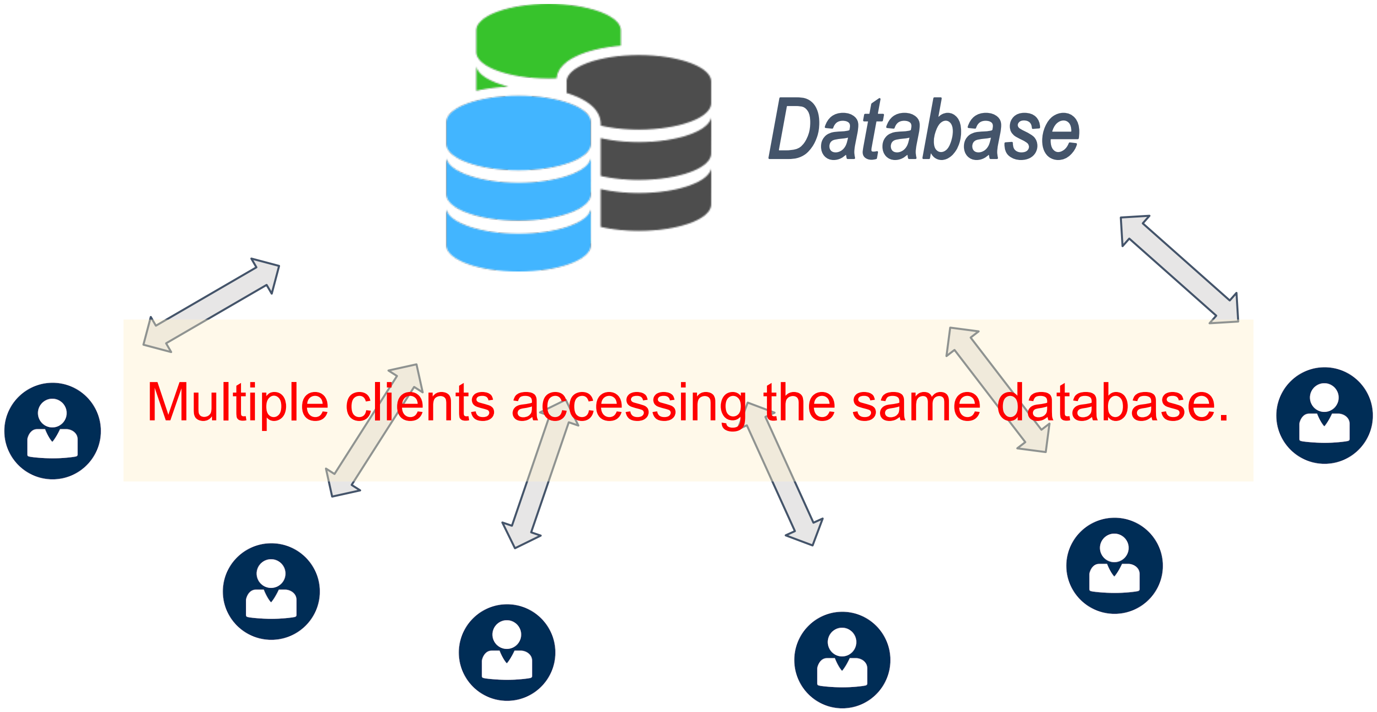
FalconDB: Blockchain-based Collaborative Database

Yanqing Peng[1], Min Du[2], Feifei Li[1], Raymond Cheng[2], Dawn Song[2]

University of Utah[1] UC Berkeley[2]

Shared database - multiple clients



Multiple clients accessing the same database.

Example: Crowdsourcing restaurant rating

Tell me all the Chinese restaurants in New York with high rating.

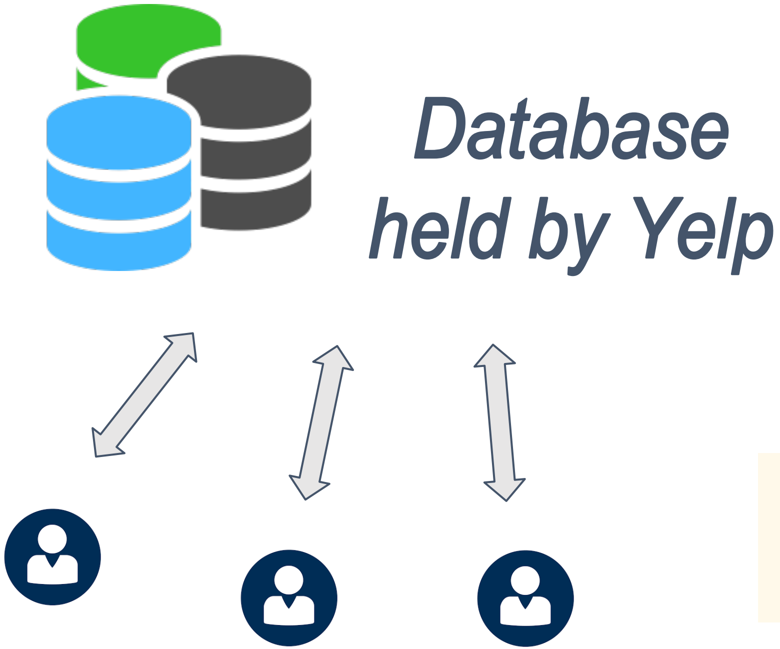
The restaurant “XX kitchen” is good! 5 stars!

```
select * from Restaurants R
where R.Zip = 94707
and R.Category=“Chinese”
and R.Avg_rat > 4;
```

Collaborative Database

```
insert into Rate
values (1000, 2, 5);
```

Trusted database server solution



Server decides:

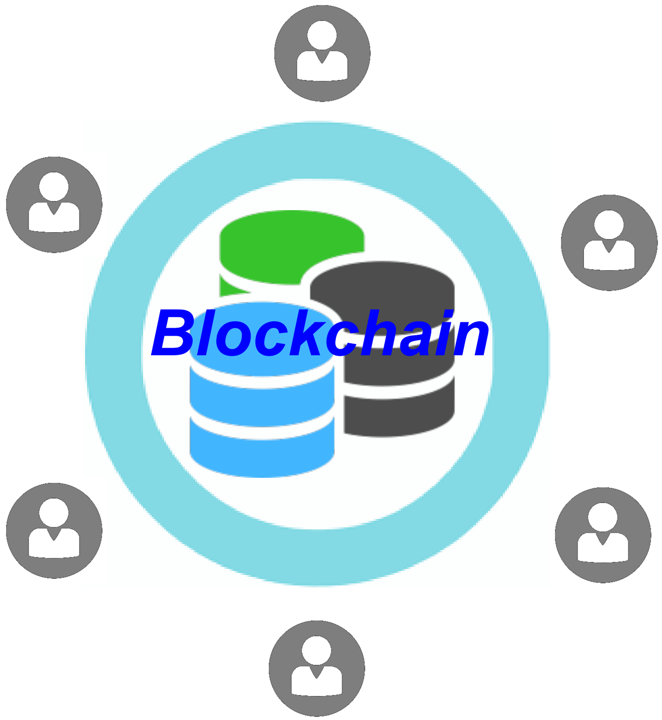
1. Transaction order;
2. The result of a query;
3. If an update should go through;

   ....

Google docs | **Overleaf** | **Github**

How about a decentralized rating system without centralized server?

Shift the trust



Blockchain-based solution

Consensus among all participants:

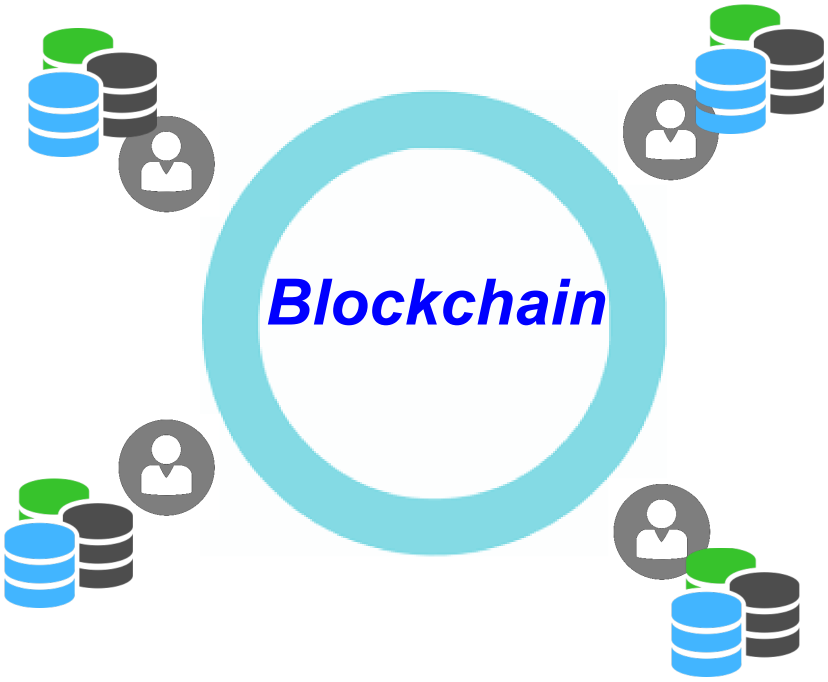
1. Transaction order;
2. The result of a query;
3. If an update should go through;

   ….

Guarantees transparency and immutability.

Shift the trust

Naive *Blockchain-based solution*



Each database client stores a full copy of the database, and run consensus in a permissioned blockchain network.

FluereeDB

BigchainDB

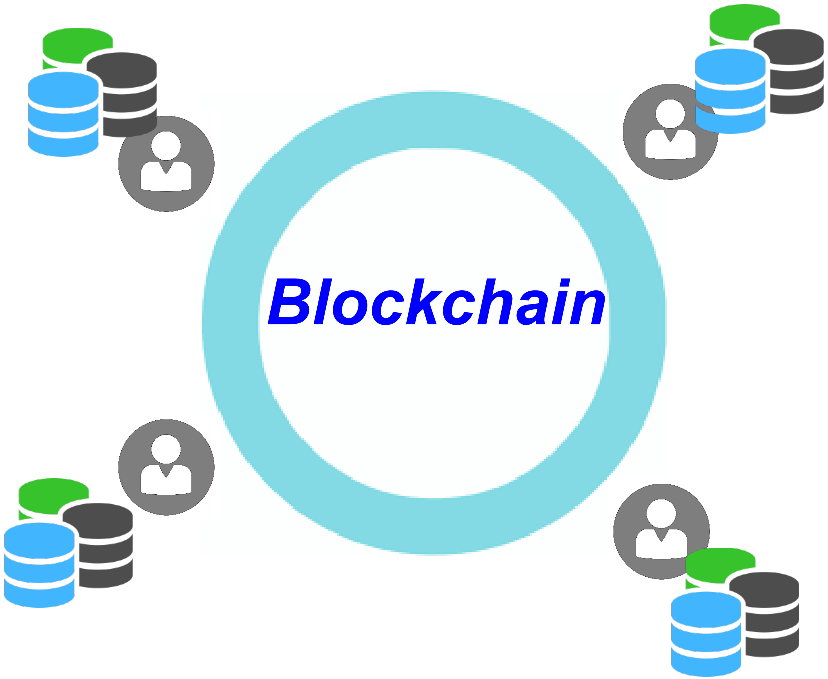
swarmDB

High storage cost!

Find a restaurant with an old smartphone?

Shift the trust

Light blockchain node



Individual users querying full nodes.

How to ensure integrity without trusting full nodes?

Tool: Verifiable Computation

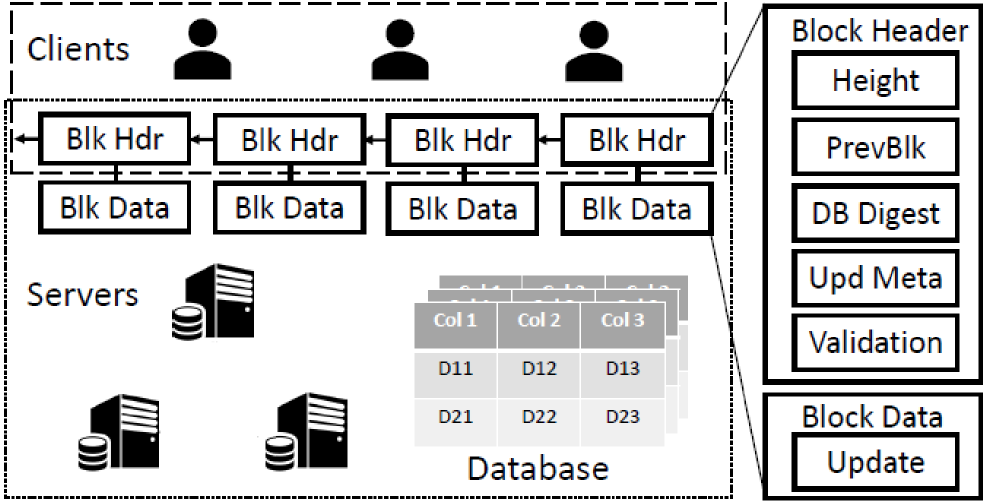
- ➢Assume the service provider and the users both have a “digest” of the data;
- ➢Service provider returns results with cryptographic proof based on the digest;
- ➢Users verify integrity of results using the proof based on the digest.
  - ➢Guarantee: if the digest is correct, then the validation process passes if and only if the query result is correct

Use Blockchain to ensure the correctness of digest

❖Enabling trust on full nodes:

- ➢Light nodes query full nodes, and verify the results using VC.

FalconDB: architecture overview



❖Block:

➢contains all update logs

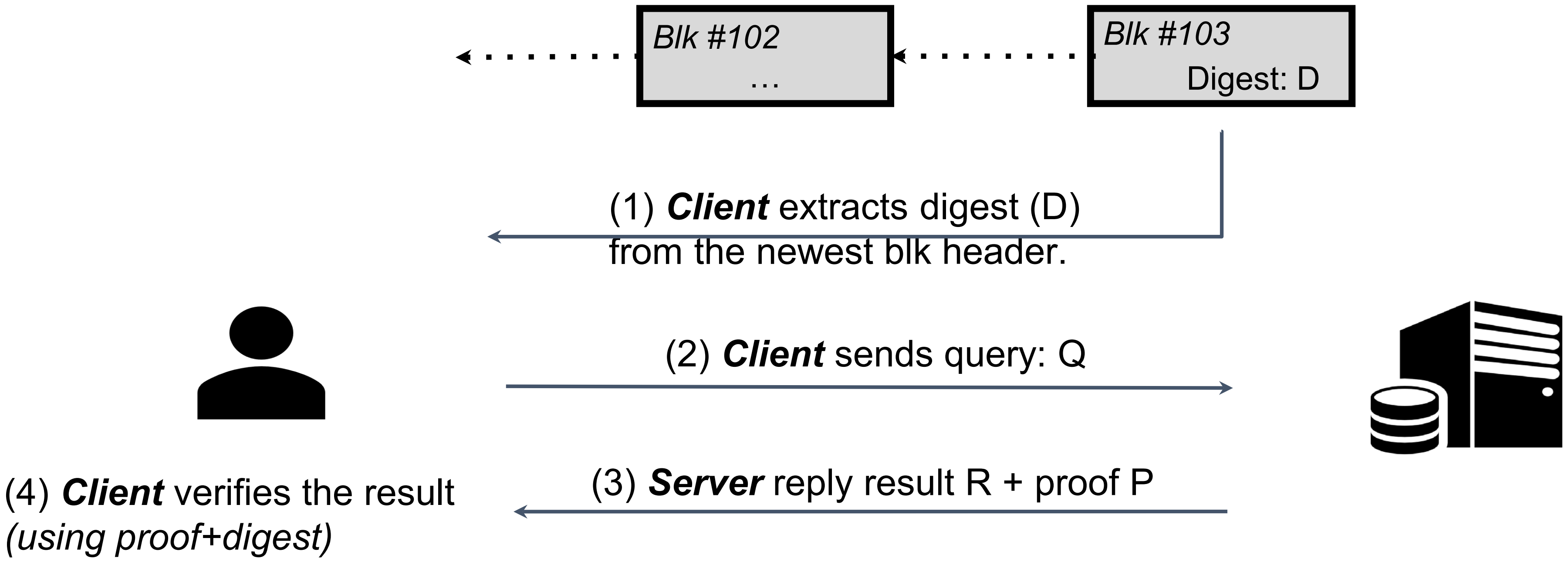
❖Full nodes (servers):

➢ store data + update logs (block data)

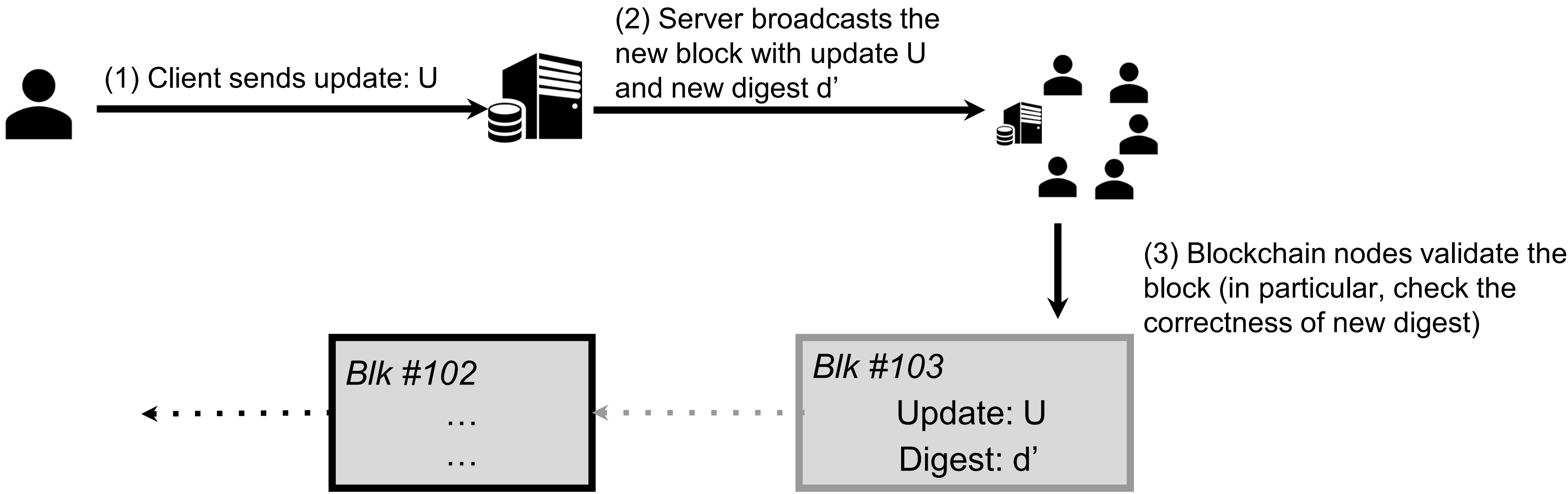
❖Light nodes (clients):

➢ store digest of data (block header)

FalconDB: verifiable query execution



FalconDB: updating the database



Challenge: VC overhead

1. **Fetch newest digest**
2. Send query

3. Execute query
4. **Generate proof**
5. Return result
6. **Return proof**

7. Validate the result with proof and digest

Overhead: Steps *1*, **4**, *6*, *7* (temporal)

Step 4 could be very slow for complicated queries

- ➢ Normal query: ~10s; Big query: ~6000s
- ➢ Bottleneck of query/update!

Asynchronous proof generation

❖ Recall:

➢ Query execution is fast; proof generation is slow.

❖ Observation:

➢ With ADS, all dishonest behavior from server could be detected.

❖ Solution:

➢ Optimistically trust the results from server and ask for a proof later.

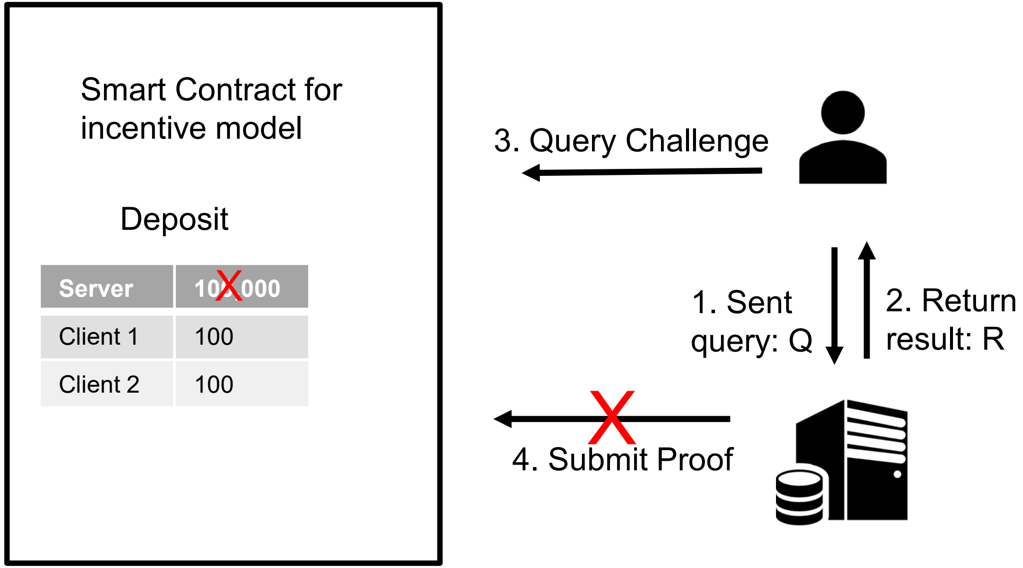
➢ Proof generation become asynchronous; won't block any process.

➢ Impose high penalty if the server fails to provide proof later.

➢ Use external smart contract as incentive model.

Incentive model

- Servers deposit to a smart contract.
- Client can challenge the server.
- Failed to provide a proof in time -> deposit is confiscated.
- Successfully provide a proof -> client pays the server a transaction fee.



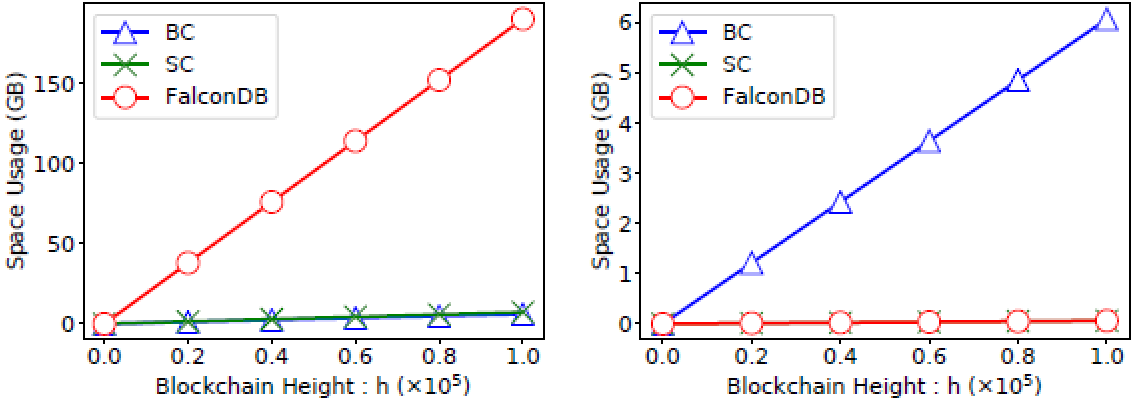
FalconDB advantages

- Low requirements on clients
  - Allow participating from any device
- Secure
  - Result Integrity
  - Transparent and Immutable updates
  - Clients can’t be cheated even all full nodes are malicious
- High performance
  - Query Performance = Traditional Server-Client
  - Blockchain performance / validation performance / update performance = Most state-of-the-art work
    - Since blockchain consensus and ADS are used as blackboxes, we can always replace them by the newest work

Evaluation Setup

- Baseline:
  - "Naïve Blockchain": each node maintains the full database; sync updates with blockchain
  - "Smart Contract": each full node maintains the database; light nodes query by submitting queries to the blockchain
- FalconDB setting:
  - 5 servers, 27 clients
- Database workload:
  - a single table with n rows and m columns

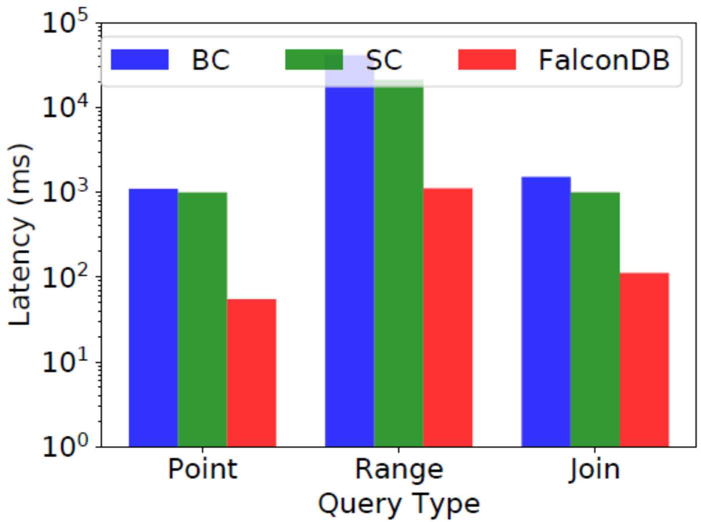
Evaluation

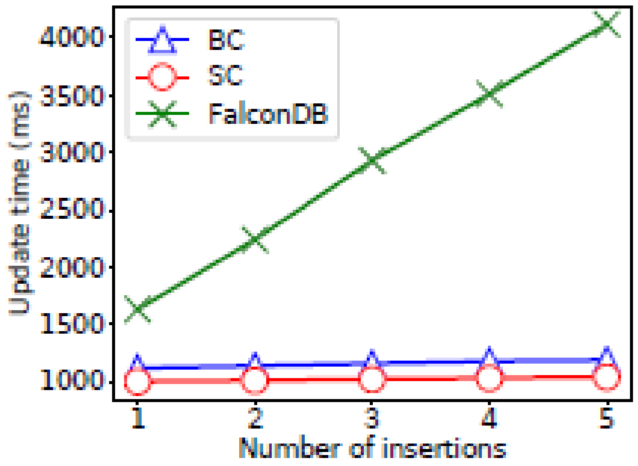


(a) Server **(b) Client**

- Space cost on Servers and Clients
- FalconDB shifts the high storage cost from local clients to server only.

Evaluation





- Query and Update performance
- FalconDB has best query performance
- Updates are slower but acceptable

Summary

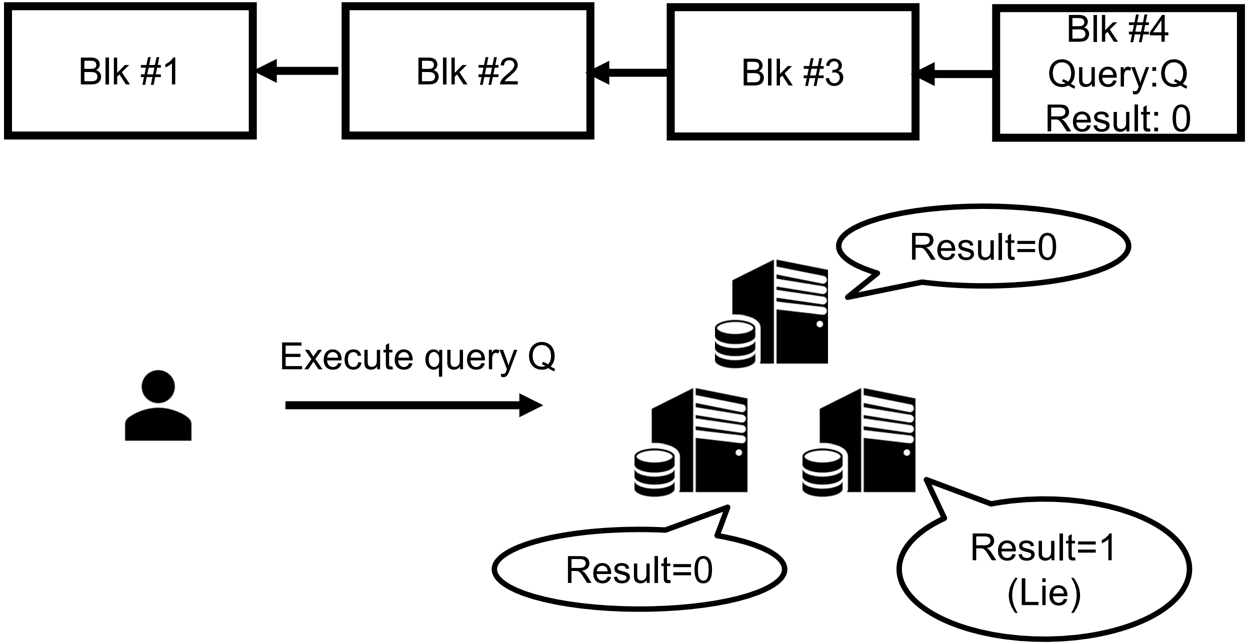
FalconDB:

- ➢ A blockchain based collaborative database.
- ➢ Clients store a little piece of data, and connect to servers to issue query/update.
- ➢ Clients are able to verify query/update results with authenticated data structures (ADS).
- ➢ High performance, high security guarantee.

THANK YOU!

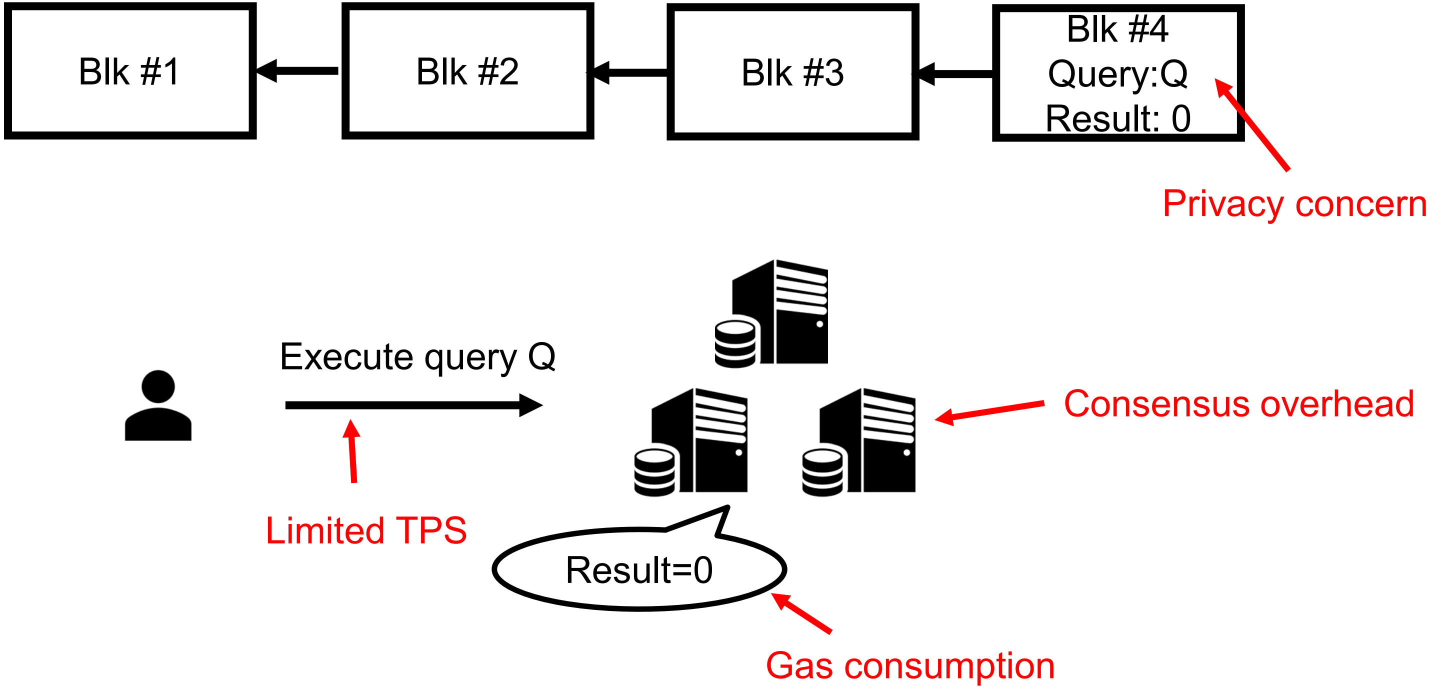
Backup Slides

Baseline Solution #1: Smart Contract



Baseline Solution #1: Smart Contract

Drawbacks...



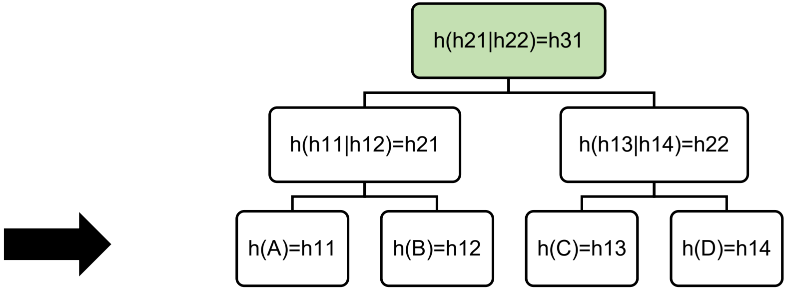
Baseline Solution #2: Verifiable Computation

- ❖Verification Computation (VC)
  - ➢Service provider returns results with cryptographic proof;
  - ➢Users verify integrity of results using the proof.

- ❖Enabling trust on full nodes:
  - ➢Light nodes query full nodes, and verify the results using VC.

VC as a blackbox

- Summary
  - Data D -> ADS S * Digest δ



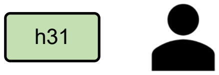
Input: array data

Output: merkle tree over the array

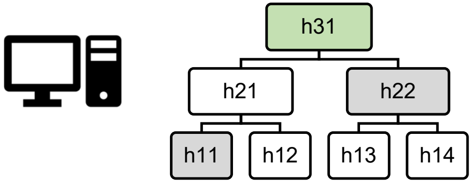
Example: array data with Merkle tree as VC data structure

VC as a blackbox

- Query (Server side)
  - Data D * Query Q -> Result R * Proof π
- Verify Query (Client side)
  - Digest δ * Query Q * Result R * Proof π -> {0, 1}



Send: Array[1]=?



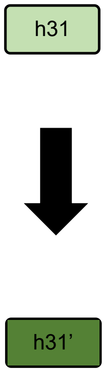
Return: Array[1]=B
Return: Proof = (h11, h22)

Check: h(h(h11|h(B))|h22)=digest?

Example: array data with Merkle tree

VC as a blackbox

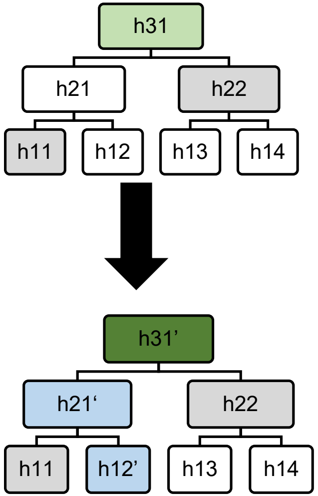
- Update (Server side)
  - Data D * Update $\mathbf{U}$ -> Updated data D' * Digest of updated data δ' * Proof π
- Verify Update (Client side)
  - Old digest δ * New digest δ' * Update $\mathbf{U}$ * Proof π -> {0, 1}



Send: Array[1]:=B’

Check: h(h(h11|h12)|h22)=digest?
Check: h31’= h(h(h11|h(B))|h22)?
Update: digest=h31’

Calculate: h12’ = h(B);
h21’=h(h11|h12’);
h31’=h(h21’|h22)
Return: Proof = (h11, h12, h22, h31’)



Example: array data with Merkle tree

Comparison between different approaches

	Client storage	Client set-up	Query execution side	Query Throughput	Query Latency	Trust
Centralized	**No**	**No**	**Server**	**High**	**Low**	Server
Blockchain	High	Slow	Client	Low	High	**No**
Smart contract	**Low**	**Quick**	**Server**	Low	High	**No**
Blockchain +VC	**Low**	**Quick**	**Server**	**High**	Depends on proof generation	**No**

Challenge

Other tech details

- ❖ Blockchain scalability
  - ➢ Algorand
- ❖ Easy history retrieval
  - ➢ Temporal database model
- ❖ Supporting DB transactions
  - ➢ Optimistic Concurrency Control