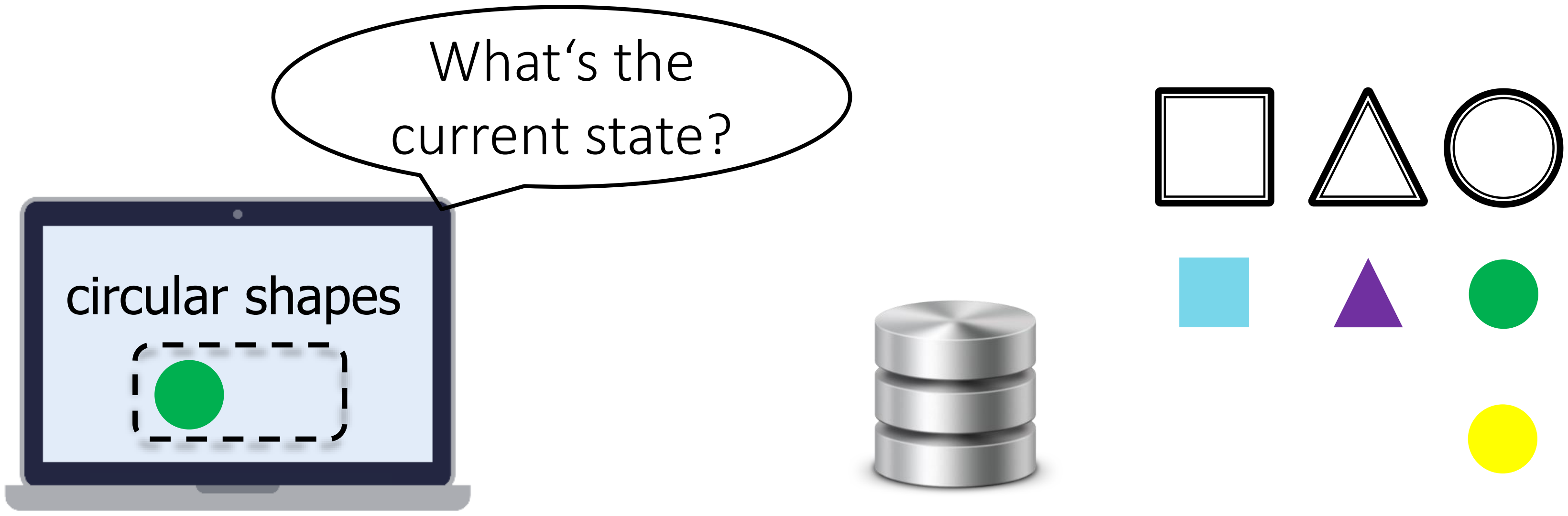
The Case for Change Notifications
in Pull-Based Databases

Wolfram Wingerath, Felix Gessert, Steffen Friedrich, Erik Witt and Norbert Ritter

Wolfram Wingerath
wingerath@informatik.uni-hamburg.de
March 6th, 2017, Stuttgart

Traditional Databases

No Request? No Data!



Query maintenance: periodic polling

→ Inefficient

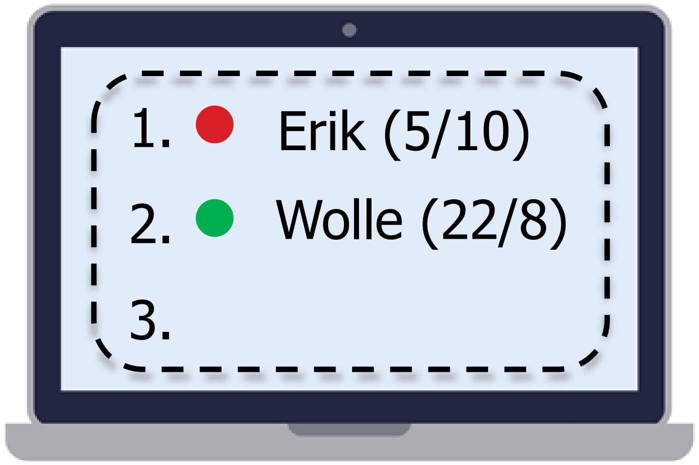
→ Slow

Ideal: Push-Based Data Access

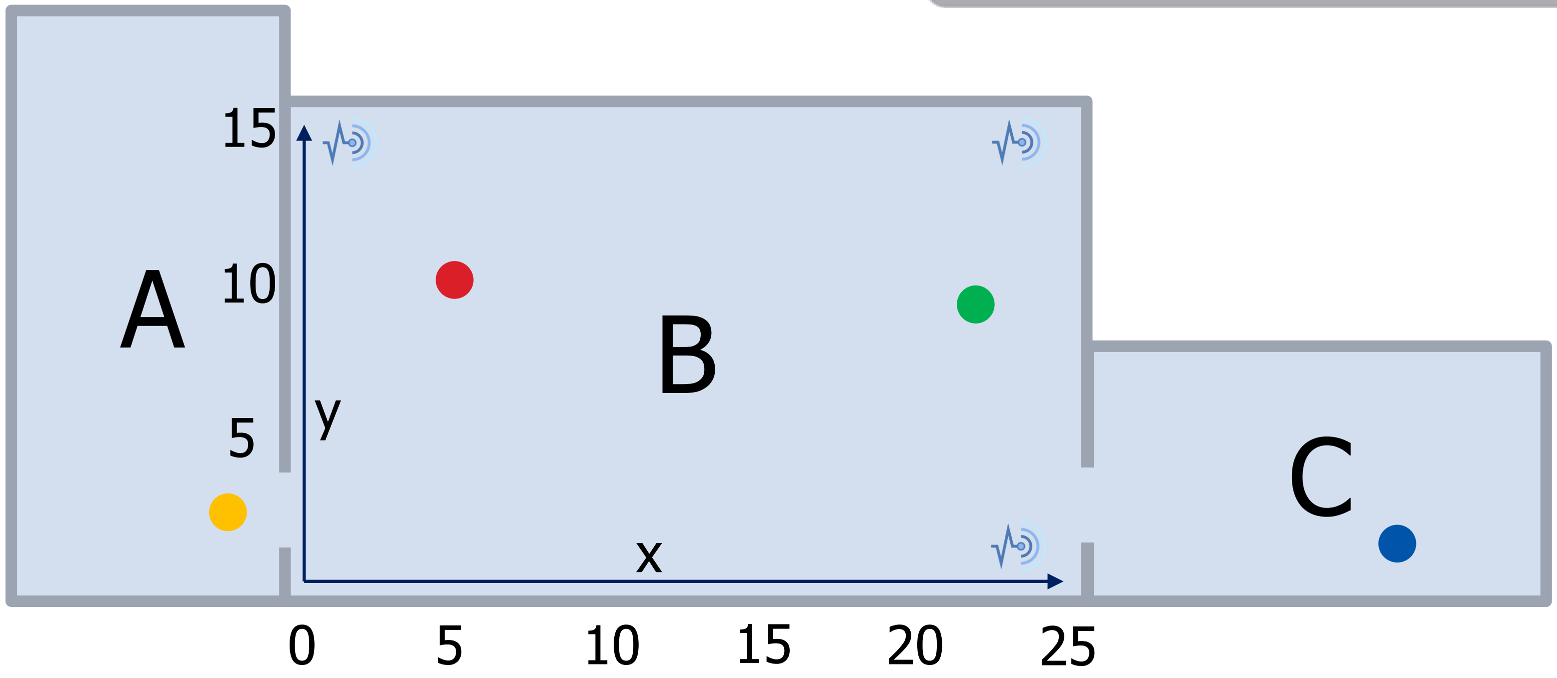
Self-Maintaining Results

Find people in Room B:

```
db.User.find()
     .equal('room','B')
     .ascending('name')
     .limit(3)
     .streamResult()
```



1. ● Erik (5/10)
2. ● Wolle (22/8)
3.



Real-Time Databases

Firebase

Overview:

- **Real-time state synchronization** across devices
- **Simplistic data model:** nested hierarchy of lists and objects
- **Simplistic queries**: mostly navigation/filtering
- **Fully managed**, proprietary
- **App SDK** for App development, mobile-first
- **Google services integration**: analytics, hosting, authorization, ...

History:

- 2011: chat service startup Envolve is founded
  → was often used for cross-device state synchronization
  → state synchronization is separated (Firebase)
- 2012: Firebase is founded
- 2013: Firebase is acquired by Google

Firebase
Real-Time State Synchronization

- **Tree data model**: application state ~JSON object
- **Subtree synching**: push notifications for specific keys only
  → Flat structure for fine granularity

→ Limited expressiveness!

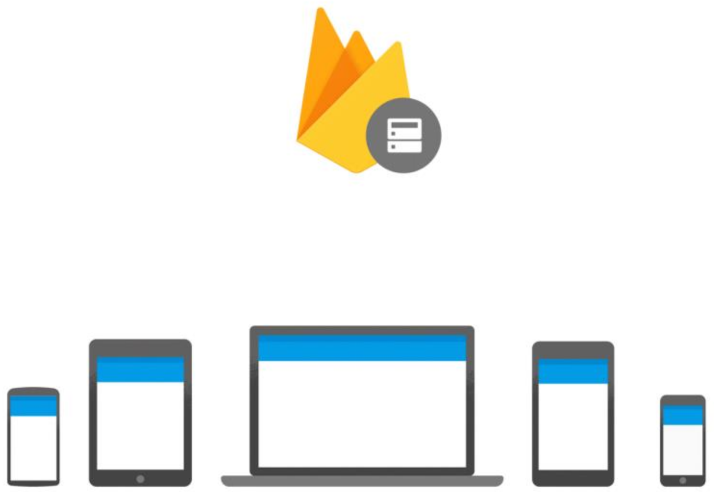


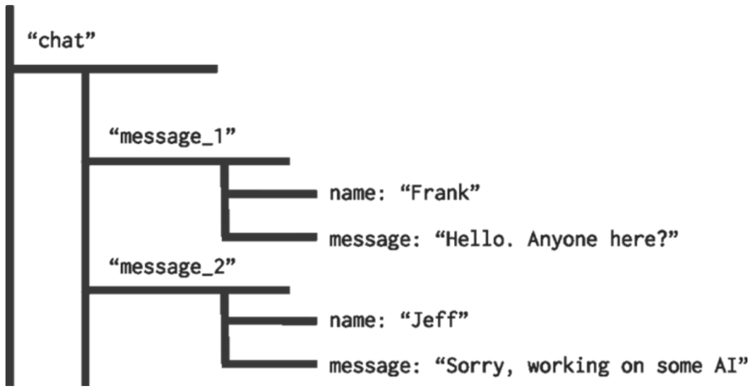
Illustration taken from: Frank van Puffelen, *Have you met the Realtime Database? (2016)*
https://firebase.googleblog.com/2016/07/have-you-met-realtime-database.html (2017-02-27)

Firebase

Query Processing in the Client

- Push notifications for **specific keys** only
  - Order by a **single attribute**
  - Apply a **single filter** on that attribute

- Non-trivial query processing in client
  → does not scale!


"chat"
"message_1"
name: "Frank"
message: "Hello. Anyone here?"
"message_2"
name: "Jeff"
message: "Sorry, working on some AI"

Jacob Wenger, on the Firebase Google Group *(2015)*
https://groups.google.com/forum/#!topic/firebase-talk/d-XjaBVL2Ko (2017-02-27)

Illustration taken from: Frank van Puffelen, *Have you met the Realtime Database? (2016)*
https://firebase.googleblog.com/2016/07/have-you-met-realtime-database.html (2017-02-27)

Meteor

Overview:

- **JavaScript Framework** for interactive apps and websites
  - **MongoDB** under the hood
  - **Real-time** result updates, full MongoDB expressiveness
- **Open-source**: MIT license
- **Managed service**: Galaxy (Platform-as-a-Service)

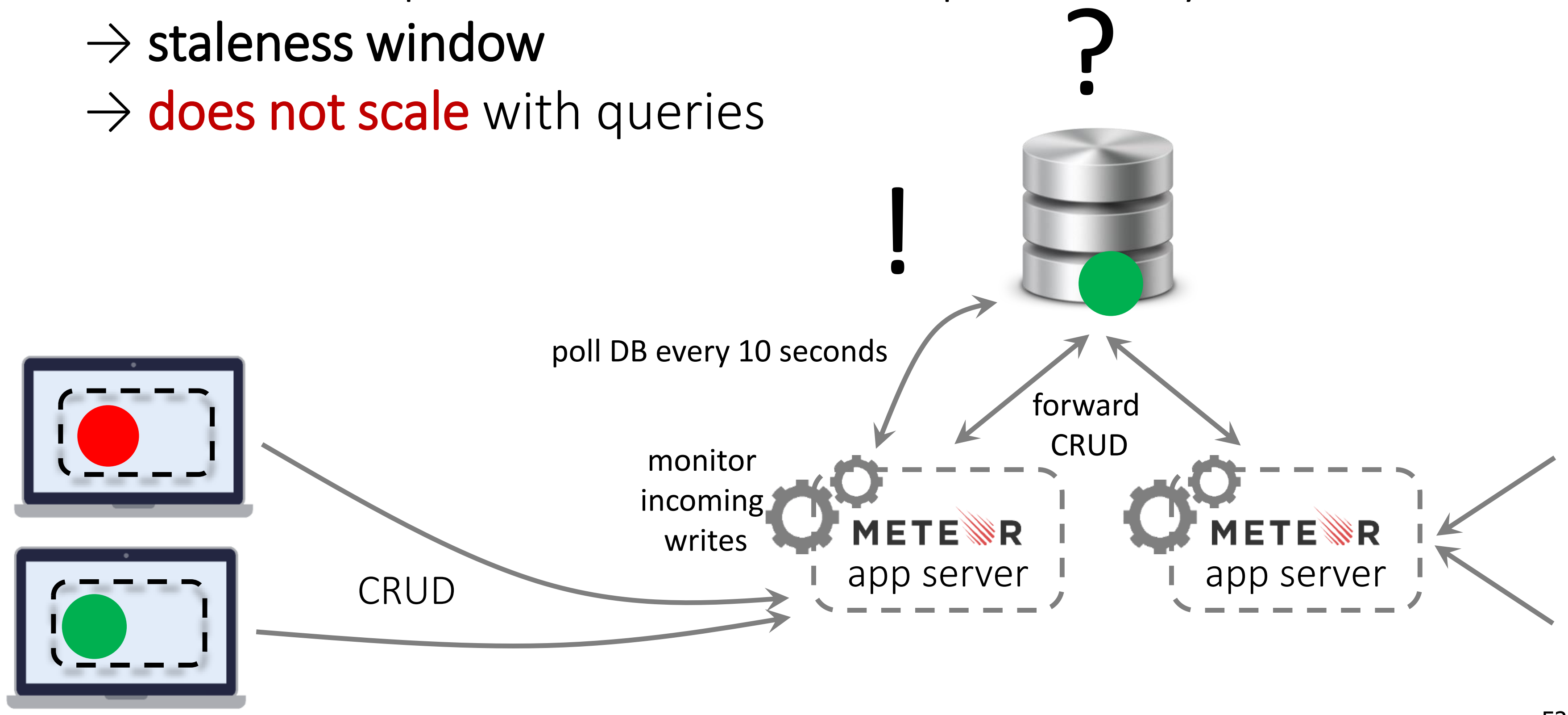
History:

- 2011: *Skybreak* is announced
- 2012: Skybreak is renamed to Meteor
- 2015: Managed hosting service Galaxy is announced

Live Queries

Poll-and-Diff

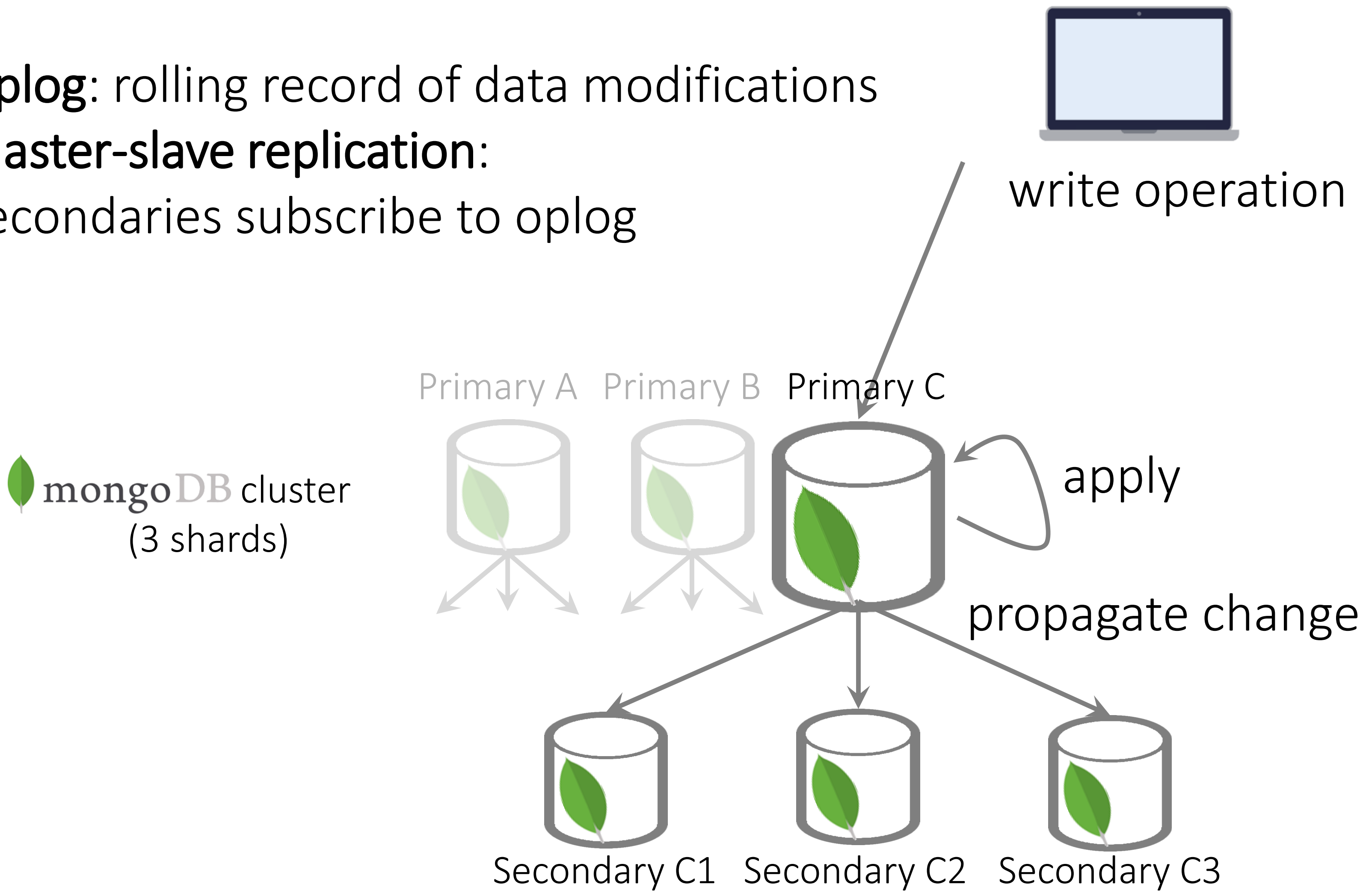
- **Change monitoring**: app servers detect relevant changes
  → *incomplete* in multi-server deployment
- **Poll-and-diff**: queries are re-executed periodically
  → **staleness window**
  → **does not scale** with queries



Oplog Tailing

Basics: MongoDB Replication

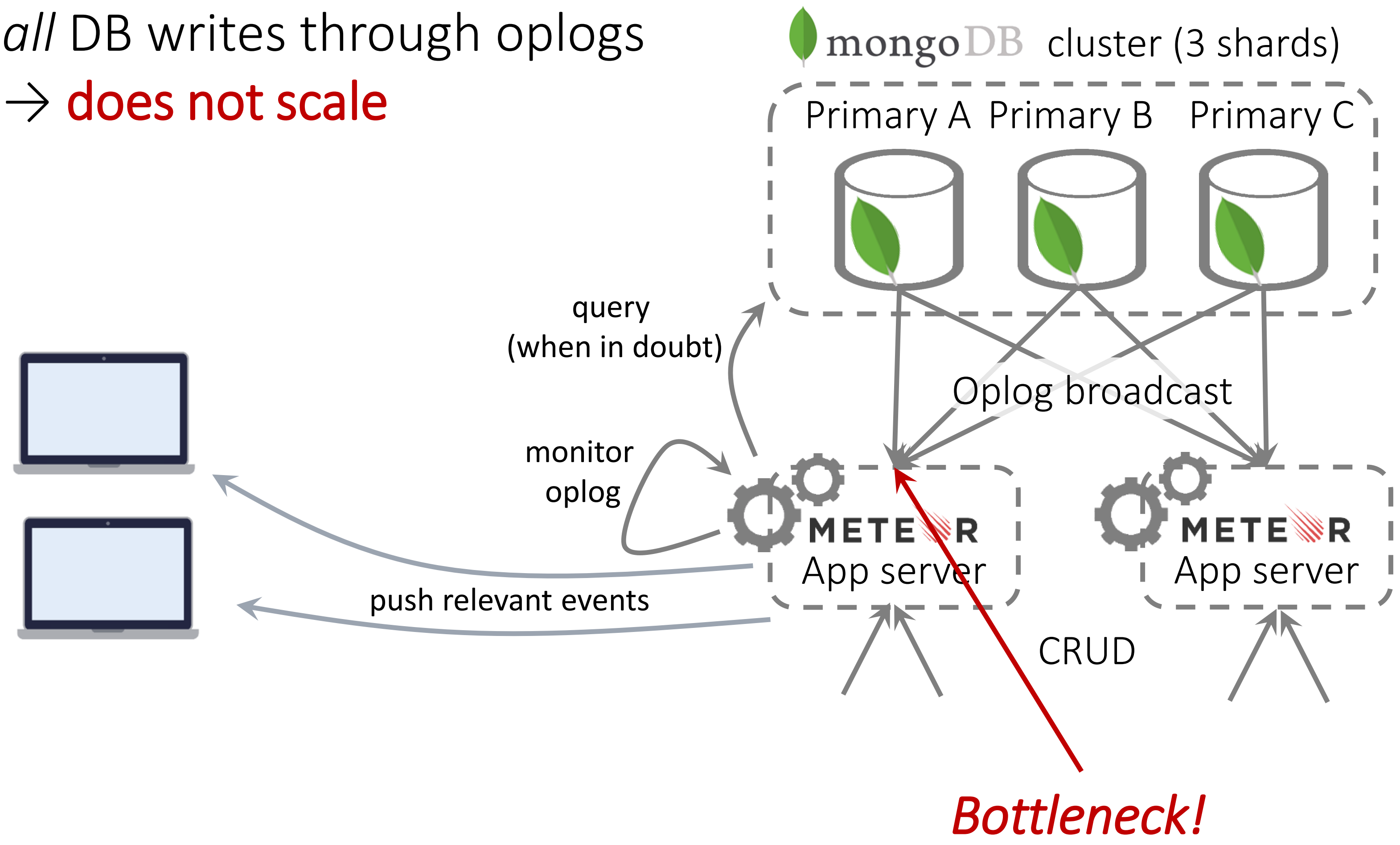
- **Oplog**: rolling record of data modifications
- **Master-slave replication**:
  Secondaries subscribe to oplog



Oplog Tailing

Tapping into the Oplog

- *Every* Meteor server receives *all* DB writes through oplogs → does not scale



Oplog Tailing

Oplog Info is Incomplete

What game does Bobby play?

→ if baccarat, he takes first place!

→ if something else, nothing changes!

Partial update from oplog:

```
{ name: „Bobby", score: 500 } // game: ???
```

Baccarat players sorted by high-score

```
1. { name: „Joy", game: „baccarat", score: 100 }
2. { name: „Tim", game: „baccarat", score: 90 }
3. { name: „Lee", game: „baccarat", score: 80 }
```

RethinkDB

RethinkDB

Overview:

- **„MongoDB done right“**: comparable queries and data model, but also:
    - **Push-based queries** (filters only)
    - **Joins** (non-streaming)
    - **Strong consistency**: linearizability
- **JavaScript SDK** (*Horizon*): open-source, as managed service
- **Open-source**: Apache 2.0 license

History:

- 2009: RethinkDB is founded
- 2012: RethinkDB is open-sourced under AGPL
- 2016, May: first official release of Horizon (JavaScript SDK)
- 2016, October: RethinkDB announces shutdown
- 2017: RethinkDB is relicensed under Apache 2.0

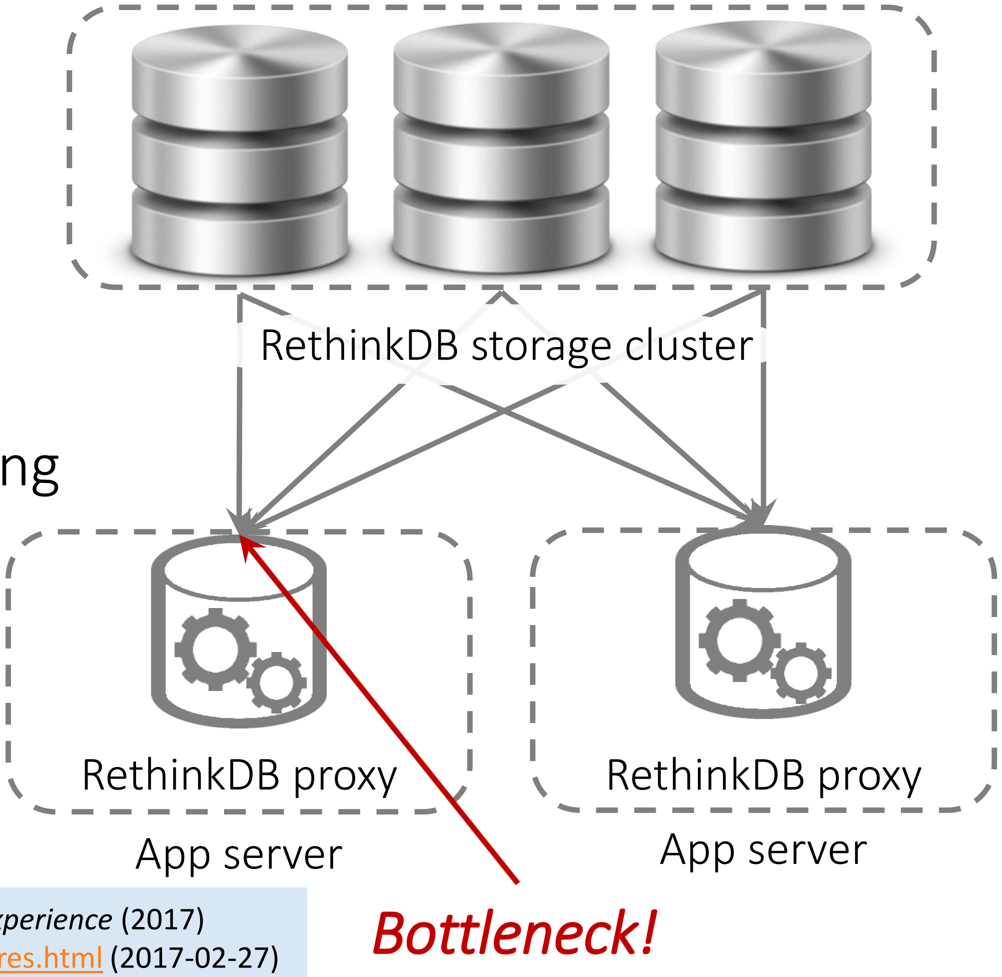
RethinkDB

Changefeed Architecture

RethinkDB

- Range-sharded data
- **RethinkDB proxy**: support node without data
  - Client communication
  - Request routing
  - Real-time query matching

- *Every* proxy receives *all* database writes → does not scale



William Stein, *RethinkDB versus PostgreSQL: my personal experience* (2017)
http://blog.sagemath.com/2017/02/09/rethinkdb-vs-postgres.html (2017-02-27)

Daniel Mewes, *Comment on GitHub issue #962: Consider adding more docs on RethinkDB Proxy* (2016)
https://github.com/rethinkdb/docs/issues/962 (2017-02-27)

Parse

Overview:

- **Backend-as-a-Service** for mobile apps
  - **MongoDB:** largest deployment world-wide
  - **Easy development**: great docs, push notifications, authentication, ...
  - **Real-time** updates for most MongoDB queries
- **Open-source**: BSD license
- **Managed service**: discontinued

History:

- 2011: Parse is founded
- 2013: Parse is acquired by Facebook
- 2015: more than 500,000 mobile apps reported on Parse
- 2016, January: Parse shutdown is announced
- 2016, March: **Live Queries** are announced
- 2017: Parse shutdown is finalized

Parse
LiveQuery Architecture

- **LiveQuery Server**: no data, real-time query matching
- *Every* LiveQuery Server receives *all* database writes
  → does not scale

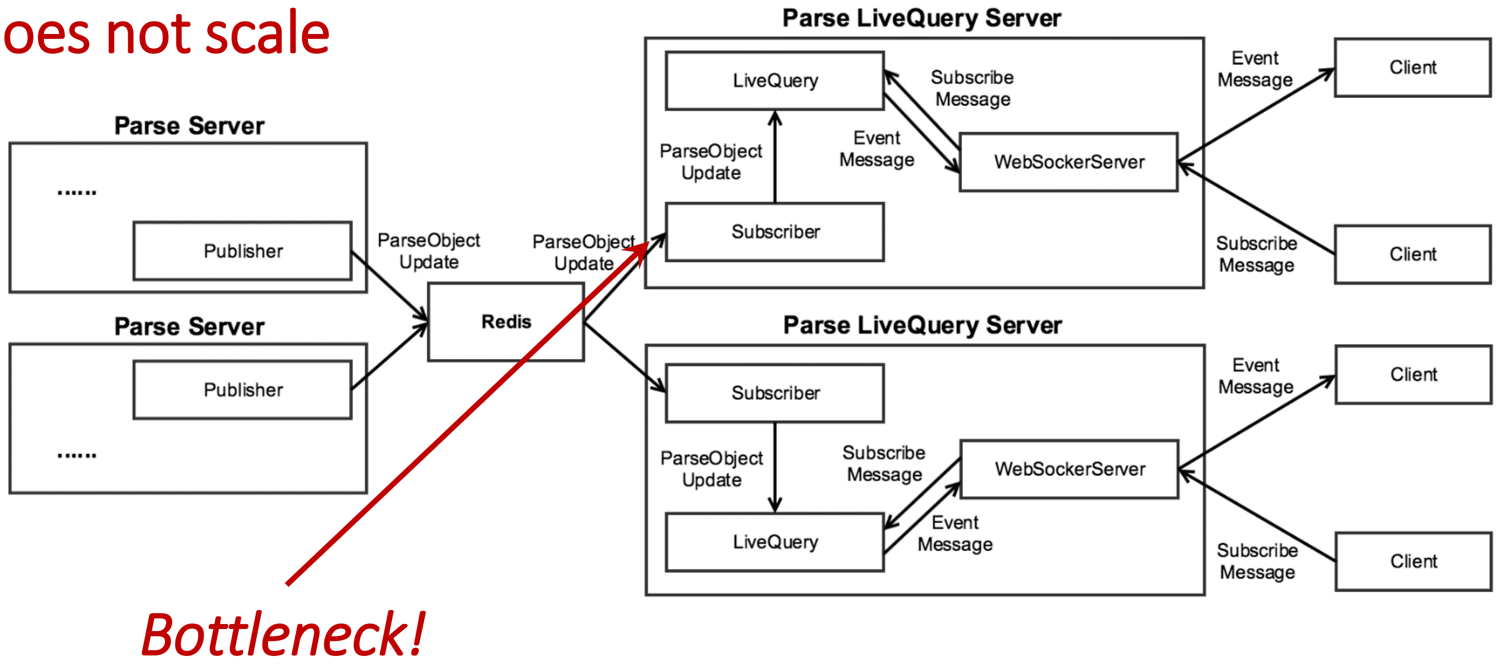


Illustration taken from:
http://parseplatform.github.io/docs/parse-server/guide/#live-queries (2017-02-22)

Comparison by Real-Time Query

Why Complexity Matters

	matching conditions	ordering	Firebase	Meteor	RethinkDB	Parse
Todos	created by „Bob“	ordered by deadline	✓	✓	✓	✗
Todos	created by „Bob“ AND with status equal to „active“		✗	✓	✓	✓
Todos	with „work“ in the name		✗	✓	✓	✓
		ordered by deadline	✗	✓	✓	✗
Todos	with „work“ in the name AND status of „active“	ordered by deadline AND then by the creator‘s name	✗	✓	✓	✗

Quick Comparison

DBMS vs. RT DB vs. DSMS vs. Stream Processing

	Database Management	Real-Time Databases	Data Stream Management	Stream Processing
Data	persistent collections		persistent/ephemeral streams	
Processing	one-time	one-time + continuous	continuous	
Access	random	random + sequential	sequential	
Streams	structured			structured, unstructured
	ORACLE, PostgreSQL, MySQL, IBM DB2	Firebase, METEOR, RethinkDB, Parse	PIPELINEDB, EsperTech, sqlstream, influxdata	STORM, samza, Flink, Spark Streaming

Discussion

Common Issues

Every database with real-time features suffers from several of these problems:

- *Expressiveness*:
  - Queries
  - Data model
  - Legacy support
- *Performance*:
  - Latency & throughput
  - Scalability
- *Robustness*:
  - Fault-tolerance, handling malicious behavior etc.
  - Separation of concerns:

    → **Availability**:
    will a crashing real-time subsystem take down primary data storage?

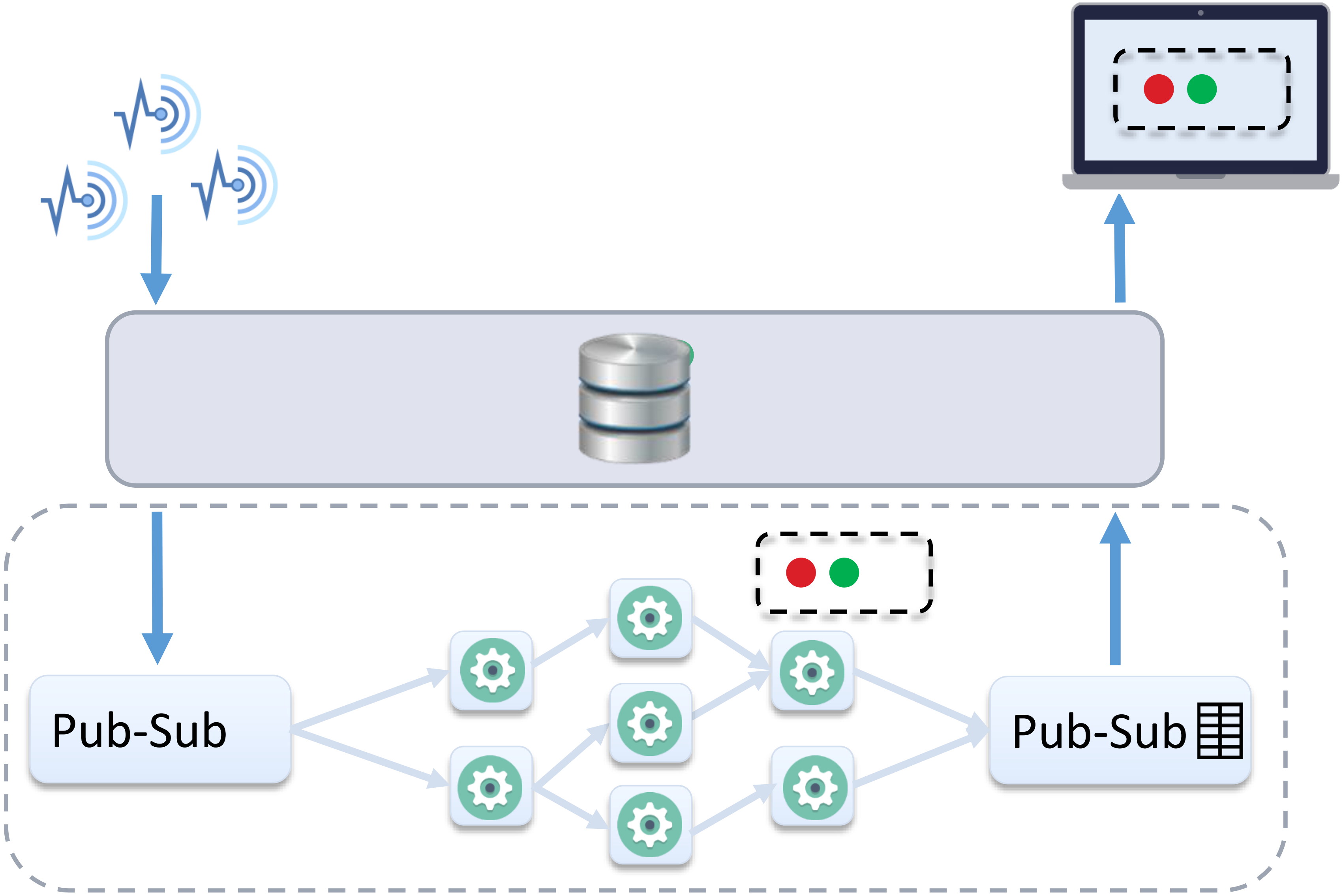
    → **Consistency**:
    can real-time be scaled out independently from primary storage?

Engineering Efforts:

Add-On Real-Time Queries

InvaliDB

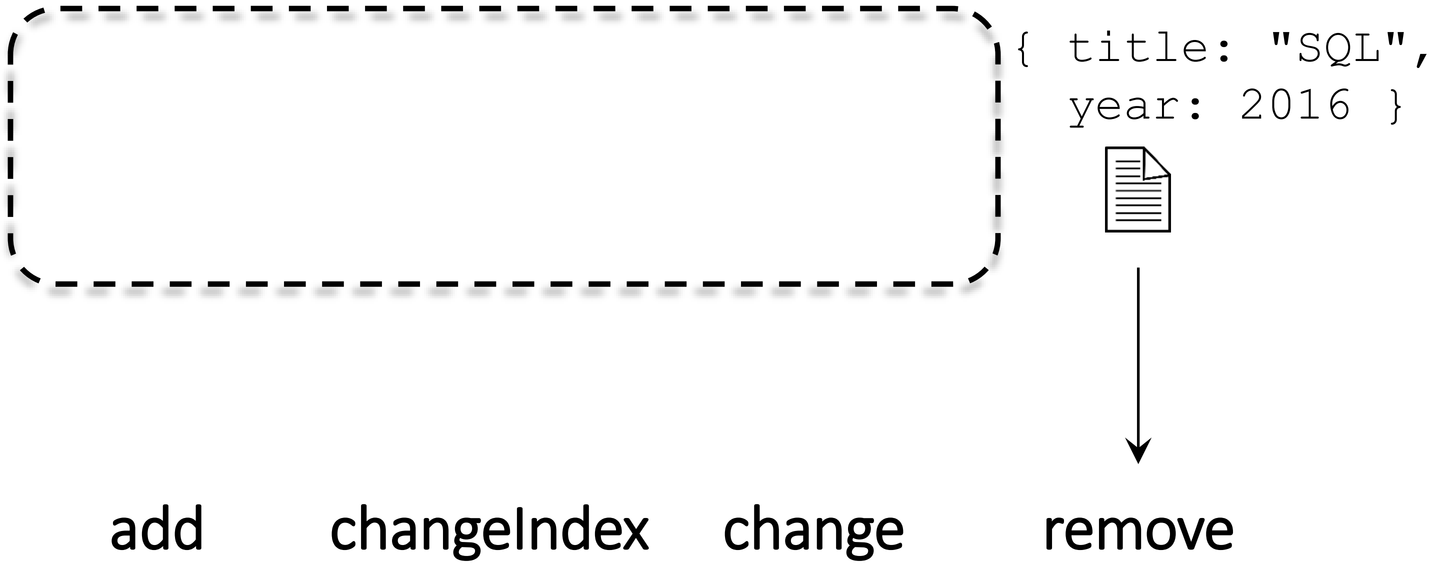
External Query Maintenance



InvaliDB

Change Notifications

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



InvaliDB

Filter Queries: Distributed Query Matching

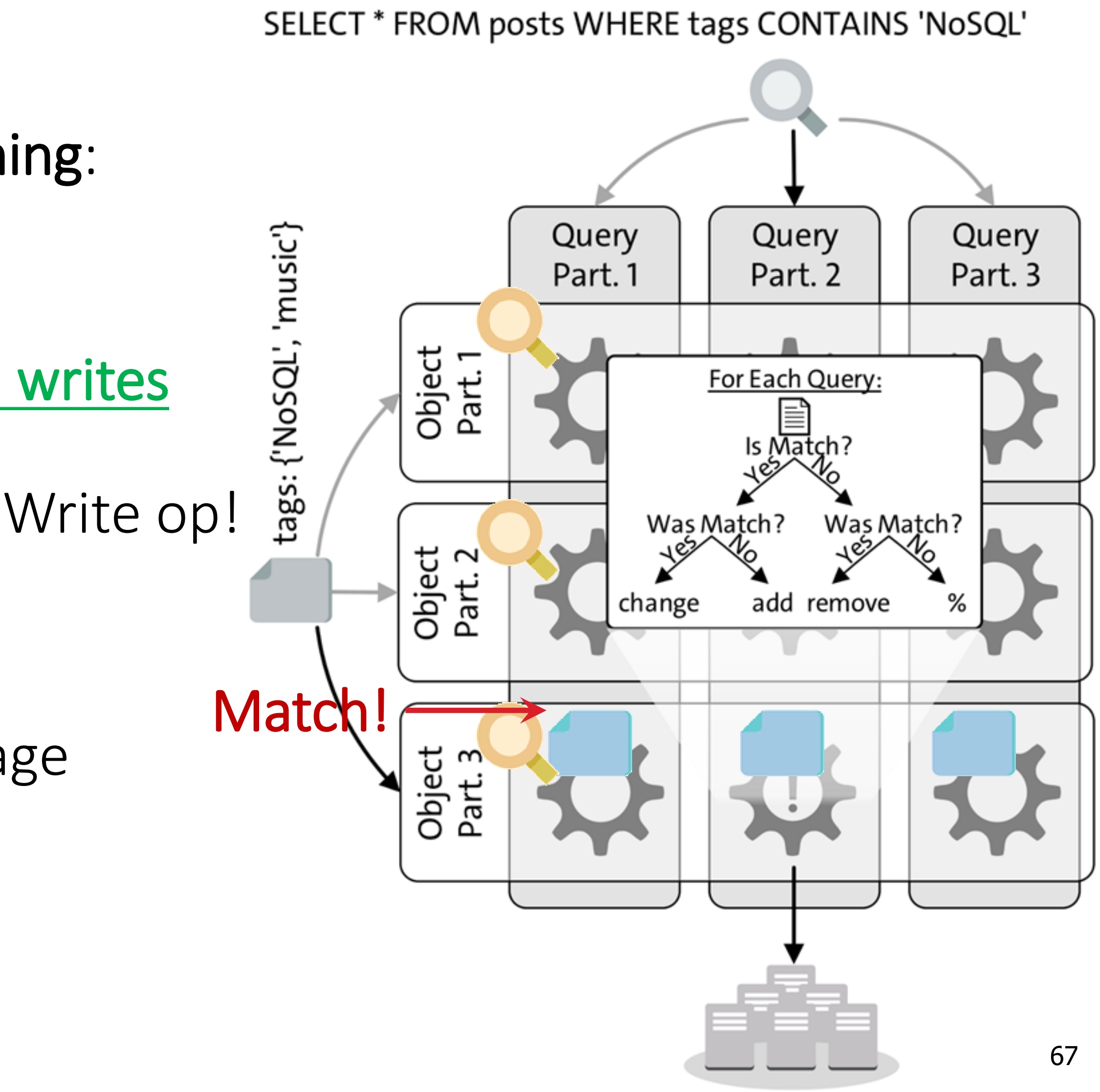
Two-dimensional partitioning:

- *by Query*
- *by Object*

→ scales with queries and writes

Implementation:

- Apache Storm
- Topology in Java
- MongoDB query language
- **Pluggable query engine**

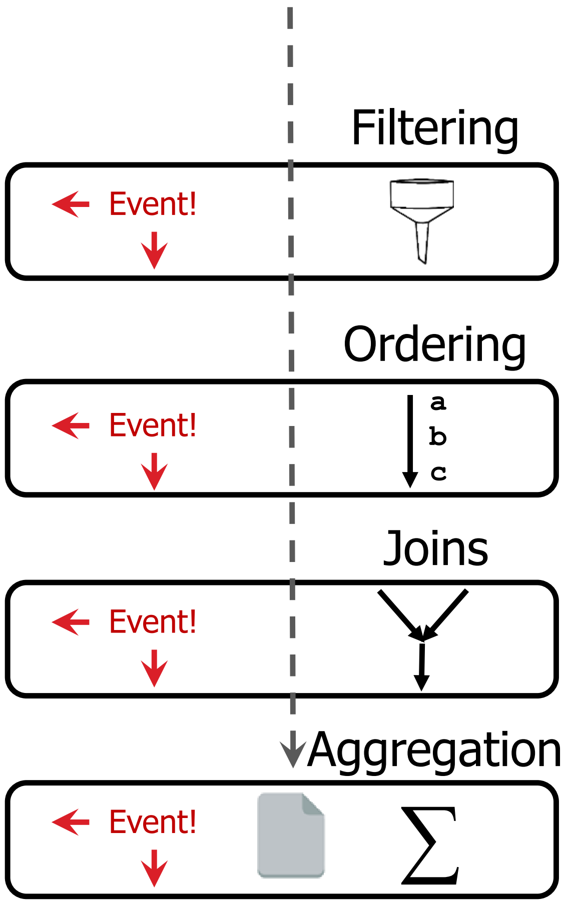


InvaliDB

Staged Real-Time Query Processing

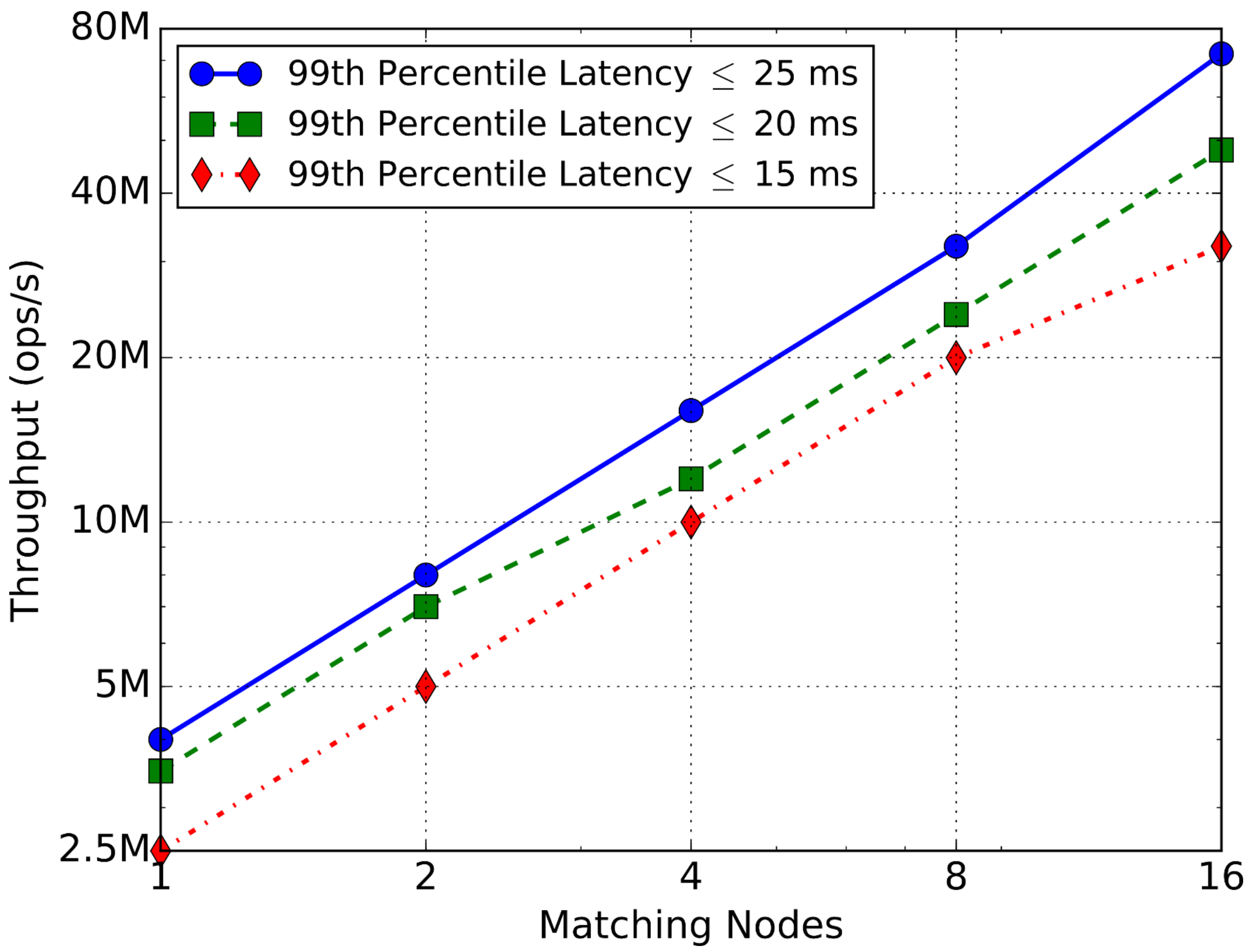
Change notifications go through up to 4 query processing stages:

1. **Filter queries**: track matching status
   → *before*- and after-images
2. **Sorted queries**: maintain result order
3. **Joins**: combine maintained results
4. **Aggregations**: maintain aggregations



InvaliDB

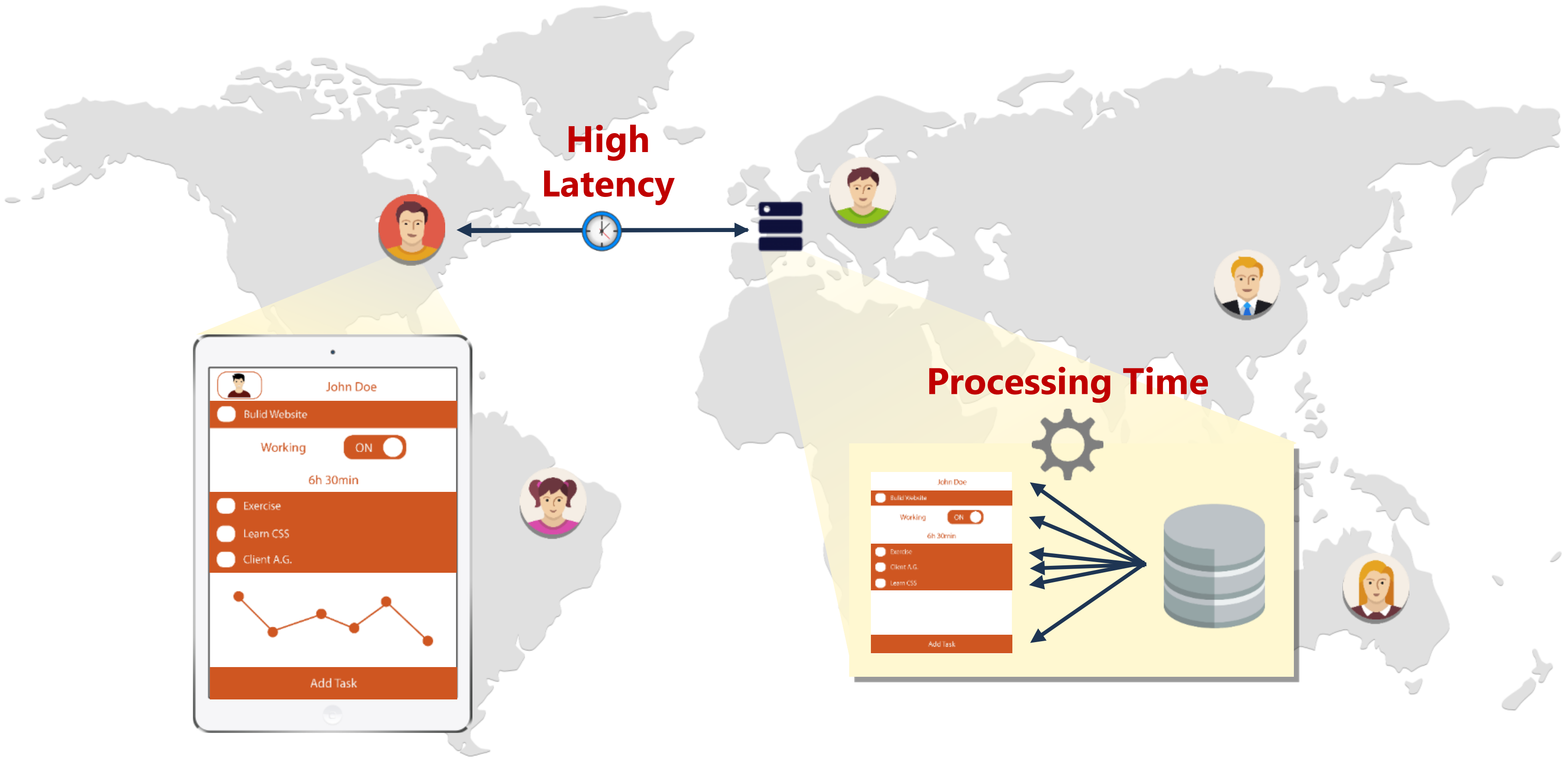
Low Latency + Linear Scalability



Research in Hamburg

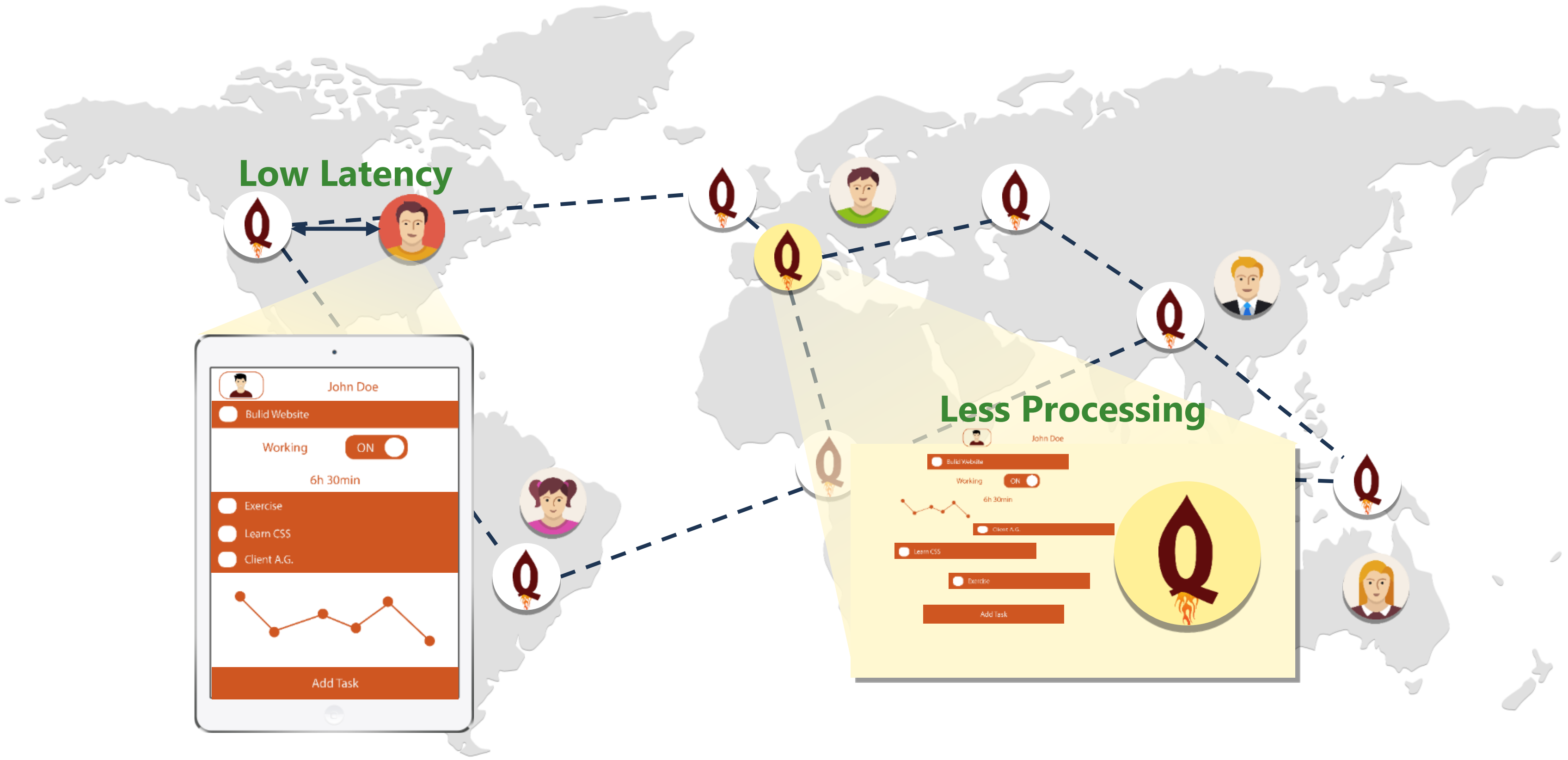
Delivering Dynamic Content

Two Bottlenecks: Latency und Processing



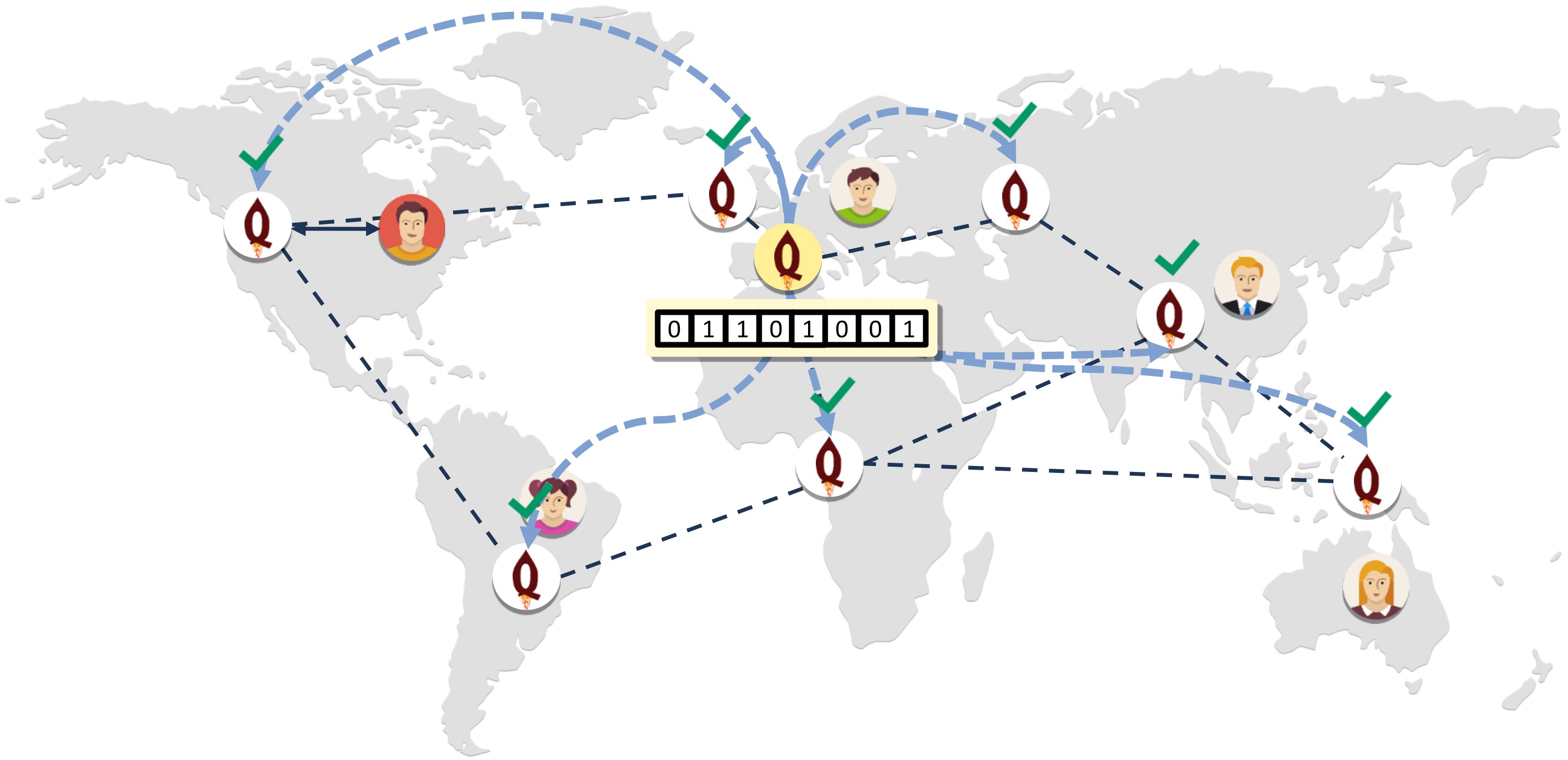
Solution: Global Caching

Fresh Data from Ubiquitous Web Caches



Caching Dynamic Content

Now Feasible: Invalidating Updated Queries



Wrap-up

- **Push-based data access**
  - Natural for many applications
  - Hard to implement on top of traditional (pull-based) databases
- **Real-time databases**
  - Natively push-based
  - Not legacy-compatible
  - Barely scalable
- **InvaliDB**
  - Add-On push-based queries
  - Database-independent
  - Linear scalability
  - Filter, sorting, joins, aggregations

Questions?