L2 Interoperability via Collaborative SNARKs

or how to synchronously compose while horizontally scaling

Ben Fisch

CEO/Co-Founder, Espresso Systems

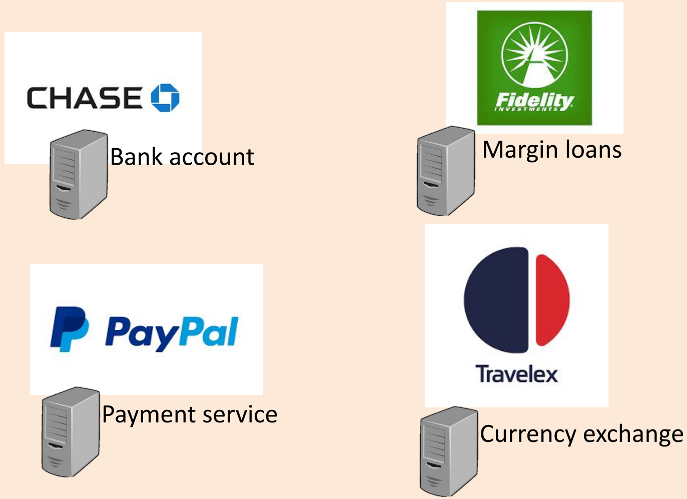
Assistant Professor, Yale University

Espresso Systems

Web 3 is about composable “money legos”

Traditional web apps

- Independently operated
- Need to trust each operator
- Communicate/interact asynchronously & requires business relationship



Bank account

Margin loans

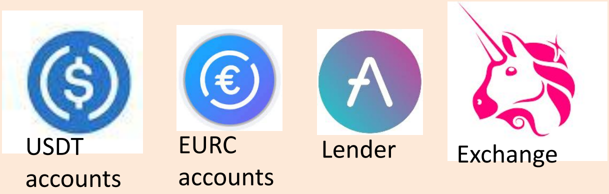
Payment service

Currency exchange

Ethereum apps (“dapps”)

- Updated synchronously, share memory
- Trust only in Ethereum security
- Dapps are composable “money legos”

Developers and users can permissionlessly compose apps created by independent developers



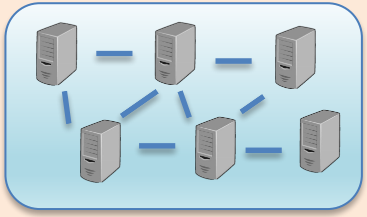
USDT accounts

EURC accounts

Lender

Exchange

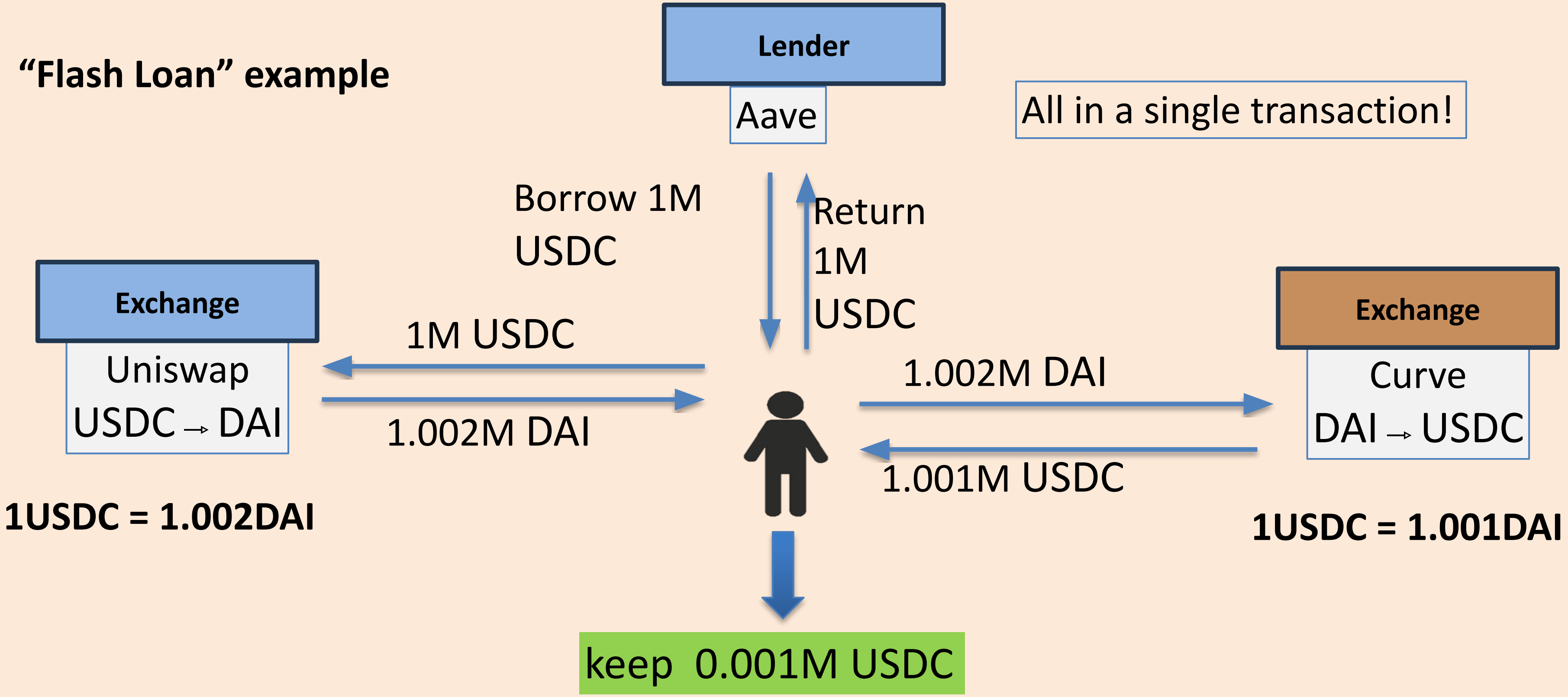
Decentralized operators reach consensus



Web 3 is about composable “money legos”

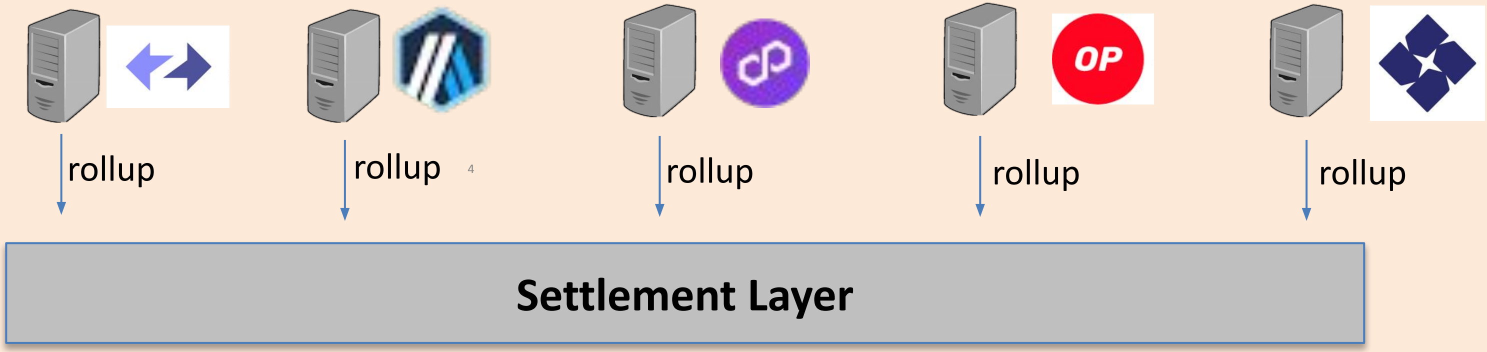
“Flash Loan” example

All in a single transaction!



Rollups horizontally scale Ethereum

A multichain world was born out of a need to scale



<u>Advantages</u>

- *Sharding* of computation across applications
- Powerful nodes help weaker nodes verify state
- VM diversity

What is composability?

- Bridging assets (e.g. moving Eth) from one chain to another
- Cross-chain messages and dependencies
- Cross-chain function calls
- …

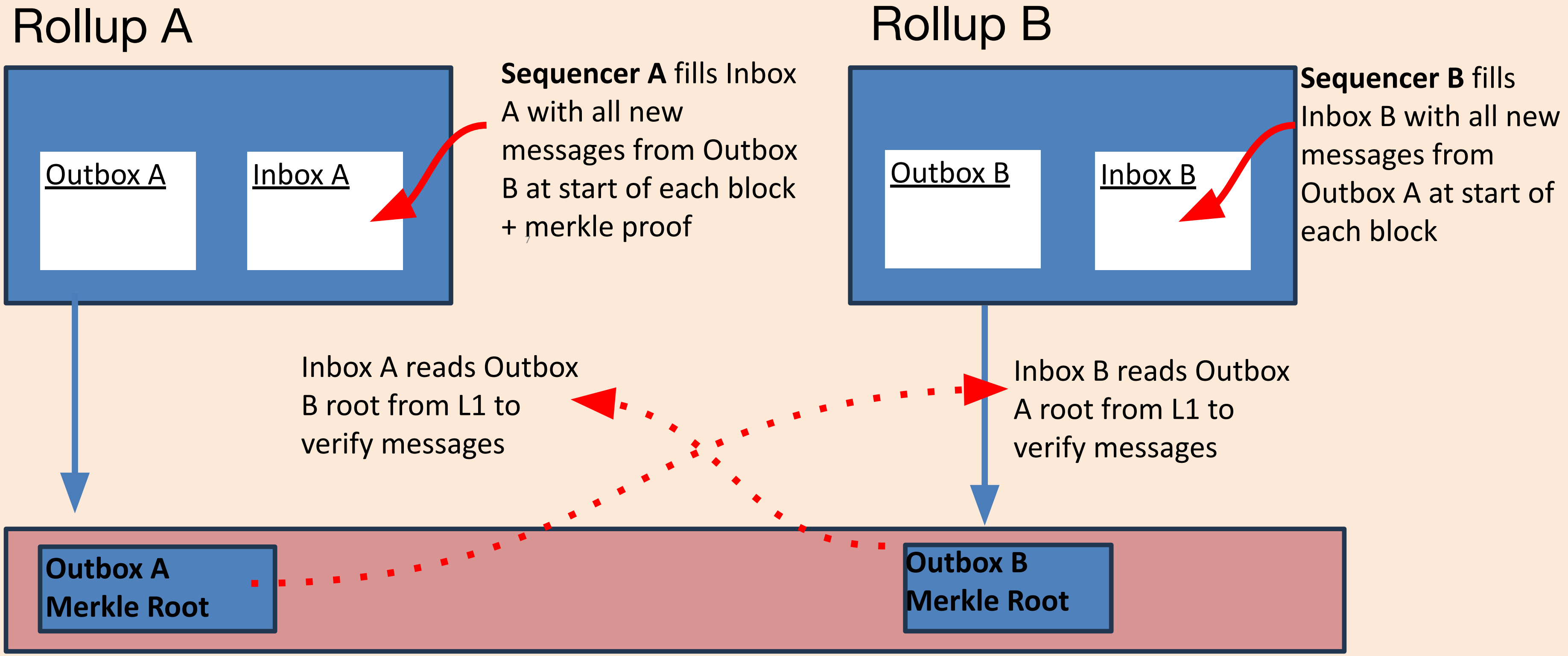
Cross-chain ACID transactions

In a *synchronously composable* cluster of chains, users can express cross-chain transactions/intents (e.g., atomic swap) that have the ACID property:

- **Atomicity** – All parts complete or none do. No in-between state.
- **Consistency** – The rules of each chain are preserved.
- **Isolation** – No interference with read/writes to any chain.
- **Durability** – There is consensus on a persistent global txn log.

Durability & consistency satisfied by any L2s sharing an L1 …

Asynchronous message-passing via L1



Rollup A

Outbox A

Inbox A

Sequencer A fills Inbox A with all new messages from Outbox B at start of each block + merkle proof

Rollup B

Outbox B

Inbox B

Sequencer B fills Inbox B with all new messages from Outbox A at start of each block

Inbox A reads Outbox B root from L1 to verify messages

Inbox B reads Outbox A root from L1 to verify messages

Outbox A Merkle Root

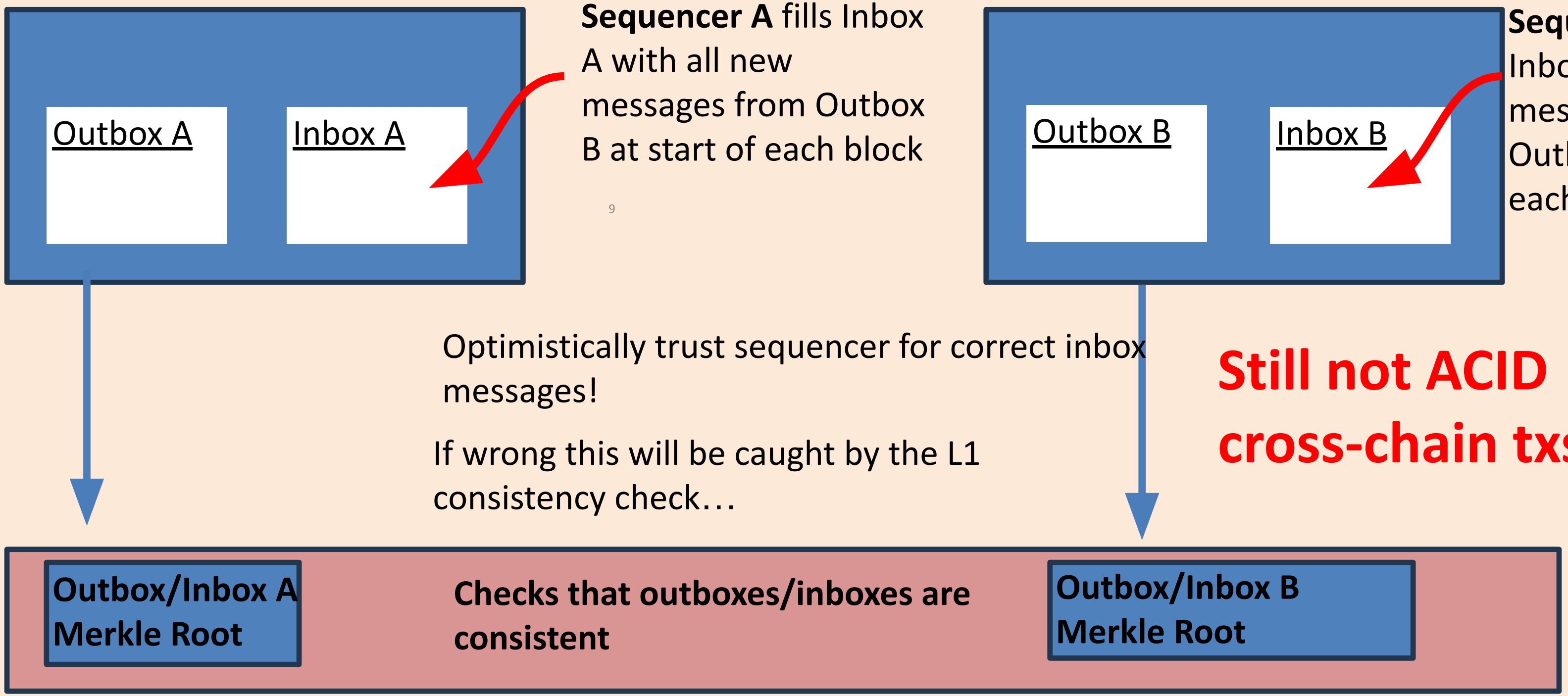
Outbox B Merkle Root

Asynchronous message-passing via L1

- Cross-chain transactions have very high latency
  - Long wait before chain can read message from other chain

- Cross-chain transactions are not ACID
  - **No Isolation:** Other transactions can *read* an intermediary state (e.g., a locked asset on one chain in the case of a cross-chain swap)
  - **Liveness risk:** e.g., censorship on one chain can break atomicity of a cross-chain swap

Message-passing with L1 consistency check



Rollup A

Outbox A

Inbox A

Sequencer A fills Inbox A with all new messages from Outbox B at start of each block

Rollup B

Outbox B

Inbox B

Sequencer B fills Inbox B with all new messages from Outbox A at start of each block

Optimistically trust sequencer for correct inbox messages!

If wrong this will be caught by the L1 consistency check…

Still not ACID cross-chain txs

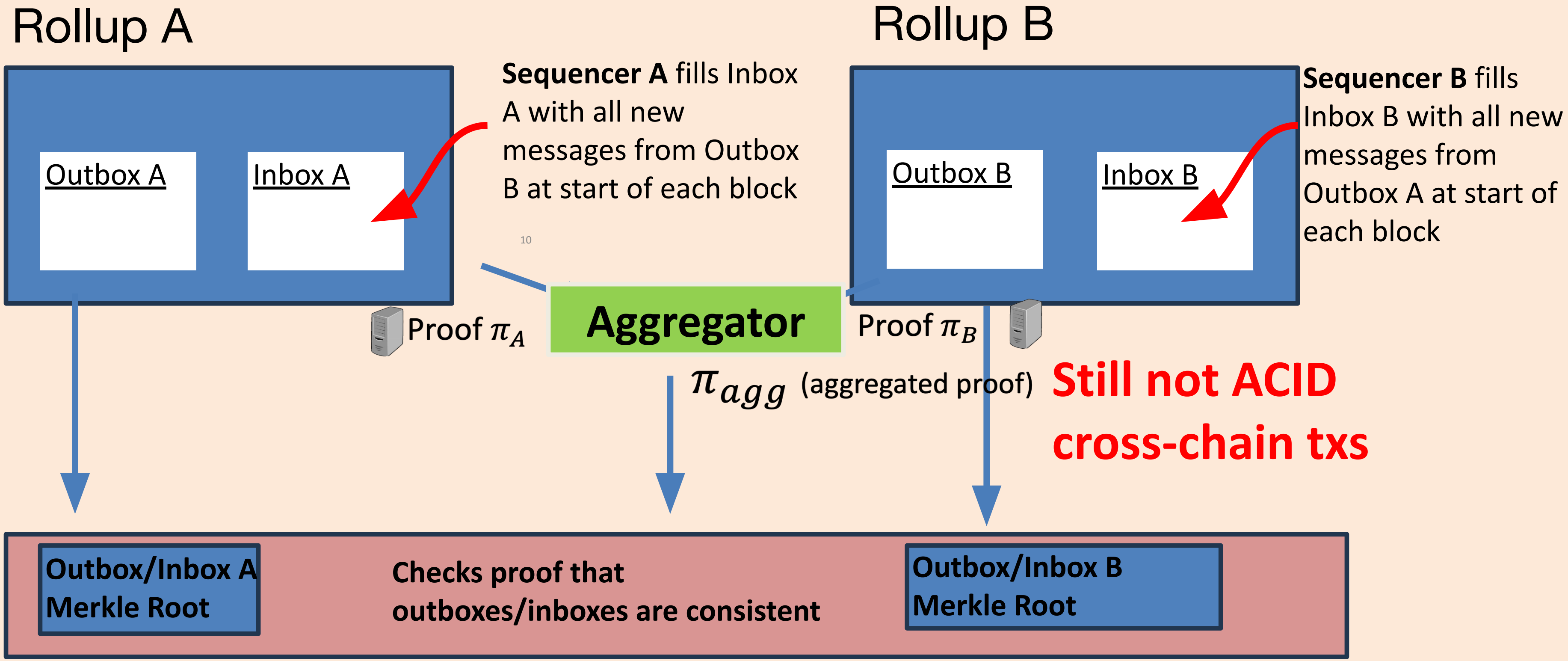
Outbox/Inbox A Merkle Root

Checks that outboxes/inboxes are consistent

Outbox/Inbox B Merkle Root

L1 contracts synchronized to reject state updates if consistency check fails

Message-passing with L1 consistency check



Rollup A

Outbox A

Inbox A

Sequencer A fills Inbox A with all new messages from Outbox B at start of each block

Rollup B

Outbox B

Inbox B

Sequencer B fills Inbox B with all new messages from Outbox A at start of each block

Proof π_A

Aggregator

Proof π_B

π_{agg} (aggregated proof)

Still not ACID cross-chain txs

Outbox/Inbox A Merkle Root

Checks proof that outboxes/inboxes are consistent

Outbox/Inbox B Merkle Root

L1 contracts synchronized to reject state updates if consistency check fails

Why are ACID cross-chain txs hard?

- Need to emulate multiple rounds of communication between chains in a single block

- Inherent dependency on *global ordering* across all chains of individual-chain transactions

- Naïve solution is to merge all chains (rollups) into one □ want to do better, preserve horizontal scaling, VM diversity

Through the lens of collaborative SNARKs

There's an implicit unified VM in which cross-chain transactions take place

Goal: create a single ZK proof of state-transition for a distributed system of VMs with inter-process communication where:

(1) There is one prover per VM
(2) The total work done per prover remains (near) constant as the number of interacting VMs grows

Through the lens of collaborative SNARKs

Goal: create a single ZK proof of state-transition for a distributed system of VMs with inter-process communication where:

(1) There is one prover per VM

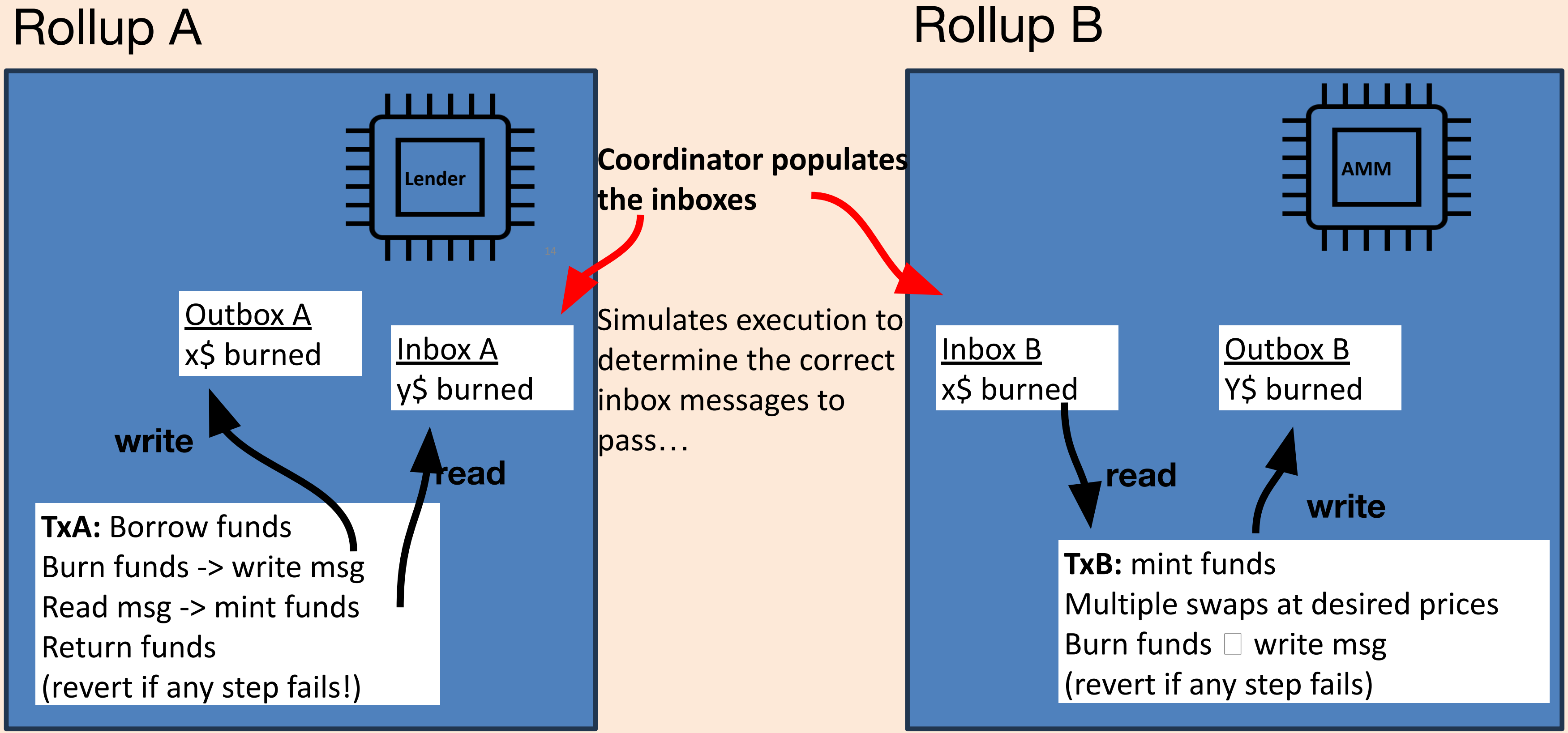
(2) The total work done per prover remains (near) constant as the number of interacting VMs grows

Each prover starts with a piece of the global witness (e.g., defined by transactions and cross-chain messages to its VM)

Generic solutions:

- Use SNARK recursion
- Distributed SNARK protocols: distributed sumcheck + FFTs

A simpler extension of async mailbox design



Rollup A

Lender

Outbox A
x$ burned

Inbox A
y$ burned

write

read

TxA: Borrow funds
Burn funds -> write msg
Read msg -> mint funds
Return funds
(revert if any step fails!)

Coordinator populates the inboxes

Simulates execution to determine the correct inbox messages to pass…

Rollup B

AMM

Inbox B
x$ burned

Outbox B
Y$ burned

read

write

TxB: mint funds
Multiple swaps at desired prices
Burn funds □ write msg
(revert if any step fails)

CIRC: Coordinated Inter-Rollup Communication

- Not fully general, but covers most practical use cases of synchronous composability among independent chains (e.g., flash loans, asset swaps, limit orders, etc)

- No VM modifications to the chains necessary

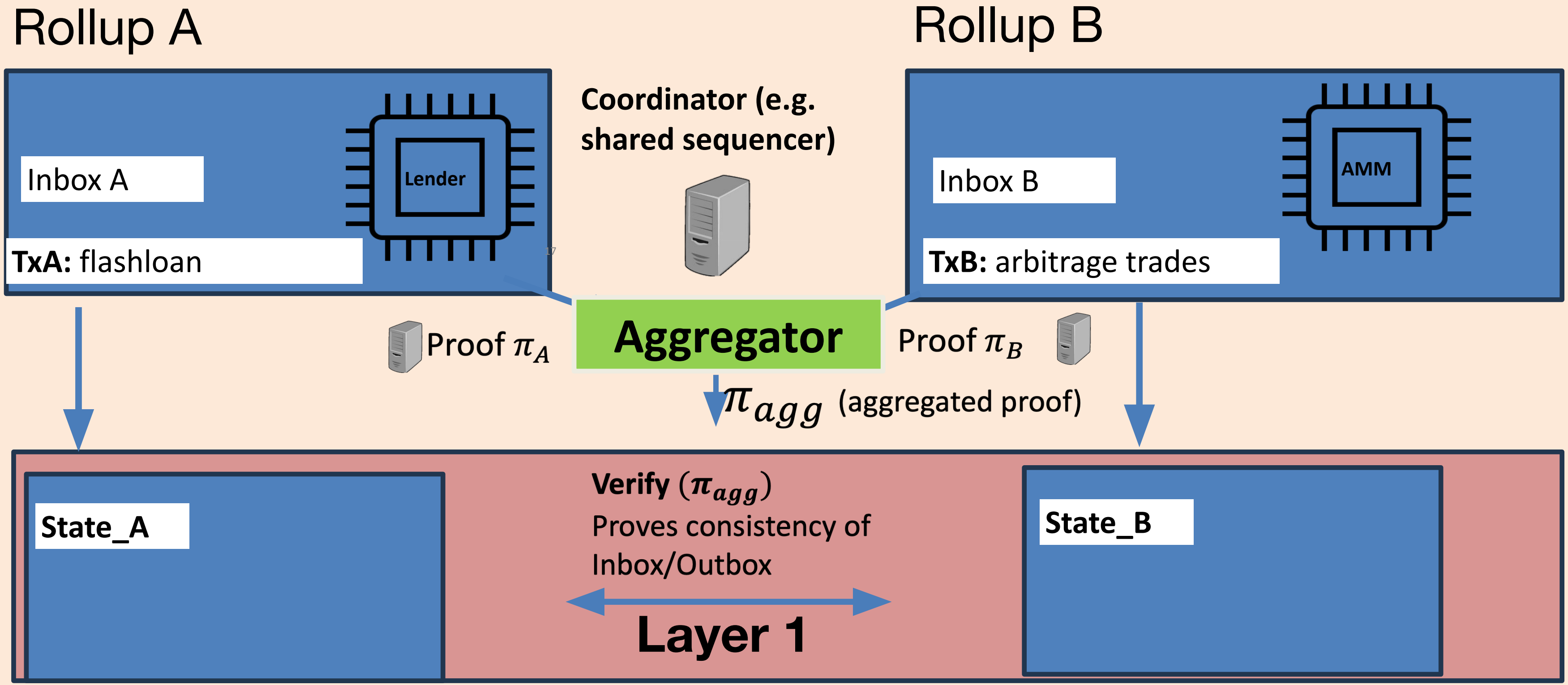
- Parallel proving (each L2 prover independently proves a single chain’s state)
    - An aggregator creates a simple aggregated proof, whose complexity is independent of the complexity of each chain

CIRC: Coordinated Inter-Rollup Communication

- **Mailboxes**:
  - implemented as contracts on each rollup
  - messages stored in authenticated key-value map (avoids need to enforce message ordering)
  - any contract can write to outbox or read from inbox, but *msg.sender* is part of the message key
  - sequencer populates inboxes at start of each block

- **Settlement layer contract:**
  - Verifies correctness of state updates for each rollup and mailbox consistency (e.g., receives aggregated proof)

CIRC: Coordinated Inter-Rollup Communication



Rollup A

Inbox A

Lender

TxA: flashloan

Coordinator (e.g. shared sequencer)

Rollup B

Inbox B

AMM

TxB: arbitrage trades

Proof π_A

Aggregator

Proof π_B

π_{agg} (aggregated proof)

State_A

Verify (π_{agg})

Proves consistency of Inbox/Outbox

Layer 1

State_B

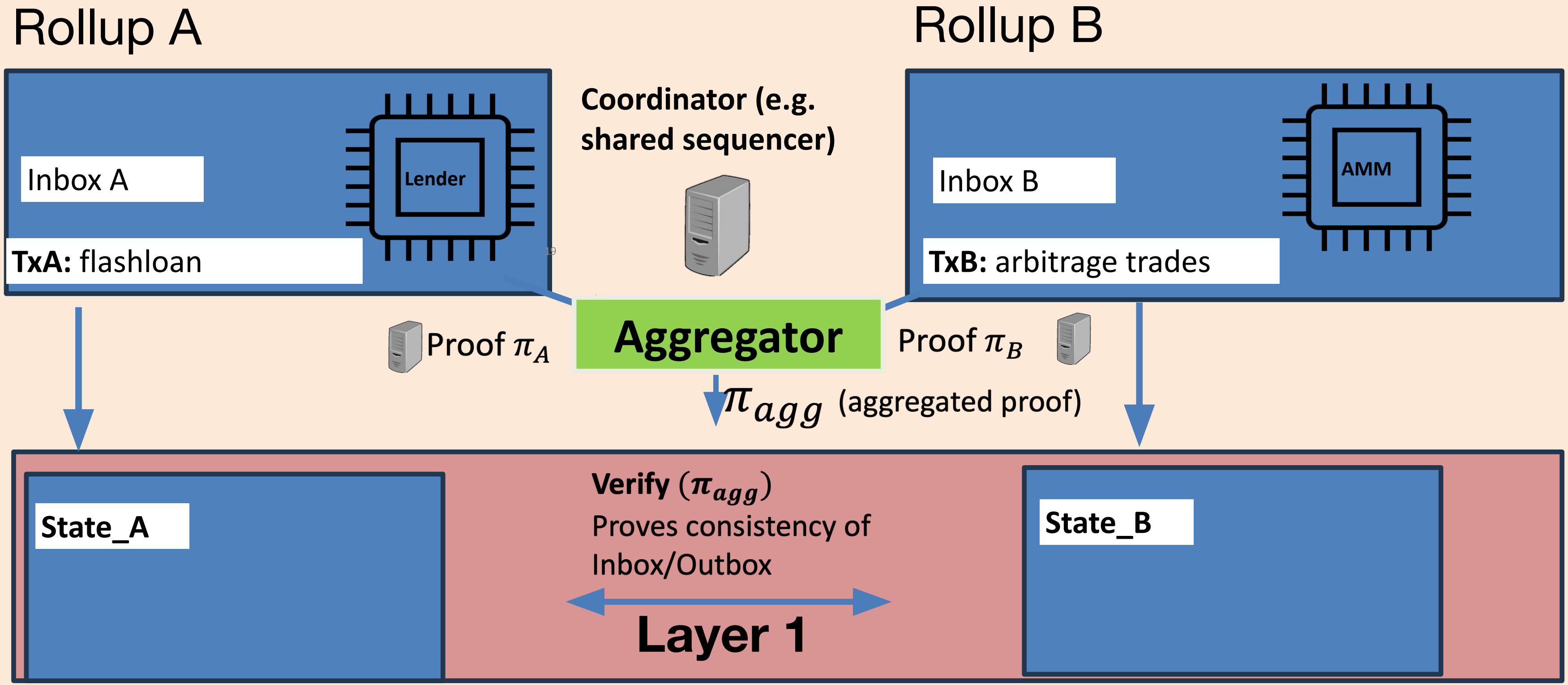
Invariant: rollup full nodes always know the state

- Once sequencer posts transaction ordering, full nodes can immediately calculate rollup state

- Full nodes don't have to wait for L1 settlement, waiting for L1 finality of sequencer published blocks suffices

- If sequencer is trusted not to equivocate then full nodes can confirm state immediately (within seconds or less)

Problem 1: full nodes don’t know state pre-settlement



Rollup A

Inbox A

Lender

TxA: flashloan

Coordinator (e.g. shared sequencer)

Rollup B

Inbox B

AMM

TxB: arbitrage trades

Proof π_A

Aggregator

Proof π_B

π_{agg} (aggregated proof)

State_A

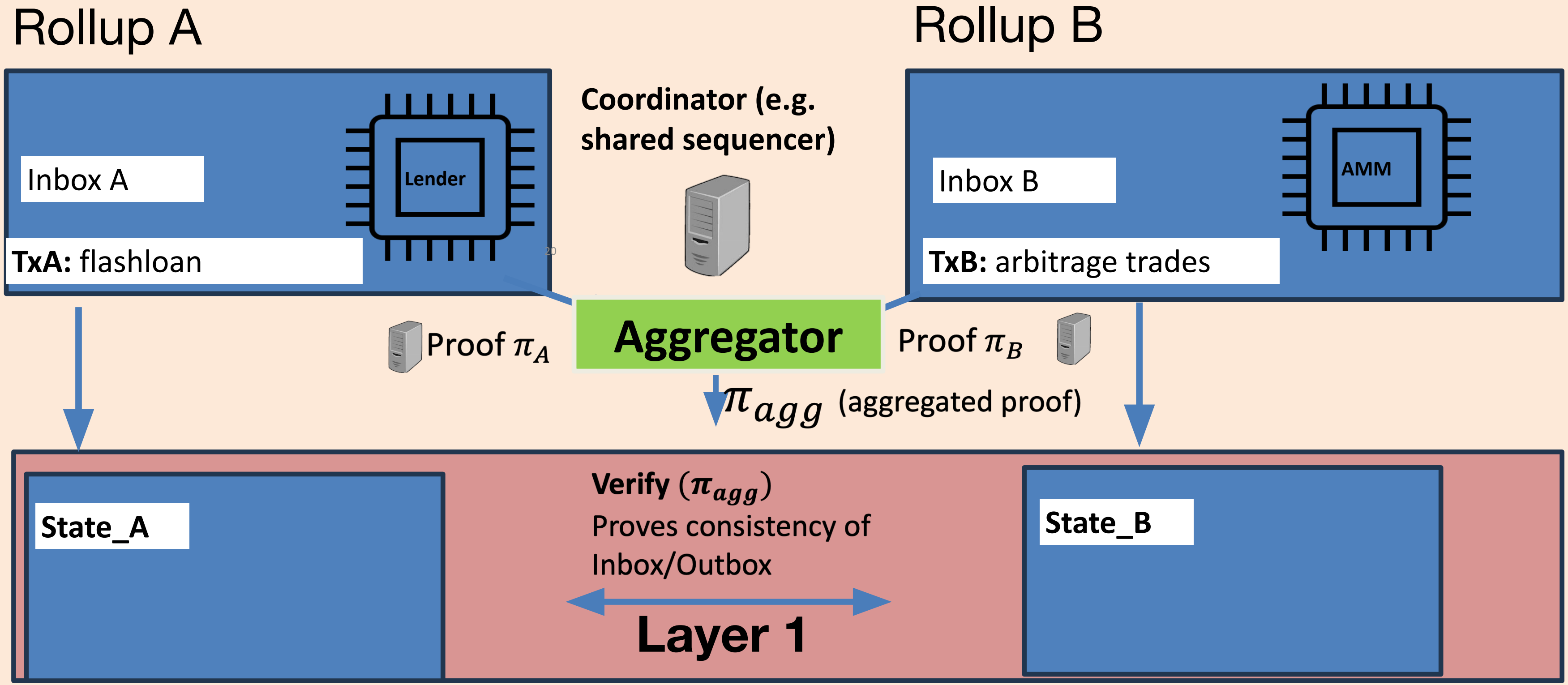
Verify $(\boldsymbol{\pi_{agg}})$

Proves consistency of Inbox/Outbox

Layer 1

State_B

Problem 2: single point of failure for fast confirmations



Rollup A

Inbox A

Lender

TxA: flashloan

Coordinator (e.g. shared sequencer)

Rollup B

Inbox B

AMM

TxB: arbitrage trades

Proof π_A

Aggregator

Proof π_B

π_{agg} (aggregated proof)

State_A

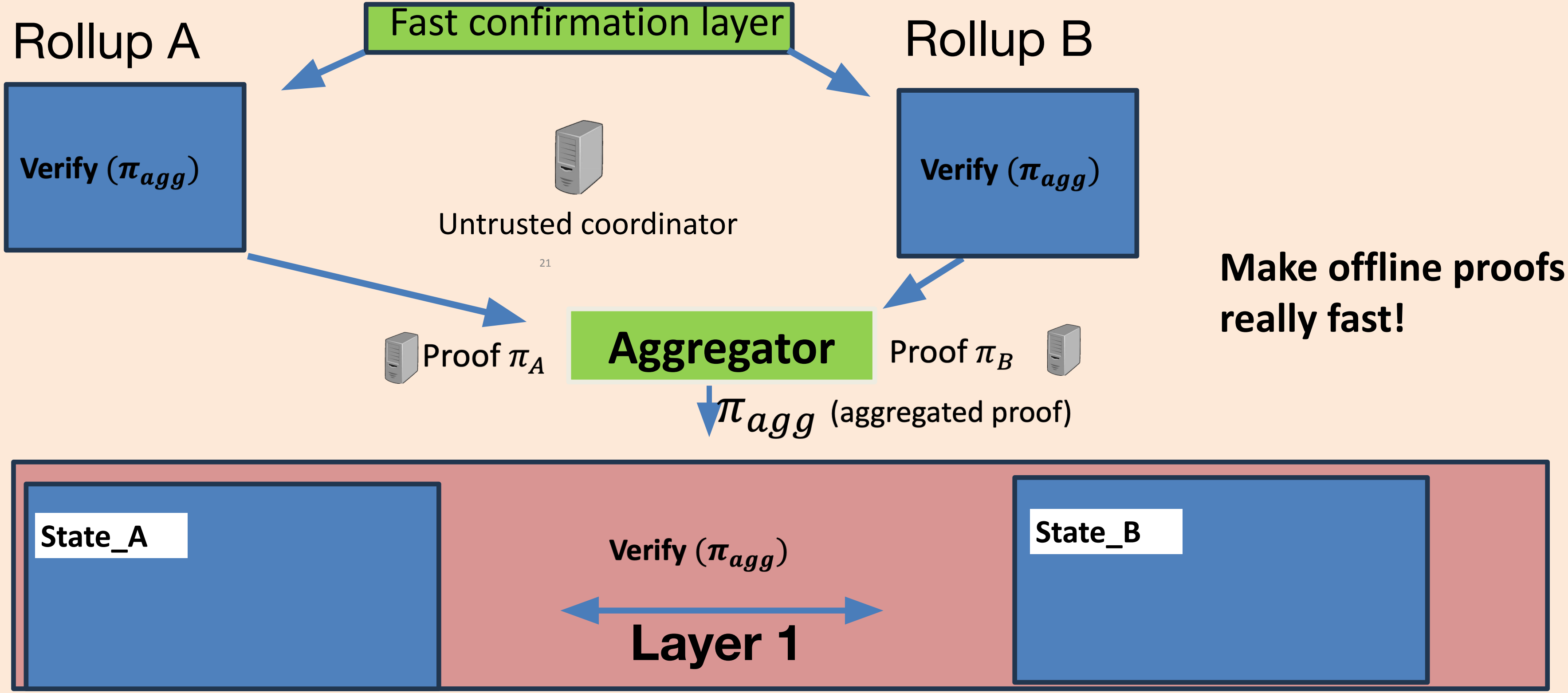
Verify (π_{agg})

Proves consistency of Inbox/Outbox

Layer 1

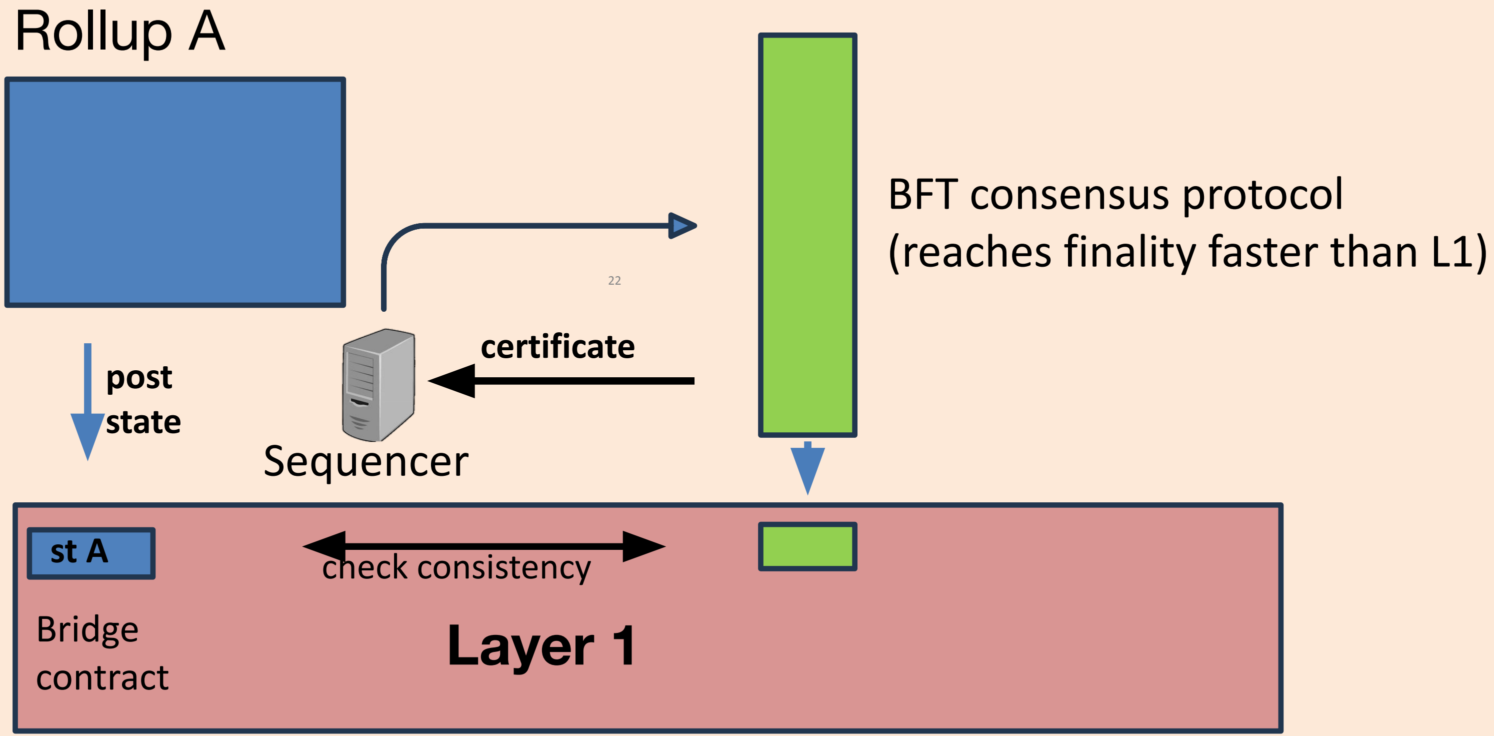
State_B

Solution: fast zk proofs + fast confirmation layer



Make offline proofs really fast!

What is a fast confirmation layer?



Rollup A

post state

Sequencer

certificate

BFT consensus protocol
(reaches finality faster than L1)

st A

check consistency

Bridge contract

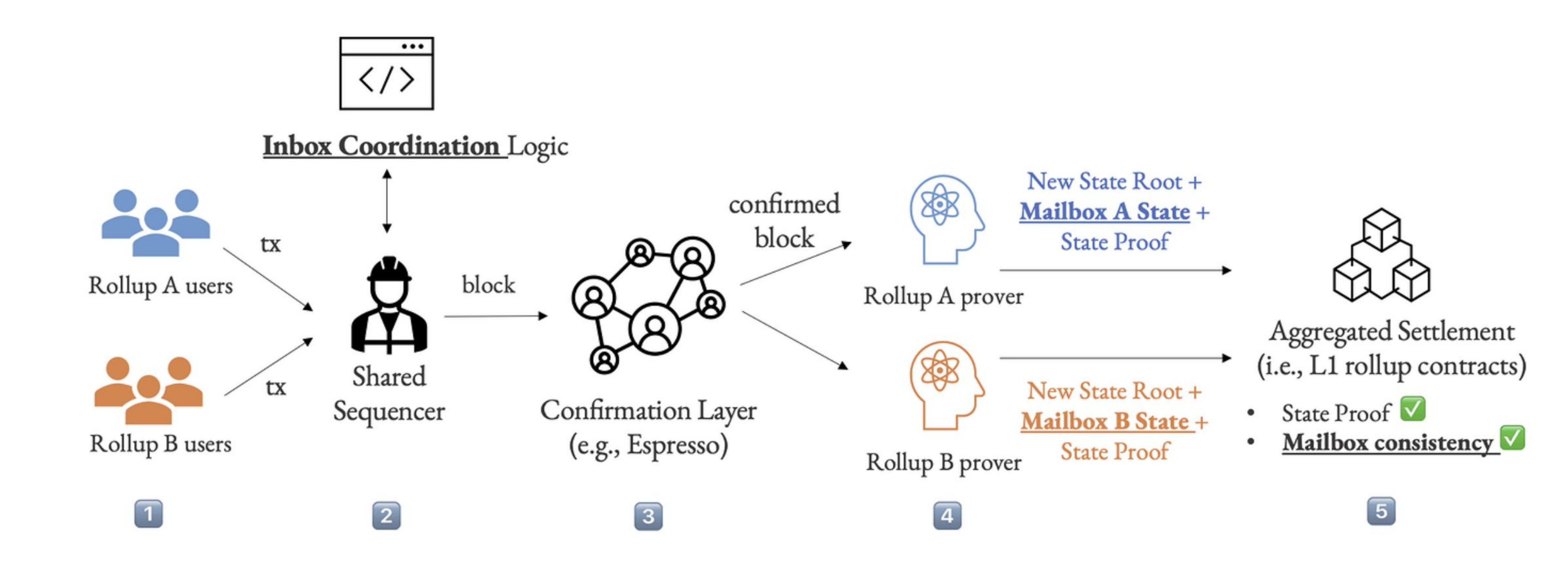
Layer 1

Fast “off-chain” ZK proofs

Goal: optimize end-to-end latency of proving, proof aggregation, and verification, given constraints on bandwidth/compute power of each full node.

- Different design constraints than ZK proofs optimized for L1 settlement
- L1 node bandwidth/compute requirements are very low to maximize diverse participation— should the same apply to L2 nodes?
- SNARKs w/ larger communication, but faster proofs: **Orion, Brakedown, Basefold, Blaze** also **NARK Accumulation**

CIRC: Coordinated Inter-Rollup Communication


Inbox Coordination Logic
Rollup A users
tx
Rollup B users
tx
Shared Sequencer
block
Confirmation Layer (e.g., Espresso)
confirmed block
Rollup A prover
New State Root + Mailbox A State + State Proof
Rollup B prover
New State Root + Mailbox B State + State Proof
Aggregated Settlement (i.e., L1 rollup contracts)
• State Proof ✅
• Mailbox consistency ✅
1
2
3
4
5

The time for cross-chain composability is *now*

Chains can’t specialize…

be the best chain for NFTs, gaming,
or the best DEX chain

... if they each need to replicate the apps of all other chains

Thank You!

Further reading:
https://espresso.discourse.group/t/circ-coordinated-inter-rollup-communication/

@EspressoSys
@benafisch