Cache Sketches

Using Bloom Filters and Web Caching Against Slow Load Times

Felix Gessert, Florian Bücklers
{fg,fb}@baqend.com
@baqendcom

Who we are

- Felix Gessert, Florian Bücklers

Research Project since 2010

Orestes

Baqend

Build faster Apps faster.

Backend-as-a-Service **Startup** since 2014

Introduction | Main Part | Conclusions

Web Performance: State of the Art → Cache Sketch: Research Approach → Using Web Caching in Applications

Presentation
is loading

Why performance matters

1040b0001ms

Average: 9,3s

Loading…

Conversions

% Traffic

% Visitors

% Revenue

AberdeenGroup

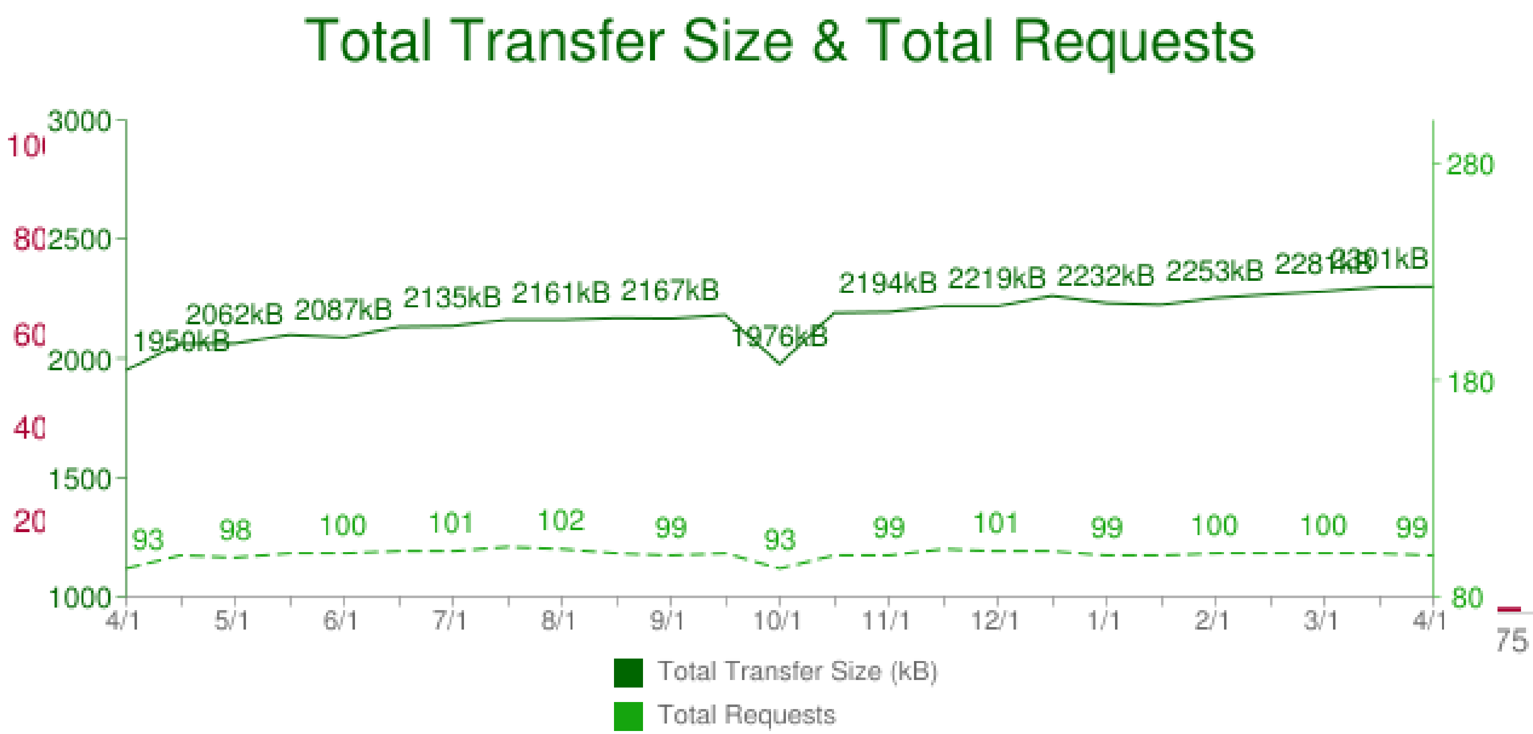
Google

YAHOO!

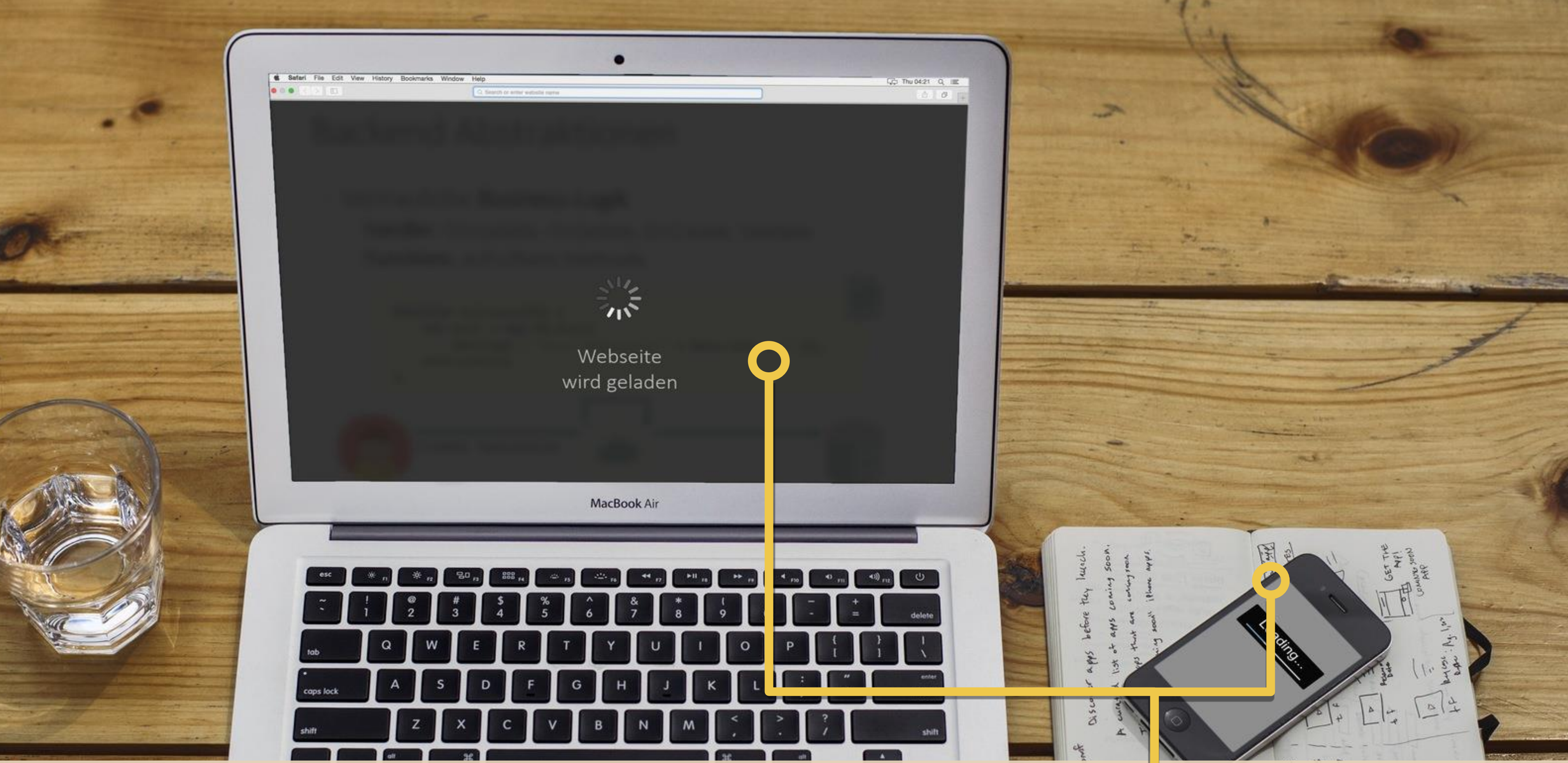
amazon.com

An Average Website

Some Statistics



http://httparchive.org/

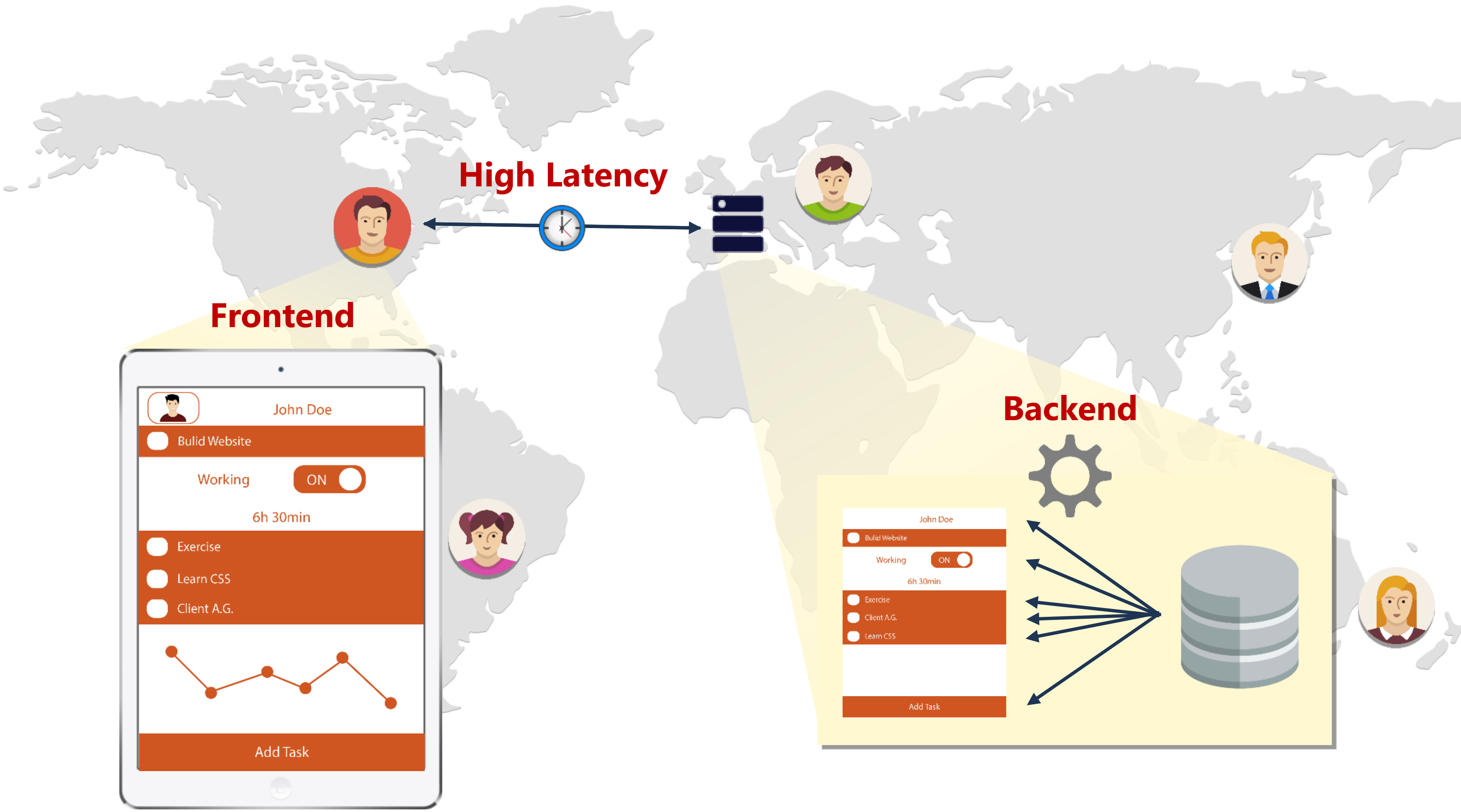


If perceived speed is such an important factor

...what causes slow page load times?

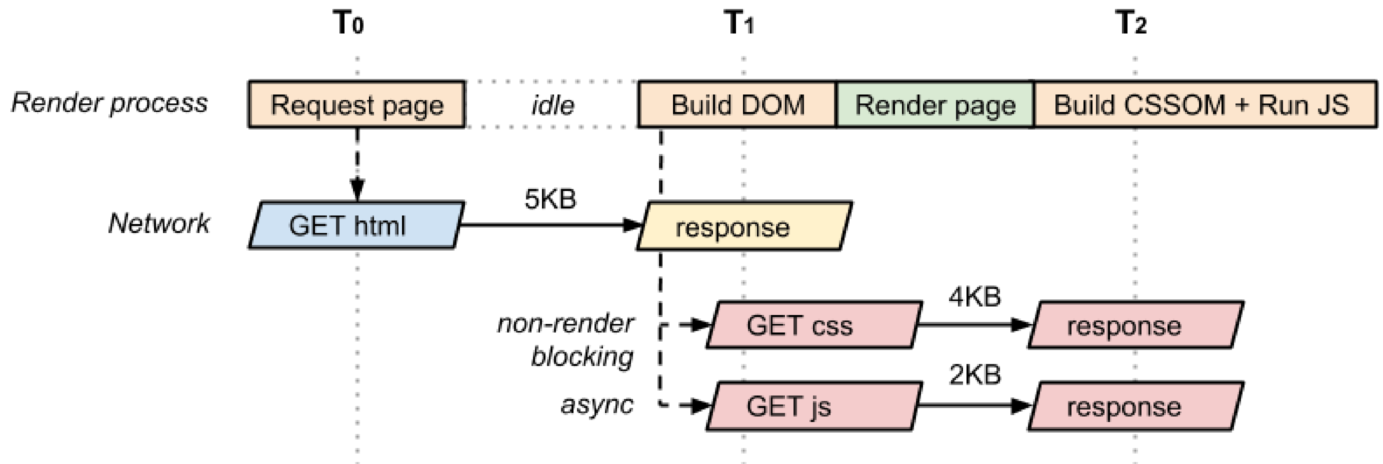
The Problem

Three Bottlenecks: Latency, Backend & Frontend



Frontend Performance

Break-down of the Critical Rendering Path



- Achieve a fast render of the page by:
  - Reducing the **critical resources** needed
  - Reducing the **critical byte**s which must be transferred
  - Loading JS, CSS and HTML templates **asynchronously**
  - Rendering the page **progressively**
  - **Minifying** & **Concatenating** CSS, JS and images

Google Developers, Web Fundamentals
https://developers.google.com/web/fundamentals/performance/critical-rendering-path/analyzing-crp.

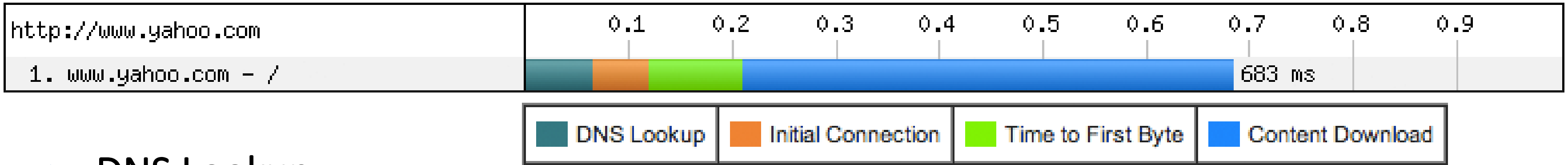
Frontend Performance

Tools to improve your page load

- Well known problem & good tooling:
  - Optimizing CSS (*postcss*)
  - Concatenating CSS and JS (*processhtml*)
  - Minification and Compression (*cssmin, UglifyJS, Google Closure, imagemin*)
  - Inline the critical CSS (*addyosmani/critical*)
  - Hash assets to make them cacheable (*gulp-rev-all*)

Network Performance

Break down of a single resource load



- DNS Lookup
  - Every domain has its own DNS lookup
- Initial connection
  - TCP makes a three way handshake → 2 roundtrips
  - SSL connections have a more complex handshake → +2 roundtrips
- Time to First Byte
  - Depends heavily on the distance between client and the backend
  - Includes the time the backend needs to render the page

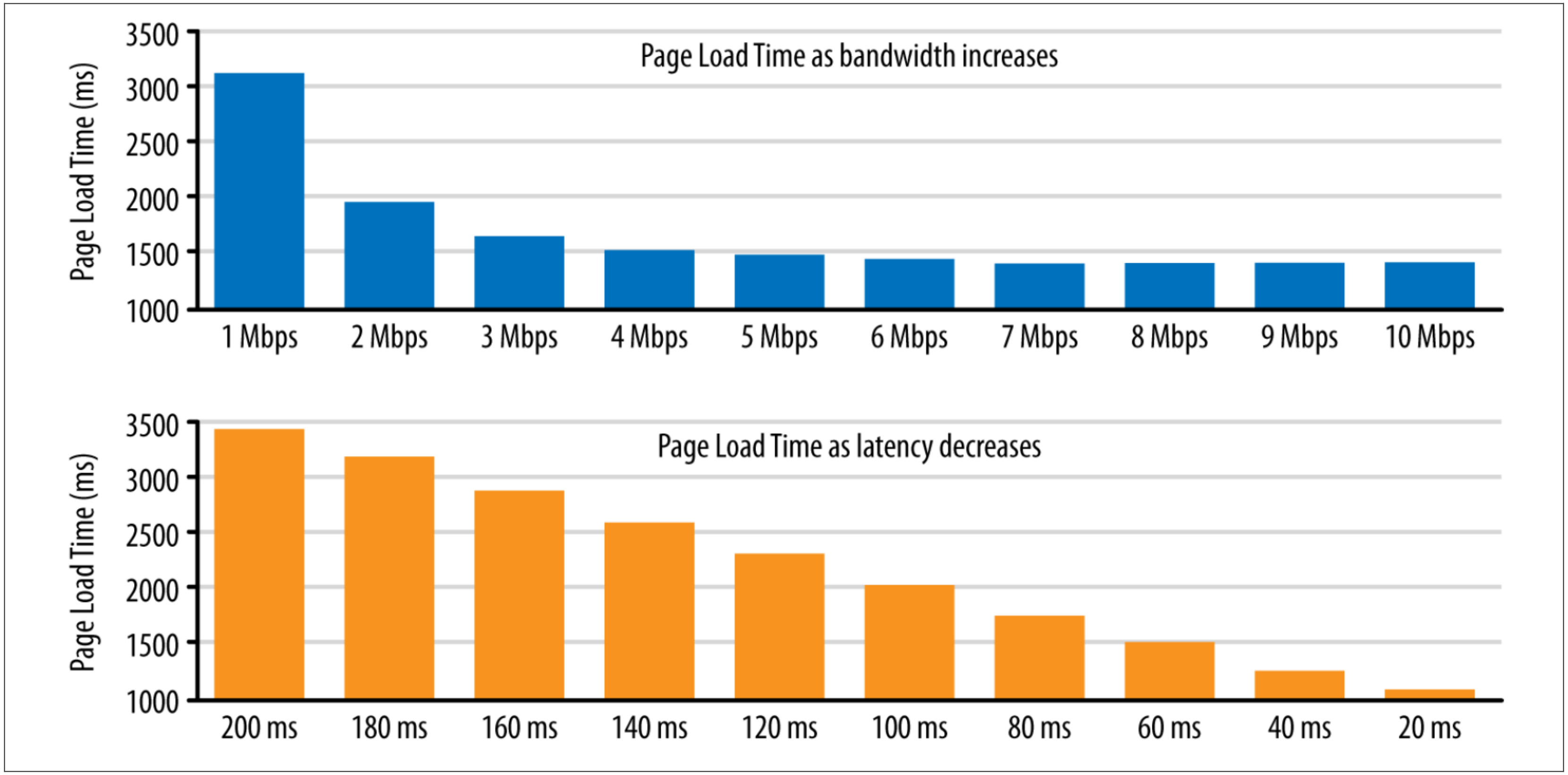
    → Session lookups, Database Queries, Template rendering …
- Content Download
  - Files have a high transfer time on new connections, since the initial congestion window is small → many roundtrips

Network Performance

Common Tuning Knobs

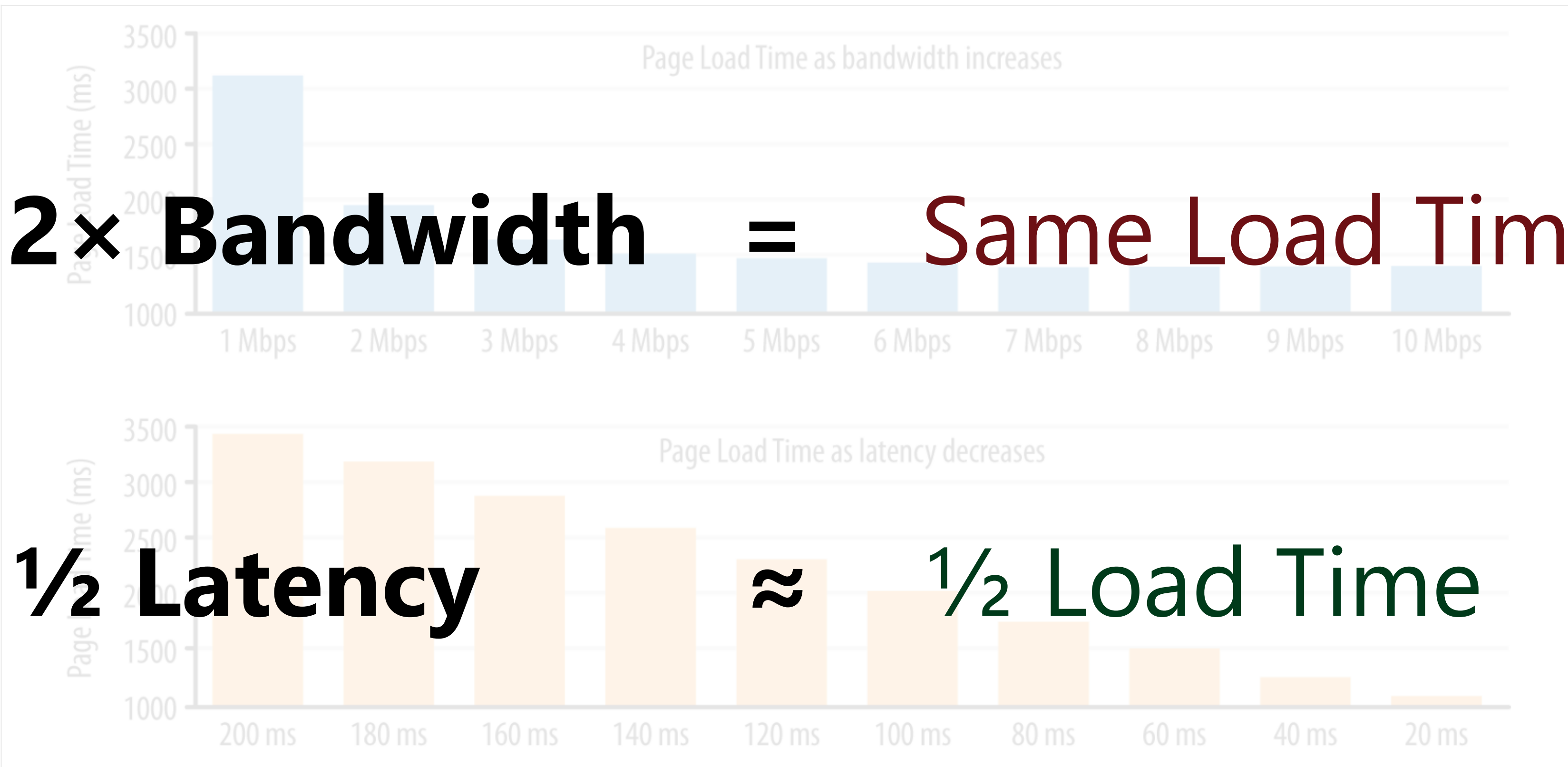
- Persistent connections, if possible HTTP/2
- Avoid redirects
- Explicit caching headers (no heuristic caching)
- Content Delivery Networks
  - To reduce the distance between client and server
  - To cache images, CSS, JS
  - To terminate SSL early and optimized
- Single Page Apps:
  - Small initial page that loads additional parts asynchronously
  - Cacheable HTML templates + load dynamic data
  - Only update sections of the page during navigation

Network Latency: Impact



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

Network Latency: Impact



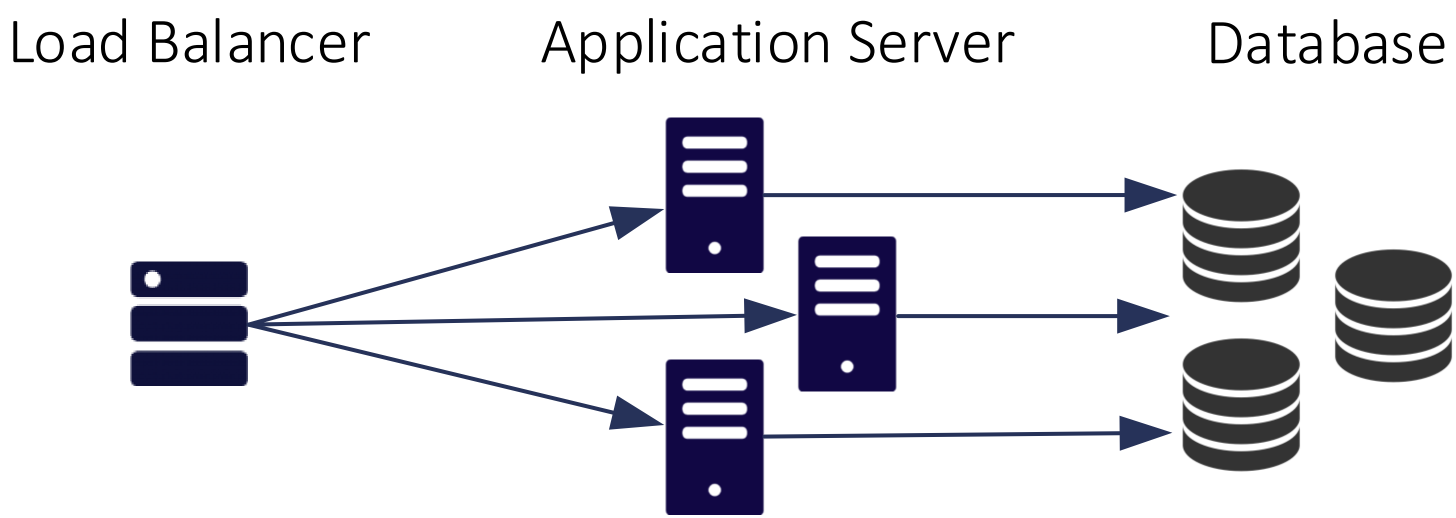
2× Bandwidth = Same Load Time

½ Latency ≈ ½ Load Time

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

Backend Performance

Scaling your backend



- Load Balancing
- Auto-scaling
- Failover

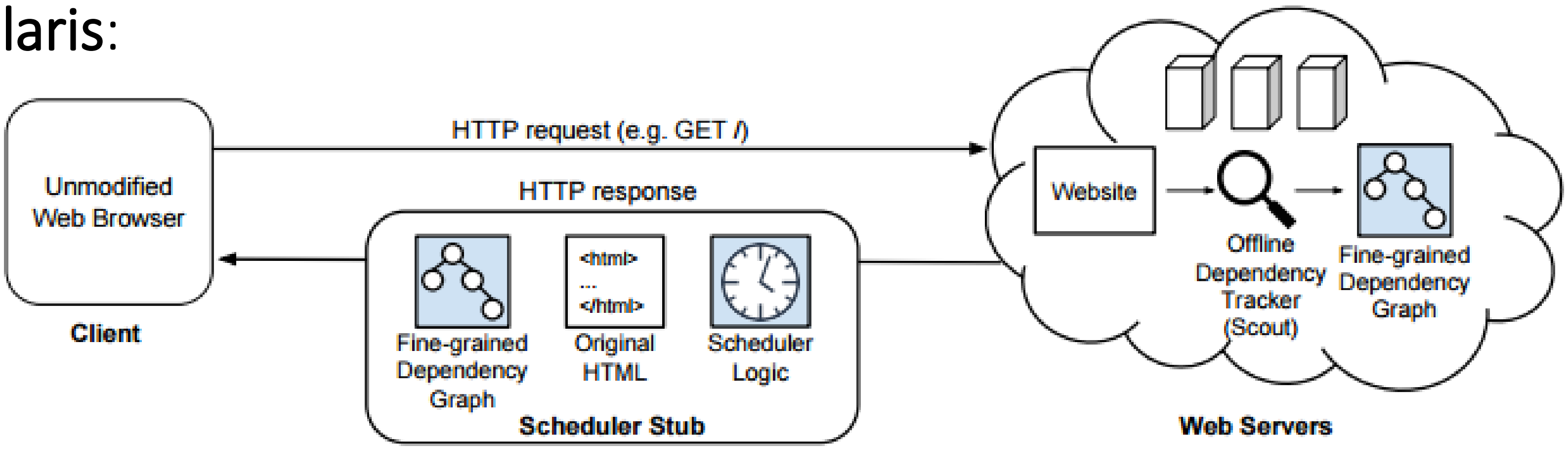
- Stateless session handling
- Minimize shared state
- Efficient Code & IO

- Horizontally scalable databases (e.g. “NoSQL”)
  - Replication
  - Sharding
  - Failover

Research Approaches

Two Examples

Polaris:



Idea: construct graph that captures real read/write and write/write JS/CSS *dependencies*

Improvement: ~30% depending on RTT and bandwidth

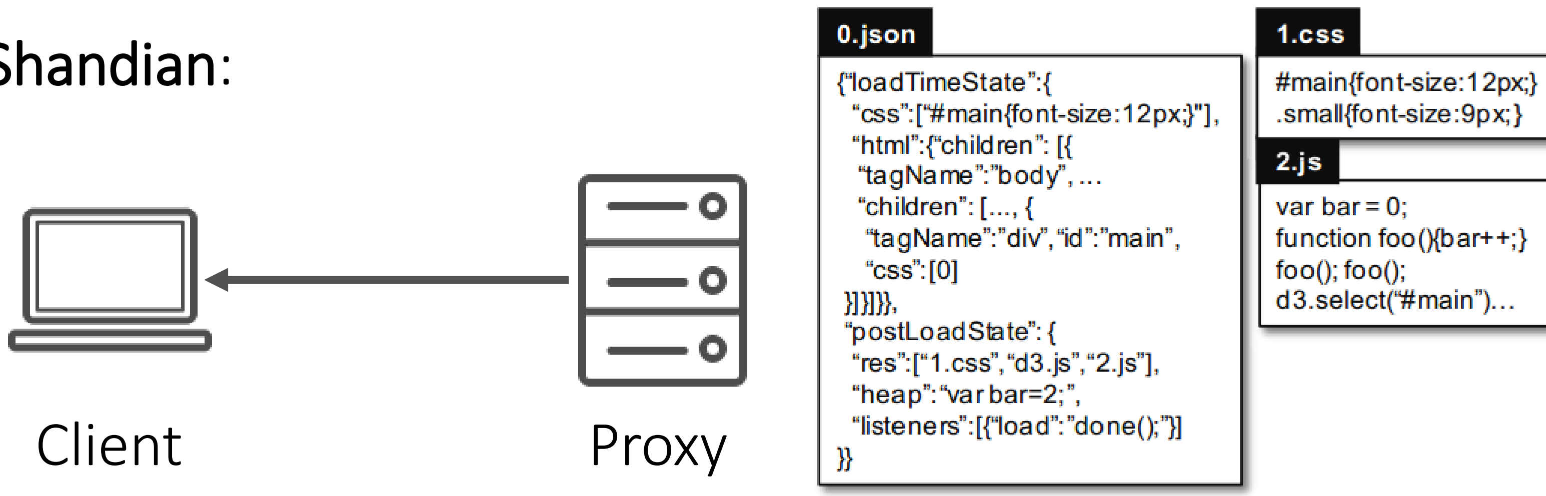
Limitation: cannot deal with *non-determinism*, requires server to generate a dependency graph *for each client view*

Netravali, Ravi, James Mickens, et al. *Polaris: Faster Page Loads Using Fine-grained Dependency Tracking,* NSDI 2016

Research Approaches

Two Examples

Shandian:



Idea: Proxy is more powerful than browser, especially mobile

→ evaluate page on proxy

Improvement: ~50% for slow Android device

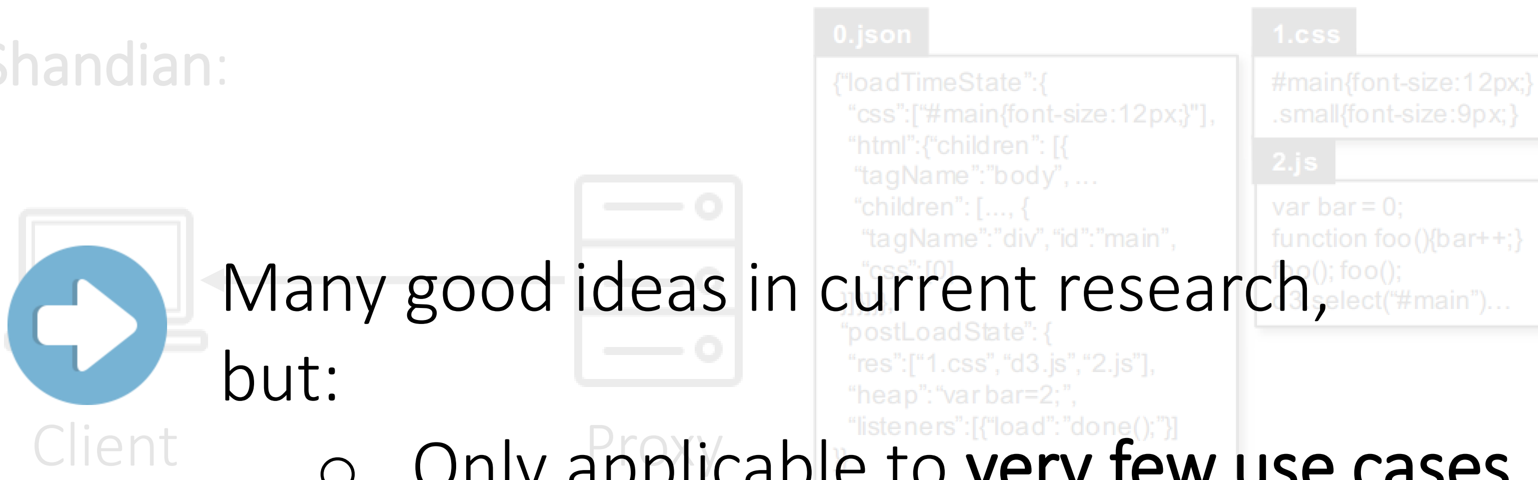
Limitation: needs *modified browser*, only useful for *slow devices*

Wang, Xiao Sophia, Arvind Krishnamurthy, and David Wetherall. "Speeding up Web Page Loads with Shandian." NSDI 2016.

Other Research Approaches

Two Examples

Shandian:



Many good ideas in current research, but:

- Only applicable to **very few use cases**
- Mostly require **modified browsers**
- **Small** performance improvements

Idea: Proxy is more powerful than browser especially mobile -> evaluate page on proxy

Improvement: ~50% for slow Android device

Limitation: needs *modified browser*, only useful for slow devices

Wang, Xiao Sophia, Arvind Krishnamurthy, and David Wetherall. "Speeding up Web Page Loads with Shandian." NSDI 2016.

Performance: State of the Art

Summarized

Frontend

- Doable with the right set of best practices
- Good support through build tools

Latency

- Caching and CDNs help, but a considerable effort and only for static content

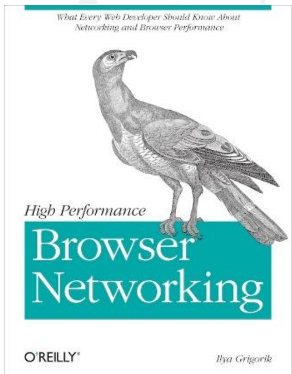
Backend

- Many frameworks and platforms
- Horizontal scalability is very difficult

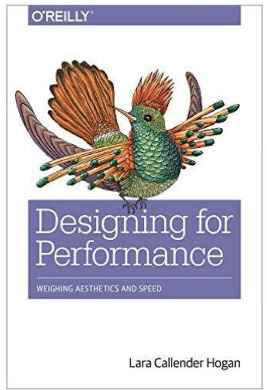
Performance: State of the Art

Summarized

Good Resources:



chimera.labs.oreilly.com/books/1230000000545



shop.oreilly.com/product/0636920033578.do

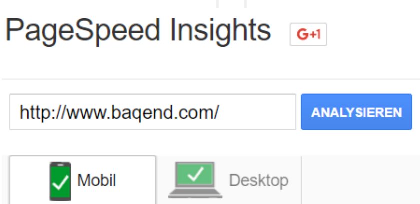
Google Developers Performance

https://developers.google.com/web/fundamentals/performance/?hl=en

Website Performance Optimization
The Critical Rendering Path

https://www.udacity.com/course/website-performance-optimization--ud884

Good Tools:



https://developers.google.com/speed/pagespeed/

GTmetrix

https://gtmetrix.com

WEBPAGETEST

http://www.webpagetest.org/

Performance: State of the Art

Summarized

Frontend

- Doable with the right set of best practices
- Good support through build tools

Latency

- Caching and CDNs help, but large effort and only for static content

Backend

- Many frameworks and platforms
- Horizontal scalability is very difficult

How to cache & scale **dynamic** content?

Introduction

Main Part

Conclusions

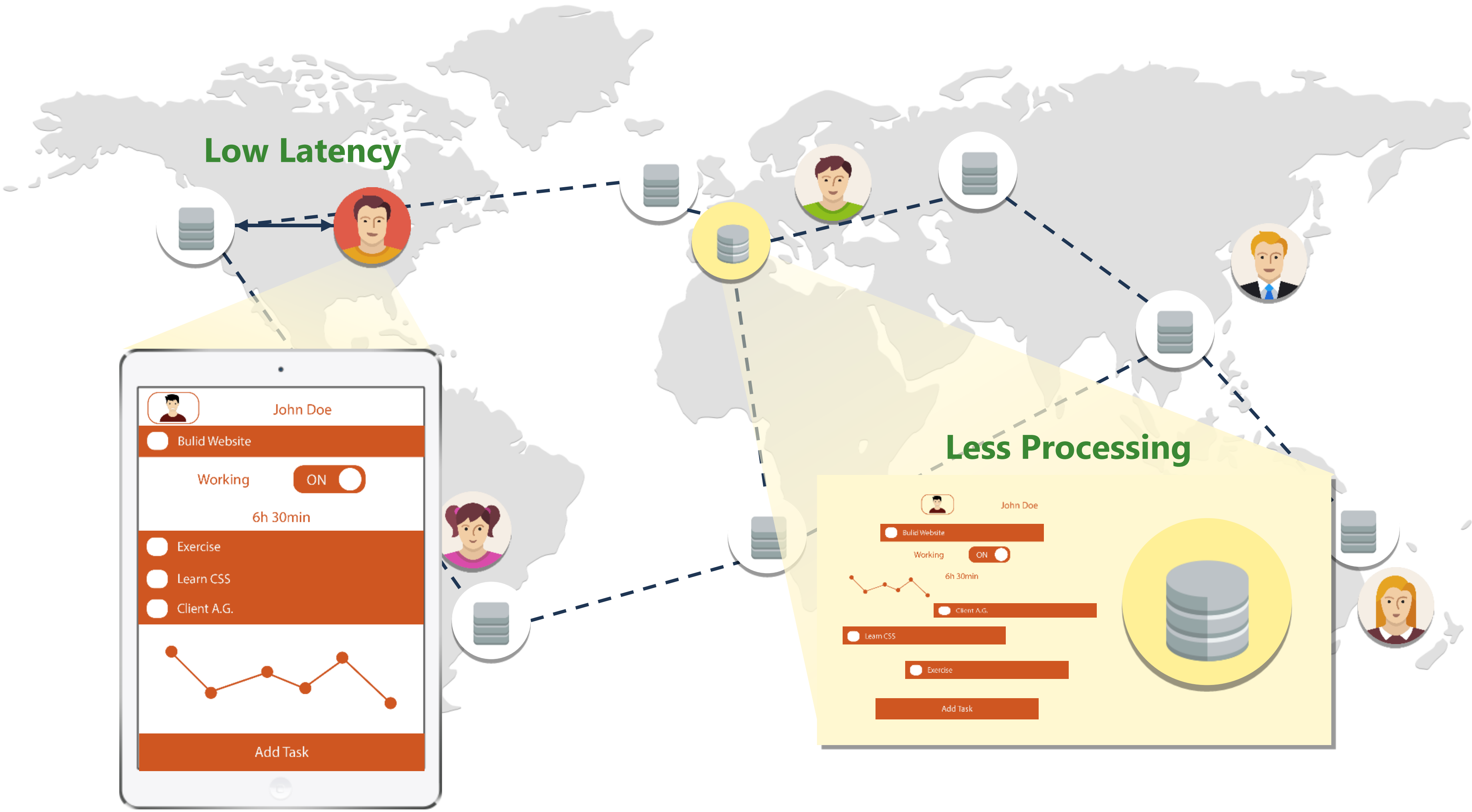
Web Performance:
State of the Art

Cache Sketch:
Research Approach

Using Web Caching in
Applications

Goal: Low-Latency for Dynamic Content

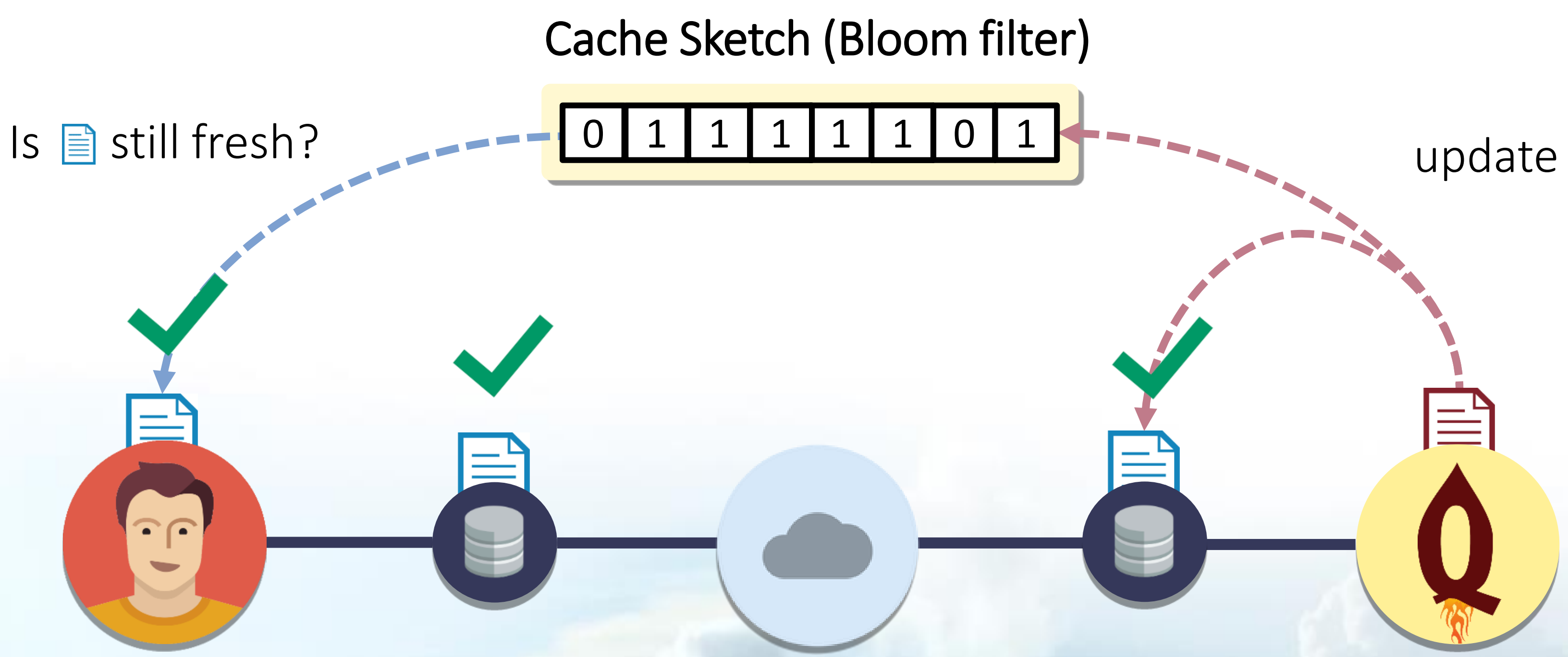
By Serving Data from Ubiquitous Web Caches



In a nutshell

In a nutshell

Solution: Proactively Revalidate Data



Innovation

Solution: Proactively Revalidate Data

F. Gessert, F. Bücklers, und N. Ritter, „ORESTES: a Scalable Database-as-a-Service Architecture for Low Latency“, in *CloudDB 2014*, 2014.

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

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

M. Schaarschmidt, F. Gessert, und N. Ritter, „Towards Automated Polyglot Persistence“, in BTW 2015.

S. Friedrich, W. Wingerath, F. Gessert, und N. Ritter, „NoSQL OLTP Benchmarking: A Survey“, in *44. Jahrestagung der Gesellschaft für Informatik*, 2014, Bd. 232, S. 693–704.

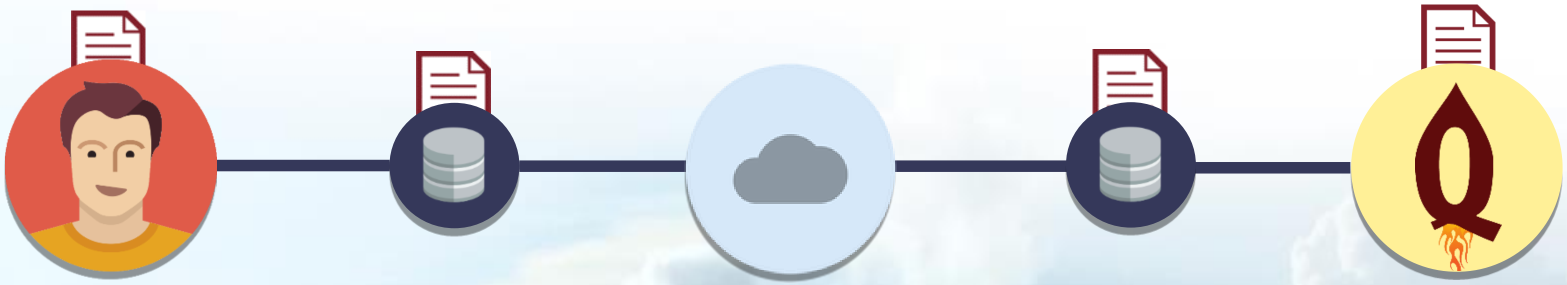
F. Gessert, S. Friedrich, W. Wingerath, M. Schaarschmidt, und N. Ritter, „Towards a Scalable and Unified REST API for Cloud Data Stores“, in *44. Jahrestagung der GI*, Bd. 232, S. 723–734.

F. Gessert, M. Schaarschmidt, W. Wingerath, S. Friedrich, und N. Ritter, „The Cache Sketch: Revisiting Expiration-based Caching in the Age of Cloud Data Management“, in BTW 2015.

F. Gessert und F. Bücklers, *Kohärentes Web-Caching von Datenbankobjekten im Cloud Computing*. Masterarbeit 2012.

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

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



Web Caching Concepts

Invalidation- and expiration-based caches

Expiration-based Caches:

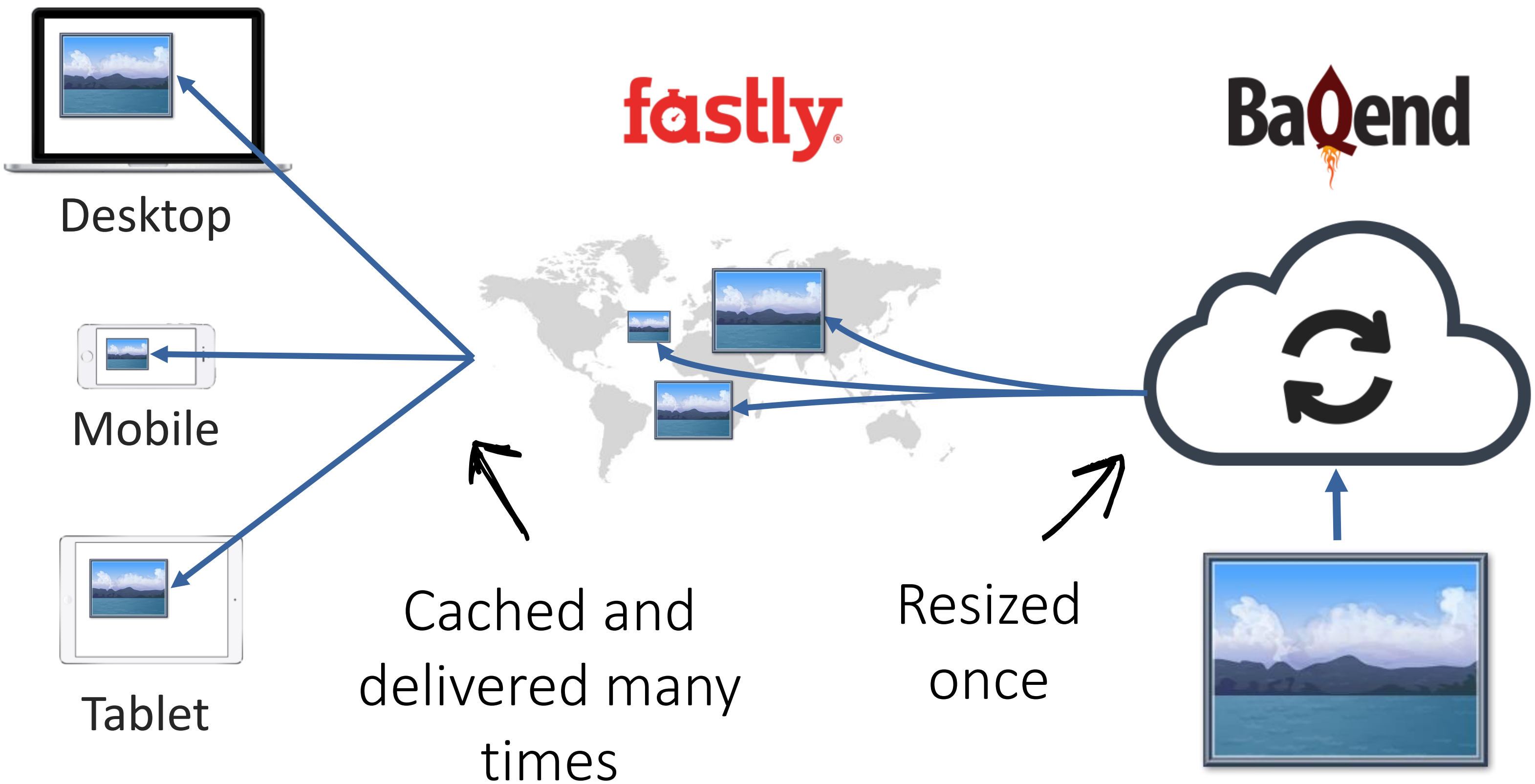
- An object x is considered fresh for TTL_x seconds
- The server assigns TTLs for each object

Invalidation-based Caches:

- Expose object eviction operation to the server

Classic Web Caching: Example

A tiny image resizer



Bloom filter Concepts

Compact Probabilistic Sets

- The „Bloom filter principle“:
  “Wherever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated.”

A. Broder und M. Mitzenmacher, „Network applications of bloom filters: A survey“, Internet Mathematics, 2004.

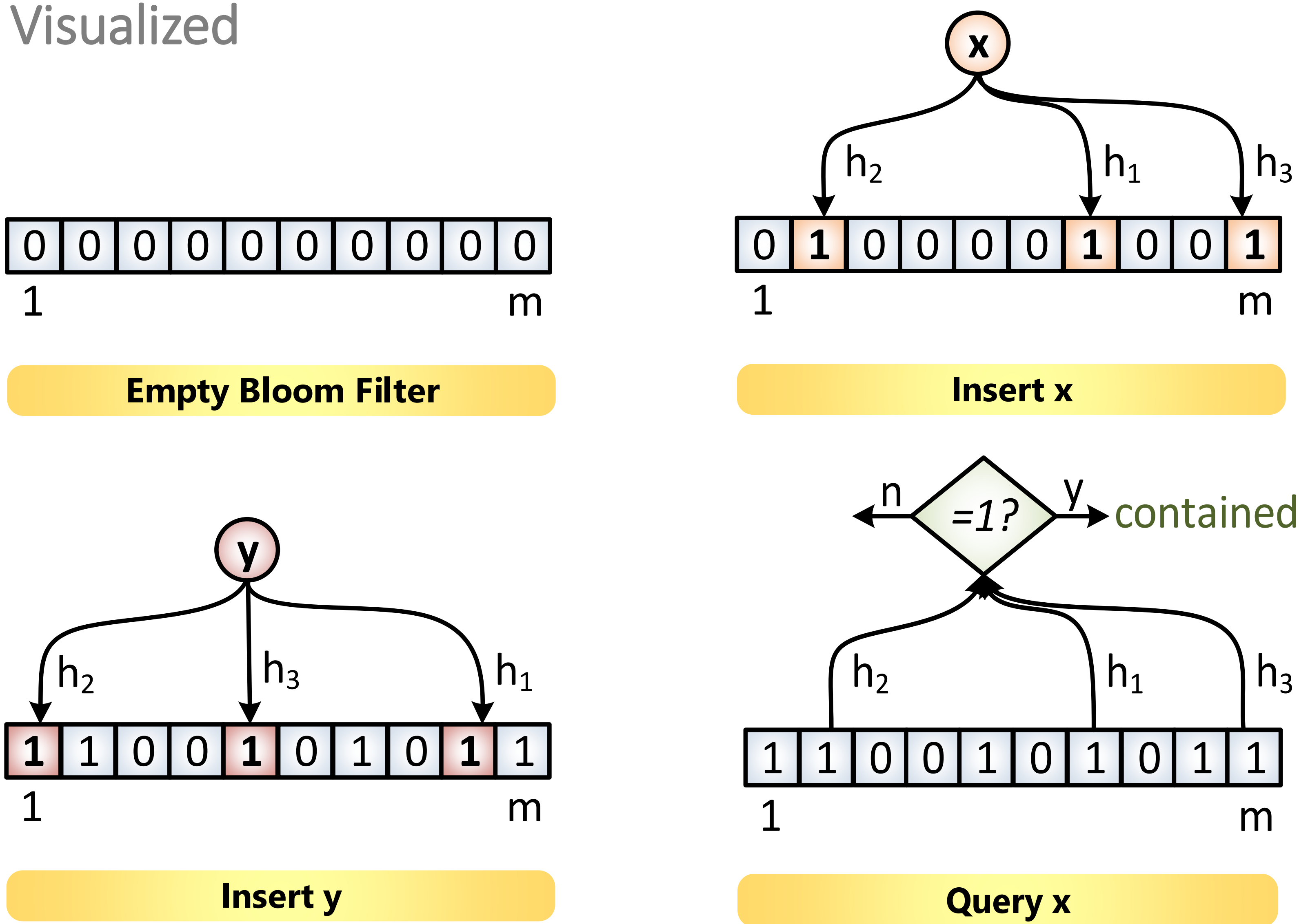
- Bit array of length **m**
- **k** independent hash functions
- **insert(obj)**: add to set
- **contains(obj)**:
  - Always returns true if the element was inserted
  - Might return true even though it was *not* inserted (**false positive**)

```
def insert(obj):
  for each position in hashes(obj):
    bits[position] = 1

def contains(obj):
  for each position in hashes(obj):
    if bits[position] == 0:
    return false;
  return true
```

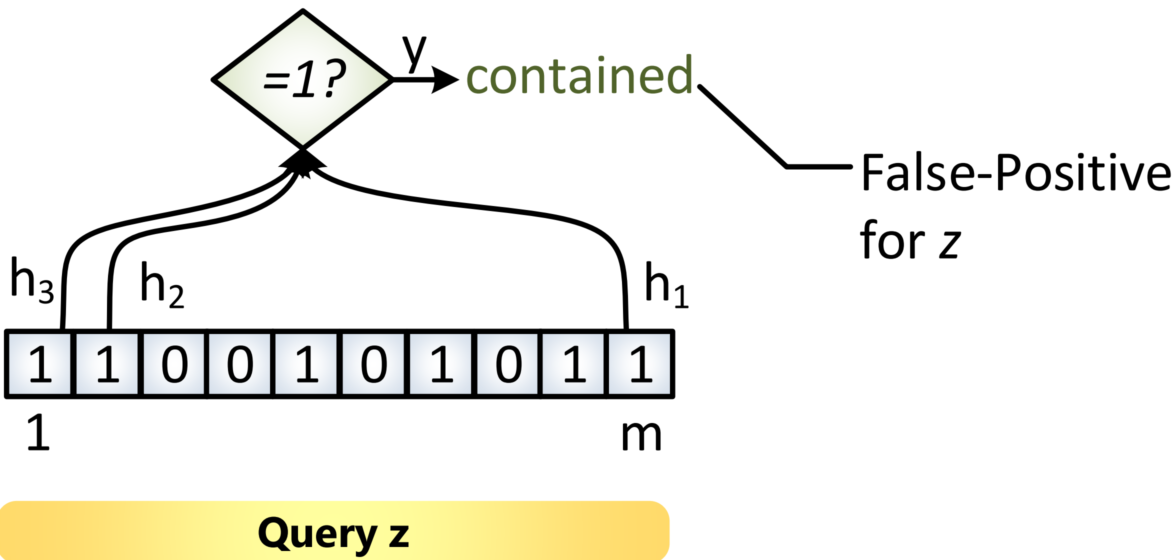
Bloom filter Concepts

Visualized



Bloom filter Concepts

False Positives



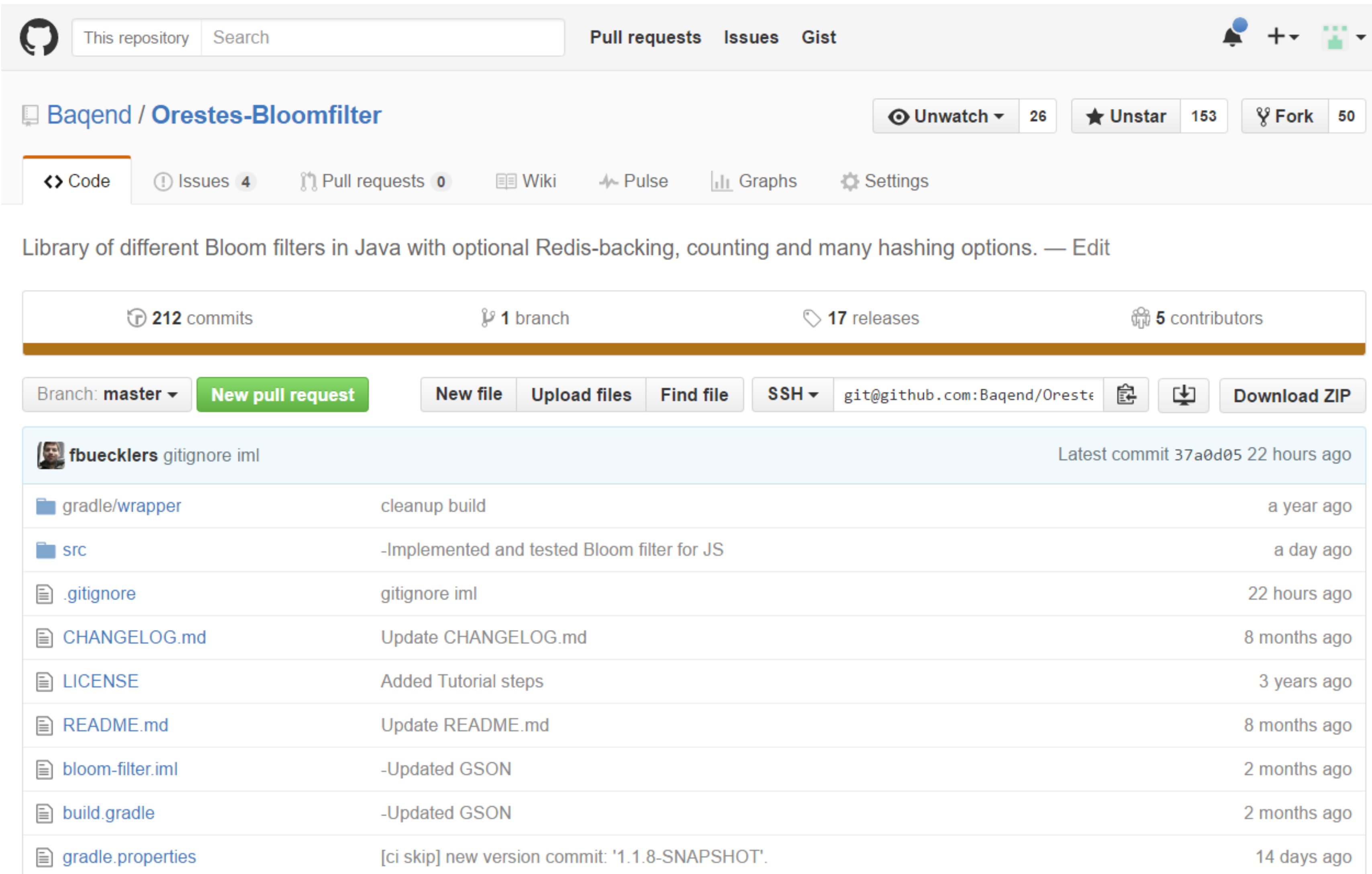
The false positive rate depends on the bits **m** and the inserted elements **n**:

$$f \approx \left(1 - e^{-\ln(2)}\right)^k \approx 0.6185^{\frac{m}{n}}$$

For ***f=1%*** the required bits per element are: *2.081 ln(1/0.01) =* ***9.5***

Our Bloom filter

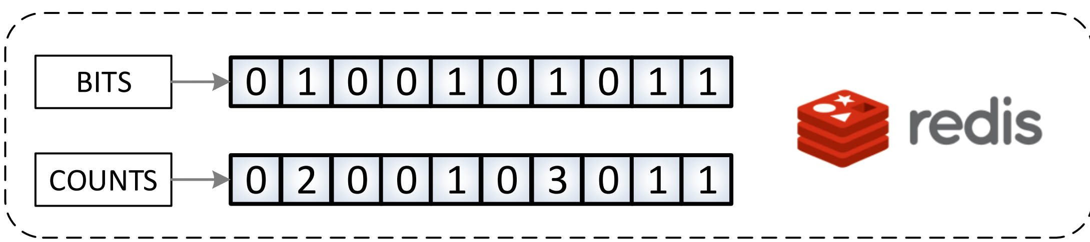
Open Source Implementation



Our Bloom filters

Example: Redis-backed Counting Bloom Filter

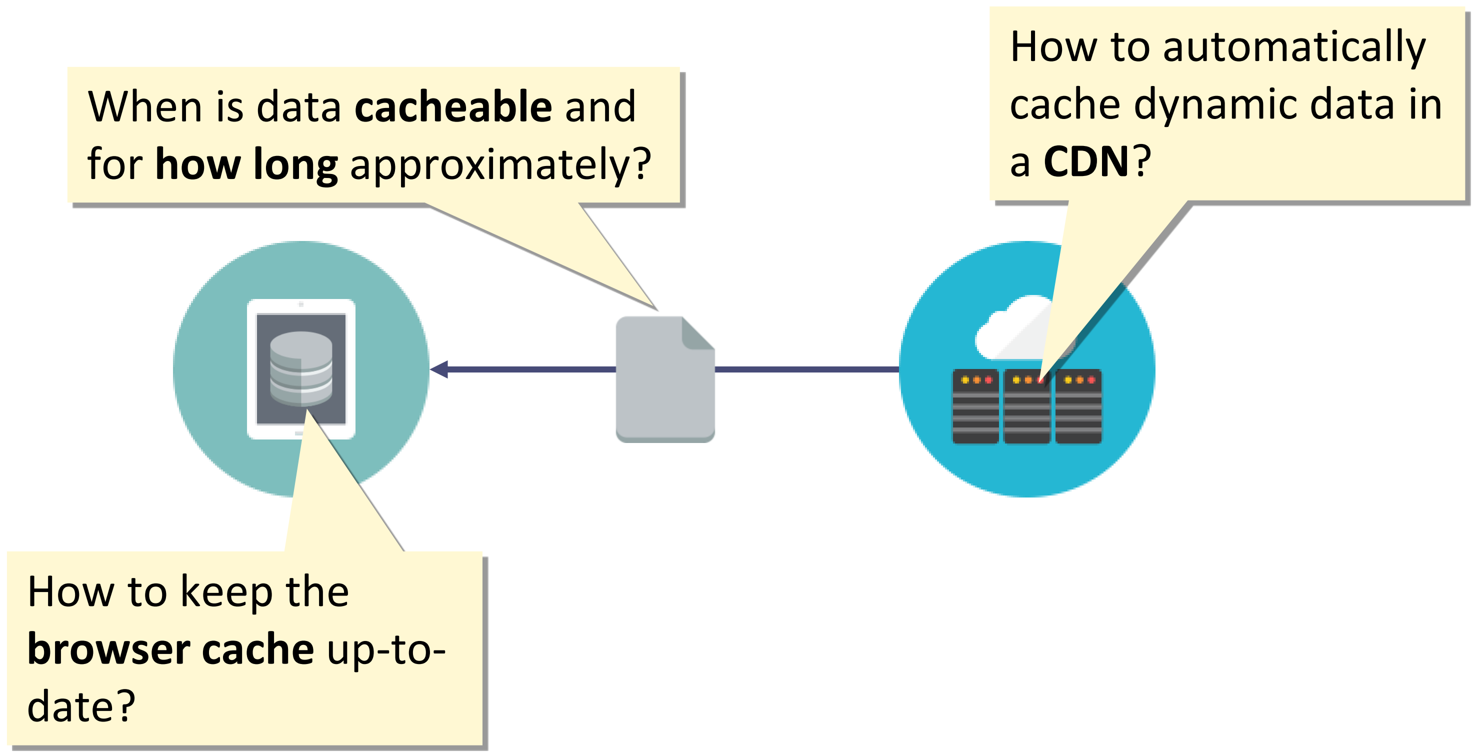
- Redis-backed Bloom filters:
  - Can be **shared** by many servers
  - Highly **efficient** through Redis' bitwise operations
  - Tunable **persistence**
- Counting Bloom Filters: use counters instead of bits to also allow **removals**
  - Stores the materialized Bloom filter for fast retrieval



The Cache Sketch approach

Caching Dynamic Data

- **Idea**: use standard *HTTP Caching* for query results and records
- **Problems**:



Orestes Architecture

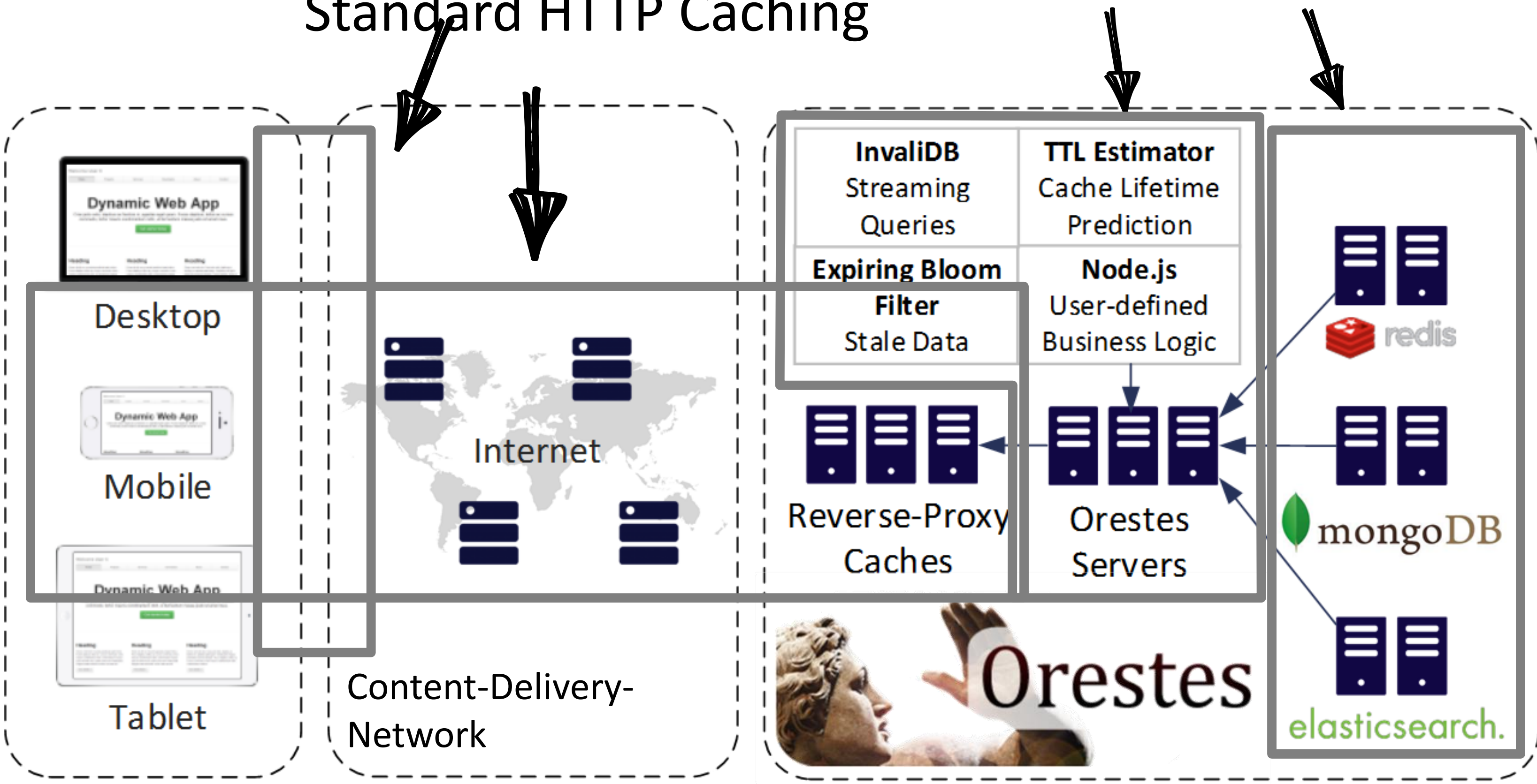
Infrastructure

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

Polyglot Storage

Unified REST API

Standard HTTP Caching

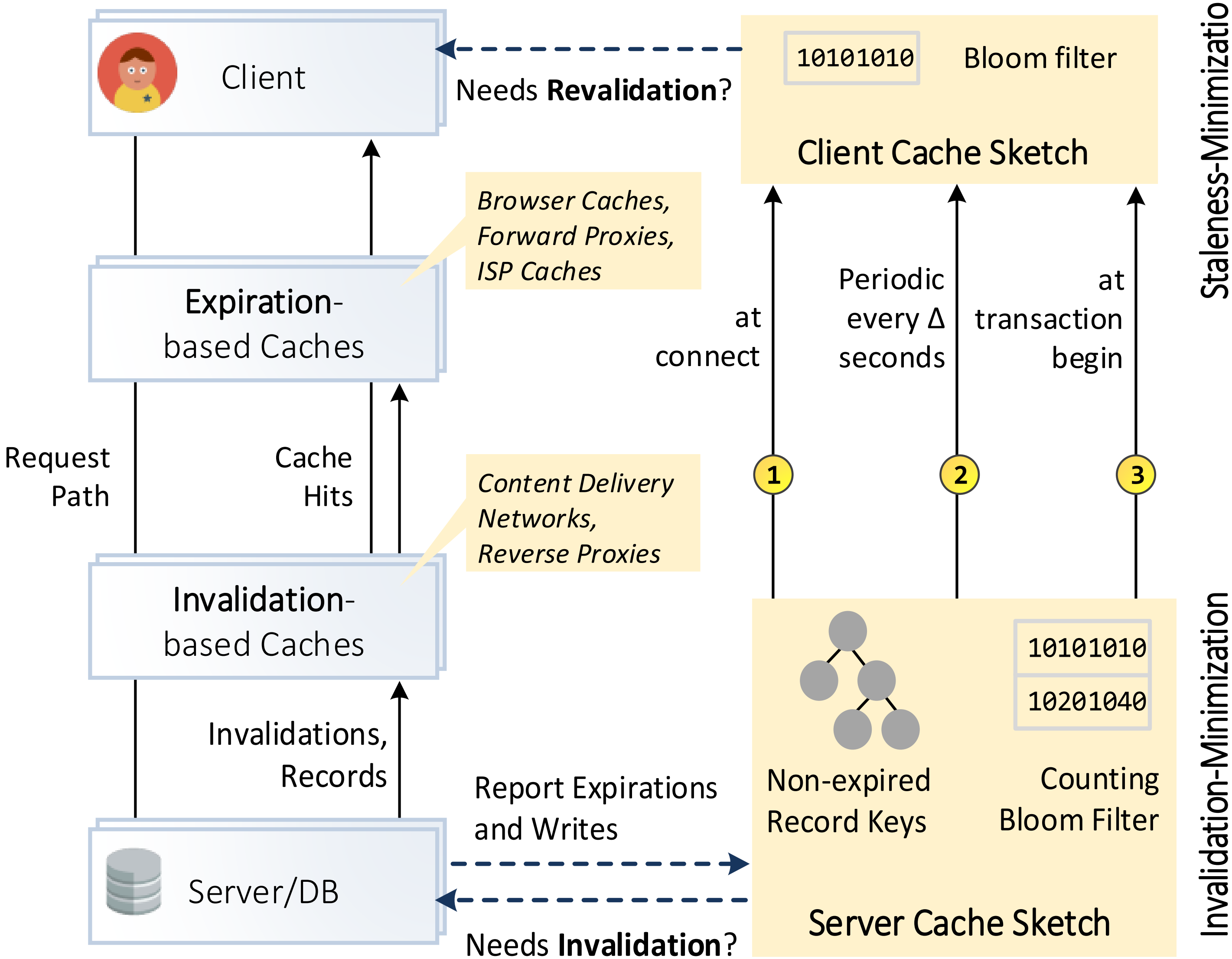


Baqend Architecture

Infrastructure

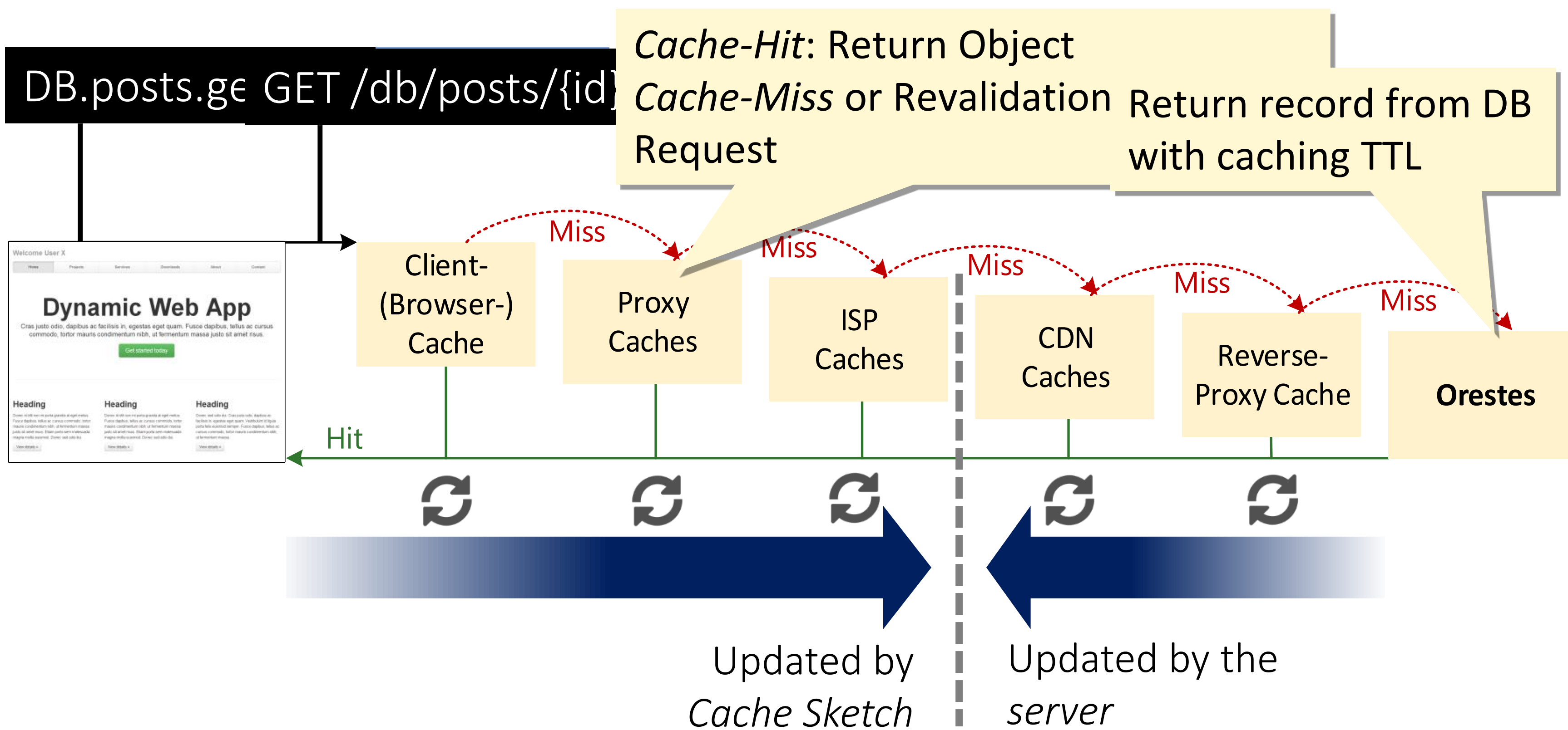
The Cache Sketch approach

Letting the client handle cache coherence



The End to End Path of Requests

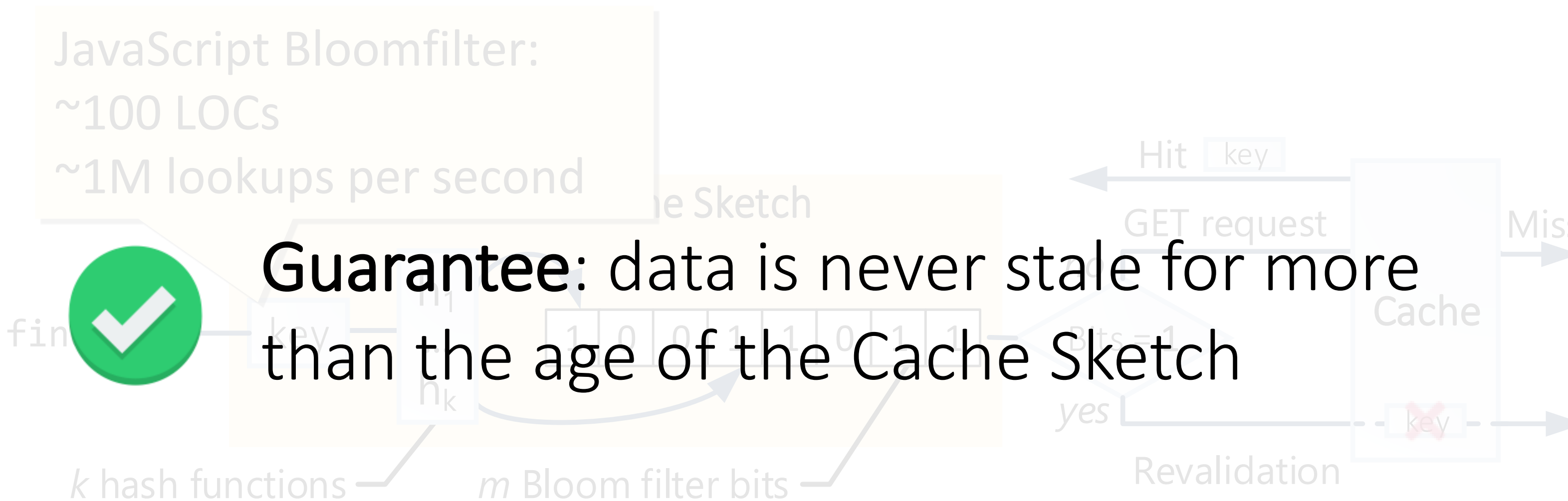
The Caching Hierarchy



The Client Cache Sketch

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

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



Guarantee: data is never stale for more than the age of the Cache Sketch

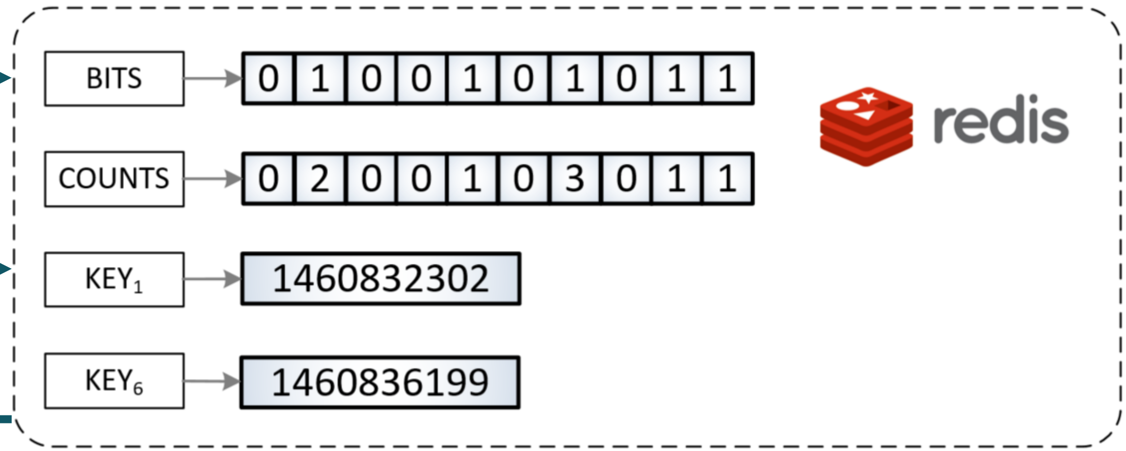
The Server Cache Sketch

Scalable Implementation

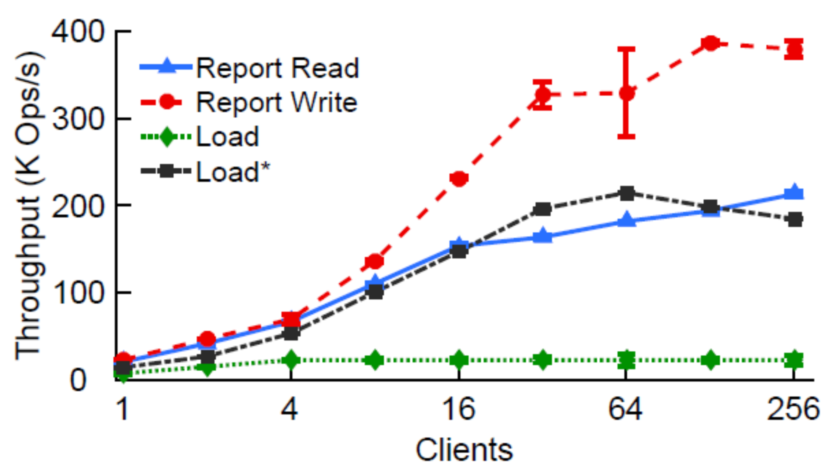
Add *key_x* if *x* unexpired and write occured

Remove x from Blom filter when expired

Load Bloom filter



Performance > 200k ops per second:



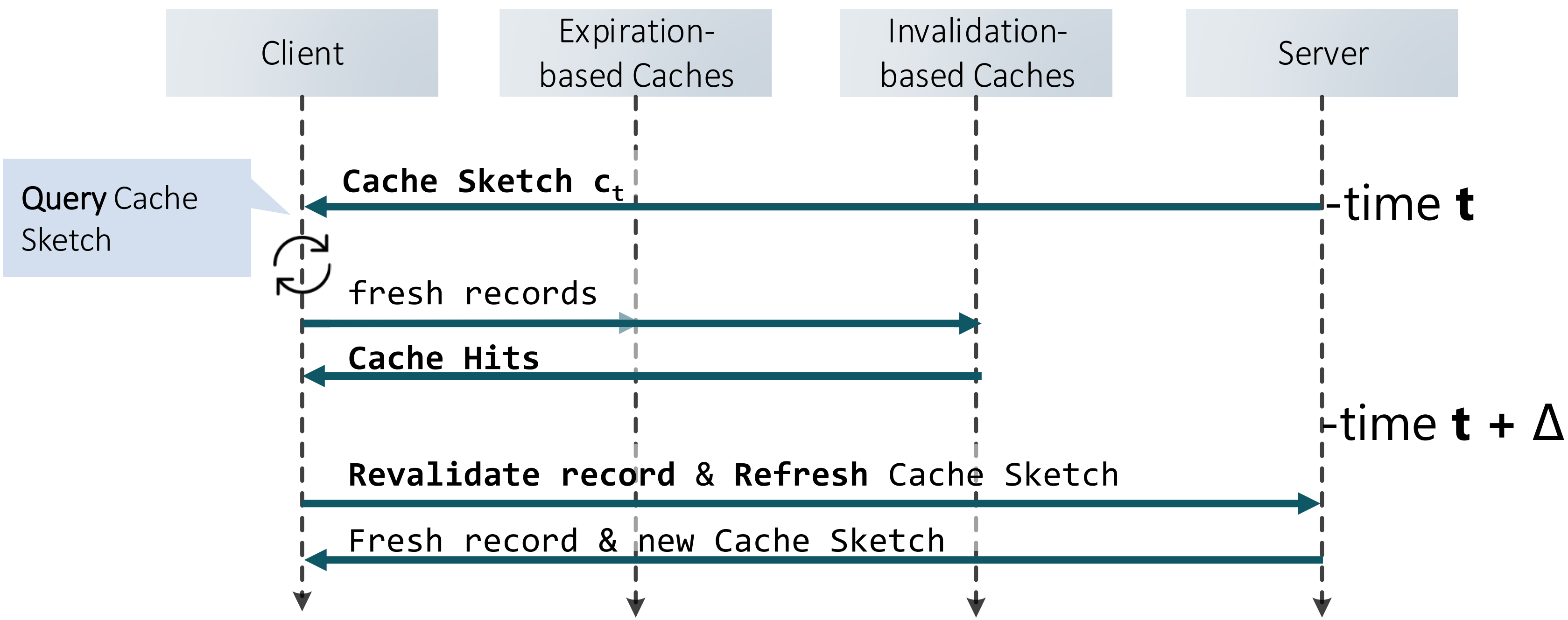
1 Faster Page Loads

- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency

2 Faster CRUD Performance

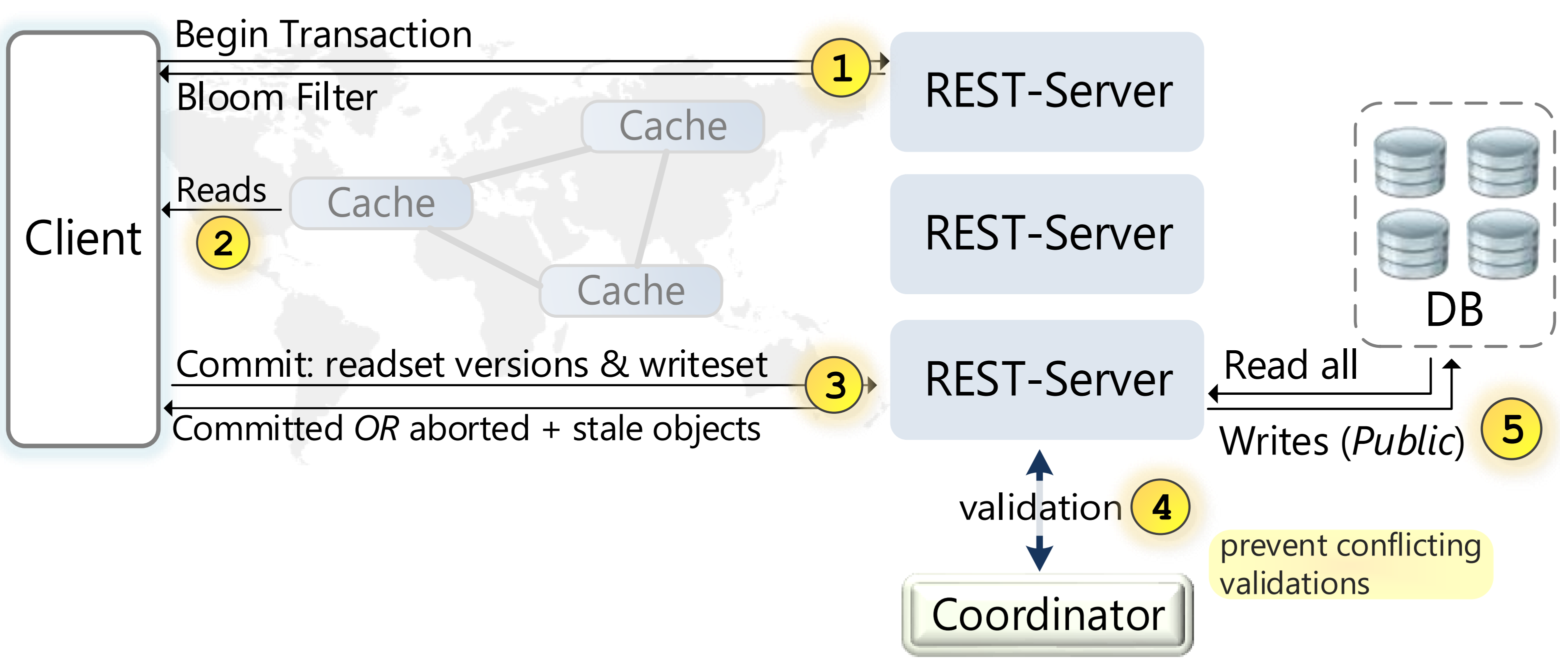
- Solution: **Δ-Bounded Staleness**
  - Clients refresh the Cache Sketch so its age never exceeds Δ

  → *Consistency guarantee*: Δ-atomicity



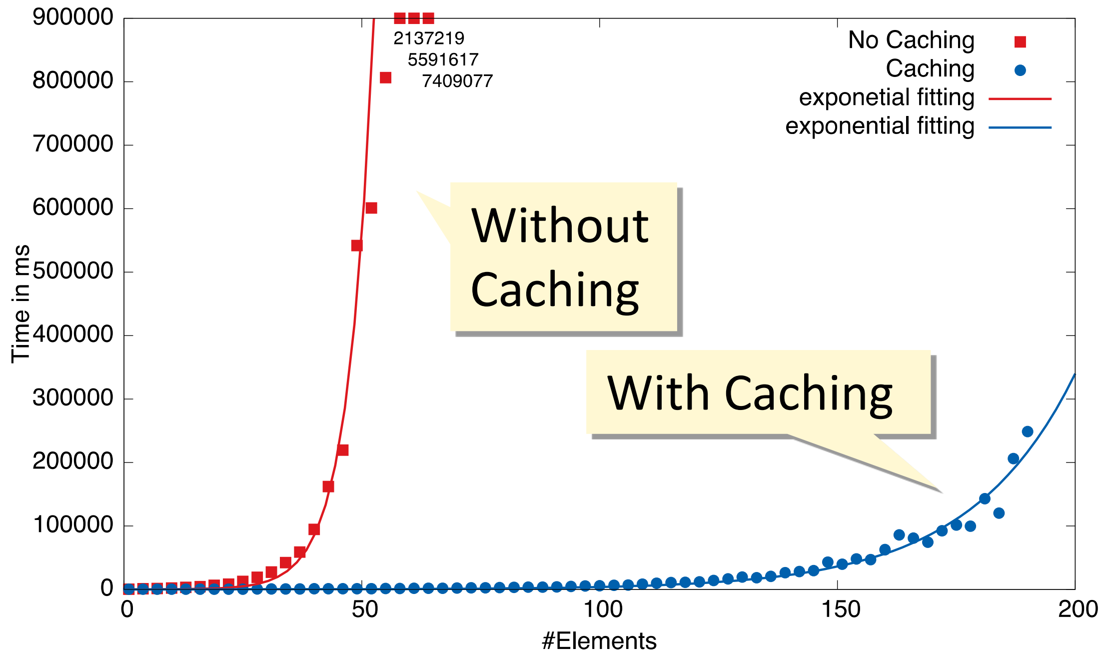
3 Scalable ACID Transcations

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



3 Scalable ACID Transcations

- **Novelty:** ACID transactions on sharded DBs like MongoDB



- Current Work: **DESY** and **dCache** building a scalable namespace for their file system on this

TTL Estimation

Determining the best TTL and cacheability

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

Shorter TTLs

- less invalidations
- less stale reads

Longer TTLs

- Higher cache-hit rates
- more invalidations

TTL Estimation

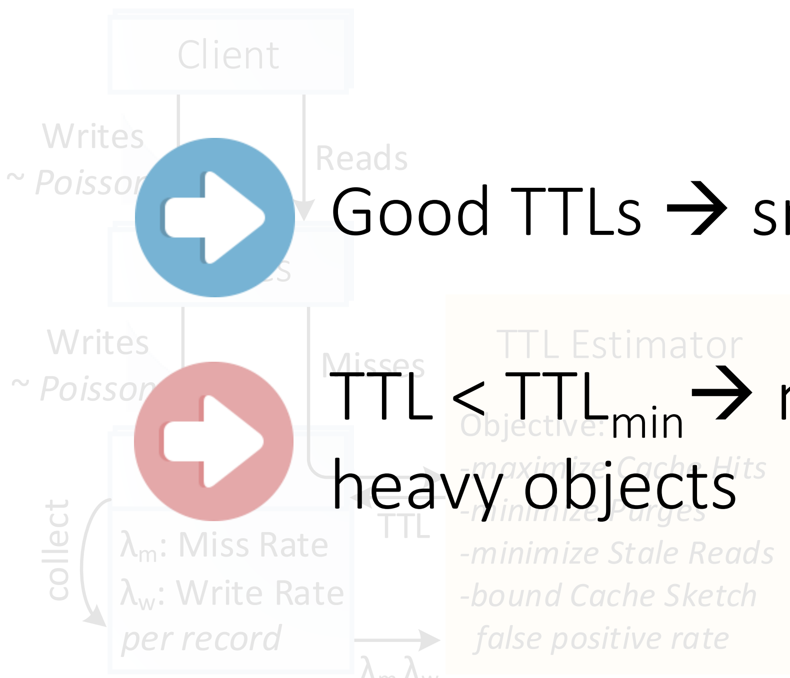
Determining the best TTL

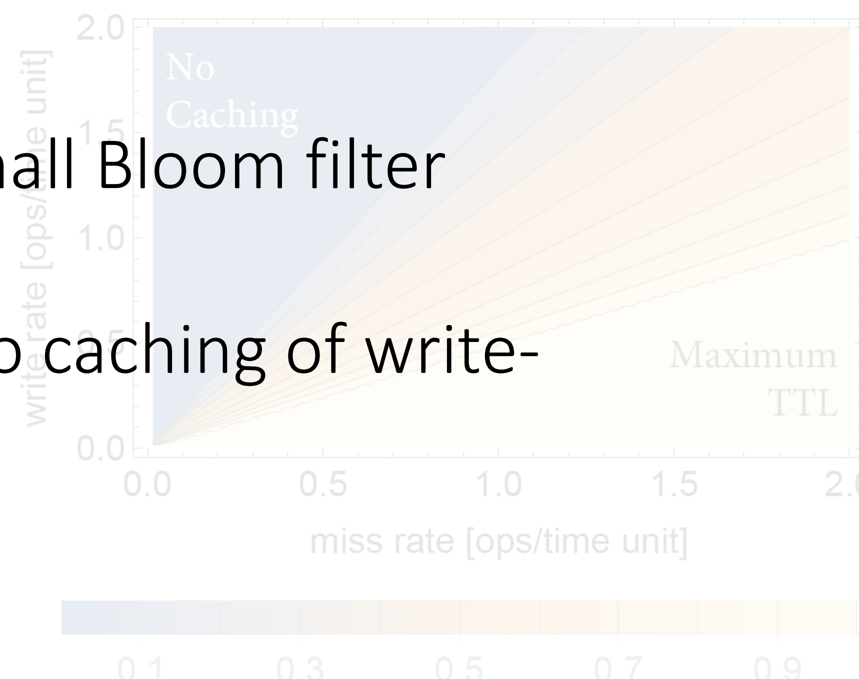
Idea:

1. Estimate average time to next write $E[T_w]$ for each record
2. Weight $E[T_w]$ using the cache miss rate

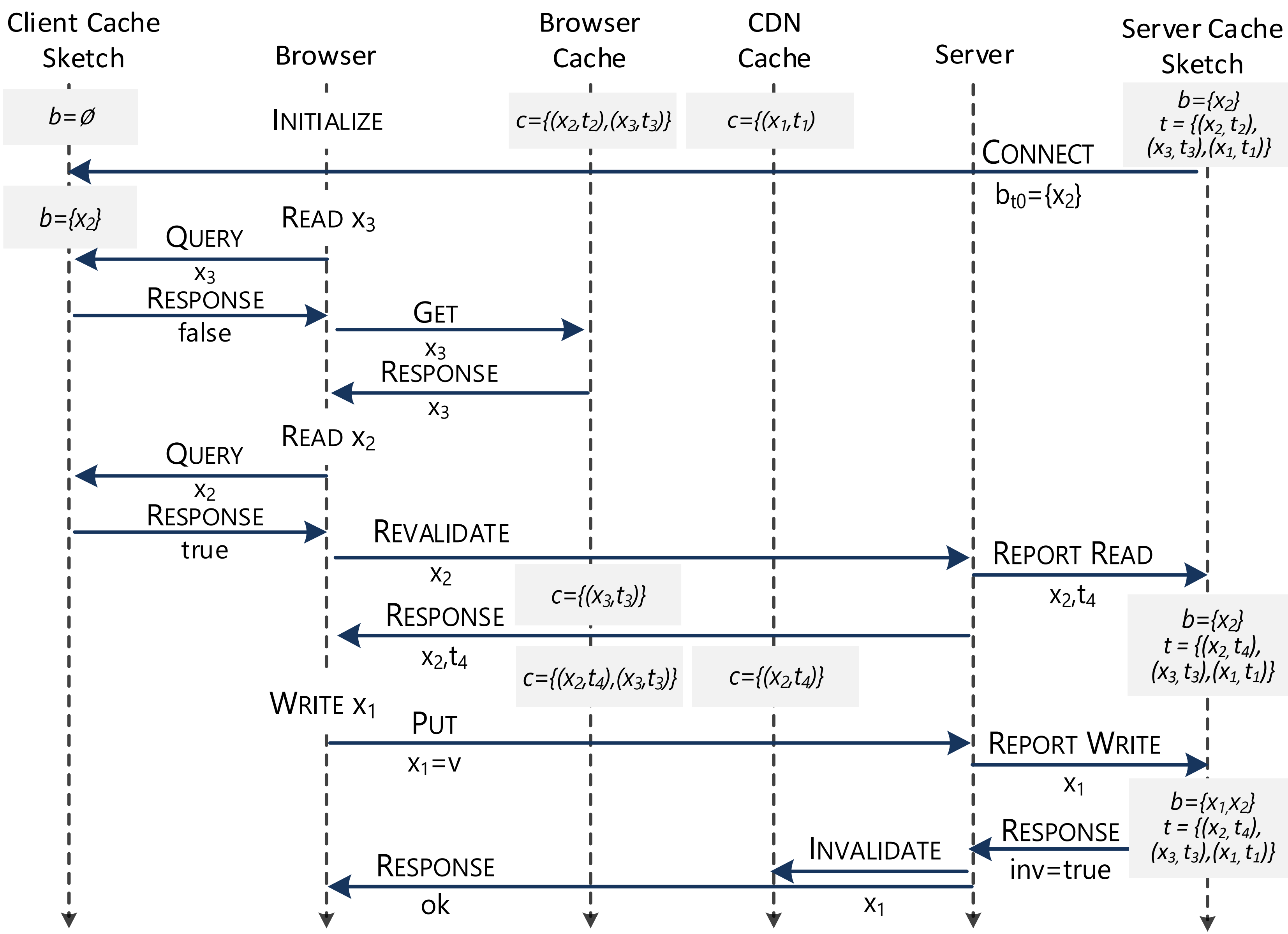
Good TTLs → small Bloom filter

TTL < TTL_{min} → no caching of write-heavy objects





End-to-End Example



Consistency

What are the guarantees?

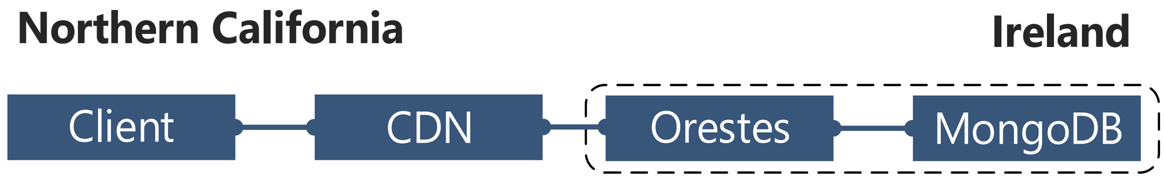
Consistency Level	How
Δ-atomicity (staleness never exceeds Δ seconds)	Controlled by age of Cache Sketch
Montonic Writes	Guaranteed by database
Read-Your-Writes and **Montonic Reads**	Cache written data and most recent read-versions in client
Causal Consistency	If read timestamp is older than Cache Sketch it is given, else revalidate
Strong Consistency (Linearizability)	Explicit revalidation (cache miss at all levels)

Always

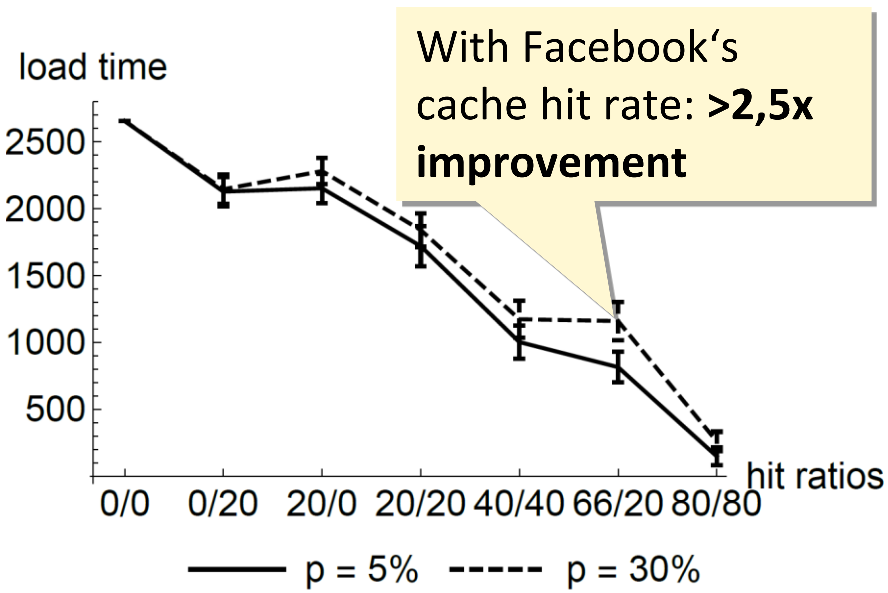
Opt-in

Performance

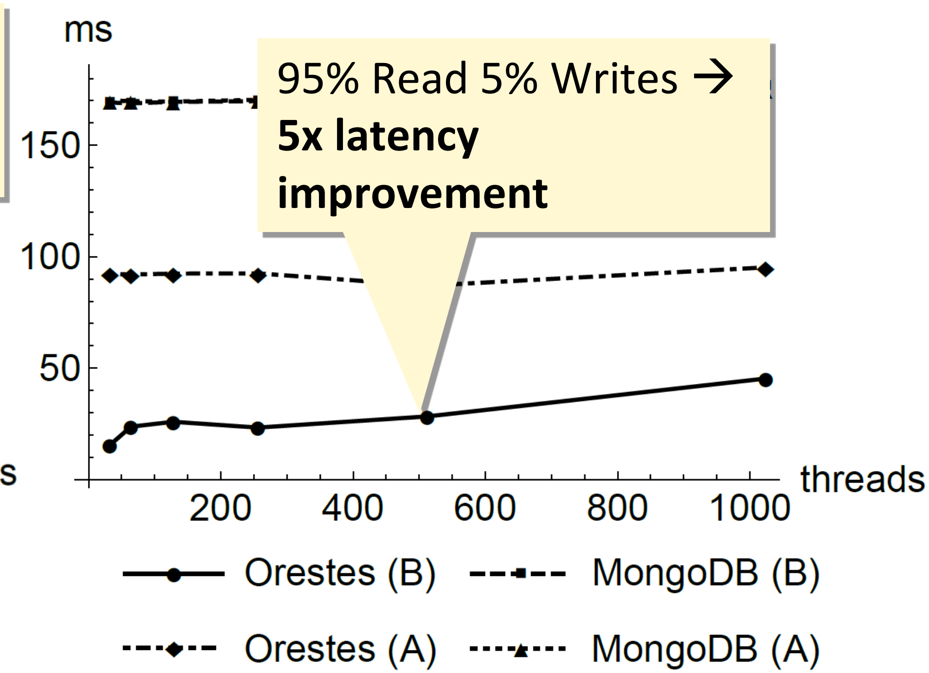
Setup:



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



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



Varnish and Fastly

What we do on the edge

Reject rate limited users

Cache all GET requests

Handle CORS pre-flight requests

Validate & renew session tokens of users

Collect access logs & report failures

Authorize the user on protected resources

The Cache Sketch

Summary

Static Data

Immutability ideal for static web caching:

`max-age=31557600`

Mutable Objects

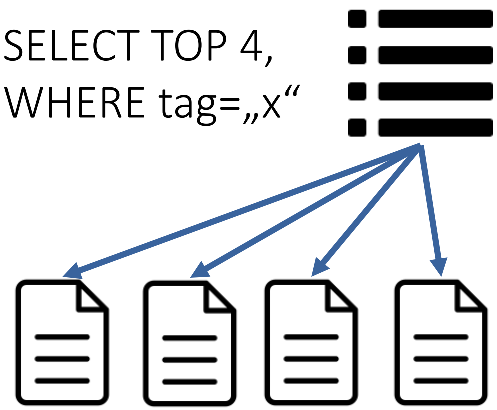

```
{
   "id":"/db/Todo/b5d9bef9-
6c1f-46a5-…",
   "version":1,
   "acl":null,
   "listId":"7b92c069-…",
   "name":"Test",
   "activities":[],
   "active":true,
   "done":false
}
```


Cache Sketch for browser cache, proxies and ISP caches

Invalidations for CDNs and reverse proxies

Queries/Aggregates

SELECT TOP 4,
WHERE tag=„x“



How to do this?

Continuous Query Matching

Generalizing the Cache Sketch to query results

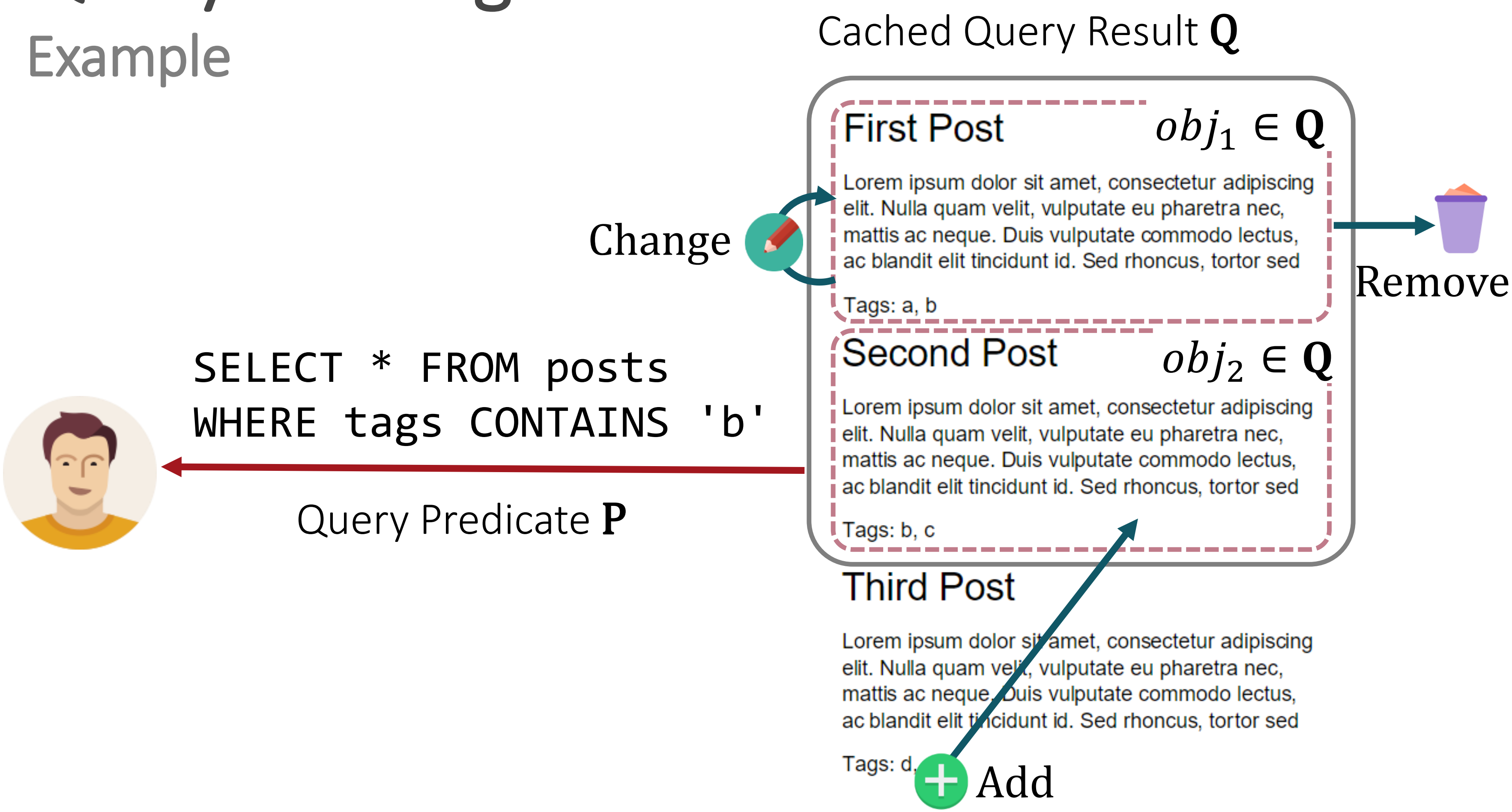
Main challenge: when to invalidate?

- **Objects**: for every update and delete
- **Queries**: as soon as the query result changes

How to detect query **result changes in real-time**?

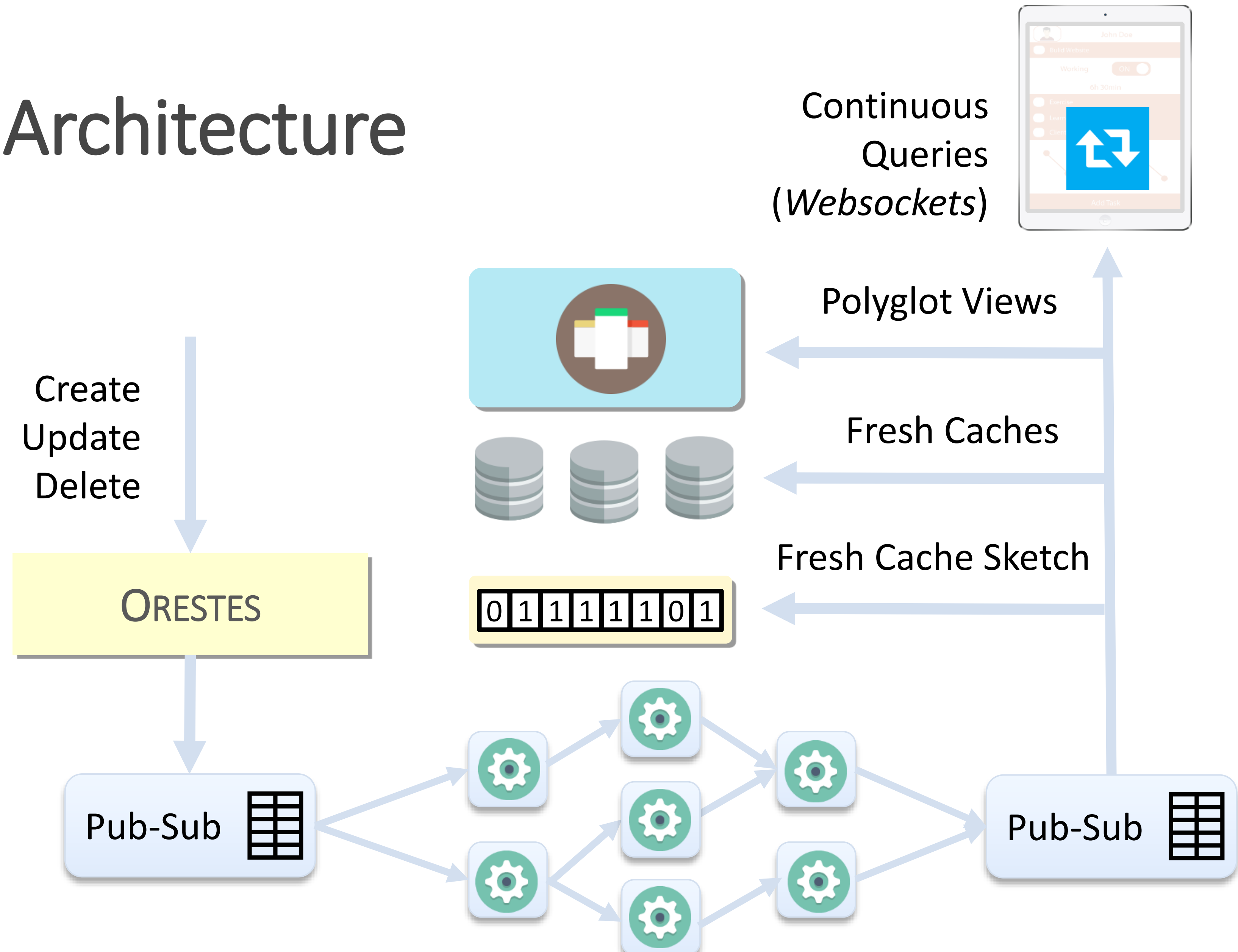
Query Caching

Example



- **Add, Change, Remove** all entail an invalidation and addition to the cache sketch

Architecture



Felix Gessert, Michael Schaarschmidt, Wolfram Wingerath, Steffen Friedrich, Norbert Ritter: *Quaestor: Scalable and Fresh Query Caching on the Web's Infrastructure*. Under Submission.

InvaliDB

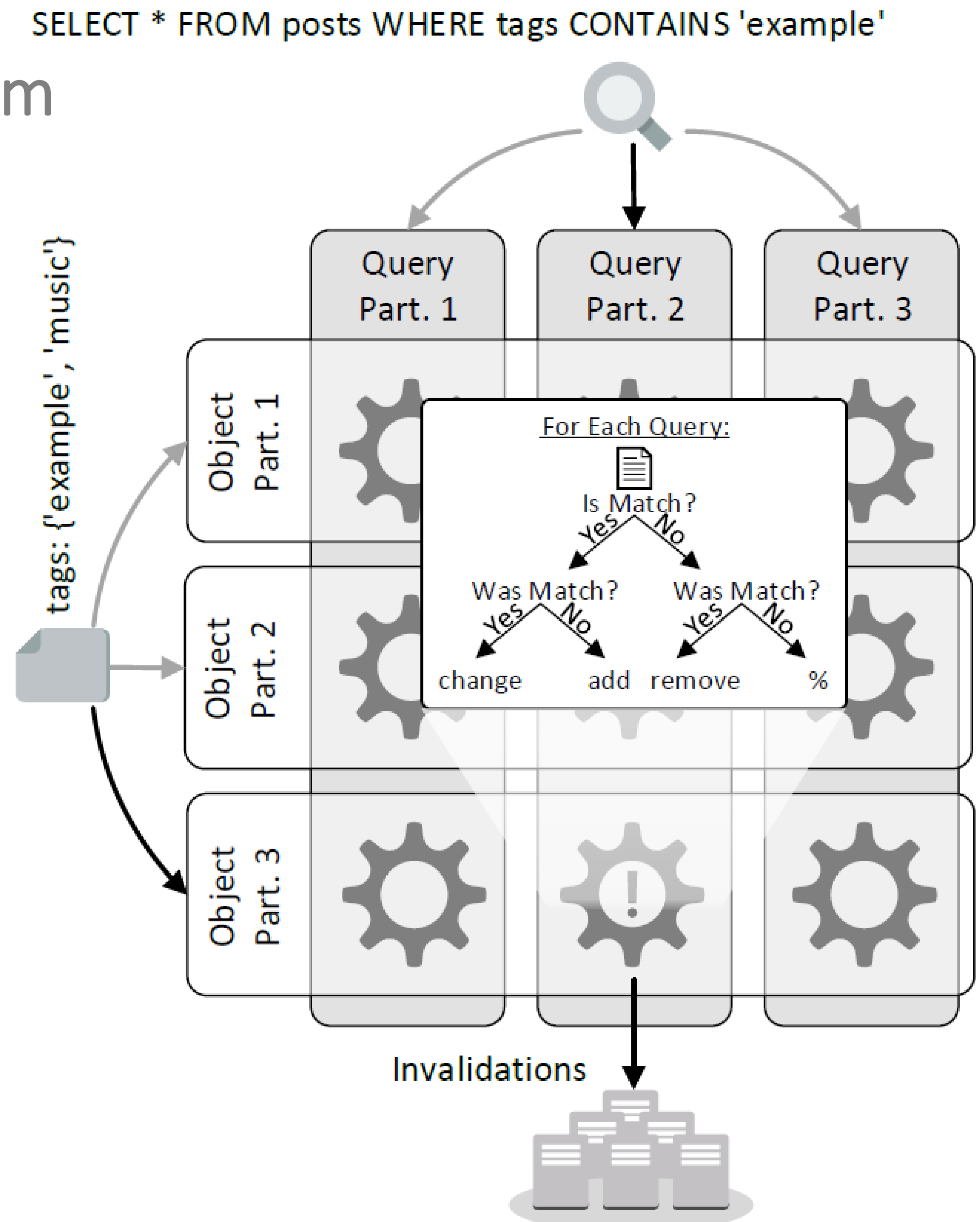
Matching on Apache Storm

Apache Storm:

- **„Hadoop of Real-Time“**
- Low-Latency **Stream Processing**
- Custom Java-based **Topologies**

InvaliDB goals:

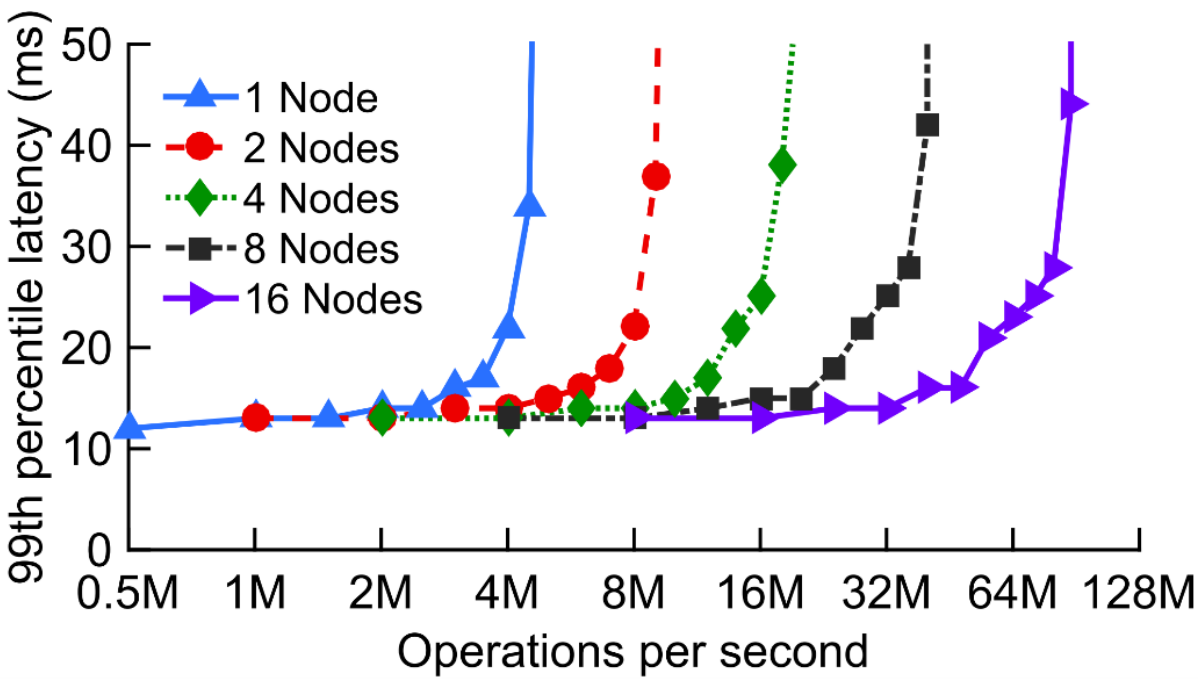
- Scalability, Elasticity, Low latency, Fault-tolerance



Query Matching Performance

Latency of detecting invalidations

- Latency mostly < 15ms, scales linearly w.r.t. number of servers and number of tables



Learning Representations

Determining Optimal TTLs and Cacheability

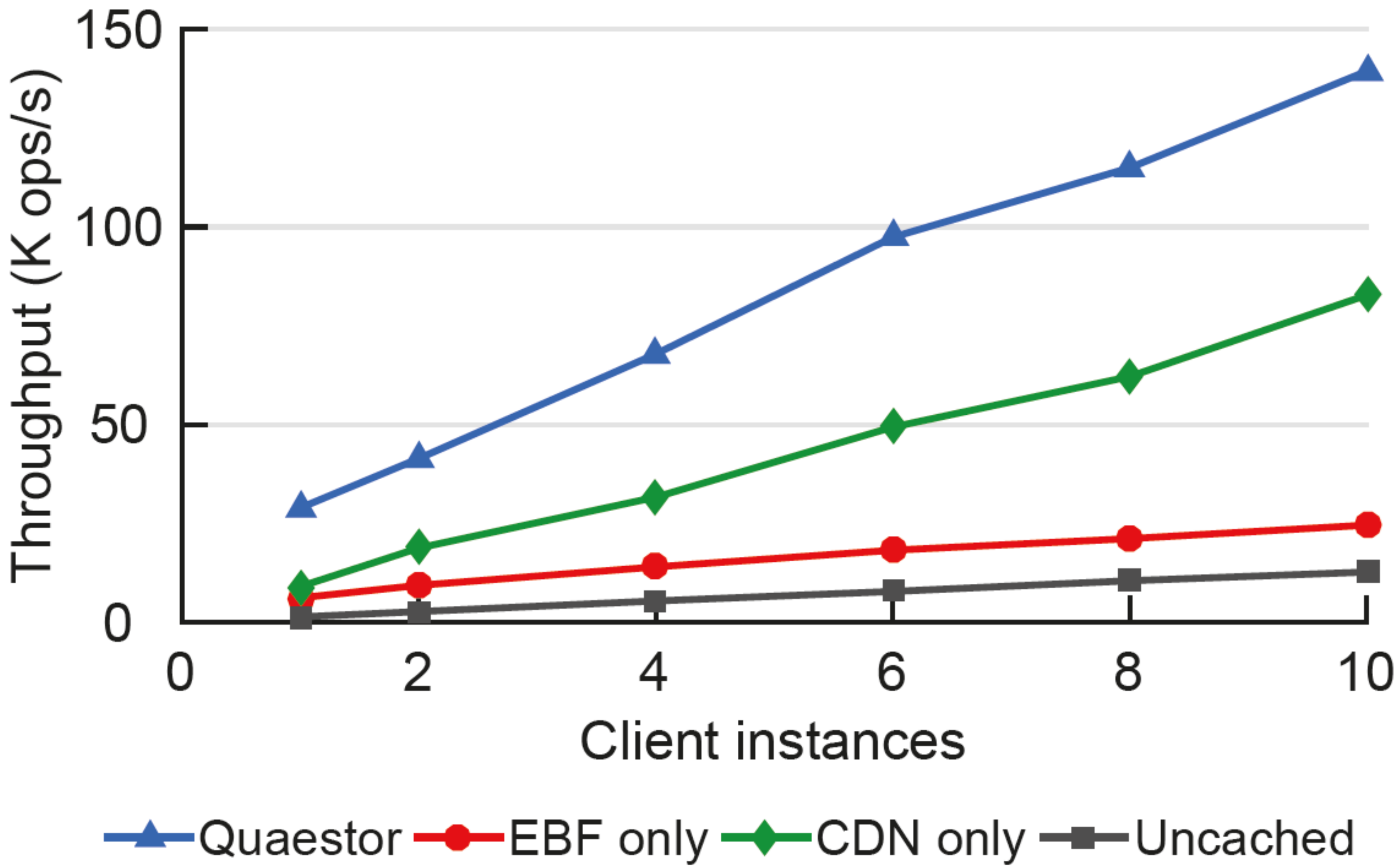
Setting: query results can either be represented as references (id-list) or full results (object-lists)

Id-Lists	Object-Lists
$[id_1, id_2, id_3]$	$[\{id{:}\,1, val{:}\,'a'\}, \{id{:}\,2, val{:}\,'b'\}, \{id{:}\,3, val{:}\,'c'\}]$
Less *Invalidations*	Less *Round-Trips*

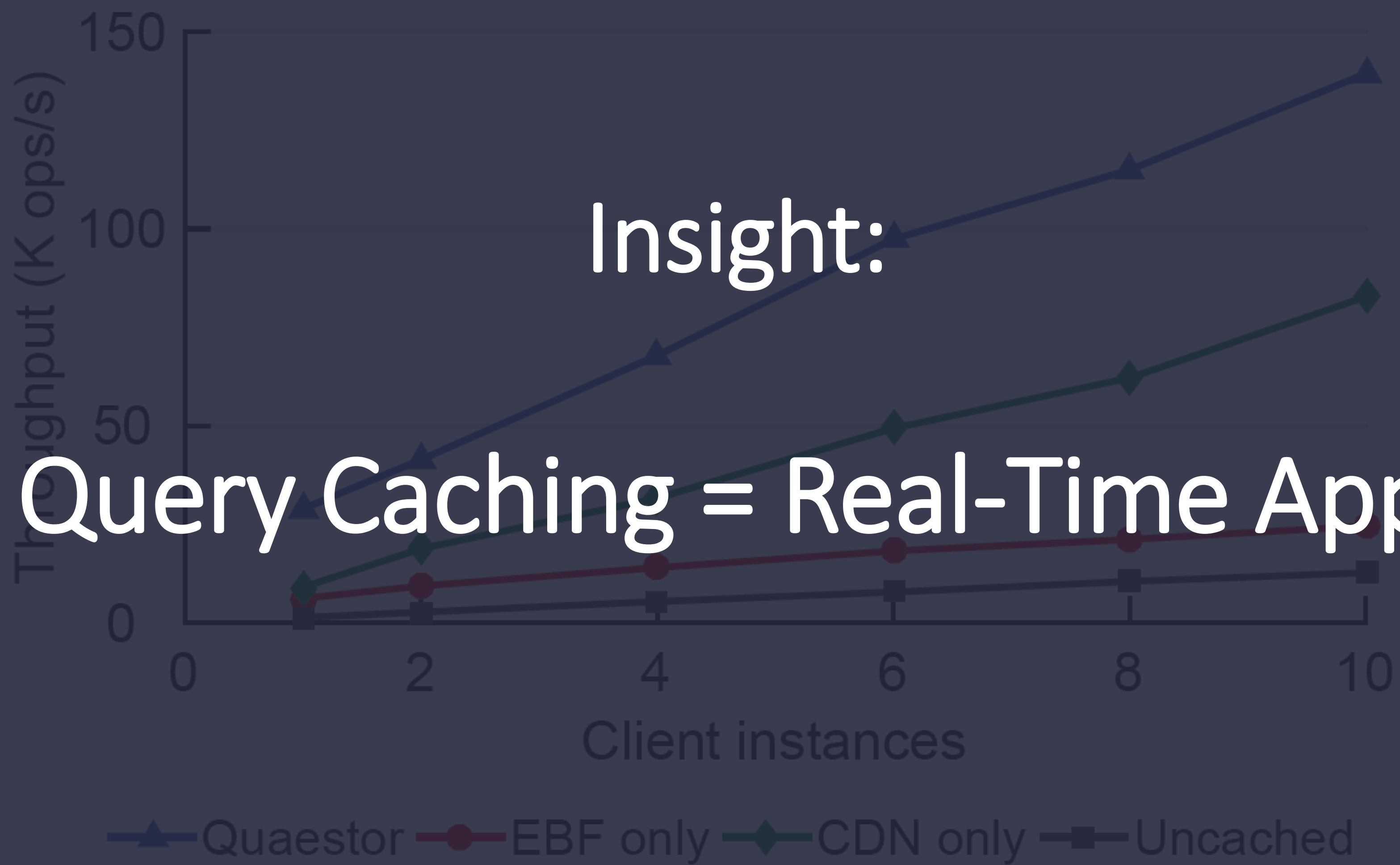
Approach: Cost-based decision model that weighs expected round-trips vs expected invalidations

Ongoing Research: Reinforcement learning of decisions

What is the impact of query caching?

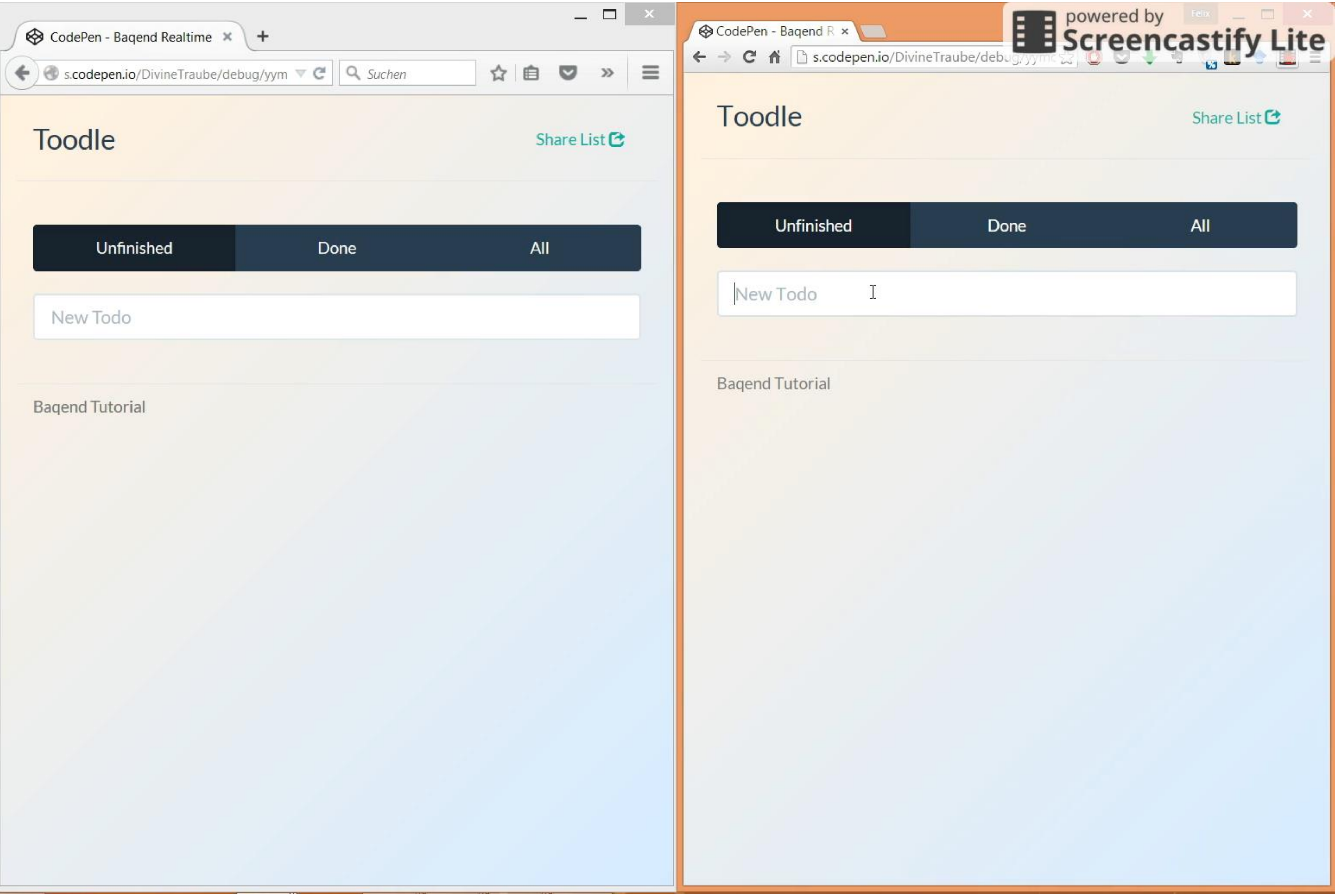


What is the impact of query caching?



Insight:

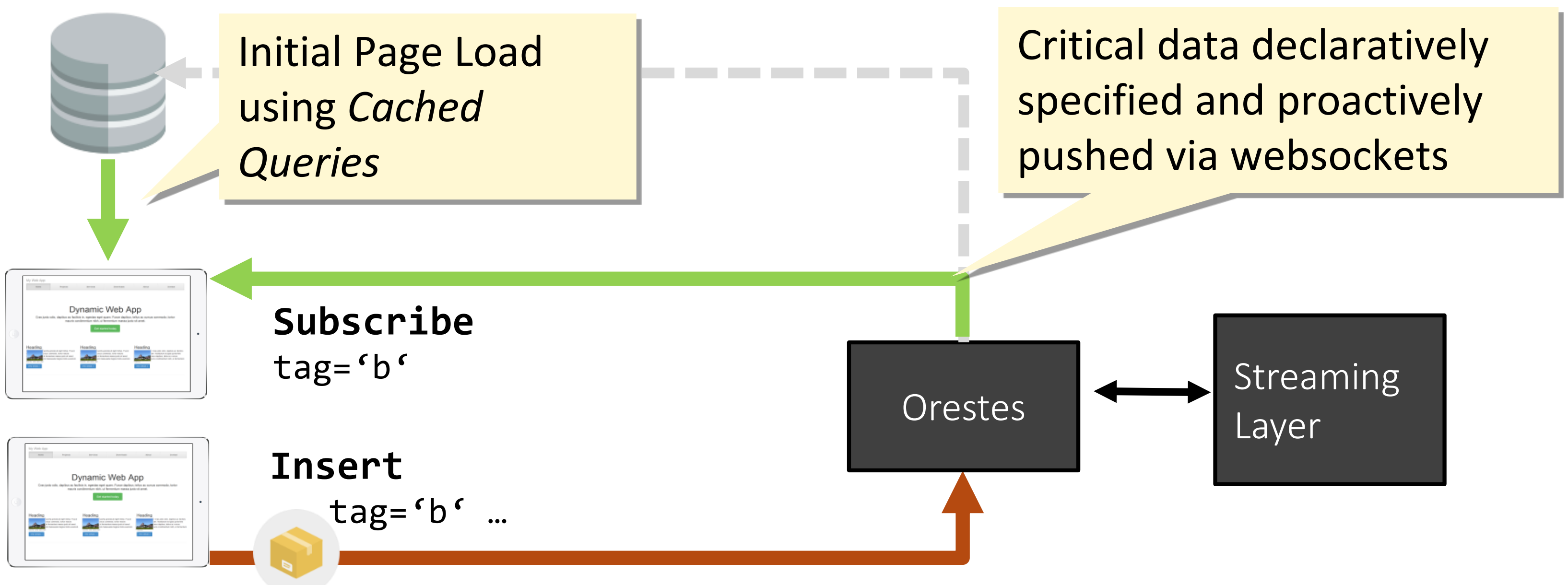
Query Caching = Real-Time Apps


CodePen - Baqend Realtime
s.codepen.io/DivineTraube/debug/yym
Suchen
Toodle
Share List
Unfinished
Done
All
New Todo
Baqend Tutorial
CodePen - Baqend R
s.codepen.io/DivineTraube/debug/yym
powered by
Screencastify Lite
Toodle
Share List
Unfinished
Done
All
New Todo
Baqend Tutorial

Continuous Queries

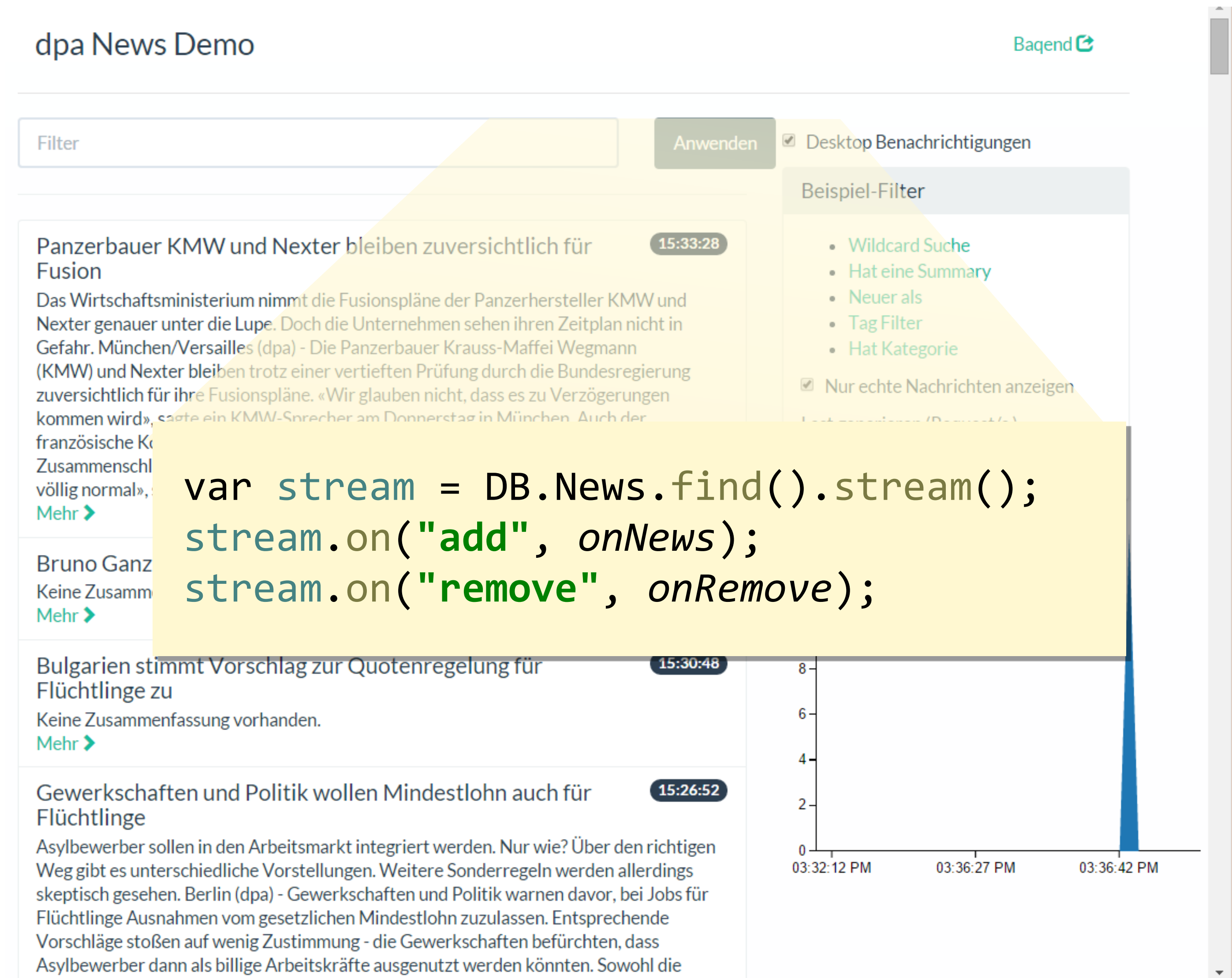
Complementing Cached Queries

- Same streaming architecture can similarly notify the browser about query result changes
- **Application Pattern:**



Continuous Query API

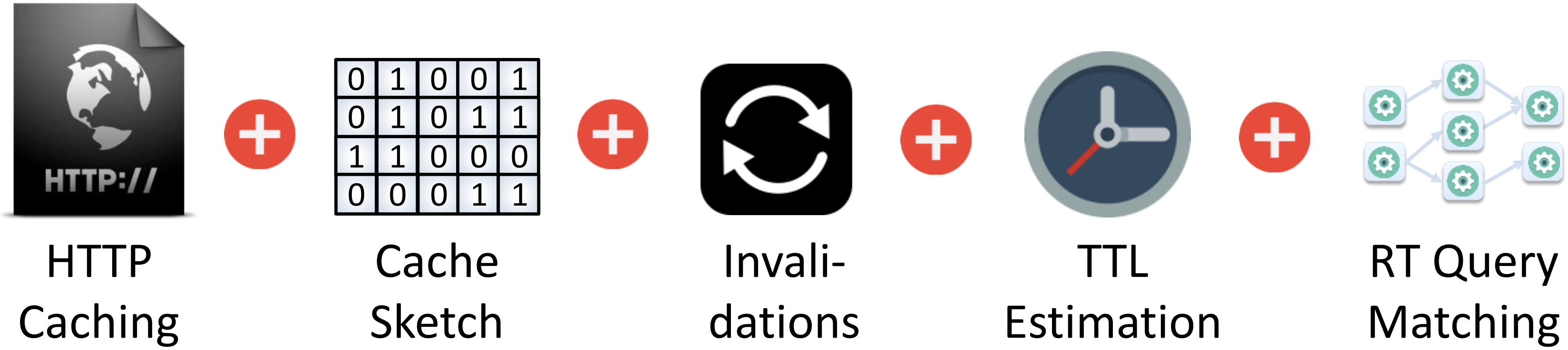
Subscribing to database updates




```
var stream = DB.News.find().stream();
stream.on("add", onNews);
stream.on("remove", onRemove);
```

Summary

- **Orestes:** DB-independent **Backend-as-a-Service**
- **Cache Sketch Approach:**
  - Client decides when to **revalidate**, server **invalidates** CDN
  - Cache Sketch = **Bloom filter** of stale IDs
  - Compatible with end-to-end ACID **transactions**
  - Query change detection in **real-time**



Introduction | Main Part | **Conclusions**

Web Performance: State of the Art

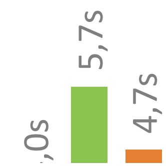
Cache Sketch: Research Approach

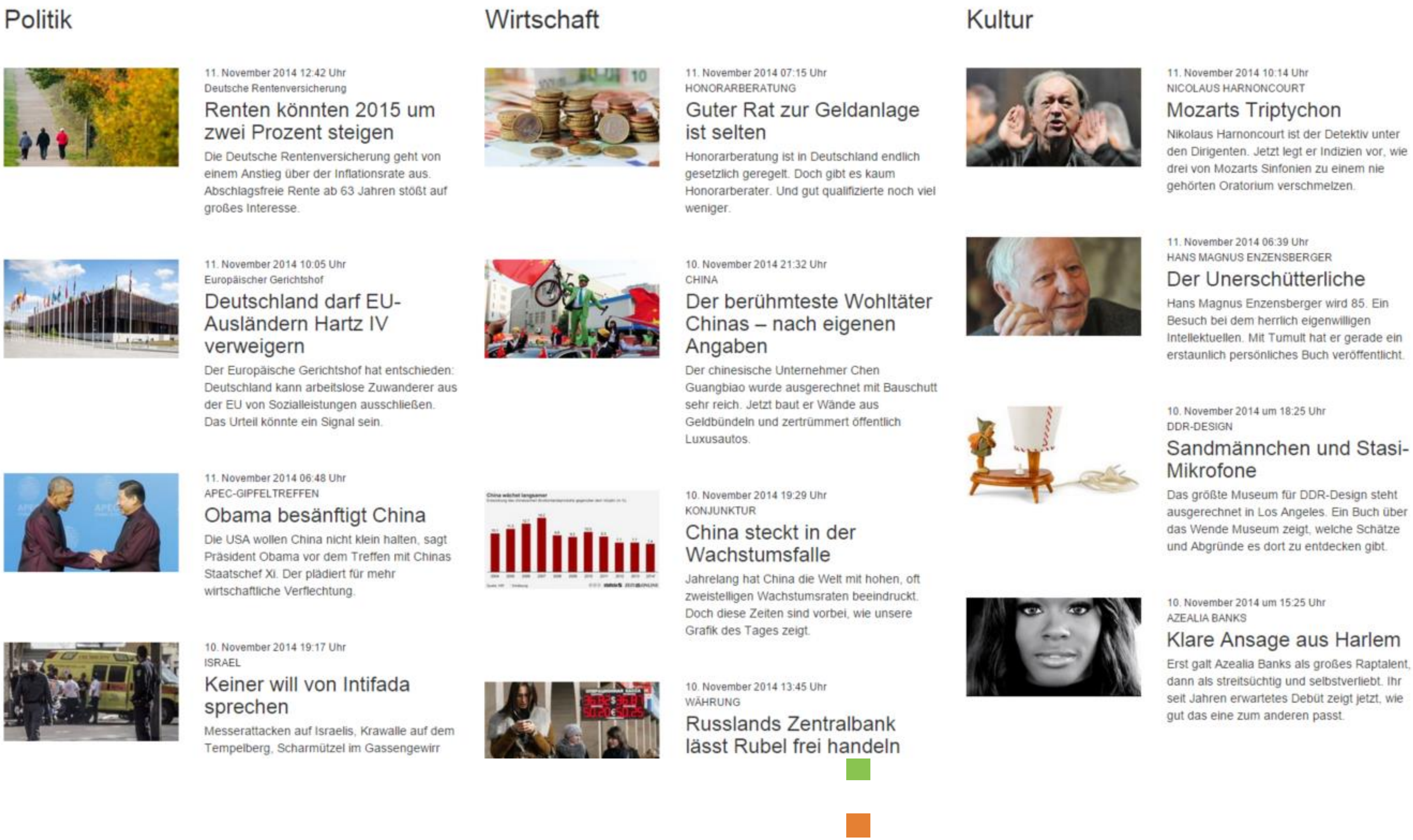
Using Web Caching in Applications

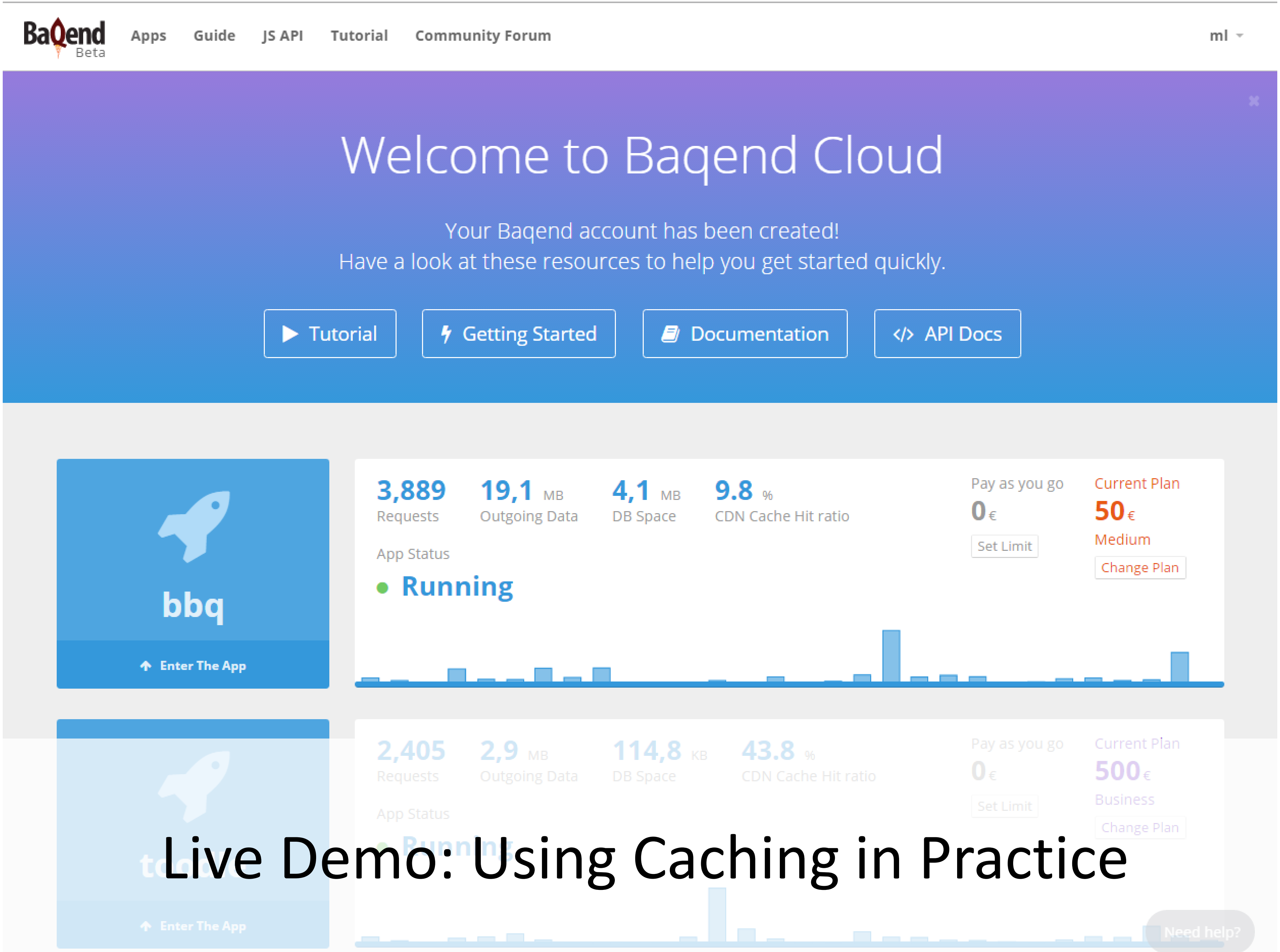

Orestes Caching
Technology as a
Backend-as-a-Service
BaQend
Build faster Apps faster.

Page-Load Times

What impact does caching have in practice?







Live Demo: Using Caching in Practice

Want to try Baqend?

- Free **Baqend Cloud** instance
- Download **Community Edition**

Thank you

Questions?

Twitter:
@baqendcom

fg@
baqend.com

baqend.com