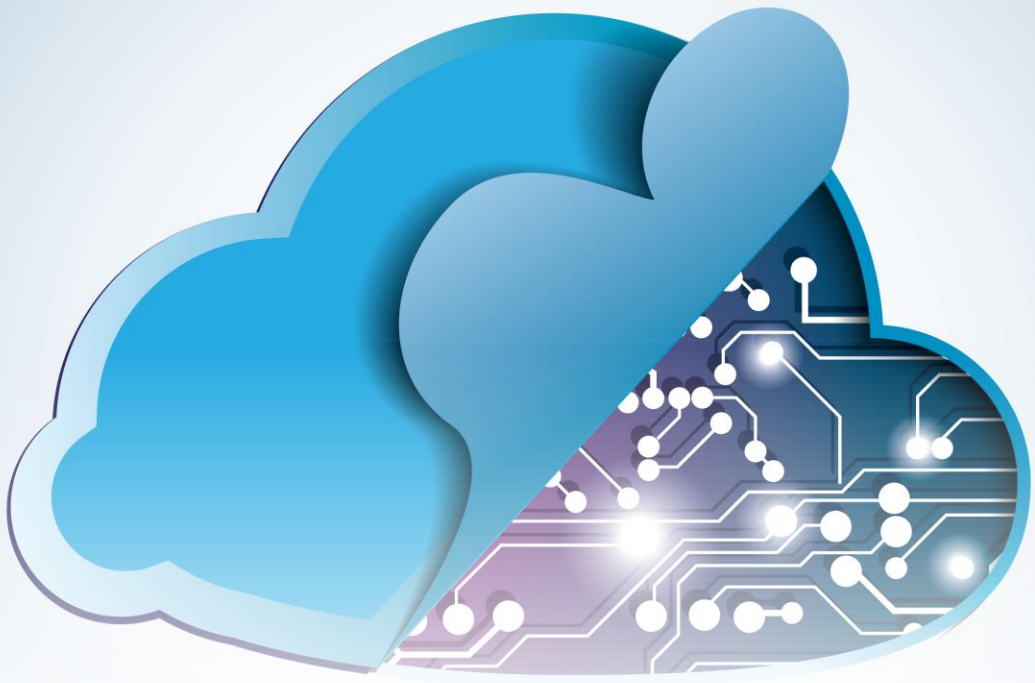


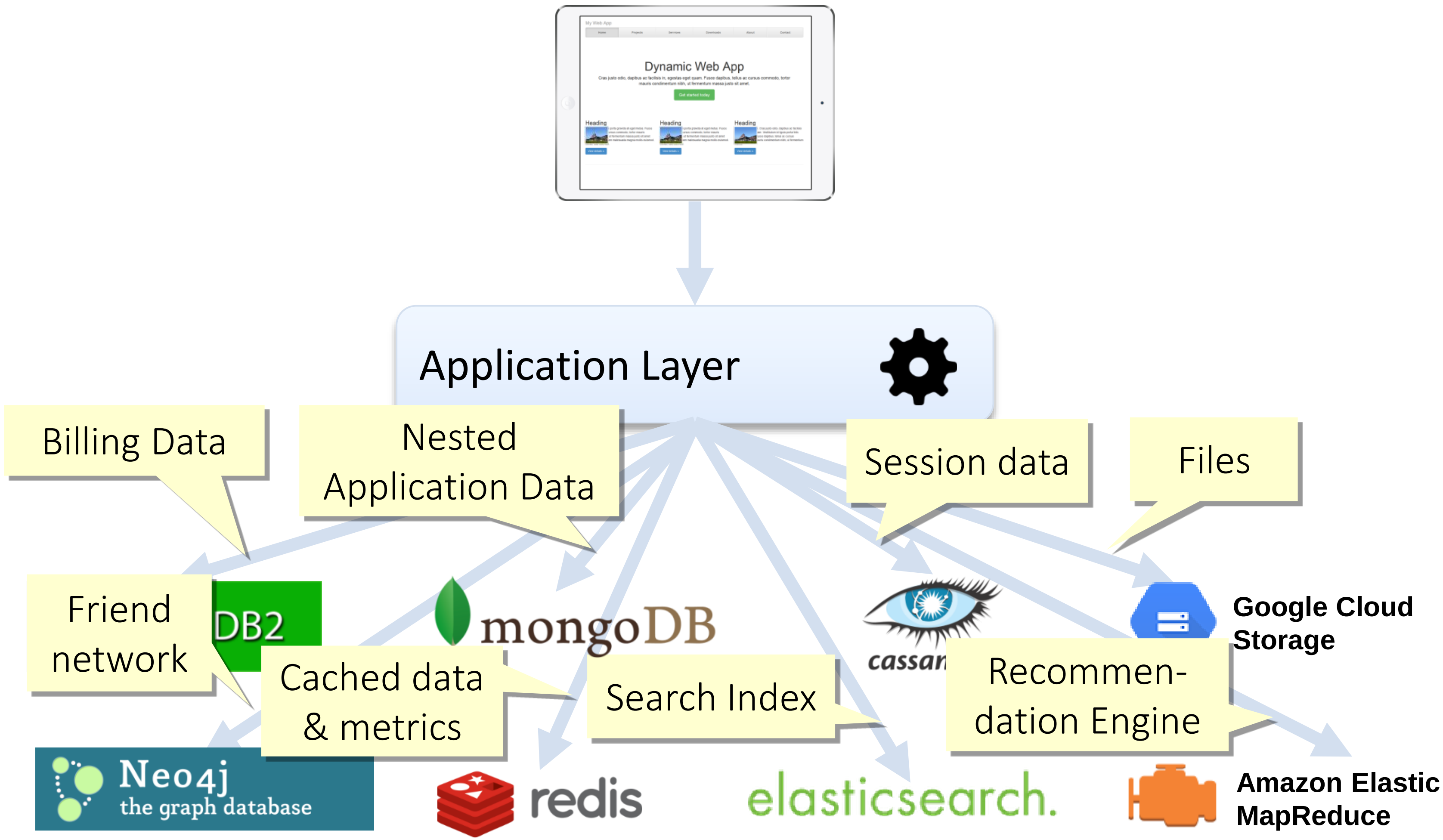
Towards Automated Polyglot Persistence

Michael Schaarschmidt, Felix Gessert, Norbert Ritter

gessert@informatik.uni-hamburg.de

Polyglot Persistence

Current best practice

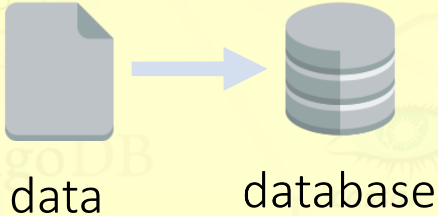


Polyglot Persistence

Current best practice

Research Question:

Can we automate the

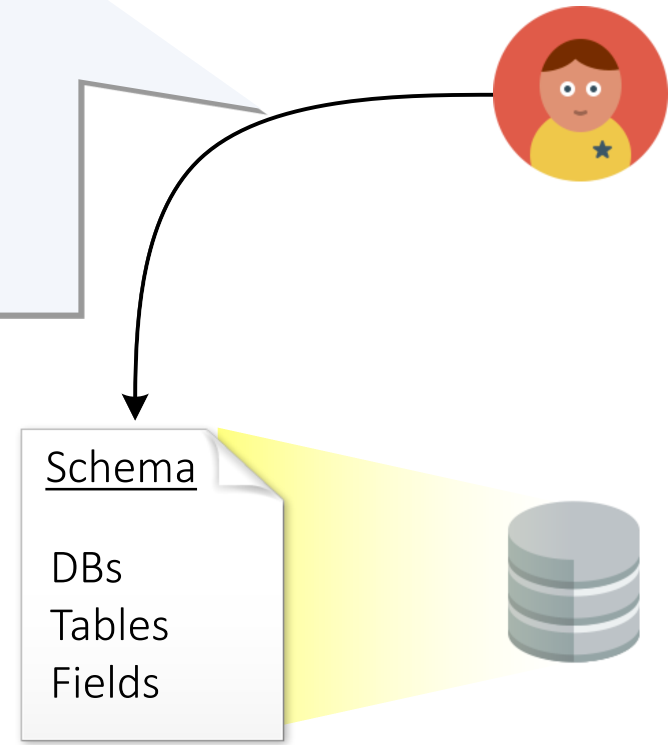


mapping problem?

Vision

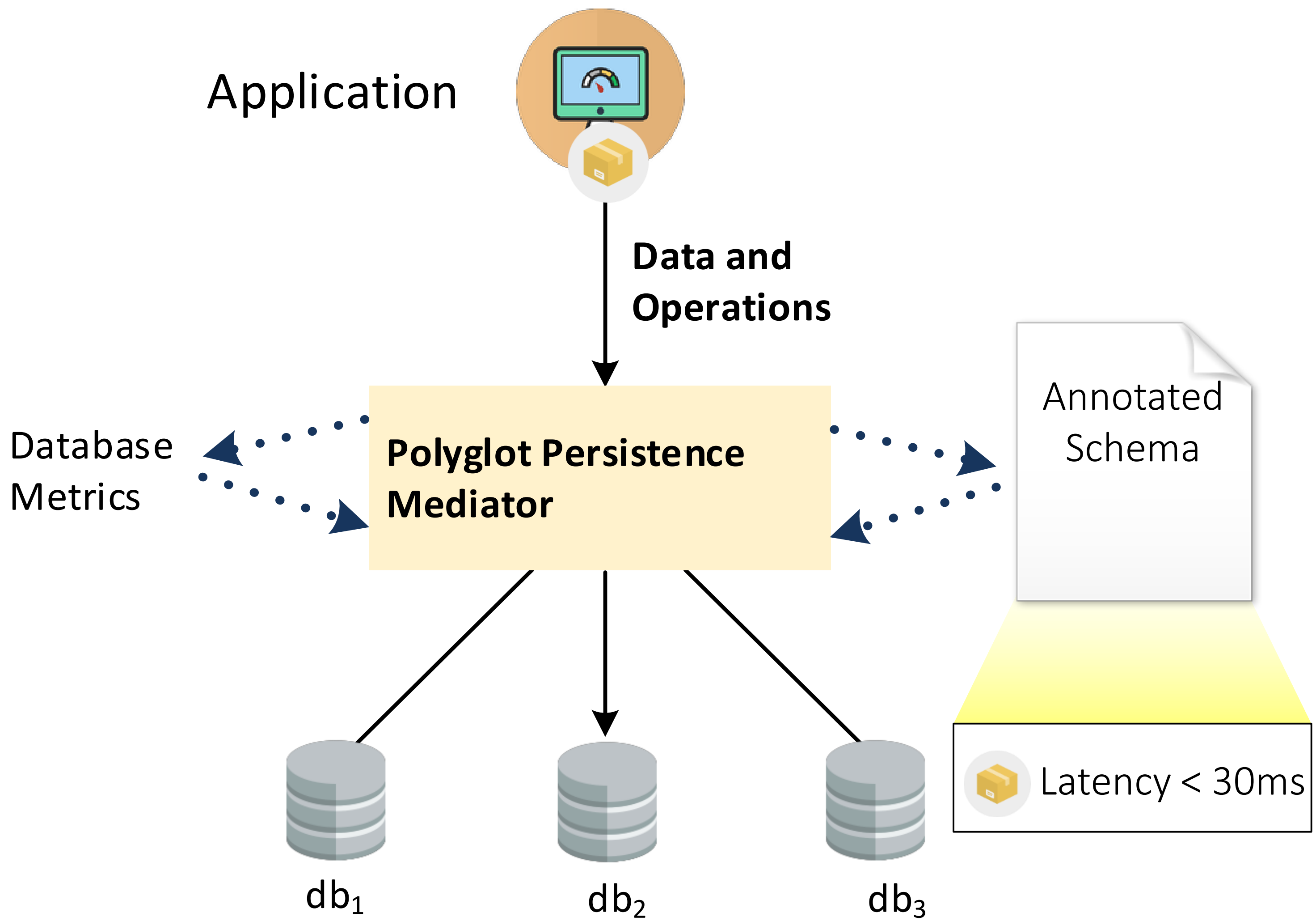
Schemas can be annotated with requirements

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



Vision

The Polyglot Persistence Mediator chooses the database

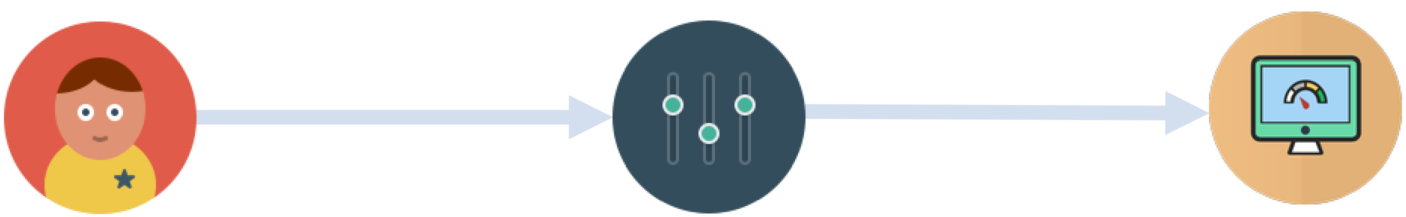


Towards Automated Polyglot Persistence

Necessary steps

- Goal:
  - Extend classic workload management to *polyglot persistence*
  - Leverage hetereogeneous (NoSQL) databases

1. Requirements — **2. Resolution** — **3. Mediation**



1. Requirements: Tenant specifies requirements as Service-Level-Agreements
2. Resolution: Find or provision a suitable combination of databases
3. Mediation: Mediate data and database operations

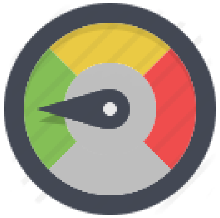
Service Level Agreements

Expressing application requirements

Functional Service Level Objectives

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

Non-Functional Service Level Objectives



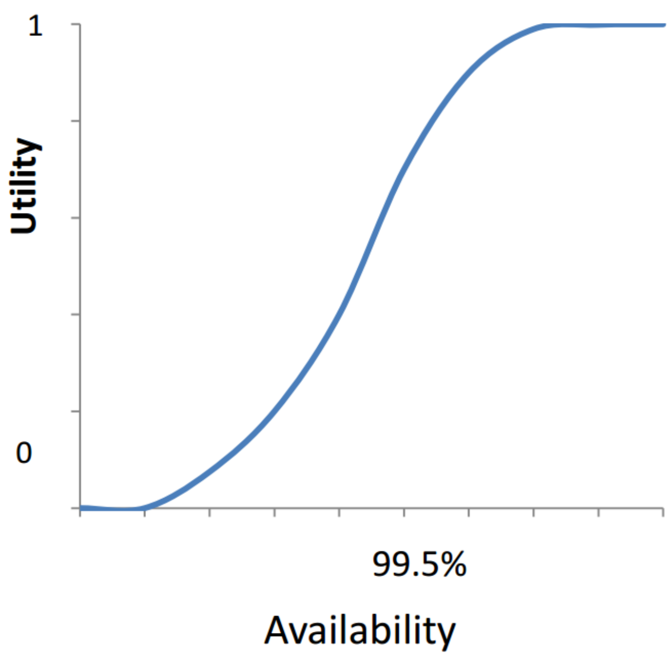
- Guarantee a certain *quality of service* (QoS)
- Determined by database system and service provider
- *Examples*:
  - **Continuous**: response time (latency), throughput
  - **Binary**: Elasticity, Read-your-writes

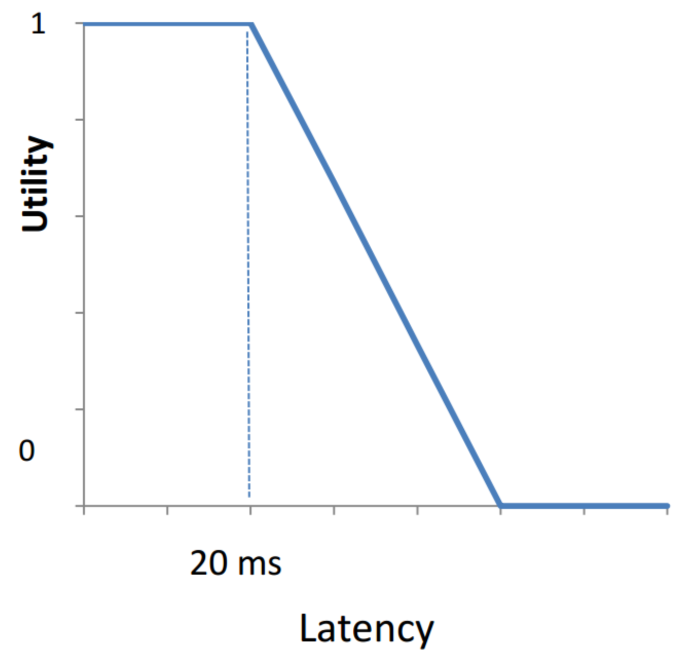
Service Level Agreements

Refining the utility of each SLO

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

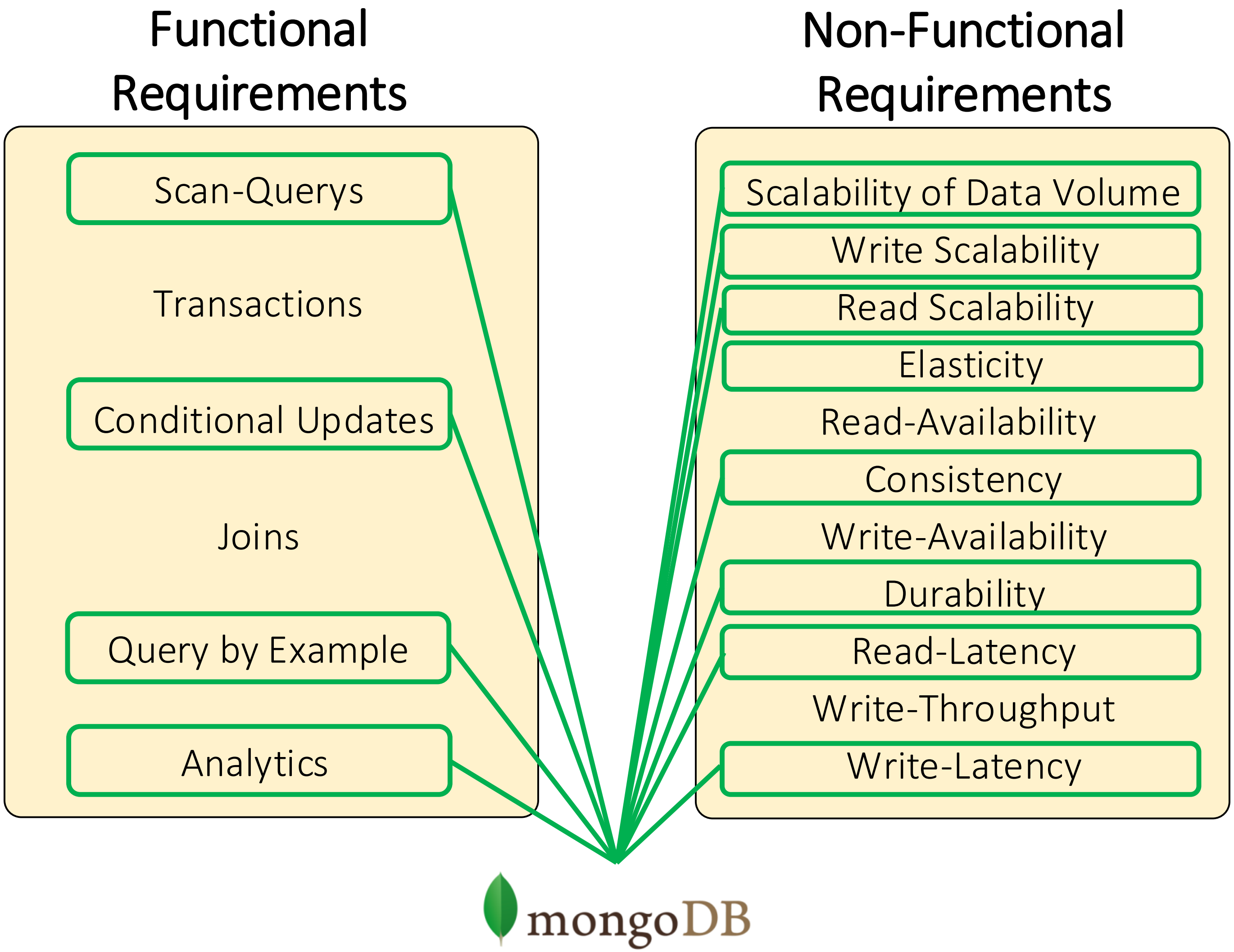
$$f_{utility}(metric) \rightarrow [0,1]$$





SLA Example

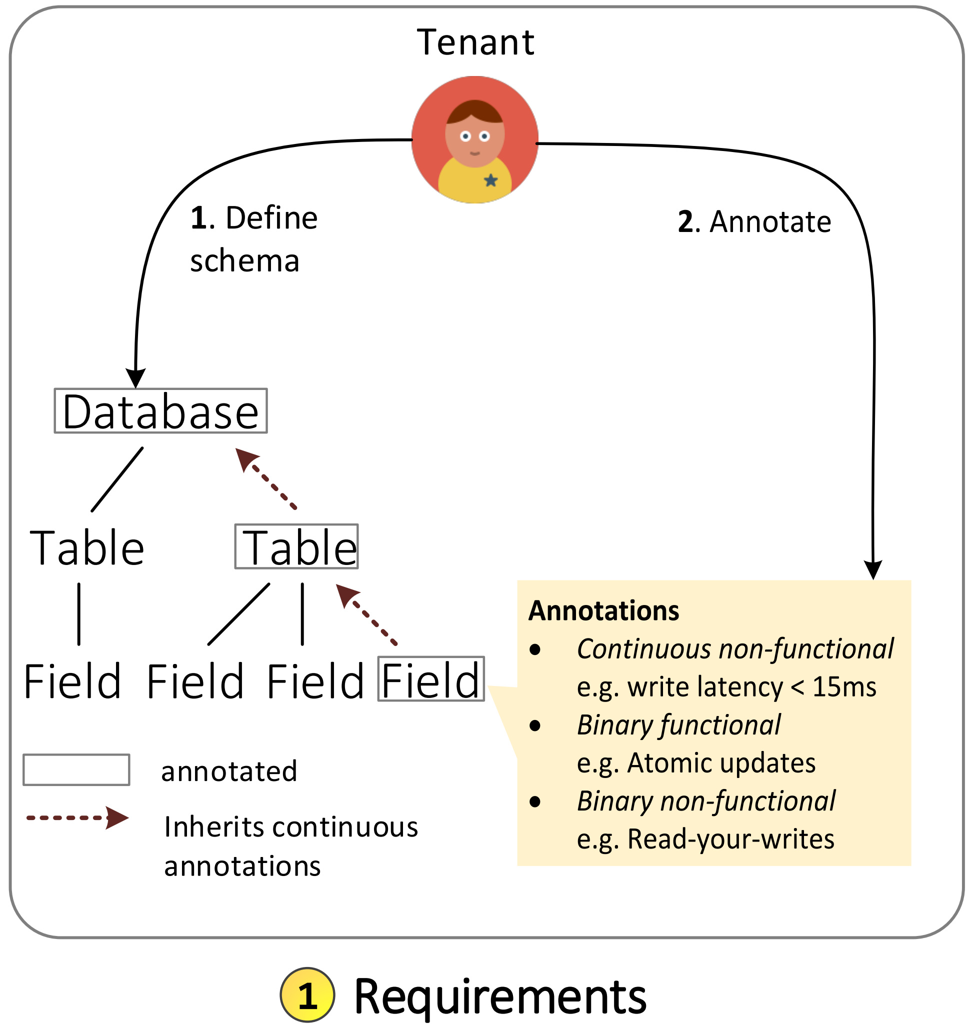
For MongoDB



Step I - Requirements

Expressing the application's needs

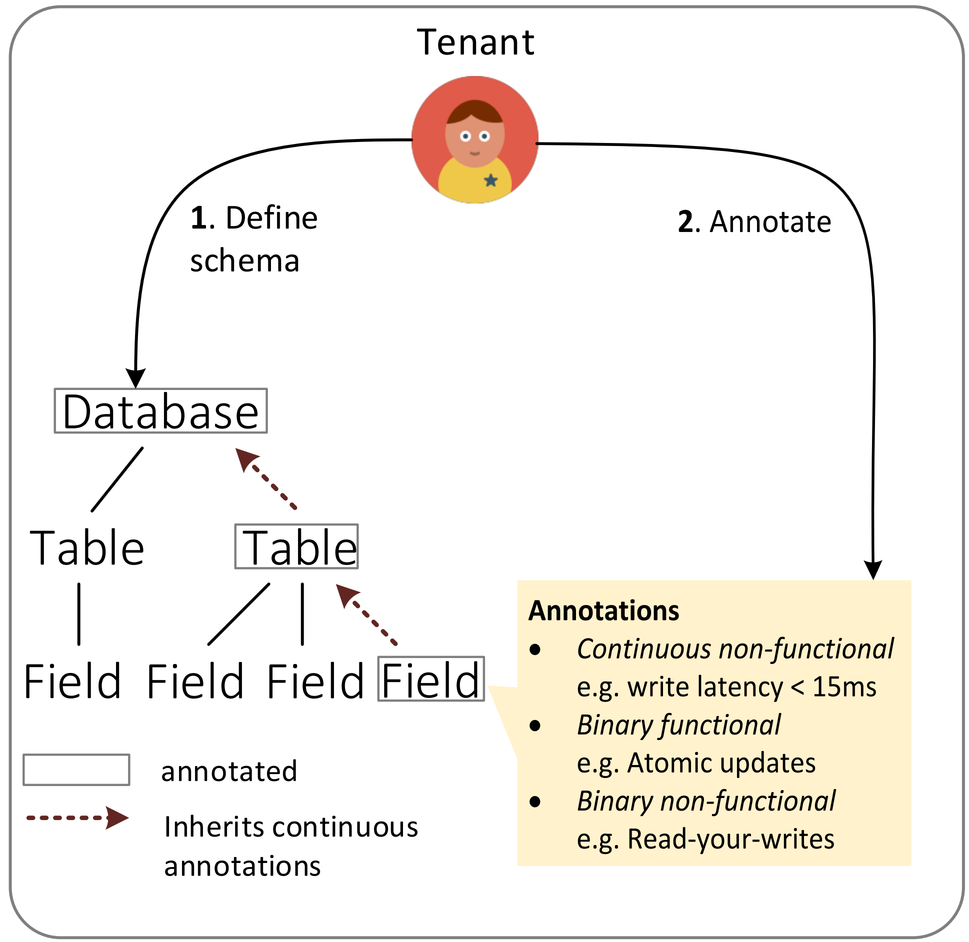
- Tenant annotates schema with his requirements



Step I - Requirements

Expressing the application's needs

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

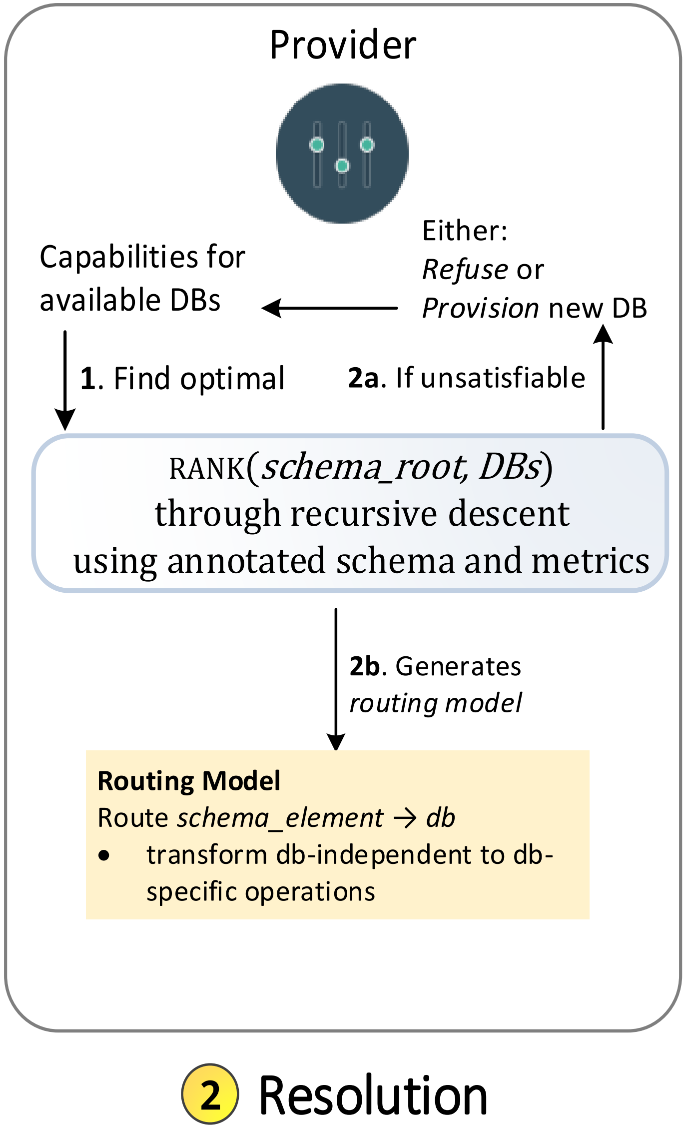


1 Requirements

Step II - Resolution

Finding the best database

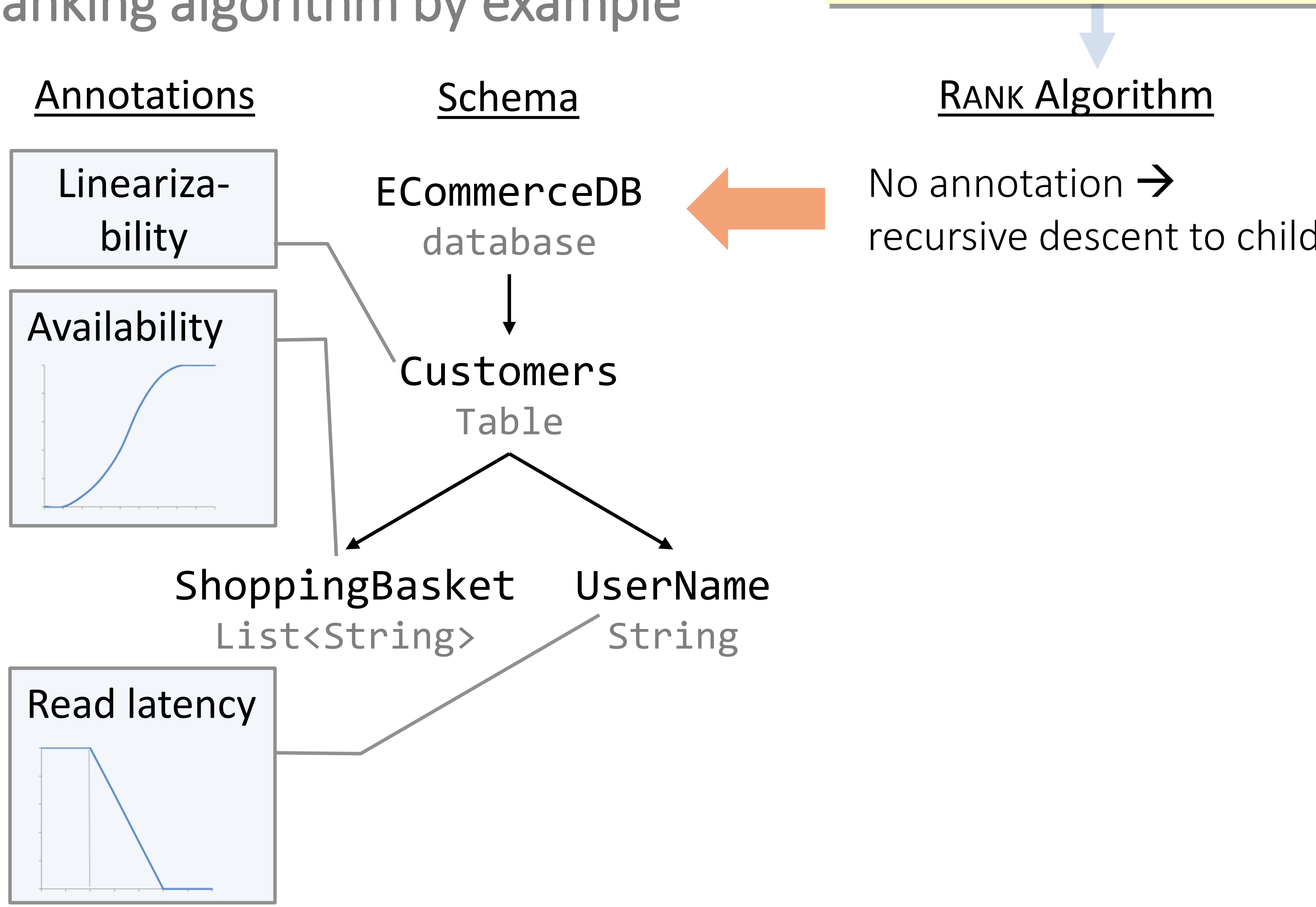
- The Provider resolves the requirements
- RANK: scores available database systems
- **Routing Model**: defines the optimal mapping from schema elements to databases



Step II - Resolution

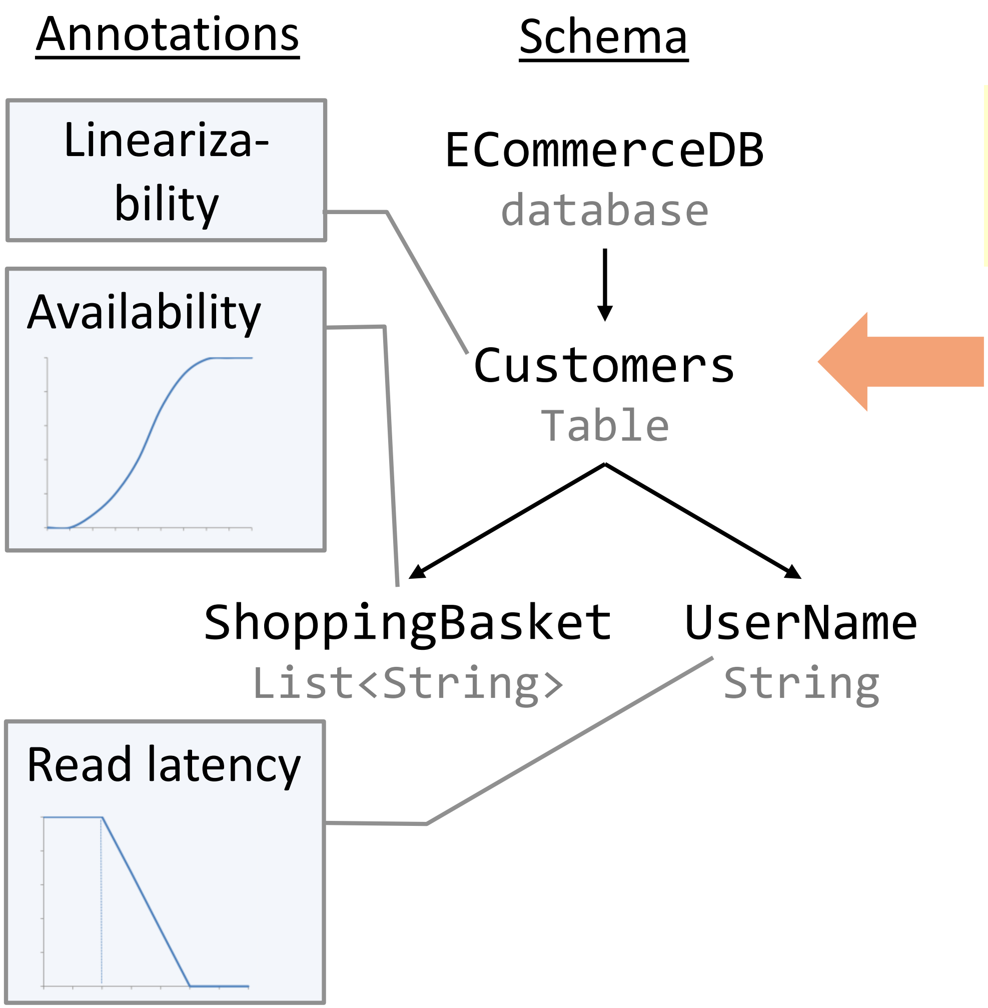
Ranking algorithm by example

DBs = { *MongoDB, Riak, Cassandra, CouchDB, Redis, MySQL, S3, Hbase* }



Step II - Resolution

Ranking algorithm by example



RANK Algorithm

DBs = { *MongoDB*, ~~*Riak*~~, ~~*Cassandra*~~, ~~*CouchDB*~~, *Redis*, *MySQL*, ~~*S3*~~, *Hbase* }

Binary requirement →

1. Exclude DBs that do not support it
2. Recursive descent

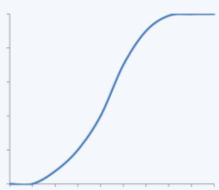
Step II - Resolution

Ranking algorithm by example

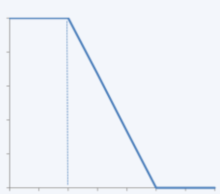
Annotations

Lineariza-bility

Availability



Read latency



Schema

`ECommerceDB` database

`Customers` Table

`ShoppingBasket` List<String>

String

RANK Algorithm

Database	Availability
MongoDB	99%→0.8
Redis	95%→0.05
MySQL	94%→ 0.04
HBase	99.9%→0.9

Continuous requirement →
∀ databases calculate

$$db \rightarrow f_{utility}(db.availability)$$

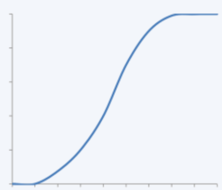
Step II - Resolution

Ranking algorithm by example

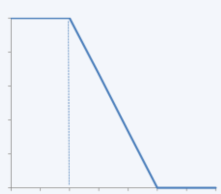
Annotations

Lineariza-
bility

Availability



Read latency



Schema

`ECommerceDB`
database

↓

`Customers`
Table

`ShoppingBasket`
List<String>

`UserName`
String

Rank Algorithm

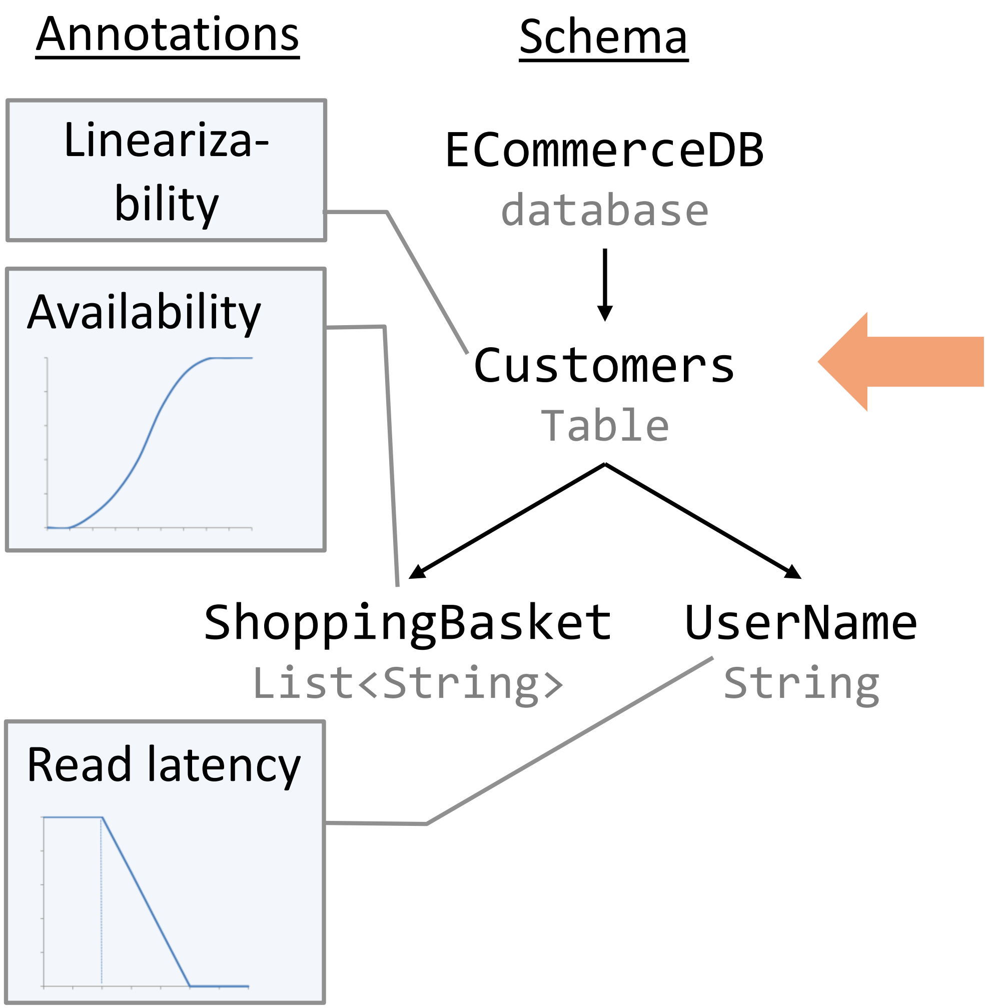
Database	Availability	Latency
MongoDB	99%→0.8	10ms→1
Redis	95%→0.05	1ms→1
MySQL	94%→ 0.04	40ms→0.2
HBase	99.9%→0.9	50ms→0.1

Continuous requirement →
∀ databases calculate

$$db \ \rightarrow f_{utility}(db.latency)$$

Step II - Resolution

Ranking algorithm by example



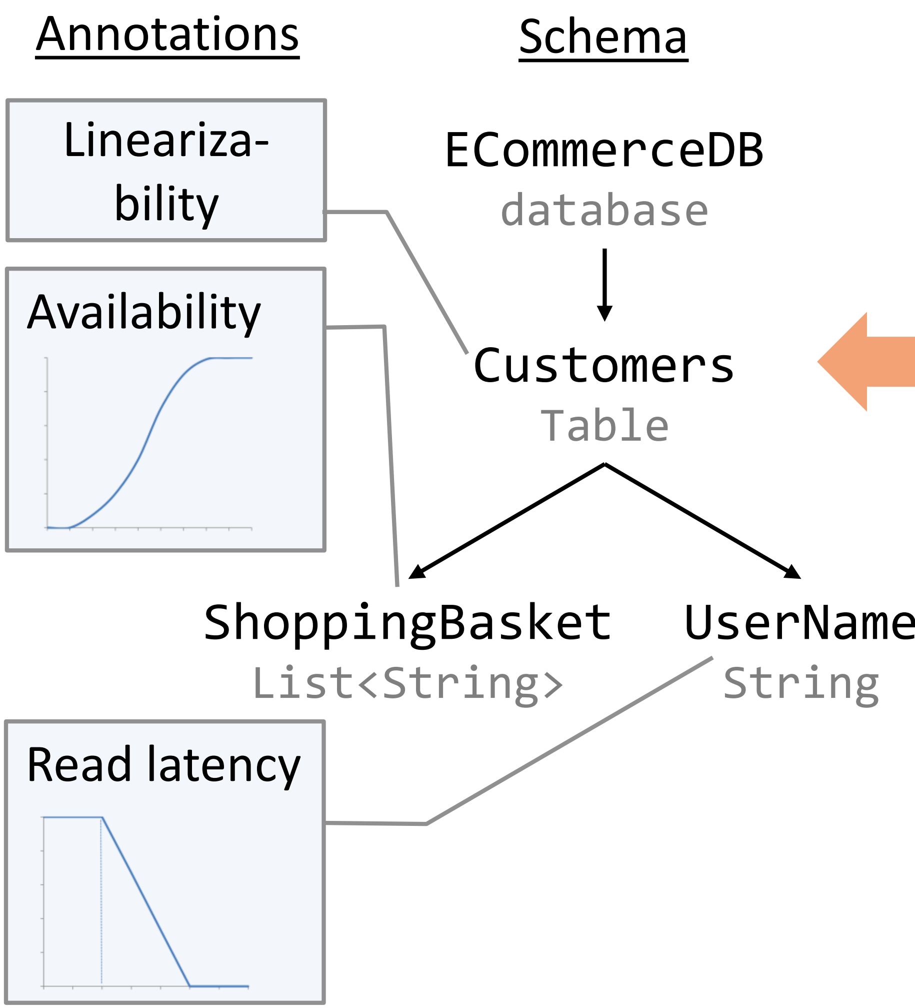
DB	Score
MongoDB	0.9
Redis	0.525
MySQL	0.12
HBase	0.5

Binary requirement →

1. Exclude DBs that do not support it
2. Recursive descent
3. Pick DB with best total score and add it to routing model

Step II - Resolution

Ranking algorithm by example



DB	Score
MongoDB	0.9
Redis	0.525
MySQL	0.12
HBase	0.5

Binary requirement →

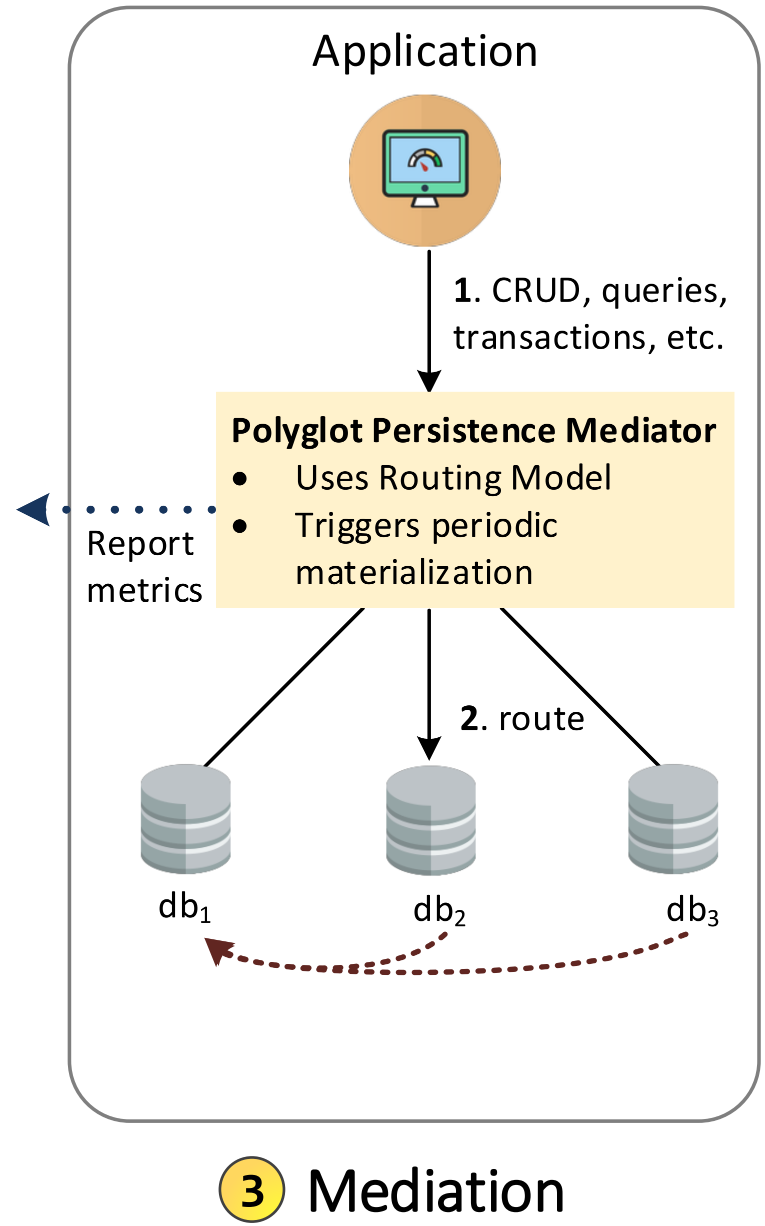
1. Exclude DBs that do not support it
2. Recursive descent
3. Pick DB with best total score and add it to routing model

Routing Model:
Customers → MongoDB

Step III - Mediation

Routing data and operations

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



Evaluation: News Article

Prototype built on ORESTES

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

Article

Europäische Zentralbank überfallen: Bankräuber erbeutet 1,14 Billionen Euro

Frankfurt (dpo) - Eine rekordverdächtige Summe hat heute Mittag ein Räuber bei einem Überfall auf die Europäische Zentralbank (EZB) in Frankfurt erbeutet. Der Mann, der inzwischen als Kleinkrimineller mit dem Namen Kalle Kowalski (43) identifiziert wurde, befindet sich derzeit mit 1,14 Billionen Euro auf der Flucht. Der Stadtteil Ostend ist vollständig abgeriegelt.
mehr...

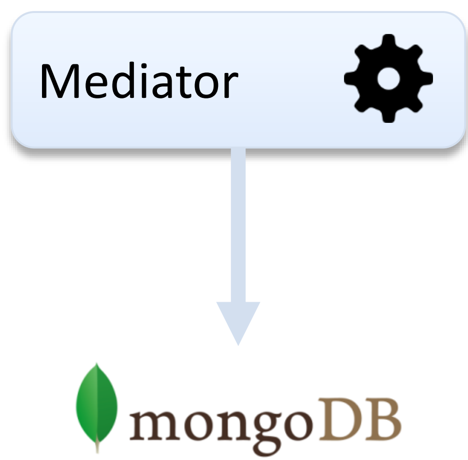
Counter

1.344.222 gelesen

Evaluation: News Article

Prototype built on 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

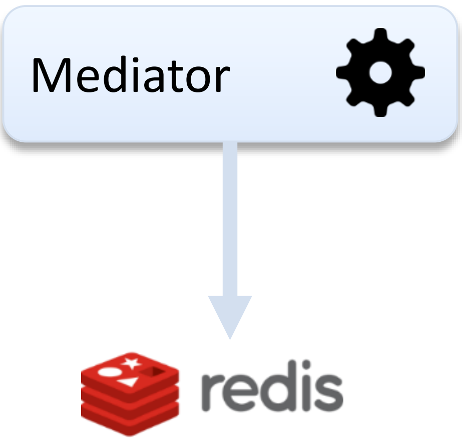
Scenario: news articles with impression counts
Objectives: low-latency top-k queries, high-throughput counts, article-queries

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



Evaluation: News Article

Prototype built on ORESTES

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

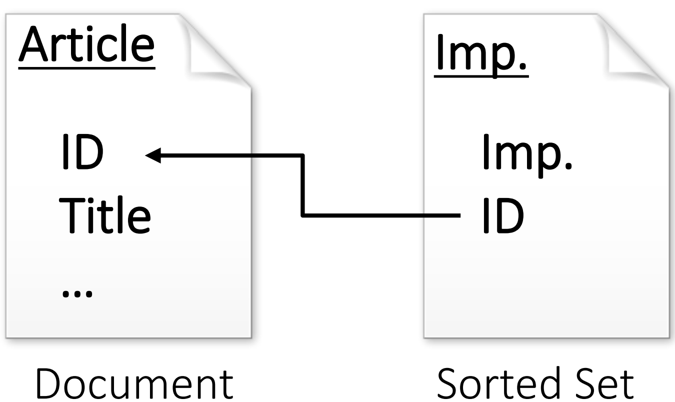
No powerful queries

Evaluation: News Article

Prototype built on ORESTES

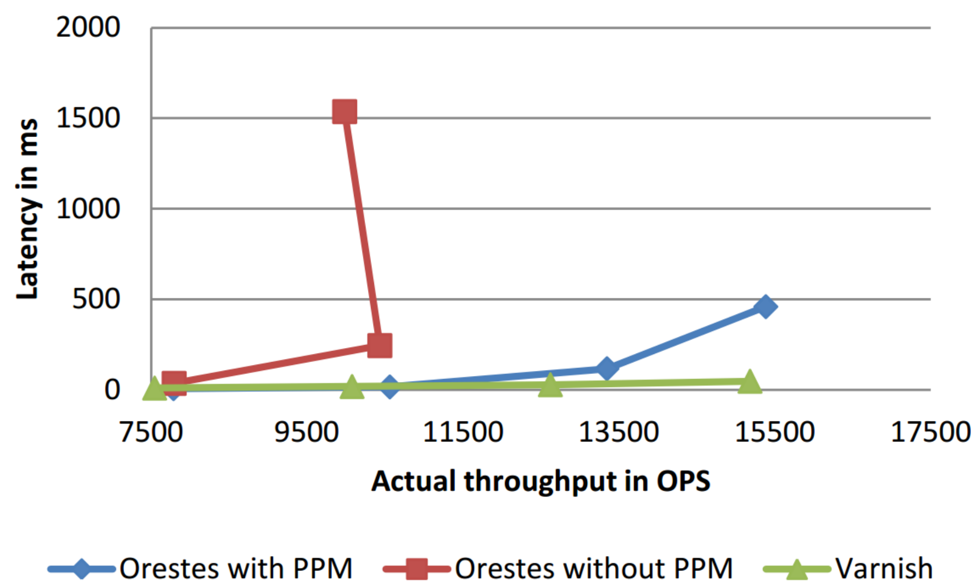
Scenario: news articles with impression counts

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



mongoDB redis

Found Resolution



Challgenges & Future Work

Workload Management: during mediation actively schedule requests based on requirements

Ranking: Predict future metrics from historic ones (*time-series analysis*) or from performance models

Database selection: minimize $P(SLA\ violation)\ *\ penalty$ (e.g. through *reinforcement learning*)

Challgenges & Future Work

Meta-DBaaS: Mediate over DBaaS-systems and factor in their SLAs

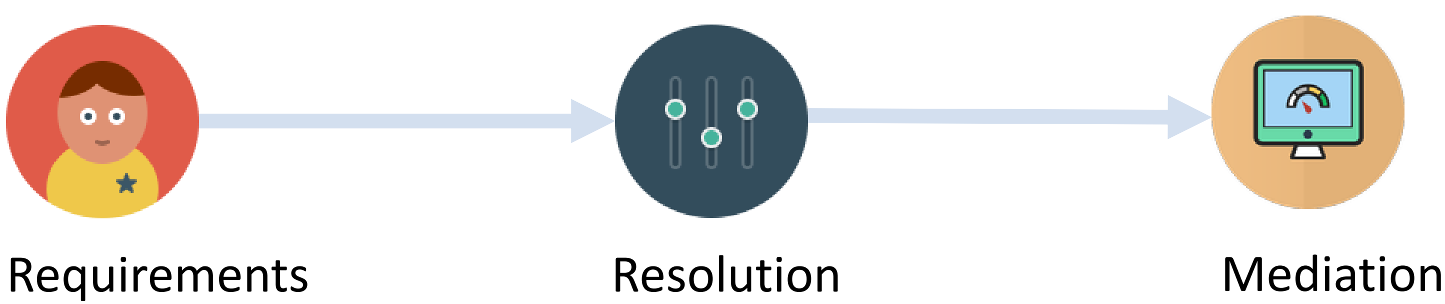
Live Migration: Enable requirement changes

Requirements: collect library of common ones

Utility: Provide intuitive, visual „knobs“ for developers

Summary

- (Manual) Polyglot Persistence is a reality - but difficult and error-prone
- **Polyglot Persistence Mediator**: SLA-driven, fine-grained selection of database systems
    1. Let the tenant define his requirements
    2. Choose or provision a database based on that
    3. Route data and operations according to that mapping



Thank you.

gessert@informatik.uni-hamburg.de
Orestes.info
Baqend.com