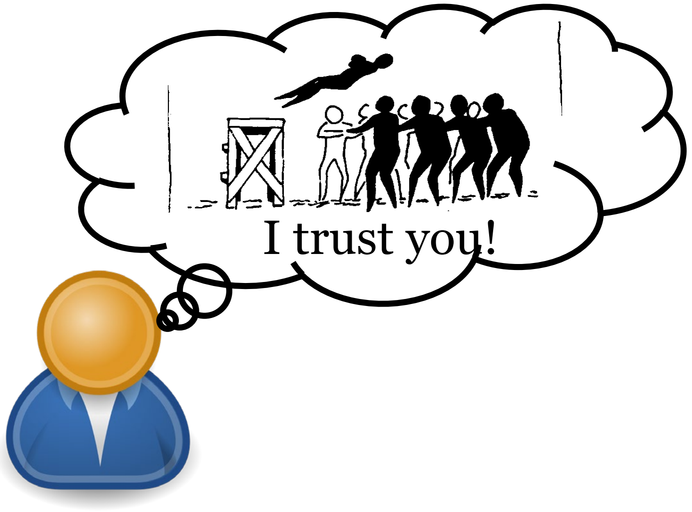
VeriDB: An SGX-based Verifiable Database

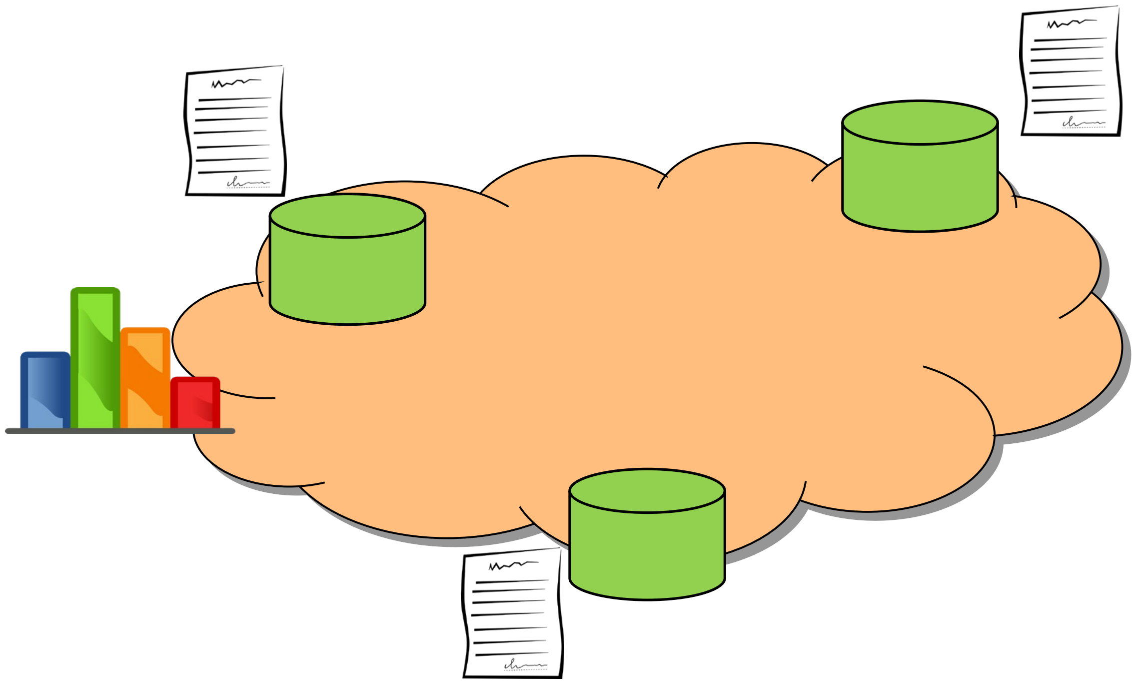
Wenchao Zhou, Yifan Cai, Yanqing Peng, Sheng Wang, Ke Ma, Feifei Li

Alibaba Group
Georgetown University
Penn University of Pennsylvania
The University of Utah

Motivation

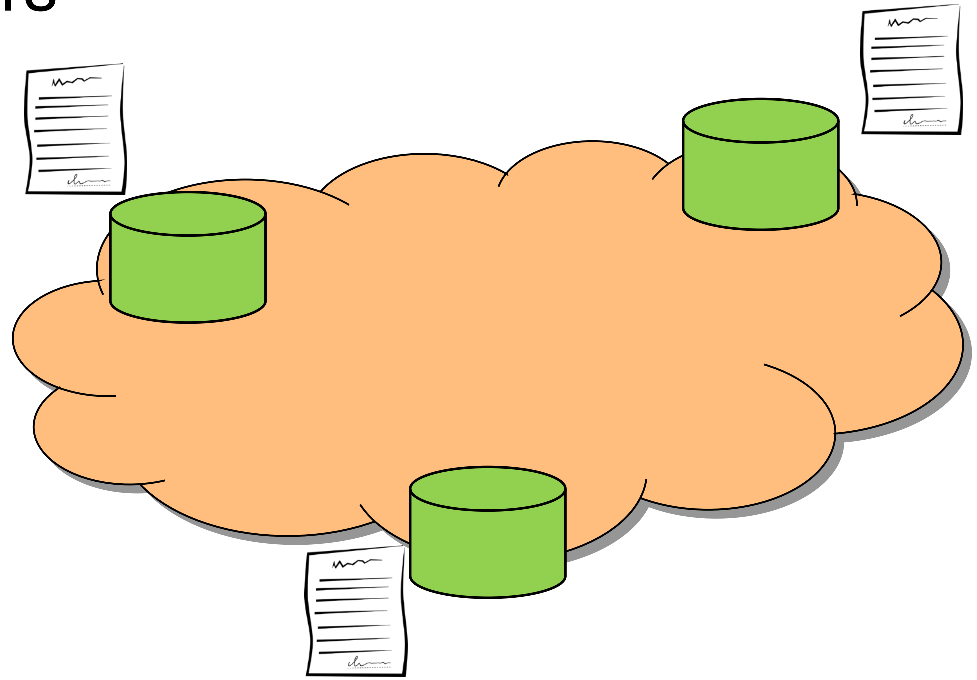
- For cloud storage and cloud computing
  - The integrity of storage and computation relies on the “trust” from users to the cloud service provider.





Motivation

- For sensitive data
  - The cloud service provider needs to give some proof of the correctness.
  - Or detect unexpected behaviors



Motivation

- The cloud may tamper with data.

What is my salary in March?

It's $4000 😈

Table "Salary"

Month	Salary
Jan	5000
Feb	6000
Mar	~~7000~~ 4000

Motivation

- The cloud may even return falsified results without tampering with data

Tell me what kinds of fruits are worth more than $1.60!

Only apples 😈

Table “Inventory”

Item	Price
Apple	2.00
Banana	1.50
Peach	1.80

Scenarios and Goals

Scenario

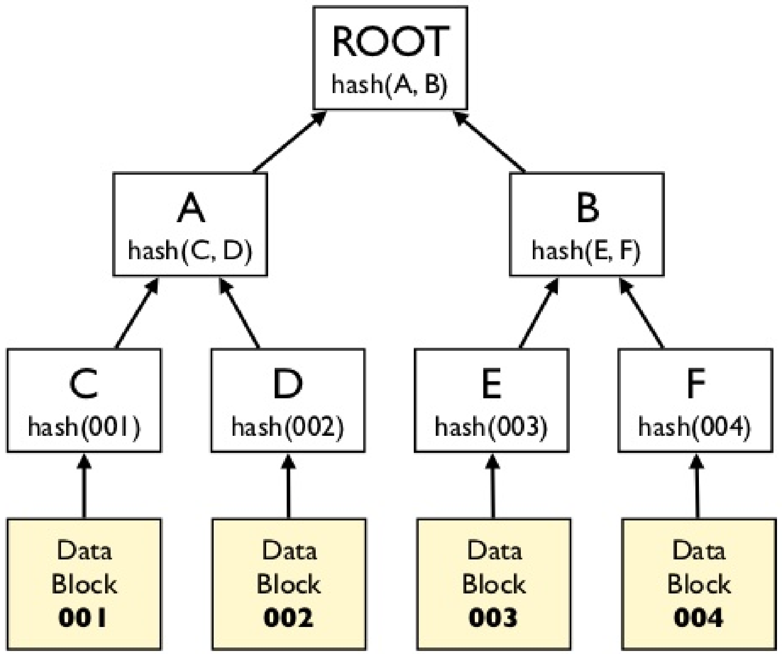
- Cloud-client
- Existence of TEE (SGX)
- Untrusted Cloud Service Provider: Byzantine Behavior

Goals

- For integrity
  - Endorsement of correct results
  - Detection of incorrect results
- For applicability
  - Support for general SQL queries
  - Low overhead

Strawman Solutions

- Use Merkle Hash Tree (MHT) to verify the integrity of data
  - The root hash would be a concurrency bottleneck
- Store all data in trusted memory
  - EPC (enclave page cache) is a scarce resource
  - Expensive swapping if EPC not enough
- Introduce significant overhead



Contribution

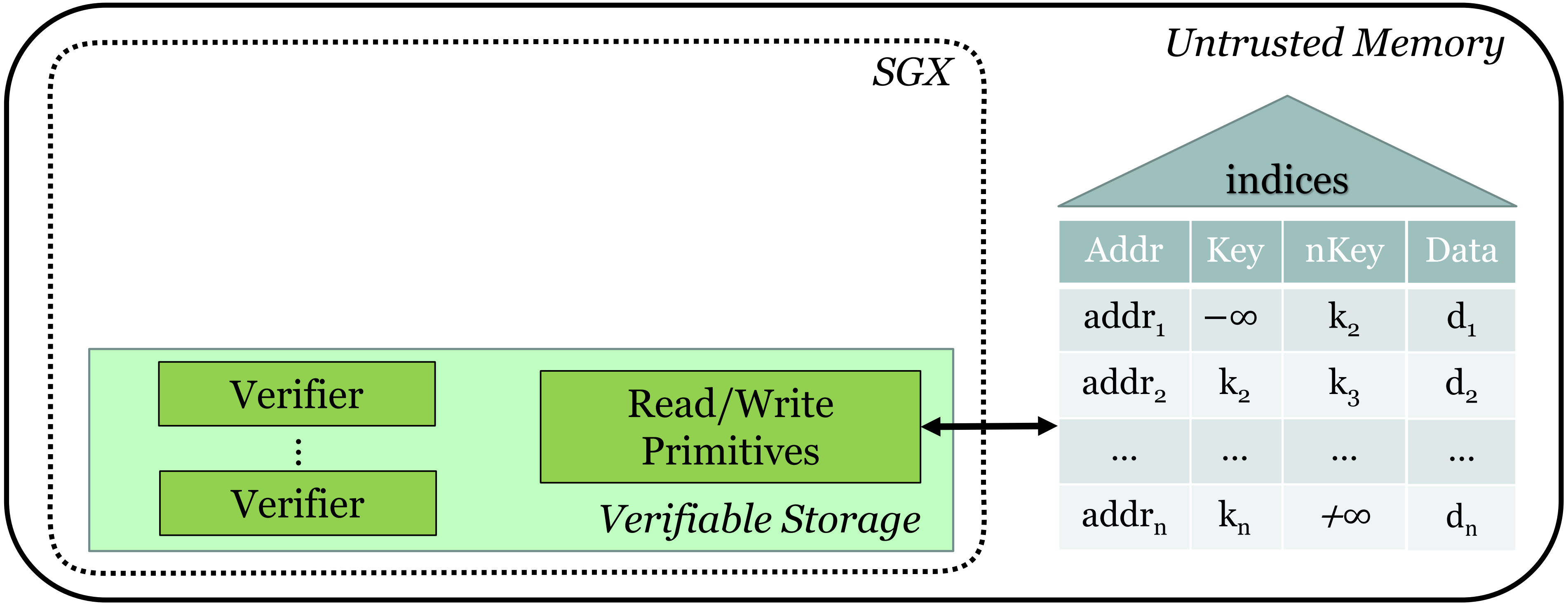
- Verifiable storage and execution
- Support verifiable general SQL queries
- Reasonable performance overhead

Outline

- Introduction
  - Motivation
  - Scenario and goals
  - Contributions
- VeriDB
  - Architecture
  - Verifiable storage and data access
  - Optimizations
- Evaluations

Architecture

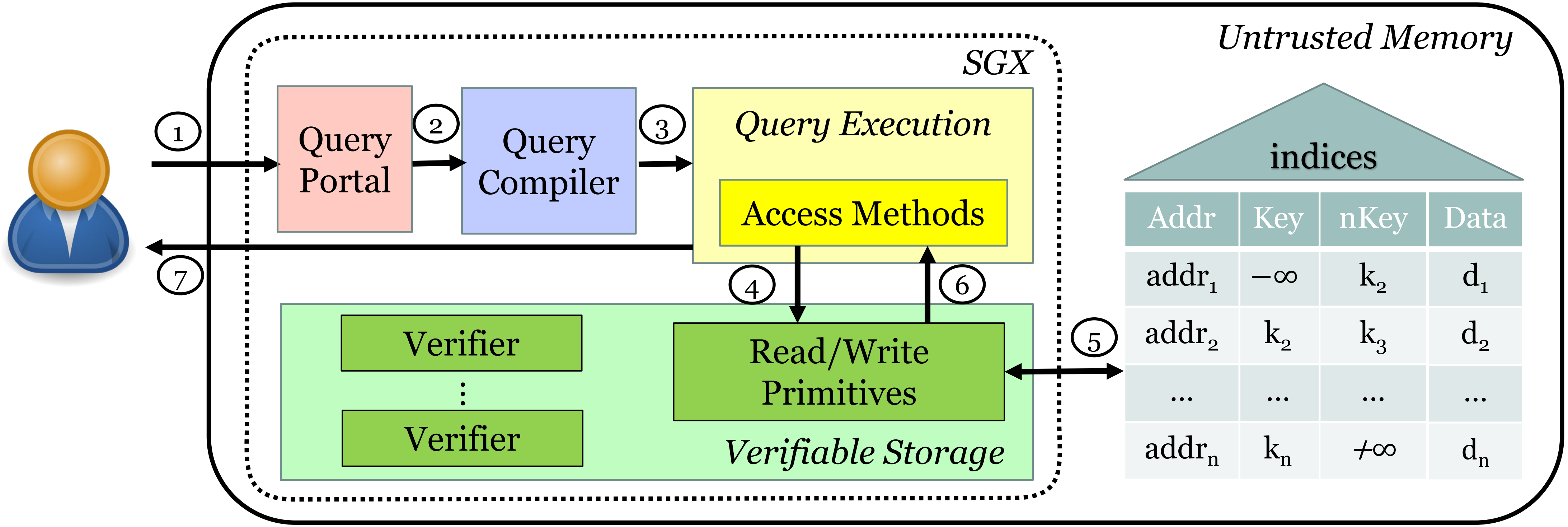
- Data stored in untrusted memory
- Read/Write primitives are stored in trusted memory
  - Ensures the integrity of storage.



Addr	Key	nKey	Data
$addr_1$	$-\infty$	k_2	d_1
$addr_2$	k_2	k_3	d_2
…	…	…	…
$addr_n$	k_n	$+\infty$	d_n

Architecture

- Interface between storage and execution
  - Reduce verifying results into verifying storage and trusted execution
- The client communicates with the query portal via a secure channel



Verifiable Storage

- Basic idea: read-write consistent memory[1]
  - The contents got from "read" must be the contents of the latest "write"
  - Maintain a read set and a write set
  - Update two sets on memory operations
  - Check if the two sets are consistent

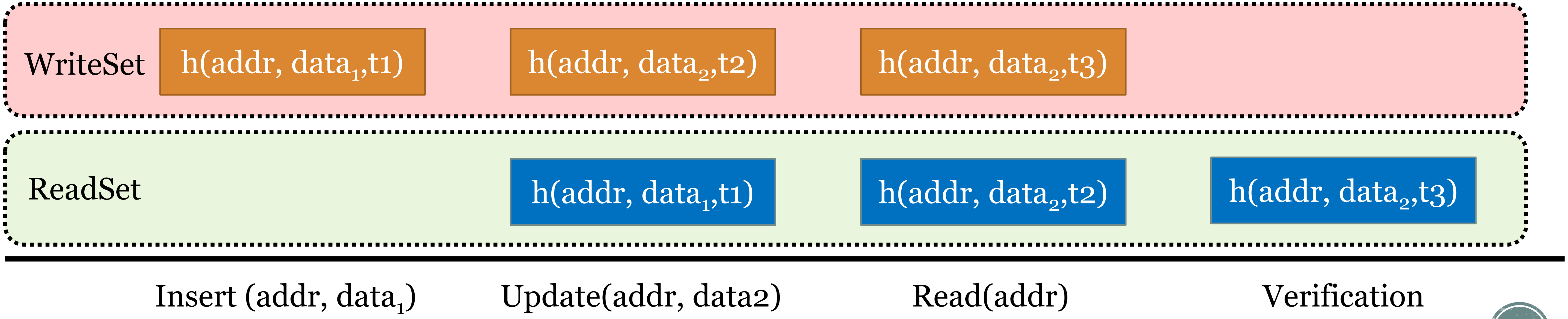
WriteSet

ReadSet

[1] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. 1991. Checking the Correctness of Memories. In 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991. IEEE Computer Society, 90–99.

Verifiable Storage

- Construct a hash of tuple h(addr, data, timestamp) on each operation.
- Update the sets by xor the hashes[2]
- Periodically,
  - The verifier reads each datum and adds to the read set.
  - Verify that ReadSet == WriteSet, otherwise throw an alarm.



[2] A. Arasu, K. Eguro, R. Kaushik, D. Kossmann, P. Meng, V. Pandey, and R. Ramamurthy. 2017. Concerto: A High Concurrency Key-Value Store with Integrity. In SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017. ACM, 251–266

Verifiable Data Access

- Key-chain of records in the table
  - Store (key, nextKey) tuples
  - Prove the existence / absence of a queried record
    - Absence of id2 < qid < id3 is proved by (id2, id3, data)

id	count	price
id_1	100	\$100
id_2	100	\$200
id_3	500	\$100
id_4	600	\$100

key	nextKey	data
$\perp$	id_1	(–, –)
id_1	id_2	(100, \$100)
id_2	id_3	(100, \$200)
id_3	id_4	(500, \$100)
id_4	$\top$	(600, \$100)

Verifiable Data Access

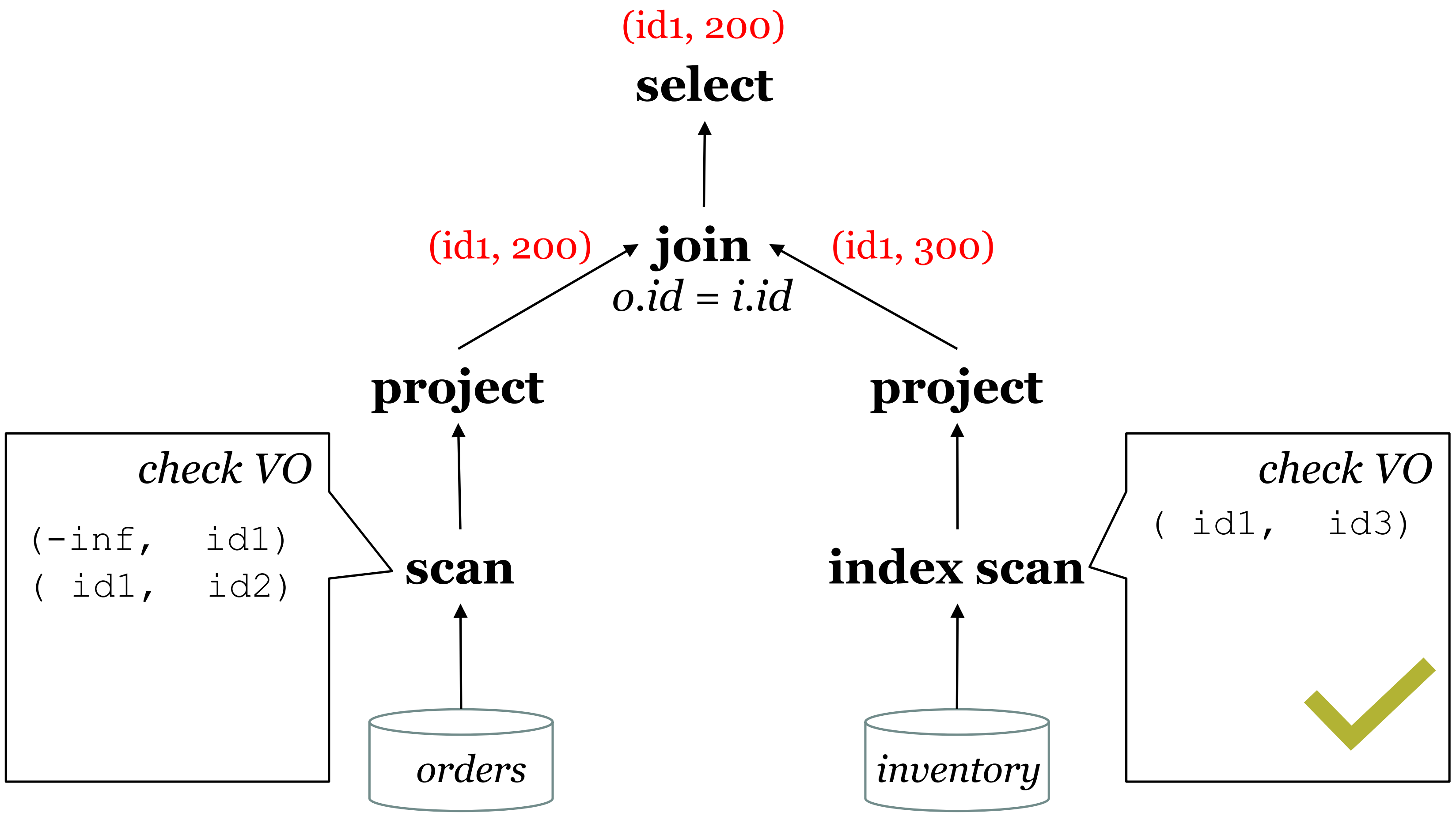
- Three principles to ensure the integrity for range queries `[startKey, endKey]`
  - We don't miss anything in the beginning
    - Find the first row where `row.nextKey >= startKey`, and start from the next row
  - We reach the expected last row
    - `lastRow.nextKey > endKey`
  - All rows are chained
    - `thisRow.key = prevRow.nextKey`

key	nextKey	data
$\perp$	id_1	$(-,-)$
id_1	id_2	(100, \$100)
id_2	id_3	(100, \$200)
id_3	id_4	(500, \$100)
id_4	$\top$	(600, \$100)

- Example: `SELECT * FROM data WHERE key >= id1 AND key <= id3`
- Verifiable storage + verifiable data access = correct results

Query execution

SELECT o.id, o.count
FROM orders as o, inventory as i
WHERE o.id = i.id, o.count <= i.count



orders

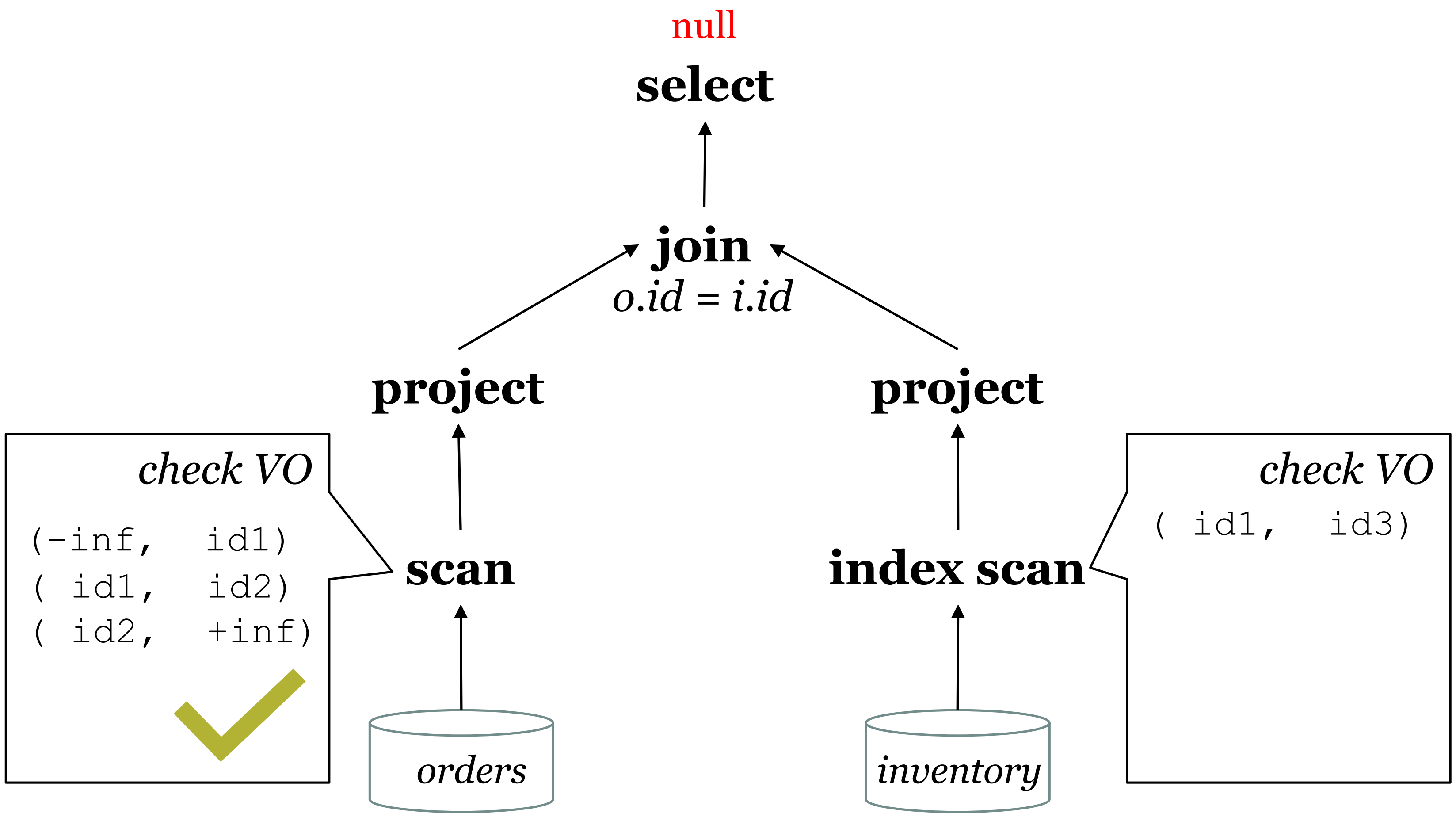
id	count	price	nextid
id1	200	$100	id2
id2	100	$200	+inf
-inf	-	-	id1

inventory

id	count	desc	nextid
id1	300	desc1	id3
id3	800	desc3	+inf
-inf	-	-	id1

Query execution

SELECT o.id, o.count
FROM orders as o, inventory as i
WHERE o.id = i.id, o.count <= i.count



orders

id	count	price	nextid
id1	200	$100	id2
id2	100	$200	+inf
-inf	-	-	id1

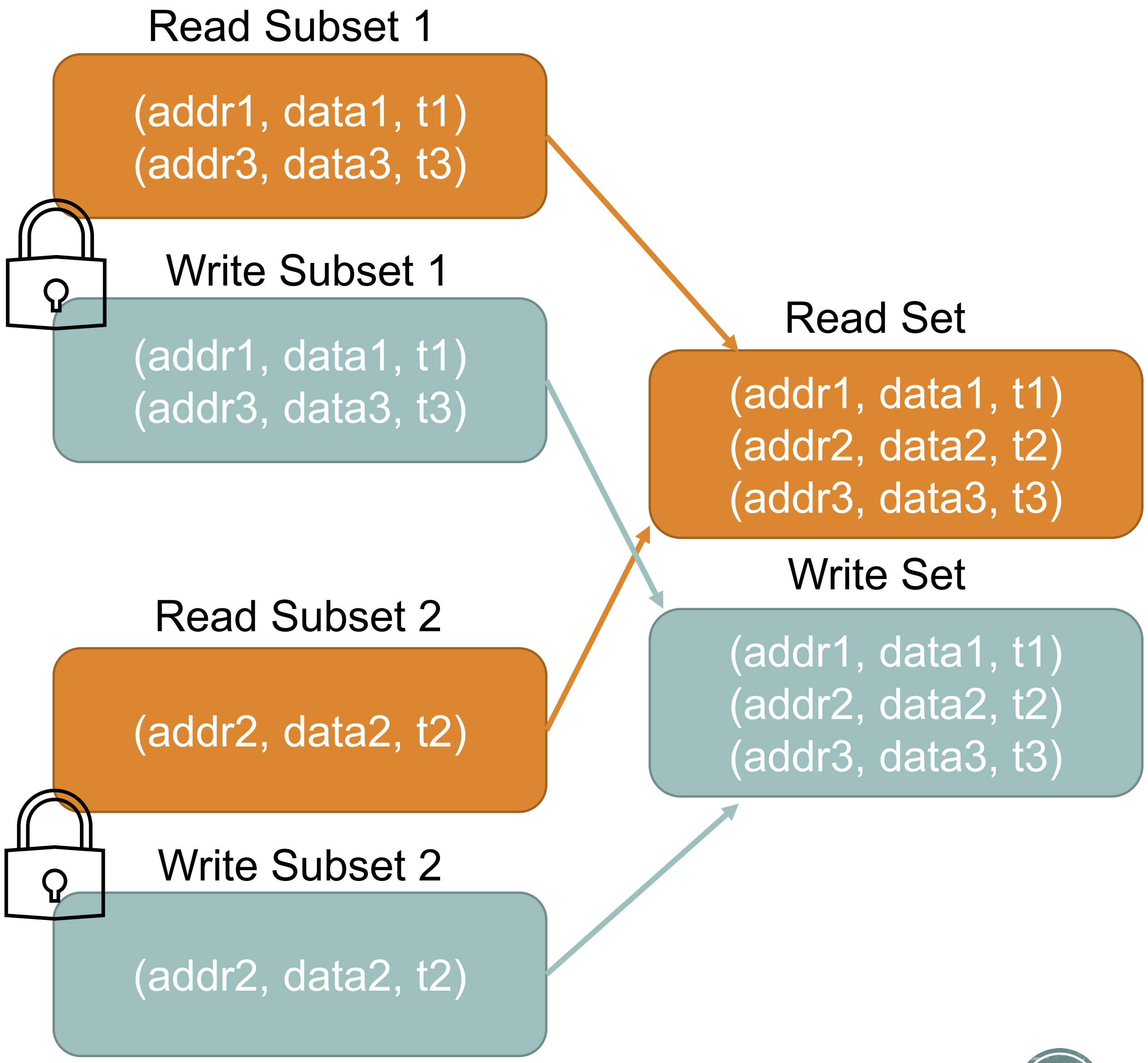
inventory

id	count	desc	nextid
id1	300	desc1	id3
id3	800	desc3	+inf
-inf	-	-	id1

Optimizations

- Use multiple RSWSs to avoid lock contention
  - Operations on `addr1`, `addr2`, and `addr3`
  - Separate the sets during update
  - Combine the sets and compare during verification

- Other optimizations
  - Avoid scanning unvisited pages
  - Excludes page metadata from verification
  - Compaction during verification

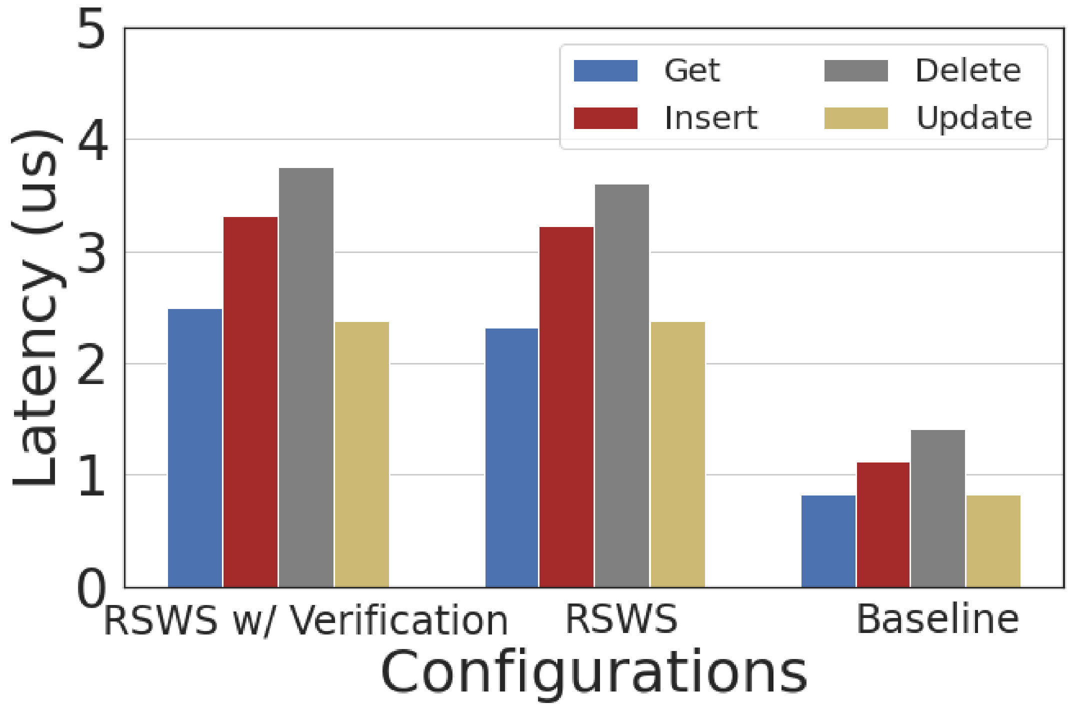


Outline

- Introduction
  - Motivation
  - Scenario and goals
  - Contributions
- VeriDB
  - Architecture
  - Verifiable storage and data access
  - Optimizations
- Evaluations

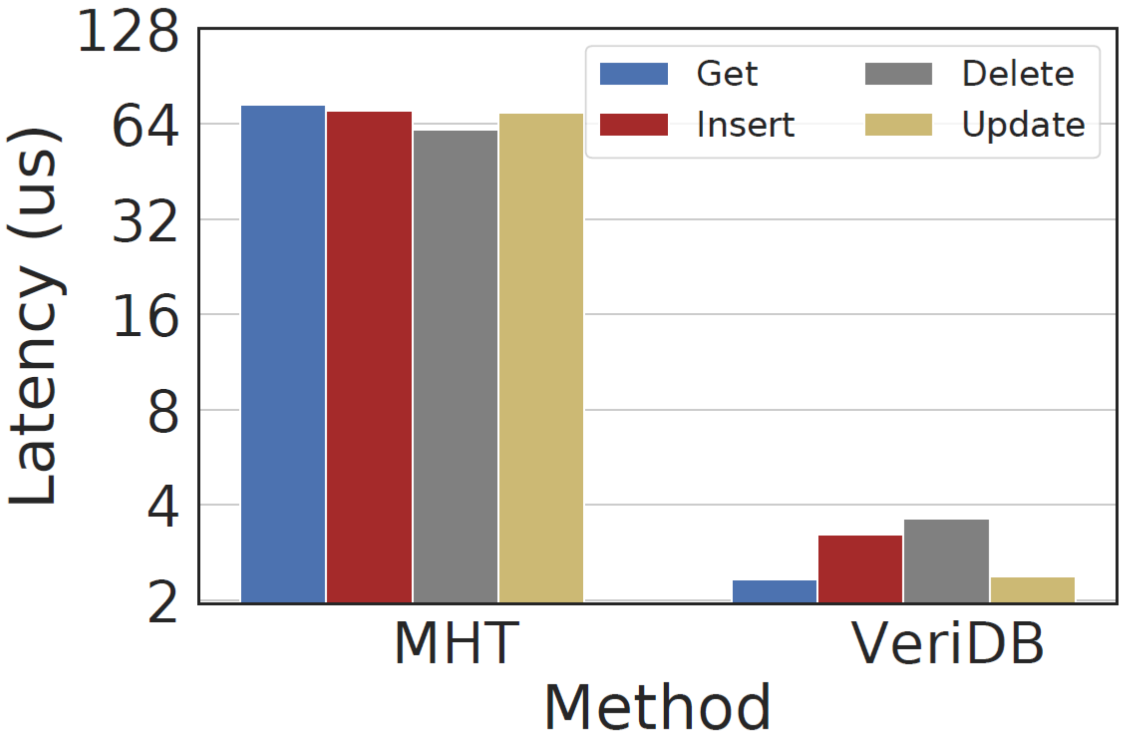
Evaluations – RSWS Updates

- Each update of read set (RS) and write set (WS) introduces 1.5 – 2.2 μs overhead
  - Hash operations make up most of the extra overhead
- "Insert" and "delete" need updates to the "nextKey" field, thus take longer time
- The verification process only introduces slight overhead



Evaluations – v.s. MB-Tree

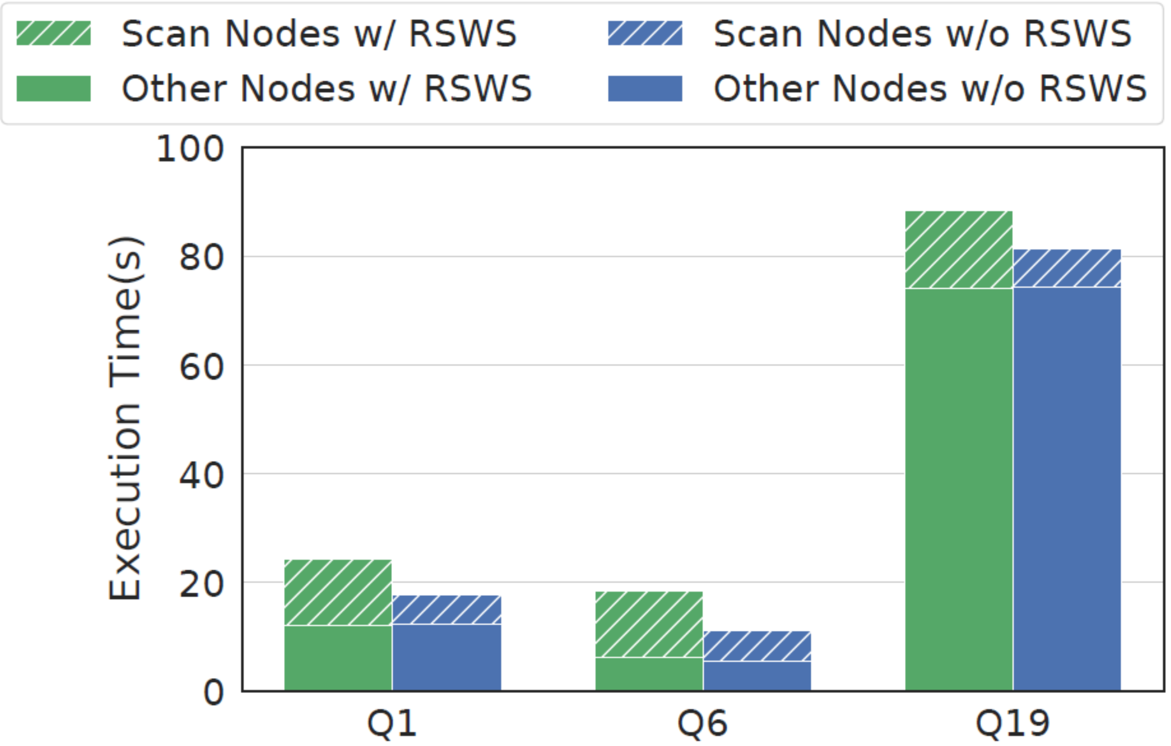
- VeriDB significantly outperforms MB-Tree[3], an MHT-based approach
  - MB-Tree involves more hash calculations
  - The root hash of MB-Tree becomes the bottleneck of concurrency



[3] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. 2006. Dynamic authenticated index structures for outsourced databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006. ACM, 121–132

Evaluations – Macro-benchmark

- Queries (TPC-H)
  - Q1 and Q6, scan, filter, and aggregate;
  - Q19, scan, filter, and join
- The performance overhead mainly comes from the scan operators.
- Overall, VeriDB introduces 9%~39% overhead.
- Other macro-benchmark results: TPC-C



Related Work

System	Support	Trust Model	Overhead	Techniques
Concerto	Key-value	Cloud-user	Relatively Low	SGX + Verifiable memory
EnclaveDB	Relational	Cloud-user	High (All in SGX)	SGX
VeritasDB	Key-value	Cloud-user	High (MHT)	MHT + ADS
FalconDB	Relational	Multi-users	High (Blockchain)	Blockchain + ADS
VeriDB	Relational	Cloud-user	Relatively Low	SGX + Verifiable memory

Conclusion

- VeriDB: an SGX-based verifiable database that supports relational tables and general SQL queries.
- Methods: reduce the problem of providing verified results to ensuring verifiable storage and verifiable access.
- Performance: ≤ 2.2 μs overhead for read/write operators and 9%-39% for analytical workloads