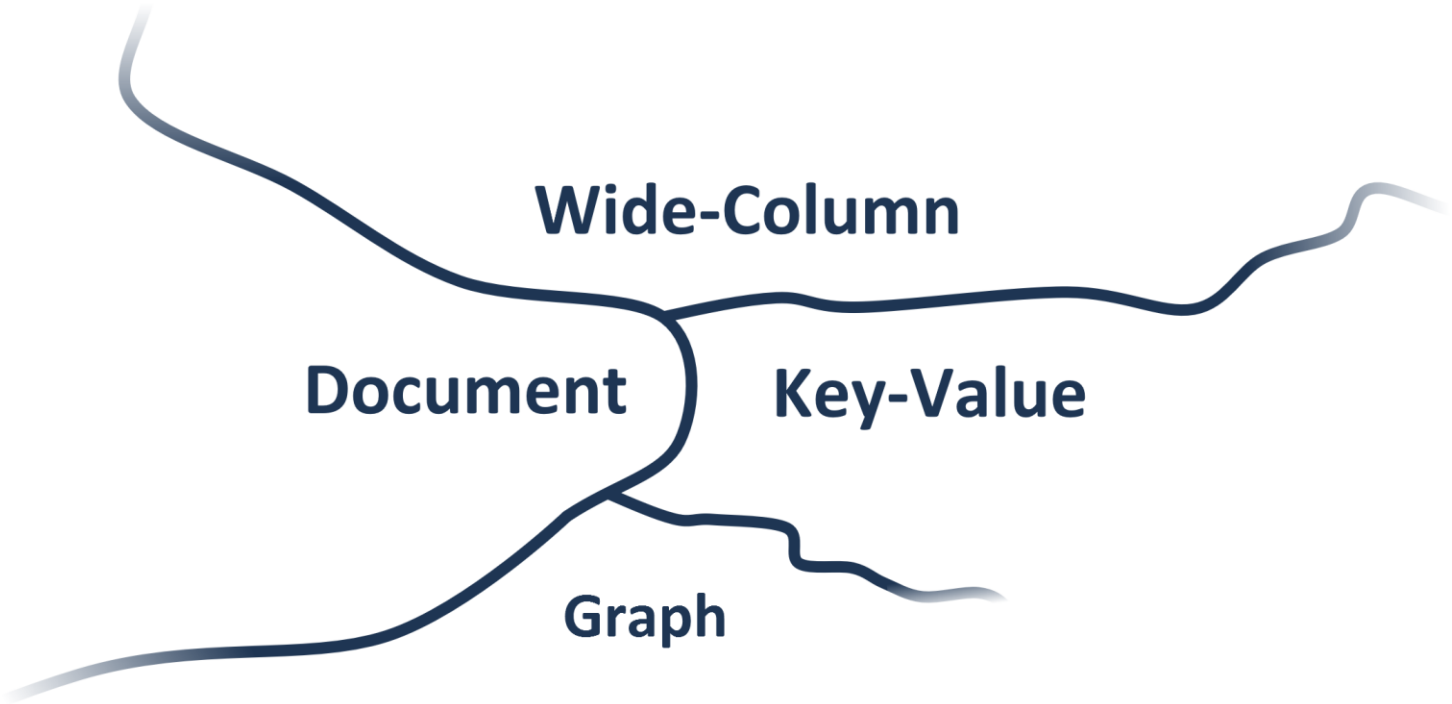
# Aktuelle Datenbanktechnologien

## in Theorie und Praxis



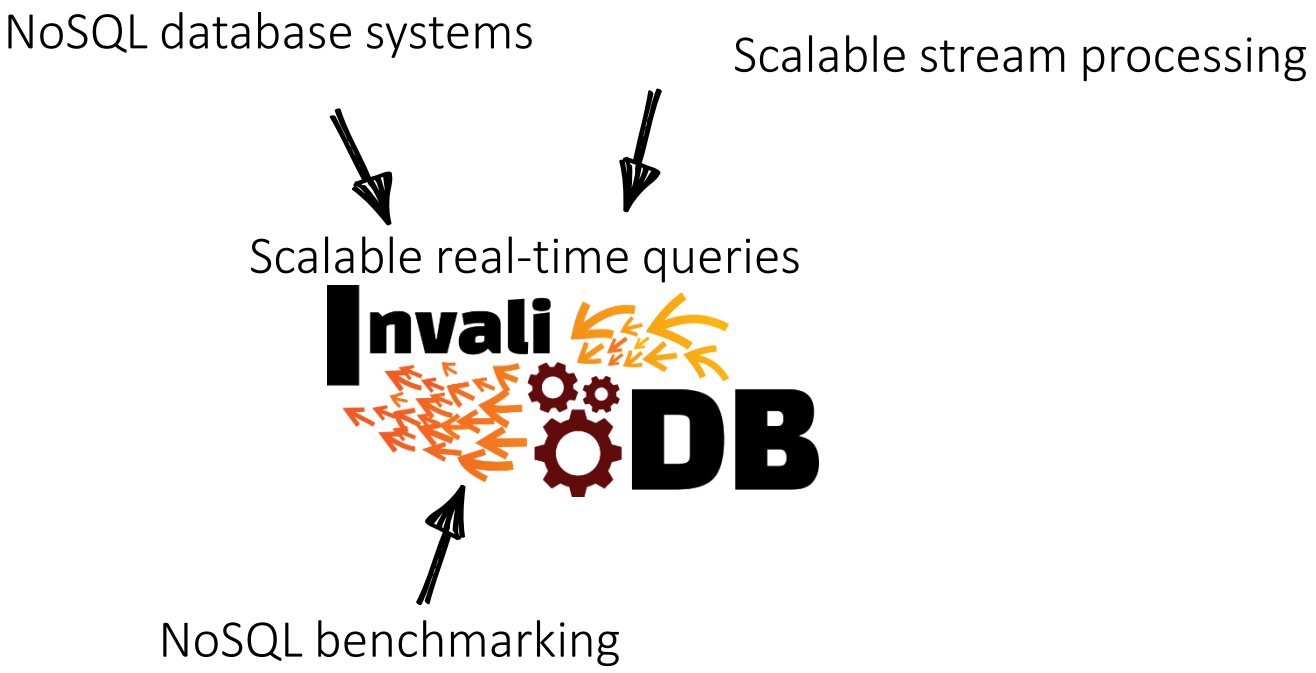

Wolfram Wingerath
wingerath@informatik.uni-hamburg.de

6. Dezember, 2016, Arbeitskreis Datenbanken, München

# About me

## Wolfram Wingerath

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



# Kudos to Felix Gessert

## For providing slides

- „Skalierbare NoSQL- und Cloud-Datenbanken in Forschung und Praxis“ http://www.btw-2015.de/?nosql
- „NoSQL Data Stores in Research and Practice“ https://www.percona.com/live/plam16/sessions/nosql-data-stores-research-and-practice

PhD research project →


More at blog.baqend.com!

BaQend
Build faster Apps faster.

# Outline

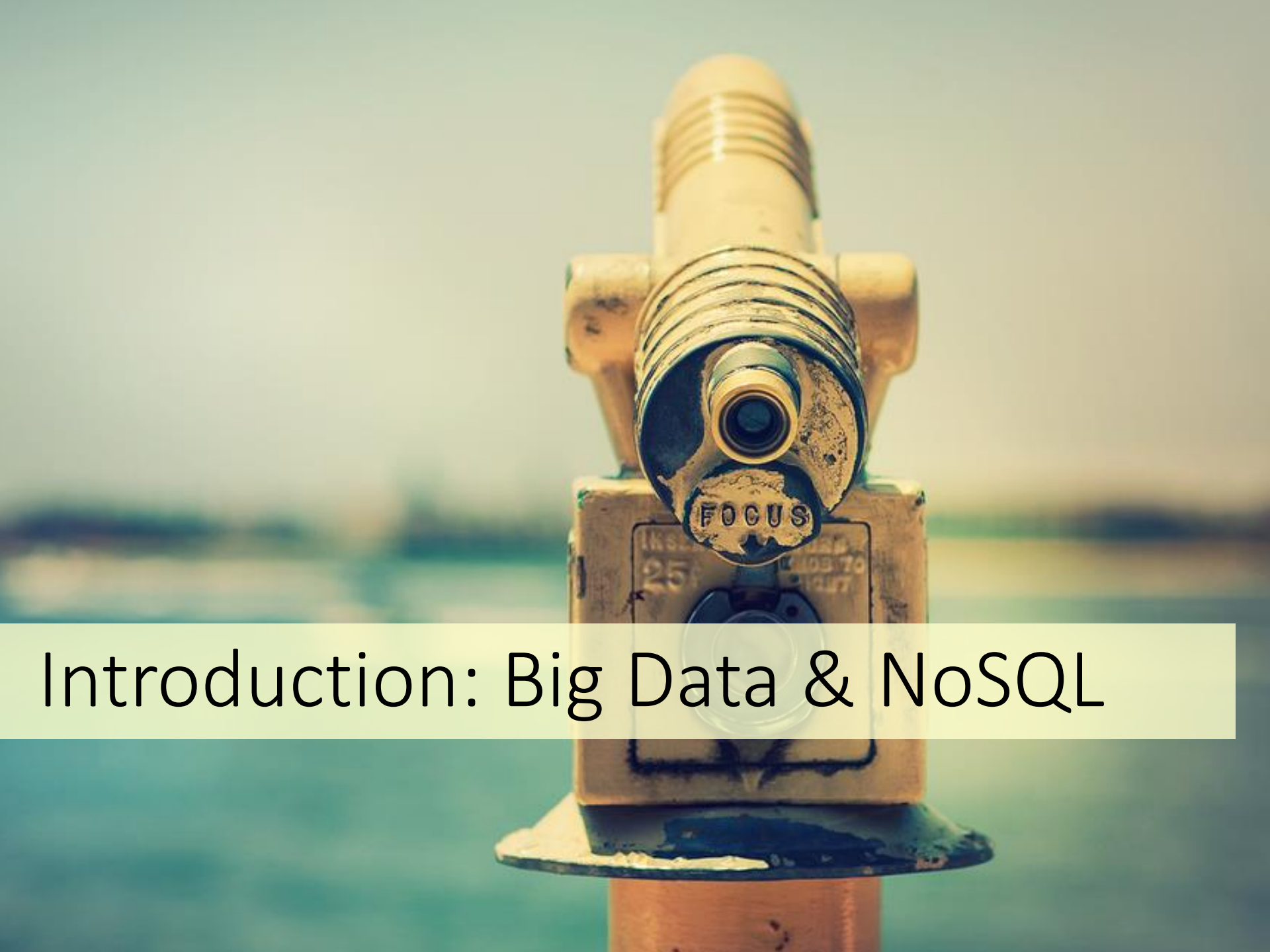
Introduction:
Big Data & NoSQL

- Motivation
  - Big Data
  - NoSQL

Background:
Trade-offs & Limits

Overview:
The 4 Classes of NoSQL

Open Challenges:
Research in Hamburg




# Introduction: Big Data & NoSQL

# Popularity

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

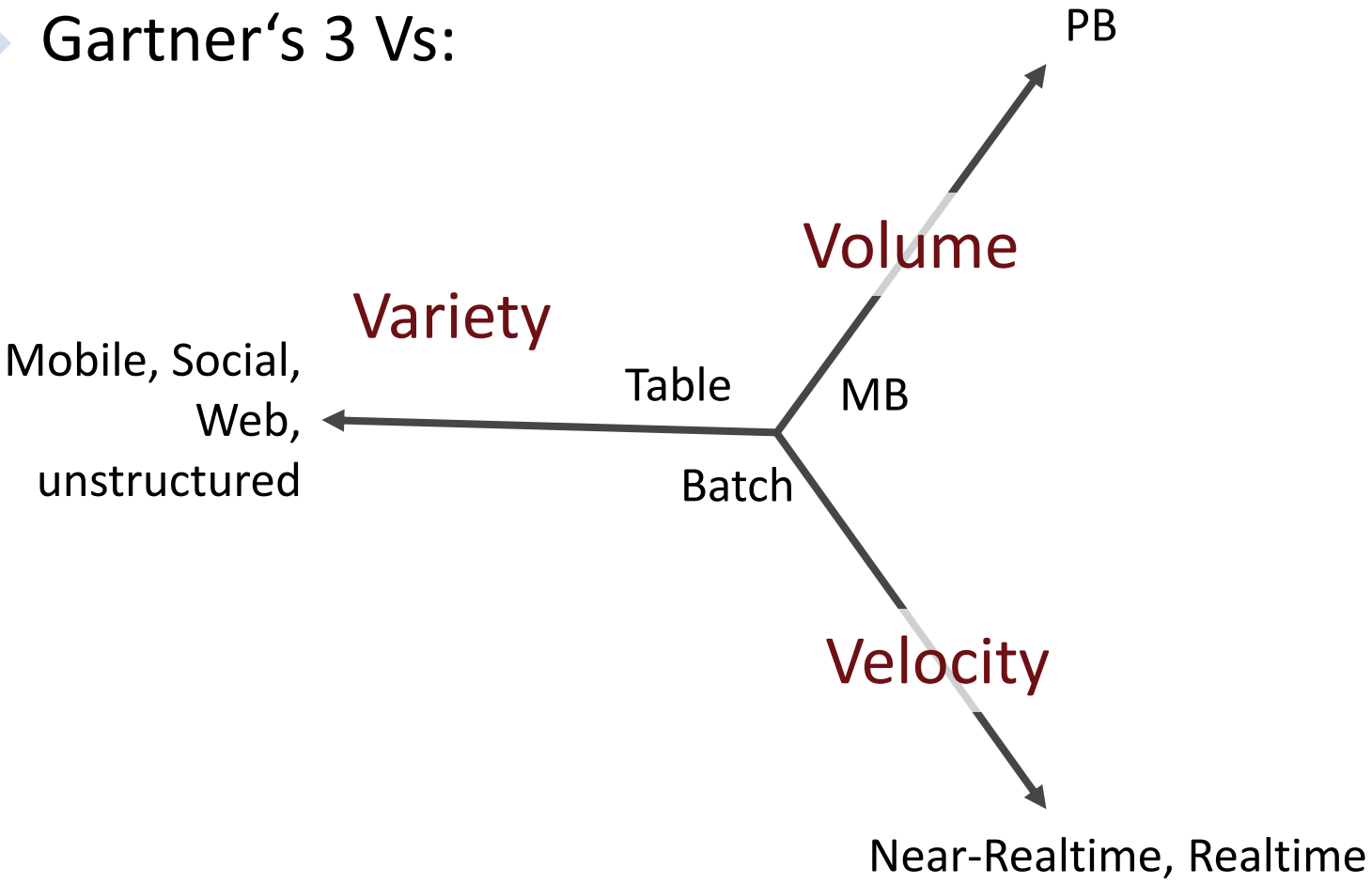
| # | System | Model | Score |
|---|---|---|---|
| **1.** | Oracle | Relational DBMS | 1462.02 |
| **2.** | MySQL | Relational DBMS | 1371.83 |
| **3.** | MS SQL Server | Relational DBMS | 1142.82 |
| **4.** | **MongoDB** | **Document store** | **320.22** |
| **5.** | PostgreSQL | Relational DBMS | 307.61 |
| **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** |

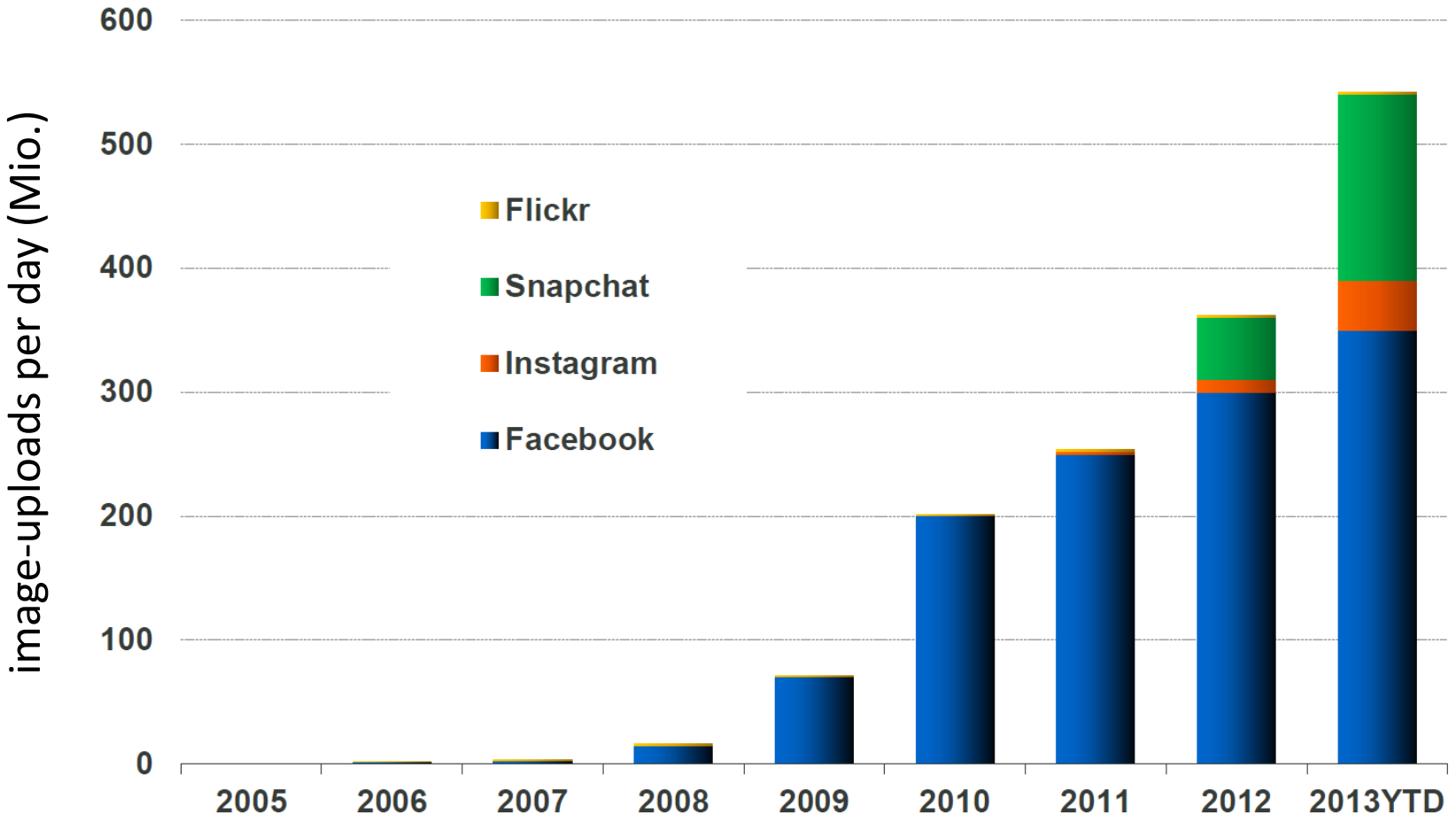
**Scoring**: Google/Bing results, Google Trends, Stackoverflow, job offers, LinkedIn

# Motivation Big Data

- Gartner‘s 3 Vs:

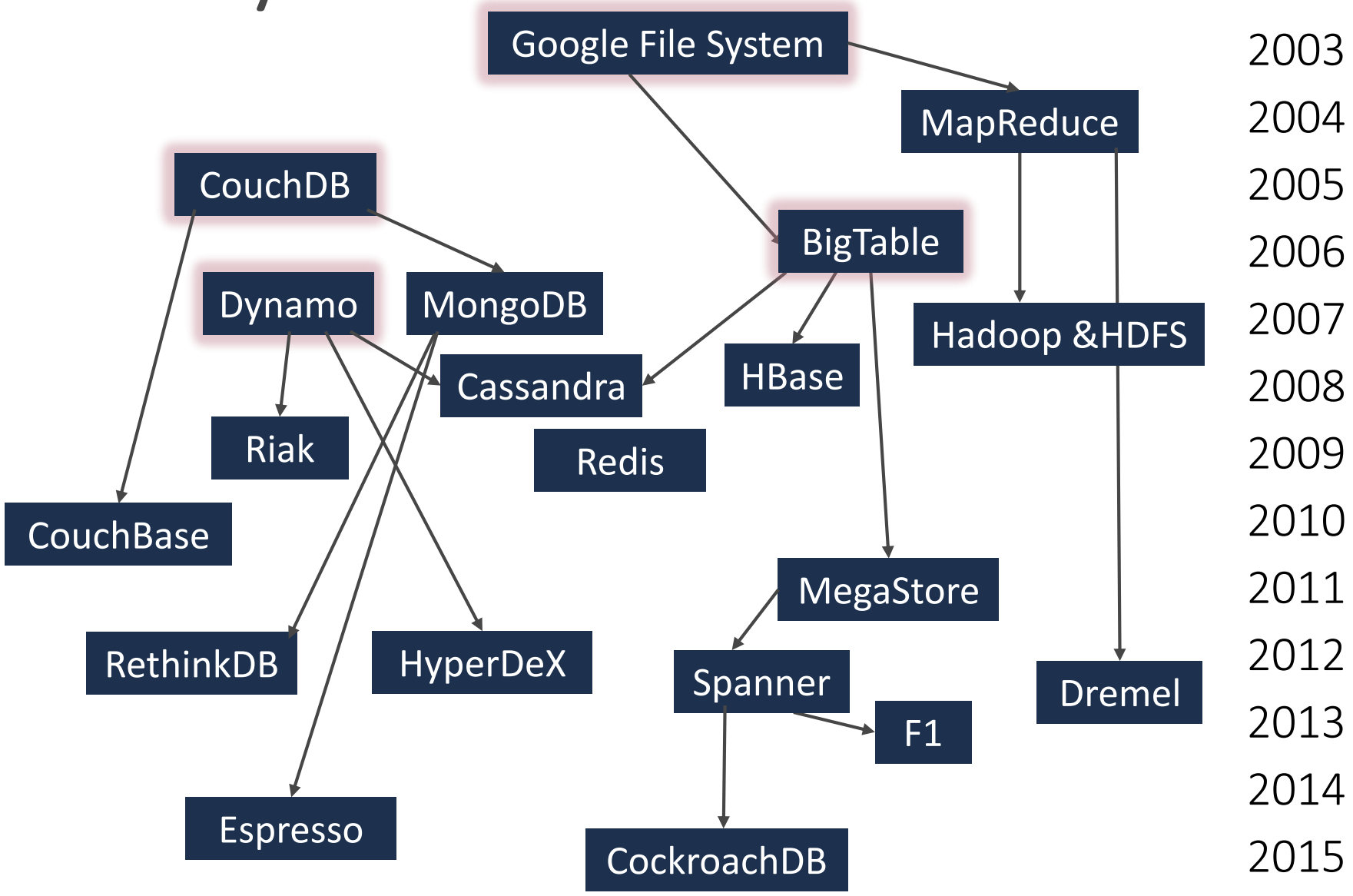


# User-Generated Content: Images




www.kpcb.com/insights/2013-internet-trends

# History


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

# Scaling the Stack

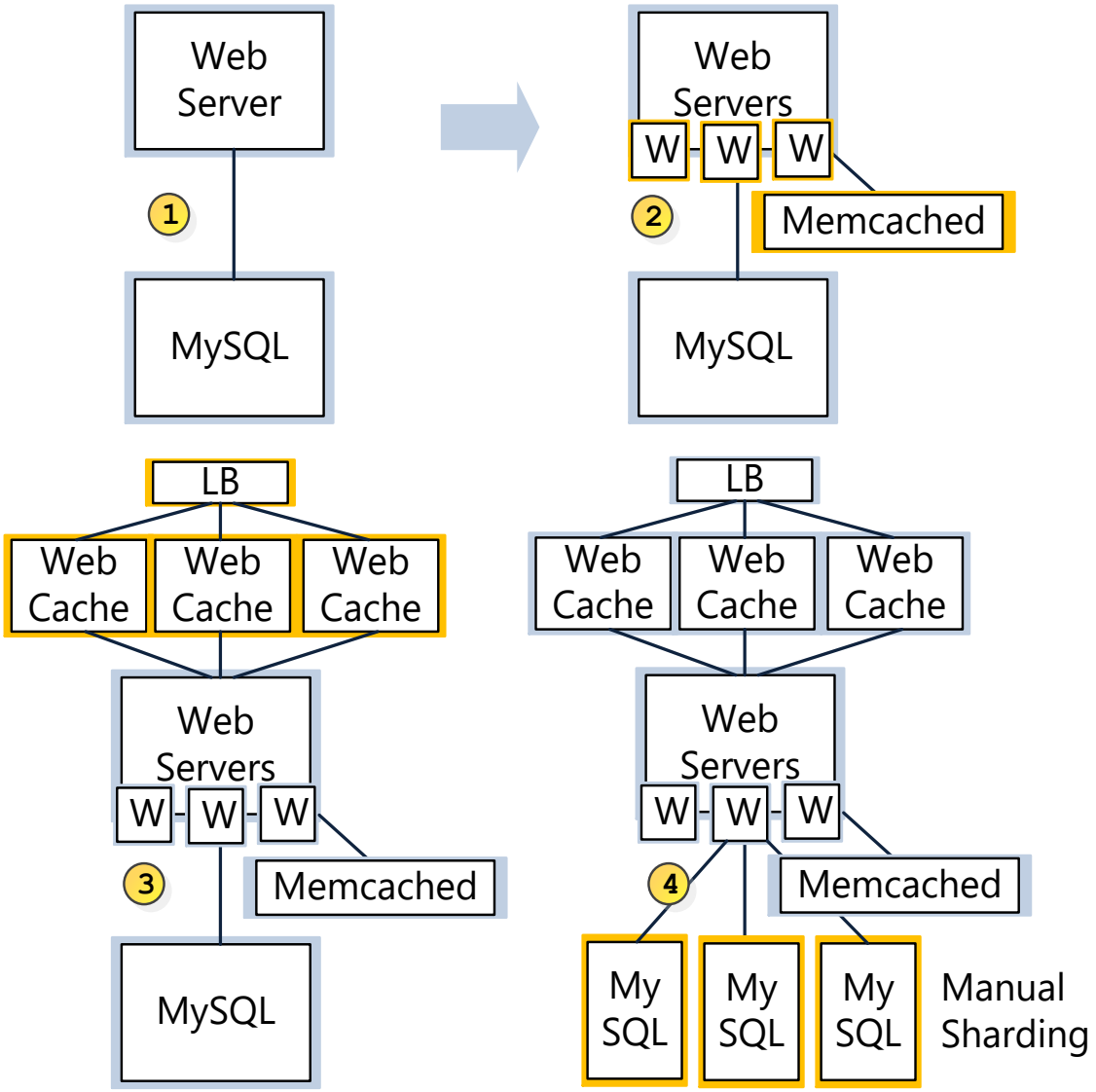
## Grow or Die

Example: **Tumblr**

- Caching
- Sharding from application

Moved towards:

- Redis
- HBase

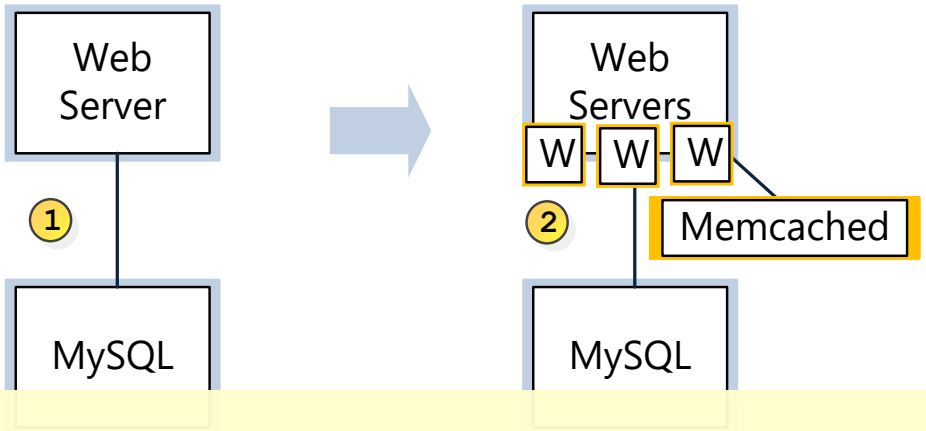


# Scaling the Stack

## Grow or Die

Example: **Tumblr**

- Caching
- Sharding from application

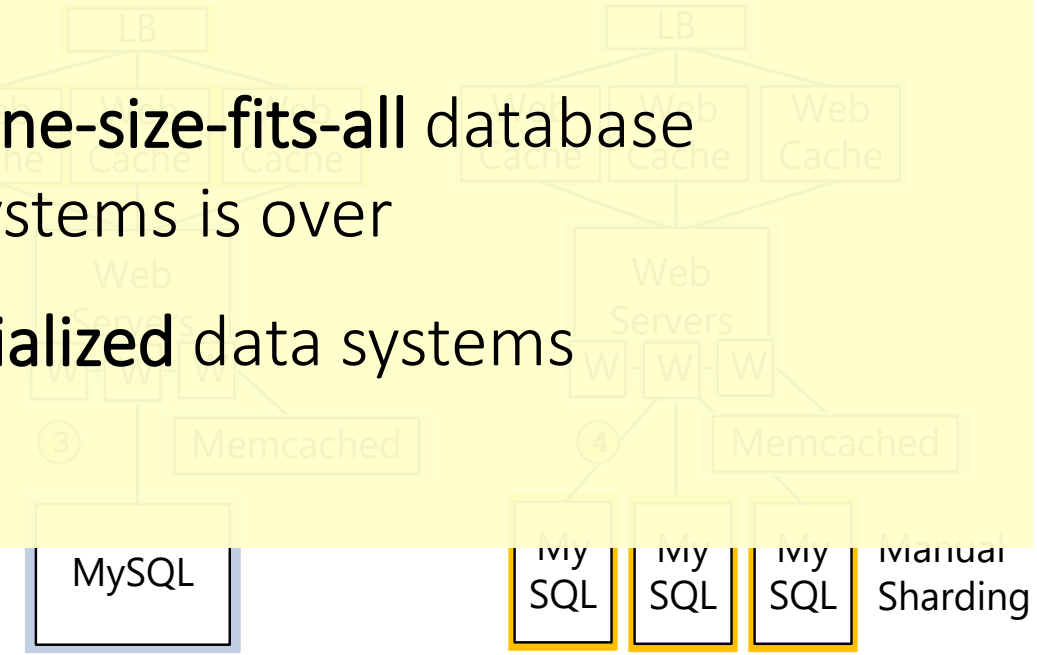



Moved towards:

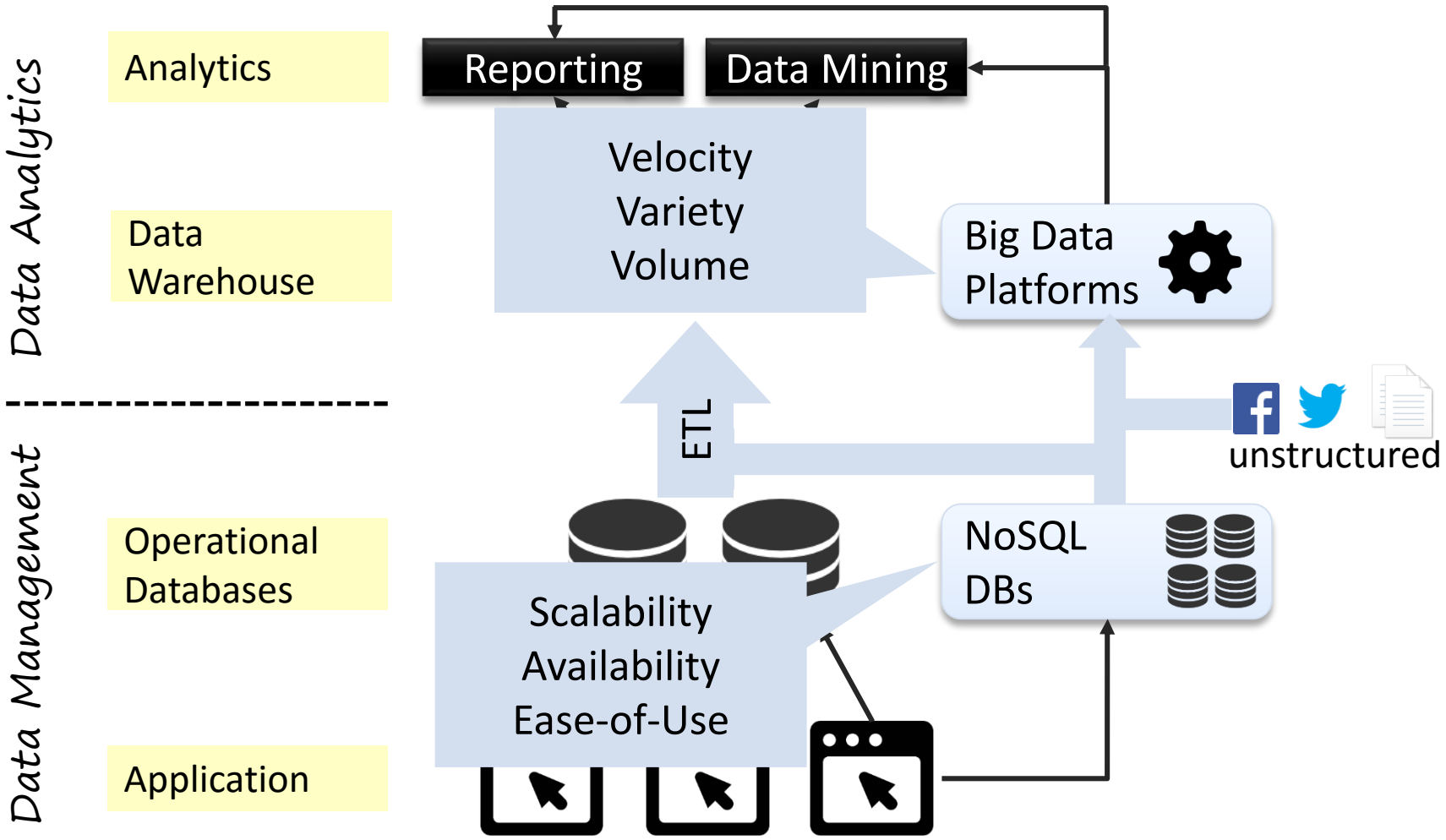
- Redis
- HBase

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

→ **Specialized** data systems



# Architectural Change



# Outline

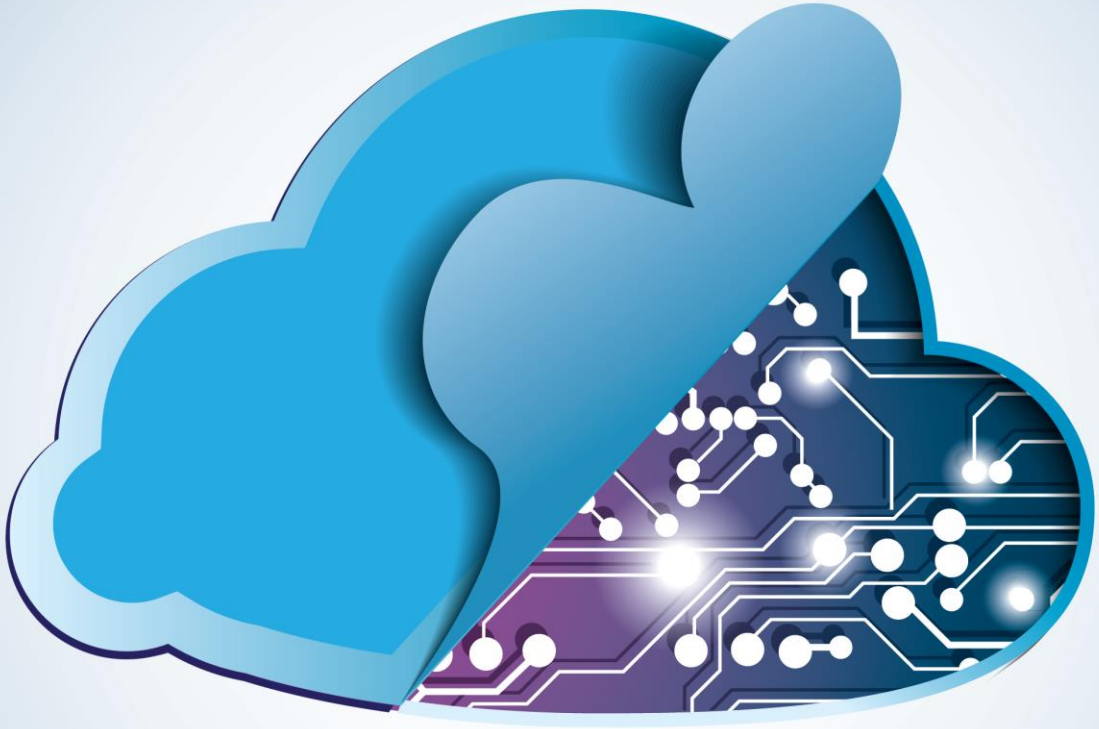
Introduction:
Big Data & NoSQL

Background:
Trade-offs & Limits

Overview:
The 4 Classes of NoSQL

Open Challenges:
Research in Hamburg

- NoSQL: What? Why?
- Data Distribution
  - Sharding
  - Replication
- CAP Theorem
- NoSQL Triangle
- ACID vs BASE



# Background: Trade-offs & Limits

# NoSQL Databases

- „NoSQL“ term coined in 2009
- Interpretation: „**N**ot **O**nly **SQL**“
- Typical properties:
  - Non-relational
  - Open-Source
  - Schema-less (*schema-free*)
  - Optimized for distribution (clusters)
  - Tunable consistency

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


Wide Column Store / Column Families

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

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

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

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

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

Cloudata Google's Big table clone like HBase, Article

Cloudera Professional Software & Services based on Hadoop.

HPCC from LexisNexis, Info, article

Stratosphere (research) system: massive parallel & flexible execution, MR generalization and extension (paper, poster).

(OpenNeptune, Qbase, KDI)

Document Store

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

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

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

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

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

RavenDB Not solution, Provides HTTP/JSON access, LINQ queries & Sharding supported, Misc

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

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

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

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

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

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

SisoDB A Document Store on top of SQL-Server.

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

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

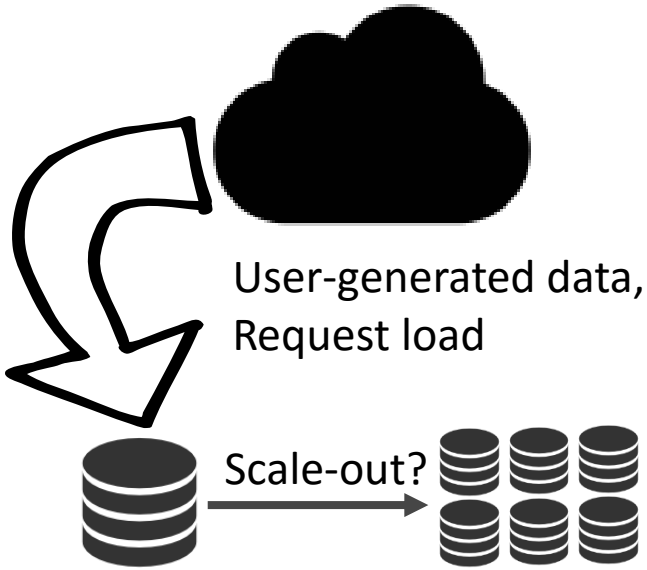
# Paradigm Shift

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

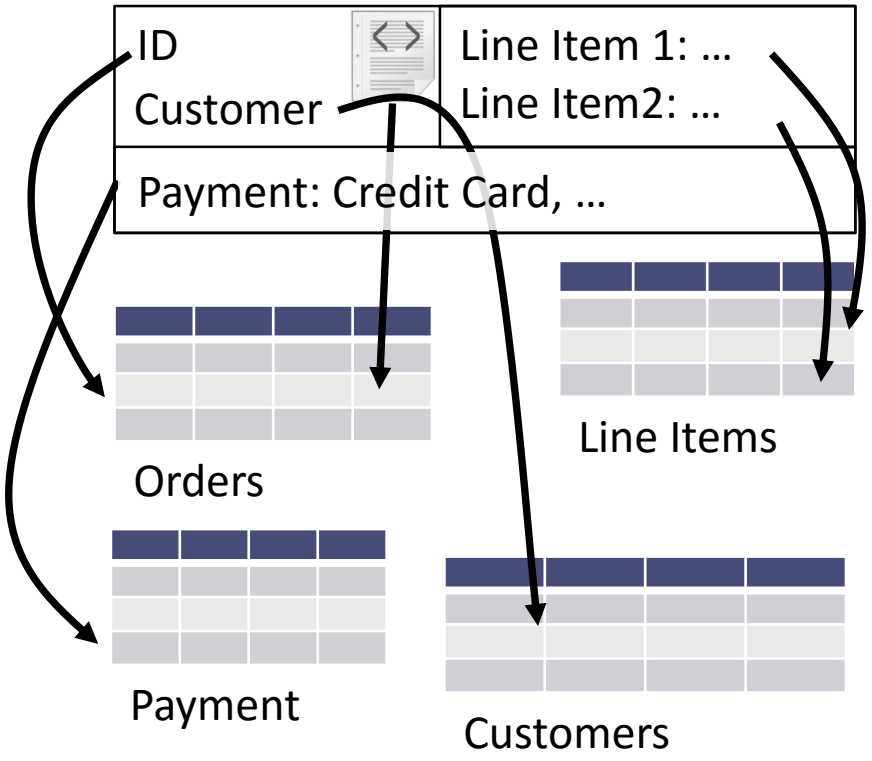
# NoSQL Databases

- Two main motivations:

## Scalability

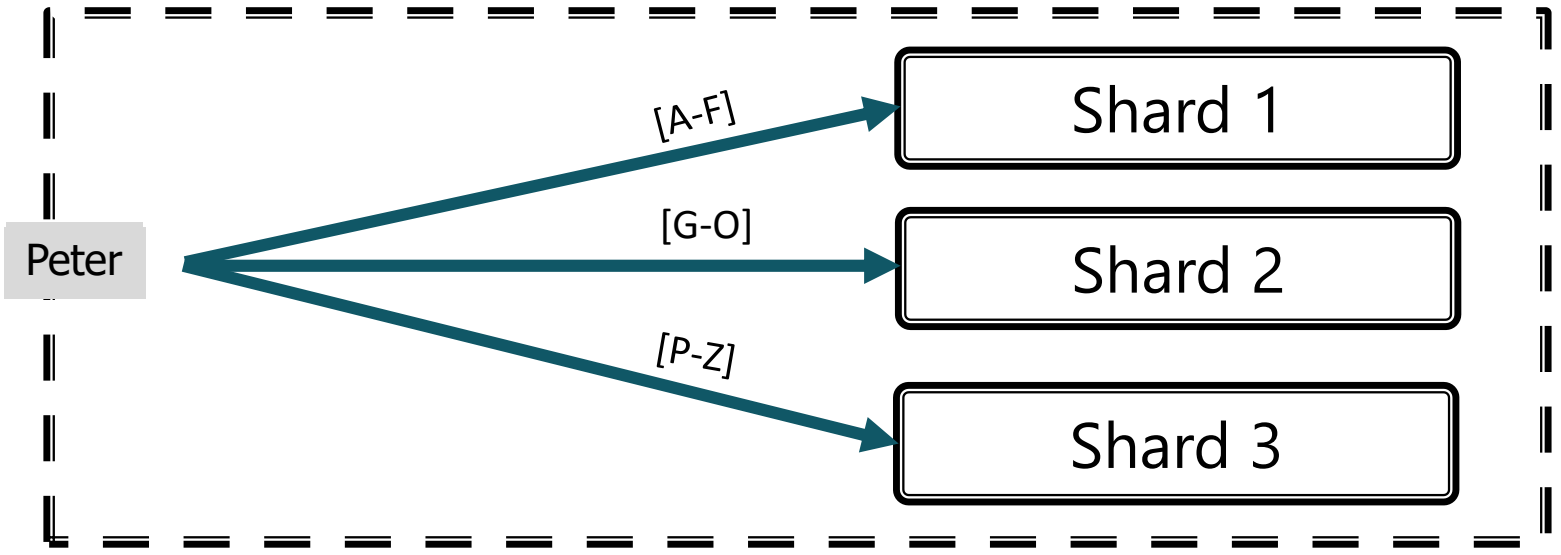



## Impedance Mismatch



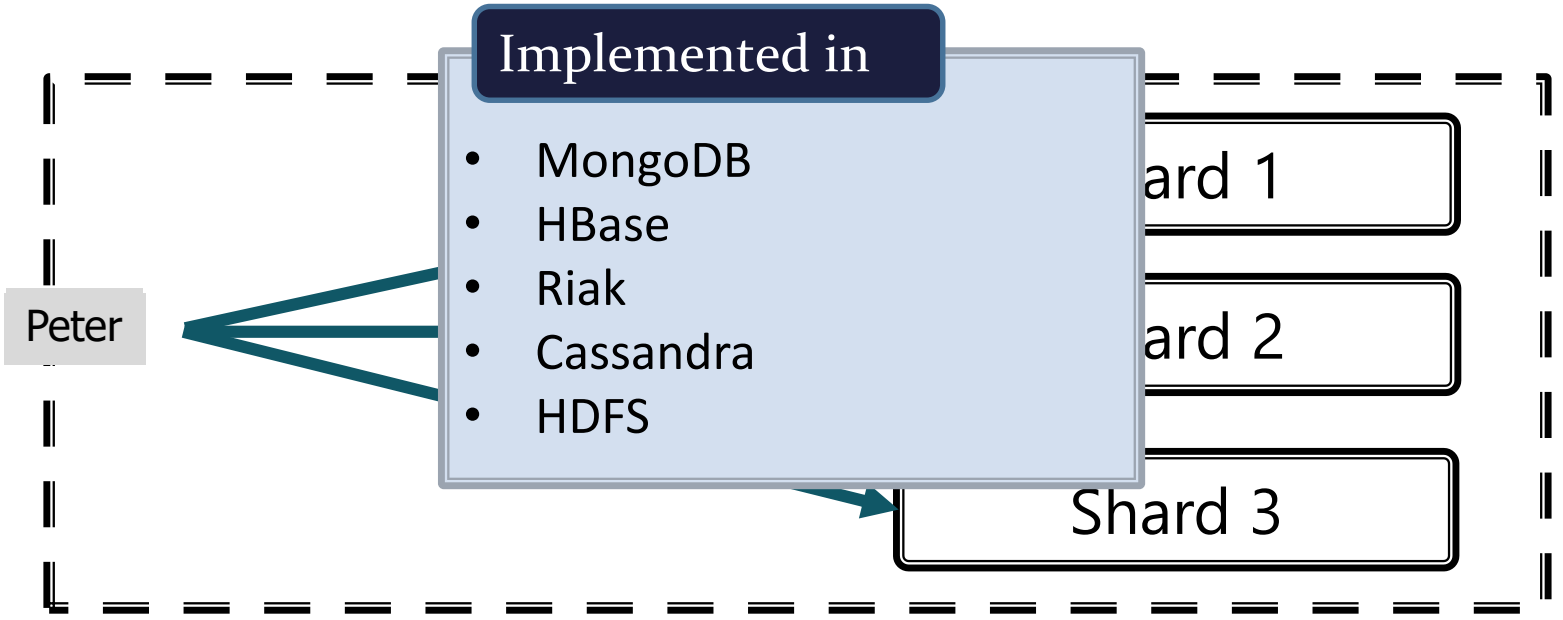
# Sharding (aka Partitioning, Fragmentation)

- Horizontal distribution of data over server nodes
- **Paritioning strategies**: Hash-based vs. Range-based
- **Difficulty**: Multi-Shard-Operations (join, aggregation)



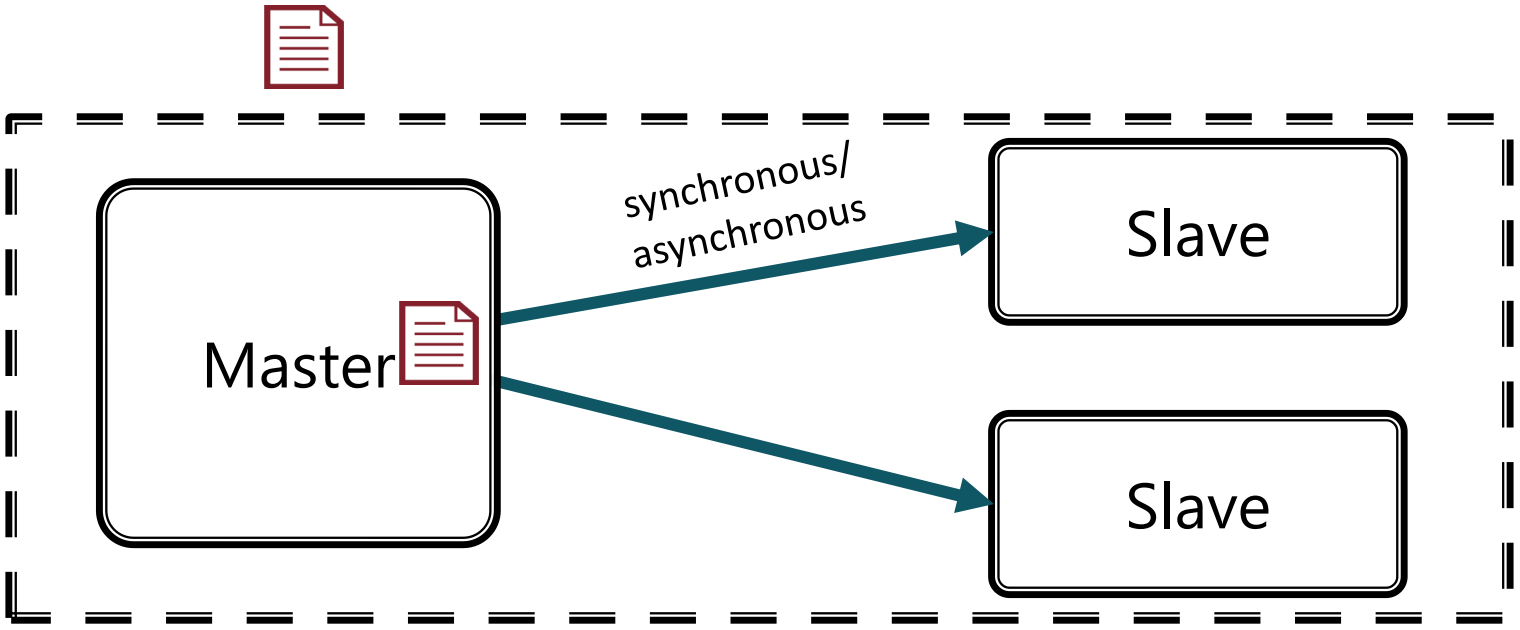
# Sharding (aka Partitioning, Fragmentation)

- Horizontal distribution of data over server nodes
- **Paritioning strategies**: Hash-based vs. Range-based
- **Difficulty**: Multi-Shard-Operations (join, aggregation)



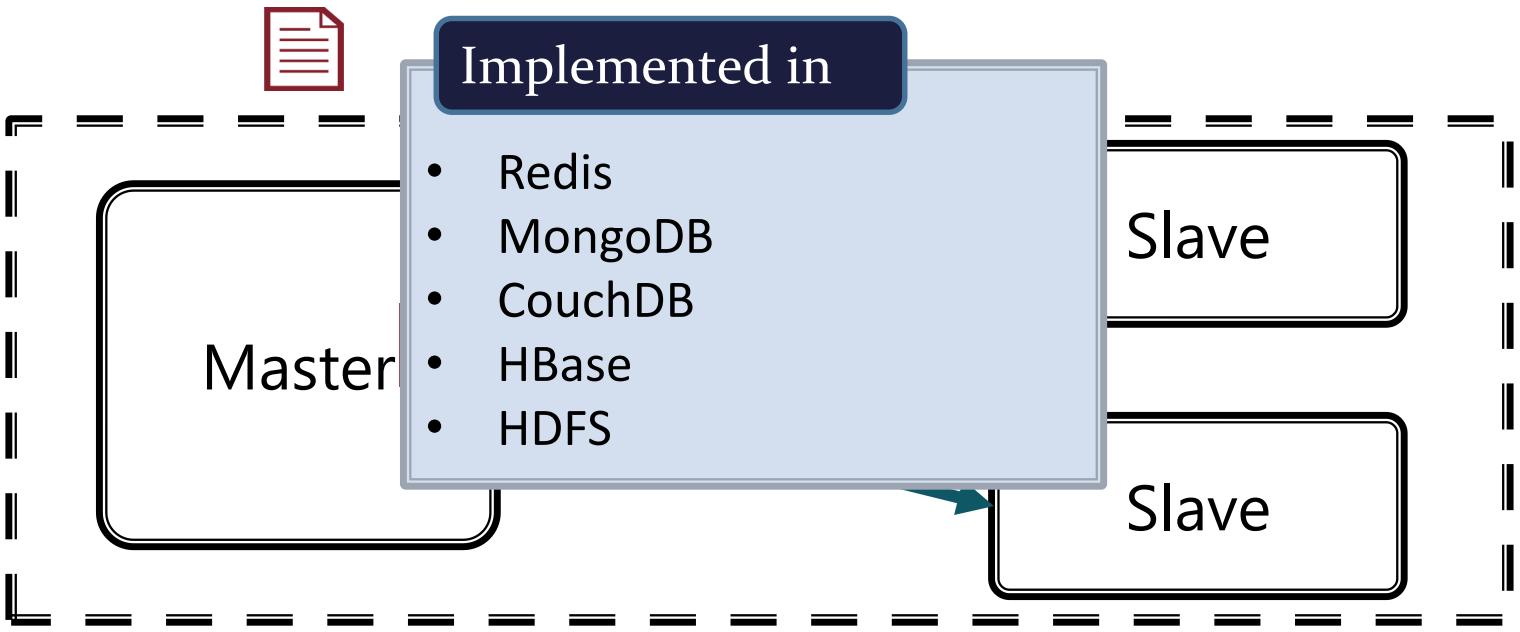
# Replication

- Stores *N* copies of each data item
- **Consistency model**: synchronous vs asynchronous
- **Coordination**: Multi-Master, Master-Slave

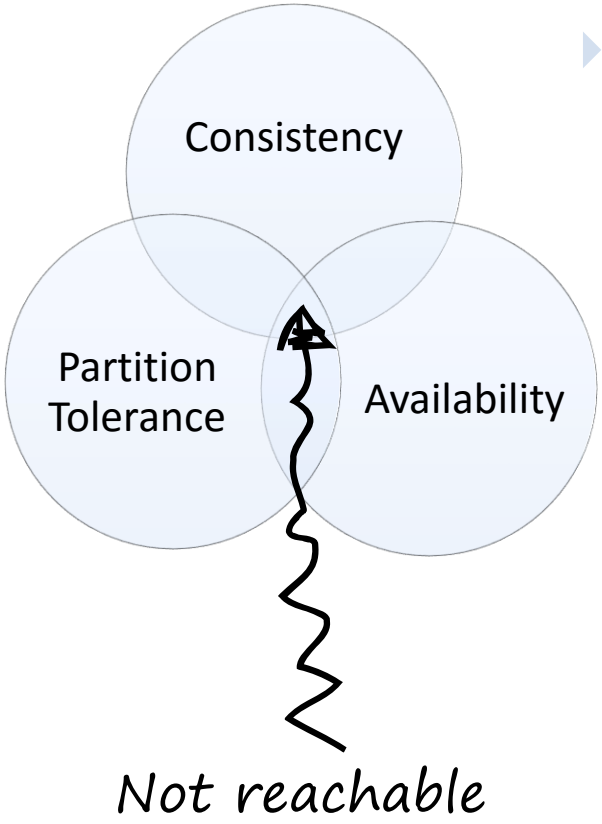


# Replication

- Stores *N* copies of each data item
- **Consistency model**: synchronous vs asynchronous
- **Coordination**: Multi-Master, Master-Slave



# CAP-Theorem



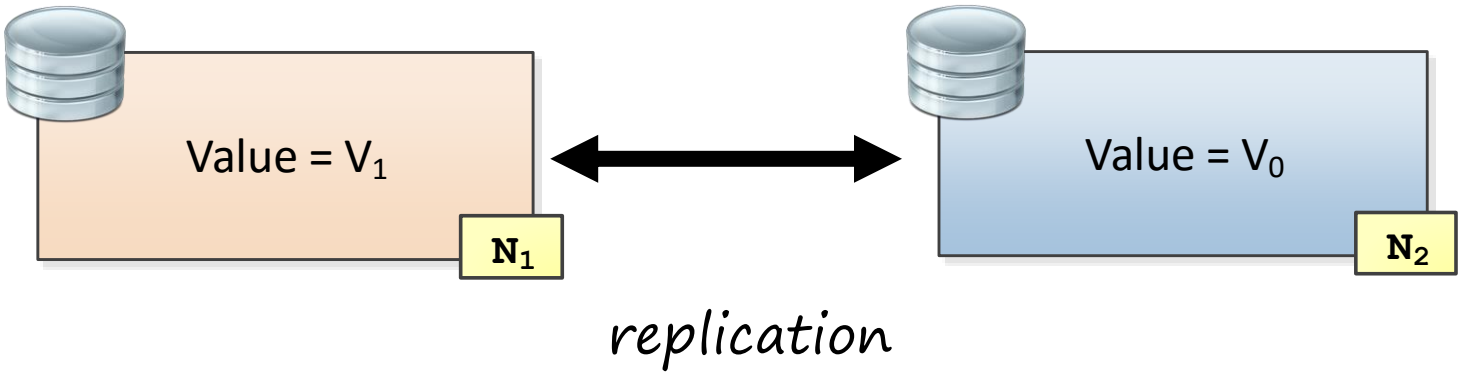

- Classifies distributed databases
- Only 2 out of 3 properties are achievable at a time:
  - **Consistency**: all clients have the same view on the data
  - **Availability**: every request to a non-failed node most result in correct response
  - **Partition tolerance**: the system has to continue working, even under arbitrary network partitions

Eric Brewer, ACM-PODC Keynote, Juli 2000

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

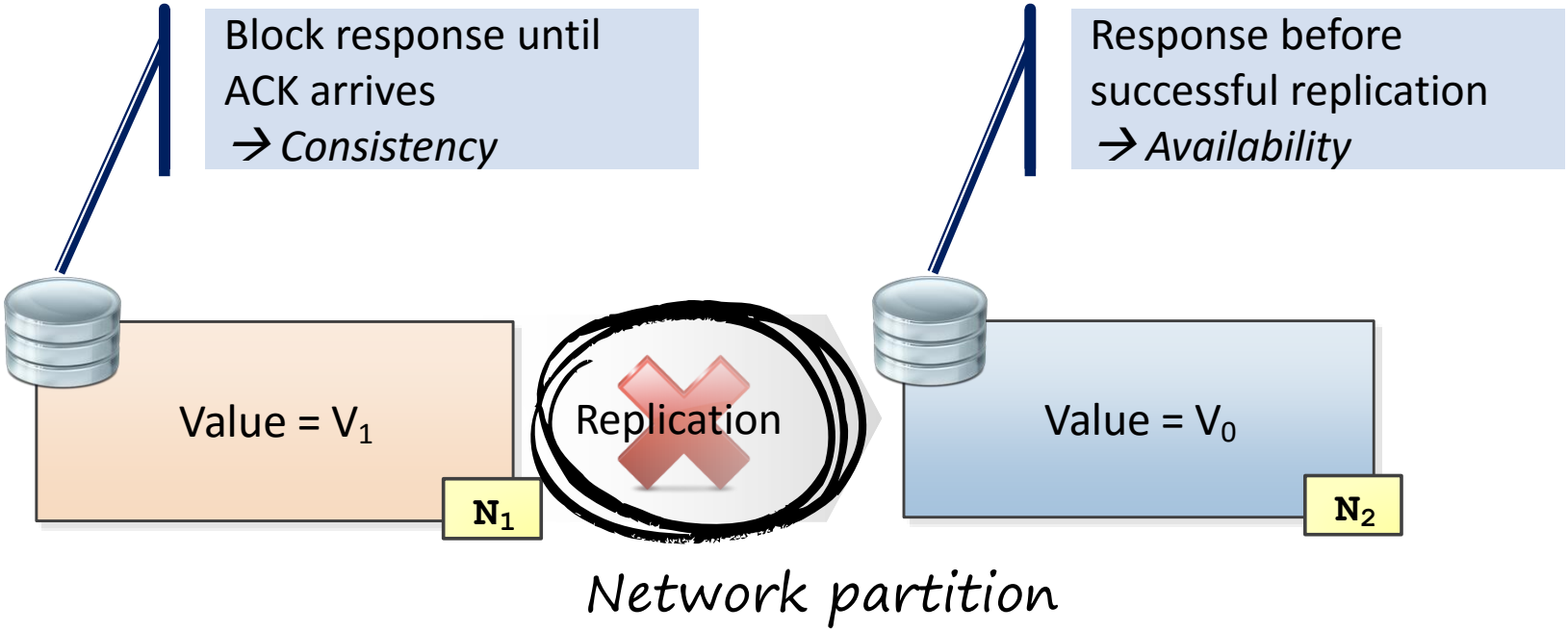
# CAP-Theorem: simplified proof

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



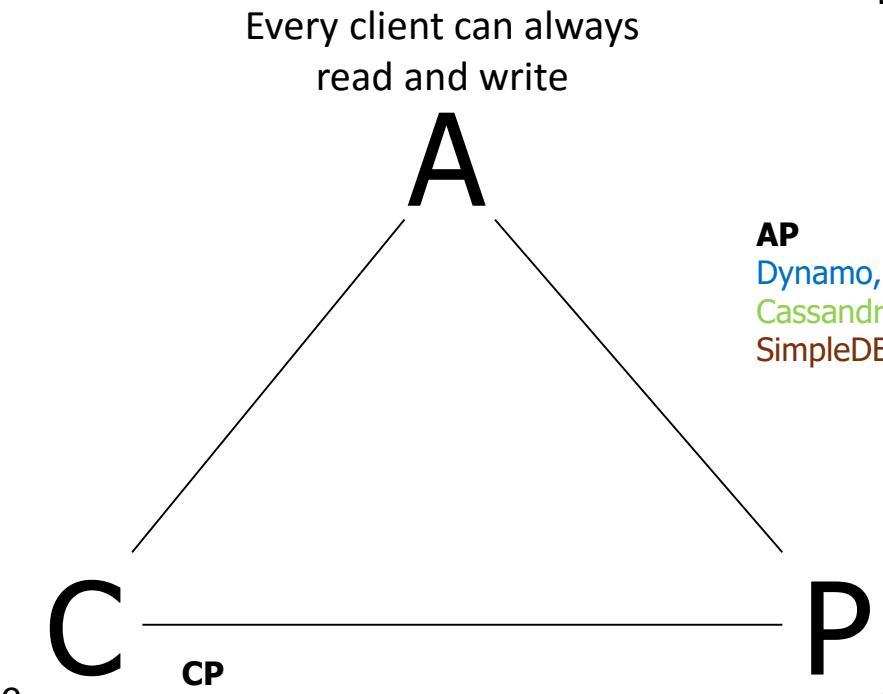
# CAP-Theorem: simplified proof

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



# NoSQL Triangle

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




**CA**
Oracle, MySQL, …

**AP**
Dynamo, Redis, Riak, Voldemort
Cassandra
SimpleDB

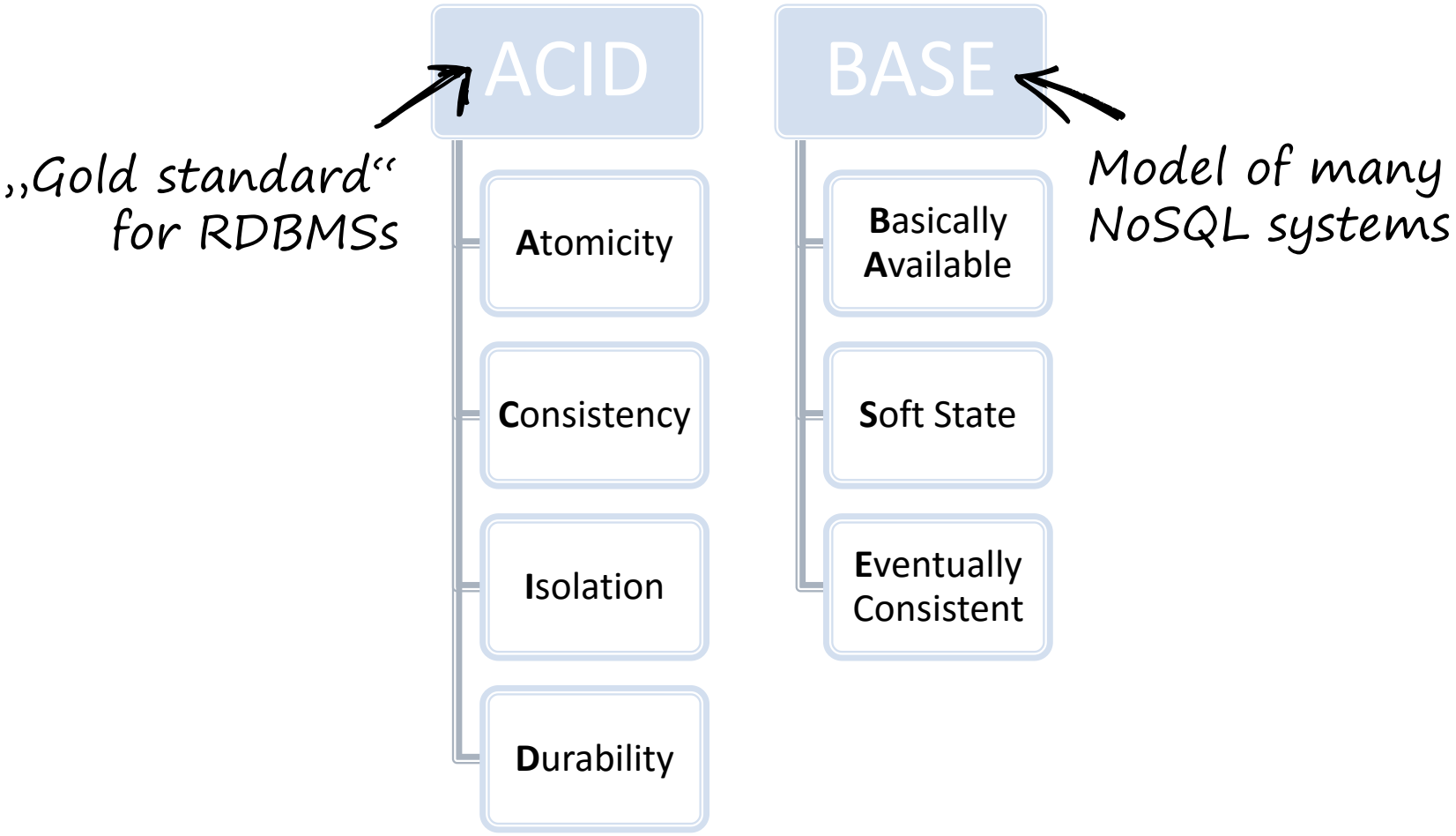
All clients share the same view on the data

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

All nodes continue working under network partitions

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

# ACID vs BASE




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

# Wrap-up: Foundations

- High data volumes, unstructured sources and new kinds of applications triggered BigData and NoSQL technologies
- Shared Nothing architectures for **horizontal scalability**
  - **Replication** enables read scalability and fault tolerance
  - **Sharding** enables write scalability and data volume scalability
- **CAP Theorem**: Consistency, Availability – two one!
  - **ACID ⇔ BASE** (Basically available, soft-state, eventually consistent)

# Outline

Introduction:
Big Data & NoSQL

Background:
Trade-offs & Limits

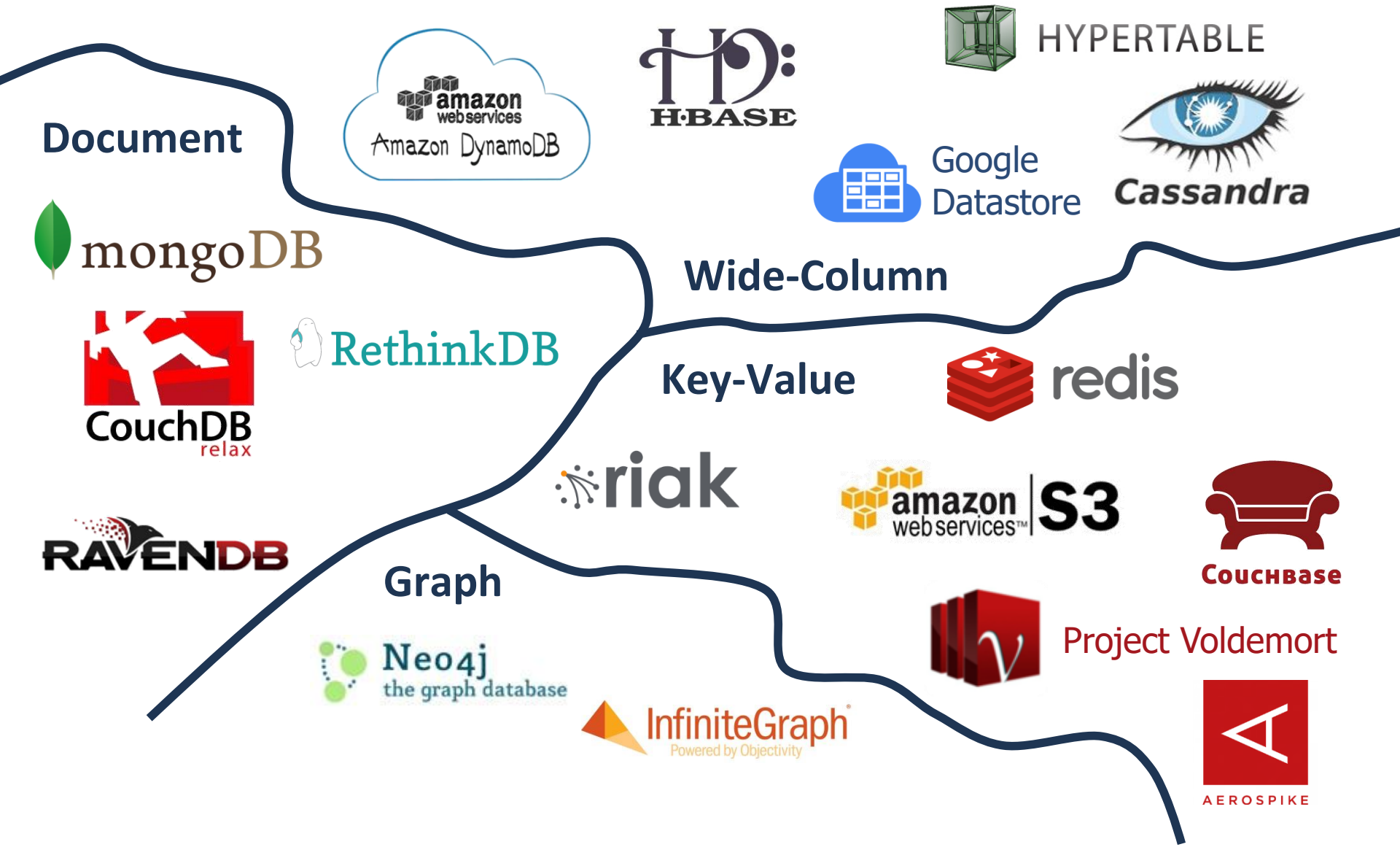
Overview:
The 4 Classes of NoSQL

Open Challenges:
Research in Hamburg

- Key-Value stores
- Wide-Column stores
- Document stores
- Graph databases
- Other classes

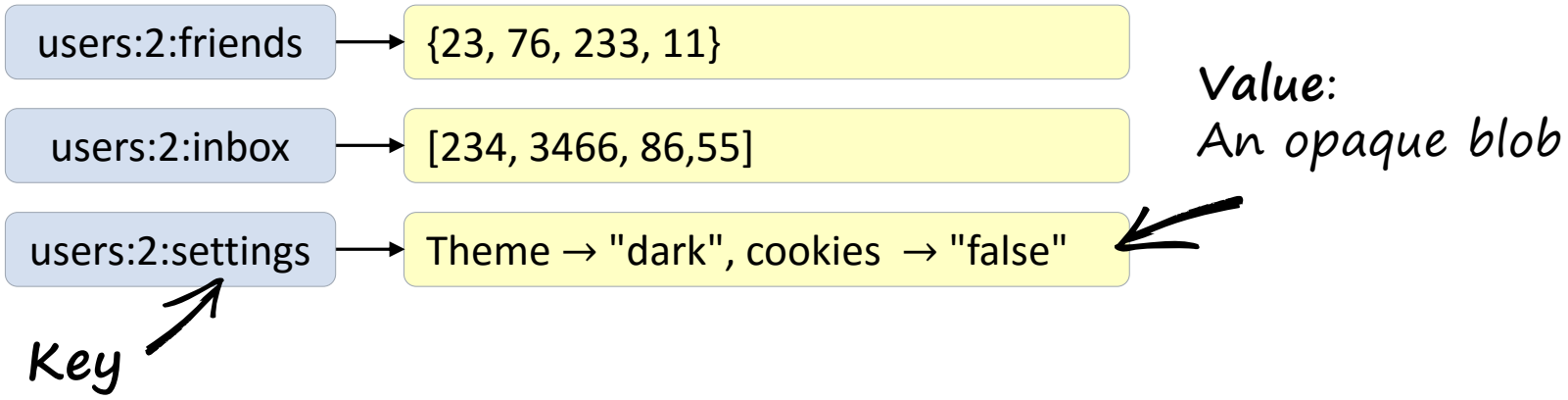

# Overview: The 4 Classes of NoSQL

# NoSQL Landscape



# Key-Value Stores

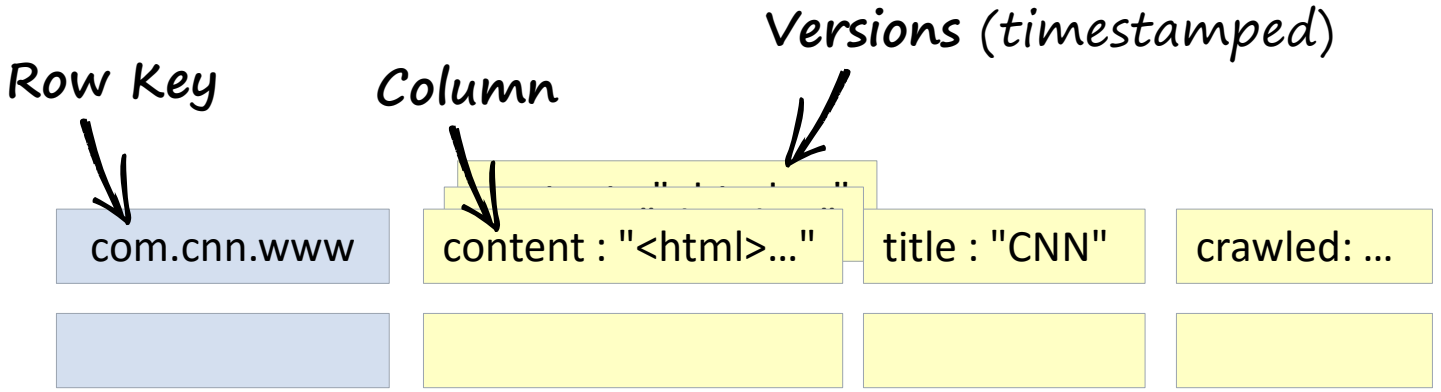
- **Data model:** (key) -> value
- **Interface**: CRUD (Create, Read, Update, Delete)




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

# Wide-Column Stores

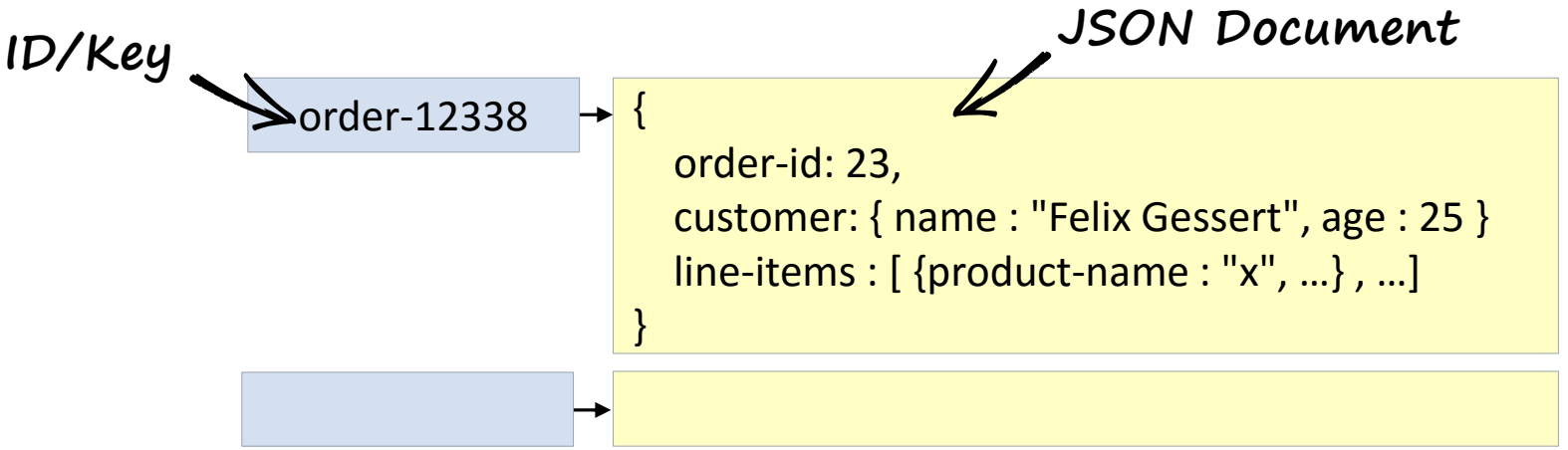
- **Data model:** (rowkey, column, timestamp) -> value
- **Interface**: CRUD, Scan




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

# Document Stores

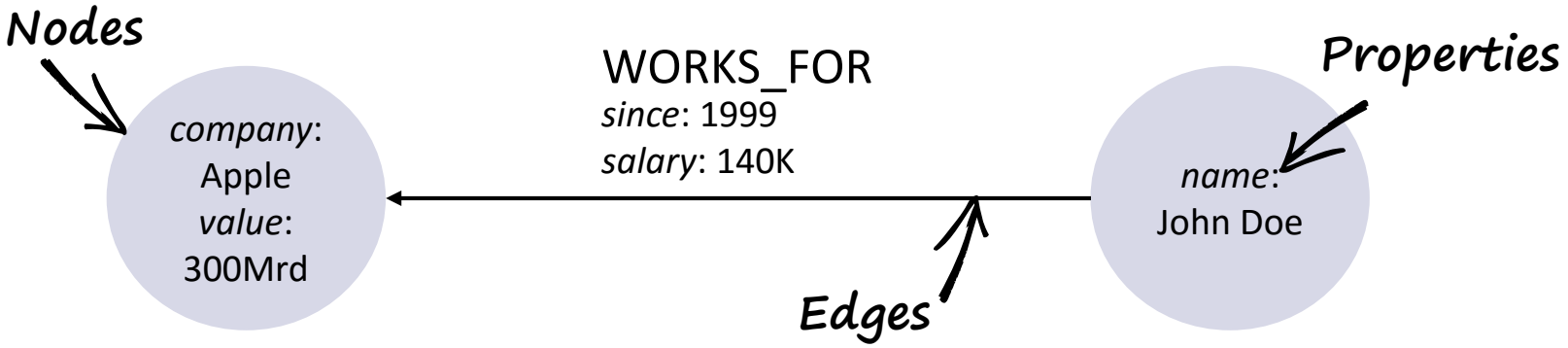
- **Data model:** (collection, key) -> document
- **Interface**: CRUD, Queries




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

# Graph Databases

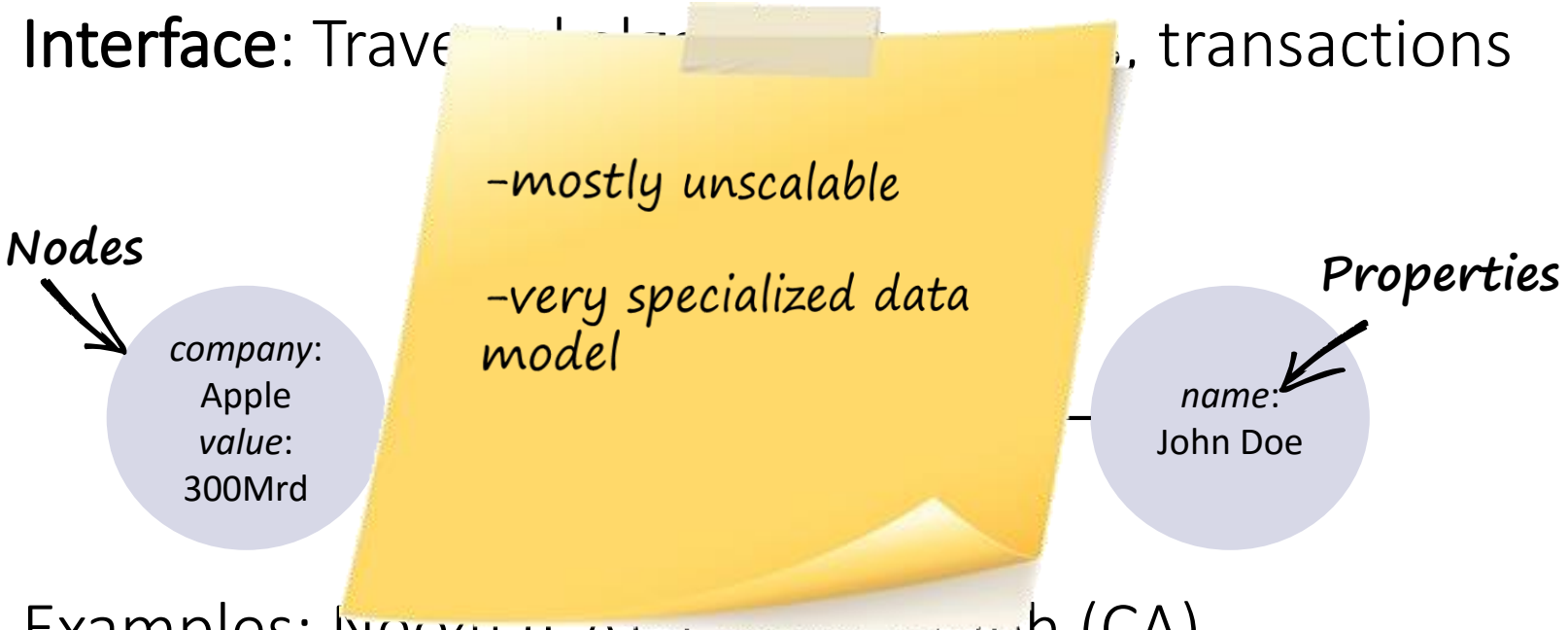
- **Data model:** G = (V, E): Graph-Property Modell
- **Interface**: Traversal algorithms, querys, transactions




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

# Graph Databases

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




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

# Others

## Not Covered Here

**Search Platforms** (Full Text Search):

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

**Multi-Model Databases:**

- Integration of feature sets in a single deployment
- *Examples*: ArangoDB (CP), OrientDB (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)

# Wrap-up

- 4 core NoSQL classes
  - **Key-Value** Stores: store opaque key-value pairs
  - **Document** Stores: store nested, rich, schema-free documents
  - **Wide-Column** Stores: extensible table data model
  - **Graph Databases**: graph-property-model (vertices and edges)
- **There's plenty more**: Multi-model databases, Object-oriented databases, Search platforms, XML databases, Big Data Frameworks, Distributed File Systems

# Outline

Introduction:
Big Data & NoSQL

Background:
Trade-offs & Limits

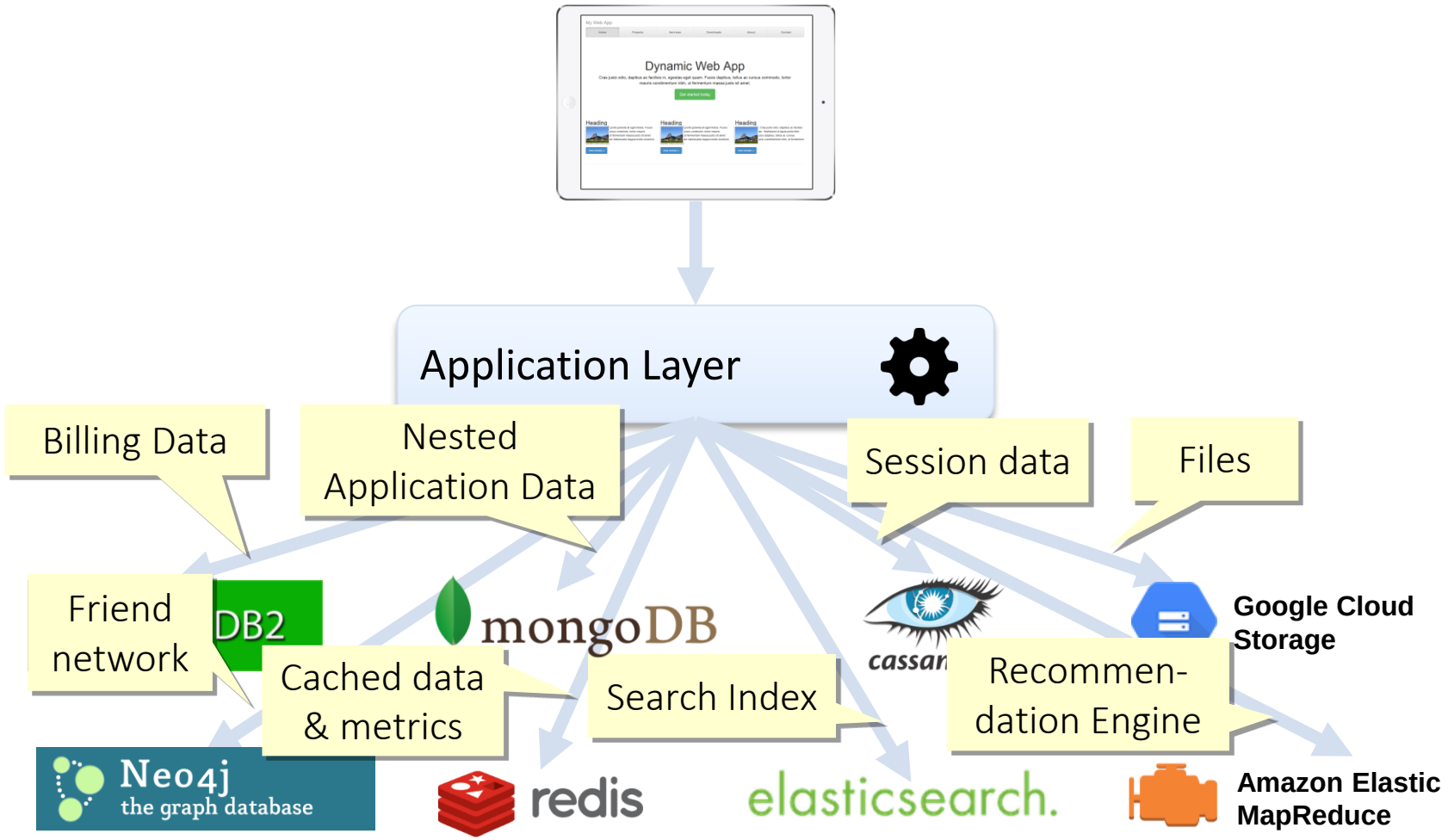
Overview:
The 4 Classes of NoSQL

Open Challenges:
Research in Hamburg

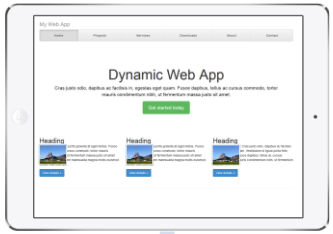
- NoSQL Decision Guidance
- NoSQL Benchmarking
- Polyglot Persistence
- Consistent Caching
- Real-Time Queries

# Open Challenges: Research in Hamburg

# Challenge: Choosing a Tool

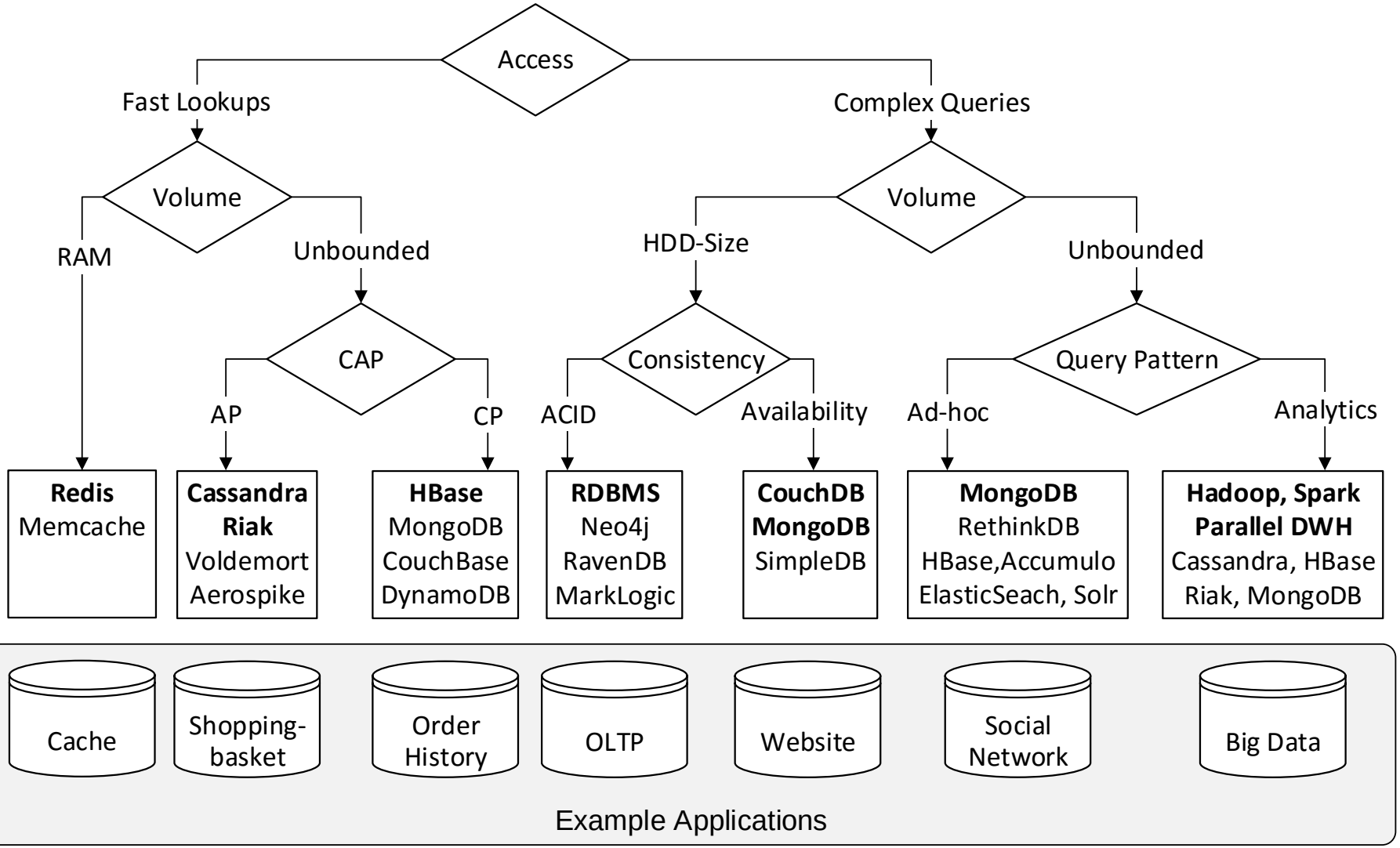


# Challenge: Choosing a Tool

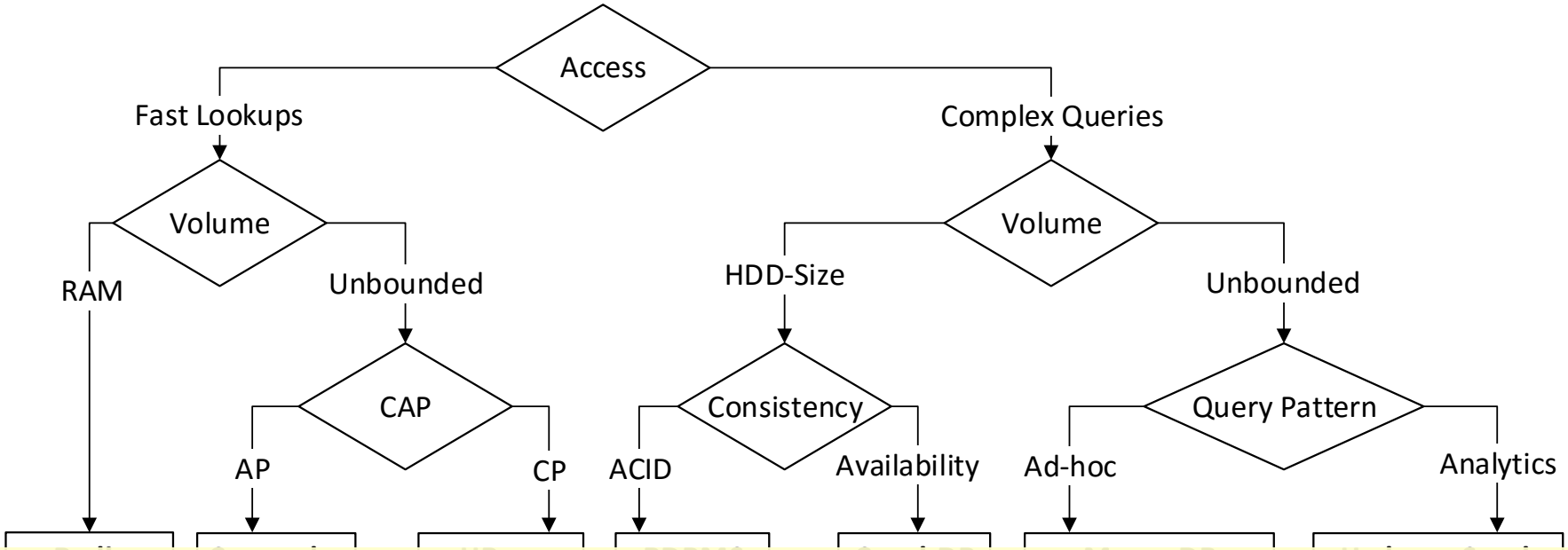



Requirement → Database

# NoSQL Decision Guidance



# NoSQL Decision Guidance




Purpose:

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

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

# Vision: NoSQL Wizard

## 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

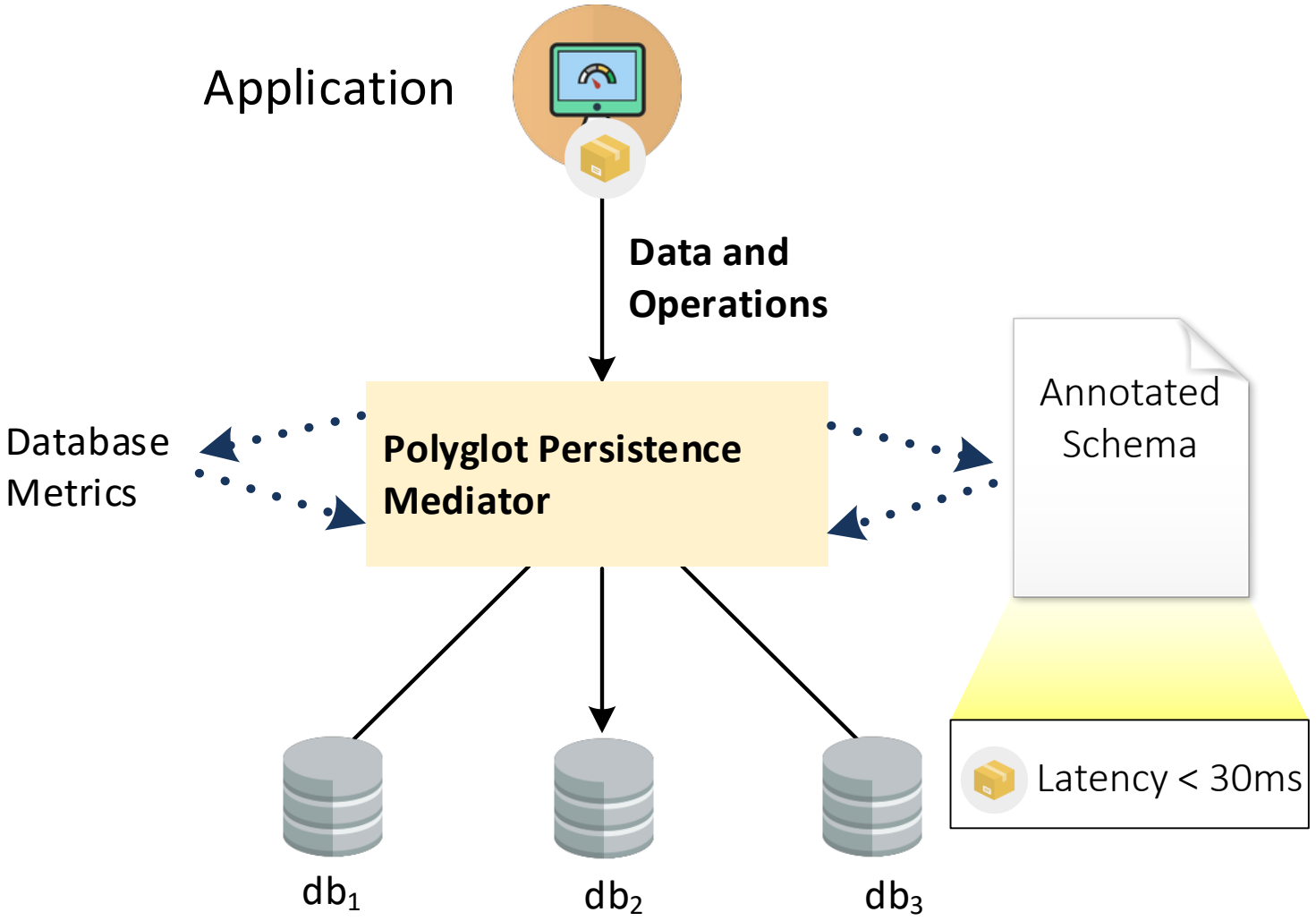
redis

★ 4/5
5/5
5/5

mongoDB

# Vision

## The Polyglot Persistence Mediator chooses the database



# Example: 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

# Challenge: NoSQL Benchmarking

## Comparison by Numbers

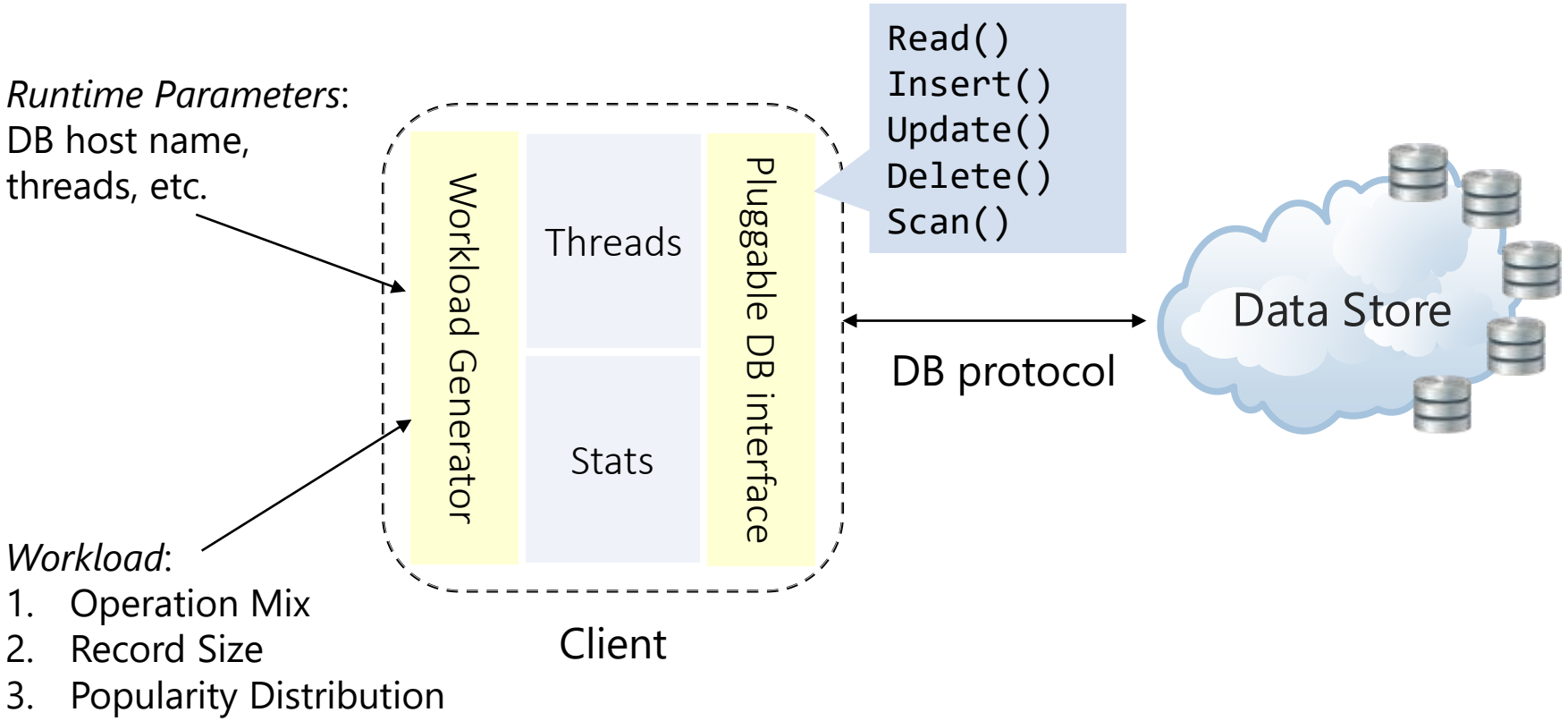
# Challenge: NoSQL Benchmarking

## Comparison by Numbers

- Throughput
- Latency
- Availability
- Consistency
- …

# State-of-the-Art: YCSB

## Yahoo Cloud Serving Benchmark

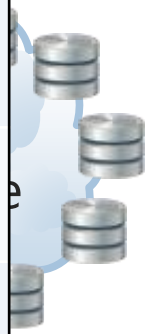


# State-of-the-Art: YCSB

## Yahoo Cloud Serving Benchmark

`Read()`

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



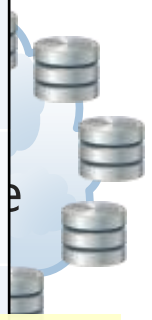
3. Popularity Distribution

# State-of-the-Art: YCSB

## Yahoo Cloud Serving Benchmark

Read()

| Workload | Operation Mix | Distribution | Example |
|---|---|---|---|
| A – Update Heavy | Read: 50%<br>Update: 50% | Zipfian | Session Store |
| B – Read Heavy | Read: 95%<br>Update: 5% | Zipfian | Photo Tagging |
| C – Read Only | Read: 100% | Zipfian | User Profile Cache |



- Availability/Resiliency?
- Consistency?
- Validation?
- Concrete Workloads?
- Distributed Clients?
- Transactions?
- Non-CRUD features?
- …

# State-of-the-Art: YCSB

## Yahoo Cloud Serving Benchmark

### 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

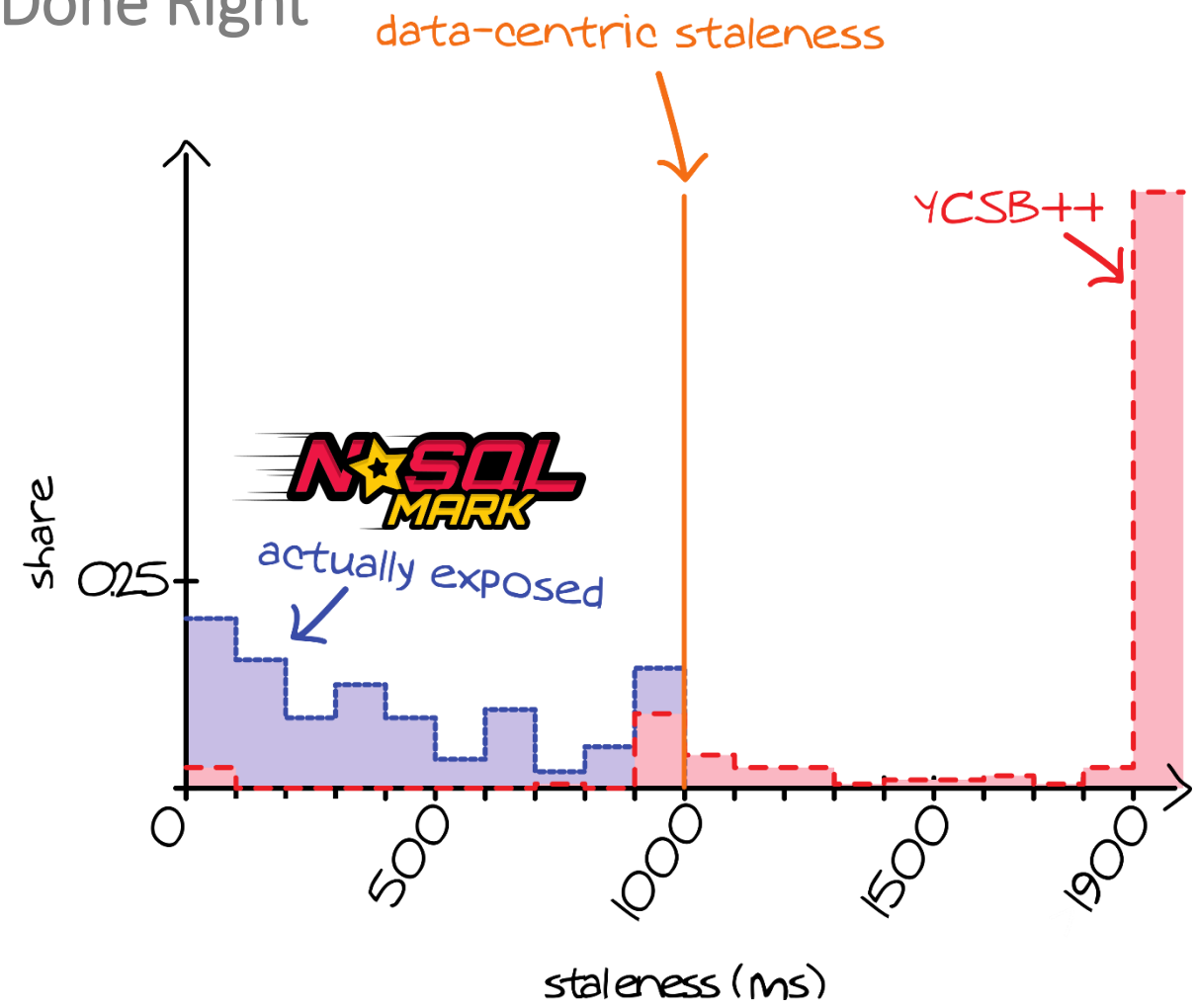
A. Dey et al. “YCSB+T: Benchmarking Web-Scale Transactional Databases”, CloudDB 2014

| | | Distri | |
|---|---|---|---|
| | | Zipfia | |
| | | Zipfia | |
| C – Read Only | Read: 100% | Zipfian | User Profile Cache |

- Availability/Resiliency?
- Consistency?
- Validation?
- Concrete Workloads?
- Distributed Clients?
- Transactions?
- Non-CRUD features?
- …

# NoSQLMark

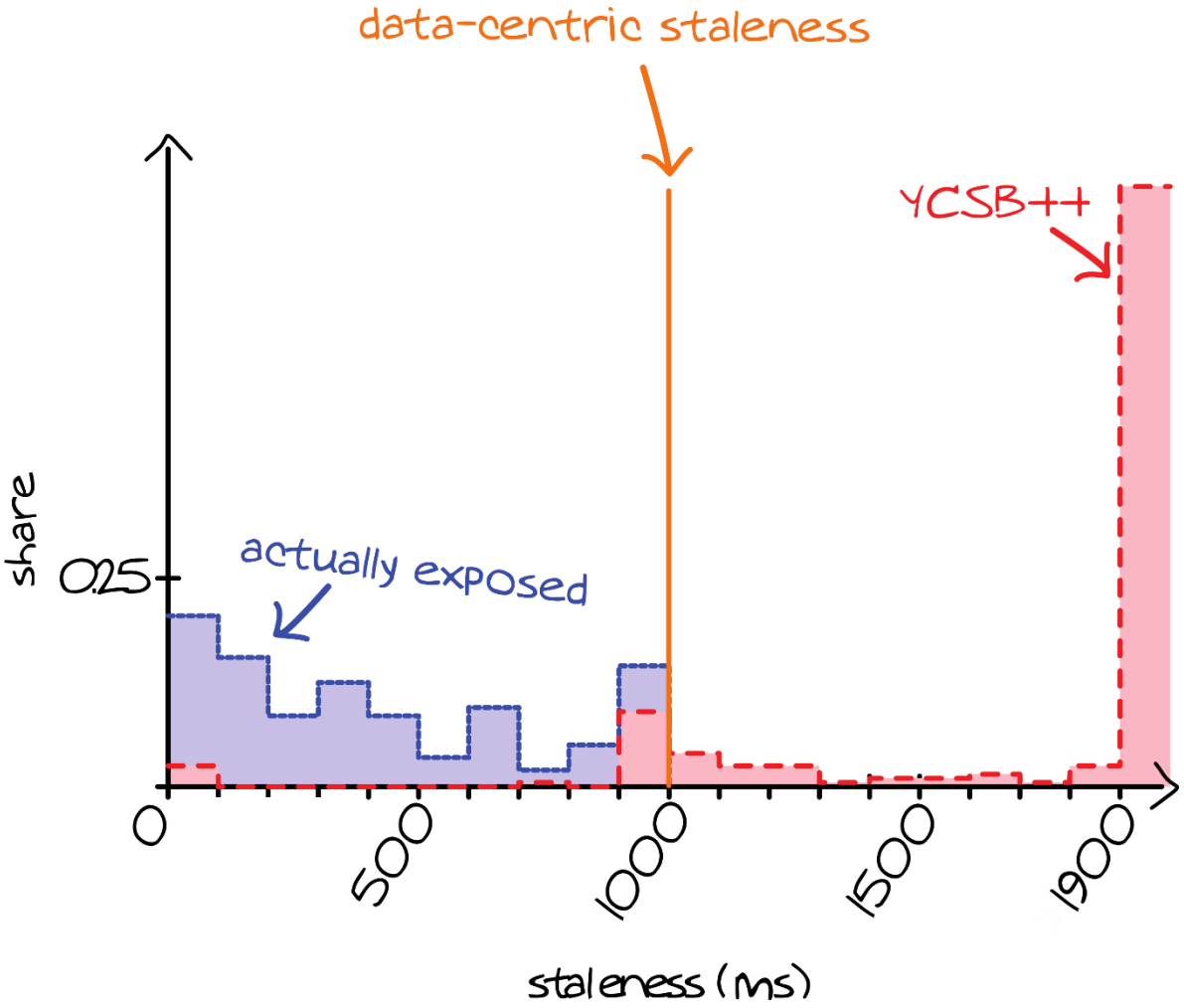
## YCSB Done Right



# NoSQLMark

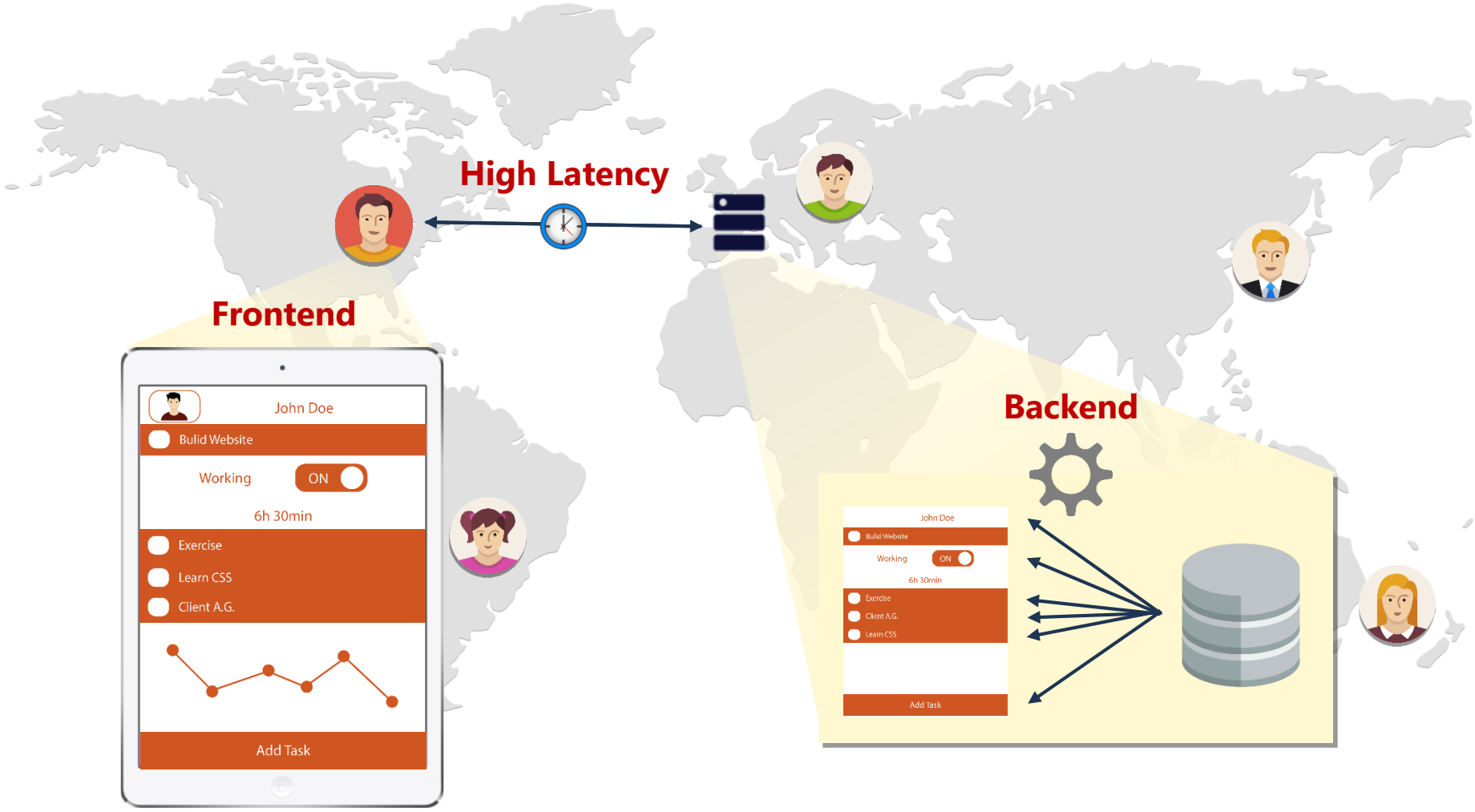
## YCSB Done Right


- Validated
- Distributed clients
- Fully asynchronous
- YCSB-compatible
- Real-world workloads
- …

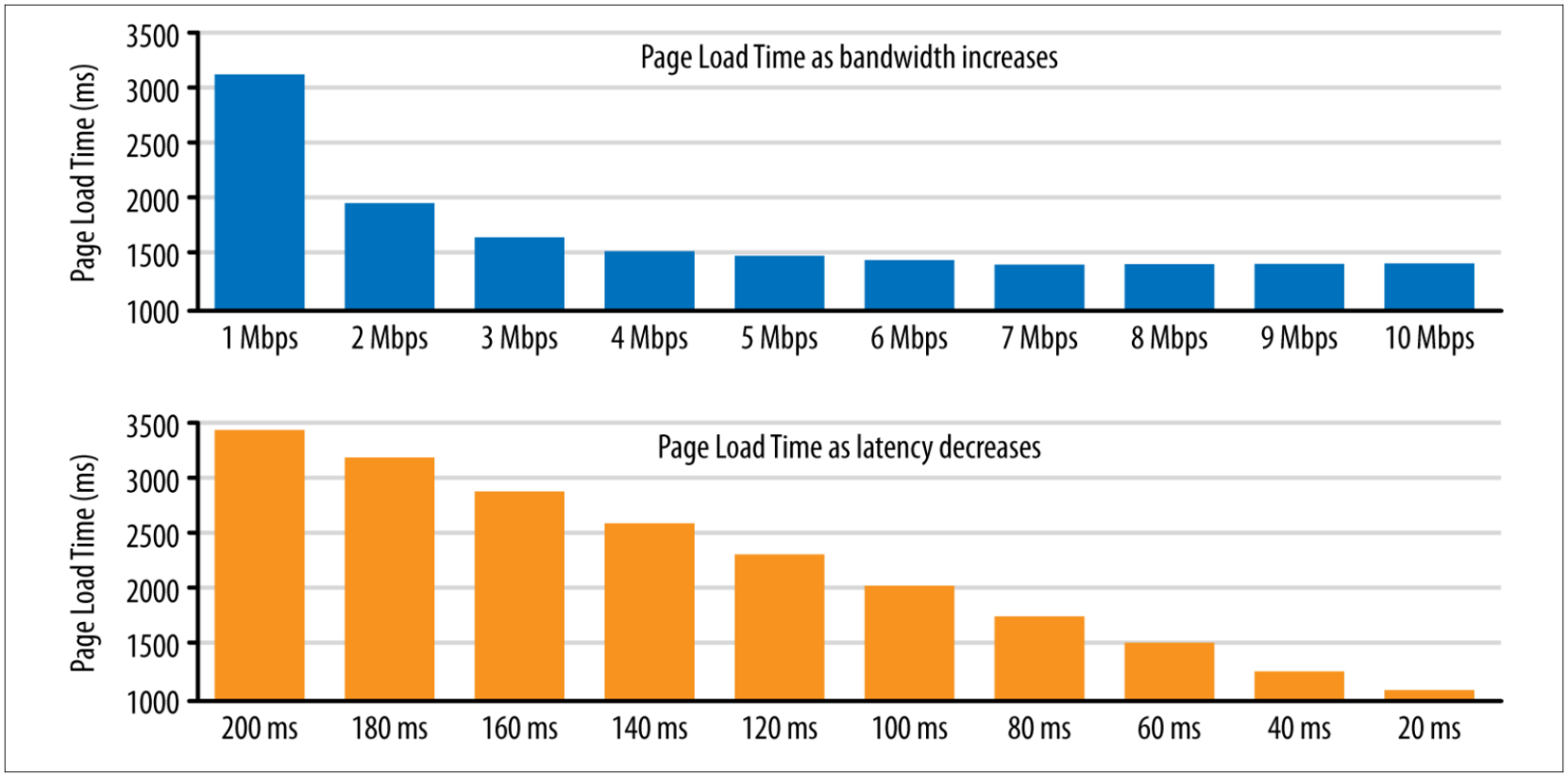


# Challenge: High Latency for Global Access

## Three Bottlenecks: Latency, Backend & Frontend

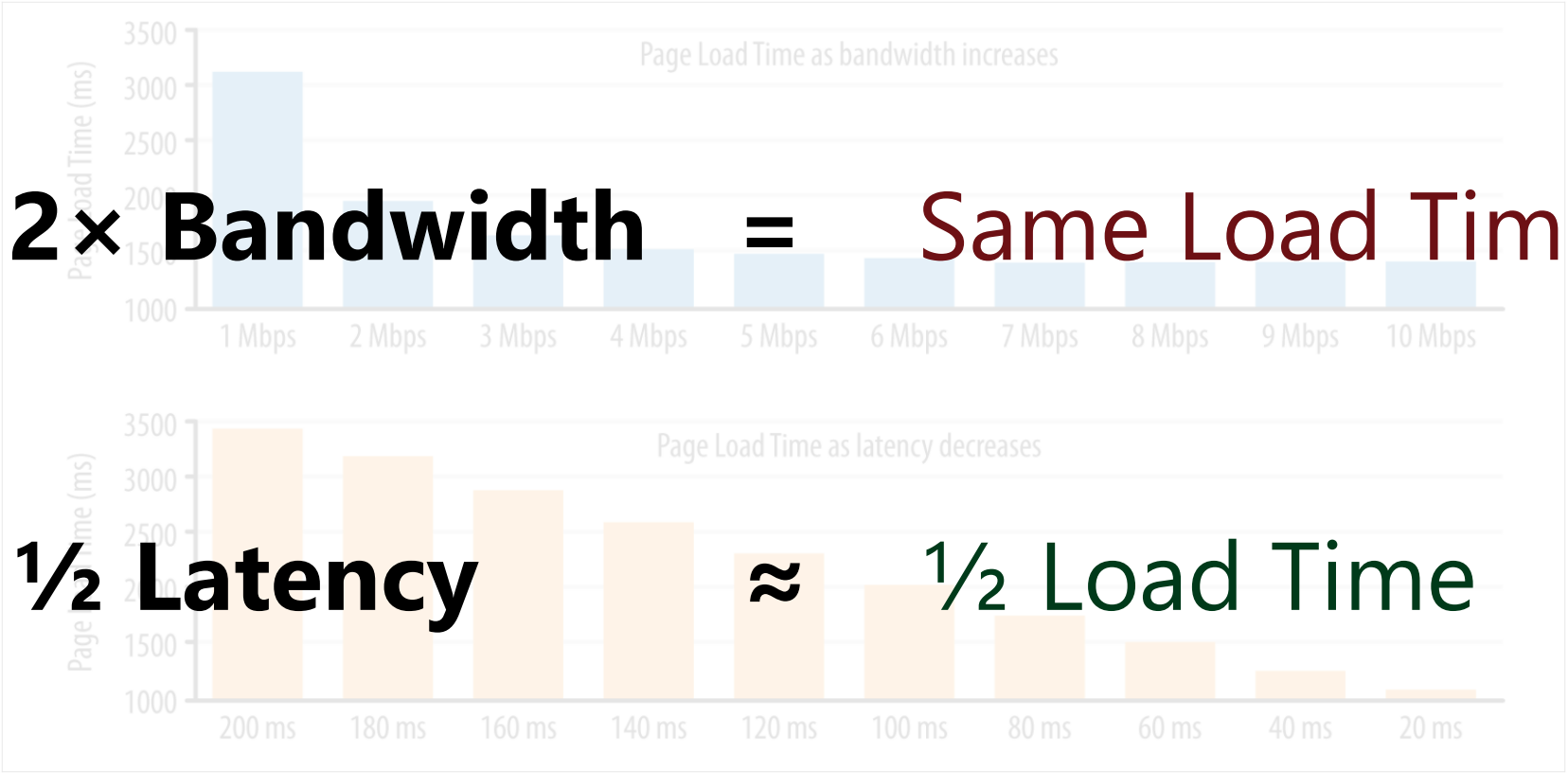


# Network Latency: Impact




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

# Network Latency: Impact



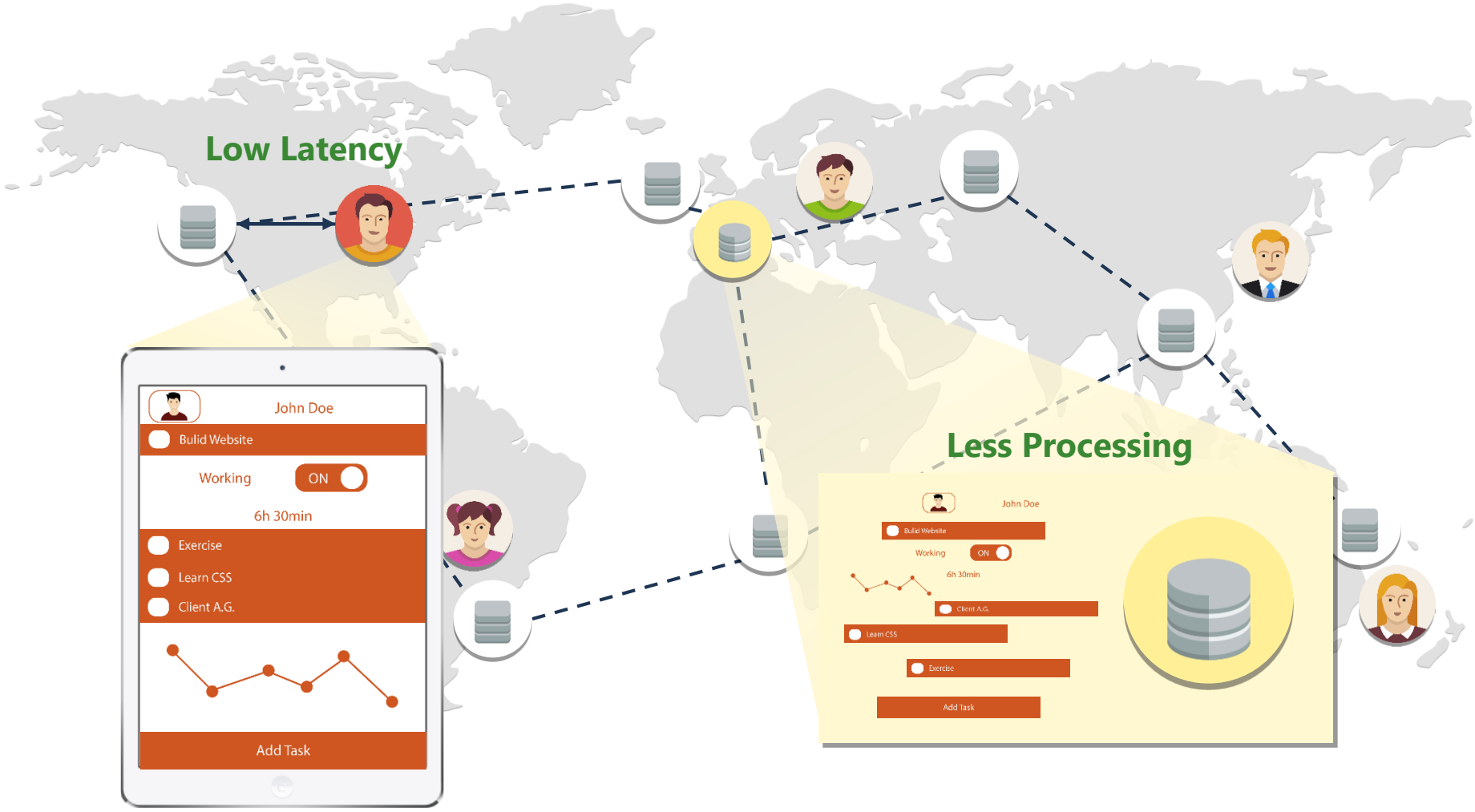

**2× Bandwidth** = Same Load Time

**½ Latency** ≈ ½ Load Time

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

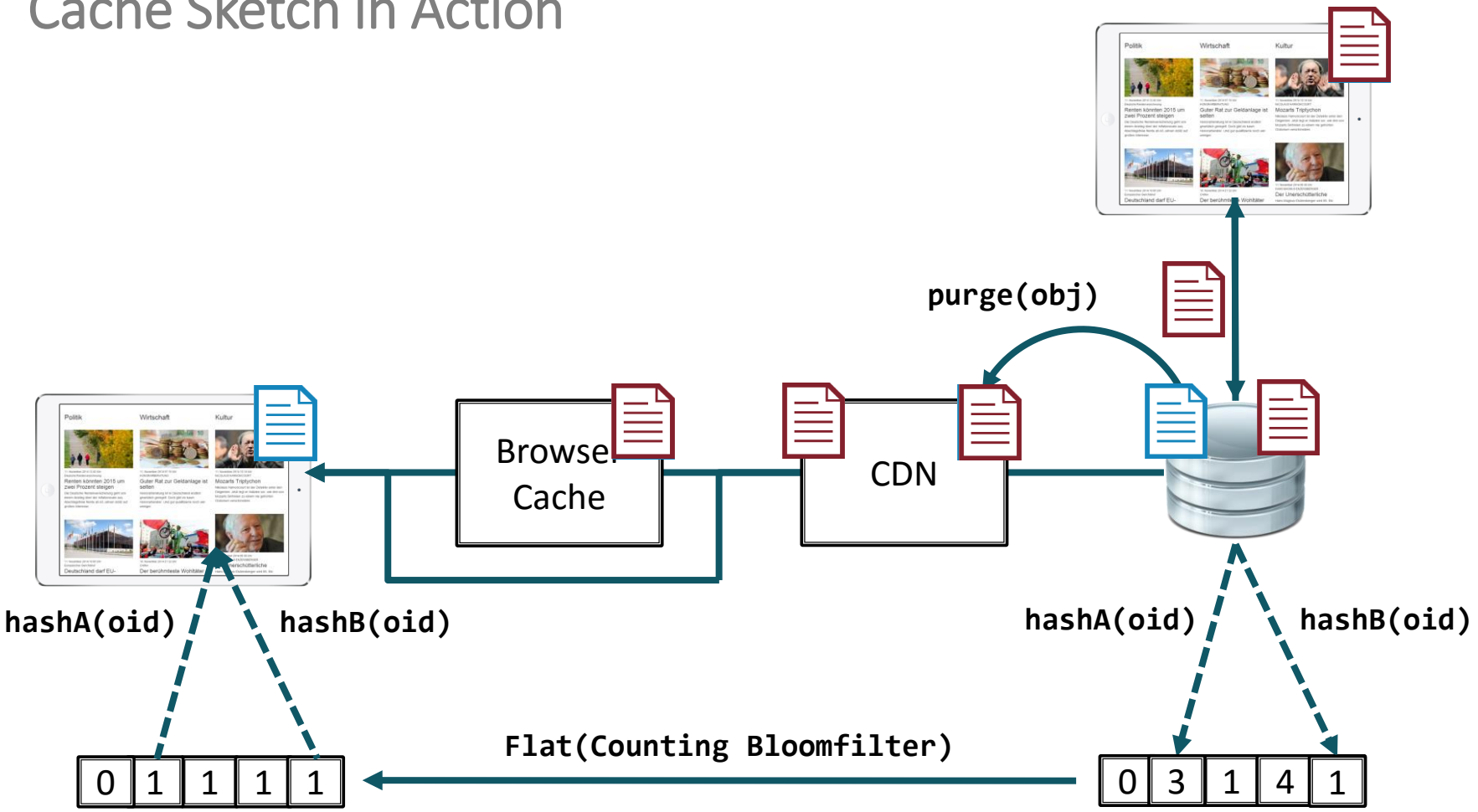
# Goal: Low-Latency for Dynamic Content

## By Serving Data from Ubiquitous Web Caches



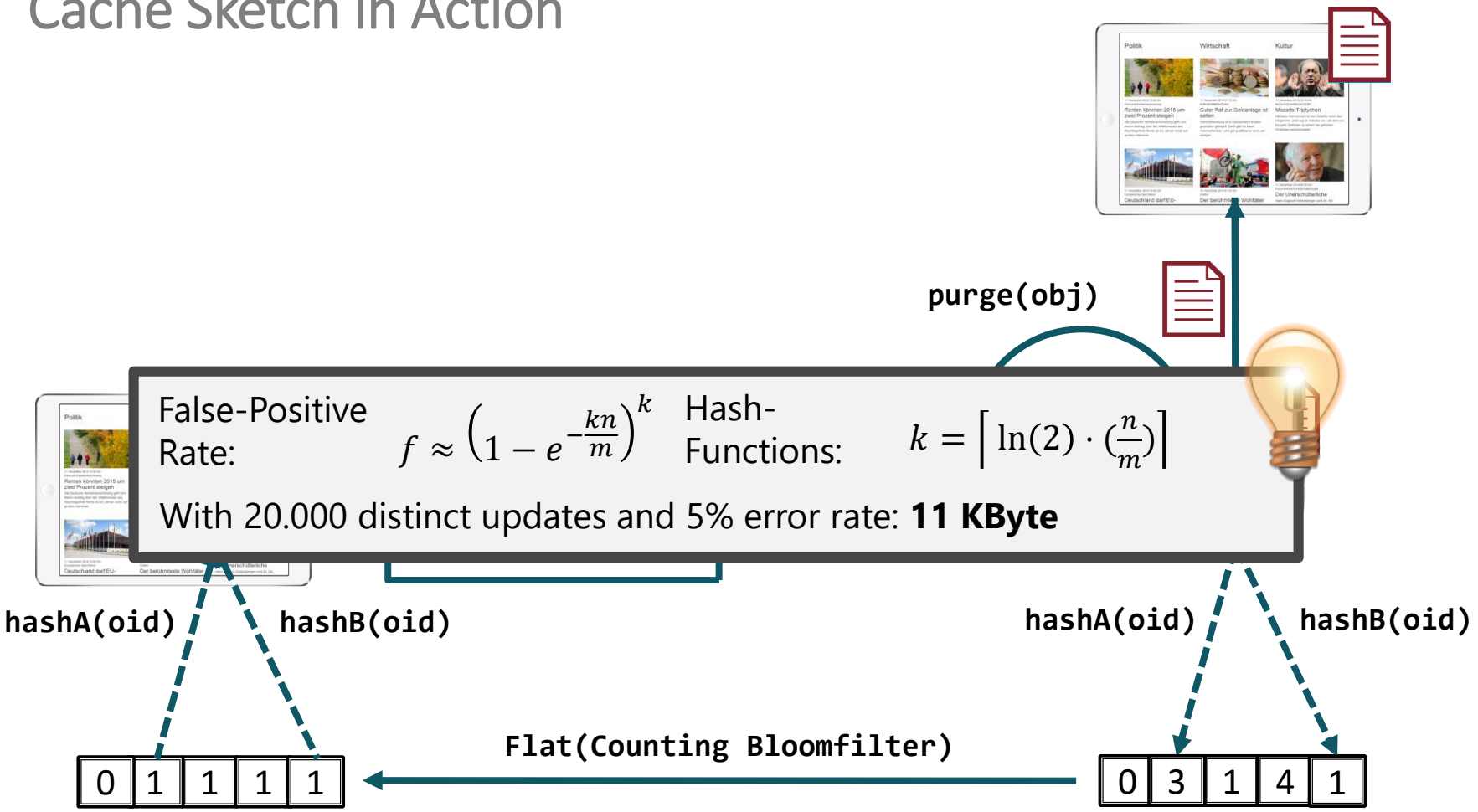
# Visually Explained

## Cache Sketch in Action



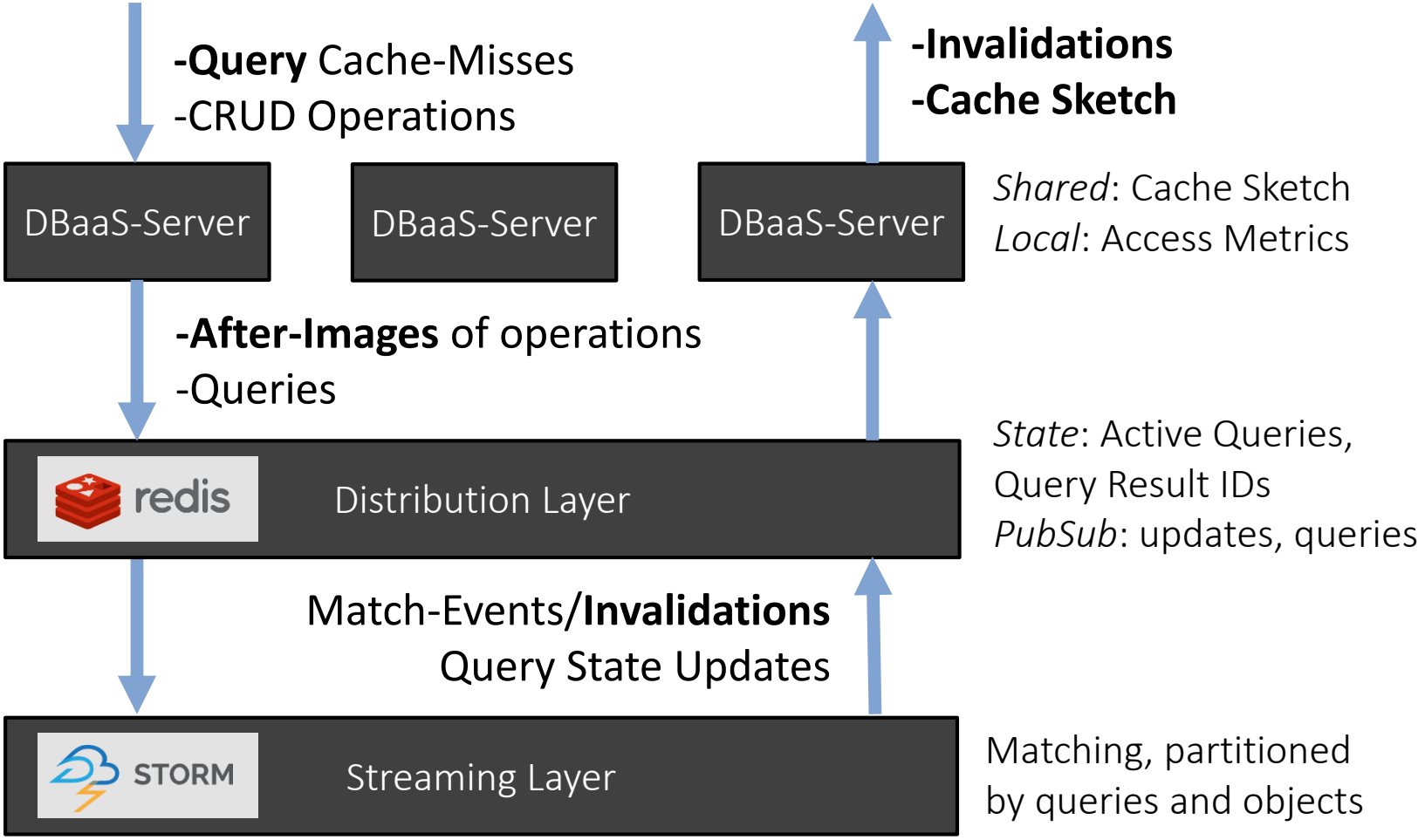
# Visually Explained

## Cache Sketch in Action



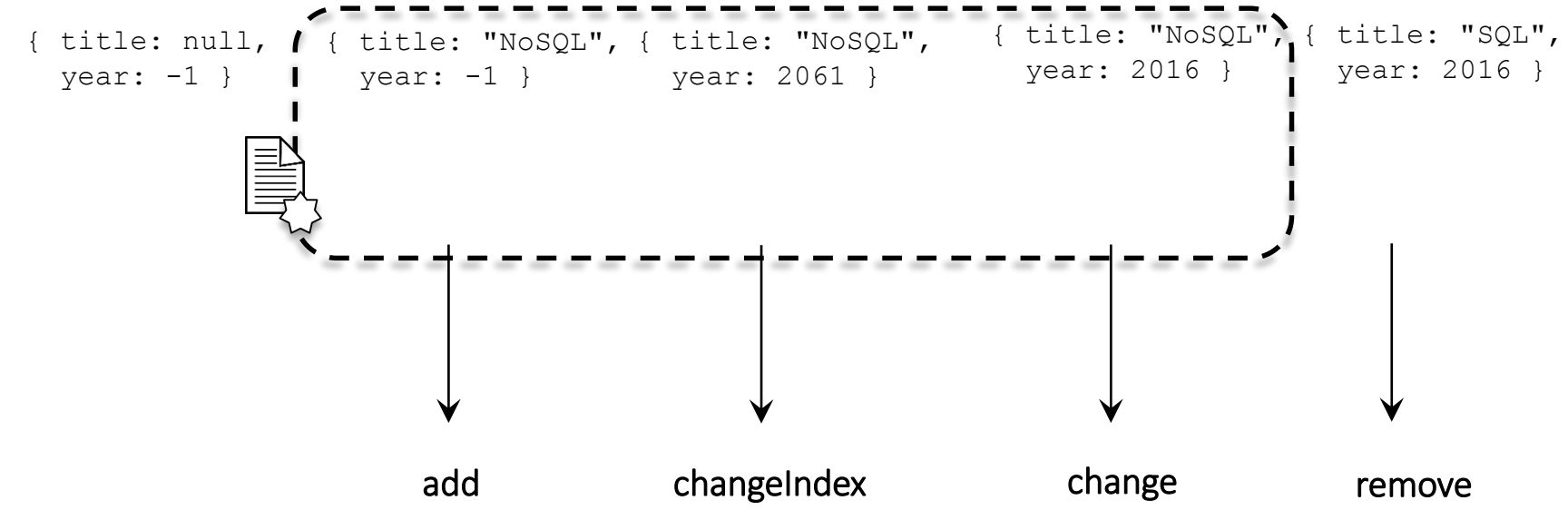
# Architecture

## Generalizing the Cache Sketch to query results



# Challenge: Maintaining Query Results

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



# Challenge: Maintaining Query Results

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

```
{ title: null,
  year: -1 }
```

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

```
{ title: "NoSQL",
  year: 2061 }
```

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

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

## The problem in code:

```
for (object in updates) {
    for (query in queries) {
        // is match?
        // was match?
    }
}
```

# State-of-the-Art: Firebase

## Real-Time Synchronization by Primary Keys


Update notifications for

- ✓ specific data items
- ✗ complex queries

https://firebase.googleblog.com/2016/07/have-you-met-realtime-database.html

# State-of-the-Art: Firebase

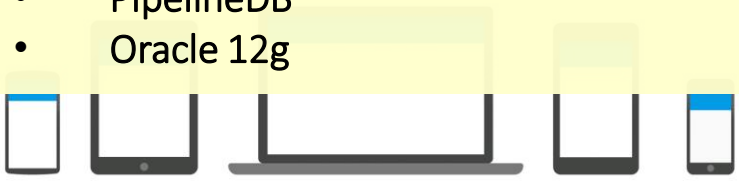
## Real-Time Synchronization by Primary Keys


Update notifications for
- ✓ specific data items
- ✗ complex queries

# Other Real-Time Systems

- RethinkDB
- Meteor
- Parse
- PipelineDB
- Oracle 12g

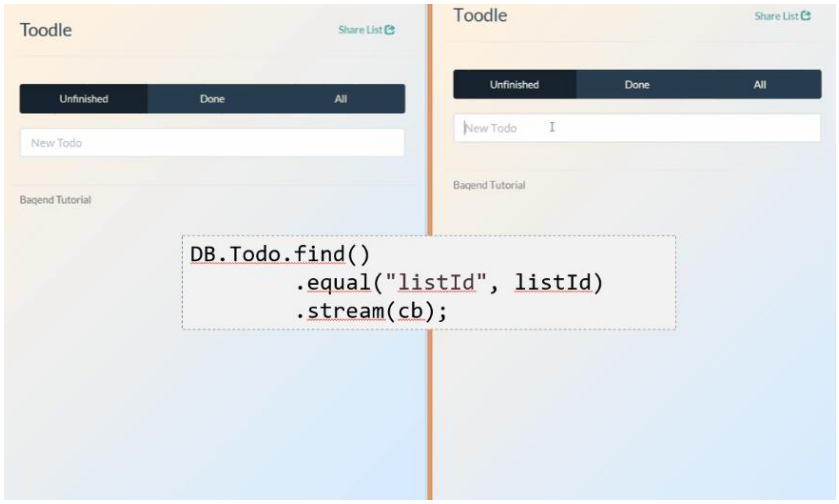
→ Problems:
- Limited query expressiveness
- Limited scalability
- Reduced availability
- High Latency



https://firebase.googleblog.com/2016/07/have-you-met-realtime-database.html

# Real-Time Queries

## More Than State Synchronization




TodoList Object

```
{
  id : "mylist",
  name : "Todo Liste",
  owner : {
    name : "Johnny",
    age : 12
  }
}
```

Embedded Object

TodoList Object

```
{
  id : "mylist",
  name : "Todo Liste",
  owner : "2333790852"
}
```

Reference

Owner Object

```
{
  id : "2333790852",
  name : "Johnny",
  age : 12
}
```

- ✓ Update notifications for complex queries

Also:

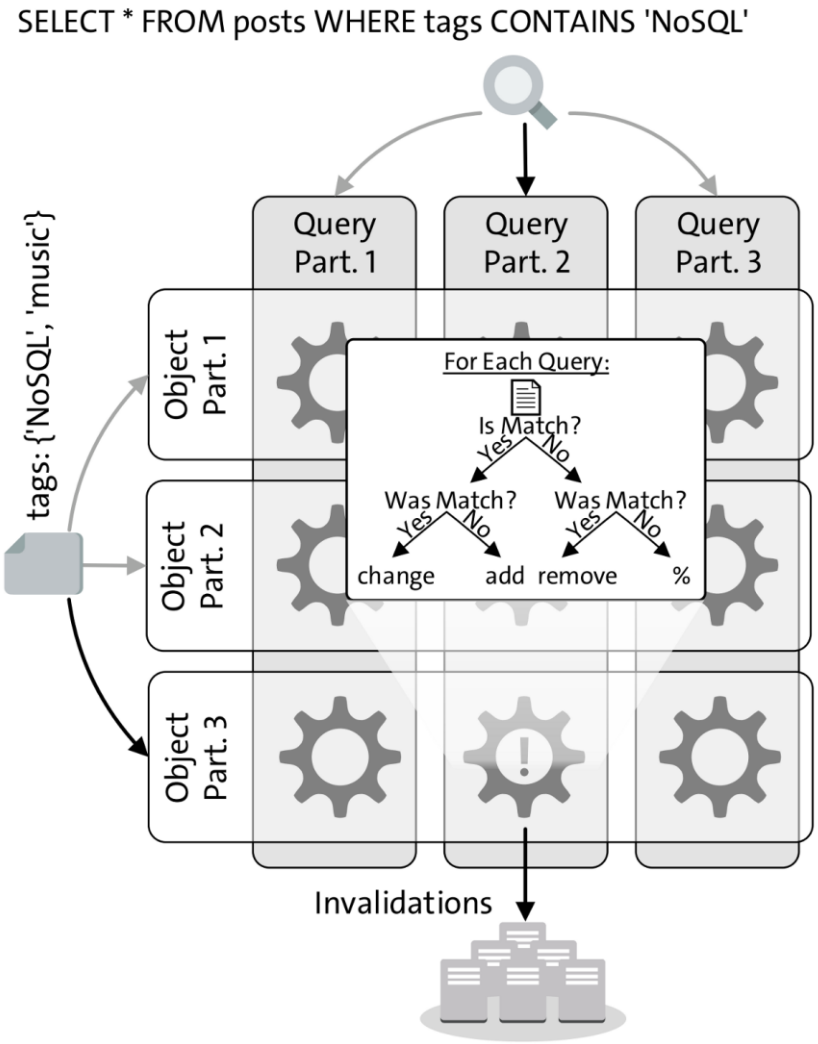
- ✓ Complex data model
- ✓ Sophisticated access control

# InvaliDB

## Distributing Real-Time Query Matching

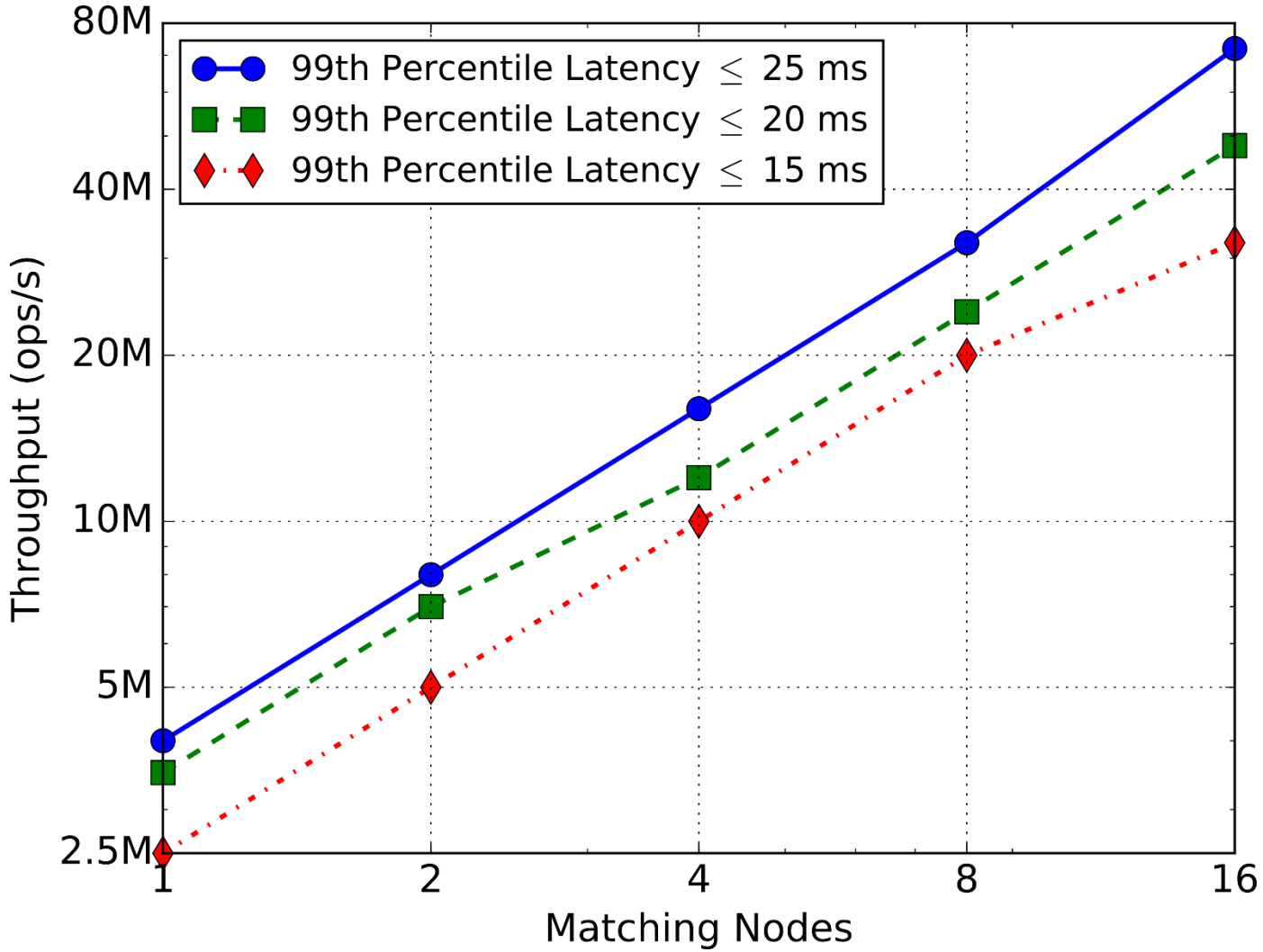
Design goals:

- Scalability
- Elasticity
- Low Latency
- Low Impact



# InvaliDB

## Distributing Real-Time Query Matching

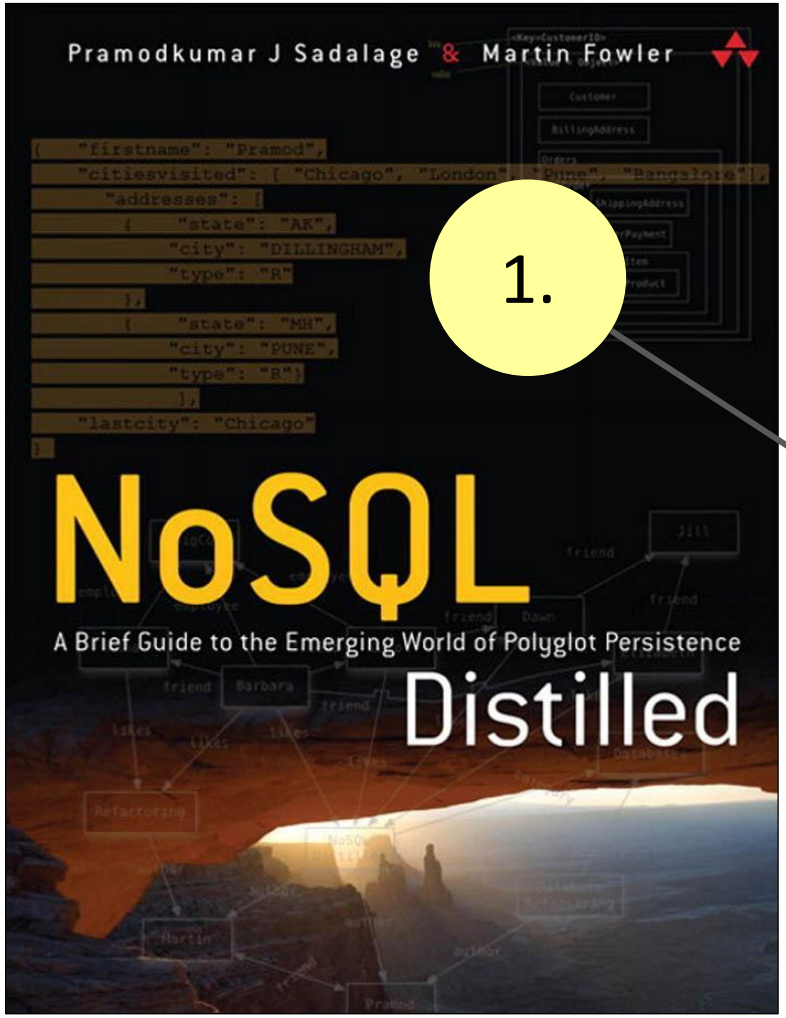


# Wrap-up

- One size doesn‘t fit all
- NoSQL can fill the gaps
- Hamburg is busy!

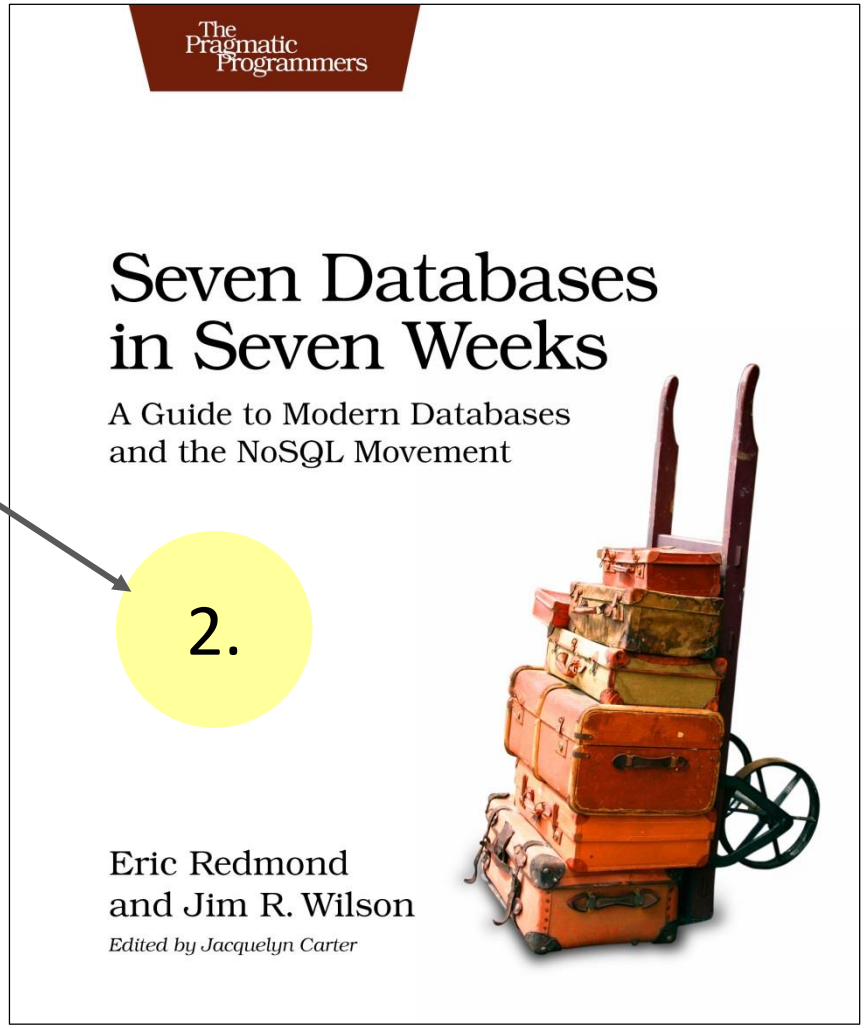
# Literature Recommendations

# Recommended Literature: NoSQL

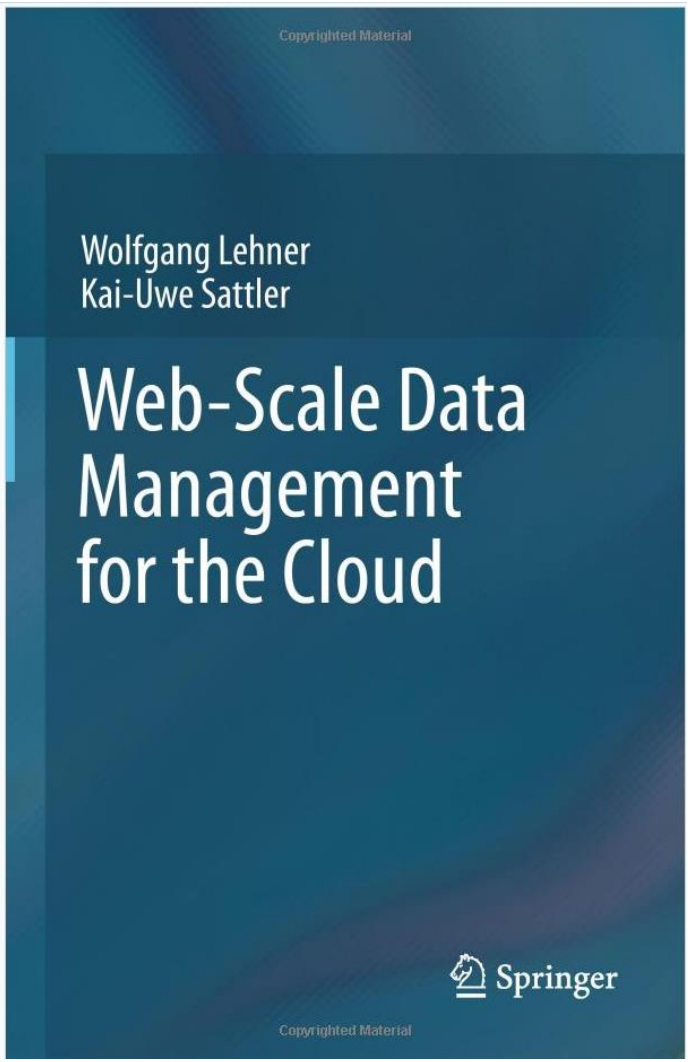


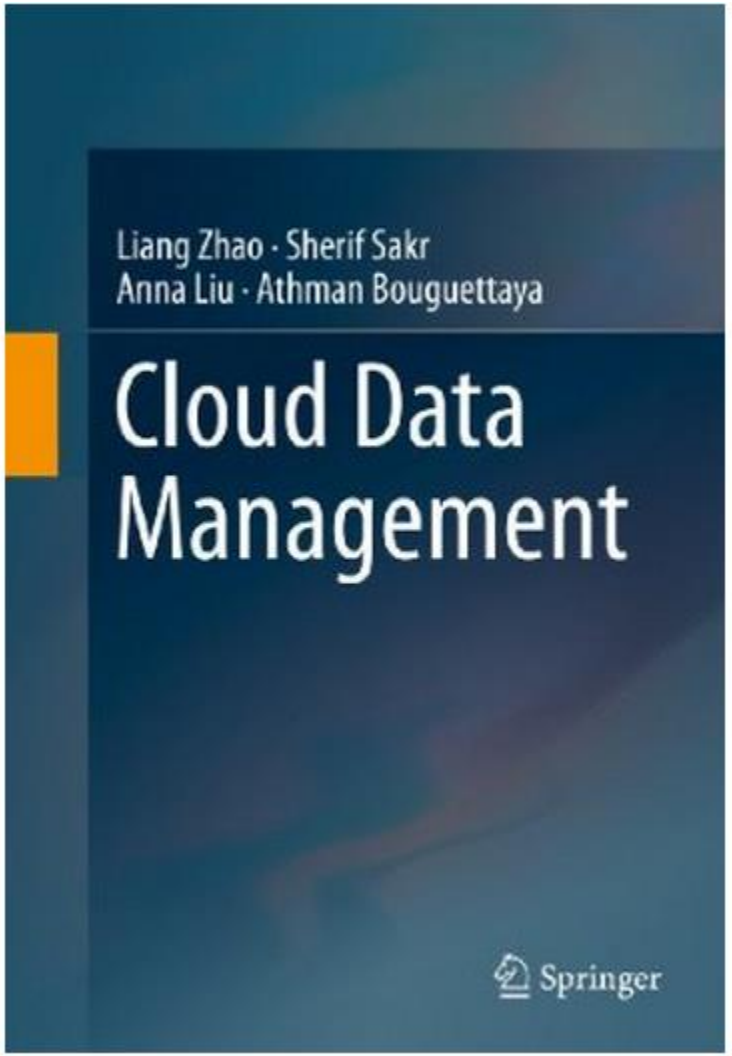

1.

2.



# Recommended Literature: Cloud DBs



# Recommended Literature: Blogs

myNoSQL

http://nosql.mypopescu.com/

NoSQL Weekly

http://www.nosqlweekly.com/


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

Hacking, Distributed

http://hackingdistributed.com/

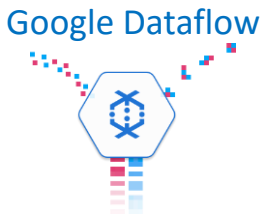
DBMS2

http://www.dbms2.com/

High Scalability

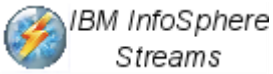
http://highscalability.com/

# Big Data Analytics





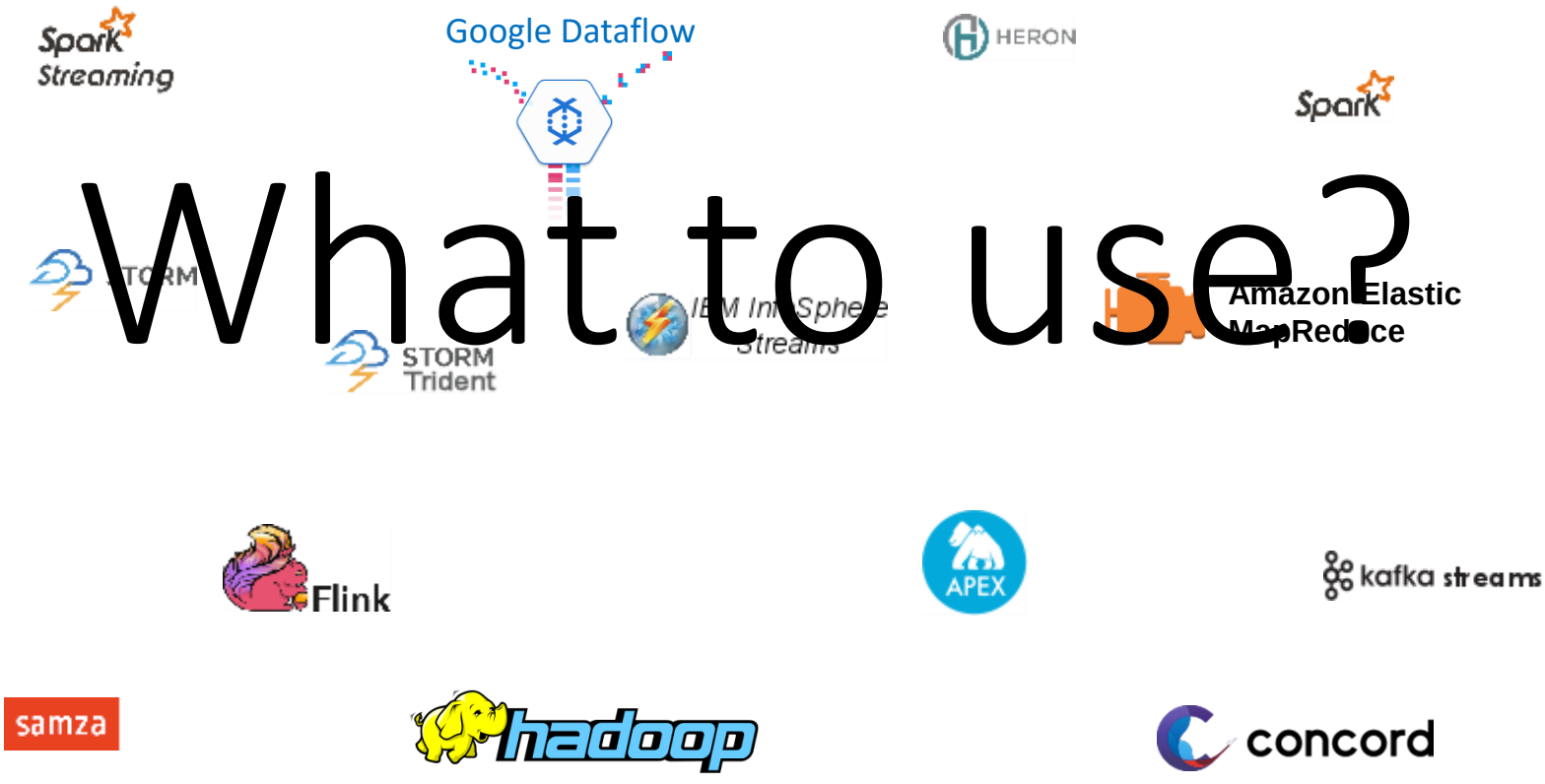




**Amazon Elastic MapReduce**



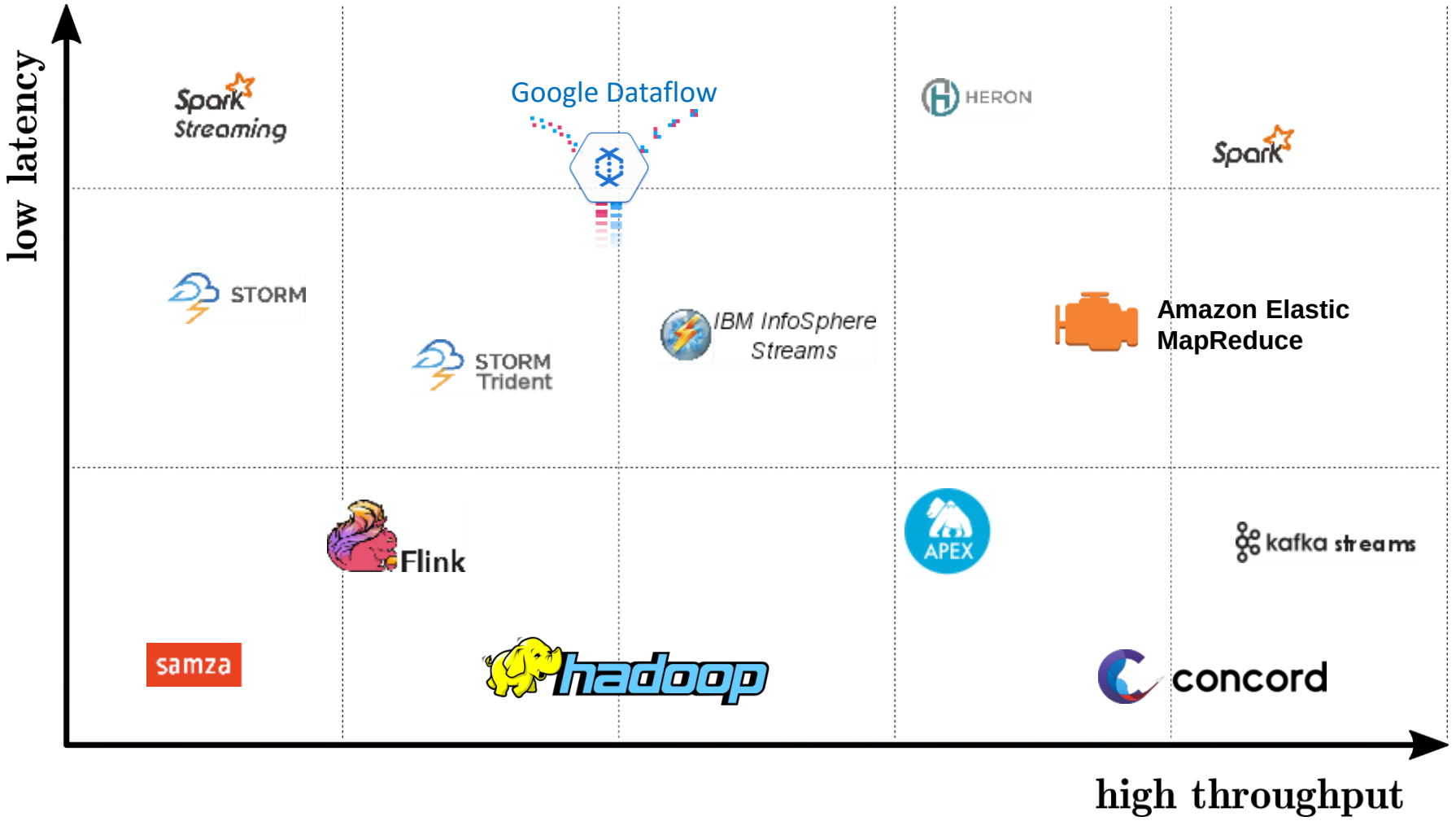



# Big Data Analytics

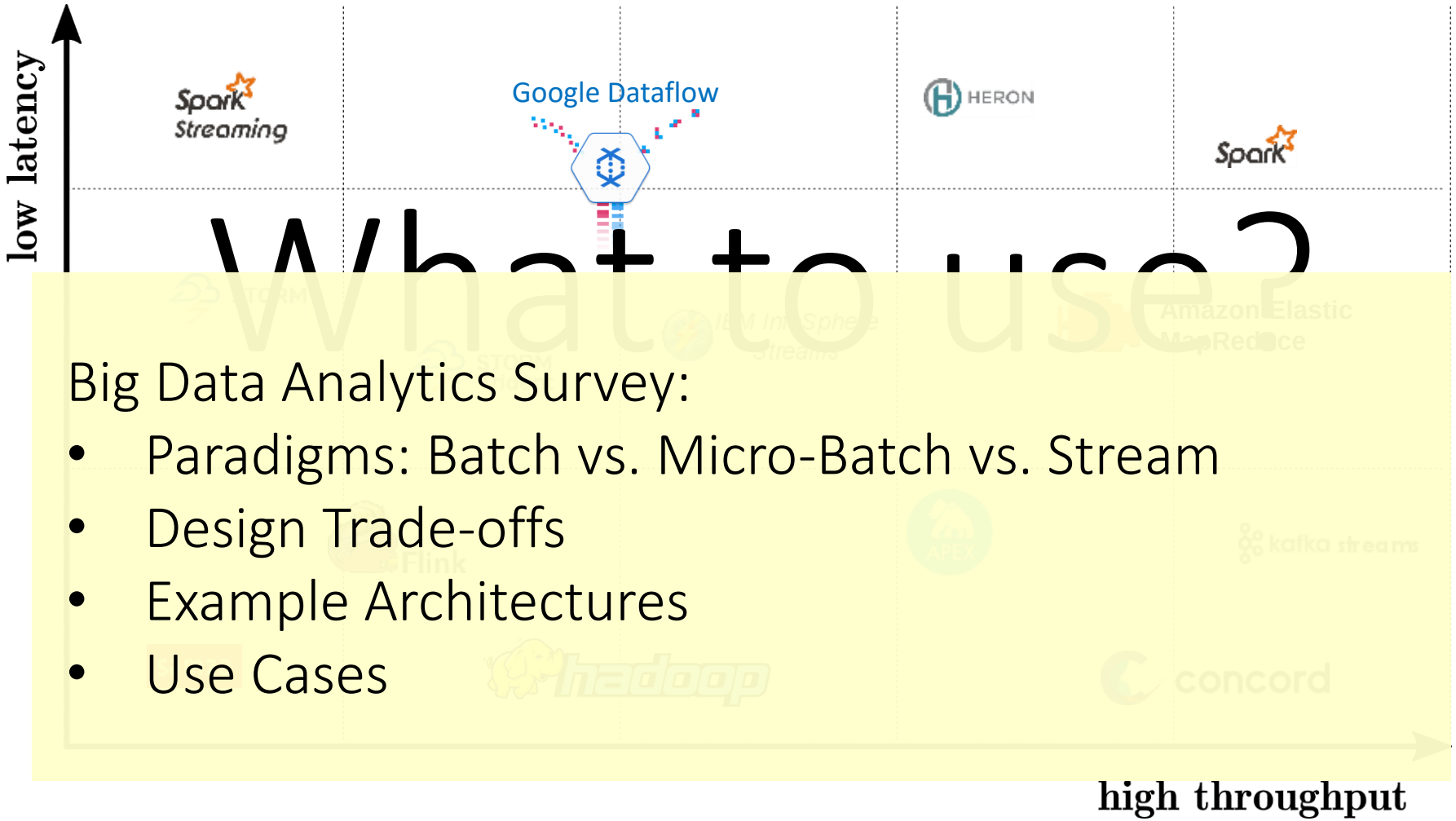



What to use?

# Big Data Analytics



# Big Data Analytics




Big Data Analytics Survey:

- Paradigms: Batch vs. Micro-Batch vs. Stream
- Design Trade-offs
- Example Architectures
- Use Cases

# Recommended Literature: Our surveys!

## NoSQL Databases: a Survey and Decision Guidance

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


### TL;DR

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

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


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

Read them at blog.baqend.com!