ORESTES: Scalability and Low Latency for Polyglot Database Services

Felix Gessert, Norbert Ritter
felix.gessert@gmail.com

About me

- Since 2 years: PhD student (advisor: Norbert Ritter)
- Since 1 year: CEO of Baqend

Research Project for PhD

Orestes

Baqend

Build faster Apps faster.

Cloud Database Startup

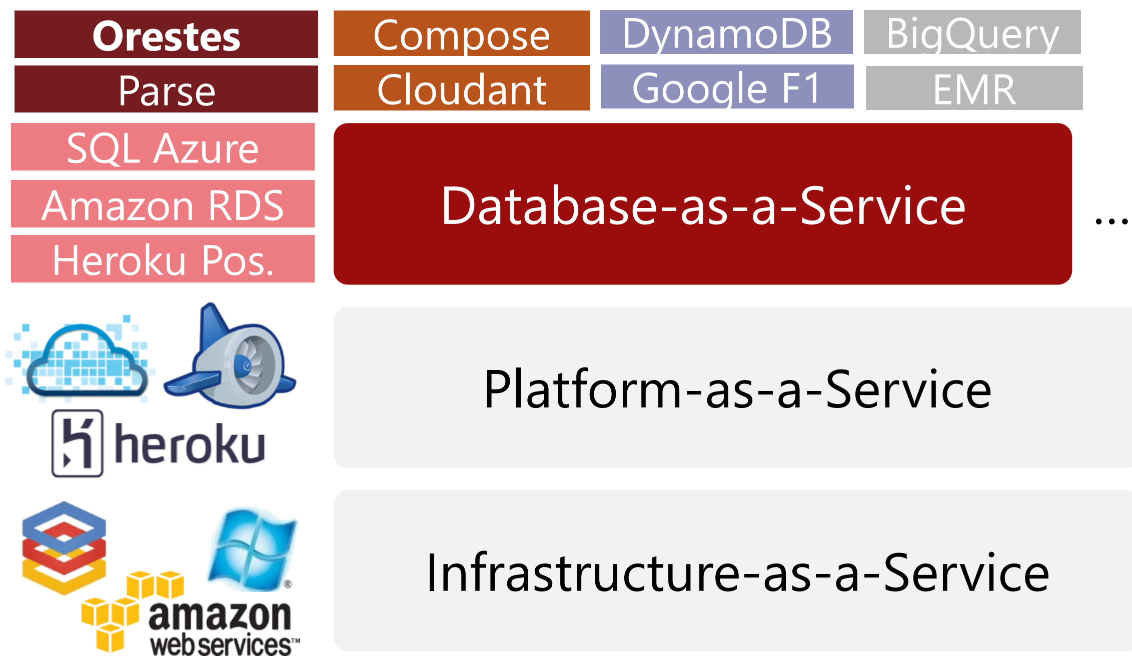
Outline

- **Motivation**
- Cache Sketch Approach
- Polyglot Persistence Mediator
- Wrap-up and future work

- Cloud Databases
- The Latency Problem of the web

Introduction: Which classes of cloud databases are there?

Cloud Databases



Cloud Databases

Orestes | Compose | DynamoDB | BigQuery

Parse | Cloudant | Google F1 | EMR

SQL Azure

Amazon RDS

Heroku Pos.

Database-as-a-Service …

Managed RDBMS

heroku

Platform-as-a-Service

amazon webservices™

Infrastructure-as-a-Service

Cloud Databases

Orestes | Compose | DynamoDB | BigQuery

Parse | Cloudant | Google F1 | EMR

SQL Azure

Amazon RDS

Heroku Pos.

Database-as-a-Service …

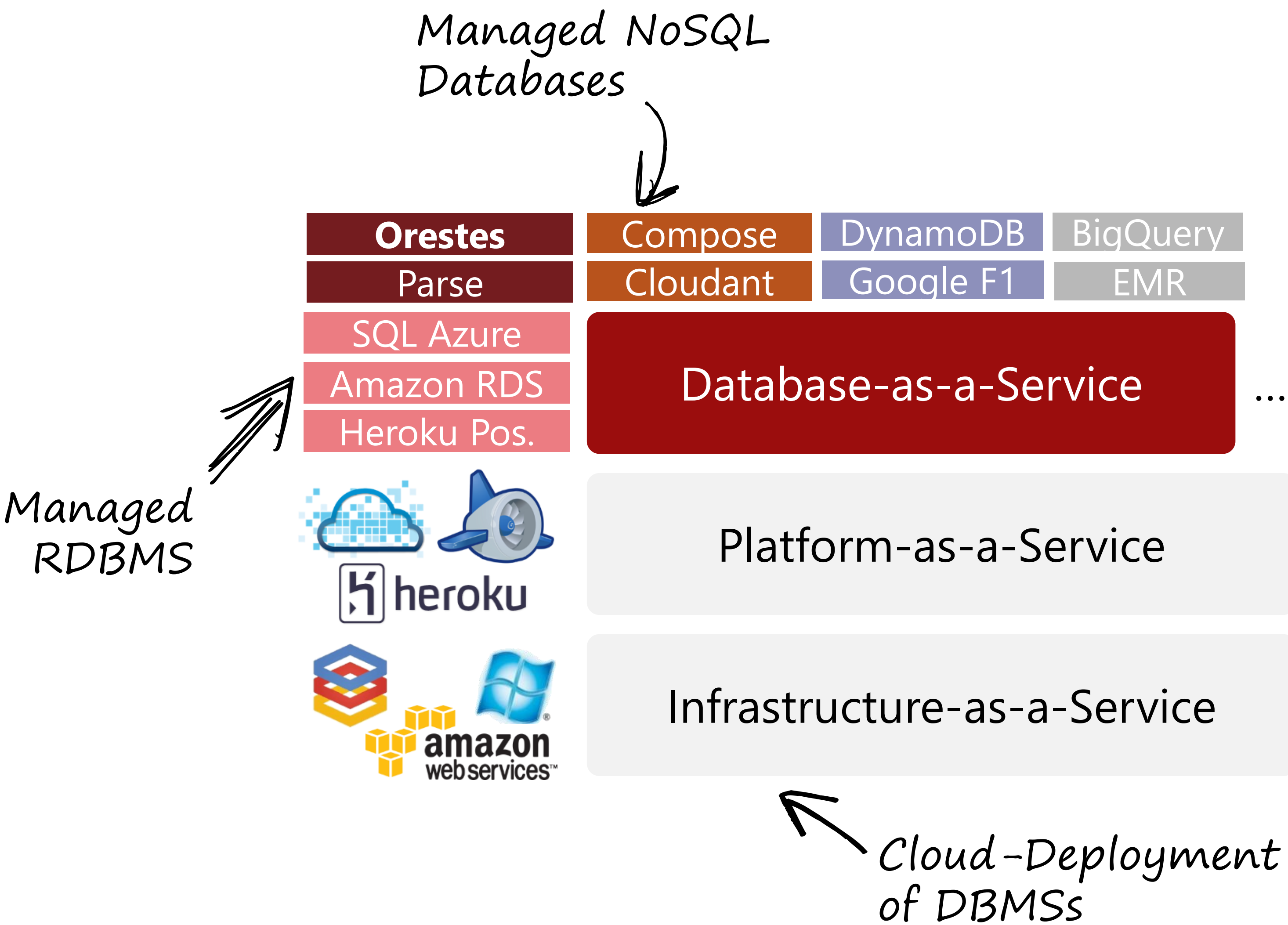
Managed RDBMS

Platform-as-a-Service

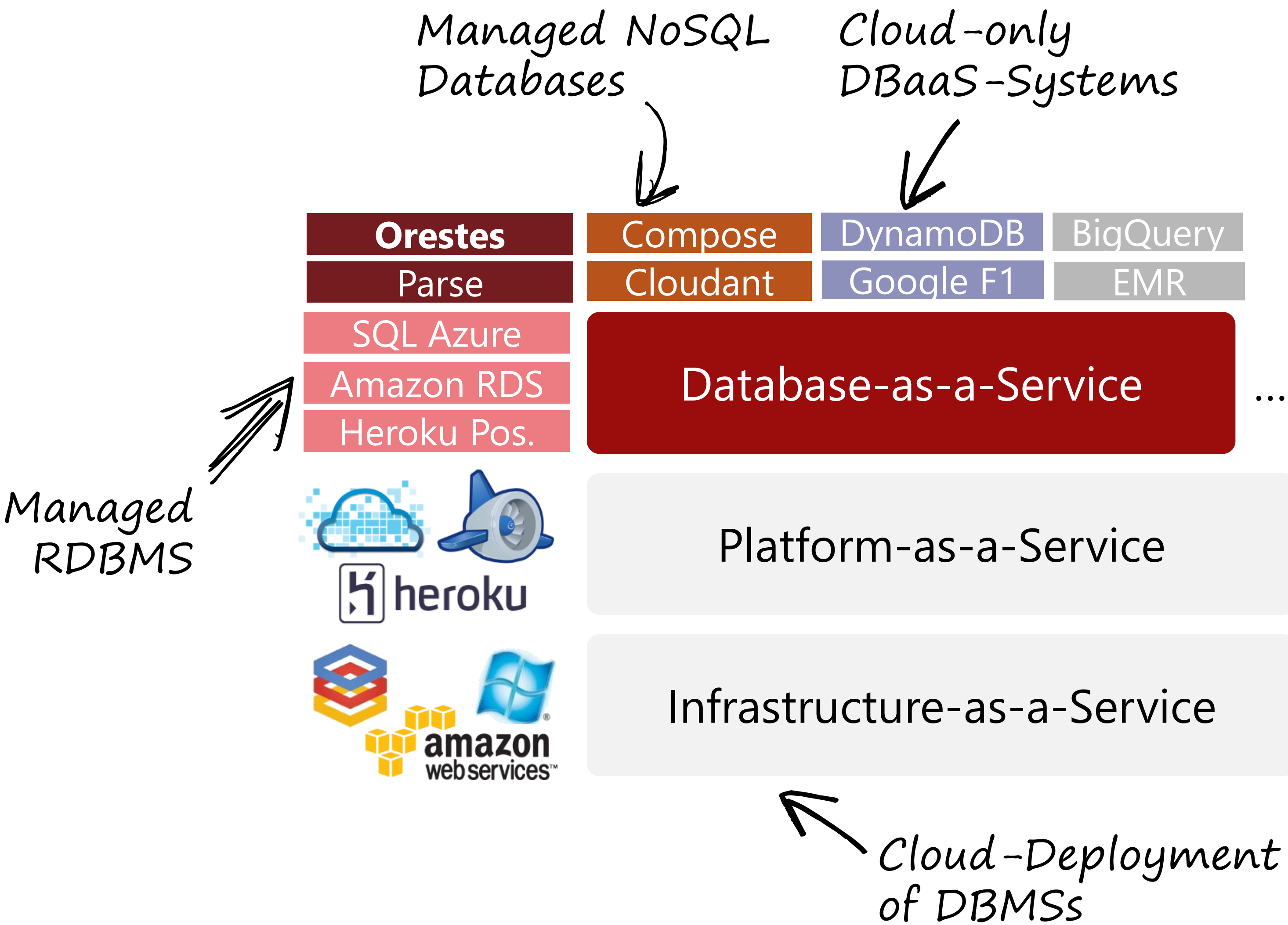
Infrastructure-as-a-Service

Cloud-Deployment of DBMSs

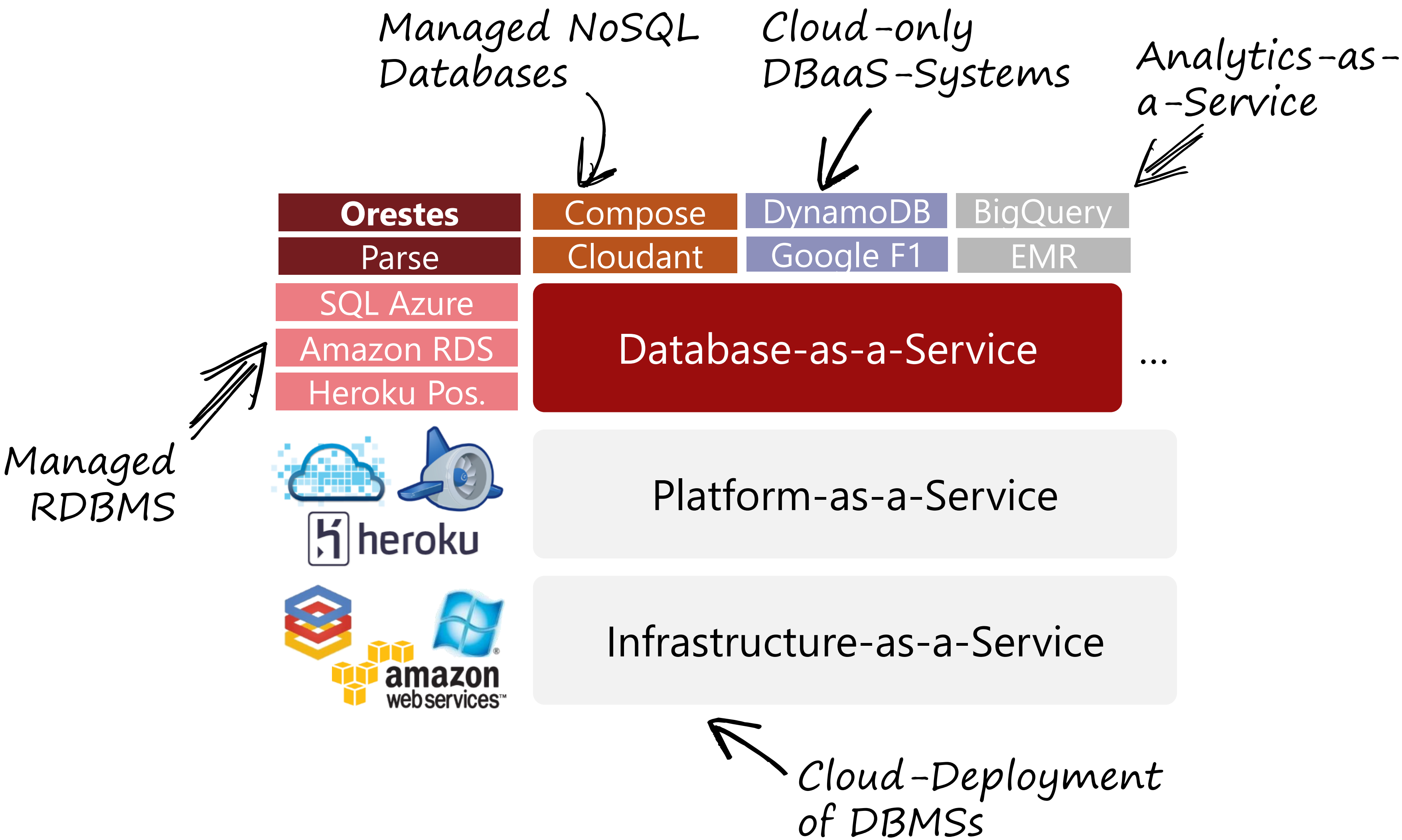
Cloud Databases



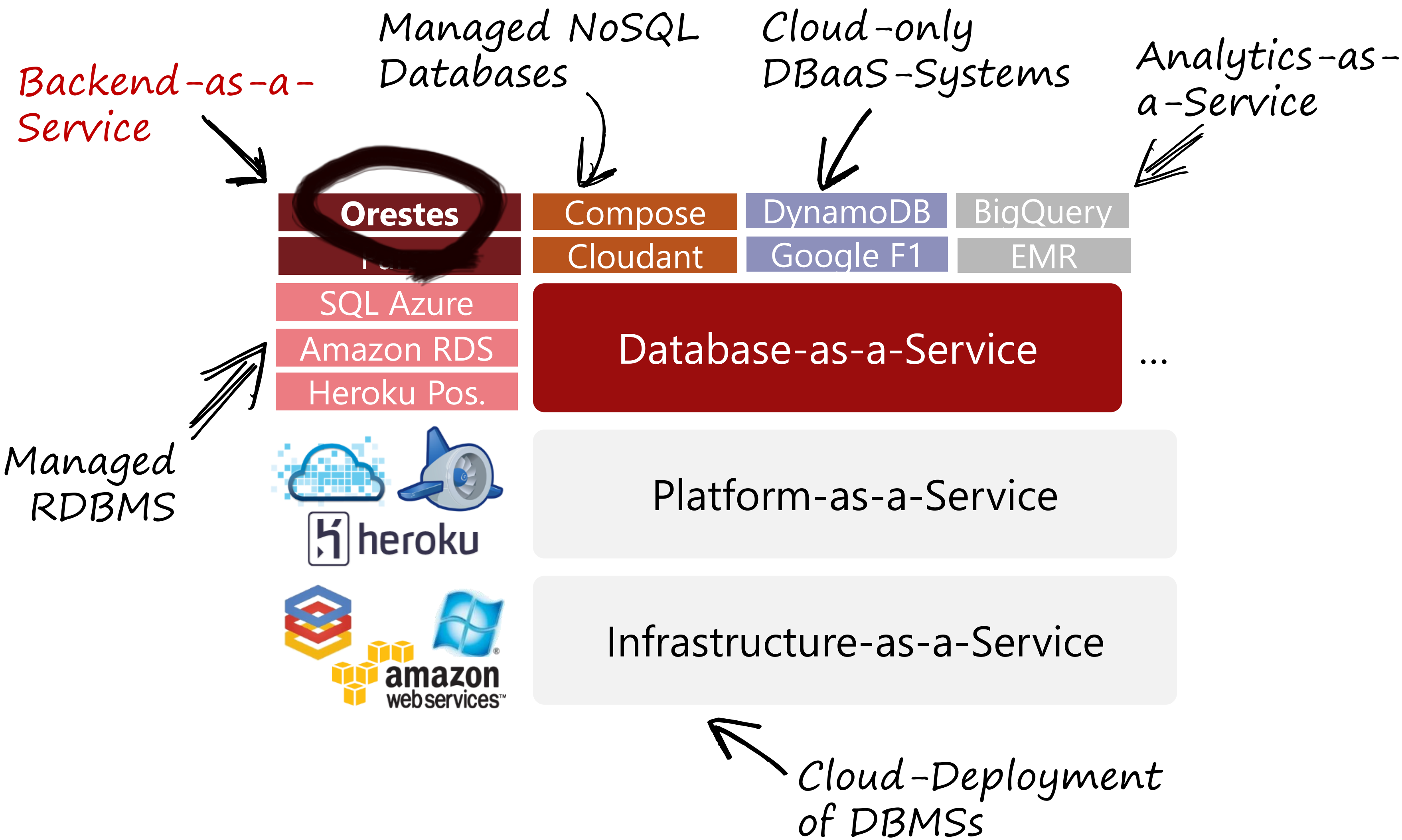
Cloud Databases



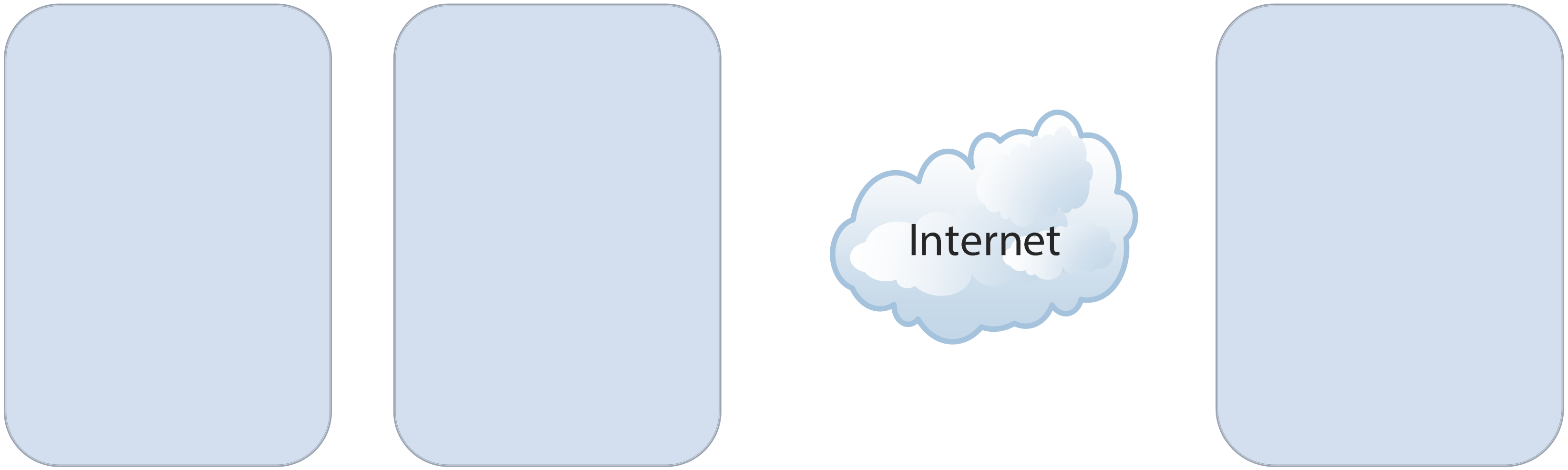
Cloud Databases



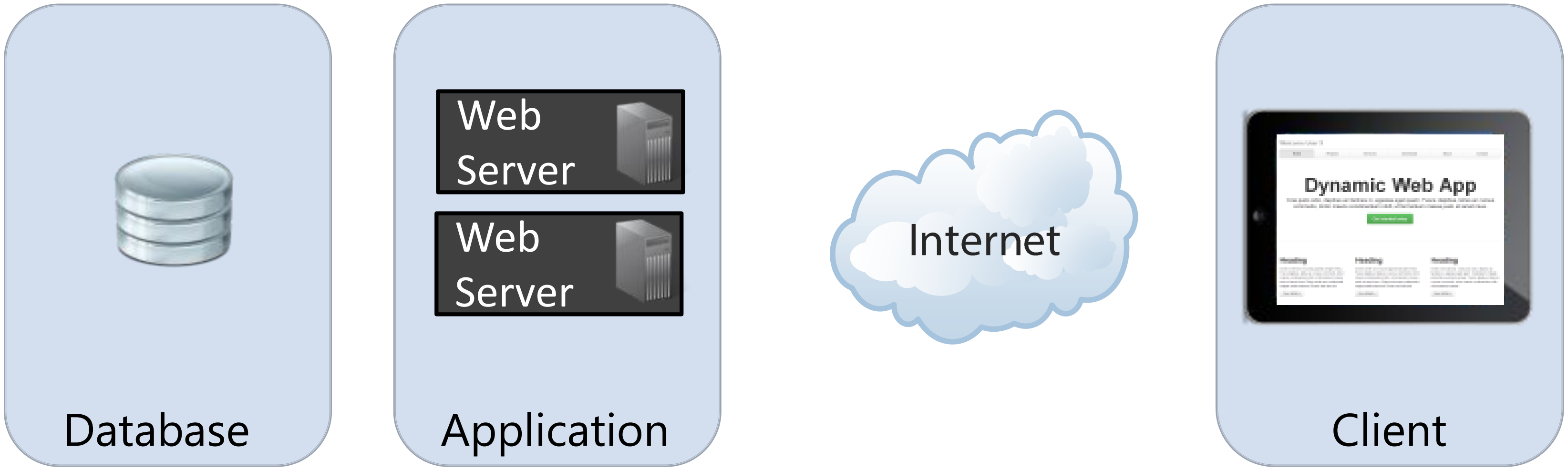
Cloud Databases



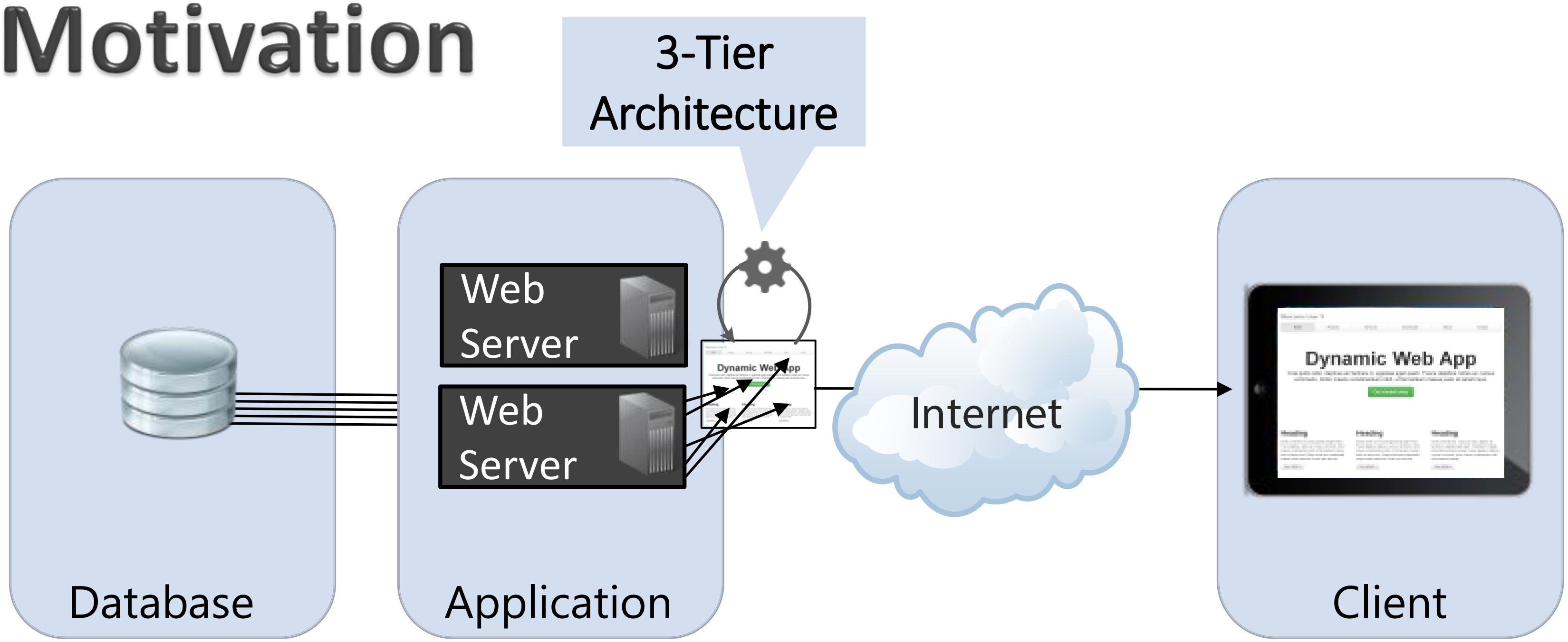
Motivation



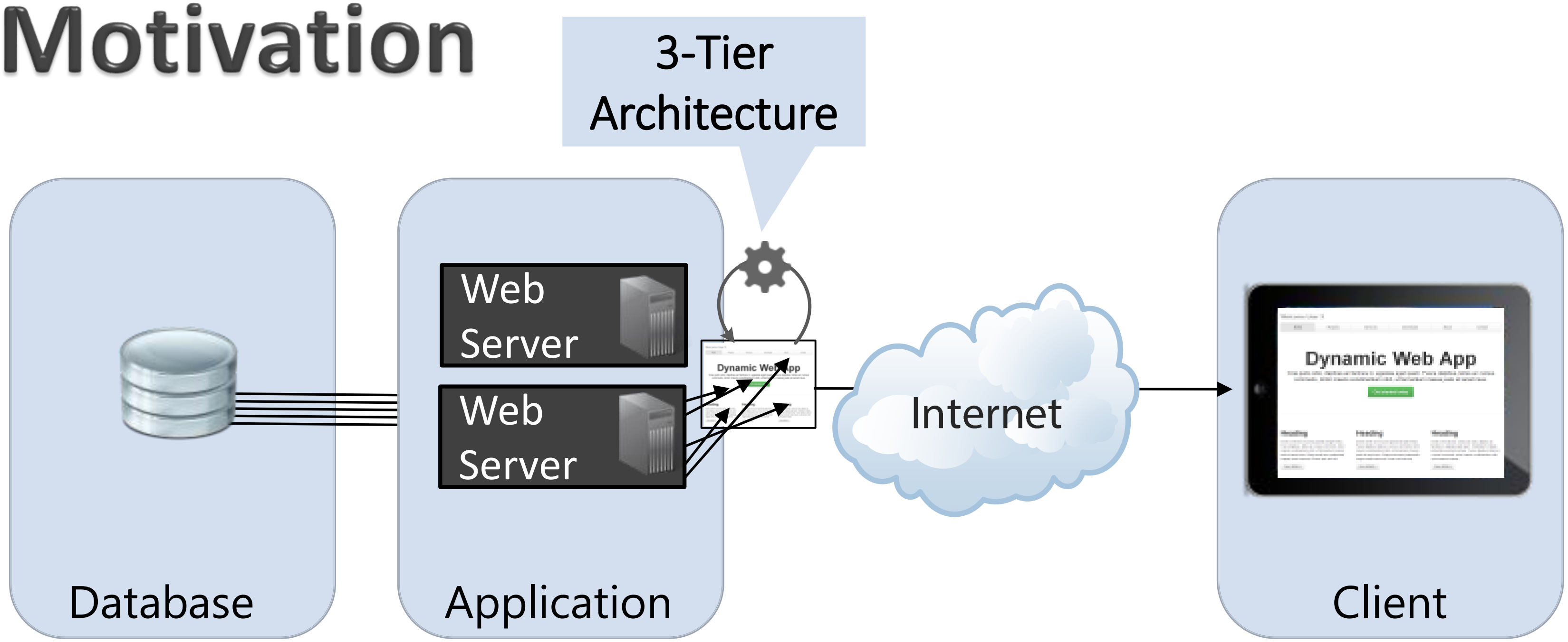
Motivation


Web Server
Web Server
Internet
Dynamic Web App
Database
Application
Client

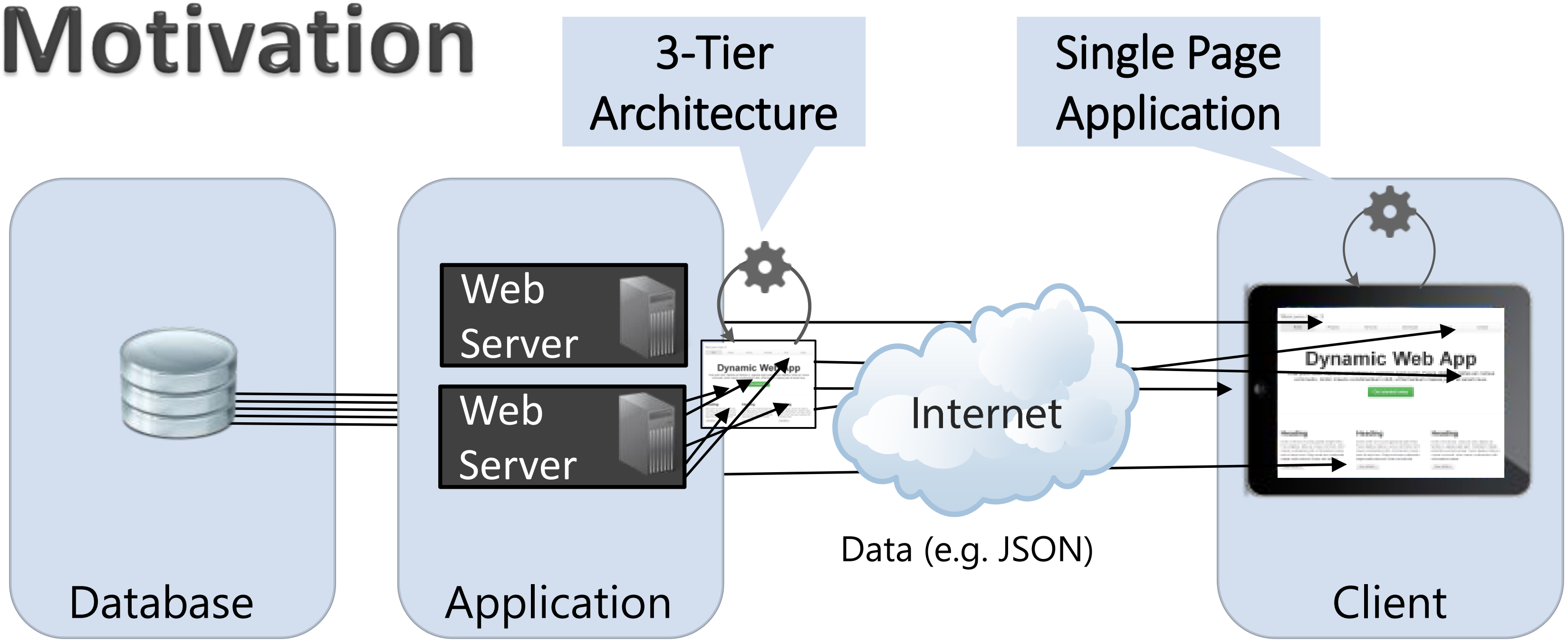
Motivation


3-Tier
Architecture
Web
Server
Web
Server
Dynamic Web App
Internet
Dynamic Web App
Database
Application
Client

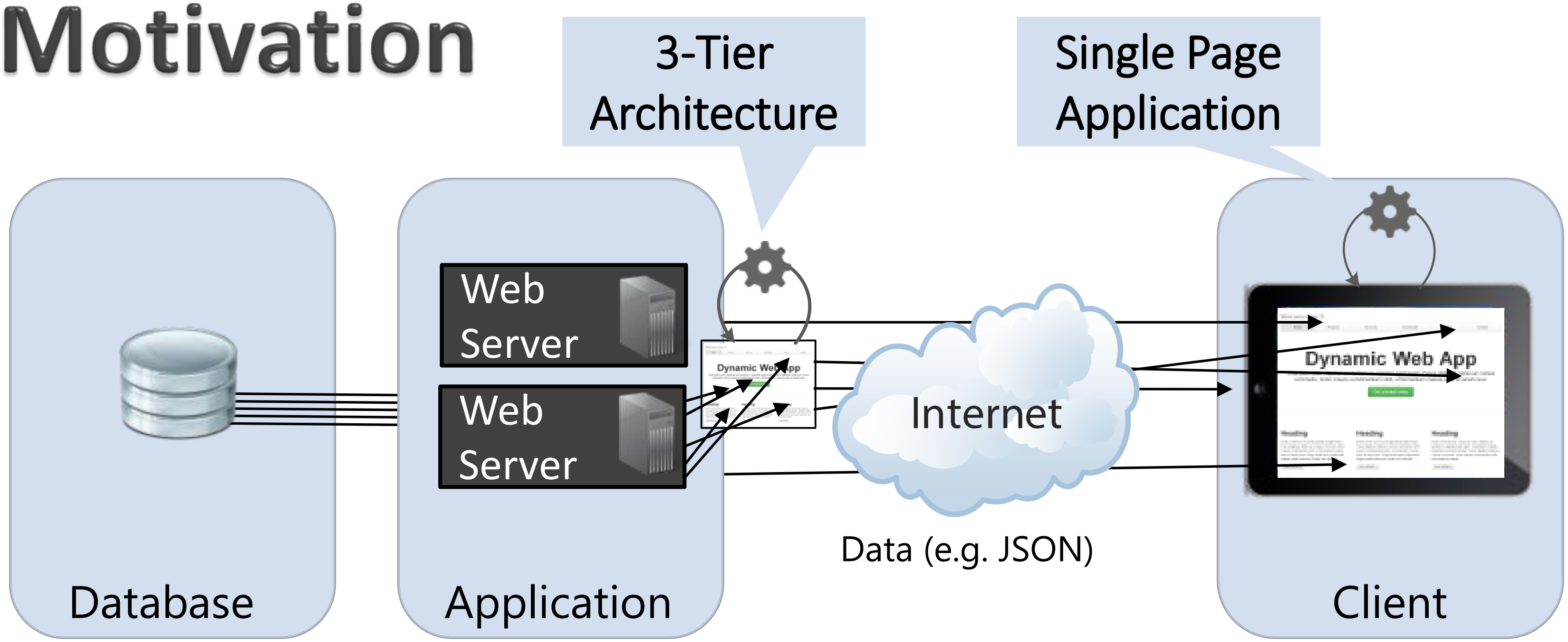
Motivation


3-Tier
Architecture
Web
Server
Web
Server
Dynamic Web App
Internet
Dynamic Web App
Database
Application
Client

Motivation

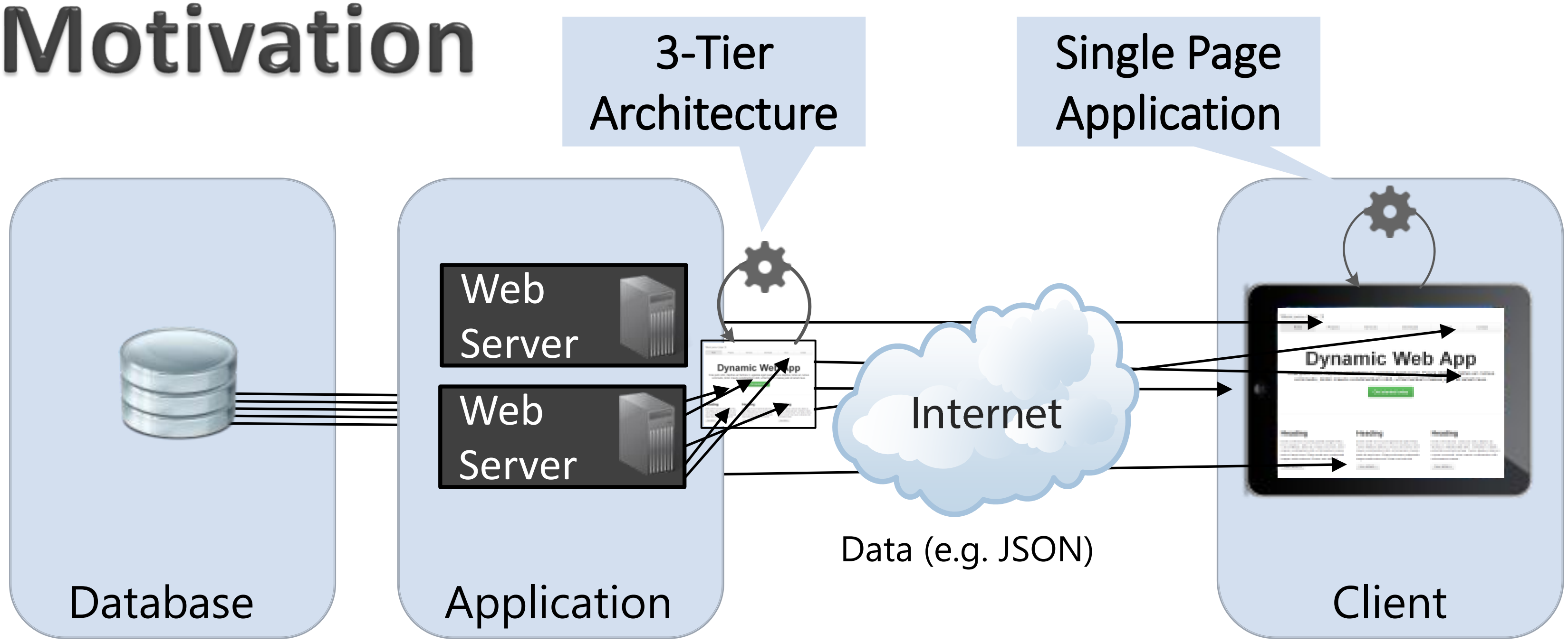

3-Tier Architecture
Single Page Application
Web Server
Web Server
Dynamic Web App
Internet
Dynamic Web App
Data (e.g. JSON)
Database
Application
Client

Motivation



Redundance
Applications reimplement
backend functionality

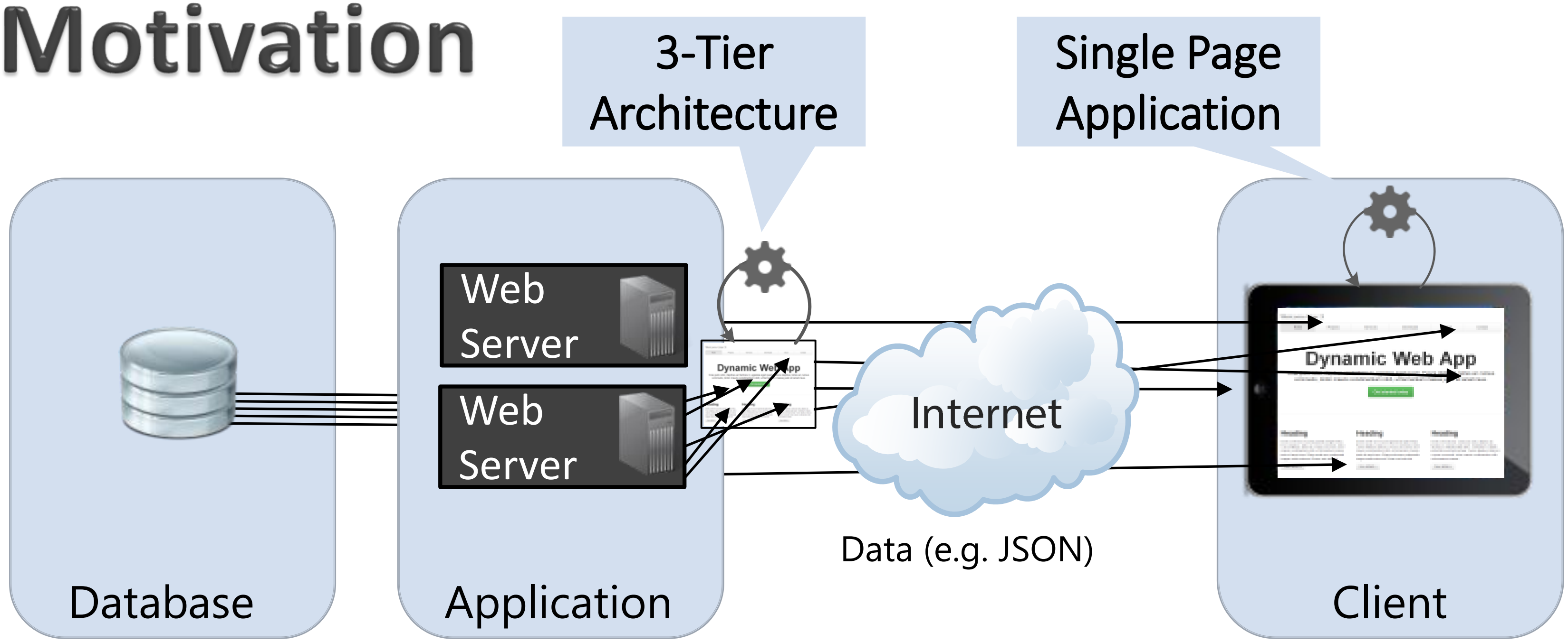
Motivation



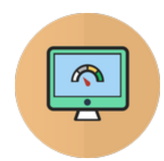
Redundance
Applications reimplement backend functionality

Database Dependence
Strong coupling between application and database system

Motivation



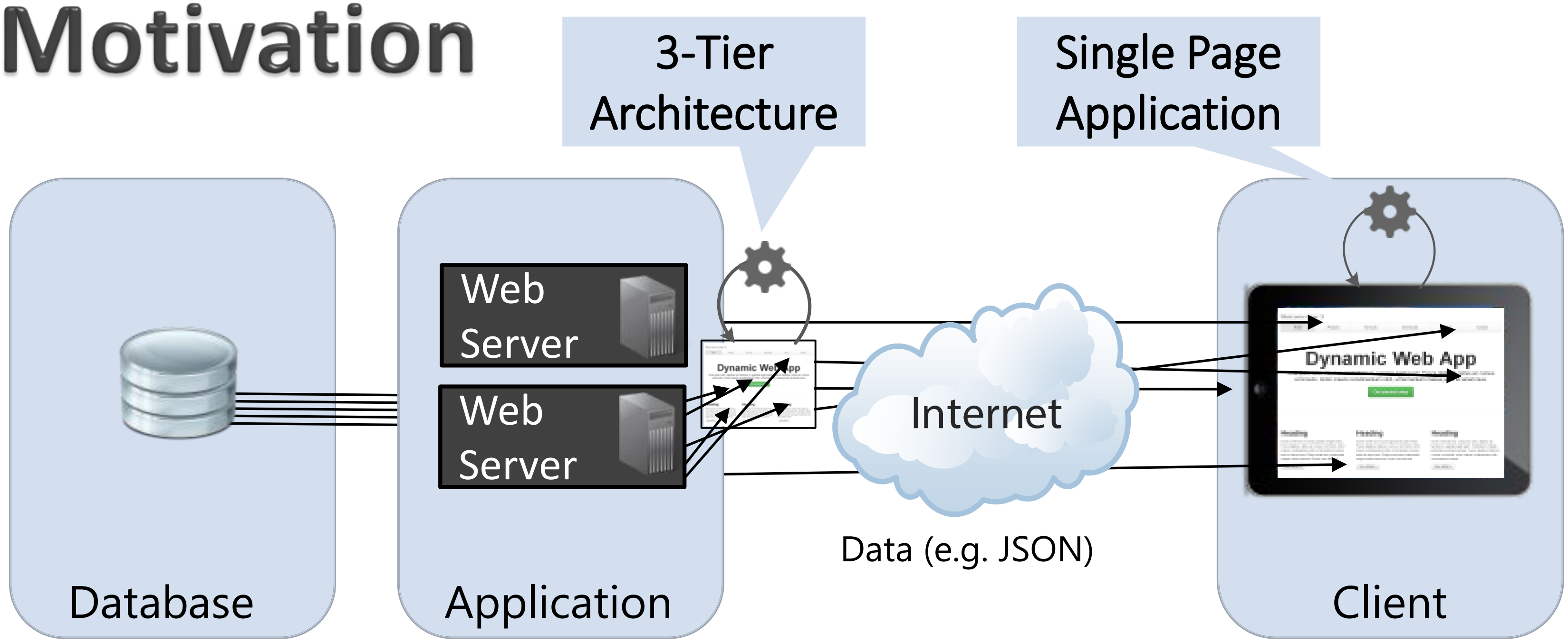
Redundance
Applications reimplement backend functionality



Performance & Scalability
Error-prone application-specific scaling policies

Database Dependence
Strong coupling between application and database system

Motivation



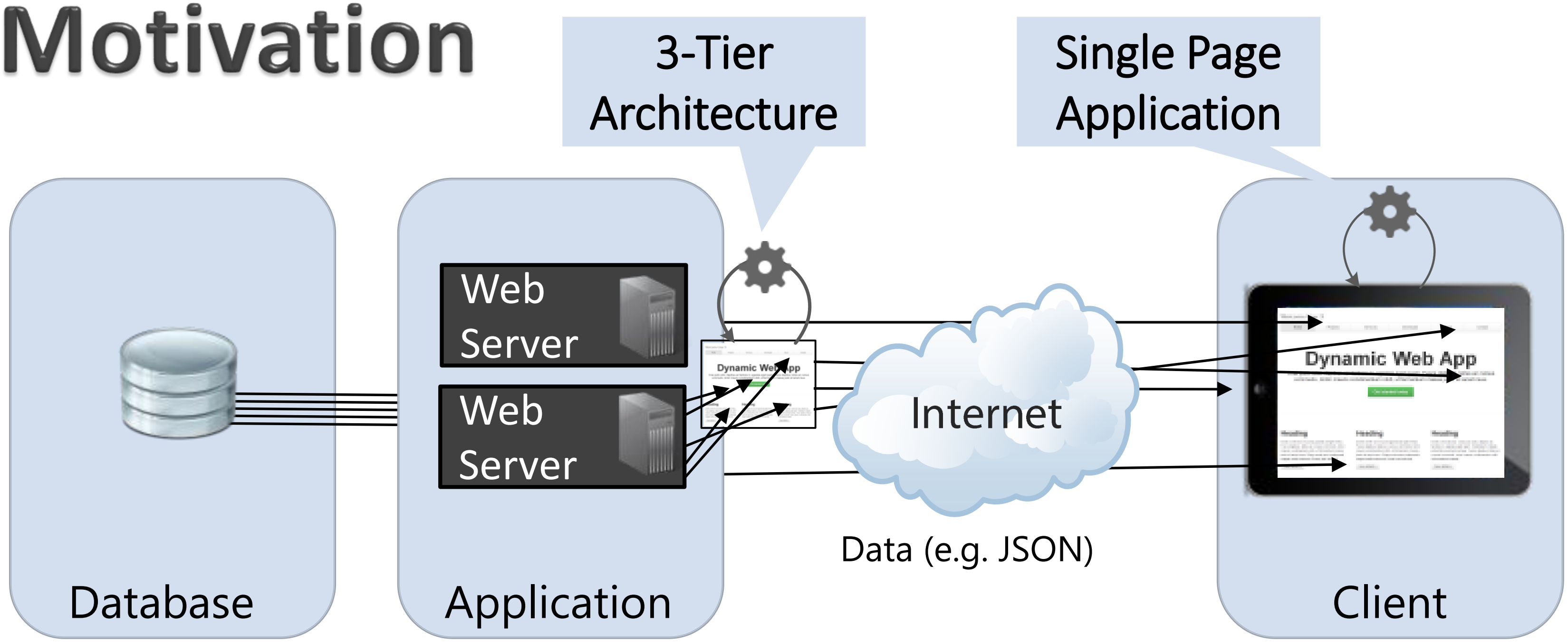
Redundance
Applications reimplement backend functionality

Performance & Scalability
Error-prone application-specific scaling policies

Database Dependence
Strong coupling between application and database system

Difficult Development
Both client and server have to be implemented

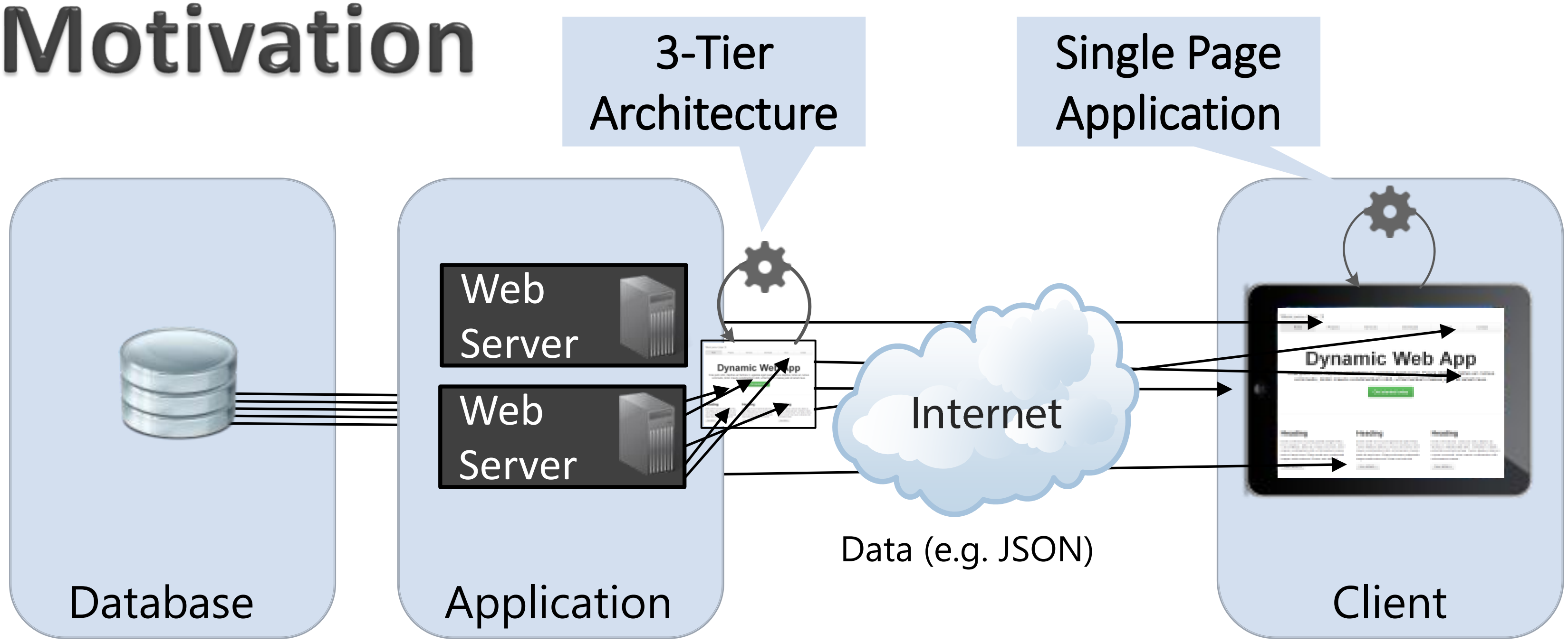
Motivation



With every **100ms** of additional page load time, revenue decreases by 1%.

Study by **Amazon**

Motivation



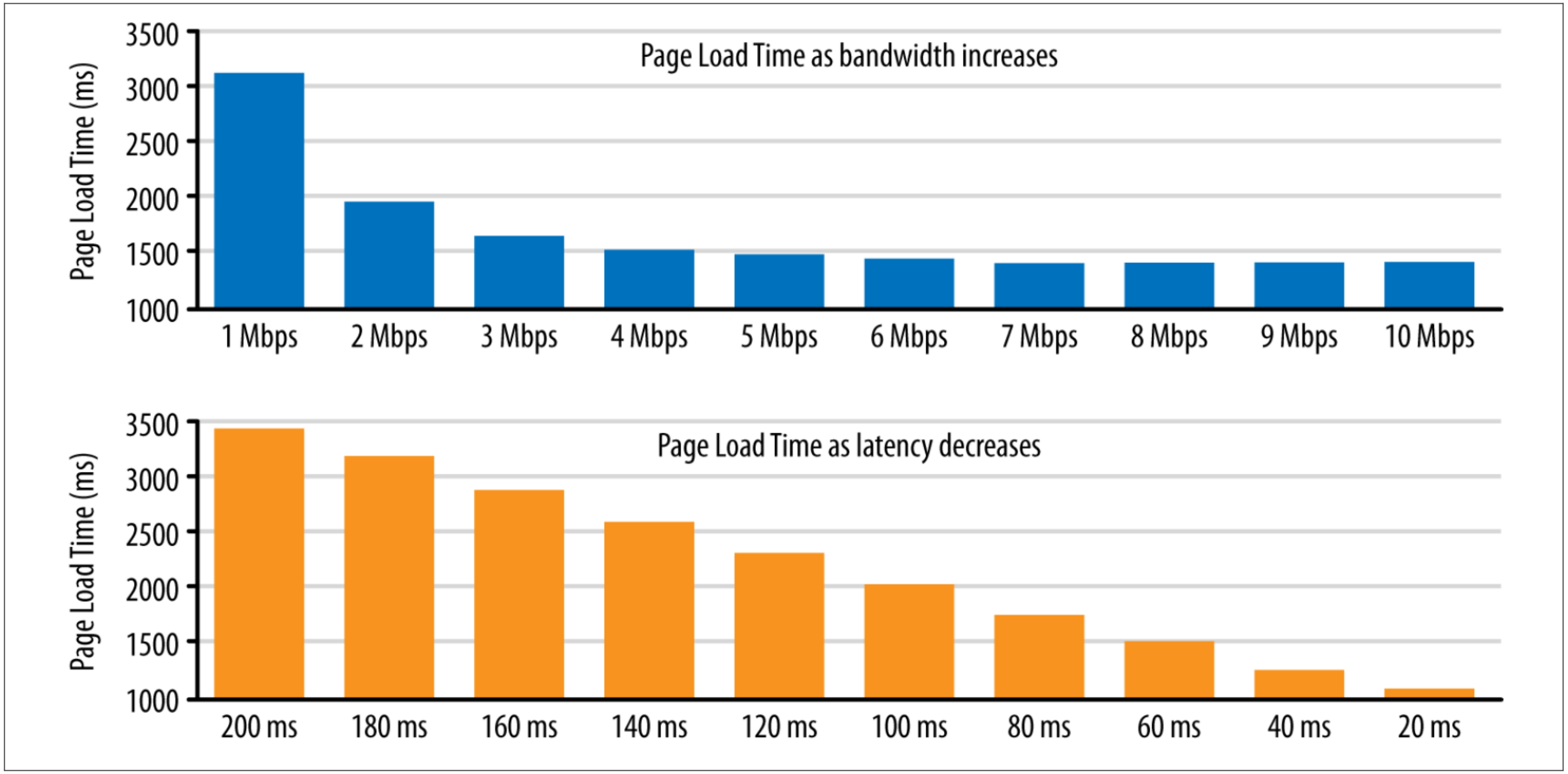
When increasing load time of search results by **500ms**, traffic decreases by 20%.

Study by **Google**

Motivation

3-Tier Architecture

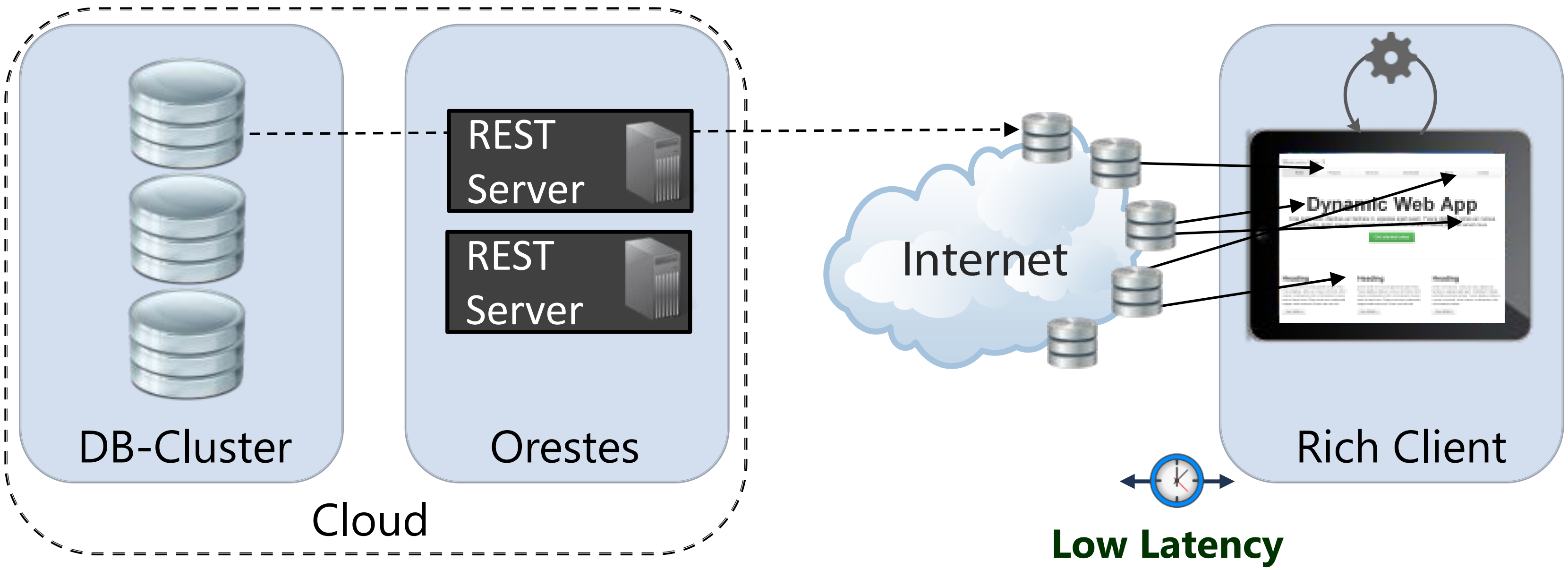
Single Page Application



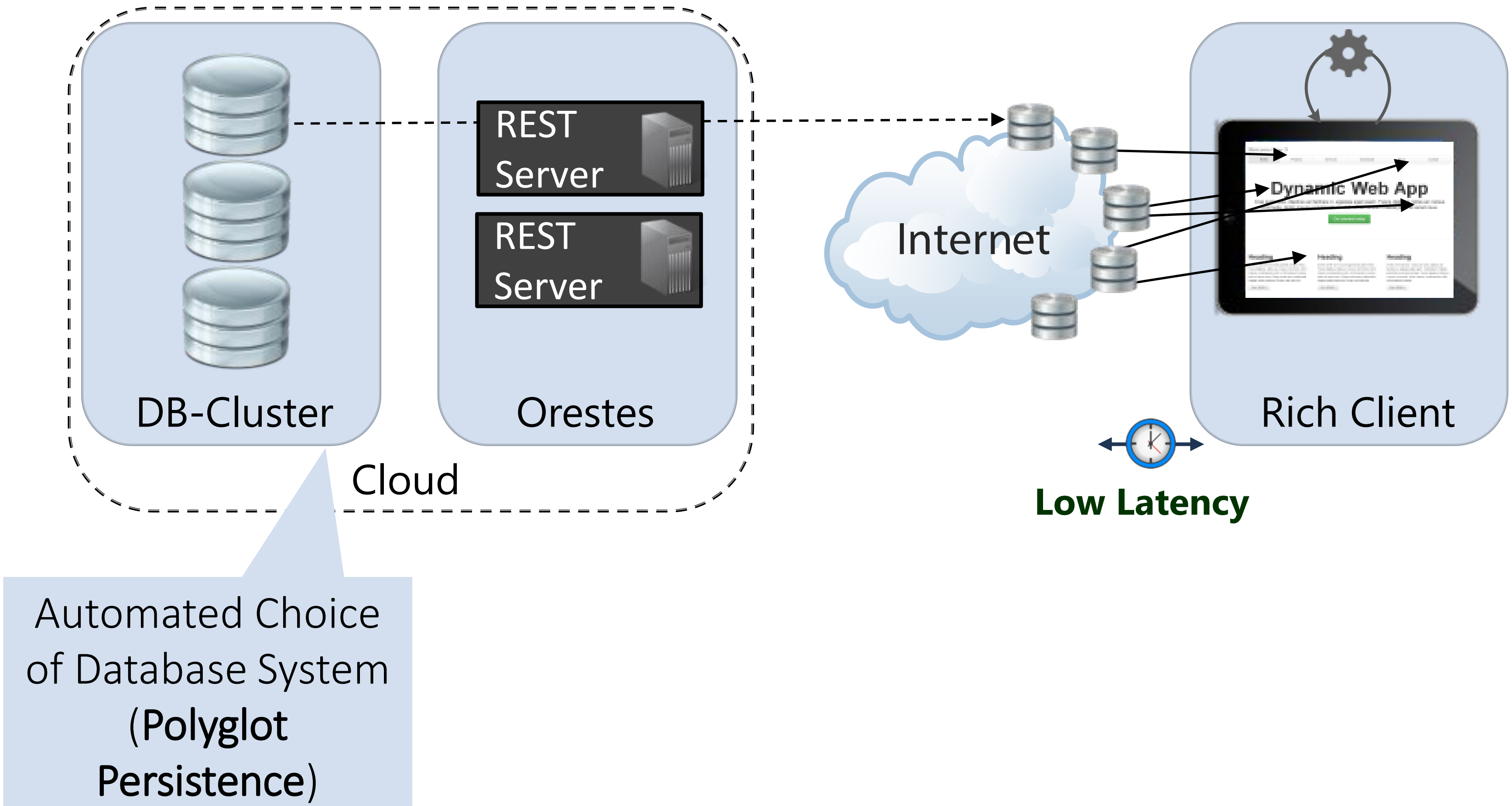
20%.

Study by **Google**

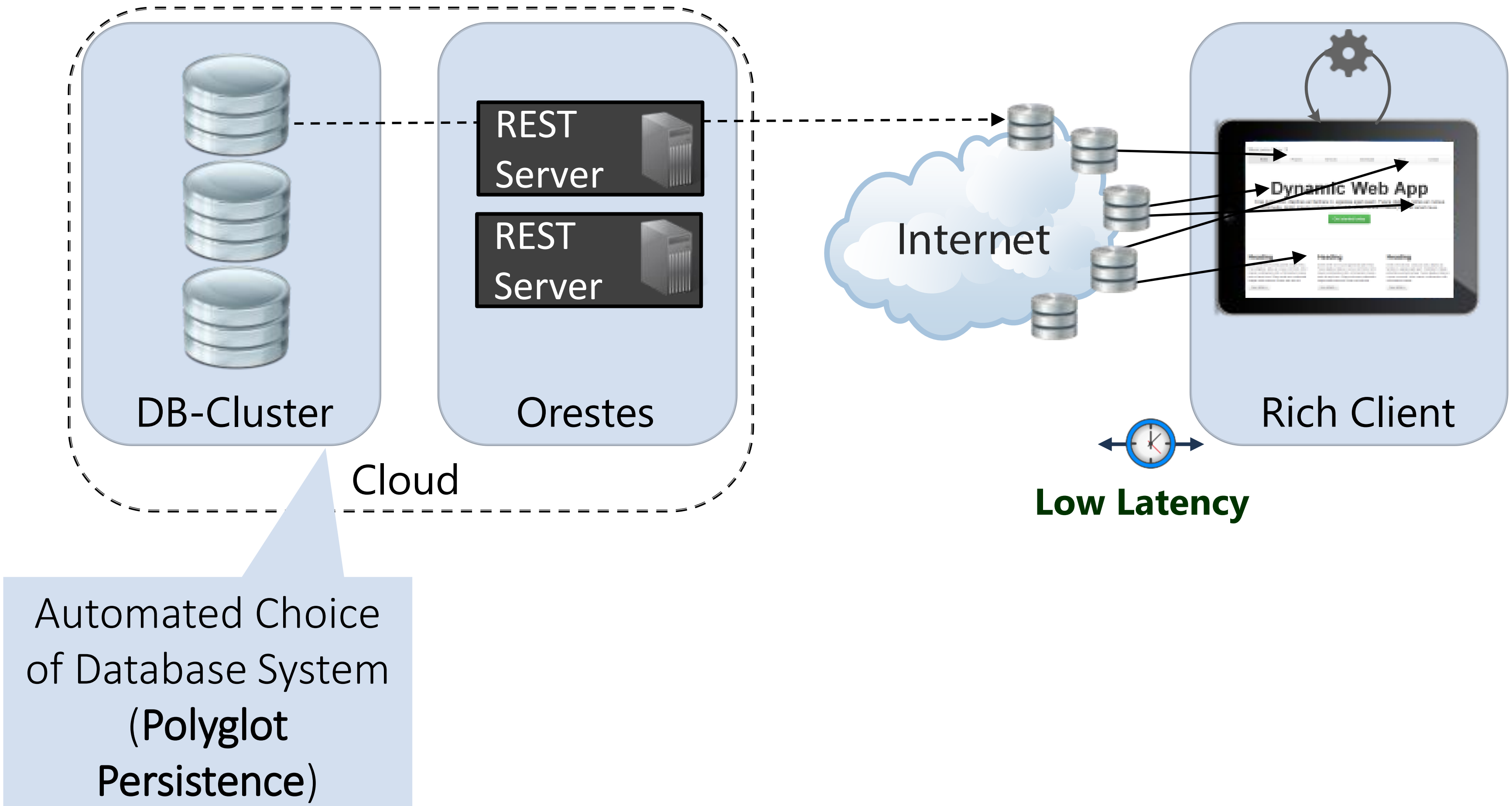
Vision: Orestes



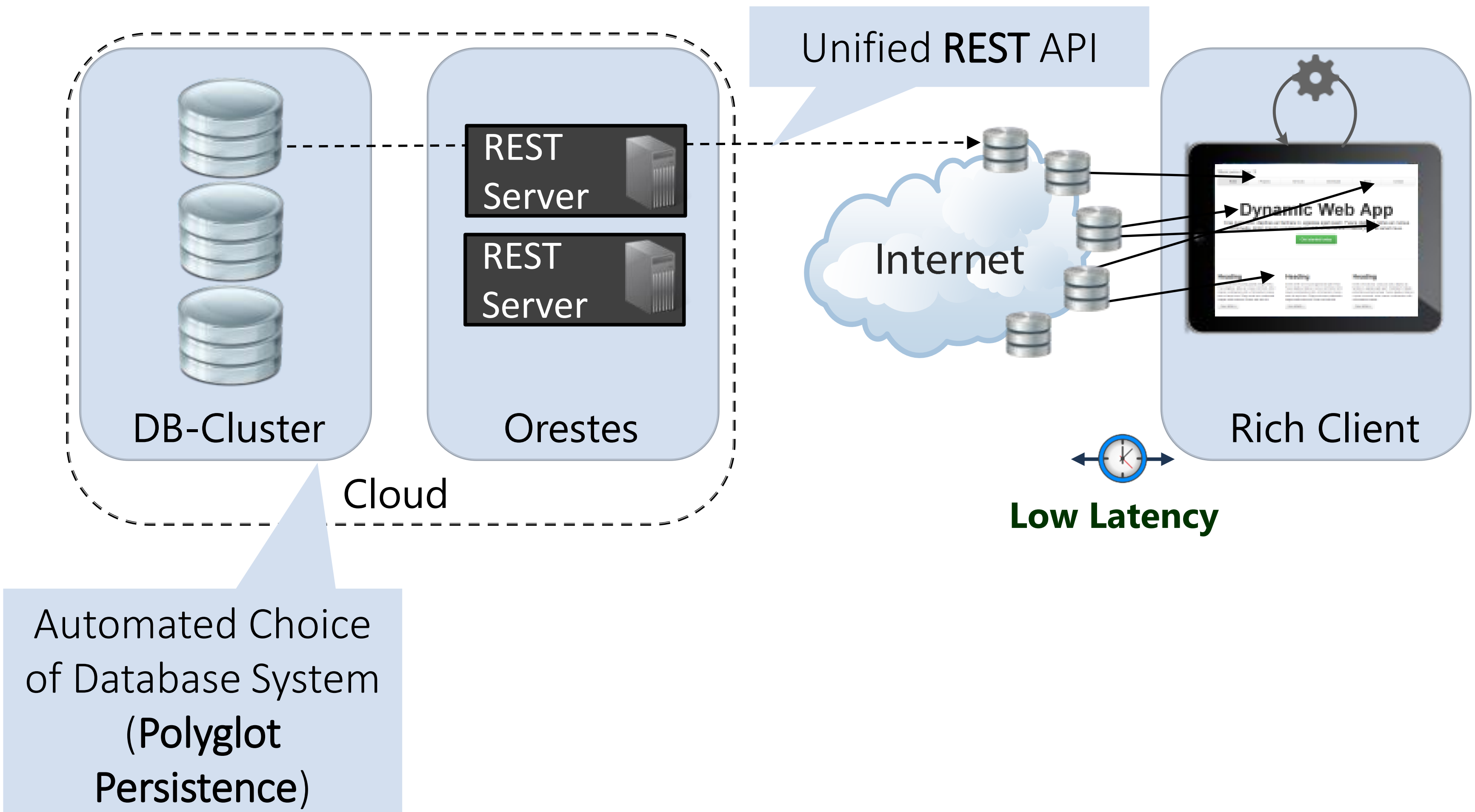
Vision: Orestes



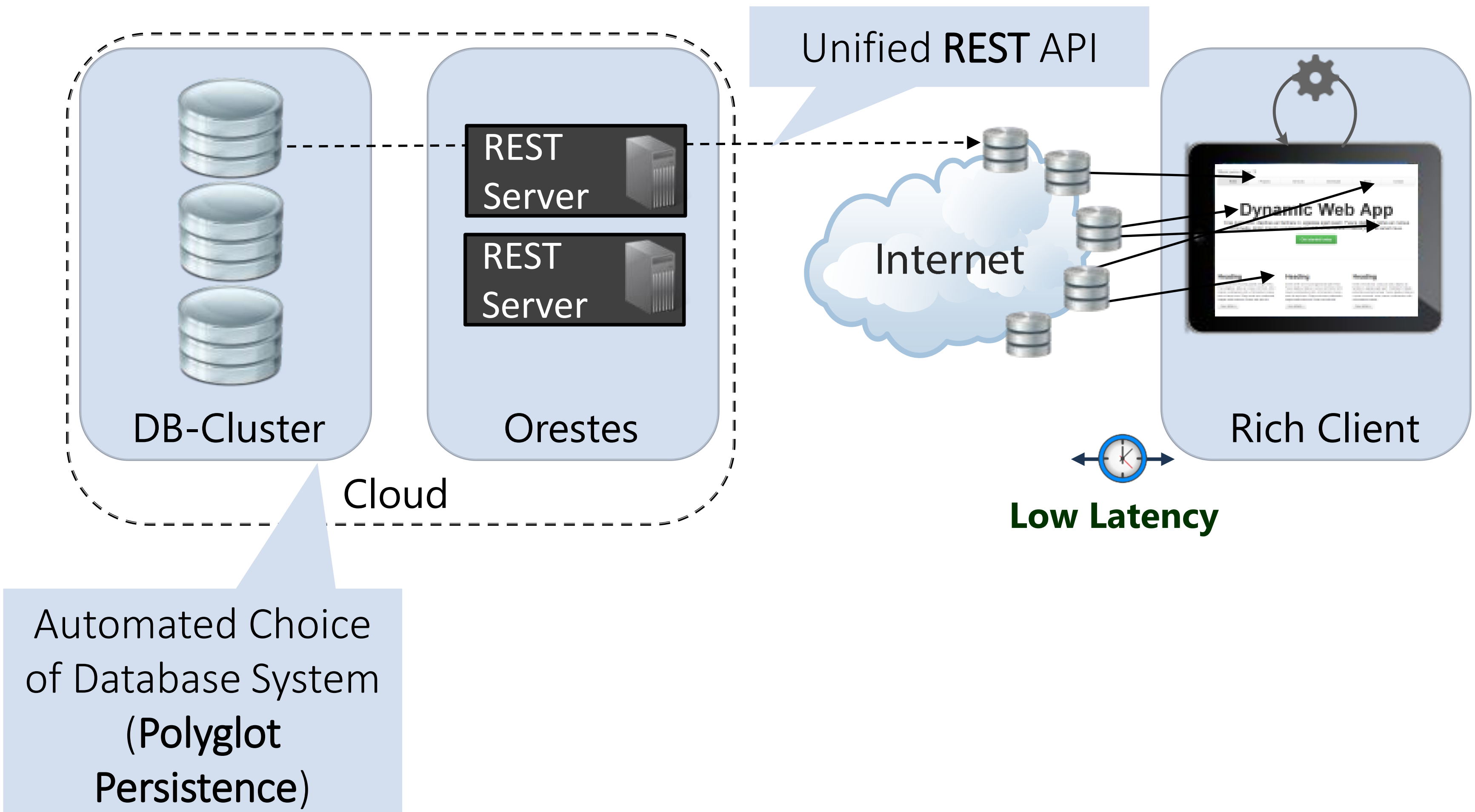
Vision: Orestes



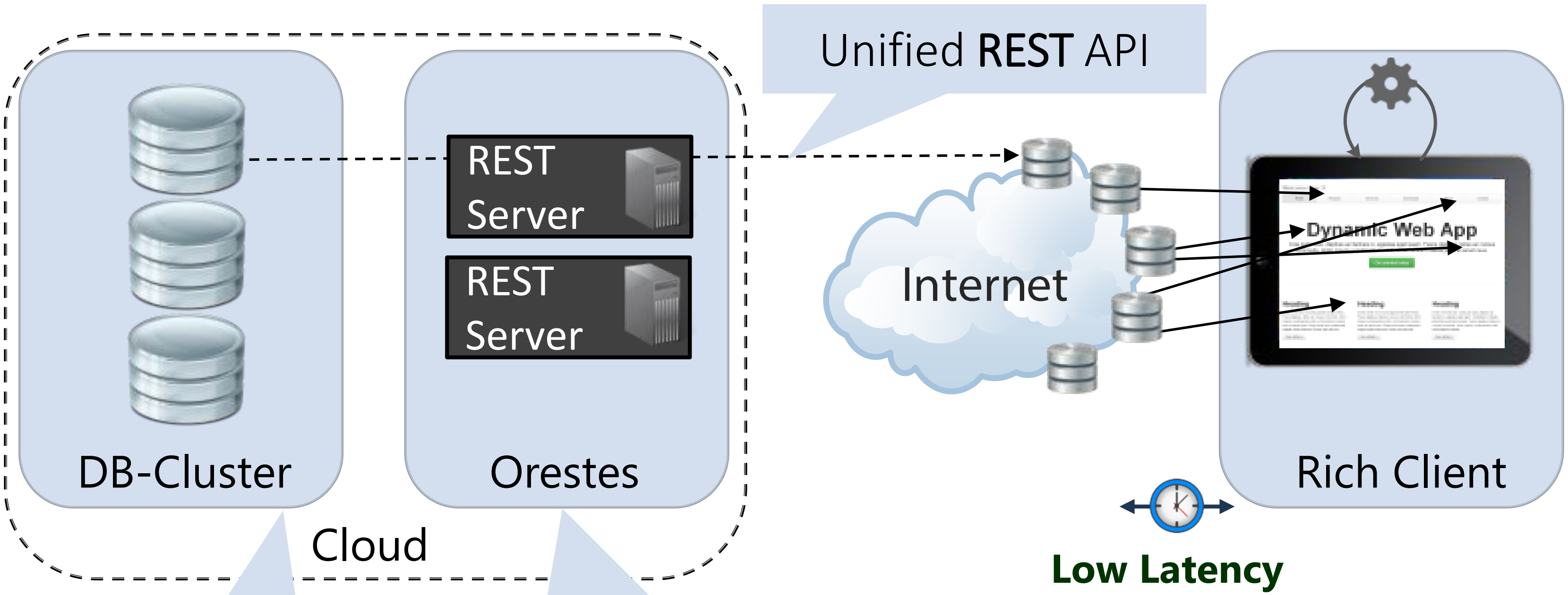
Vision: ORESTES



Vision: ORESTES



Vision: Orestes

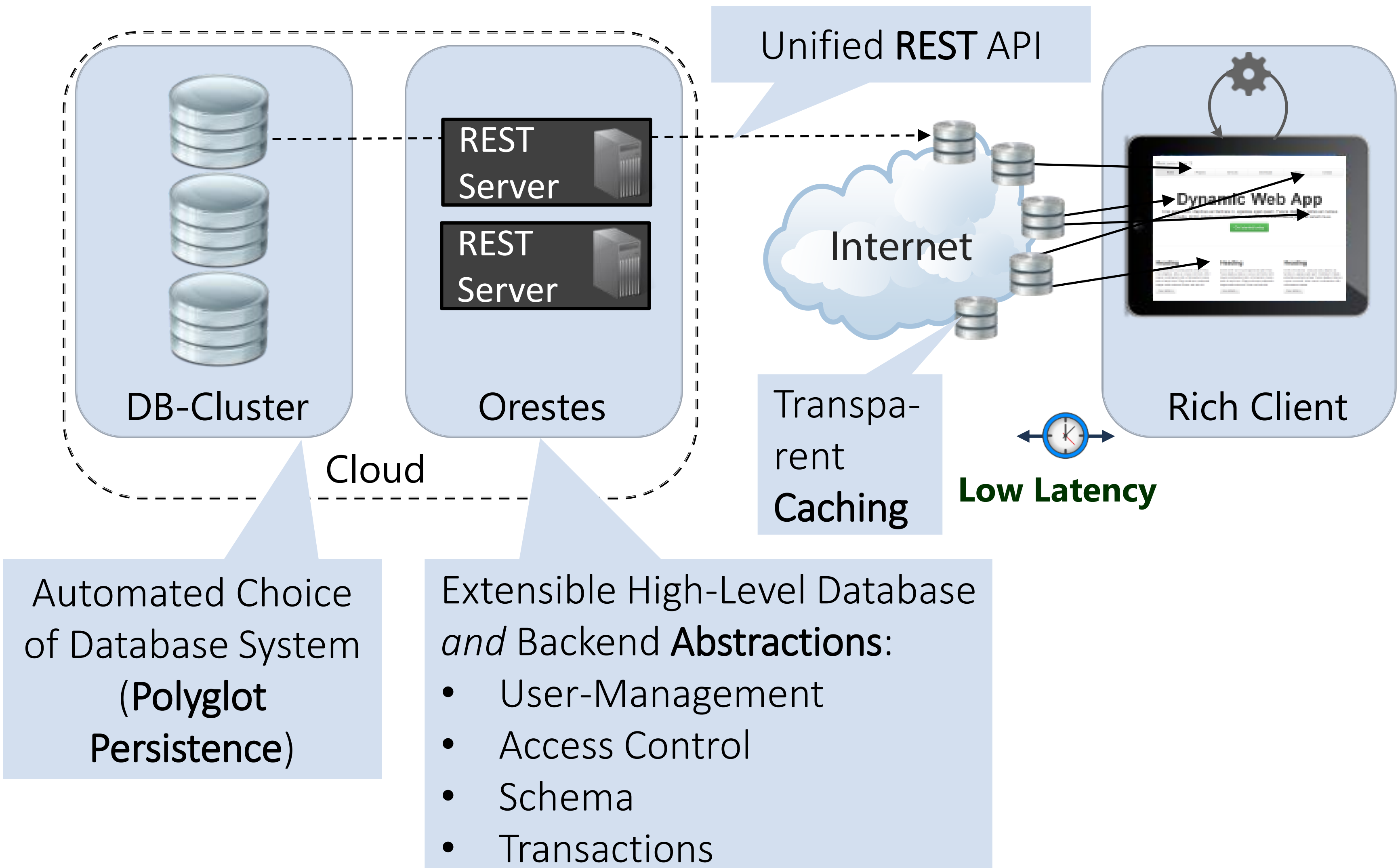


Automated Choice of Database System (**Polyglot Persistence**)

Extensible High-Level Database *and* Backend **Abstractions**:

- User-Management
- Access Control
- Schema
- Transactions

Vision: ORESTES



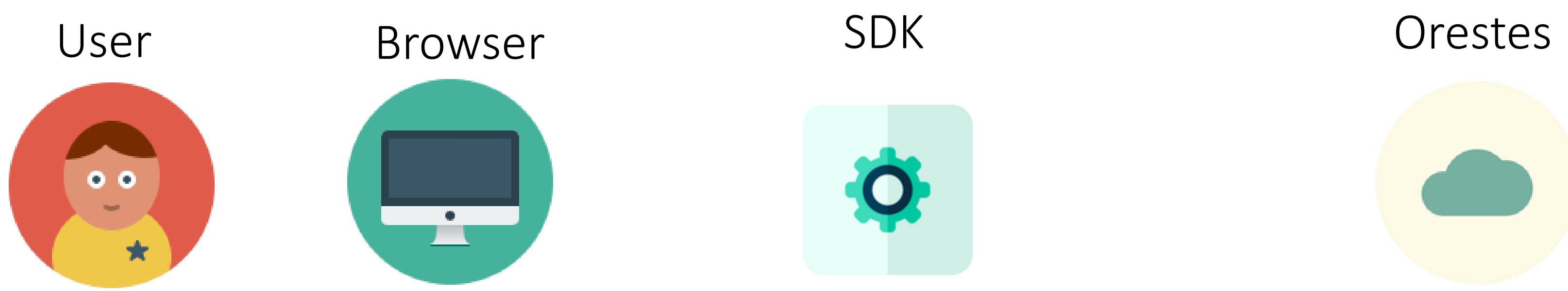
Outline

- Motivation
- **Cache Sketch Approach**
- Polyglot Persistence Mediator
- Wrap-up and future work

- REST/HTTP API
- HTTP Caching
- The Cache Sketch
  - Principle
  - Construction
  - Use

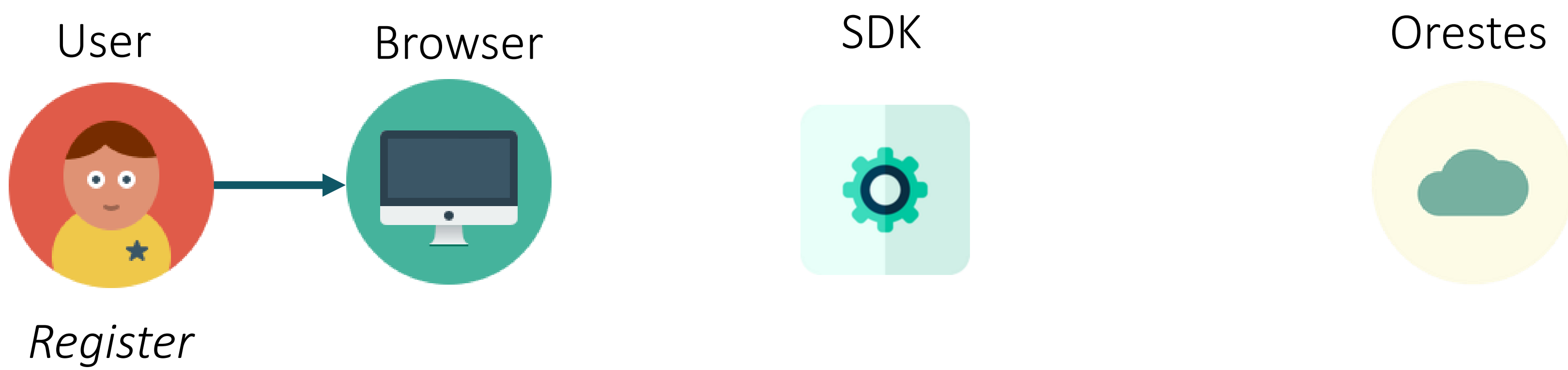
Unified REST API

- Platform-specific interfaces map to unified REST API



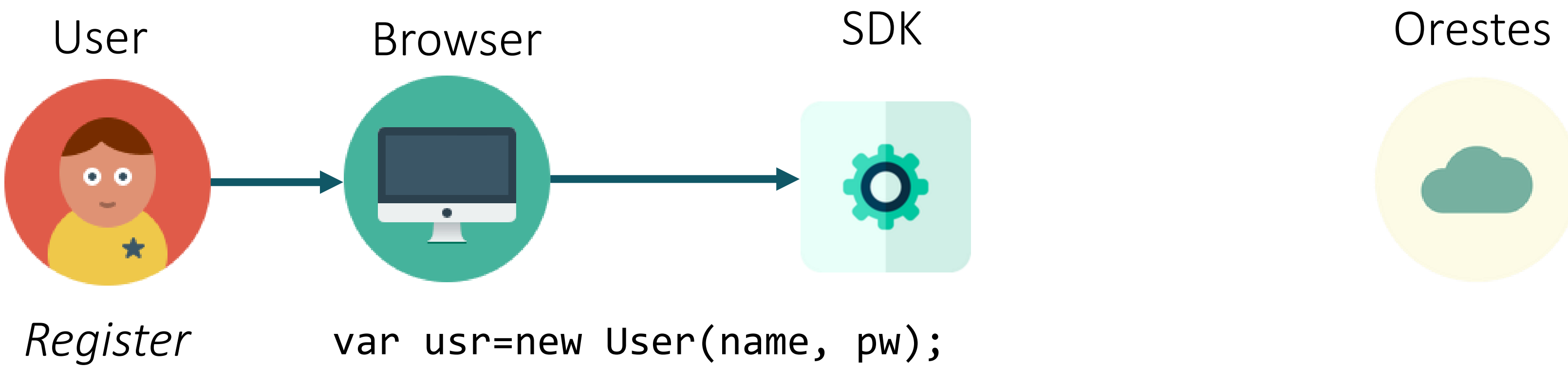
Unified REST API

- Platform-specific interfaces map to unified REST API



Unified REST API

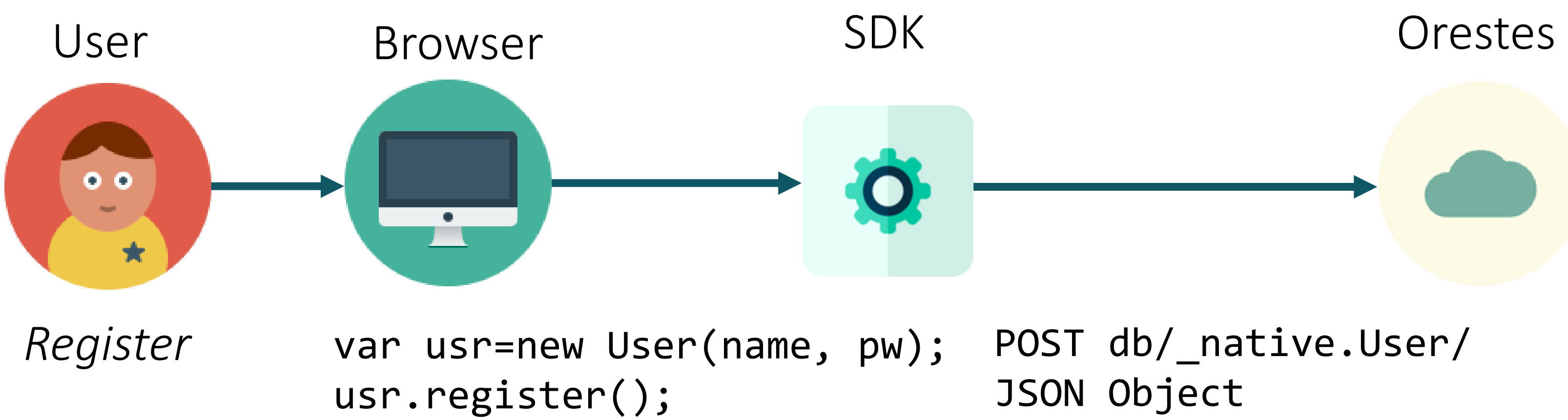
- Platform-specific interfaces map to unified REST API




```
var usr=new User(name, pw);
usr.register();
```

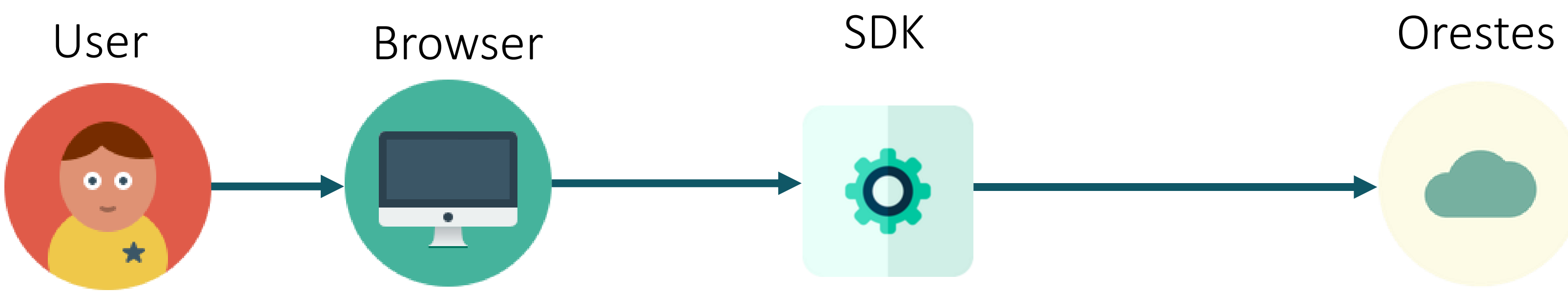
Unified REST API

- Platform-specific interfaces map to unified REST API



Unified REST API

- Platform-specific interfaces map to unified REST API



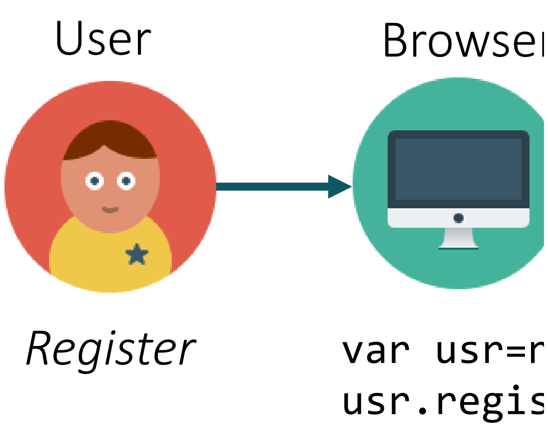
Register

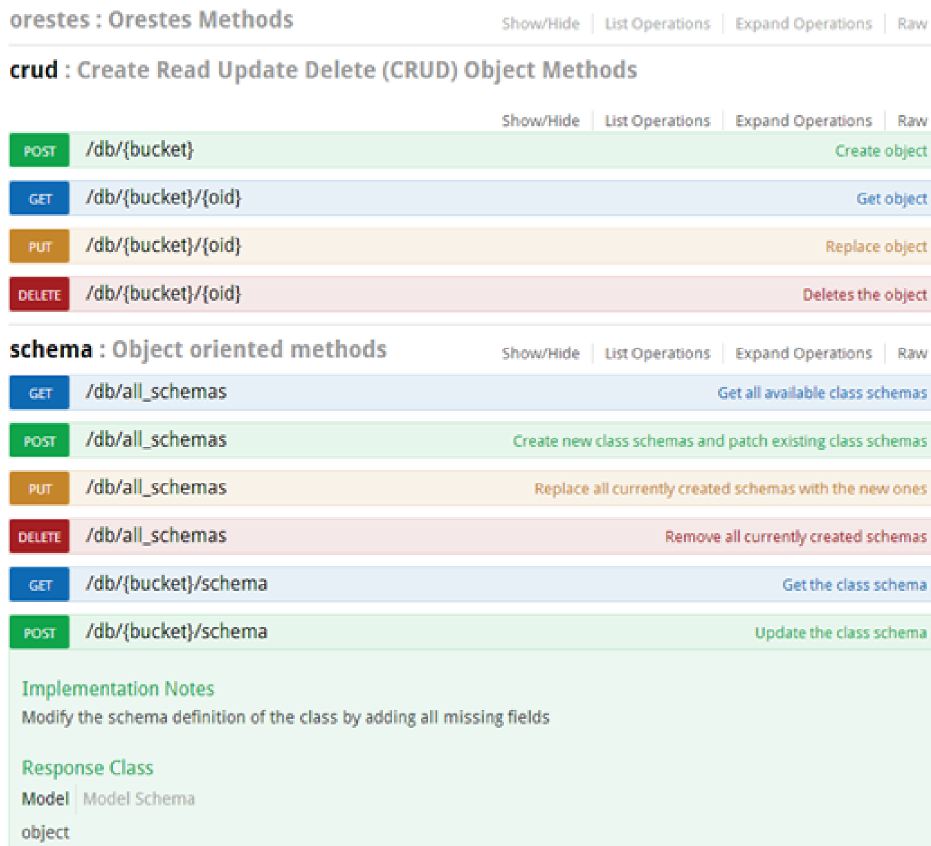
```
var usr=new User(name, pw);
usr.register();
```

```
POST db/_native.User/
JSON Object
```

Unified REST API

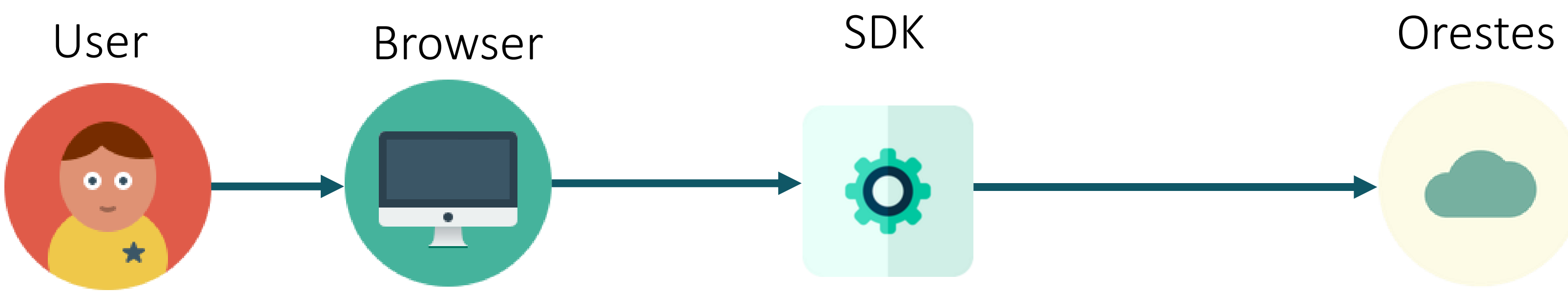
- Platform-specific ir





Unified REST API

- Platform-specific interfaces map to unified REST API



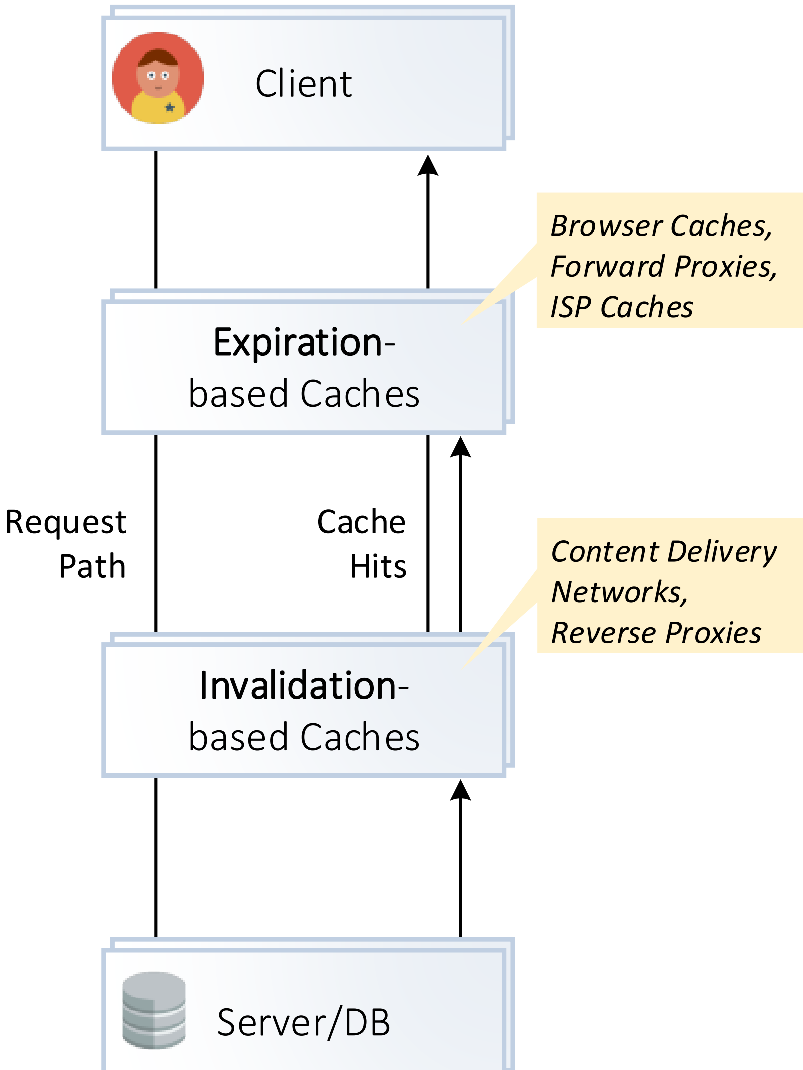
Register

```
var usr=new User(name, pw);
usr.register();
```

```
POST db/_native.User/
JSON Object
```

- REST API leverages existing HTTP infrastructure
  - Load-Balancer (*stateless communication REST constraint*)
  - Web-Caches (*caching REST constraint*)

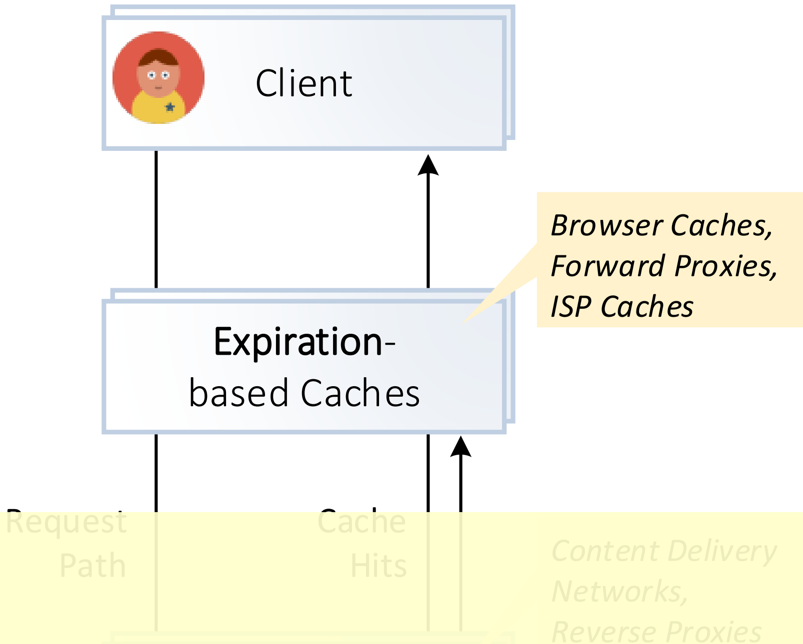
Expiration-based Caching



HTTP Caching Model:

- **Expiration-based** with defined *TTL*
- **Revalidations** check freshness at the server

Expiration-based Caching

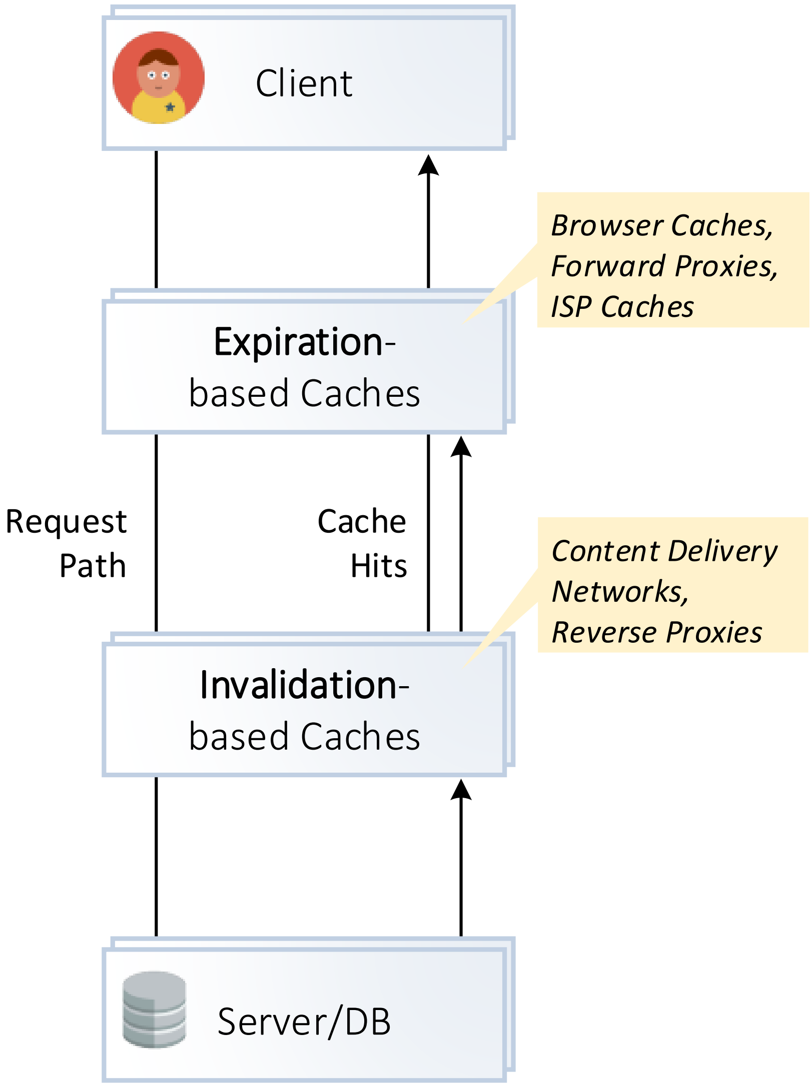


HTTP Caching Model:

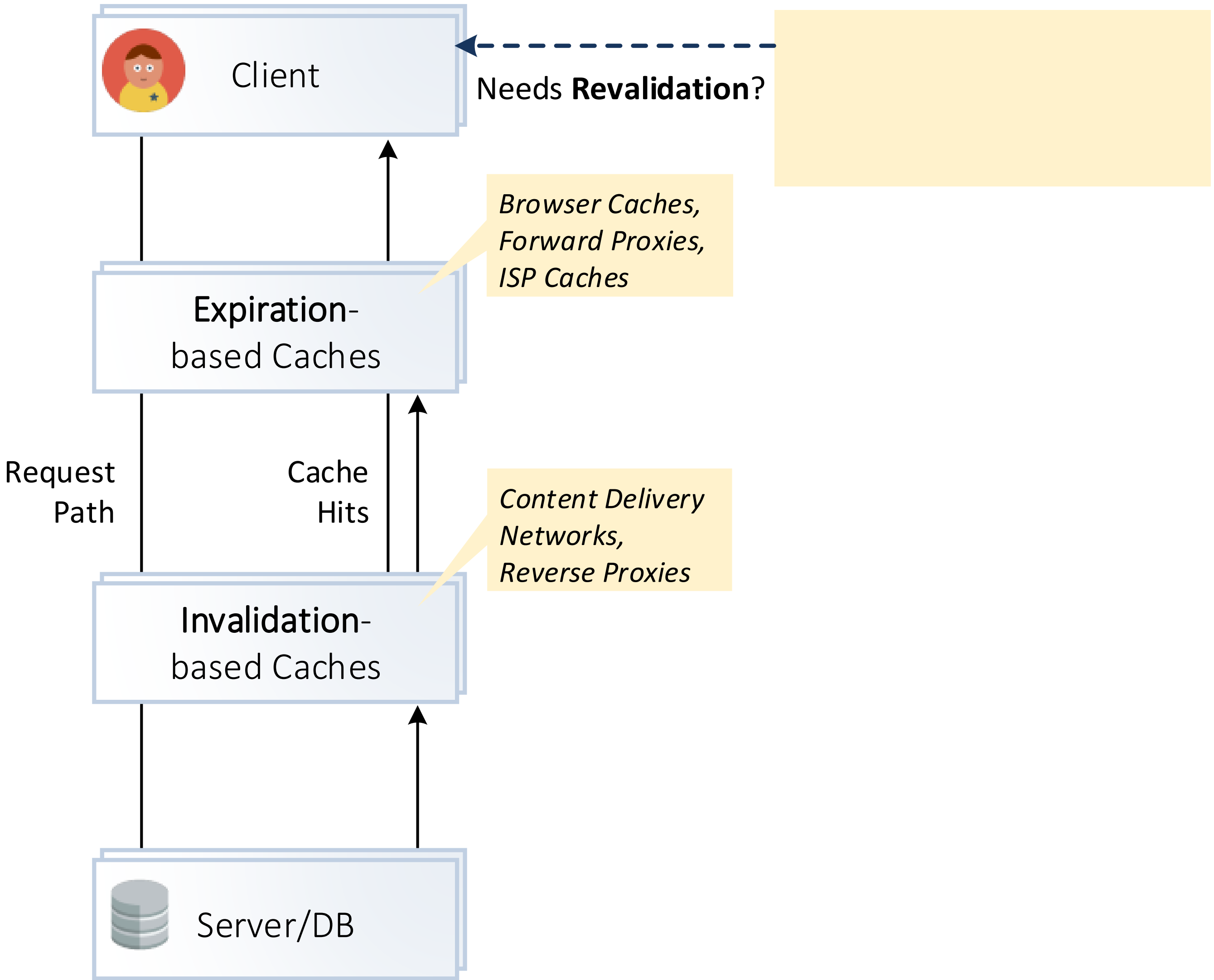
- **Expiration-based** with defined *TTL*
- **Revalidations** check freshness at the server

Research Question:
Can database services leverage the web caching infrastructure for low latency with rich consistency guarantees?

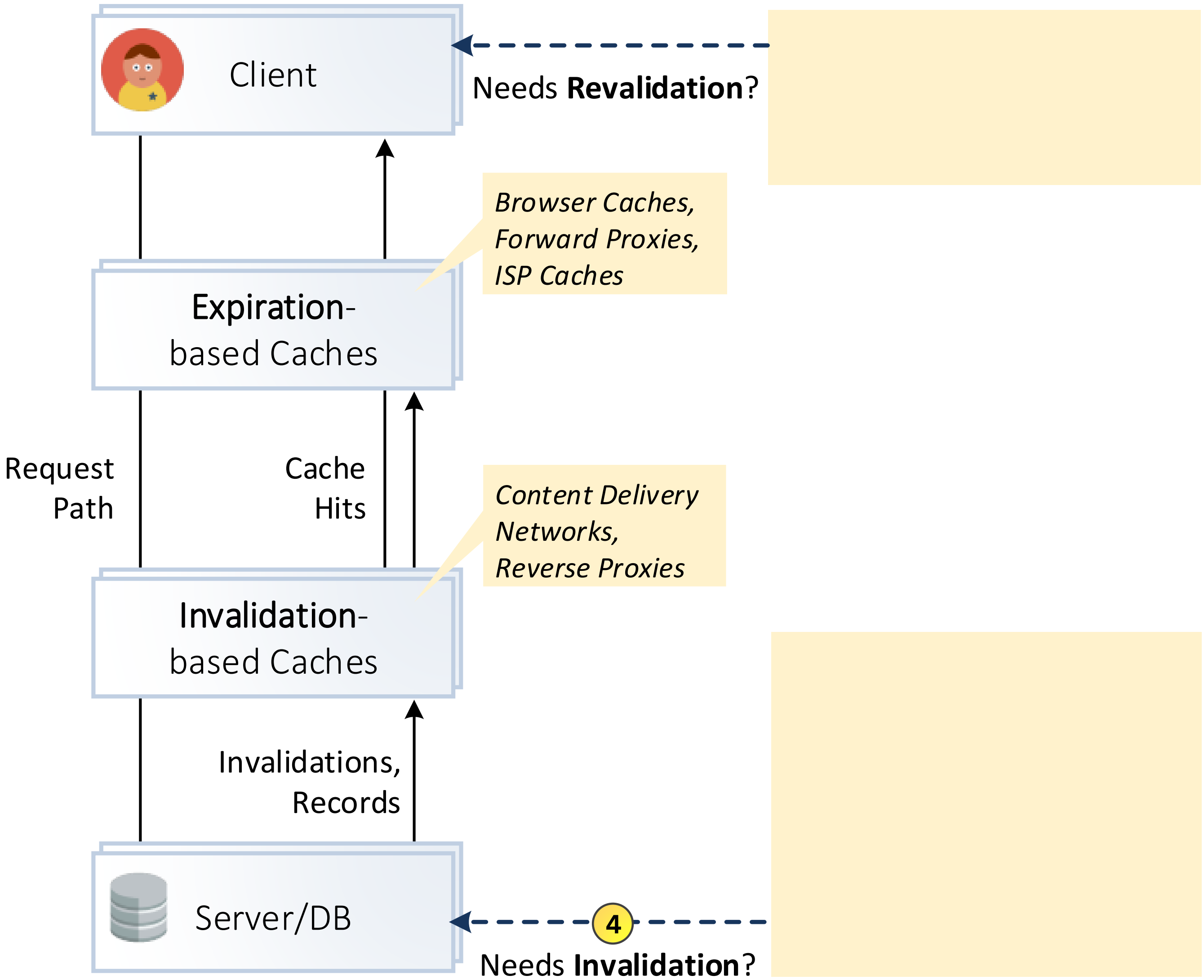
The Cache Sketch approach



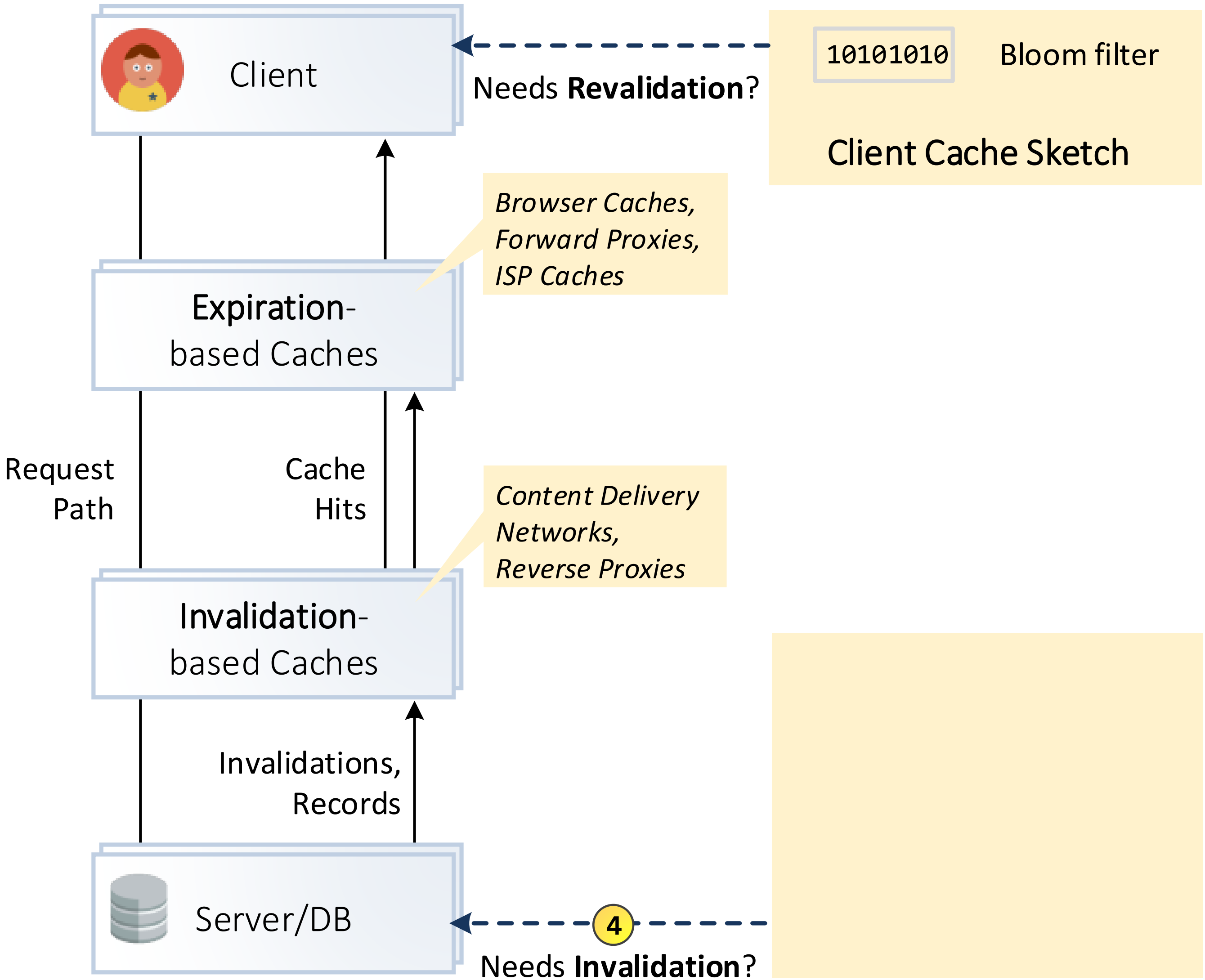
The Cache Sketch approach



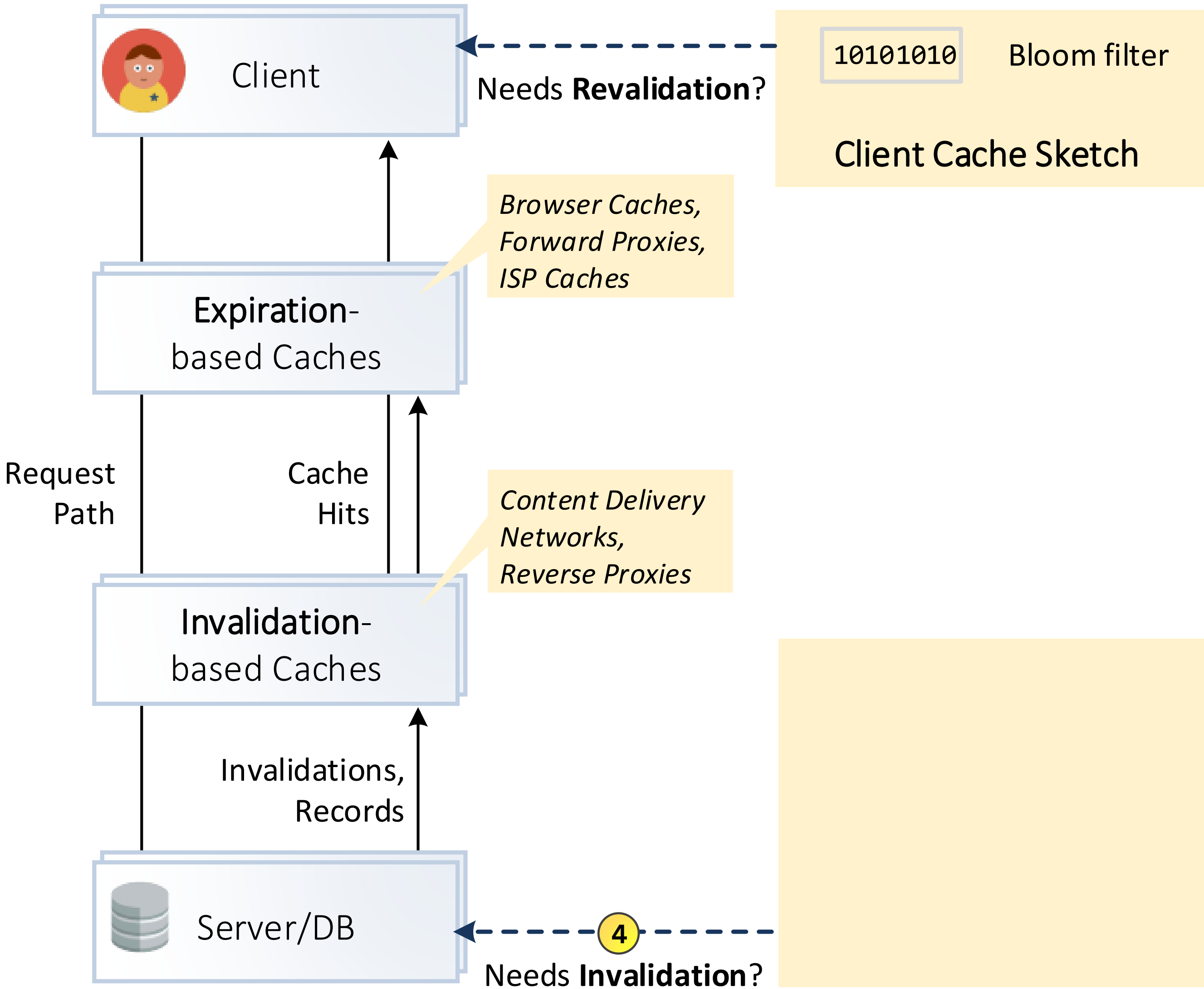
The Cache Sketch approach



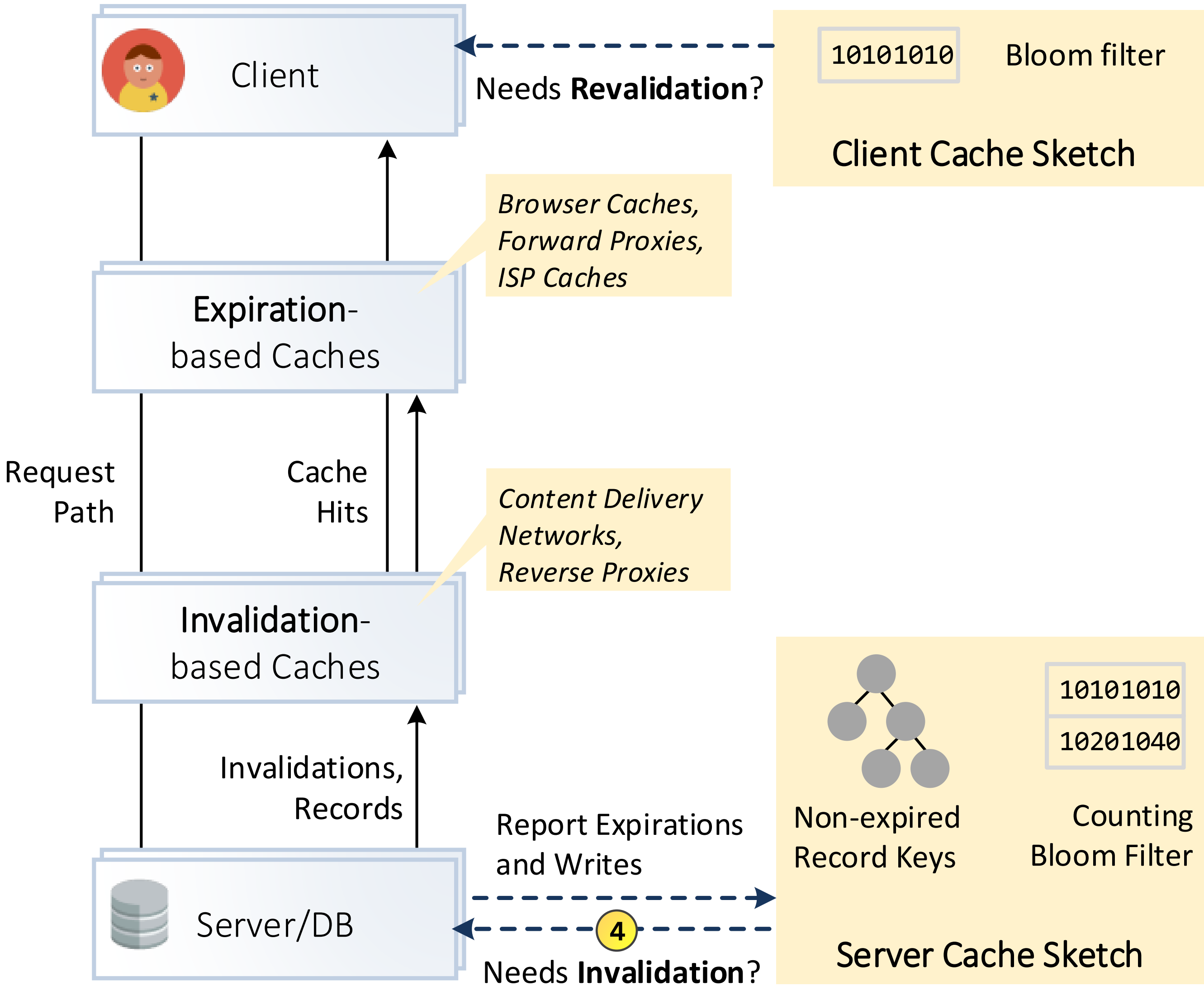
The Cache Sketch approach



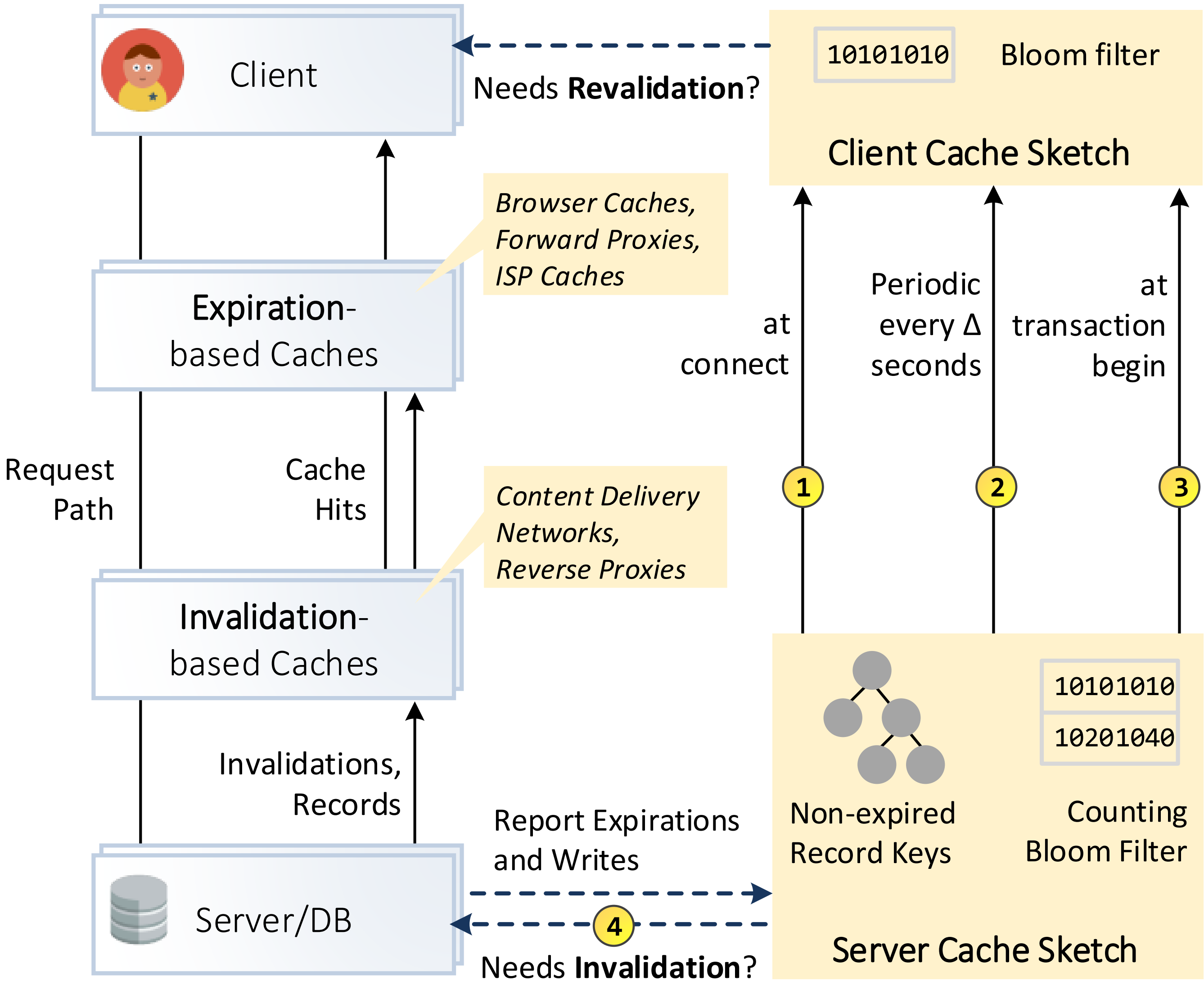
The Cache Sketch approach



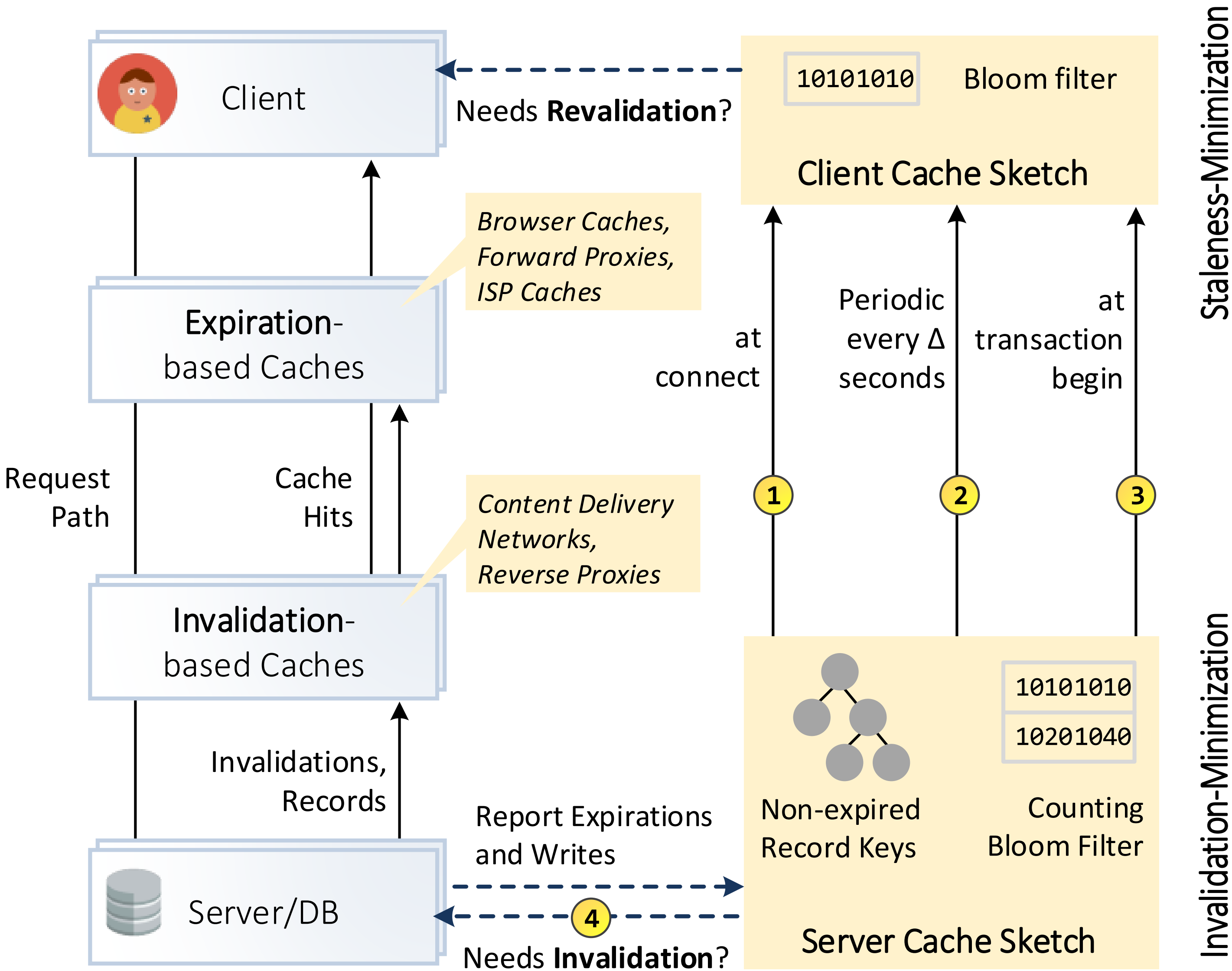
The Cache Sketch approach



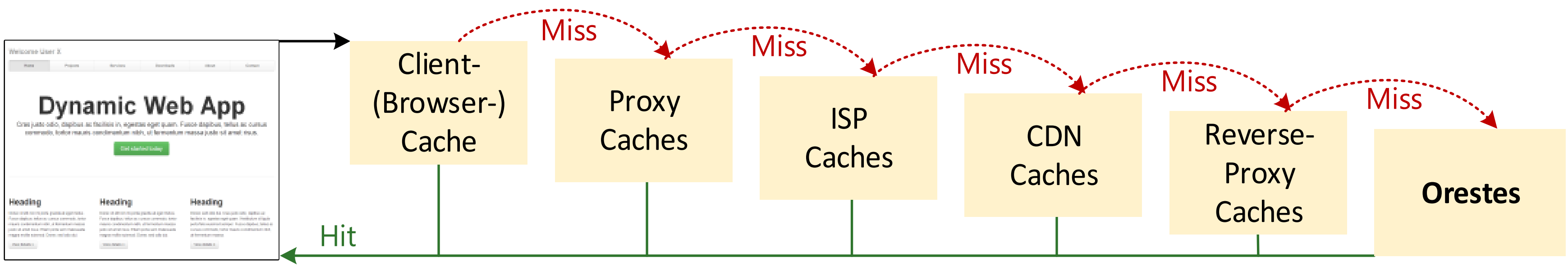
The Cache Sketch approach



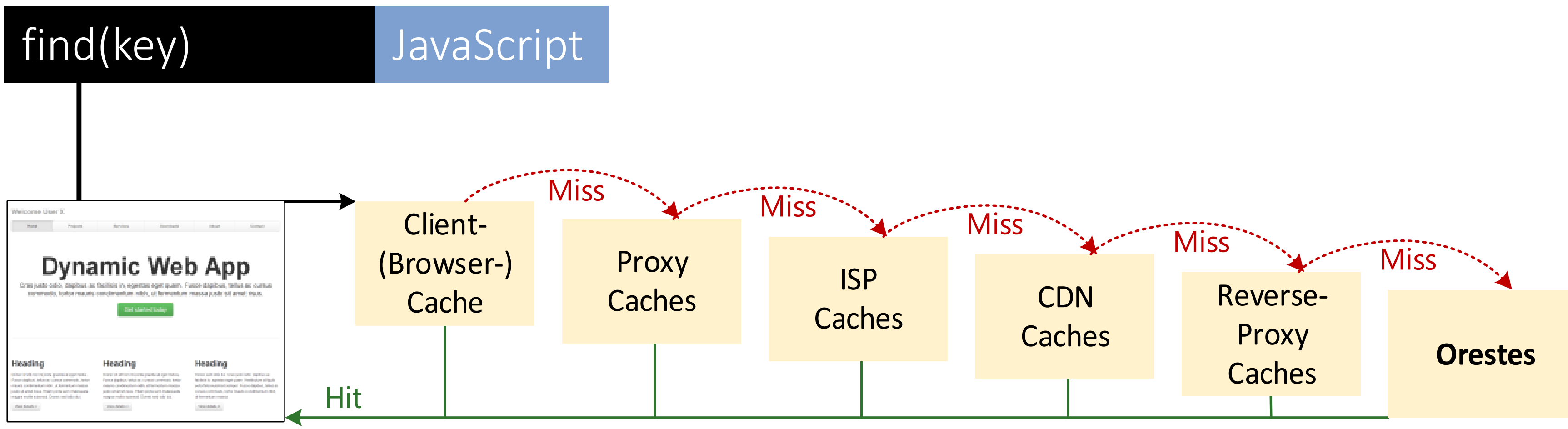
The Cache Sketch approach



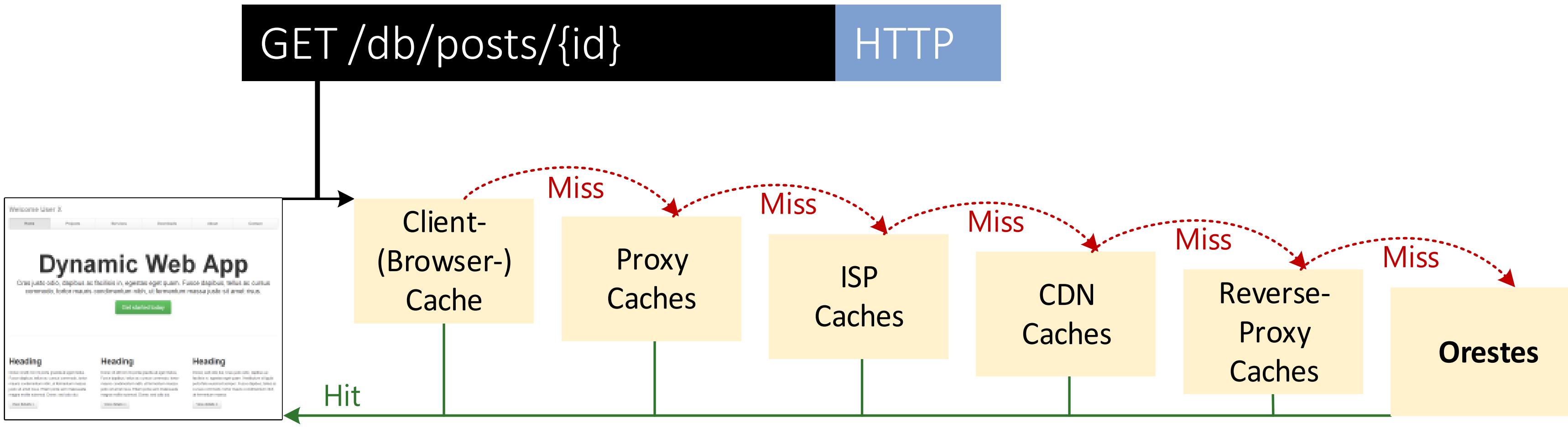
The Caching Hierarchy



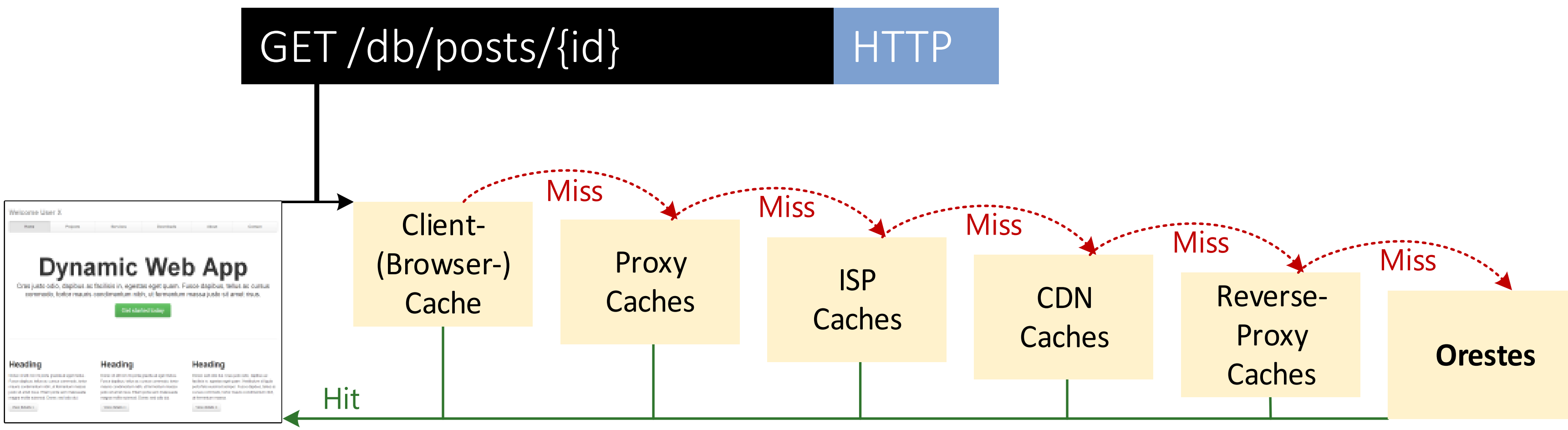
The Caching Hierarchy



The Caching Hierarchy

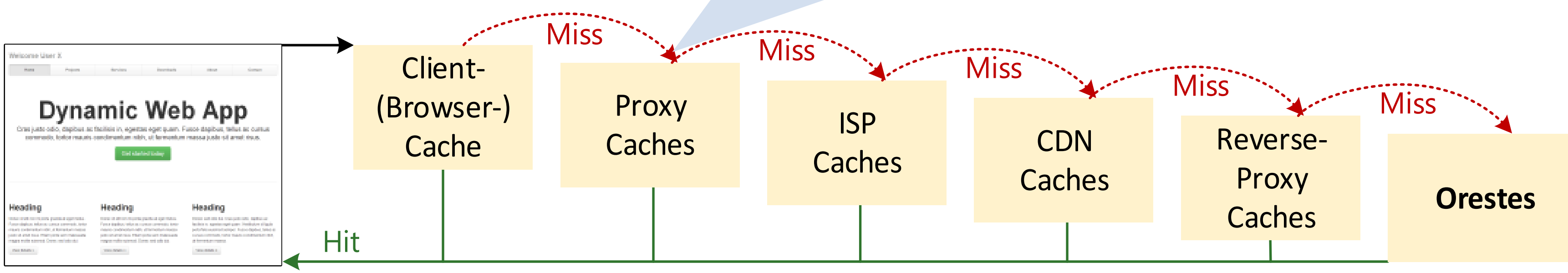


The Caching Hierarchy

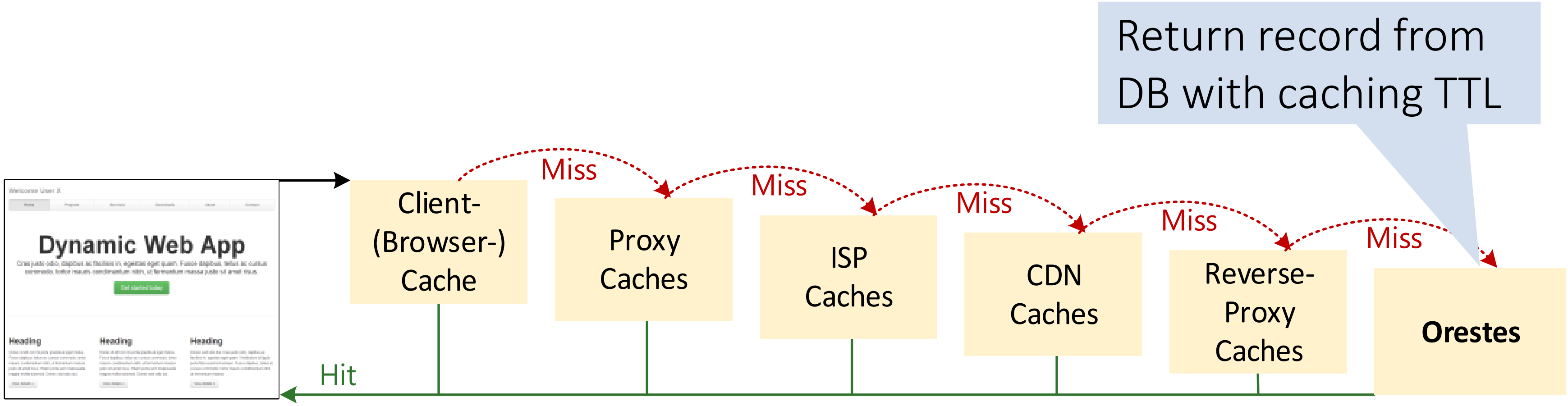


The Caching Hierarchy

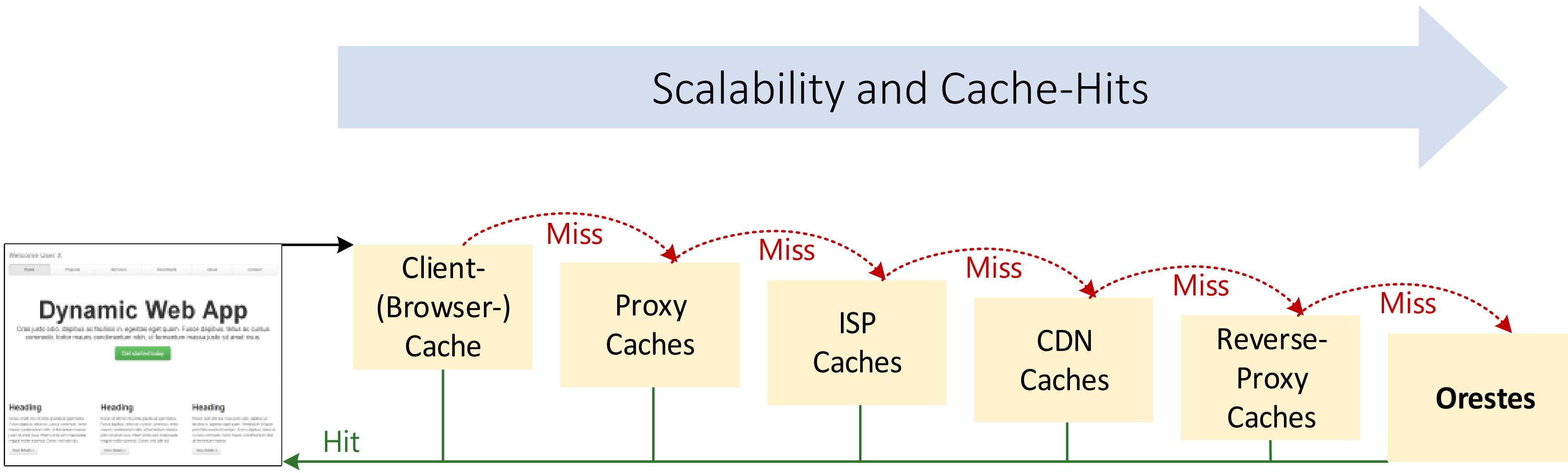
Cache-Hit: Return Object
Cache-Miss or Revalidation:
Forward Request



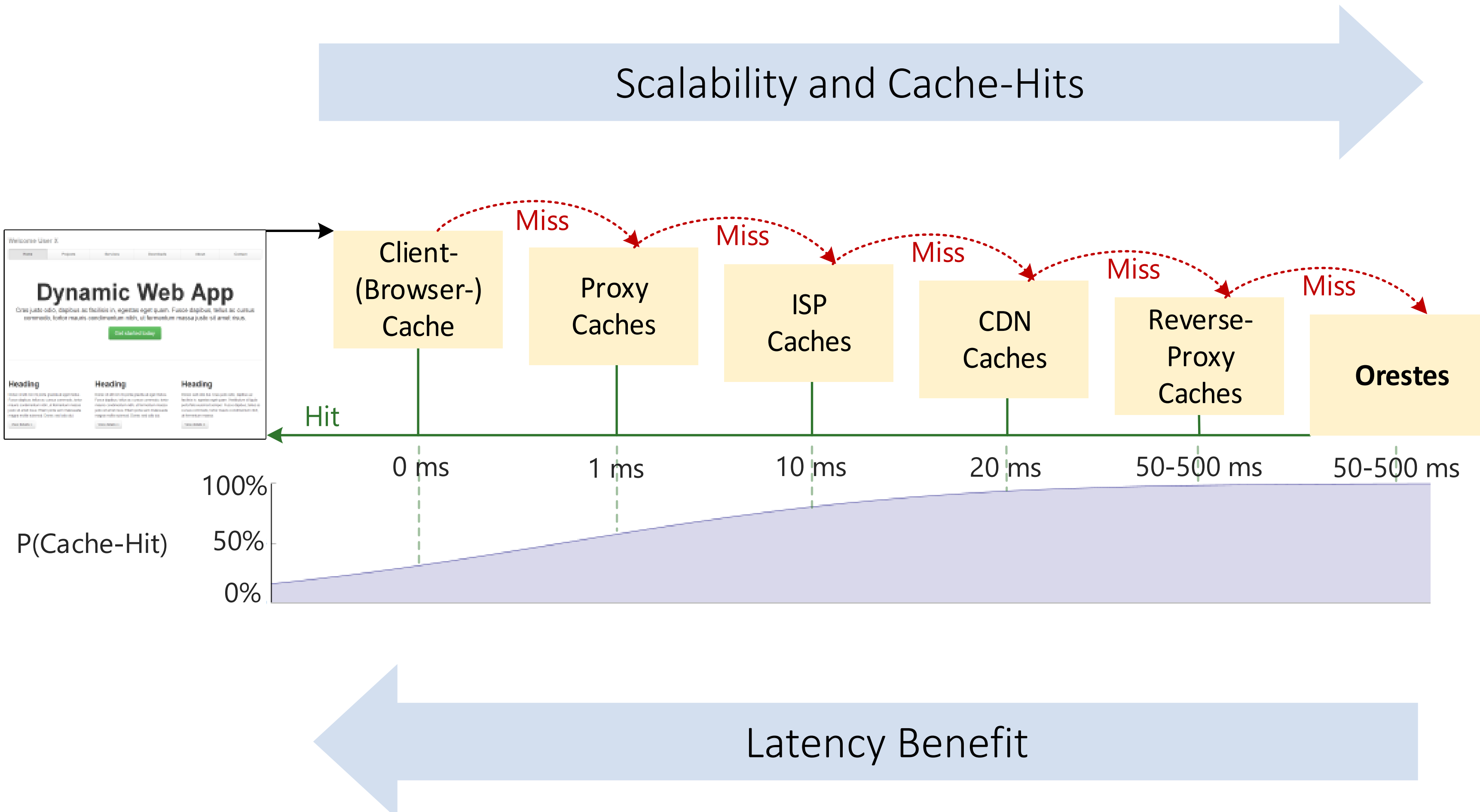
The Caching Hierarchy



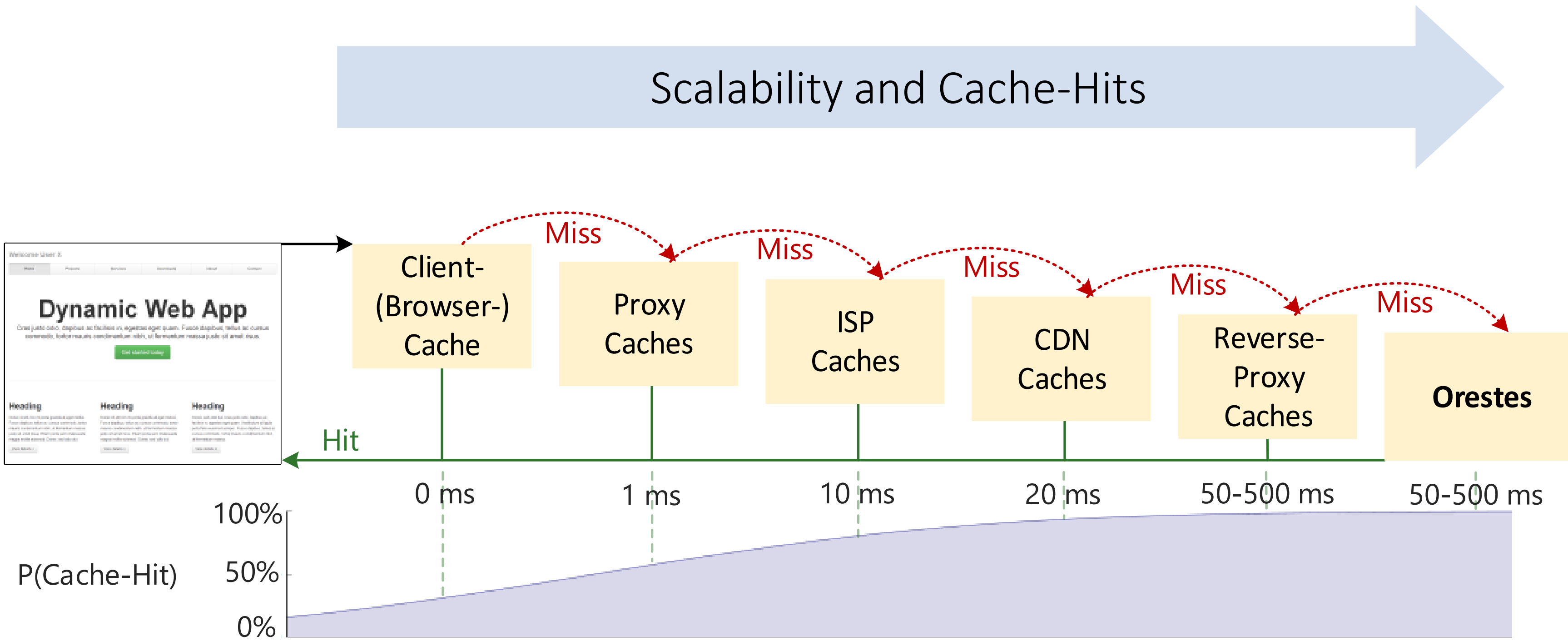
The Caching Hierarchy



The Caching Hierarchy



The Caching Hierarchy



Low Latency

Less Load on Database Service

Protection Against Flash Crowds

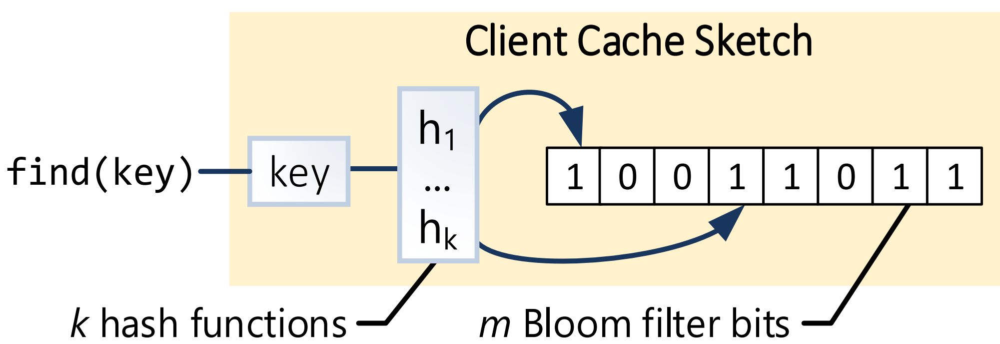
Better Availability

Clients Profit from Each Other

The Client Cache Sketch

- *Let* $\boldsymbol{c_t}$ *be the client Cache Sketch generated at time* $\boldsymbol{t}$*, containing the key* $\boldsymbol{key_x}$ *of every record* $\boldsymbol{x}$ *that was <u>written before it expired</u> in all caches, i.e. every* $\boldsymbol{x}$ *for which holds:*

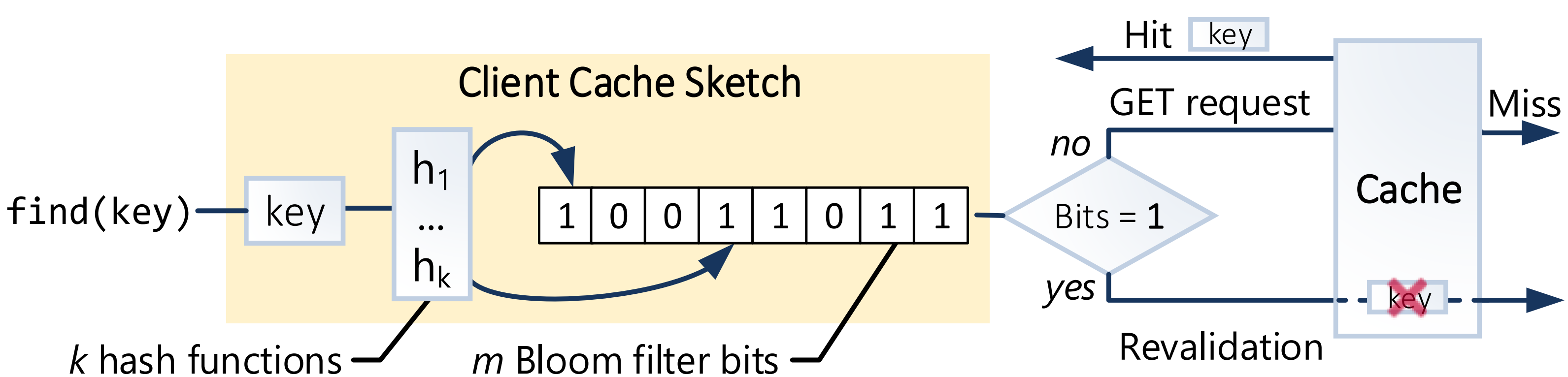
$$\exists\, r(x, t_r, TTL), w(x, t_w) : \; t_r + TTL > t > t_w > t_r$$



The Client Cache Sketch

- *Let **c_t** be the client Cache Sketch generated at time **t**, containing the key **key_x** of every record **x** that was written before it expired in all caches, i.e. every **x** for which holds:*

$$\exists\, r(x, t_r, TTL), w(x, t_w) : t_r + TTL > t > t_w > t_r$$

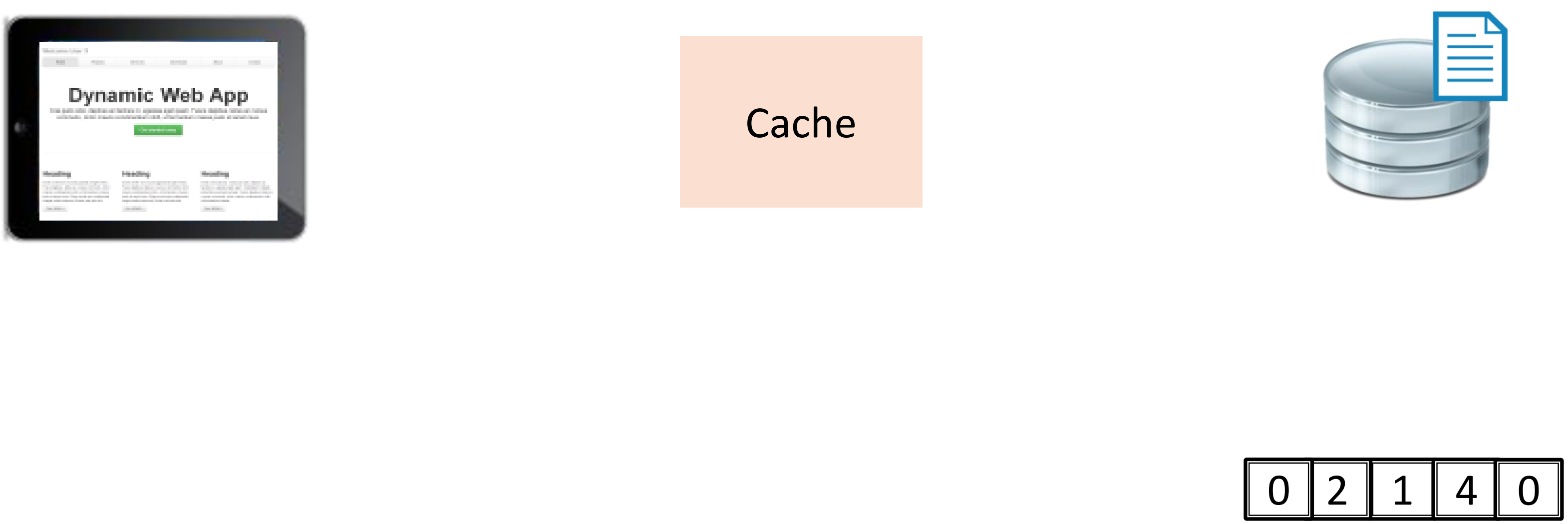


Problem I: Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency

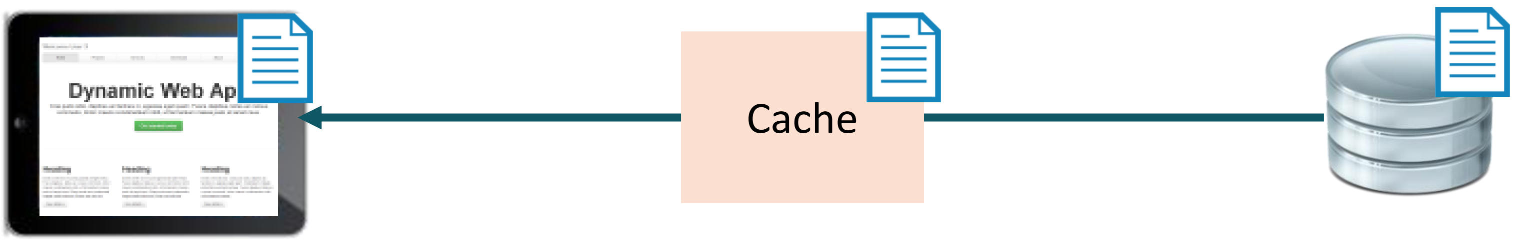
Problem I: Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



Problem I: Slow initial page loads

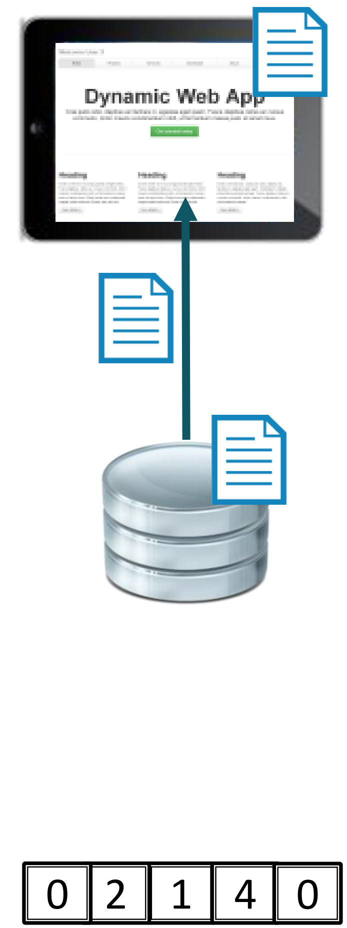
- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



0	2	1	4	0

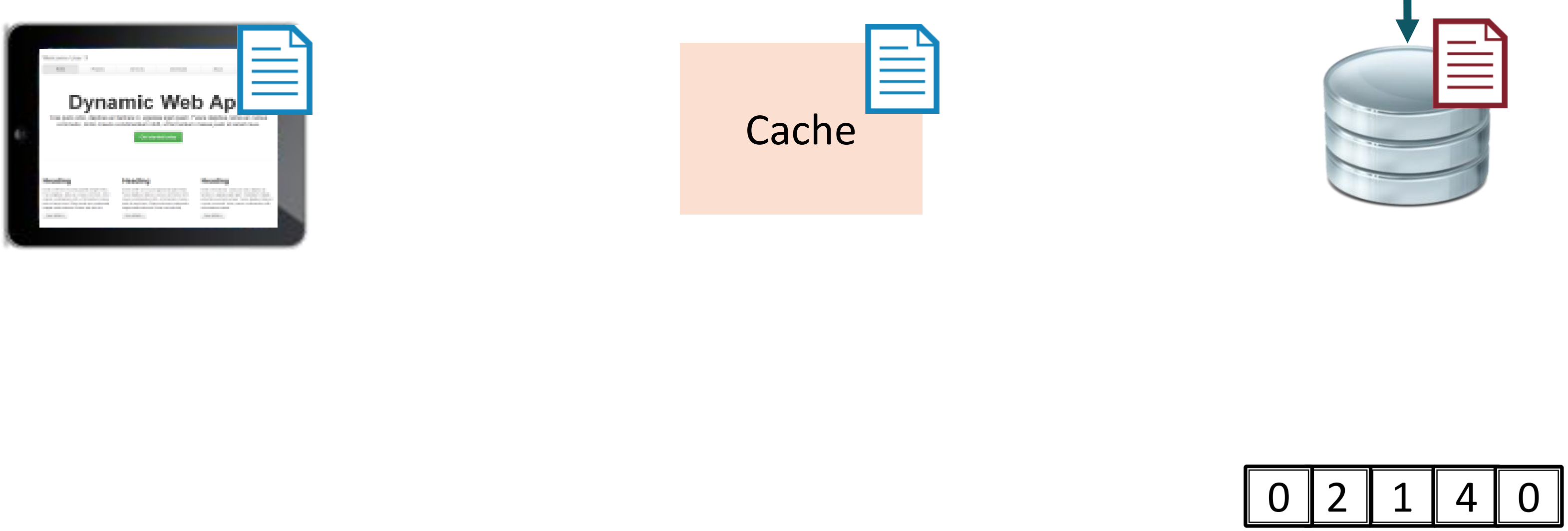
Problem I: Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



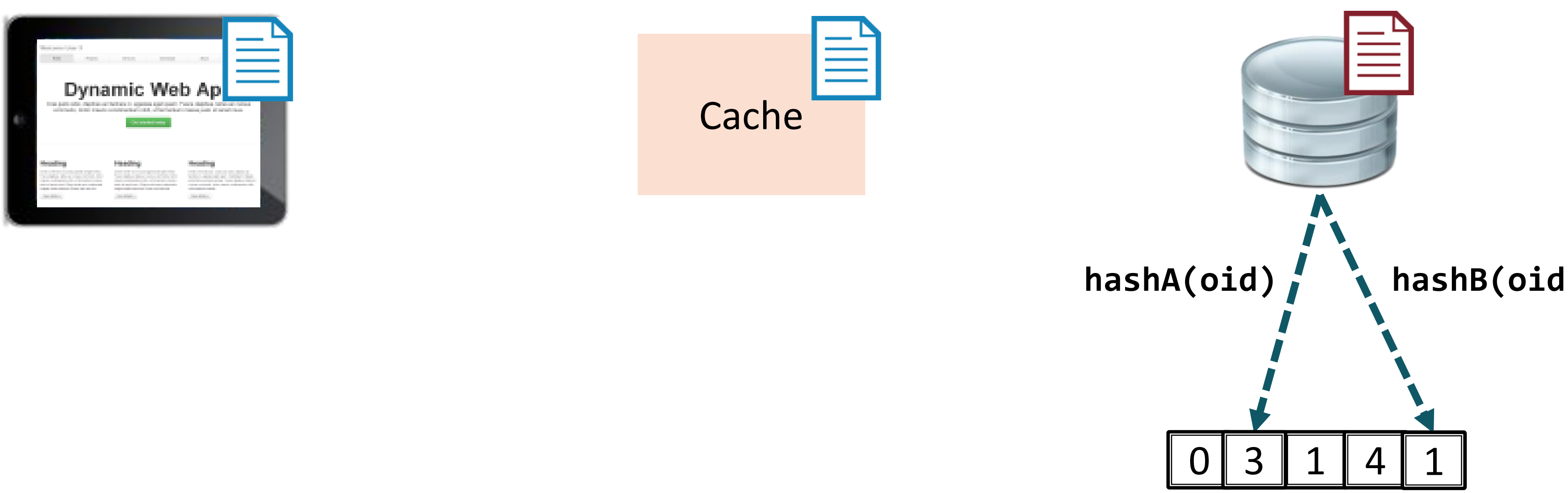
Problem I: Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



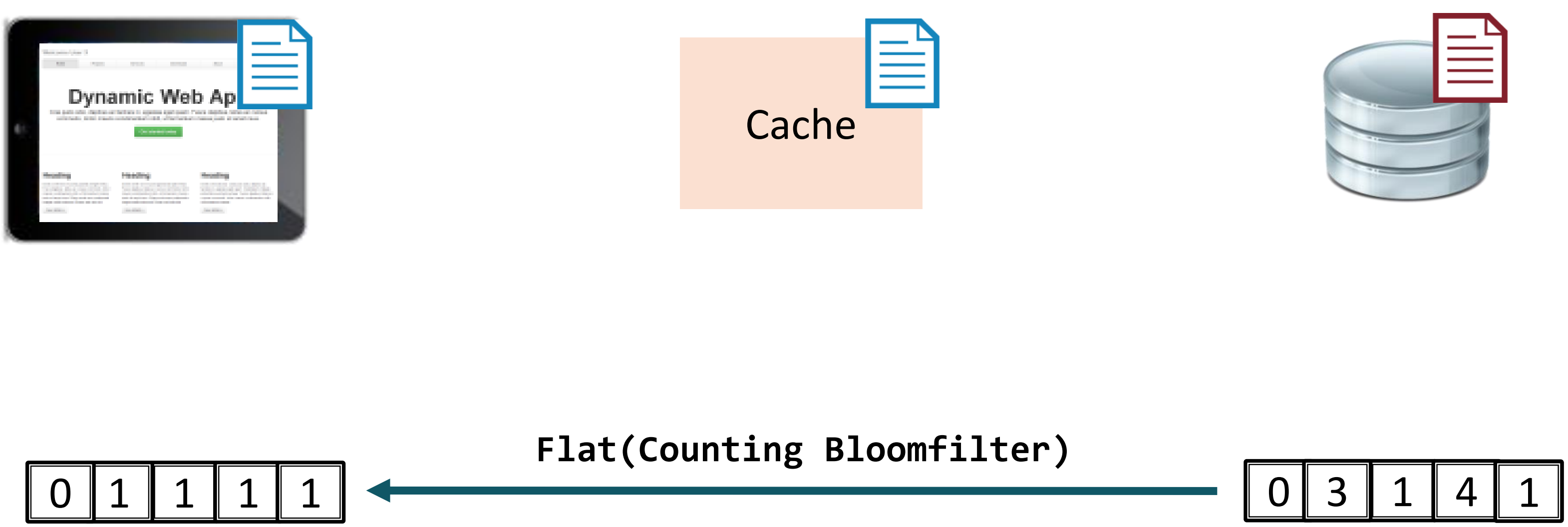
Problem I: Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



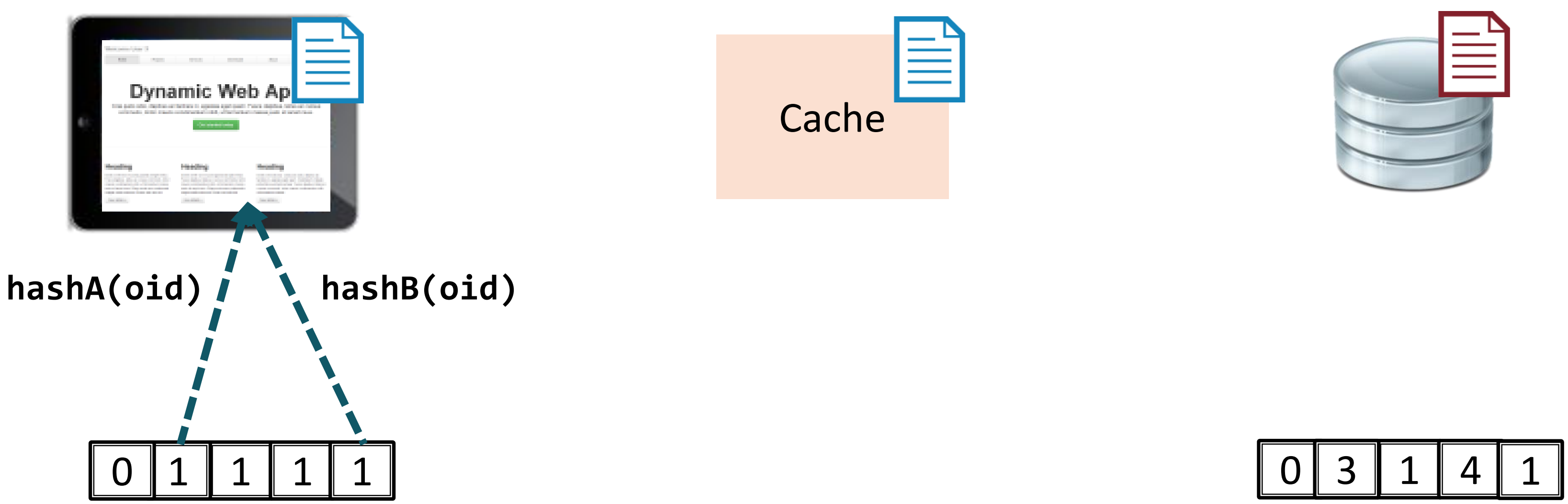
Problem I: Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



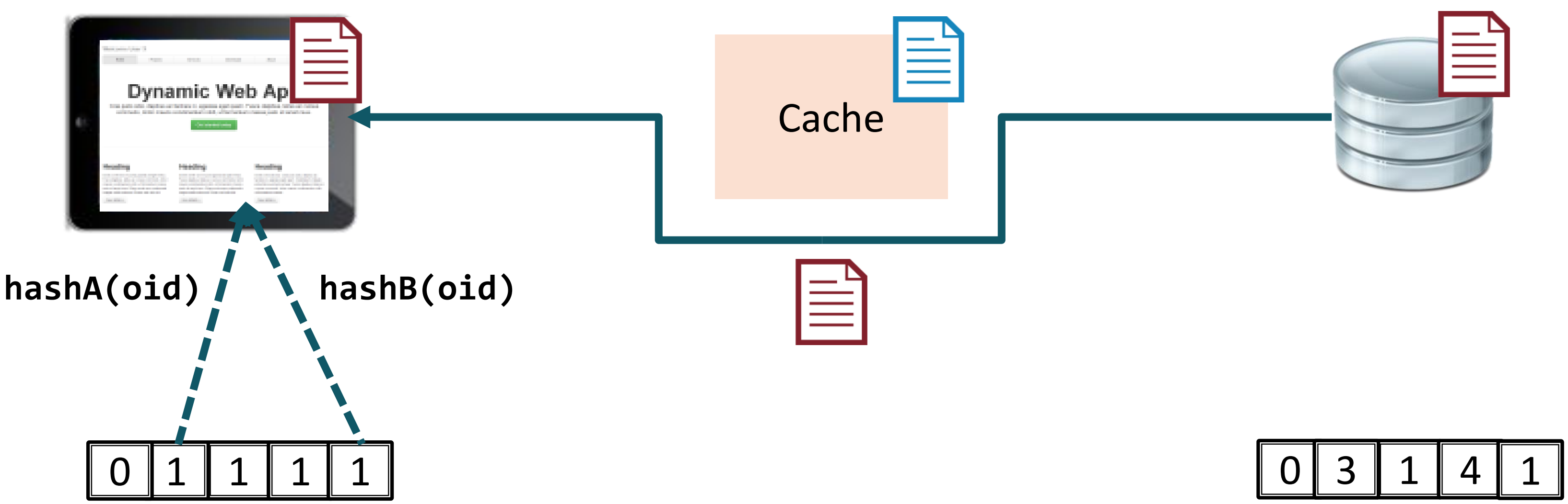
Problem I: Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



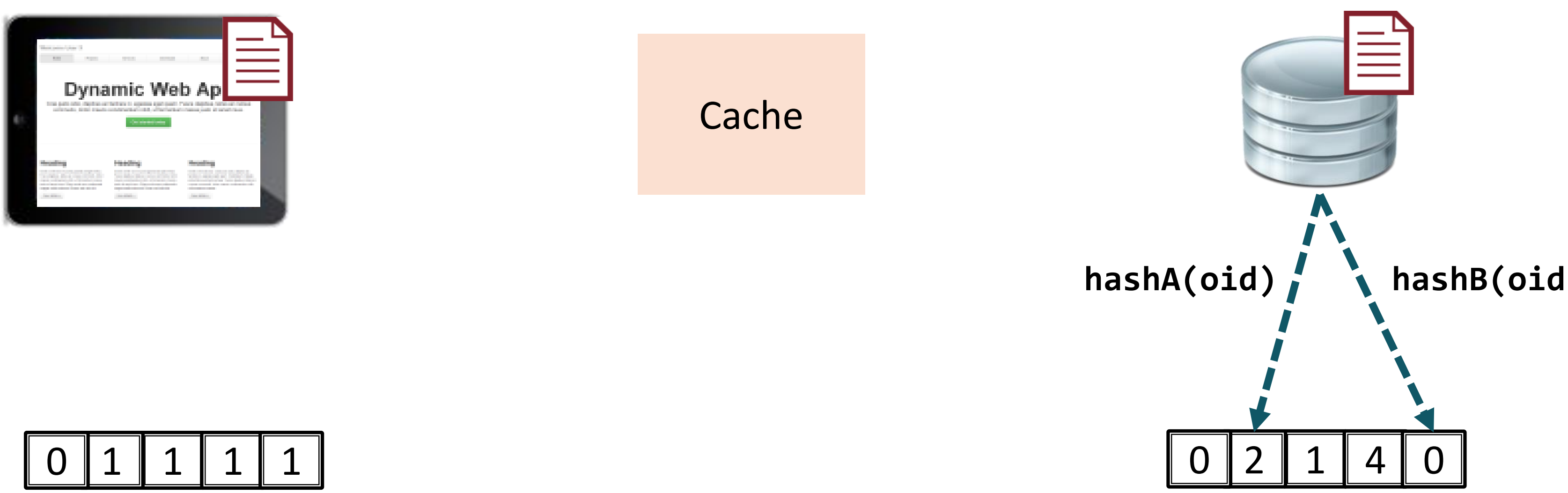
Problem I: Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



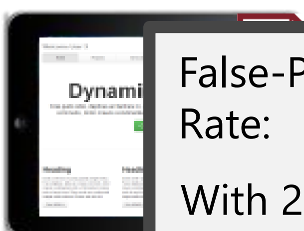
Problem I: Slow initial page loads

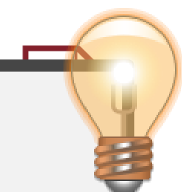
- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



Problem I: Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency

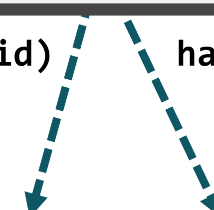




False-Positive Rate: $f \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$

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

With 20.000 distinct updates and 5% error rate: **14 KByte**

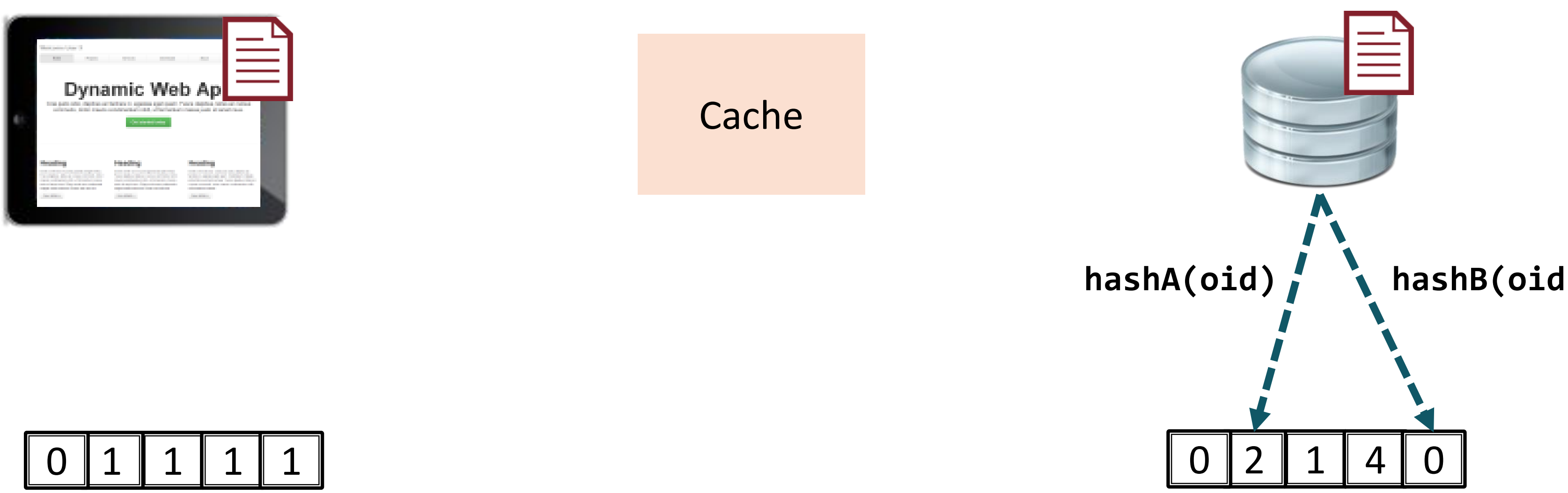


0	1	1	1	1

0	2	1	4	0

Problem I: Slow initial page loads

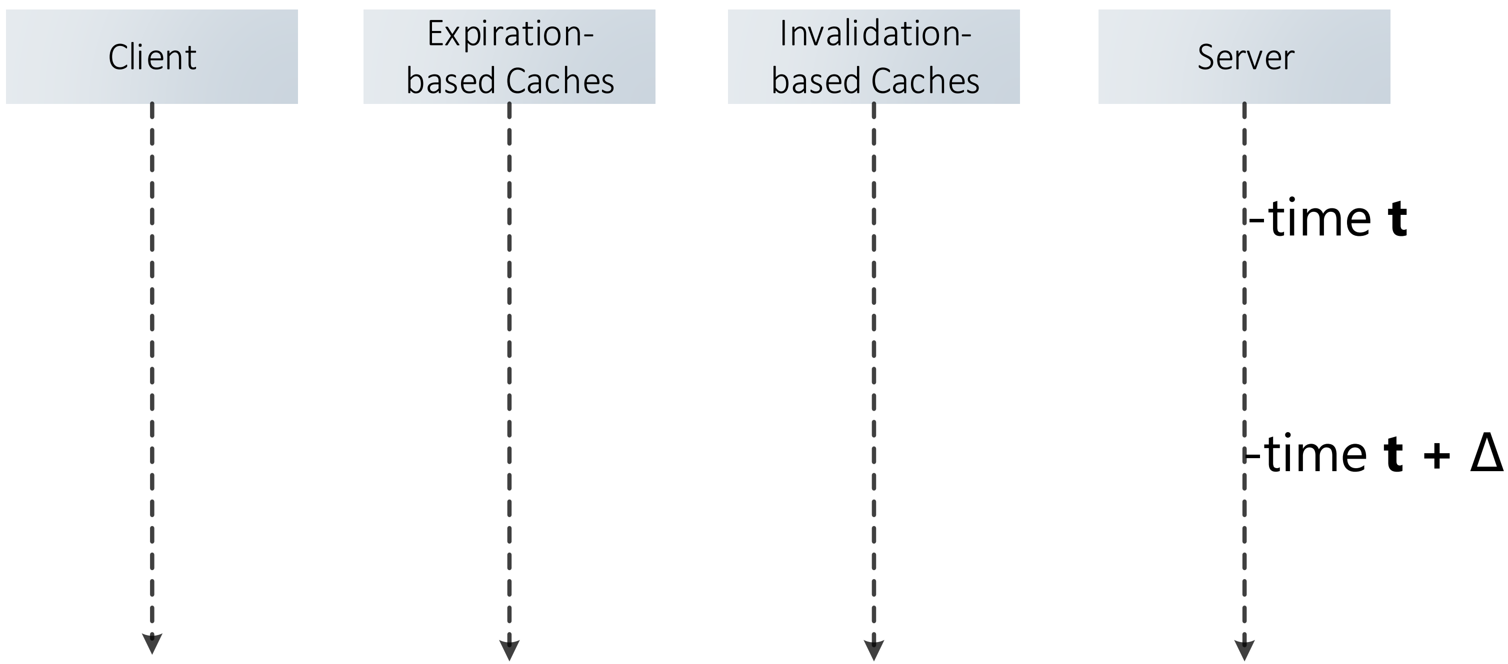
- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency



Problem II: Slow CRUD performance

- Solution: **Δ-Bounded Staleness**
  - Clients refresh the Cache Sketch so its age never exceeds Δ

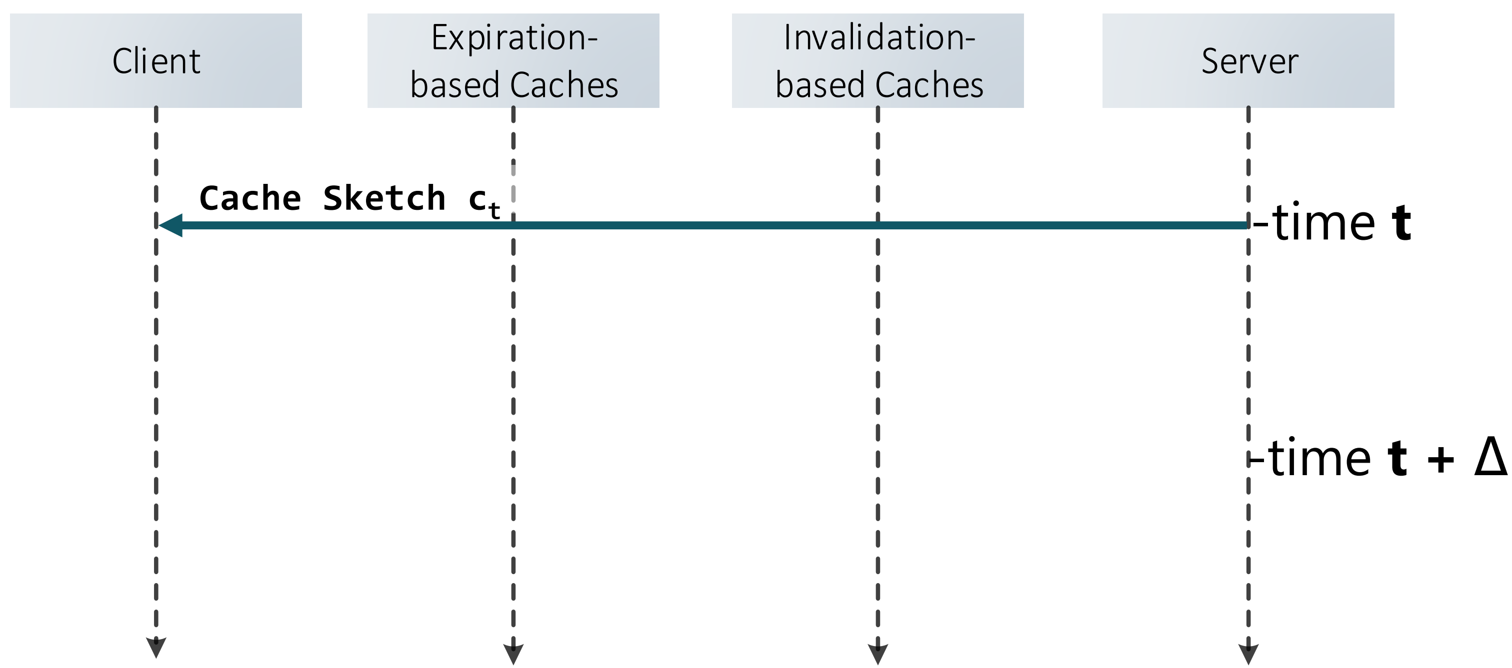
  → *Consistency guarantee*: Δ-atomicity



Problem II: Slow CRUD performance

- Solution: **Δ-Bounded Staleness**
  - Clients refresh the Cache Sketch so its age never exceeds Δ

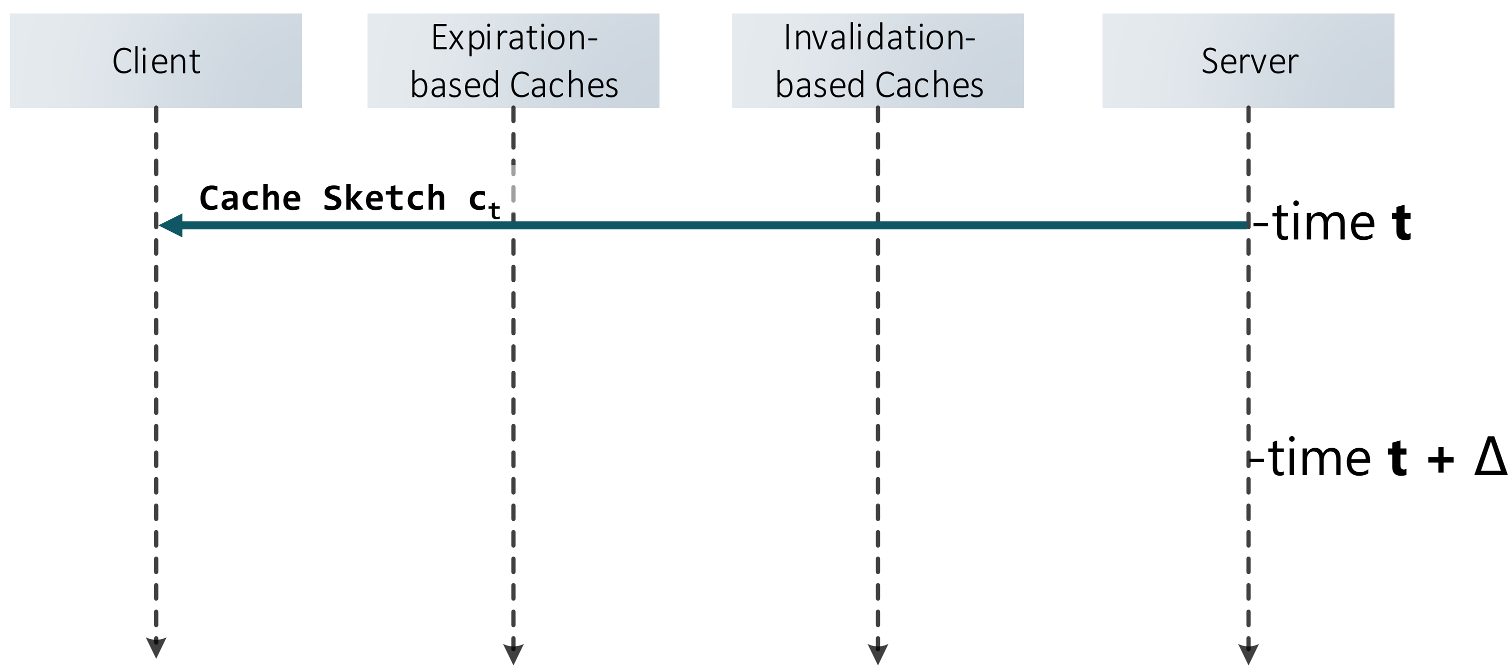
→ *Consistency guarantee*: Δ-atomicity



Problem II: Slow CRUD performance

- Solution: **Δ-Bounded Staleness**
  - Clients refresh the Cache Sketch so its age never exceeds Δ

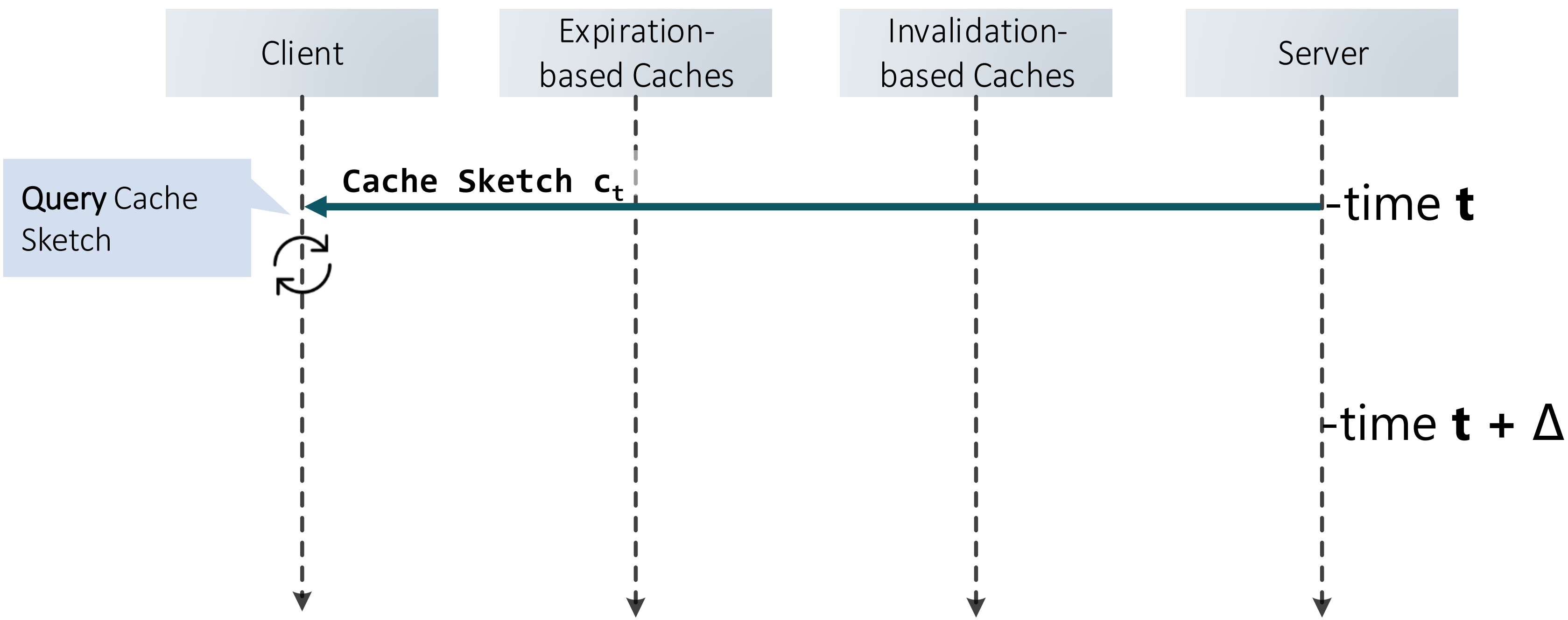
  → *Consistency guarantee*: Δ-atomicity



Problem II: Slow CRUD performance

- Solution: **Δ-Bounded Staleness**
  - Clients refresh the Cache Sketch so its age never exceeds Δ

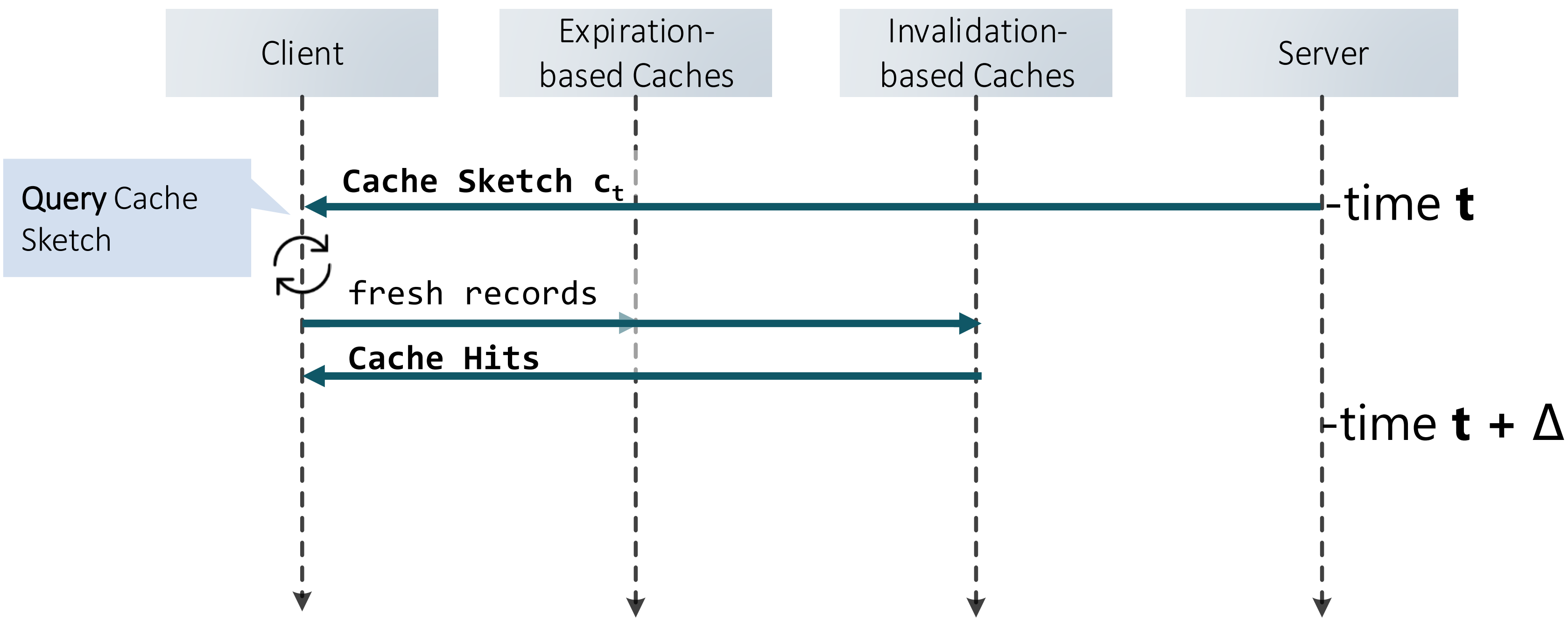
  → *Consistency guarantee*: Δ-atomicity



Problem II: Slow CRUD performance

- Solution: **Δ-Bounded Staleness**
  - Clients refresh the Cache Sketch so its age never exceeds Δ

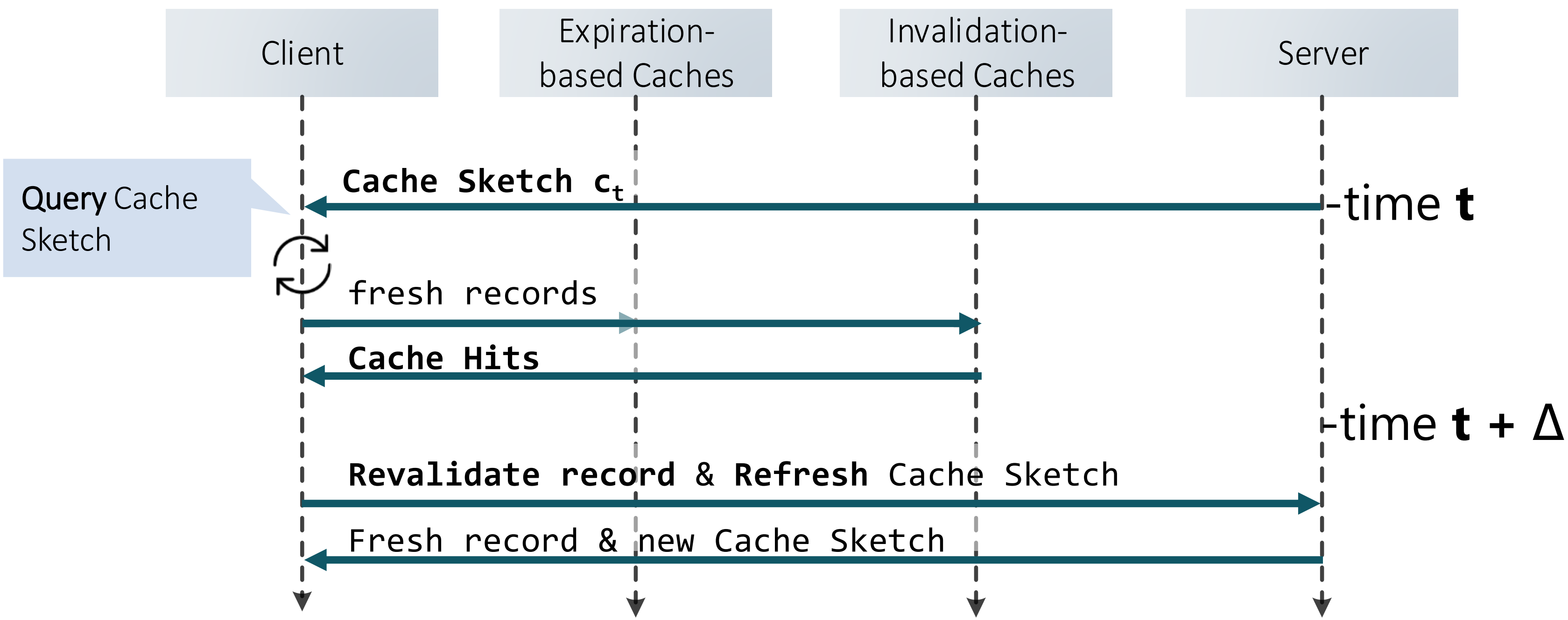
→ *Consistency guarantee*: Δ-atomicity



Problem II: Slow CRUD performance

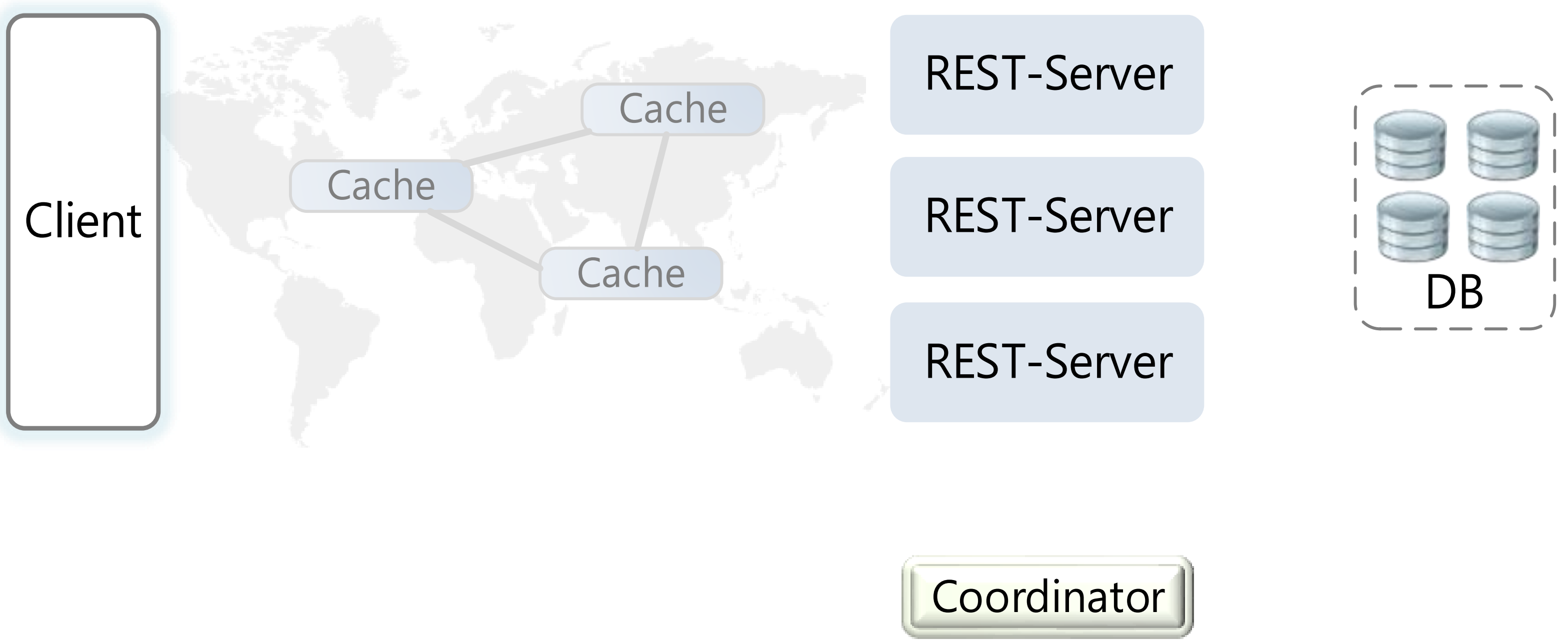
- Solution: **Δ-Bounded Staleness**
  - Clients refresh the Cache Sketch so its age never exceeds Δ

→ *Consistency guarantee*: Δ-atomicity



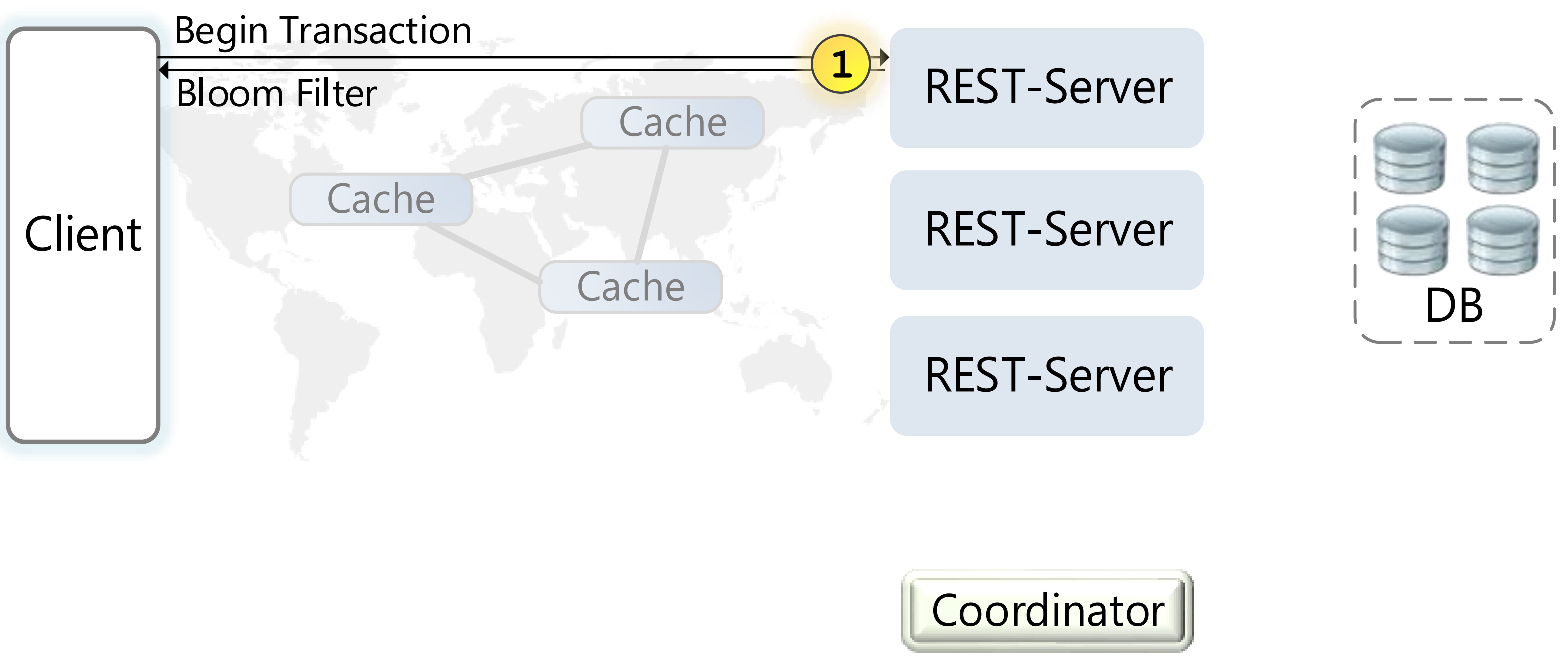
Problem III: High Abort Rates in OCC

- Solution: **Conflict-Avoidant Optimistic Transactions**
  - Cache Sketch fetched with transaction begin
  - **Cached reads → Shorter transaction duration → less aborts**



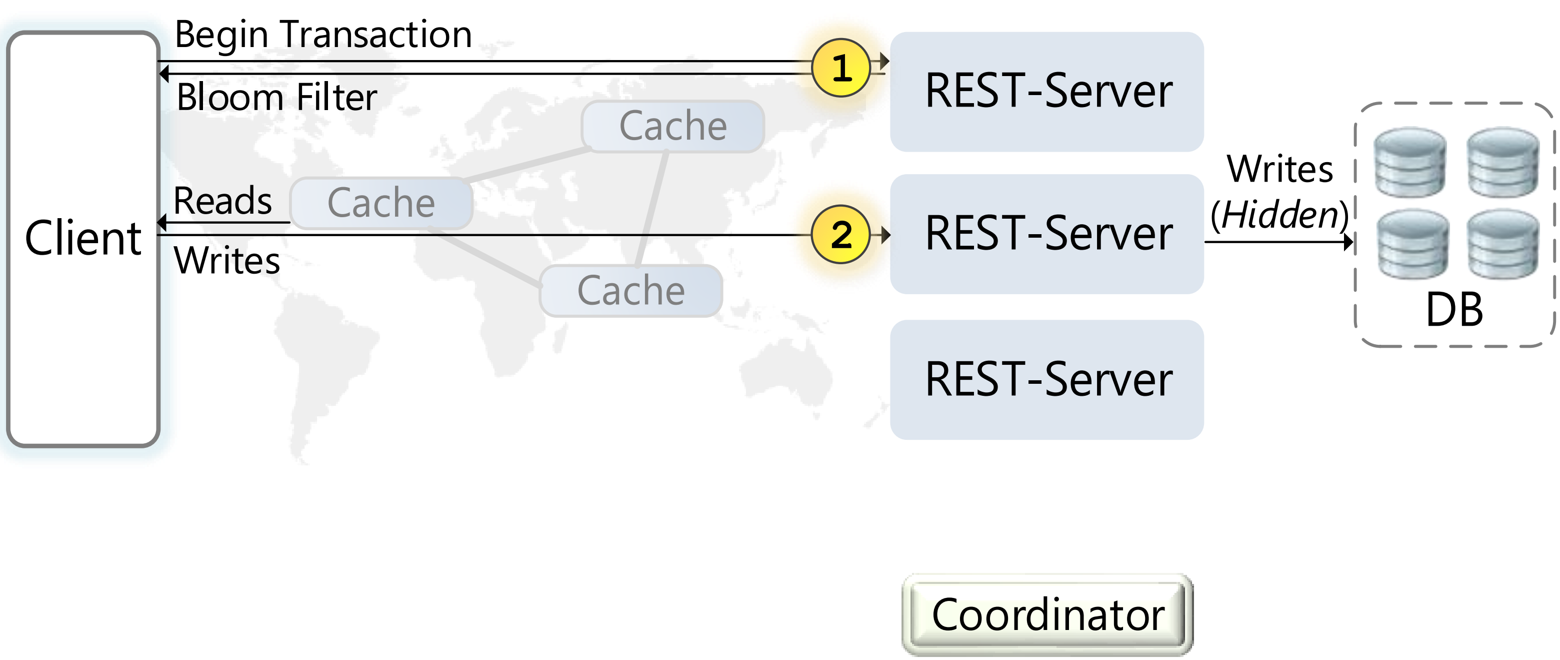
Problem III: High Abort Rates in OCC

- Solution: **Conflict-Avoidant Optimistic Transactions**
  - Cache Sketch fetched with transaction begin
  - **Cached reads → Shorter transaction duration → less aborts**



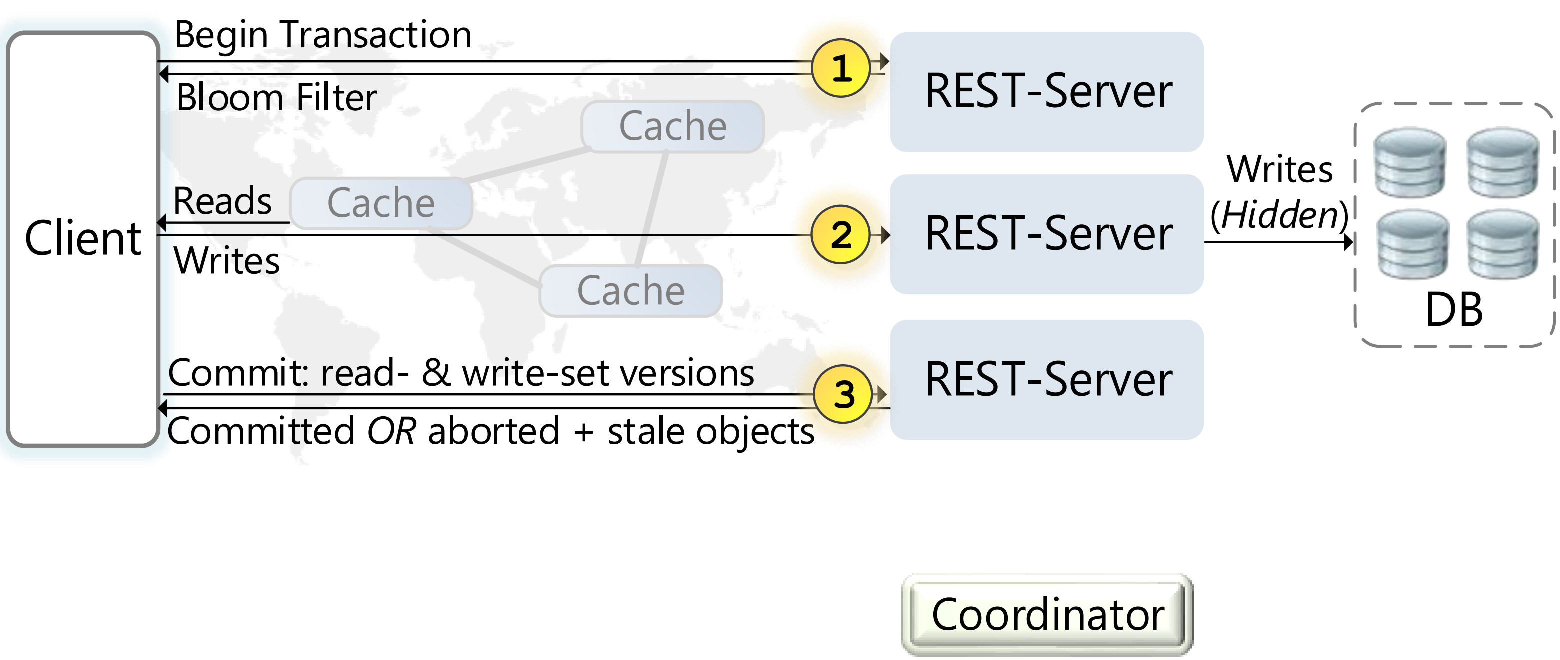
Problem III: High Abort Rates in OCC

- Solution: **Conflict-Avoidant Optimistic Transactions**
  - Cache Sketch fetched with transaction begin
  - **Cached reads → Shorter transaction duration → less aborts**



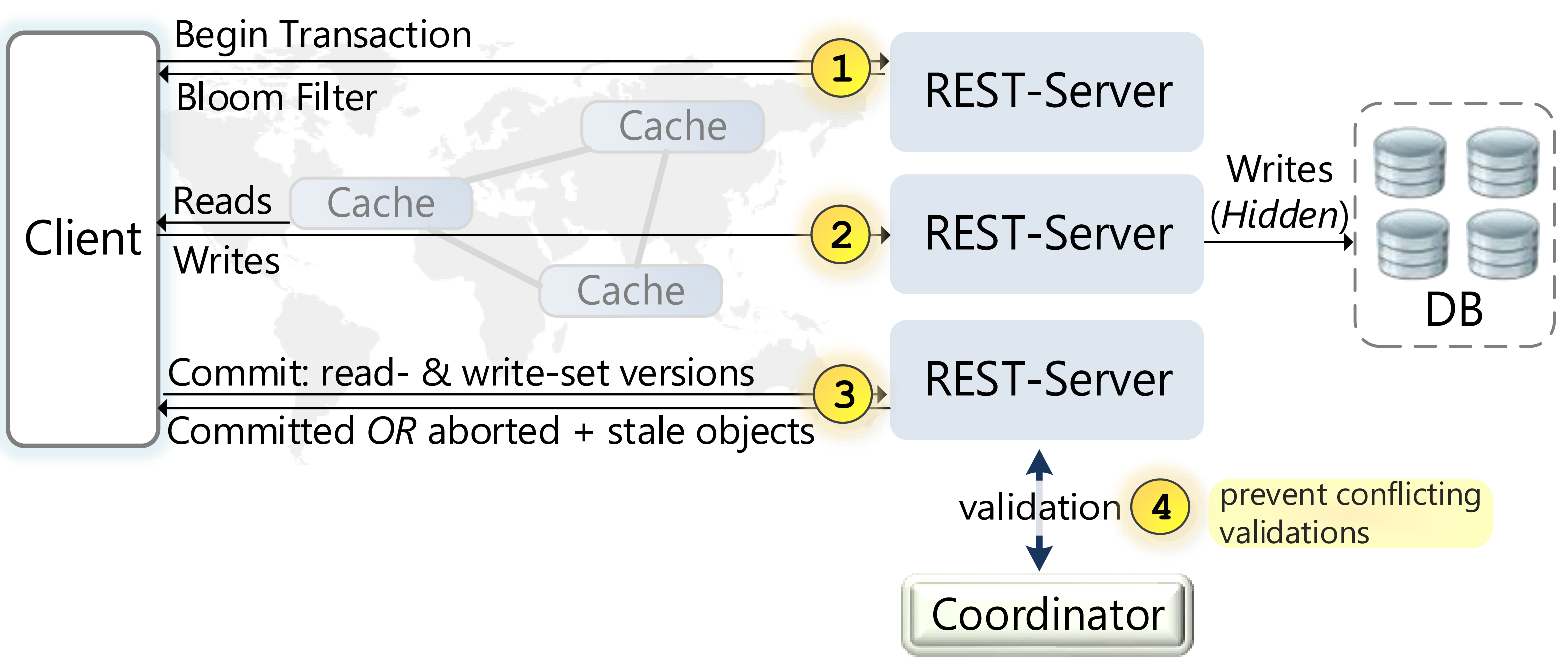
***Problem III*: High Abort Rates in OCC**

- Solution: **Conflict-Avoidant Optimistic Transactions**
  - Cache Sketch fetched with transaction begin
  - **Cached reads → Shorter transaction duration → less aborts**



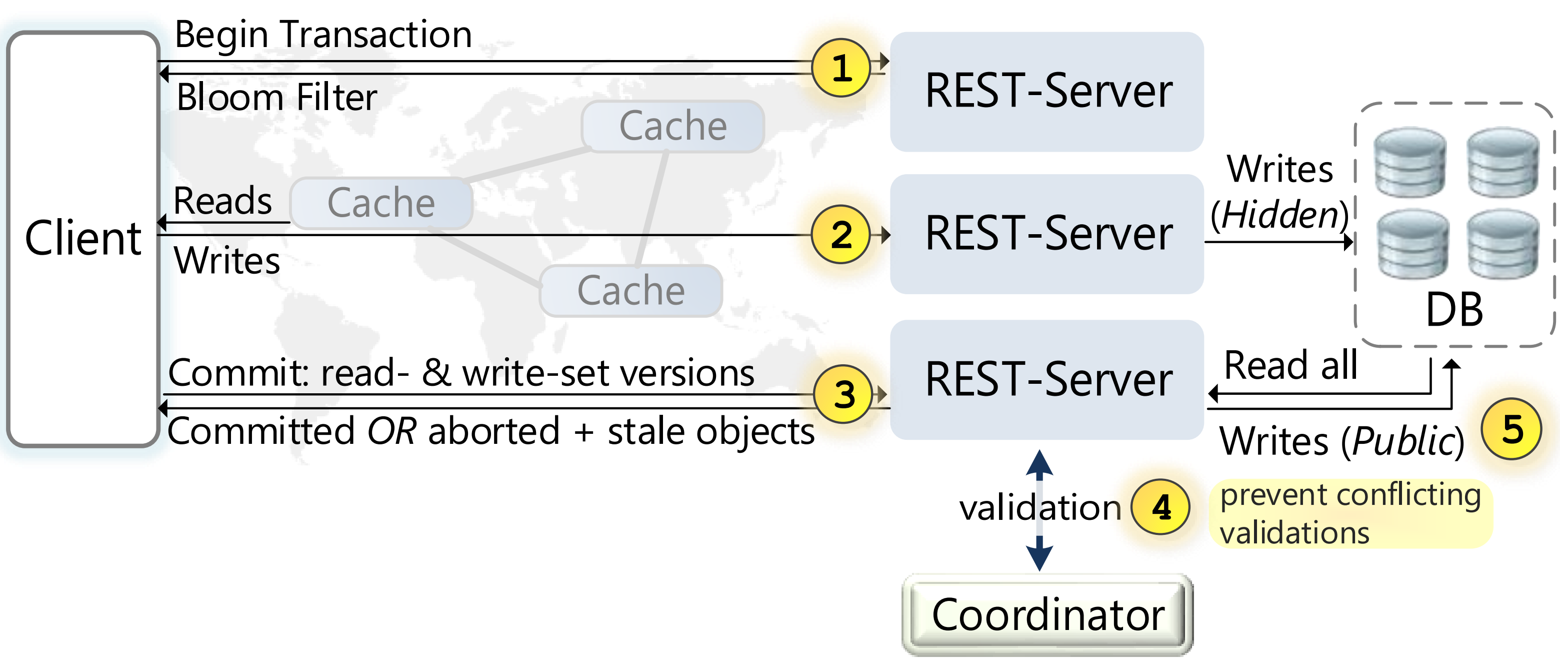
Problem III: High Abort Rates in OCC

- Solution: **Conflict-Avoidant Optimistic Transactions**
  - Cache Sketch fetched with transaction begin
  - **Cached reads → Shorter transaction duration → less aborts**



Problem III: High Abort Rates in OCC

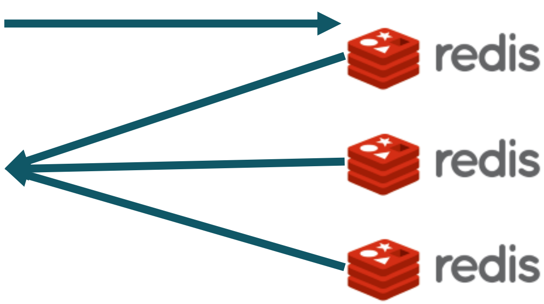
- Solution: **Conflict-Avoidant Optimistic Transactions**
  - Cache Sketch fetched with transaction begin
  - **Cached reads → Shorter transaction duration → less aborts**



The Server Cache Sketch

- Goal: Efficient *Generation of Cache Sketch* and *Invalidation Minimization*
- **Counting Bloom Filter** and **key** → **expiration** mapping

Add key_x if
x unexpired



Cache Sketch
for Table **A**

… **B**

… **C**

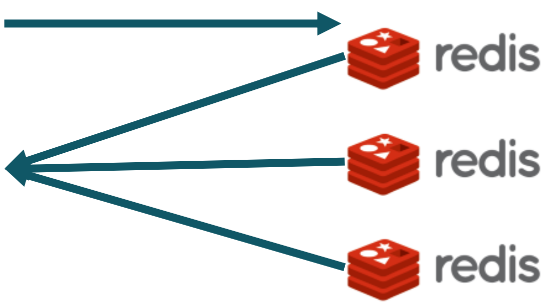
Get Cache Sketch:
Union
(Bitwise AND)

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

The Server Cache Sketch

- Goal: Efficient *Generation of Cache Sketch* and *Invalidation Minimization*
- **Counting Bloom Filter** and **key** → **expiration** mapping

Add key_x if
x unexpired

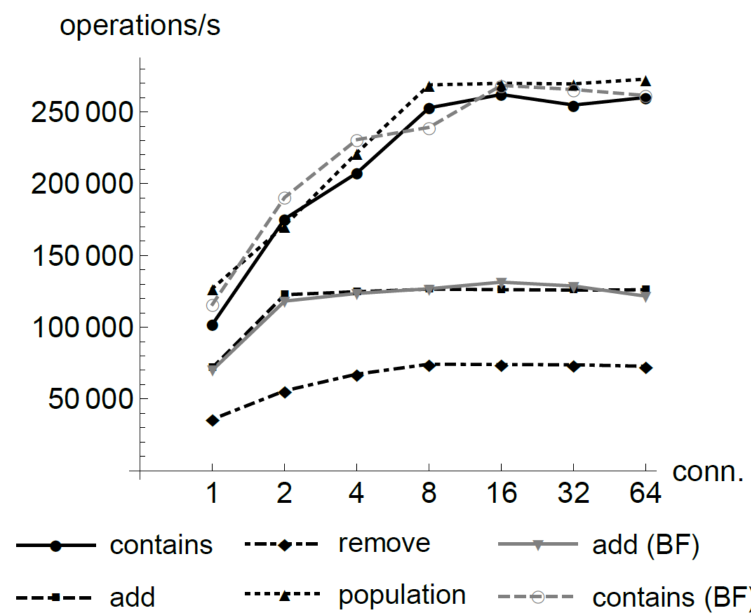


Cache Sketch for Table **A**

… **B**

… **C**

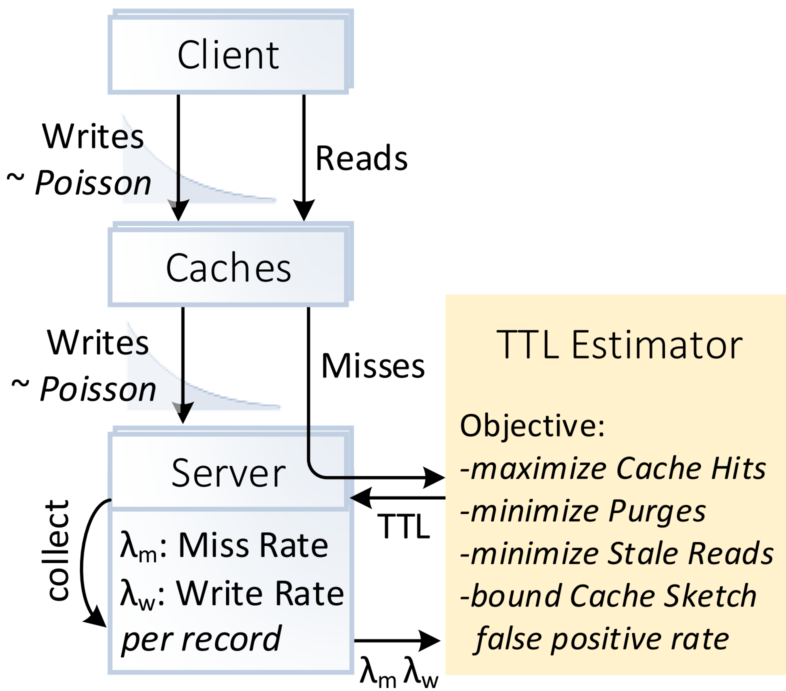
Get Cache Sketch:
Union
(Bitwise AND)



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

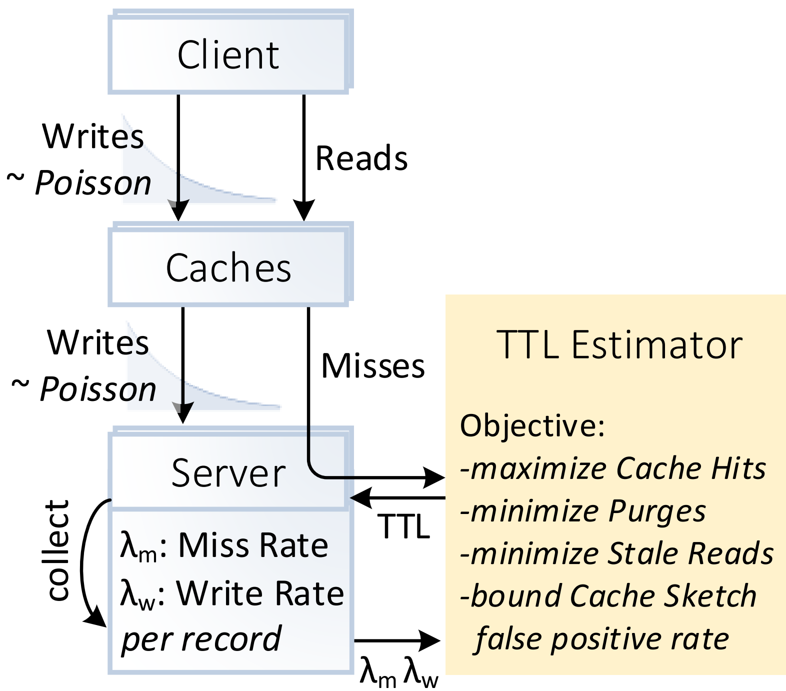
TTL Estimation

- **Problem**: if TTL ≪ time to next write, then it is contained in Cache Sketch unnecessarily long
- **TTL Estimator**: finds „best“ TTL



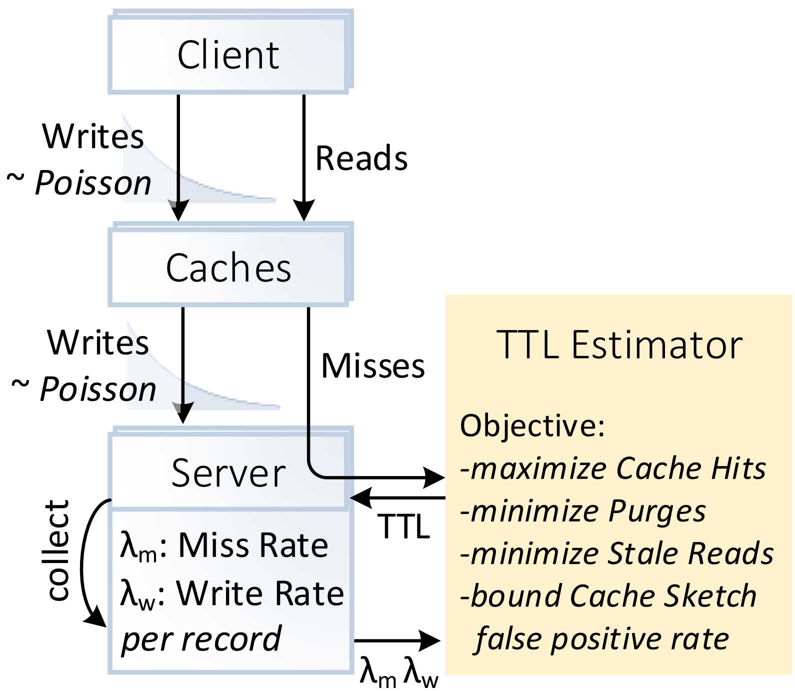
TTL Estimation

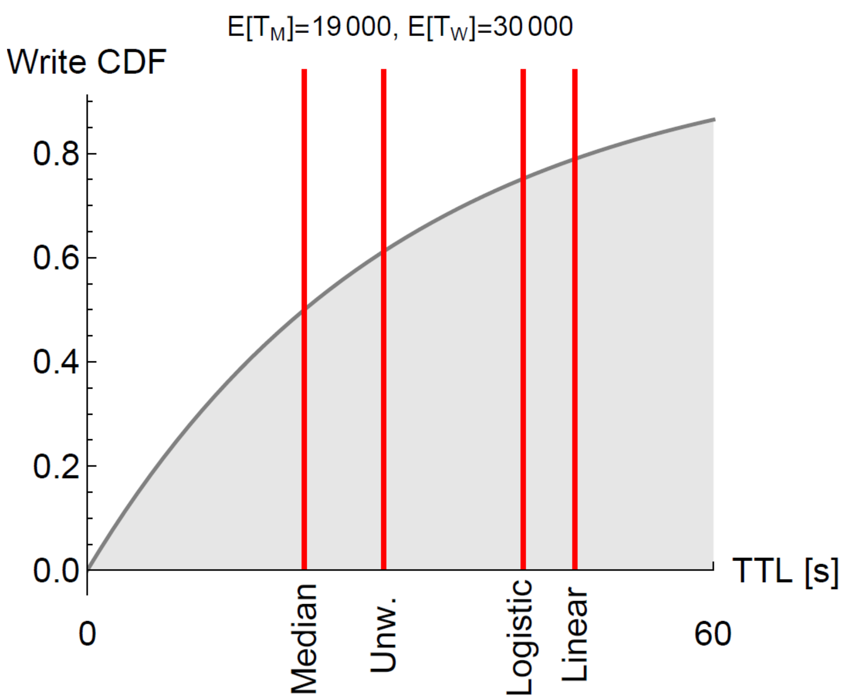
- **Problem**: if TTL ≪ time to next write, then it is contained in Cache Sketch unnecessarily long
- **TTL Estimator**: finds „best“ TTL



TTL Estimation

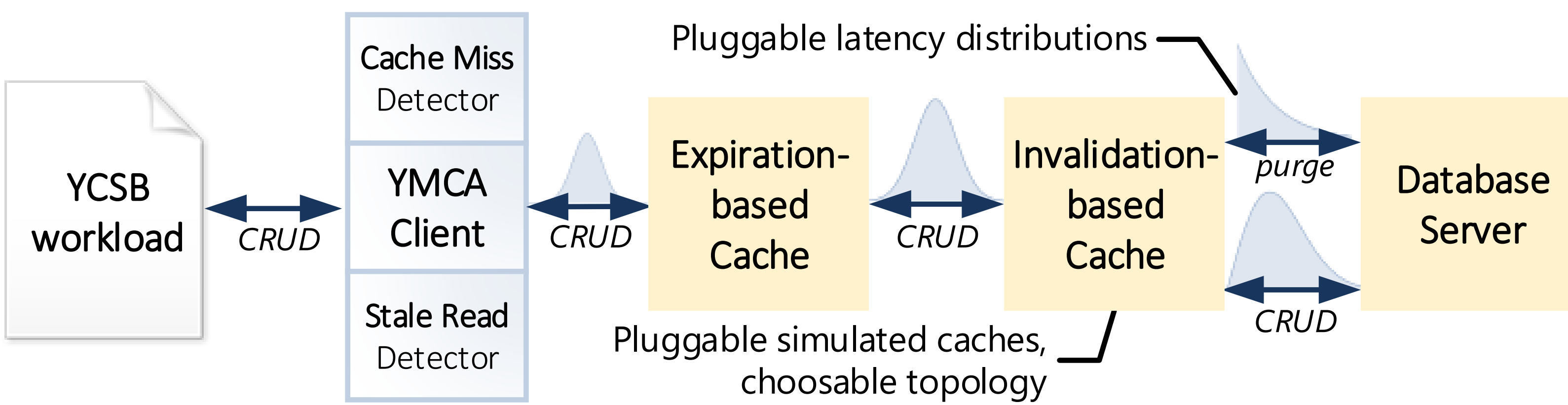
- **Problem**: if TTL ≪ time to next write, then it is contained in Cache Sketch unnecessarily long
- **TTL Estimator**: finds „best“ TTL





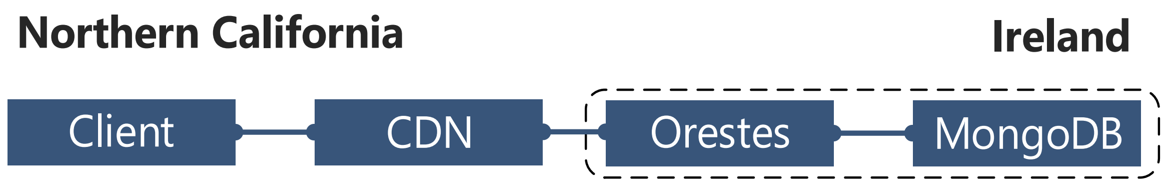
YCSB Monte Carlo Caching Simulator (YMCA)

- Goal: Analysis of arbitrary caching architectures using the standard **YCSB** benchmark
  - **Metrics**: *Latency, TP, Cache Hits, Stale Reads, Invalidations*
  - **Training of TTL Estimator**: Hill Climber finds optimal params

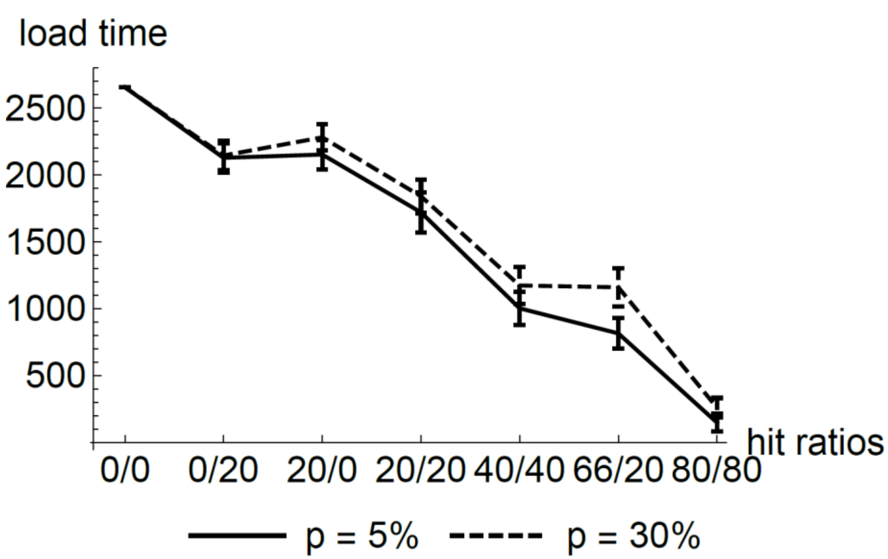


Results: Simulation & real-world

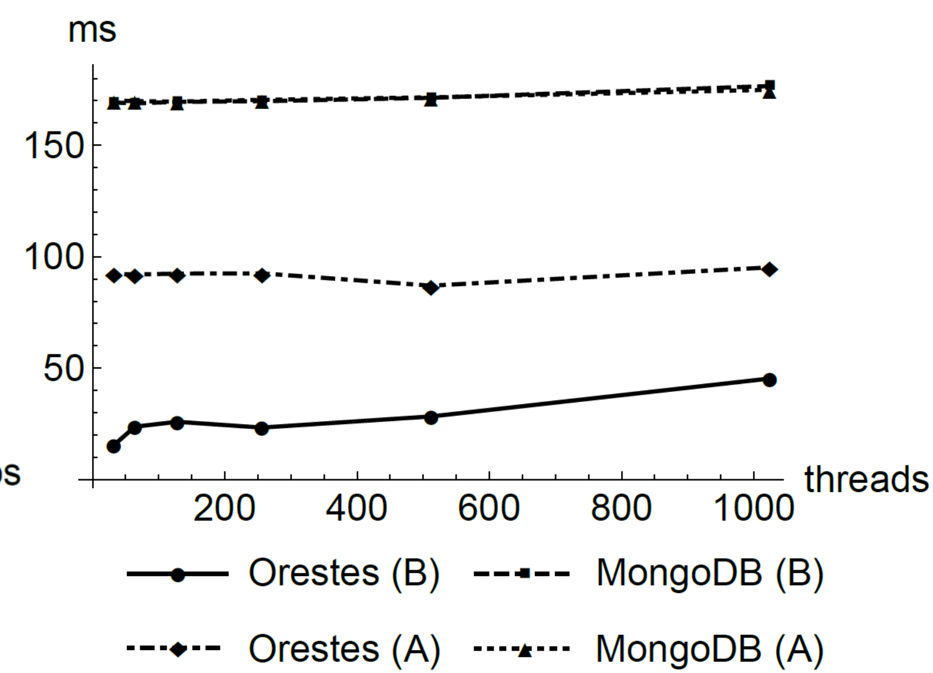
Setup:



Page load times with **cached initialization** (YMCA):



Average Latency for YCSB Workloads A and B (real):



Outline

- Motivation
- Cache Sketch Approach
- **Polyglot Persistence Mediator**
  - Idea
  - Process
  - Evaluation
- Wrap-up and future work

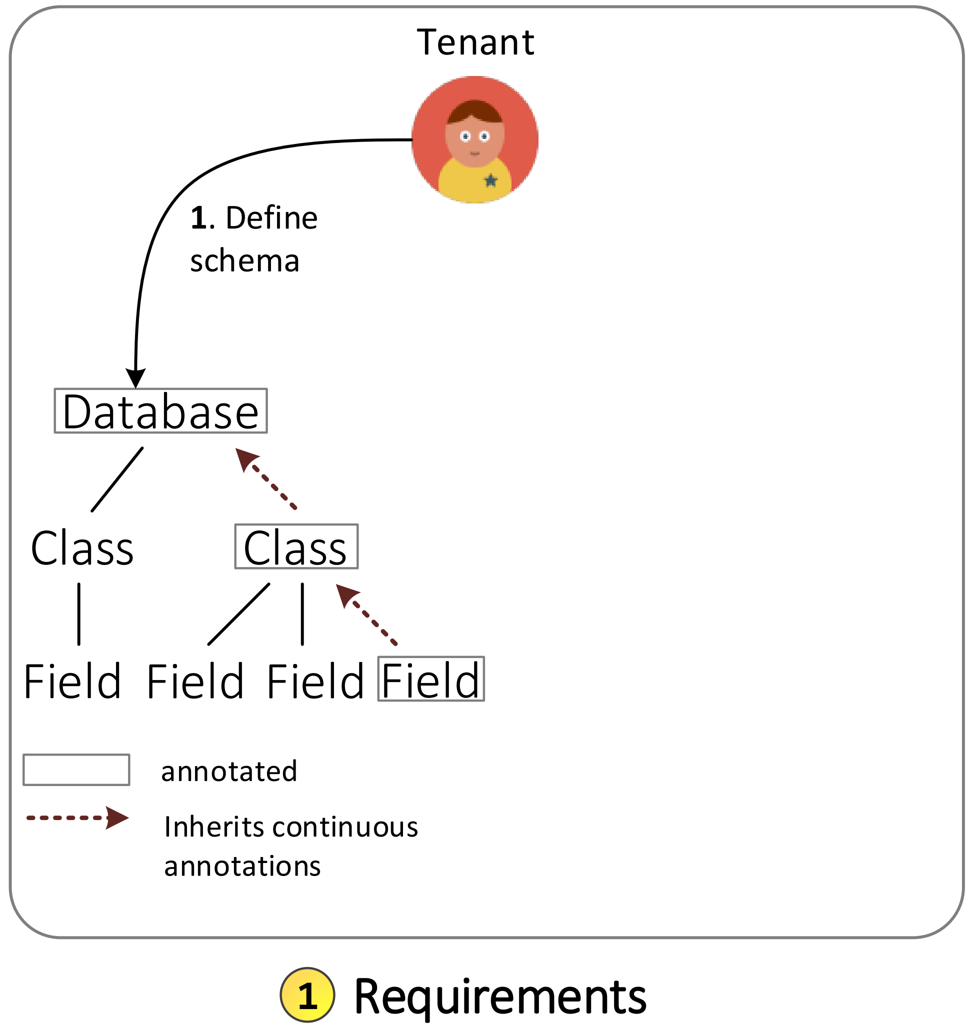
The Polyglot Persistence Mediator

- Fully transparent choice of DB, based on requirements (SLAs)
- 3-step-process:

  1. Requirements

  2. Resolution

  3. Mediation



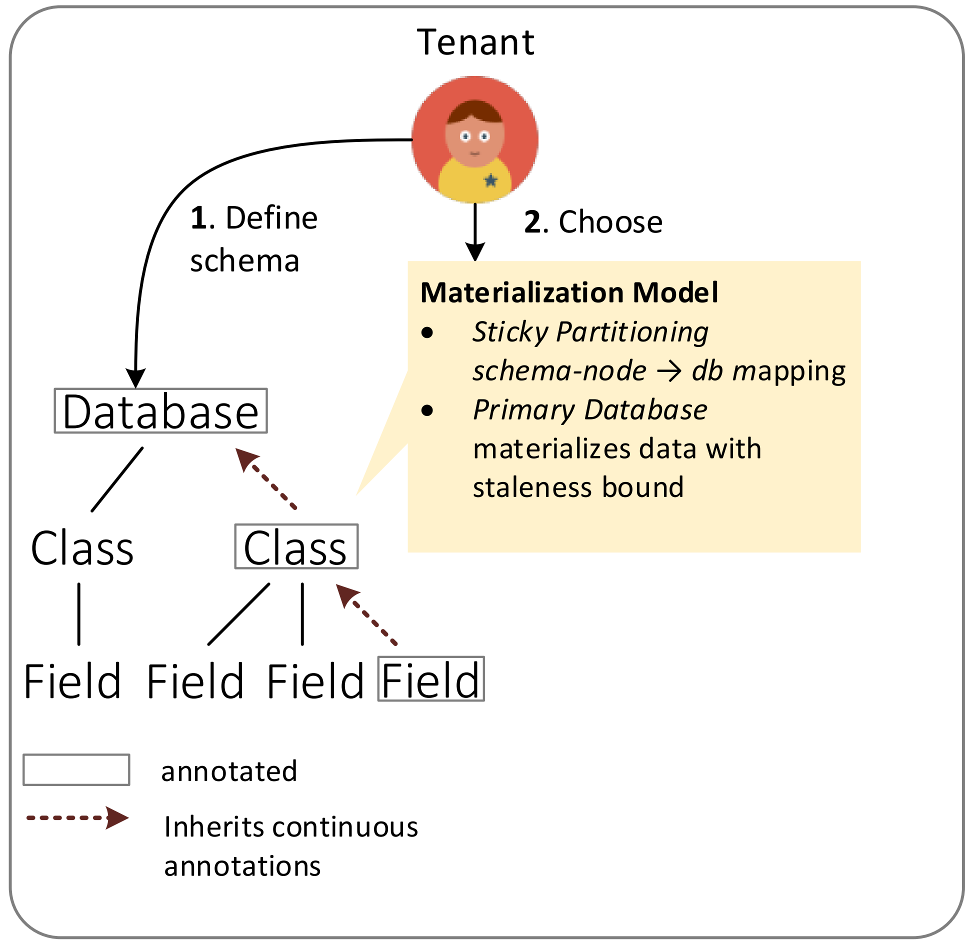
The Polyglot Persistence Mediator

- Fully transparent choice of DB, based on requirements (SLAs)
- 3-step-process:

  1. Requirements

  2. Resolution

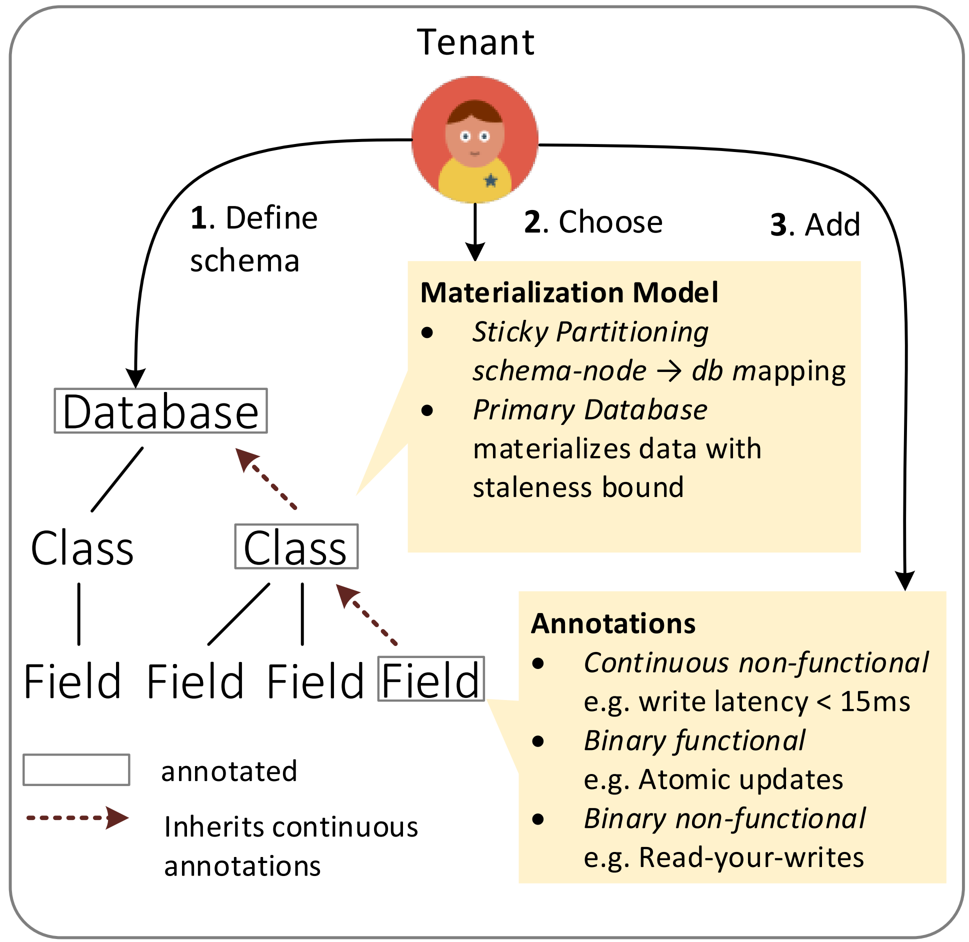
  3. Mediation



1 Requirements

The Polyglot Persistence Mediator

- Fully transparent choice of DB, based on requirements (SLAs)
- 3-step-process:
  1. Requirements
  2. Resolution
  3. Mediation



1 Requirements

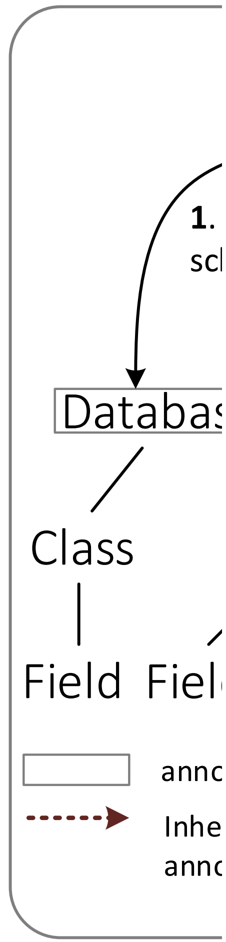
The Polyglot Persistence Mediator

- Fully transparent choice of DB, based on requirements (SLAs)
- 3-step-process:

  1. Requirements

  2. Resolution

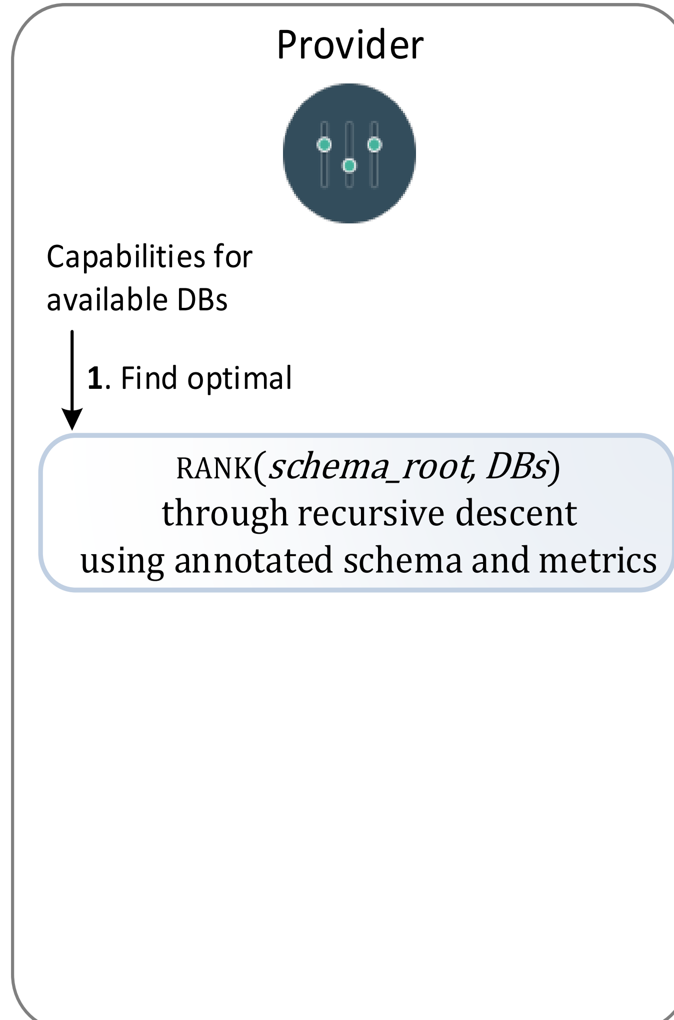
  3. Mediation



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

PPM Step 2: Resolution

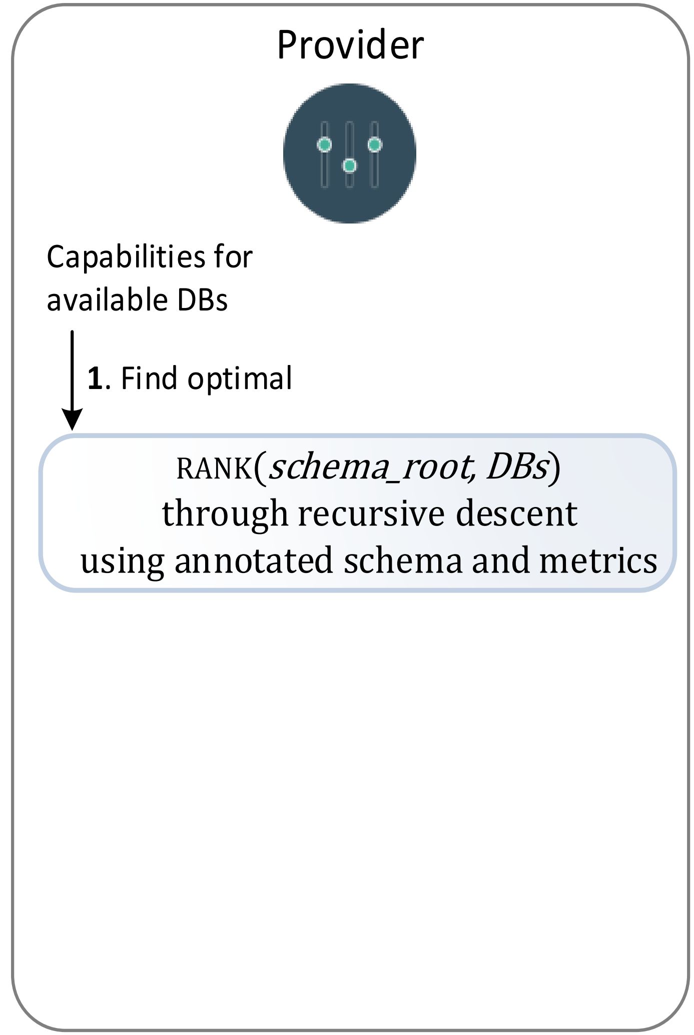
- The Provider resolves the requirements
- RANK algorithm recursively analyzes schema annotations
- For each schema element:
  - Find each DB that satisfies all binary requirements
  - Pick the one that has the best score according to utility function of historic or predicted metrics



2 Resolution

PPM Step 2: Resolution

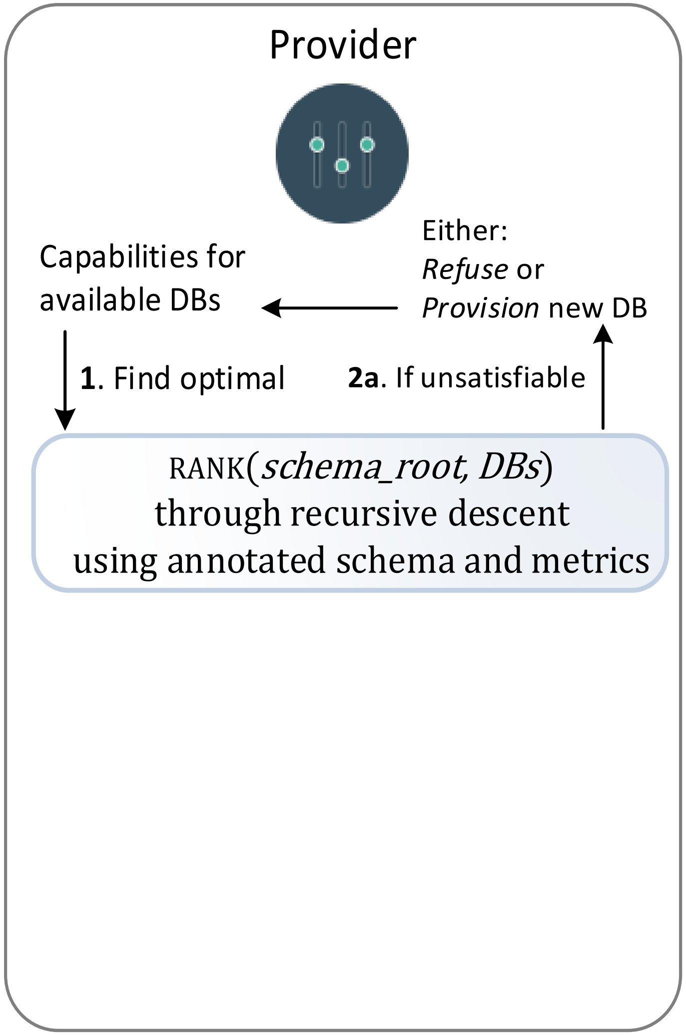
- The Provider resolves the requirements
- RANK algorithm recursively analyzes schema annotations
- For each schema element:
  - Find each DB that satisfies all binary requirements
  - Pick the one that has the best score according to utility function of historic or predicted metrics



2 Resolution

PPM Step 2: Resolution

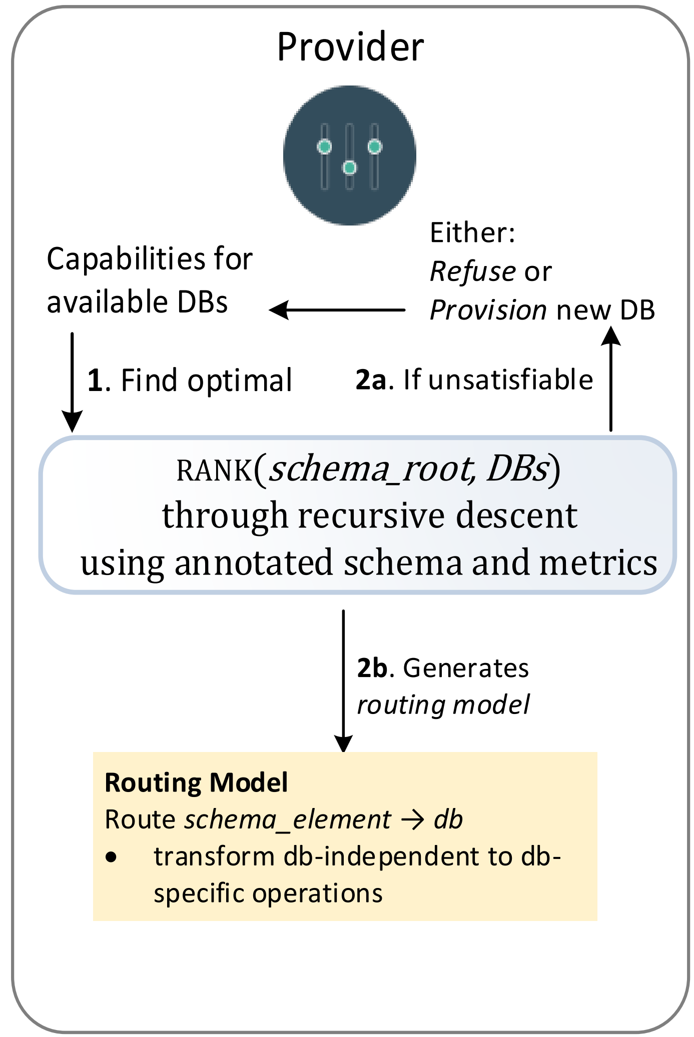
- The Provider resolves the requirements
- RANK algorithm recursively analyzes schema annotations
- For each schema element:
  - Find each DB that satisfies all binary requirements
  - Pick the one that has the best score according to utility function of historic or predicted metrics



2 Resolution

PPM Step 2: Resolution

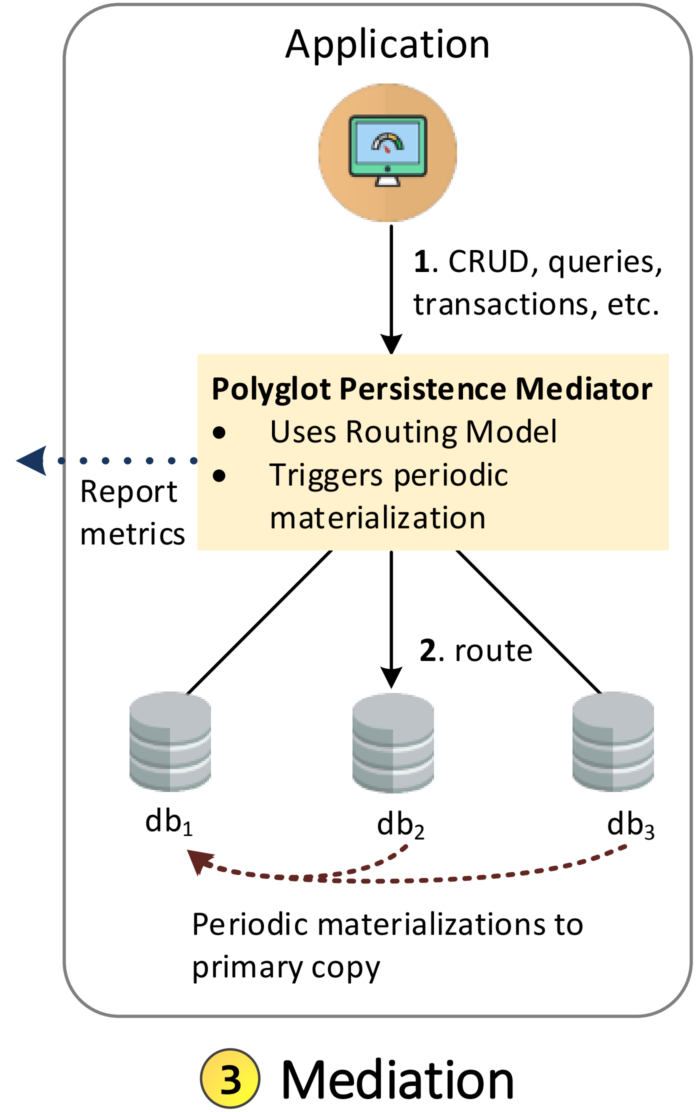
- The Provider resolves the requirements
- RANK algorithm recursively analyzes schema annotations
- For each schema element:
  - Find each DB that satisfies all binary requirements
  - Pick the one that has the best score according to utility function of historic or predicted metrics



2 Resolution

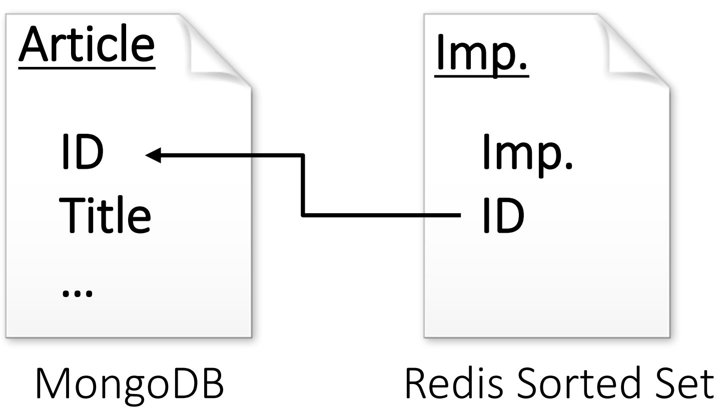
PPM Step 3: Mediation

- The PPM routes data and operations to the chosen DBs
- In the *primary database model*, it triggers periodic materializations
- Metrics (latency, availability, etc.) are reported to the resolver



Evaluation

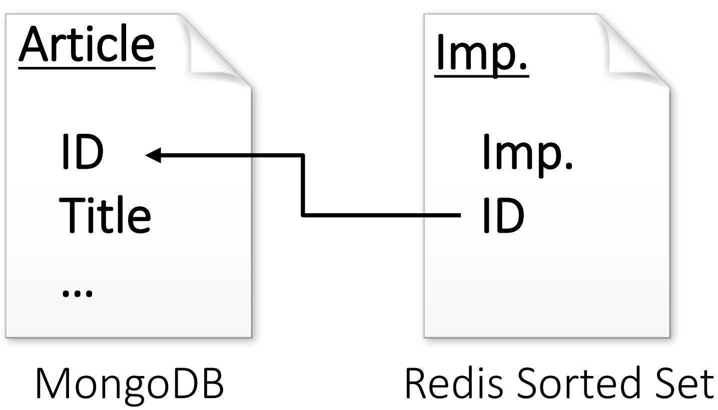
Scenario: news articles with impression counts
Objective: low-latency top-k querys, high-throughput counts



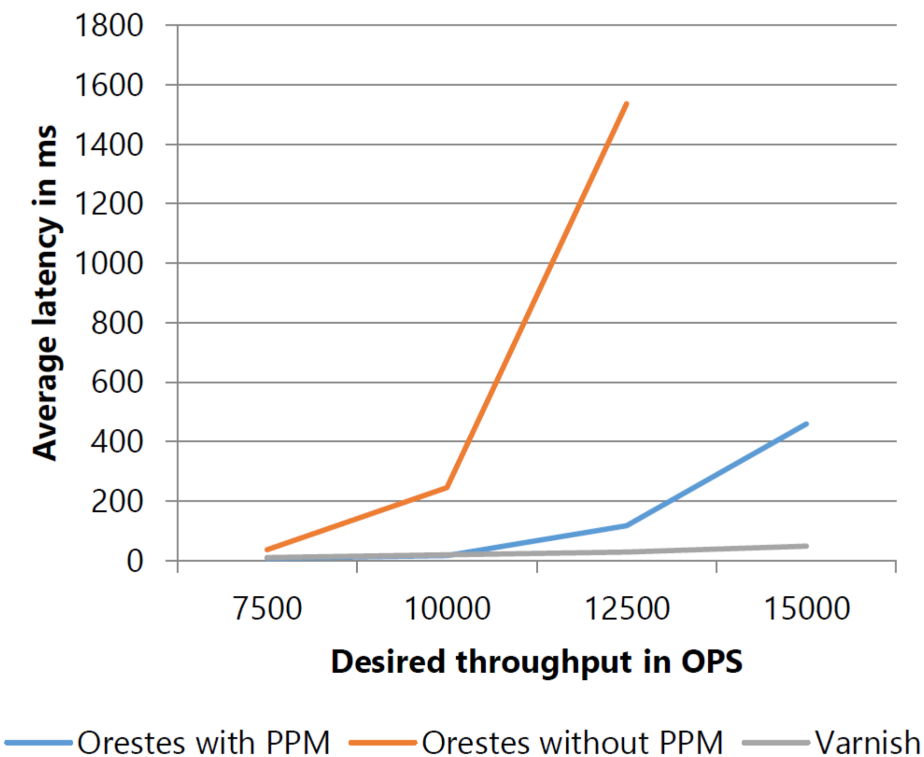
Found Resolution

Evaluation

Scenario: news articles with impression counts
Objective: low-latency top-k querys, high-throughput counts



Found Resolution



Outline

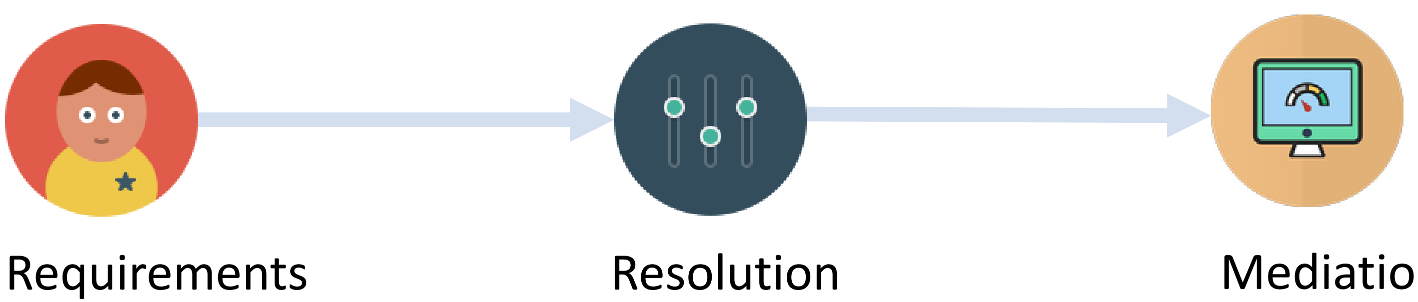
- Motivation
- Cache Sketch Approach
- Polyglot Persistence Mediator
- **Wrap-up and future work**
  - Summary
  - Research areas
  - Baqend

Summary

- **Cache Sketch**: dual approach to web caching for database services
  - Consistent (Δ-atomic) *expiration-based* caching
  - *Invalidation-based* caching with minimal purges

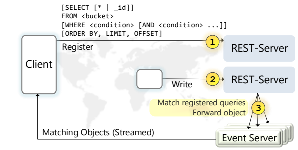
Cached Initialization	Δ-Bounded Staleness	Conflict-Avoidant Transactions	Invalidation Minimization

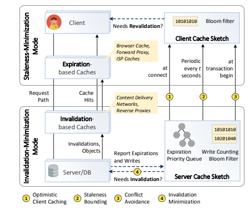
- **Polyglot Persistence Mediator**: SLA-driven, fine-grained selection of database systems

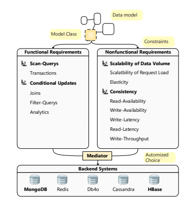


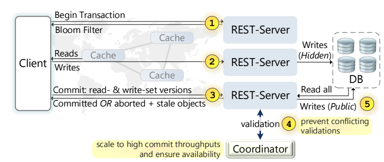
Future & Current Work

- **Cache Sketch**:
  - Online learning of best *TTL estimation*
  - Quantify *COT* properties
  - *Query* result caching
  - Extend *YMCA* to replication and sharding architectures
- **Polyglot Persistence**:
  - Common requirements/SLA „*library*“
  - Implementation of *Resolution*
  - *Live Migration*
  - Scalable *metrics* aggregation

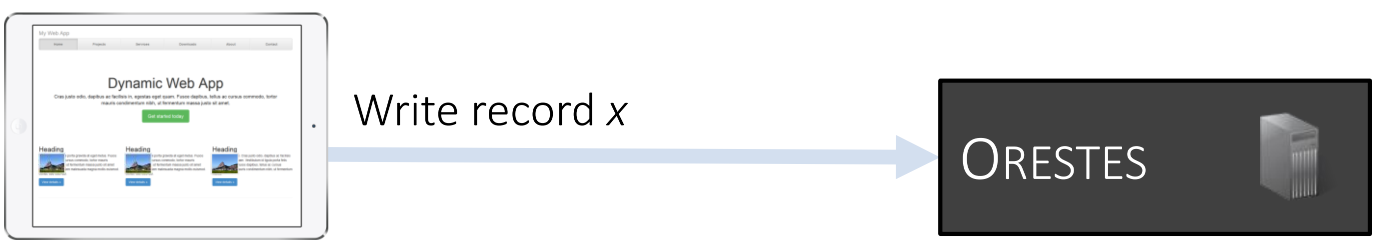






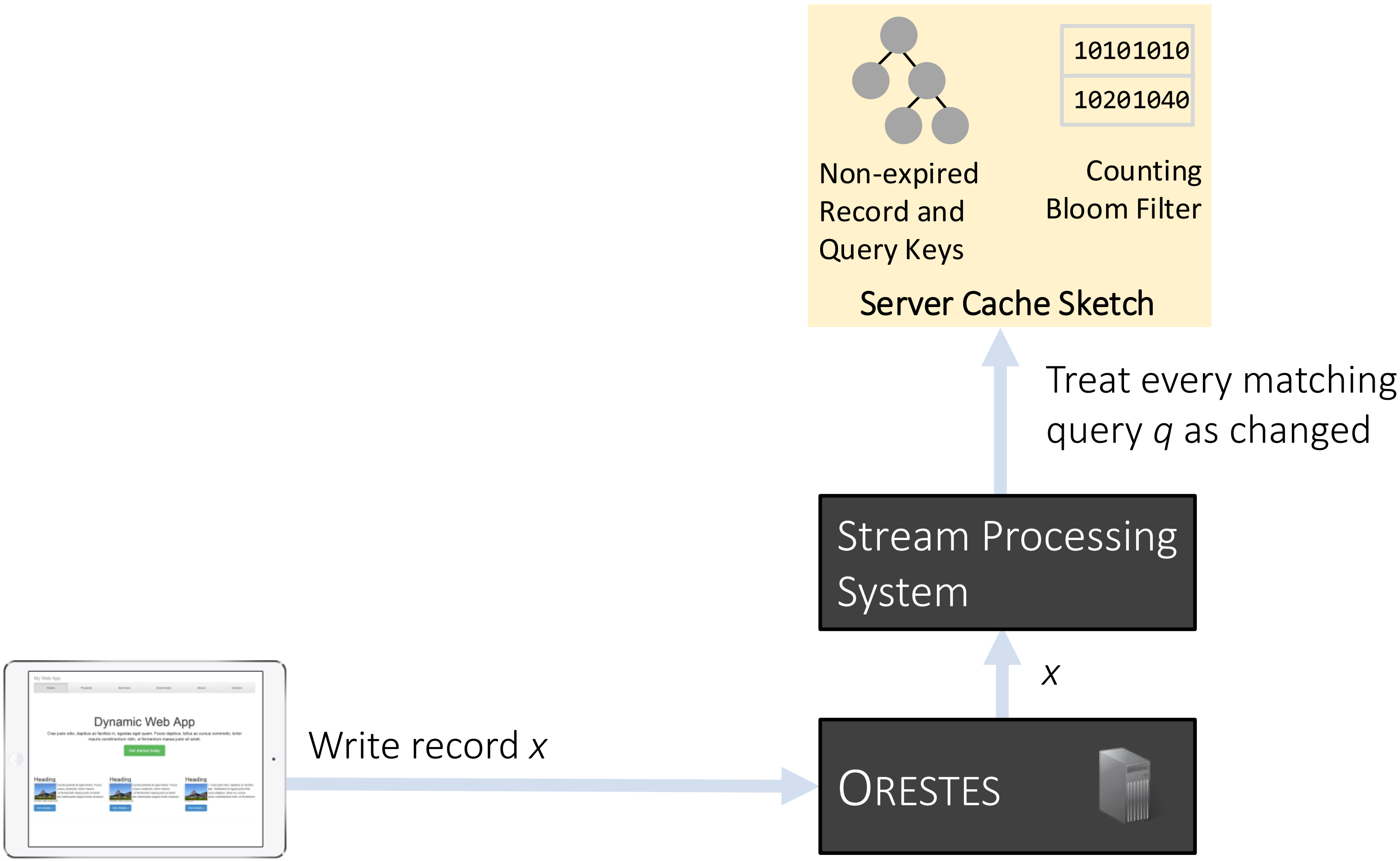


Future Work: Query Caching

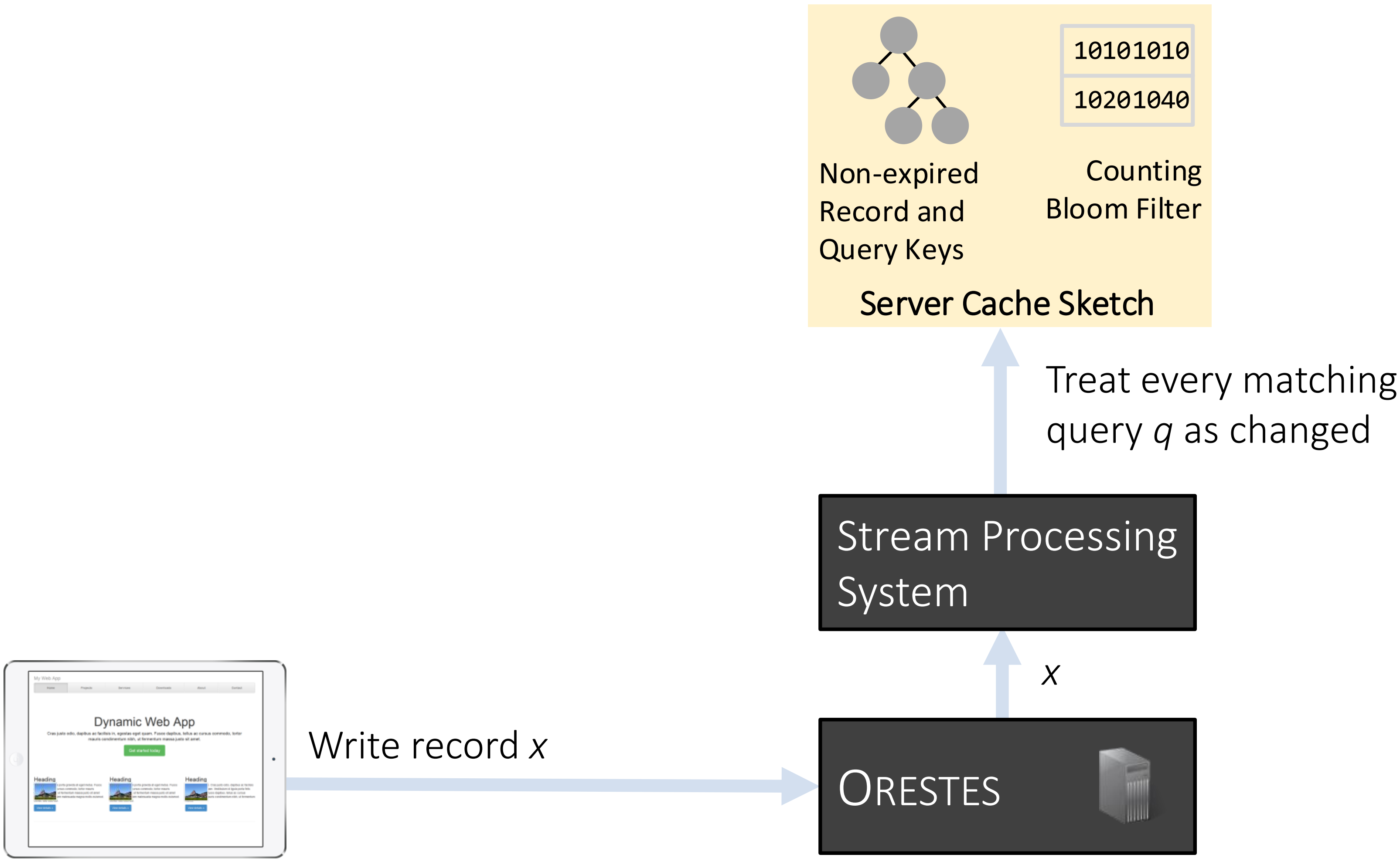


Future Work: Query Caching

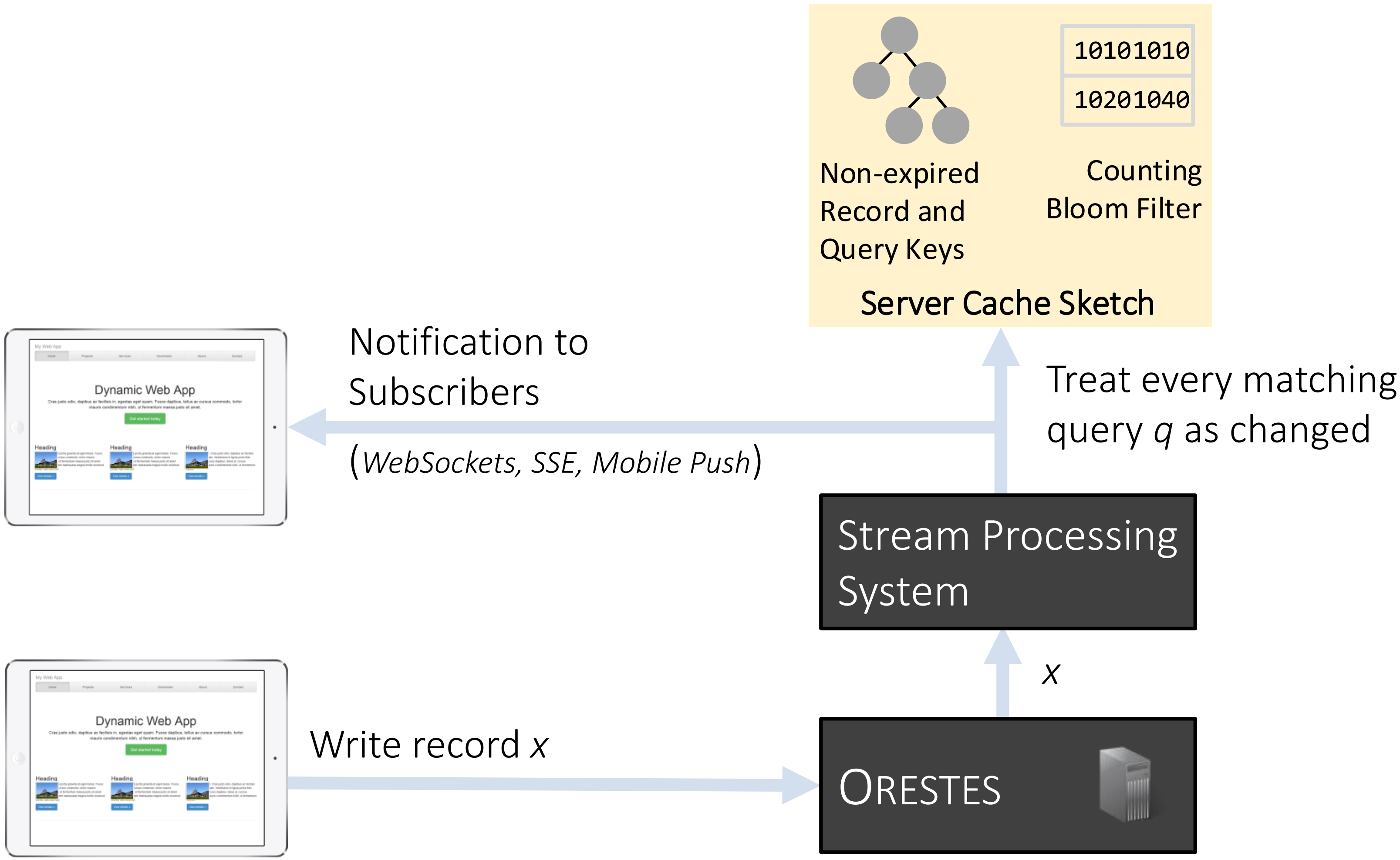
Future Work: Query Caching



Future Work: Query Caching



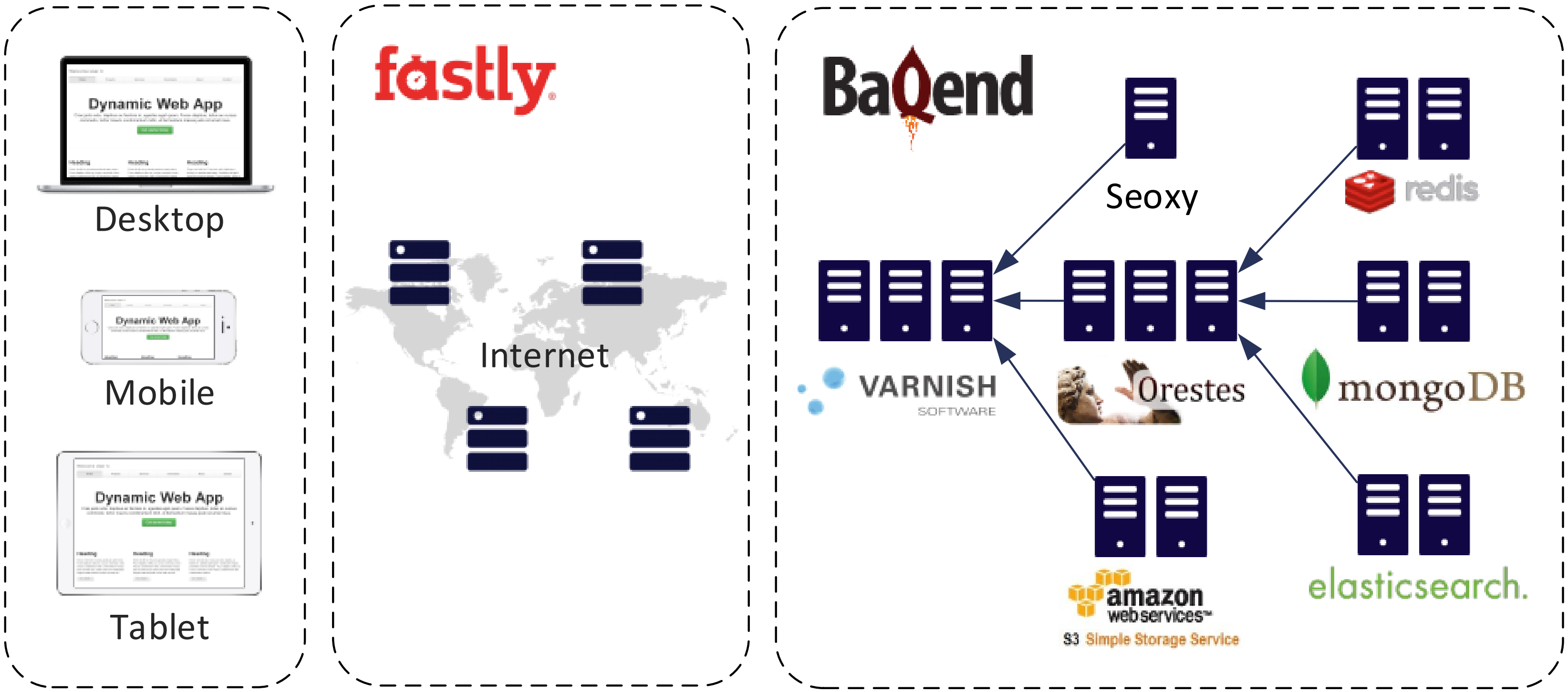
Future Work: Query Caching




BaQend
Build faster Apps faster.

Baqend

- Orestes as a startup, funded since March 2014



Thank you

Contact:

gessert@informatik.uni-hamburg.de

http://orestes.info

http://baqend.com