Heterogeneous, Efficient Auditing of Replicated State Machines

Workshop on Heterogeneous Trust in Distributed Systems

Geoff Ramseyer, David Mazières

Stanford University

October 26, 2023

Digital Cash is on the Horizon...

... but what does infrastructure for society-scale deployment need?

February 3, 2022

Project Hamilton Phase 1 Executive Summary

Federal Reserve Bank of Boston and Massachusetts Institute of Technology Digital

2020 年 10 月 9 日
日 本 銀 行

中央銀行デジタル通貨に関する日本銀行の取り組み方針

（要 旨）

情報通信技術の急速な進歩を背景に、内外の様々な領域でデジタル化が進んでいる。技術革新のスピードの速さなどを踏まえると、今後、「中央銀行デジタル通貨」（Central Bank Digital Currency：以下「ＣＢＤＣ」）に対する社会のニーズが急激に高まる可能性もある。日本銀行では、現時点でＣＢＤＣを発行する計画はないが、決済システム全体の安定性と効率性を確保する観点から、今後の様々な環境変化に的確に対応できるよう、しっかり準備しておくことが重要であると考えている。こうした認識のもと、今般、個人や企業を含む幅広い主体の利用を想定した「一般利用型ＣＢＤＣ」について、日本銀行の取り組み方針を示すこととした。

１．ＣＢＤＣを導入する場合に期待される機能と役割

ＣＢＤＣには、「ホールセール型ＣＢＤＣ」と「一般利用型ＣＢＤＣ」の２つの形態があるが、わが国において一般利用型ＣＢＤＣを導入する場合に期待される機能や役割としては、以下のようなものが考えられる。

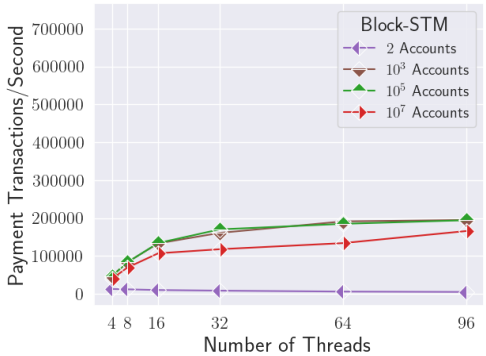
Our work aims to ensure that in the digital age citizens and firms continue to have access to the safest form of money, central bank money.

What does Digital Cash Need?

- **Extensibility**
  - Arbitrary, unknown use-cases and applications
- **Core Infrastructure Scalablility**
  - Infrastructure should run arbitrarily fast
- **End-User Scalability**
  - Heterogeneous users with varying requirements and constraints

Where do existing solutions stand today?

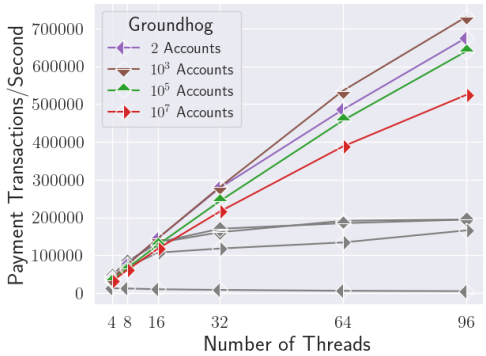
- ✓ **Adaptability**
  - Smart contracts, custom code in transactions
- ✖ **Core Infrastructure Scalability**
  - Conventional wisdom says Layer-1 scalability is impossible
- ✖ **End-User Scalability**
  - Nontrivial trust assumptions or significant hardware requirements for end users



State of the art smart contract execution engine

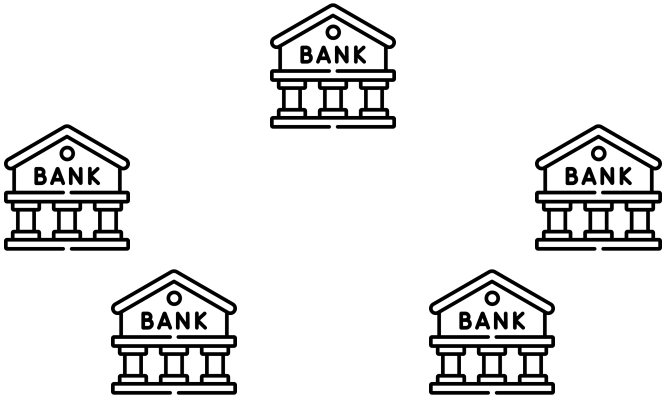
Where do existing solutions stand today?

- ✓ **Adaptability**
  - Smart contracts, custom code in transactions
- ✓ **Core Infrastructure Scalability**
  - Near-linear, Layer-1 computational scalability
- ✓ **End-User Scalability, Auditability**
  - End-user costs proportional to individual usage and requirements



Today's Talk: Near-linear core scalability, and auditing n transactions costs $\tilde{O}(n)$

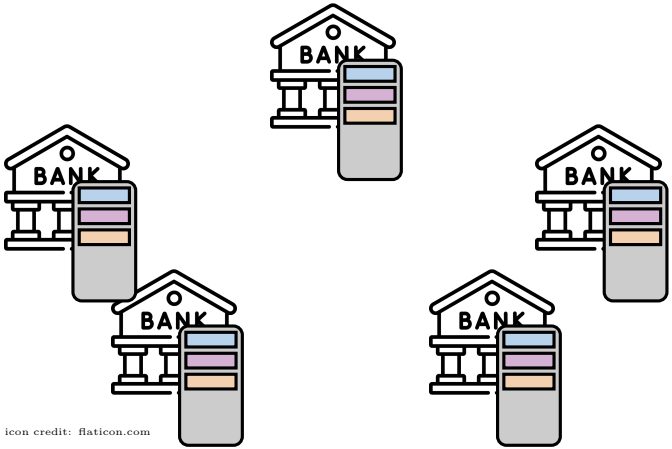
Architecture Overview



icon credit: flaticon.com

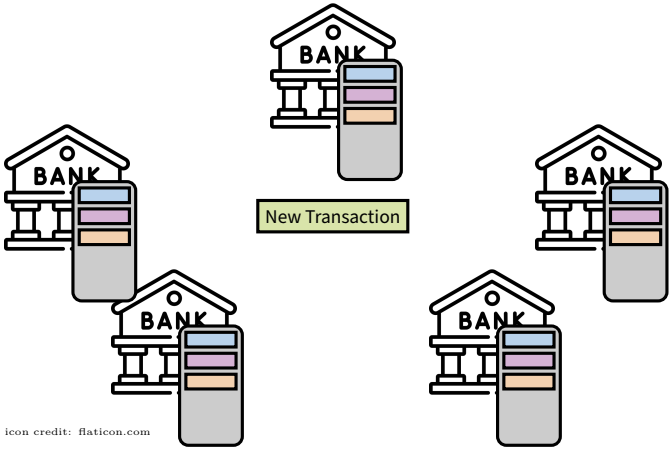
- Many replicas maintain whole copies of a shared ledger
- End-users send transactions to replicas

Architecture Overview



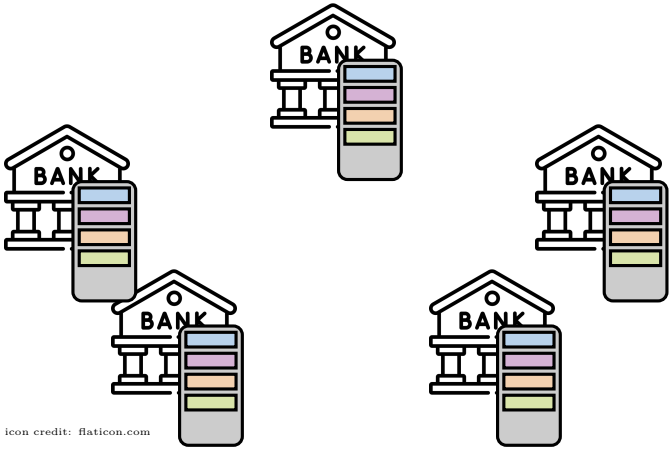
- Many replicas maintain whole copies of a shared ledger
- End-users send transactions to replicas

Architecture Overview



- Many replicas maintain whole copies of a shared ledger
- End-users send transactions to replicas

Architecture Overview



- Many replicas maintain whole copies of a shared ledger
- End-users send transactions to replicas

Architecture Overview

- **Many possible architectures (rollups, decentralized networks)**
  - Agnostic to consensus protocol
- **Might even be only one replica allowed (e.g. in a CBDC)**
- **Maybe access controls on data**

How Do End-Users Interact With Today's Designs?

- **Option 1: Run a Replica**
  - ✖ Expensive, individual's costs increase with number of total users!
- **Option 2: Trust (a majority of) network**
  - ✖ Requires a nontrivial trust assumption, may not always be satisfied
  - ✖ What if there is only one replica?
- **Option 3: Audit a replica (and produce fraud proofs)**
  - Used in optimistic rollups
  - ✓ Does not require trust in any full node
  - ✖ Costs are equal to running a full node.
  - ✖ Requires entire transaction history.
  - **How can we fix these problems?**

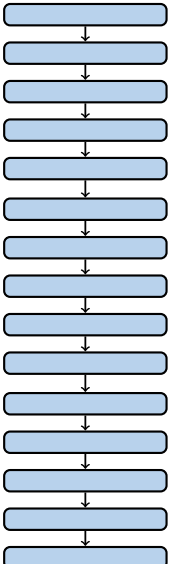
Roadmap

- **Key Idea: Unordered Blocks of Commutative Transactions**
  - Everything depends on choosing the right abstractions
- **Part 1: Commutative Transaction Semantics**
- **Part 2: Heterogeneous, Efficient Partial Audits**

Key Idea: Commutative Transaction Semantics

Most Systems:

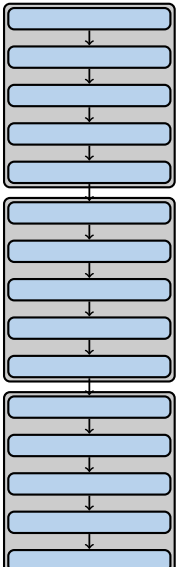
A replicated state machine *deterministically* executes (blocks of) transactions in a *total* order.



Key Idea: Commutative Transaction Semantics

Most Systems:

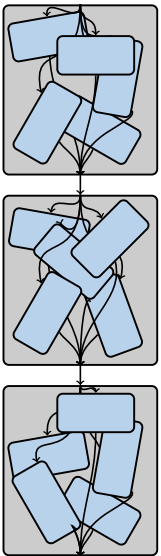
A replicated state machine *deterministically* executes (blocks of) transactions in a *total* order.



Key Idea: Commutative Transaction Semantics

Our systems:

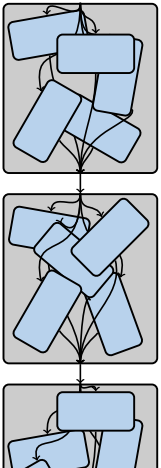
A replicated state machine *deterministically* executes blocks of unordered *commutative* transactions.



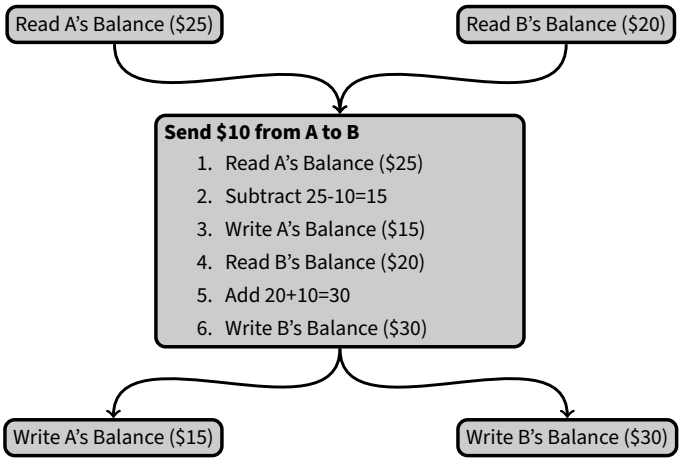
Part 1: Linearly-Scalable Smart Contract Engine

How can commutative transactions do useful work?

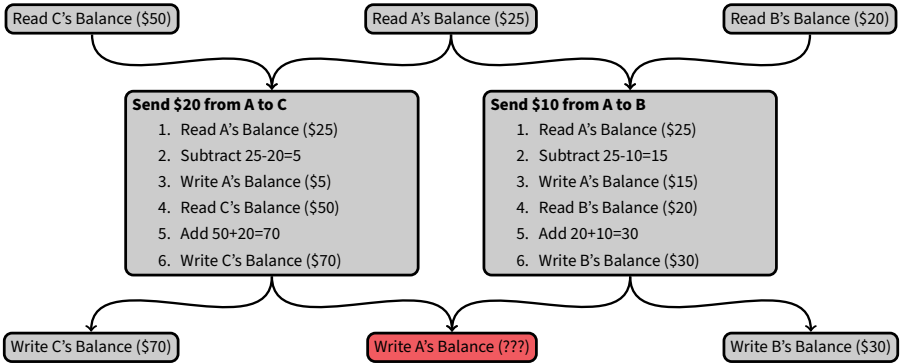
0. **Unordered batches of commutative transactions**
1. **Read from snapshots**
   - Key-value store, typed values
2. **Write through typed state changes**
3. **Maintain application constraints on data**
   - E.g., Account balances must be nonnegative



Example: Payments

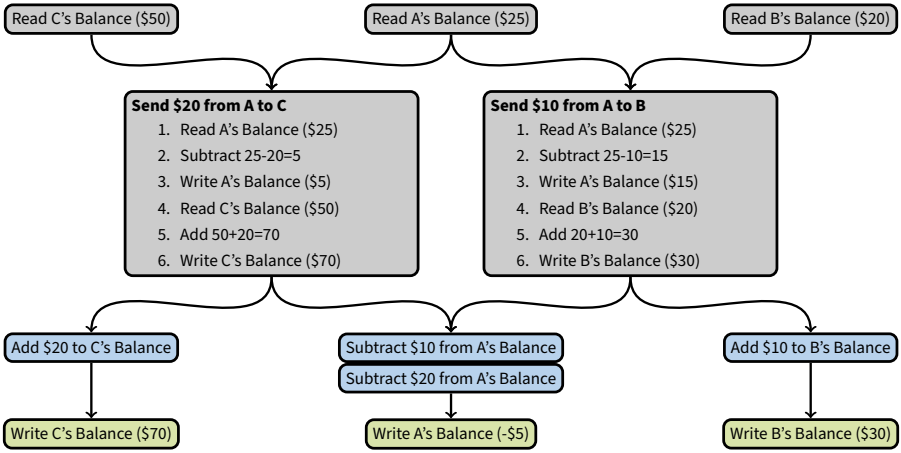


1: Reading from a Snapshot

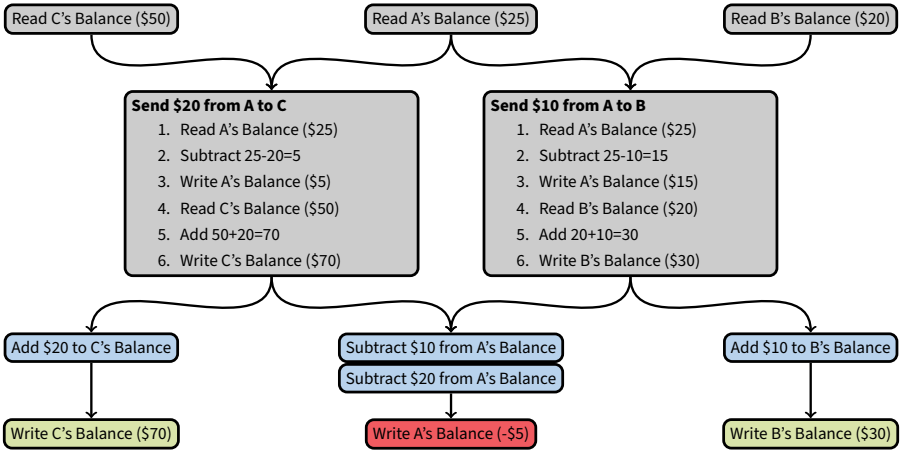


What if two transactions write to the same location?

2: Typed State Changes

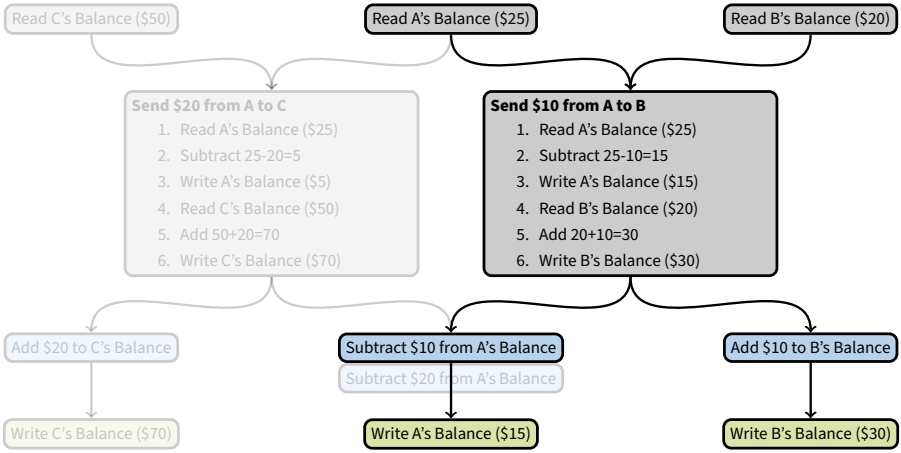


3: Constraints



What if a balance becomes negative?

3: Constraints



Preempt constraint conflicts when assembling blocks
Efficiently implementable via only hardware atomics

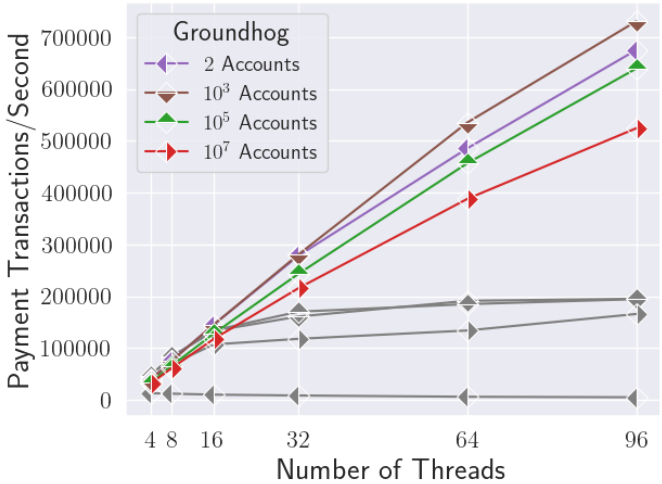
One Minimal Set of Value Types

- **Bytestrings (Read/Write)**
  - Ex: Configuration data
- **Nonnegative integers (Add/Subtract)**
  - Ex: Account balances, linear constraints
- **Ordered Sets (Insert/Clear)**
  - Ex: Replay cache, set of auction bids, set of messages

Groundhog [RM, Working Paper]

- **Implement real applications with minimal API changes**
  - Tokens, Auctions, Money-Markets
- **Implement a sequencing gadget when necessary (asynchronous message-passing)**
  - Used in production today: Near, Zilliqa, CosmWasm,...
- **Applications can choose their own level of parallelism**
  - Only acquire necessary locks, not one global lock
    - ▷ Nonnegative integer primitive directly implements a semaphore
  - Fast applications run quickly, slow ones run slowly
- **Implementation in $\sim 20,000$ LOC C++**
  - `https://github.com/scslab/smart-contract-scalability`

Payments Benchmark



Throughput (mostly) independent of contention on individual account balances

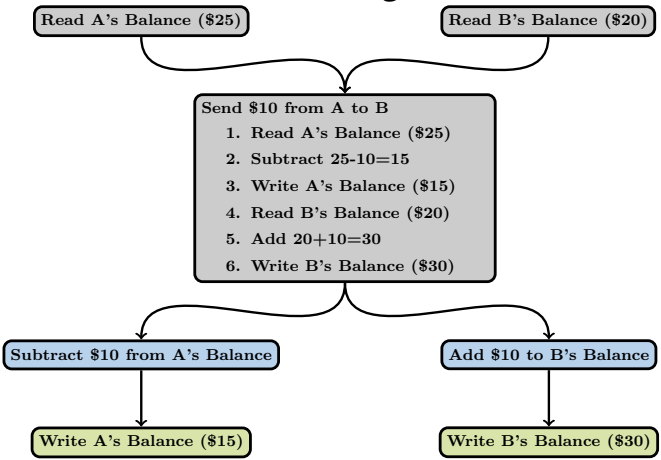
Part 2: Efficient Partial Auditing

How does commutativity enable efficient, partial audits?

- **Assumptions:**
  - User has trusted block headers, with commitments to authenticated data structures
  - User can query openings for these commitments
- **What's in a block header?**
  - Commitment to ledger state
  - Commitment to set of transactions
  - Commitment to an index of *modifications to ledger state*
    - List of modifications on each key

Auditing a Single Transaction

How can we audit a single transaction?



Why is this efficient?

- Every transaction in a block reads from the same snapshot
- Writes to ledger state are indexed in the block header
- Compare with traditional, sequential model

Example Audits

- **Smart contract developer audits all keys (and the transactions that touch them) in their smart contract**
- **User audits their own account balance, over some time period**
  - Cost proportional to number of transactions and length of the time period
- **Bank audits all transactions in its tokenized deposits**
  - Cost proportional only to number of depositors and their activity

Related Work

- **Shards, rollups divide global state into pieces, allowing audits of individual shards.**
  - Coarse granularity in audit requirements, usually reduce resources by only a constant factor
  - Cross-shard communication, bridge mechanisms also require audit
- **Cross-shard communication, bridge mechanisms also require audit**

Conclusion

The right abstractions enable layer-1 decentralized infrastructure that is *extensible*, *computationally scalable*, and *efficiently auditable*

- **Transaction *commutativity* enables:**
  - Parallelizable implementation: near-linear scalability
  - Efficient audits: each user pays only for their own usage.

Thank you!