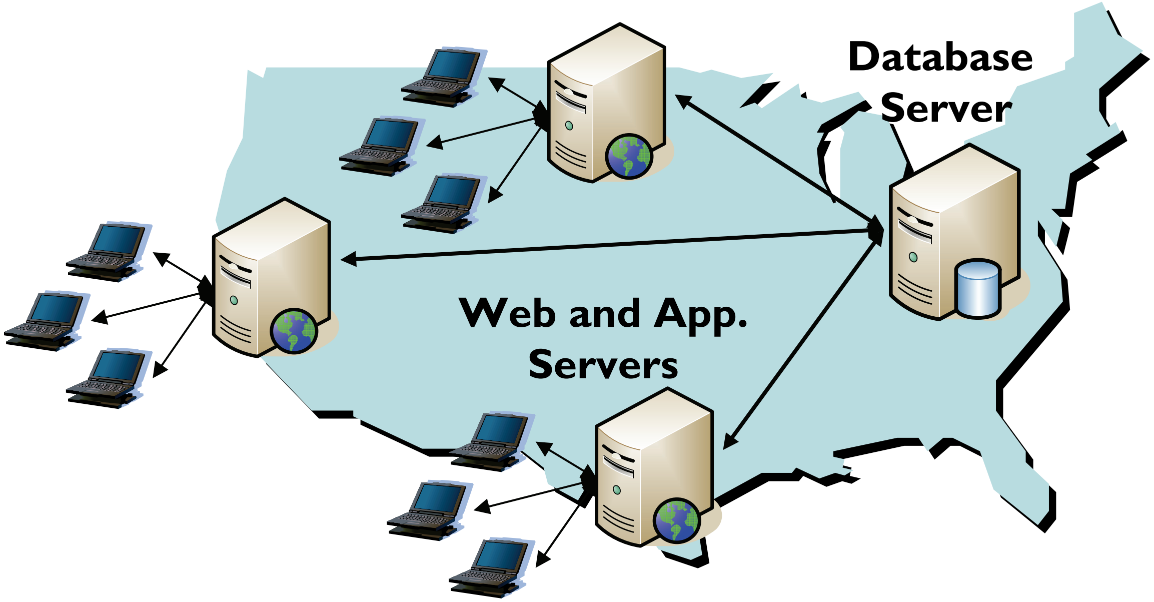
Consistency-preserving Caching of Dynamic Database Content

Niraj Tolia
M. Satyanarayanan

Carnegie Mellon University

Motivation



- Easy to geographically distribute web and app. servers
- Harder to distribute databases
  - Pick Two: Consistency, Availability, Tolerance to Partitions
- How can you optimize the use of the WAN?

Outline

- Motivation
- Ganesh
  - Overview
  - Design and Implementation
- Evaluation
- Conclusion

High-level Overview of Ganesh

- Optimizes WAN usage for database accesses
  - Without interpreting queries or results
  - In a database-independent manner
- Uses a Content Addressable cache to
  - Detect similarity between query results
  - Eliminate redundancy over WAN link

- Tradeoff increased computation for network savings

Finding Result Similarity

(A 10,000 foot view)

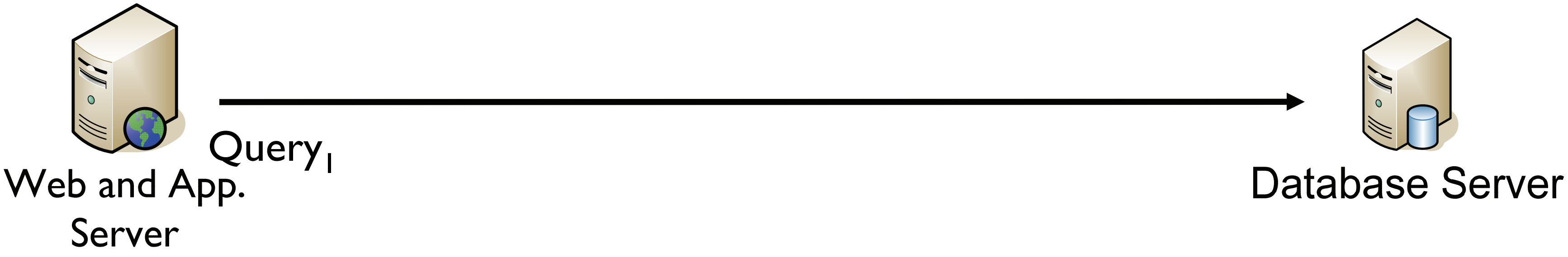
Web and App.
Server

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)



Web and App. Server

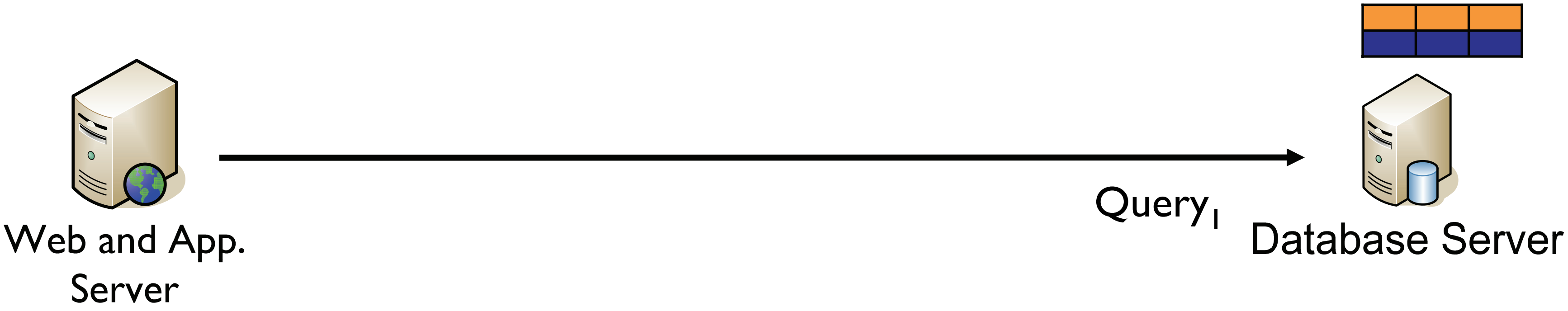
Query_1

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)



Web and App. Server

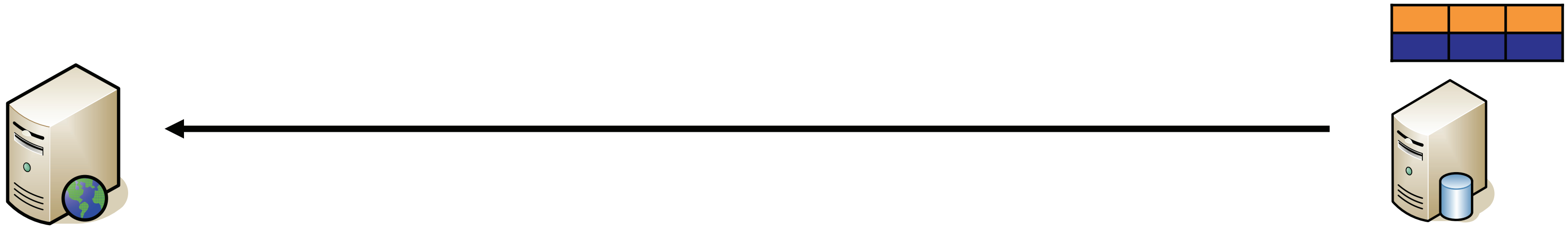
Query$_1$

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)



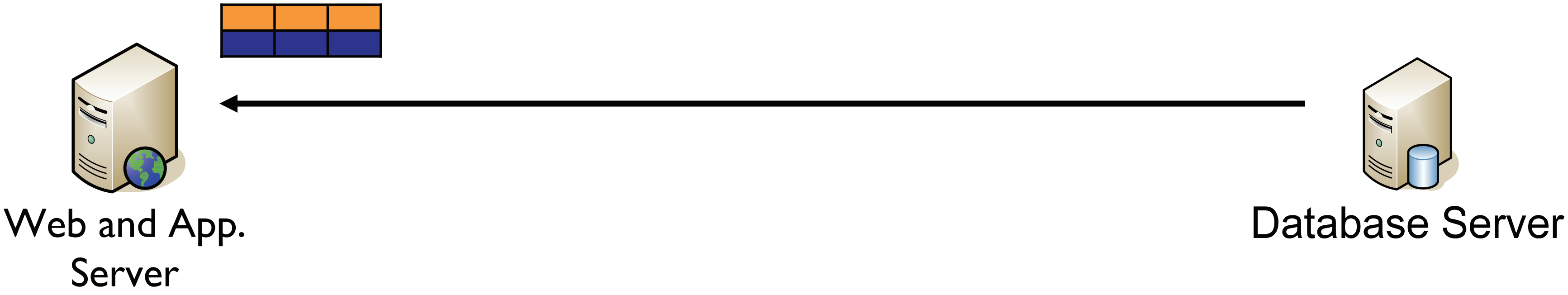
Web and App. Server

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)



Web and App.
Server

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)

Web and App. Server

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)

Web and App.
Server

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)

Web and App.
Server

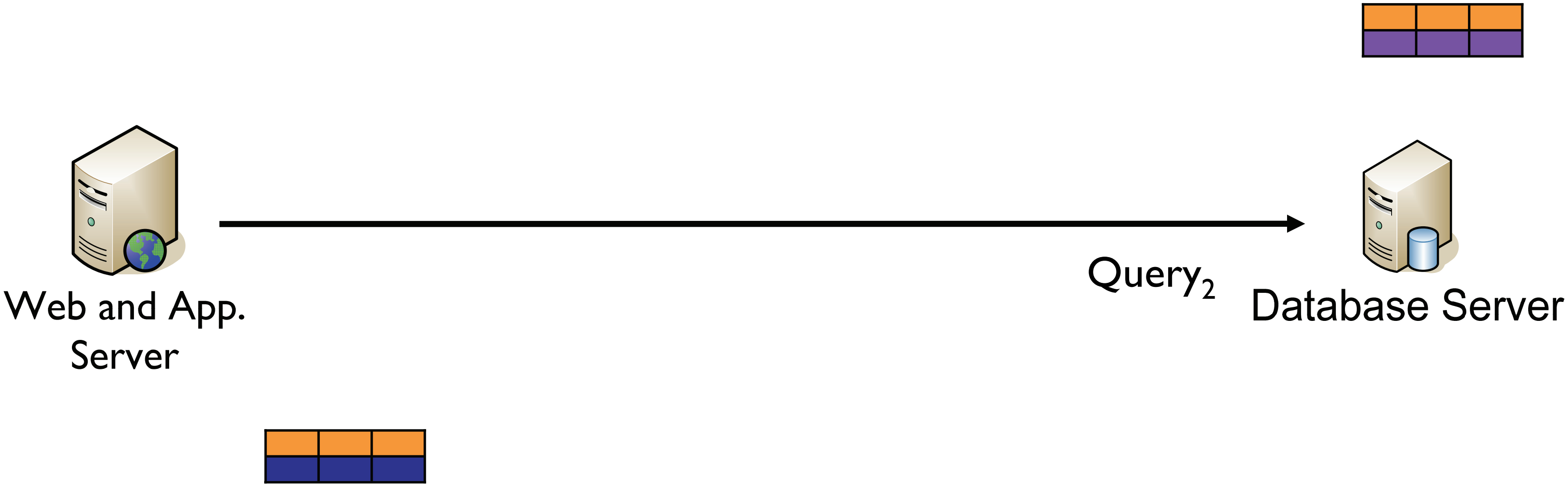
Query$_2$

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)



Web and App.
Server

Query$_2$

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)

SELECT queries

Finding Result Similarity

(A 10,000 foot view)

`Data ID`

Web and App. Server

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)

SELECT queries

Finding Result Similarity

(A 10,000 foot view)

Web and App. Server

Database Server

SELECT queries

Finding Result Similarity

(A 10,000 foot view)

SELECT queries

UPDATE queries are not optimized

Design Goals

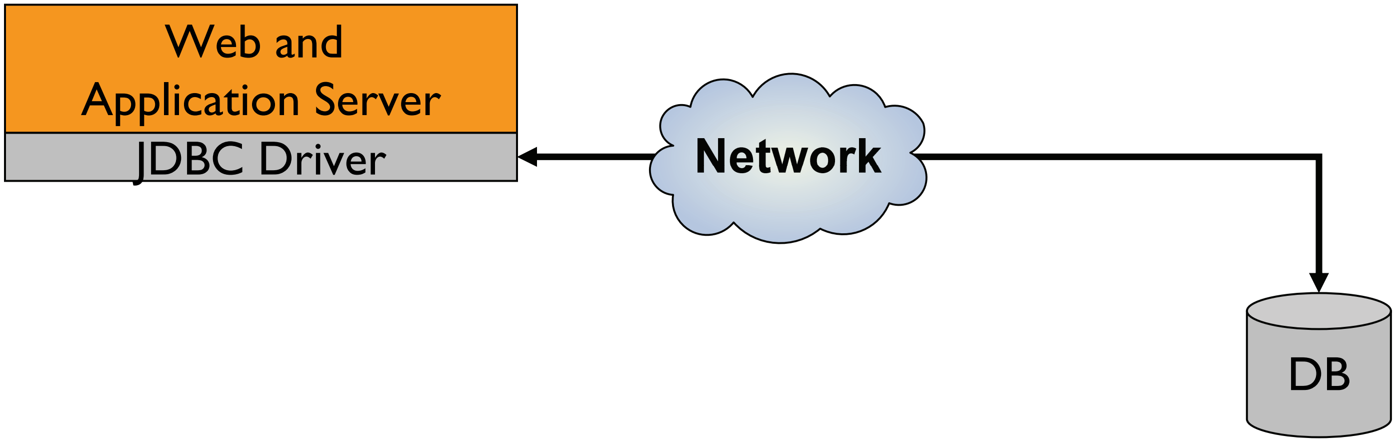
- Transparent
  - To both the application server and the database
- Does not weaken consistency
- Efficiently detects similarity

Design - Transparency

- Doesn’t require modifications to the application and database server

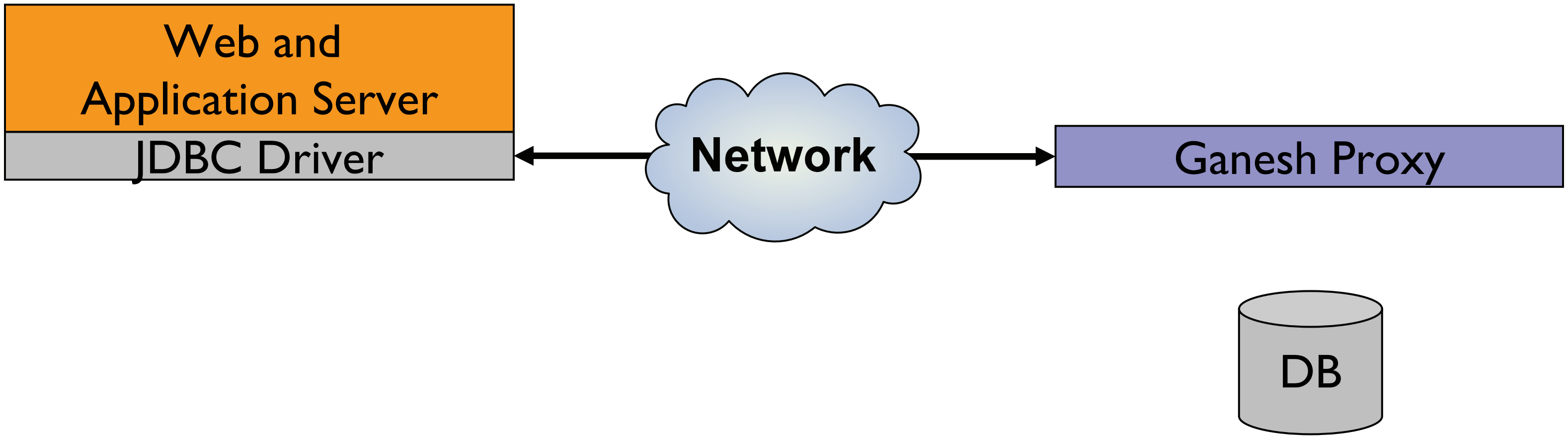
Design - Transparency

- Doesn't require modifications to the application and database server



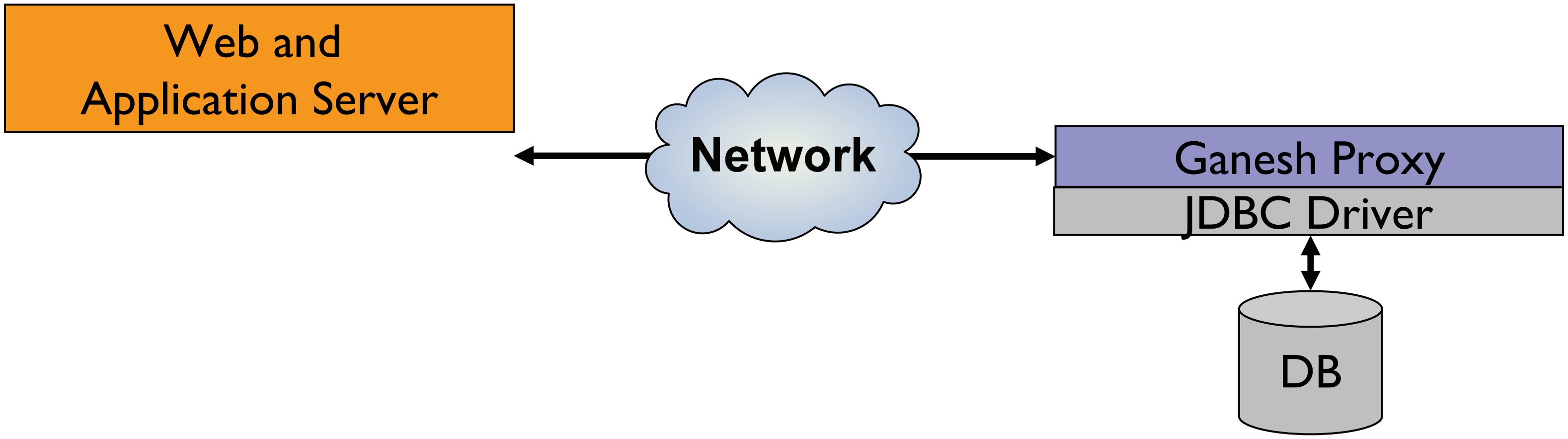
Design - Transparency

- Doesn't require modifications to the application and database server



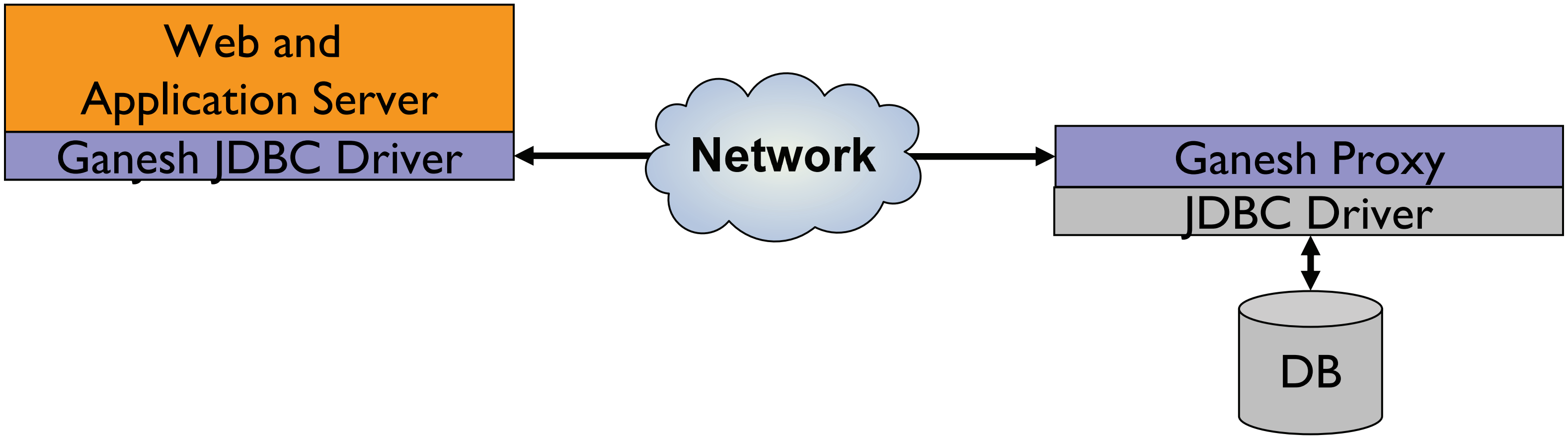
Design - Transparency

- Doesn’t require modifications to the application and database server



Design - Transparency

- Doesn't require modifications to the application and database server

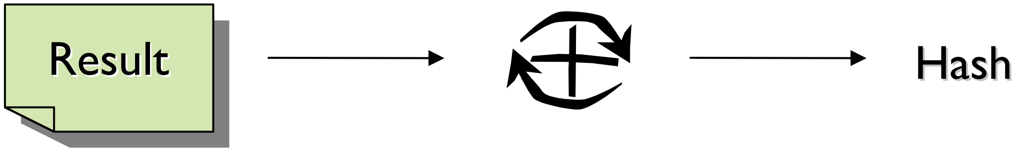


Proxy-based Architecture

- Ganesh JDBC Driver
  - Thin but smart -- conservative network use
  - Contains in-memory cache
    - Caches previous results at different granularities
    - Uses a LRU replacement policy
- Ganesh Proxy
  - Forwards queries to the database
  - Eliminates redundancy in results

Detecting Similarity

- Ganesh uses Content Addressability



Cryptographic Hash

- Hash value is a globally unique identifier
  - Independent of any particular system
  - Infeasible to find another object with same hash
  - If hash values are equal, so are the source objects
- Any small change in source completely changes hash

Exploiting Structure

- Exploit structure in results – they look like tables

```
SELECT  name, address, zip, email  FROM USERS
```

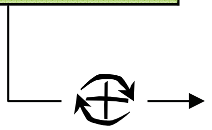
Name	Address	Zip	Email
John Doe	412 Avenue	15213	jd2@eg.com
Mary Major	821 Lane	15232	mm@eg.com
John Stiles	701 Street	T1L1J4	js@eg.com

Exploiting Structure

- Exploit structure in results – they look like tables

```
SELECT  name, address, zip, email  FROM USERS
```

Name	Address	Zip	Email
John Doe	412 Avenue	15213	jd2@eg.com
Mary Major	821 Lane	15232	mm@eg.com
John Stiles	701 Street	T1L1J4	js@eg.com



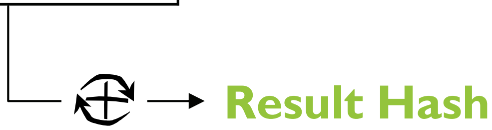
Result Hash

Exploiting Structure

- Exploit structure in results – they look like tables

SELECT name, address, zip, email FROM USERS

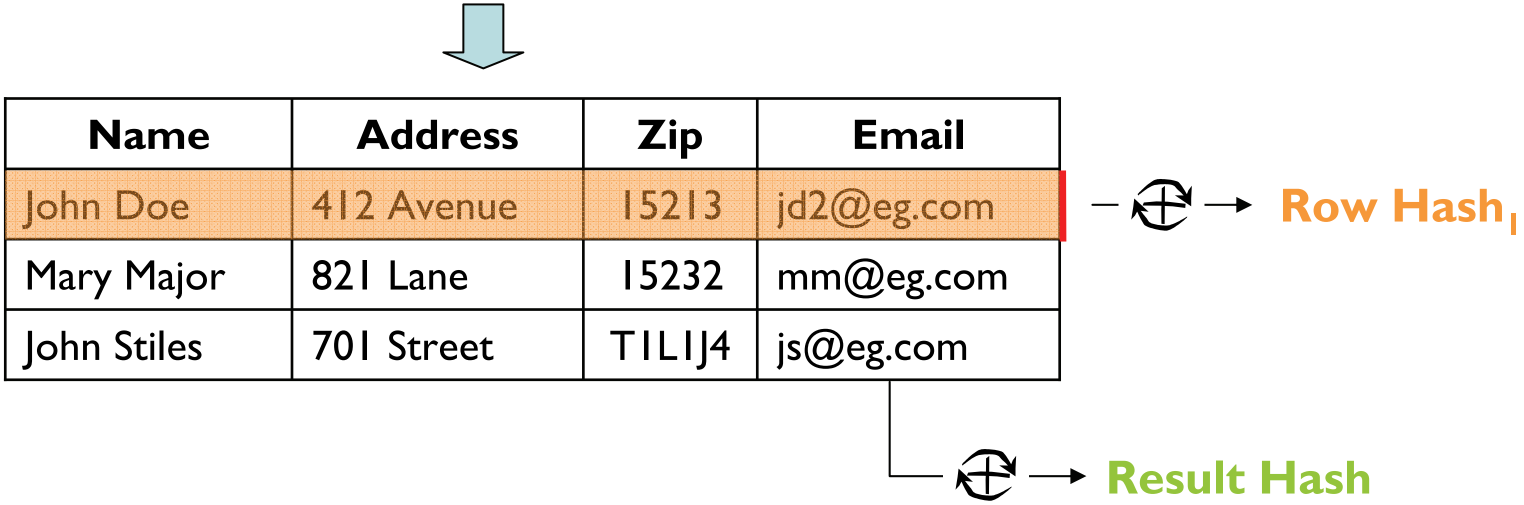
Name	Address	Zip	Email
John Doe	412 Avenue	15213	jd2@eg.com
Mary Major	821 Lane	15232	mm@eg.com
John Stiles	701 Street	T1L1J4	js@eg.com


Result Hash

Exploiting Structure

- Exploit structure in results – they look like tables

SELECT name, address, zip, email FROM USERS



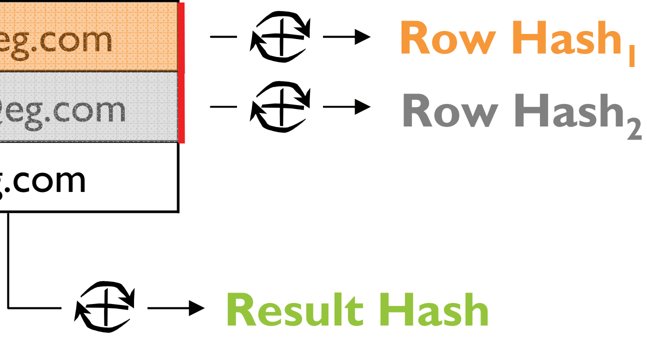
Name	Address	Zip	Email
John Doe	412 Avenue	15213	jd2@eg.com
Mary Major	821 Lane	15232	mm@eg.com
John Stiles	701 Street	T1L1J4	js@eg.com

Exploiting Structure

- Exploit structure in results – they look like tables

SELECT name, address, zip, email FROM USERS

Name	Address	Zip	Email
John Doe	412 Avenue	15213	jd2@eg.com
Mary Major	821 Lane	15232	mm@eg.com
John Stiles	701 Street	T1L1J4	js@eg.com



Row Hash$_1$

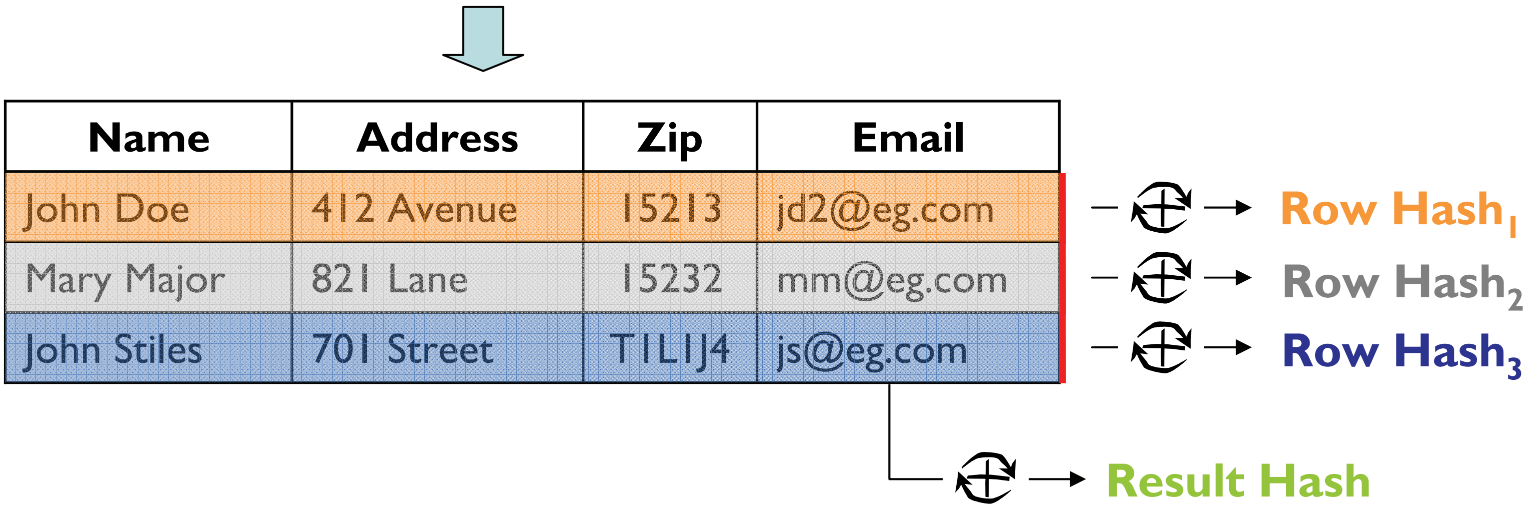
Row Hash$_2$

Result Hash

Exploiting Structure

- Exploit structure in results – they look like tables

SELECT name, address, zip, email FROM USERS



Name	Address	Zip	Email
John Doe	412 Avenue	15213	jd2@eg.com
Mary Major	821 Lane	15232	mm@eg.com
John Stiles	701 Street	T1L1J4	js@eg.com

Row Hash$_1$

Row Hash$_2$

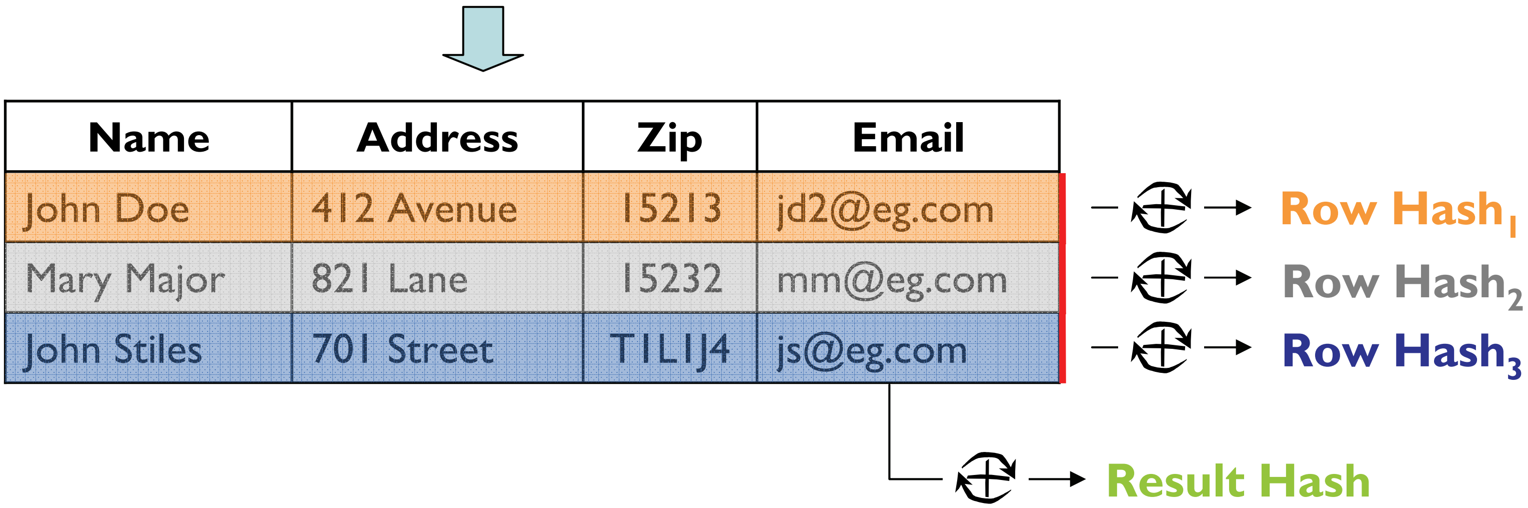
Row Hash$_3$

Result Hash

Exploiting Structure

- Exploit structure in results – they look like tables

SELECT name, address, zip, email FROM USERS



Name	Address	Zip	Email
John Doe	412 Avenue	15213	jd2@eg.com
Mary Major	821 Lane	15232	mm@eg.com
John Stiles	701 Street	T1L1J4	js@eg.com

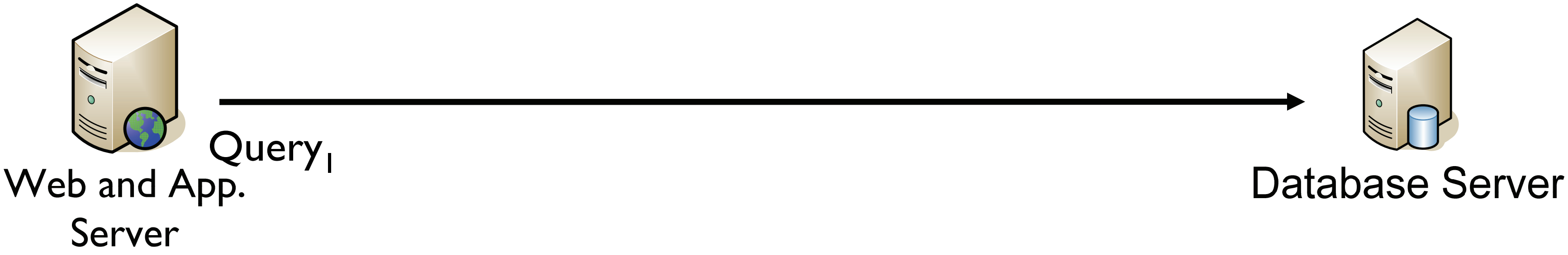
- This process does not interpret data

Putting it all together

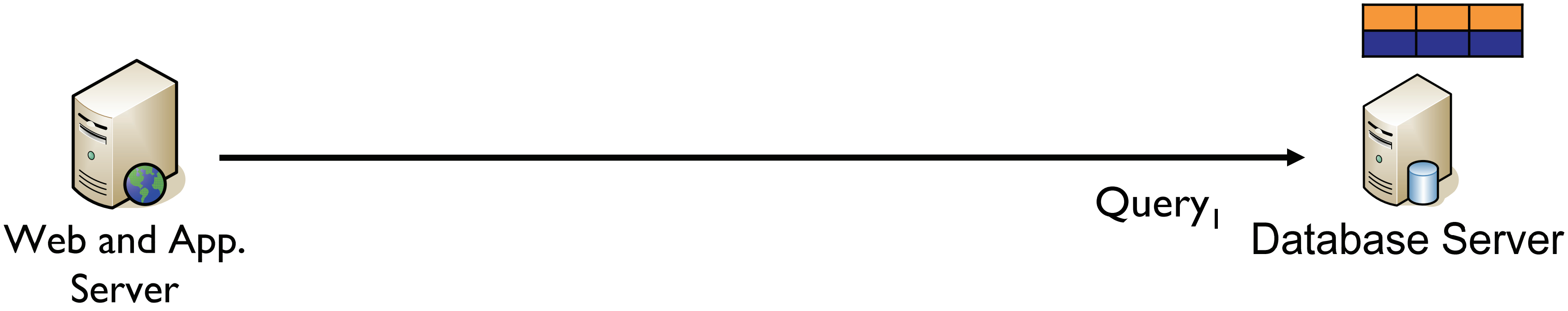
Web and App.
Server

Database Server

Putting it all together



Putting it all together

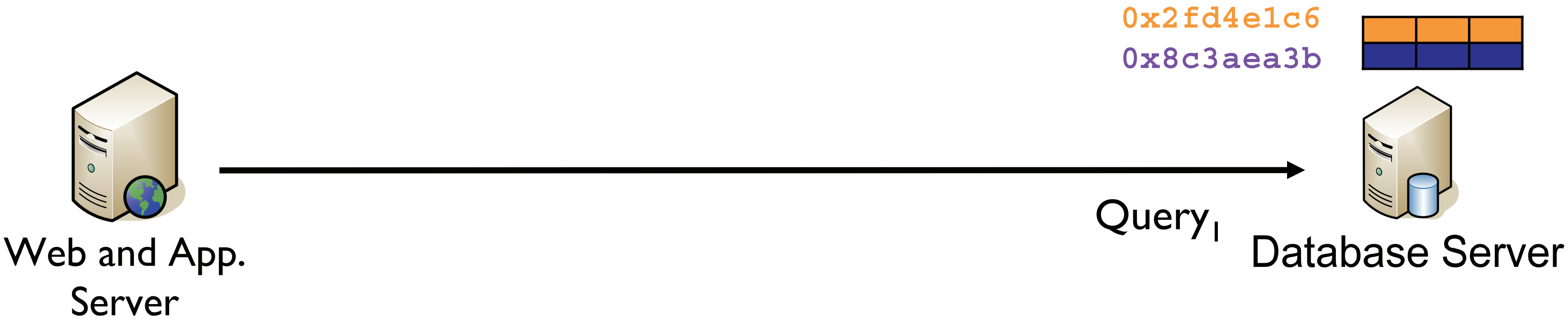


Web and App. Server

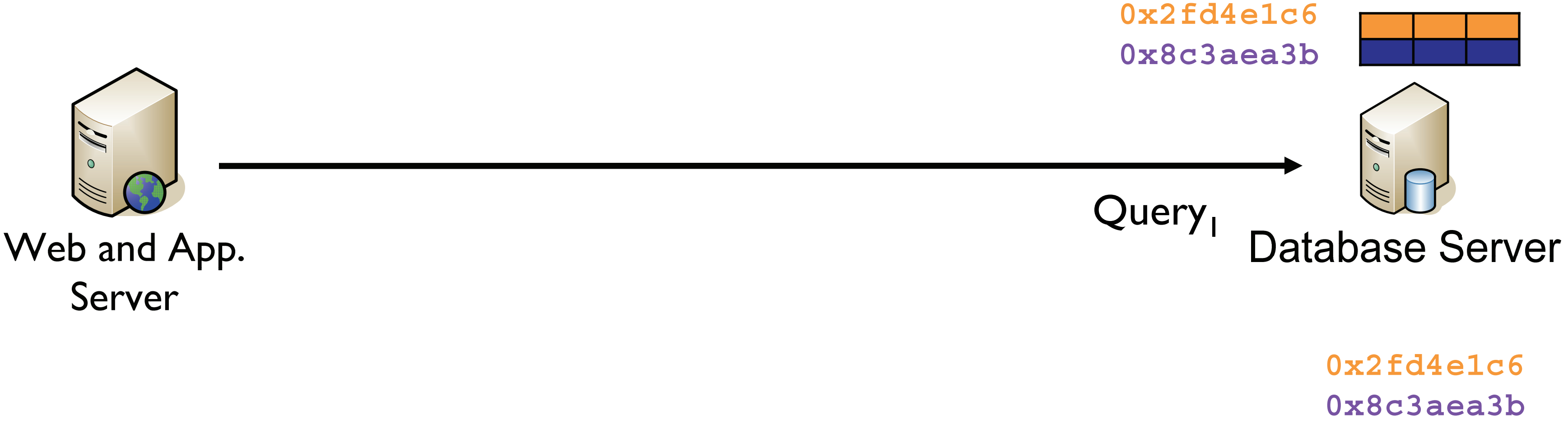
Query$_1$

Database Server

Putting it all together



Putting it all together



0x2fd4e1c6

0x8c3aea3b

Web and App. Server

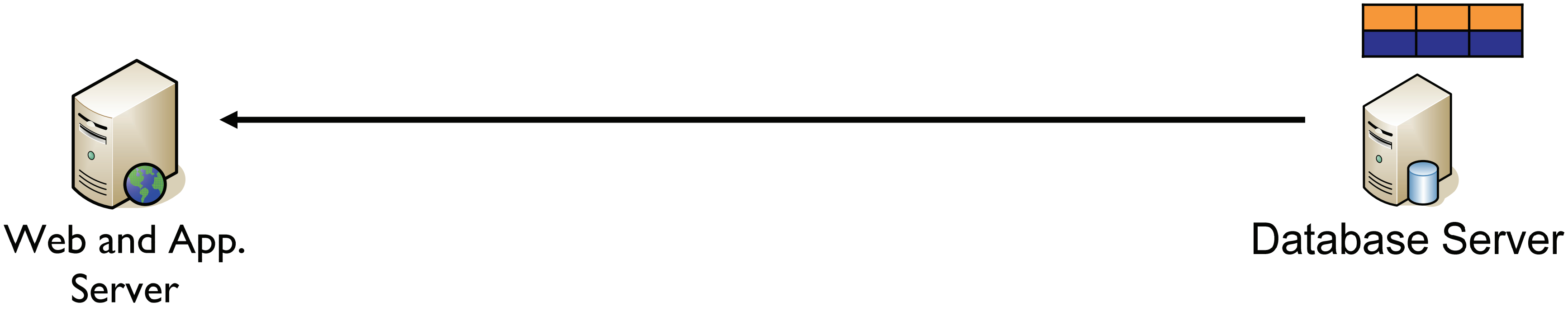
Query_1

Database Server

0x2fd4e1c6

0x8c3aea3b

Putting it all together



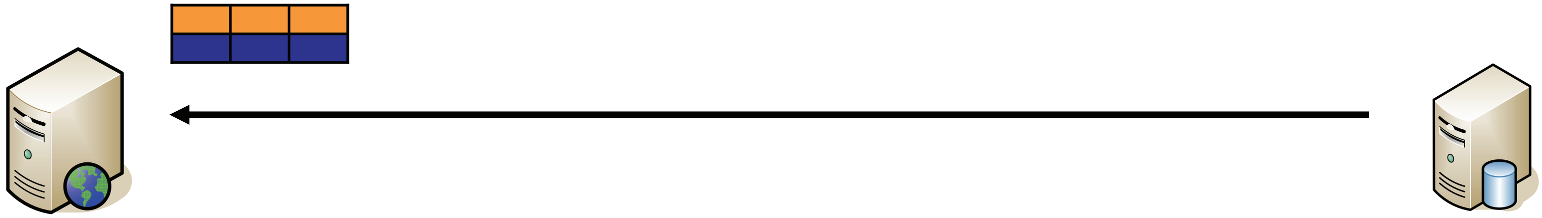
Web and App. Server

Database Server

`0x2fd4e1c6`

`0x8c3aea3b`

Putting it all together



Web and App.
Server

Database Server

`0x2fd4e1c6`
`0x8c3aea3b`

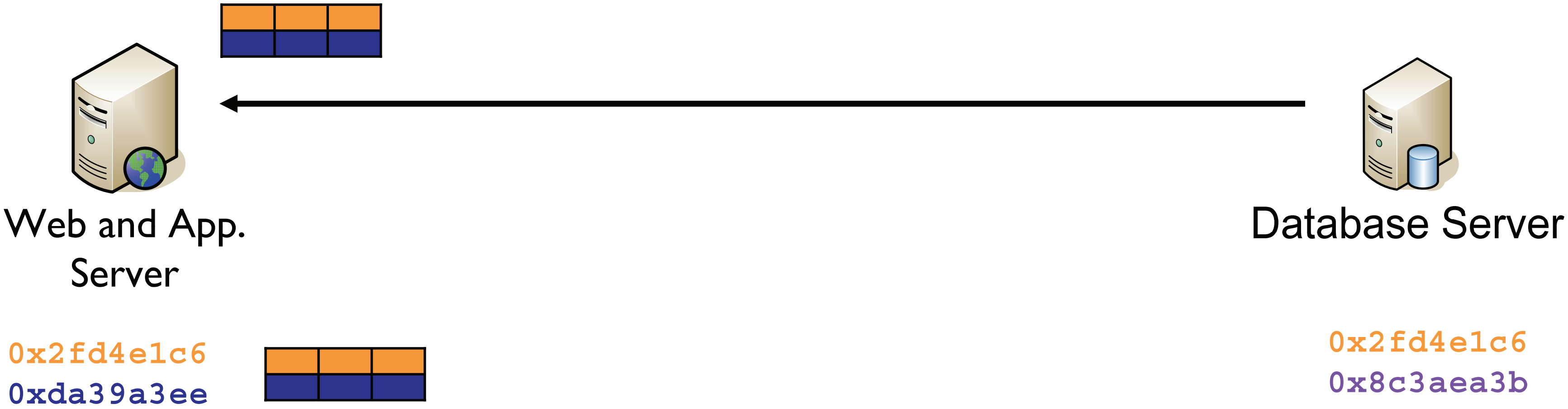
Putting it all together

Web and App.
Server

Database Server

0x2fd4e1c6
0x8c3aea3b

Putting it all together



Web and App. Server

Database Server

0x2fd4e1c6
0xda39a3ee

0x2fd4e1c6
0x8c3aea3b

Putting it all together

Web and App. Server

Database Server

0x2fd4e1c6
0xda39a3ee

0x2fd4e1c6
0x8c3aea3b

Putting it all together

Query$_2$

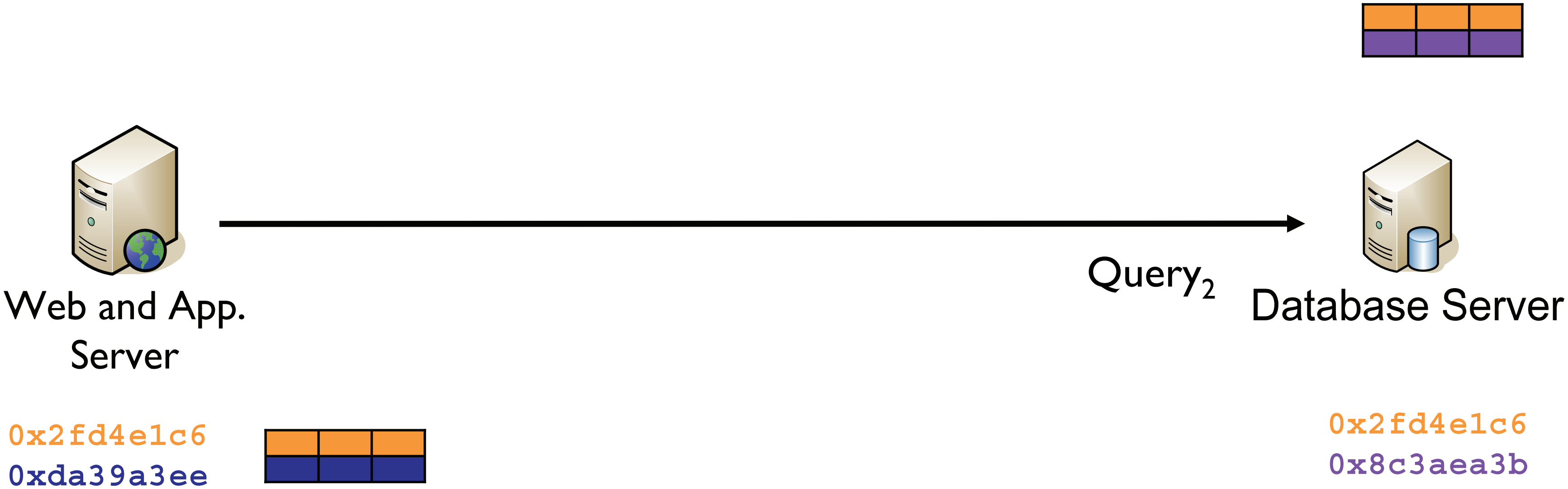
Web and App.
Server

0x2fd4e1c6
0xda39a3ee

Database Server

0x2fd4e1c6
0x8c3aea3b

Putting it all together



Web and App. Server

Query$_2$

Database Server

`0x2fd4e1c6`
`0xda39a3ee`

`0x2fd4e1c6`
`0x8c3aea3b`

Putting it all together

0x2fd4e1c6
0x8c3aea3b

Web and App. Server

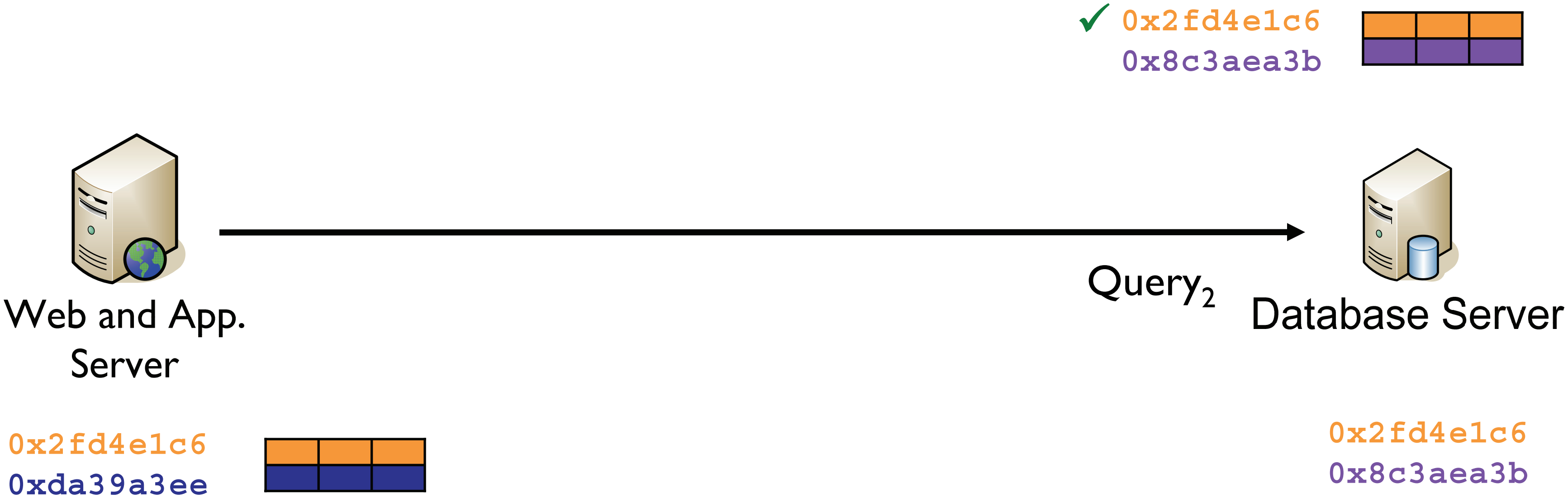
Query$_2$

Database Server

0x2fd4e1c6
0xda39a3ee

0x2fd4e1c6
0x8c3aea3b

Putting it all together



✓ 0x2fd4e1c6
0x8c3aea3b

Web and App. Server

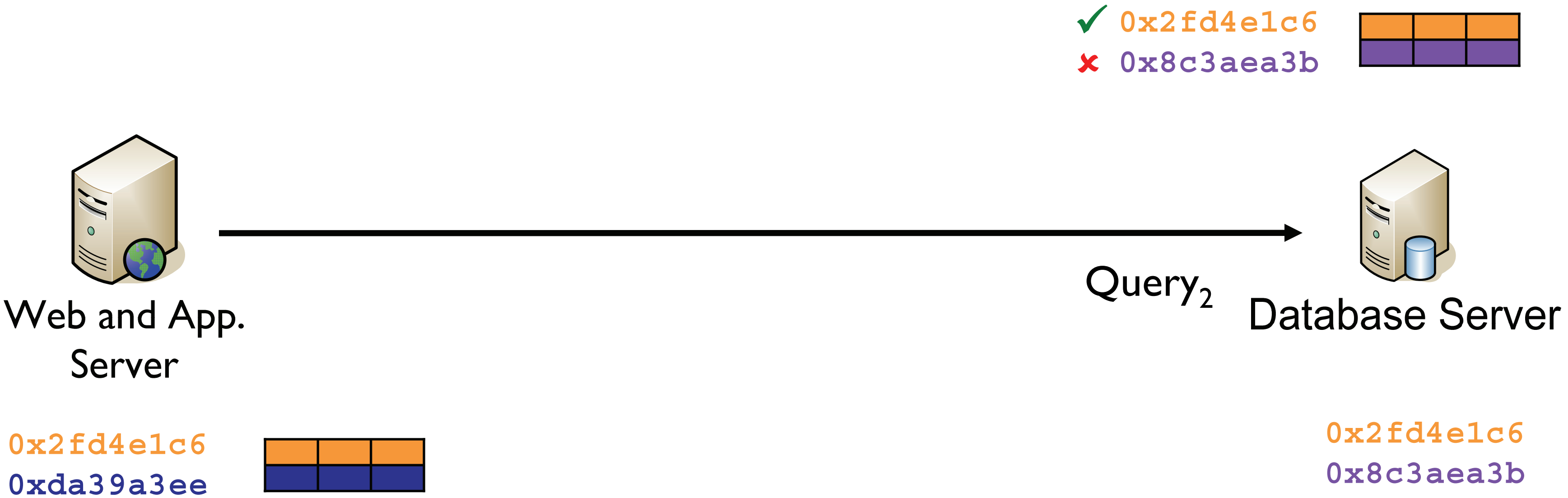
Query$_2$

Database Server

0x2fd4e1c6
0xda39a3ee

0x2fd4e1c6
0x8c3aea3b

Putting it all together



✓ 0x2fd4e1c6

✗ 0x8c3aea3b

Web and App.
Server

Query$_2$

Database Server

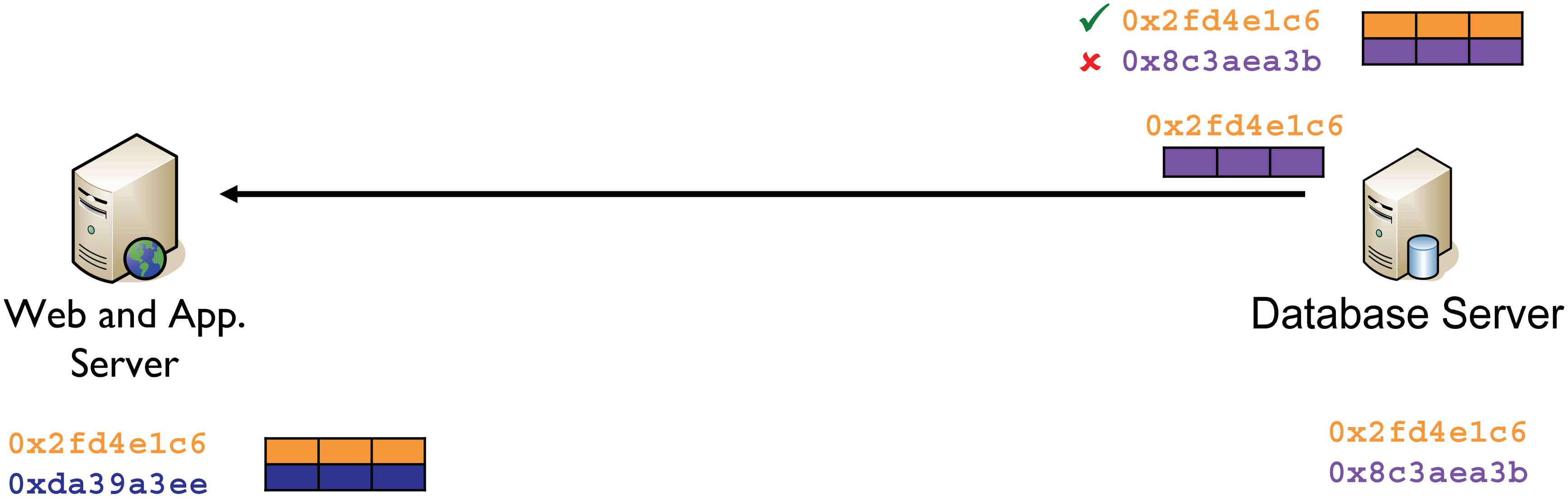
0x2fd4e1c6

0xda39a3ee

0x2fd4e1c6

0x8c3aea3b

Putting it all together



✓ 0x2fd4e1c6

✗ 0x8c3aea3b

0x2fd4e1c6

Web and App. Server

Database Server

0x2fd4e1c6
0xda39a3ee

0x2fd4e1c6
0x8c3aea3b

Putting it all together

✓ 0x2fd4e1c6

✗ 0x8c3aea3b

0x2fd4e1c6

Web and App. Server

Database Server

0x2fd4e1c6

0xda39a3ee

0x2fd4e1c6

0x8c3aea3b

Putting it all together

✓ 0x2fd4e1c6
✗ 0x8c3aea3b

Web and App. Server

Database Server

0x2fd4e1c6
0xda39a3ee

0x2fd4e1c6
0x8c3aea3b

Putting it all together

✓ 0x2fd4e1c6
✗ 0x8c3aea3b

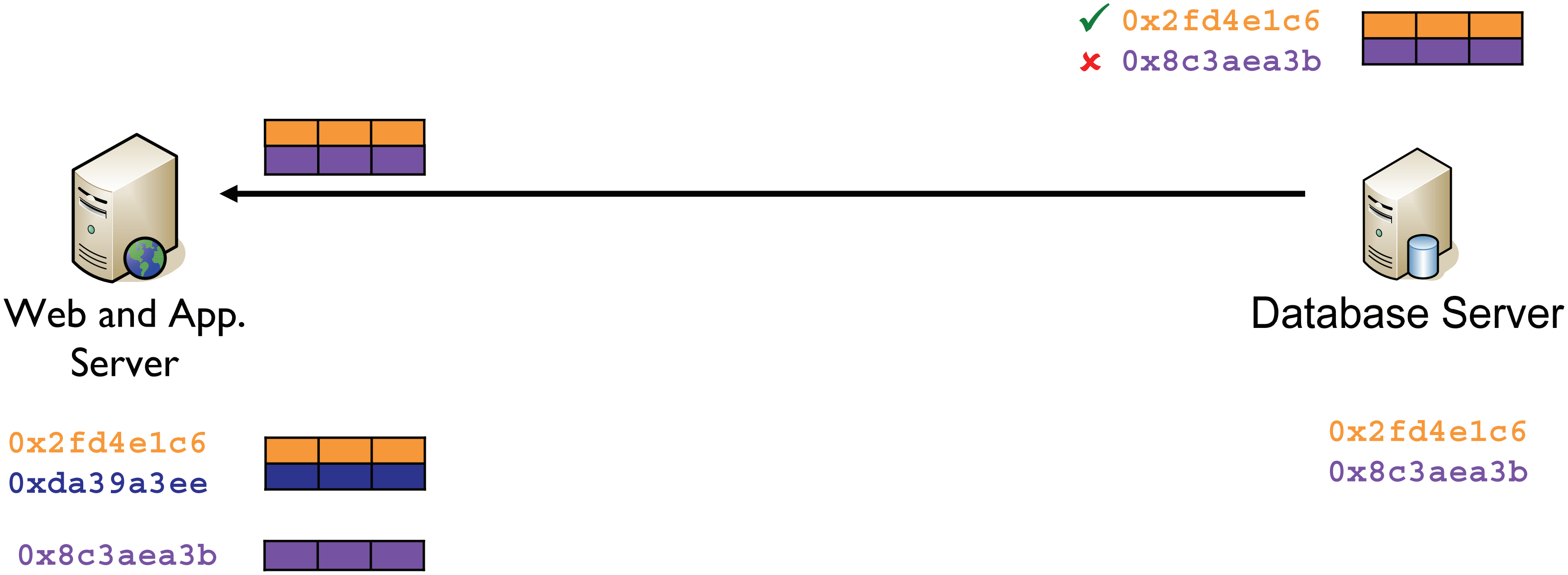
Web and App.
Server

Database Server

0x2fd4e1c6
0xda39a3ee

0x2fd4e1c6
0x8c3aea3b

Putting it all together



✓ 0x2fd4e1c6
✗ 0x8c3aea3b

Web and App. Server

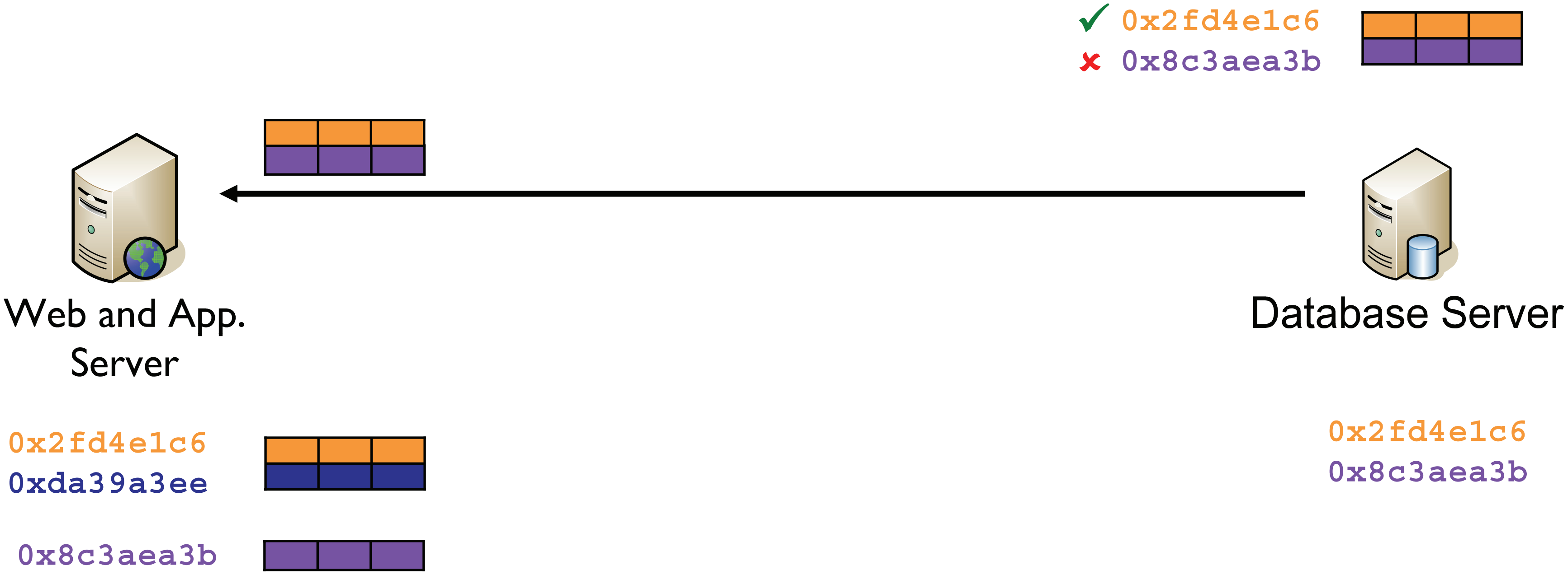
Database Server

0x2fd4e1c6
0xda39a3ee

0x8c3aea3b

0x2fd4e1c6
0x8c3aea3b

Putting it all together



✓ 0x2fd4e1c6
✗ 0x8c3aea3b

Web and App. Server

Database Server

0x2fd4e1c6
0xda39a3ee

0x8c3aea3b

0x2fd4e1c6
0x8c3aea3b

- No explicit cache-coherence algorithm needed

Outline

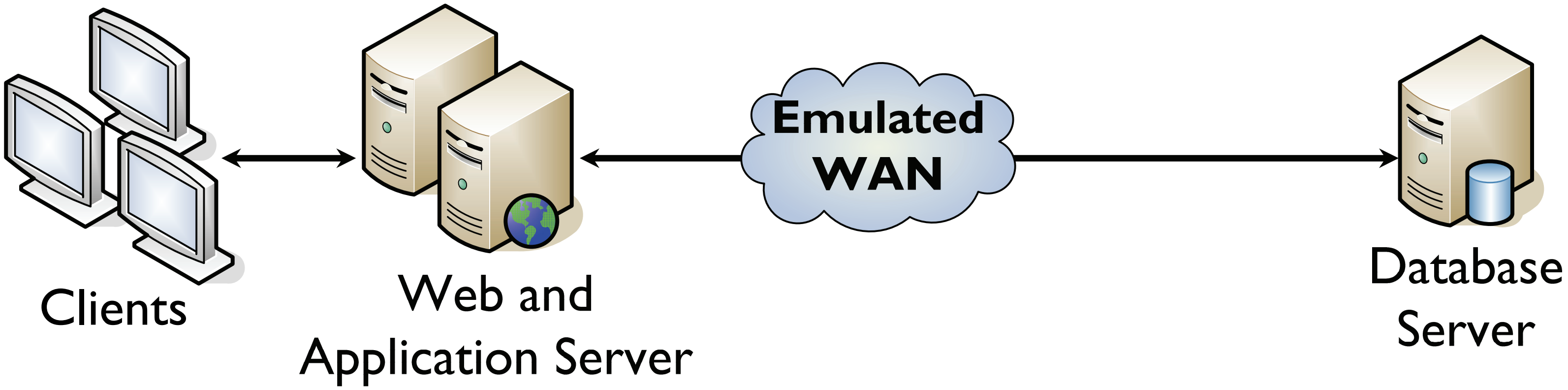
- Motivation
- Ganesh
    - Overview
    - Design and Implementation
- Evaluation
    - Benchmarks and Experimental Setup
    - Results
    - Did exploiting structure help?
- Conclusion

Evaluation – Benchmarks

- AUCTION (RUBiS) – Models eBay
  - Browsing, bidding, add auctions and feedback, …
- BBOARD (RUBBoS) – Models Slashdot
  - Reading articles, adding comments, moderating, …

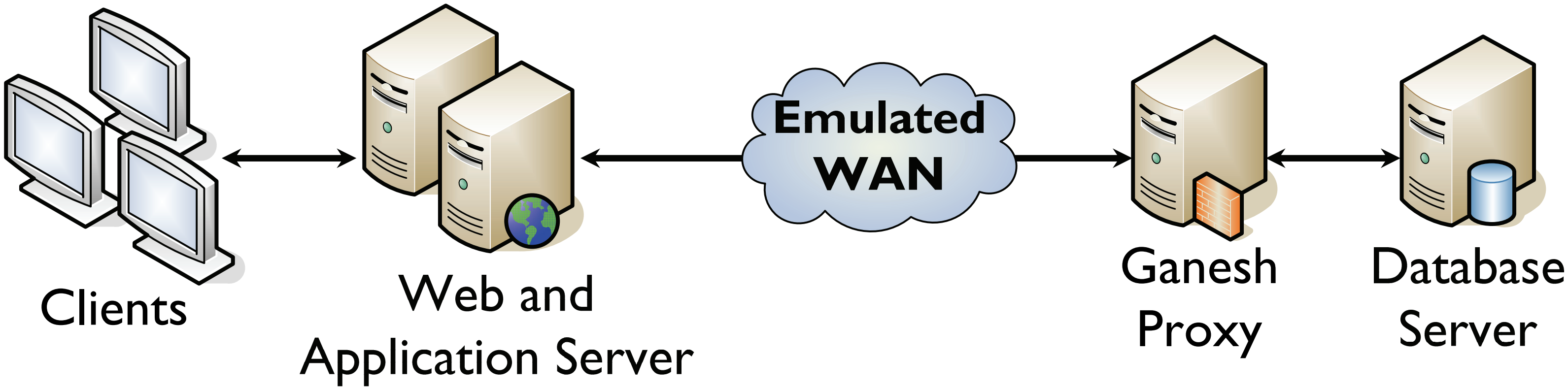
- Benchmarks have RW and RO variants
- Performance metrics
  - Throughput (Requests/sec)
  - Latency (Average request response time)

Experimental Setup



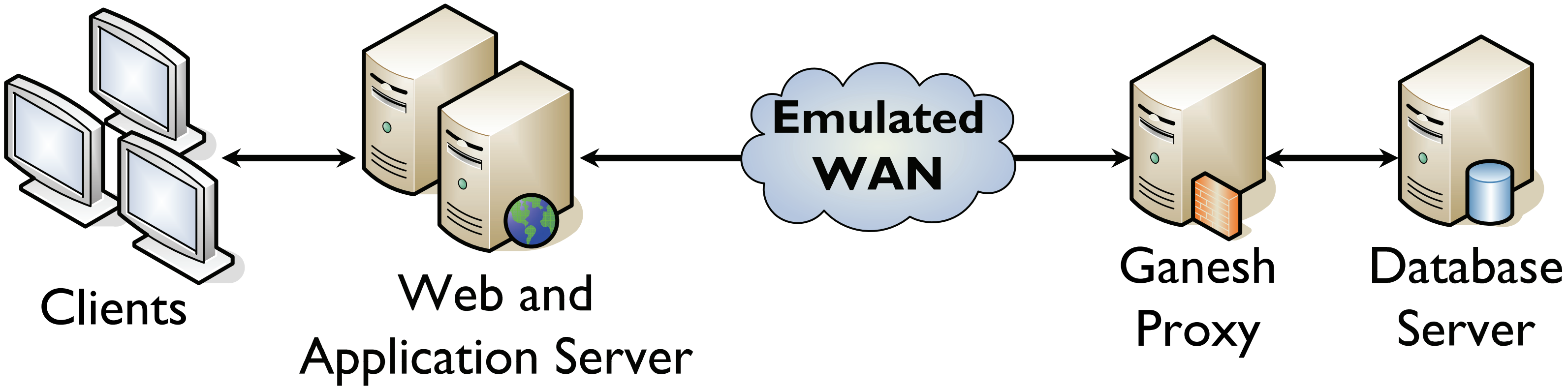
- Two configurations:
  - Native: Unmodified Setup

Experimental Setup



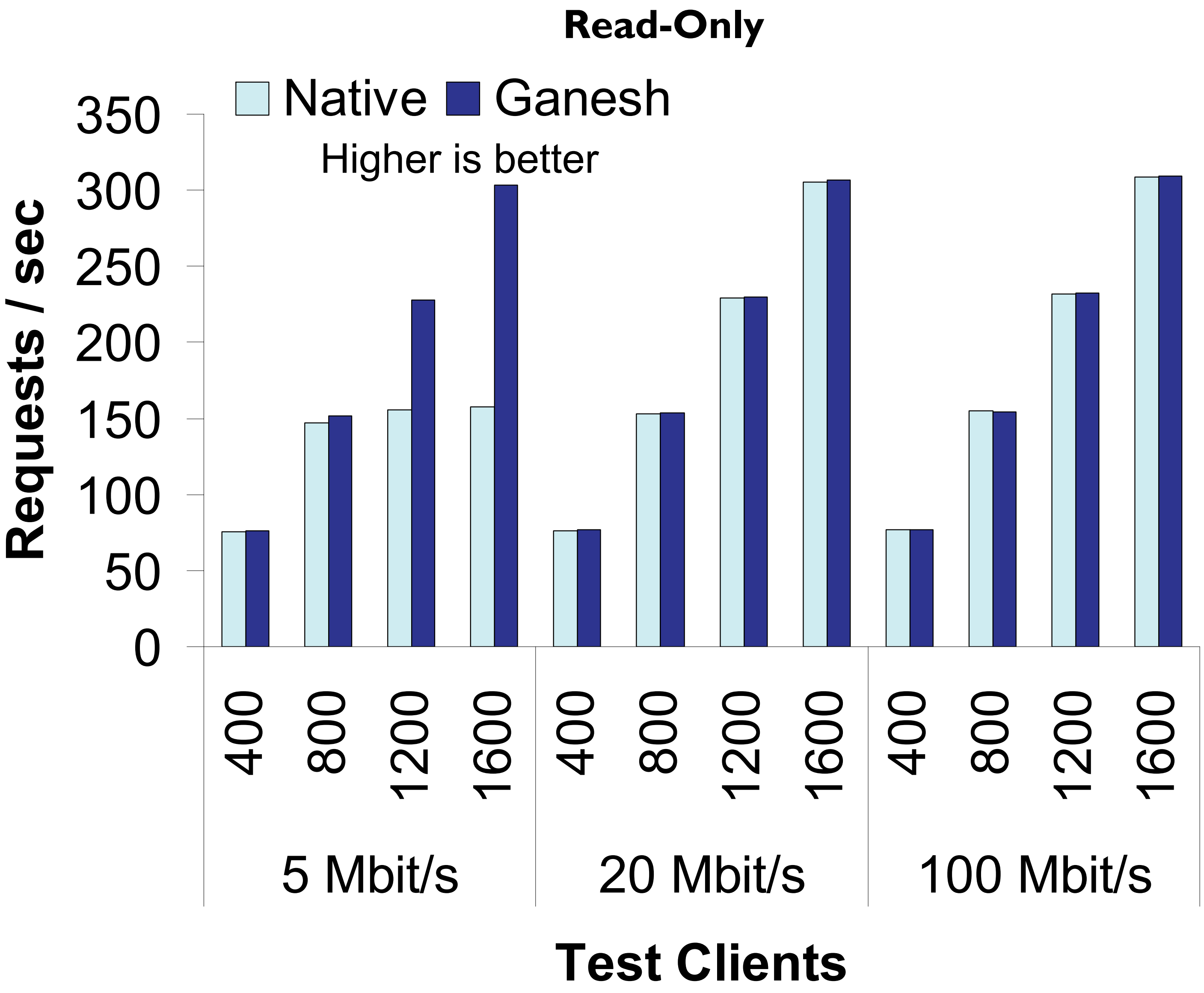
- Two configurations:
  - Native: Unmodified Setup
  - Ganesh: Content-based optimizations used

Experimental Setup



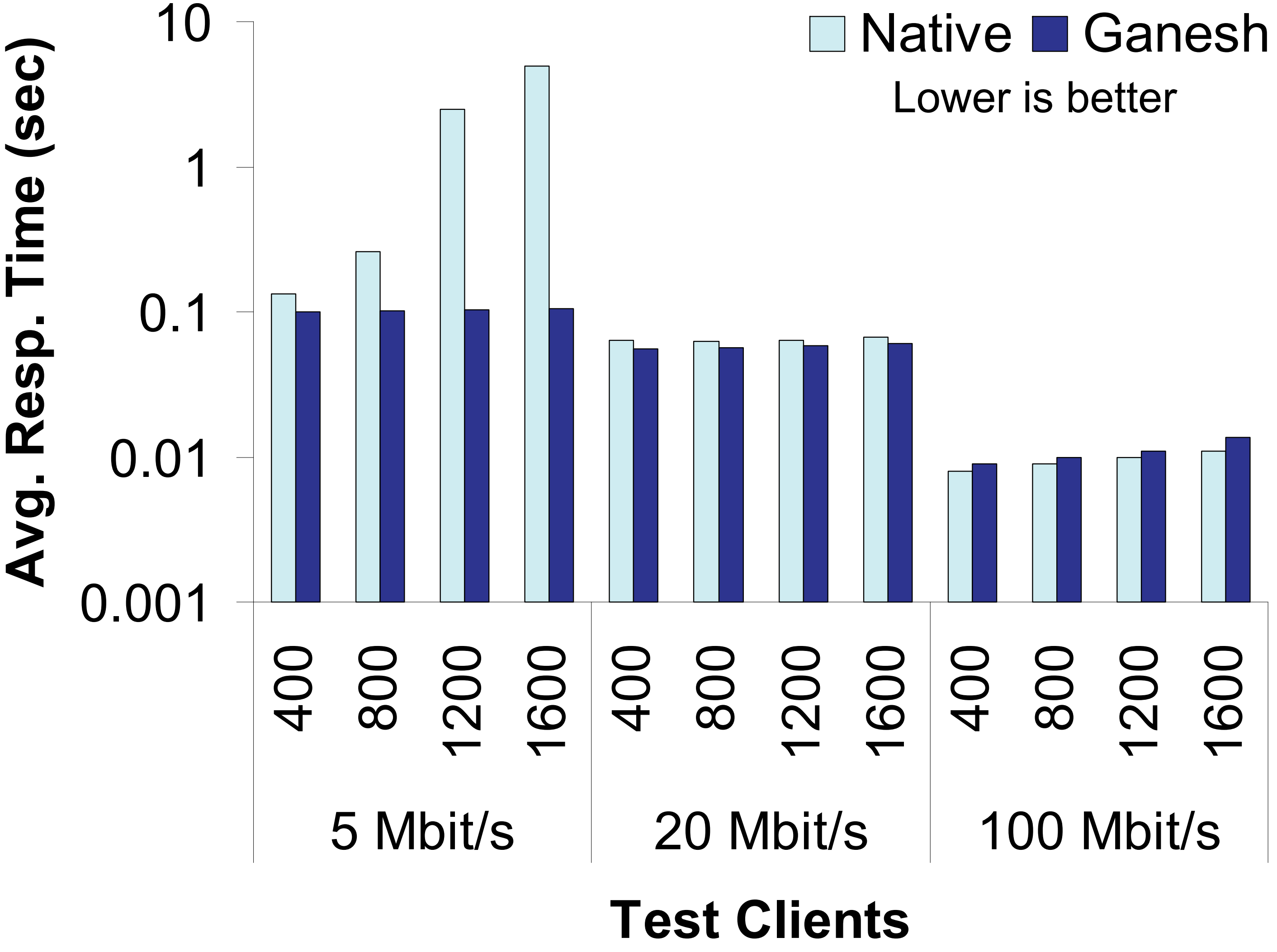
- Two configurations:
  - Native: Unmodified Setup
  - Ganesh: Content-based optimizations used
- Evaluation Parameters
  - 5 Mbit/s + 66ms, 20 Mbit/s + 33ms, 100 Mbit/s
  - 400, 800, 1200, and 1600 clients

AUCTION – Throughput

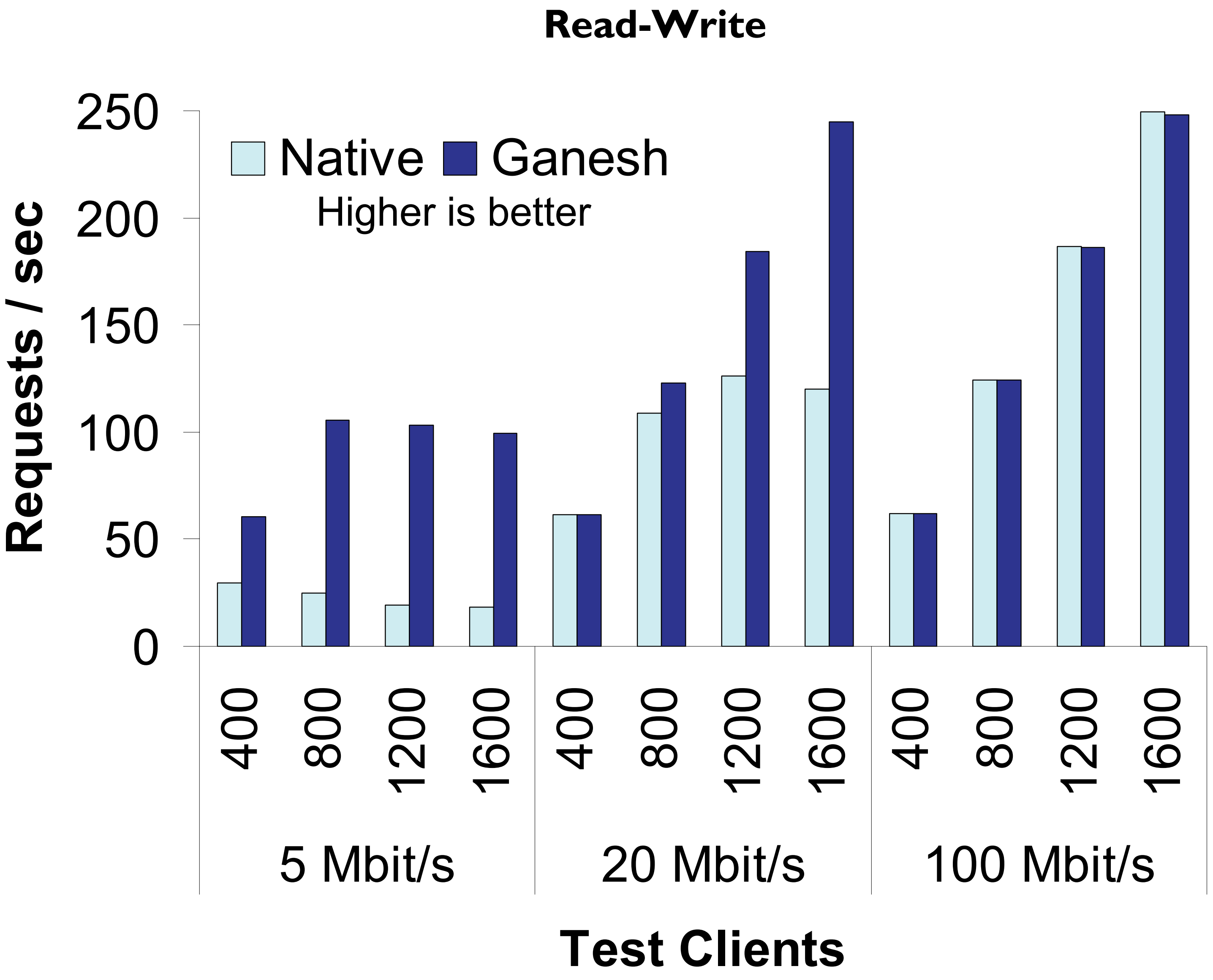


AUCTION - Latency

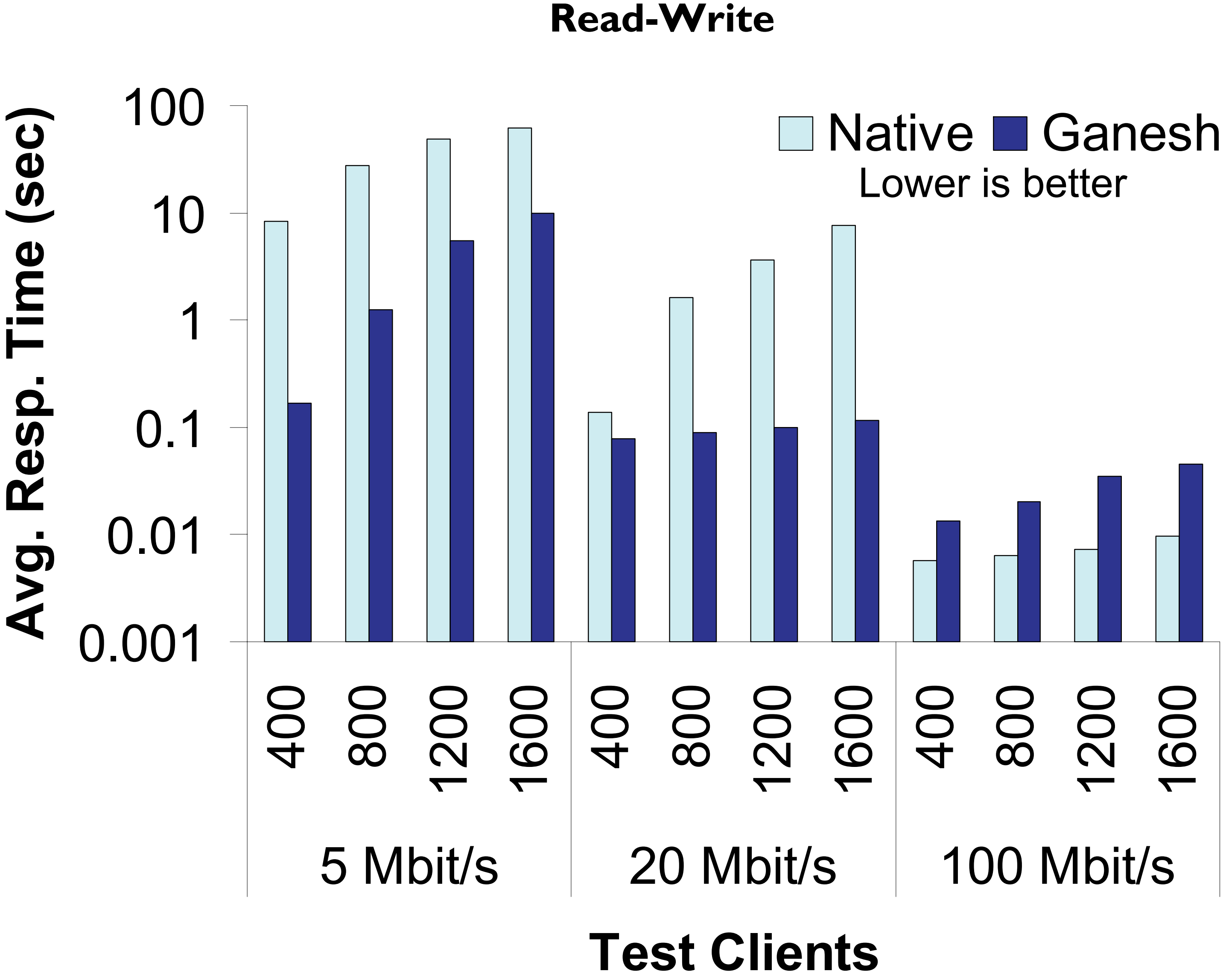
Read-Only



BBOARD - Throughput

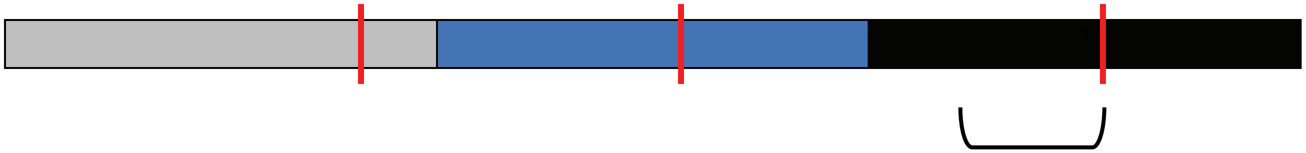


BBOARD - Latency



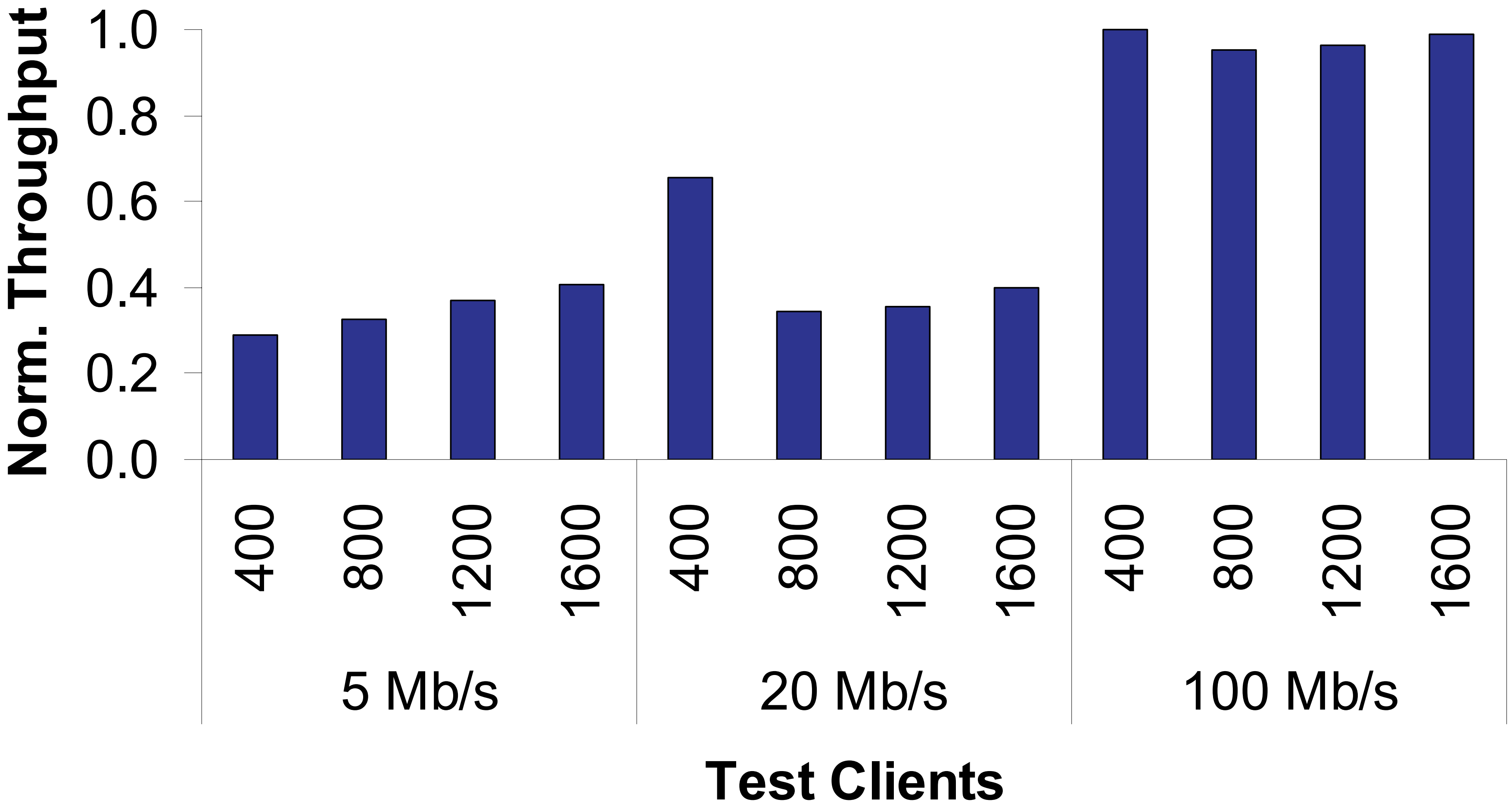
Rabin Fingerprinting vs. Structure

- Could we have used Rabin fingerprinting?
  - Extensive use in storage systems
  - Chunks data using a stochastic process



  - Works well for in-place updates, deletes, insertions
  - Does ***not*** work well for query results
    - Reordering of data (ORDER BY)
    - Also hard to pick average chunk size

Rabin vs. Exploiting Structure



Related Work

- Caching Dynamic Database Content
  - DBCache, DBProxy, MTCache, …
  - Per-application consistency model [Gao03, GlobeDB]
  - Backend-scalability [C-JDBC, SSS, Ganeymed]
- Content-Addressable Systems
  - TCP-level duplicate elimination [Riverbed, Spring00]
  - P2P backup [Pastiche], Storage [Venti, SiS]
  - Distributed File Systems [LBFS, CASPER, PAST]

Conclusion

- Ganesh: Optimizes database access over the WAN
  - Transparent
  - Does not weaken consistency
- Prototype built around Java and the JDBC interface