Nothing Architectures

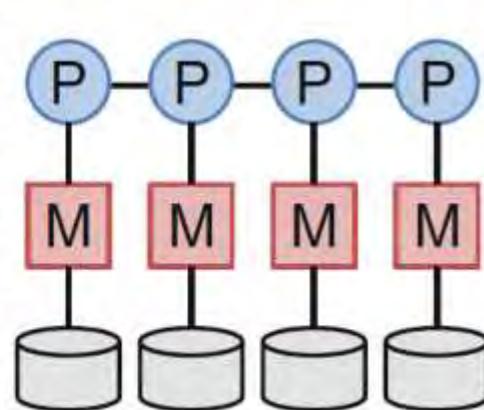
Shift towards higher distribution & less coordination:



Shared Memory
e.g. "Oracle 11g"



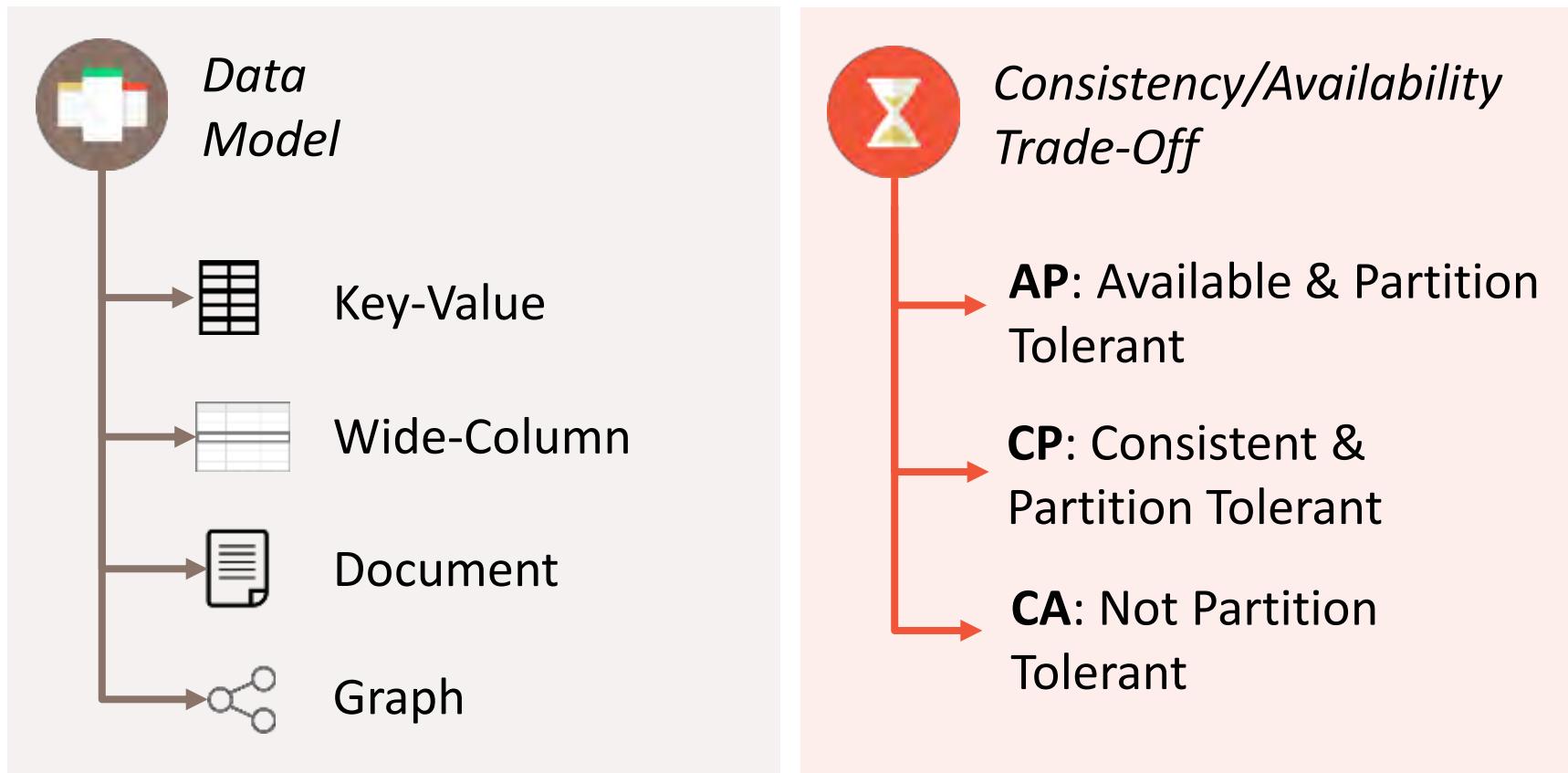
Shared Disk
e.g. "Oracle RAC"



Shared Nothing
e.g. "NoSQL"

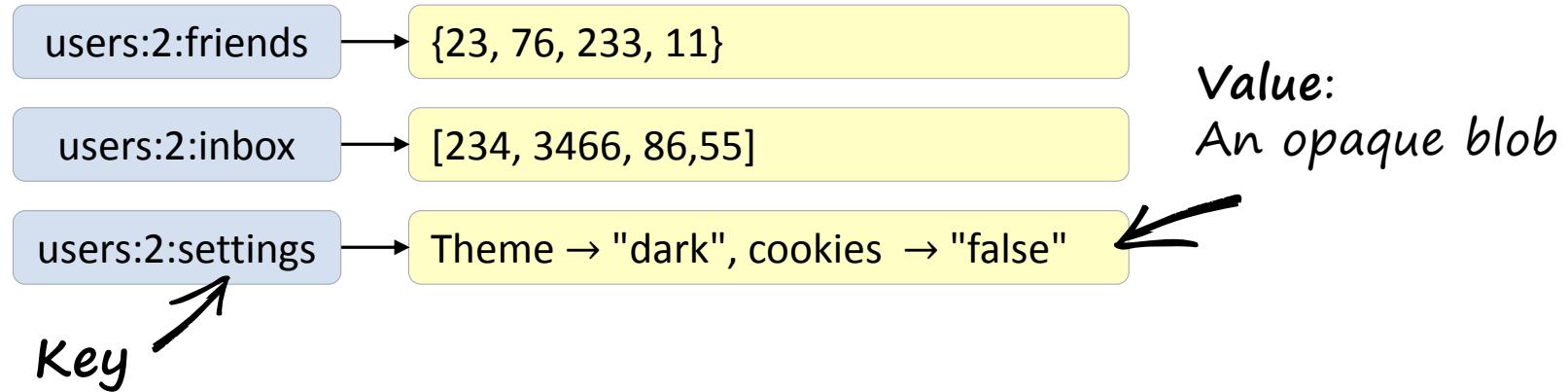
NoSQL System Classification

- ▶ Two common criteria:



Key-Value Stores

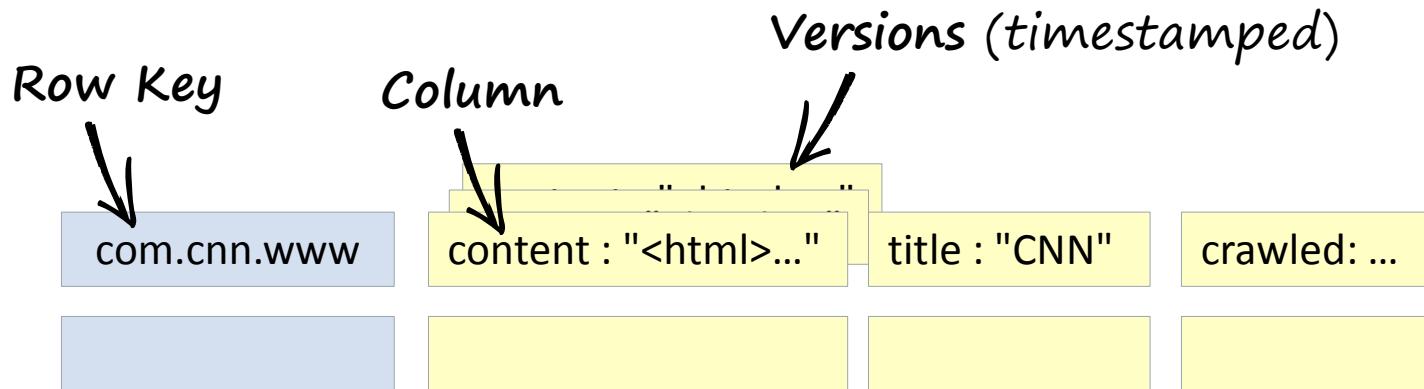
- ▶ Data model: (key) -> value
- ▶ Interface: CRUD (Create, Read, Update, Delete)



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

Wide-Column Stores

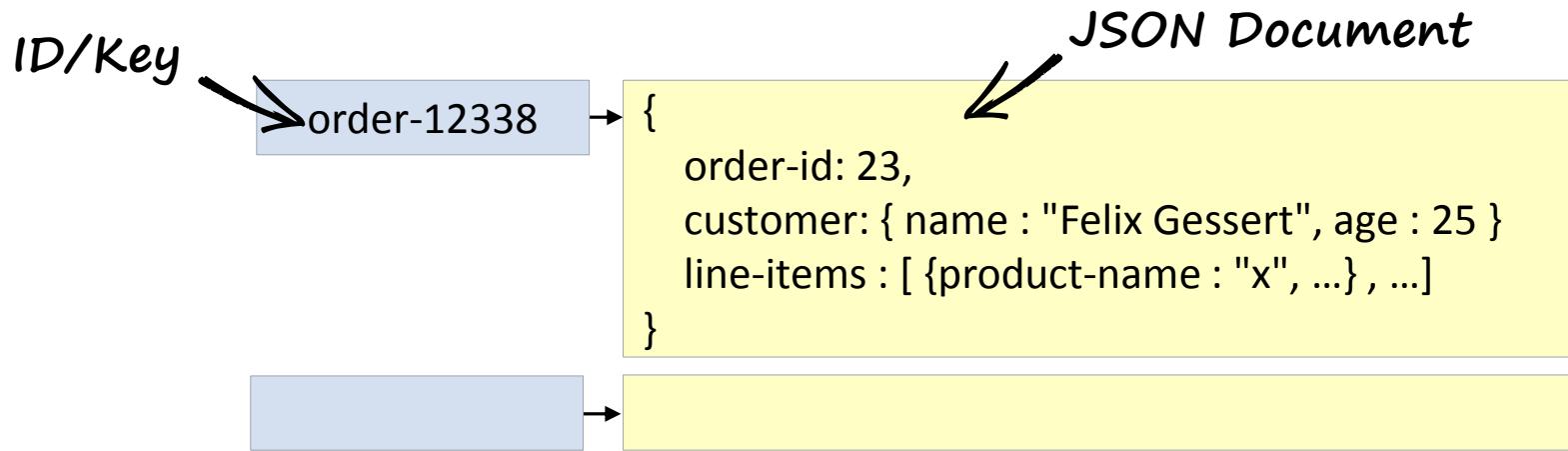
- ▶ Data model: (rowkey, column, timestamp) -> value
- ▶ Interface: CRUD, Scan



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

Document Stores

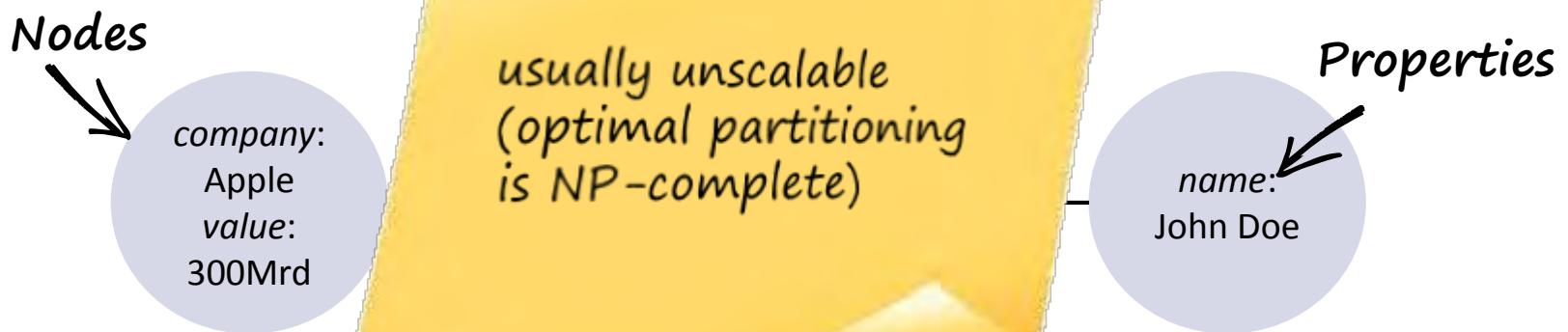
- ▶ Data model: (collection, key) -> document
- ▶ Interface: CRUD, Querys, Map-Reduce



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

Graph Databases

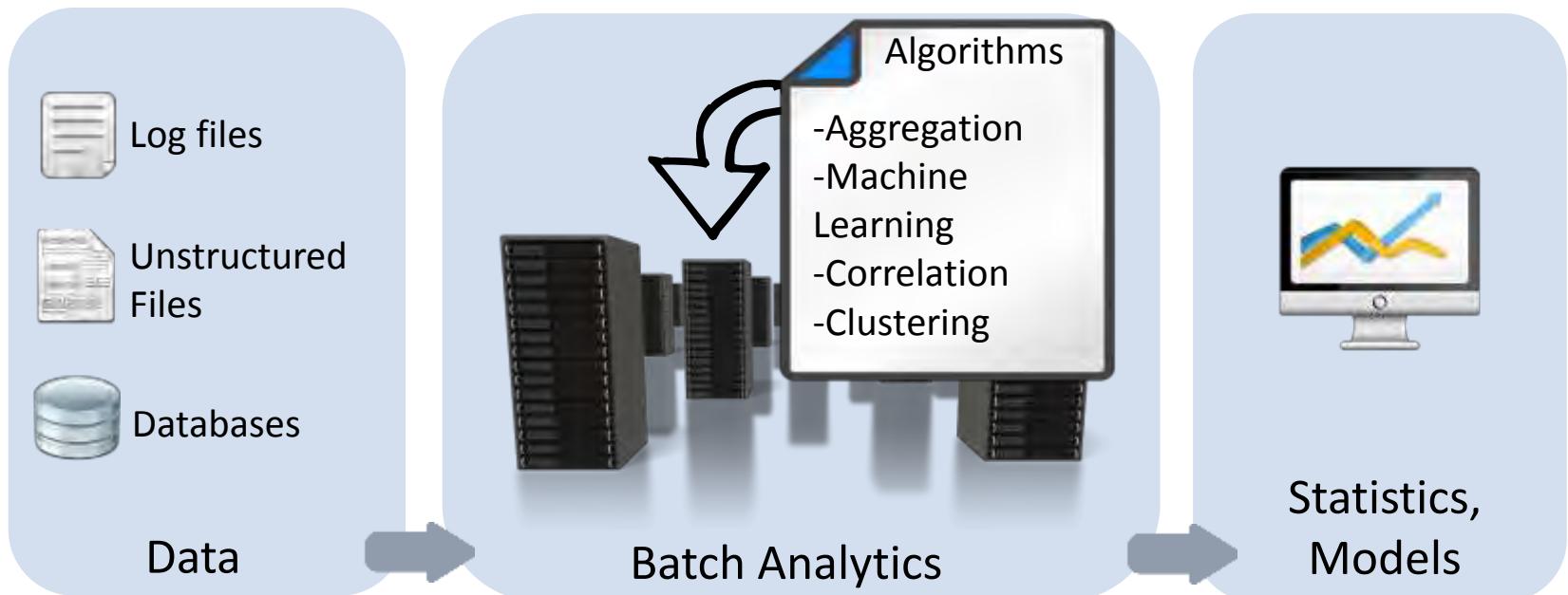
- ▶ Data model: $G = (V, E)$: Graph-Property Modell
- ▶ Interface: Traversal, transactions



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

Big Data Batch Processing

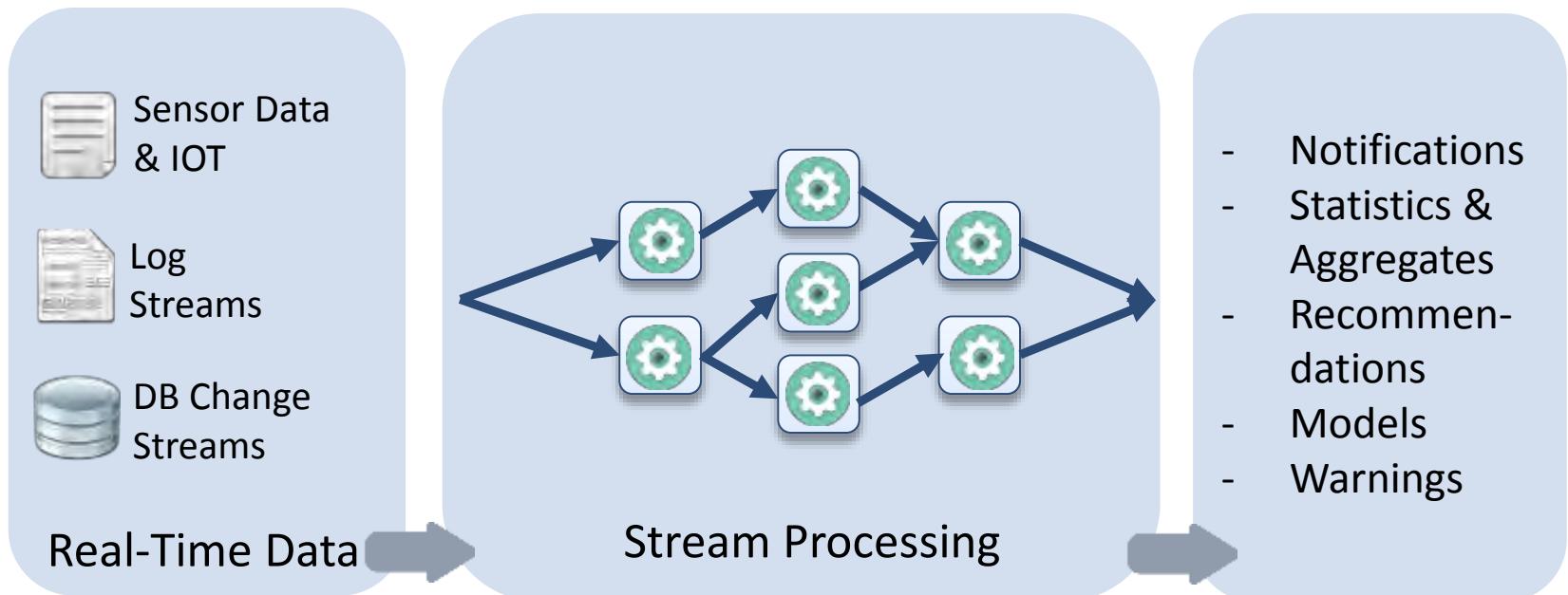
- ▶ **Data model:** arbitrary (frequently unstructured)
- ▶ Examples: Hadoop, Spark, Flink, DryadLink, Pregel



Big Data Stream Processing

Covered in Depth in the Last Part

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



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:

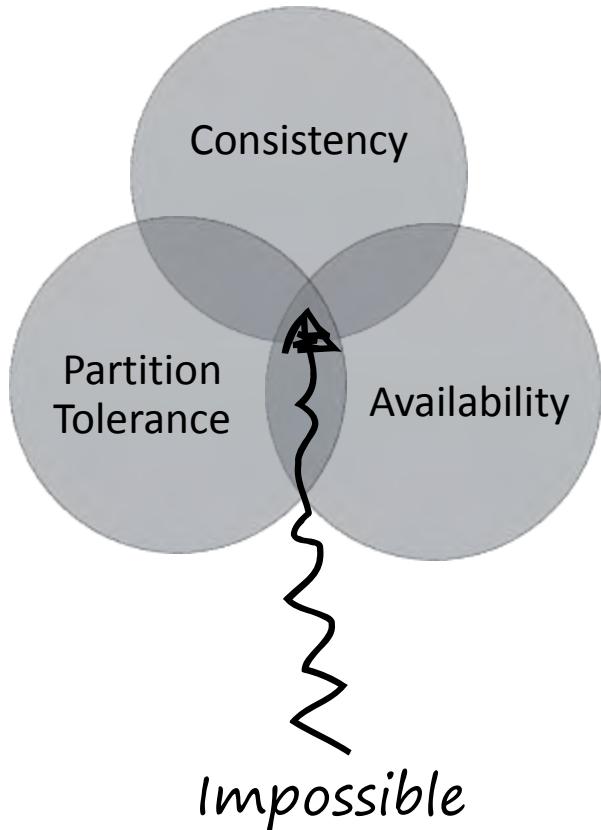
- Strong coupling of programming language and DB
- *Examples:* Versant (CA), db4o (CA), Objectivity (CA)



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 must result in correct response
- **Partition tolerance**: the system has to continue working, even under arbitrary network partitions



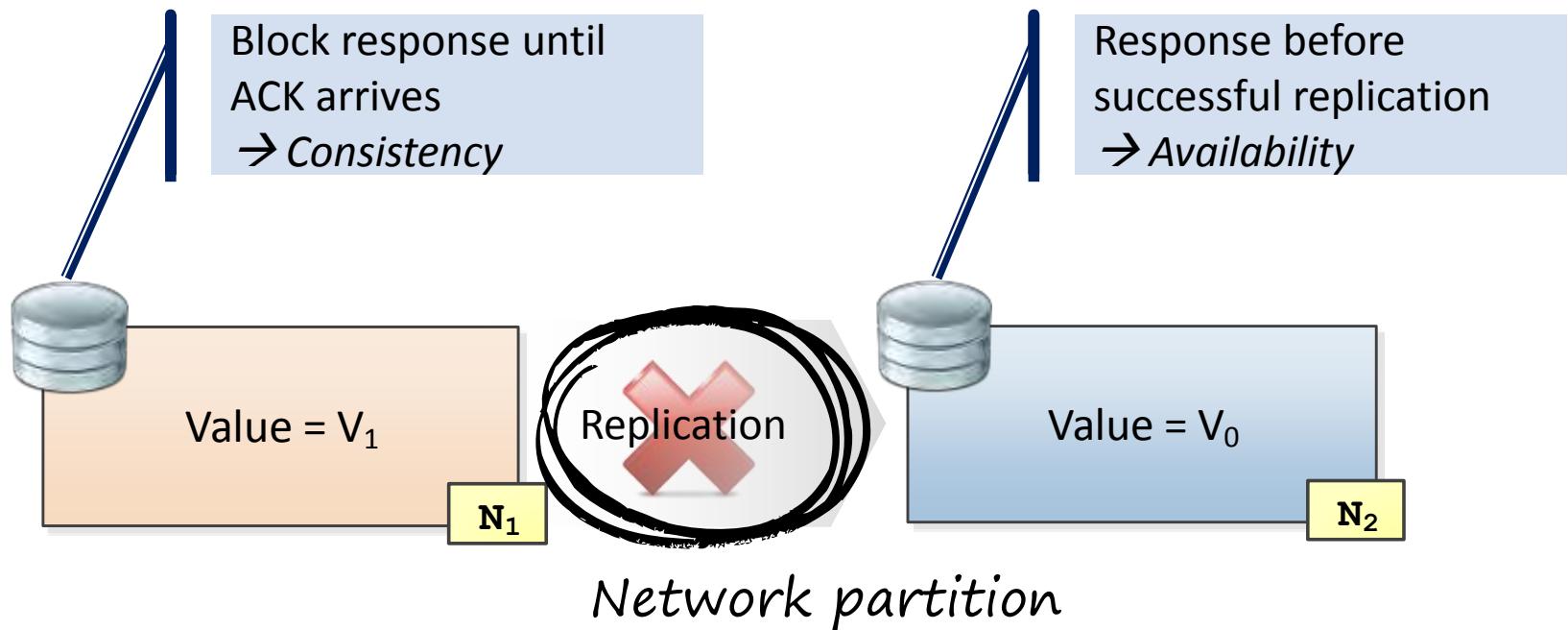
Eric Brewer, ACM-PODC Keynote, Juli 2000



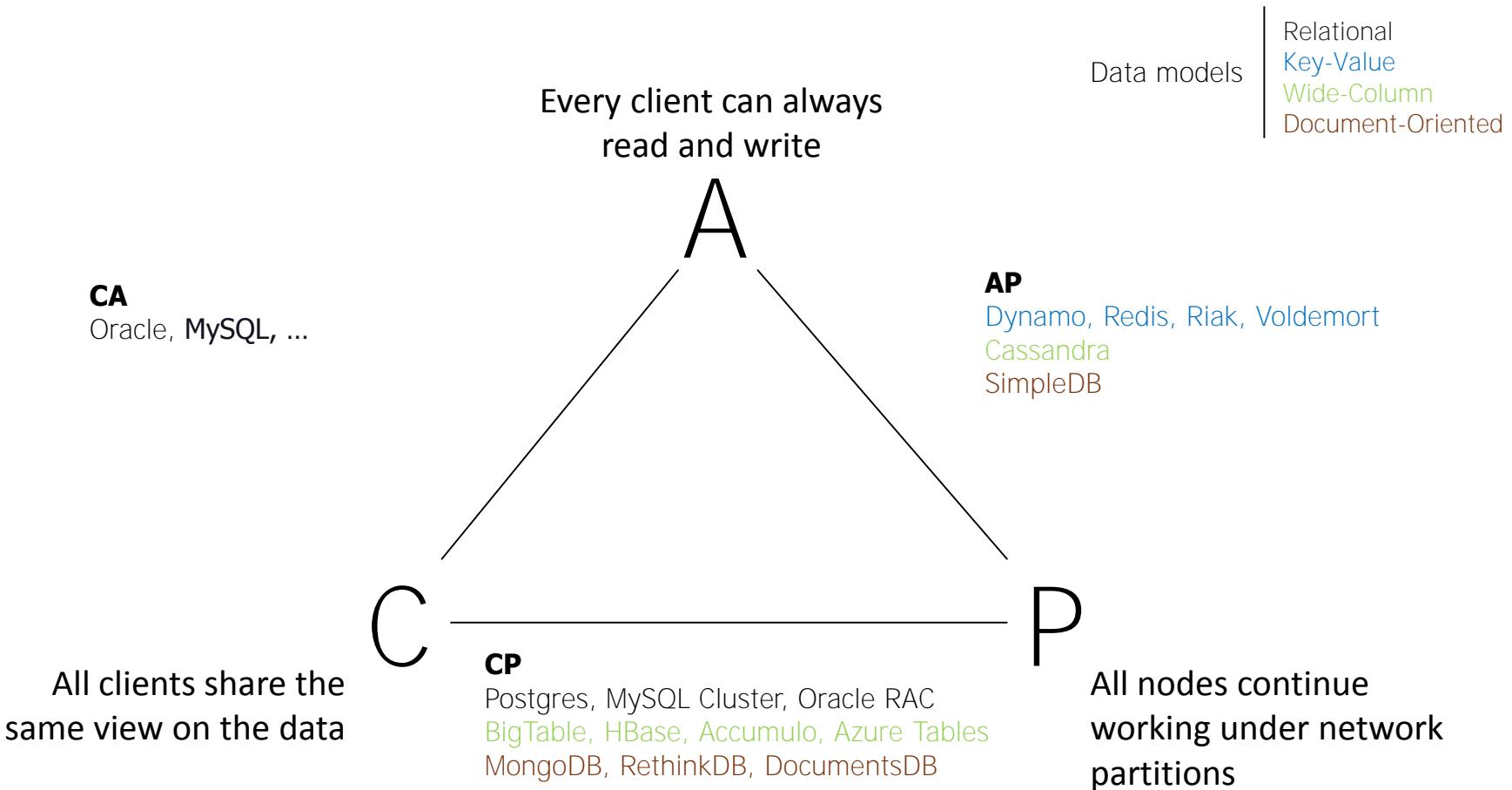
Gilbert, Lynch: Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services, SigAct News 2002

CAP-Theorem: simplified proof

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

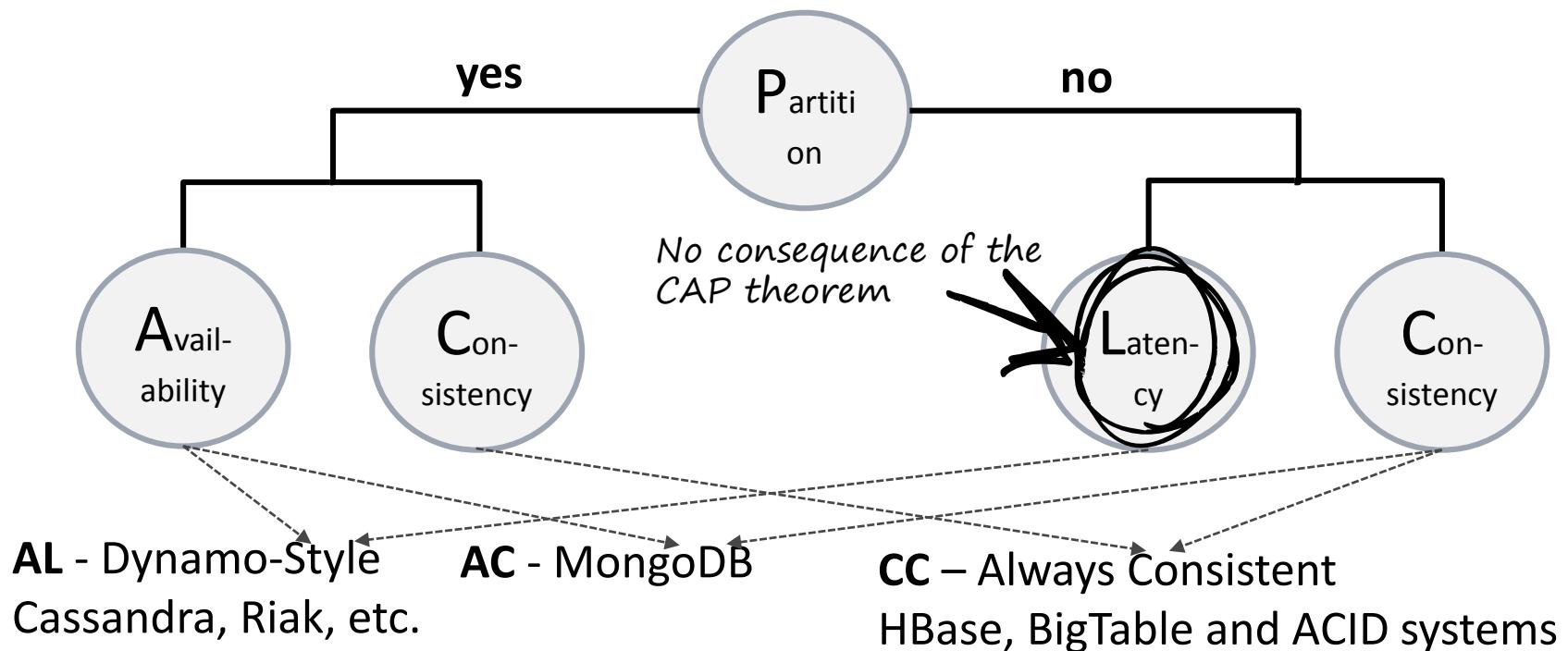


NoSQL Triangle



PACELC – an alternative CAP formulation

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

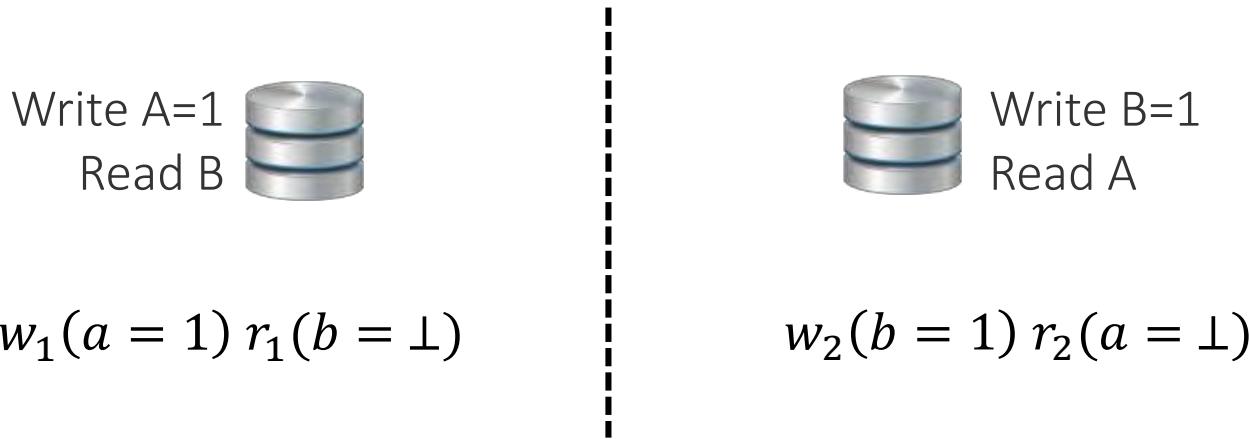


Abadi, Daniel. "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story."

Serializability

Not Highly Available Either

Global serializability and availability are incompatible:



- ▶ 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

▶ 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

Safety
Properties

▶ 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**)



Where CAP fits in

Negative Results in Distributed Computing

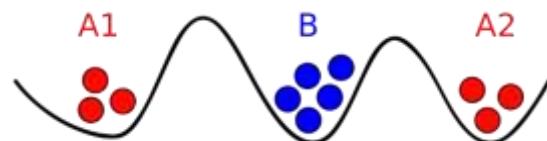
***Asynchronous Network,
Unreliable Channel***

Atomic Storage

Impossible:
CAP Theorem

Consensus

Impossible:
2 Generals Problem



***Asynchronous Network,
Reliable Channel***

Atomic Storage

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

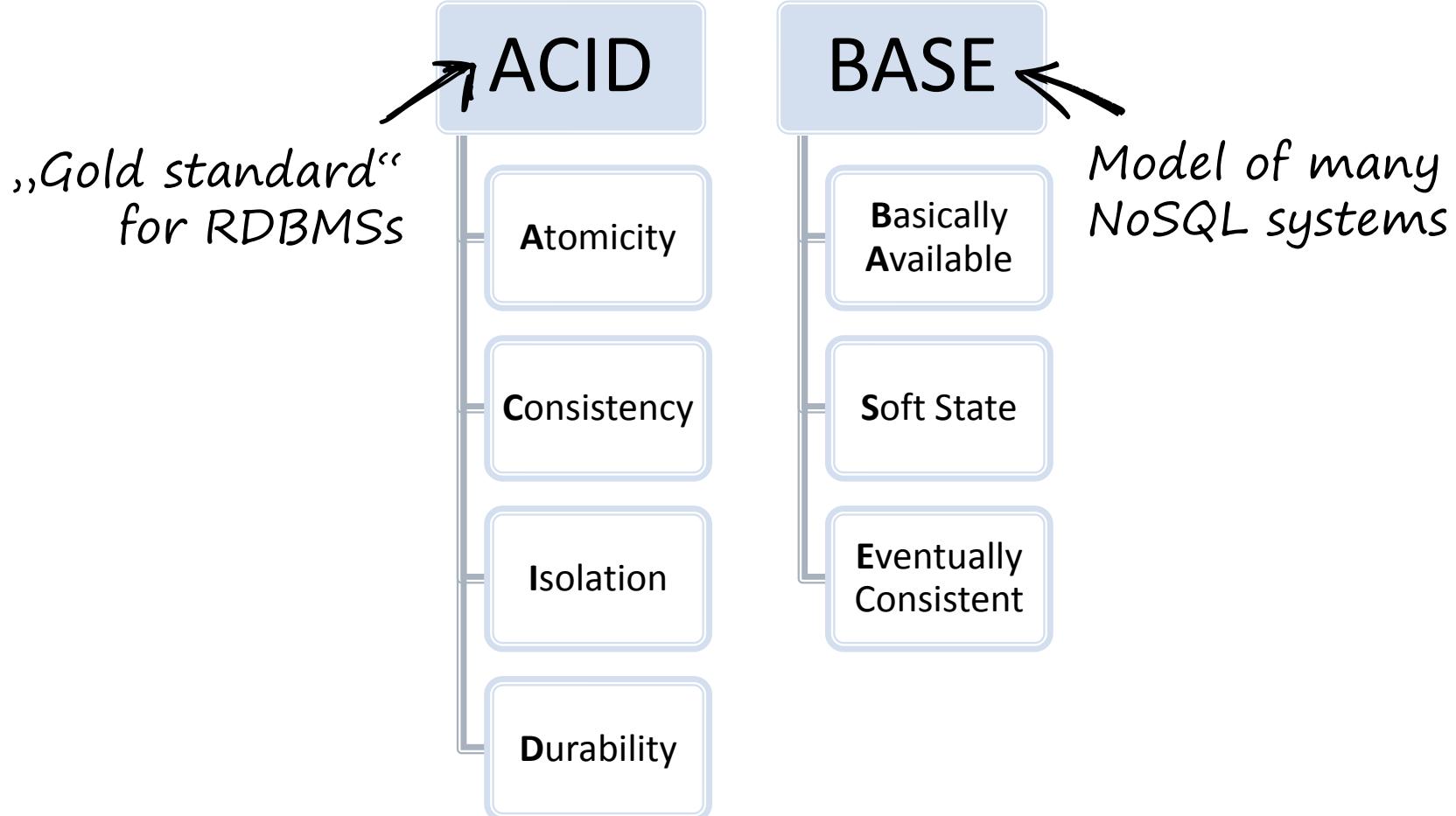
Consensus

Impossible:
Fisher Lynch Patterson (FLP)
Theorem



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

ACID vs BASE



Weaker guarantees in a database?!

Default Isolation Levels in RDBMSs

Database	Default Isolation	Maximum Isolation
Actian Ingres 10.0/10S	S	S
Aerospike		RC
Clustrix CLX 4100		?
Greenplum 4.1		S
IBM DB2 10 for z/OS		S
IBM Informix 11.50		RR
MySQL 5.6		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

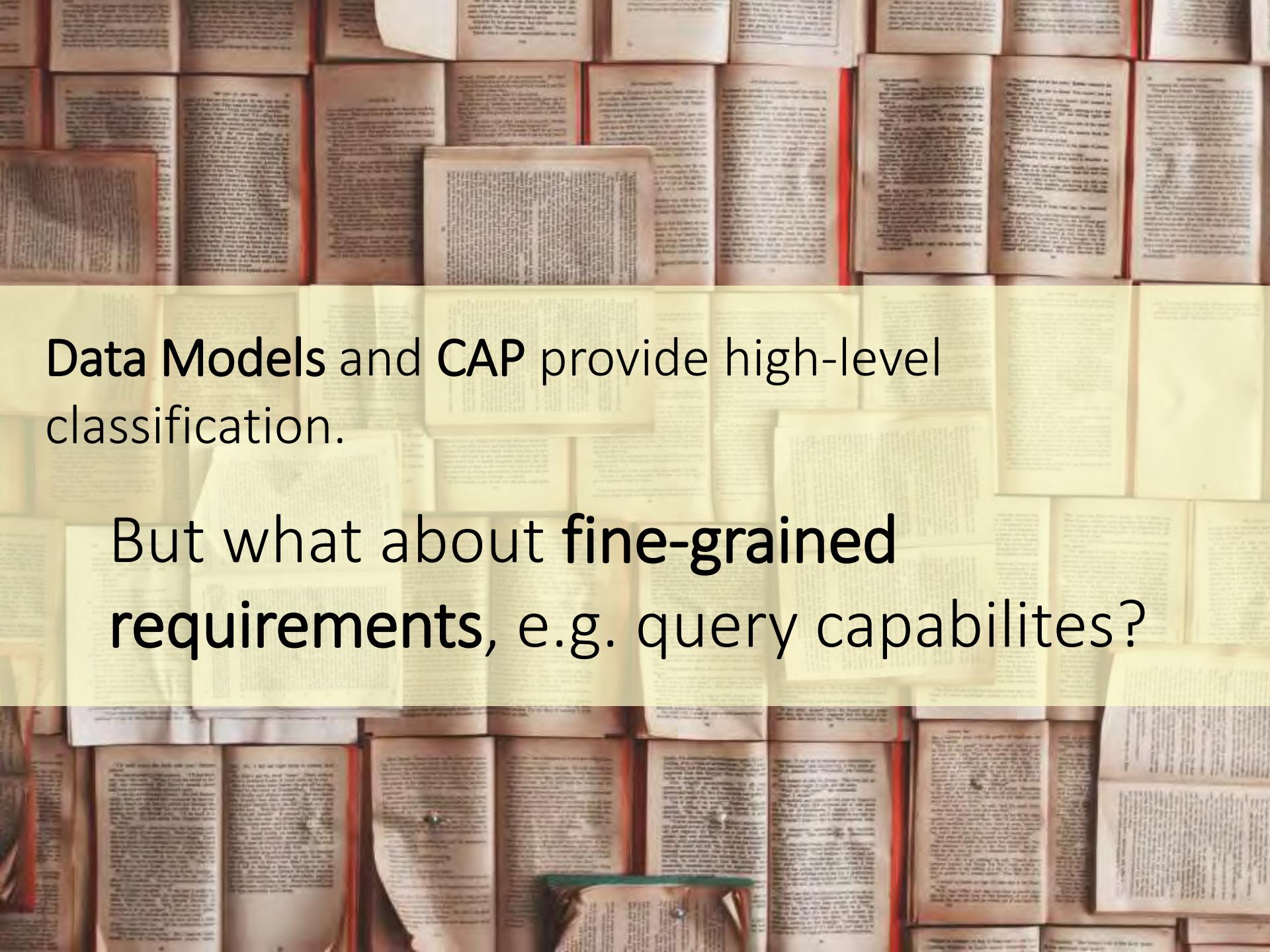
Theorem:
Depends

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 capabilities?

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 Requirements from the application



Aggregation and Analytics

enable

Techniques

Sharding

- Range-Sharding
- Hash-Sharding
- Entity-Group Sharding
- Consistent Hashing
- Shared-Disk

Replication

- Commodity Consistency
- Synchronous
- Asynchronous
- Primary Copy
- Update Replication

Storage Management

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

Query

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

enable

Non-Functional

Data Scalability

Write Scalability

Read Scalability

Operational Requirements

Consistency

Write Latency

Read Latency

Write Throughput

Read Throughput

Write Availability

Durability

Central techniques NoSQL databases employ



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

Functional

Techniques

Non-Functional

Scan Queries

ACID Transactions

Conditional or Atomic Writes

Joins

Sorting

Sharding

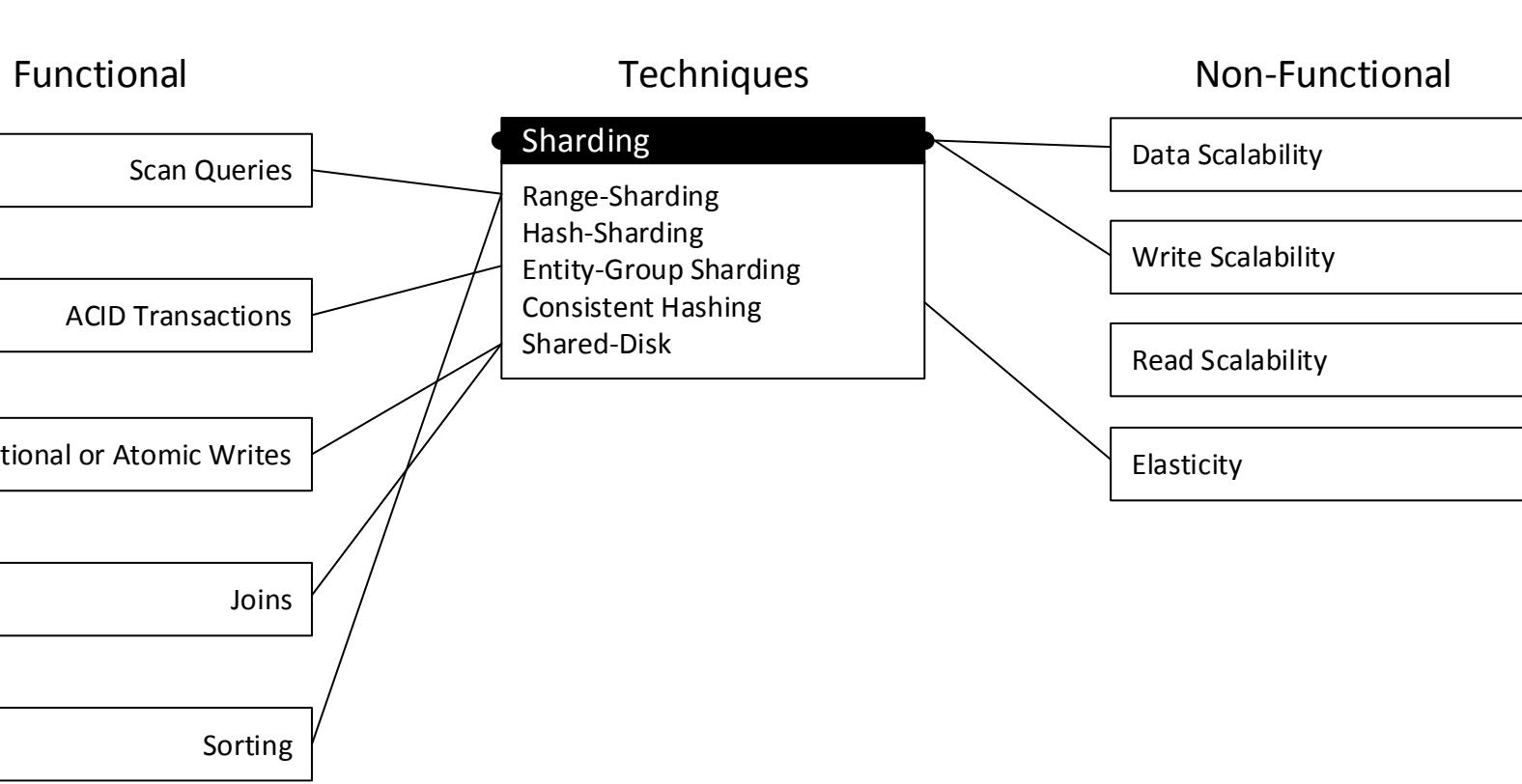
- Range-Sharding
- Hash-Sharding
- Entity-Group Sharding
- Consistent Hashing
- Shared-Disk

Data Scalability

Write Scalability

Read Scalability

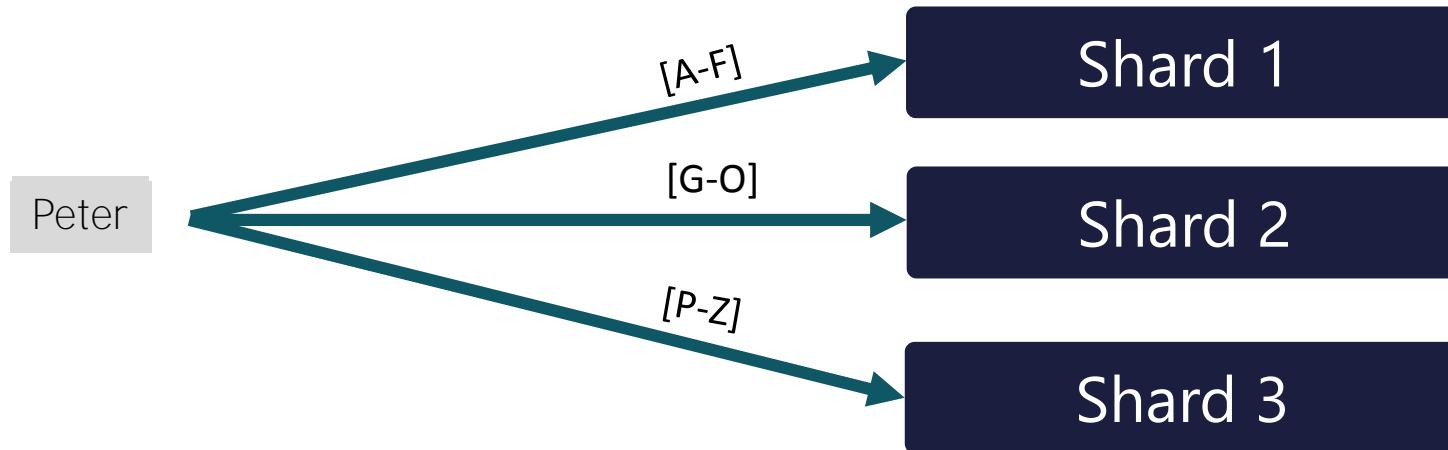
Elasticity



Sharding (aka Partitioning, Fragmentation)

Scaling Storage and Throughput

- ▶ Horizontal distribution of data over nodes



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

Sharding

Approaches

Hash-based Sharding

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

Implemented in

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

Implemented in

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

Implemented in

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

Range-based Sharding

- Assigns ranges defined over fields
- Pro: Enables *Range Scans* and *Scalability*
- Contra: Repartitioning/balancing

Entity-Group Sharding

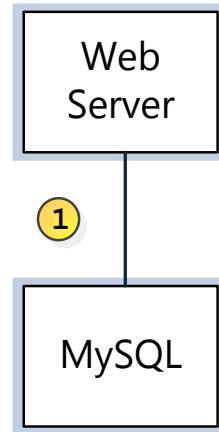
- Explicit data co-location for single entities
- Pro: Enables *ACID Transactions* and *Consistency*
- Contra: Partitioning not easily done



Problems of Application-Level Sharding

Example: Tumblr

- ▶ Caching
- ▶ Sharding from application



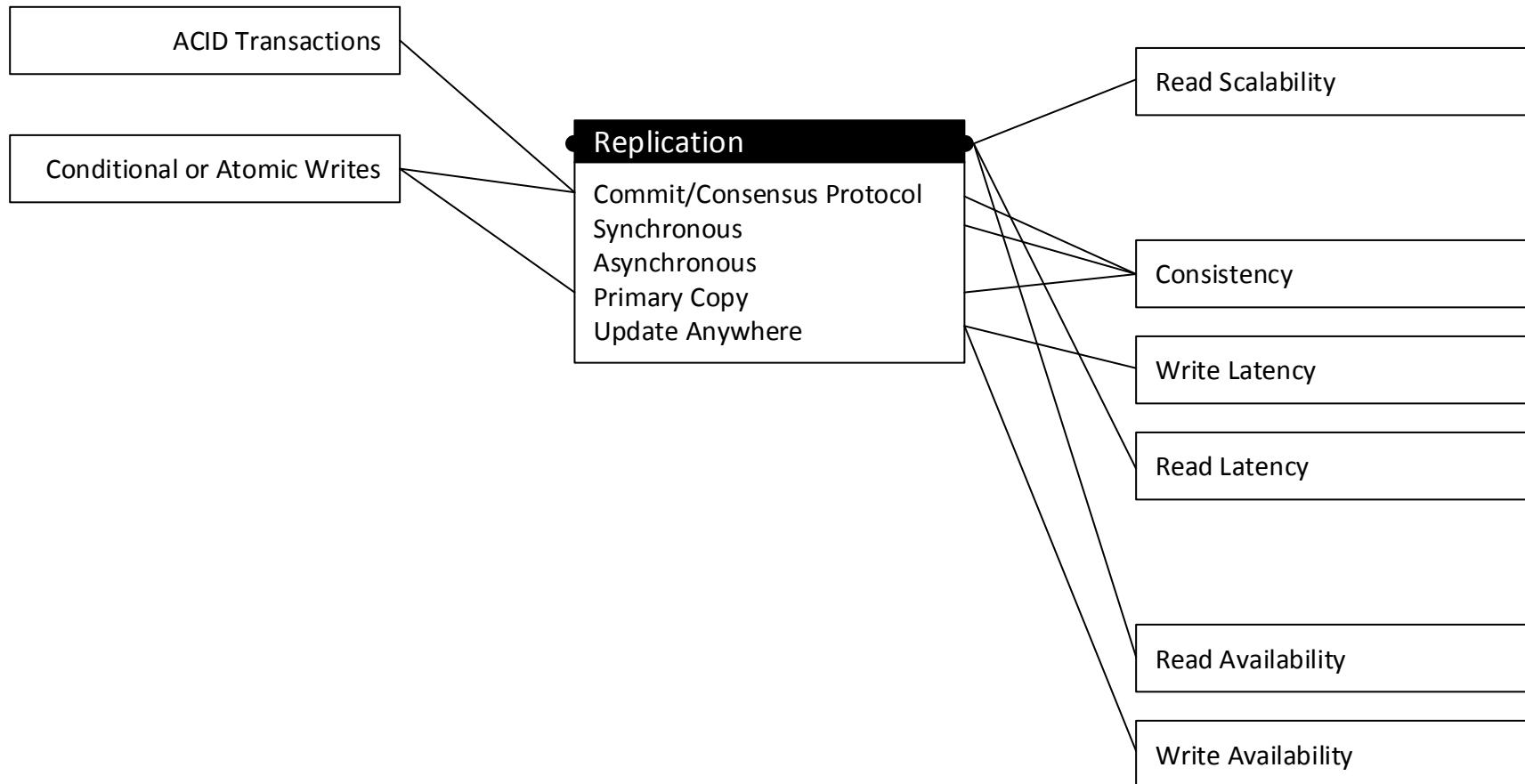
Moved towards:

- ▶ Redis
- ▶ HBase

Functional

Techniques

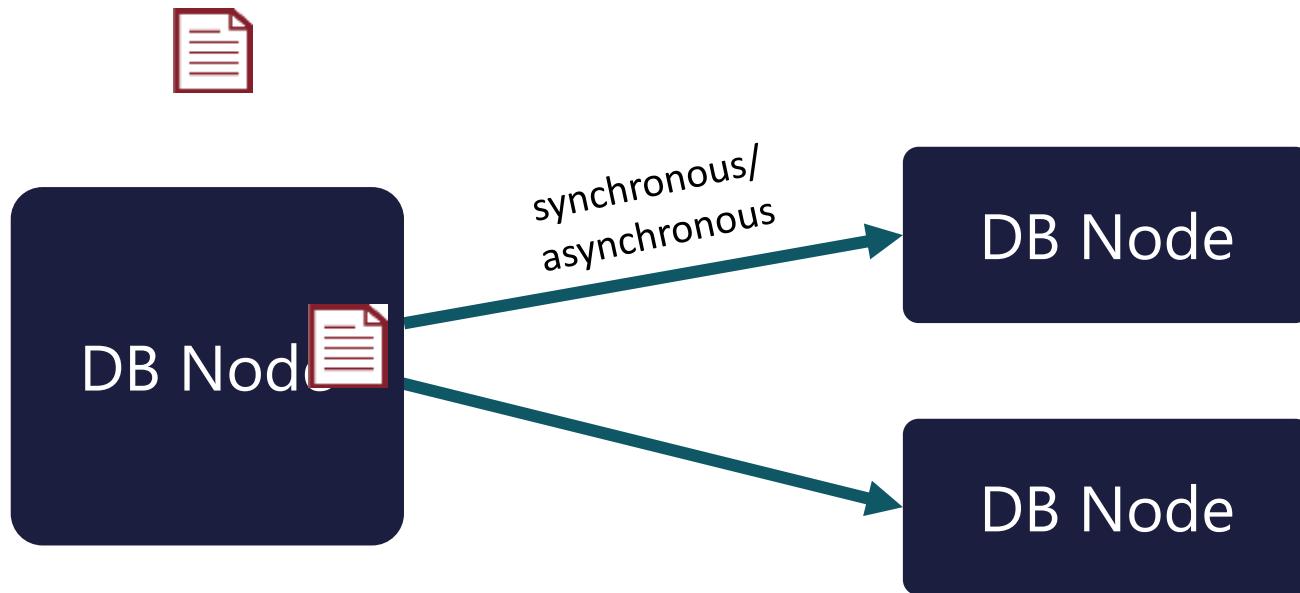
Non-Functional



Replication

Read Scalability + Failure Tolerance

- ▶ Stores N copies of each data item



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



Replication: When

Asynchronous (lazy)

- Writes are acknowledged immediately
- Performed through *log shipping*
- **Pro:** Fast writes, no coordination required
- **Contra:** Replica data potentially stale

Implemented in

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

Synchronous (eager)

- The node accepting writes synchronizes updates/transactions before acknowledging them
- **Pro:** Consistent
- **Contra:** needs a commit protocol, unavailable under certain network partitions

Implemented in

BigTable, HBase, Accumulo, CouchBase, MongoDB, RethinkDB



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

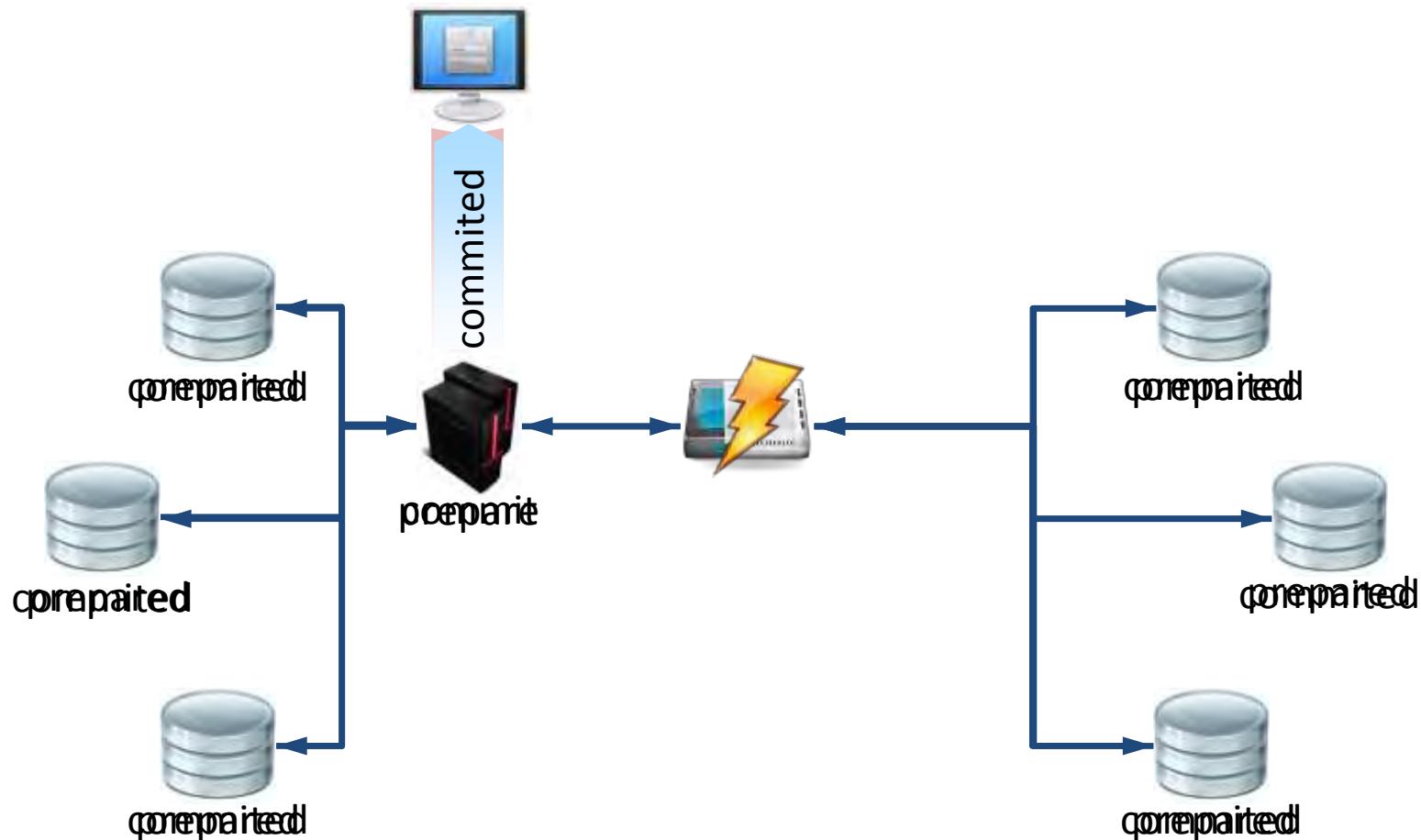
Multi-Master (*Update anywhere*)

- 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

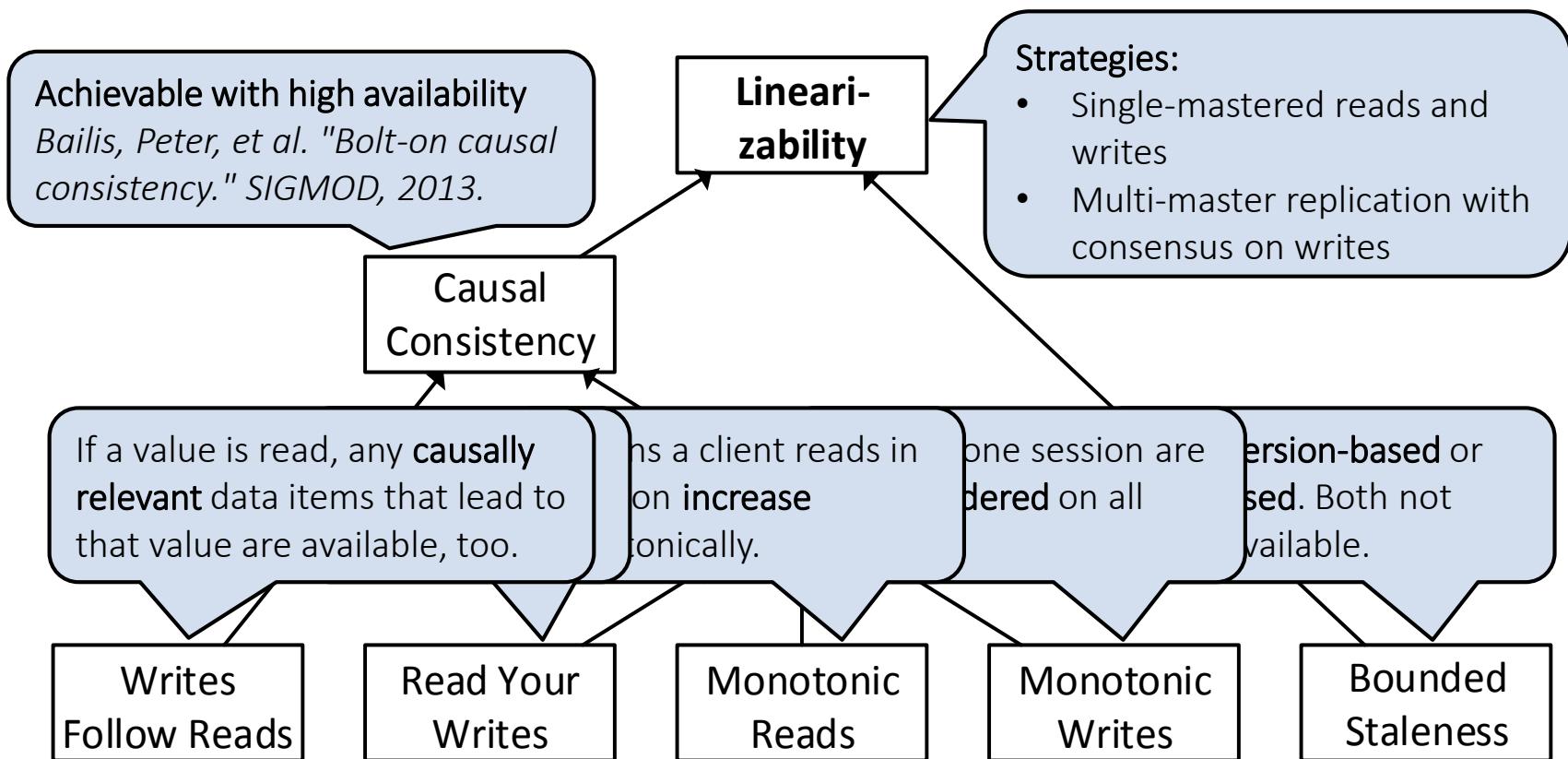


Synchronous Replication

Example: Two-Phase Commit is not partition-tolerant



Consistency Levels

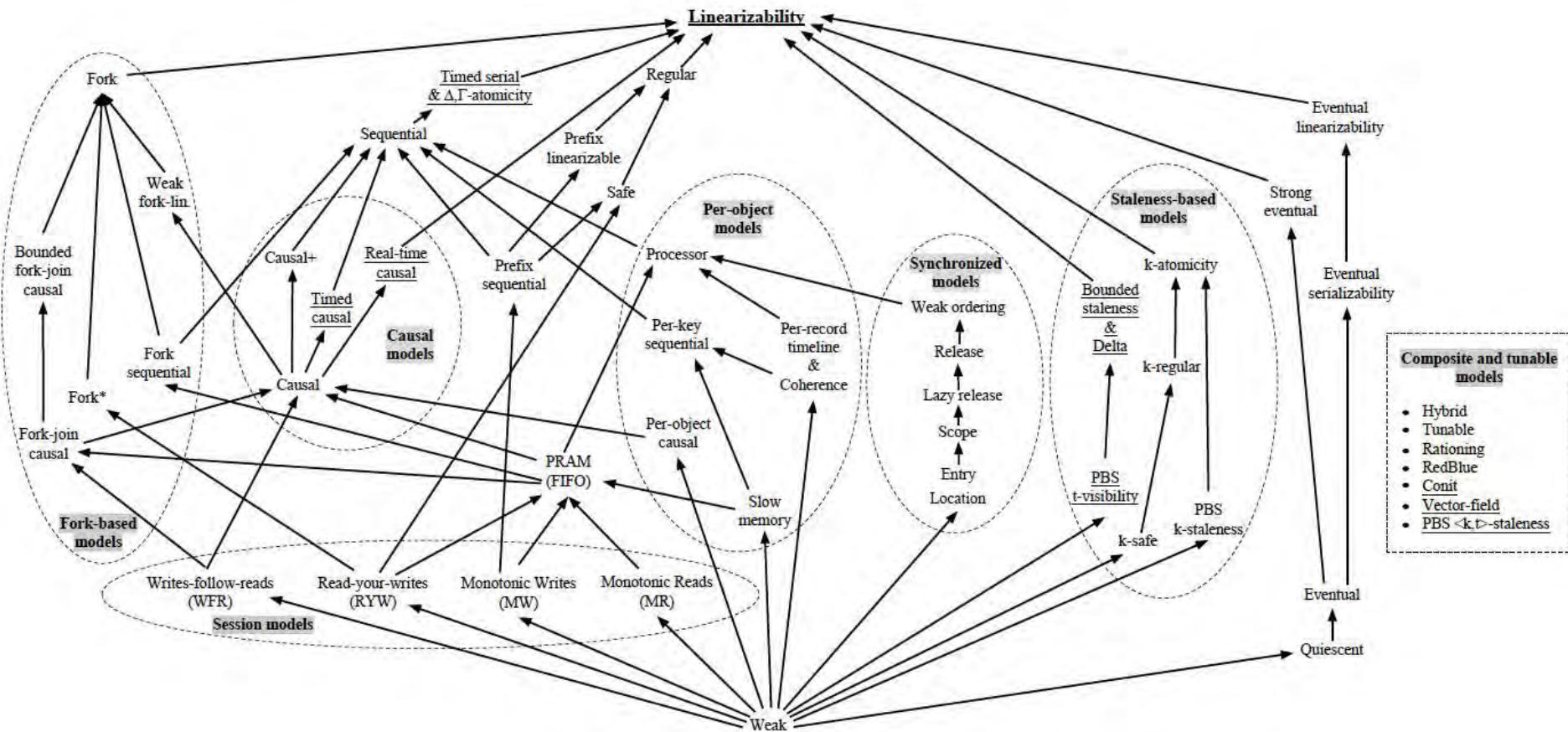


Viotti, Paolo, and Marko Vukolić. "Consistency in Non-Transactional Distributed Storage Systems." arXiv (2015).



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

Problem: Terminology



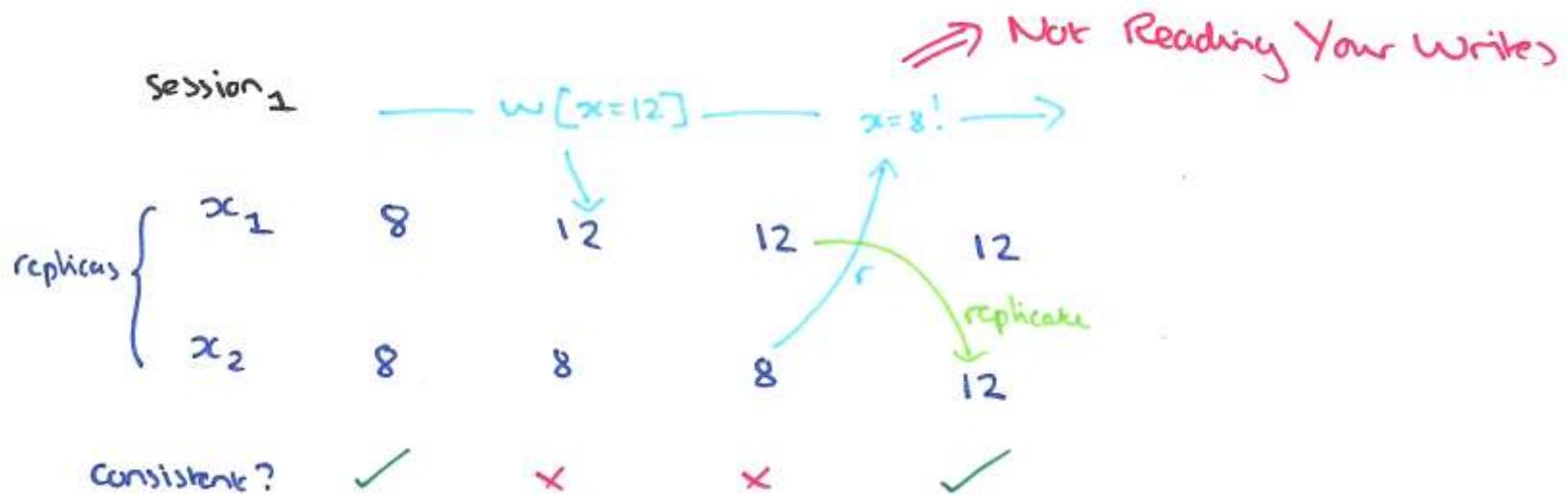
V., Paolo, and M. Vukolić. "Consistency in Non-Transactional Distributed Storage Systems." ACM CSUR (2016).



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

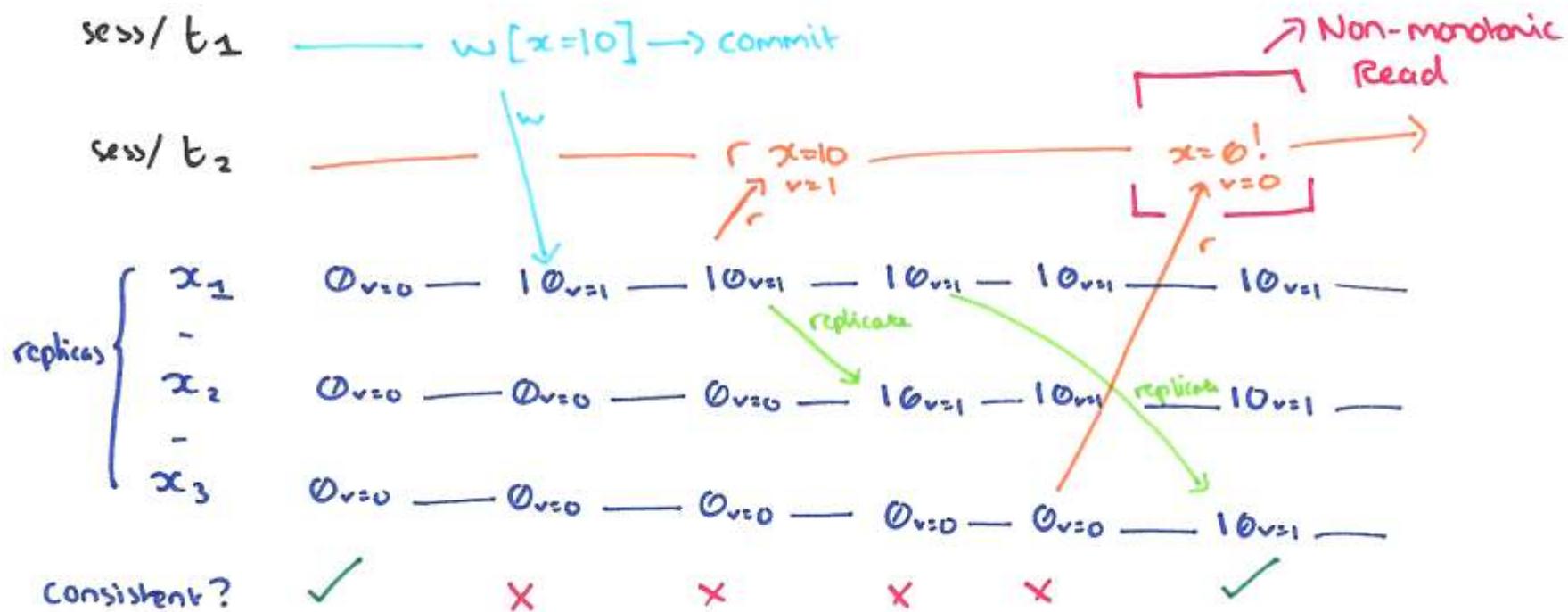
Read Your Writes (RYW)

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.



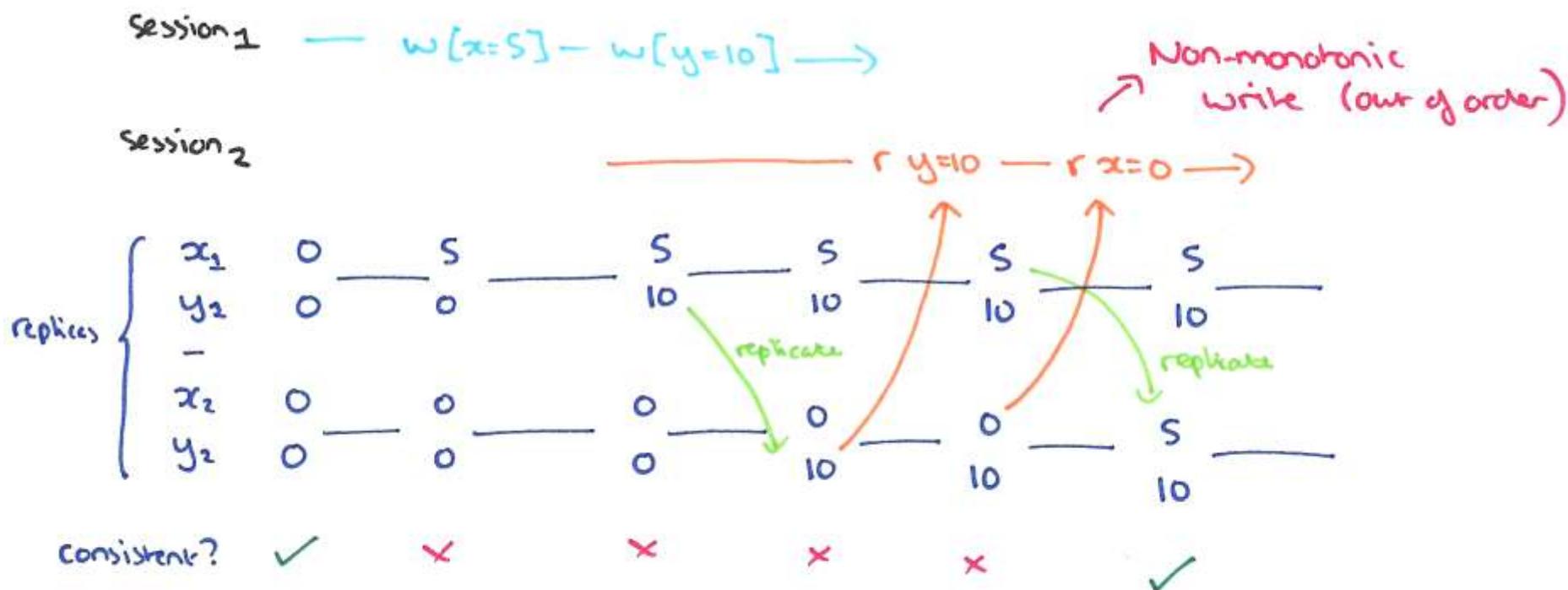
Monotonic Reads (MR)

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



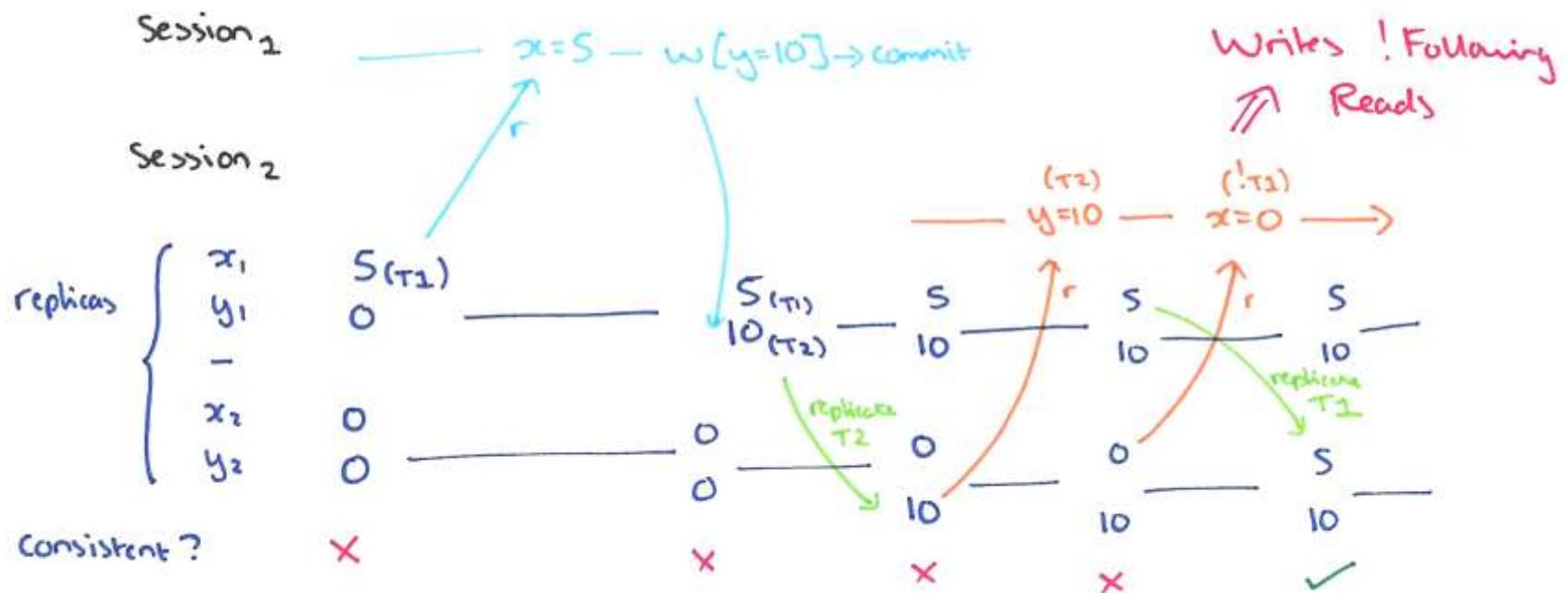
Monotonic Writes (MW)

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.



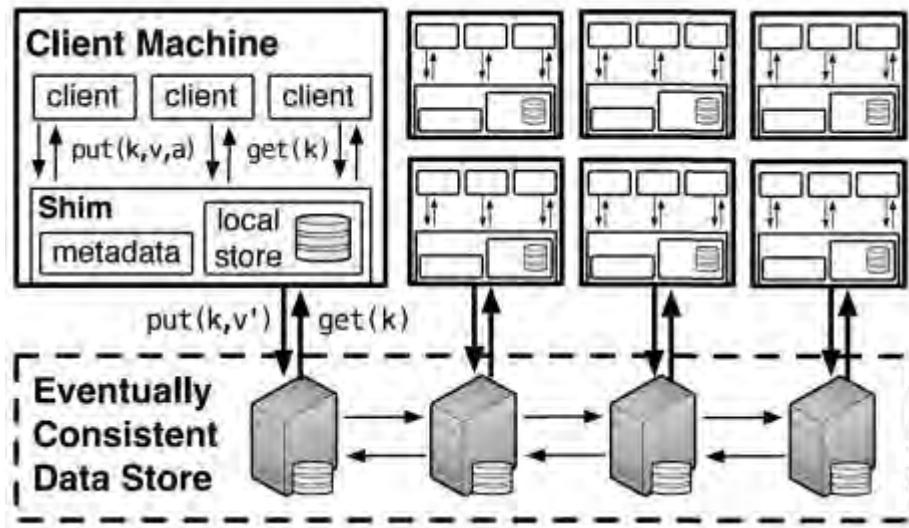
Writes Follow Reads (WFR)

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.



PRAM and Causal Consistency

- ▶ Combinations of previous session consistency guarantees
 - 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.

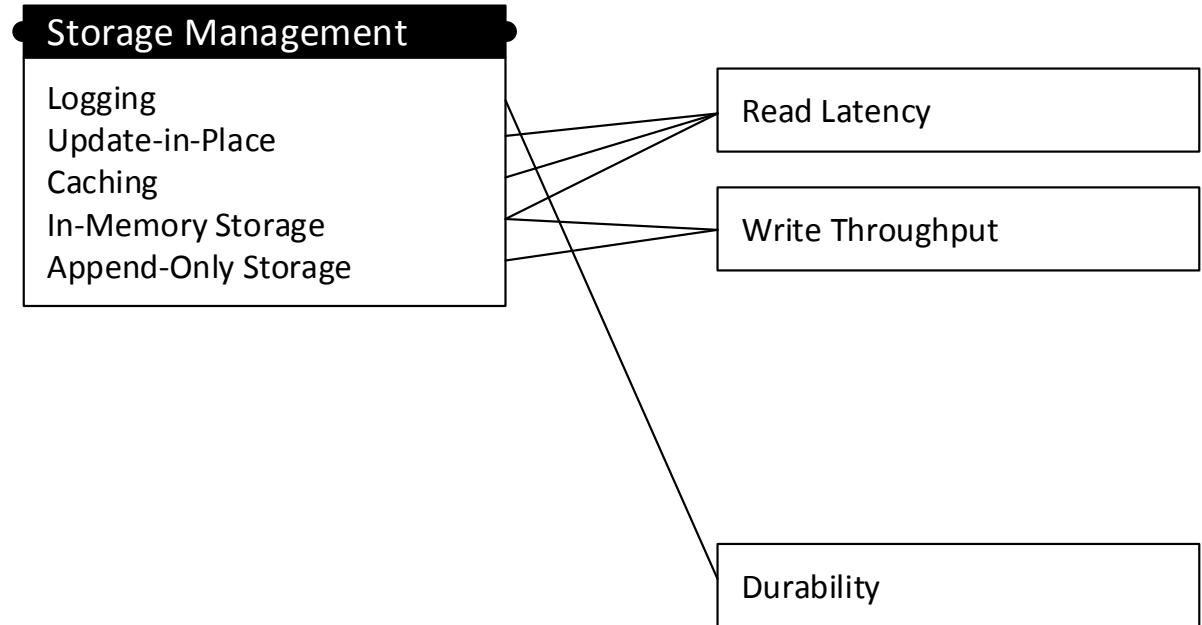
- ▶ Both are *not* achievable with high availability



Functional

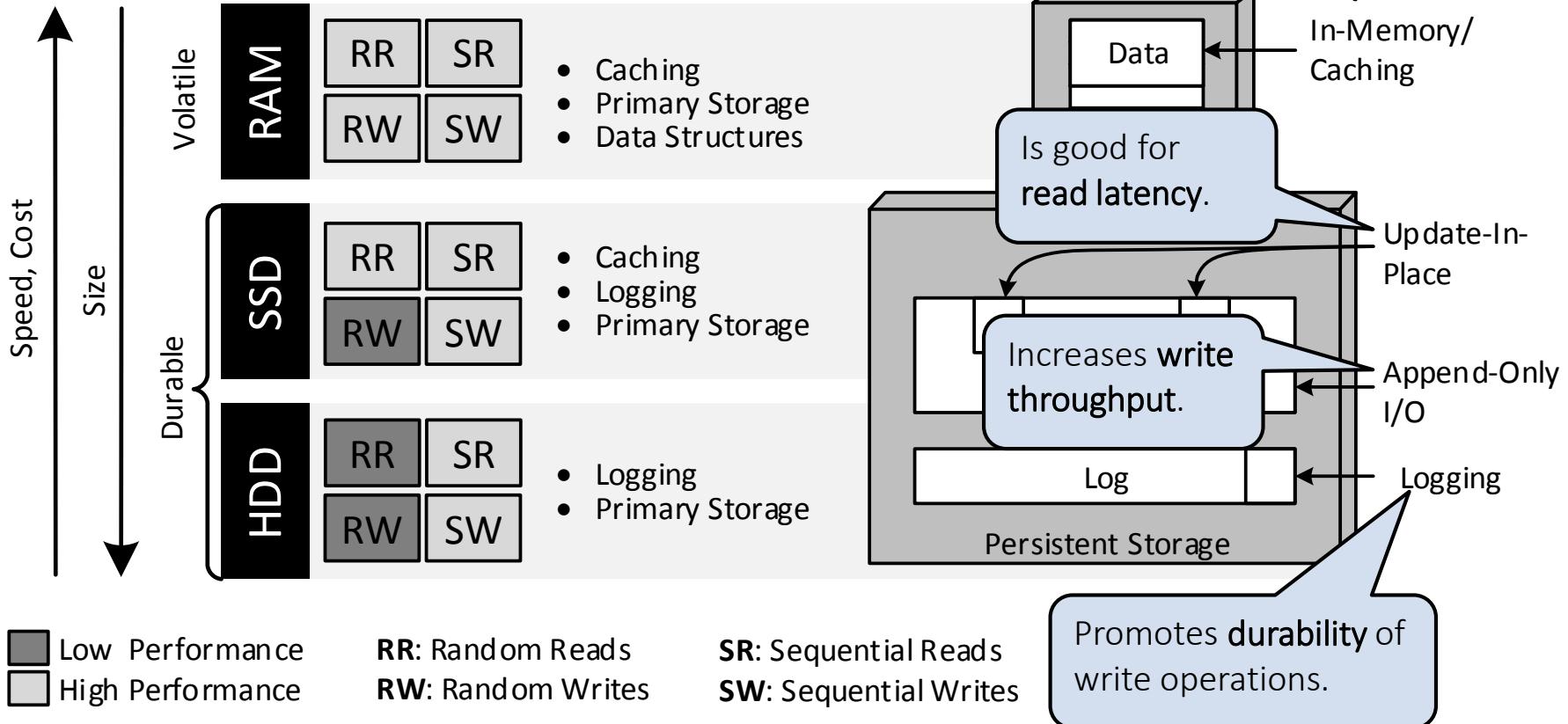
Techniques

Non-Functional



NoSQL Storage Management

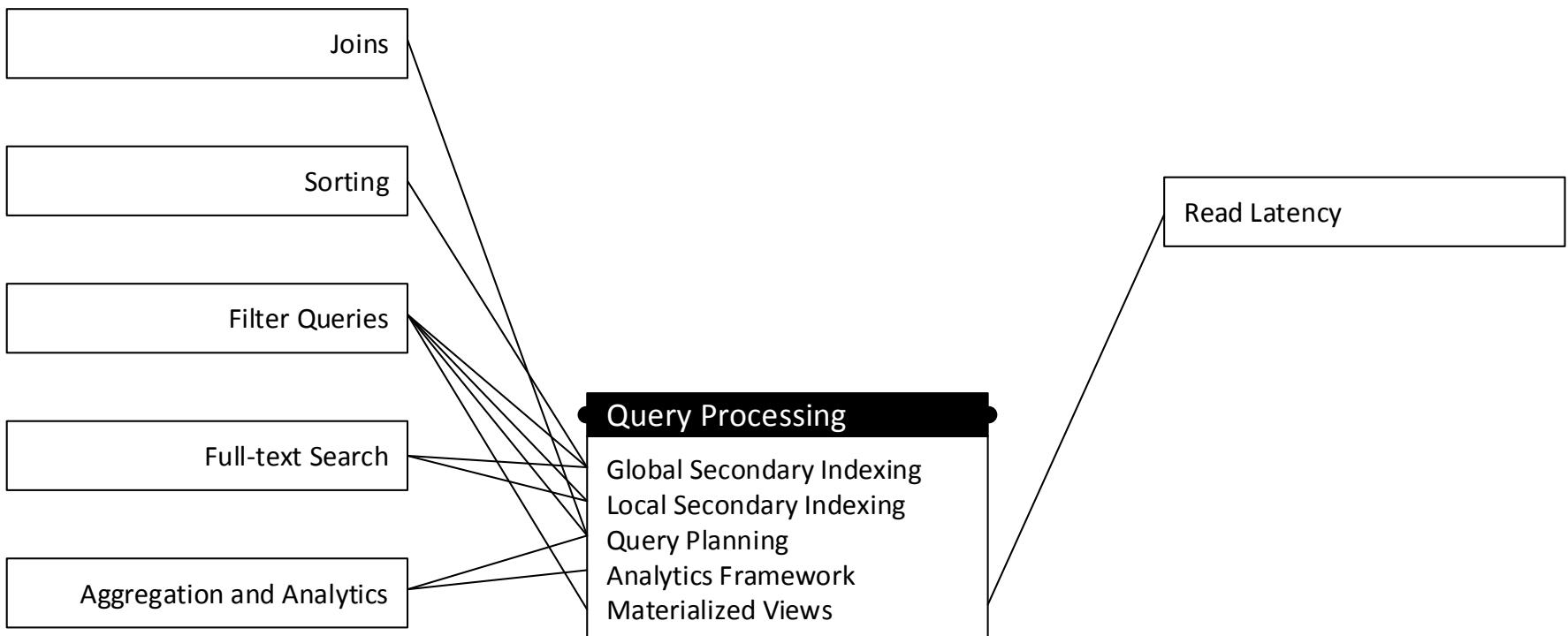
In a Nutshell



Functional

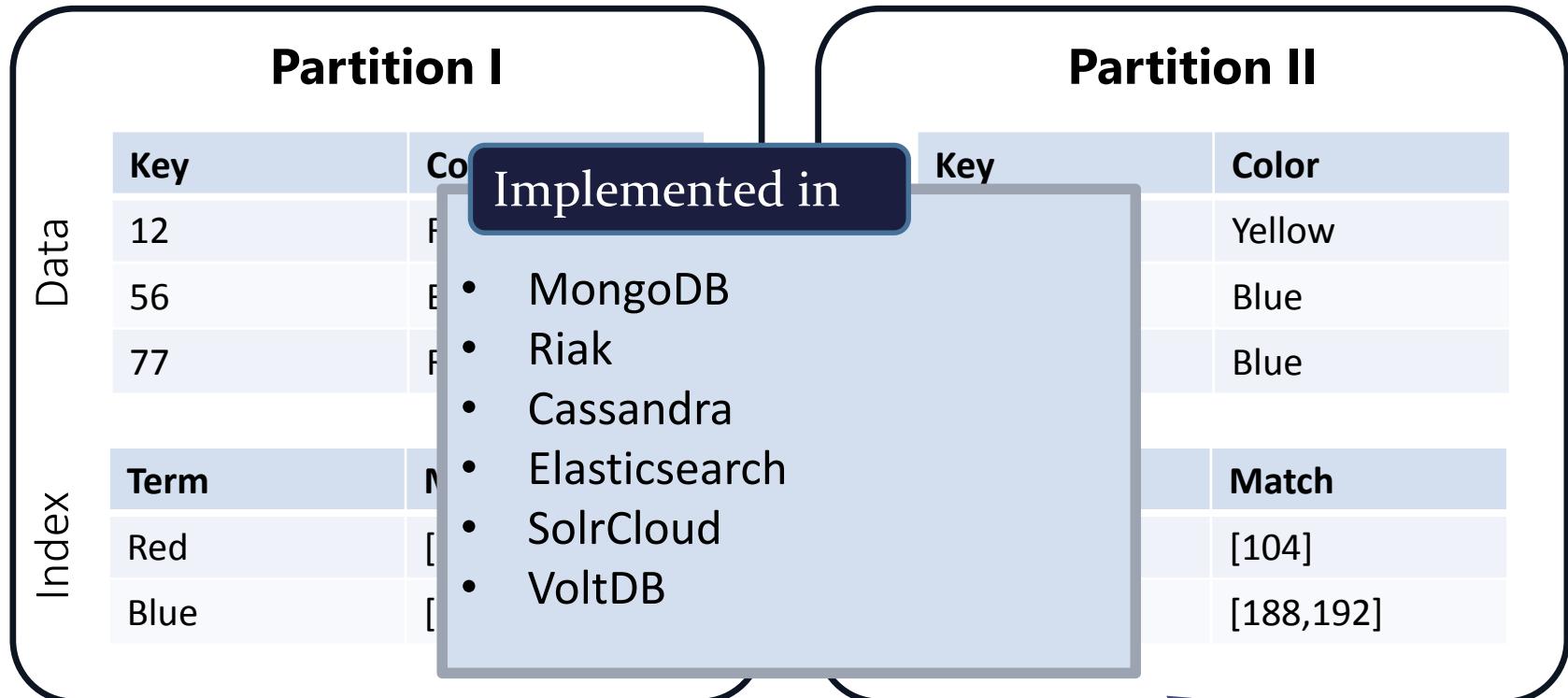
Techniques

Non-Functional



Local Secondary Indexing

Partitioning By Document



Scatter-gather query pattern.

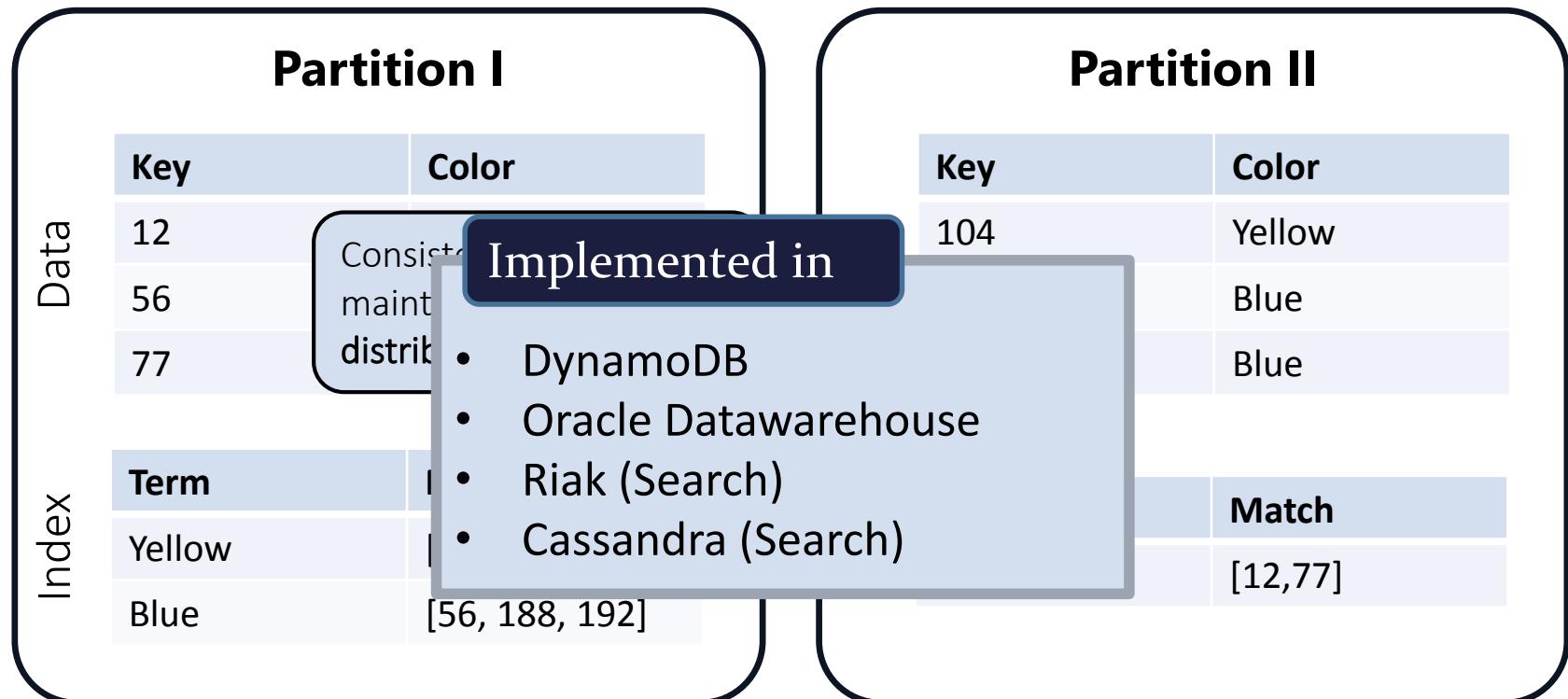
WHERE color=blue



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

Global Secondary Indexing

Partitioning By Term



Targeted Query

WHERE color=blue



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

Query Processing Techniques

Summary

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



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

Outline



NoSQL Foundations and Motivation



The NoSQL Toolbox:
Common Techniques



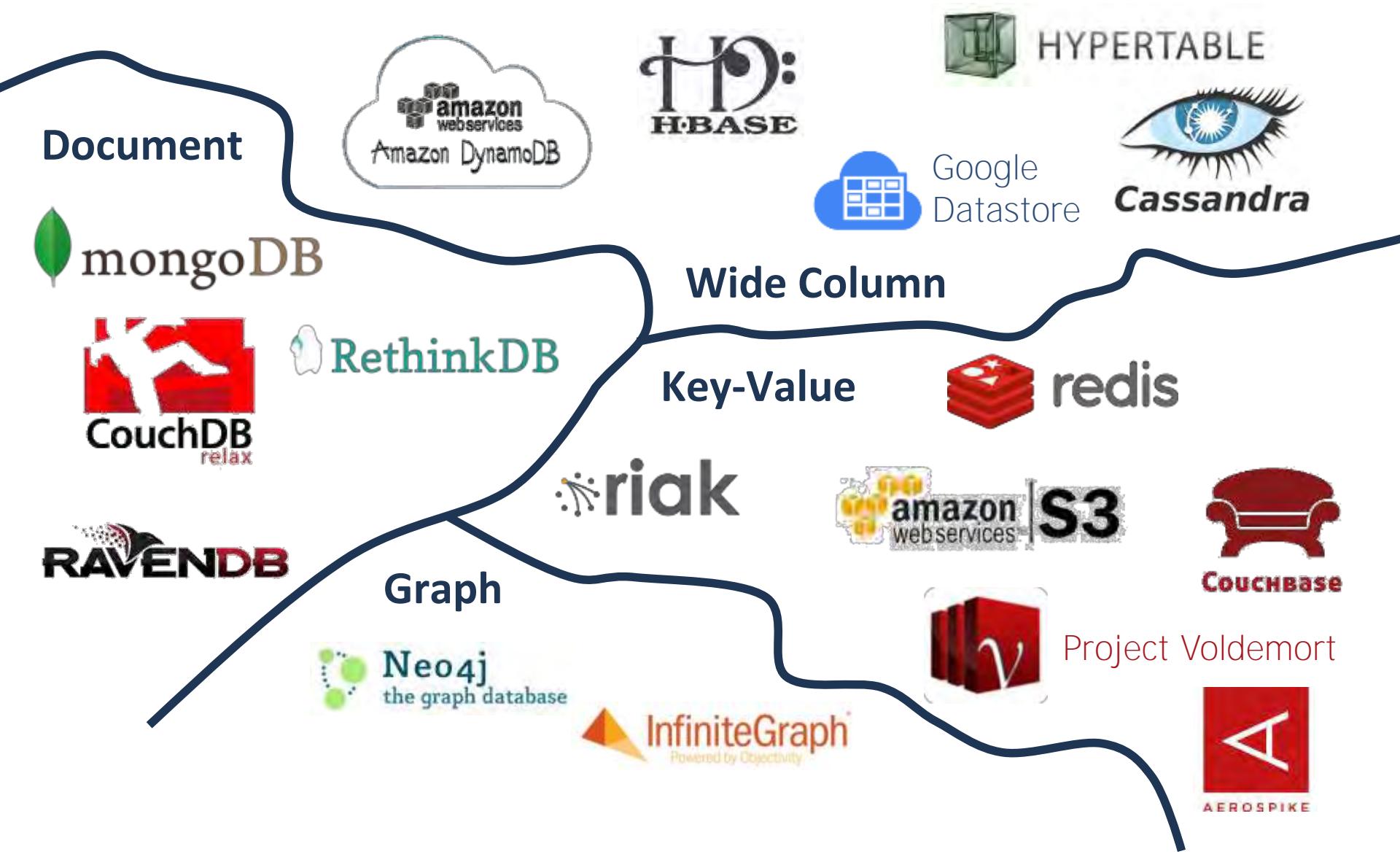
NoSQL Systems &
Decision Guidance



Scalable Real-Time
Databases and Processing

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

NoSQL Landscape



Popularity

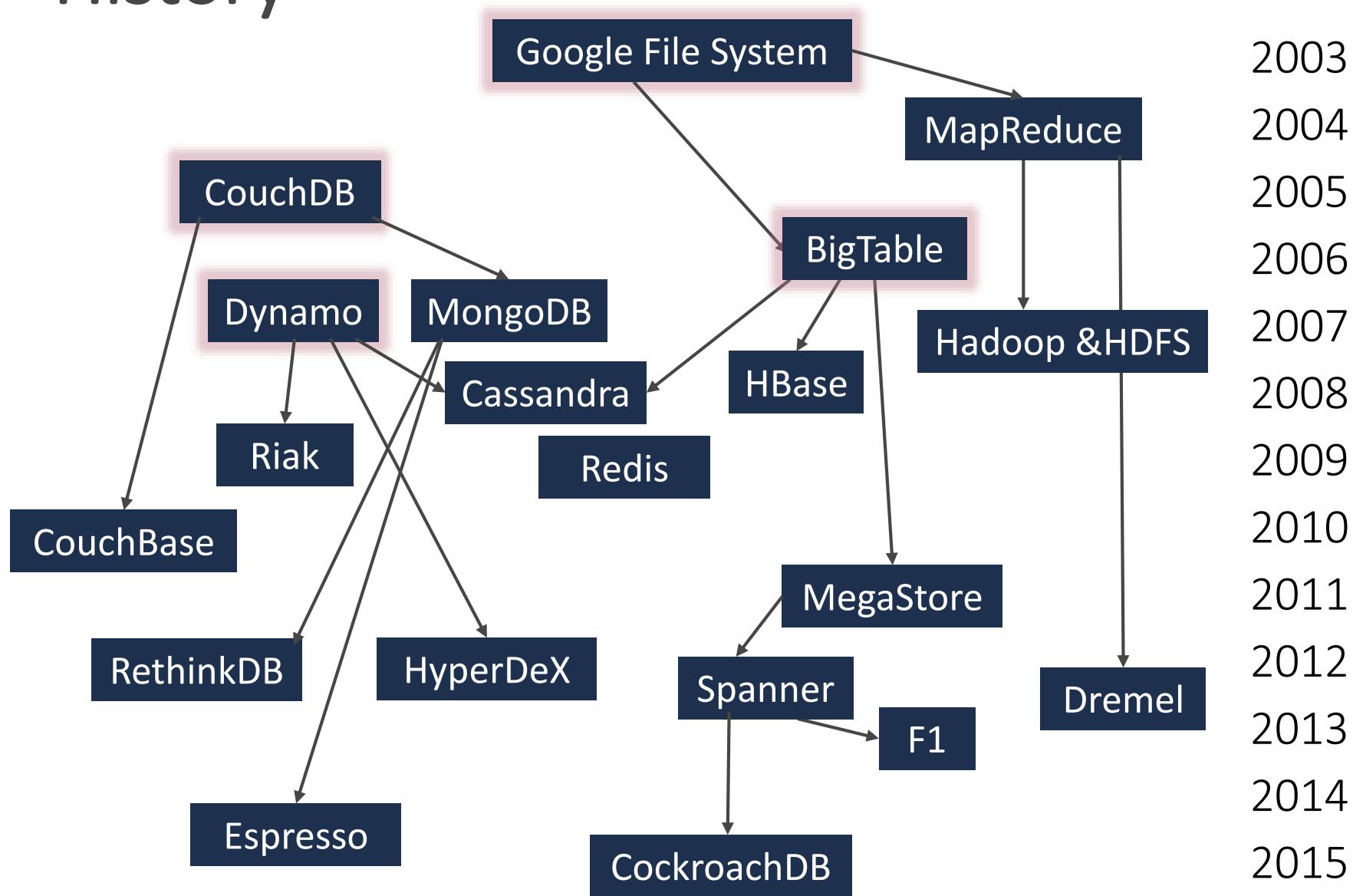
<http://db-engines.com/de/ranking>

#	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



NoSQL foundations

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



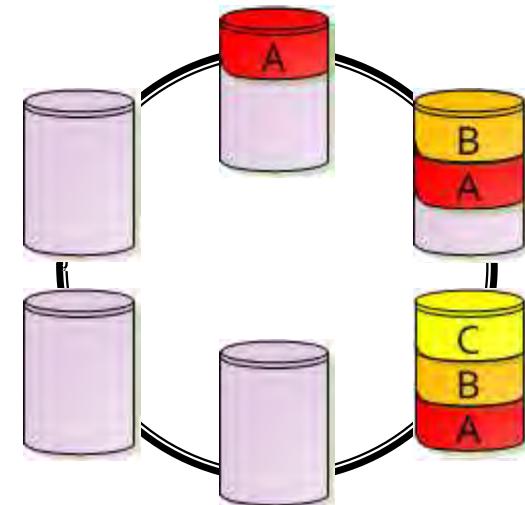
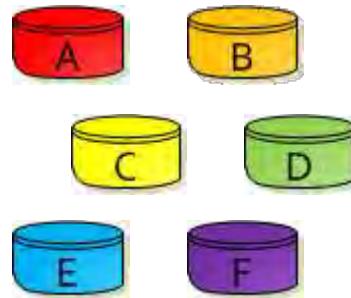
Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."



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

Dynamo (AP)

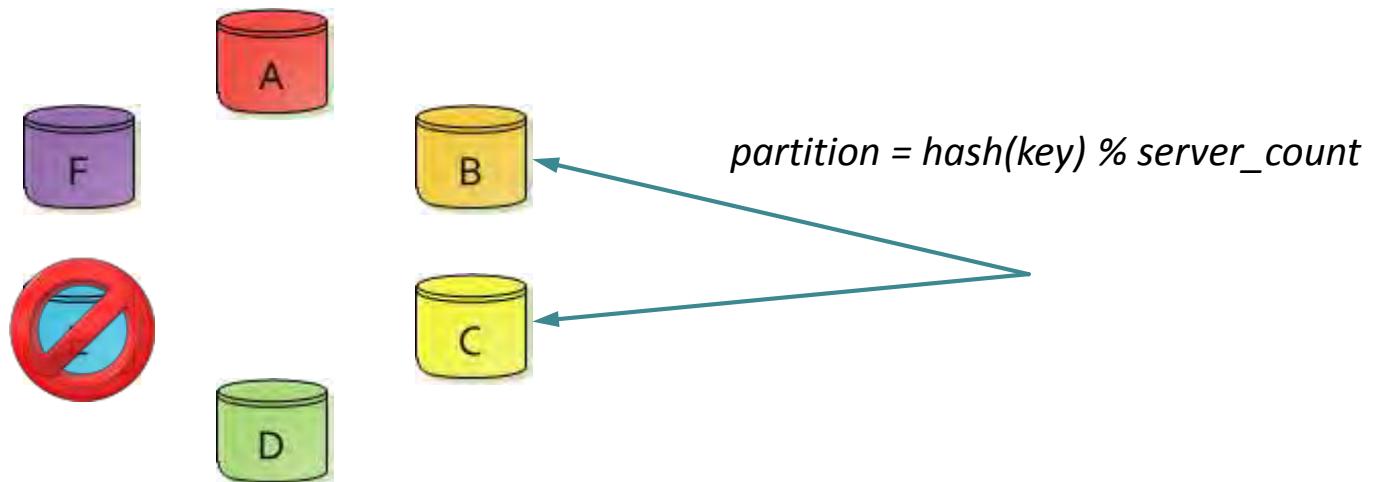
- ▶ Developed at Amazon (2007)
- ▶ Sharding of data over a ring of nodes
- ▶ Each node holds multiple partitions
- ▶ Each partition replicated **N** times



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

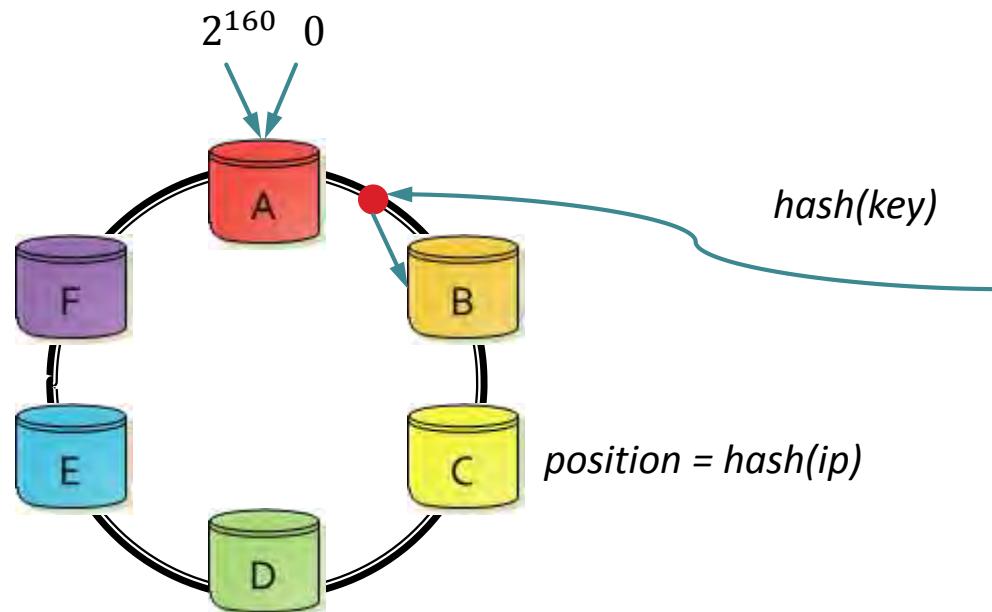
Consistent Hashing

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



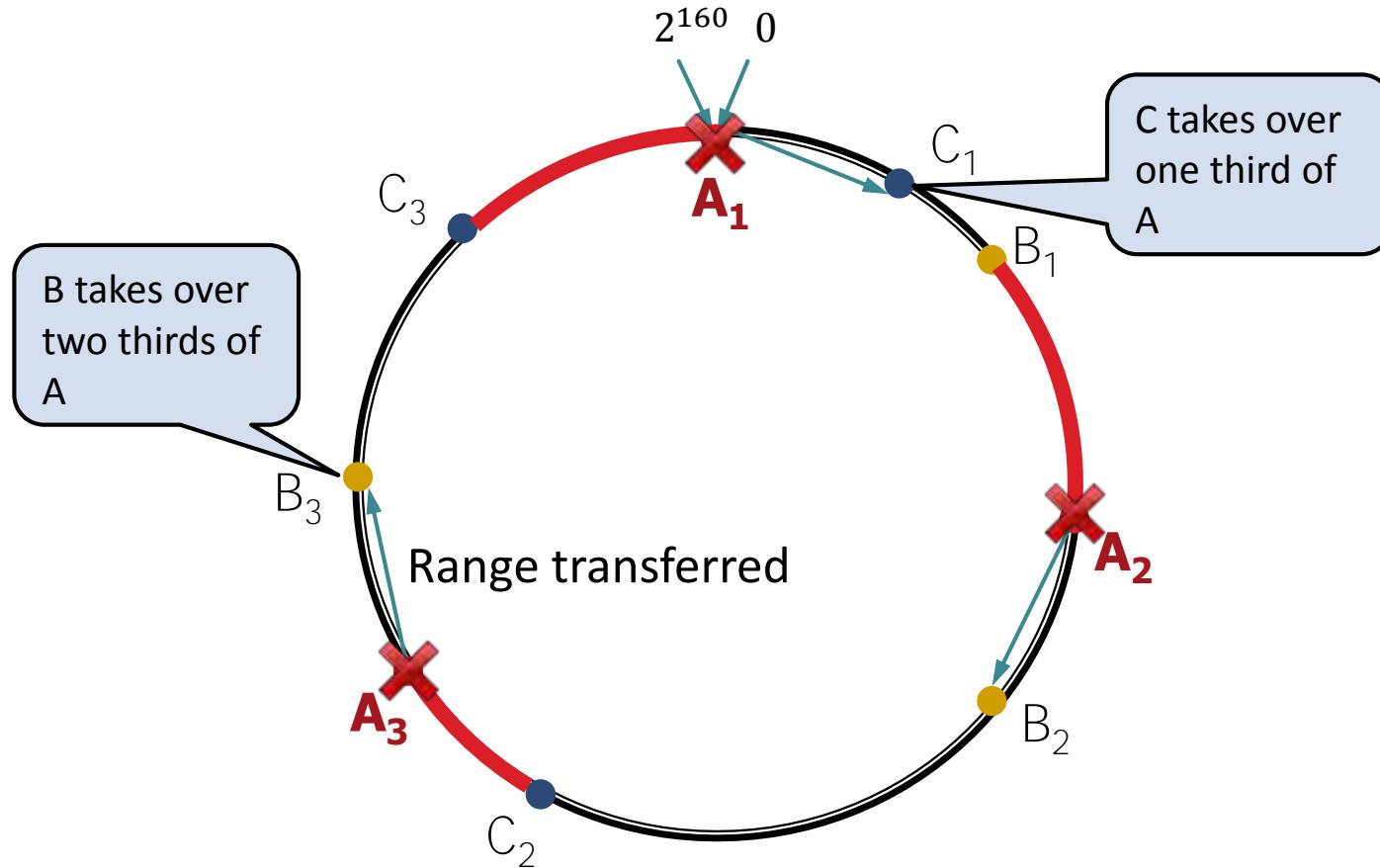
Consistent Hashing

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



Consistent Hashing

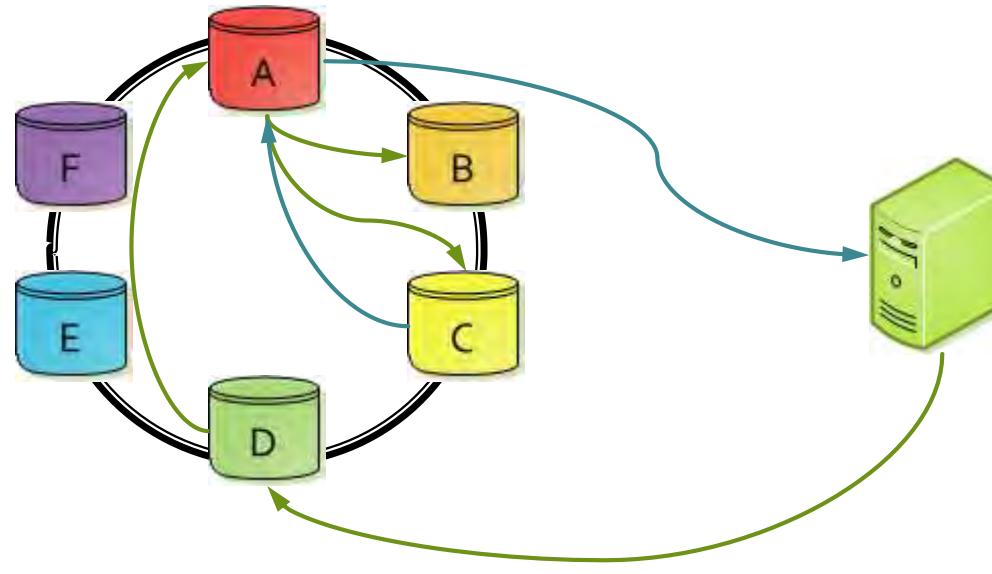
- Extension: Virtual Nodes for Load Balancing



Reading

Parameters R, W, N

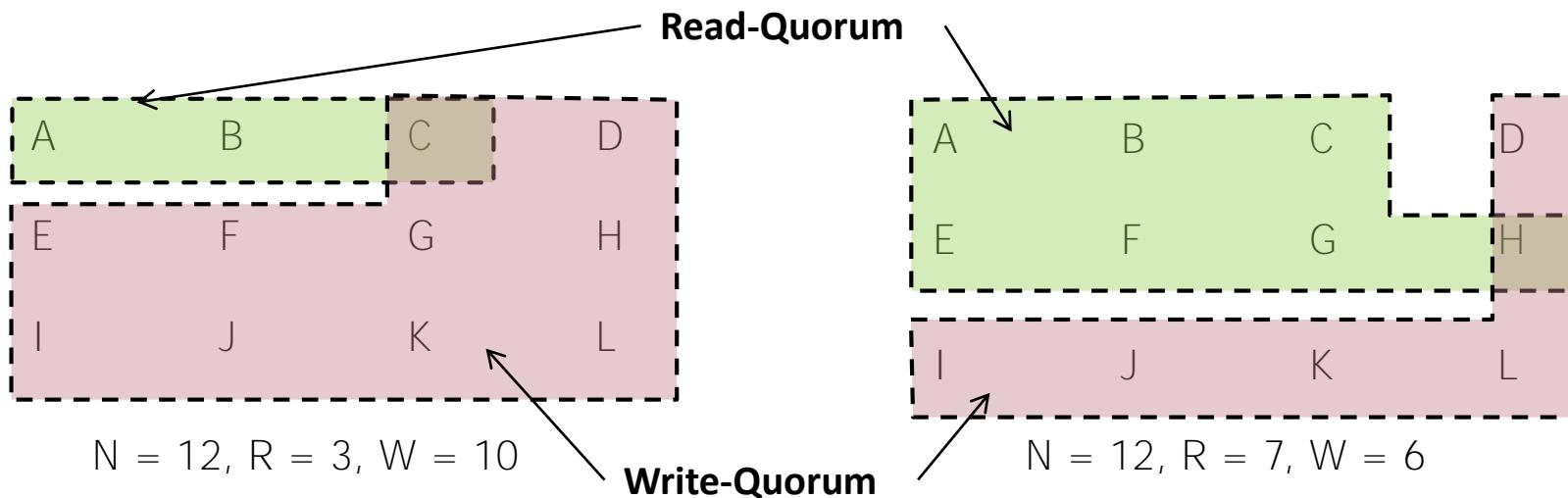
- ▶ An arbitrary node acts as a coordinator
- ▶ N: number of replicas
- ▶ R: number of nodes that need to confirm a read
- ▶ W: number of nodes that need to confirm a write



N=3
R=2
W=1

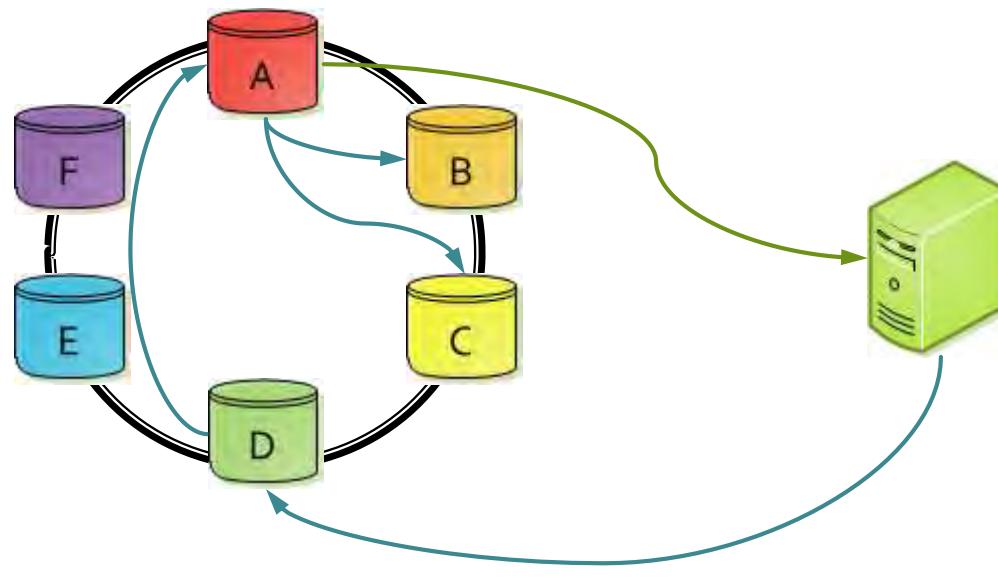
Quorums

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



Writing

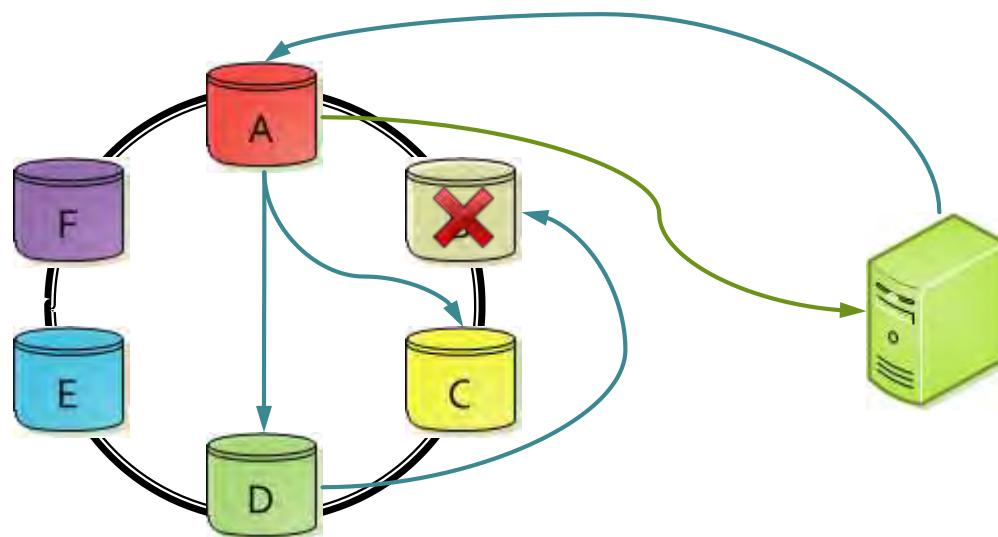
- ▶ W Servers have to acknowledge



N=3
R=2
W=1

Hinted Handoff

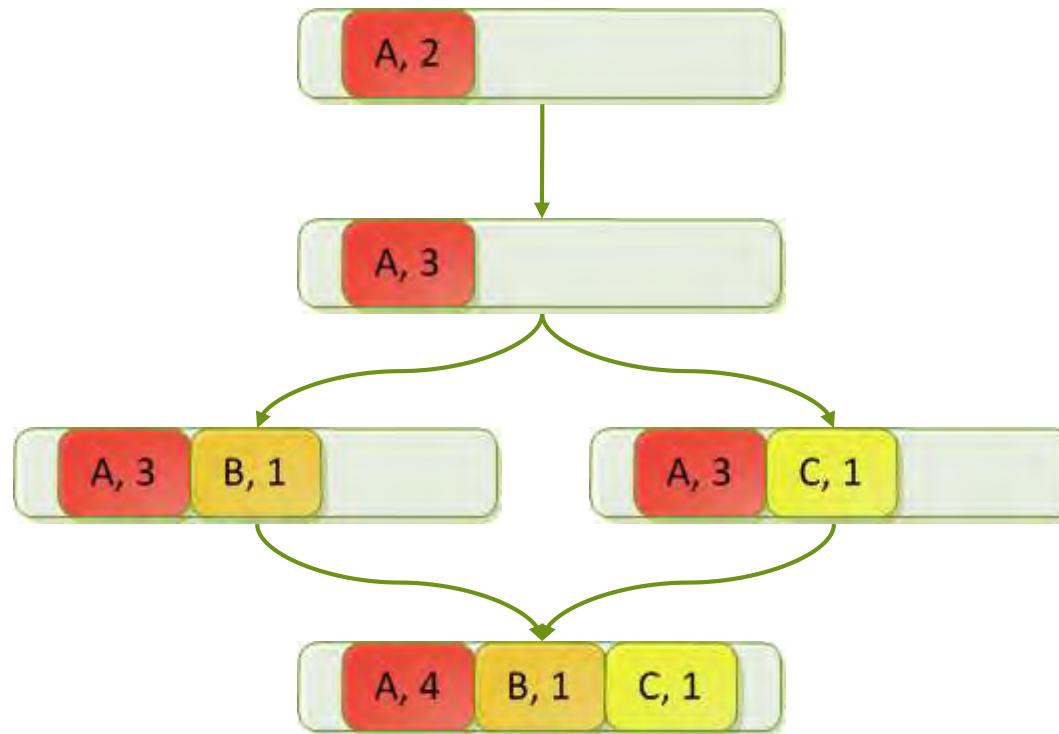
- ▶ Next node in the ring may take over, until original node is available again:



N=3
R=2
W=1

Vector clocks

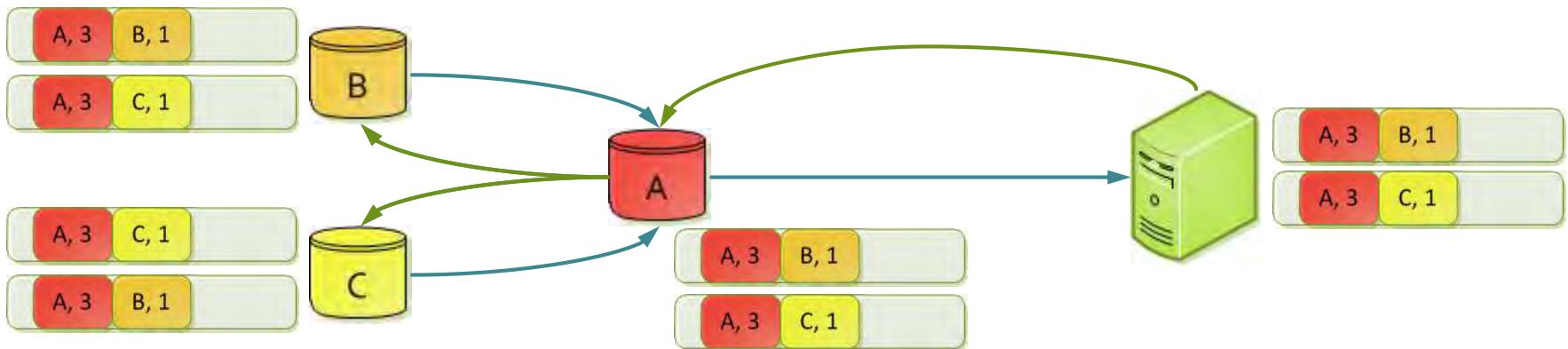
- ▶ Dynamo uses **Vector Clocks** for versioning



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

Versioning and Consistency

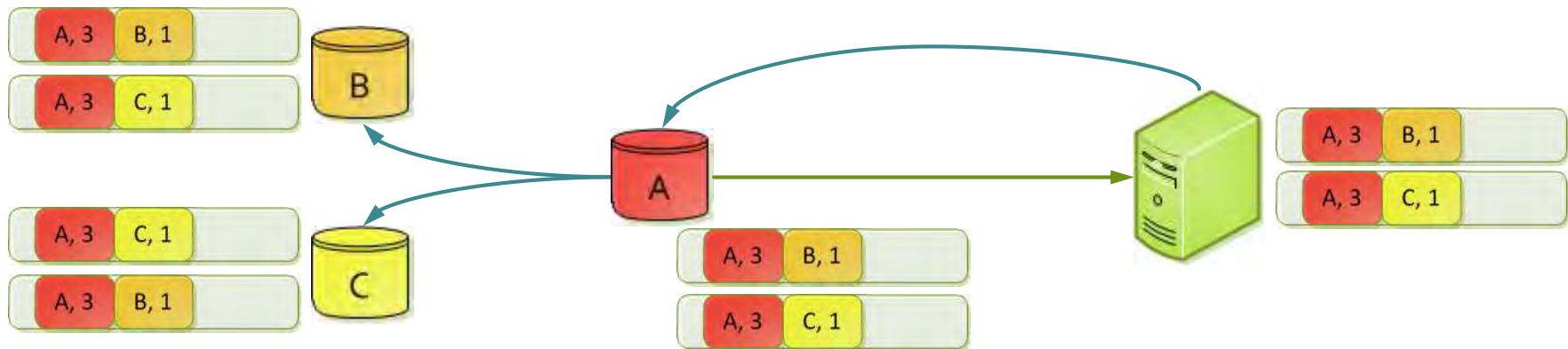
- ▶ $R + W \leq N \Rightarrow$ no consistency guarantee
- ▶ $R + W > N \Rightarrow$ newest acked value included in reads
- ▶ **Vector Clocks** used for versioning



Read Repair

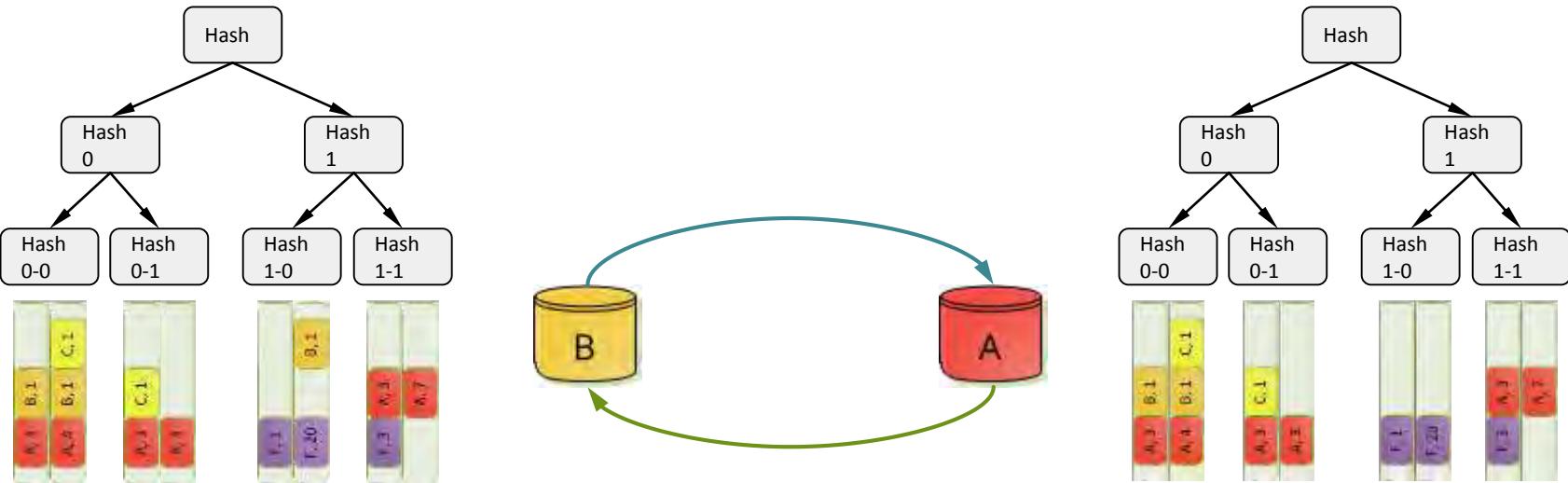
Conflict Resolution

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



Merkle Trees: Anti-Entropy

- ▶ Every Second: Contact random server and compare



Quorum

▶ Typical Configurations:

Performance
(Cassandra Default)

N=3, R=1, W=1

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

Quorum, fast
Writing:

N=3, R=3, W=1

Quorum, fast
Reading

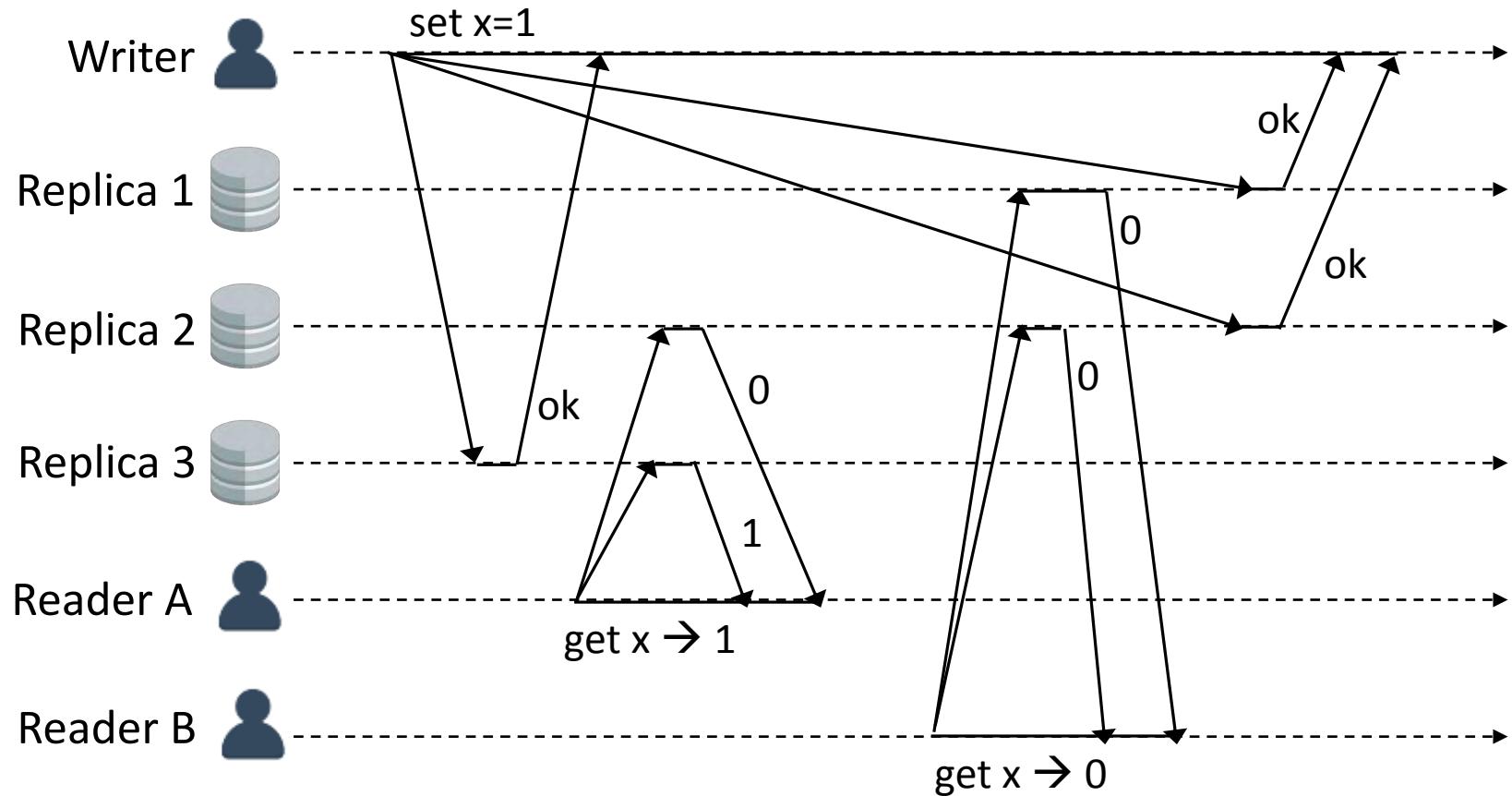
N=3, R=1, W=3

Trade-off (Riak
Default)

N=3, R=2, W=2

$R + W > N$ does not imply linearizability

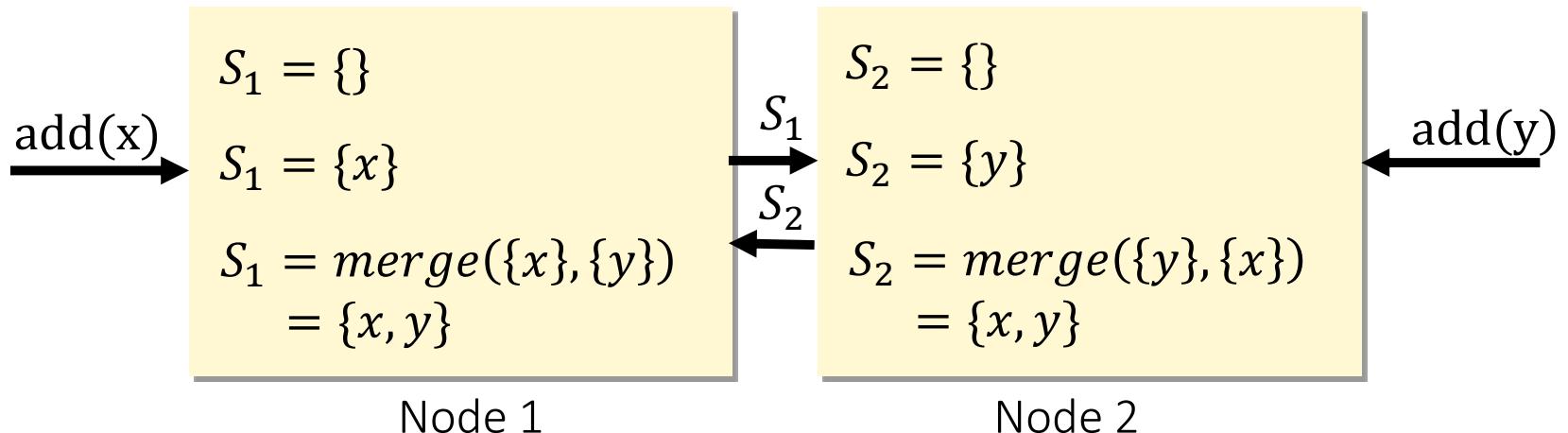
- Consider the following execution:



CRDTs

Convergent/Commutative Replicated Data Types

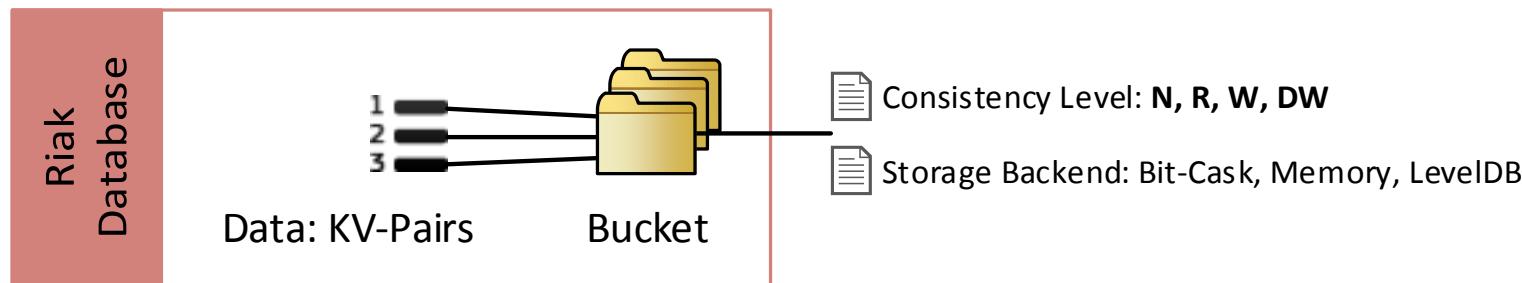
- ▶ Goal: avoid manual conflict-resolution
- ▶ Approach:
 - State-based – commutative, idempotent merge function
 - Operation-based – broadcasts of commutative updates
- ▶ Example: State-based Grow-only-Set (G-Set)



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

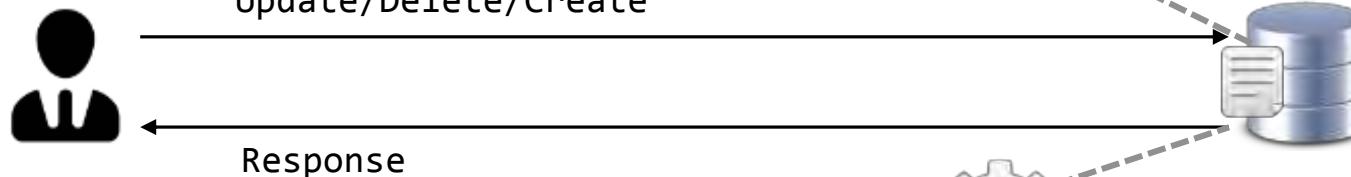
- ▶ Implemented as *state-based CRDTs*:

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



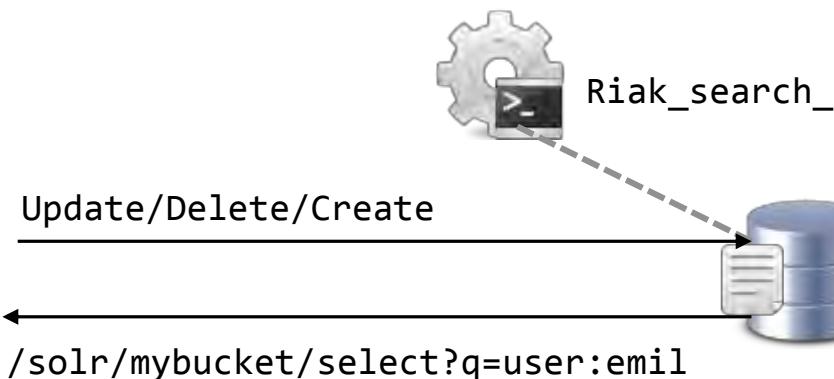
Hooks & Search

▶ Hooks:



JS/Erlang Pre-Commit Hook

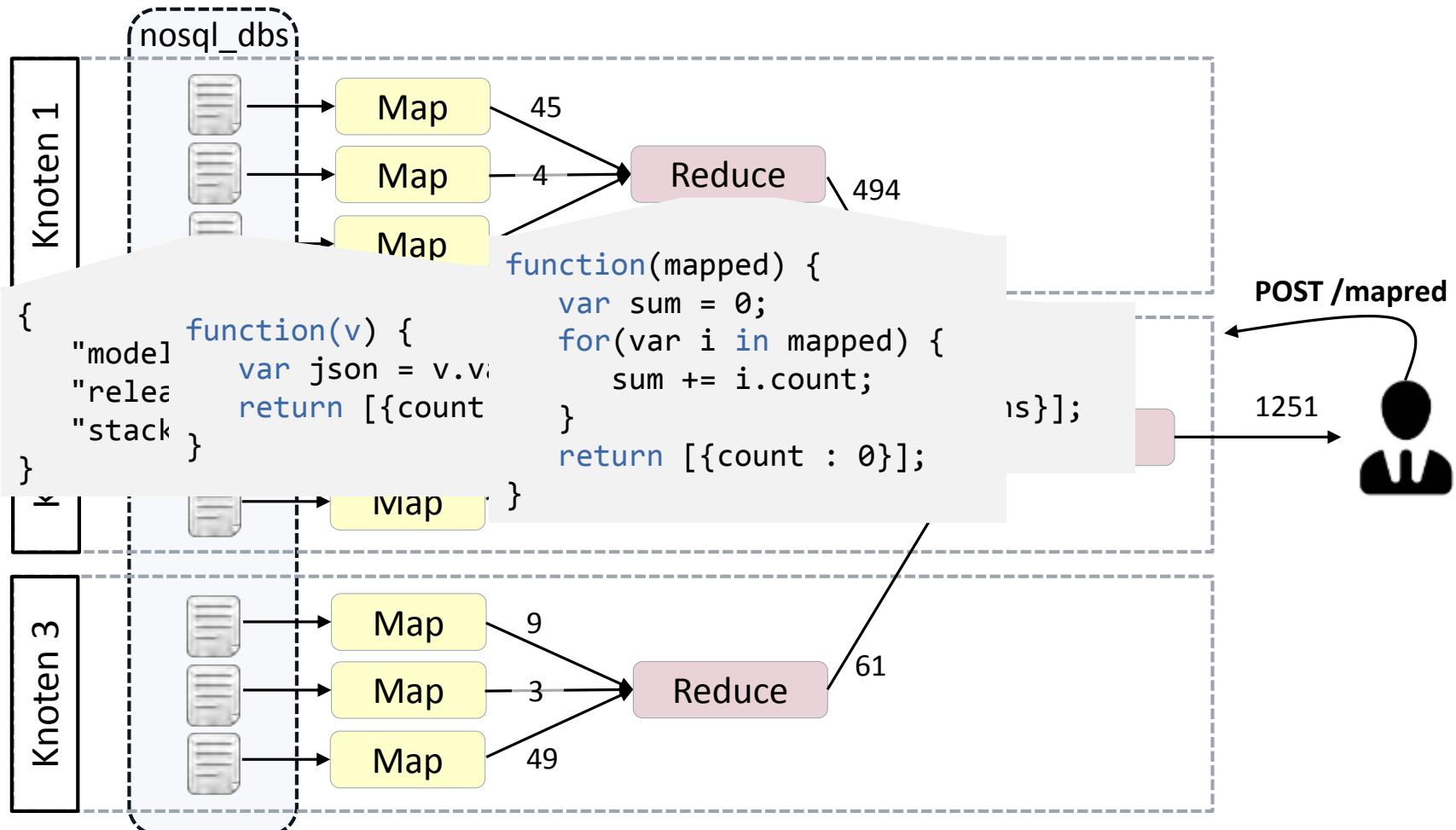
▶ Riak Search:



Term	Dokument
database	3,4,1
rabbit	2

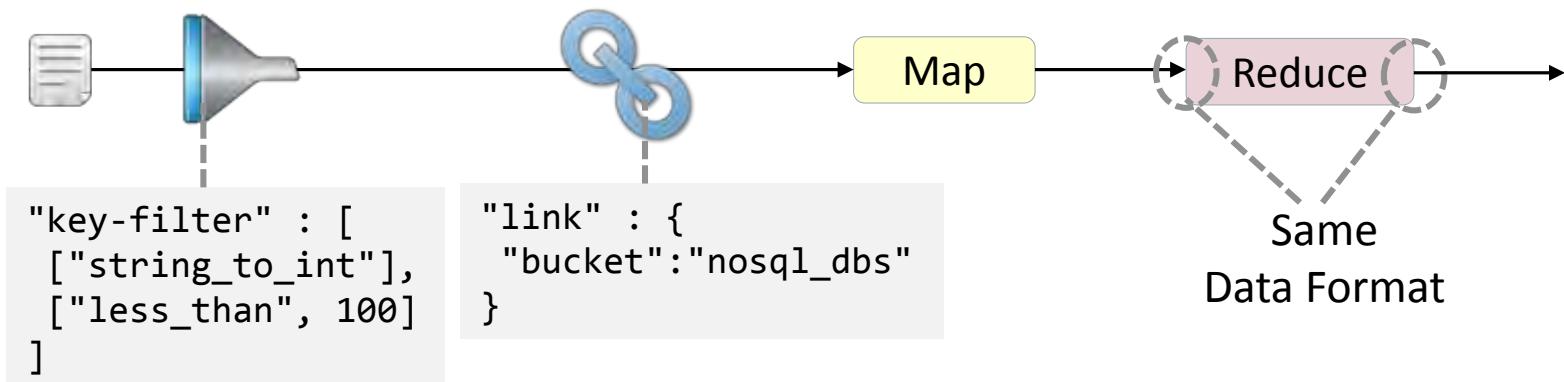
Search Index

Riak Map-Reduce

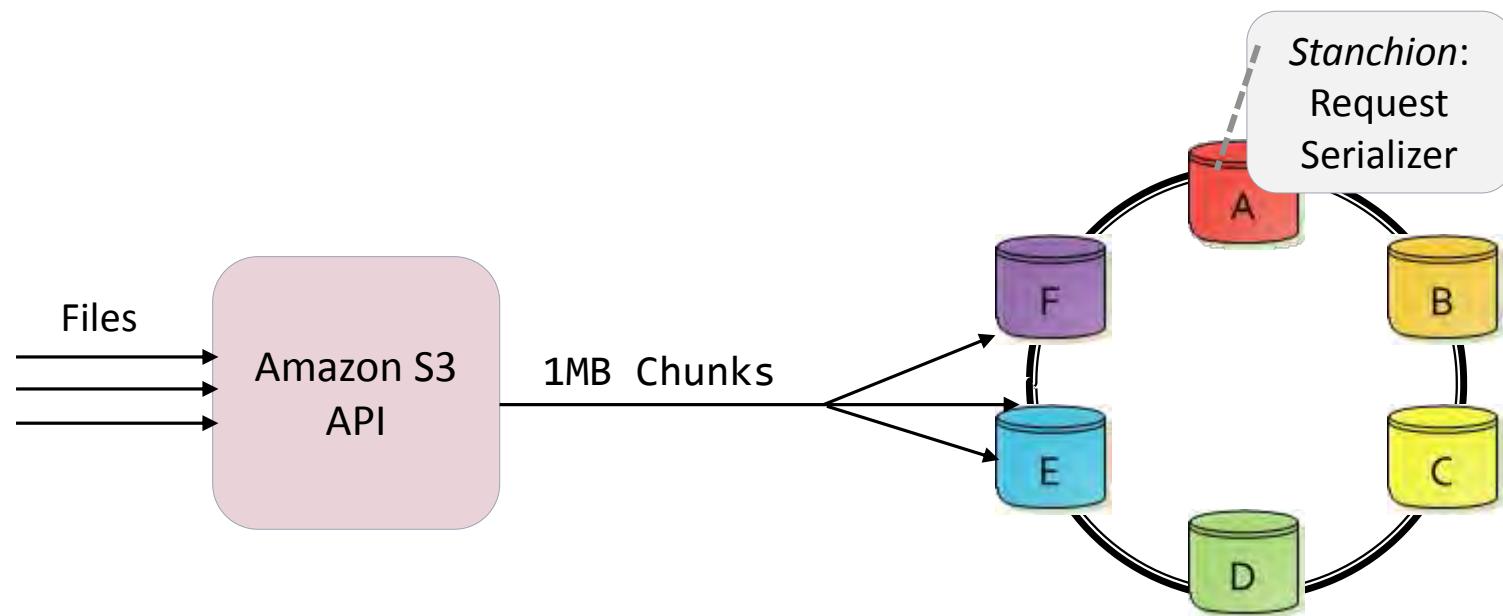


Riak Map-Reduce

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



Riak Cloud Storage



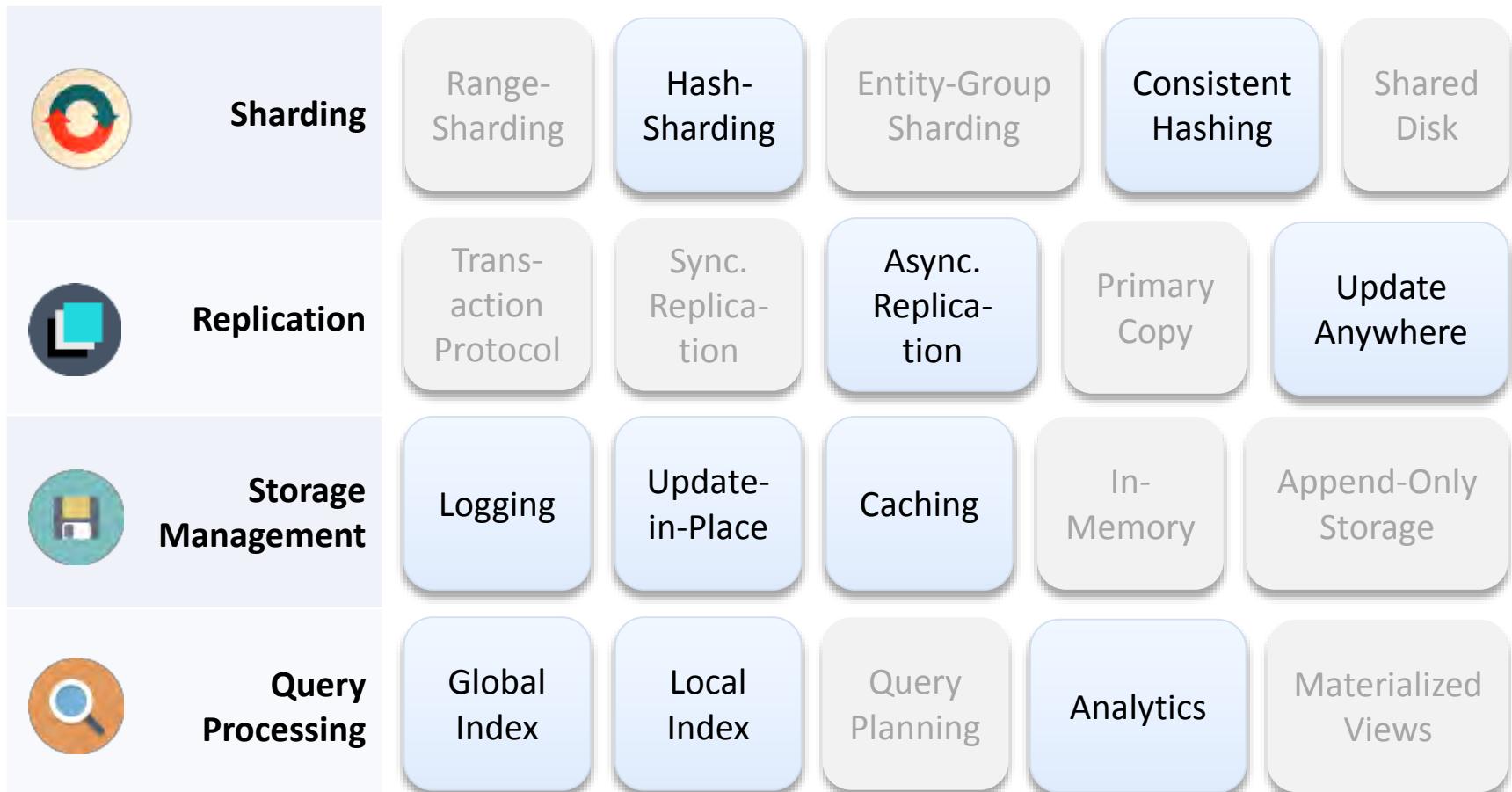
Summary: Dynamo and Riak



- ▶ 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





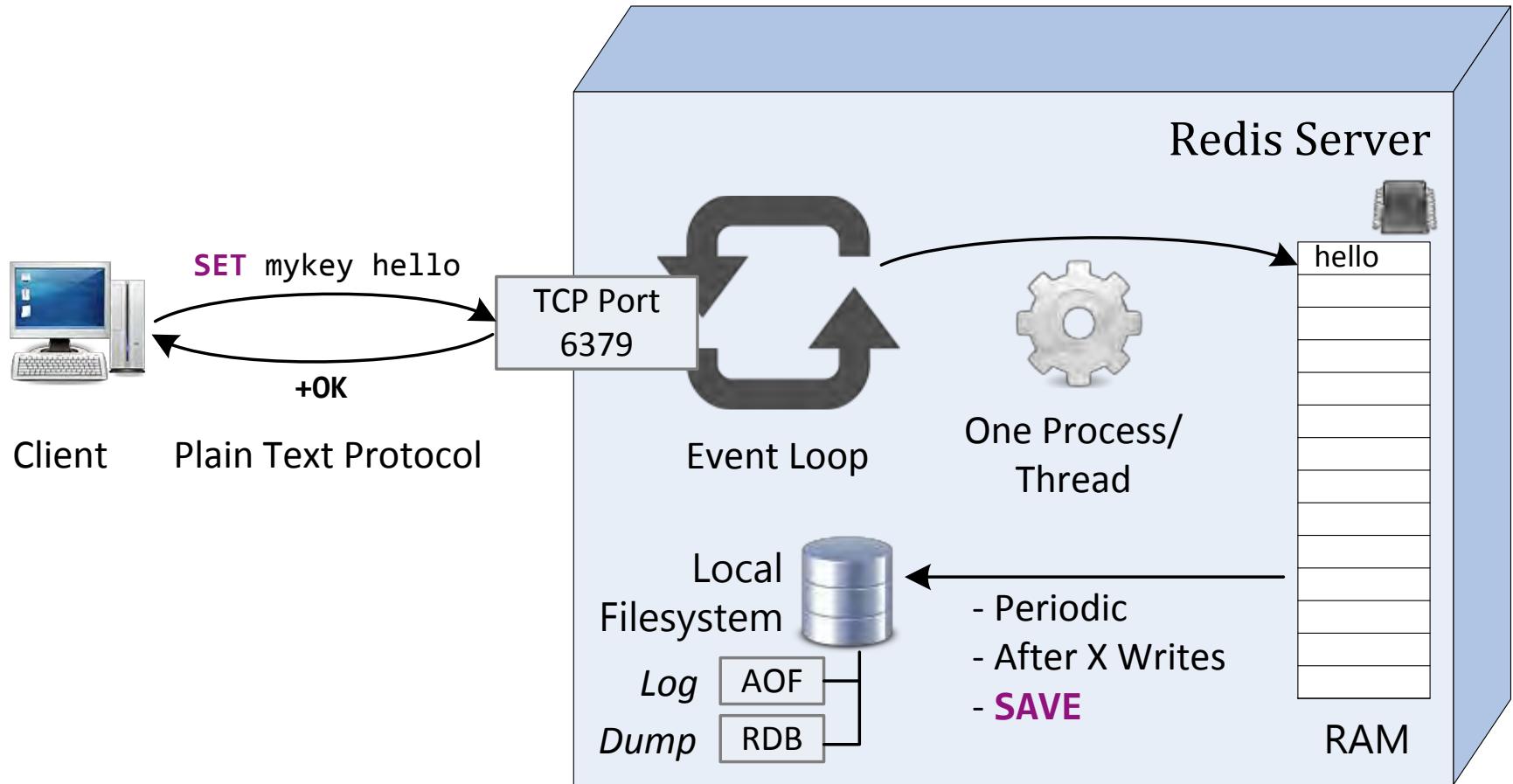
Redis (CA)

- ▶ Remote Dictionary Server
- ▶ 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
Model:
Key-Value
License:
BSD
Written in:
C

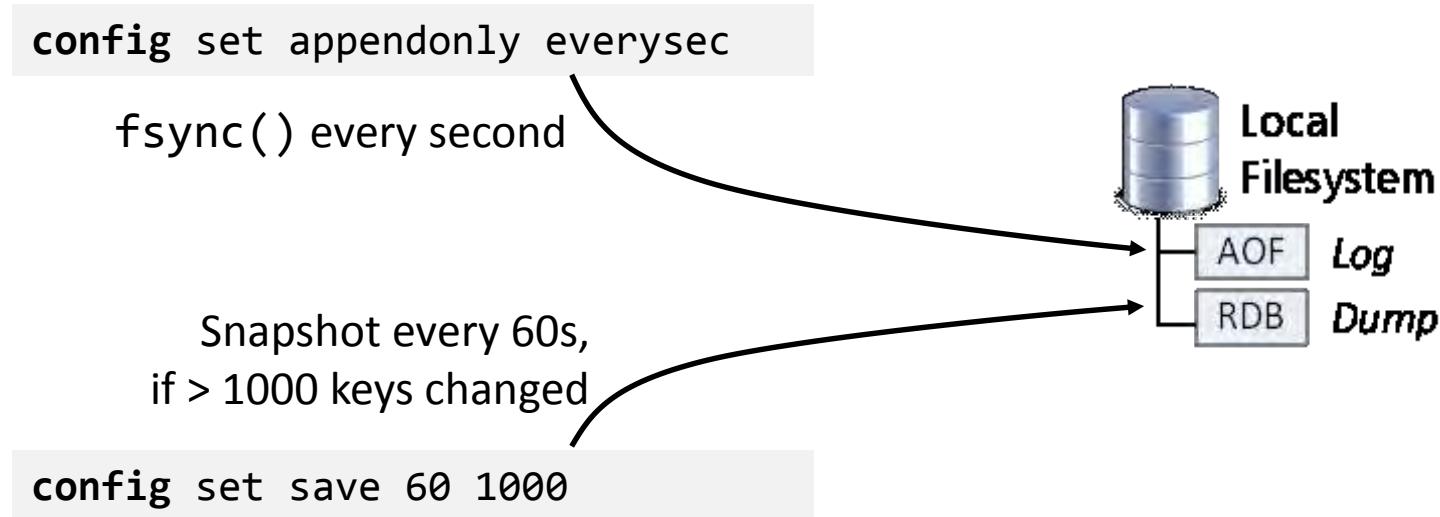
Redis Architecture

- ▶ Redis Codebase \cong 20K LOC



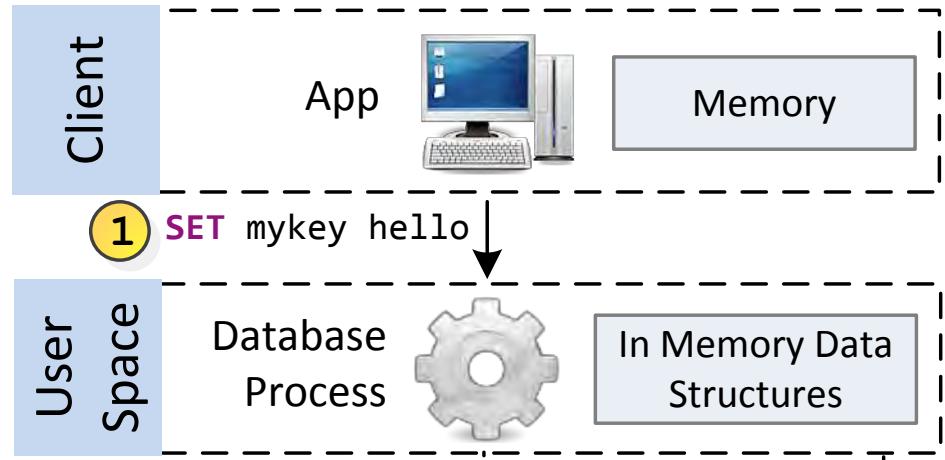
Persistence

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



Persistence

1. Resistance to client crashes
2. Resistance to DB process crashes
3. Resistance to hardware crashes with *Write-Through*
4. Resistance to hardware crashes with *Write-Back*



Persistence: Redis vs an RDBMS

▶ PostgreSQL:

> **synchronous_commit** on

Latency > Disk Latency, Group Commits, Slow

> **synchronous_commit** off

> **appendfsync** **everysec**

periodic fsync(), data loss limited

> **fsync** **false**

Data corruption and loss possible

> **pg_dump**

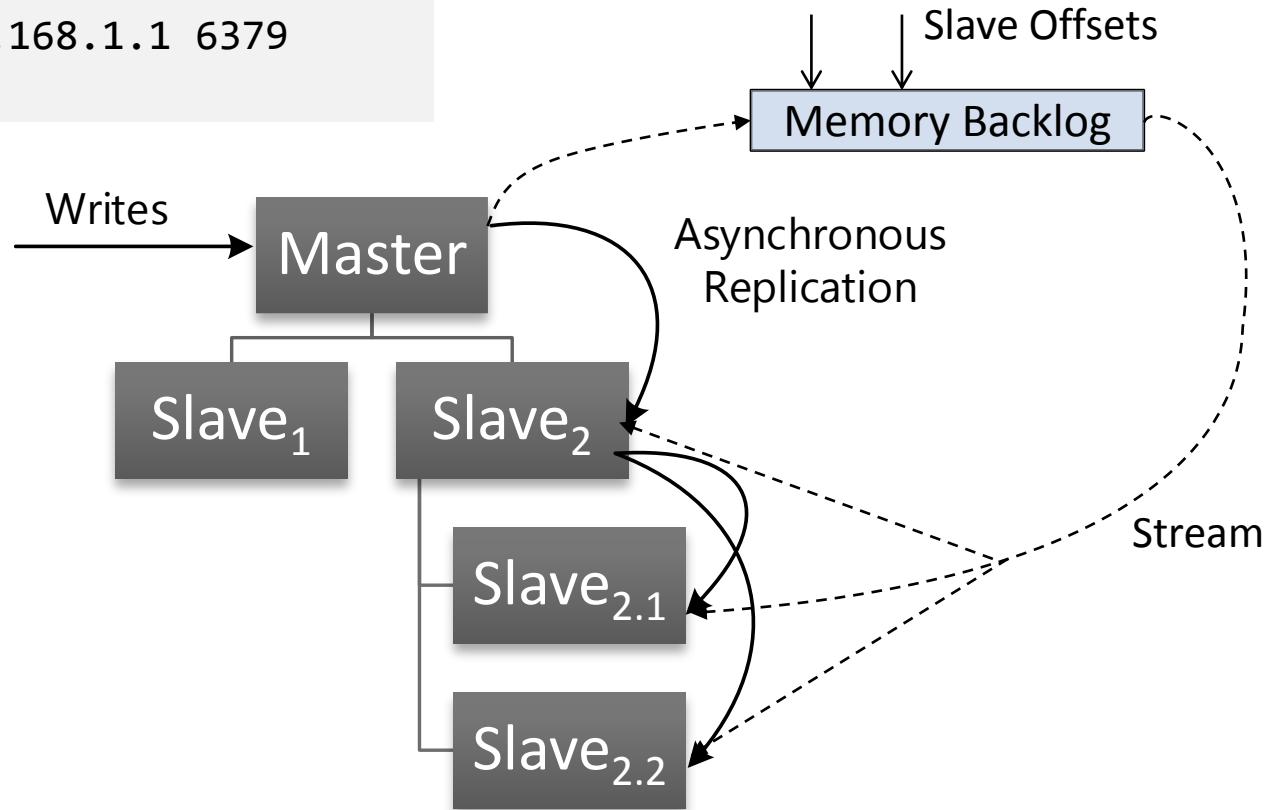
> **appendfysnc** **no**

Data loss possible, corruption prevented

> **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

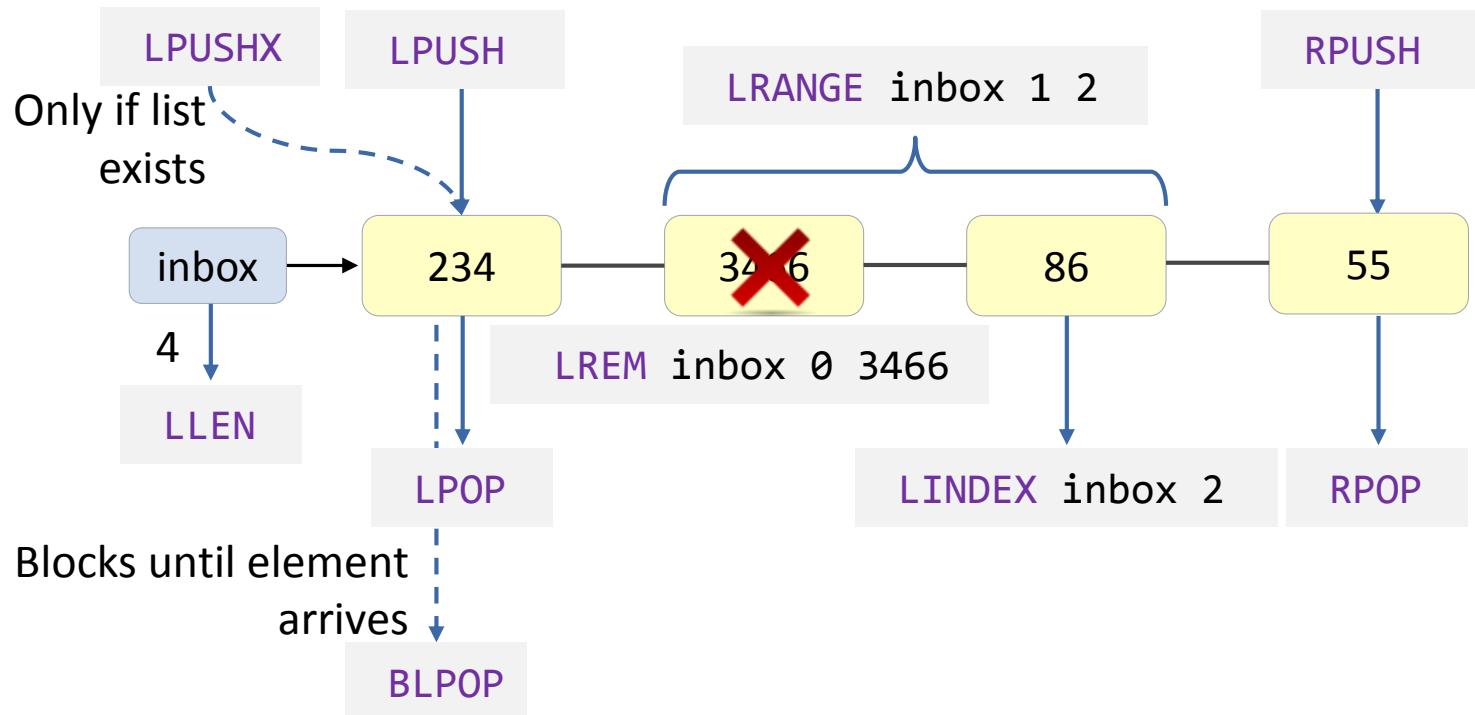
top-posters → 466 → "2", 344 → "16"

Pub/Sub

users:2:notifs → "{event: 'comment posted', time : ...}"

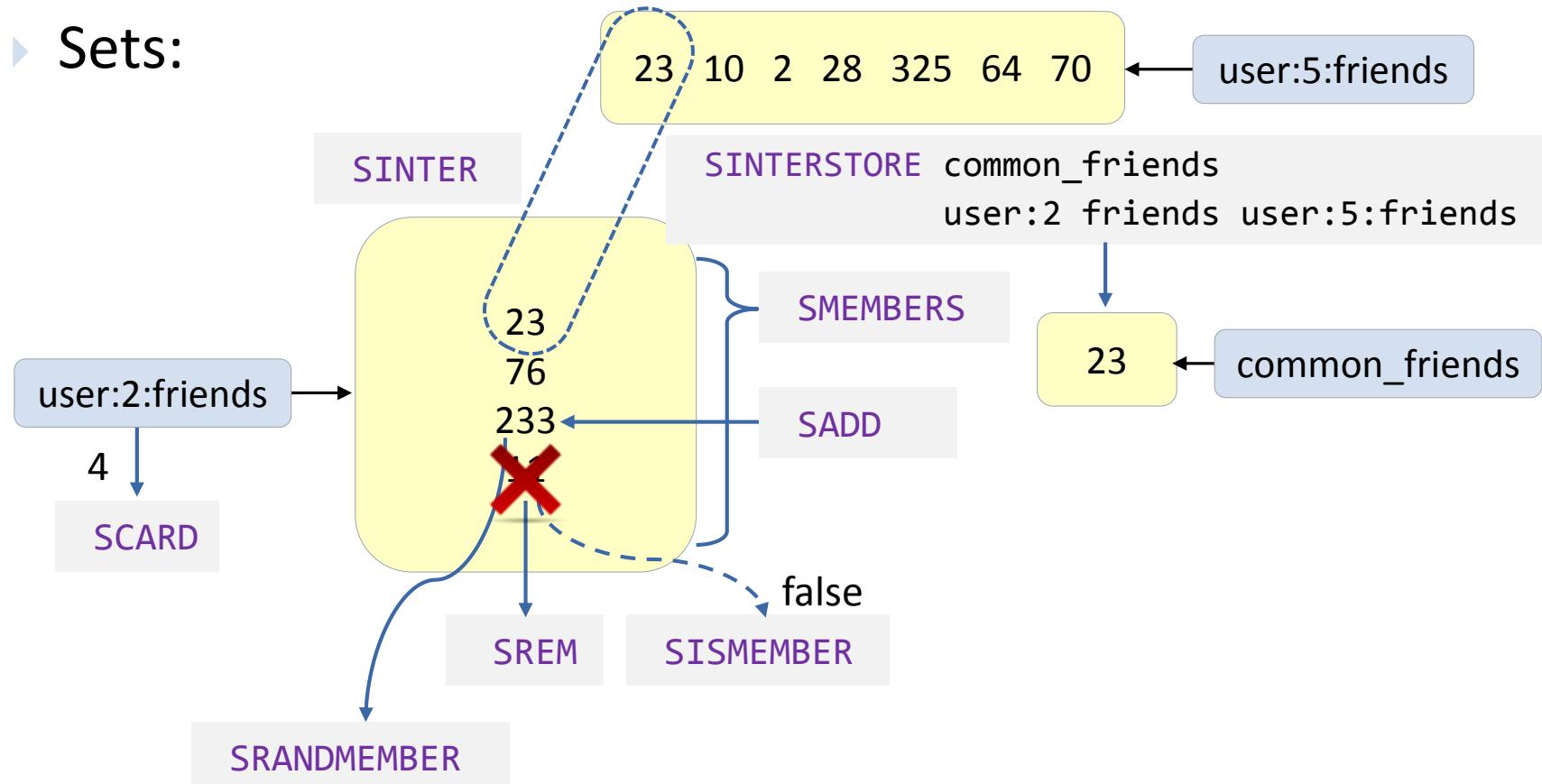
Data Structures

▶ (Linked) Lists:



Data Structures

Sets:



Data Structures

▶ Pub/Sub:

users:2:notifs

→ "{event: 'comment posted', time : ...}"

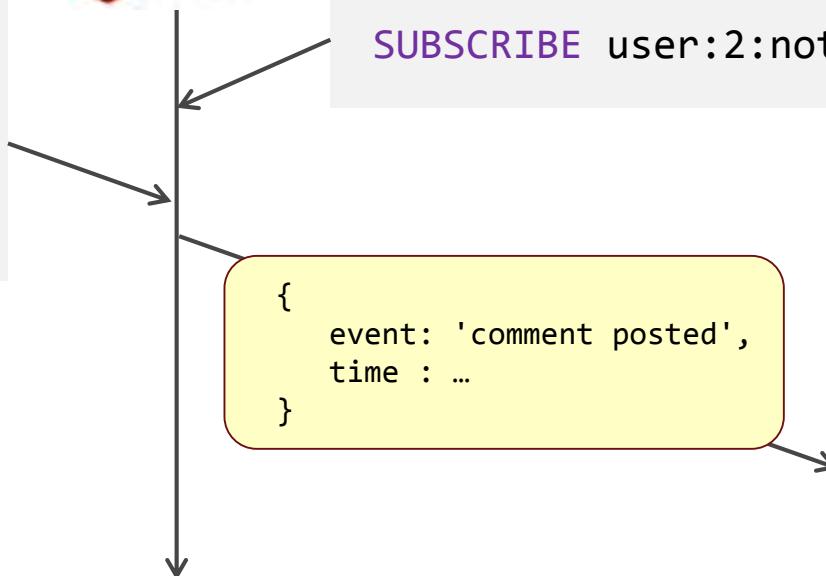
```
PUBLISH user:2:notifs
"{
  event: 'comment posted',
  time : ...
}"
```



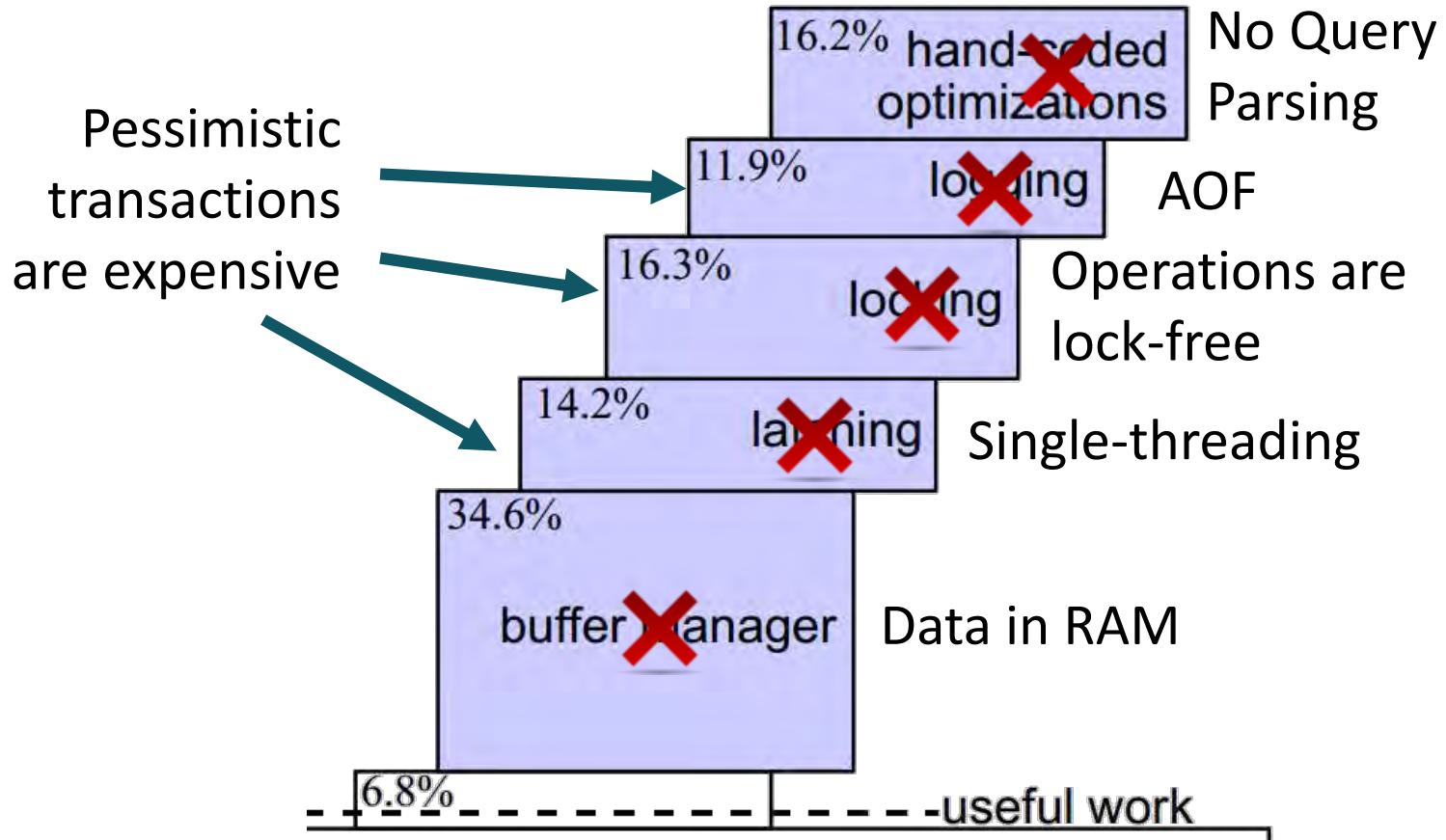
redis

SUBSCRIBE user:2:notifs

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



Why is Redis so fast?



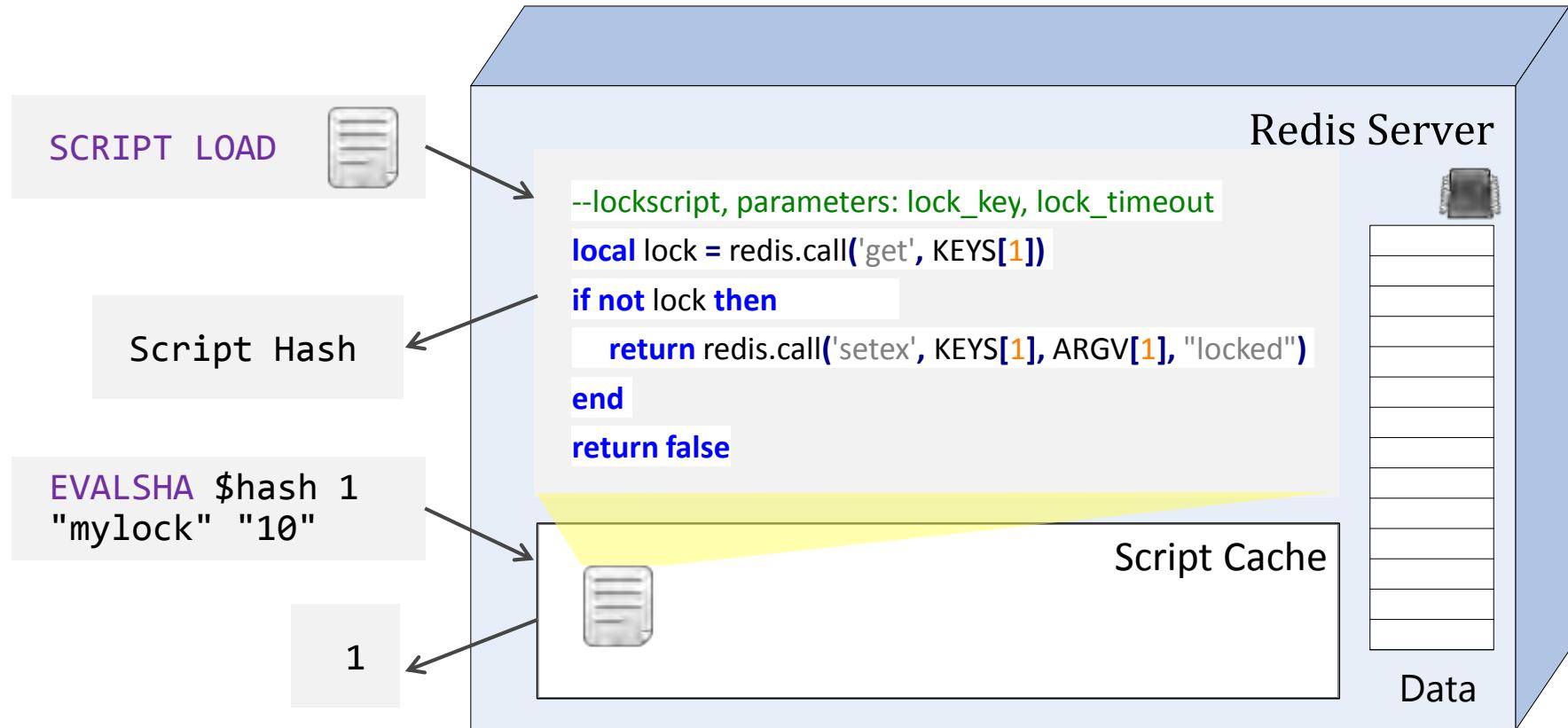
Optimistic Transactions

- ▶ 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
```

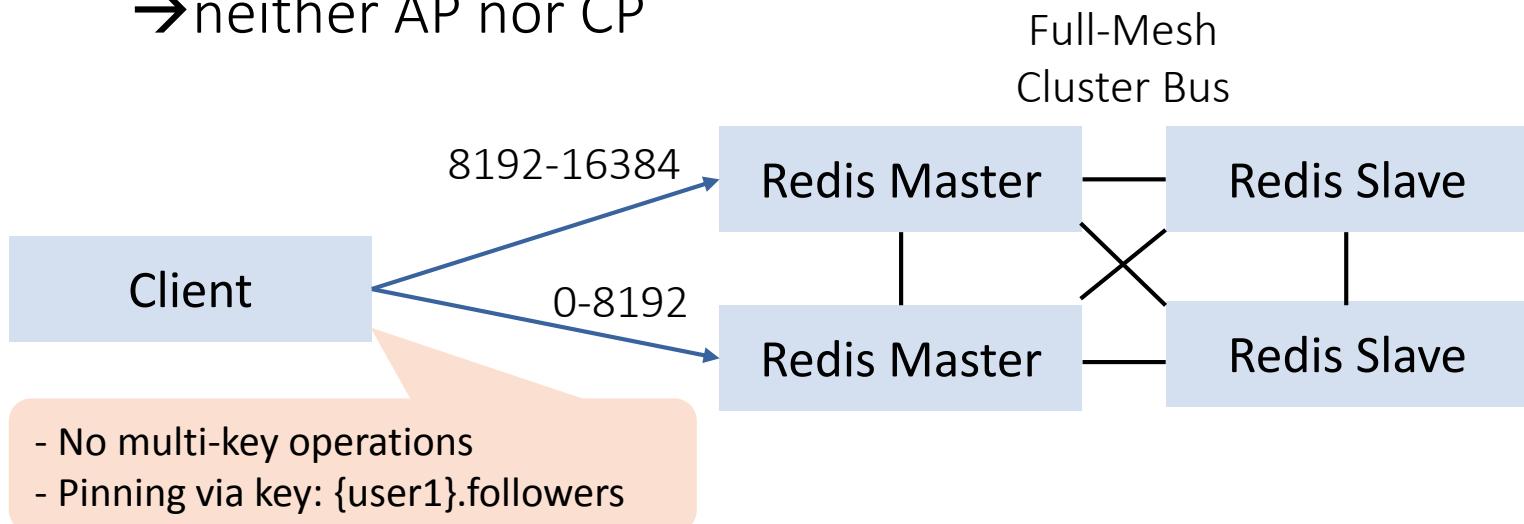
Lua Scripting



Redis Cluster

Work-in-Progress

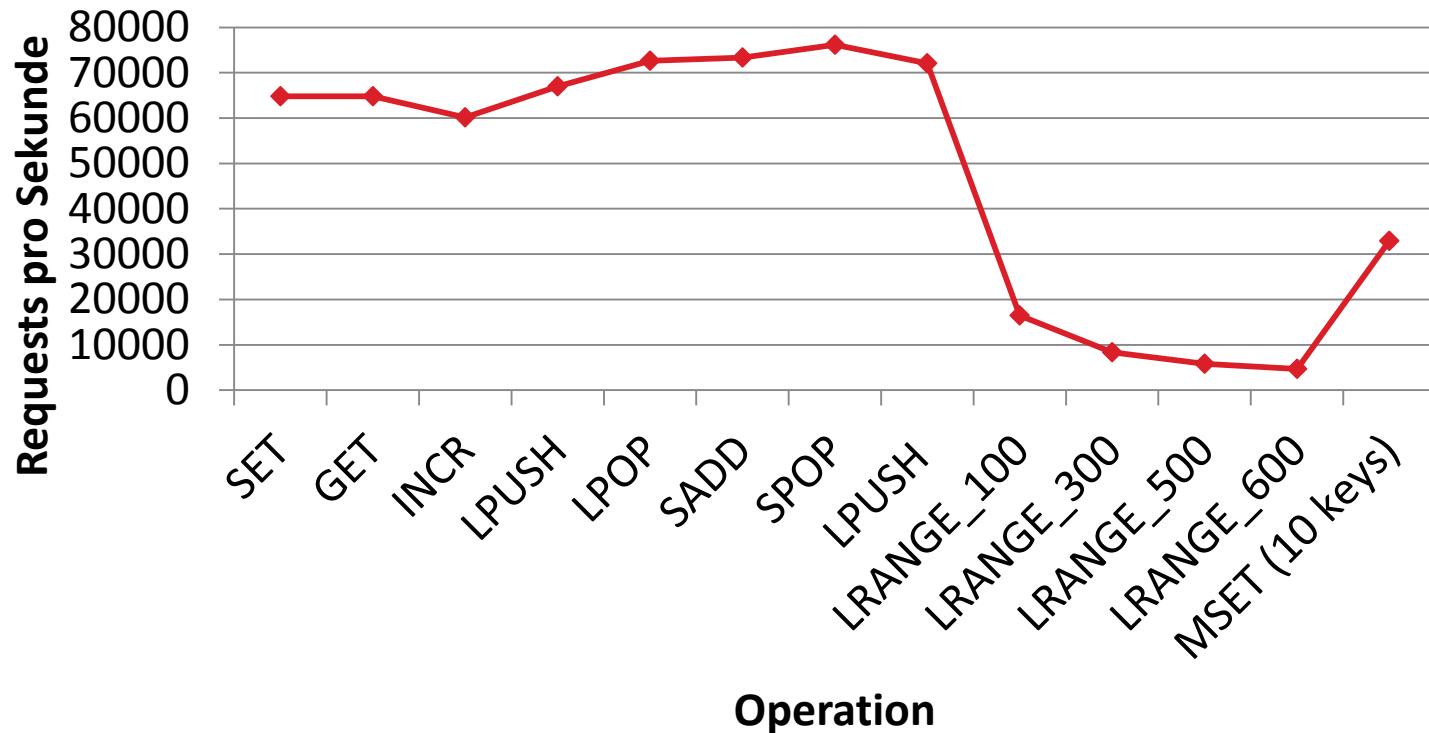
- ▶ **Idea:** Client-driven hash-based sharing (CRC32, „hash slots“)
- ▶ **Asynchronous replication with failover** (variant of Raft's leader election)
 - **Consistency:** not guaranteed, last failover wins
 - **Availability:** only on the majority partition
→ neither AP nor CP



Performance

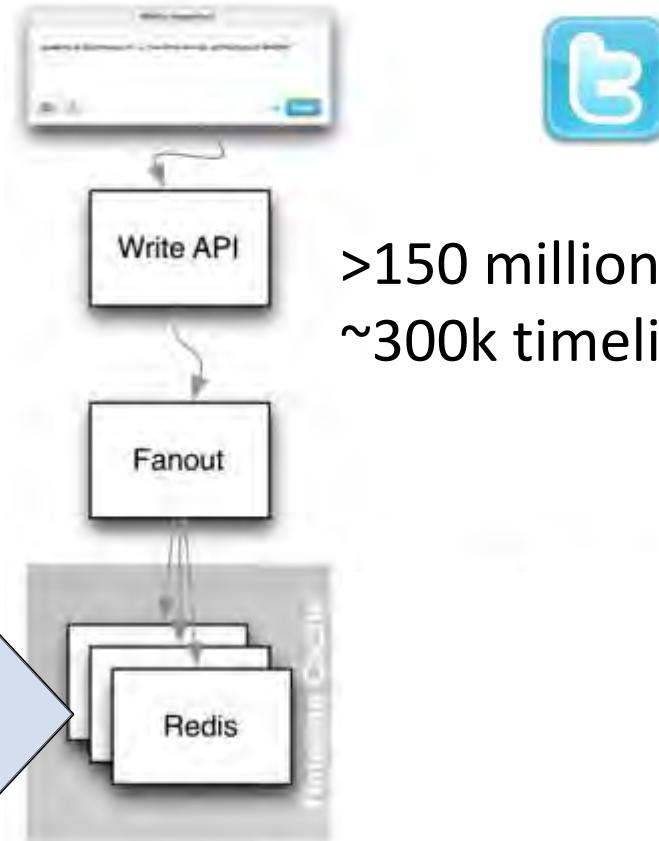
- ▶ Comparable to Memcache

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



Example Redis Use-Case: Twitter

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



Classification: Redis

Techniques

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

Google BigTable (CP)

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

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

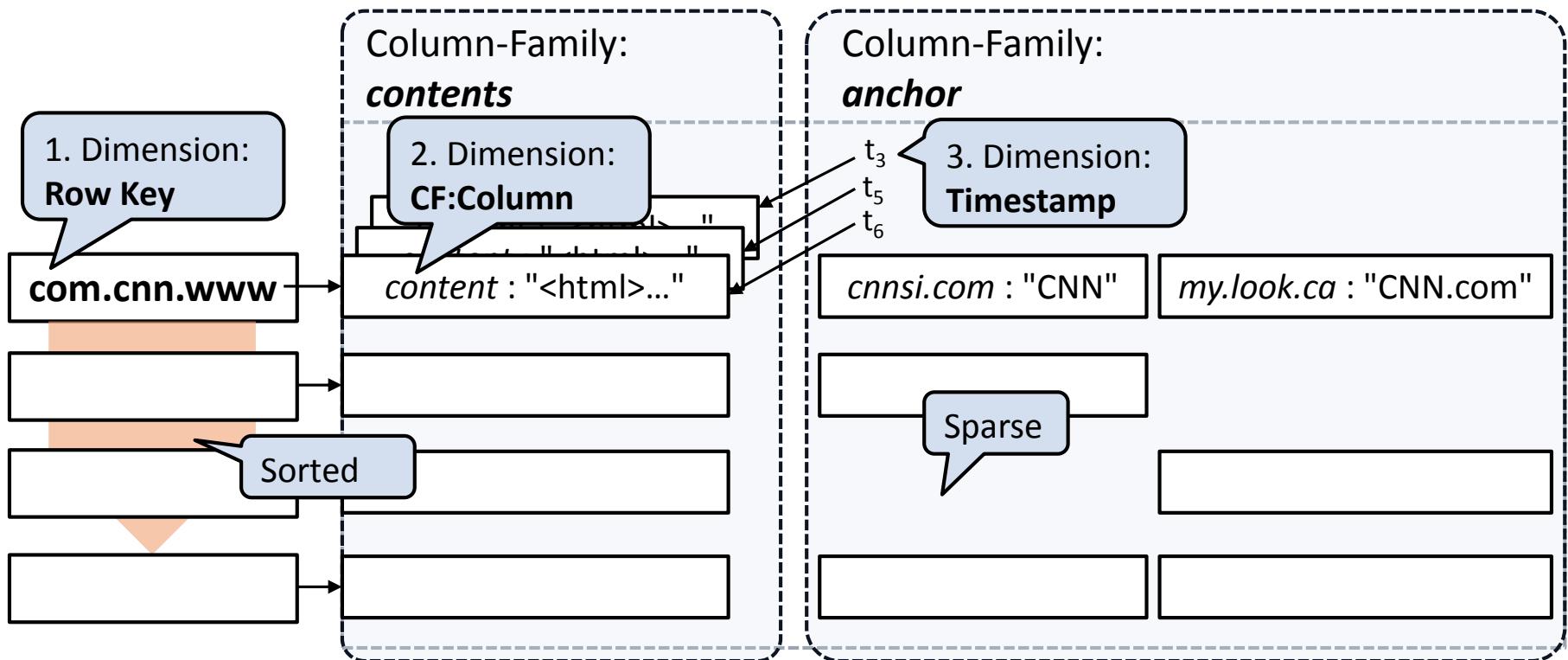
- ▶ Data model also used in: **HBase**, **Cassandra**, **HyperTable**, **Accumulo**



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

Wide-Column Data Modelling

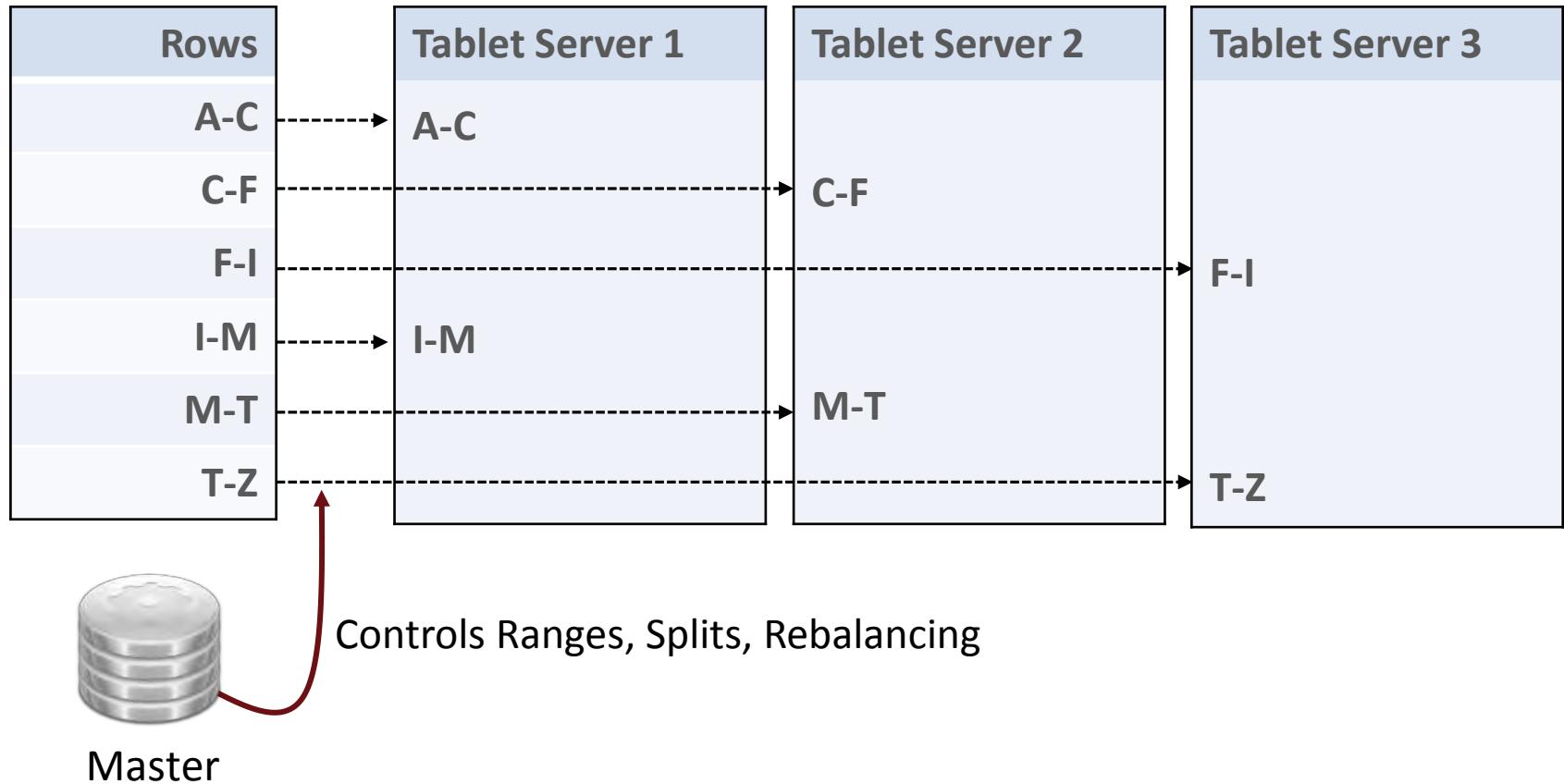
- ▶ Storage of crawled web-sites („Webtable“):



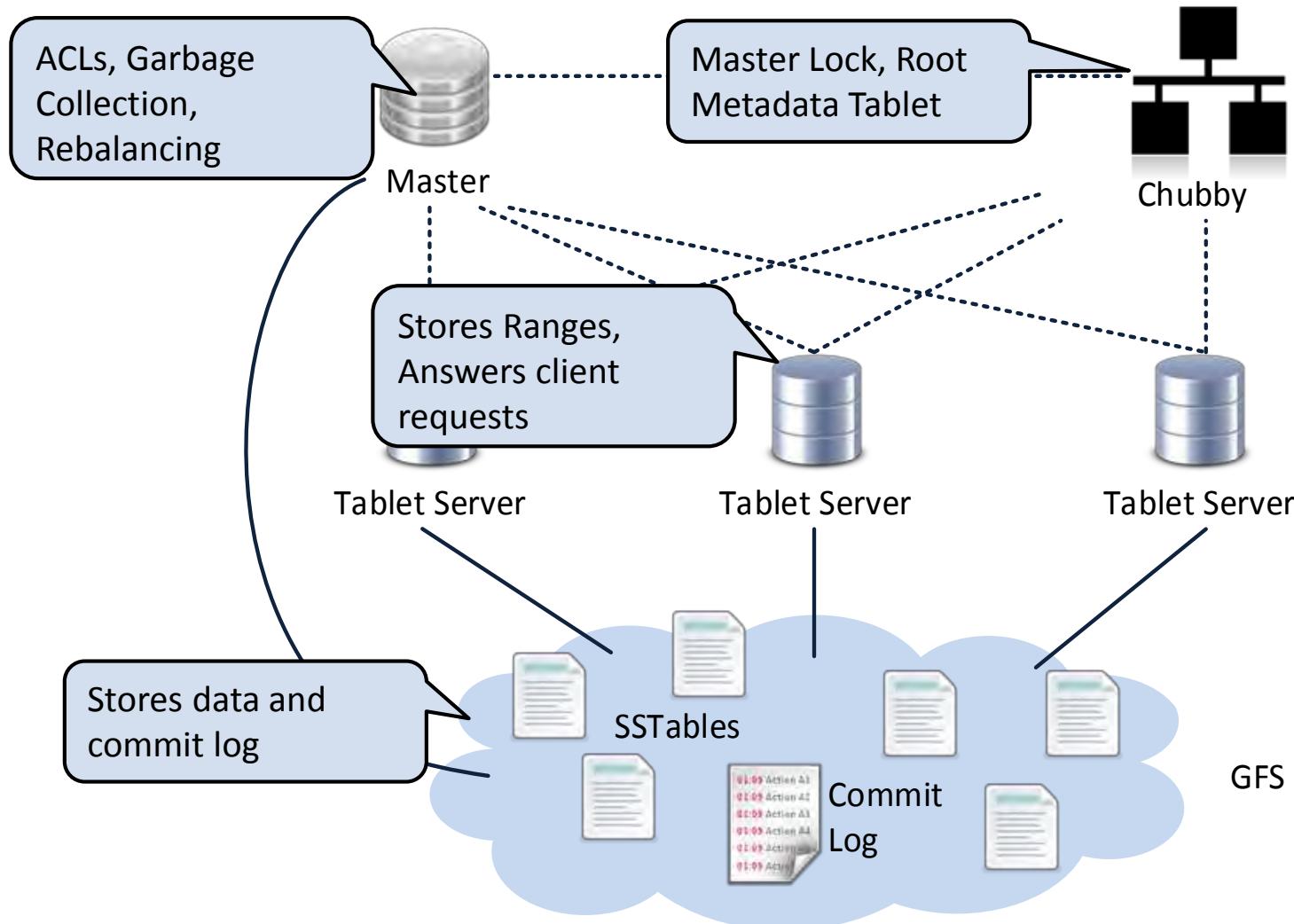
Range-based Sharding

BigTable Tablets

Tablet: Range partition of ordered records

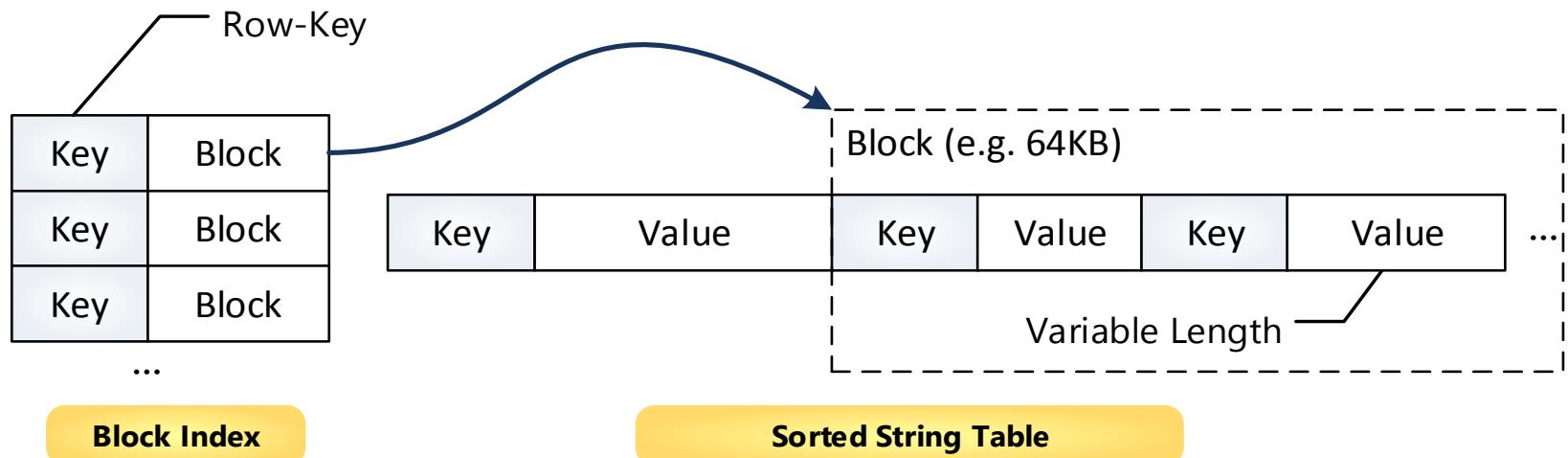


Architecture



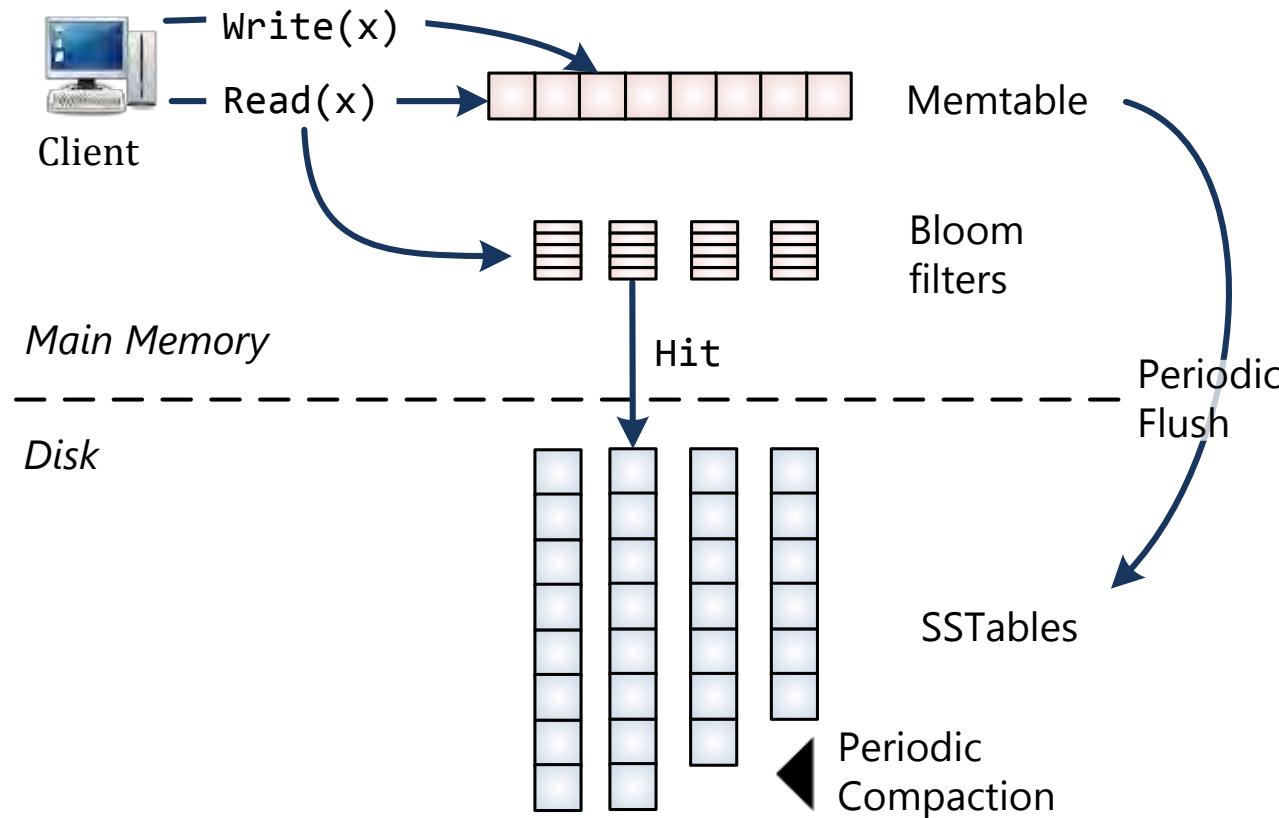
Storage: Sorted-String Tables

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



Storage: Optimization

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



Apache HBase (CP)

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

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

HBase Storage

▶ Logical to physical mapping:

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



Example: Facebook Insights



MD5(Reversed Domain) + Reversed Domain + URL-ID									Row Key
6PM Total	6PM Male	...	01.01 Total	01.01 Male	...	Total	Male	...	
10	7		100	65		567			
CF:Daily				CF:Monthly				CF>All	

Atomic HBase Counter

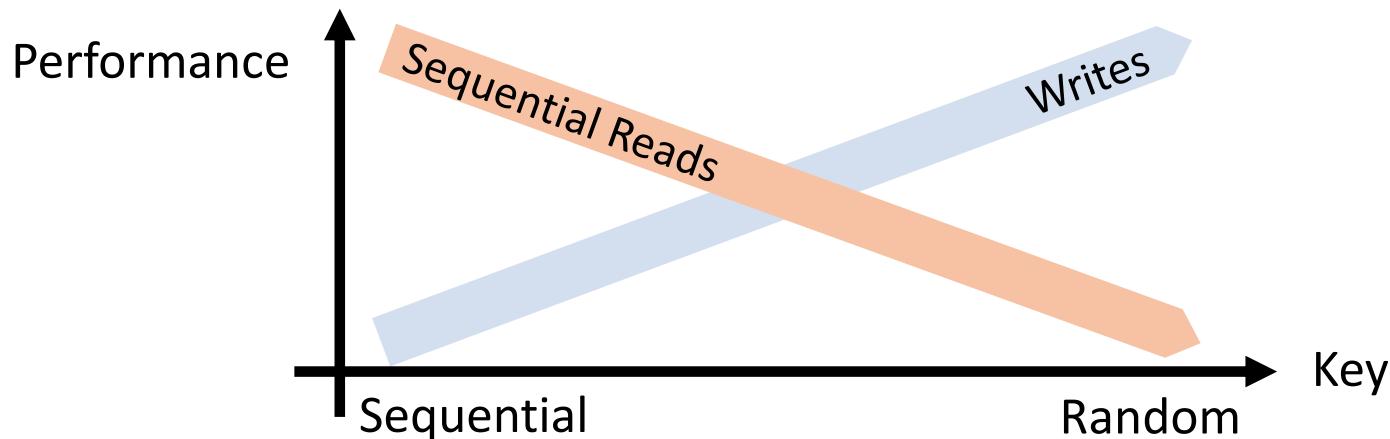
TTL – automatic deletion of old rows



Lars George: "Advanced HBase Schema Design"

Schema Design

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



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**

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 - registered 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)

- ▶ 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

Cassandra

Model:

Wide-Column

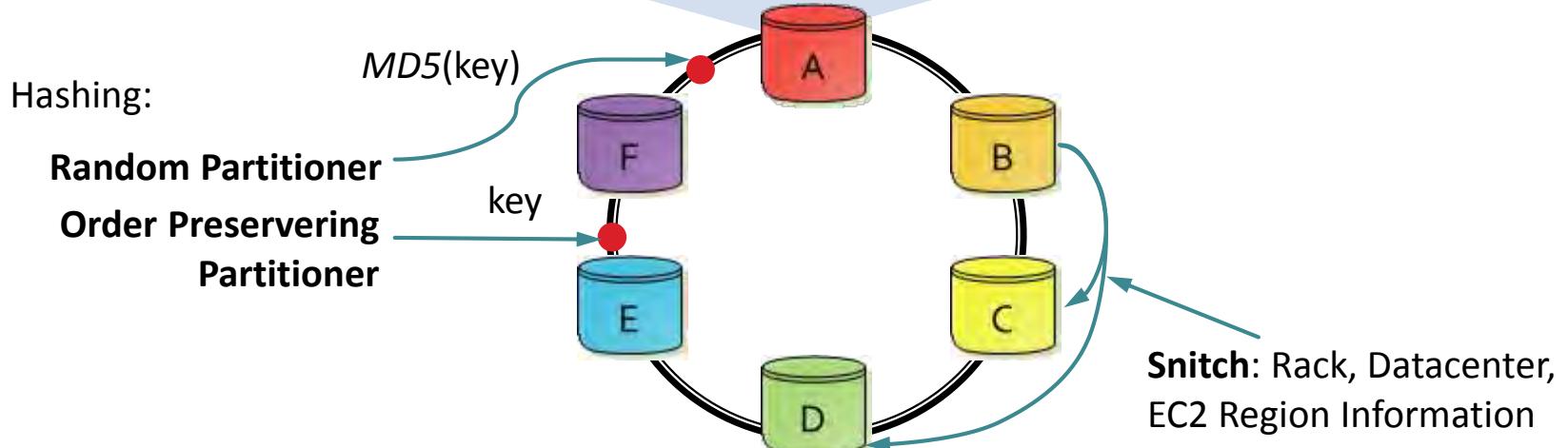
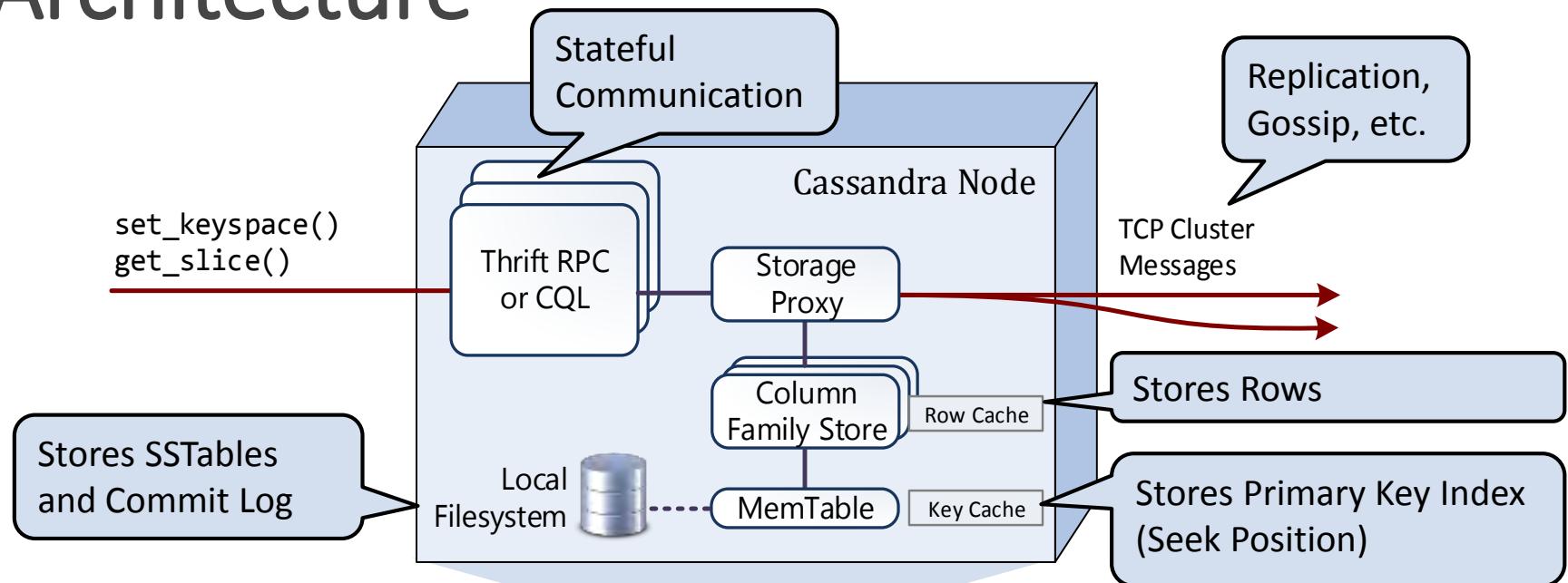
License:

Apache 2

Written in:

Java

Architecture



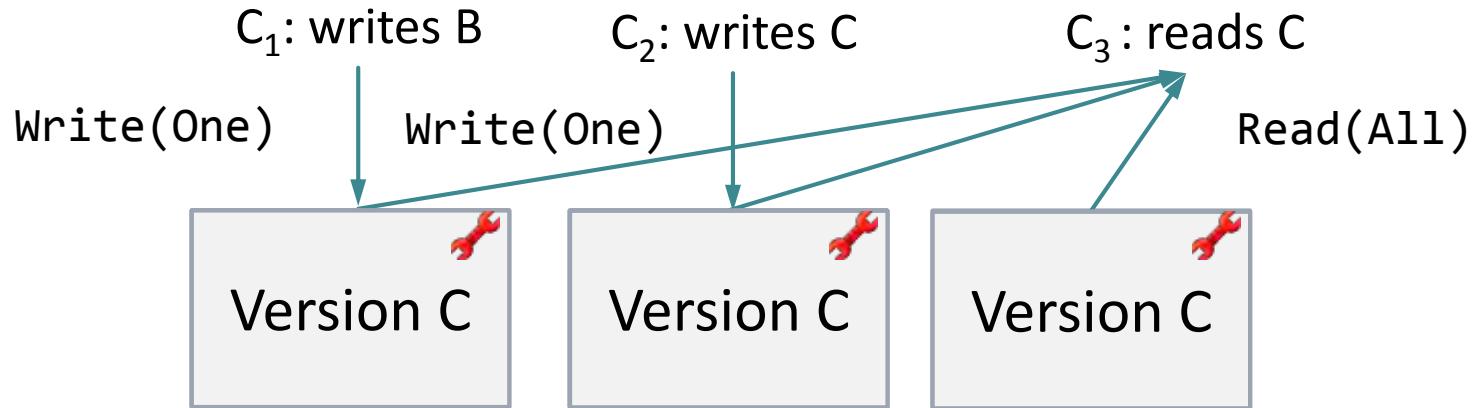
Consistency

- ▶ 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

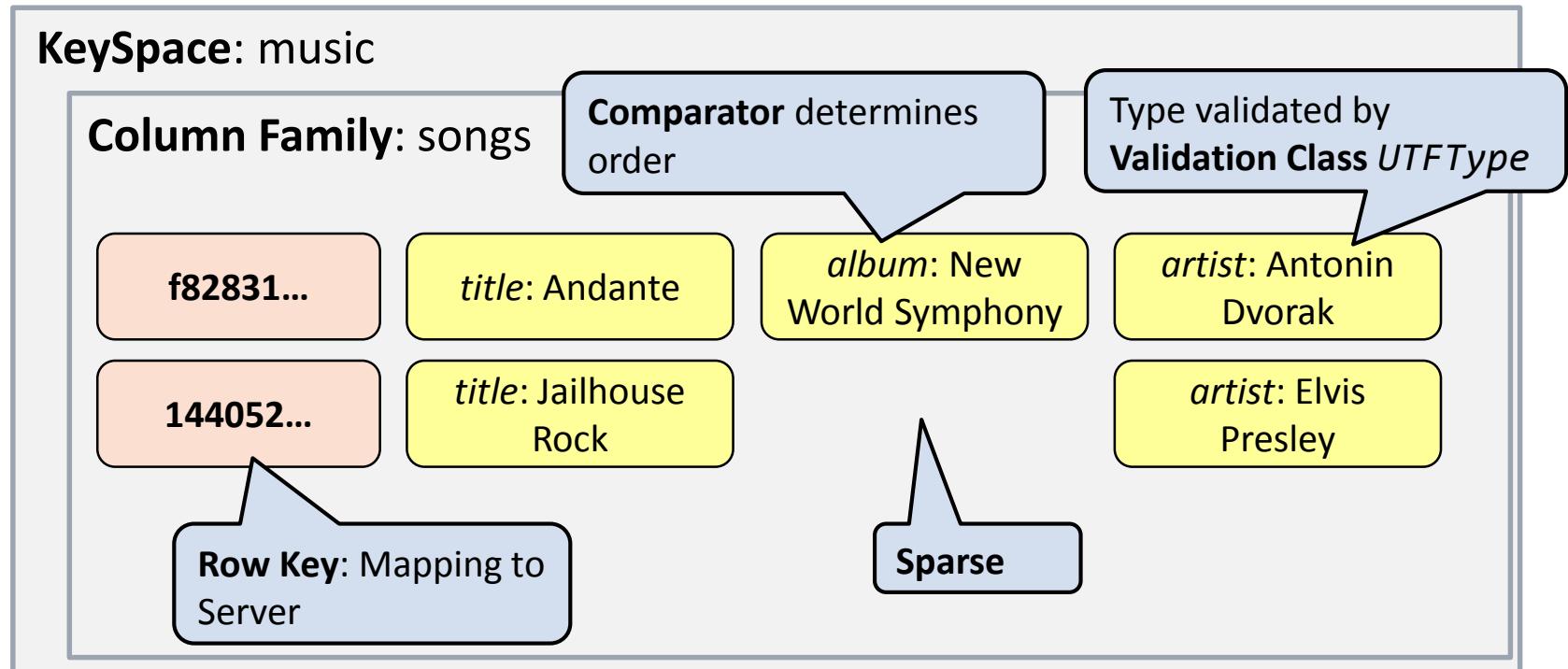
Consistency

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



Storage Layer

- ▶ Uses BigTables Column Family Format



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;
```

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

Partition Key

Clustering Columns:
sorted per node

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	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
	Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
	Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views

MongoDB (CP)

- ▶ From **humongous** \cong 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), ...

MongoDB

Model:

Document

License:

GNU AGPL 3.0

Written in:

C++

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

```
{  
    "_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 : [ {  
        "text": "#nowwatching Iron Man 3",  
        "user": "Franz Kafka",  
        "ordinates": 1  
    },  
    {"text": "#nowwatching Iron Man 3",  
     "retweet": false,  
     "date": ISODate("2013-05-29T13:15:51Z")  
    }]  
}
```

Movie Document

Genre

Actor

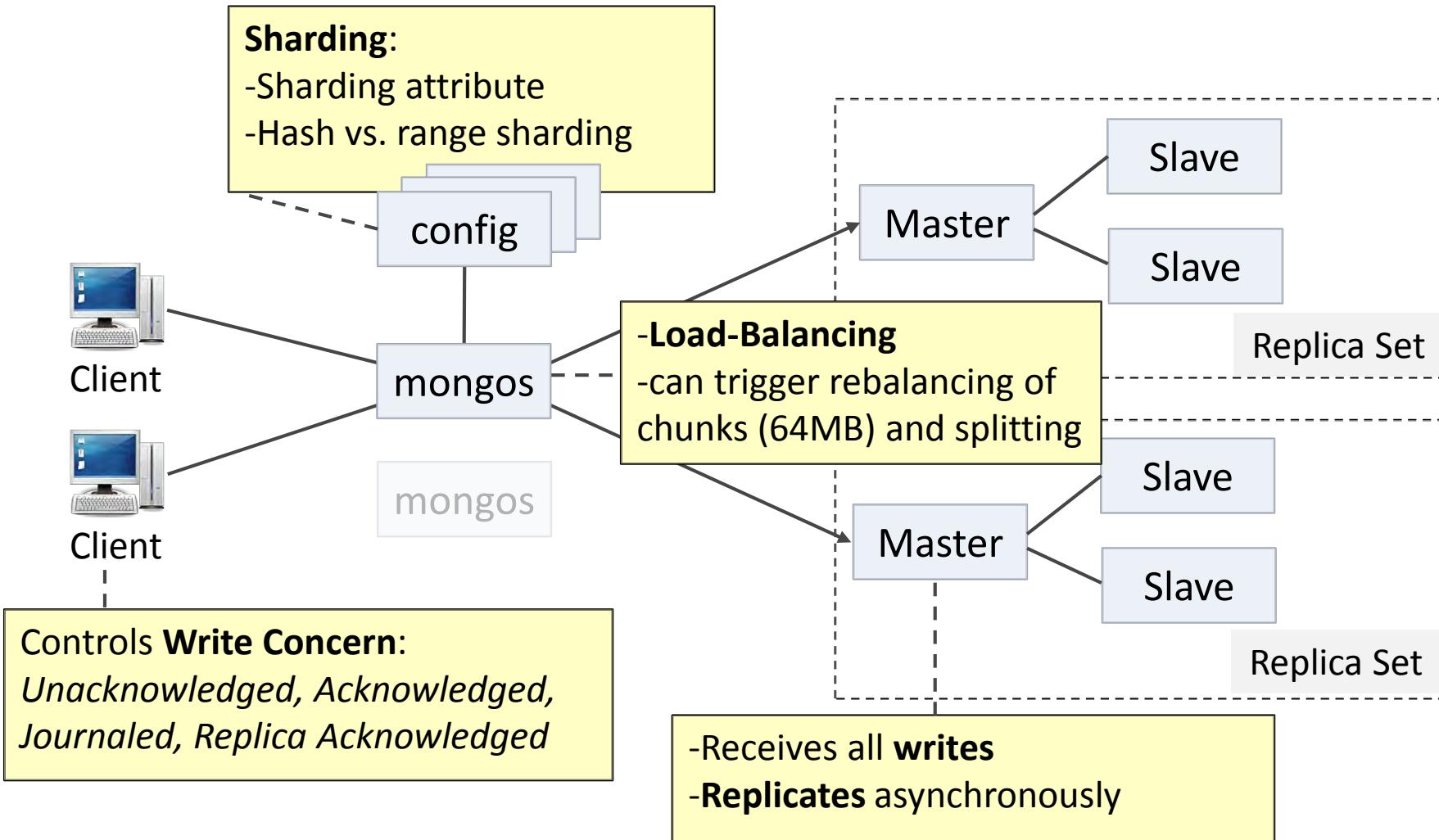
Tweet

User

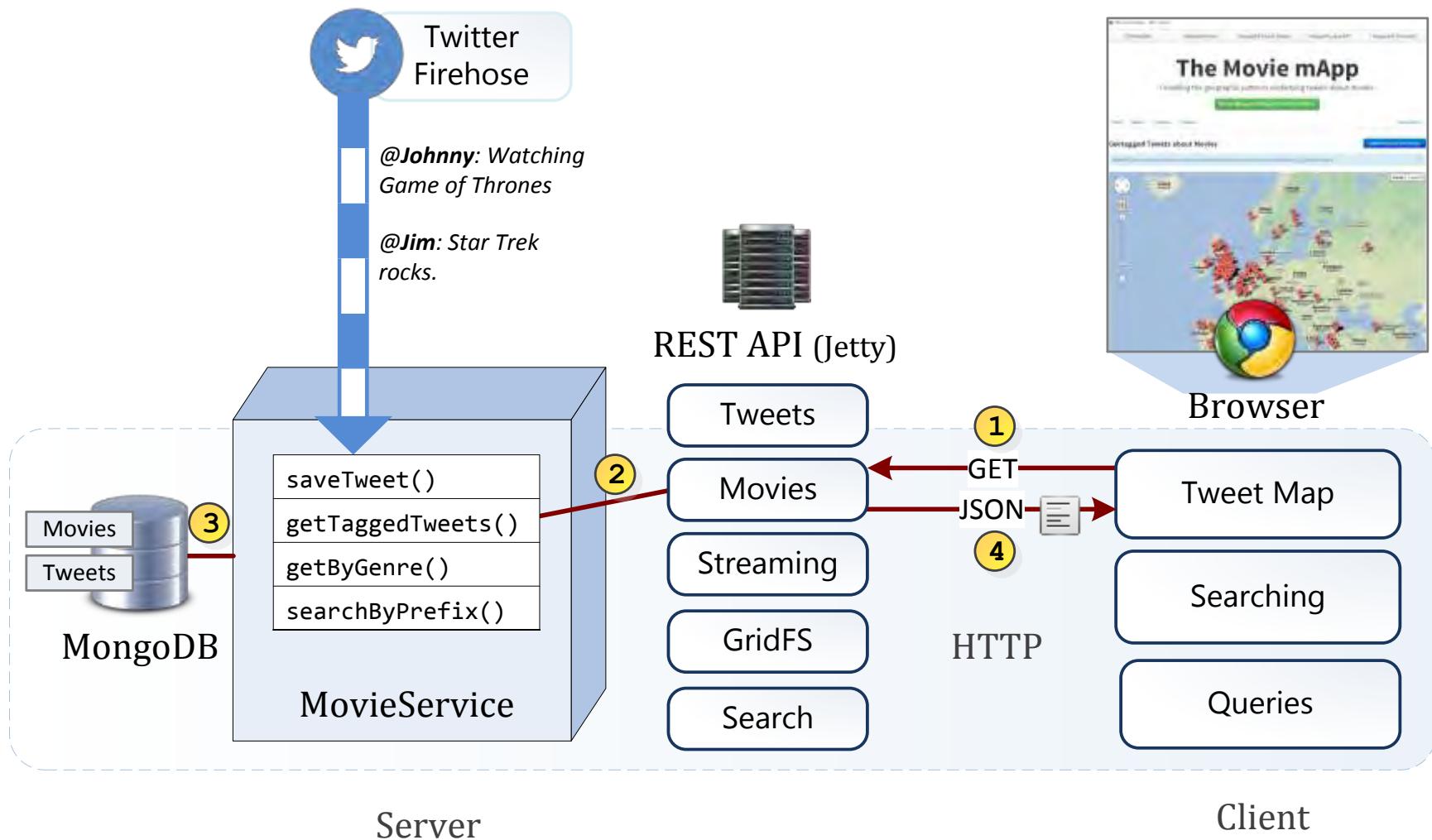
name
location

Principles

Sharding und Replication



MongoDB Example App



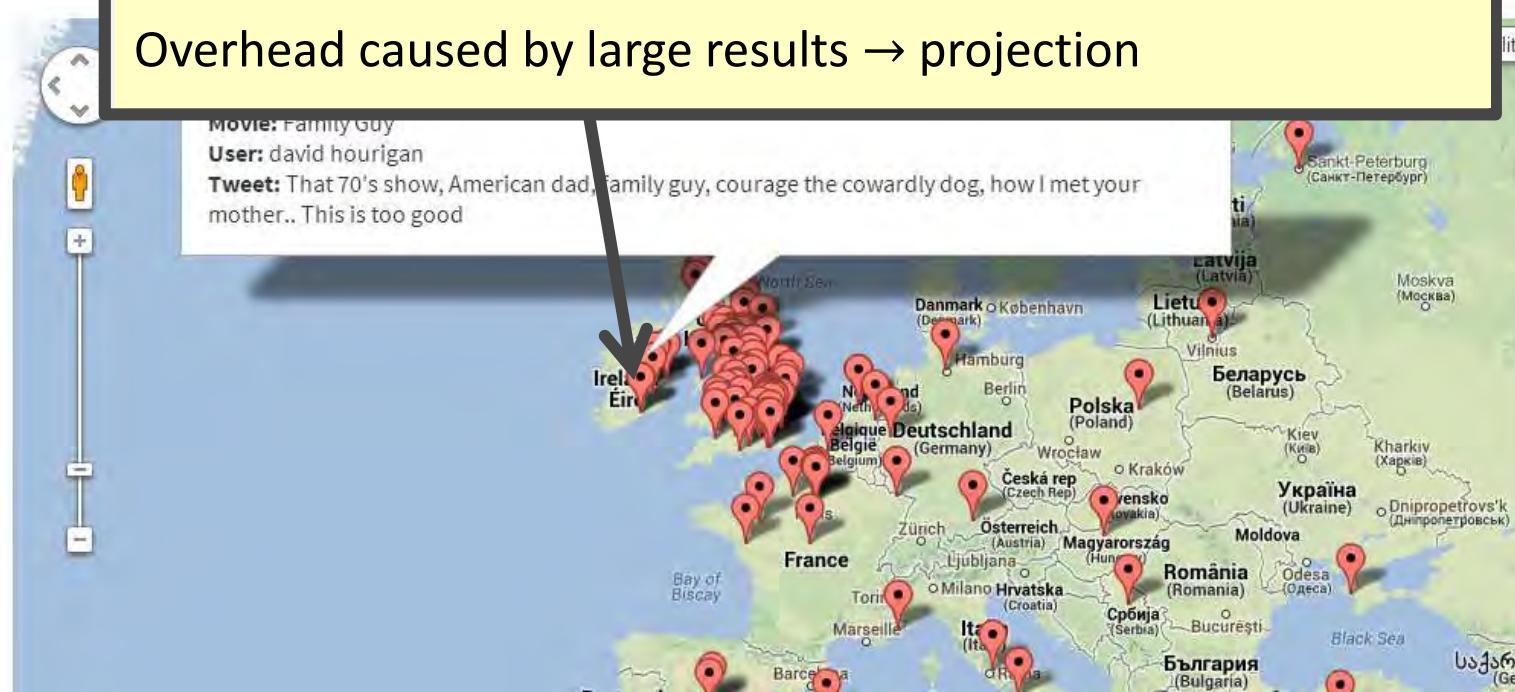
MongoDB by Example

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



The Movie mApp

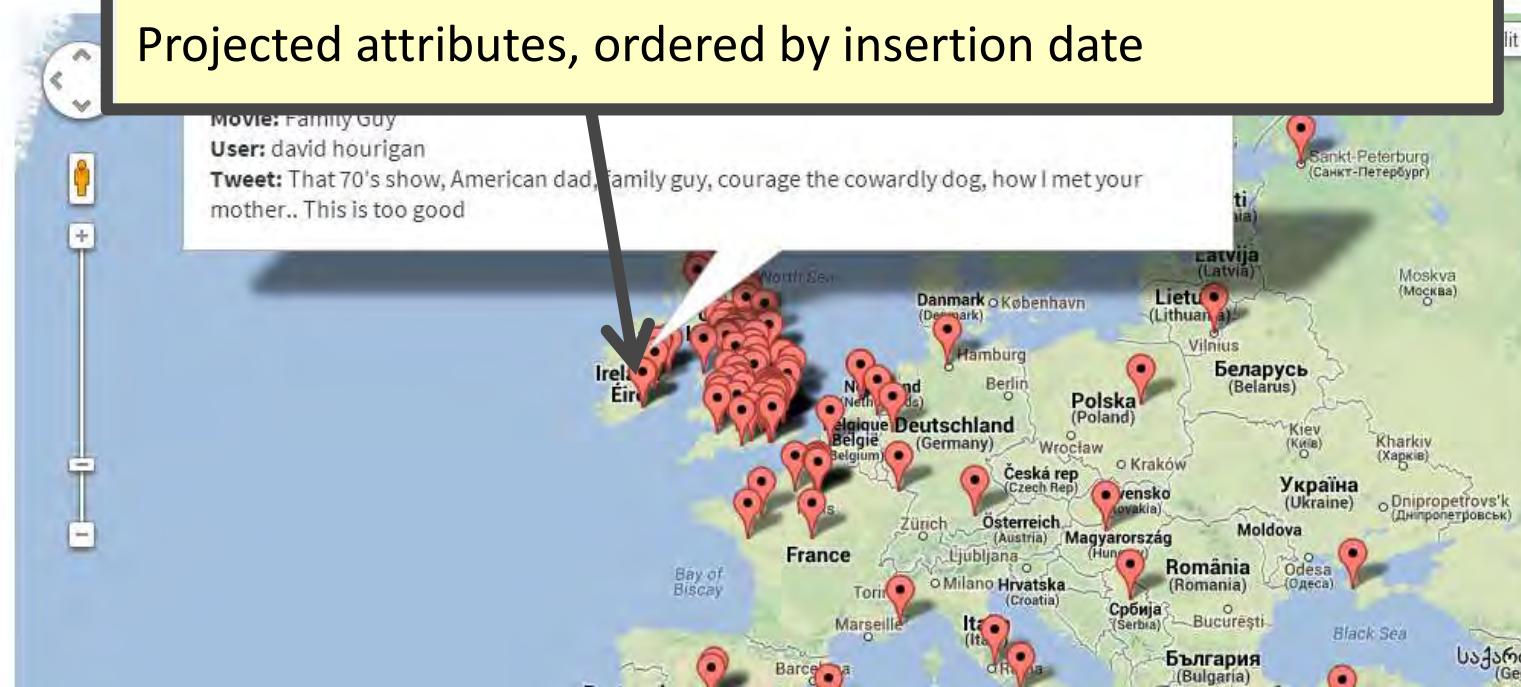
Unveiling the geographic patterns underlying tweets about movies.

Show Mongo at <http://127.0.0.1:28017/>

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



Projected attributes, ordered by insertion date



Search for Movie and Its Tweets

Movie Incep

Inception

Inception: Motion Comics

Inception: 4Movie Premiere Special

Stream Tweets in Background

Keywords (comma-separated)

Comma-separated Movie Names

Total Tweets to Stream

100

 Only geotagged tweets**Start Streaming**

Title	Incep
Poster	

```
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
```

Upload:

Datei auswählen

Keine ausgewählt

Title Inception

Poster



Import Poster from IMDB

@TRIXIA : #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);
```

Upload

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

Upload:

Datei auswählen

Keine ausgewählt

Comment

Editable. You can edit and save this comment.

One o

```
fs = new GridFs(db);
fs.createFile(inputStream).save();
```

Year 2010

Rating 8.8

Votes 542923

Runtime 148 mi

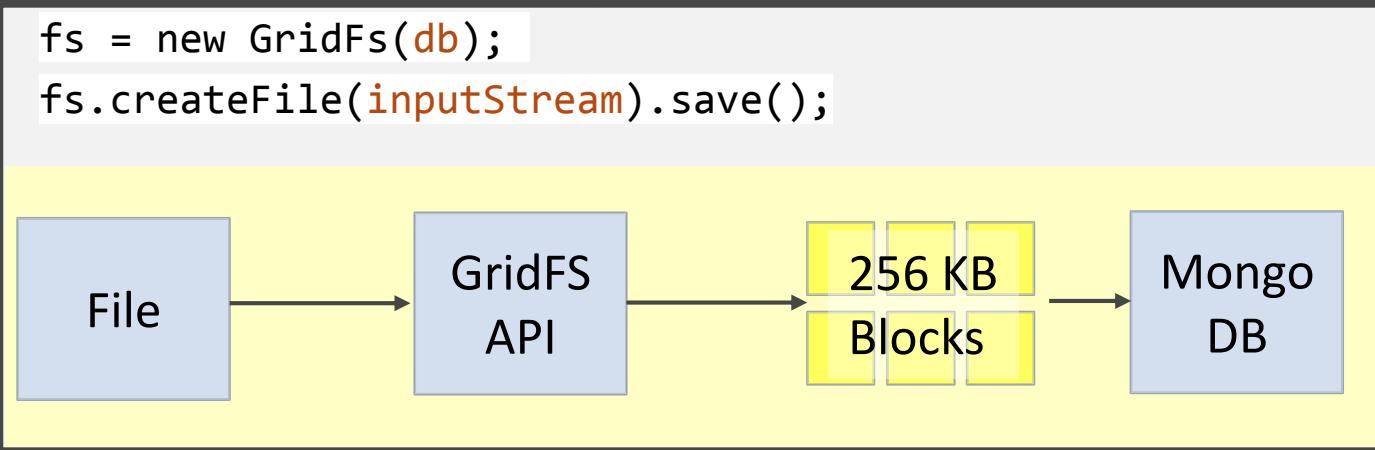
Genre Action

Plot Dom Cobb is a skilled thief, the absolute best in the dangerous art of extraction, stealing valuable secrets from deep within the subconscious during the dream state, when the mind is at its most vulnerable. Cobb's rare ability has made him a coveted player in this

@TRIXIA: #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 Hedges: What if Stacy's mom was Jessie's girlfriend and her number was 867-5309? #Inception



Query Tweets

Query

Get Tweets Near: lat,lng,radius-in-km

Go

Parameter

51.54155217692421,10.406249463558197,1000

Result Limit

10



User

Tweet

Created at

Coordinates

MitchellyMonica

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

J. Z.

Party Hardy

nadine stachowiak

Geospatial Queries:

- Distance
- Intersection
- Inclusion

2013

Query Tweets

Query

Indexed Fulltext Search on Tweets

Go

Parameter

StAr trek

Result Limit

100

Show 25

search results per page

Filter search results:

User

Tweet

Created at

Coordinates

manwonman

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

Mia Clrss Hrnndz ❤

ANGGI_

Stefany Ezra Elvina

Full-text Search:

- Tokenization, Stop Words
- Stemming
- Scoring

Vanessa Yung

Star Trek into Darkness

2013

Wed May 29
19:21:06 +0000
2013

-2.986771,53.404051

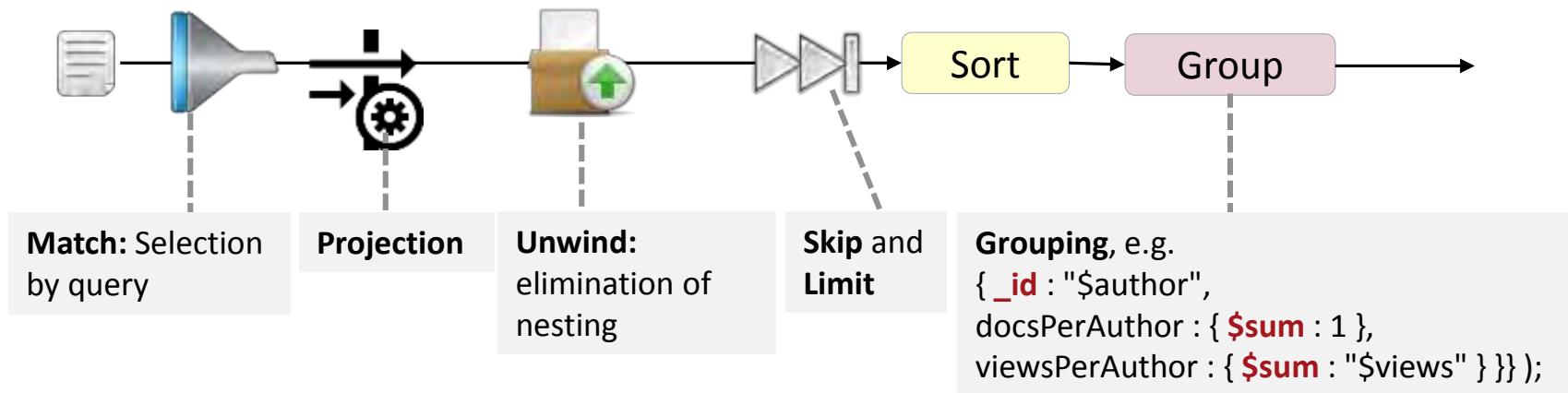
tam wilson

Finally getting to see Star Trek! (at @DCADundee Contemporary Arts
for Star Trek Into Darkness 3D) http://t.co/0ojg4KMBL5Wed May 29
18:48:56 +0000

-2.97489166,56.45753477

Analytic Capabilities

- ▶ Aggregation Pipeline Framework:

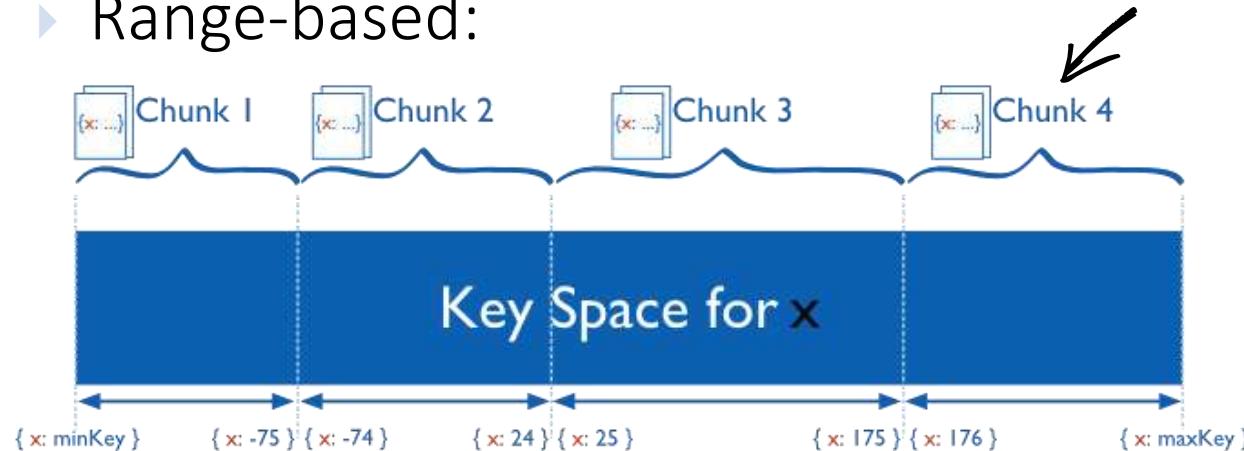


- ▶ Alternative: JavaScript MapReduce

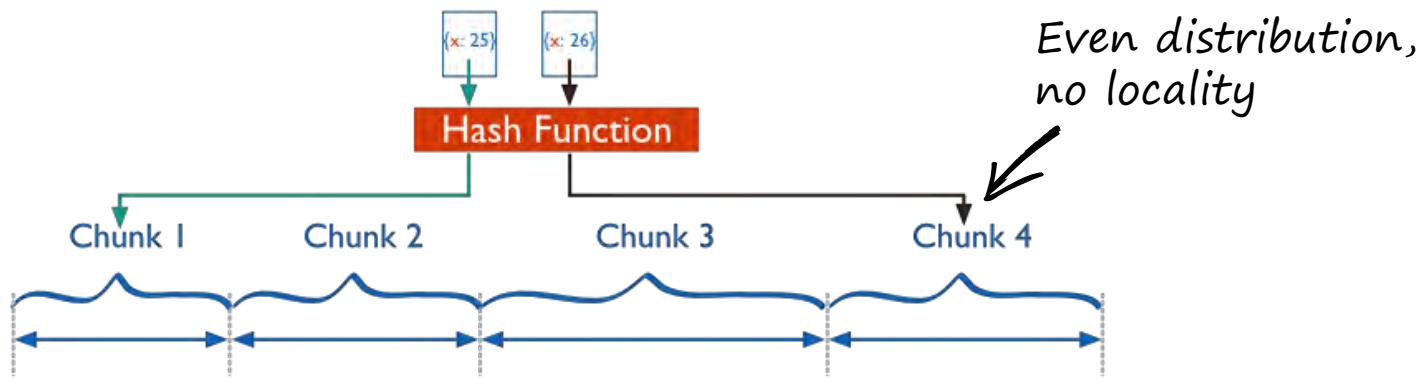
Sharding

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

- ▶ Range-based:

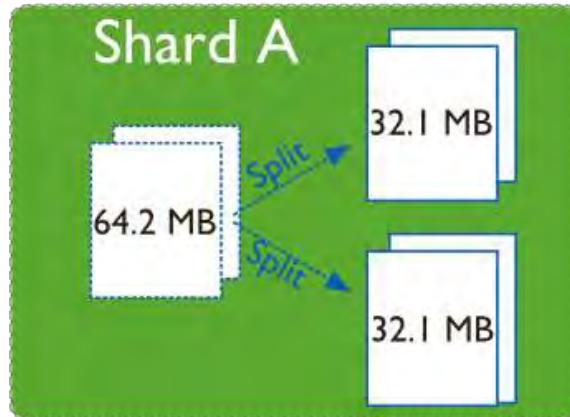


- ▶ Hash-based:

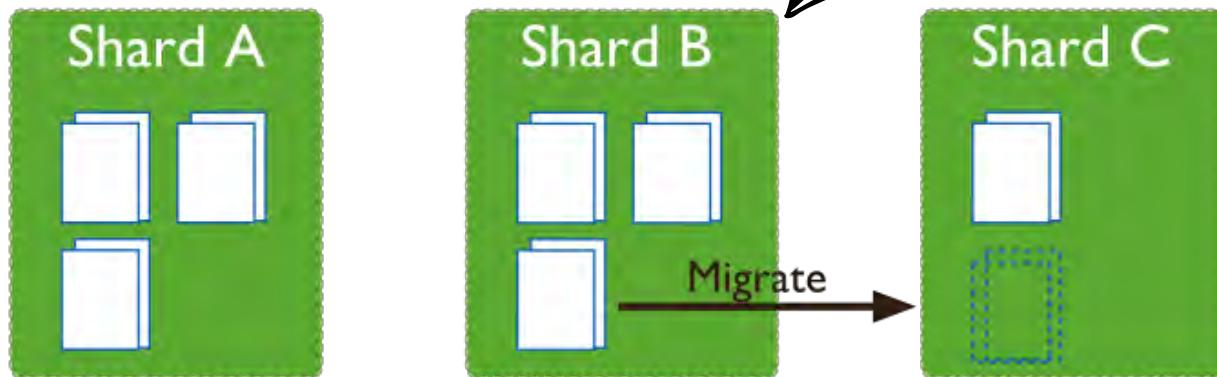


Sharding

- ▶ Splitting:



- ▶ Migration:



Classification: MongoDB

Techniques

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

Other Systems

Graph databases

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

Other Systems

Key-Value Stores

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

Other Systems

Document Stores

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

Other Systems

Wide-Column Stores

- ▶ **Accumulo** (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

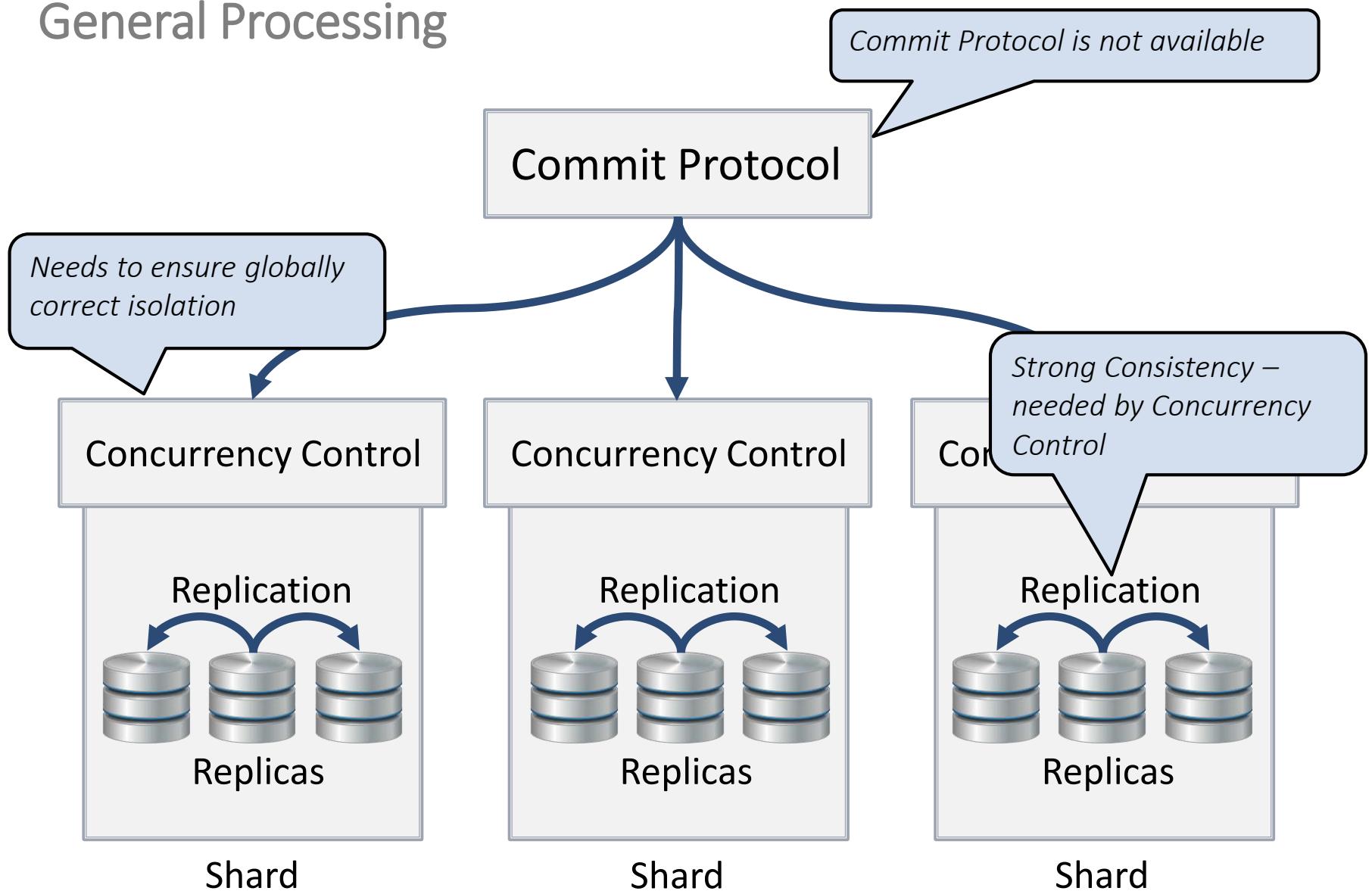
Durability

Isolation Levels:

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

Distributed Transactions

General Processing



Distributed Transactions

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

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

URL

Root Table

Child Table

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.

LOCAL COMMIT PROTOCOL,
optimistic concurrency
control

Distributed Transactions

MDCC – Multi Datacenter Concurrency Control

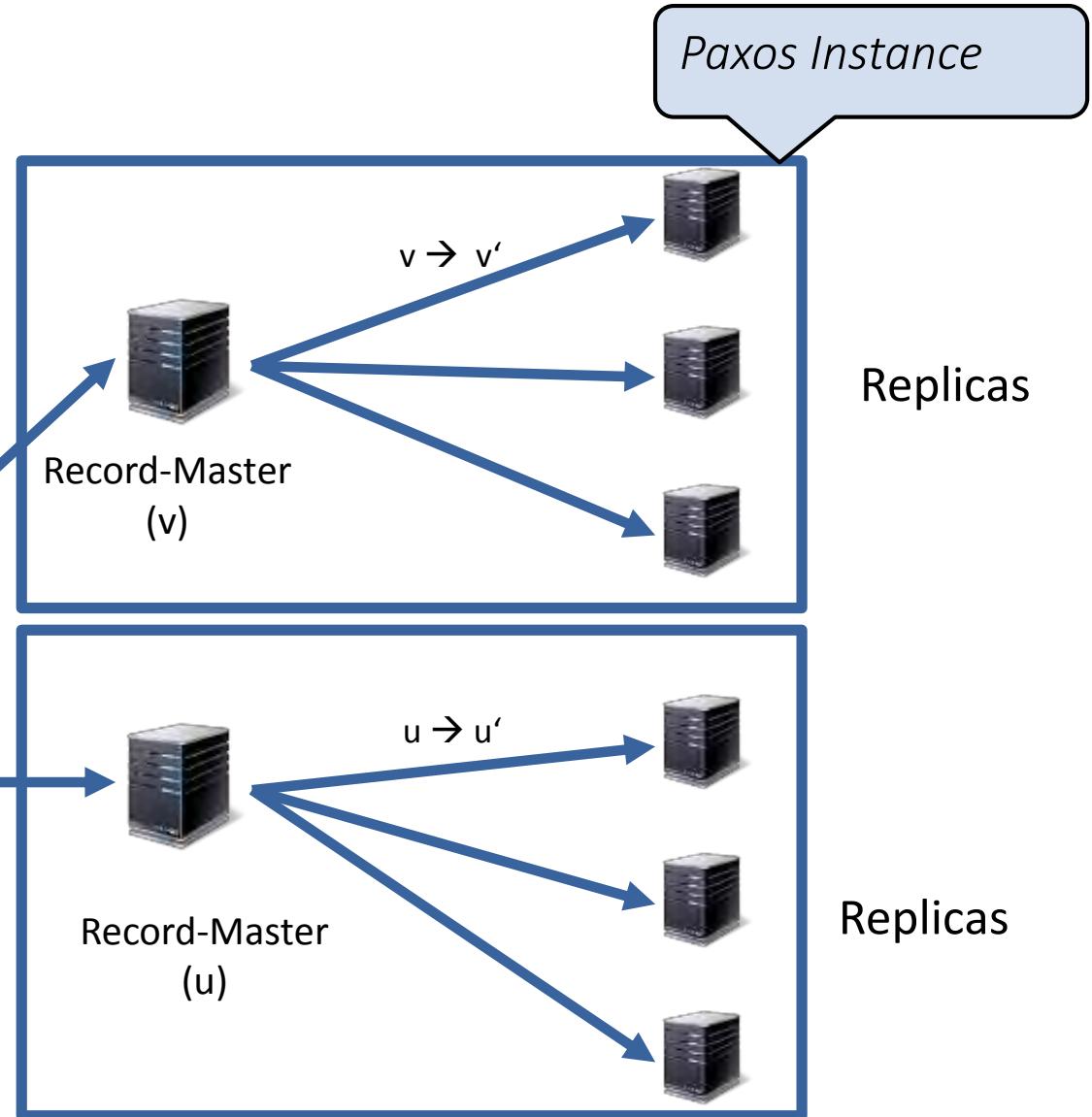
Properties:

- Read Committed Isolation
- Geo Replication
- Optimistic Commit

$$T1 = \{v \rightarrow v', u \rightarrow u'\}$$



App-Server
(Coordinator)



Distributed Transactions

RAMP – Read Atomic Multi Partition Transactions

Properties:

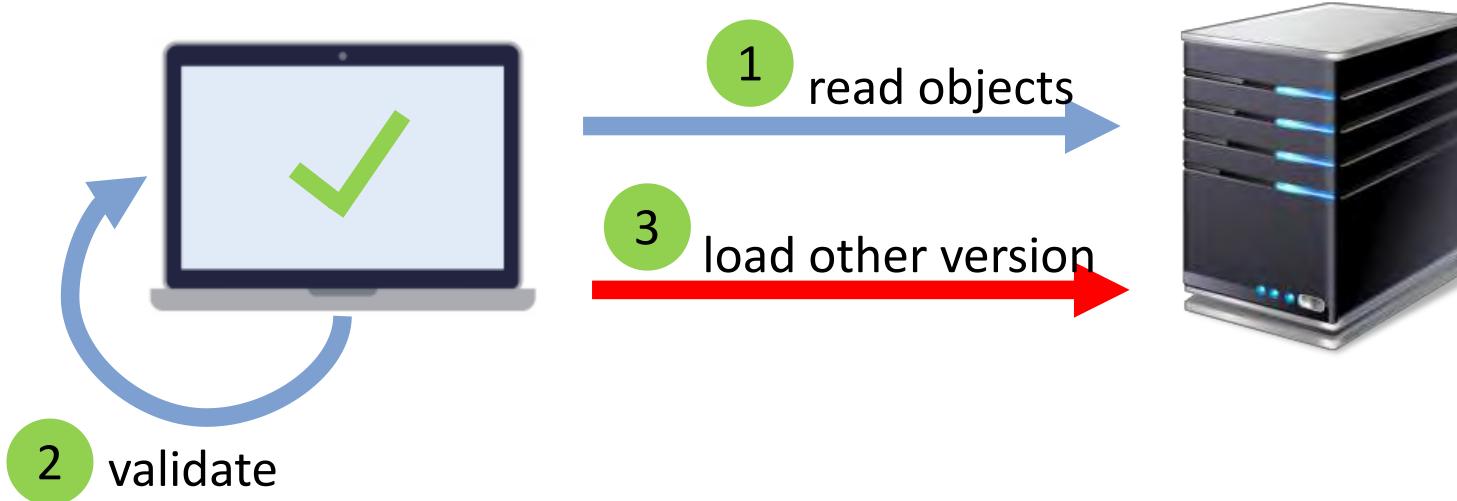
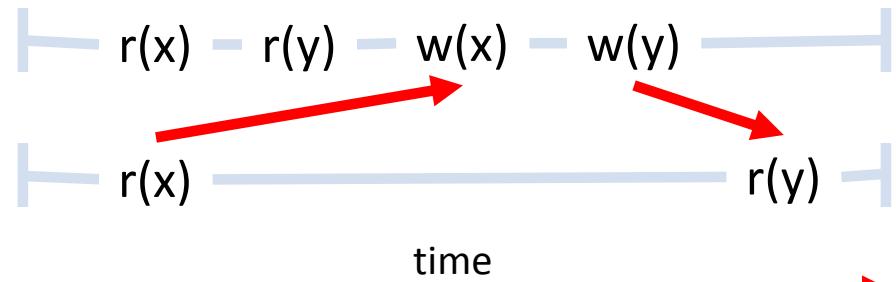
 Read Atomic Isolation

 Synchronization Independence

 Partition Independence

 Guaranteed Commit

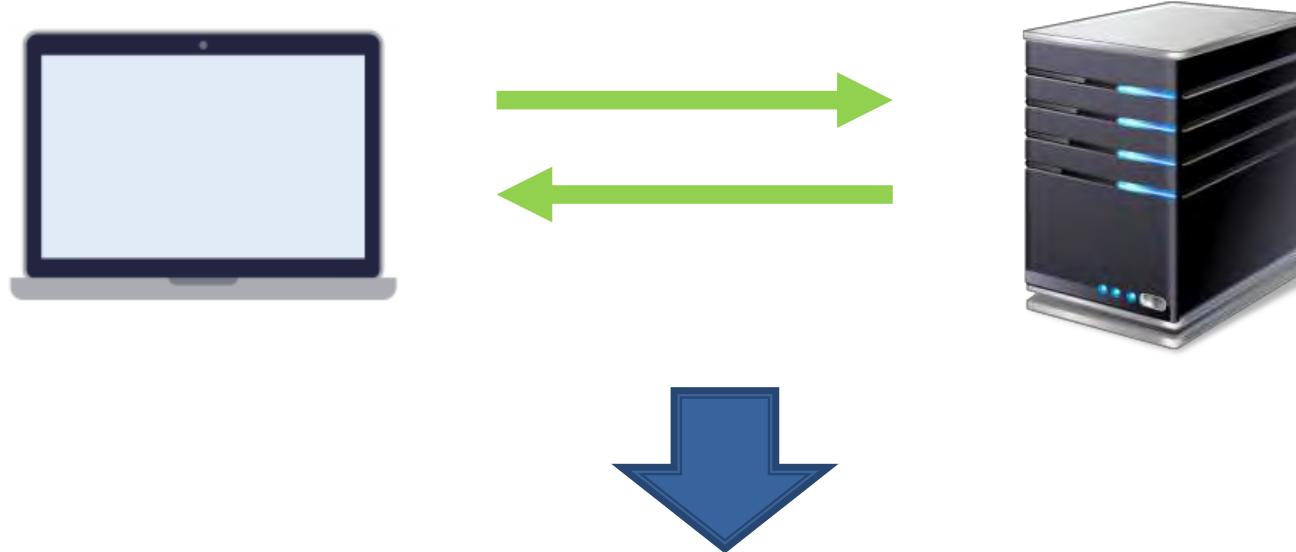
Fractured Read



Distributed Transactions in the Cloud

The Latency Problem

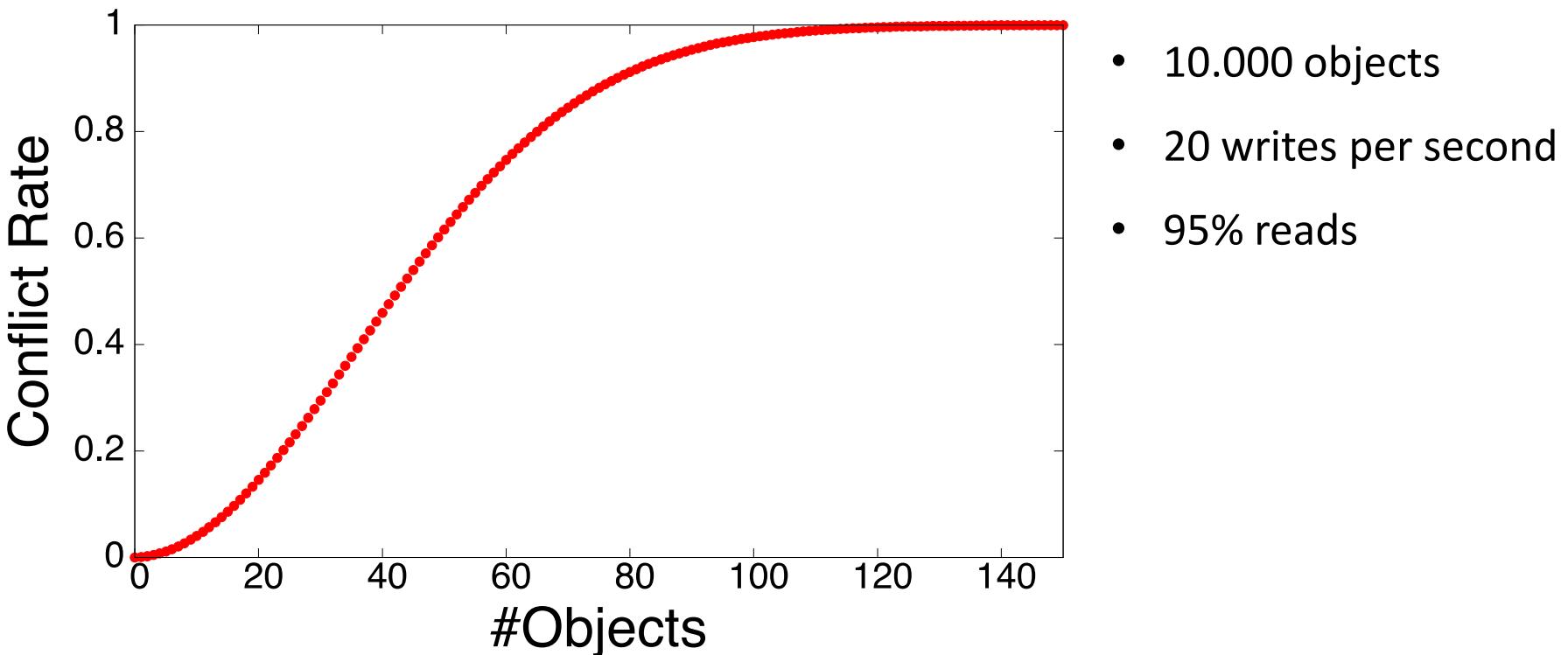
Interactive Transactions:



Optimistic Concurrency Control

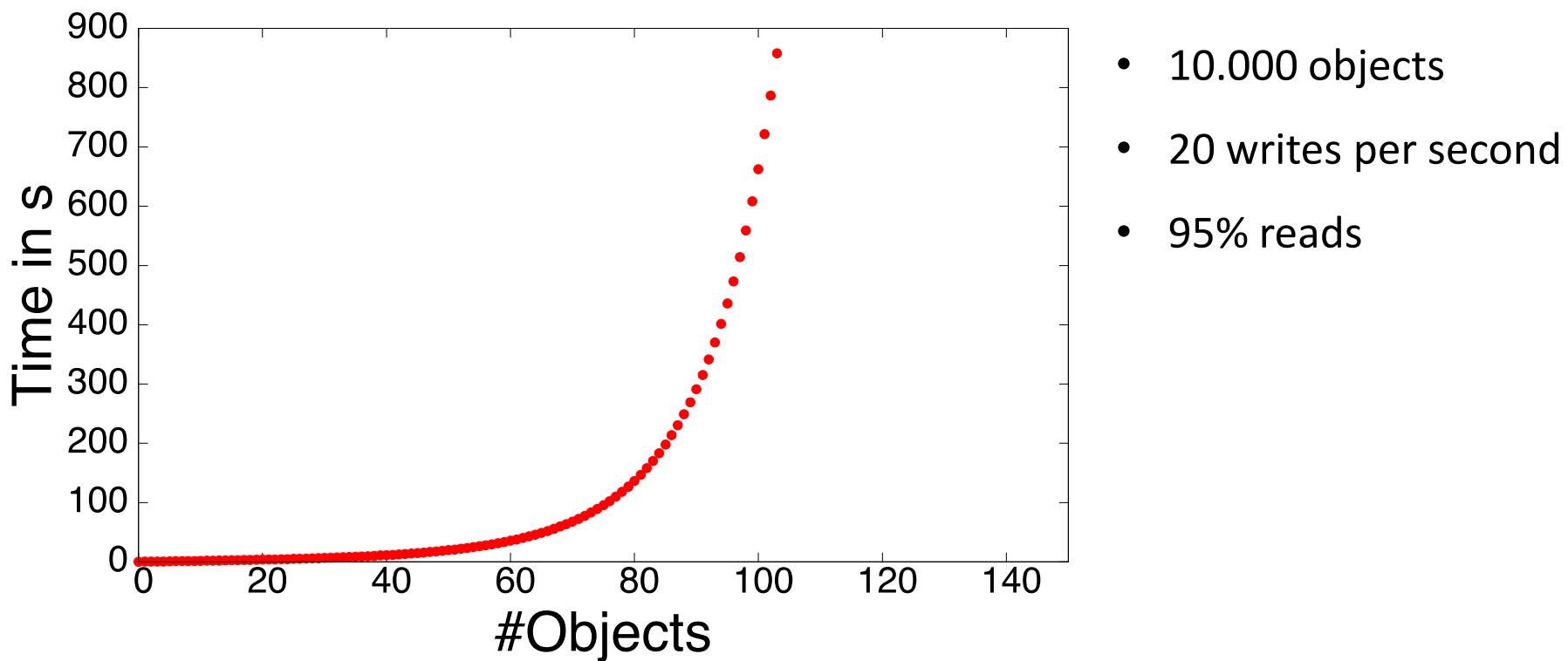
Optimistic Concurrency Control

The Abort Rate Problem



Optimistic Concurrency Control

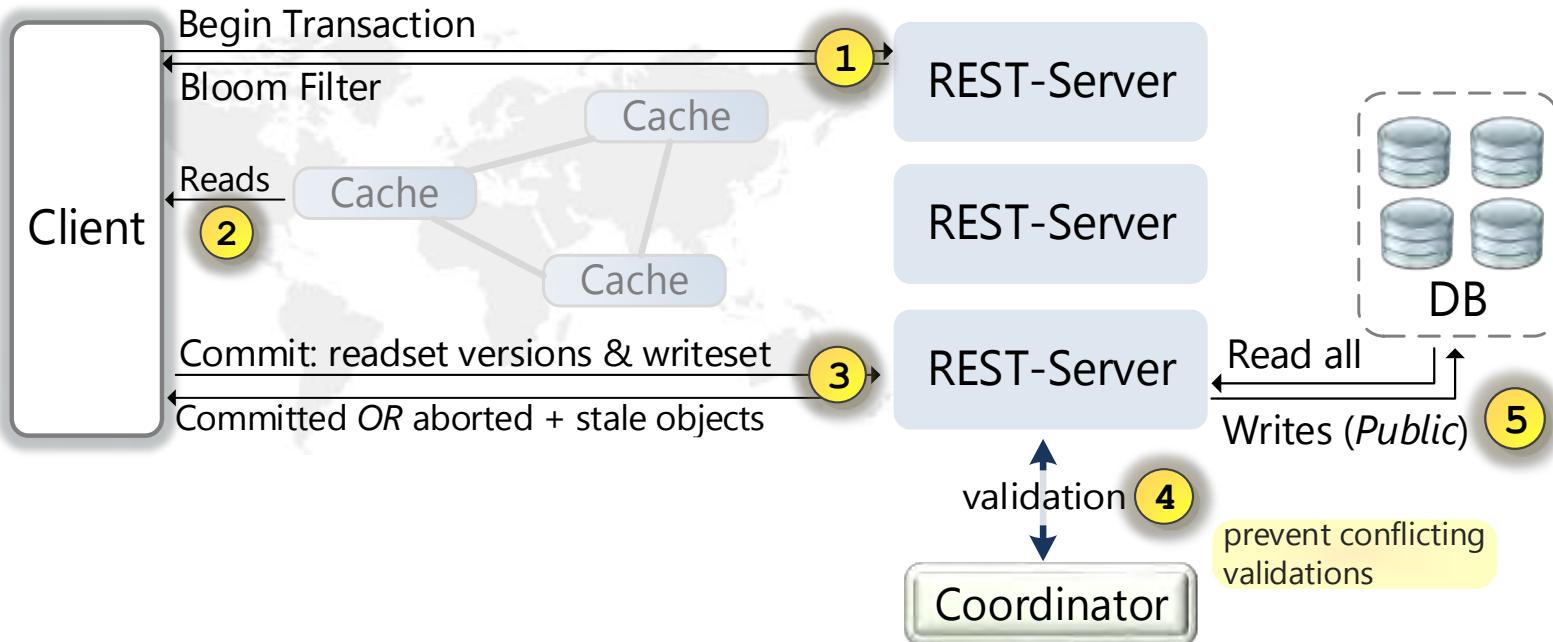
The Abort Rate Problem



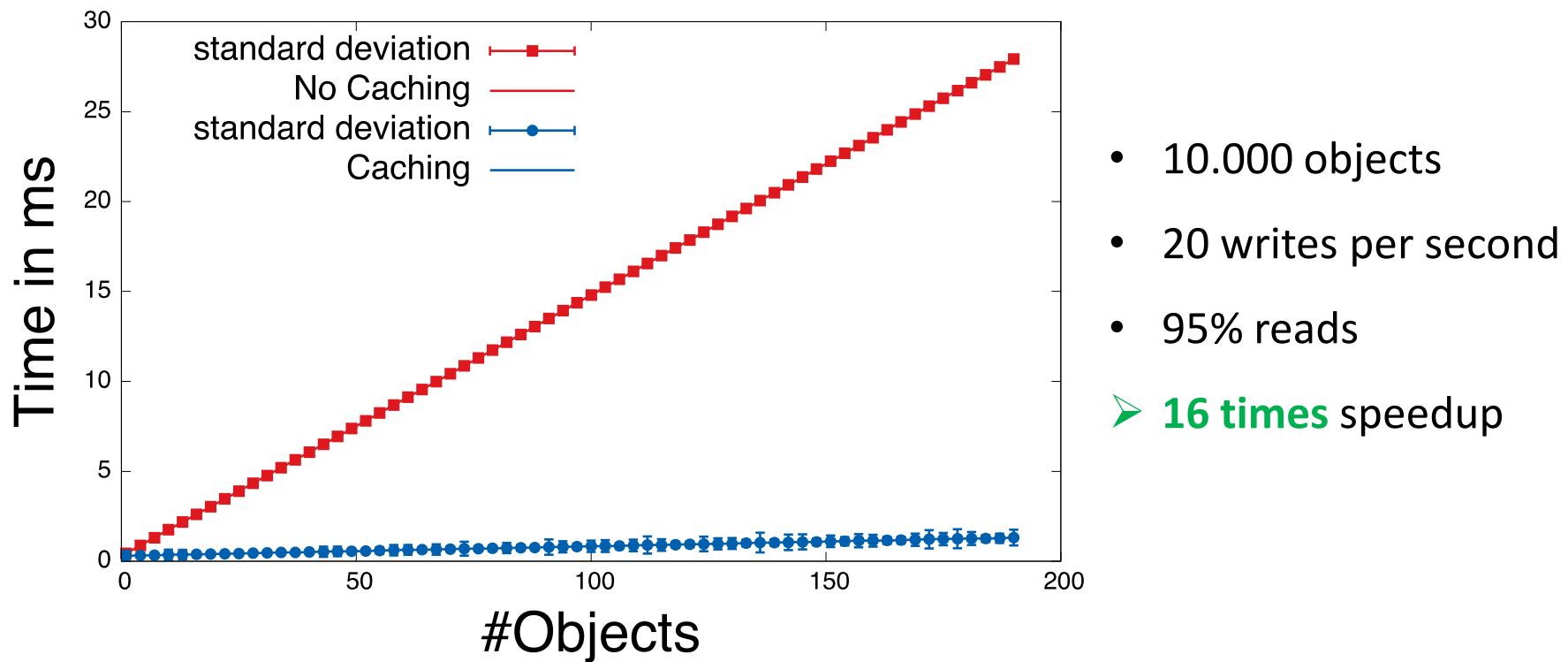
Distributed Cache-Aware Transaction

Scalable ACID Transactions

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

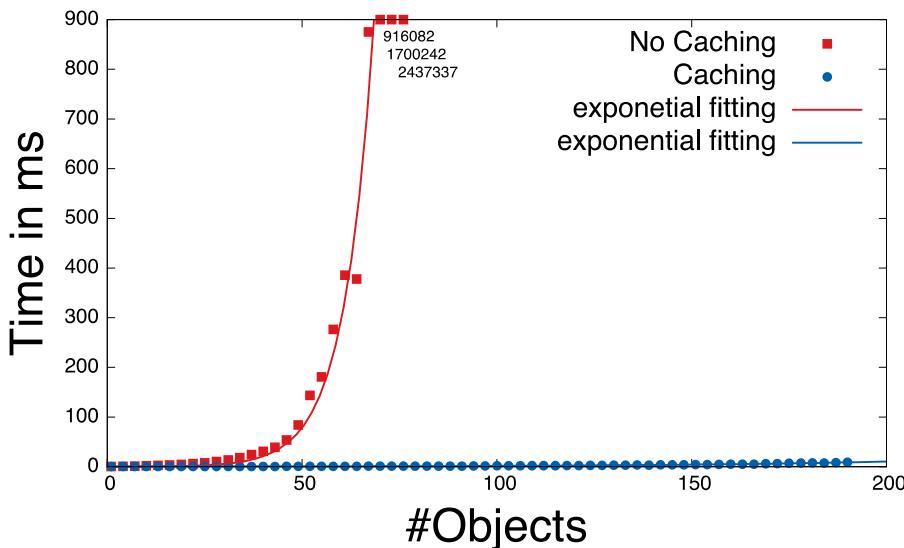
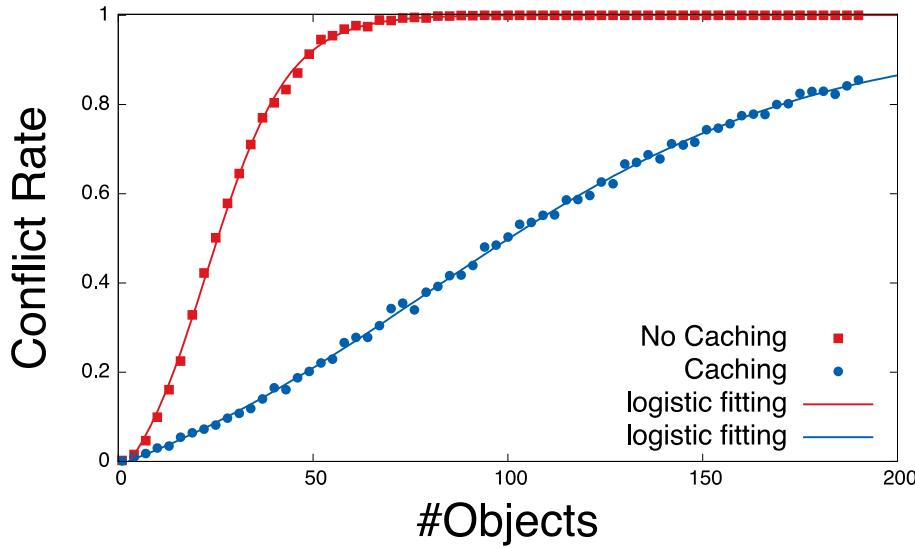


Distributed Cache-Aware Transaction Speed Evaluation



Distributed Cache-Aware Transaction

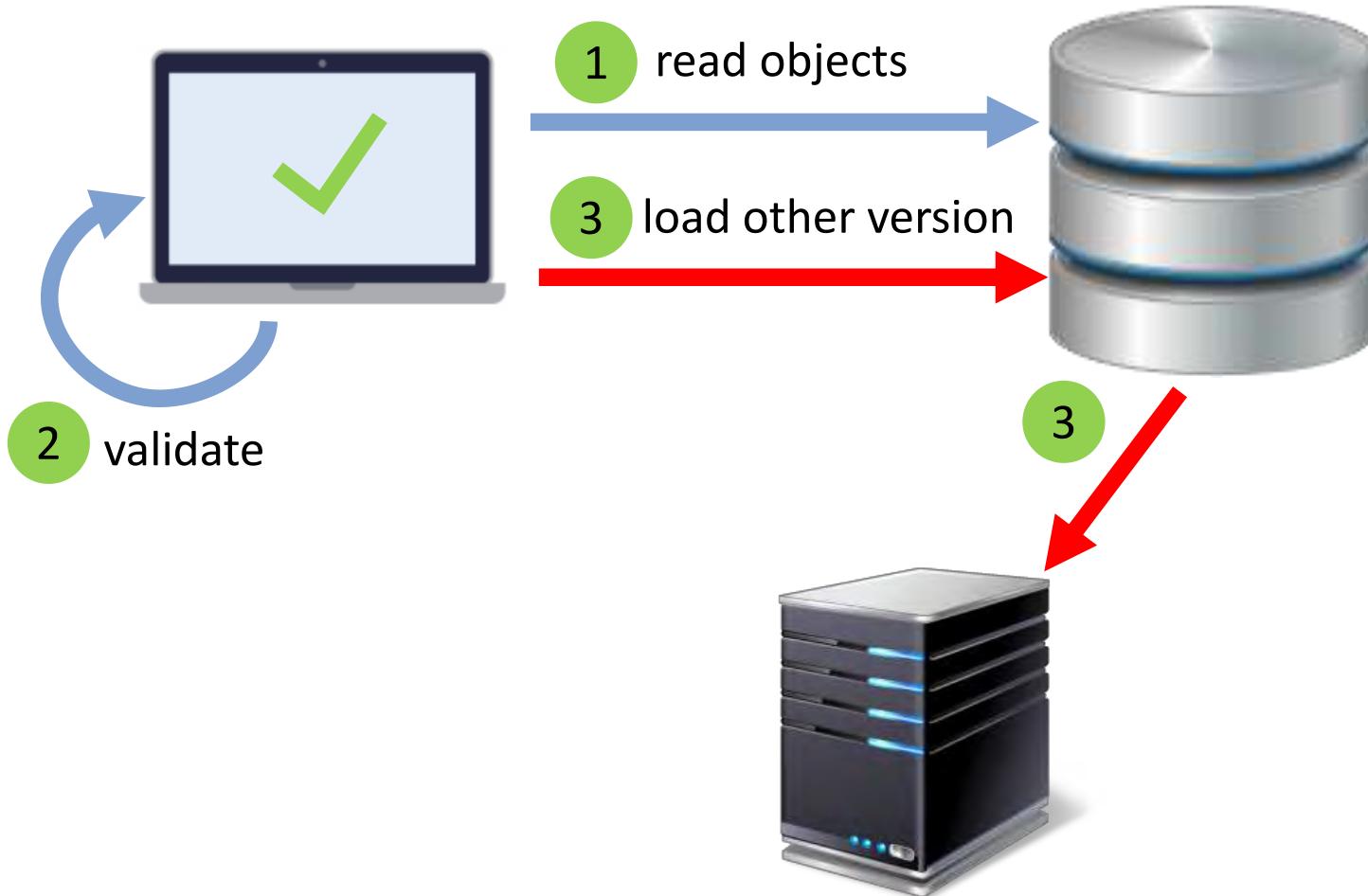
Abort Rate Evaluation



- 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 Challenges

Encrypted Databases

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

SQL-Proxy

Encrypts and decrypts,

Relational Cloud

DBaaS Architecture:

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



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



- Early approach
- Not adopted in practice, yet



Dream solution:

Full Homomorphic Encryption

Research Challenges

Transactions and Scalable Consistency

Google's F1

Consistent

mit

Data



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.

Dynamo

Eventual

Yahoo PNuts

Timeline p

COPS

Causality

MySQL (async)

Serializable

Megastore

Serializable

Spanner

Snapshot Isolation

MDCC

Partition



Currently very few NoSQL DBs implement consistent Multi-DC replication

Selected Research Challenges

NoSQL Benchmarking

- ▶ YCSB (Yahoo Cloud Serving Benchmark)

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

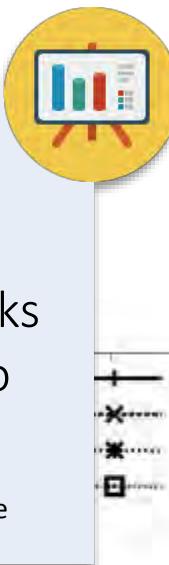
Selected Research Challenges

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

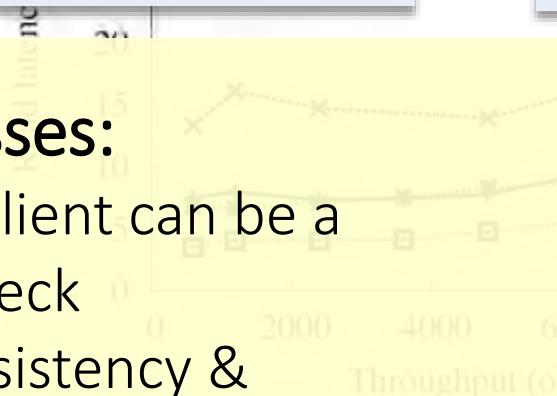
 A. Dey et al. “YCSB+T: Benchmarking Web-Scale Transactional Databases”, CloudDB 2014

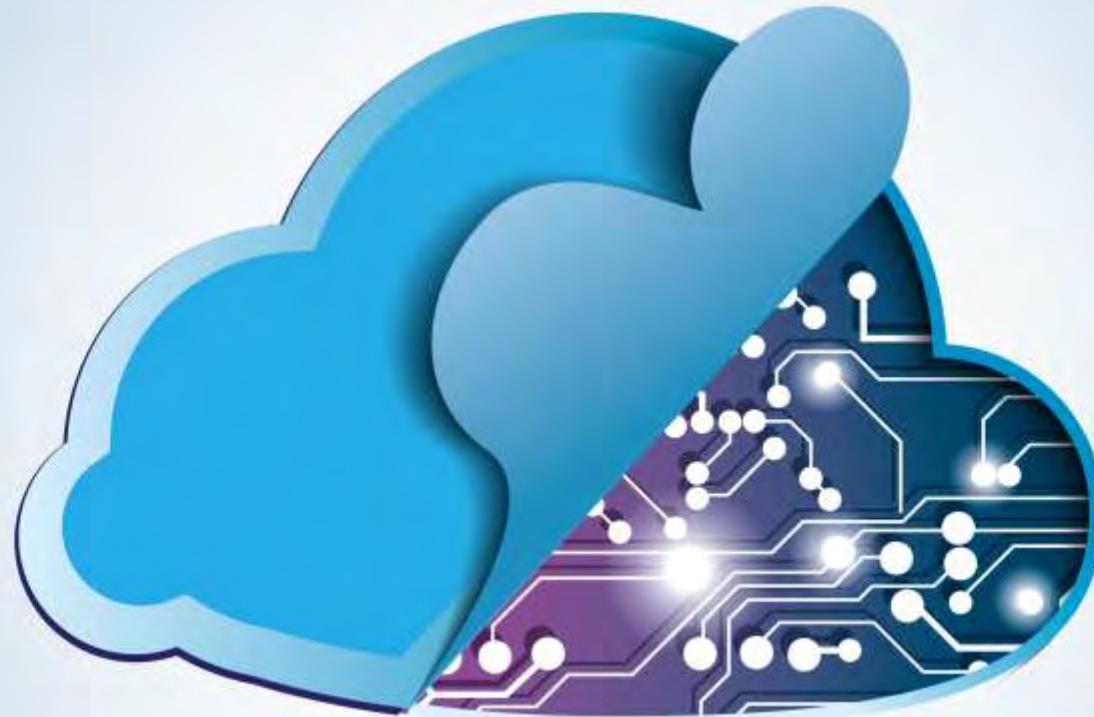


Weaknesses:

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

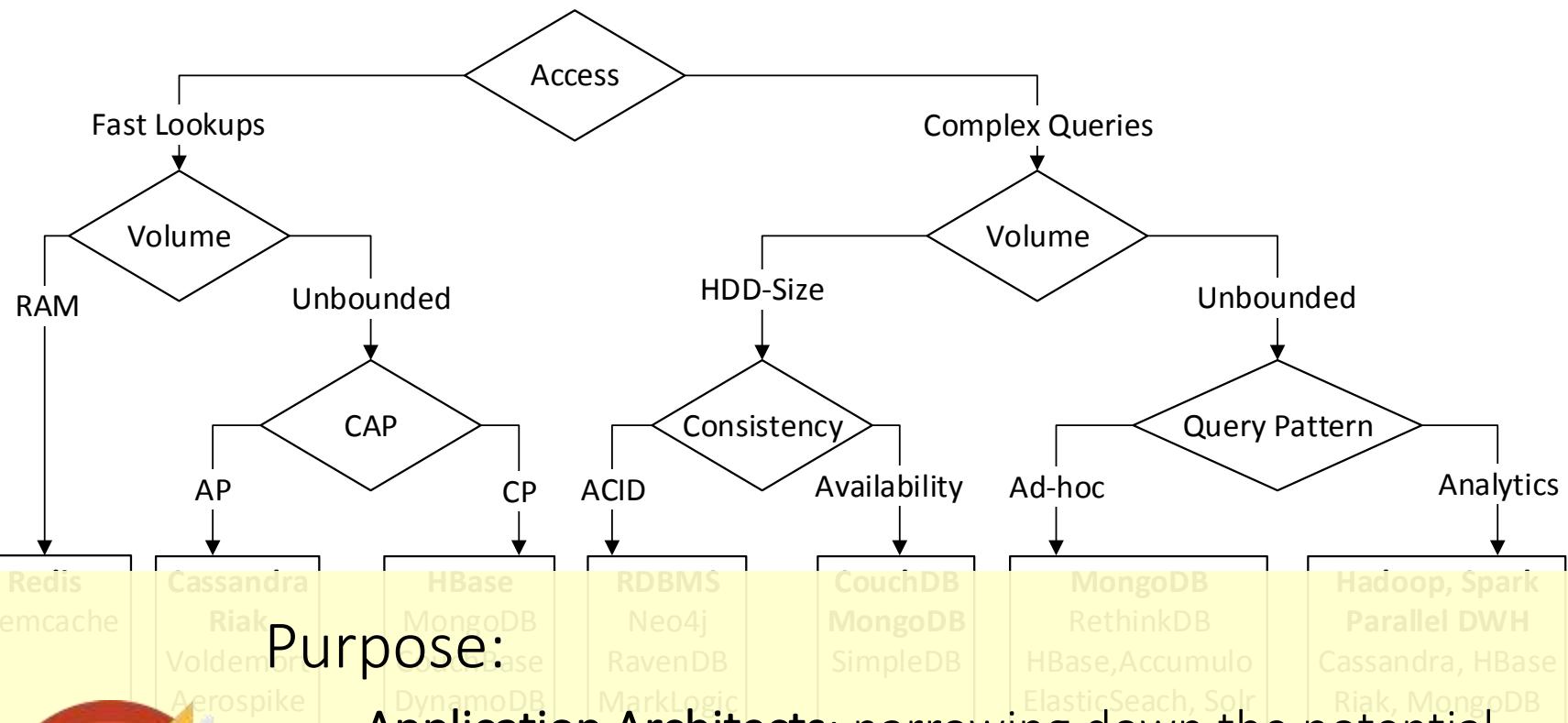
- No Transaction Support
No specific application
→ CloudStone, CARE, TPC extensions?





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

NoSQL Decision Tree



Purpose:

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

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

System Properties

According to the NoSQL Toolbox

- ▶ For fine-grained system selection:

Functional Requirements									
	Scan Queries		ACID Transactions		Conditional Writes		Joins		Analytics
	Support	Complexity	Support	Complexity	Support	Complexity	Support	Complexity	
Mongo	x			x			x		x
Redis	x		x	x					
HBase	x			x			x		x
Riak								x	x
Cassandra	x			x			x		x
MySQL	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	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	x	x	x	x									
Redis						x	x	x	x	x	x	x																
HBase	x				x	x	x	x	x	x	x	x	x	x	x													
Riak	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	x									
Cassandra	x	x				x	x	x	x	x	x	x	x	x	x	x	x	x	x									
MySQL			x			x	x	x	x	x	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

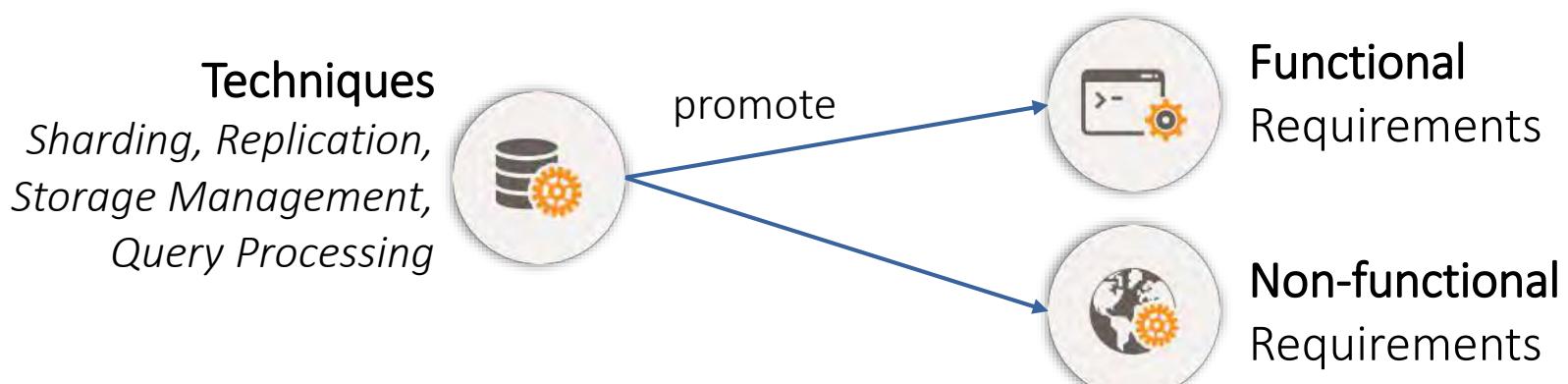


mongoDB

Summary



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

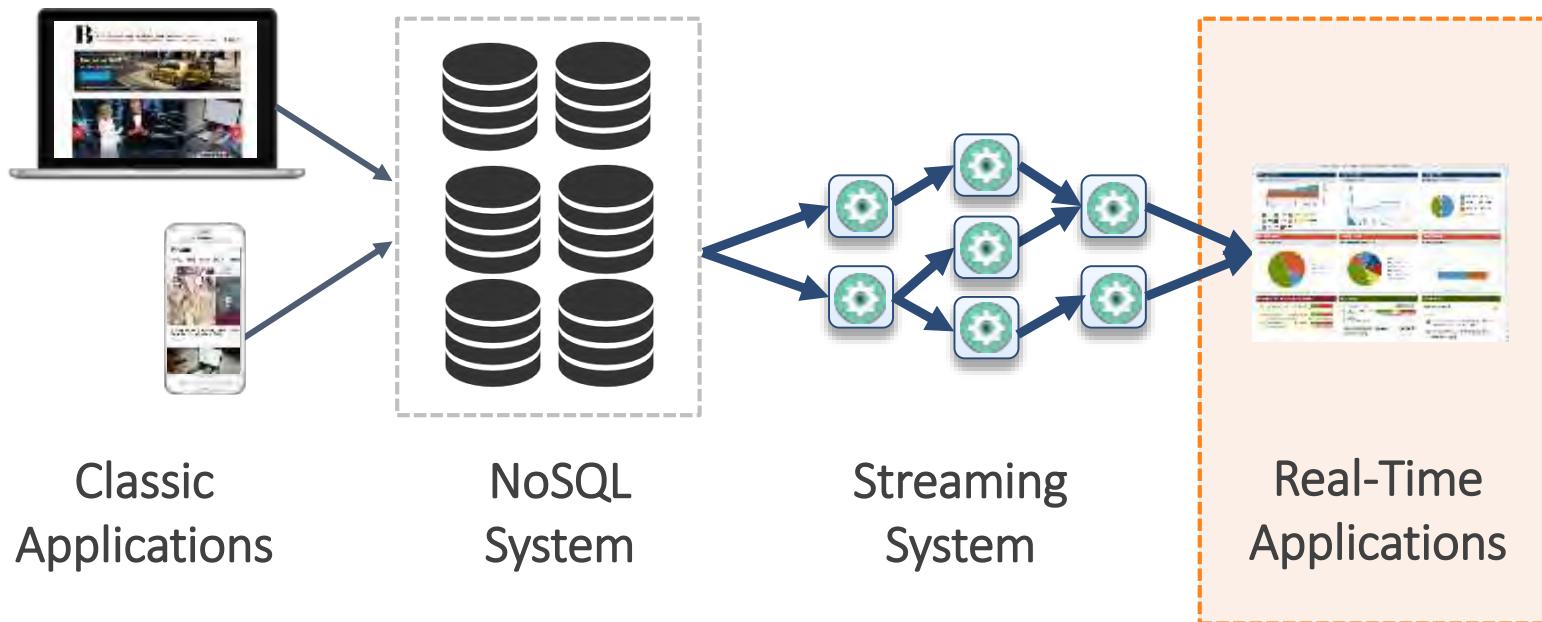


- ▶ Decision Tree

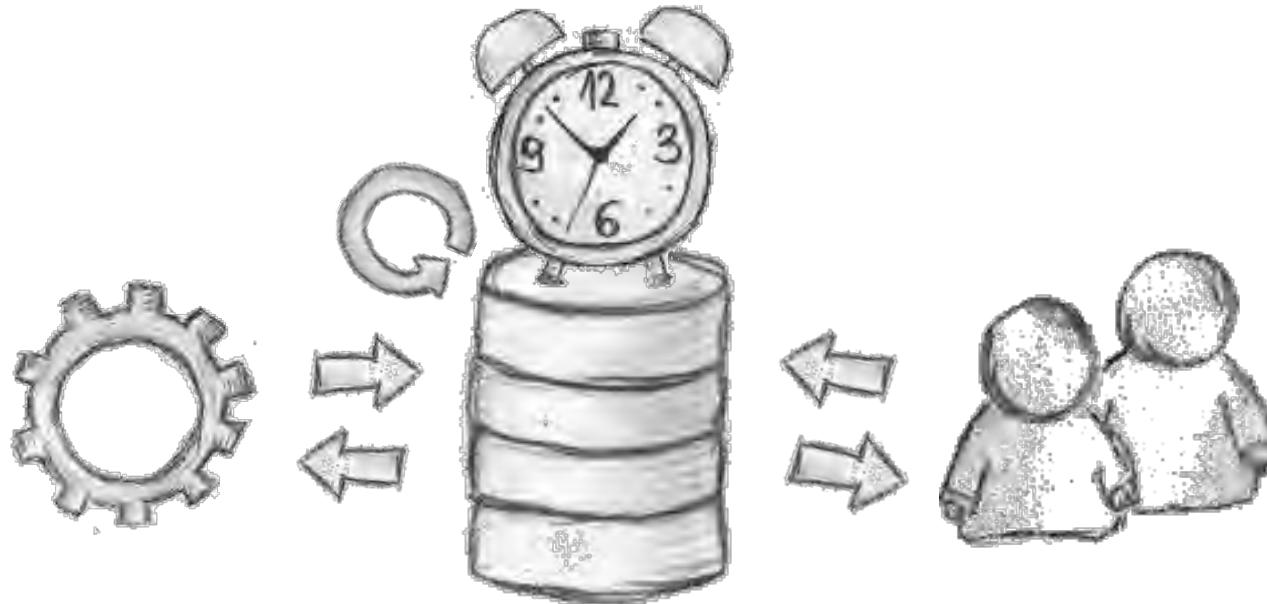
Summary



- ▶ 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



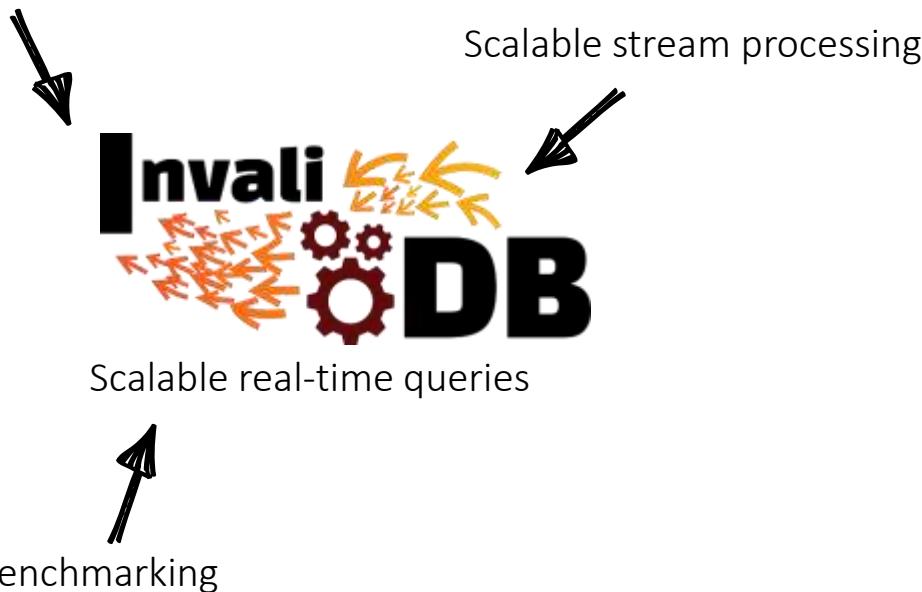
Wolfram Wingerath
wingerath@informatik.uni-hamburg.de
March 7th, 2017, Stuttgart

About me

Wolfram Wingerath

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

NoSQL database systems



Outline



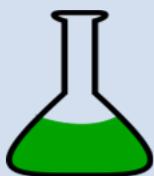
Scalable Data Processing:
Big Data in Motion



Stream Processors:
Side-by-Side Comparison

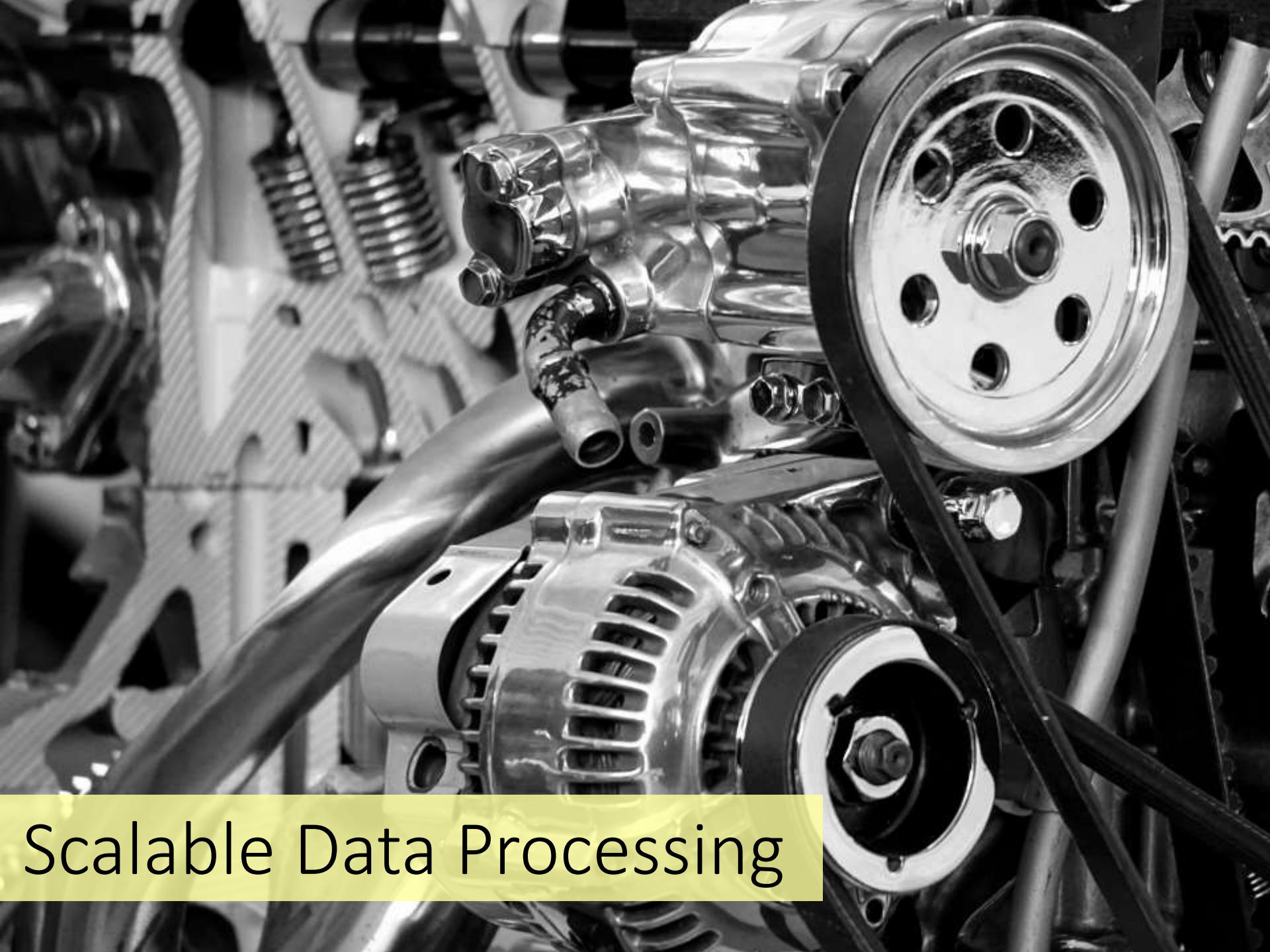


Real-Time Databases:
Push-Based Data Access



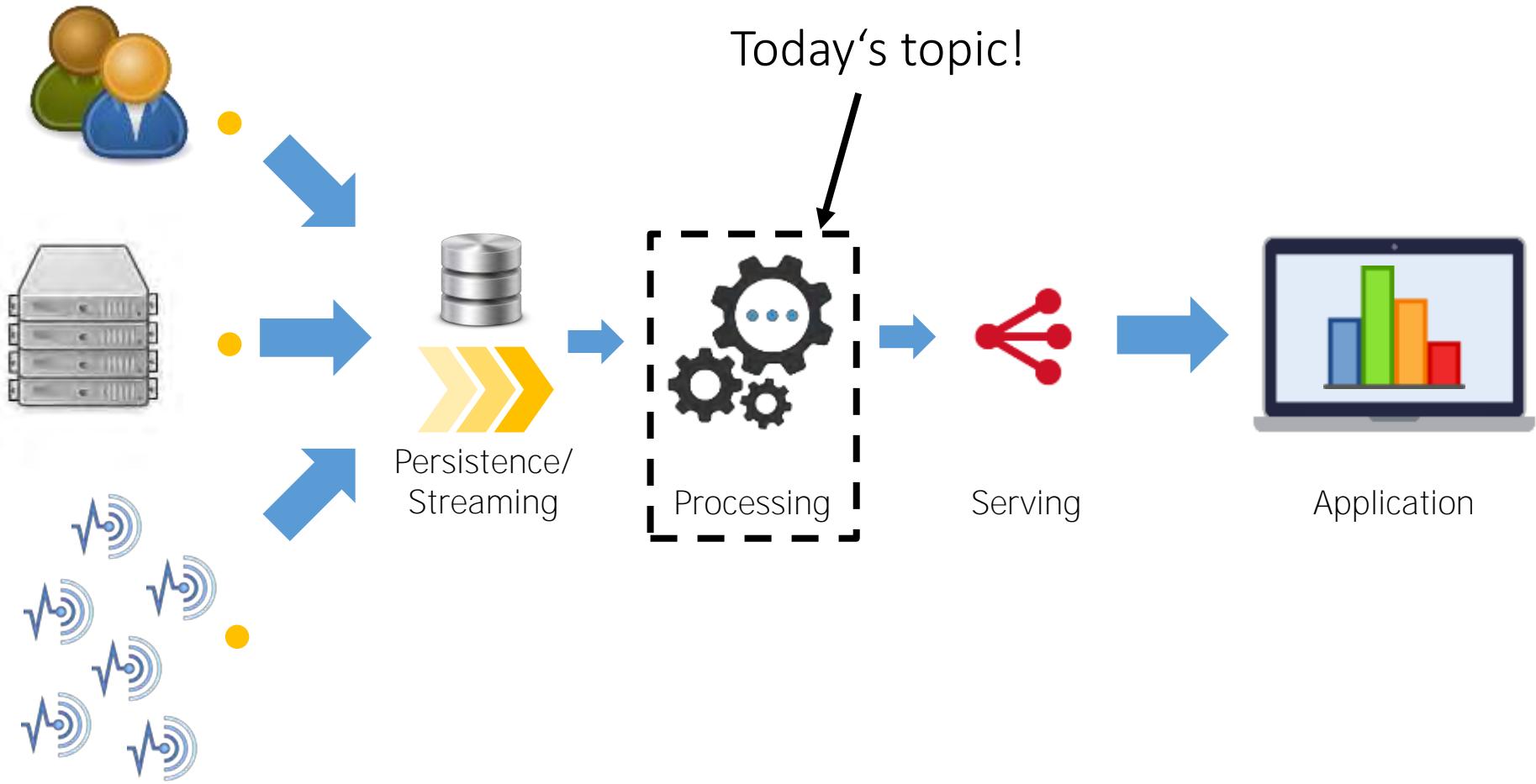
Current Research:
Opt-In Push-Based Access

- 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.:

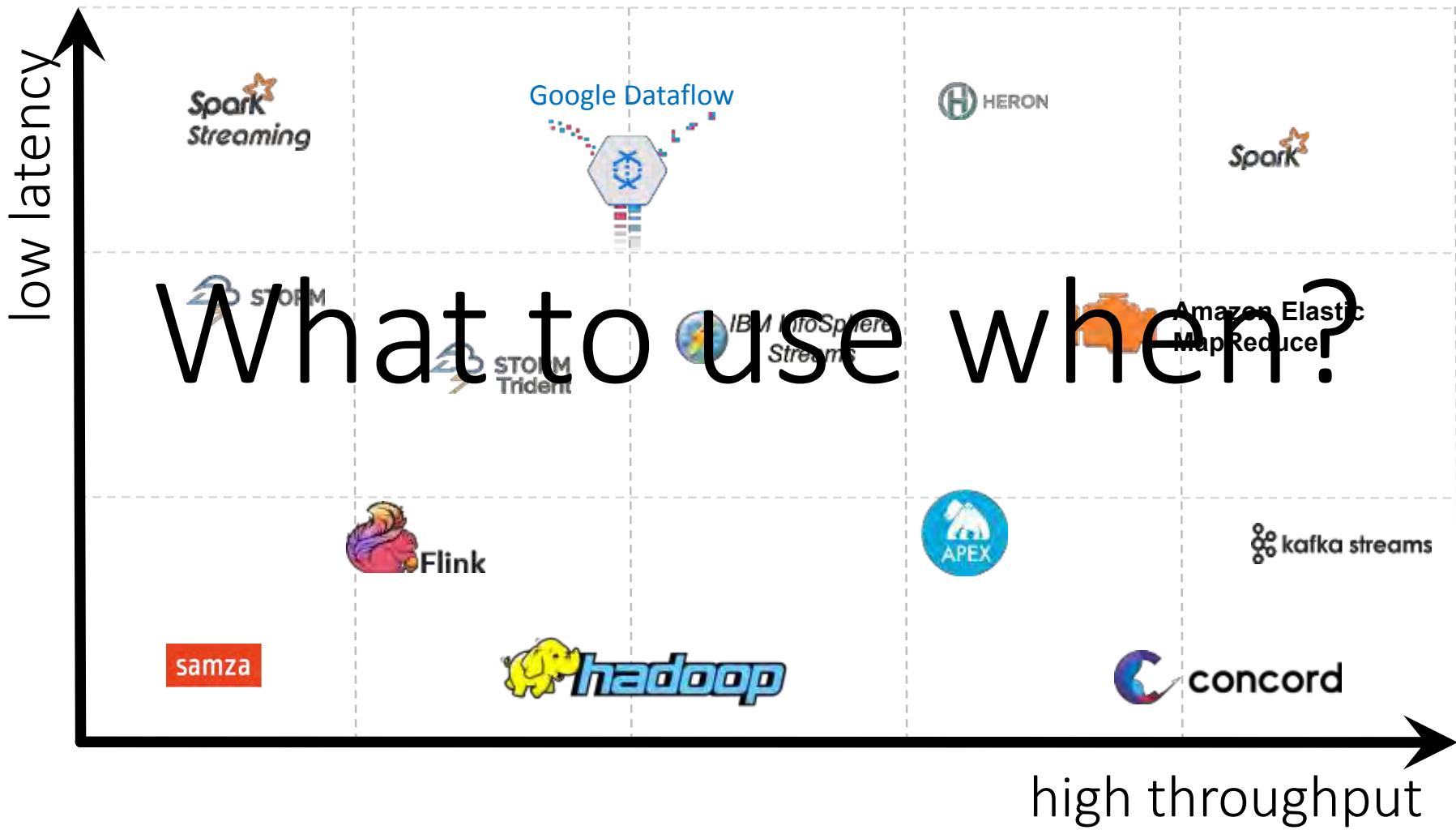
- **Deployment**: code distribution, starting/stopping work
- **Monitoring**: health checks, application stats
- **Scheduling**: assigning work to machines, rebalancing
- **Fault-tolerance**: restarting failed workers, rescheduling failed work

Running on single-node



Big Data Processing Frameworks

What are your options?



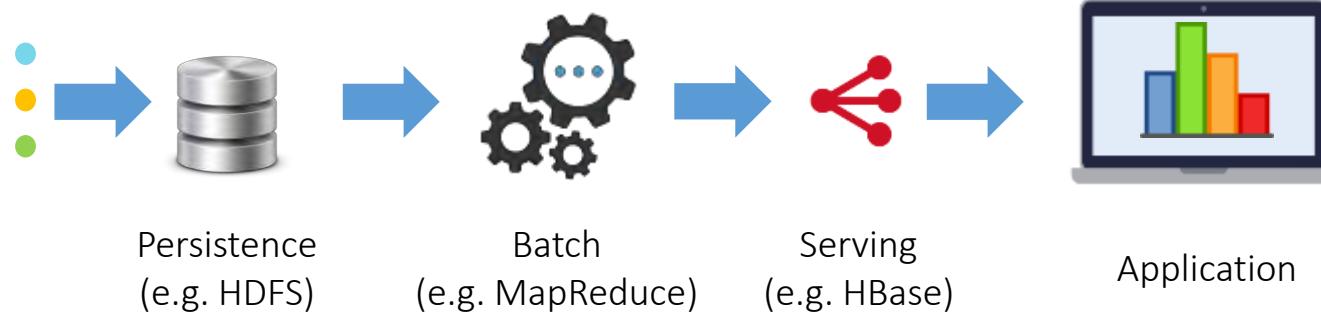
Batch Processing

„Volume“

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

But:

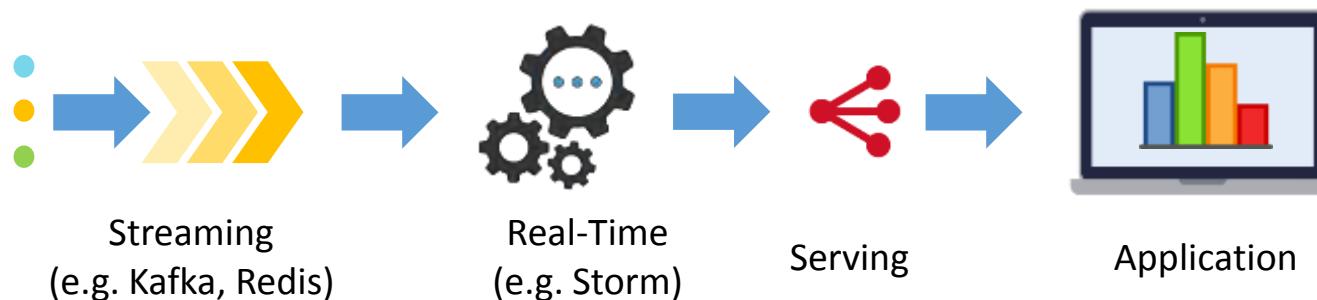
- **High latency**: jobs periodically (e.g. during night times)



Stream Processing

„Velocity“

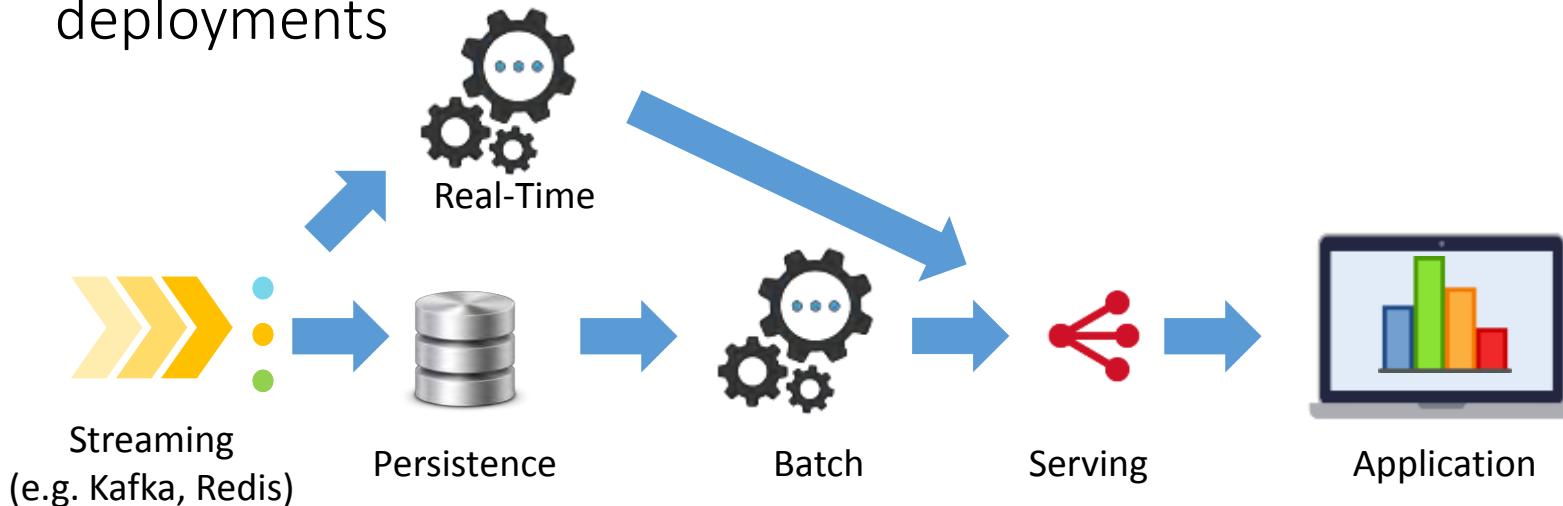
- Low end-to-end latency
- Challenges:
 - Long-running jobs: no downtime allowed
 - Asynchronism: data may arrive delayed or out-of-order
 - Incomplete input: algorithms operate on partial data
 - More: fault-tolerance, state management, guarantees, ...



Lambda Architecture

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

- **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



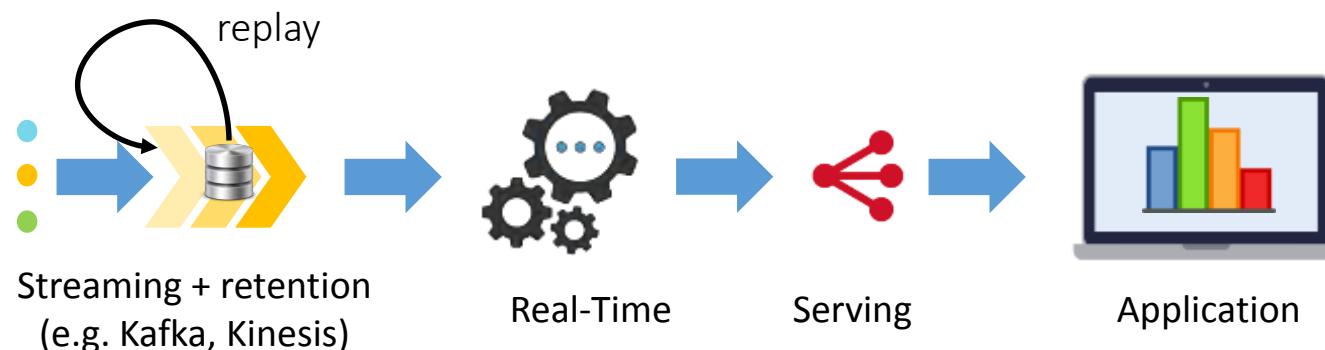
Nathan Marz, *How to beat the CAP theorem* (2011)

<http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html>

Kappa Architecture

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

- 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



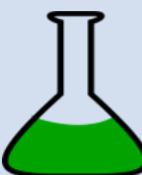
Scalable Data Processing:
Big Data in Motion



Stream Processors:
Side-by-Side Comparison



Real-Time Databases:
Push-Based Data Access



Current Research:
Opt-In Push-Based Access

- Processing Models:
Stream \leftrightarrow 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



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:

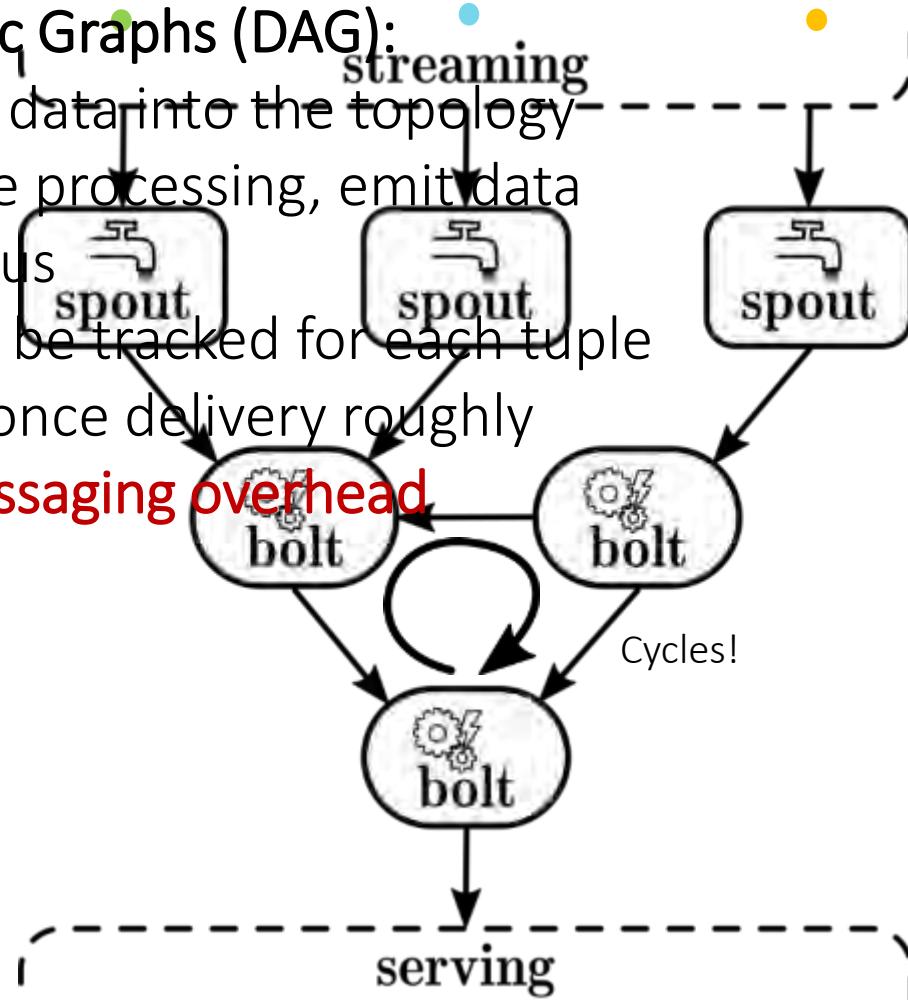
- 2010: start of development at BackType (acquired by twitter)
- 2011: open-sourced
- 2014: Apache top-level project

Dataflow

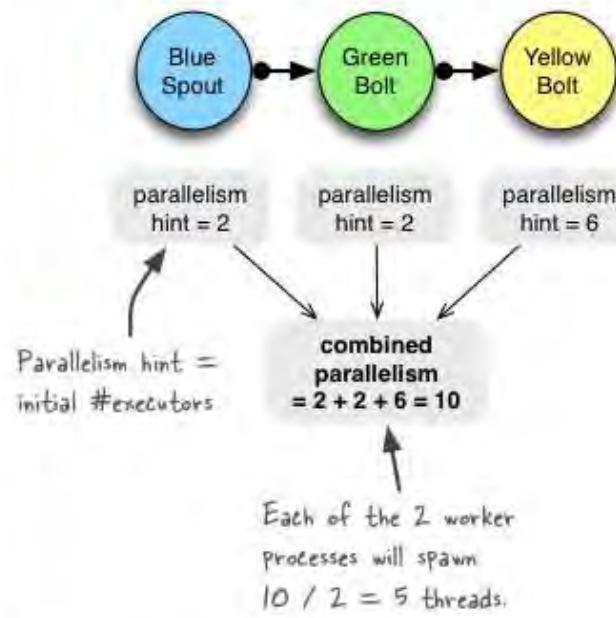
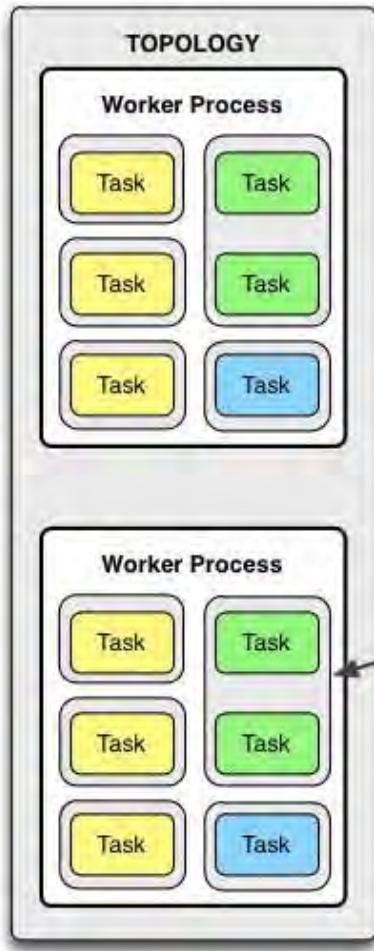


Directed Acyclic Graphs (DAG):

- 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



The green bolt was configured to use two executors and four tasks. For this reason each executor runs two tasks for this bolt.



Illustration taken from:

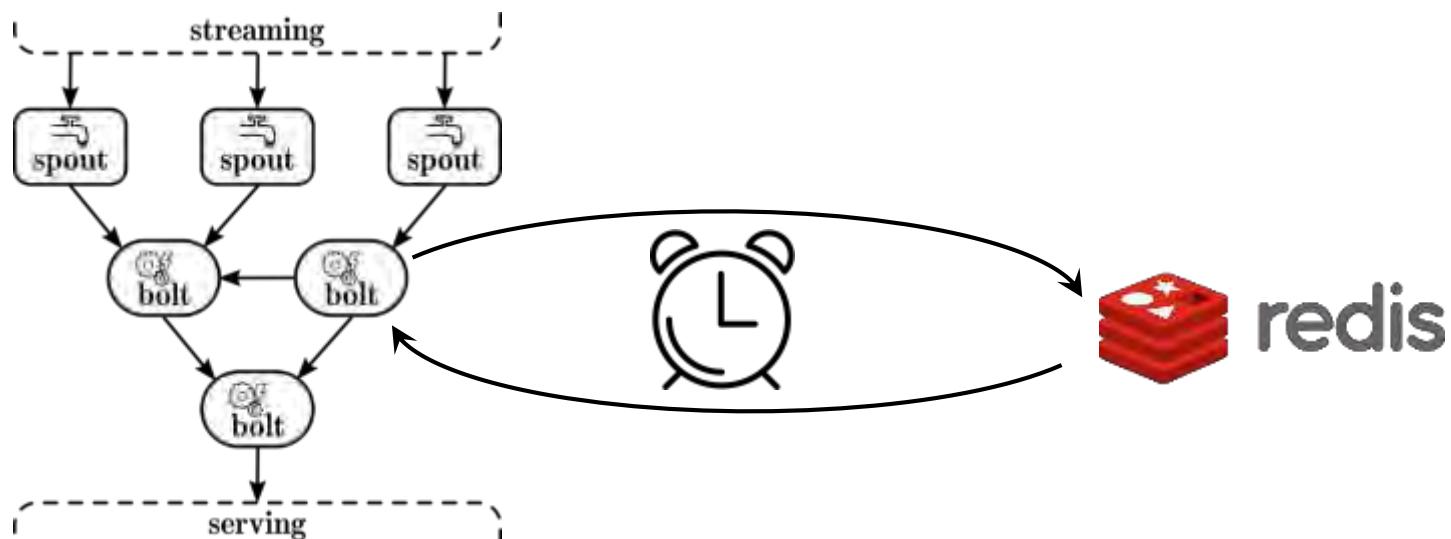
<http://storm.apache.org/releases/1.0.1/Understanding-the-parallelism-of-a-Storm-topology.html> (2017-02-19)

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

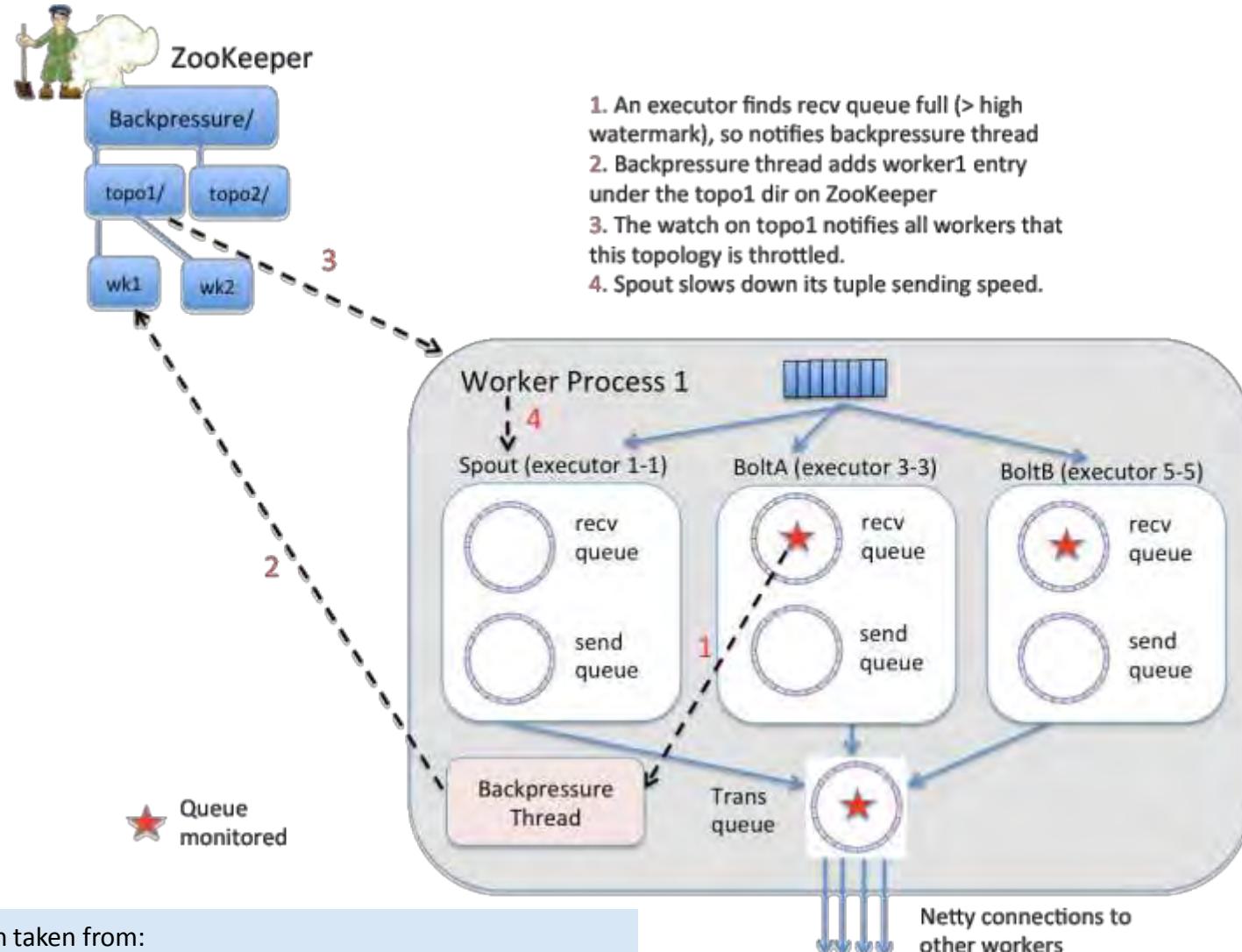
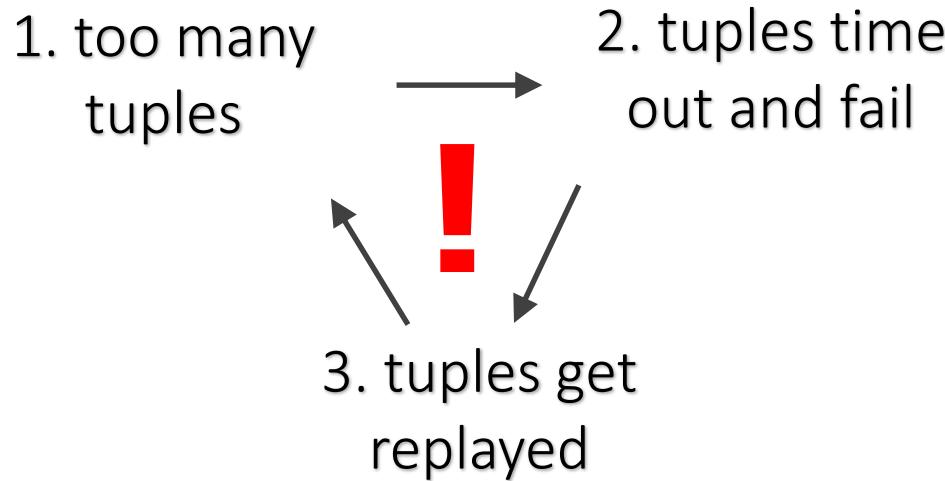


Illustration taken from:

<https://issues.apache.org/jira/browse/STORM-886> (2017-02-21)

Back Pressure

Throttling Ingestion on Overload



Approach: monitoring bolts' inbound buffer

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

Trident

Stateful Stream Joining on Storm



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

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

A diagram showing a spout component on the left with two arrows pointing to it from the text "Can block the topology" and "Does not scale:". The arrows point to the "Non-transactional" row and the "Opaque transactional" row respectively.

Does not scale:

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



Illustration taken from:

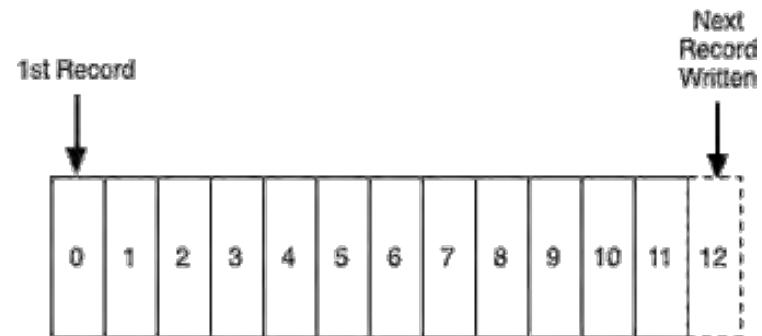
<http://storm.apache.org/releases/1.0.2/Trident-state.html> (2017-02-26)

Samza

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)

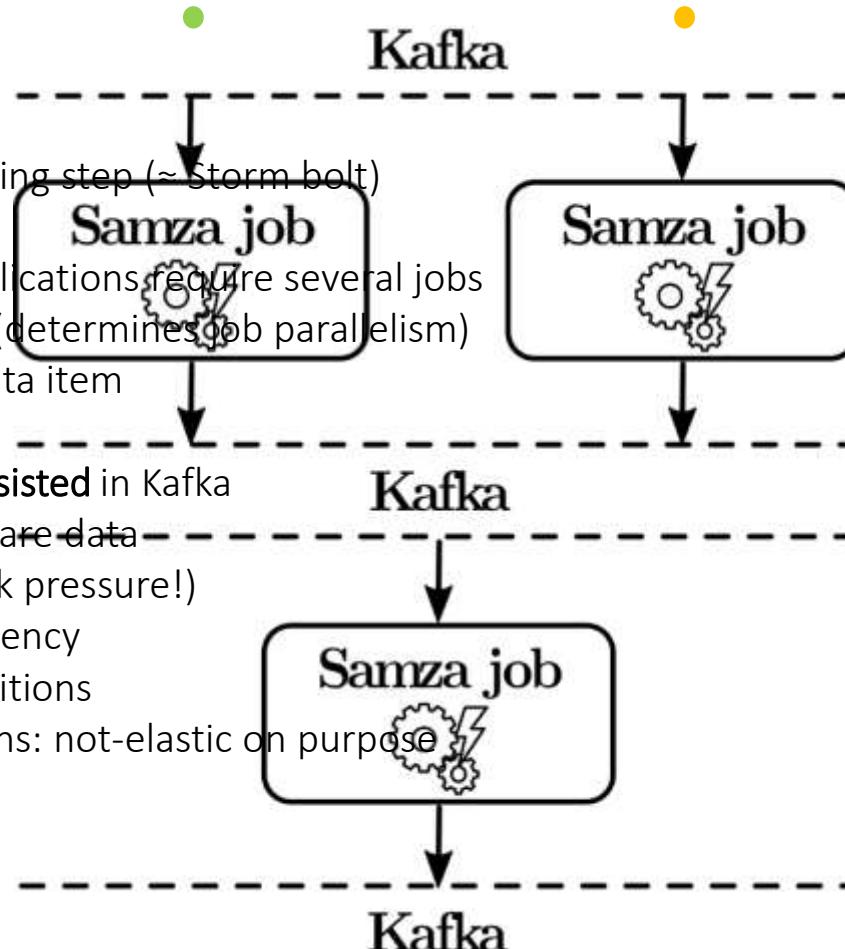
<https://www.oreilly.com/ideas/questioning-the-lambda-architecture> (2017-03-02)

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



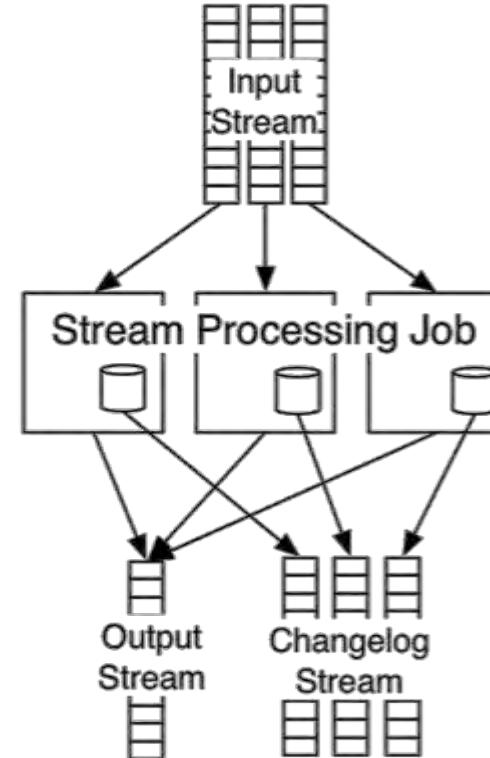
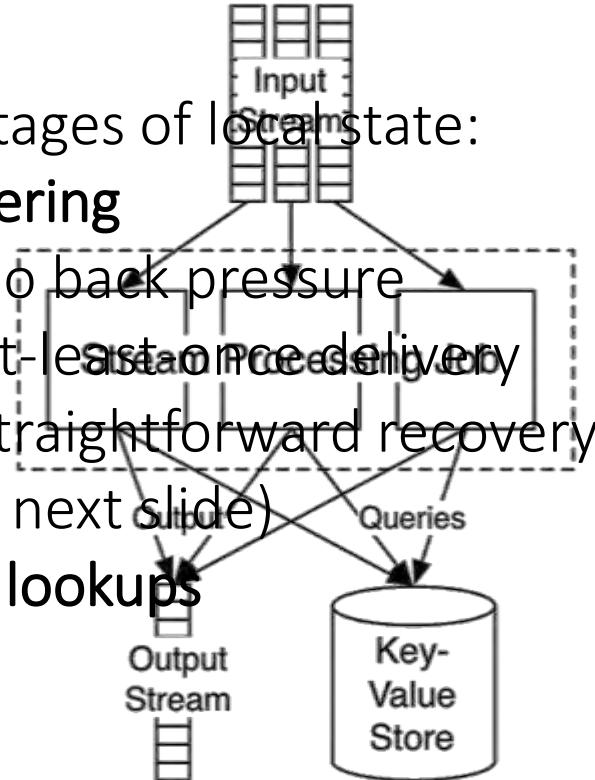
Samza

Local State

Samza

Advantages of local state:

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



Remote State vs. Local State



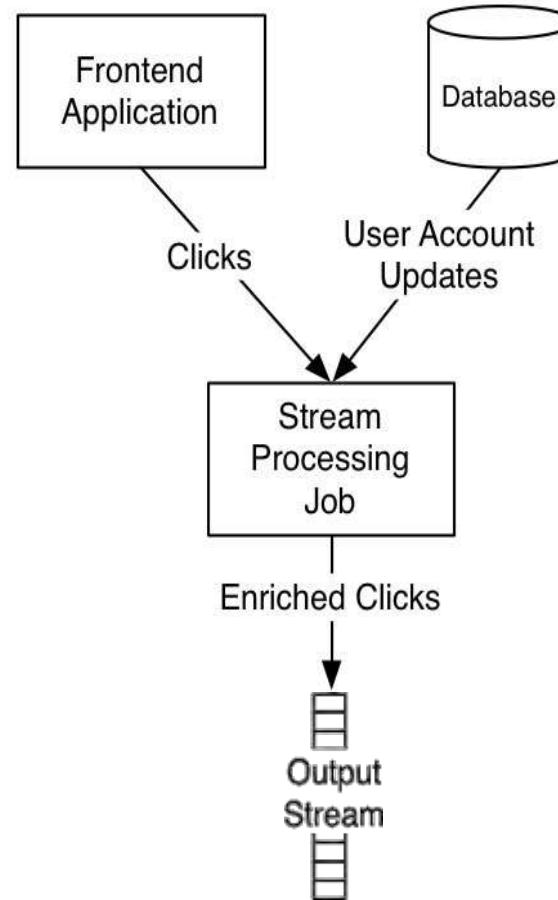
Illustrations taken from: Jay Kreps, *Why local state is a fundamental primitive in stream processing* (2014)

<https://www.oreilly.com/ideas/why-local-state-is-a-fundamental-primitive-in-stream-processing> (2017-02-26)

Dataflow

Example: Enriching a Clickstream

Samza



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

Samza

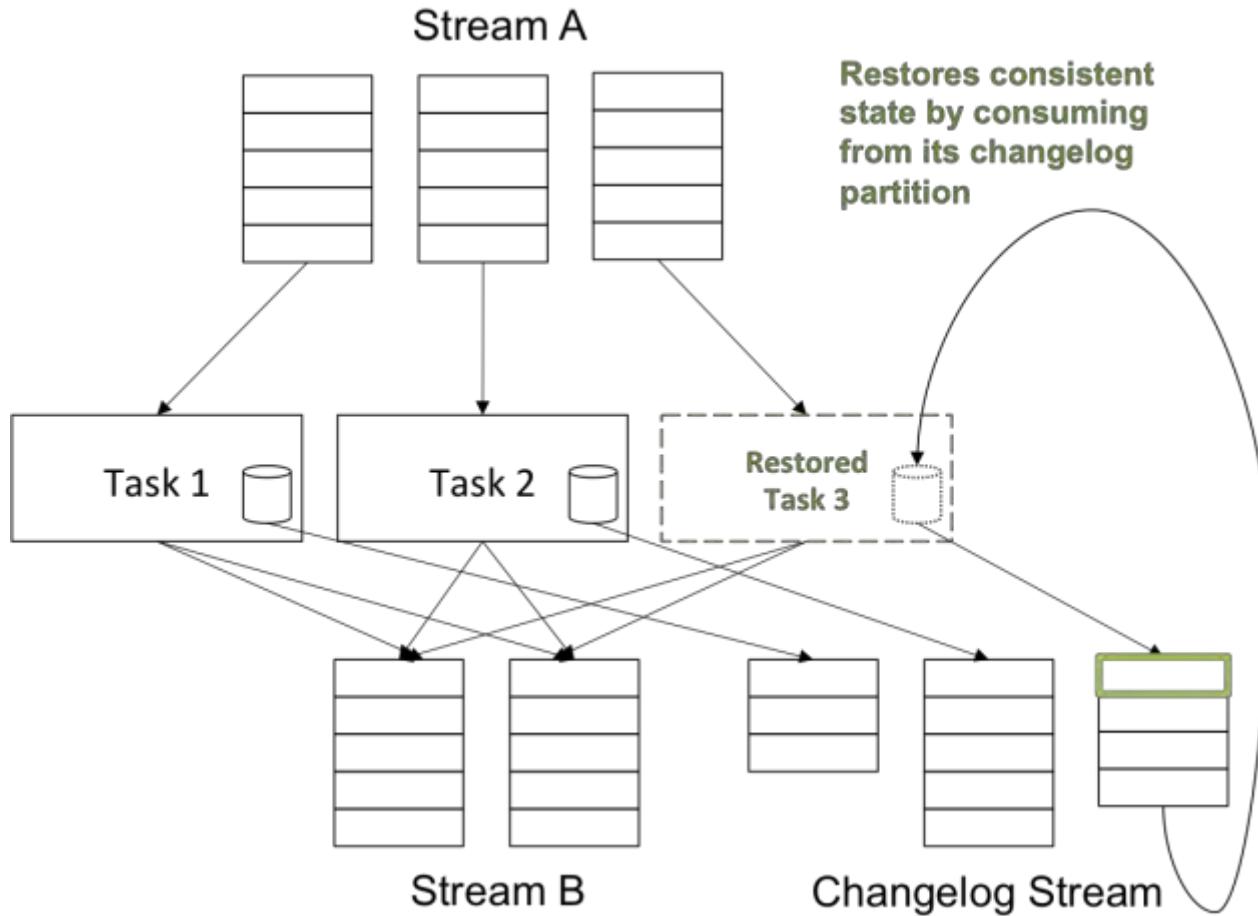


Illustration taken from: Navina Ramesh, *Apache Samza, LinkedIn's Framework for Stream Processing* (2015)
<https://thenewstack.io/apache-samza-linkedin-s-framework-for-stream-processing> (2017-02-26)

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

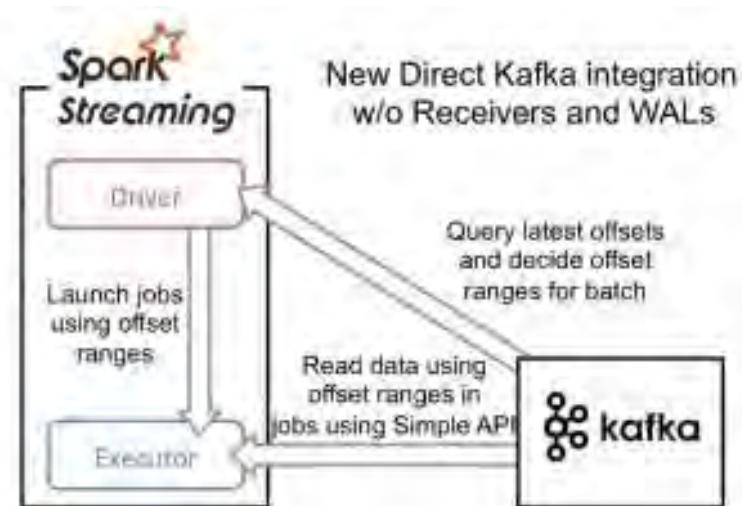
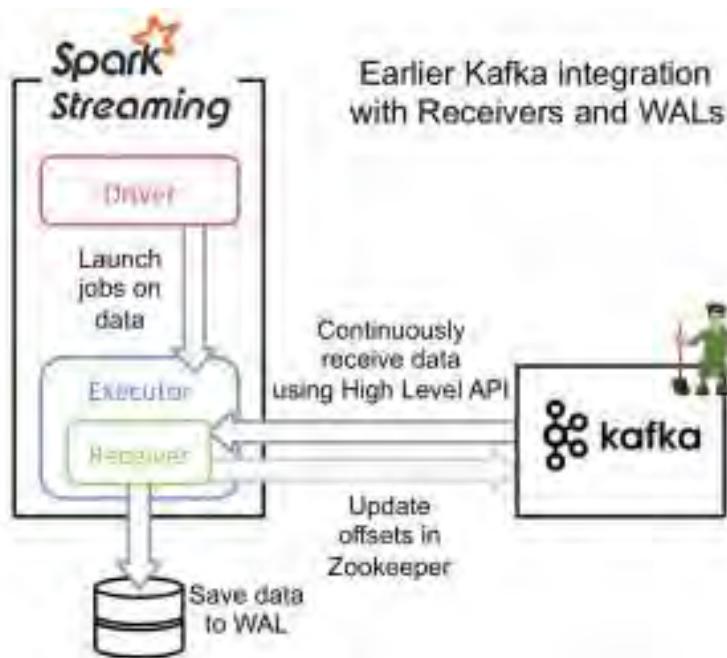


Illustration taken from:

<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:

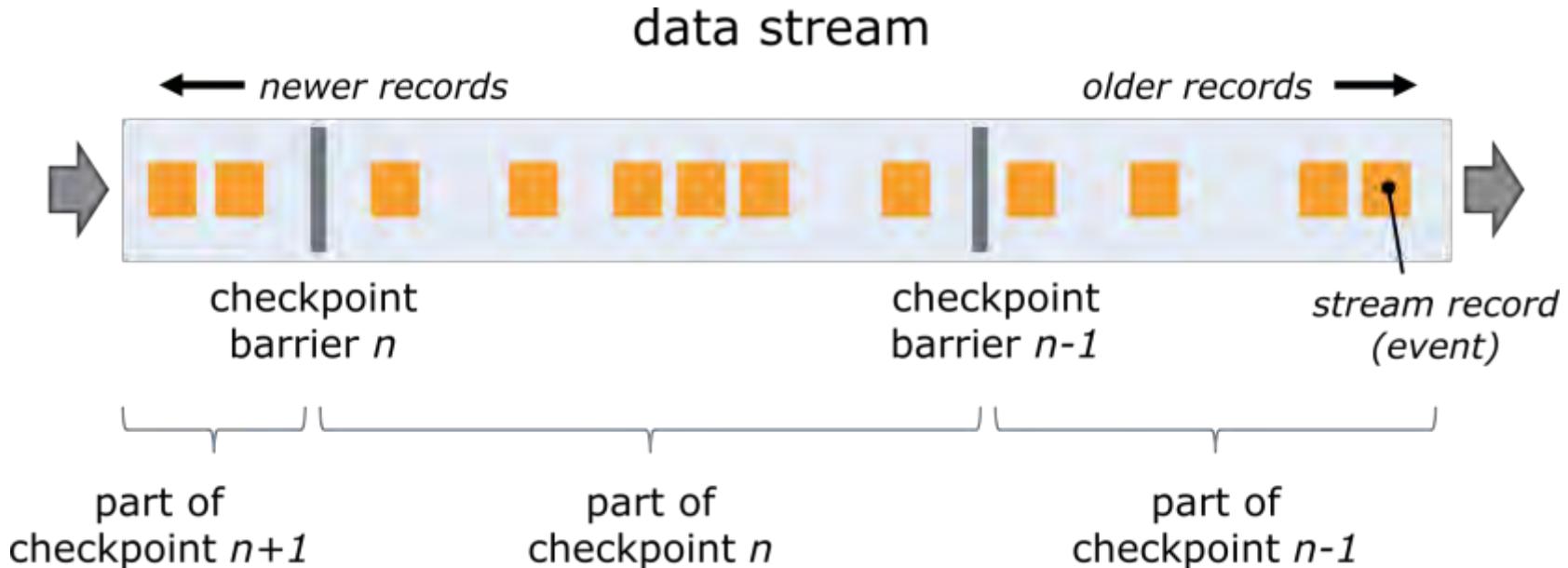
- **Native stream processor:** Latency <100ms feasible
- **Abstract API** for stream and batch processing, stateful, exactly-once delivery
- **Many libraries:**
 - Table and SQL: distributed and streaming SQL
 - CEP: complex event processing
 - Machine Learning
 - Gelly: graph processing
 - Storm Compatibility: 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



- **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)

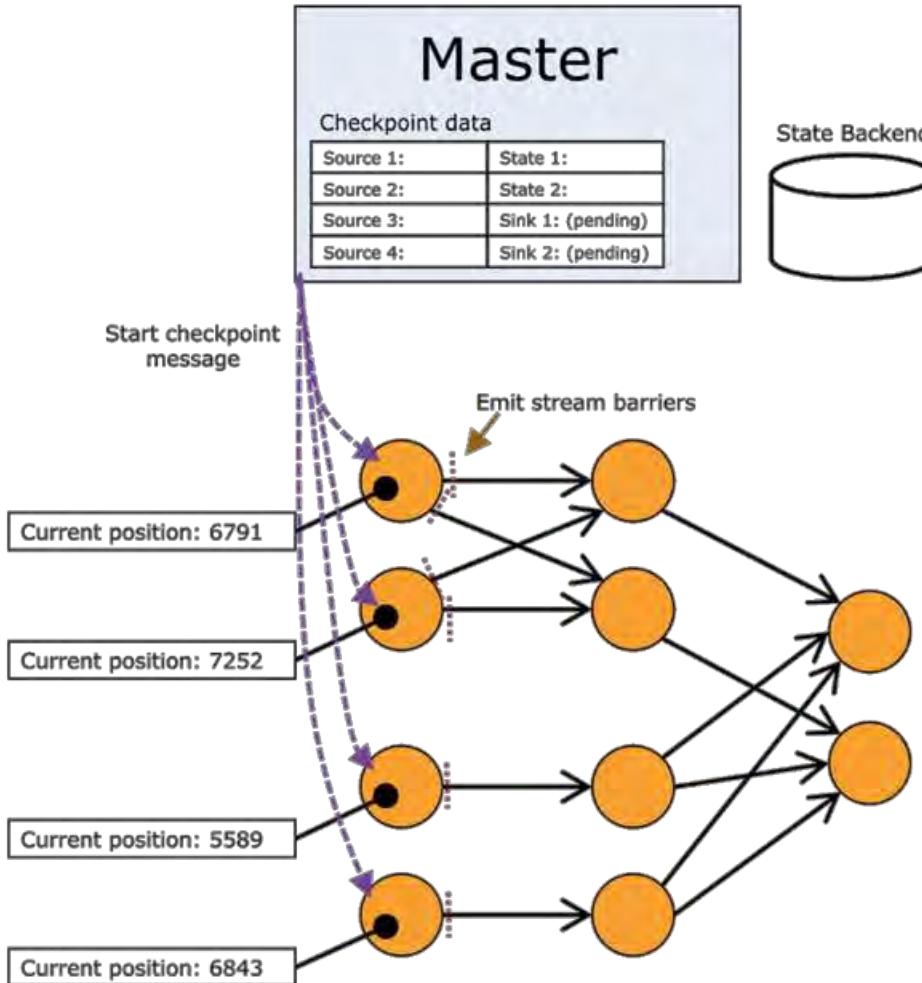


Illustration taken from: Robert Metzger, *Architecture of Flink's Streaming Runtime* (ApacheCon EU 2015)

<https://www.slideshare.net/robertmetzger1/architecture-of-flinks-streaming-runtime-apachecon-eu-2015> (2017-02-27)

State Management

Checkpointing (2/4)

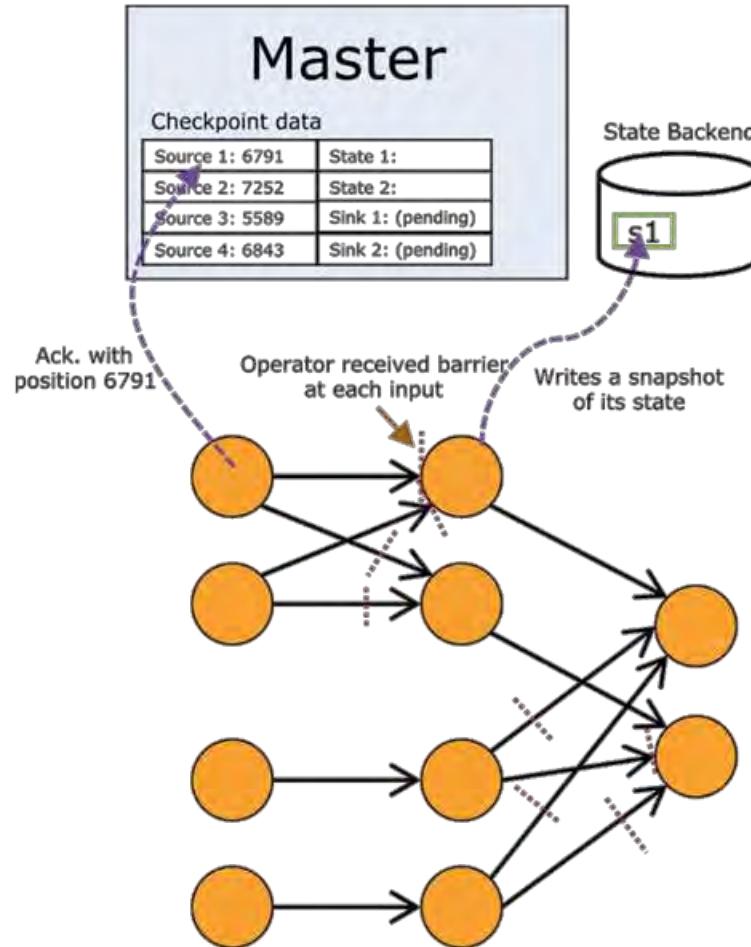


Illustration taken from: Robert Metzger, *Architecture of Flink's Streaming Runtime* (ApacheCon EU 2015)

<https://www.slideshare.net/robertmetzger1/architecture-of-flinks-streaming-runtime-apachecon-eu-2015> (2017-02-27)

State Management

Checkpointing (3/4)

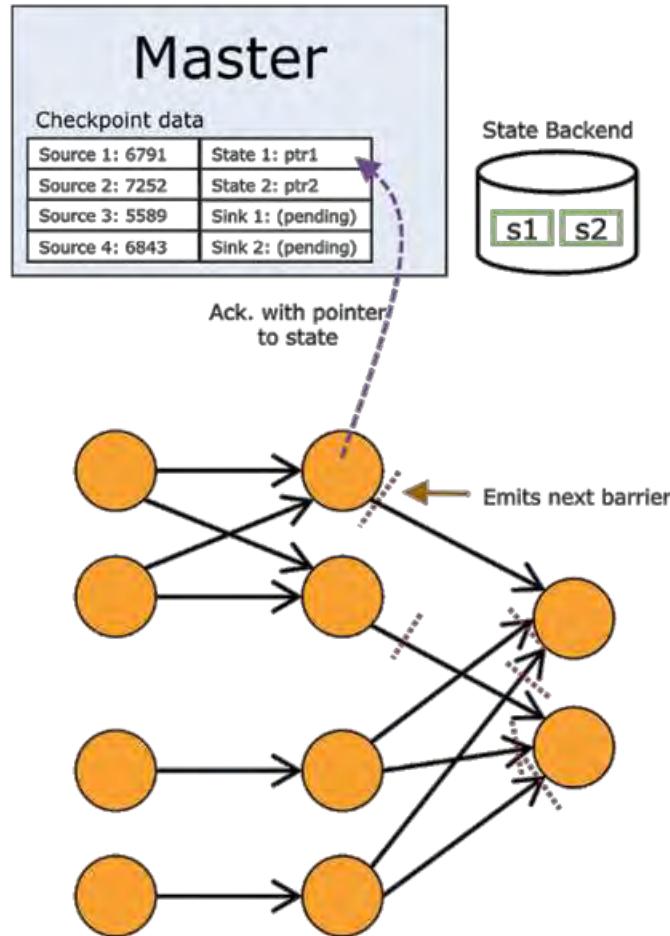
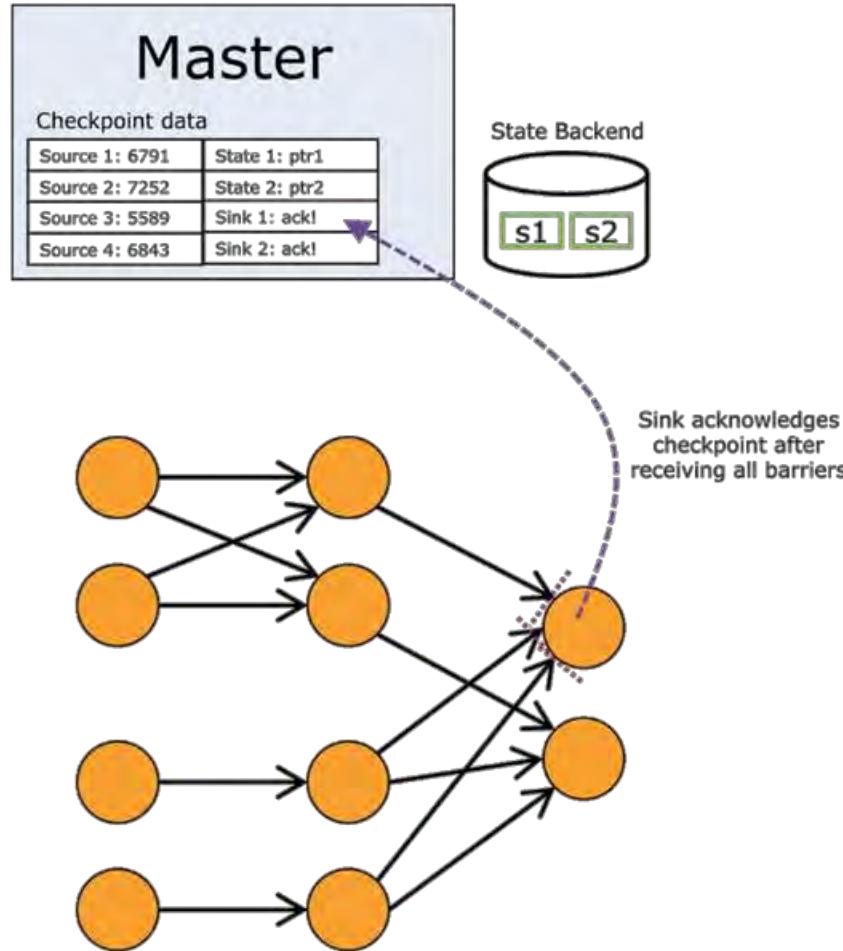


Illustration taken from: Robert Metzger, *Architecture of Flink's Streaming Runtime* (ApacheCon EU 2015)

<https://www.slideshare.net/robertmetzger1/architecture-of-flinks-streaming-runtime-apachecon-eu-2015> (2017-02-27)

State Management Checkpointing (4/4)



Other Systems

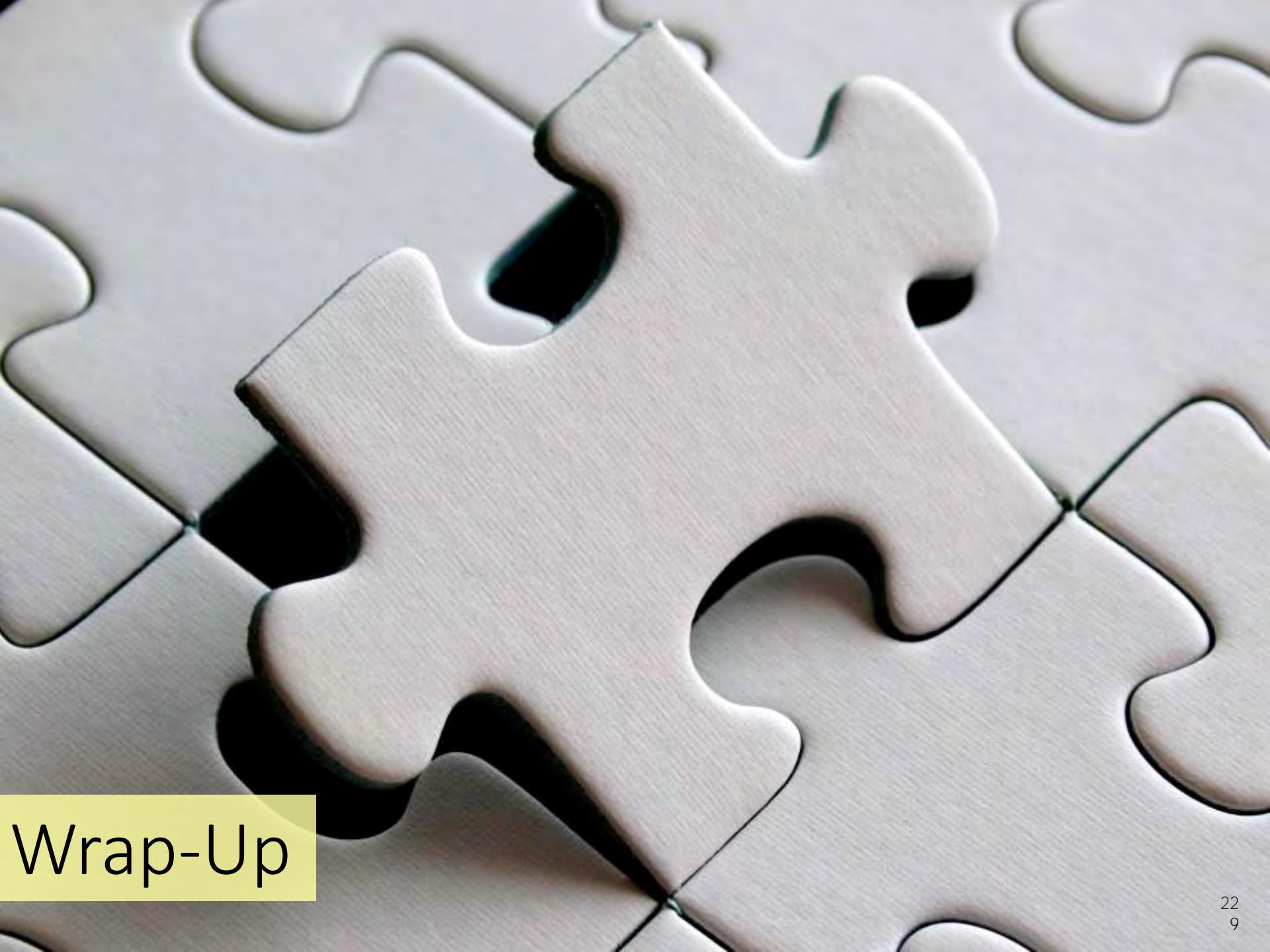


- **Heron**: open-source, Storm successor
- **Apex**: stream and batch process so with many libraries
- Dataflow**: Fully managed cloud service for batch and stream processing, proprietary
- **Beam**: open-source runtime-agnostic API for Dataflow programming model; runs on Flink, Spark and others
- **KafkaStreams**: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



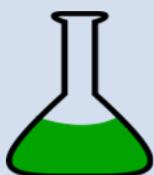
Scalable Data Processing:
Big Data in Motion



Stream Processors:
Side-by-Side Comparison



Real-Time Databases:
Push-Based Data Access



Current Research:
Opt-In Push-Based Access

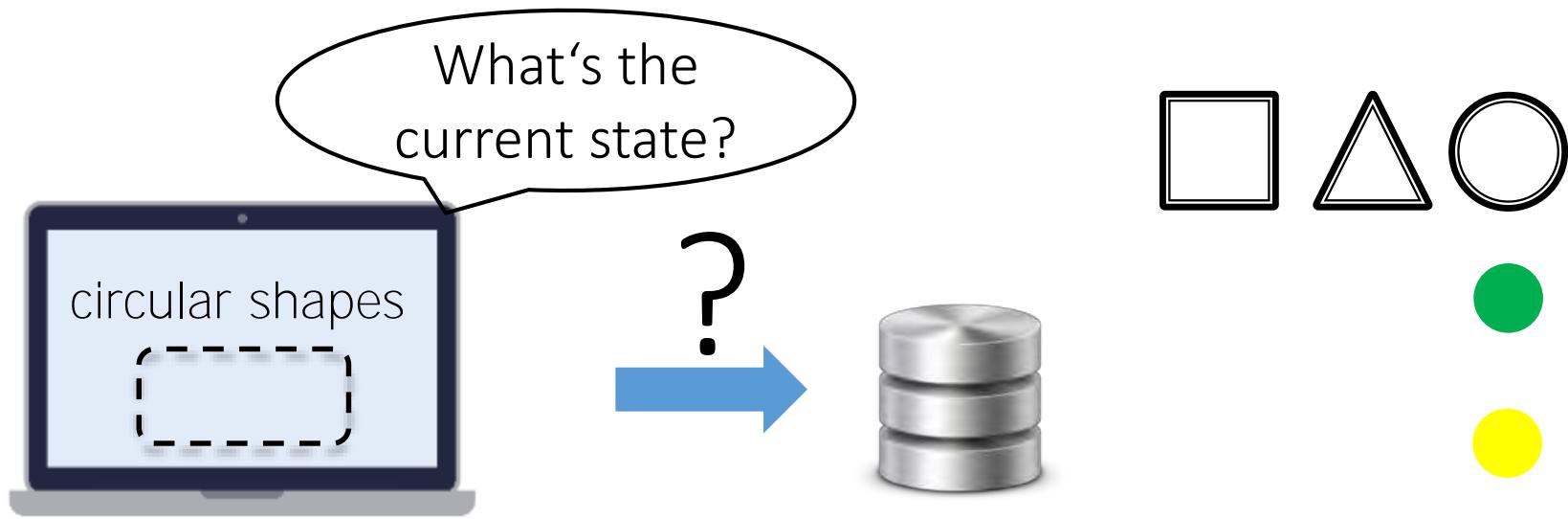
- Pull-Based vs Push-Based Data Access
- DBMS vs. RT DB vs. DSMS vs. Stream Processing
- Popular Push-Based DBs:
 - Firebase
 - Meteor
 - RethinkDB
 - Parse
 - Others
- Discussion



Real-Time Databases

Traditional Databases

No Request? No Data!



Query maintenance: periodic polling

- Inefficient
- Slow

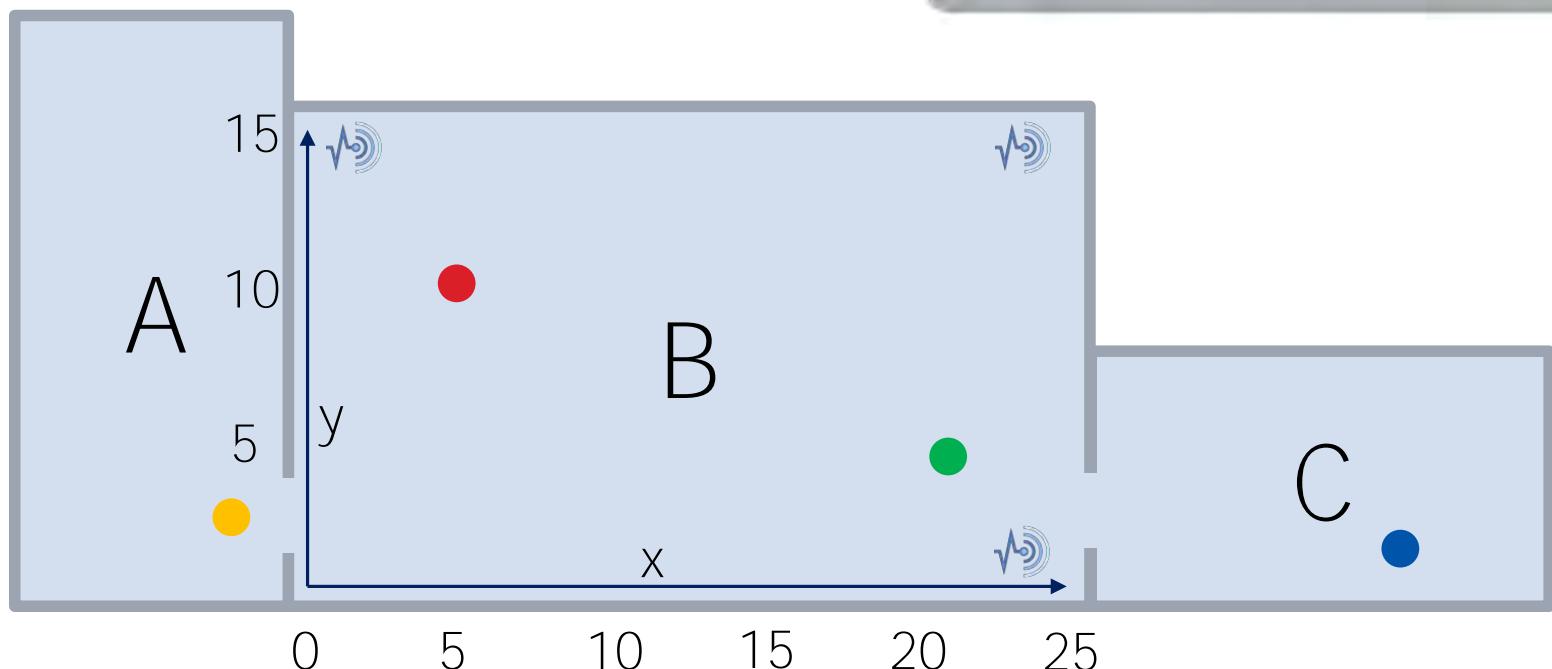
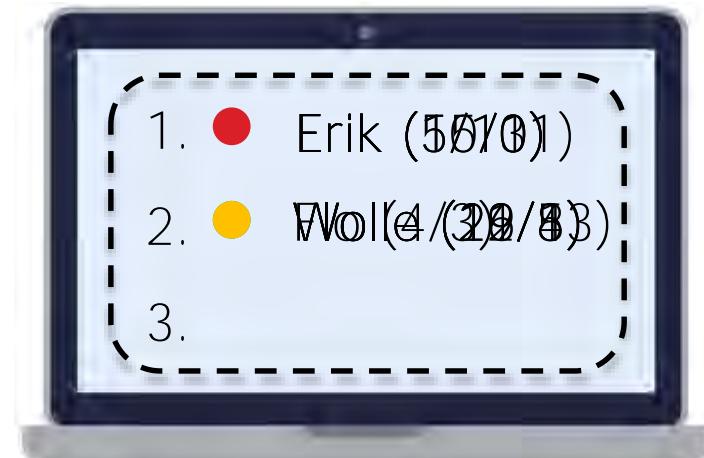
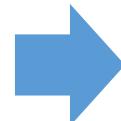


Ideal: Push-Based Data Access

Self-Maintaining Results

Find people in Room B:

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





LONDON



NEW YORK



TOKYO



MOSCOW

Popular Real-Time Databases

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

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



→ *Limited expressiveness!*

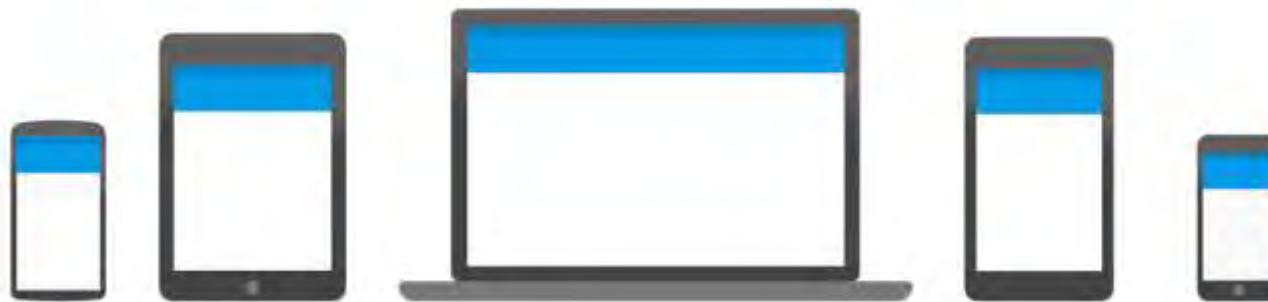


Illustration taken from: Frank van Puffelen, *Have you met the Realtime Database?* (2016)
<https://firebase.googleblog.com/2016/07/have-you-met-realtime-database.html> (2017-02-27)

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.firebaseio-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)

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:

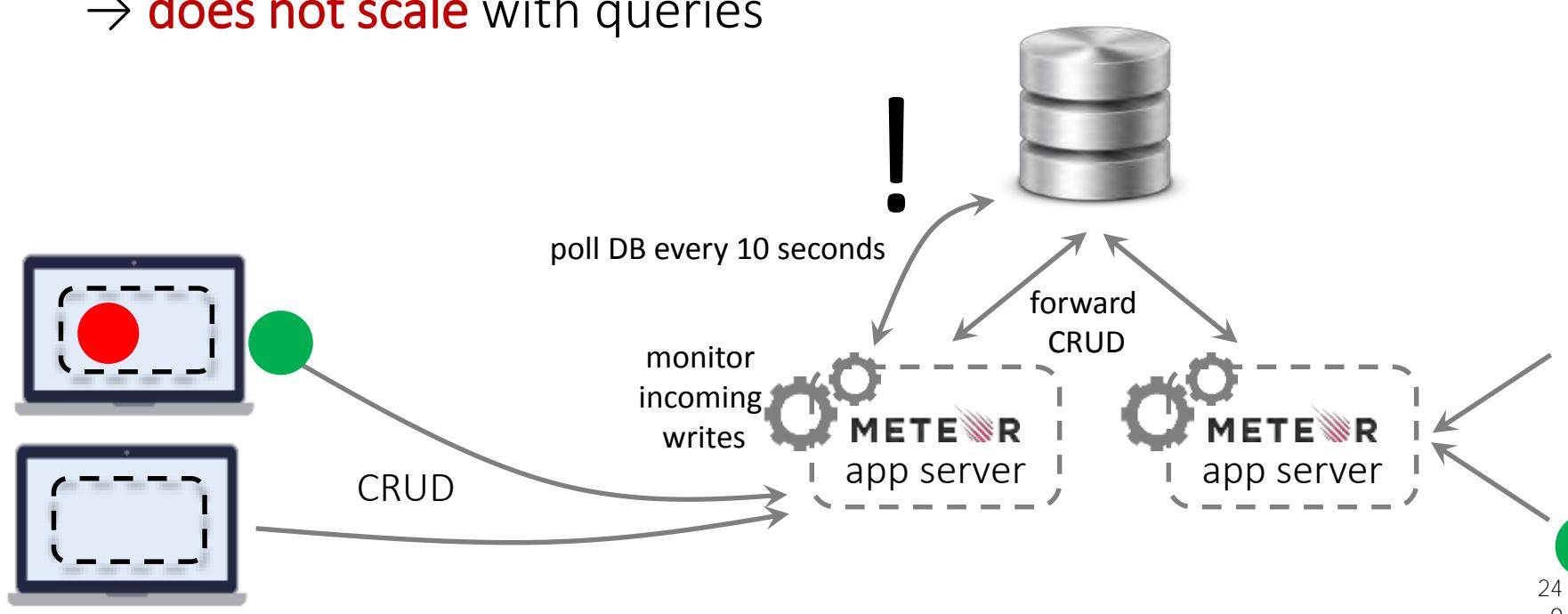
- 2011: *Skybreak* is announced
- 2012: Skybreak is renamed to Meteor
- 2015: Managed hosting service Galaxy is announced

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

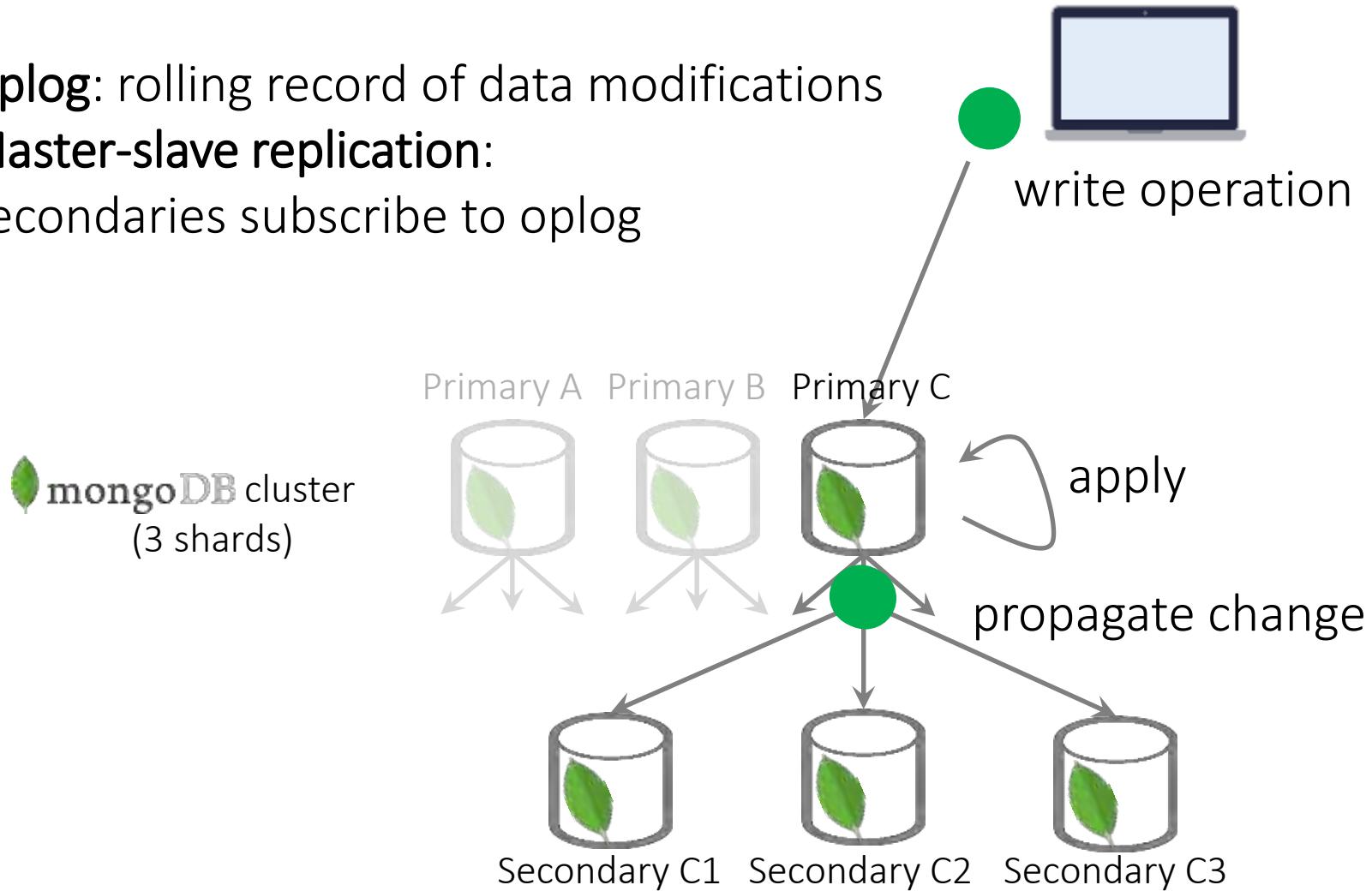


Oplog Tailing



Basics: MongoDB Replication

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

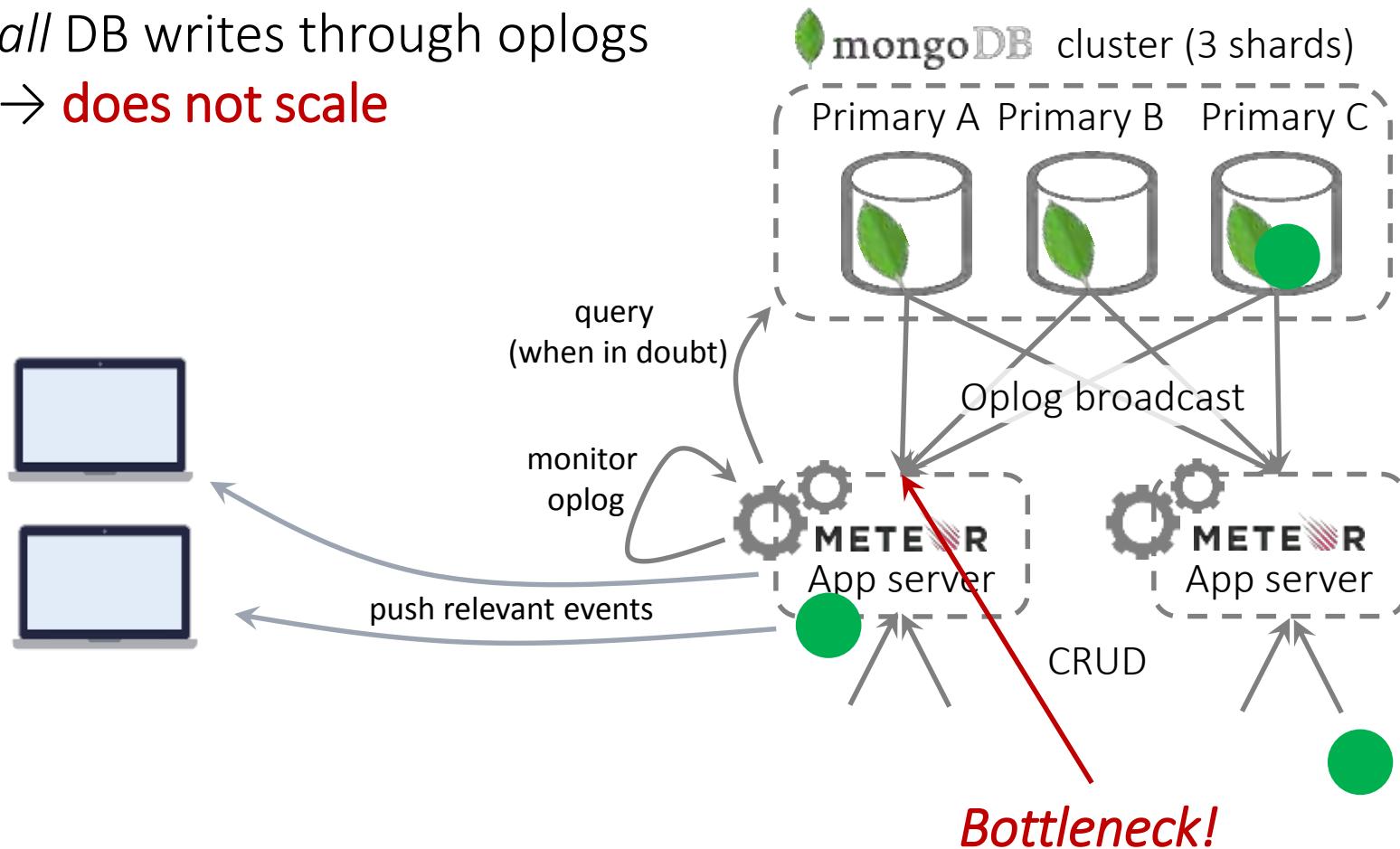


Oplog Tailing

Tapping into the Oplog



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



Oplog Tailing

Oplog Info is Incomplete



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-score



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

## 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

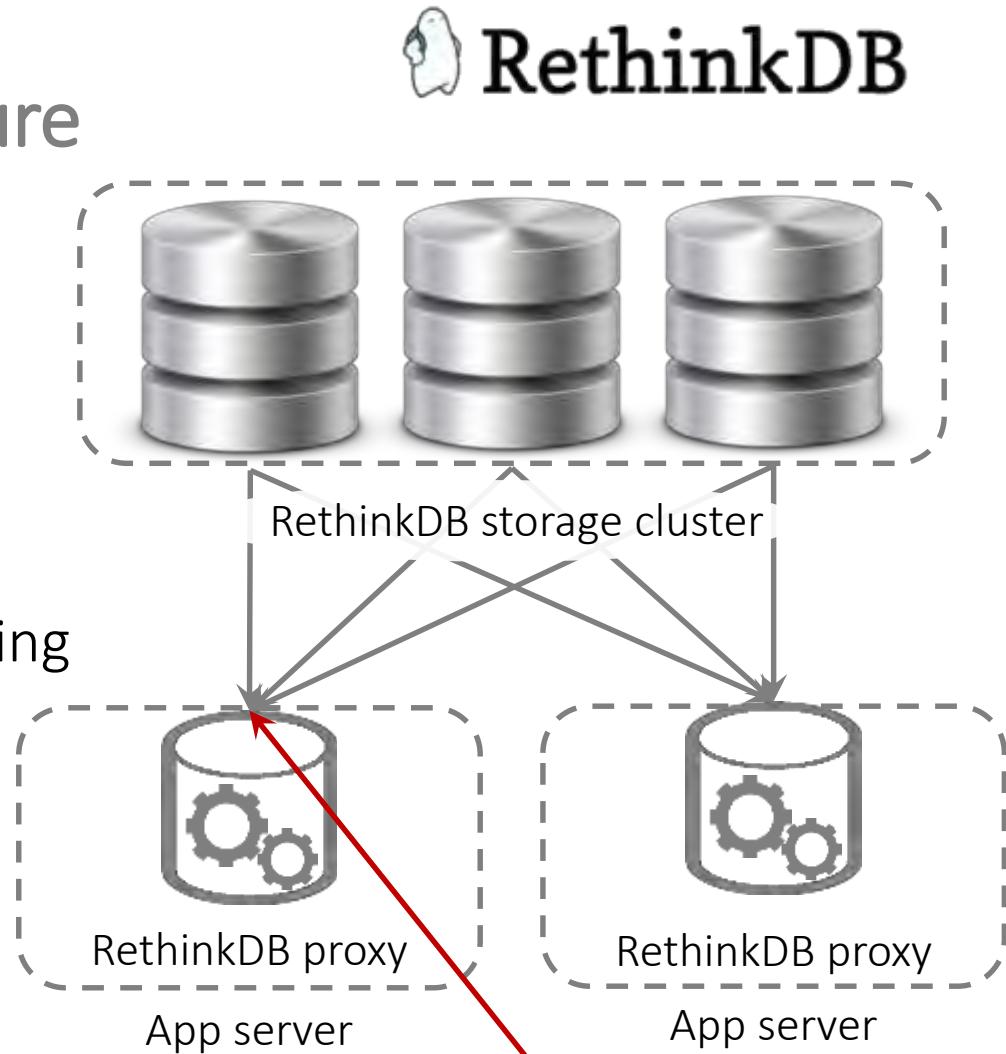
## History:

- 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

- 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



## 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

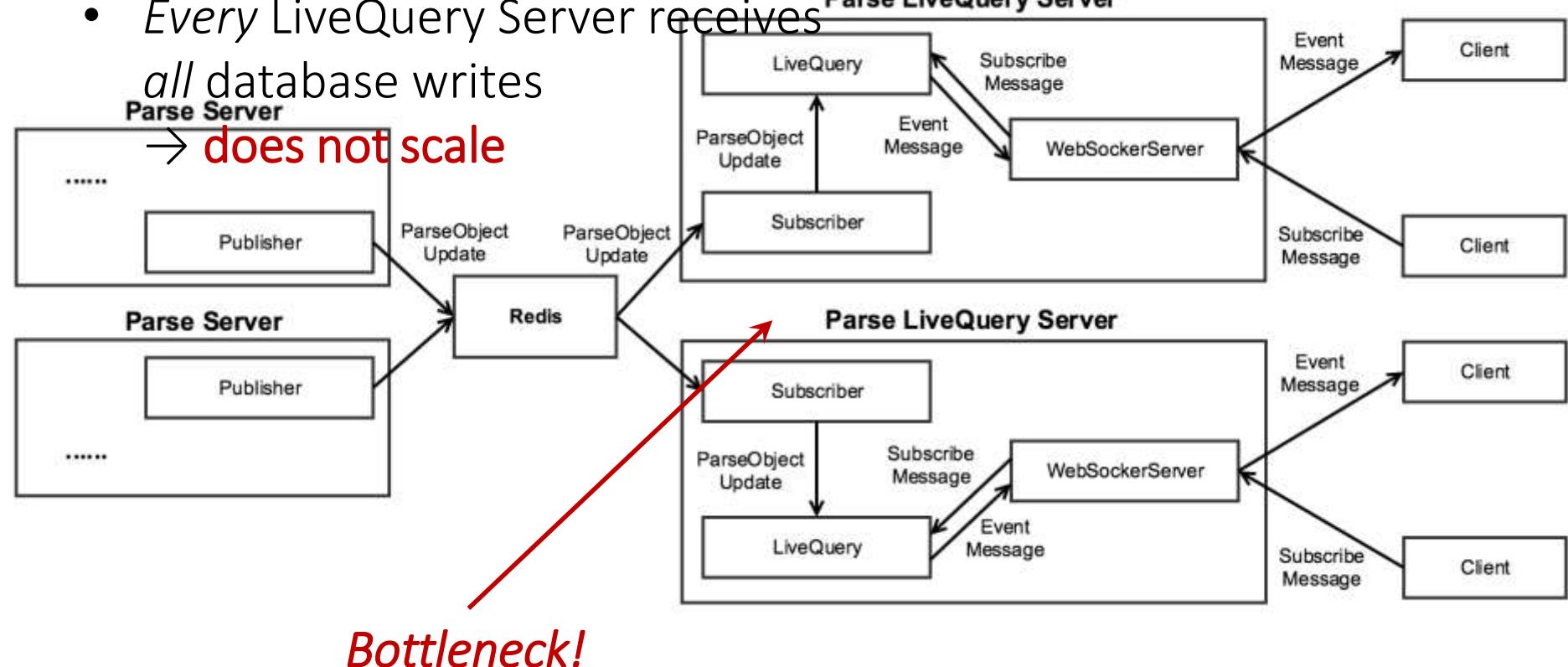
# Parse

## LiveQuery Architecture



- **LiveQuery Server:** no data, real-time query matching
- *Every LiveQuery Server receives all database writes*

→ does not scale



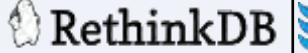
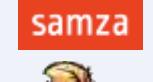
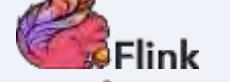
# Comparison by Real-Time Query

## Why Complexity Matters

|       | matching conditions                                      | ordering                                                 | Firebase | Meteor | RethinkDB | Parse |
|-------|----------------------------------------------------------|----------------------------------------------------------|----------|--------|-----------|-------|
| Todos | created by „Bob“                                         | ordered by deadline                                      | ✓        | ✓      | ✓         | ✗     |
| Todos | created by „Bob“<br>AND<br>with status equal to „active“ |                                                          | ✗        | ✓      | ✓         | ✓     |
| Todos | with „work“ in the name                                  |                                                          | ✗        | ✓      | ✓         | ✓     |
|       |                                                          | ordered by deadline                                      | ✗        | ✓      | ✓         | ✗     |
| Todos | with „work“ in the name<br>AND<br>status of „active“     | ordered by deadline<br>AND<br>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                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Data</b>                                                                                                                                                                                                                                                                                                                                                                                                                            | persistent collections |                                                                                                                                                                                                                                                                                                                                                                                                                        | persistent/ephemeral streams                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Processing</b>                                                                                                                                                                                                                                                                                                                                                                                                                      | one-time               | one-time + continuous                                                                                                                                                                                                                                                                                                                                                                                                  | continuous                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Access</b>                                                                                                                                                                                                                                                                                                                                                                                                                          | random                 | random + sequential                                                                                                                                                                                                                                                                                                                                                                                                    | sequential                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Streams</b>                                                                                                                                                                                                                                                                                                                                                                                                                         | structured             |                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                         | structured,<br>unstructured                                                                                                                                                                                                                                                                                                                                                                                                    |
|   <br>  |                        |  <b>Firebase</b><br> <b>METEOR</b><br> <b>RethinkDB</b><br> <b>Parse</b> |  <b>PIPELINE DB</b><br> <b>EsperTech</b><br> <b>sqlstream</b><br> <b>influxdata</b> |  <b>STORM</b><br> <b>samza</b><br> <b>Flink</b><br> <b>Spark Streaming</b> |

# 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

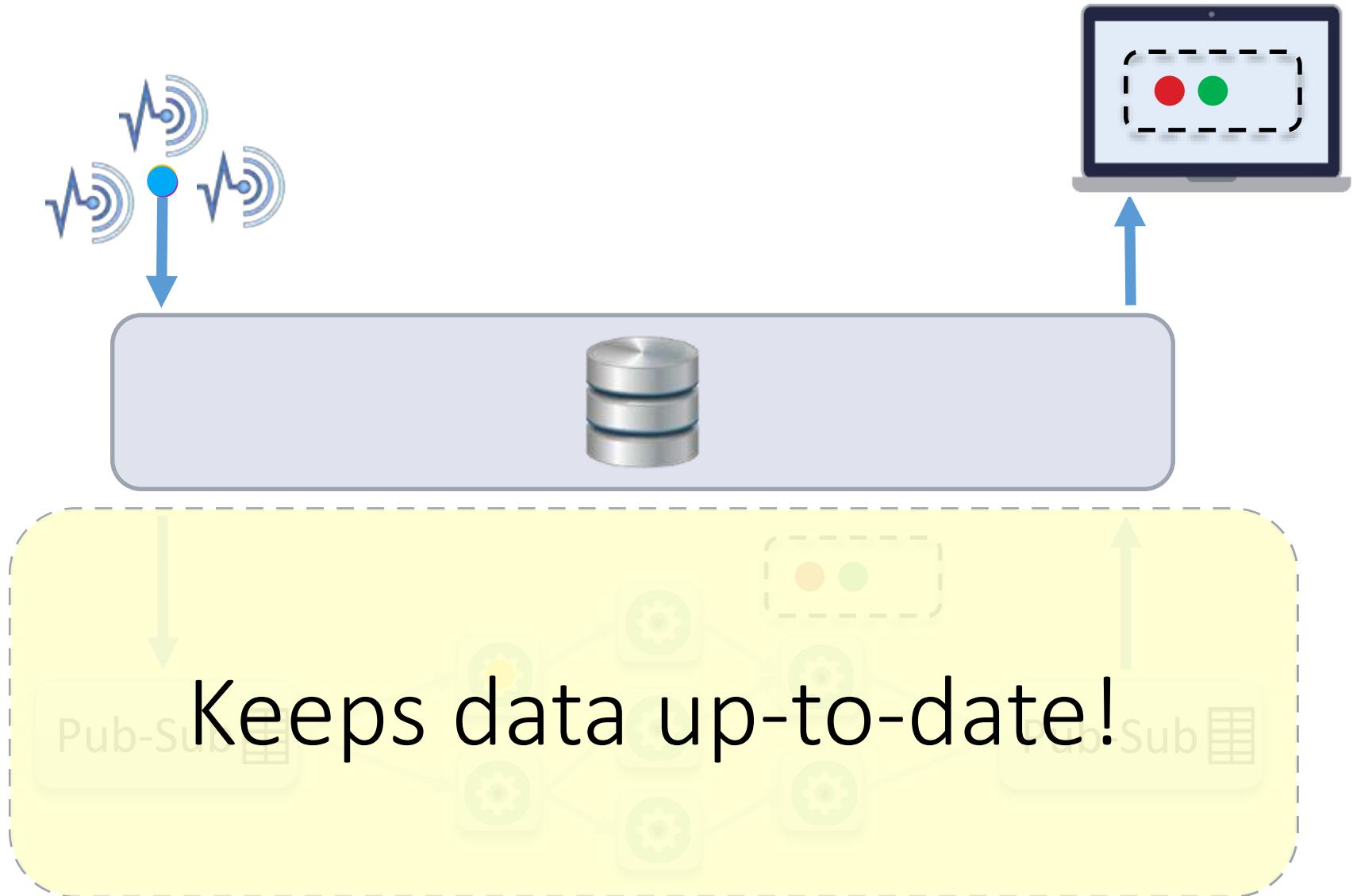
- InvaliDB:  
Opt-In Real-Time Queries
- Distributed Query Matching
- Staged Query Processing
- Performance Evaluation
- Wrap-Up



Current Research

# InvaliDB

## External Query Maintenance



# InvaliDB

## Change Notifications

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

```
{ title: null, year: -1 } { title: {"NoSQL": {"NoSQL": "NoSQL"}, year: -1 } { 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**

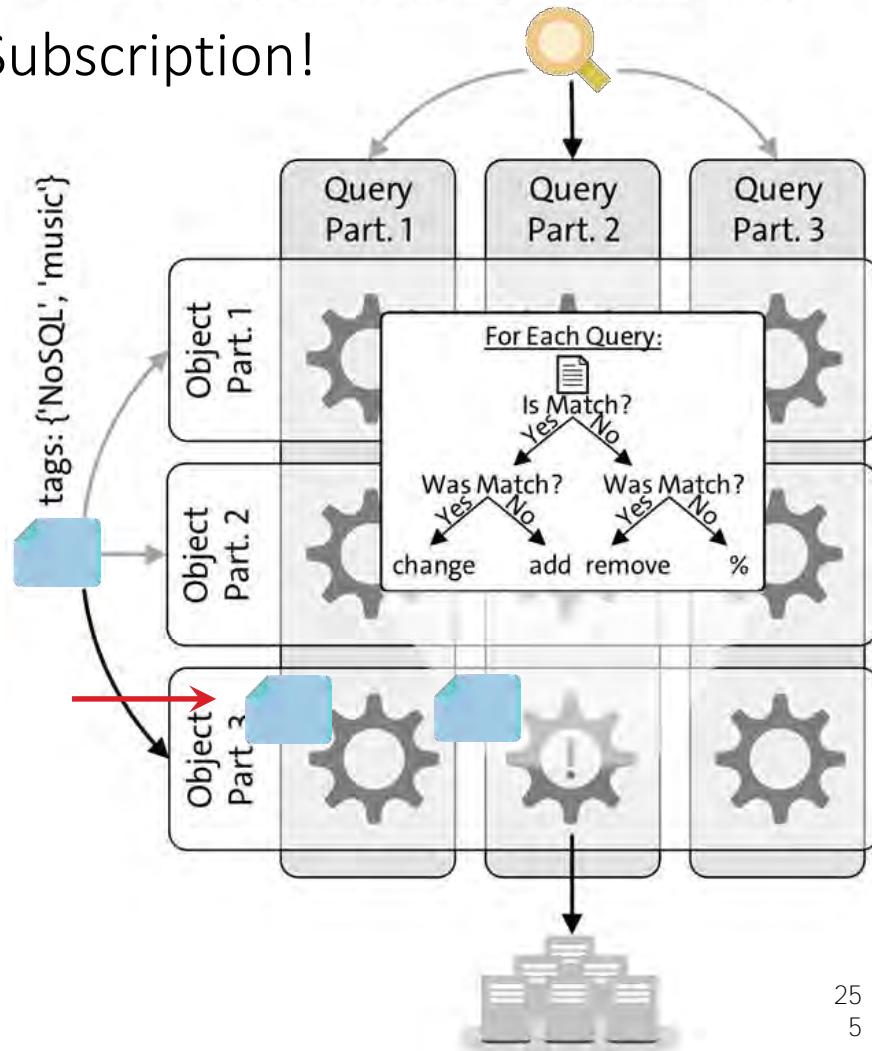
Implementation:

- Apache Storm
- Topology in Java
- MongoDB query language
- Pluggable query engine

Write op!

SELECT \* FROM posts WHERE tags CONTAINS 'NoSQL'

Subscription!

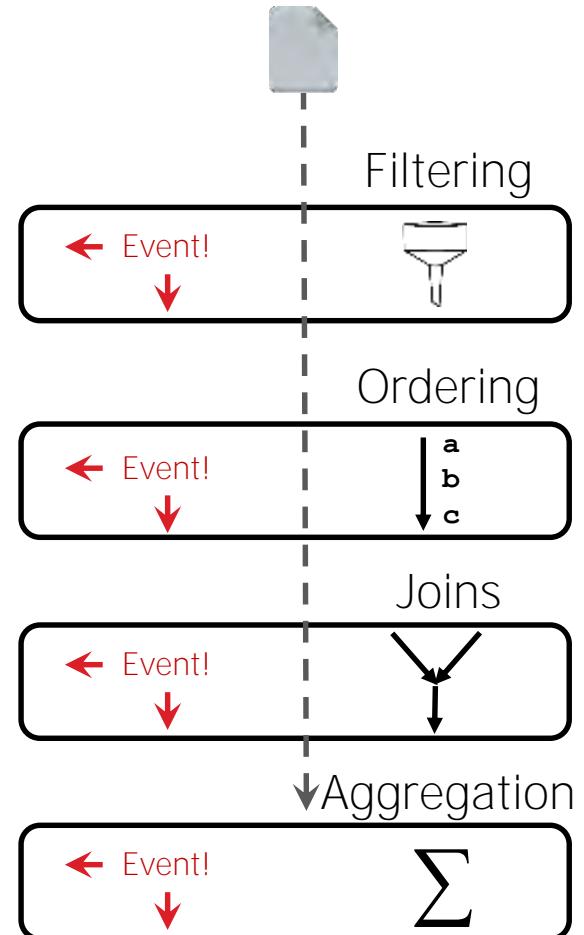


# InvaliDB

## Staged Real-Time Query Processing

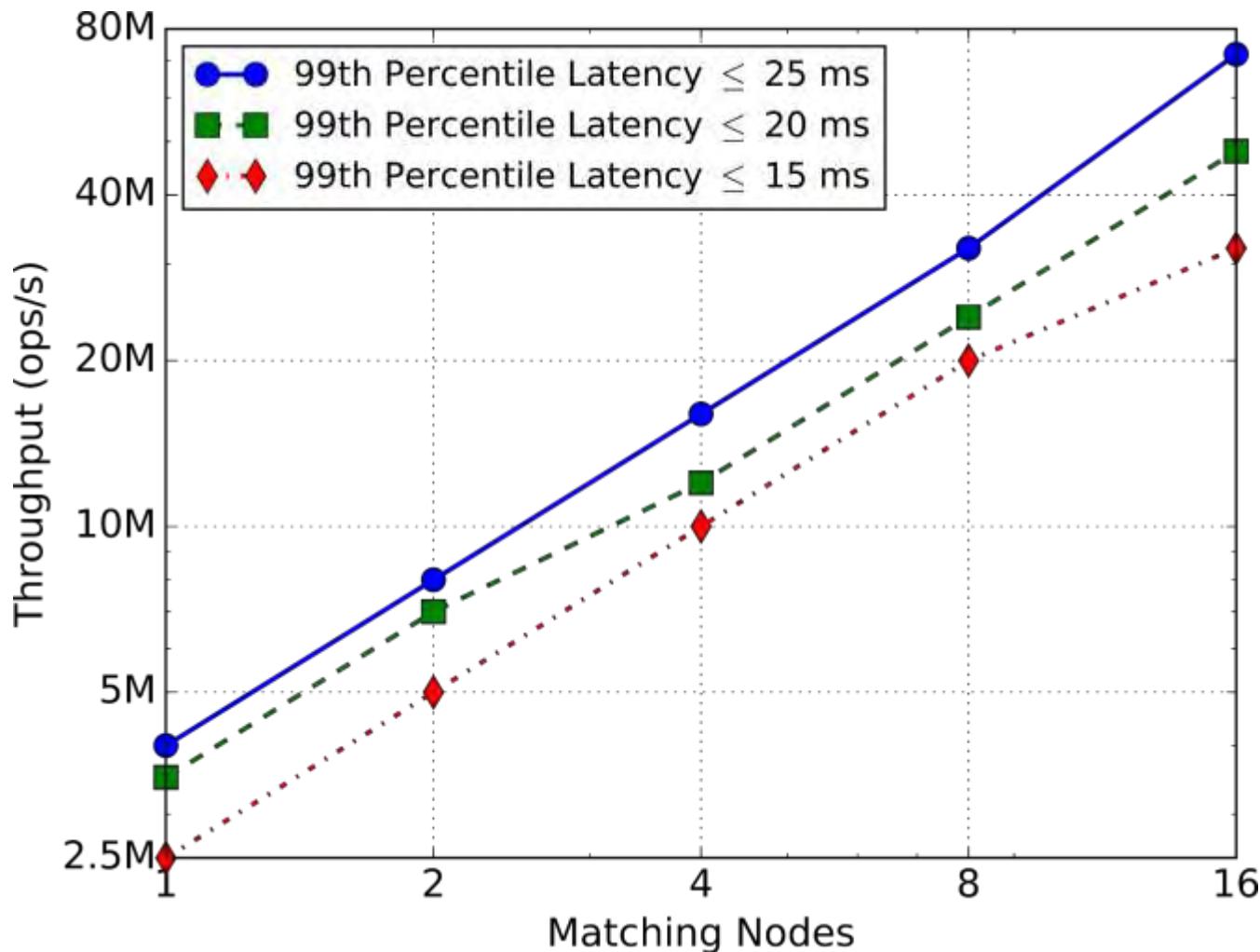
Change notifications go through up to 4 query processing stages:

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



A photograph of a woman with long dark hair, seen from behind, looking out over a body of water towards a large cargo ship docked at a port. The sky is a warm, golden color of a sunset or sunrise. The ship's hull and the port's industrial structures are illuminated by numerous lights, creating a reflection on the water. The overall atmosphere is contemplative and scenic.

Our NoSQL research at the  
University of Hamburg



Presentation  
is loading

# Why performance matters

1000000ms

Average: 9,3s

Loading...



Conversions

Traffic

Visitors

Revenue

Aberdeen *Group*

Google

YAHOO!

amazon.com®



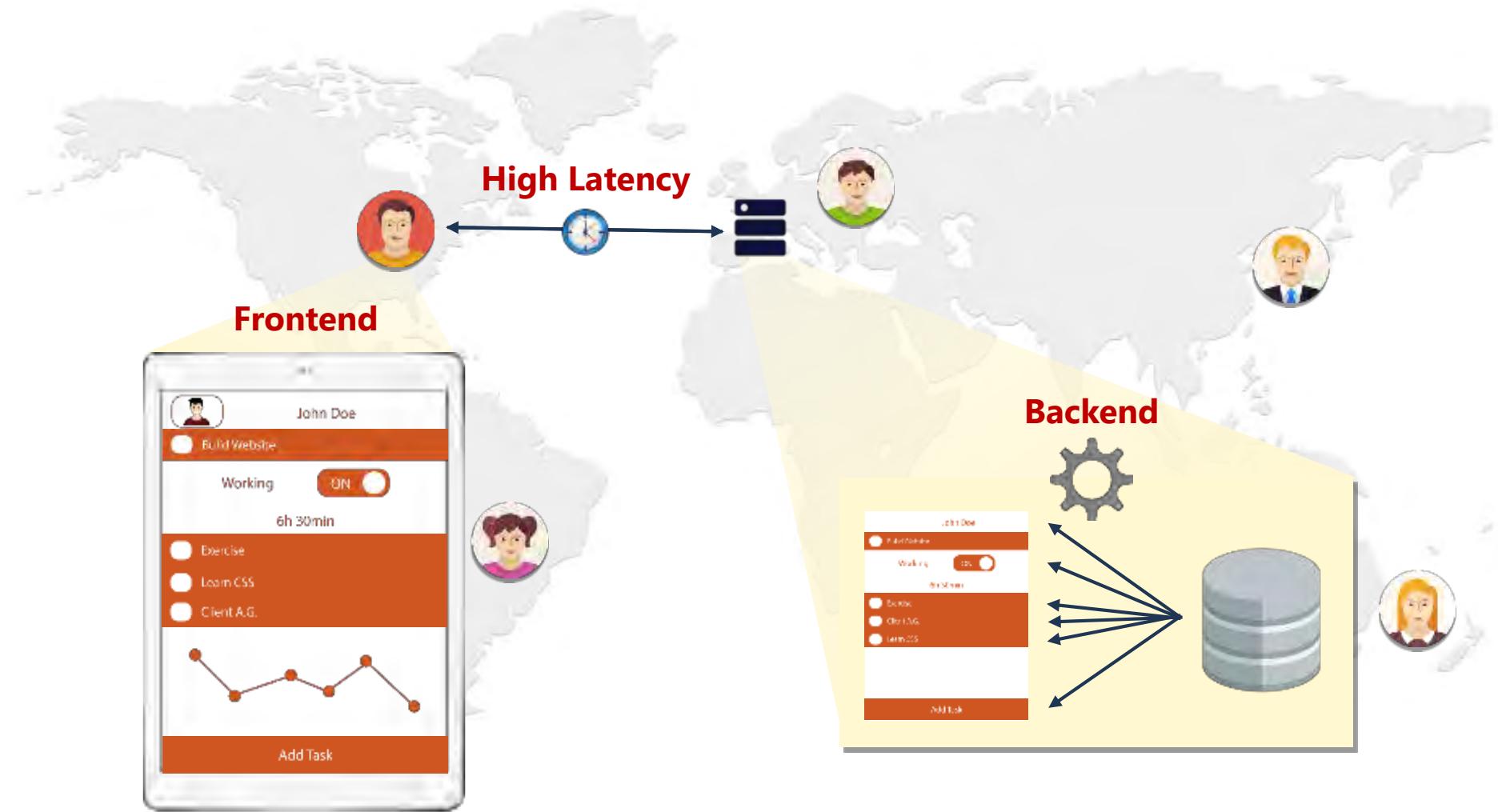
If perceived speed is such an important factor



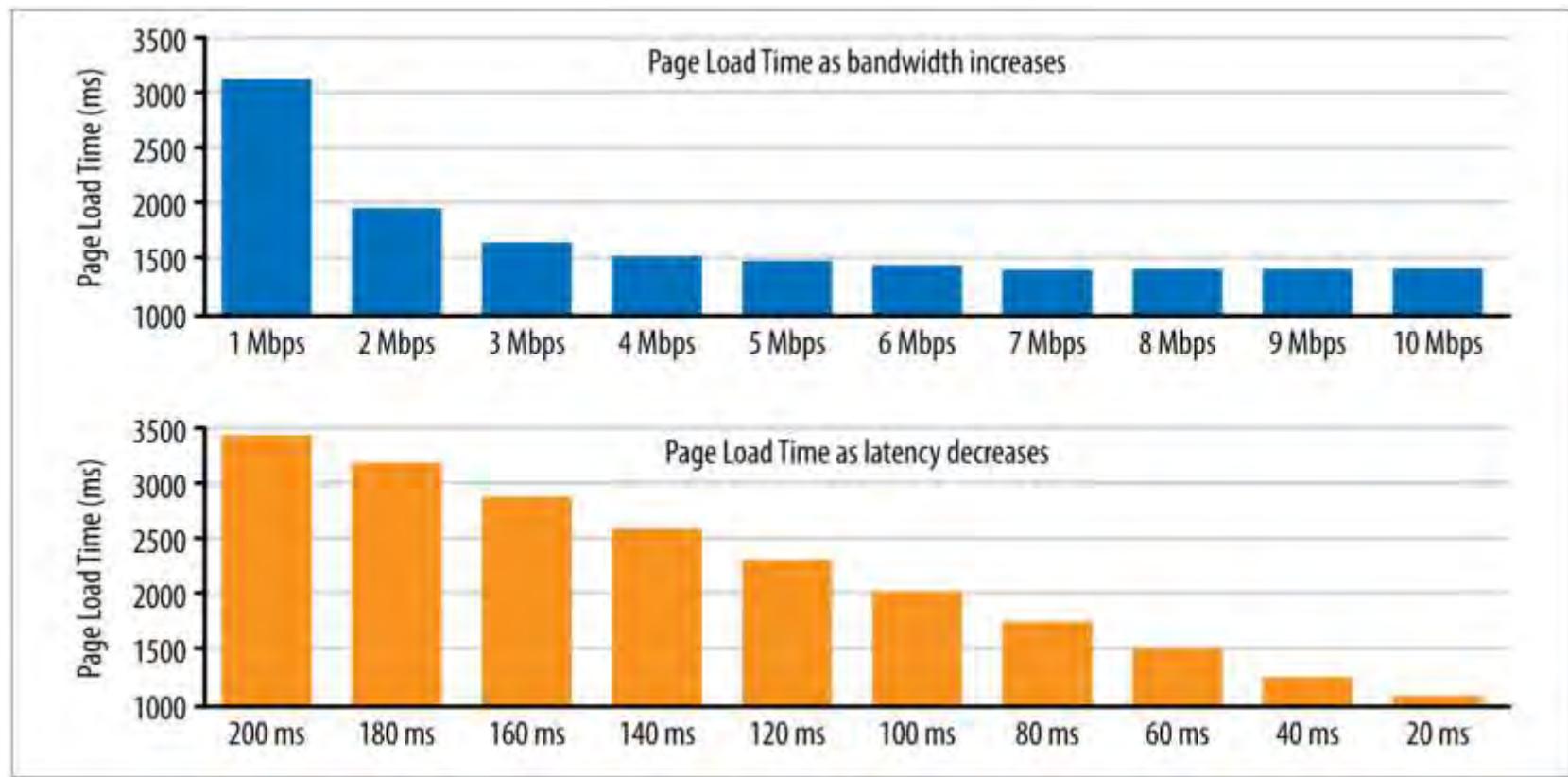
...what causes slow page load times?

# The Problem

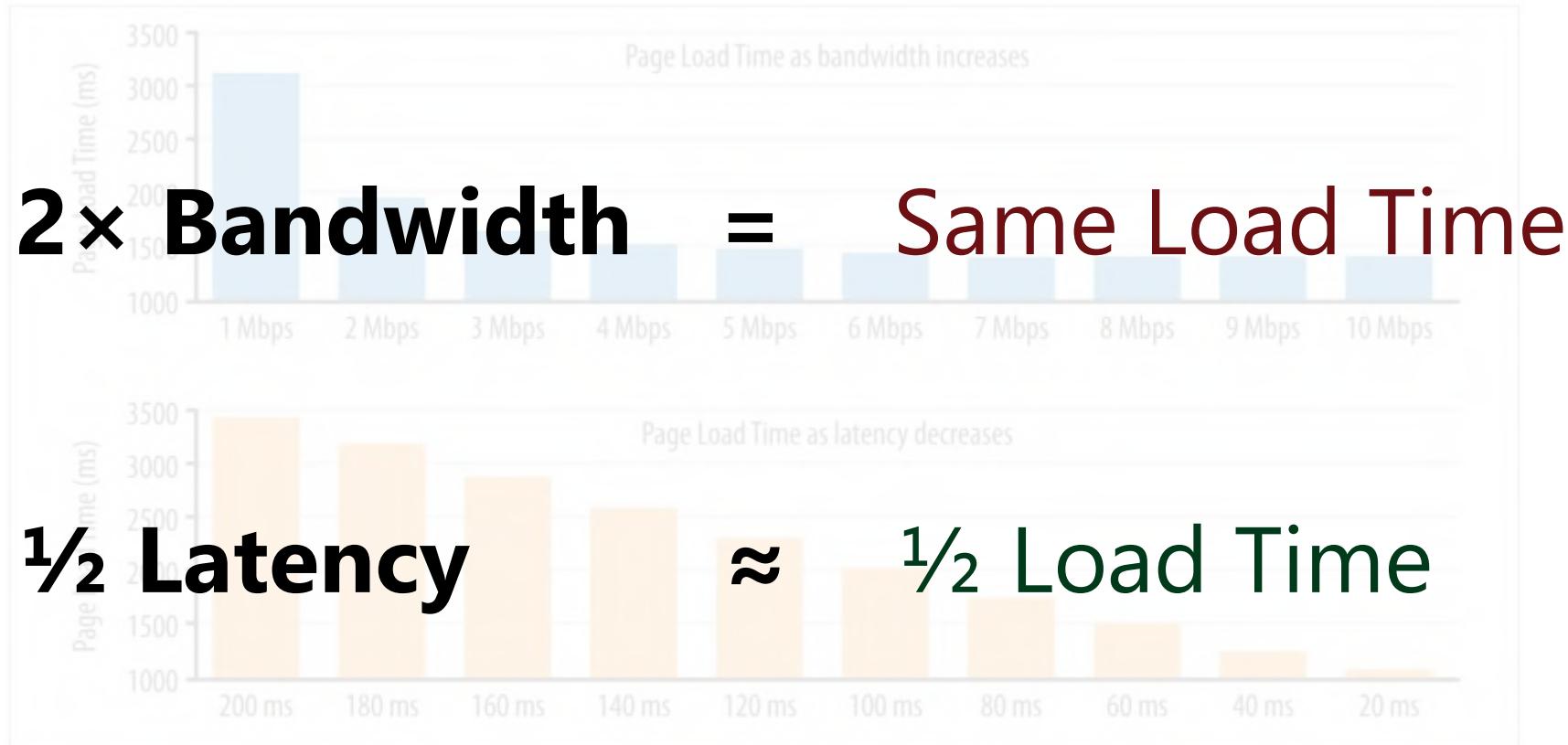
Three Bottlenecks: Latency, Backend & Frontend



# Network Latency: Impact

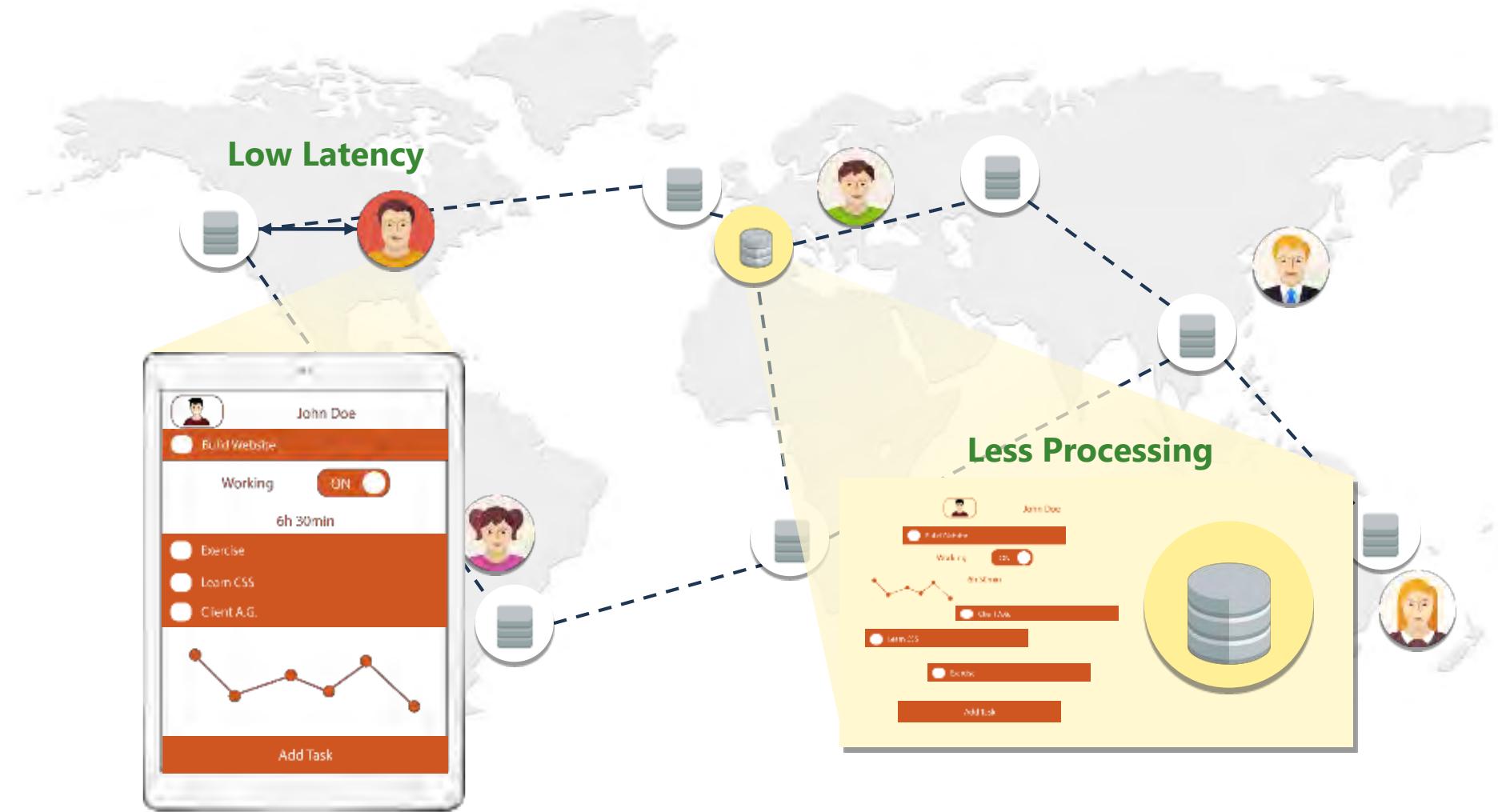


# Network Latency: Impact



# Goal: Low-Latency for Dynamic Content

By Serving Data from Ubiquitous Web Caches

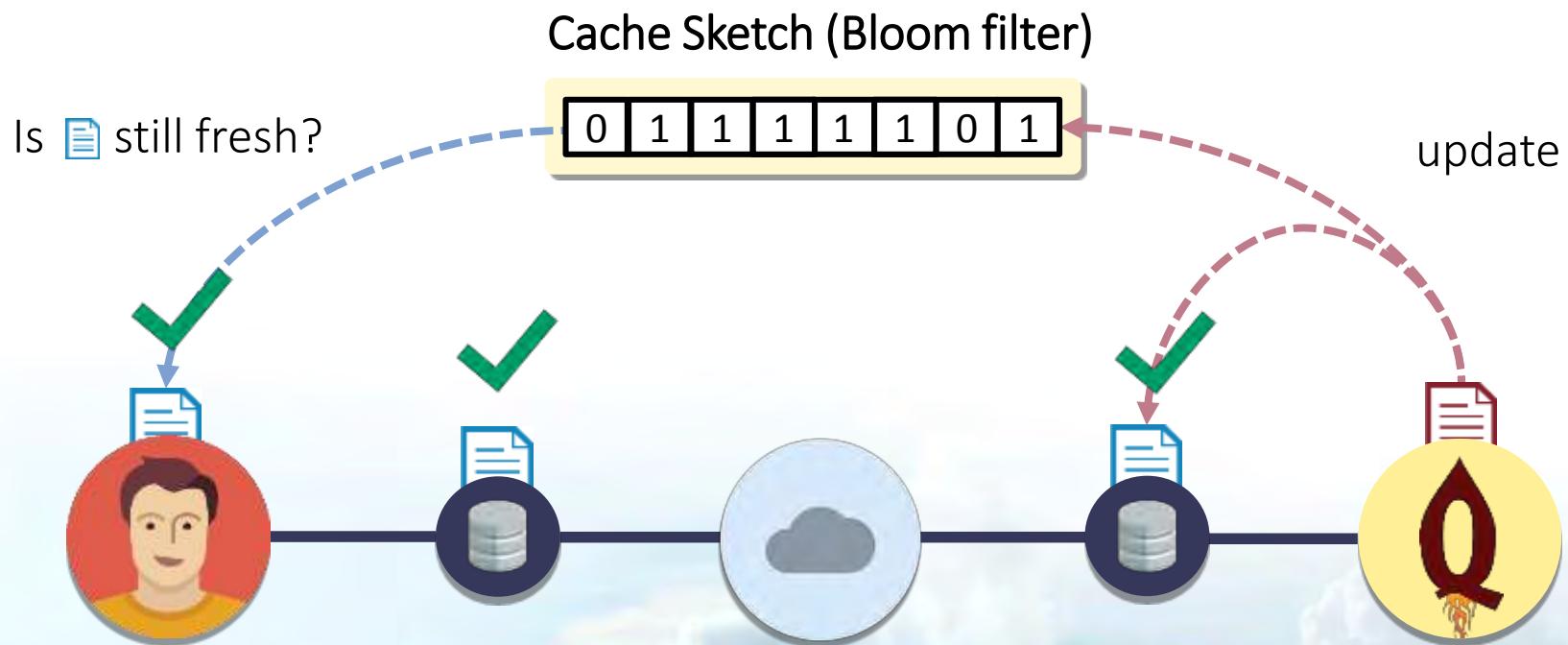


# In a nutshell



# In a nutshell

Solution: Proactively Revalidate Data



# 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 und F. Bücklers, „ORESTES: ein System für horizontal skalierbaren Zugriff auf Cloud-Datenbanken“, in *Informatiktage 2013*, 2013.

 F. Gessert und F. Bücklers, *Performanz- und Reaktivitätssteigerung von OODBMS vermittels der Web-Caching-Hierarchie*. Bachelorarbeit, 2010.

 M. Schaarschmidt, F. Gessert, und N. Ritter, „Towards Automated Polyglot Persistence“, 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, 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, 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, *Kohärentes Web-Caching von Datenbankobjekten im Cloud Computing*. Masterarbeit 2012.

 W. Wingerath, S. Friedrich, und F. Gessert, „Who Watches the Watchmen? On the Lack of Validation in NoSQL Benchmarking“, in *BTW 2015*.

 F. Gessert, „Skalierbare NoSQL- und Cloud-Datenbanken in Forschung und Praxis“, *BTW 2015*



# Page-Load Times

## What impact does caching have in practice?

.0s  
5.7s  
4.7s

### Politik



11. November 2014 12:42 Uhr  
Deutsche Rentenversicherung

#### Renten könnten 2015 um zwei Prozent steigen

Die Deutsche Rentenversicherung geht von einem Anstieg über der Inflationsrate aus. Abschlagsfreie Rente ab 53 Jahren stößt auf großes Interesse.



11. November 2014 10:05 Uhr  
Europäischer Gerichtshof

#### Deutschland darf EU-Ausländern Hartz IV verweigern

Der Europäische Gerichtshof hat entschieden: Deutschland kann arbeitslose Zuwanderer aus der EU von Sozialleistungen ausschließen. Das Urteil könnte ein Signal sein.



11. November 2014 05:48 Uhr  
APEC-GIPFELTREFFEN

#### Obama besänftigt China

Die USA wollen China nicht klein halten, sagt Präsident Obama vor dem Treffen mit Chinas Staatschef Xi. Der plädiert für mehr wirtschaftliche Verflechtung.



10. November 2014 19:17 Uhr  
ISRAEL

#### Keiner will von Intifada sprechen

Messerattacken auf Israelis, Krawalle auf dem Tempelberg, Schermützel im Gassengewirr

### Wirtschaft



11. November 2014 07:15 Uhr  
HONORARBERATUNG

#### Guter Rat zur Geldanlage ist selten

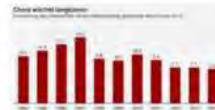
Honorarberatung ist in Deutschland endlich gesetzlich geregelt. Doch gibt es kaum Honorarberater. Und gut qualifizierte noch viel weniger.



10. November 2014 21:32 Uhr  
CHINA

#### Der berühmteste Wohltäter Chinas – nach eigenen Angaben

Der chinesische Unternehmer Chen Guangbiao wurde ausgerechnet mit Bauschutt sehr reich. Jetzt baut er Wände aus Geldbündeln und zertrümmert öffentlich Luxusautos.



10. November 2014 19:29 Uhr  
KONJUNKTUR

#### China steckt in der Wachstumsfalle

Jahrelang hat China die Welt mit hohen, oft zweistelligen Wachstumsraten beeindruckt. Doch diese Zeiten sind vorbei, wie unsere Grafik des Tages zeigt.



10. November 2014 13:45 Uhr  
WÄHRUNG

#### Russlands Zentralbank lässt Rubel frei handeln



### Kultur



11. November 2014 10:14 Uhr  
NICOLAUS HARNONCOURT

#### Mozarts Triptychon

Nikolaus Harnoncourt ist der Detektiv unter den Dirigenten. Jetzt legt er Indizien vor, wie drei von Mozarts Sinfonien zu einem nie gehörten Oratorium verschmelzen.



11. November 2014 06:39 Uhr  
HANS MAGNUS ENZENSBERGER

#### Der Unerschütterliche

Hans Magnus Enzensberger wird 85. Ein Besuch bei dem herrlich eigenwilligen Intellektuellen. Mit Tumult hat er gerade ein erstaunlich persönliches Buch veröffentlicht.



10. November 2014 um 18:25 Uhr  
DDR-DESIGN

#### Sandmännchen und Stasi-Mikrofone

Das größte Museum für DDR-Design steht ausgerechnet in Los Angeles. Ein Buch über das Wende Museum zeigt, welche Schätze und Abgründe es dort zu entdecken gibt.



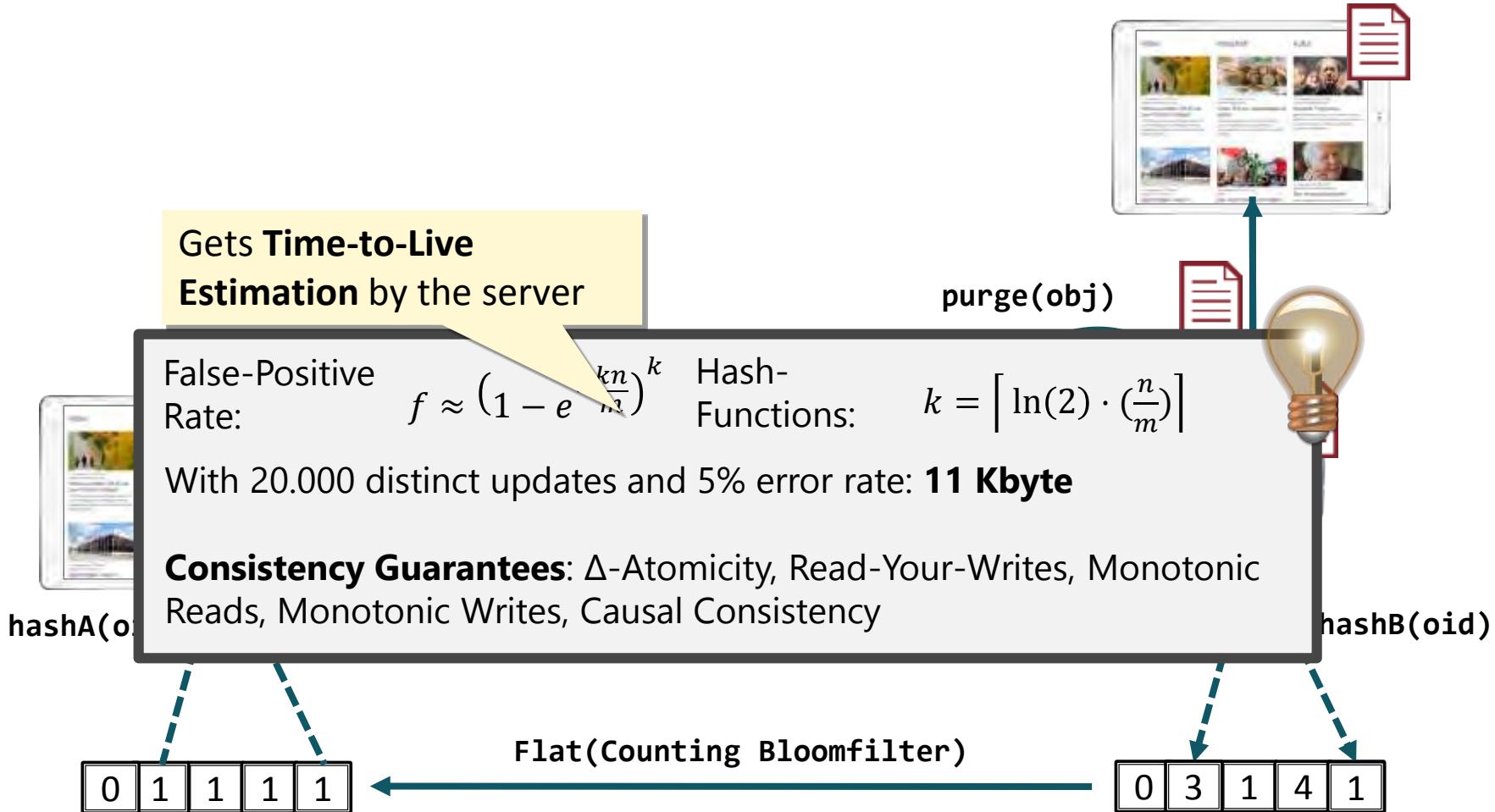
10. November 2014 um 15:25 Uhr  
AZEALIA BANKS

#### Klare Ansage aus Harlem

Erst galt Azealia Banks als großes Raptalent, dann als streitsüchtig und selbstverliebt. Ihr seit Jahren erwartetes Debüt zeigt jetzt, wie gut das eine zum anderen passt.

# Bloom filters for Caching

## End-to-End Example

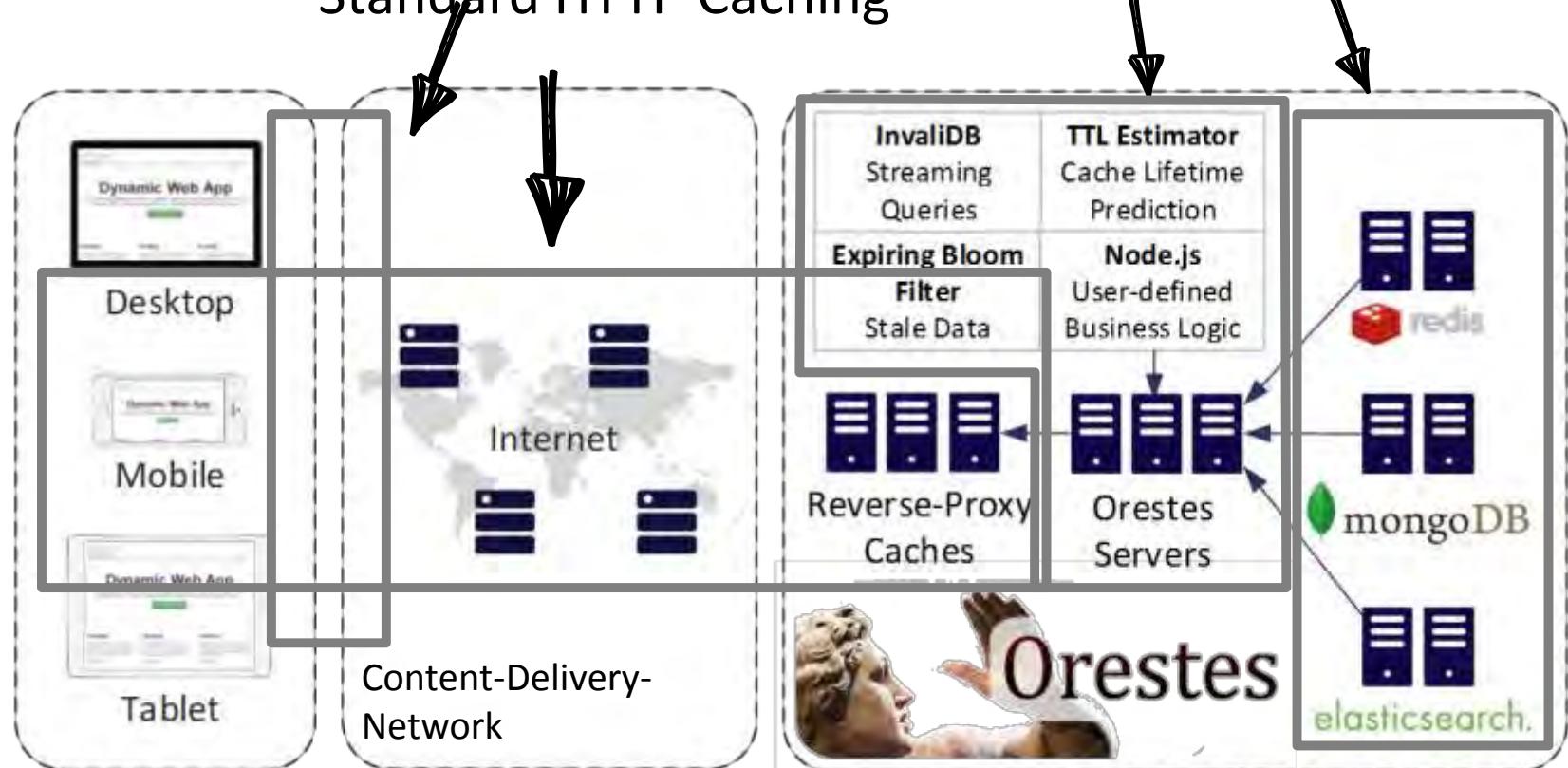


# Orestes Architecture

## Infrastructure

Backend-as-a-Service Middleware:  
Caching, Transactions, Schemas,

Unified REST API  
Standard HTTP Caching  
Invalidation Detection  
Hot Storage





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.

A screenshot of a mobile application interface titled "Tutorial App". At the top, there are three buttons: "Todo" (unchecked), "Done" (checked), and "All". Below these buttons, a message says "No items here.". At the bottom, there is a text input field with the placeholder "New Todo". At the very bottom of the screen, the text "Baqend Tutorial" is visible.



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](https://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.

### NoSQL Databases: A Survey and Decision Guidance

#### 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

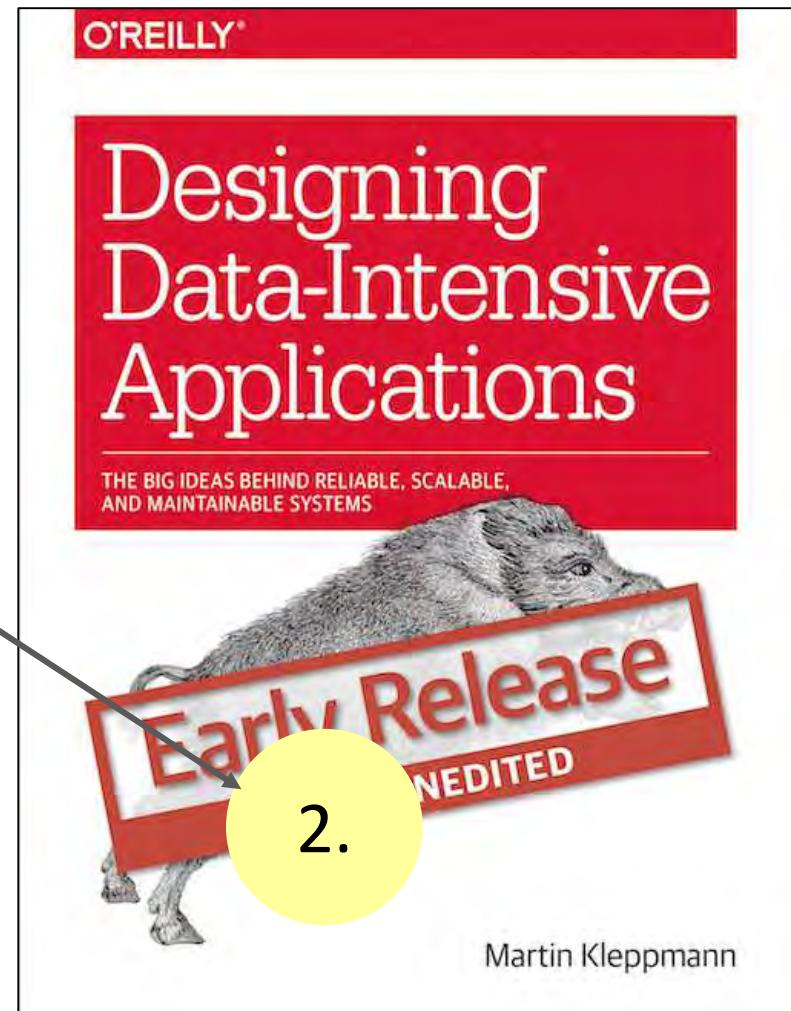
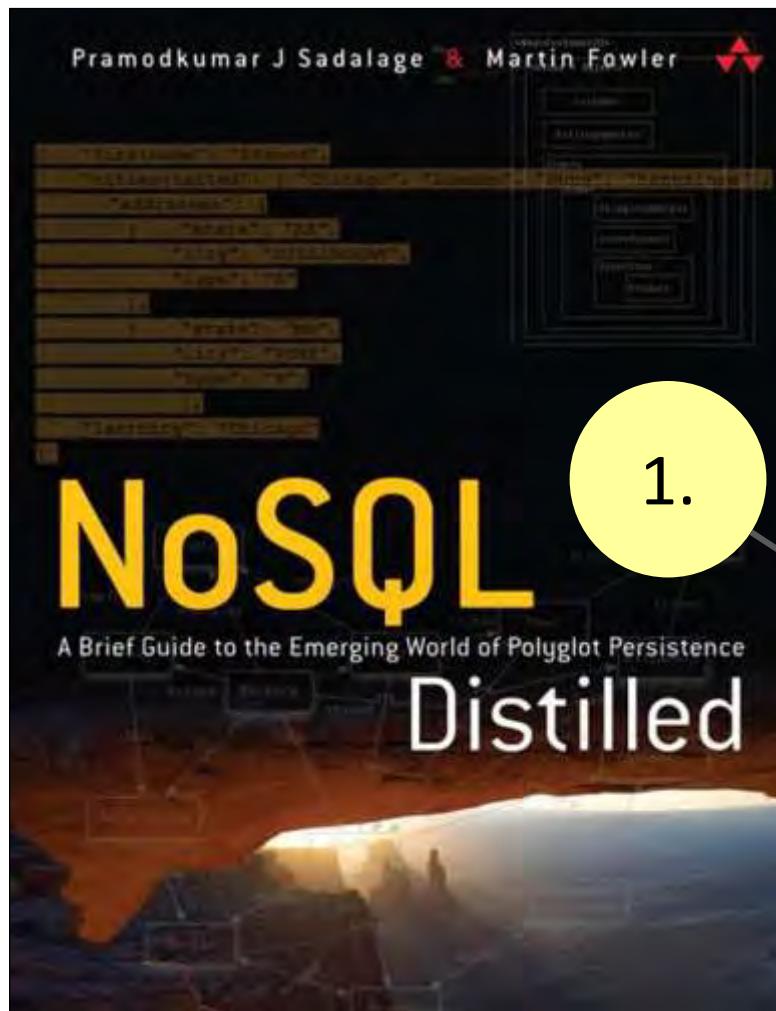


### Scalable Stream Processing: A Survey of Storm, Samza, Spark and Flink

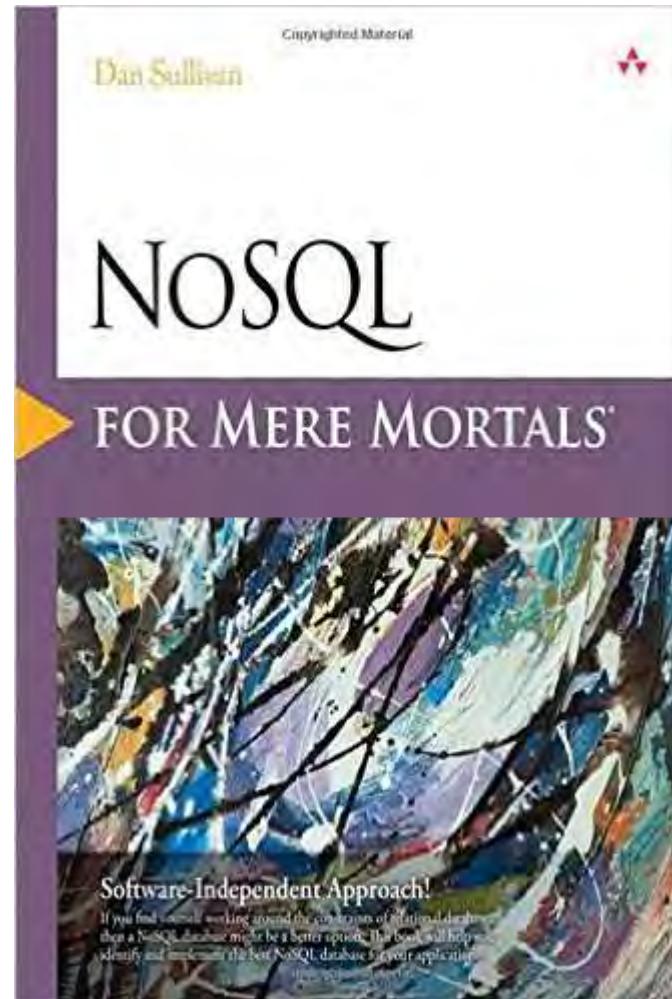
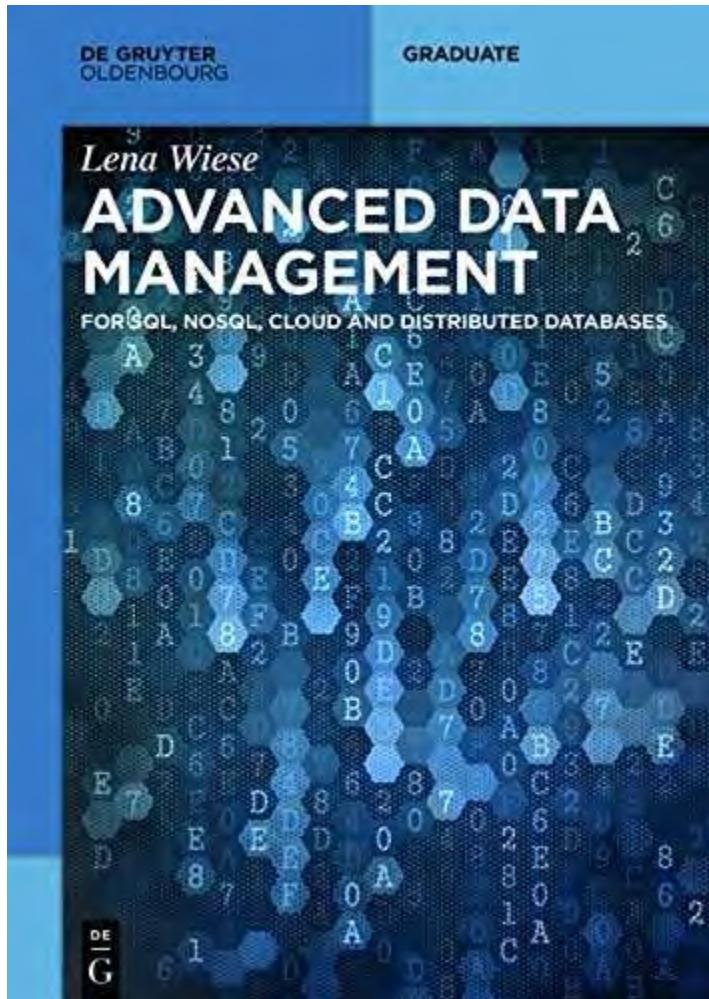
With this article, we would like to share our insights on real-time data processing we gained building Baqend. This is an updated version of our most recent stream processor survey which is another cooperation with the University of Hamburg (authors: Wolfram Wingerath, Felix Gessert, Steffen Friedrich 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!](http://blog.baqend.com)

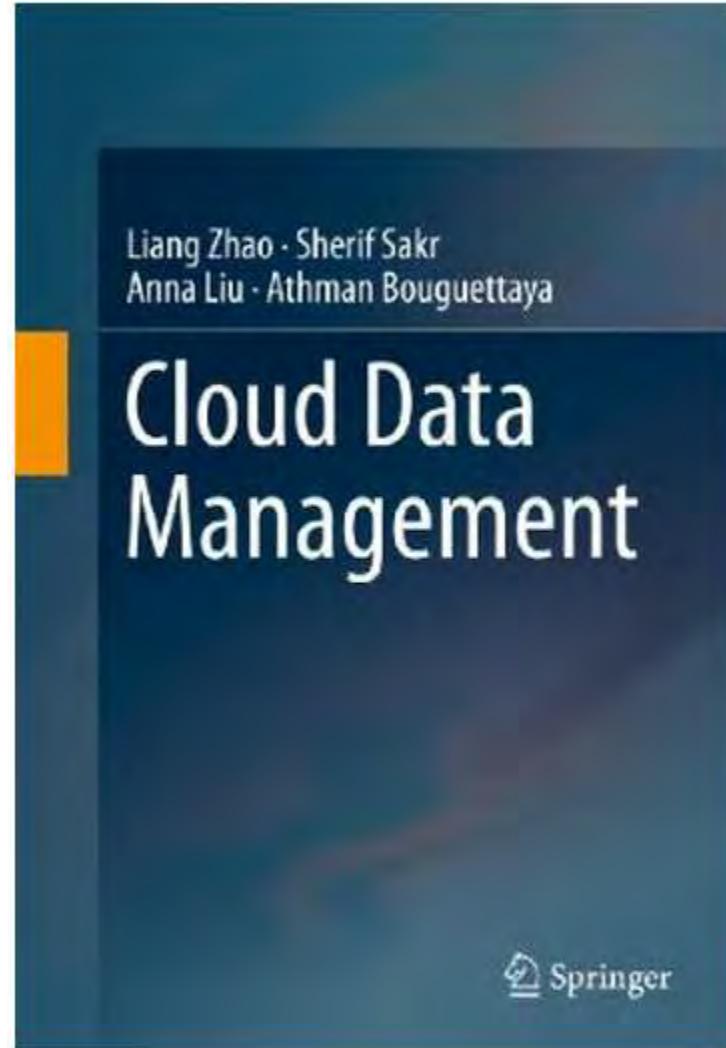
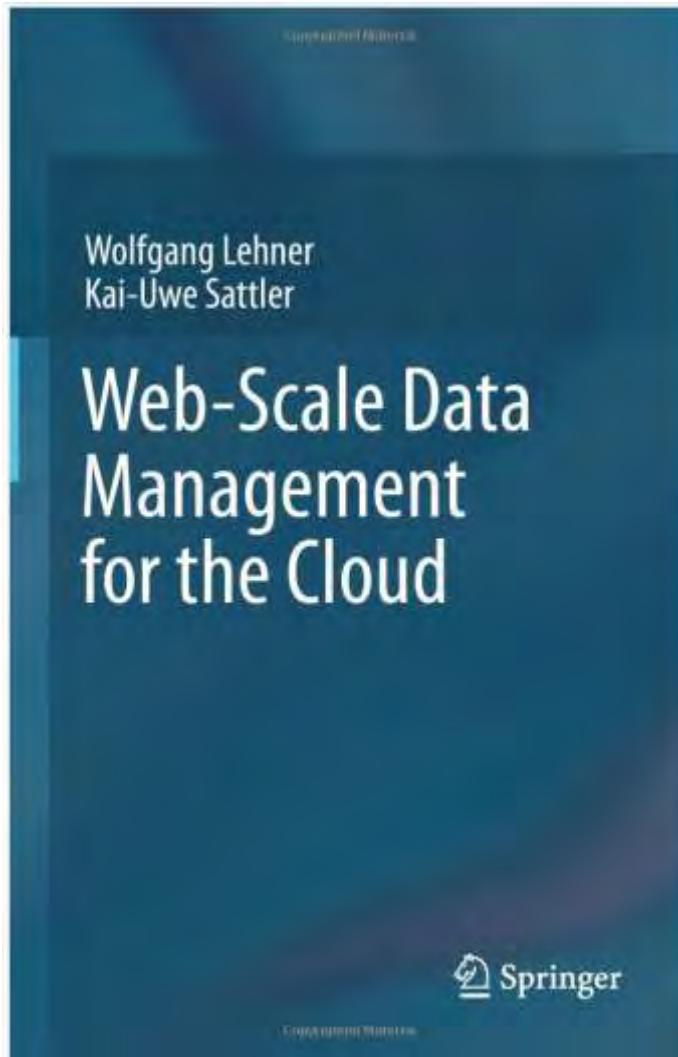
# Recommended Literature



# Recommended Literature



# Recommended Literature: Cloud-DBs



# Recommended Literature: Blogs



<http://medium.baqend.com/>



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



<https://aphyr.com/>



<http://muratbuffalo.blogspot.de/>



<http://www.nosqlweekly.com/>



<https://martin.kleppmann.com/>



<http://highscalability.com/>



<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

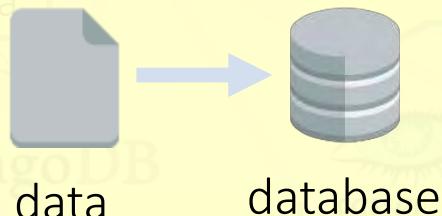
Current best practice



*Research Question:*

Can we automate the

mapping problem?

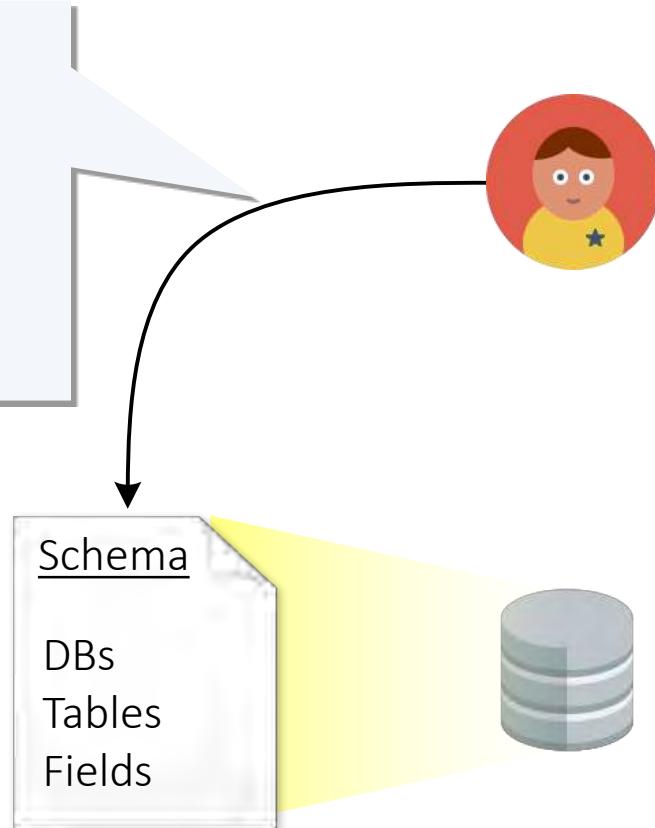


# Vision

Schemas can be annotated with requirements

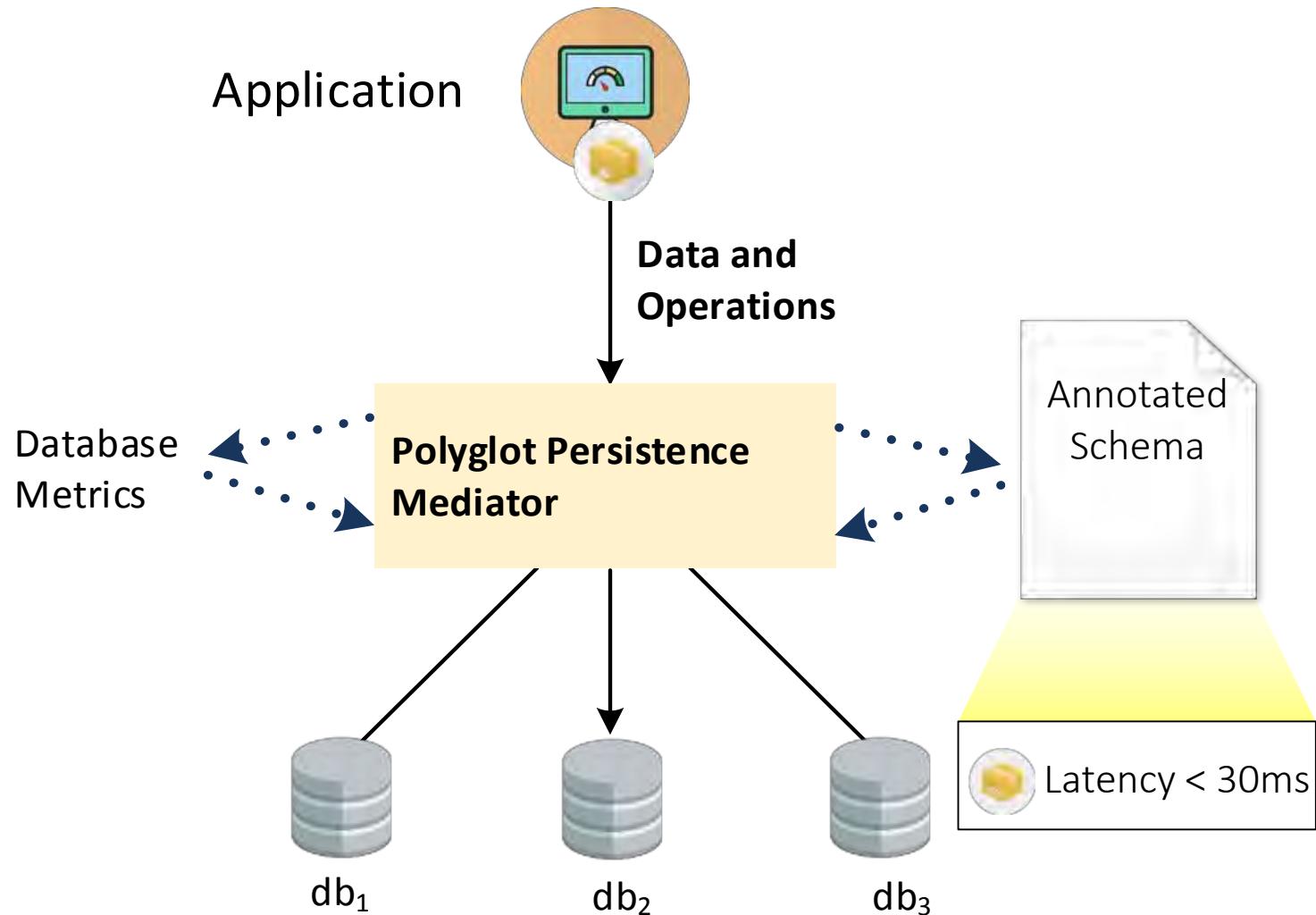


- Write Throughput > 10,000 RPS
- Read Availability > 99.9999%
- Scans = **true**
- Full-Text-Search = **true**
- Monotonic Read = **true**



# Vision

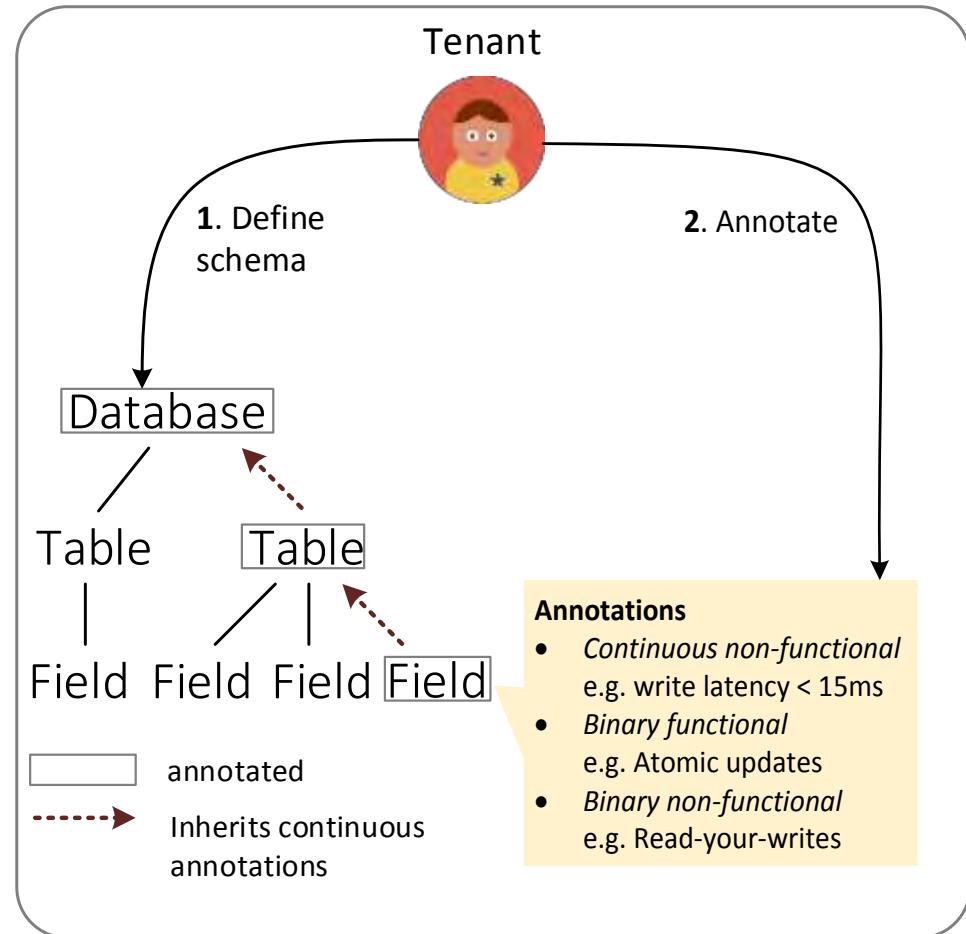
The Polyglot Persistence Mediator chooses the database



# Step I - Requirements

Expressing the application's needs

| 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 Scalability      | 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    |

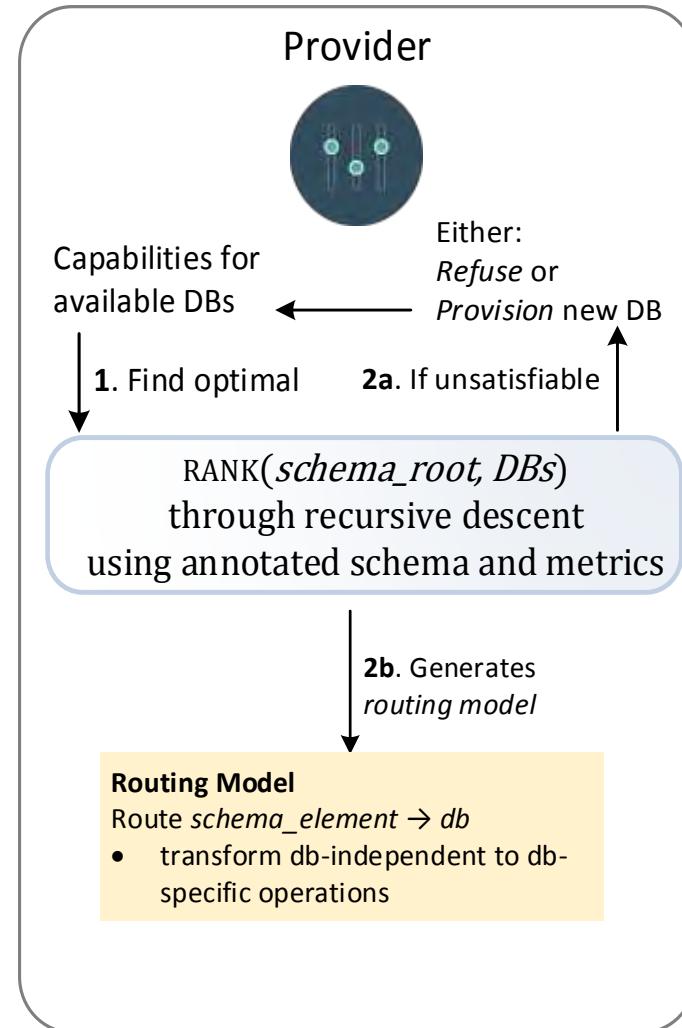


1 Requirements

# 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

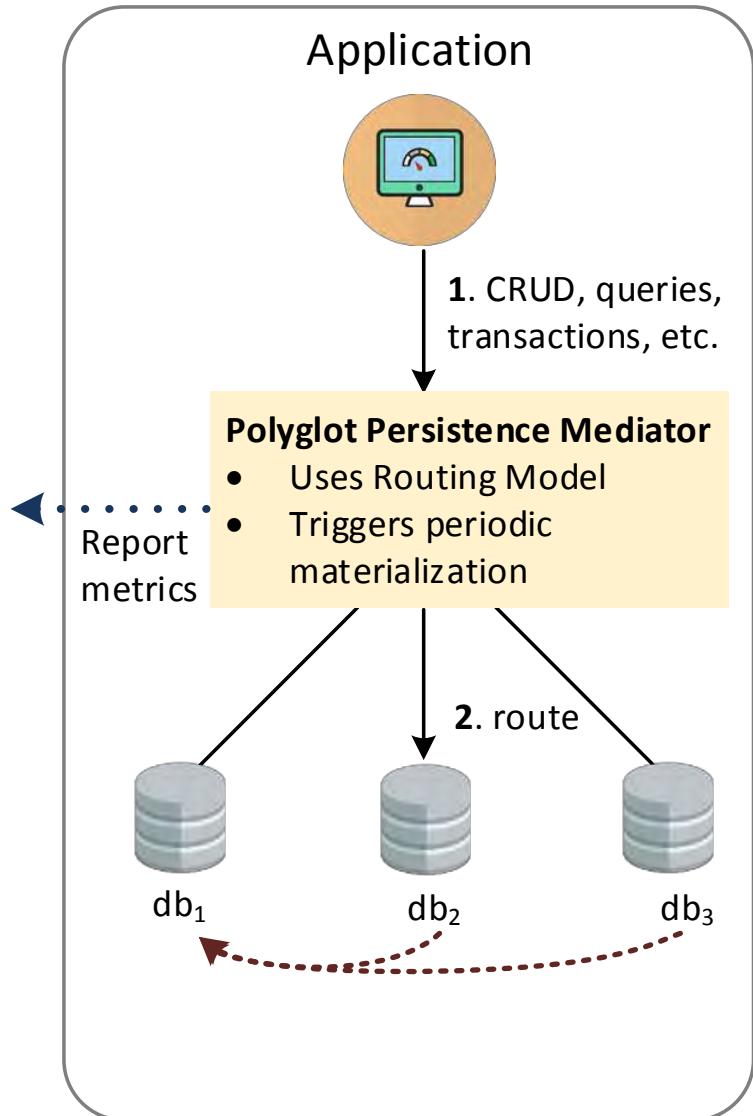


2 Resolution

# 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



3 Mediation

# 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

Article

A screenshot of a news article on the Hacker News website. The page has a light gray background with a white header bar. The header bar contains the text "Hacker News" in bold black font, followed by links: "new | threads | comments | show | ask | jobs | submissions". Below the header, there is a list of articles. The first article is titled "NoSQL Databases: A Survey and Decision Guidance (medium.com)". It shows "297 points" and was posted "9 days ago". The article is categorized as "past | web | 73 comments | in pocket speichern". At the bottom right of the article card, the text "read by 53,222" is displayed. Two large black arrows point from the left and right sides of the slide towards this text. The entire article card is enclosed in a thin gray border.

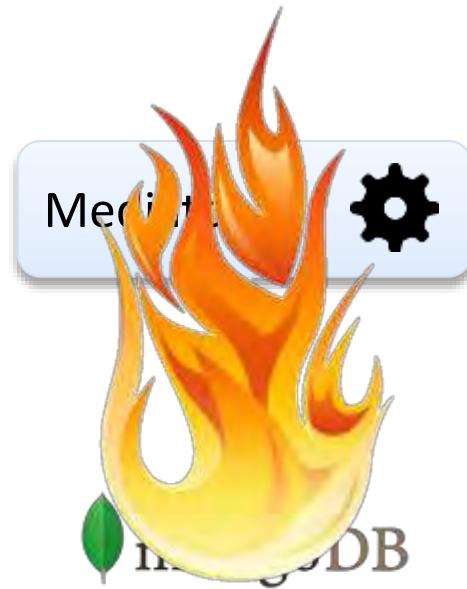
Counter

# Evaluation: News Article

Prototype built on ORESTES

**Scenario:** news articles with impression counts

**Objectives:** low-latency top-k queries, high-throughput counts, article-queries



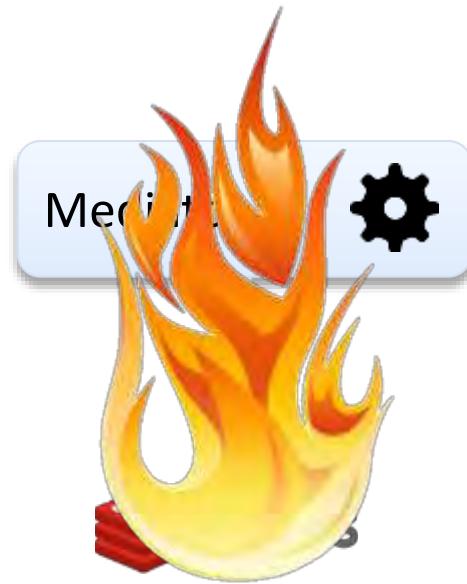
Counter updates kill performance

# Evaluation: News Article

Prototype built on ORESTES

**Scenario:** news articles with impression counts

**Objectives:** low-latency top-k queries, high-throughput counts, article-queries



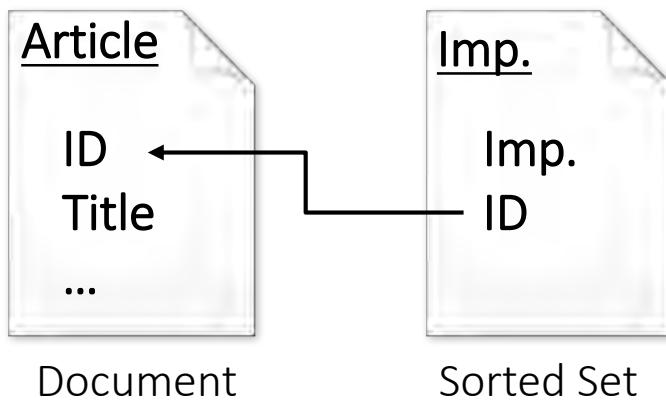
No powerful queries

# Evaluation: News Article

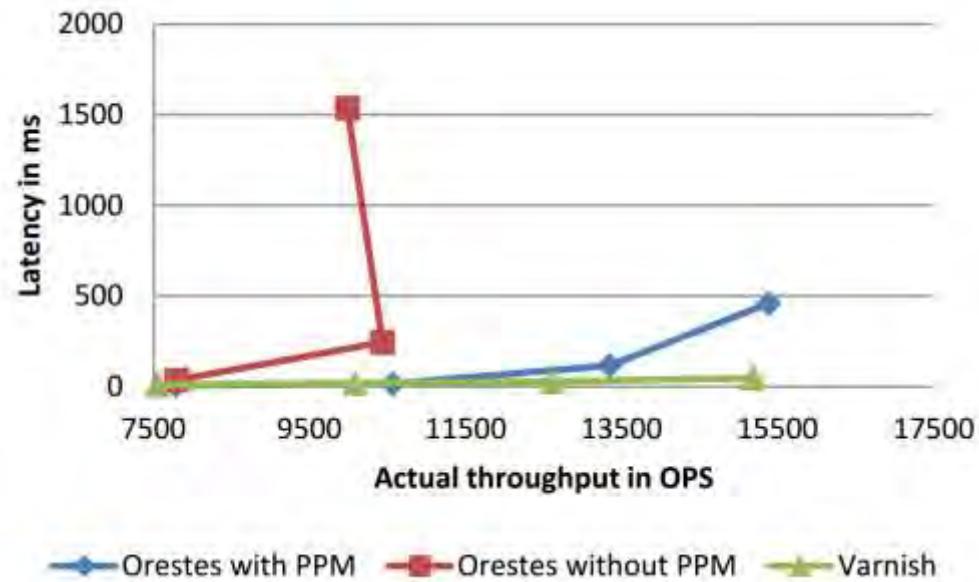
Prototype built on ORESTES

**Scenario:** news articles with impression counts

**Objectives:** low-latency top-k queries, high-throughput counts, article-queries

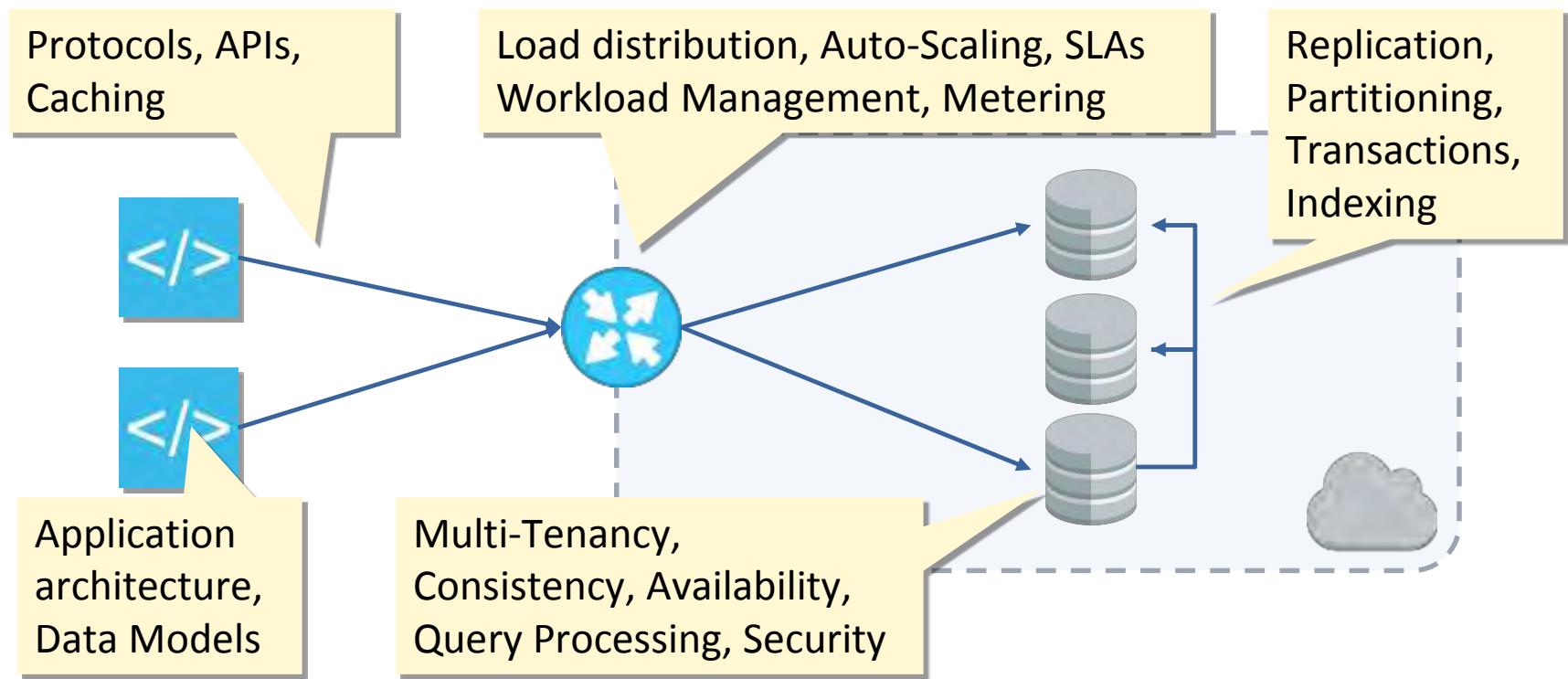


*Found Resolution*

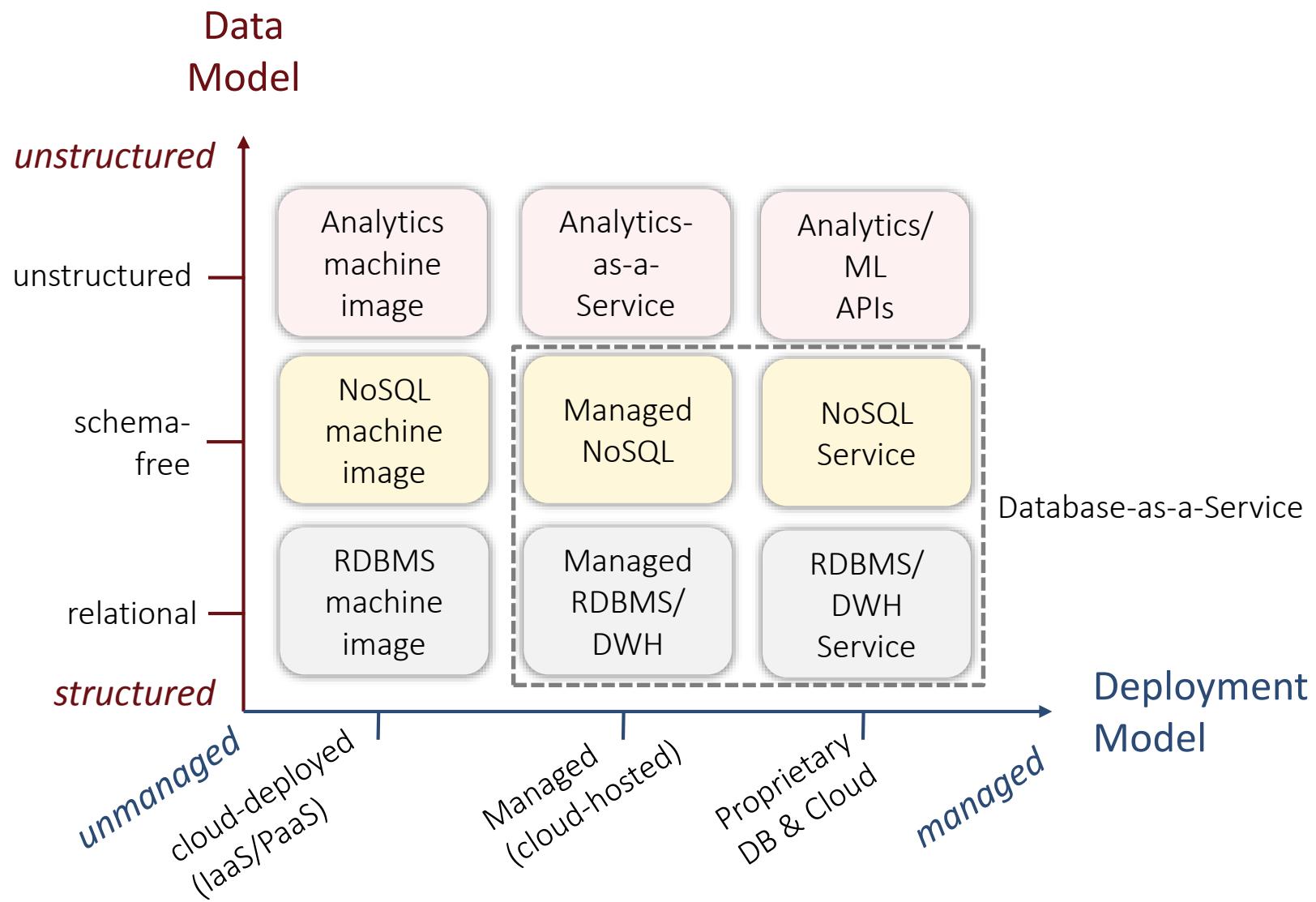


# Cloud Data Management

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

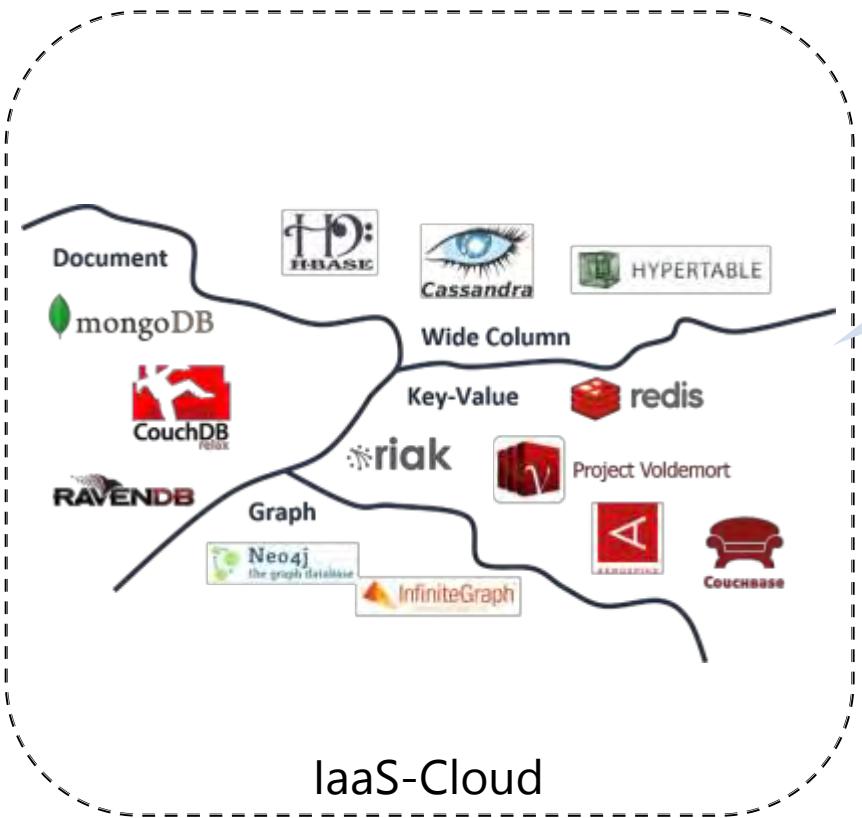


# Cloud-Database Models



# Cloud-Deployed Database

Database-image provisioned in IaaS/PaaS-cloud



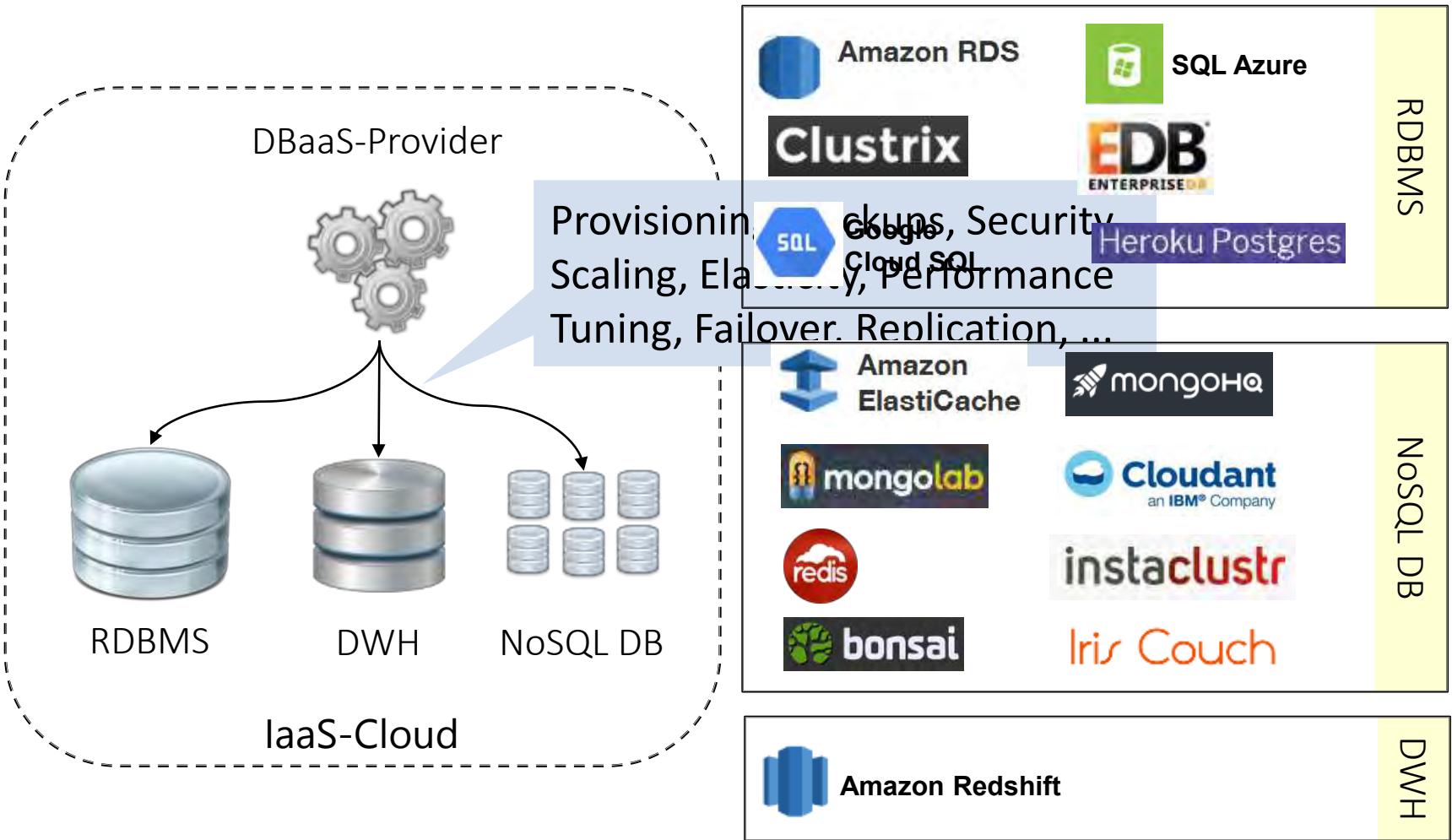
IaaS/PaaS deployment of  
database system

Does not solve:

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

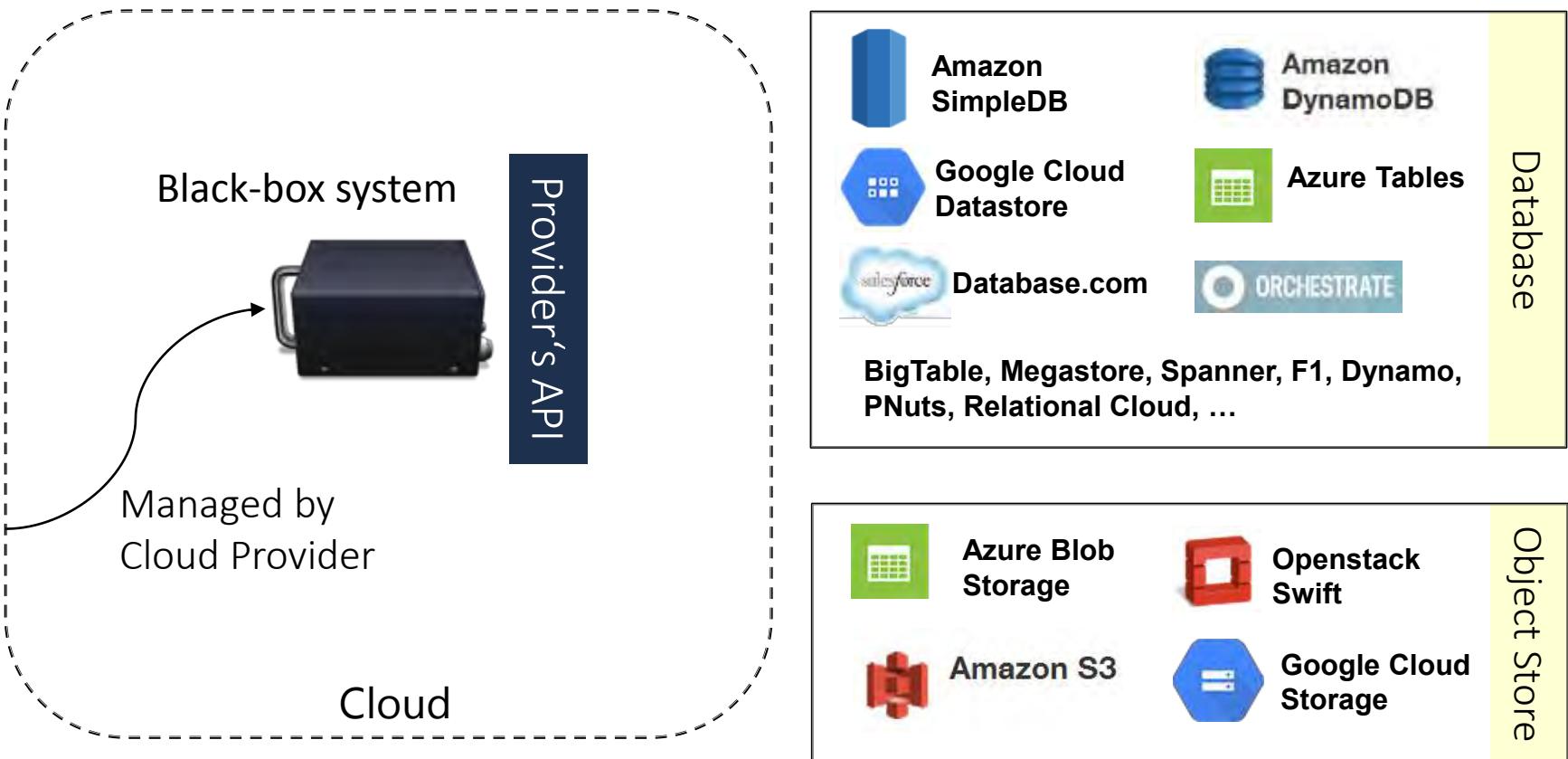
# Managed RDBMS/DWH/NoSQL DB

Cloud-hosted database



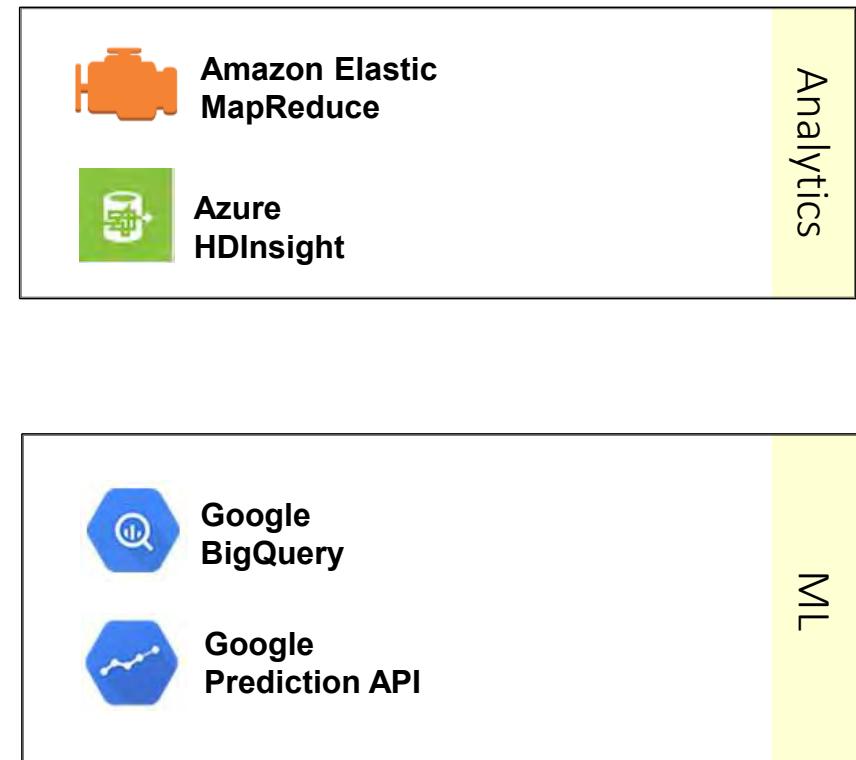
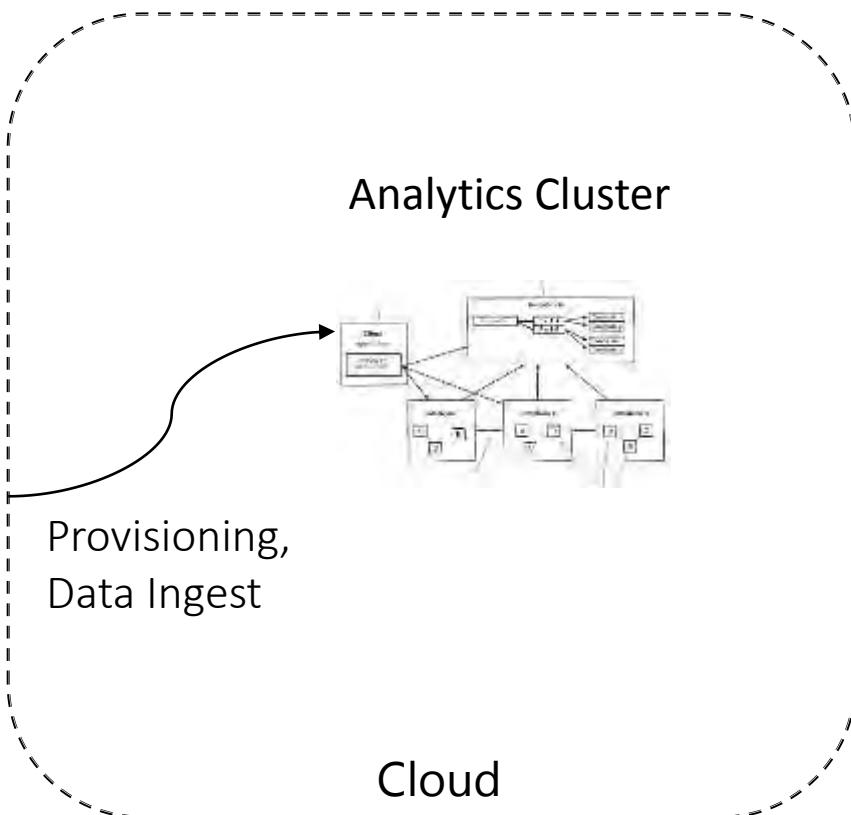
# Proprietary Cloud Database

Designed for and deployed in vendor-specific cloud environment



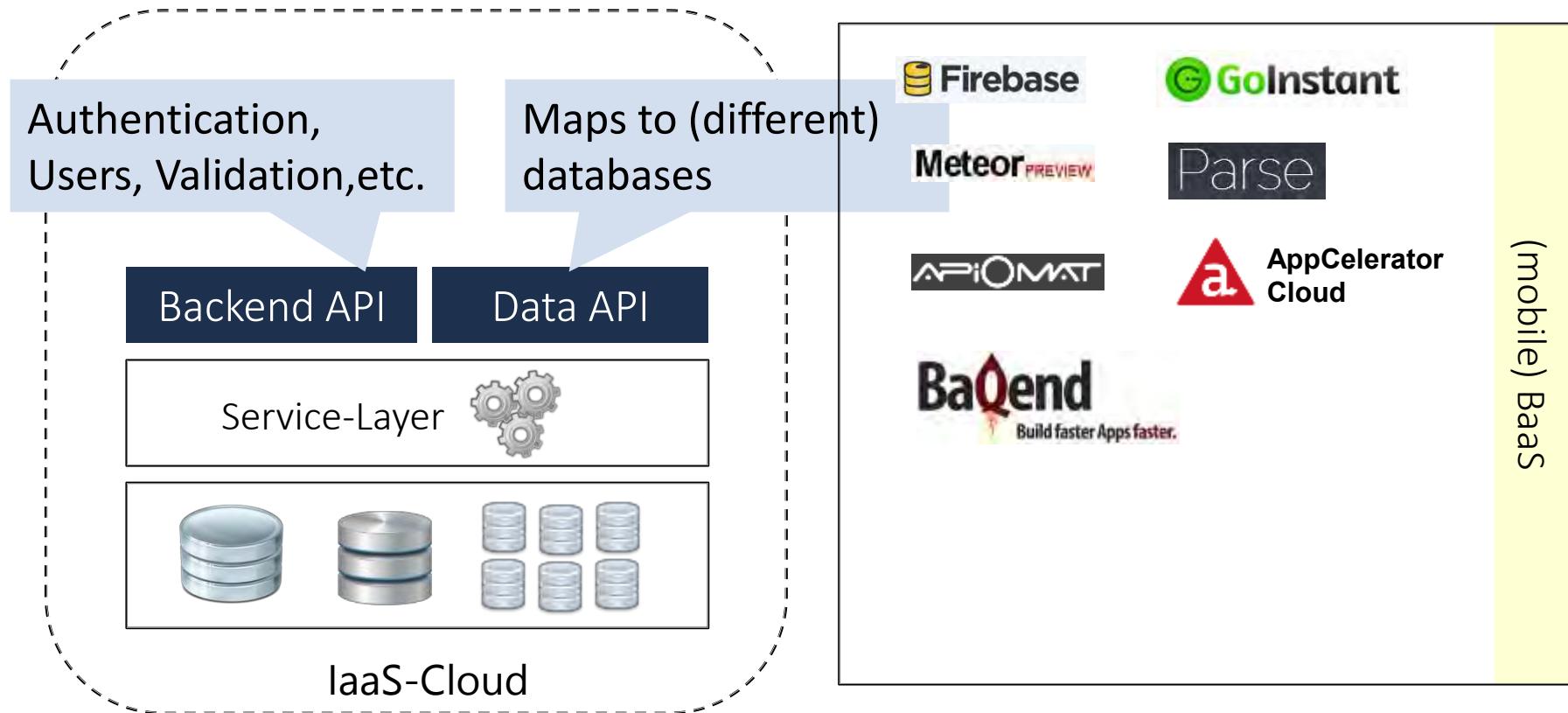
# Analytics-as-a-Service

Analytic frameworks and machine learning with service APIs



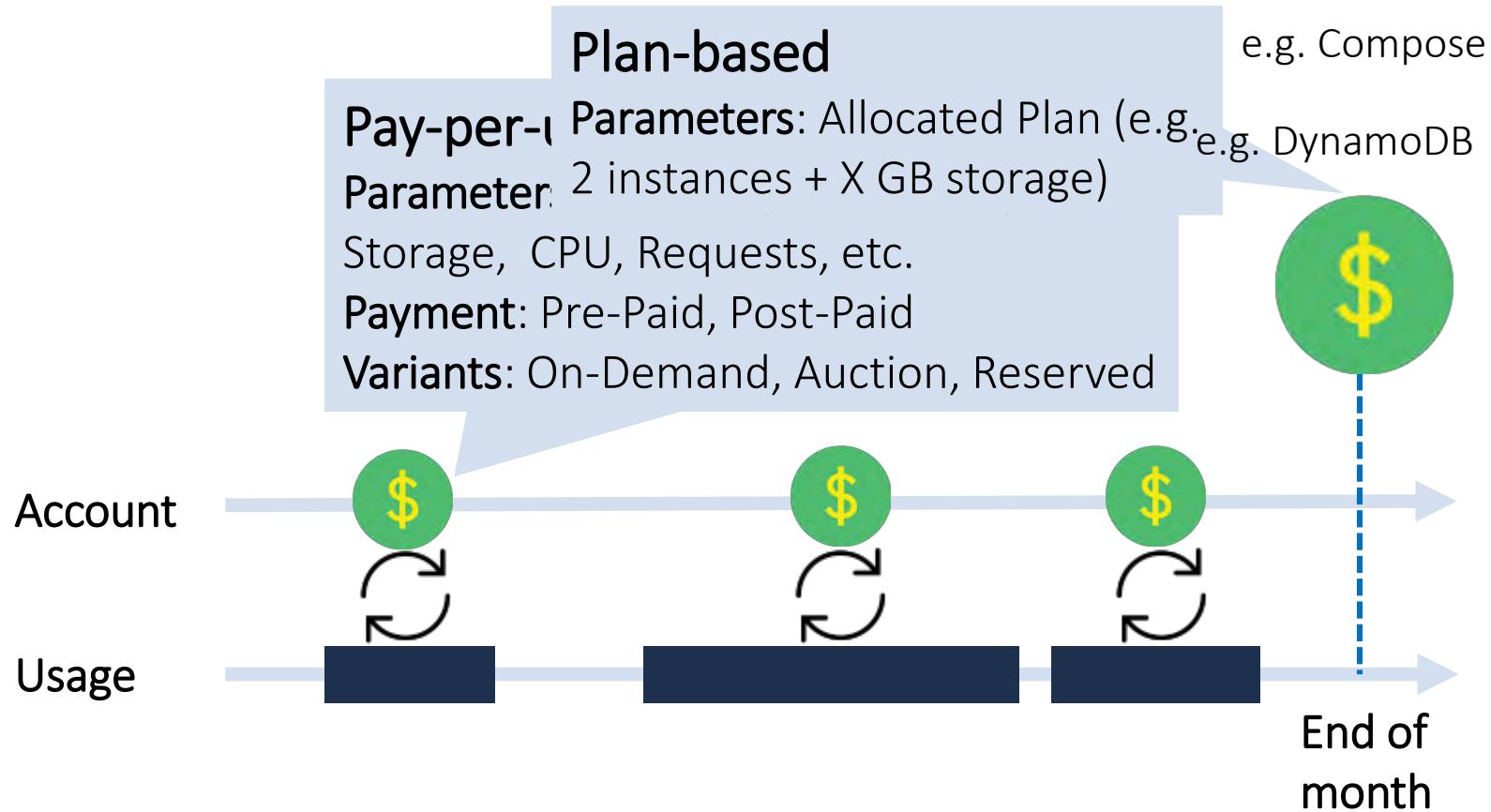
# Backend-as-a-Service

DBaaS with embedded custom and predefined application logic



# Pricing Models

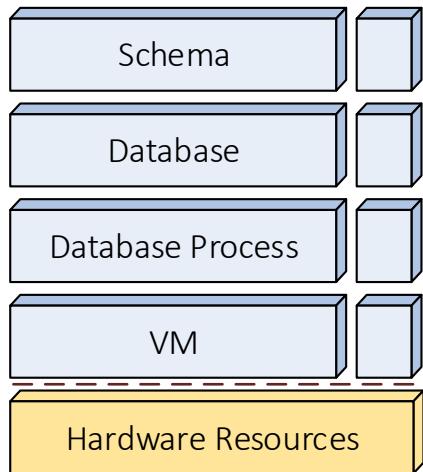
## Pay-per-use and plan-based



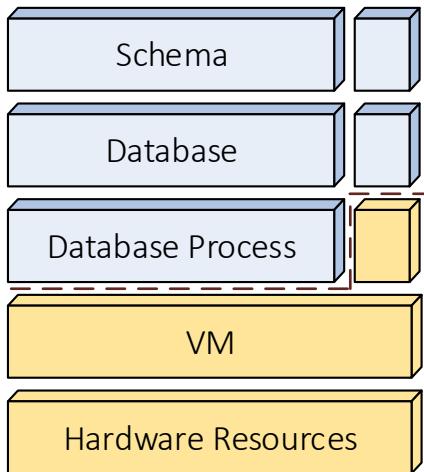
# Database-as-a-Service

## Approaches to Multi-Tenancy

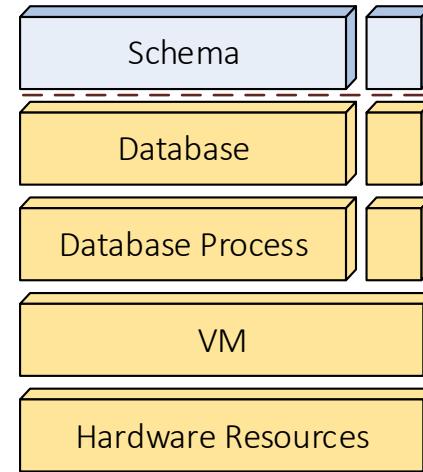
Private OS



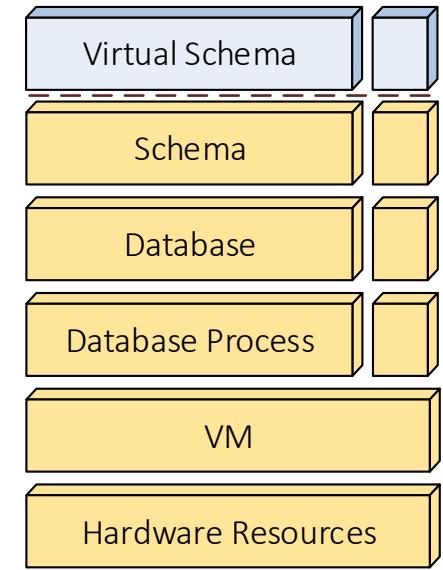
Private Process/DB



Private Schema



Shared Schema



e.g. Amazon RDS

e.g. Compose

e.g. Google DataStore

Most SaaS Apps



# Multi-Tenancy: Trade-Offs

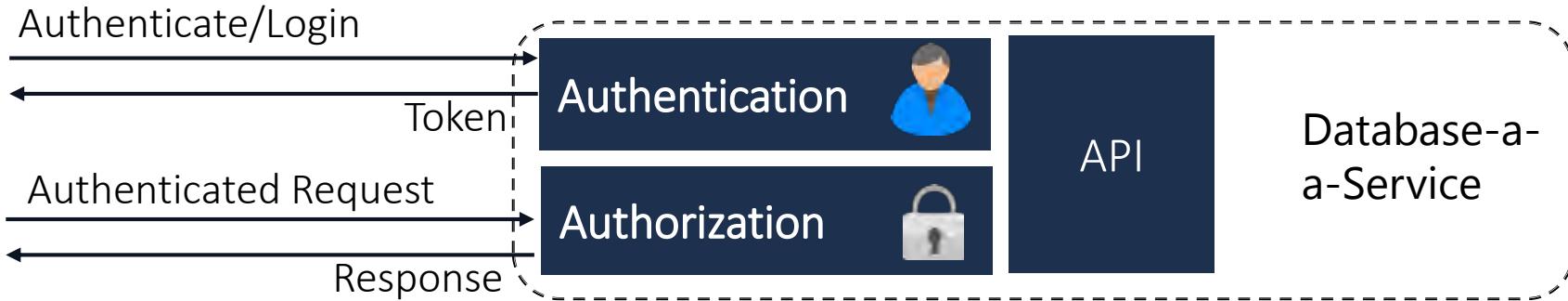
|                       | App.<br>indep. | Ressource<br>Util. | Isolation | Maintenance,<br>Provisioning |
|-----------------------|----------------|--------------------|-----------|------------------------------|
| Private OS            |                |                    |           |                              |
| Private<br>Process/DB |                |                    |           |                              |
| Private Schema        |                |                    |           |                              |
| Shared Schema         |                |                    |           |                              |



# Authentication & Authorization

## Checking Permissions and Identity

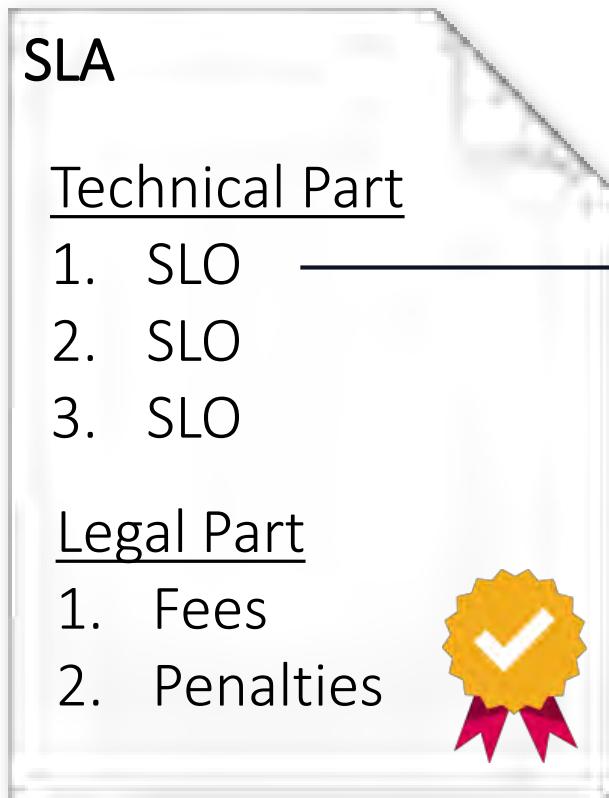
| Internal Schemes | External Identity Provider | Federated Identity (Single Sign On) |
|------------------|----------------------------|-------------------------------------|
| e.g. Amazon IAM  | e.g. OpenID                | e.g. SAML                           |



| 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



- **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

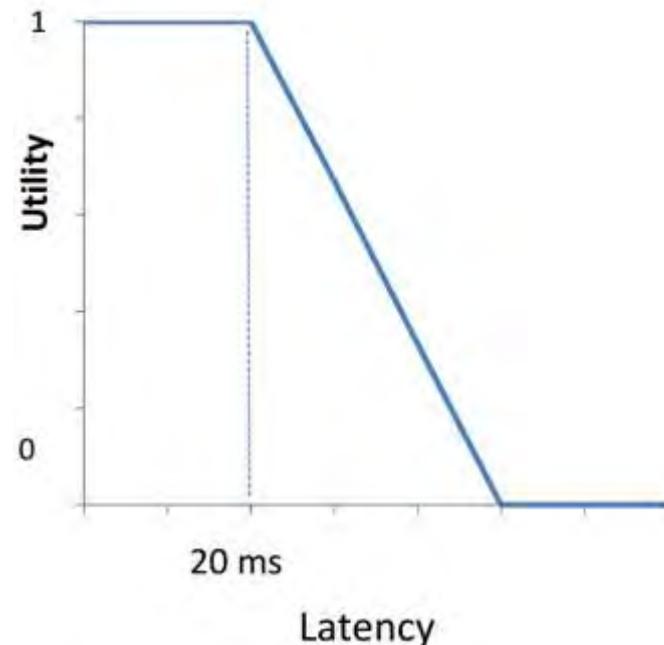
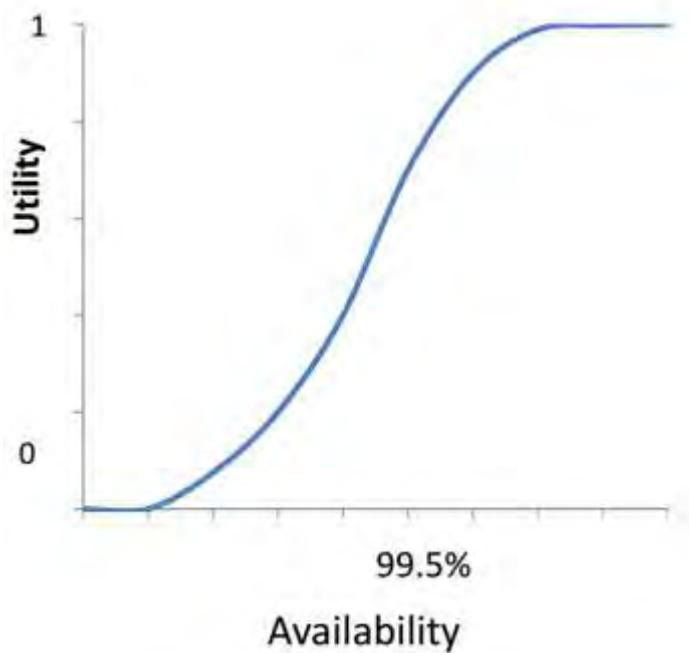


# Service Level Objectives

Making SLOs measurable through utilities

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

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

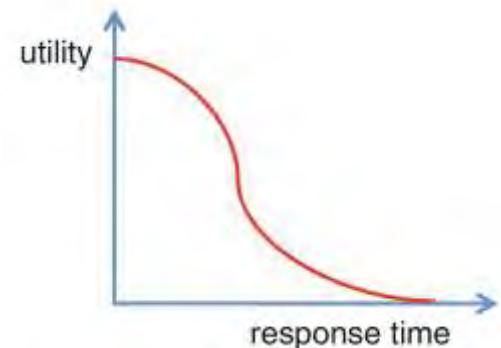


# Workload Management

## Guaranteeing SLAs

Typical approach:

Maximize:



response time

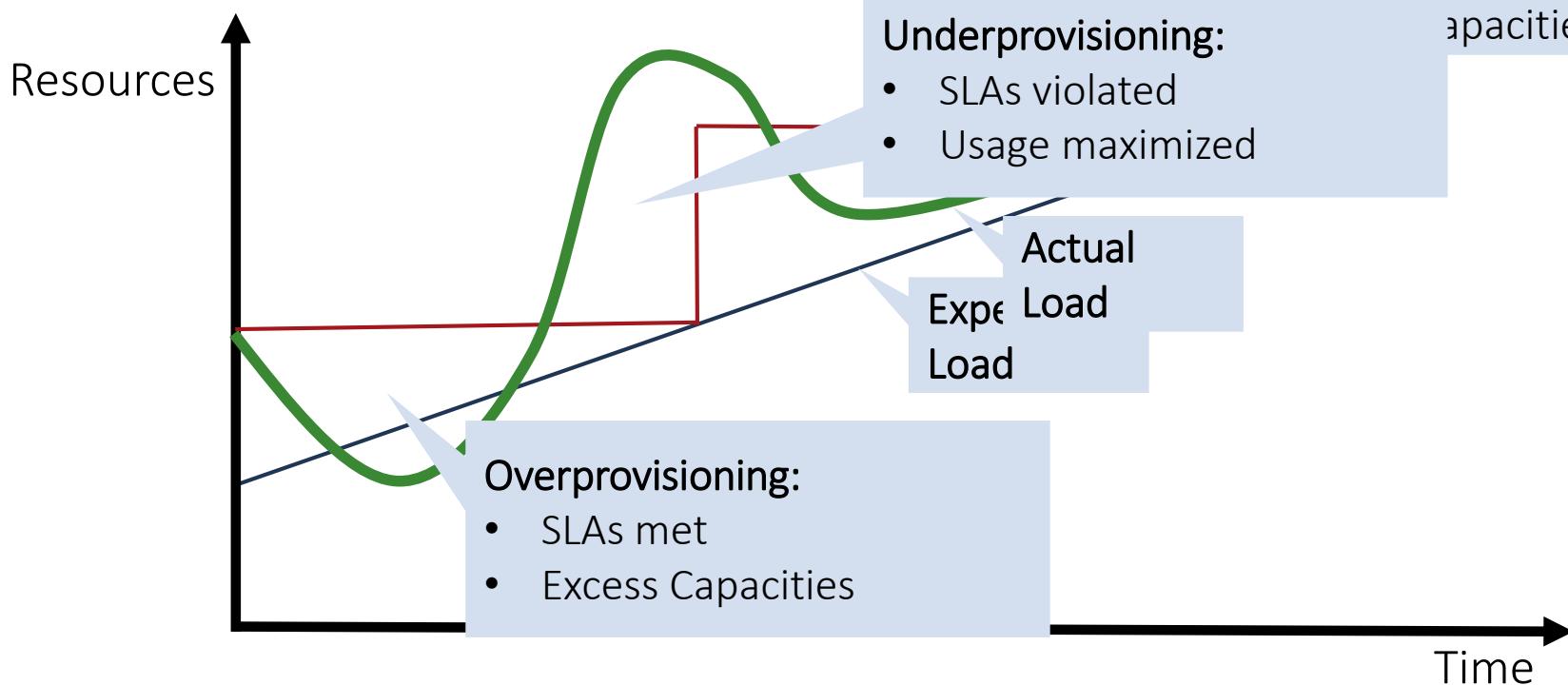


W. Lehner, U. Sattler "Web-scale Data Management for the Cloud"  
Springer, 2013

# Resource & Capacity Planning

From a DBaaS provider's perspective

**Goal:** minimize penalty and resource costs



# SLAs in the wild

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

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

# DBaaS Example

## Amazon RDS

### ▶ Relational Database Service

Step 1: Engine Selection  
Step 2: Production?  
Step 3: DB Instance Details  
Step 4: Additional Options

**Management Options**

Enable Automatic Backups:  Yes  No

The number of days for which automated backups are retained

MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Amazon Aurora, Amazon Neptune, Amazon DocumentDB

Period: 1 days

Automated backups are created if automated backups are enabled

Backup Window:  Select Window  No Preference

The weekly time range (in UTC) during which system maintenance can occur

Maintenance Window:  Select Window  No Preference

|                |                                |
|----------------|--------------------------------|
|                | 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

| No Index: Lookup only (!) by full table scan |                                       |                              |           |        |
|----------------------------------------------|---------------------------------------|------------------------------|-----------|--------|
| Partition Key                                | Row Key<br><i>(sortiert)</i>          | Timestamp<br><i>(autom.)</i> | Property1 | P      |
| intro.pdf                                    | v1.1                                  | 14/6/2013                    | ...       | ...    |
| intro.pdf                                    | v1.2                                  | 15/6/2013                    | ...       | ...    |
| p                                            | Hash-distributed to partition servers | 11/6/2013                    | ...       | Sparse |

REST API

Atomic "Entity-Group Batch Transaction" possible

Partition

Partition

▶ Similar to Amazon **SimpleDB** and **DynamoDB**

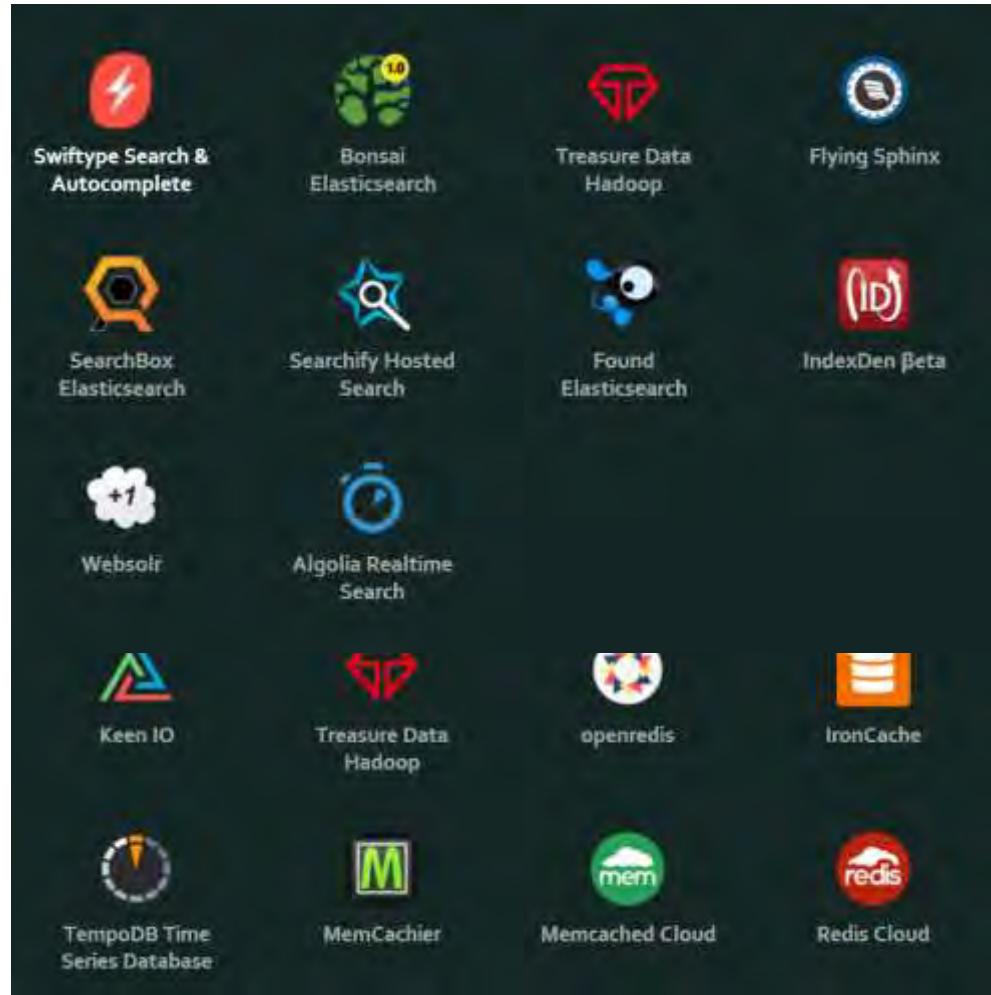
- Indexes all attributes
- Rich(er) queries
- Many Limits (size, RPS, etc.)

- 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'])
```



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

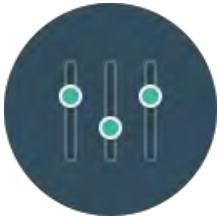
Dep

```
$ git
```

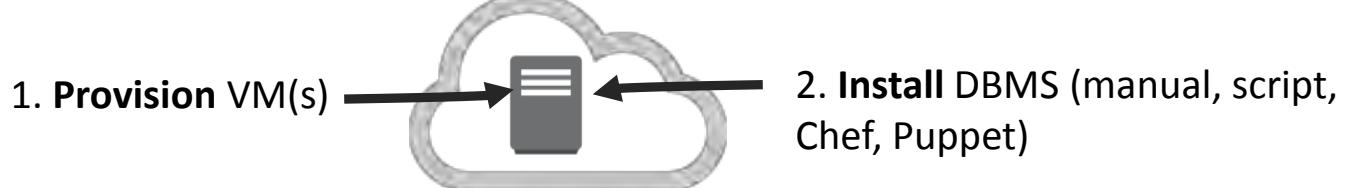
# Cloud-Deployed DB

An alternative to DBaaS-Systems

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



- ▶ Method I: DIY



- ▶ Method II: Deployment Tools

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

Login, cluster-size etc.



Amazon EC2



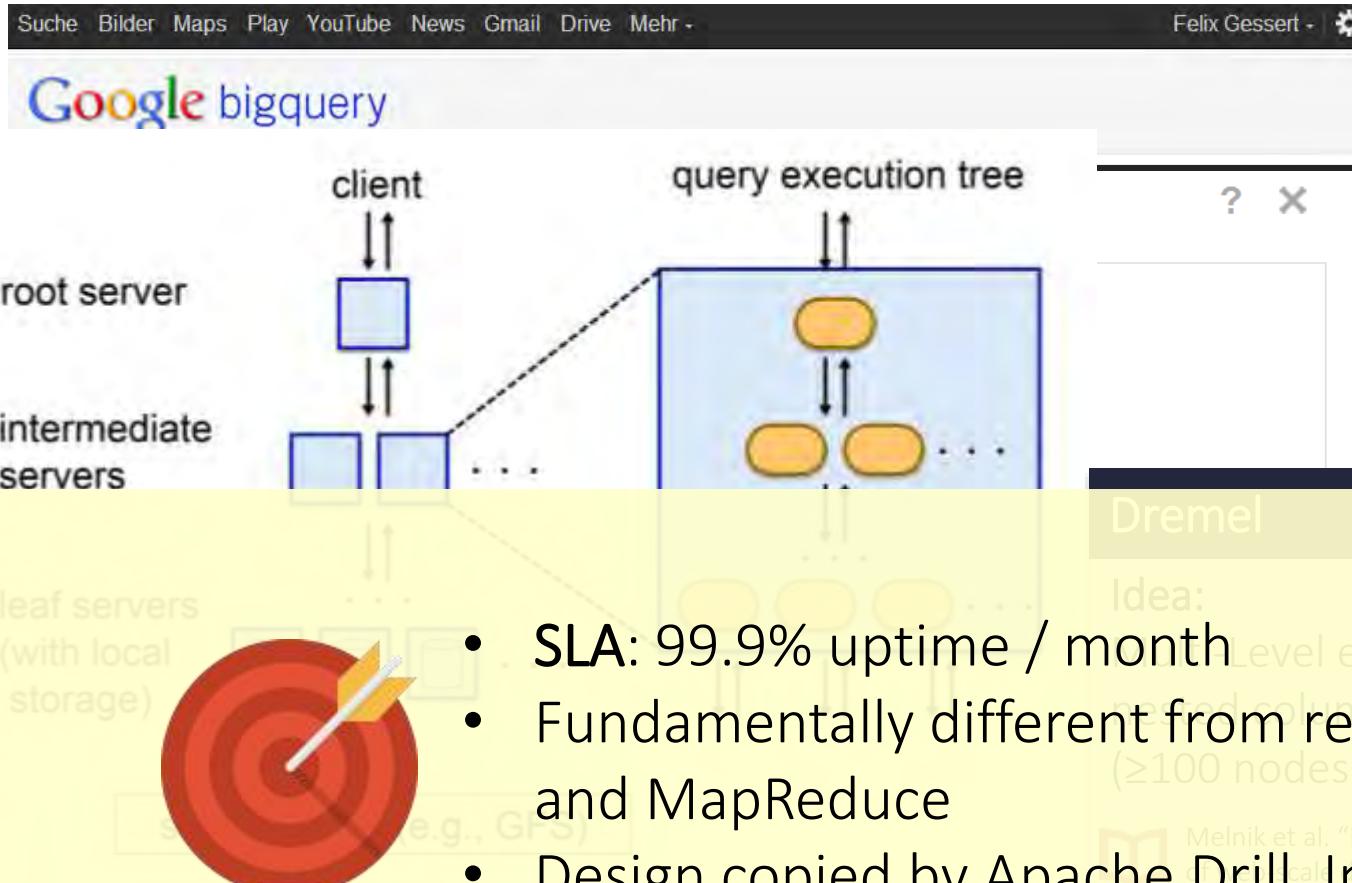
- ▶ Method III: Marketplaces



Google  
BigQuery

# Google BigQuery

- Idea: Web-scale analysis of nested data



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

# Managed NoSQL services

## Summary

|                | Model       | CAP | Scans        | Sec. Indices | Largest Cluster | Learn-ing | Lic.   | DBaaS       |
|----------------|-------------|-----|--------------|--------------|-----------------|-----------|--------|-------------|
| <b>HBase</b>   | Wide-Column | CP  | Over Row Key |              | ~700            | 1/4       | Apache | (EMR)       |
| <b>MongoDB</b> | Doc-ument   | CP  | yes          |              | >100<br><500    | 4/4       | GPL    |             |
| <b>Riak</b>    | Key-Value   | AP  |              |              | ~60             | 3/4       | Apache | (Softlayer) |

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

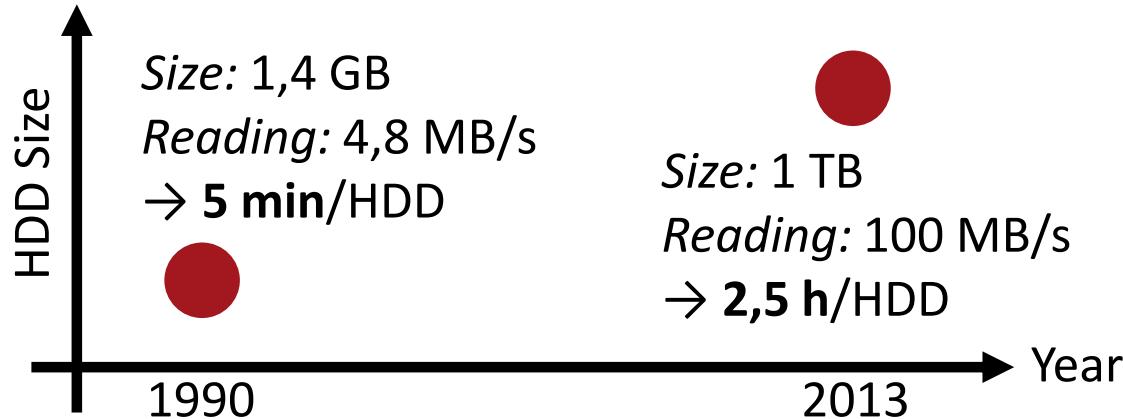
## Summary

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

# Big Data Frameworks



# Hadoop Distributed FS (CP)

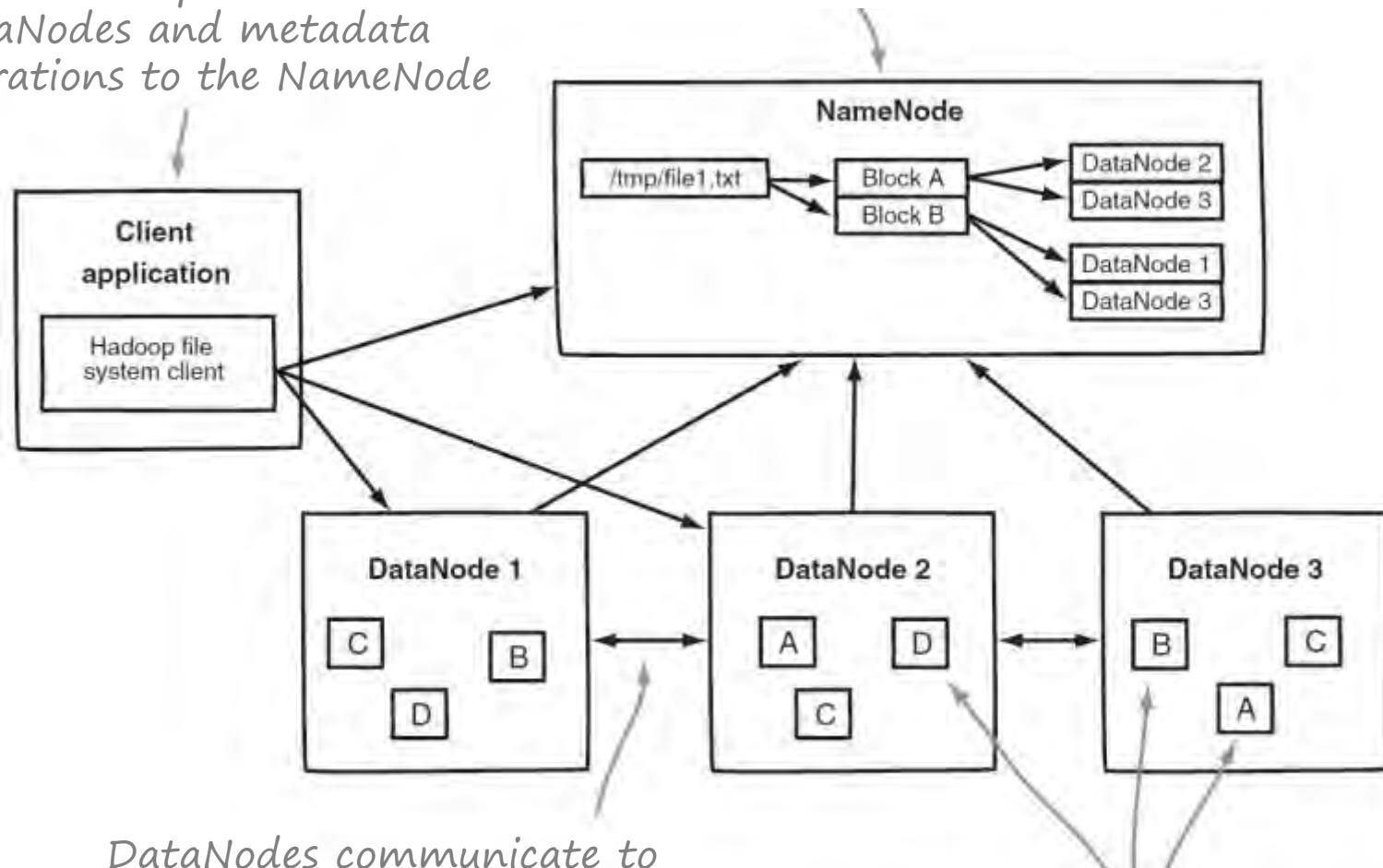


| HDFS        |
|-------------|
| Model:      |
| File System |
| License:    |
| Apache 2    |
| Written in: |
| Java        |

- ▶ 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

Sends data operations to DataNodes and metadata operations to the NameNode

Holds filesystem data and block locations in RAM



DataNodes communicate to perform 3-way replication

Files are split into blocks and scattered over DataNodes



# 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:

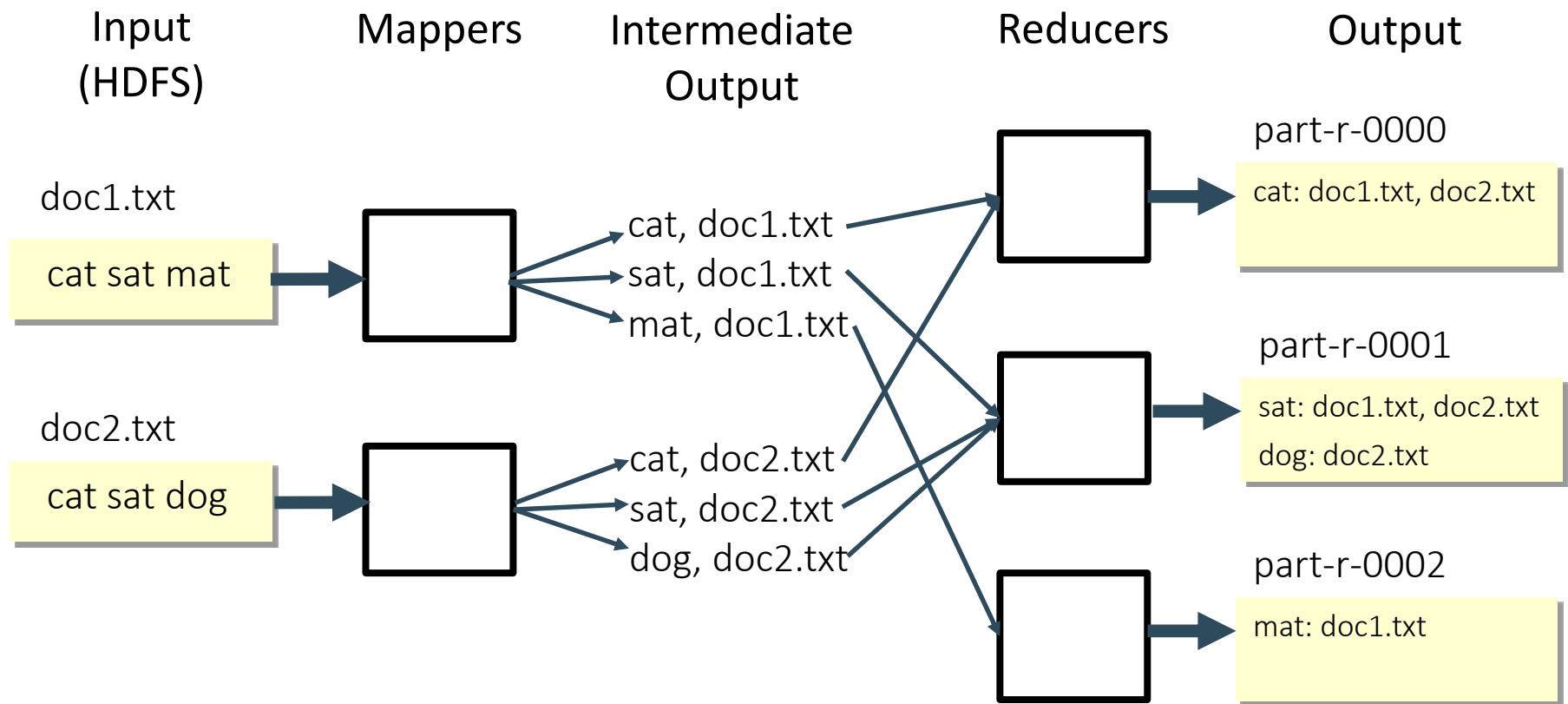
Apache 2

Written in:

Java

# MapReduce: Example

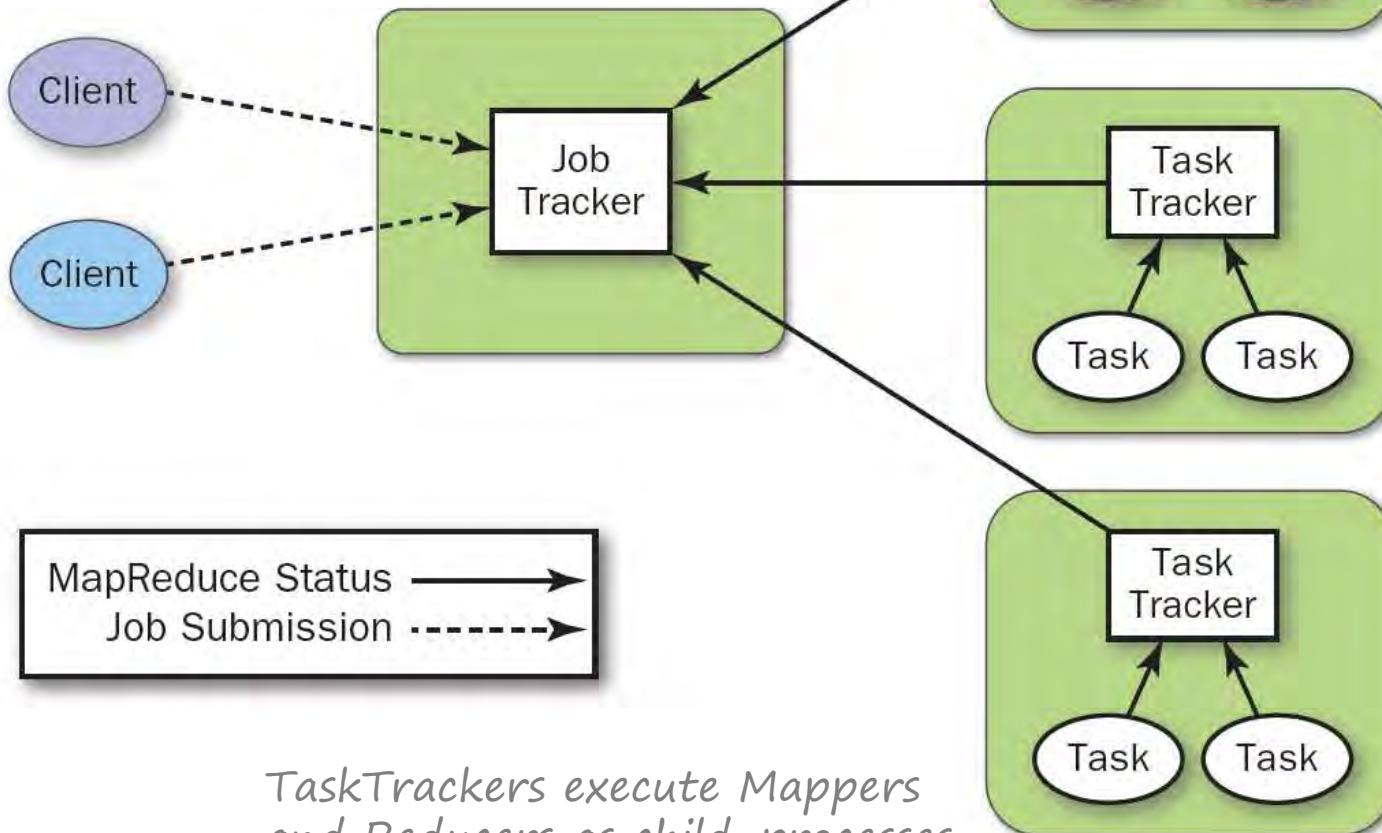
## Constructing a reverse-index



# Cluster Architecture

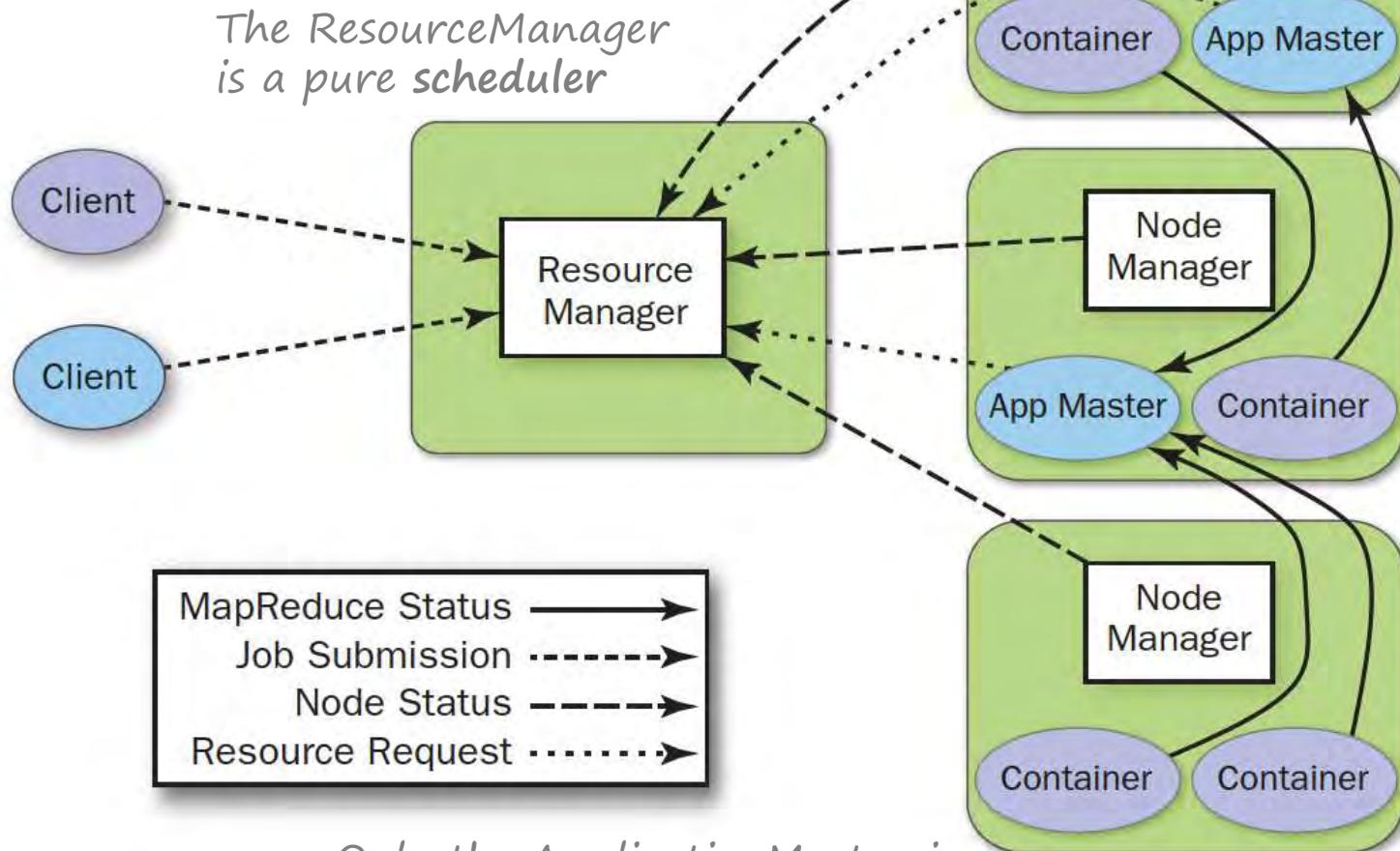
The client sends job and configuration to the Jobtracker

The JobTracker coordinates the cluster and assigns tasks



# Cluster Architecture

## YARN – Abstracting from MR



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

# Summary: Hadoop Ecosystem

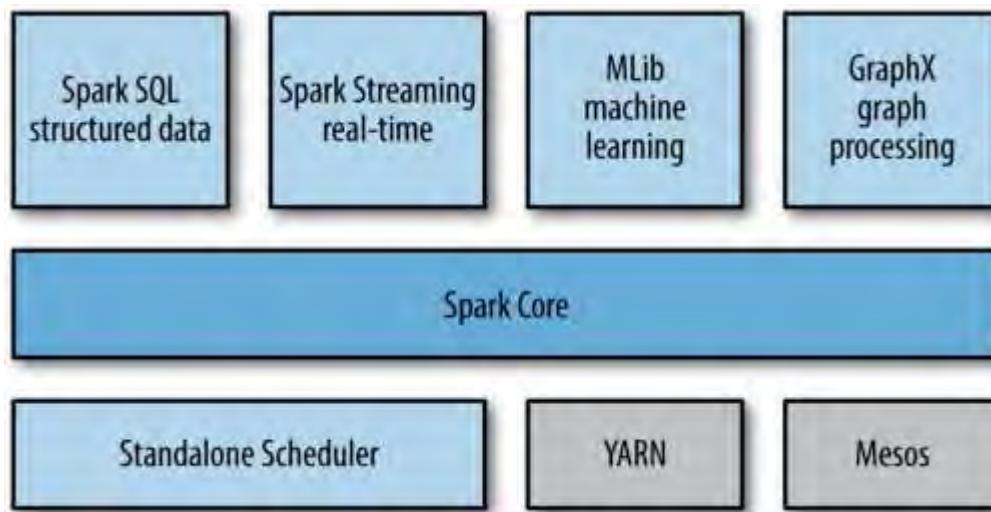


- ▶ **Hadoop:** Ecosystem for Big Data Analytics
- ▶ **Hadoop Distributed File System:** scalable, shared-nothing file system for throughput-oriented workloads
- ▶ **Map-Reduce:** Paradigm for performing scalable distributed batch analysis
- ▶ Other Hadoop projects:
  - **Hive:** SQL(-dialect) compiled 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                      |

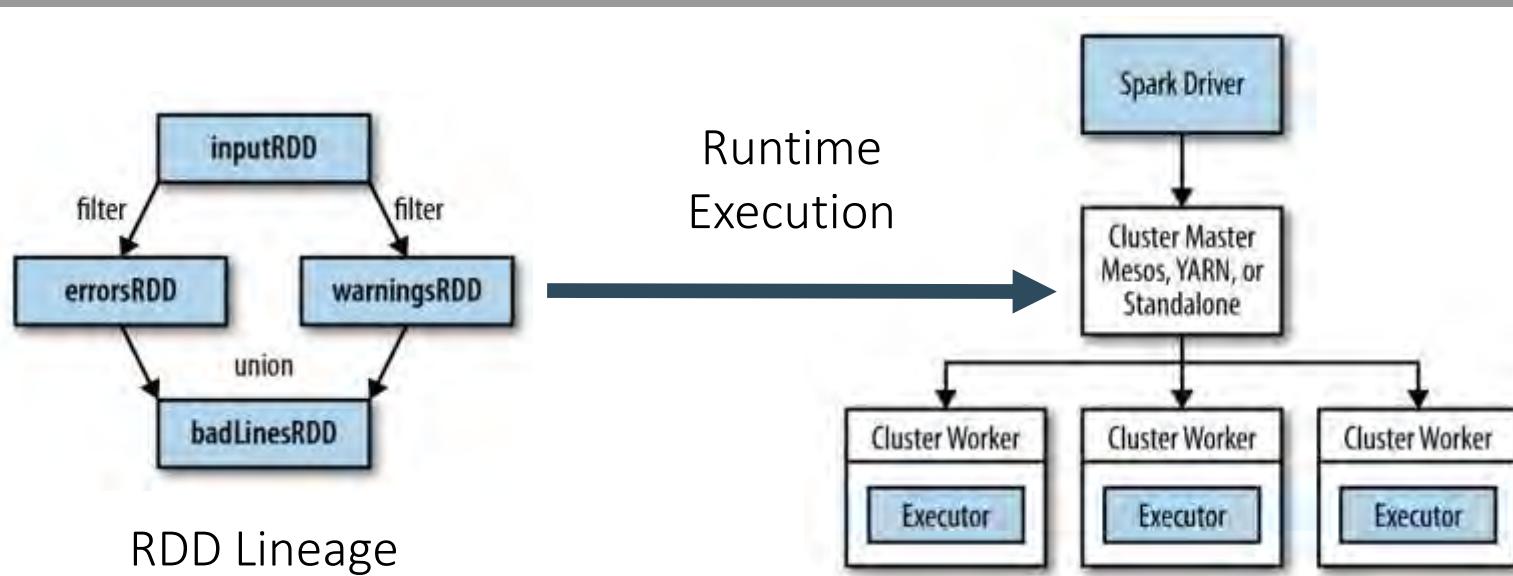


# Spark

## Example RDD Evaluation

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

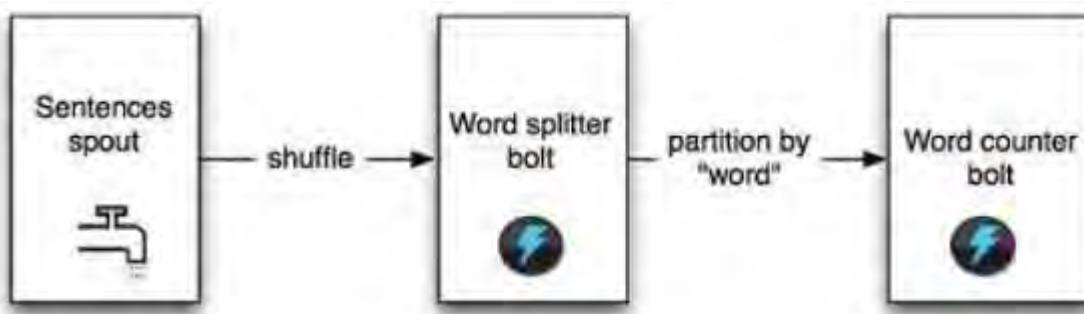
```
errors = sc.textFile("log.txt").filter(lambda x: "error" in x)
warnings = inputRDD.filter(lambda x: "warning" in x)
badLines = errorsRDD.union(warningsRDD).count()
```



# Storm

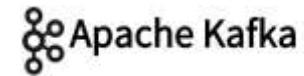
- ▶ Distributed Stream Processing Framework
- ▶ Topology is a DAG of:
  - **Spouts:** Data Sources
  - **Bolts:** Data Processing Tasks
- ▶ Cluster:
  - Nimbus (Master) ↔ Zookeeper ↔ Worker

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



# 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



## Kafka

Model:

Distributed Pub-Sub-System

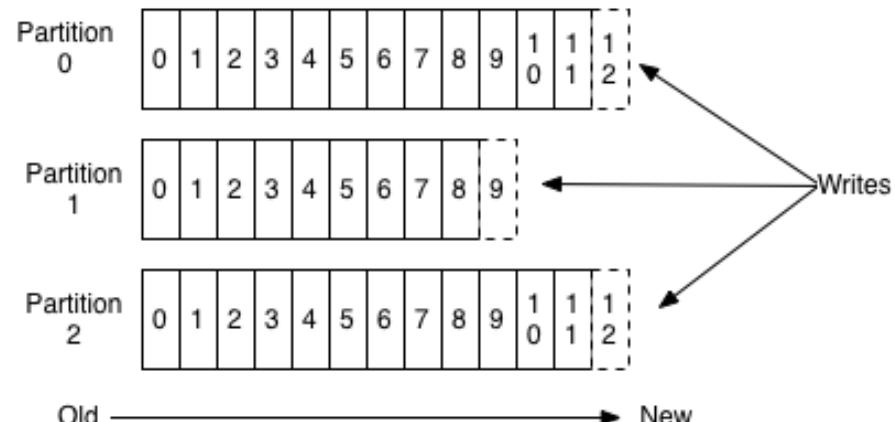
License:

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“