ForkBase: An Efficient Storage Engine for Blockchain and Forkable Applications

Sheng Wang, Anh Dinh, Qian Lin, Zhongle Xie, Meihui Zhang[#], Qingchao Cai, Gang Chen[^], Beng Chin Ooi, Pingcheng Ruan

National University of Singapore
[#]Beijing Institute of Technology
[^]Zhejiang University

Emerging Opportunities for Storage Systems

- **A Rich Pool of Storage Engines**
  - Relational, Key-Value, Document …
  - Developers can select best solutions for their applications

- **Any New Requirements from Emerging Applications?**
  - Blockchain (Tamper Evidence)
  - Collaborative dataset management (Fork & merge, Deduplication)
  - …
  - Native storage support?

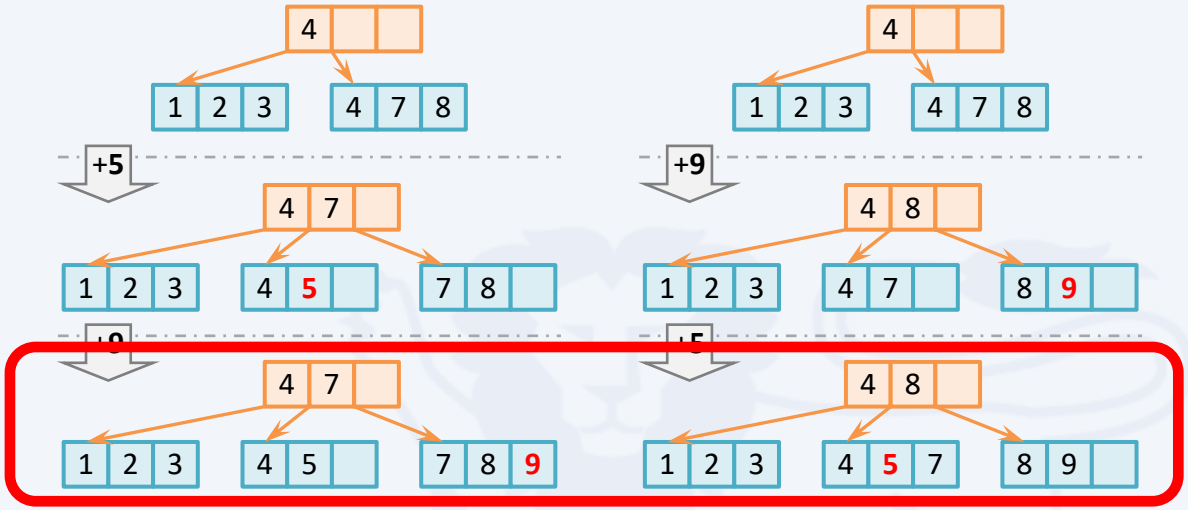
Motivation – Versioning

- **Collaborative (Structured) Dataset Management**

- **More Deduplication Scenarios**
  - Implicit deduplication
    - **Data model handles versions as separate records (e.g. Wikipedia)**
  - Global deduplication
    - **Users upload same (sub-)datasets independently (e.g. cloud repository)**

Motivation – Implicit/Global Deduplication

- **Unstructured Data**
  - Well studied in file system community
  - Canonical method: chunk (e.g., content-based chunking) deduplication
- **Structured Data**
  - Page dedup is ineffective
    - **e.g., B+tree**
  - Nature of indexes
    - **internal representation vary from update seqs**

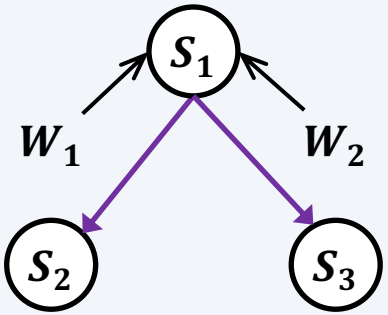


- **ForkBase** proposes a new index class
  - Structurally-Invariant Reusable Indexes (SIRI)
  - Effective page-level deduplication

Motivation – Ubiquitous Forking

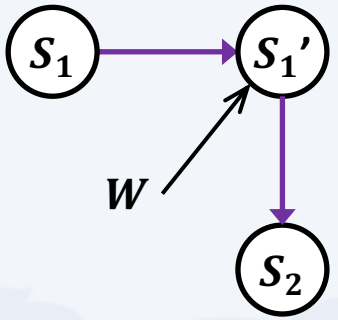
Fork on Conflict

System forks a branch upon conflict detected



Fork on Demand

User creates a branch on purpose



Forking Semantics

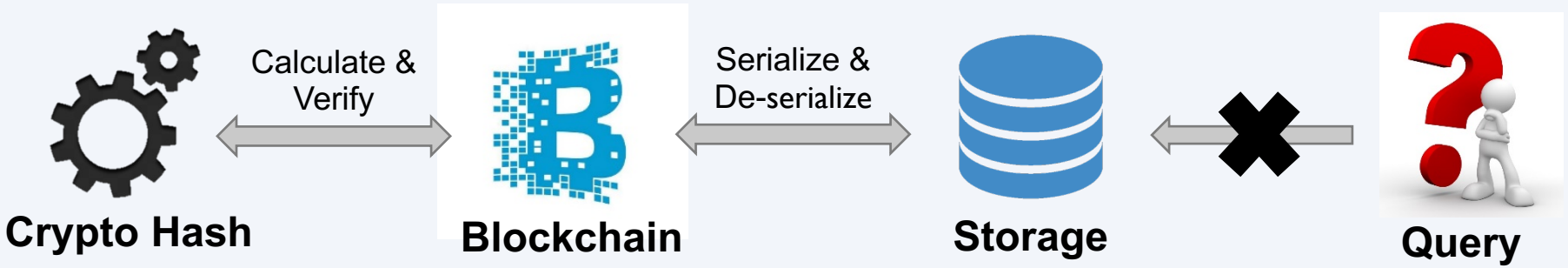
amazon DynamoDB bitcoin

DataHub

- **ForkBase** natively supports both forking semantics
  - simplify application logic
  - lower development efforts
  - preserving high performance

Motivation – Blockchains

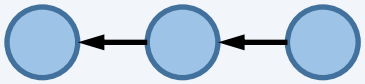
- **State-of-the-art Storage for Blockchain**
  - Simple key-value stores: LevelDB & RocksDB



- **ForkBase is the first Native Storage for Blockchains**
  - Tamper-evident data types for modeling complex data structures
  - Built-in support for forking semantics and query functionalities
  - …
  - Decouple data-related logics from upper layers
  - Make Blockchains analytics-ready

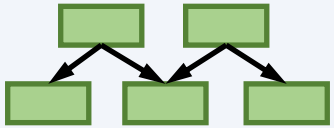
Core Designs

Versioning & Tamper Evidence



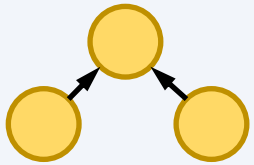
Merkle DAG

Indexing & Deduplication

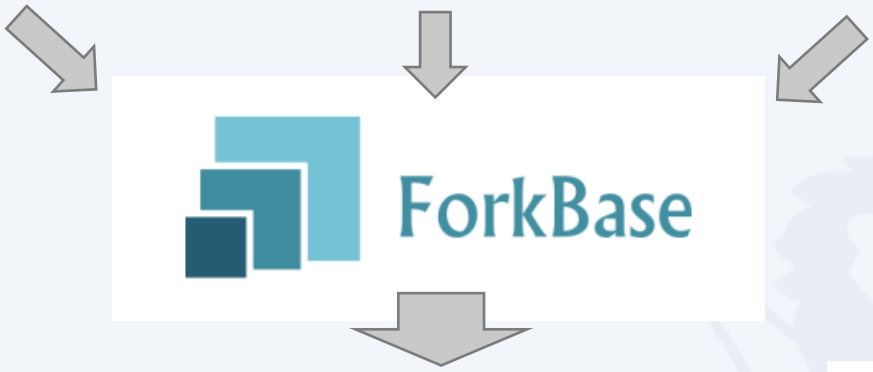


SIRI Indexes

Collaboration Workflows

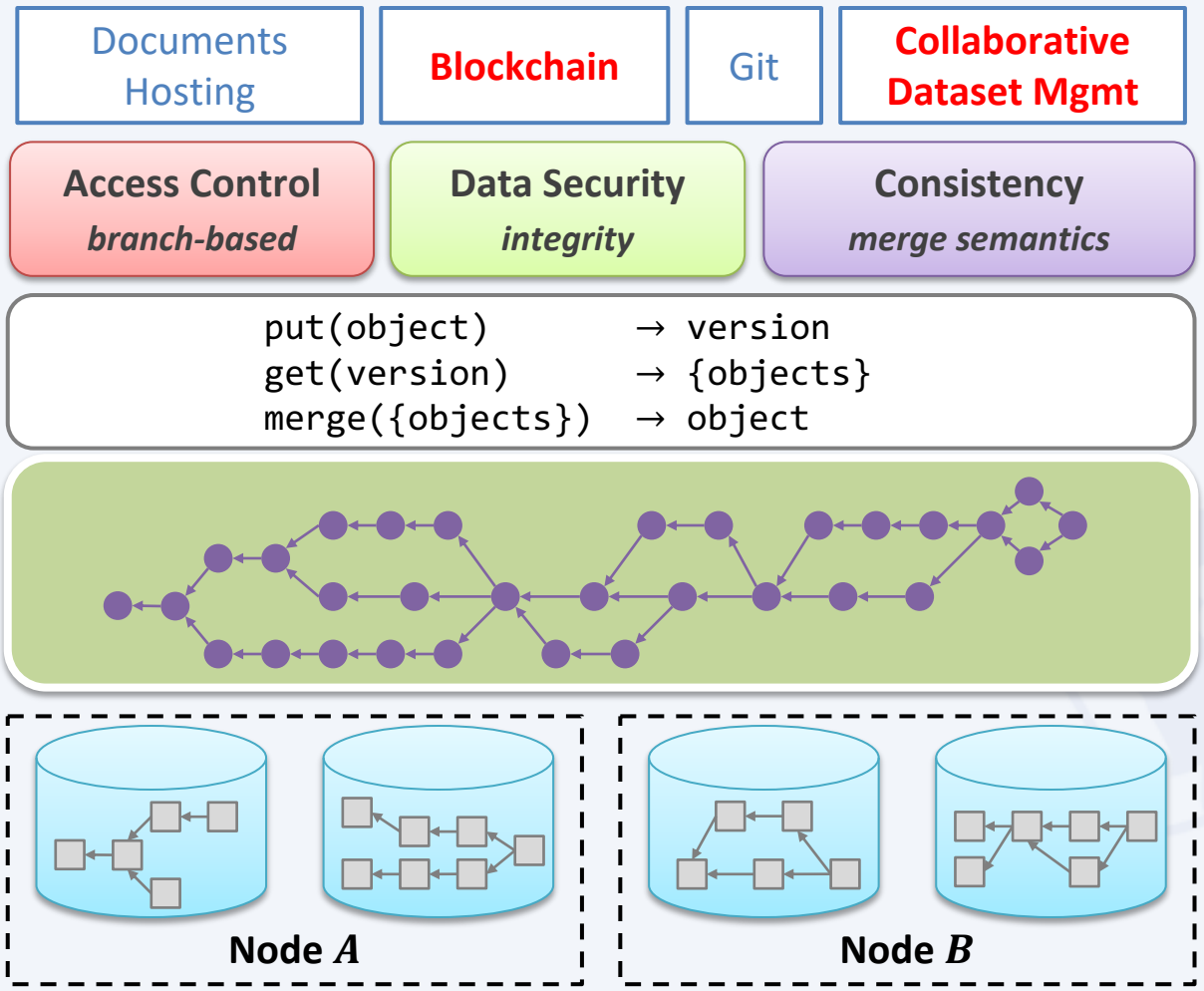


Fork Semantics



git (versioning) + **database** (query) + **blockchain** (integrity)

Component Stack



Applications

Semantic Views
(application-oriented)

Data Access APIs
(data types, fork semantics)

Branch Representation
(versioning, tamper evidence)

Chunk Storage
(deduplication, immutability)

Structurally-Invariant Reusable Indexes (SIRI)

- **Structurally Invariant**
  - For any index instance I_1 and I_2 :
  - $record(I_1) = record(I_2) \Leftrightarrow page(I_1) = page(I_2)$
    - **Internal representation is exclusively determined by content**

- **Recursively Identical**
  - For any index instance I_1 and I_2 such that $record(I_2) = record(I_1) + r$:
  - $|page(I_2) - page(I_1)| \ll |page(I_2) \cap page(I_1)|$
    - **Evolved/updated indexes can reuse most of existing pages**

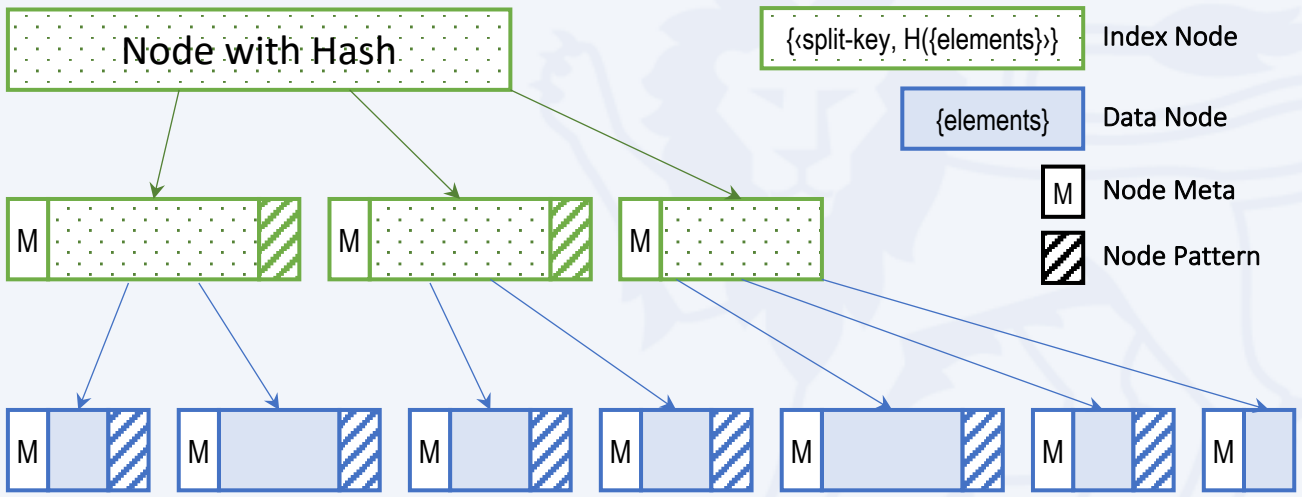
- **Universally Reusable**
  - For any index instance I_1 and $p \in page(I_1)$, there exists instance I_2 :
  - $(|page(I_2)| > |page(I_1)|) \wedge (p \in page(I_2))$
    - **Any page is reusable, i.e., not exclusively belong to an instance**

Pattern-Oriented-Split Tree (POS-Tree)

- **An Implementation of SIRI**
  - **Content-based** node split → structural invariant
    - **Split a node when a pre-defined pattern occurs**
  - **B+-tree** variant → probabilistically balanced
  - **Merkle tree** variant → tamper evidence, deduplication
  - Chunkable **data types**: blob, list, map, set …

Index node pattern
H(entry) < C
(efficiency)

Data node pattern
Rolling Hash
(randomness)



Generic Forking Semantics

Fork on Demand

Fork – create an tagged branch from another branch
Read – return committed data from a branch
Commit – update a branch with new data
Diff – find the differences between branches
Merge – merge two branches and their histories

Fork on Conflict

Read – choose and read a branch based on a **policy**:
* any/exact/descendant/…
Commit – update a branch based on a **policy**:
* exact/descendant/…
List – return all conflicted branches
Resolve – resolve conflicts and merge branches

Dataset Management

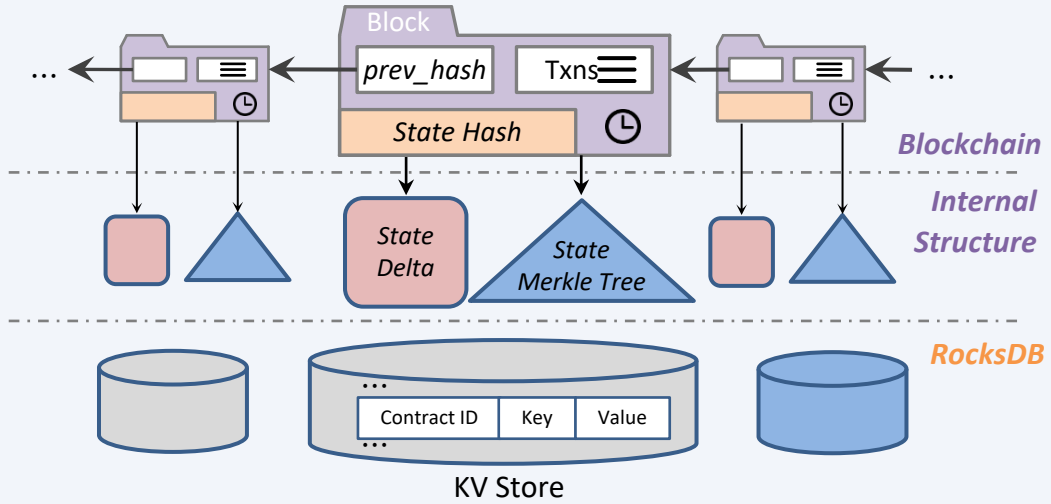
```
db.Commit(repo, main_brc, data);
db.Fork(repo, main_brc, new_brc);
dataset = db.Read(repo, new_brc);
dataset.add(new_table);
db.Commit(dataset);
db.Merge(repo, main_brc, new_brc);
```

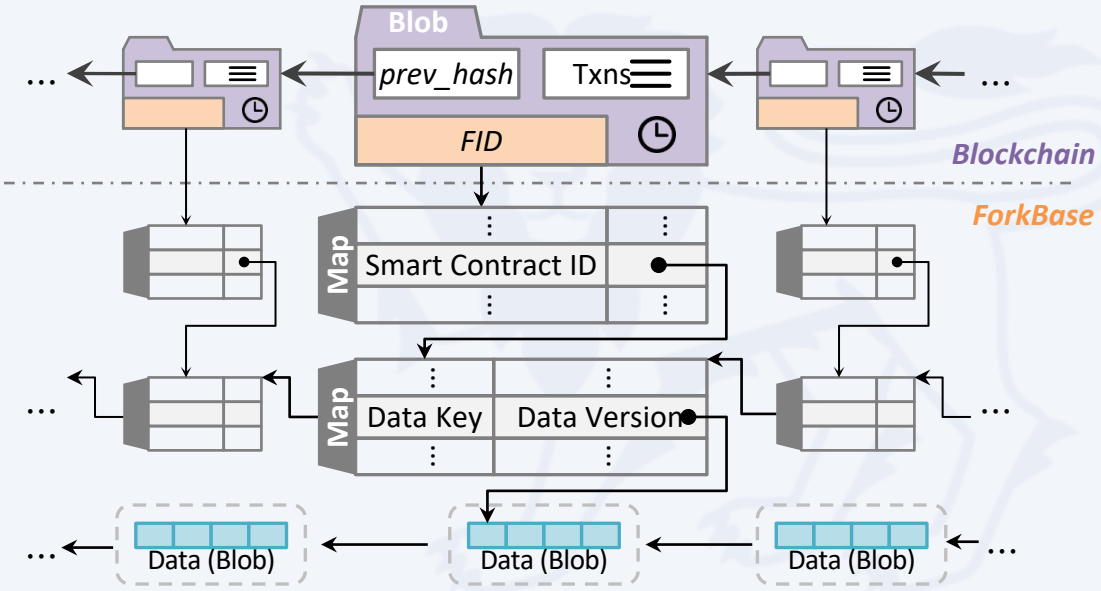
Blockchain

```
// commit or receive a block
db.Commit(domain, block, exact);
// check if the chain is forked
heads = db.List(domain);
// choose the longest chain
db.Resolve(domain, heads, UDF);
```

Blockchain Data Models in ForkBase

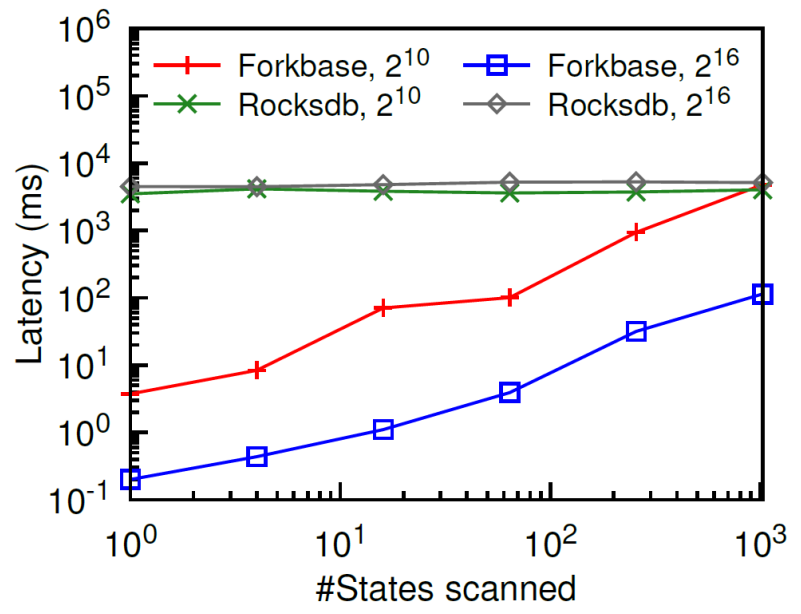
- **Key-Value Store**
  - Customized structures
    - **Linked block**
    - **State Merkle tree**
    - **State delta**
    - **…**
  - Hard to implement
- **ForkBase**
  - Achieve with built-in types
    - **UBlob**
    - **UMap**
    - **…**
  - Easy to maintain
    - **10+ lines for each structure**



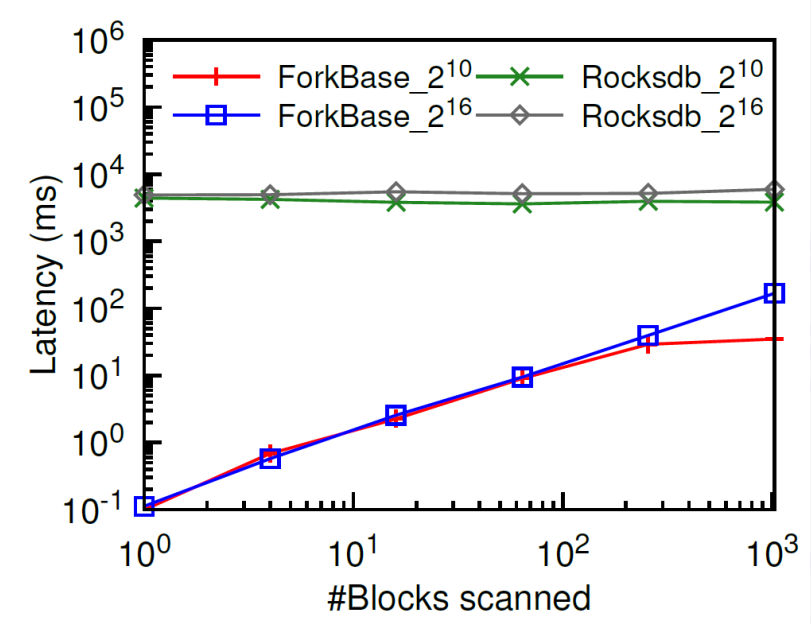


Analytics-Ready Blockchain Backend

- **Analytic on blockchain is expensive**
  - Need to scan whole block history to extract information
- **Built-in data types support fast analytics**



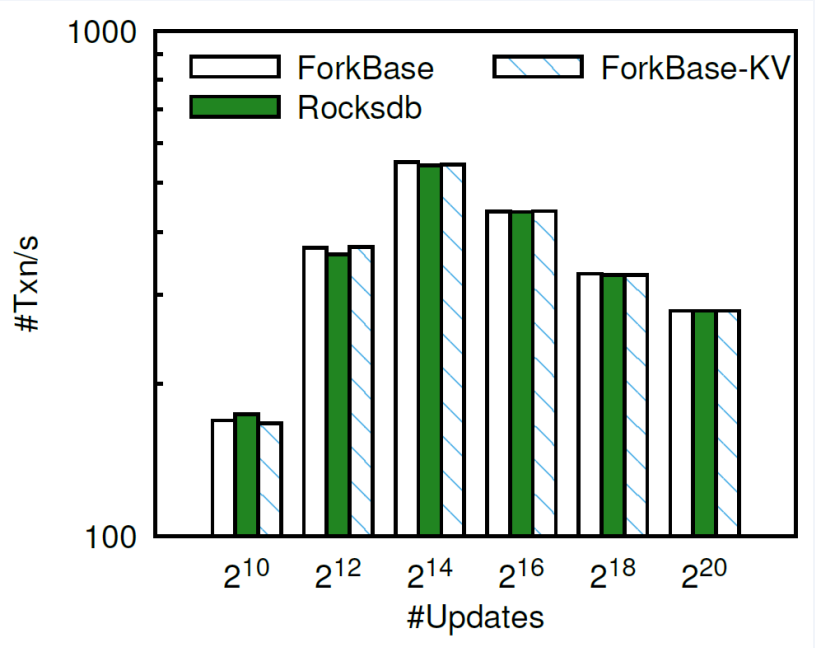
State Scan Query



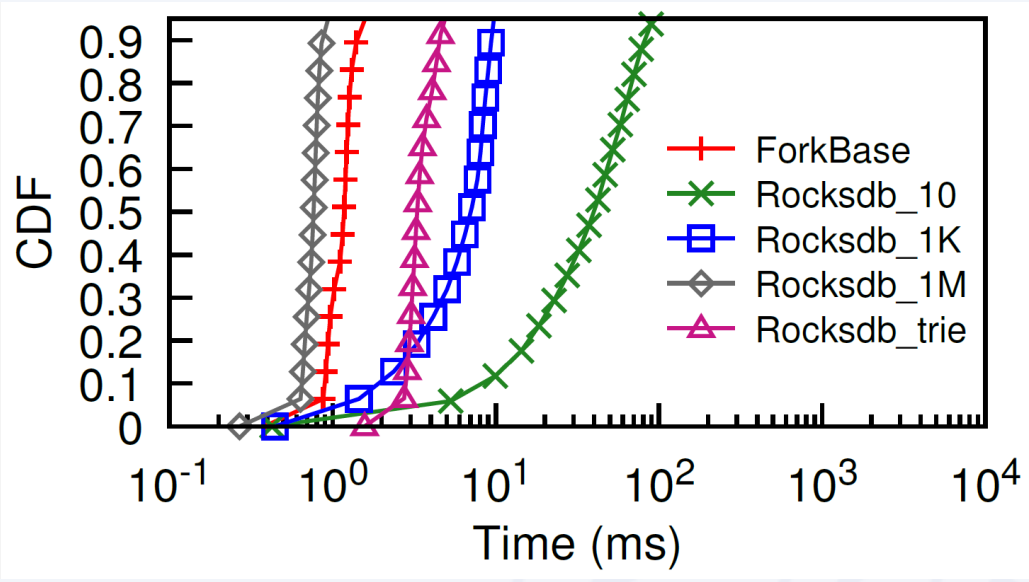
Block Scan Query

Online Operations in Blockchain

- **Fast transactions with stable performance**
  - Analytic functionalities do not degrade common online operations
  - Data access latency is more stable (UMap v.s. Merkle Tree)



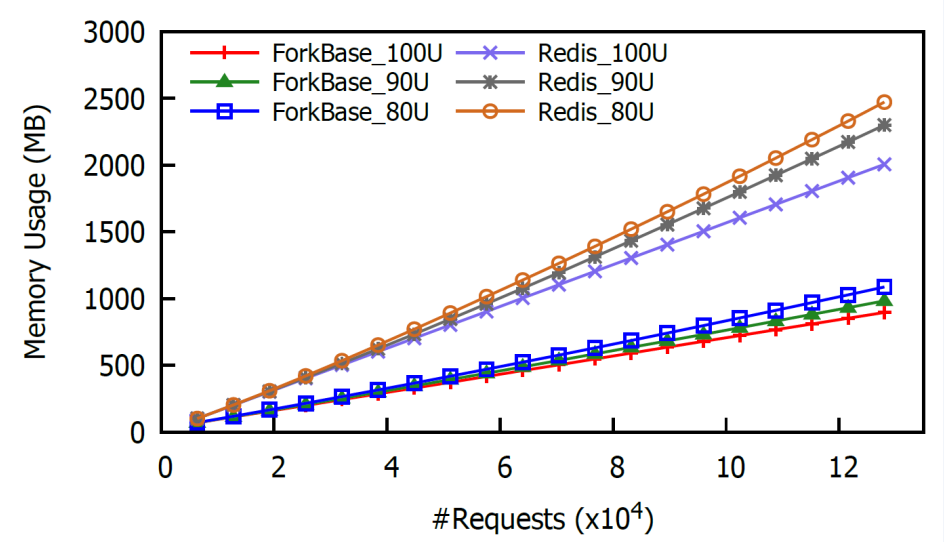
Client Throughput



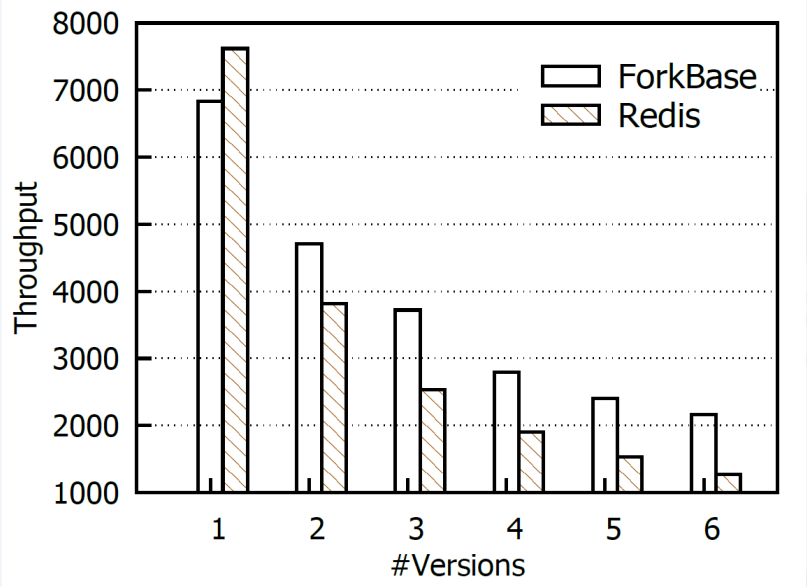
Data Access Latency

Wiki Engine for Documents

- **Implicit deduplication (POS-Tree) improves performance**
  - Redundant content are automatically detected and removed
  - Identical chunks can be cached/reused to enable delta fetching



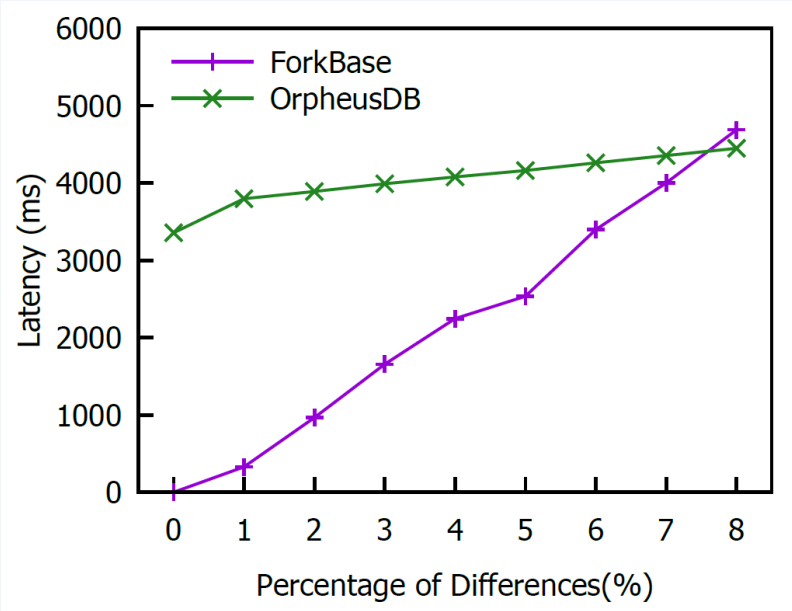
Storage Consumption



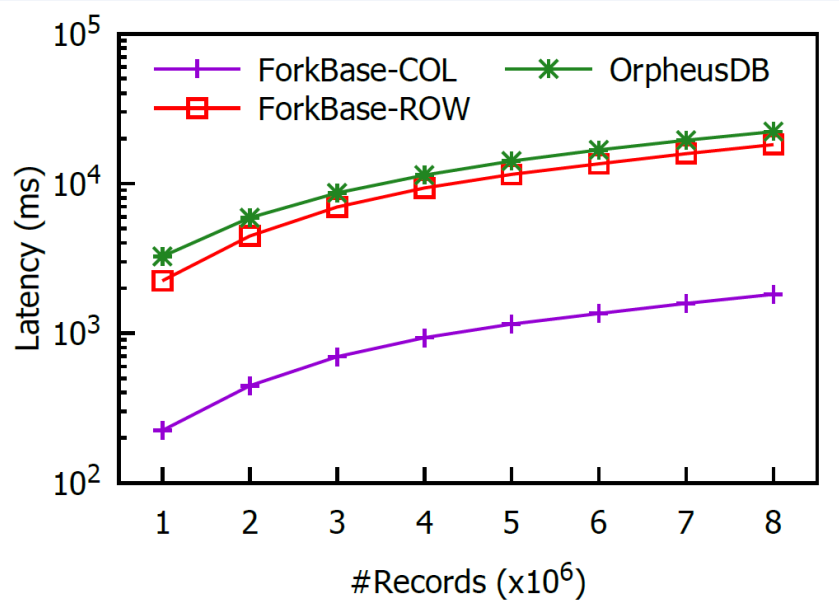
Consecutive Read Throughput

Collaborative Dataset Management

- **Built-in data types enable modeling of relational tables**
  - POS-Tree supports fast pruning of identical sub-trees
  - No need to reconstruct a whole dataset before querying it



Diff Query Latency



Aggregation Query Latency

Highlights

- **SIRI Indexes & POS-Tree → Effective Deduplication**
- **Forking Semantics → Various Collaboration Workflows**
- **Powerful Building Blocks for Emerging Applications**
  - Versioning, forking semantics, tamper-evidence, deduplication …
  - Simplify application logics
  - Reduce development efforts
  - Deliver better performance
    - **storage consumption, query performance …**

Thank you!

Questions & Answers