Scalable Cloud Data Management

Felix Gessert, Norbert Ritter, Wolfram Wingerath

ritter@informatik.uni-hamburg.de

SummerSOC 2019

Universität Hamburg

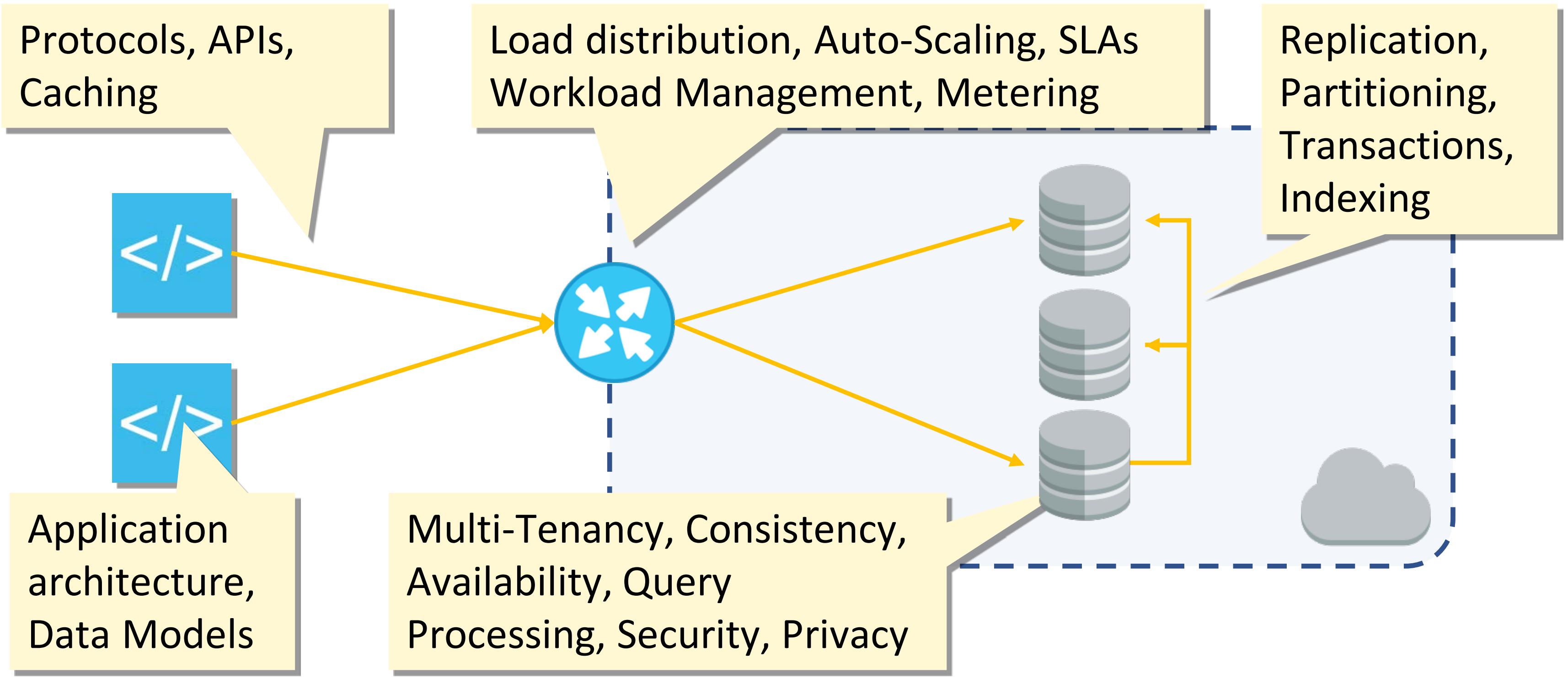
DBIS

Baqend

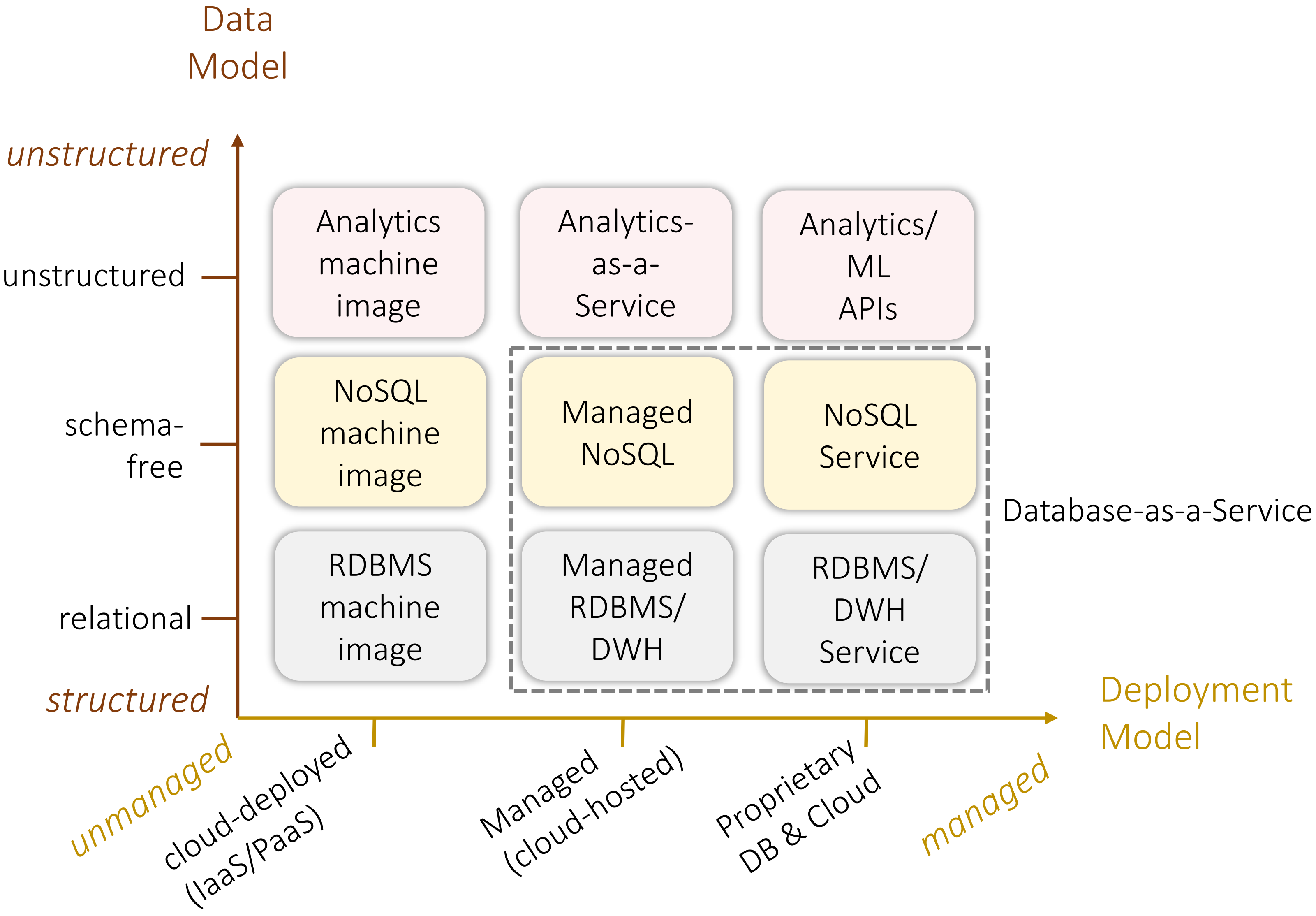
www.baqend.com

Cloud Data Management

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

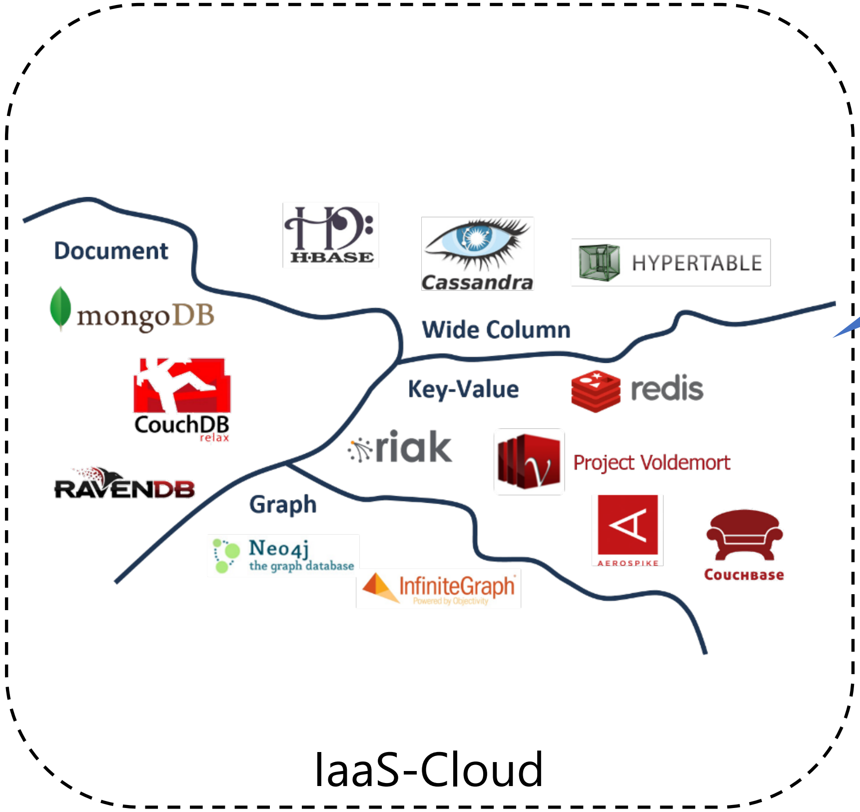


Cloud-Database Models



Cloud-Deployed Database

Database-image provisioned in IaaS/PaaS-cloud



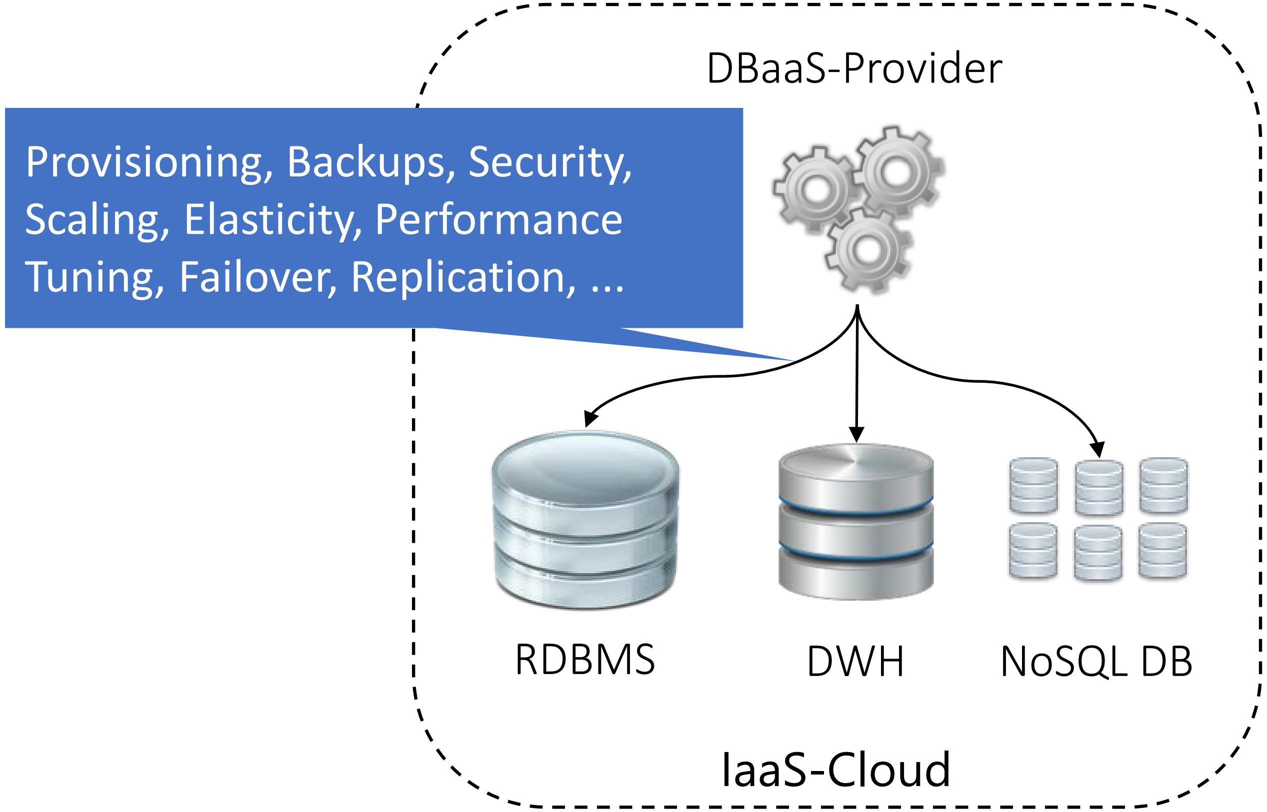
IaaS/PaaS deployment of database system

Does not solve:

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

Managed RDBMS/DWH/NoSQL DB

Cloud-hosted database



RDBMS

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

NoSQL DB

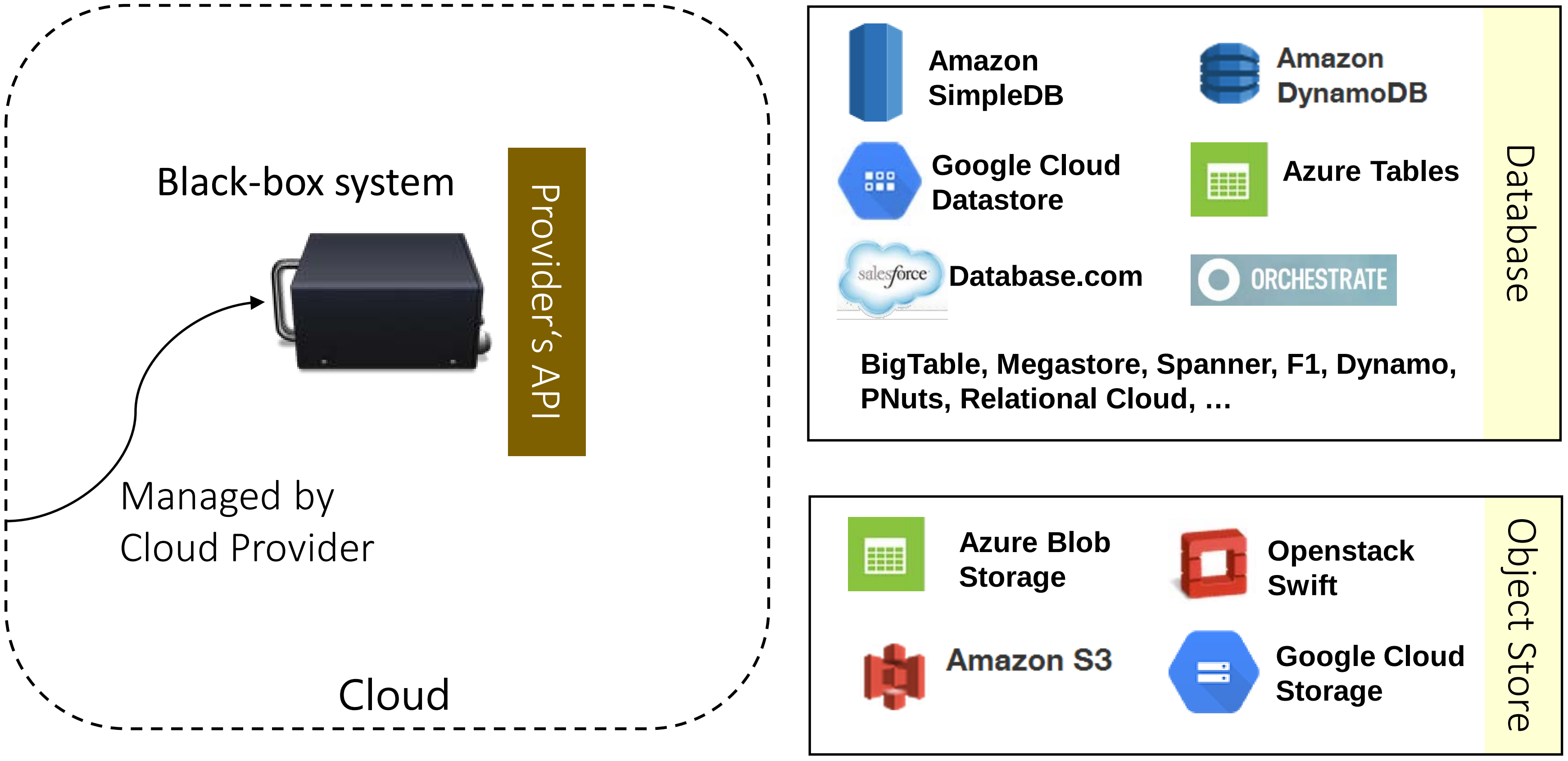
- Amazon ElastiCache
- mongoHQ
- mongolab
- Cloudant an IBM® Company
- redis
- instaclustr
- bonsai
- Iris Couch

DWH

- Amazon Redshift

Proprietary Cloud Database

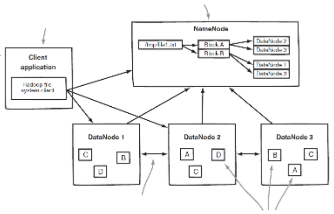
Designed for and deployed in vendor-specific cloud environment



Analytics-as-a-Service

Analytic frameworks and machine learning with service APIs

Analytics Cluster



Provisioning,
Data Ingest

Cloud

Amazon Elastic MapReduce

Azure HDInsight

Analytics

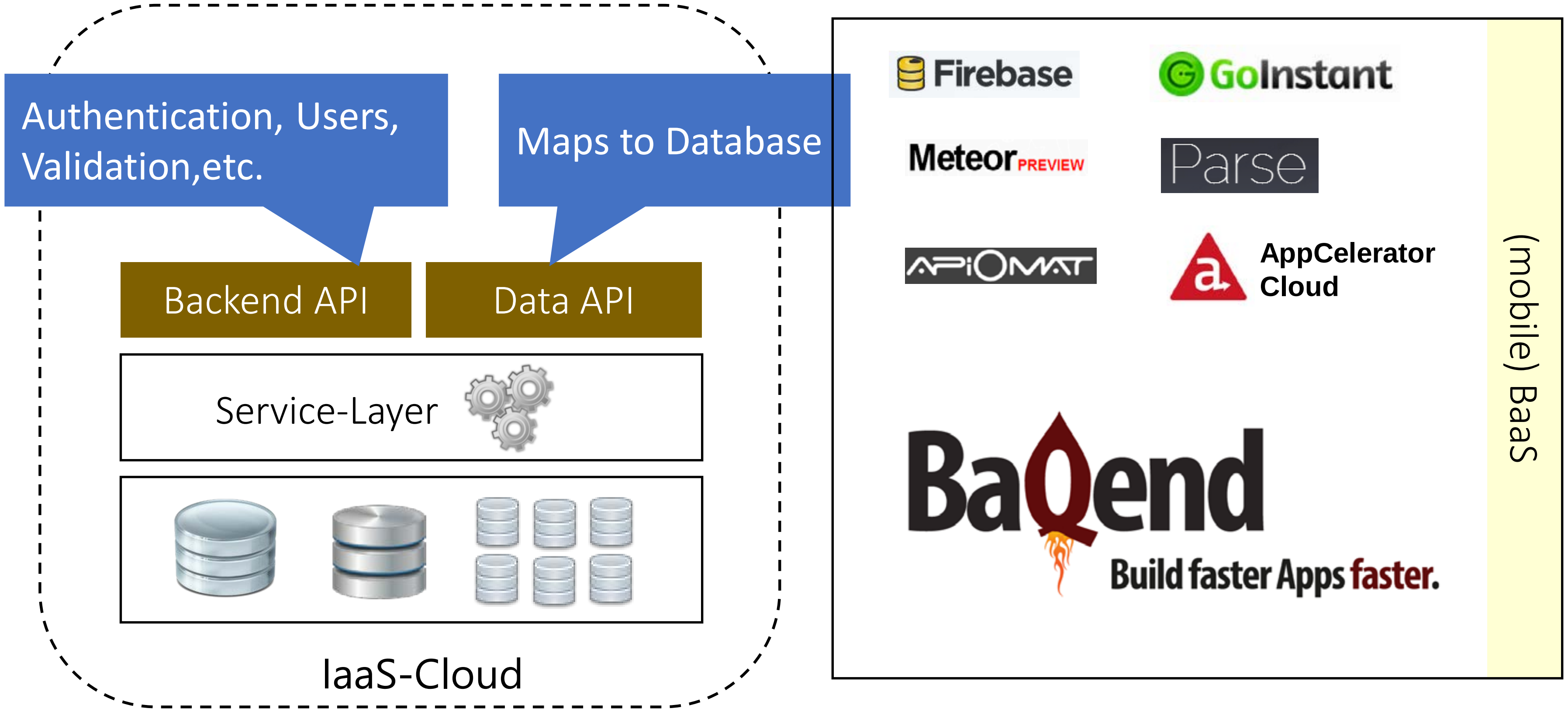
Google BigQuery

Google Prediction API

ML

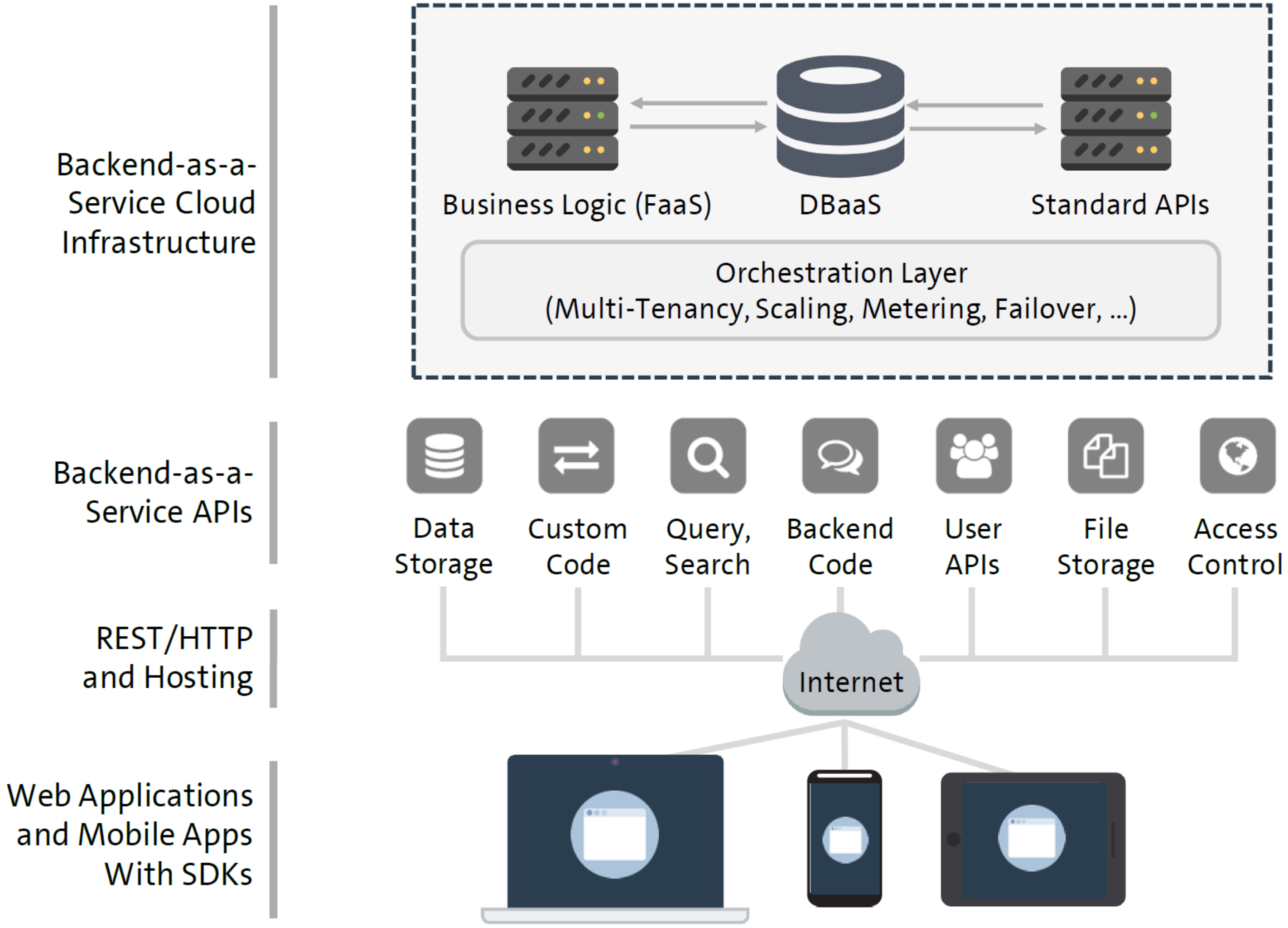
Backend-as-a-Service

DBaaS with embedded custom and predefined application logic



Backend-as-a-Service

DBaaS with embedded custom and predefined application logic



Backend-as-a-Service

DBaaS with embedded custom and predefined application logic

- rather recent trend
  - progress currently driven by industry projects
    (similarly to early cloud computing and big data processing)
  - structured research yet to be established
- most comfortable approach for applications
- but many unsolved problems
  - latency challenge: all clients access the service via high-latency WAN
  - persistence on top of one single database technology
    - service/NoSQL-DBS selection problem
    - usually, tenants colocated on a shared database cluster
      → database system configuration (e.g., the replication protocol)
      prescribes the guarantees for each tenant

Service Level Agreements (SLAs)

Specification of Application/Tenant Requirements

SLA

Technical Part

1. SLO
2. SLO
3. SLO

Legal Part

1. Fees
2. Penalties

Service Level Objectives:

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

Service Level Agreements

Expressing application requirements

Functional Service Level Objectives

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

Non-Functional Service Level Objectives

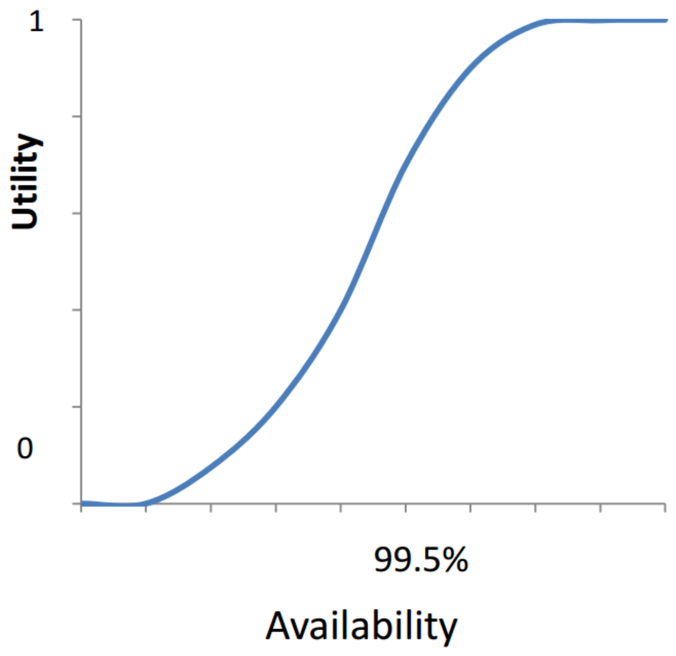
- Guarantee a certain *quality of service* (QoS)
- Determined by database system and service provider
- *Examples*:
  - **Continuous**: response time (latency), throughput
  - **Binary**: Elasticity, Read-your-writes

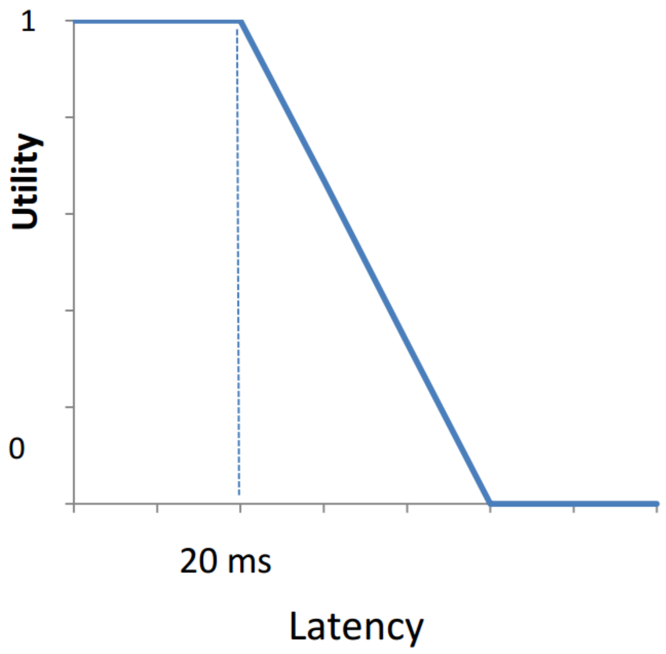
Service Level Objectives

Making SLOs measurable through utilities

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

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

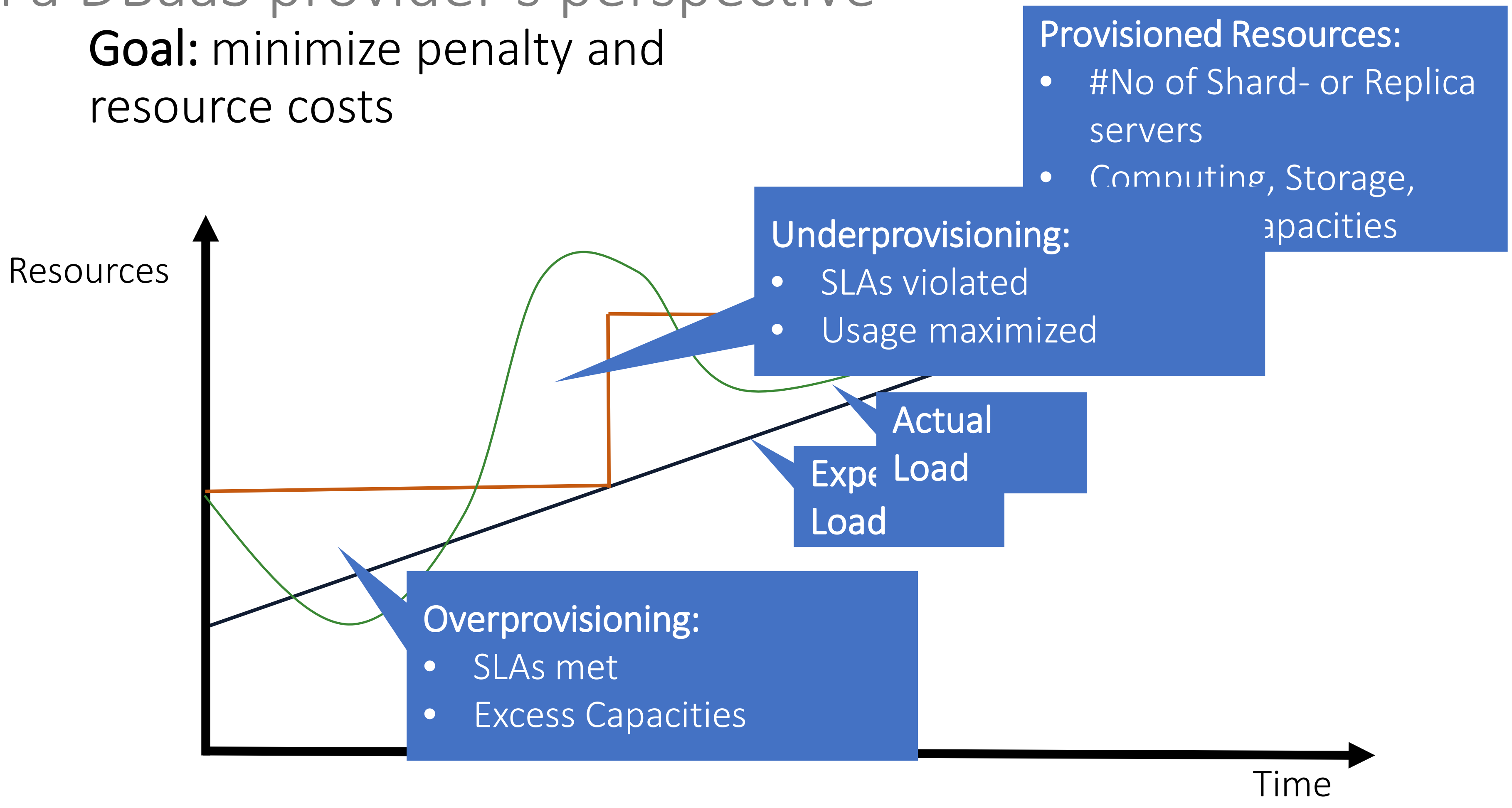




Resource & Capacity Planning

From a DBaaS provider‘s perspective

Goal: minimize penalty and resource costs



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

SLAs in the wild

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

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

Managed NoSQL Services

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

Proprietary Database Services

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

Our SCDM Approach

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

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. Aim at fully managed Backend (BaaS)

1 Introduction

Traditional relational database management systems (RDBMSs) provide

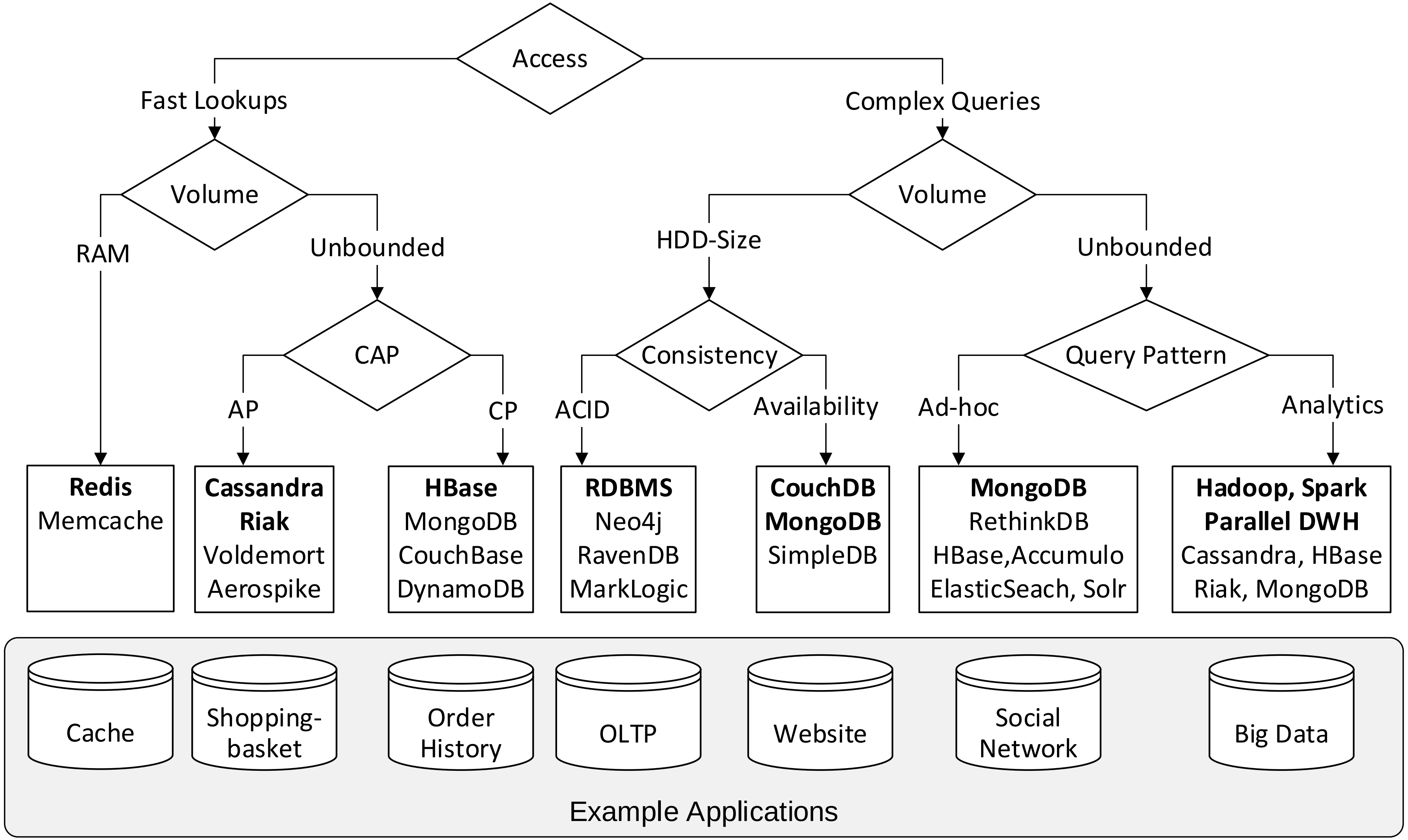
2. Exploit modern (NoSQL) Database Technology

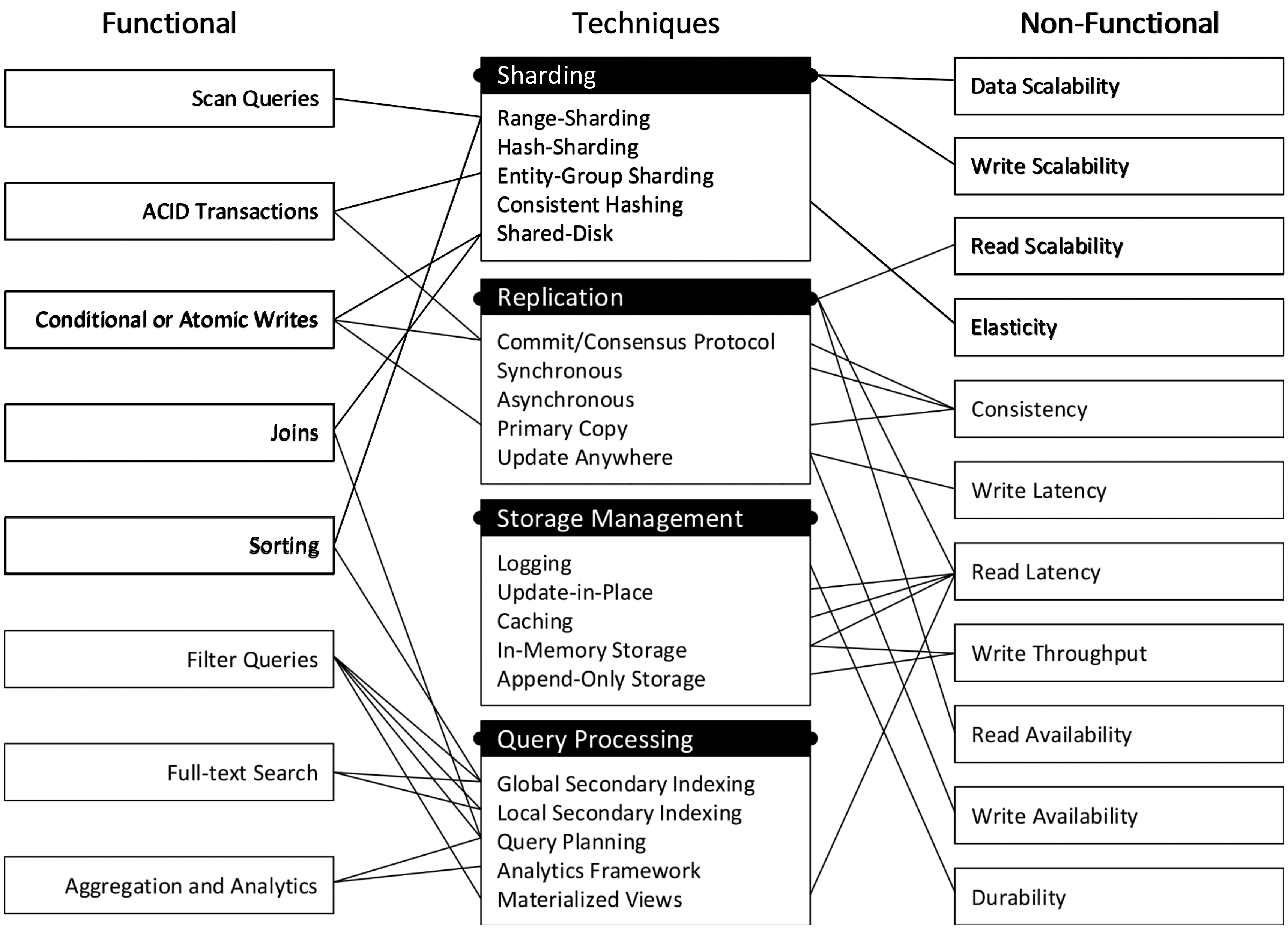
so vast that it cannot be stored or processed by traditional database solutions. User-generated content in social networks or data retrieved from large sensor networks are only two examples of this phenomenon commonly referred to as **Big Data** [35]. A class of novel data storage systems able to cope 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

NoSQL Decision Tree




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

System Properties

According to the NoSQL Toolbox

- For fine-grained system selection:

	Functional Requirements							
	Scan Queries	**ACID Transactions**	**Conditional Writes**	**Joins**	**Sorting**	**Filter Query**	**Full-Text Search**	**Analytics**
Mongo	x		x		x	x	x	x
Redis	x	x	x					
HBase	x		x		x			x
Riak							x	x
Cassandra	x		x		x		x	x
MySQL	x	x	x	x	x	x	x	x

System Properties

According to the NoSQL Toolbox

- For fine-grained system selection:

	Non-functional Requirements										
	Data Scalability	Write Scalability	Read Scalability	Elasticity	Consistency	Write Latency	Read Latency	Write Throughput	Read Availability	Write Availability	Durability
Mongo	x	x	x		x	x	x		x		x
Redis			x		x	x	x	x	x		x
HBase	x	x	x	x	x	x		x			x
Riak	x	x	x	x		x	x	x	x	x	x
Cassandra	x	x	x	x		x		x	x	x	x
MySQL			x		x						x

System Properties

According to the NoSQL Toolbox

- For fine-grained system selection:

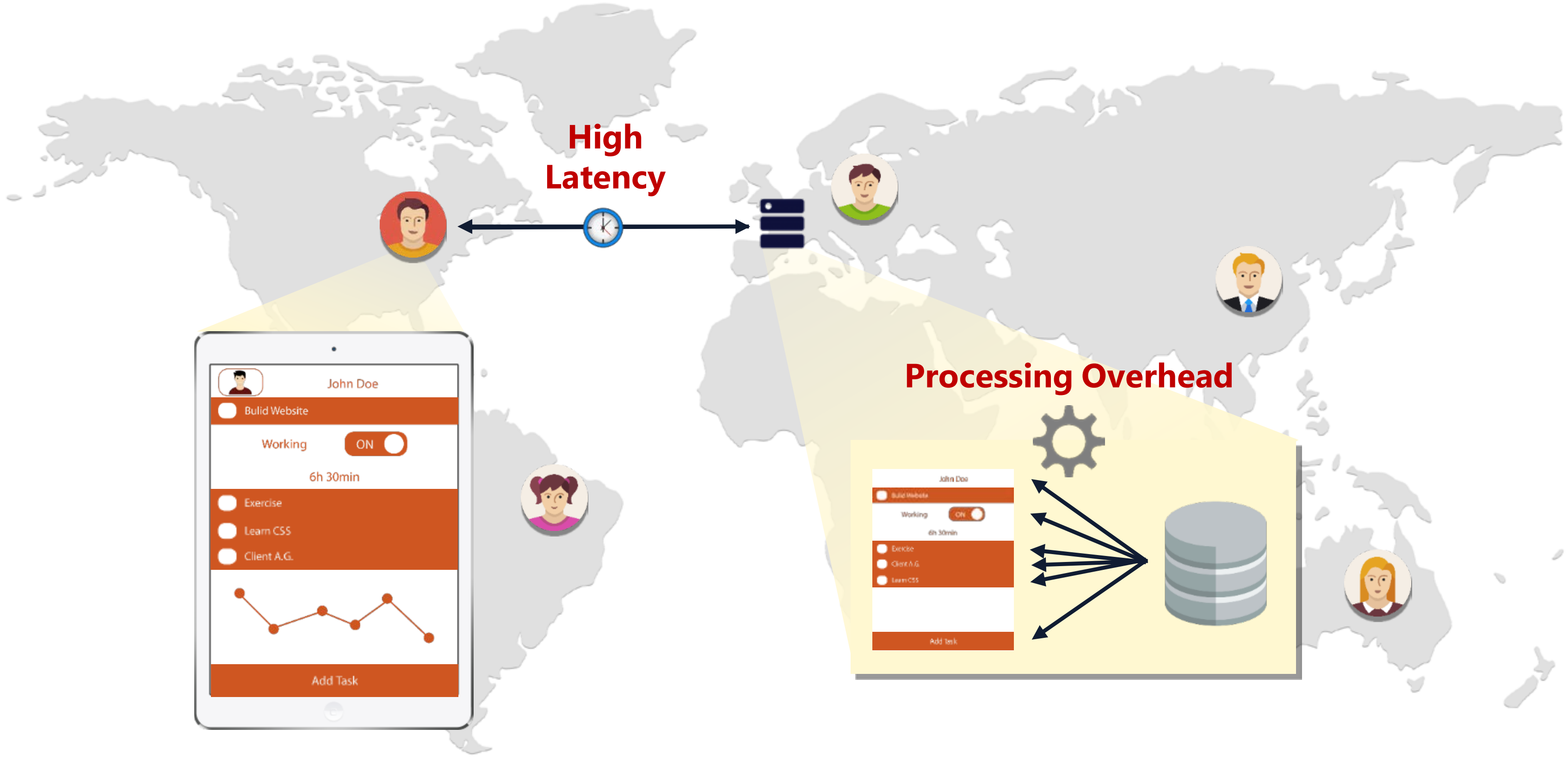
	Techniques																			
	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared-Disk	Transaction Protocol	Sync. Replication	Async. Replication	Primary Copy	Update Anywhere	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage	Global Indexing	Local Indexing	Query Planning	Analytics Framework	Materialized Views
Mongo	x	x					x	x	x		x		x	x	x		x	x	x	
Redis								x	x		x		x							
HBase	x						x		x		x		x		x					
Riak		x		x				x		x	x	x	x			x	x		x	
Cassandra		x		x				x		x	x		x		x	x	x			x
MySQL					x			x	x		x	x	x				x	x		

NoSQL DBS support applications in achieving horizontal scalability and backend performance through differentiated trade-offs in functionality and consistency!

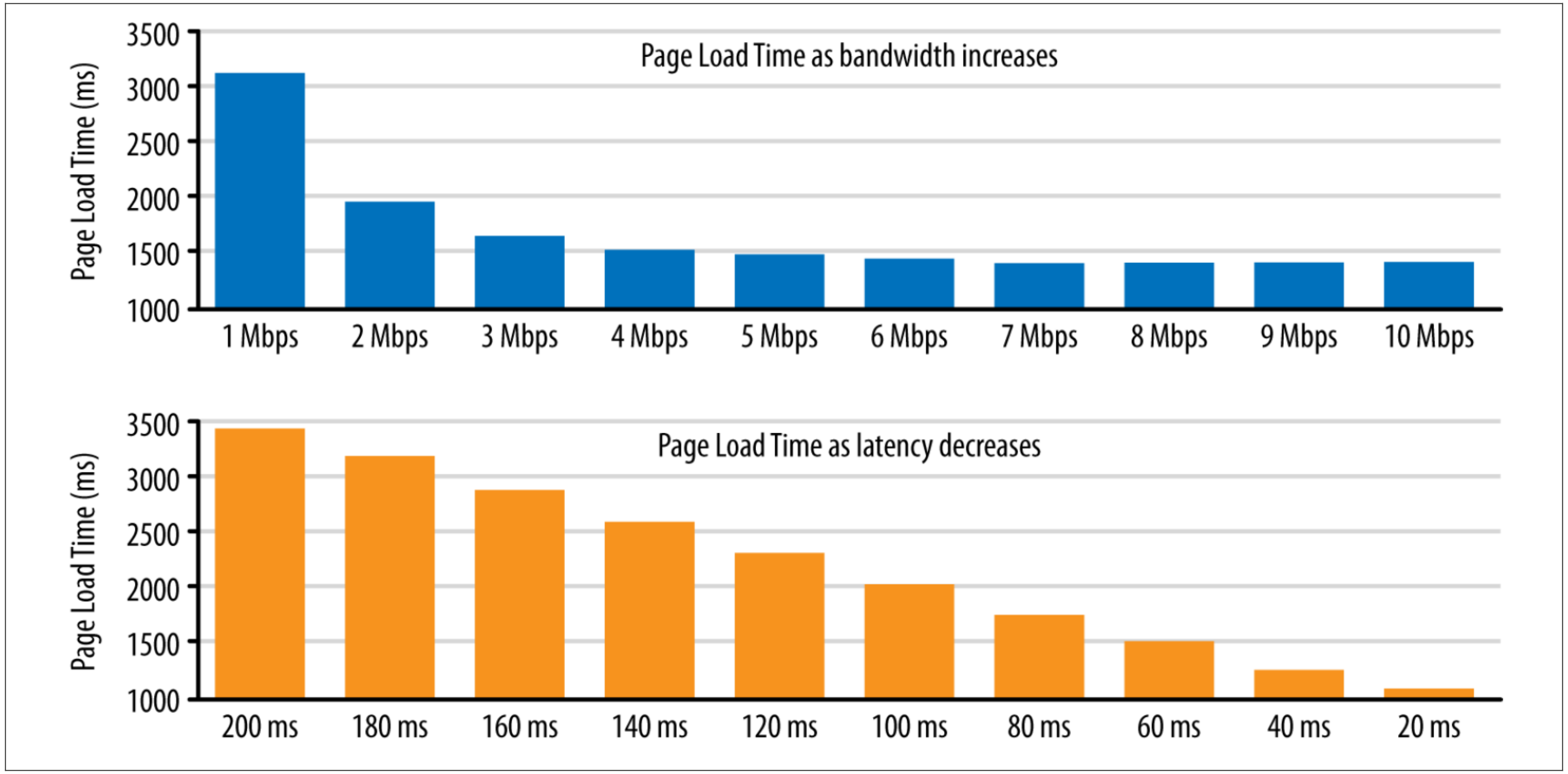
3. Consider the entire path
from the (mobile) application
through the net
to the data backend!

Challenge: Slow Websites / mobile Apps

Two Bottlenecks: Latency and Processing

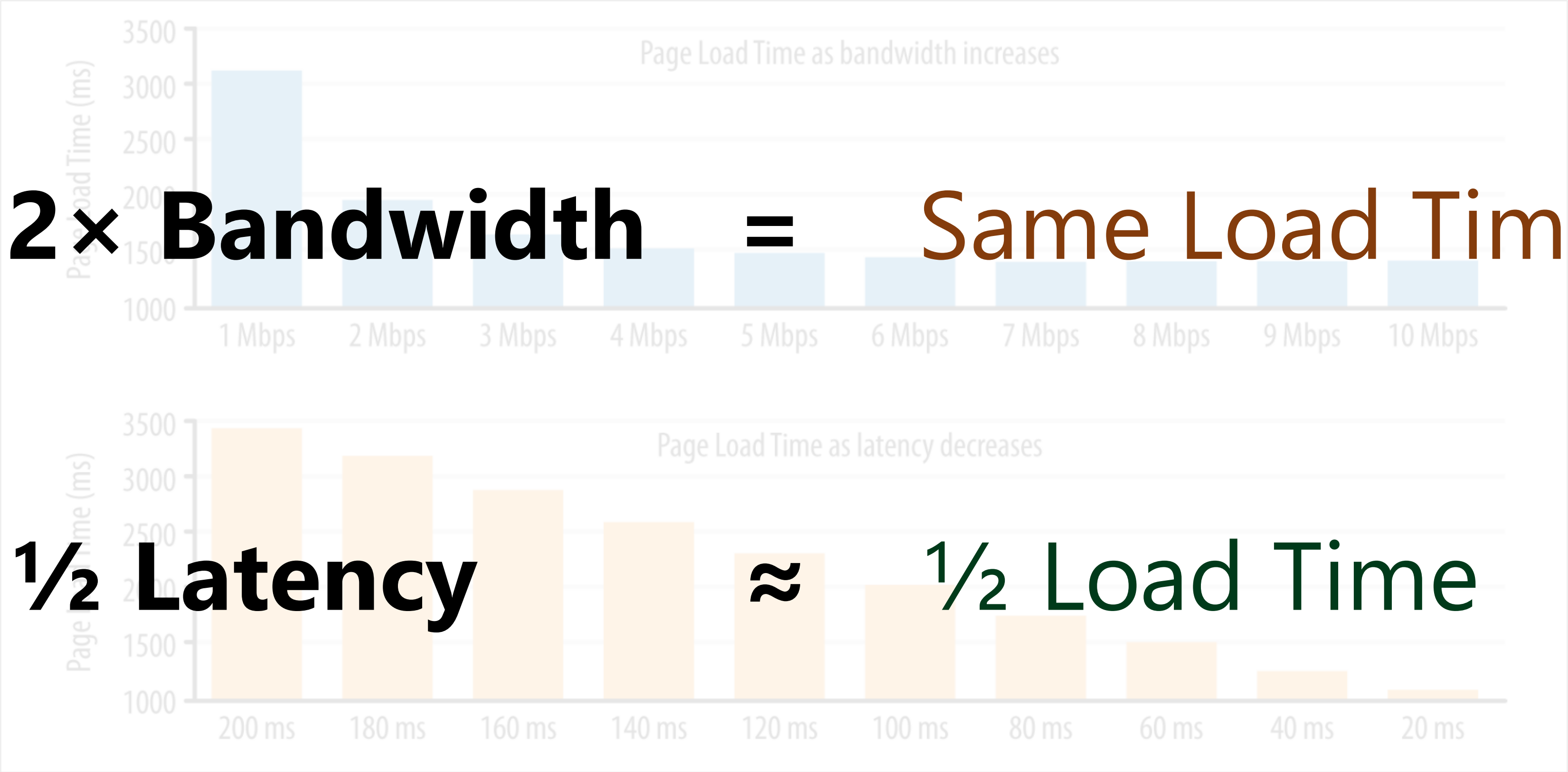


Network Latency: Impact



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

Network Latency: Impact



2× Bandwidth = Same Load Time

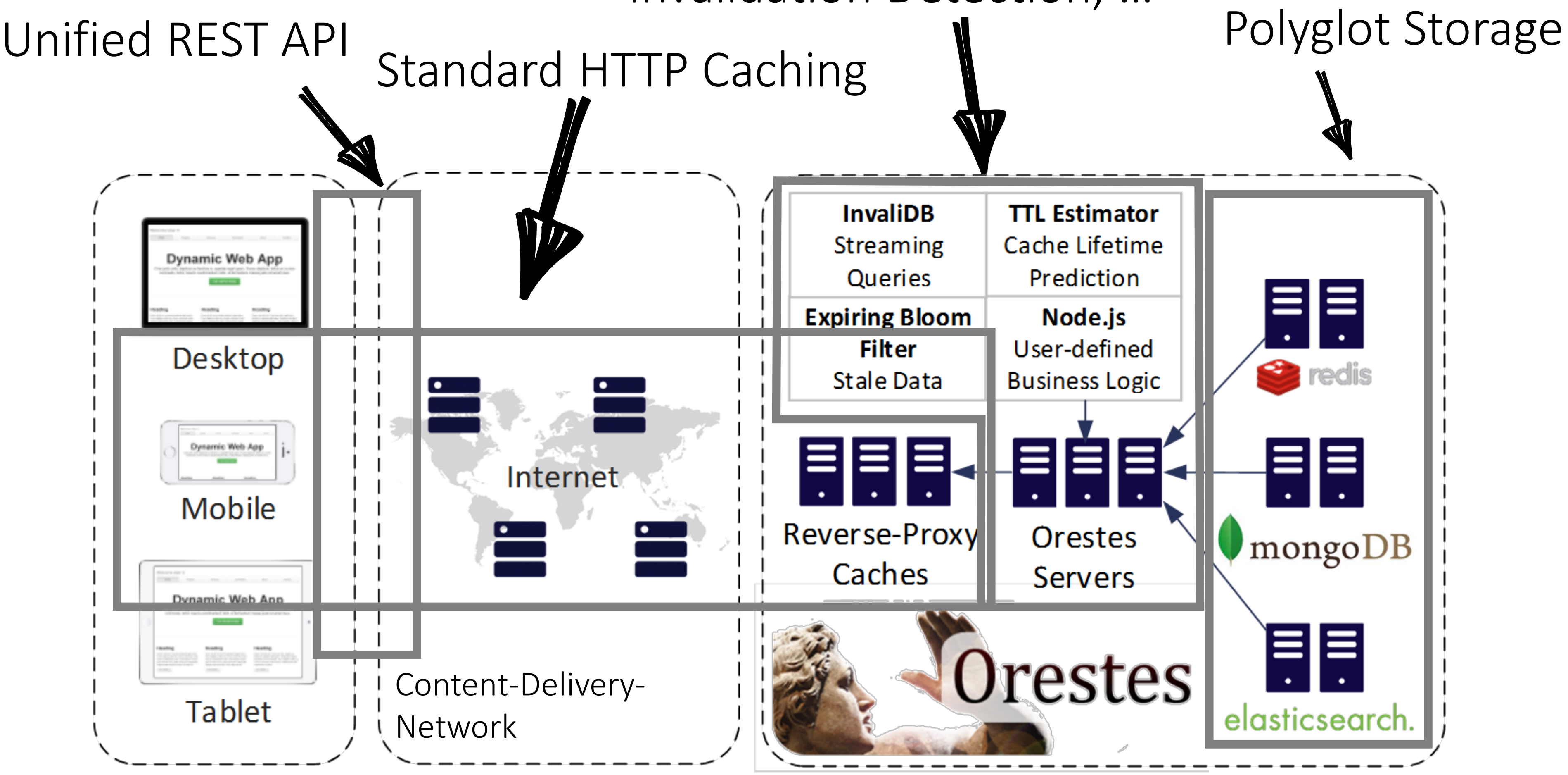
½ Latency ≈ ½ Load Time

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

Orestes Architecture

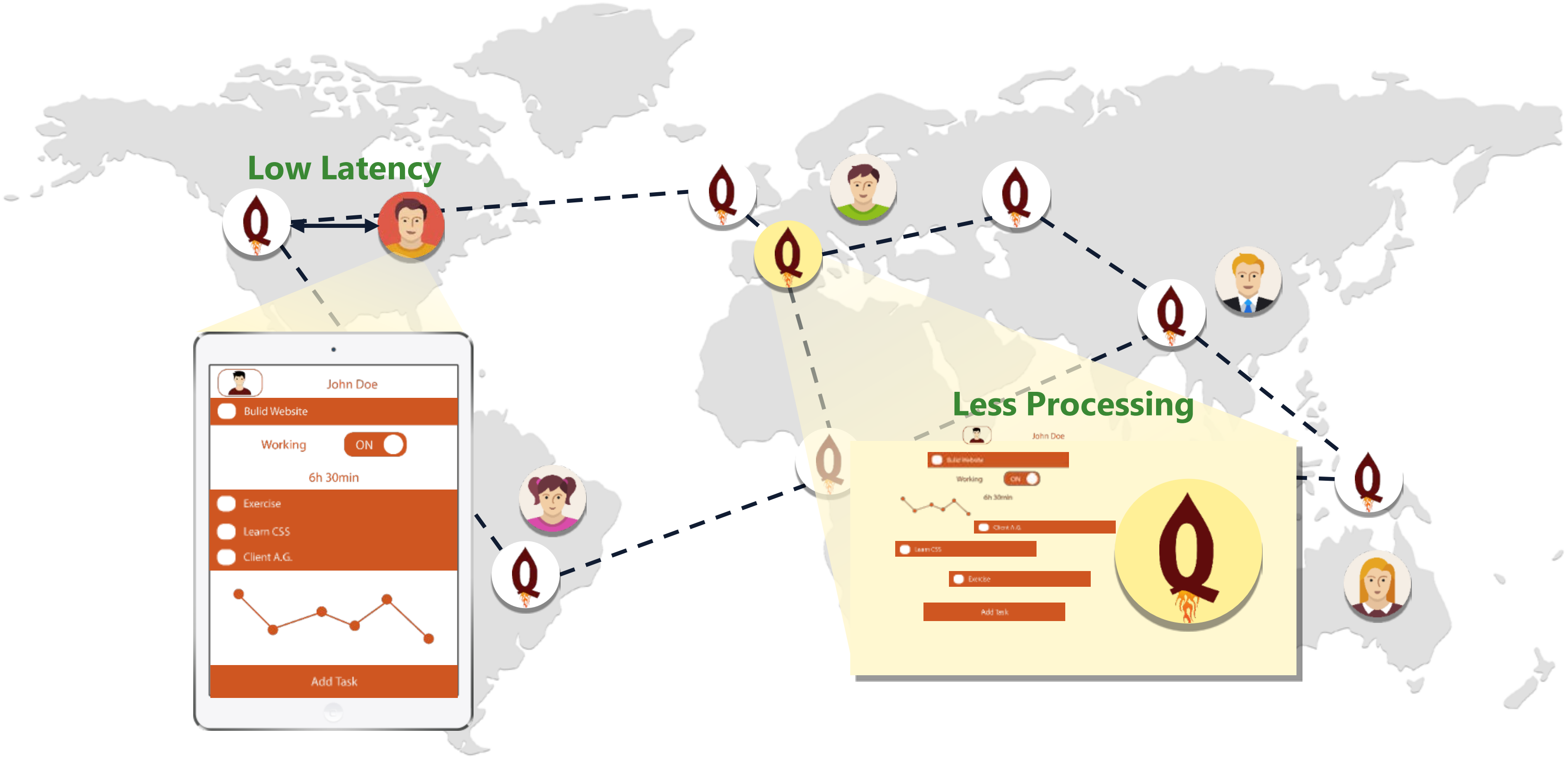
Infrastructure

F. Gessert, Low Latency for Cloud Data Management. PhD thesis, University of Hamburg, 2018.



Solution: Global Caching

Fresh Data From Distributed Web Caches

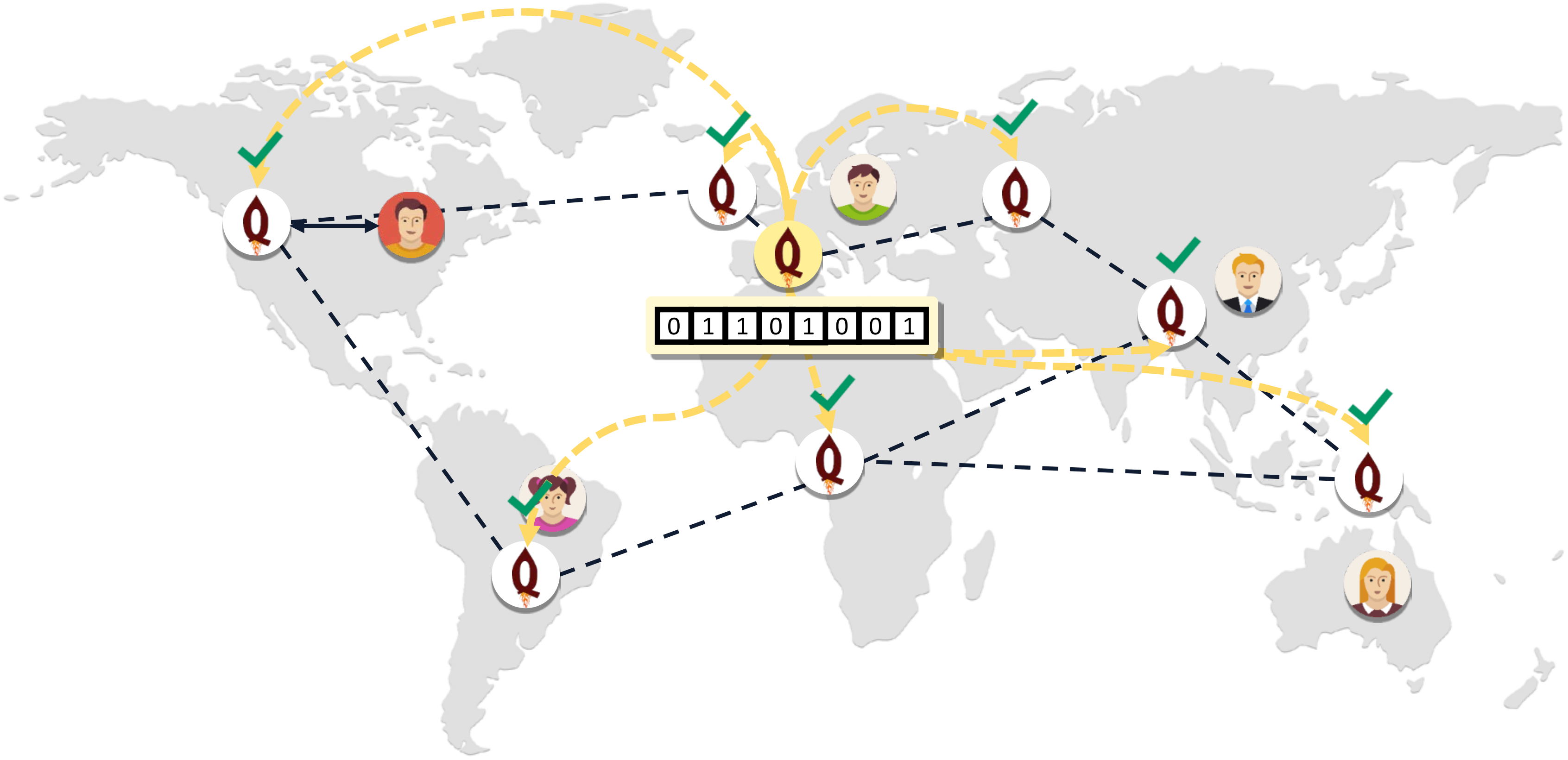


BaQend

www.baqend.com

New Caching Algorithms

Solve Consistency Problem

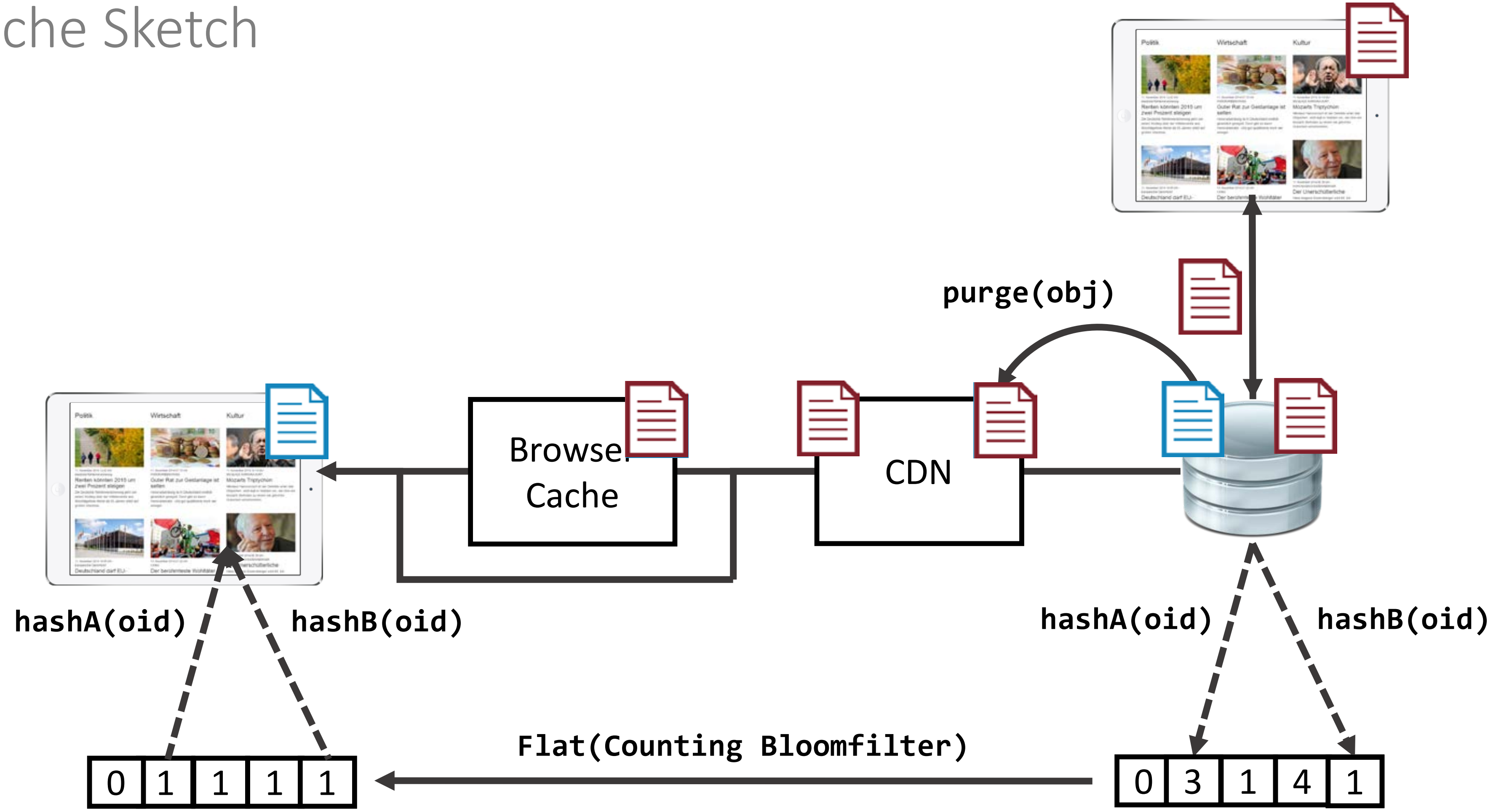


BaQend

www.baqend.com

Consistent Web Caching

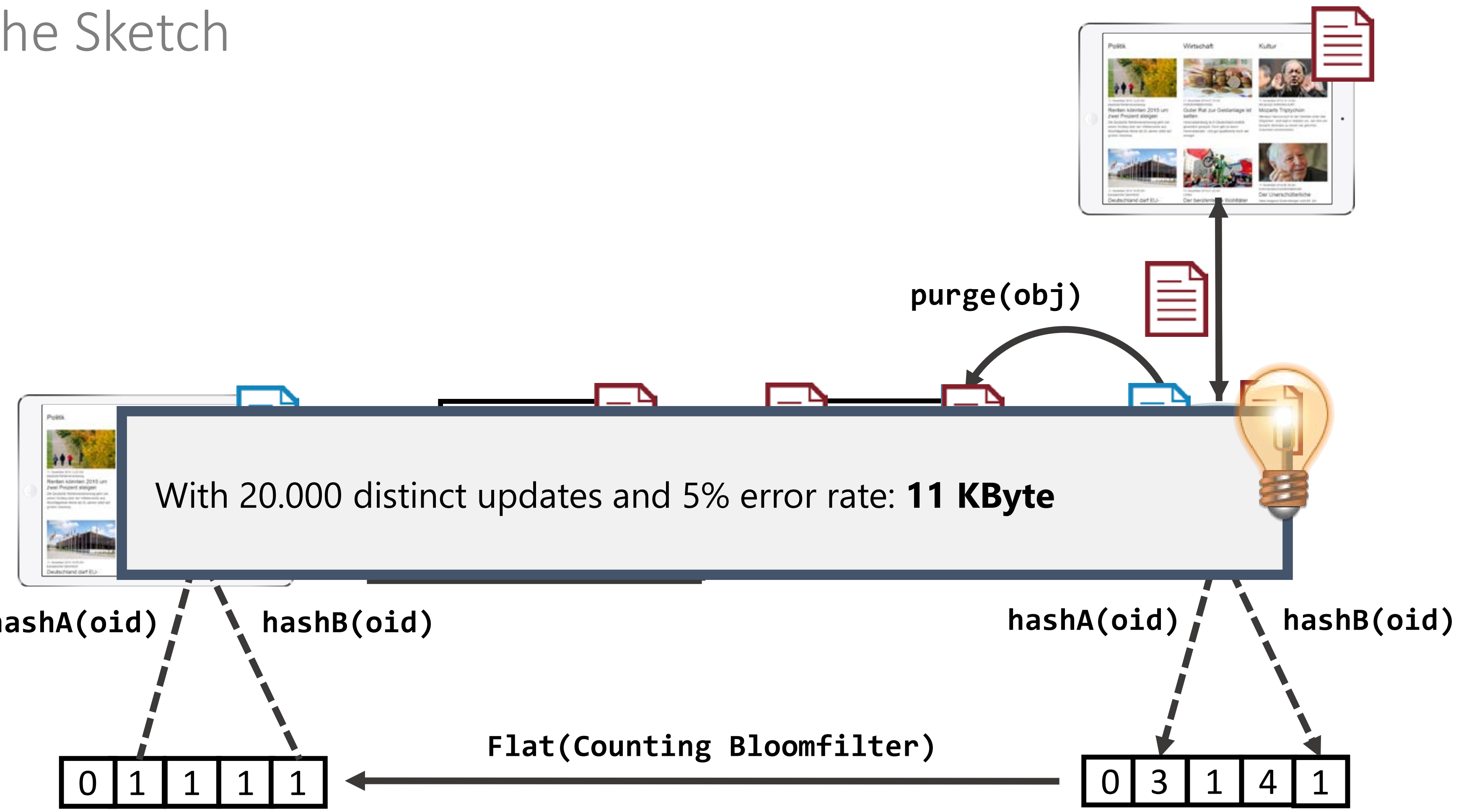
The Cache Sketch



BaQend
www.baqend.com

Consistent Web Caching

The Cache Sketch



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

4. Make the backend push-based, additionally (real-time queries)!

Challenge: Real-Time Queries

C_1: Scalability:
- Handle additional queries
- Handle increasing throughput

C_2: Expressiveness:
- Content search? Composite filters?
- Ordering? Limit? Offset?

Research Question: *„How can expressive push-based real-time queries be implemented on top of an existing pull-based database in a scalable and generic manner?“*

C_3: Legacy Support:
- Real-time queries for *existing databases*
- *Decouple* OLTP from real-time workloads

C_4: Abstract API
- Data independence
- Self-maintaining queries

Overview Real-time DBSs

	METEOR		RethinkDB	Parse	Firebase
	Poll-and-Diff	Change Log Tailing			Unknown
Write Scalability	✓	✗	✗	✗	✗
Read Scalability	✗	✓	✓	✓	? (100k connections)
Composite Filters (AND/OR)	✓	✓	✓	✓	○ (AND In Firestore)
Sorted Queries	✓	✓	✓	✗	○ (single attribute)
Limit	✓	✓	✓	✗	✓
Offset	✓	✓	✗	✗	○ (value-based)
Self-Maintaining Queries	✓	✓	✗	✗	✗
Event Stream Queries	✓	✓	✓	✓	✓

Overview Stream-Processors

	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	✓	✓	no (buffering)	✓	✓
Ordering	✗	between batches	within partitions	between batches	within partitions
Elasticity	✓	✓	✗	✓	✗

Overview Stream-Processors

	Storm	Trident	Samza	Spark Streaming	Flink (streaming)
Strictest Guarantee	at-least-once	exactly-once	at-least-once	[illegible]	[illegible]
Achievable Latency	≪100 ms	[illegible]	[illegible]	[illegible]	[illegible]
State M[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible] time	micro-batch	one-at-a-time
[illegible]	[illegible]	✓	no (buffering)	✓	✓
[illegible]	[illegible]	[illegible]	[illegible]	between batches	within partitions
Elasticity	✓	✓	✗	✓	✗

"**Storm** […] and **Flink** […] show **sub-second latencies** at relatively high throughputs with **Storm** having the **lowest 99th percentile** latency. **Spark** streaming […] supports high **throughputs**, but at a relatively **higher latency**."

From https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at

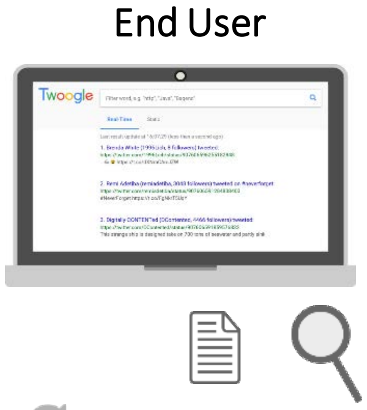
InvaliDB: A Scalable Real-Time Database Design

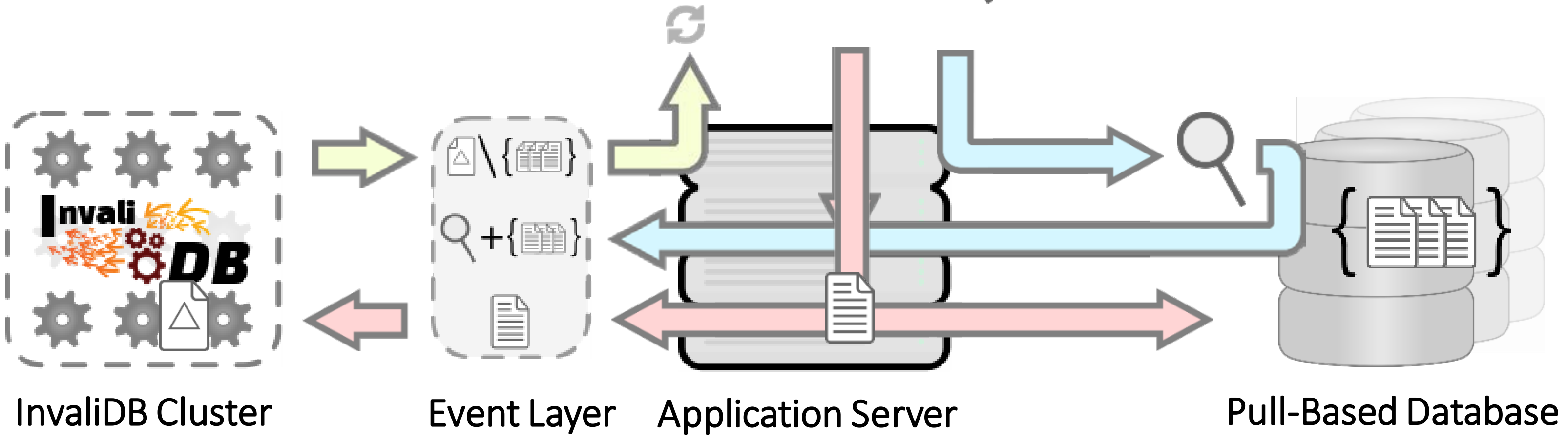
System Model & Overview

Realtime-as-a-Service For Heterogeneous Tenants:

1. Query Subscription
2. Write Ingestion
3. Change Propagation

- resource pooling: high matching performance & overall efficiency
- multi-tenancy: low provisioning overhead per application server





Real-Time & OLTP Workloads Decoupled:

- isolated failure domains
- separated resource requirements & independent scaling

Baqend
www.baqend.com

InvaliDB: A Scalable Real-Time Database Design

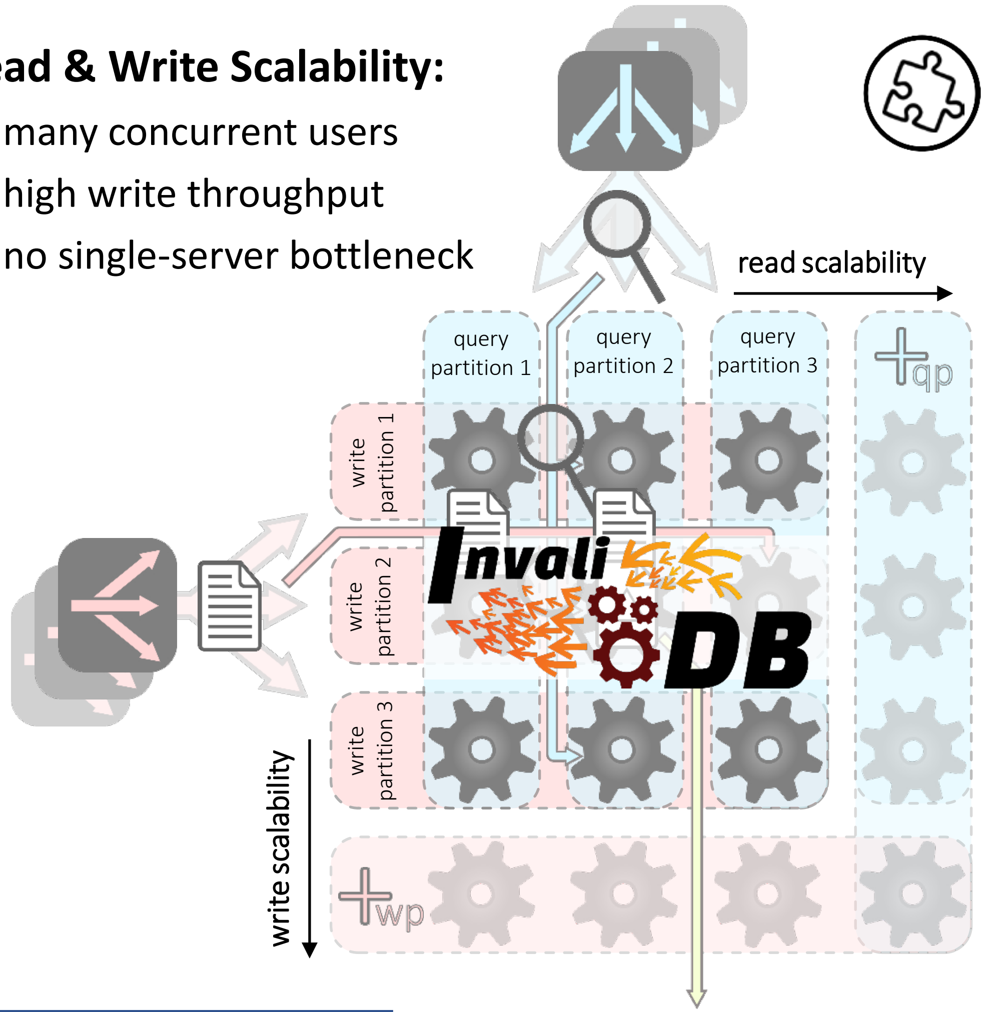
Two-Dimensional Workload Partitioning

Read & Write Scalability:

- many concurrent users
- high write throughput
- no single-server bottleneck

Pluggable Query Engine:

- legacy-compatibility
- multi-tenancy across databases

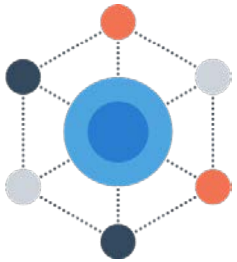


W. Wingerath, Scalable Push-based Real-time Queries on top of Pull-based Databases. PhD thesis, University of Hamburg, 2019.

Baqend realises 1. to 4. as commercially available service

www.baqend.com

Platform



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

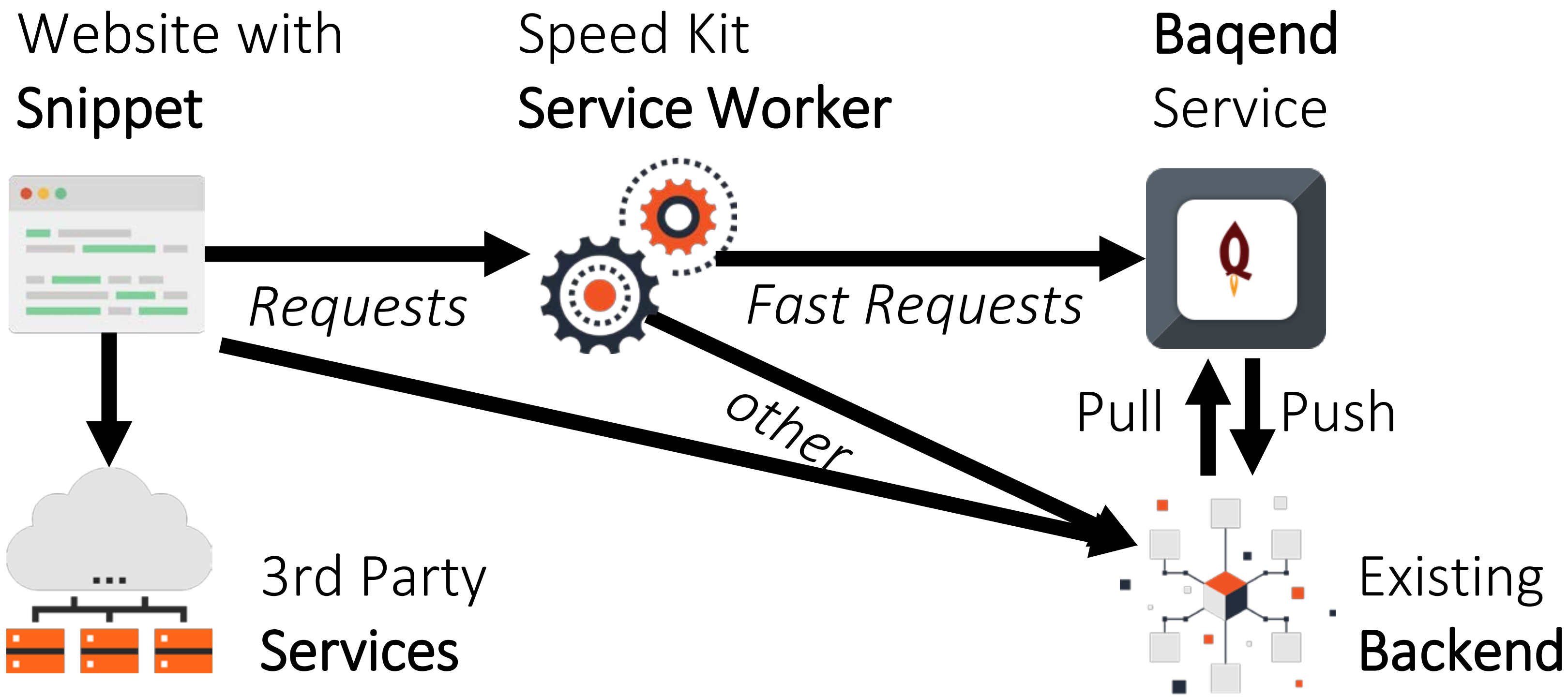
Speed Kit

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

Try It Out!

Speed Kit

Baqend Caching for Legacy Websites



Speed Kit

Measure Your Page Speed!

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

Baqend

www.baqend.com

Speed Kit

Built for Market Leaders

For a large e-commerce company like Baur, supreme performance and a snappy user experience are vital. **Speed Kit** helps Baur.de stay ahead of the competition by accelerating page loads through **cutting-edge technology**. Finally, there is a German player in the web performance market that does not only pioneer a **superior approach**, but also shines through competent onboarding and immediate support.

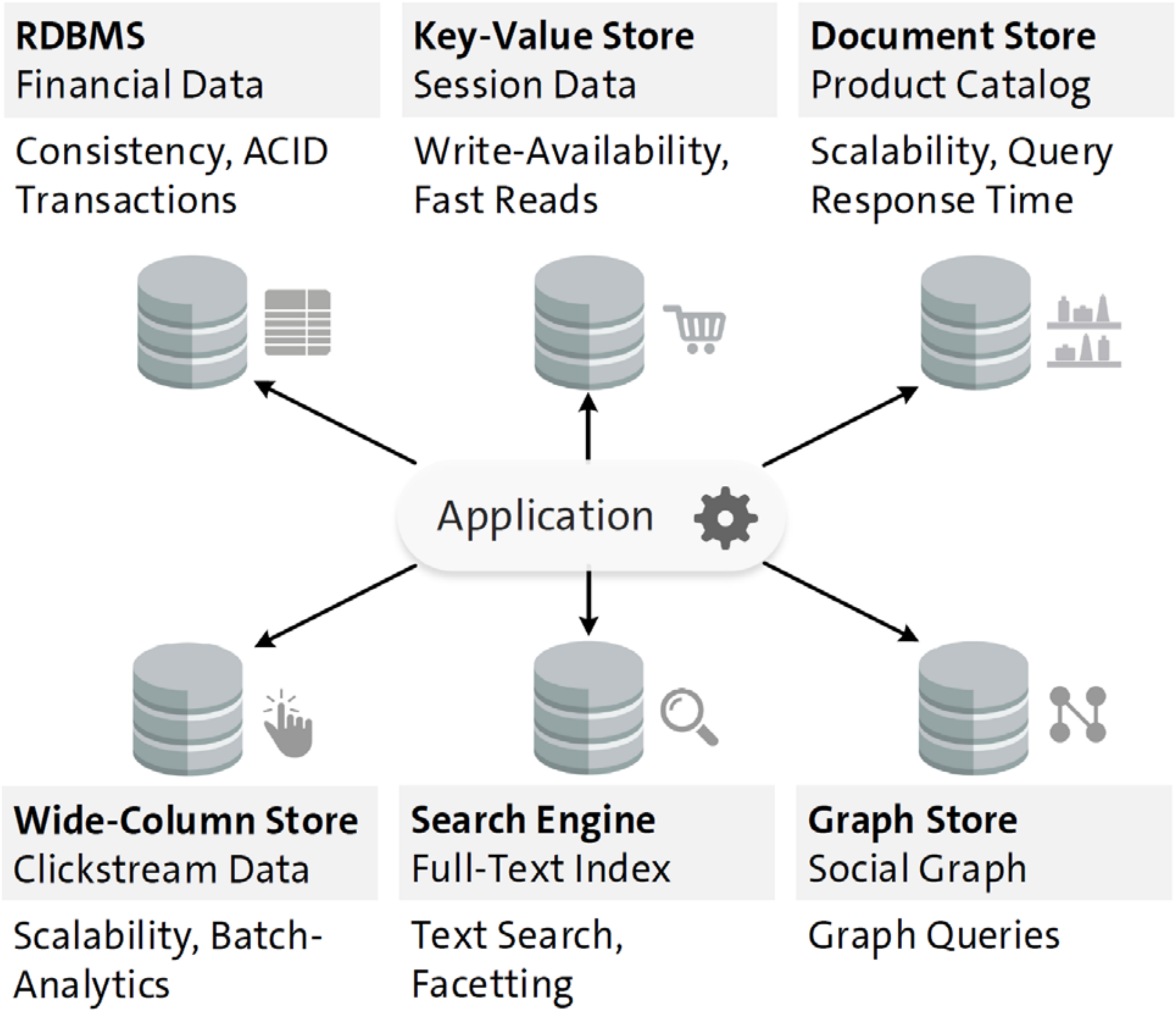
Revenue: 1 000 000 000 € for 2018
Traffic: 70 000 000 PIs per month

BAUR

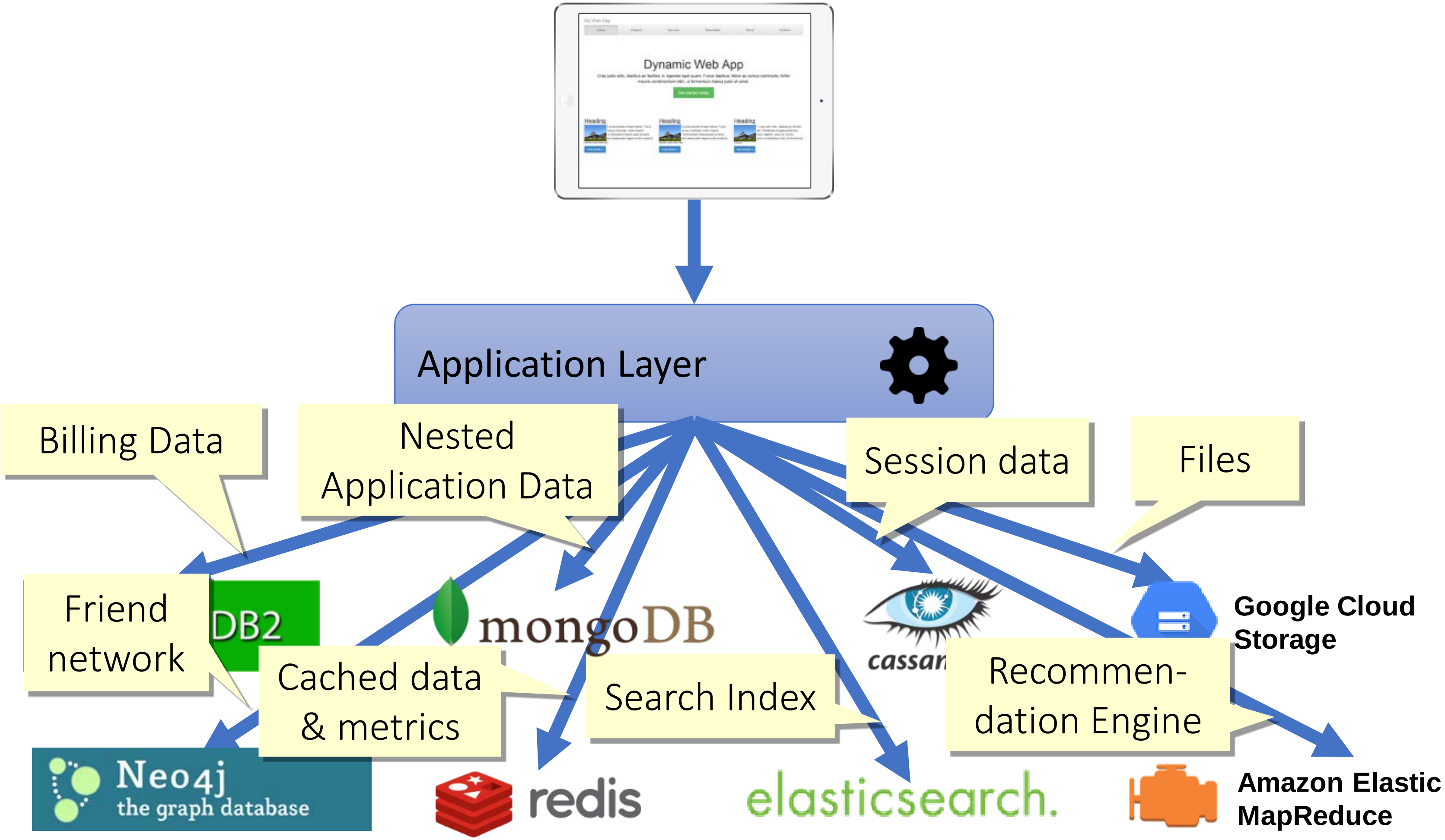
A member of the otto group

Vision

5. Provide Polyglot Persistence!



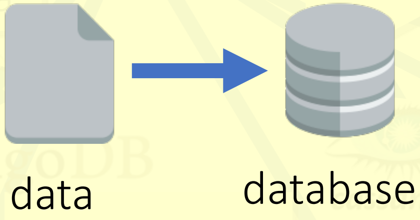
Challenge: ‚automated‘ mediation



Challenge: ‚automated‘ mediation

Research Question:

Can we automate the

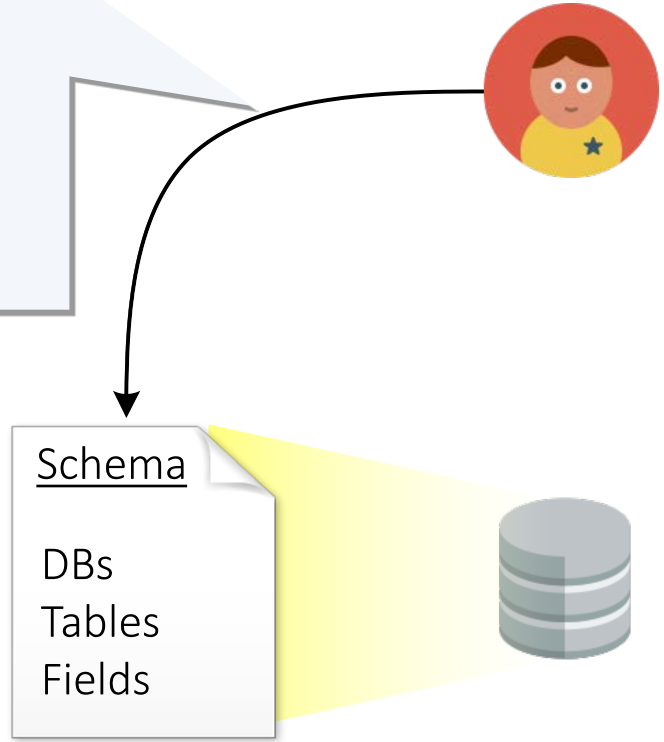


mapping problem?

Vision

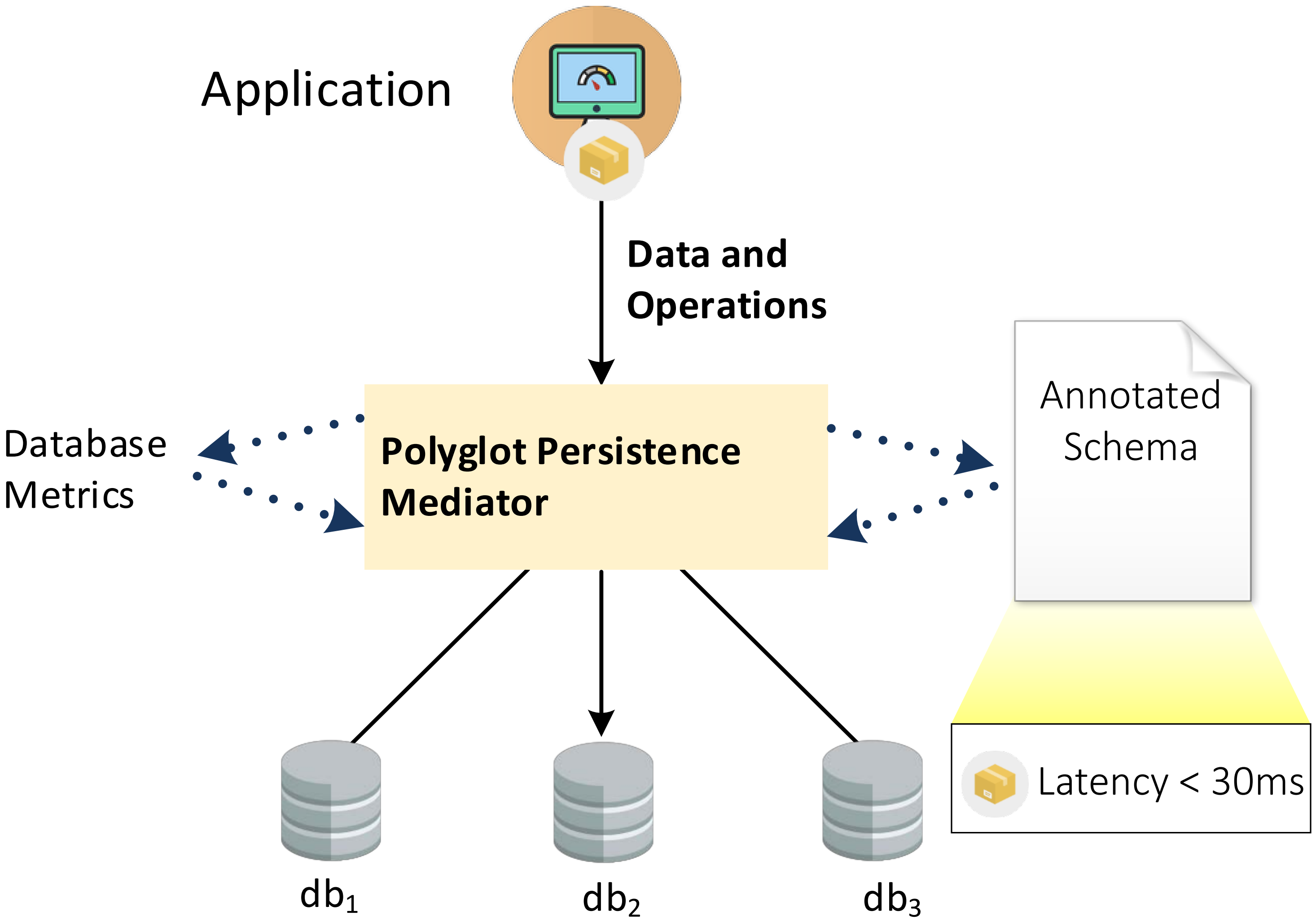
Schemas can be annotated with requirements

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



Vision

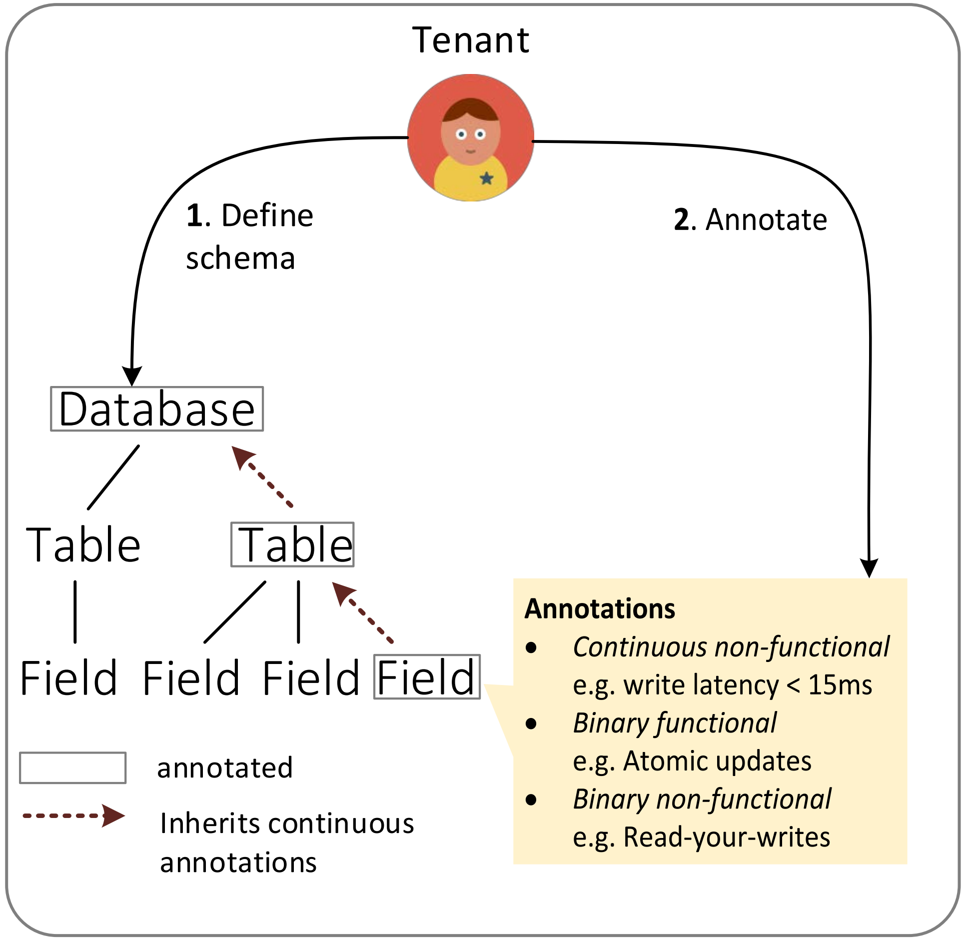
The Polyglot Persistence Mediator chooses the database



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 Scalabilty	Non-Functional	Field/Class/DB
Elasticity	Non-Functional	Field/Class/DB
Durability	Non-Functional	Field/Class/DB
Replicated	Non-Functional	Field/Class/DB
Linearizability	Non-Functional	Field/Class
Read-your-Writes	Non-Functional	Field/Class
Causal Consistency	Non-Functional	Field/Class
Writes follow reads	Non-Functional	Field/Class
Monotonic Read	Non-Functional	Field/Class
Monotonic Write	Non-Functional	Field/Class
Scans	Functional	Field
Sorting	Functional	Field
Range Queries	Functional	Field
Point Lookups	Functional	Field
ACID Transactions	Functional	Class/DB
Conditional Updates	Functional	Field
Joins	Functional	Class/DB
Analytics Integration	Functional	Field/Class/DB
Fulltext Search	Functional	Field
Atomic Updates	Functional	Field/Class

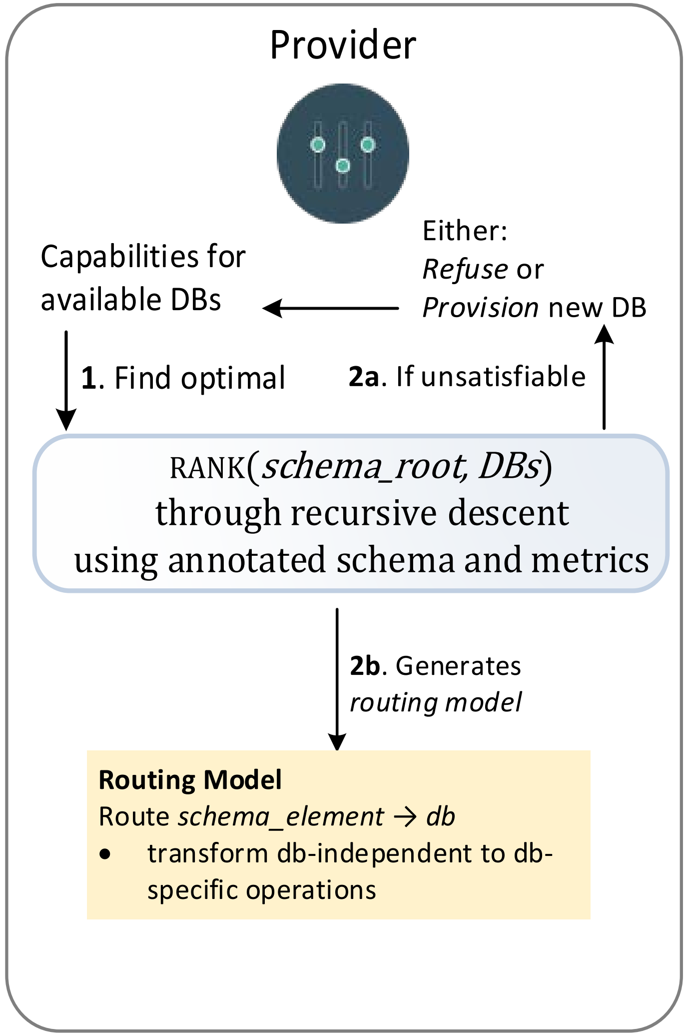


1 Requirements

Step II - Resolution

Finding the best database

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

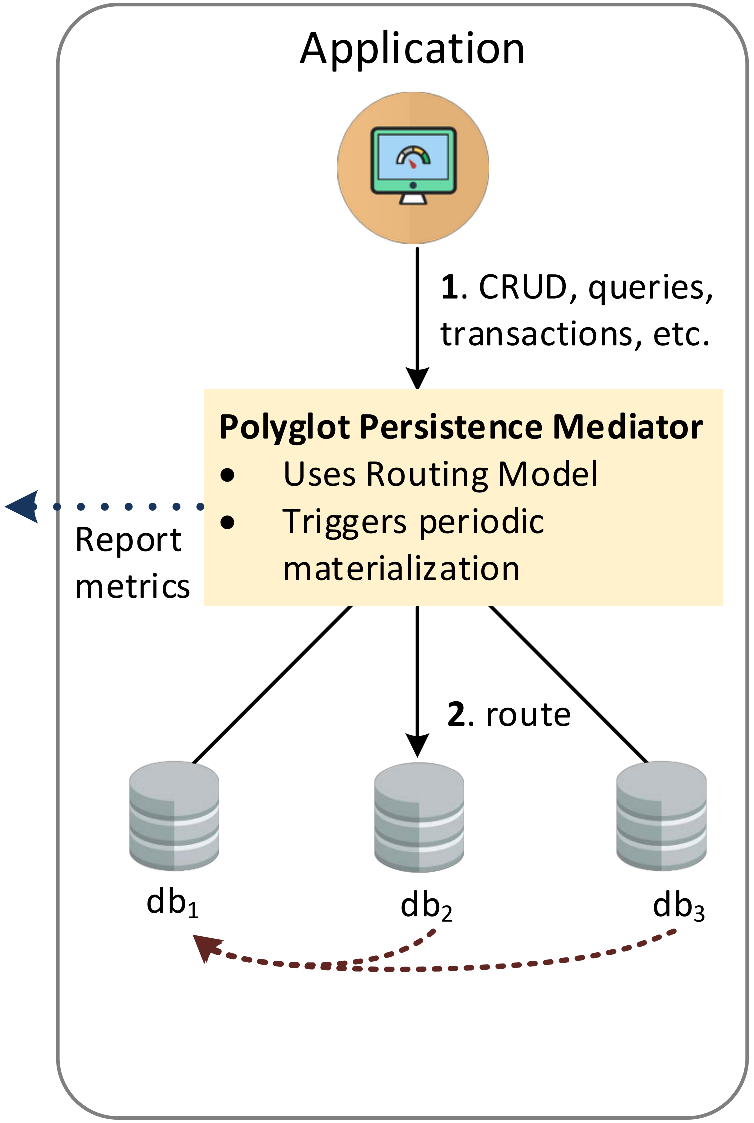


2 Resolution

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

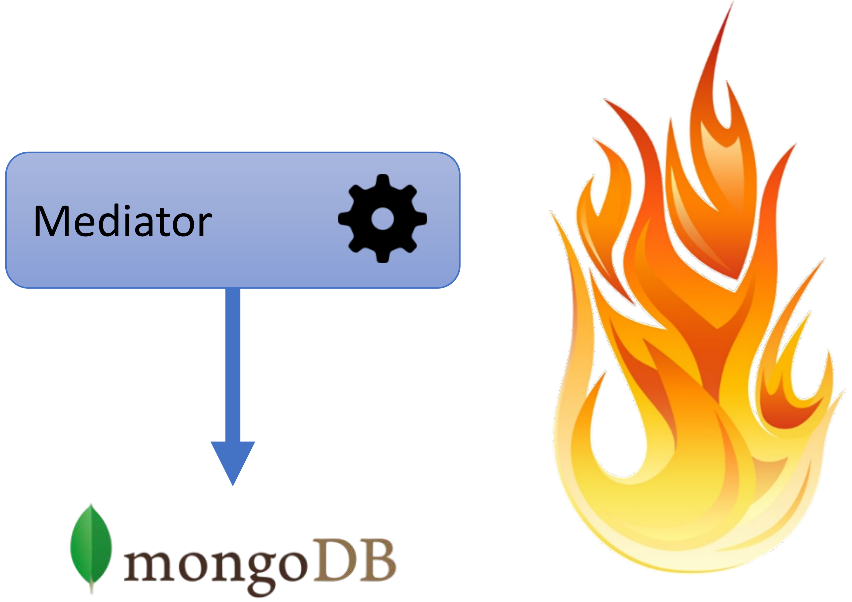
Scenario: news articles with impression counts

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

Evaluation: News Article

Prototype built on ORESTES

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

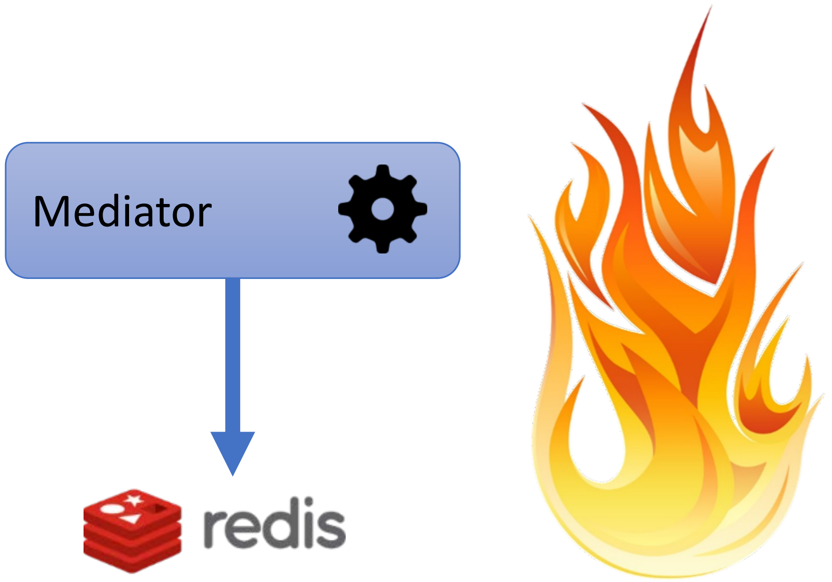


Counter updates kill performance

Evaluation: News Article

Prototype built on ORESTES

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

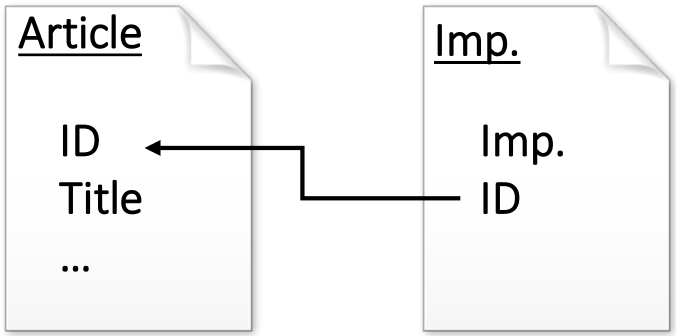


No powerful queries

Evaluation: News Article

Prototype built on ORESTES

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



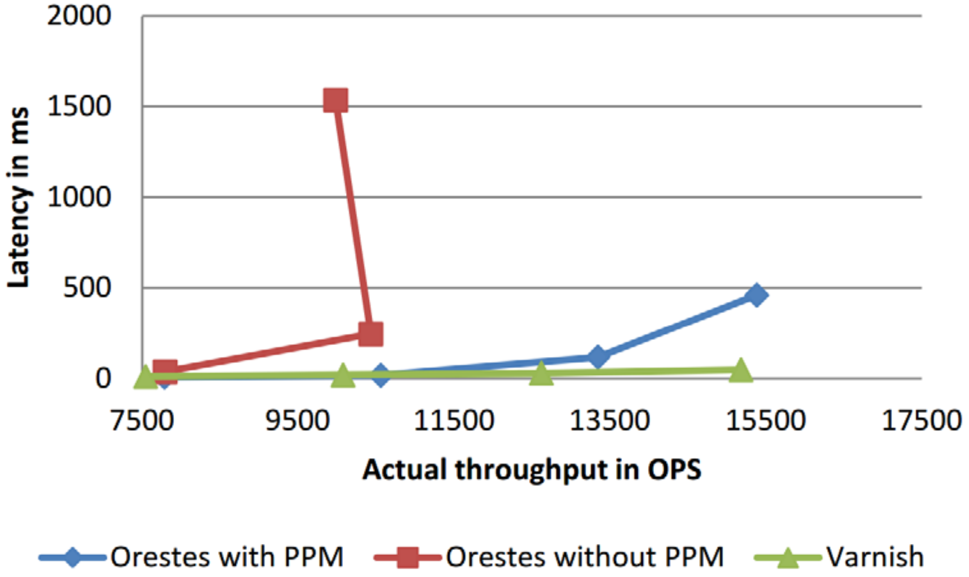
Document

Sorted Set

mongoDB

redis

Found Resolution



Polyglot Persistence: Challenges

- **Database Selection**: Actively minimize SLA violations
- **Utility Functions/SLAs**: Capture trade-offs comprehensively
- **Meta-DBaaS**: Mediate over DBaaS-Systems, unify SLAs
- **Live Migration**: adapt to changing requirements
- **Workload Management**: Adaptive Runtime Scheduling
- **Transaction Management**: Alignment of ACID with NoSQL and scalability
- **Multi-Tenancy/Privacy**: Dream: full homomorphic encryption

CLOSING TIME

Summary

S

C

D

M

1. Aim at fully managed Backend (BaaS)

2. Exploit modern (NoSQL) DB Technology

3. Consider the entire path from the (mobile) Application through the Net to the Data Backend!

4. Make the backend push-based, additionally (real-time queries)!

5. Provide Polyglot Persistence!

6. *Other Problems? … certainly!*