Blockchain and Database

A match made in the cloud

About Me

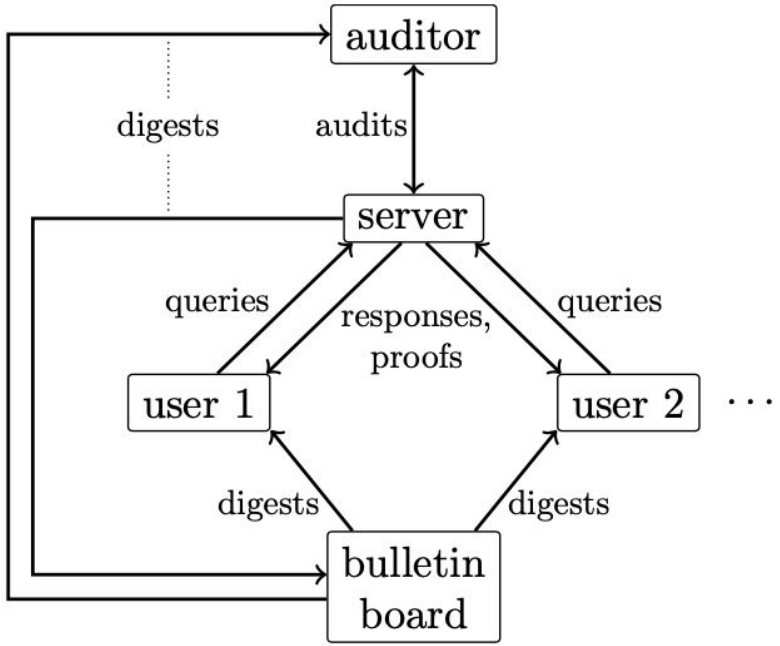
- Senior Lecturer at Deakin University
- Previously:
  - SUTD (Assistant Professor)
  - NUS (Senior Research Fellow)
- Research interest
  - Databases
  - Security
  - Distributed systems

An Observation

- Settings:
  - Some data involved multiple users
  - Computation on the data
  - Outsourced to untrusted servers
- Example: blockchains, key management
- The blockchain way:
  - Consensus ensure that bad things do not happen
    - Given some assumption
- The certificate transparency way:
  - Servers made accountable via auditing
    - Detect bad things after the fact

Transparency

- Untrusted server
  - Publish digests over the data
- Client audits a server
- Third-party / global auditor
  - Ensure fork consistency



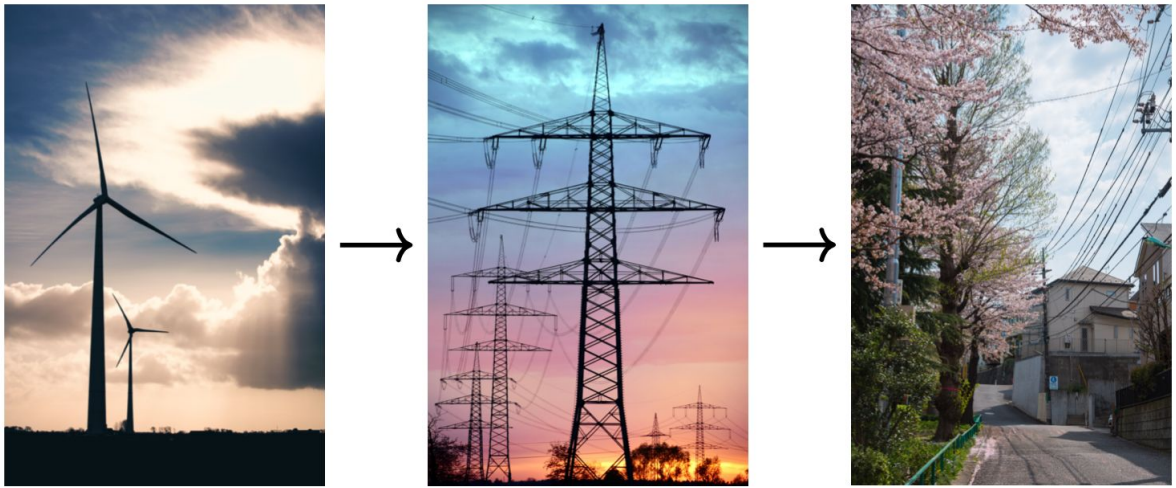
Transparency

- Prevention vs. detection
  - Cost:
    - Blockchains vs. databases
  - Assumptions:
    - Upper bound on failures vs. window of vulnerability
- Transparency is gaining traction!
  - Applications: key/certificate transparency
  - Systems: QLDB, LedgerDB, SQLLedger



Dynamic Pricing

- Retailers: purchase electricity from wholesale market
- Retailer sells to consumers
- Consumer pays bill based on usage

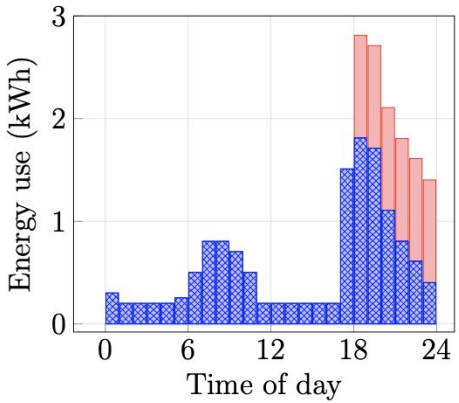


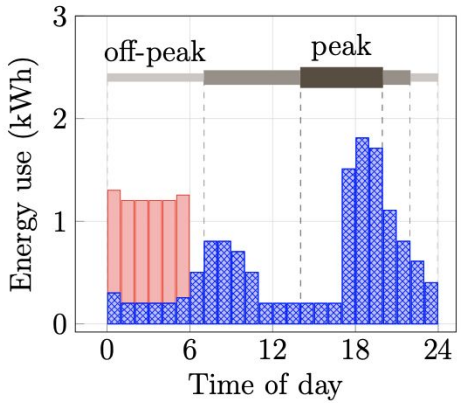
Dynamic Pricing

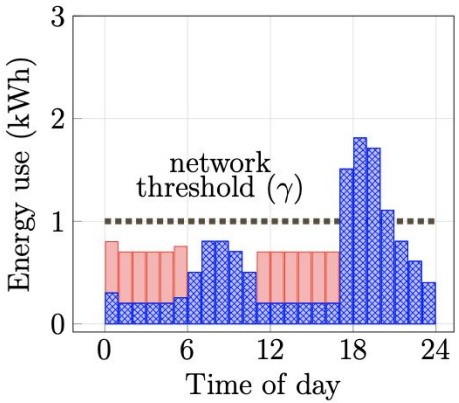
- Retailer’s cost is lowest if total demands spread out over the day
  - -> want consumer to shift loads to low-demand period
- Smart meters:
  - Fine-grained tracking of electricity usage
- Dynamic pricing
  - Different rates based on usage
  - Higher rate if exceeding some thresholds

Dynamic Pricing

- Pricing scheme:
  - Charged based on system-level demand
  - Peak rate applied to consumer if:
    - Individual demand exceeds t1
    - System-level demand exceeds t2

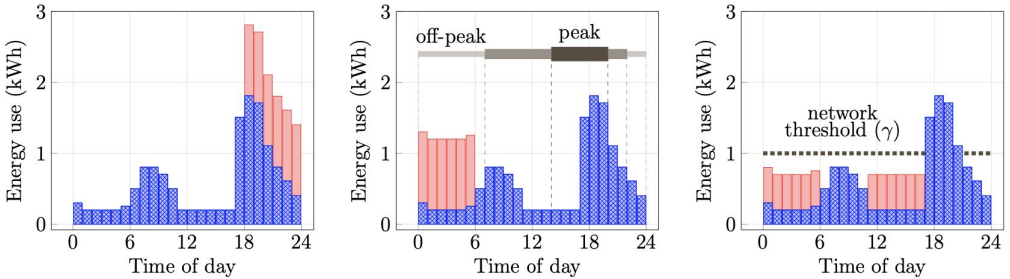






Dynamic Pricing

- Threat model
  - Retailer exaggerates system-level demand
    - More money
  - Consumer A curious about consumer B's usage
- Goals:
  - Transparency: retailer cannot exaggerates beyond a bound
    - Defined by number of malicious/fake users
  - Privacy: does not reveal data to curious consumers



Dynamic Pricing

- Building blocks:
  - Commitments
  - ZK range proofs
- Baseline:
  - Retailer computes C for all data and sums
  - Retailer computes range proofs for all data and sums
  - Retailers sends all commitments and proofs to users, publishes hashes on bulletin boards
  - Consumer checks all proofs
    - And her data is included in C

- Additively homomorphic *commitment protocol*:

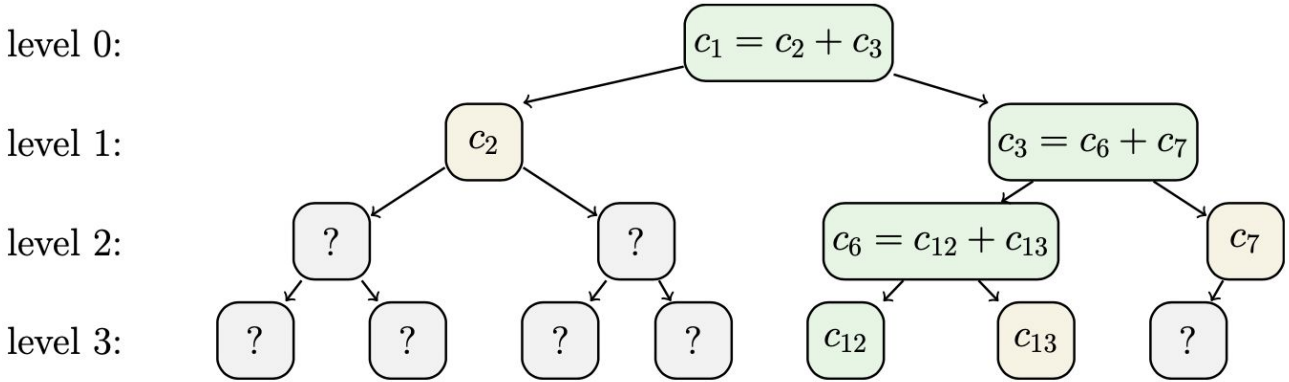
$$C(v_1, r_1) + C(v_2, r_2) = C(v_1 + v_2, r_1 + r_2)$$

- *Zero-knowledge range proofs* for the above protocol:

$$\{(c, v_{max}), (v, r) : C(v, r) = c, v \in [0, v_{max}]\}$$

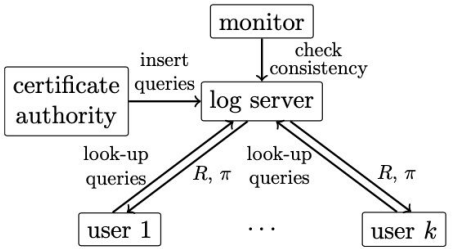
Dynamic Pricing

- Merkle tree based solution:
  - Retailer builds Merkle tree on commitments
  - Sends inclusion proofs to consumer
  - Consumer verifies proofs
  - Auditor checks all range proofs

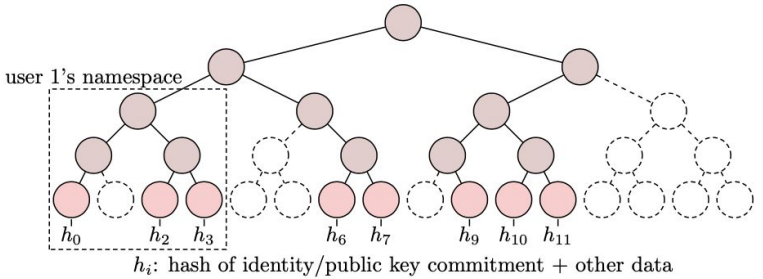


Transparent Data Services

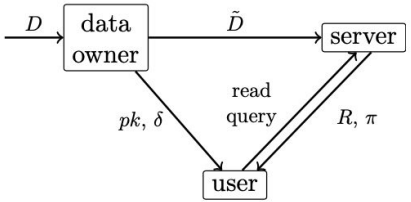
- More general computation than just SUM
- SOTA: key transparency, blockchains, general ADS



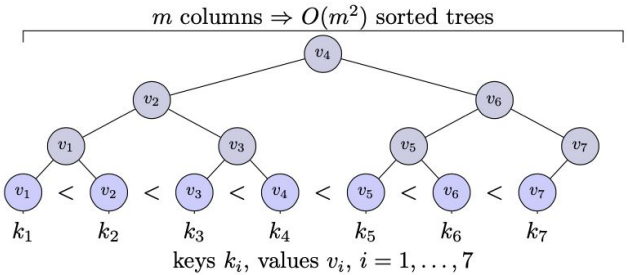
(a) System model of CT, CONIKS, and Merkle^2.



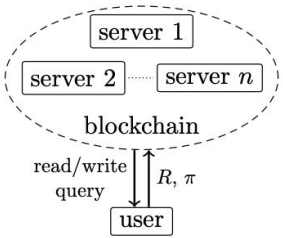
(b) ADS of CONIKS.



(c) System model of IntegriDB.



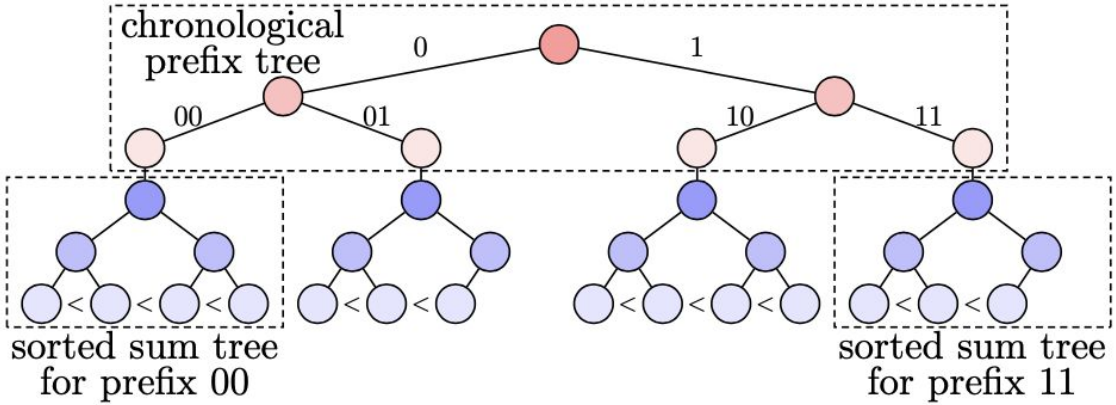
(d) ADS of IntegriDB and FalconDB.



(e) System model of FalconDB.

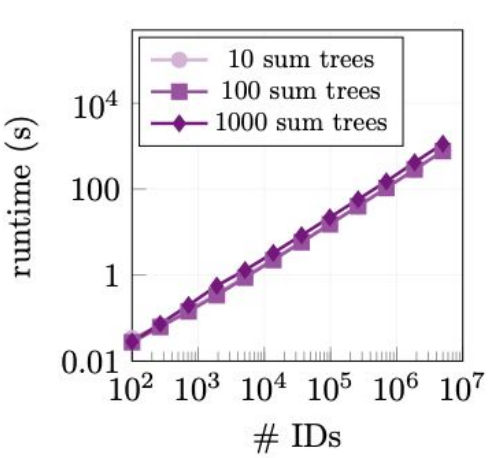
Data Services

- Support rich operations: SUM, MIN/MAX, QUANTILES
- Applications: smart grids, congesting pricing, advertising
- Building blocks:
  - SUM tree + prefix tree
  - Leaves are sorted commitments
- Richer operations on top:
  - MIN
  - SUM
  - QUANTILES

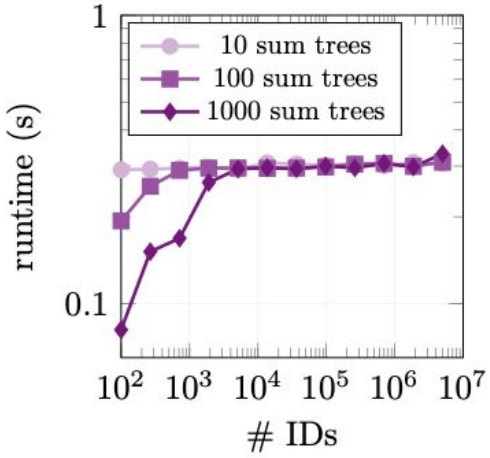


Transparent Data Services

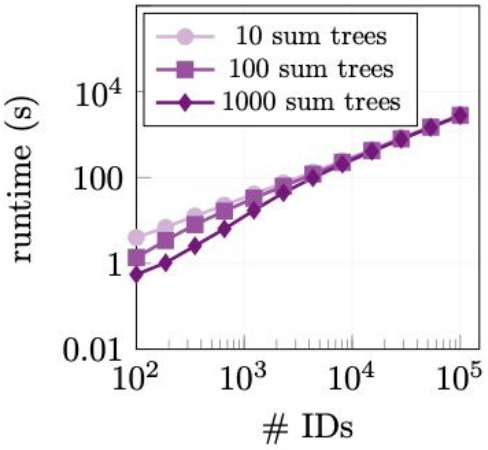
- Performance



(a) tree construction



(b) query execution



(c) full audit

Ledger Databases

- What is a ledger database
  - Execute user operations, maintaining a history of operations
  - Integrity: server cannot tamper with the result
    - E.g.: update x to A -> query x will return A
  - Append-only: server cannot change the history of operations
- Vast design space:
  - Threat model: consensus vs. auditing
  - Abstraction: key-value APIs vs. transaction
  - Performance: proof sizes, latency, throughput, etc.

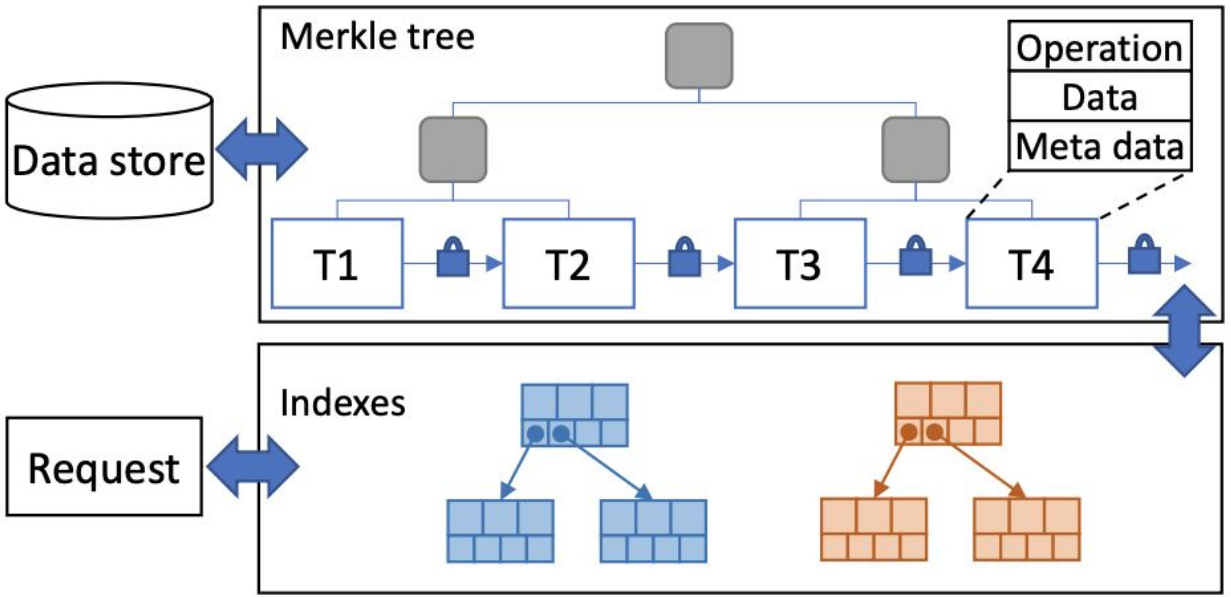
Ledger Databases

- Current systems

System	Abstraction	Threat model	Append-Only Proof	Current-Value Proof	Throughput
QLDB [2]	Transaction	Audit	$O(\log N)$	$O(N)$	Low
LedgerDB [31]	Transaction	Audit	$O(\log N)$	$O(N)$	Medium
Forkbase [28]	Key-value	Audit	$O(N)$	$O(\log m)$	Medium
Blockchain [4]	Transaction	Consensus	$O(1)$	$O(1)$	Low
CreDB [20]	Transaction	Trusted hardware	$O(1)$	$O(1)$	Low
Trillian [15], ECT [26], Merkle^2 [16]	Key-value	Audit	$O(\log m)$	$O(\log m)$	Low
GLassDB	Transaction	Audit	$O(\log B)$	$O(\log B + \log m)$	High

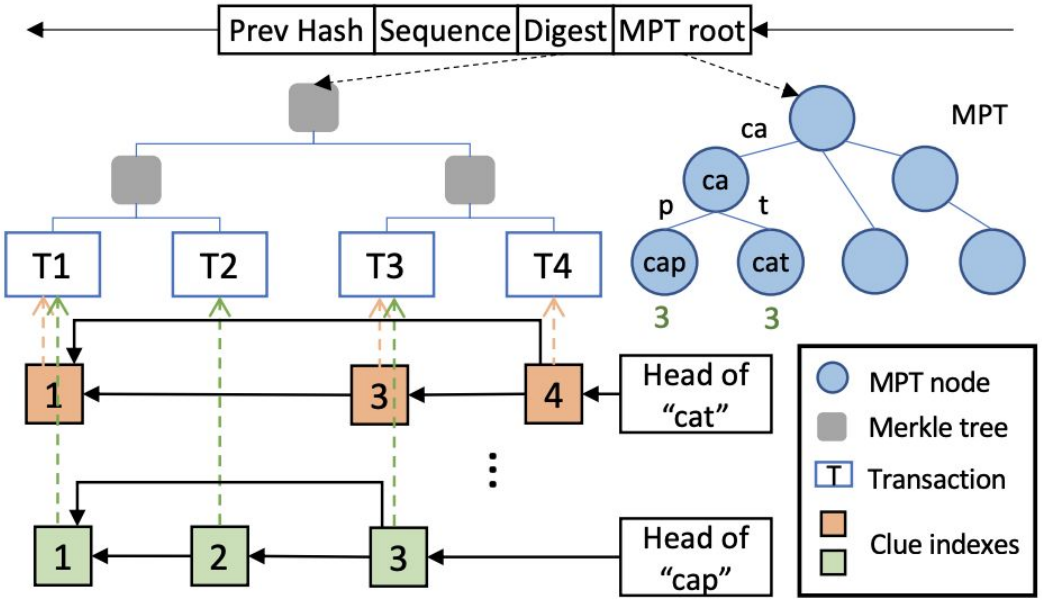
Systems

- QLDB
  - Inefficient
  - Index not protected



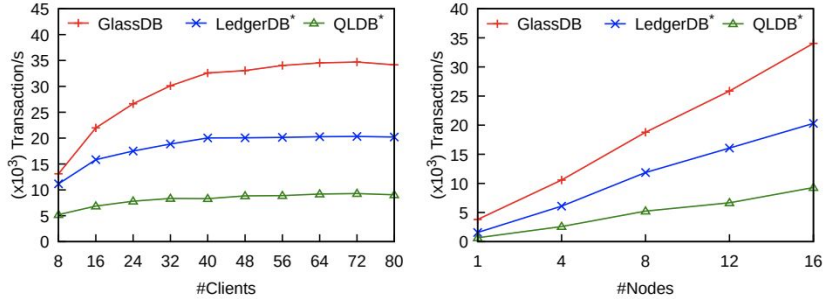
Systems

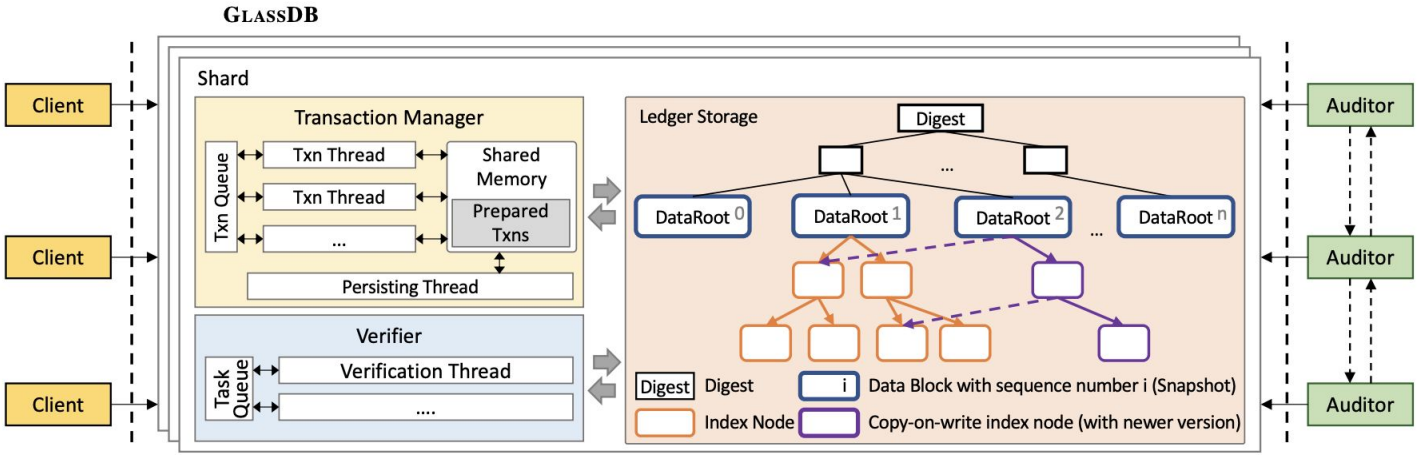
- LedgerDB
  - Better performance
  - Verification is still expensive



Systems

- GlassDB:
  - Concurrent transactions
  - Integrity protected index
    - Structured Invariant Reusable Index
  - High performance





Going Forward

- Transparency in ML?
- Testing transparent systems
  - Anyone fuzzed CT yet?
  - Blockchain:
    - Smart contracts (seems crowded)
    - Consensus layer?
    - Storage layer?
    - Application layer (DeFi: so many incidents!)

Thank you

- Consider submitting to VDBS workshop

  https://veridbsys.github.io/