Scaling Blockchain with Off-chain Approach

COS 418: *Distributed Systems*
Lecture 18 optional

Zhenyu Song

Outline

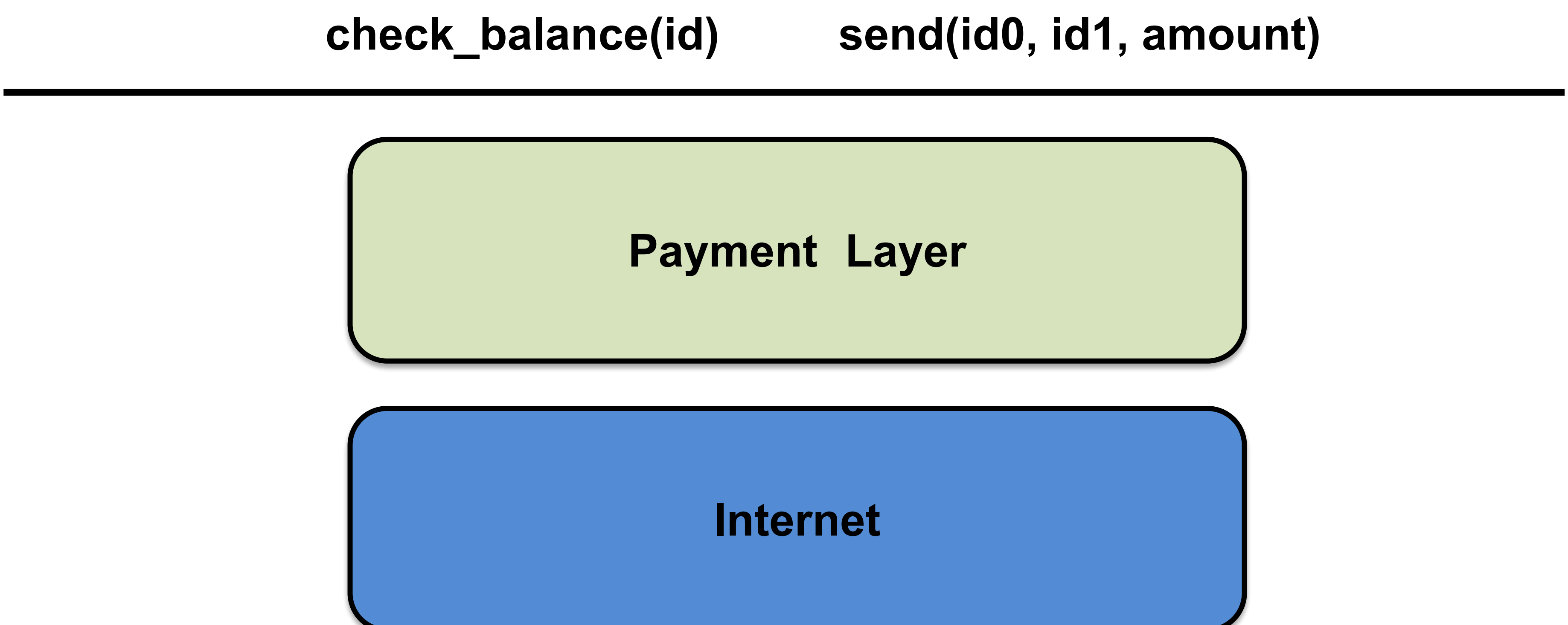
- Short introduction to Bitcoin
- Scaling limitation and payment channel
- Payment network
- Smart contract and state channel

Why Bitcoin? All about Trust

- Problem with current payment system
  - Reversible: bank can reverse your payment
    - The whole system is built on the trust of third-party, e.g., trust the bank not reversing your transaction
    - Introduce additional cost
  - From a systems perspective, it's better to build a non-reversible payment system first
    - Can build reversible system on top of it
  - Big goal: code is law

Distributed Payment Layer

- A stateful layer: support state transition with constraints
  - For payment layer: the total sum of balance is unchanged



Design Intuition

- Replicate payment history on each node
- Nodes run consensus protocol to make the history identical

Intro to Cryptography Signature

Public-Key Cryptography

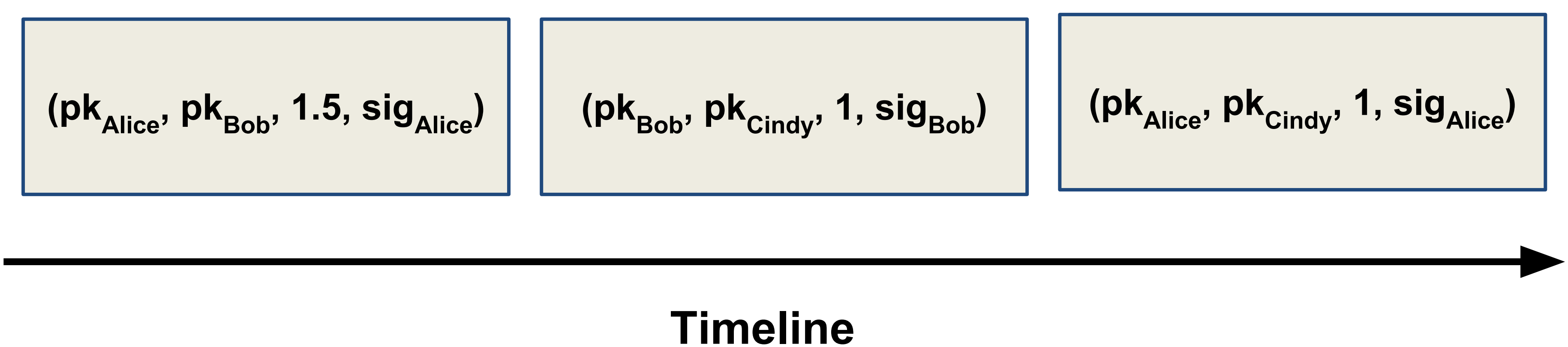
- **Each party has (public key, secret key)**

- **Alice's secret key: sk**
  - Known only by Alice
  - Alice uses sk to generate new signatures on messages

- **Alice's public key: pk**
  - Known by anyone
  - Bob uses pk to verify signatures *from* Alice

Primitive: Payment Transaction

- Each tx can be viewed as (pk_{src}, pk_{dst}, amount, sig_{src})
  - We use public keys as identifiers
  - Signature is to prove the owner made the transaction
- Bitcoin is an append-only log of transactions
  - How do we make it append-only?



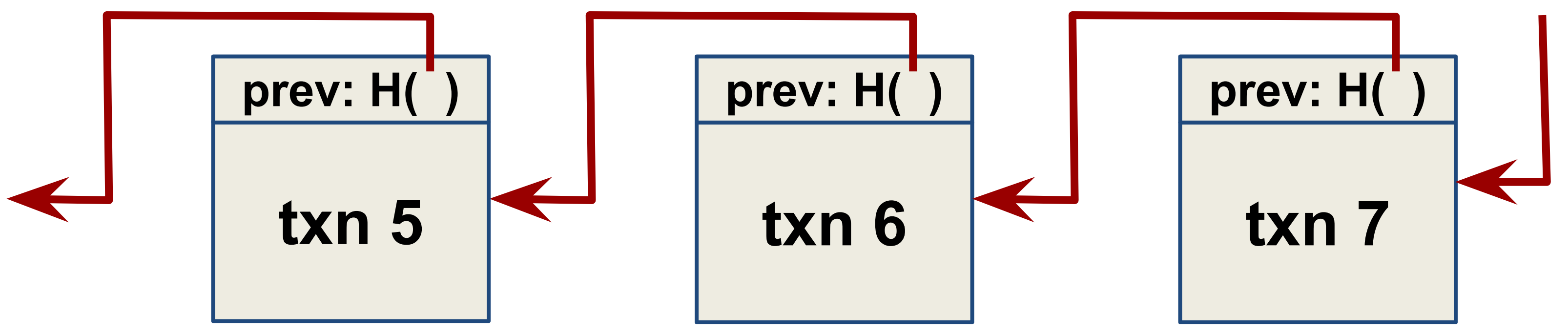
Intro to Cryptography Hash

Cryptography Hash Functions

- Take message *m* of arbitrary length and produces fixed-size (short) number *H(m)*

- One-way function
  - Efficient: Easy to compute H(m)
  - Hiding property: Hard to find an m, given H(m)
  - Collisions exist, but hard to find
    - For SHA-1, finding any collision requires 2^{80} tries. Finding a specific collision requires 2^{160} tries.

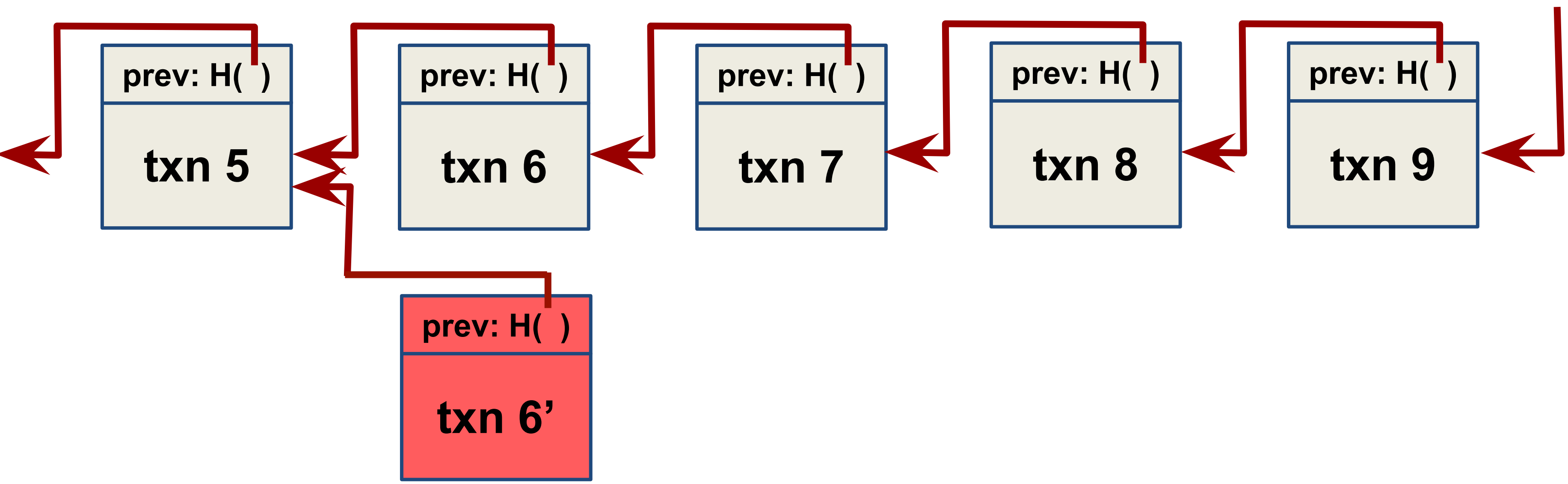
Blockchain: Append-only Hash Chain



- To prevent entities modifying transactions already committed, each block contains the hash of previous block

- This gives a sequential order
  - Given a block, all blocks before it are fixed

Resolve Forking: Proof of Work



- Generating a new block requires computation
  - Cooperative nodes always accept longest chain
- Creating fork requires rate of malicious work >> rate of correct work
  - So, the older the block, the safer it is from being deleted

Bitcoin Proof of Work

Find **nonce** such that

hash (**nonce** || prev_hash || block data) < target

i.e., hash has certain number of leading 0's

What about changes in total system hashing rate?

- Target is recalculated every 2 weeks
- Goal: one new block every 10 minutes

Outline

- Short introduction to Bitcoin
- Scaling limitation and payment channel
- Payment network
- Smart contract and state channel

Limitation of Scaling

- Throughput limitation for Bitcoin
  - 1 block ~ 10 min
    - Can increase the throughput by batching transactions, i.e., increase block size
    - Currently 1 block = 1 MB max, ~ 2000 txns
      - 3-4 txns / sec
    - Visa payment system: typically 2,000 txns / sec
  - Can we scale infinitely by batching more txs?

Limitation of Scaling

- Scaling by batching?
  - Short answer: not infinitely
  - The fundamental limitation on sequential consistency
  - Blocks are designed to be in sequential order
    - Each block propagates to rest nodes all over the world before another block is generated
  - And also another problem: latency
    - If we view the system as a computer, the frequency is bounded by light speed across the earth
  - Conclusion: the throughput and latency in the system are fundamentally bounded by global network

Does People Give up in Scaling?

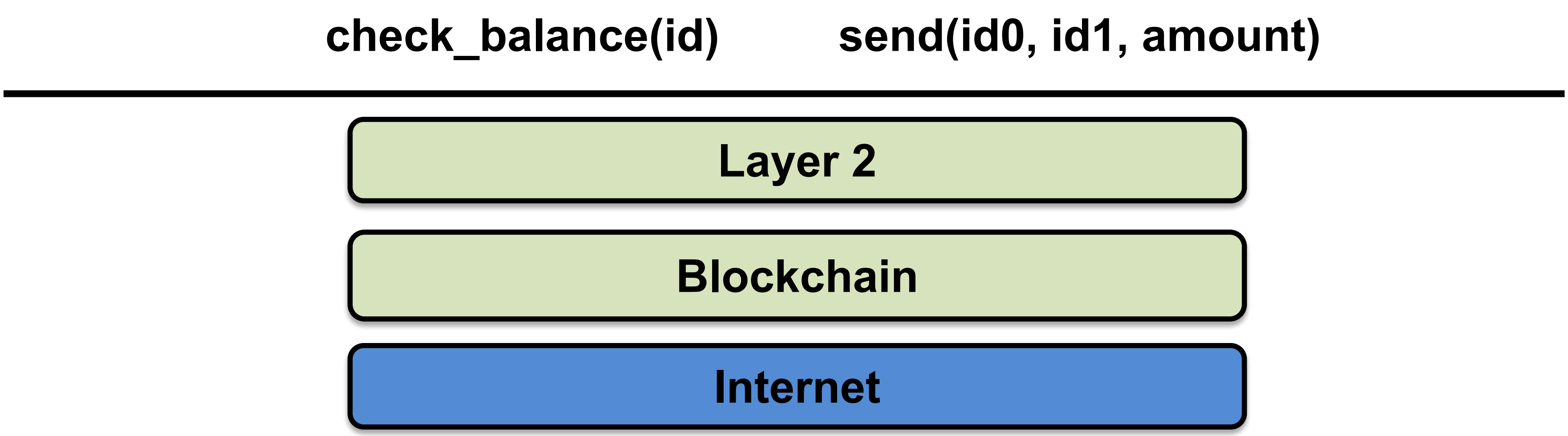
- Two classes:
  - Change blockchain design
    - Sharding
    - DAG
  - Build another layer on top of blockchain: layer 2
    - State channel
    - Side chain

Why Layer 2? Throughput/latency!

- The throughput/latency is fundamentally bounded by sequential consistency
  - We pay by ordering all transactions
- But do we need to order all transactions?
  - No. If order is not necessary, we don't need to.
  - Therefore we can leverage it to scale

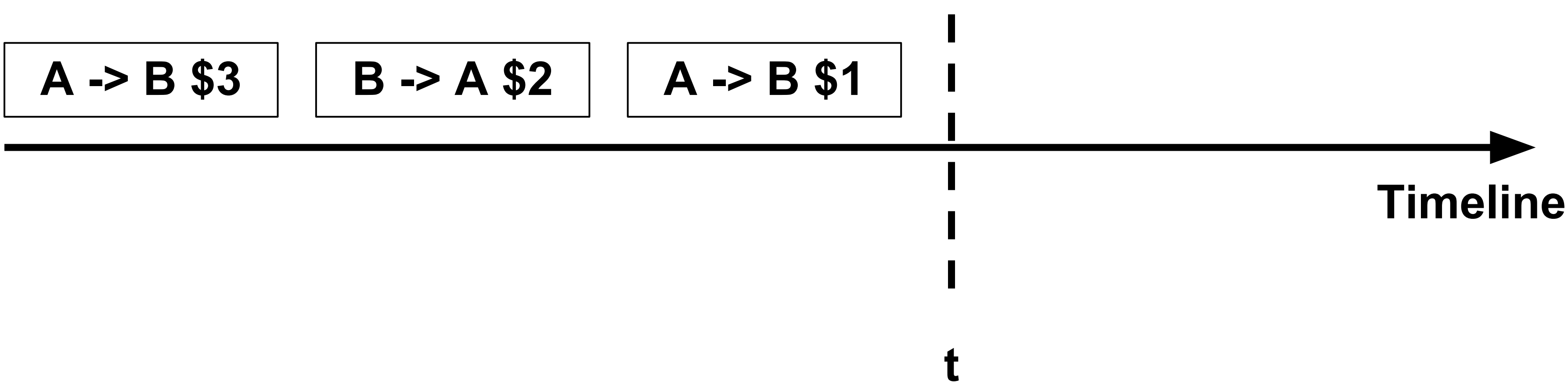
Payment Layer with Layer 2

- Layer 2 offloads most of transactions
  - Blockchain layer doesn’t see all transactions
  - This is why called off-chain
- Intuition: payment transactions are special
  - Payment transactions can be merged
  - We don’t keep the order, even don’t keep origin txs



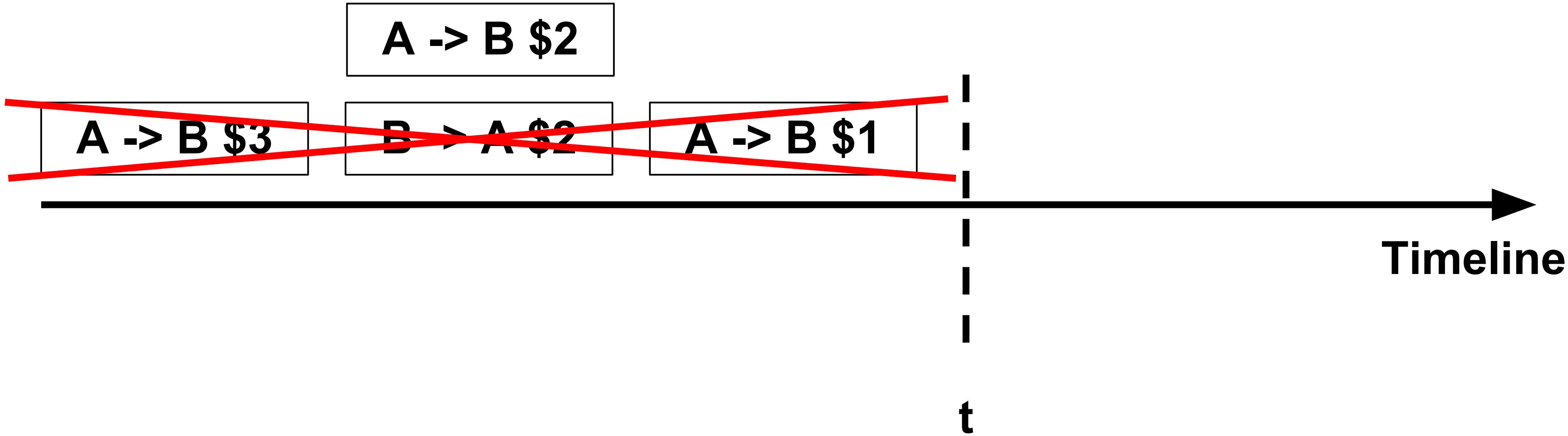
Payment Channel Example

- Suppose we are at time t.
  - A, B already had several payments
  - Does the system need to order the first 3 txs at t?



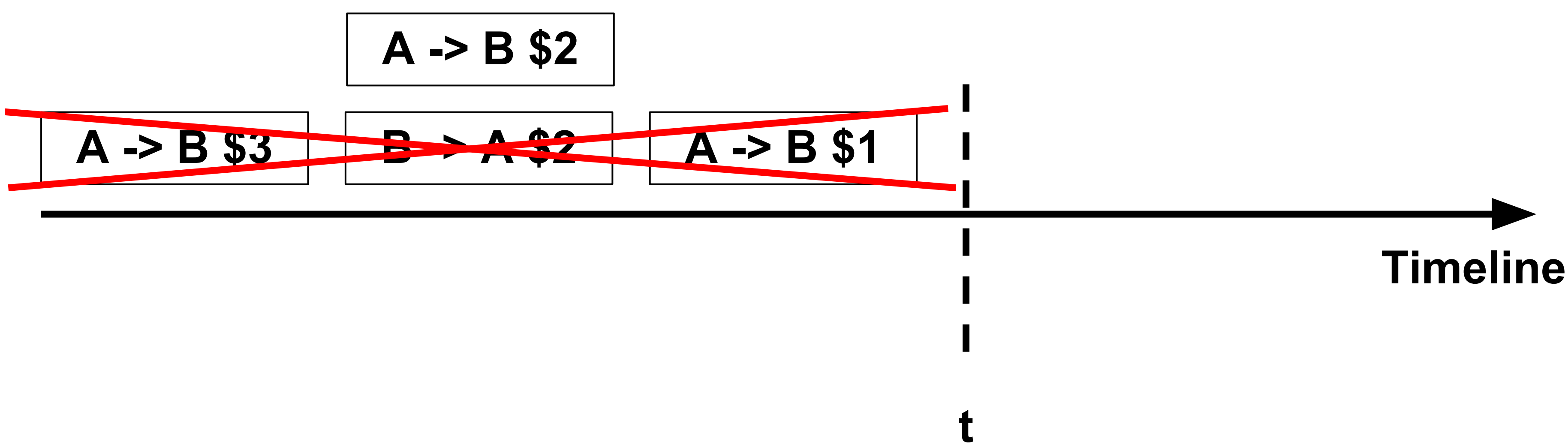
Payment Channel Example

- No, we can replace the first 3 txs with merged tx (A -> B $2)
  - But remember Blockchain is append-only
  - How
    - Users delay to put transactions on to blockchain
    - Users only put merged tx



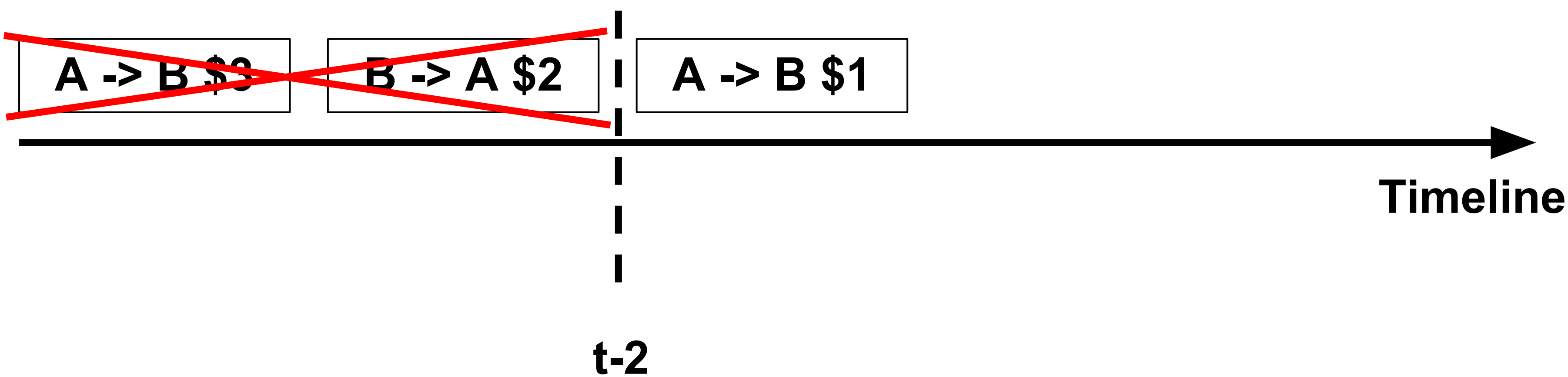
Payment Channel Example

- This is an important intuition of layer 2
  - Blockchain acts as a court system. Users commit to blockchain only when their interaction settle down or there is a disagreement.
- What are the challenges?



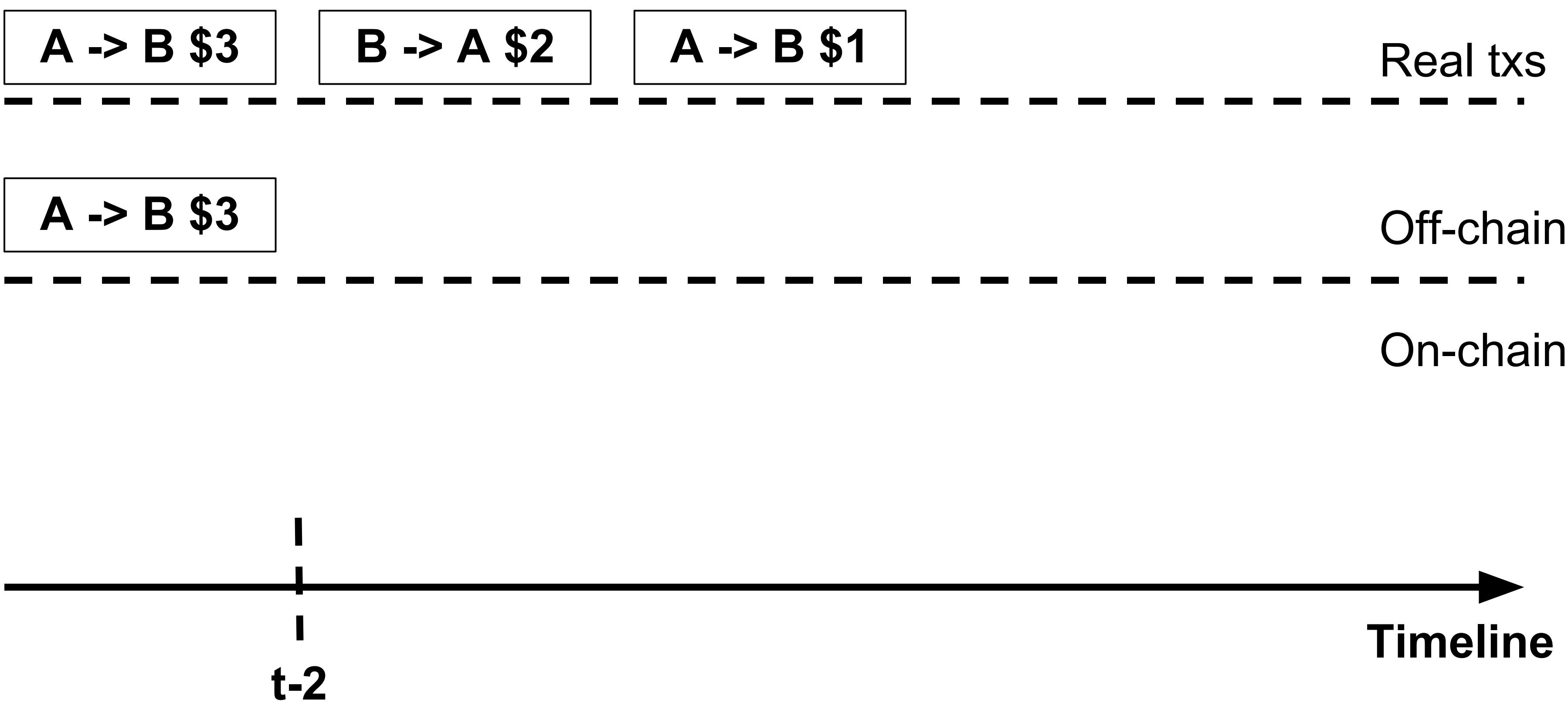
Payment Channel Example

- Support we are at time t-2.
  - A paid B $3, and got an apple
  - B paid A $2, and got an orange
  - If there is no record on-chain, how can B prove to others that A owes it $1?



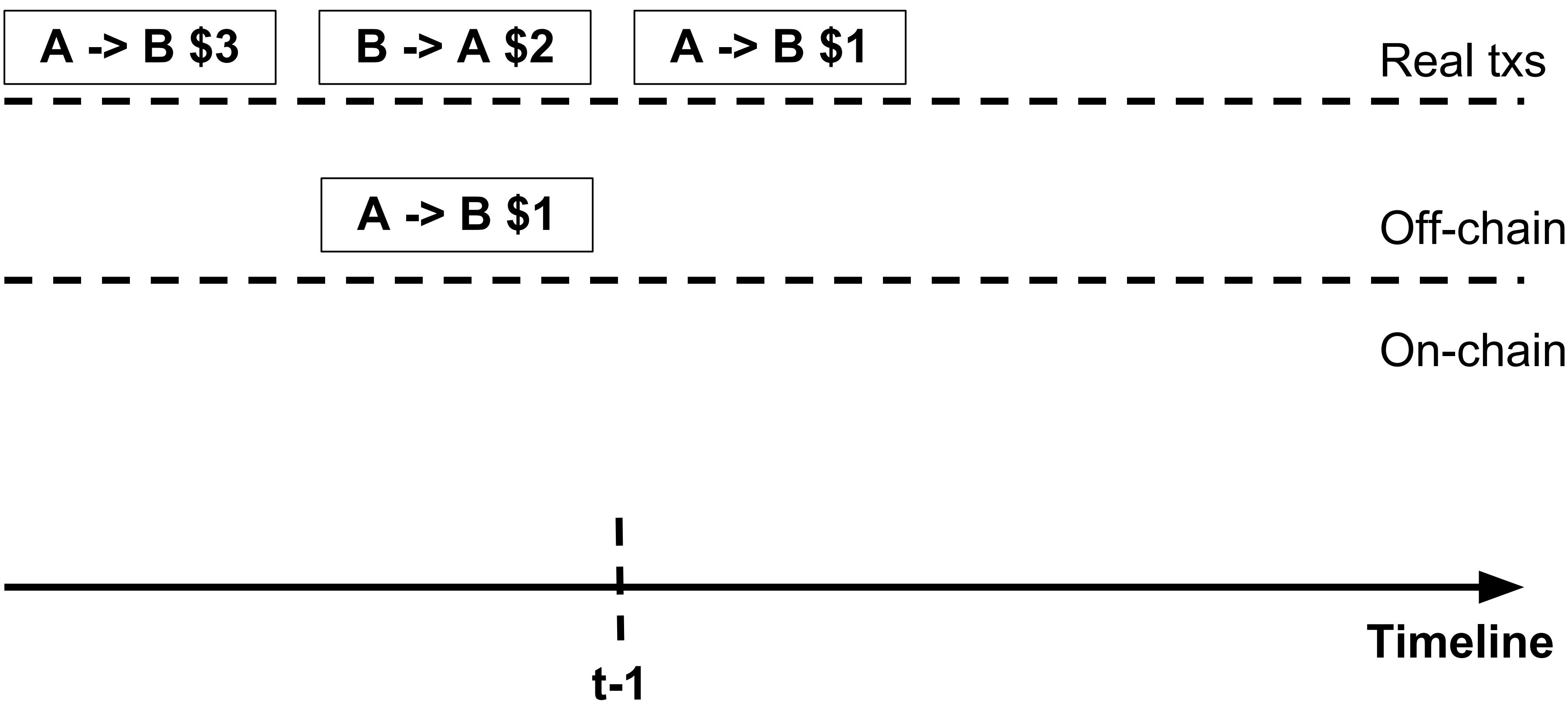
Payment Channel Example

- We create a new merged transaction after each interaction. And submit it on-chain when finalizing the result



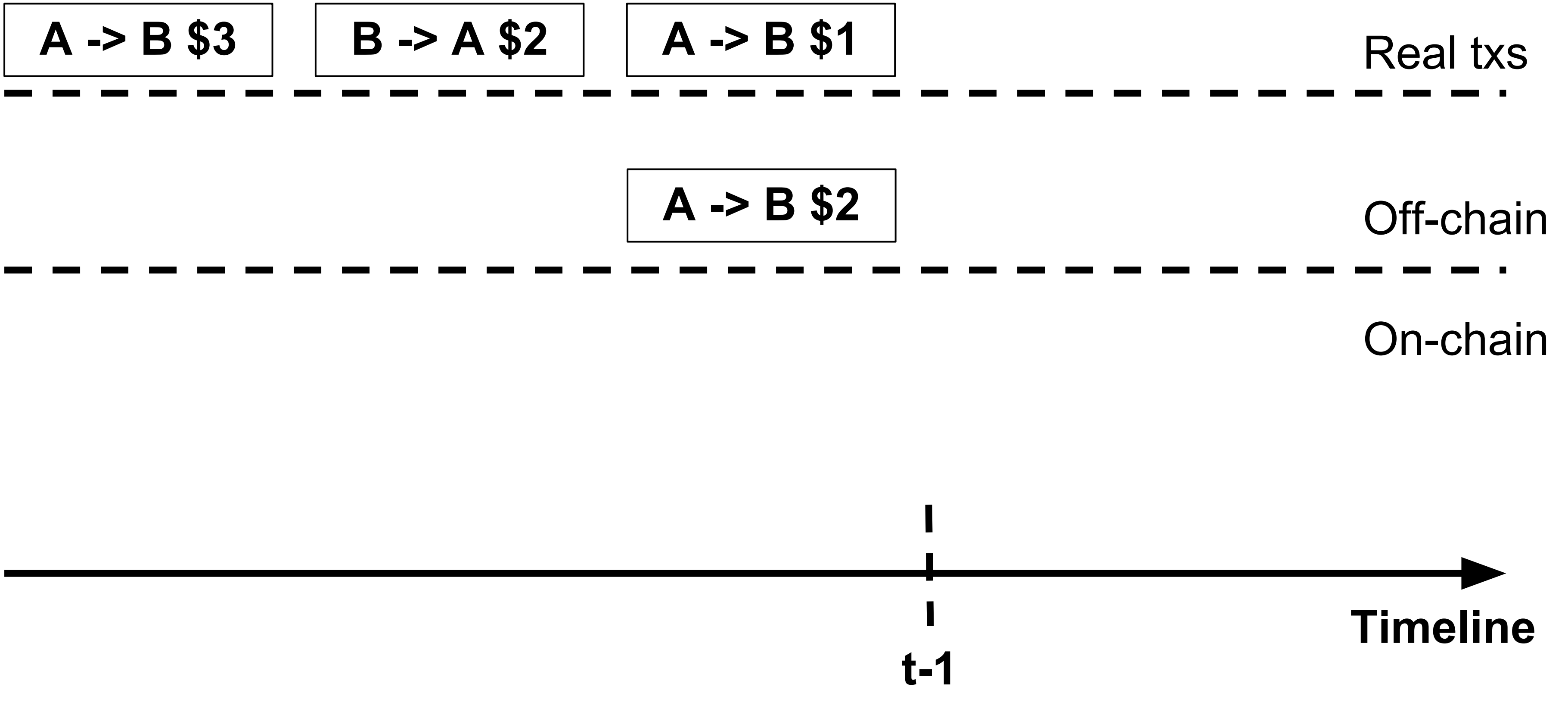
Payment Channel Example

- We create a new merged transaction after each interaction. And submit it on-chain when finalizing the result



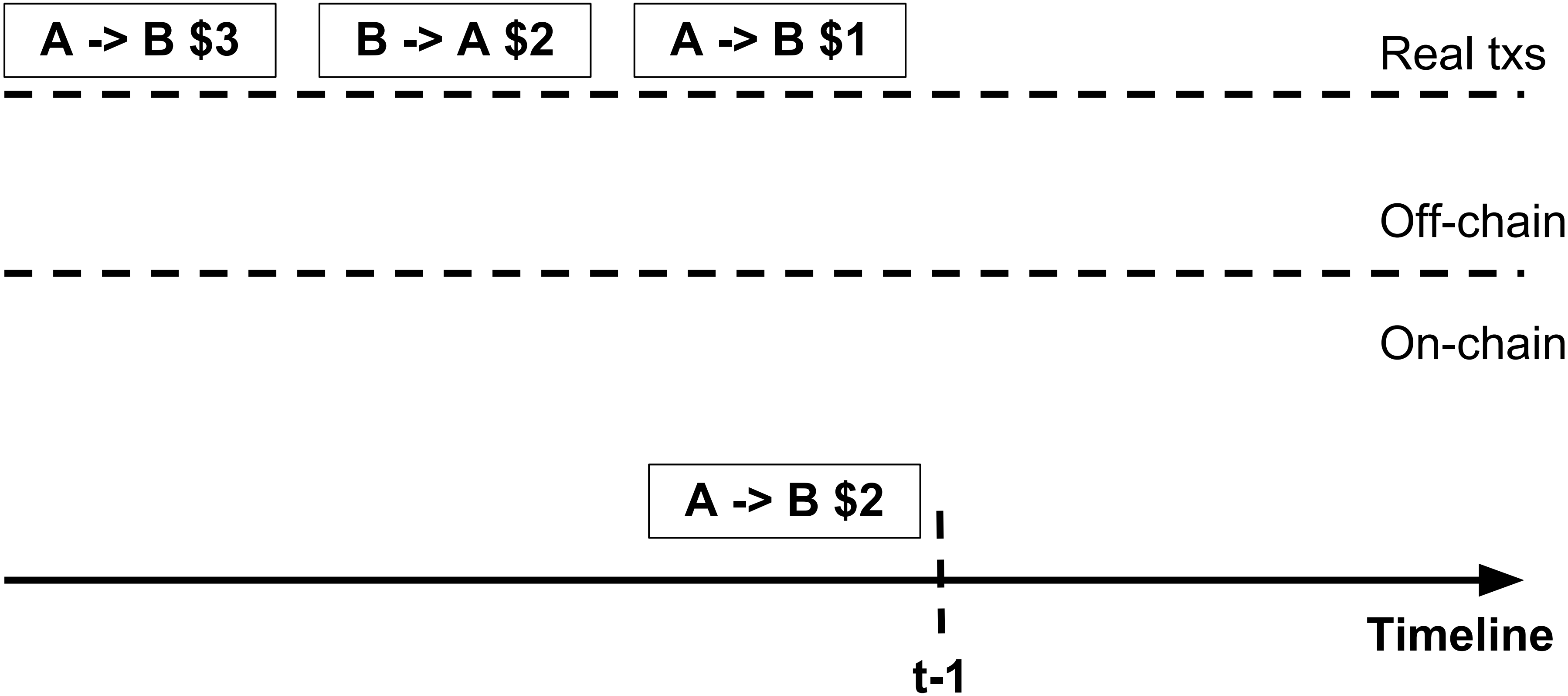
Payment Channel Example

- We create a new merged transaction after each interaction. And submit it on-chain when finalizing the result



Payment Channel Example

- We create a new merged transaction after each interaction. And submit it on-chain when finalizing the result



Additional Payment Channel Problems

- Still we have problems
  - What if one malicious party withdraw all its balance to other account before off-chain transaction commit?
  - How do we valid the merged transaction?
  - What if malicious party submit old merged tx instead of new one?

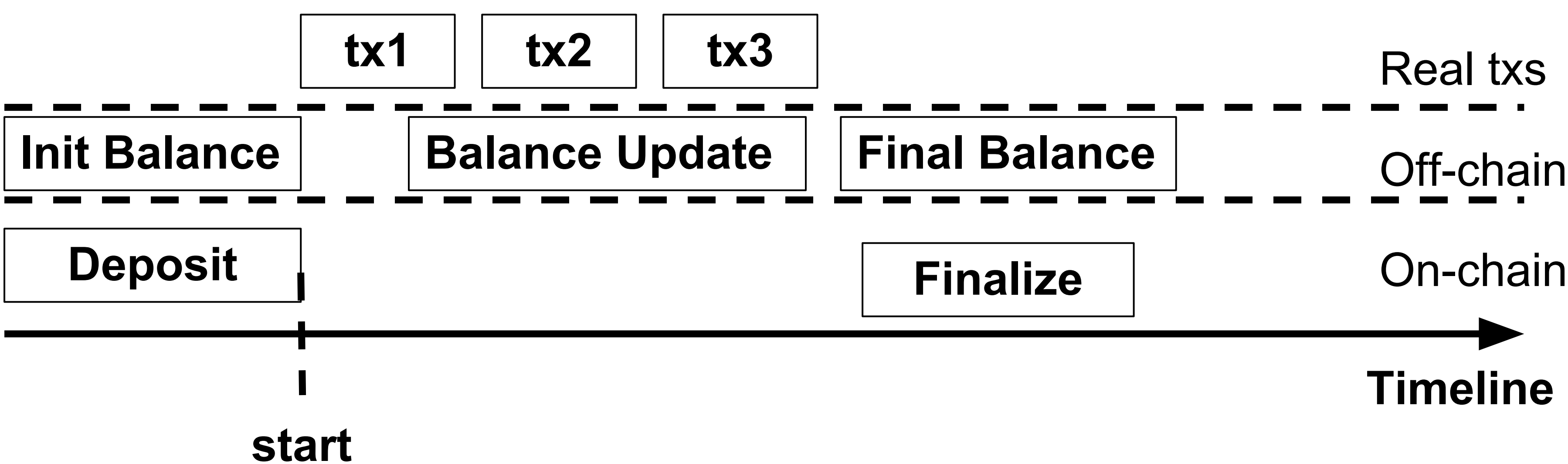
Intro to Multi-signature

Multi-signature

- We can sign a message with multiple secret keys
- And we can verify message with multi-signature using multiple public keys
  - Example
    - Sign message with sk_{Alice}, sk_{Bob}
    - Verify message with pk_{Alice}, pk_{Bob}

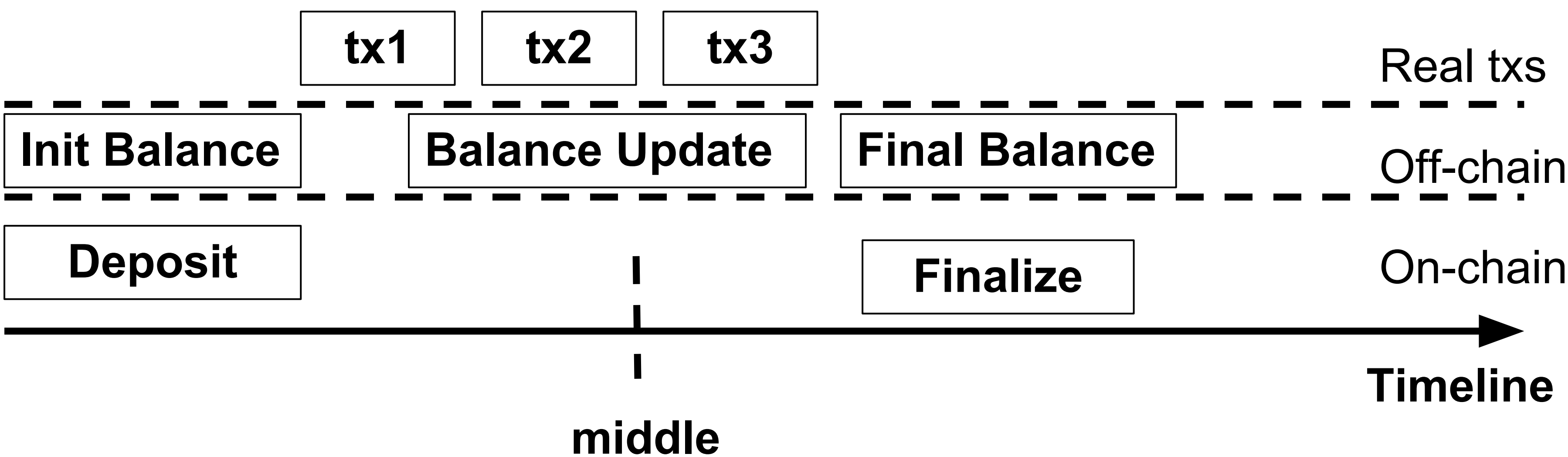
Payment Channel Formal Design

- At start, A and B deposit balance to an account AB controlled by A and B jointly
  - Any txs sending by AB need multi-signature of A and B
- At the same time, A and B sign the init balance with multi-signature
  - Init balance is actually a tx sending deposit money back from AB



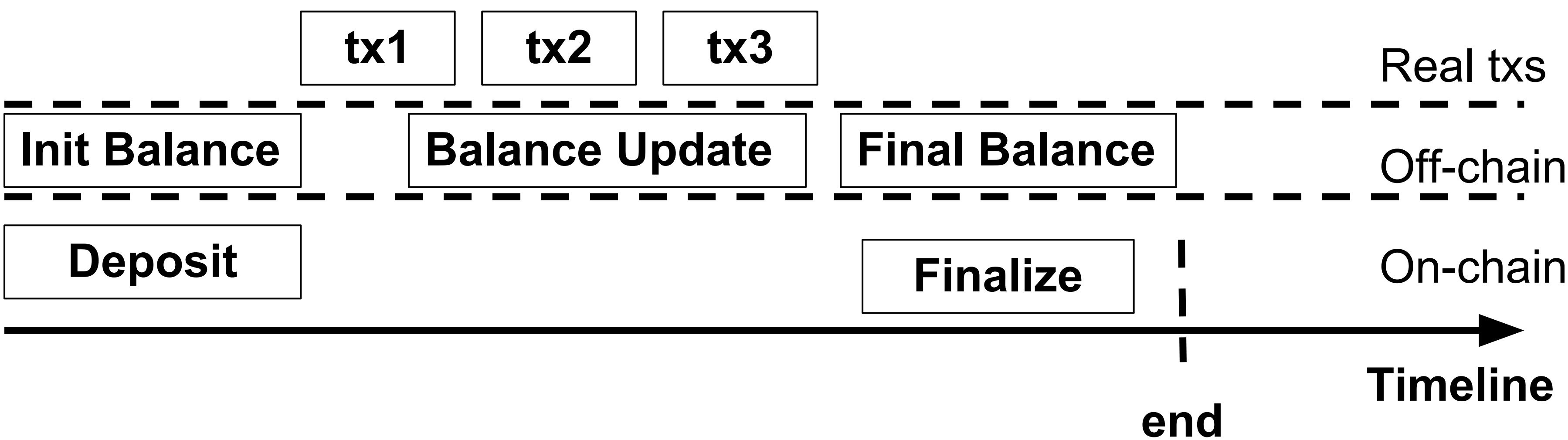
Payment Channel Formal Design

- After each transaction, A and B sign a new balance with multi-signature
  - That is also a transaction sending money from AB back to A and B according to balance update



Payment Channel Formal Design

- At the end, A and B sign the final balance with multi-signature
  - That is also a tx sending money back to A and B
- After that, one of then submit this transaction to Blockchain



Solution to Problems

- What if one malicious party withdraw all its balance to other account before off-chain transaction commit?
  - Initial deposit to a multi-signature controlled account
- How do we valid the merged transaction?
  - Multi-signature
- What if malicious party submit old merged tx instead of new one?
  - Merged tx has a nonce (sequence number).
  - The nonce is increasing every tx.
  - Final tx has highest nonce.
  - When merged tx is submitted, there is a disputing period. In that period, any user can submit a newer merged tx.

Payment Channel Cont'

- Why this called payment channel?
  - We can view it as a stateful link between two parties
  - The state is the current balance
- This is difficult to implement in Bitcoin, but not hard in Ethereum
  - Ethereum is Turing-complete. You can write program on it

Outline

- Short introduction to Bitcoin
- Scaling limitation and payment channel
- Payment network
- Smart contract and state channel

Why Payment Network?

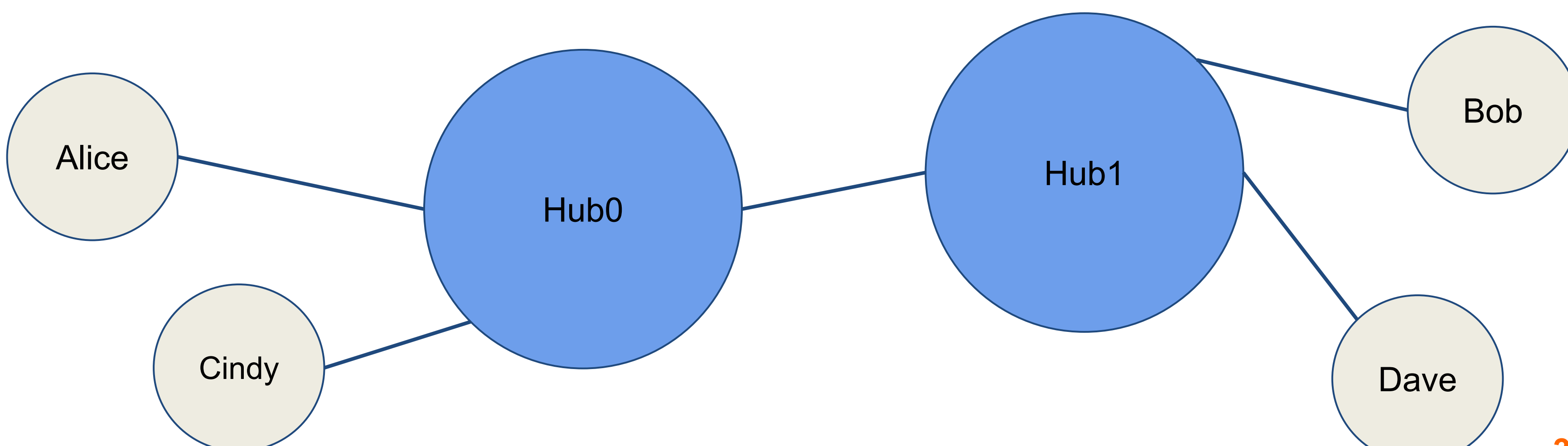
- Payment channel reduces # txs on-chain for pairs of users

- But to use it, you need to open payment channel first
  - The cost to maintain many channels is high

- Thus why we introduce payment network:
  - Reducing complexity from $O(n^2)$ to $O(n)$

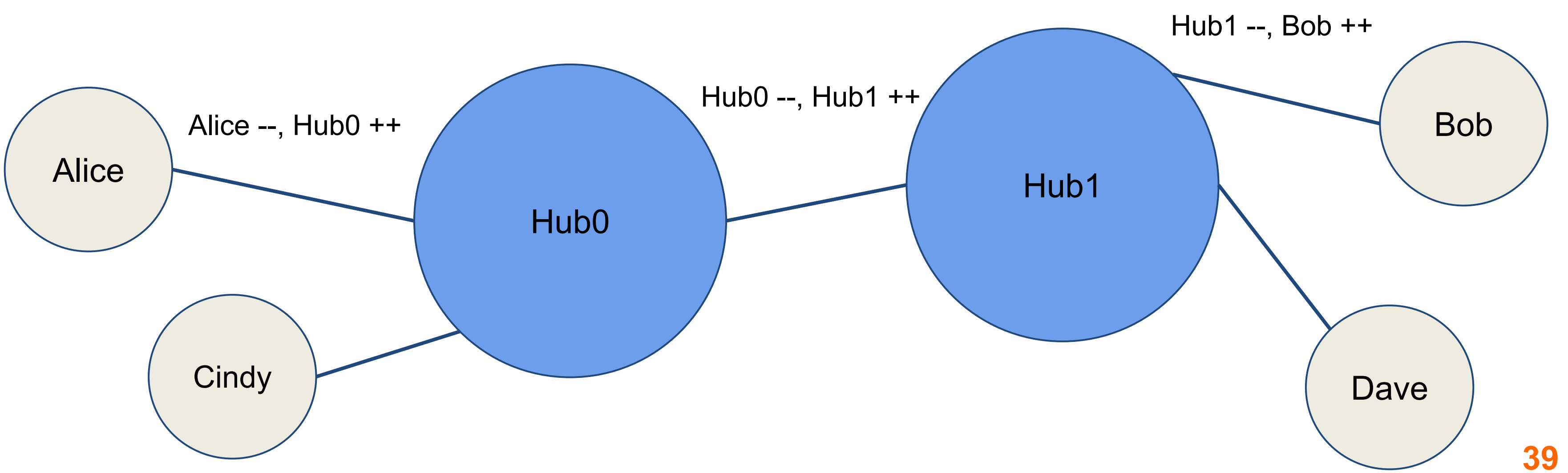
Payment Network Intuition

- There are two kinds of nodes: users and hubs
- Hubs act as routers
  - The links between user-hub and hub-hub are payment channels
  - Each payment is done by multiple payment channel changes



Payment Network Intuition

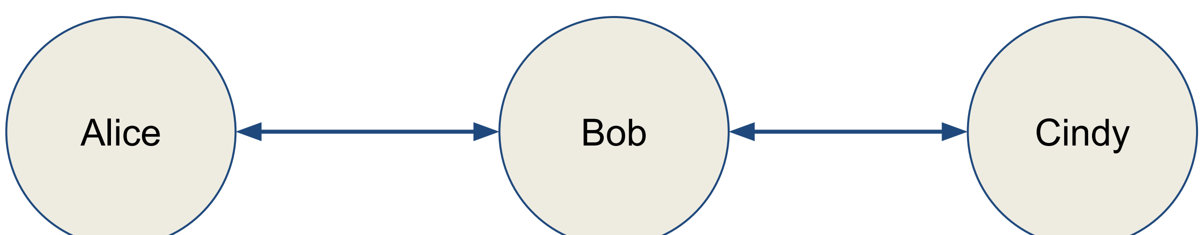
- Example: Alice -> Bob $1
  - 3 related channel changes
- Challenge:
  - There can be malicious hubs / users. How can we make sure the state changes are atomic?
    - Otherwise, hub can take the payment for free



Intro to Hashed Timelock Contracts (HTLC)

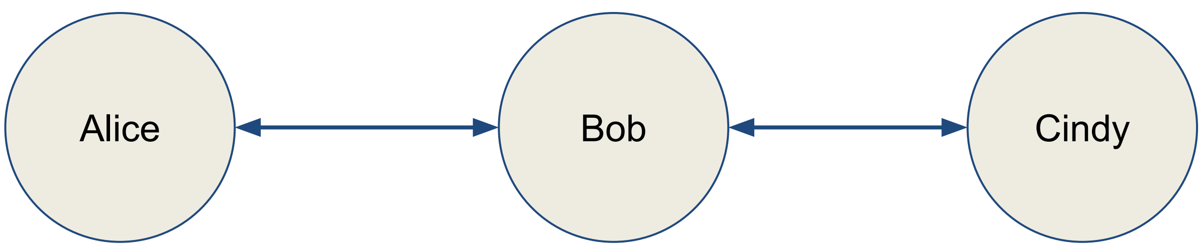
Hashed Timelock Contracts

- Suppose Alice has payment channel with Bob, and Bob has payment channel with Cindy
- Alice is going to pay Cindy $100



Hashed Timelock Contracts

- Alice is going to pay Cindy $100
  - Cindy generate a random number m and its hash H(m). Cindy gives H(m) to Alice
  - Alice updates the channel to a conditional payment: she will pay Bob $101 if Bob shows m
  - Bob wants $101, so he updates the channel with Cindy to a conditional payment: he will pay Cindy $100 if Cindy shows m
  - Cindy knows m, so she shows m and gets $100. This makes Bob know m
  - Bob shows m to Alice, and he gets $101
  - All can be done off-chain.



Payment Network Summary

- Based on HTLC, we are able to concat multiple payment channels
- We have payment network that scales. The bottleneck moves from on-chain to off-chain
- There are still problems, e.g.,
  - How to do routing in state links?
  - Who pays deposit in hubs?
  - Links in the end of slides

Outline

- Short introduction to Bitcoin
- Scaling limitation and payment channel
- From payment channel to payment network
- Smart contract and state channel

Why Smart Contract?

- Bitcoin system is not expressive enough
- It’s hard to implement our previous design
  - Multi-signature, nonce, conditional payment and more complex resolving logic

Smart Contract Intuition

- Bitcoin is a special state machine (payment system)
  - It’s log based
  - Each entry is a payment transaction

- Why not design a general state machine?
  - Also log based
  - Each entry is an instruction

Short Intro to Ethereum

- There are accounts and contracts
  - Account is like Bitcoin account controlled by a user
    - Account has balance
    - Account can do payment
    - Account can call function of contract
  - Contract is state + stateless functions
    - Contract has balance
    - Contract can do payment
    - Contract can call functions of itself/other contracts

Short Intro to State Channel

- With smart contract, we can do complex actions related with value transfer
  - Play chess with conditional payment
- But Ethereum has similar throughput / latency limitations as Bitcoin
  - We can use similar approach to scale it
  - We call it state channel
    - Instead of agreement on balance, we agree on state (byte array)
    - Resolving logic knows the mapping from state to balance
  - Similarly, we can build networks of state channel

Limitation

- State channels only deal with interaction between two parties
  - Can scale to multiple, but not a lot
    - Because we need multi-signature
- Information is not able to be shared between state channels before finalizing
- Need system to monitor on-chain status and dispute

Summary

- How to build a layer 2 payment network to scale up irreversible payment system
  - Not ordering all txs on-chain
  - Reduce the complexity by network
- Here are some useful links:
  - https://www.celer.network
    - Have a chess game on Blockchain testnet (Android only)
  - https://www.learnchannels.org
  - https://offchainlabs.com/
    - Princeton