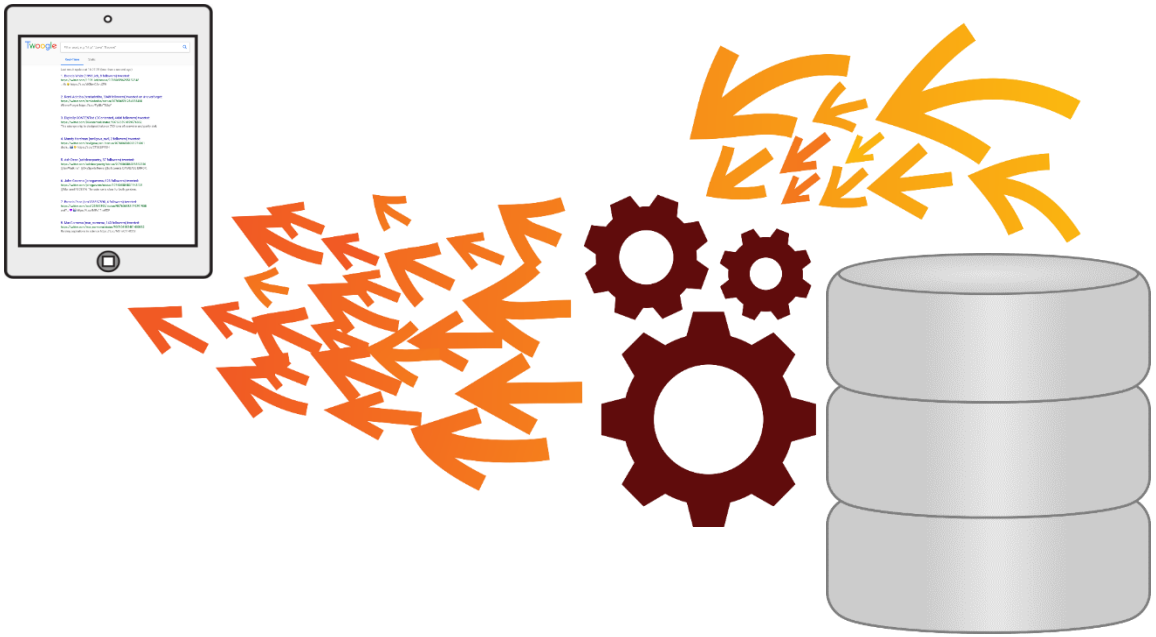
InvaliDB: Scalable Push-Based Real-Time Queries

on Top of Pull-Based Databases



Wolfram Wingerath, Felix Gessert, Norbert Ritter

ICDE 2020, Dallas/USA

Universität Hamburg

Baqend

InvaliDB: Scalable Push-Based Real-Time Queries

on Top of Pull-Based Databases

Wolfram Wingerath, Felix Gessert, Norbert Ritter

ICDE 2020, Dallas/USA

Universität Hamburg

Baqend

Outline

Problem Statement
Intro & Research Question

Related Work
State of the Art & Open Issues

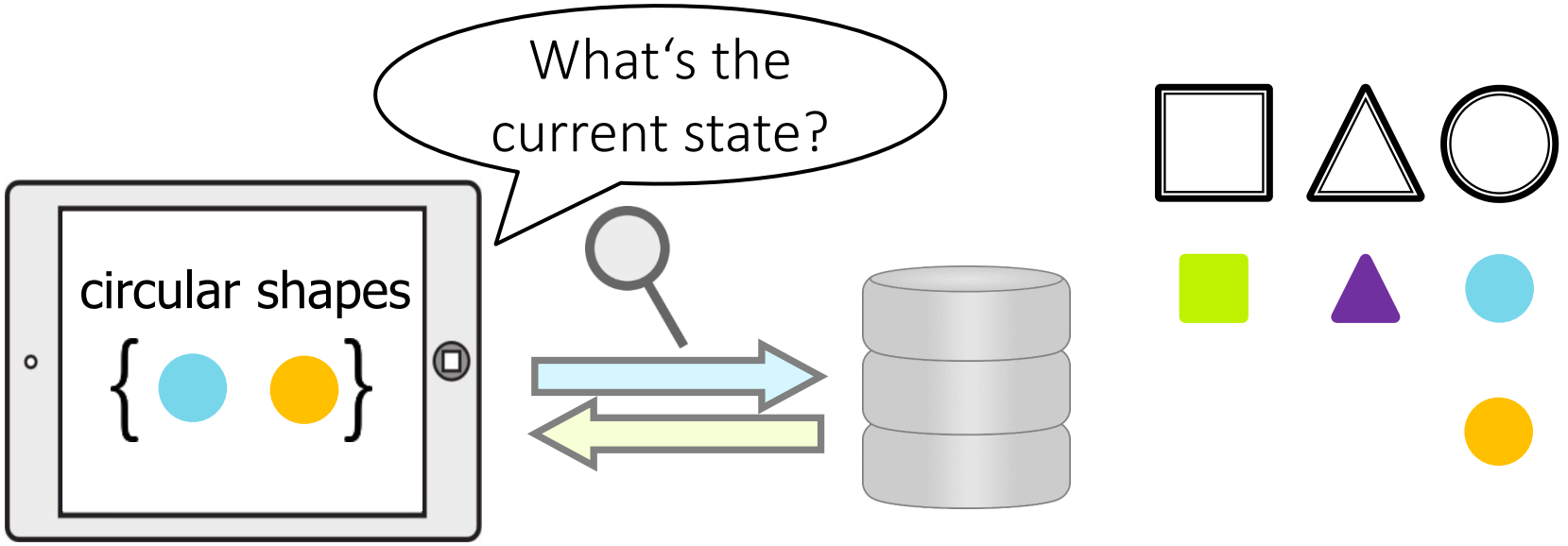
A Scalable RTDB Design
InvaliDB: Concept & Prototype

Discussion
Applications & Outlook

- **Pull vs. Push**
  - Traditional DB Queries
  - Why Real-Time Queries?
  - How to Provide Them?

Traditional Databases

The Problem: No Request – No Data!



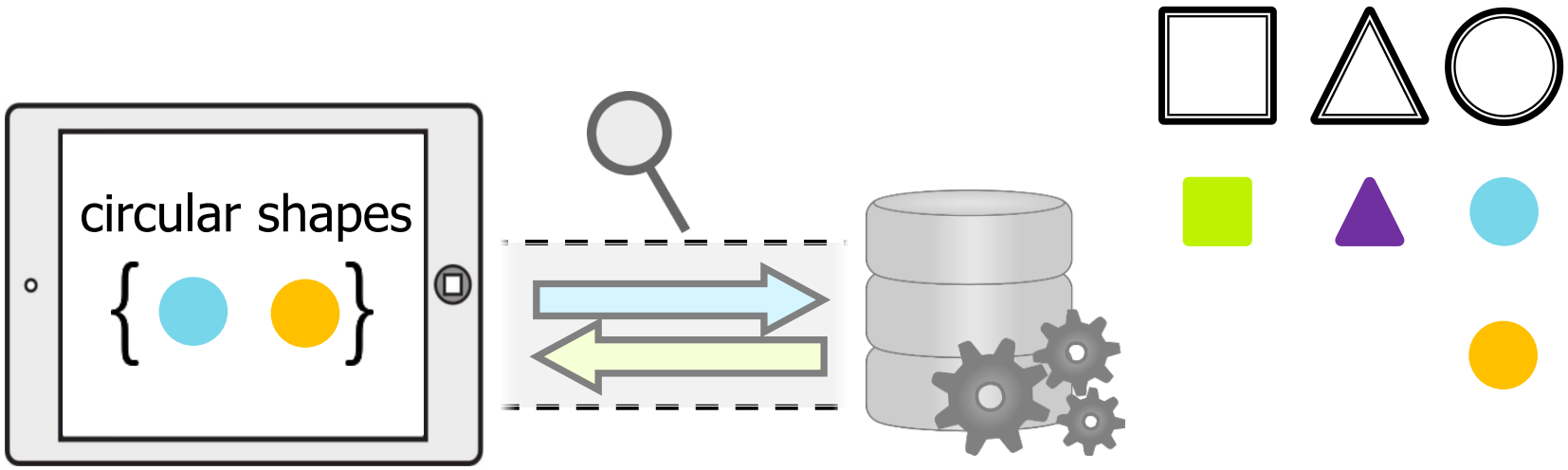
Periodic Polling for query result maintenance:

→ inefficient

→ slow

Real-time Databases

Always Up-to-Date With Database State



Real-Time Queries for query result maintenance:

→ efficient

→ fast

Real-Time Query Maintenance

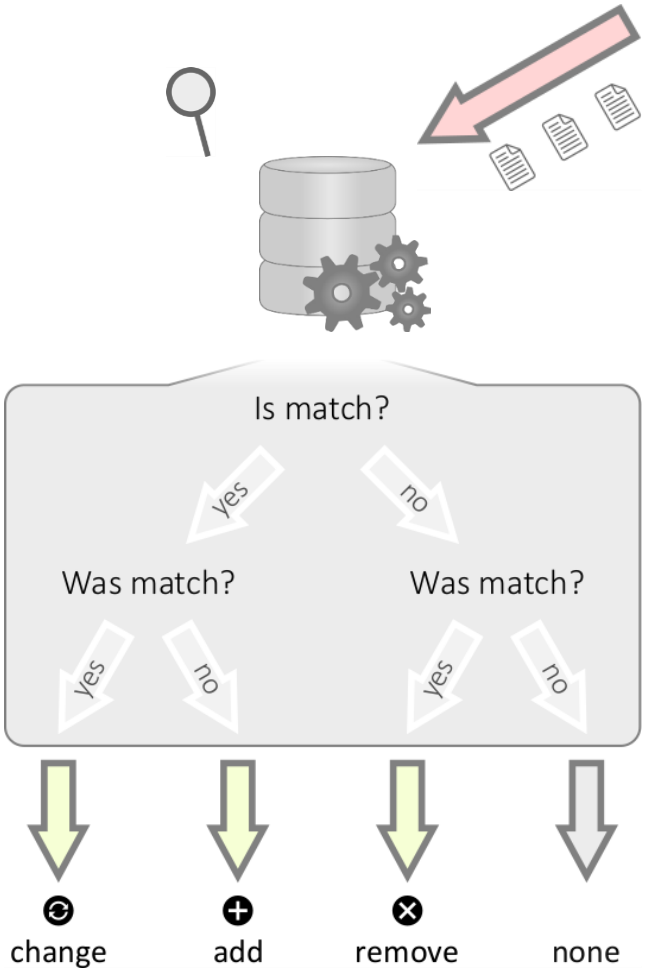
Matching Every Query Against Every Update

→ Potential ***bottlenecks***:

- *Number of queries*
- *Write throughput*
- *Query complexity*

Similar processing for:

- Triggers
- ECA rules
- Materialized views



Outline

Problem Statement
Intro & Research Question

Related Work
State of the Art & Open Issues

A Scalable RTDB Design
InvaliDB: Concept & Prototype

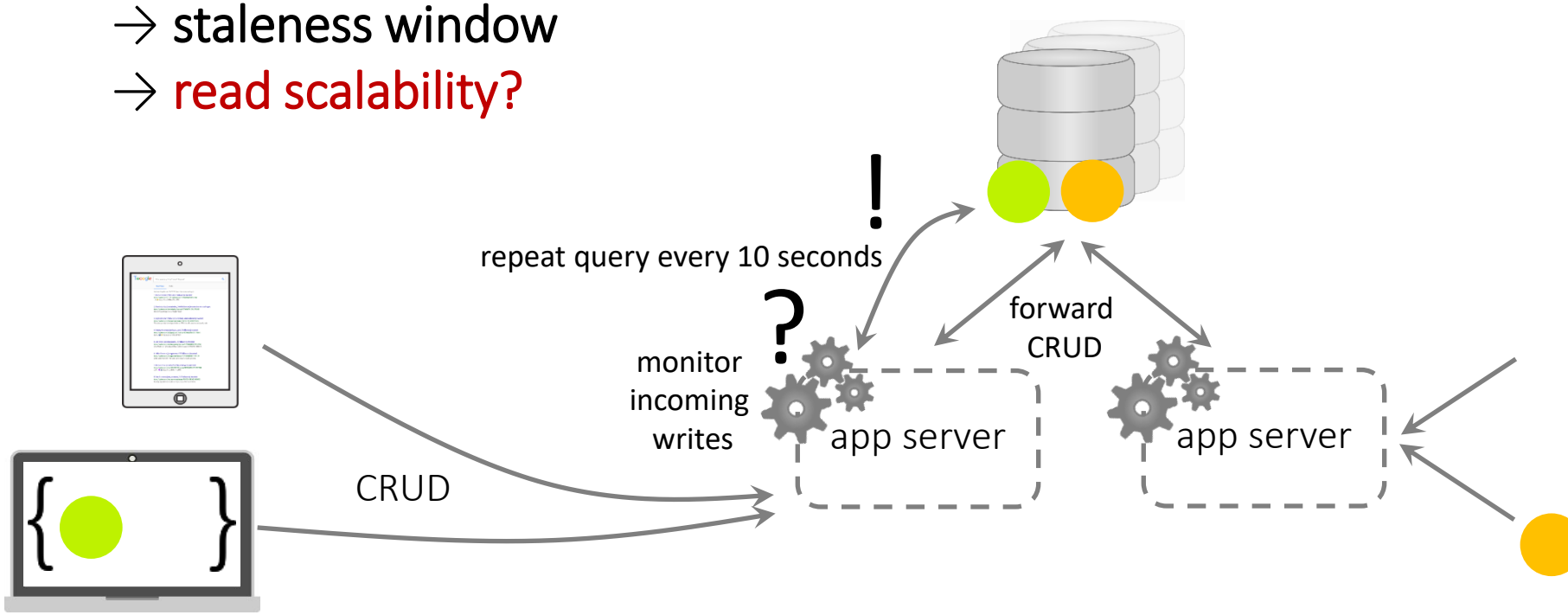
Discussion
Applications & Outlook

- **Real-Time Databases**
  - Poll-and-Diff
  - Oplog Tailing
- **System Comparison**
  - Meteor
  - RethinkDB
  - Parse
  - Firebase
  - InvaliDB

Typical Maintenance Mechanisms (1/2)

Poll-and-Diff

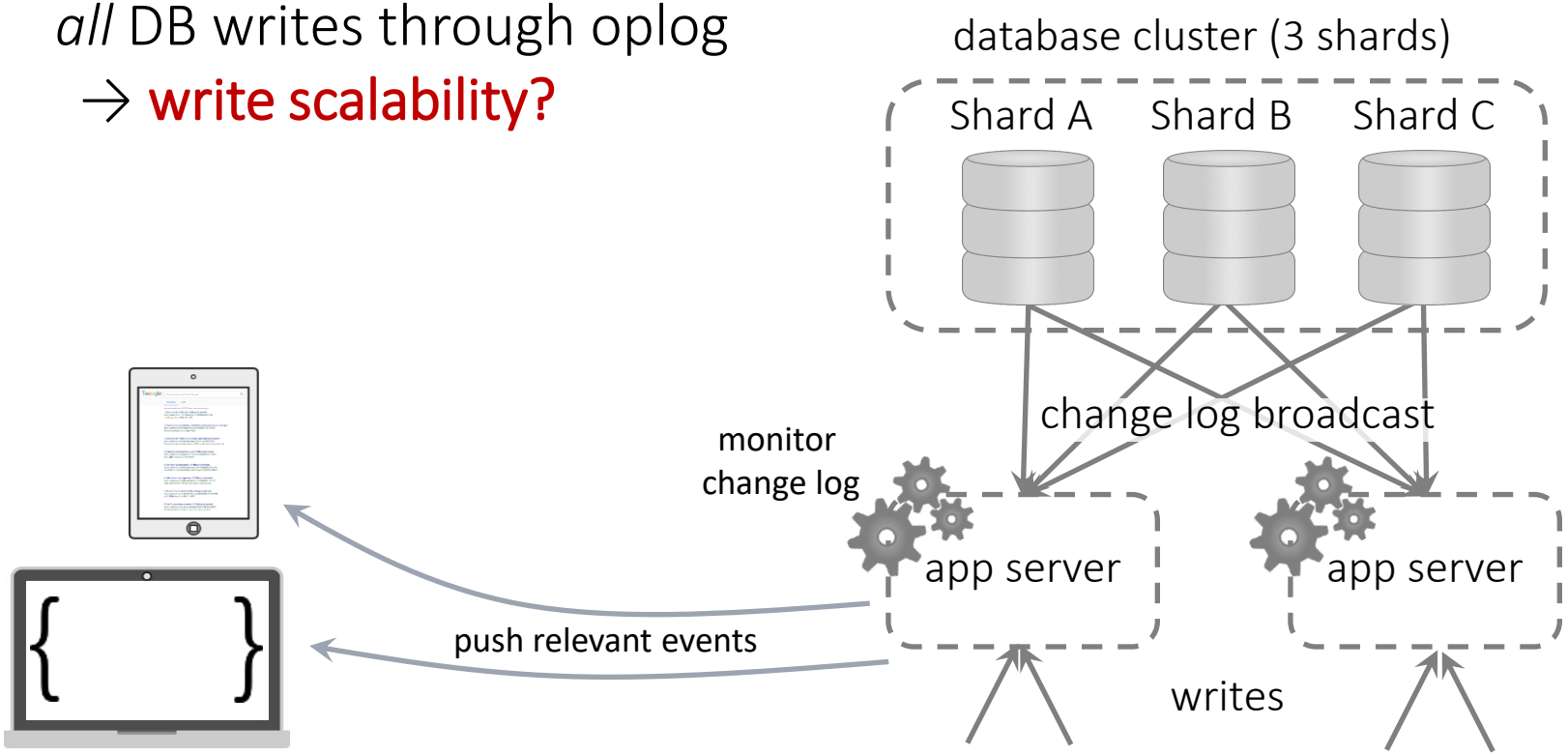
- **Local change monitoring**: app servers detect local changes
  → *incomplete* in multi-server deployment
- **Poll-and-diff**: global changes are discovered through polling
  → **staleness window**
  → **read scalability?**



Typical Maintenance Mechanisms (2/2)

Change Log Tailing

- *Every* application server receives *all* DB writes through oplog
  → write scalability?



Real-Time Database Comparison

	METEOR	RethinkDB	Parse	Firebase	InvaliDB
	Poll-and-Diff	Change Log Tailing		Unknown	2-D Partitioning
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	✓	✓	✓	✓	✓	✓

Outline

Problem Statement
Intro & Research Question

Related Work
State of the Art & Open Issues

A Scalable RTDB Design
InvaliDB: Concept & Prototype

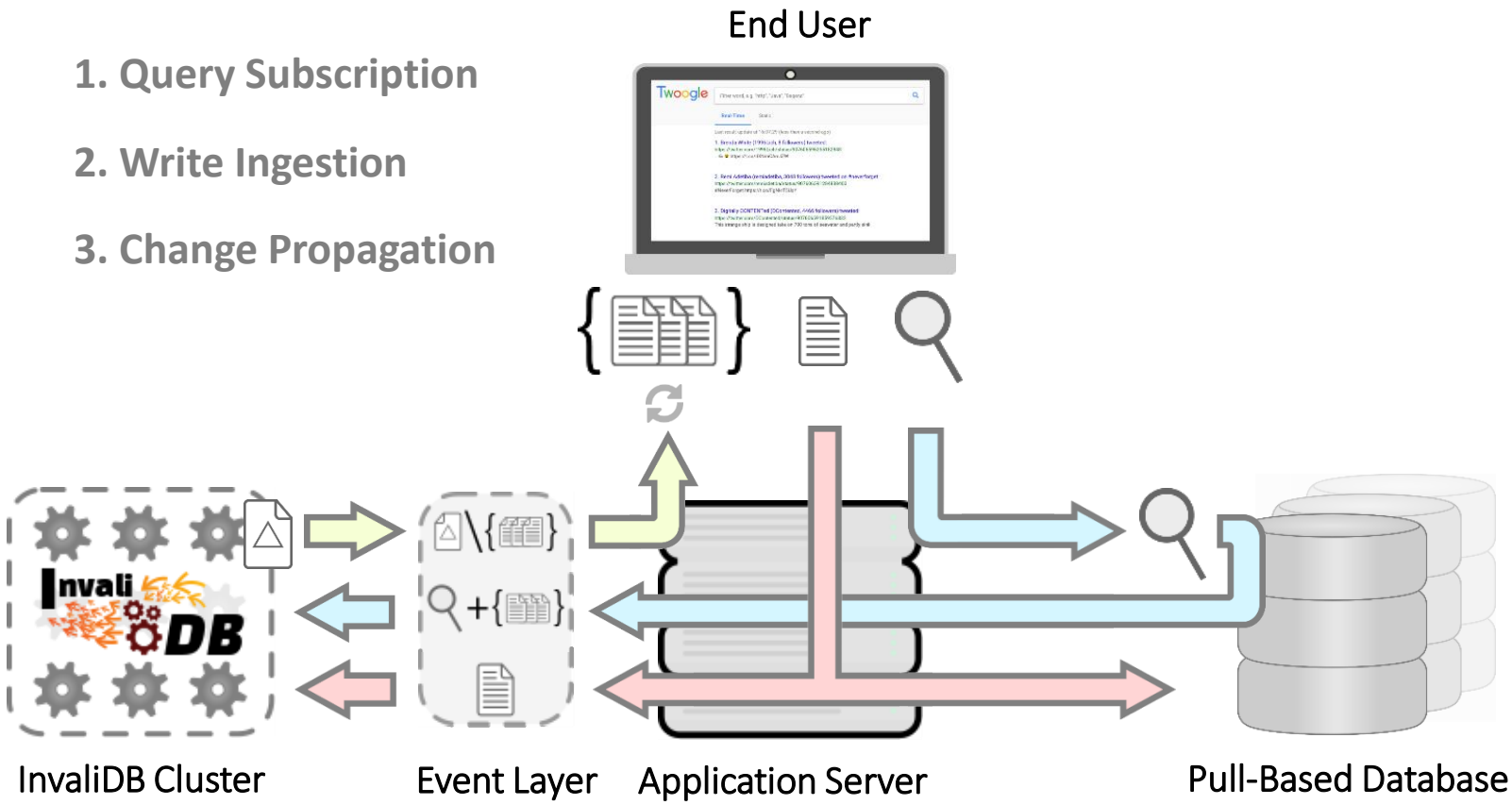
Discussion
Applications & Outlook

- **System Model & Overview**
  - Query Subscription
  - Write Ingestion
  - Change Propagation
- **Real-Time Query Processing**
  - Two-Dimensional Workload Partitioning

InvaliDB: A Scalable Real-Time Database Design

System Model & Overview

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



InvaliDB: A Scalable Real-Time Database Design

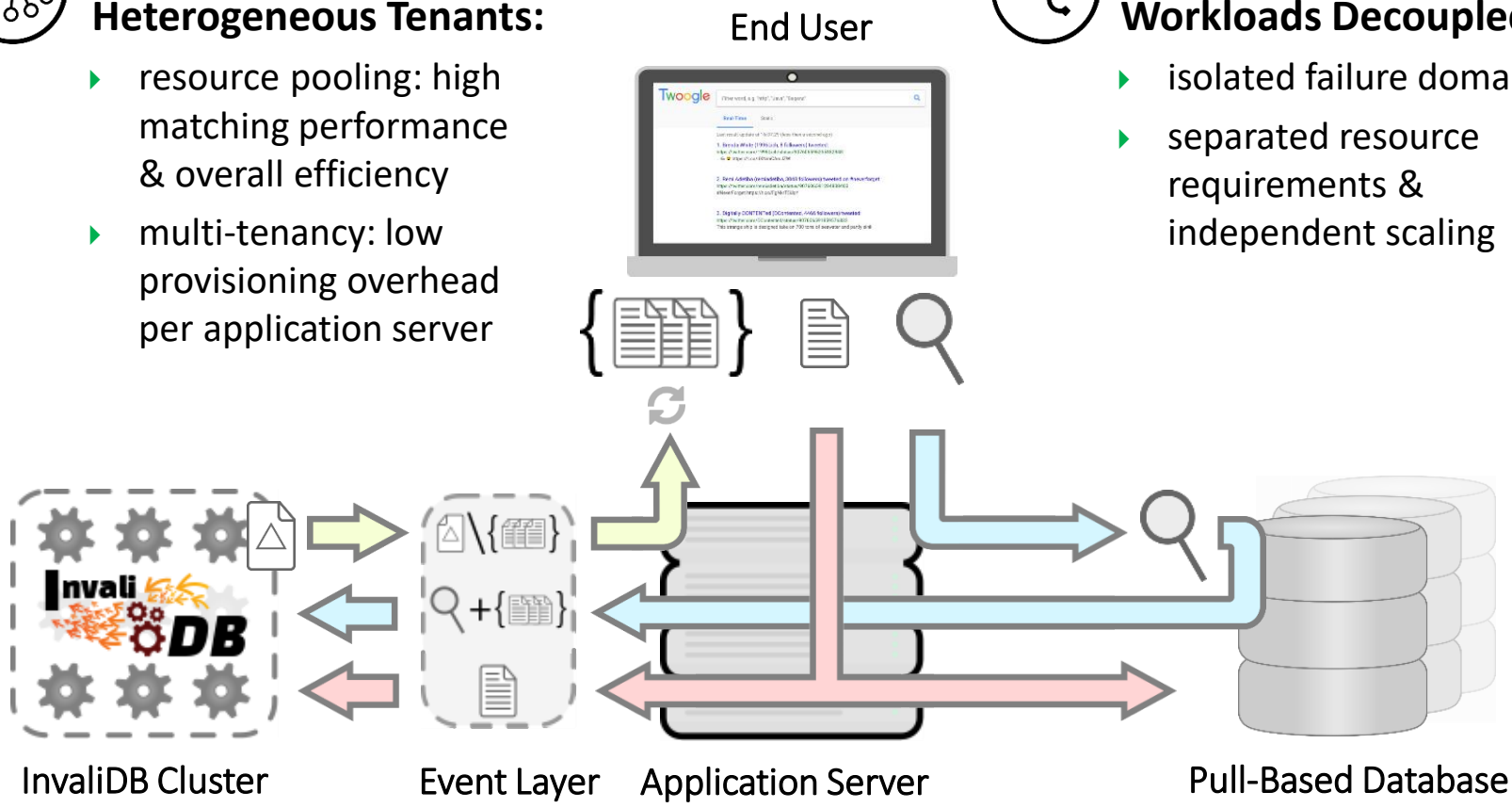
System Model & Overview

Realtime-as-a-Service For Heterogeneous Tenants:

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

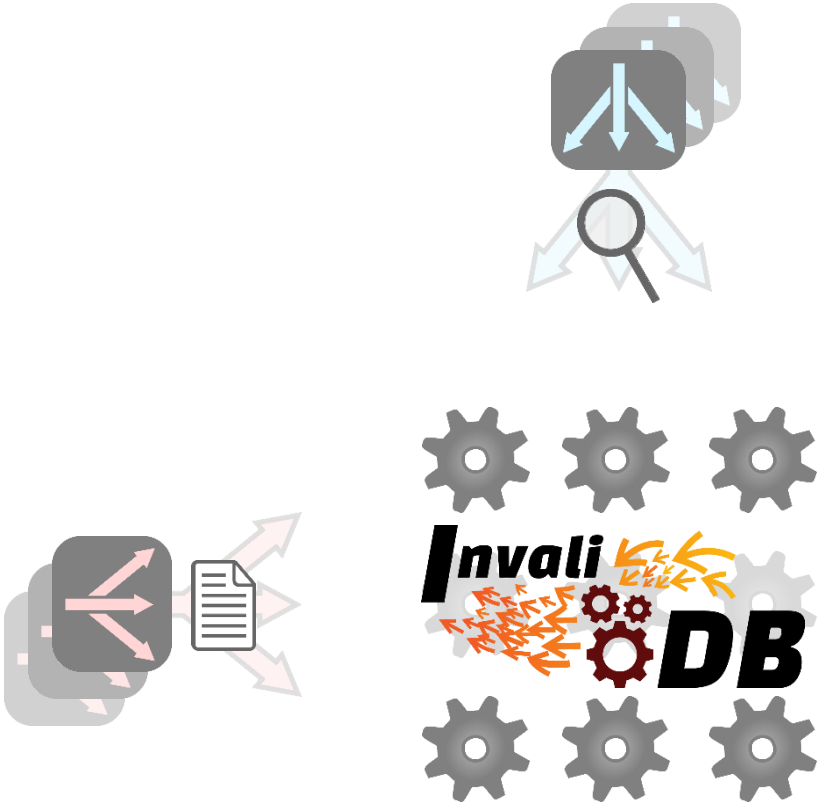
Real-Time & OLTP Workloads Decoupled:

- isolated failure domains
- separated resource requirements & independent scaling



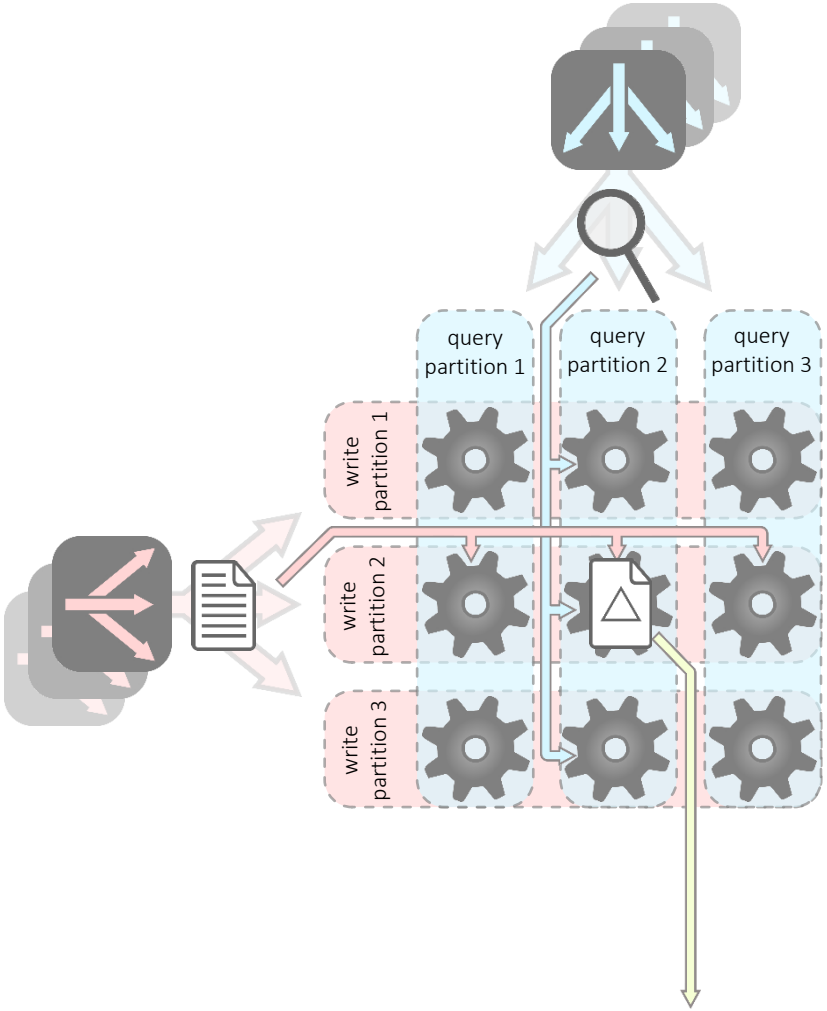
InvaliDB: A Scalable Real-Time Database Design

Two-Dimensional Workload Partitioning



InvaliDB: A Scalable Real-Time Database Design

Two-Dimensional Workload Partitioning

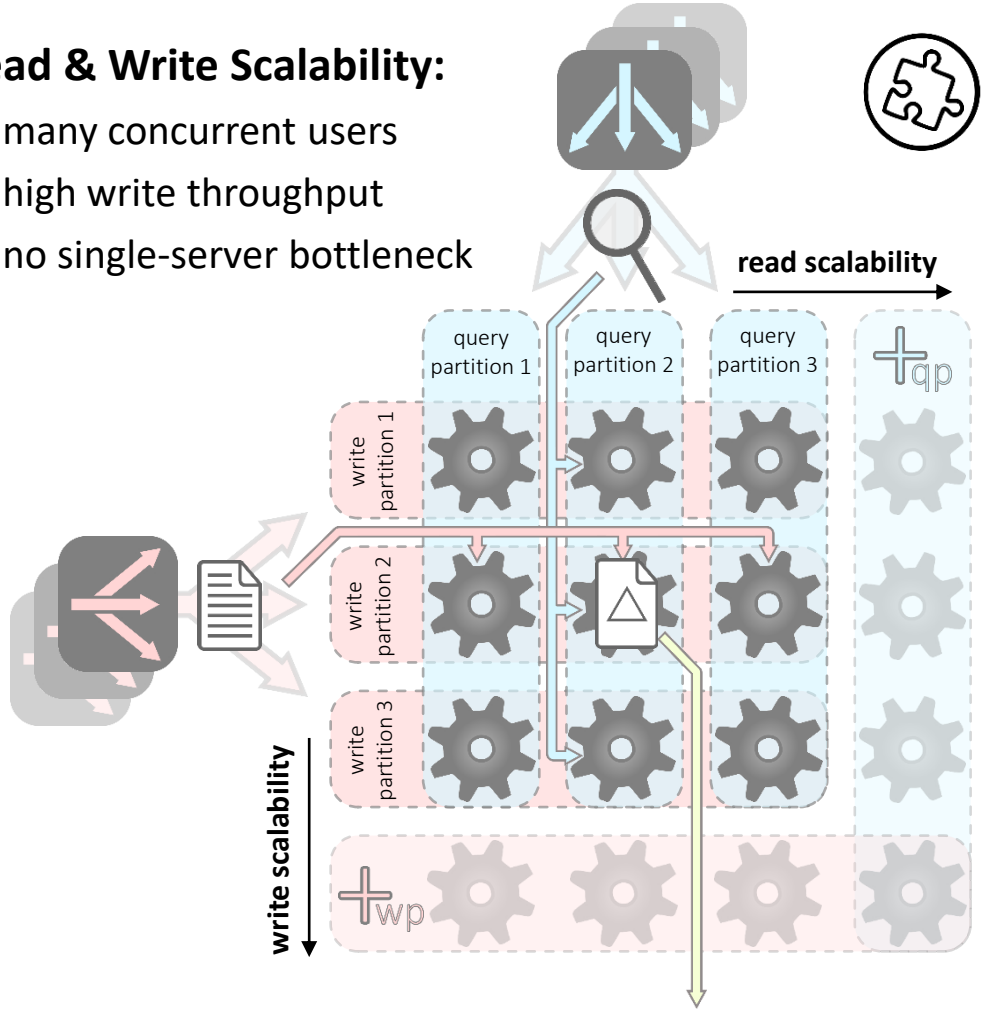


InvaliDB: A Scalable Real-Time Database Design

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

Production System

Query Processing

- low latency
- customizability
- tried & tested

Event Layer

- low latency
- high per-node throughput
- ease of deployment

Database

- typical RTDB expressiveness
- typical NoSQL datastore
- wildly popular

InvaliDB Cluster

Event Layer

Application Server

Pull-Based Database

Outline

Problem Statement
Intro & Research Question

Related Work
State of the Art & Open Issues

A Scalable RTDB Design
InvaliDB: Concept & Prototype

Discussion
Applications & Outlook

- **Application Scenarios**
  1. Real-Time Queries
  2. Query Caching
- **Future Work**
  - Extending Semantics
  - Trade-Offs & Tuning
  - New Use Cases
- **Summary & Contributions**

Use Case 1: Real-Time Queries

An Easy-to-Use JavaScript API

```
var query = DB.Tweet.find()
              .matches('text', /my filter/)
              .descending('createdAt')
              .limit(10)
              .offset(20);
```

Pull-Based Query

```
query.resultList(result => ...);
```

Google

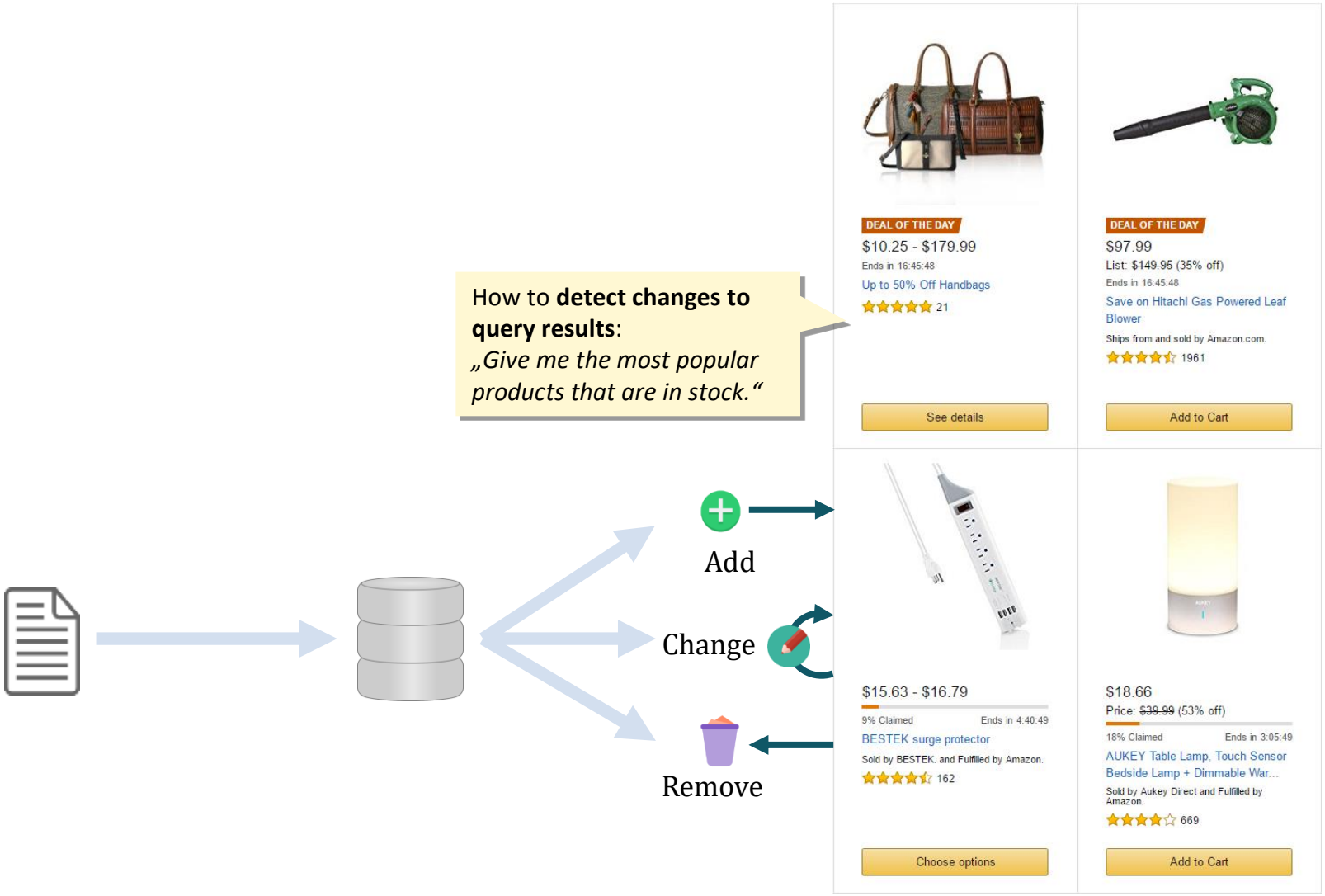
Real-Time Query

```
query.resultStream(result => ...);
```

Twoogle

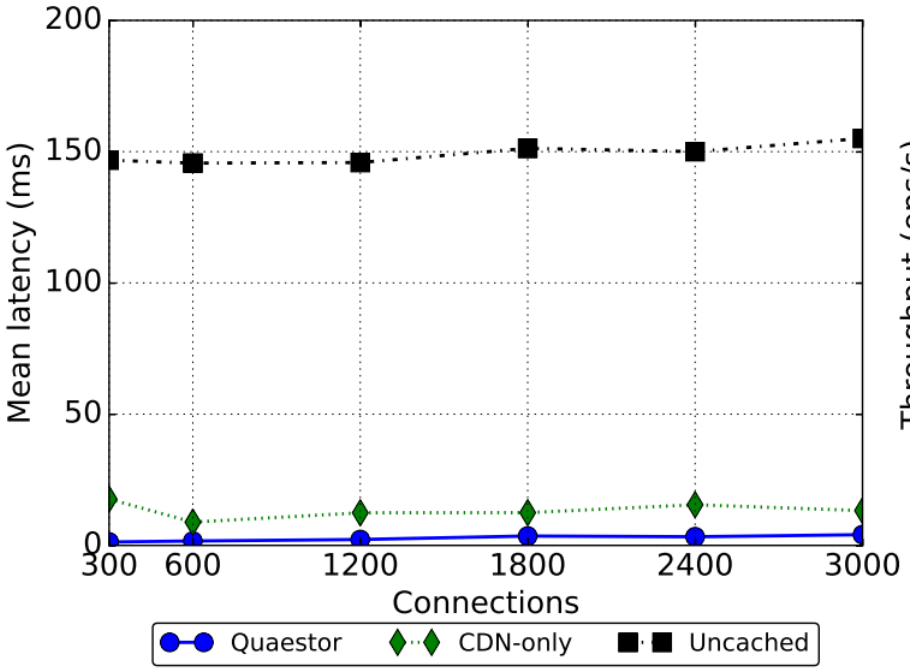
Use Case 2: Consistent Query Caching

InvaliDB For Invalidating DB Queries

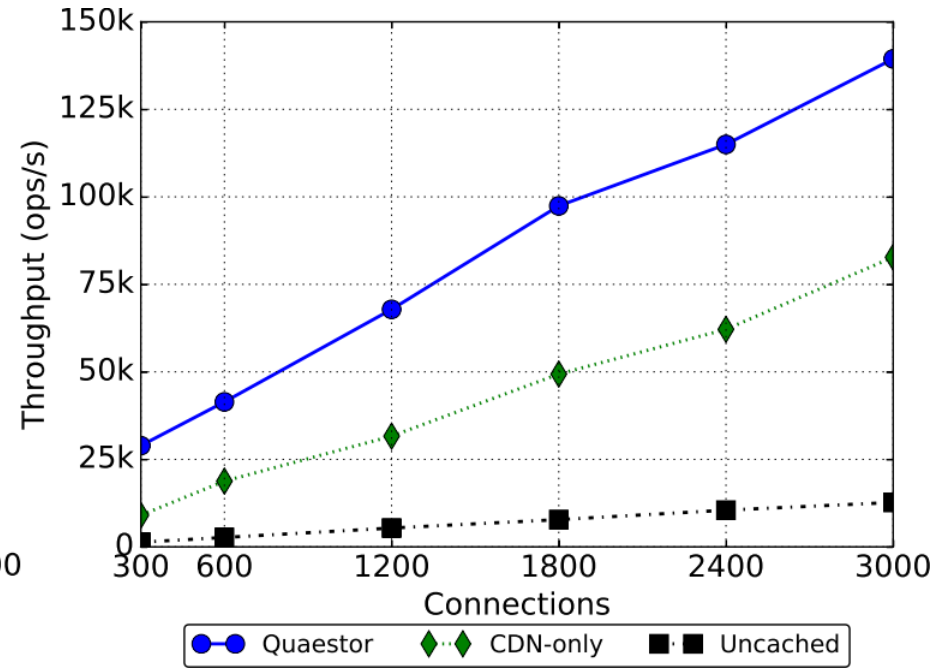


Use Case 2: Consistent Query Caching

Improving Pull-Based Query Performance



Latency

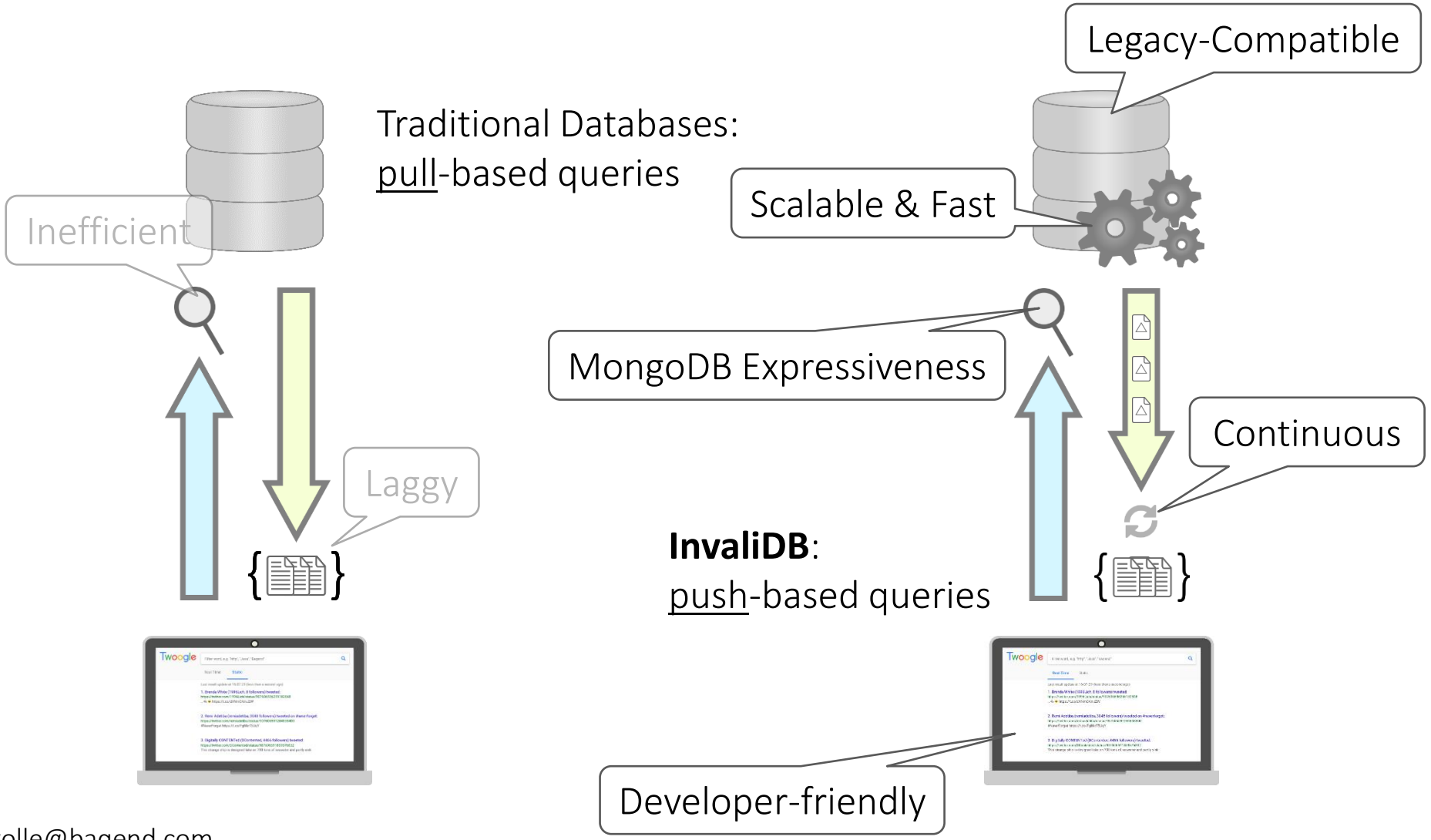


Throughput

Quaestor: *Query Web Caching for Database-as-a-Service Providers*, VLDB '17.

InvaliDB

A Scalable Real-Time Database Design



wolle@baqend.com