Bitcoin and Blockchain

COS 418: *Distributed Systems*
Lecture 18
Zhenyu Song
[Credit: Selected content adapted from Michael Freedman.
Slides refined by Chris Hodsdon and Theano Stavrinos]

Why Bitcoin? All about Trust

- Problem with current payment system
  - Reversible: bank can reverse your payment
    - Need trust from third party, e.g., trust the bank to not reverse 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

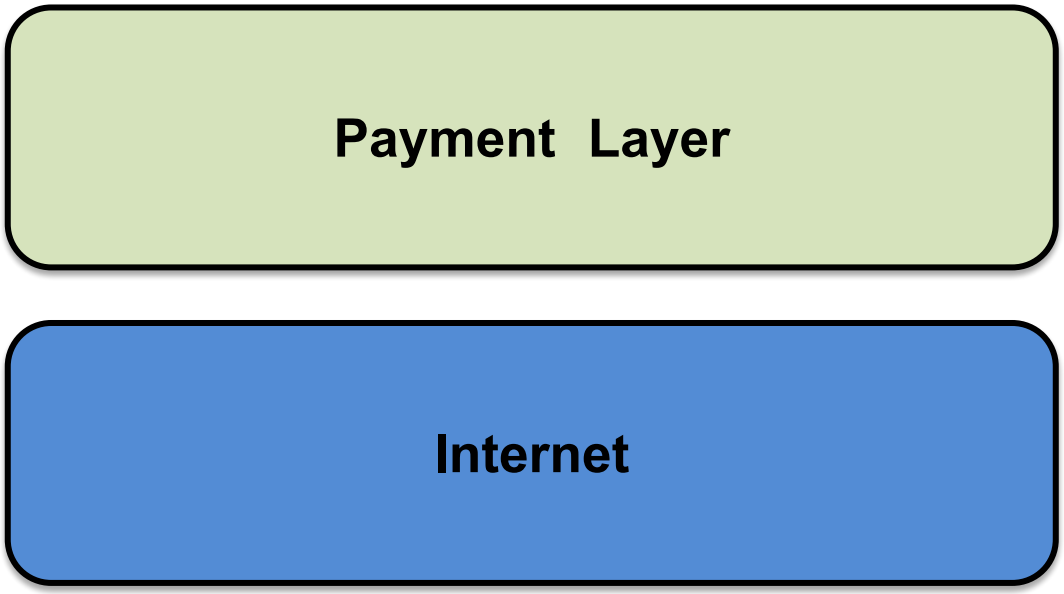
Design Intuition

- Run a consensus protocol for a consistent view of payment history
  - The protocol guarantees the irreversibility
  - Anyone can participate
  - As long as a majority of servers are cooperative, the system is safe

Distributed Payment Layer

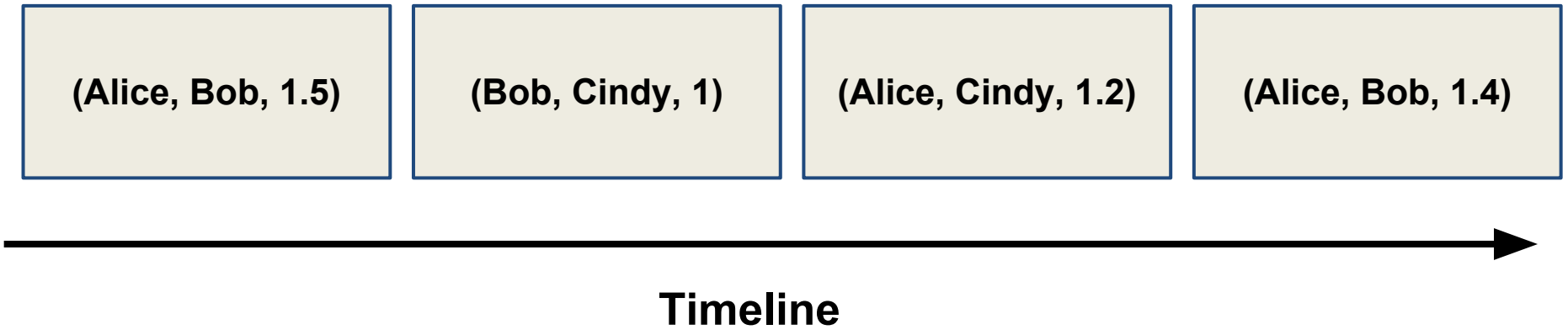
- A stateful layer: state transition has constraints
  - After a payment, the total sum of balance is unchanged

check_balance(id) **send(id0, id1, amount)**



Transaction Log

- Instead of directly storing key-value pairs, we store a list of transactions: (id_{src}, id_{dst}, amount)
  - This means id_{src} sent id_{dst} amount of BTC
  - We can construct the balance by iterating along the list



Problem: Proof of Transaction

- Users need to prove they actually made the transaction
  - Bitcoin uses Cryptography Signature
  - Alice signs the transaction before she shows the transaction to others

Intro to Cryptography Signature

Public-Key Cryptography

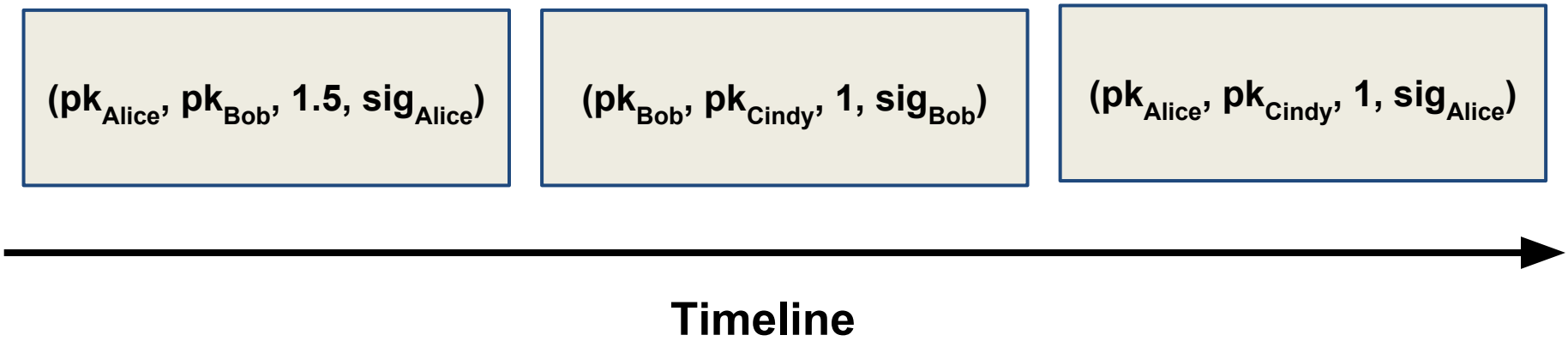
- **Each party has (public key, secret key)**

- **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

Combine Signature with Transaction

- We use public keys as identifiers
- Each entry becomes (pk_{src}, pk_{dst}, amount, sig_{src})
- Is this good enough?



Problem: How to Build Consensus

- Any entity can insert/delete transactions anytime
  - Suppose Alice only has 1 BTC, she may be able to spend it several times
    - Alice pays Bob 1 BTC, and receives the product from Bob
    - Later this transaction is deleted, so Alice has 1 BTC again
    - And Alice still keeps the product
  - This is called the double spending problem
- To solve this: use Cryptography Hash to make the transactions append only

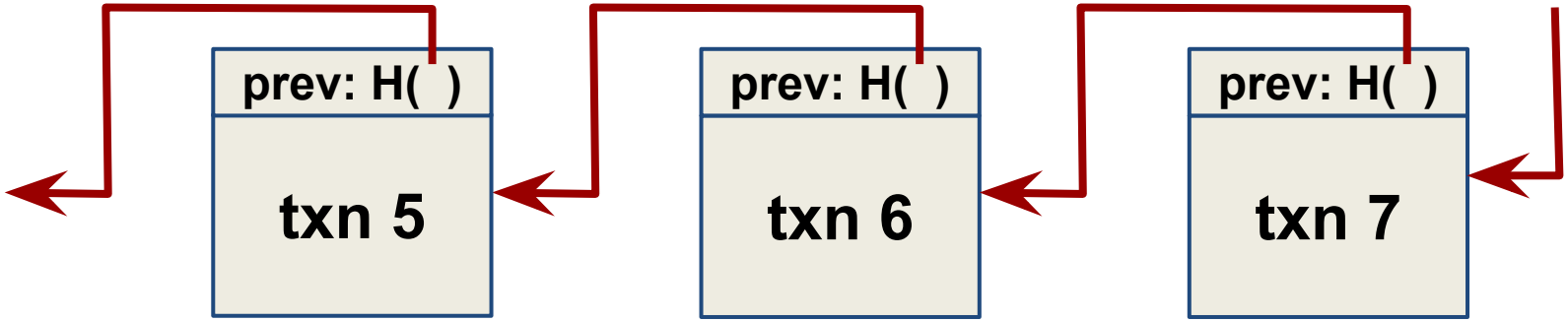
Intro to Cryptography Hash

Cryptography Hash Functions

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

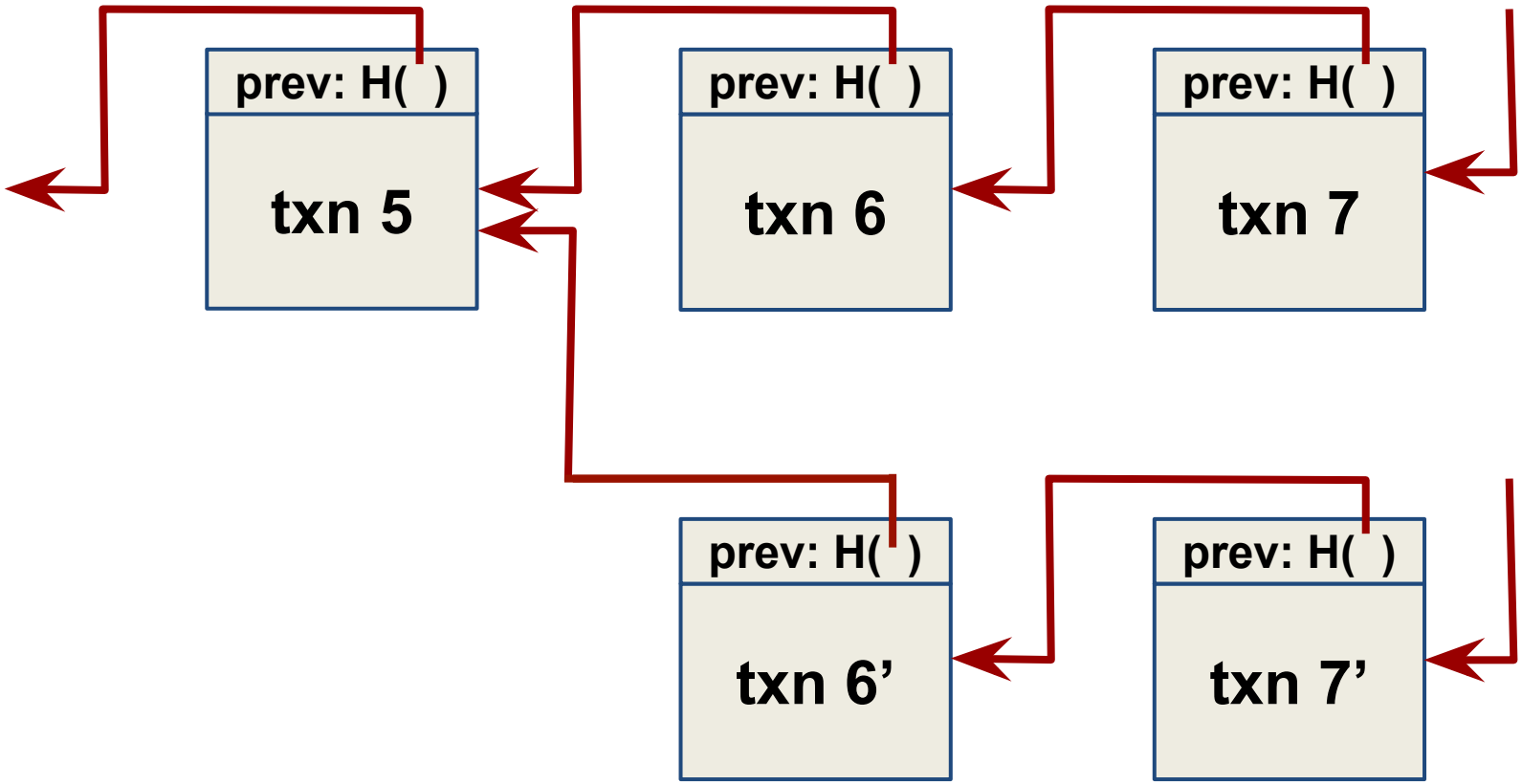
- One-way function
  - Efficient: Easy to compute *H(m)*
  - Hiding property: Hard to find an *m*, given *H(m)*
  - Collisions exist, but hard to find
    - For SHA-1, finding any collision requires 2^{80} tries. Finding a specific collision requires 2^{160} tries.

Blockchain: Append-only Hash Chain

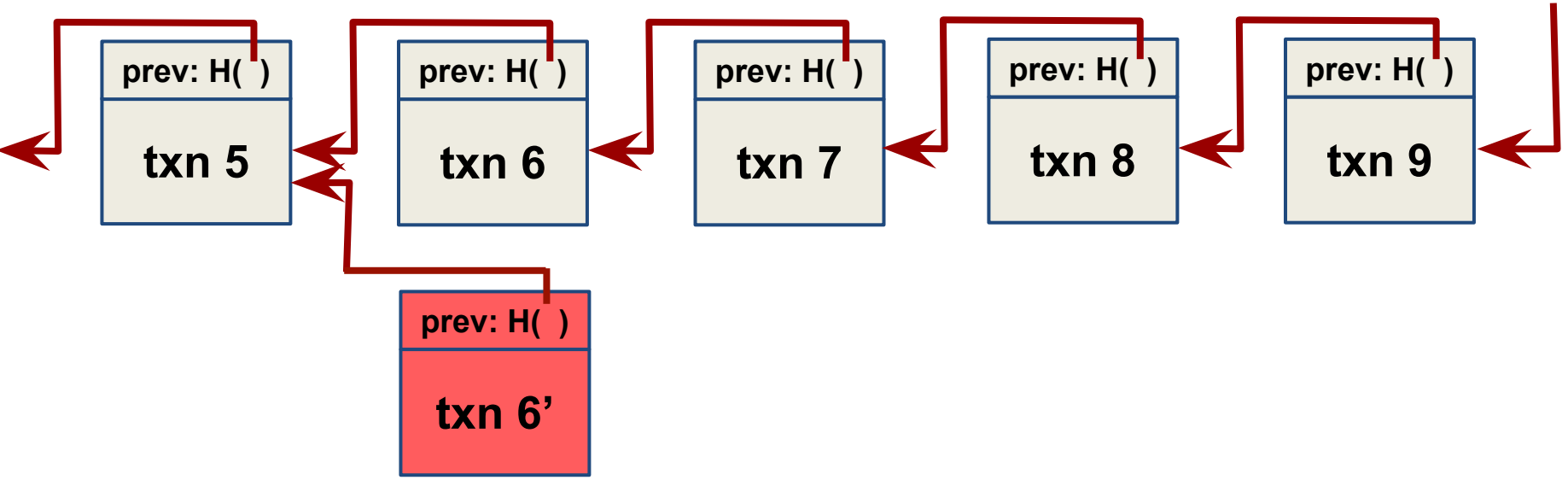


- Each block contains the hash of previous block
  - Block proposer includes the hash
- This gives a sequential order
  - Not real time. Assigned by proposer
- Did we succeed in building consensus?

Problem Remains: Forking



Key Idea: Proof of Work



- New design: 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

Use Hashing to Determine Work!

- Recall hash functions are one-way/collision resistant
  - Hard to find an *m*, given *H(m)*
- But what about finding partial collisions?
  - *m* whose hash has most significant bit = 0?
  - *m* whose hash has most significant bit = 00?
  - Assuming output is randomly distributed, complexity grows exponentially with # bits to match

Bitcoin Proof of Work

Find **nonce** such that

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

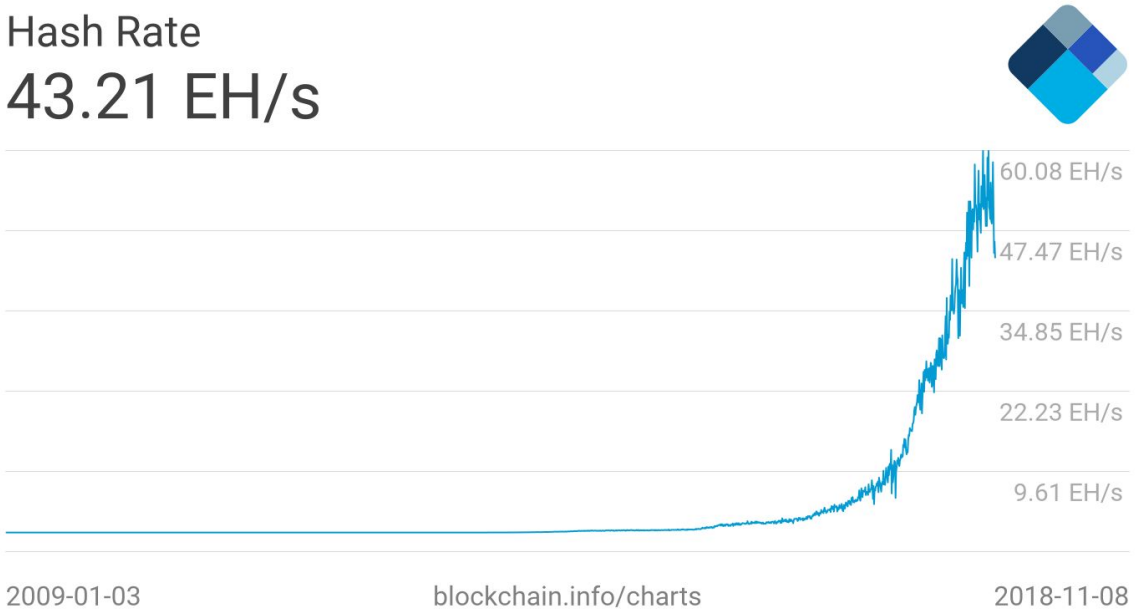
i.e., hash has certain number of leading 0's

What about changes in total system hashing rate?

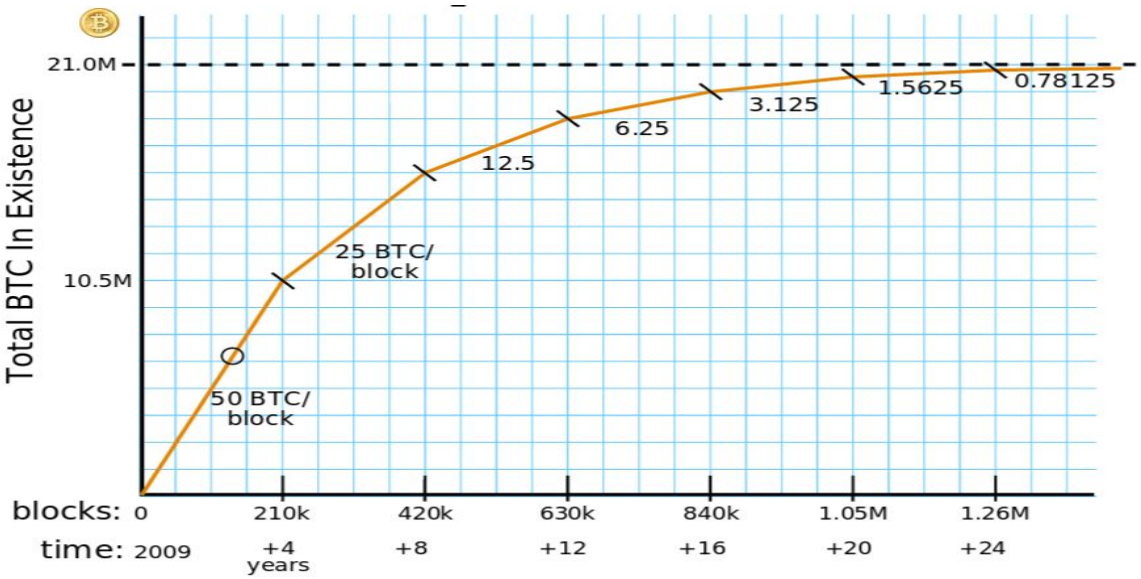
- Target is recalculated every 2 weeks
- Goal: one new block every 10 minutes

Hash Rate Trends of Bitcoin

- To achieve this with normal PCs (less than 40 MH/s), requires 1,000,000,000,000 PCs
- The tech trend: CPU -> GPU -> FPGA -> ASIC



Why Consume All This Energy?



- Creating a new block creates bitcoin!
  - Initially 50 BTC, decreases over time, currently 12.5
  - New bitcoin assigned to party named in new block
  - Called “mining” as you search for gold/coins

Incentivizing Correct behavior?

- Race to find nonce and claim block reward, at which time race starts again for next block

- Correct behavior is to accept longest chain
  - So miners incentivized only to work on longest chain; otherwise solution is not accepted
  - Remember blocks on other forks still “create” bitcoin, but only matters if chain is in collective conscious (majority)

Design Detail: Transaction Format

- In real Bitcoin, a transaction is not in the format of a tuple of (pk_{src}, pk_{dst}, amount, sig_{src})
- Why? Miner can append the same transaction multiple times!
- This is similar in “at most once” execution

Real Transaction Format

- Four fields
  - Hash: hash of this transaction
  - Inputs: hashes of one/more previous transactions
  - Outputs: one/more (amount, public key) pairs
  - Signatures: signatures by each input coin owners
- Example: Alice sends Bob 1 BTC. Alice's 1 BTC is from the previous transaction x.

Hash: h
Inputs: h_x
Outputs: 1 -> pk_{Bob}
Signature: Sig_{Alice}

Real Transaction Format

- Each input must be fully spent
  - Then what if the owner only wants to spend part of the coin? Make the owner a receiver for rest of the coin
- Example: Alice uses a 3-BTC tx as input, and sends 1 BTC to Bob, and the rest to herself

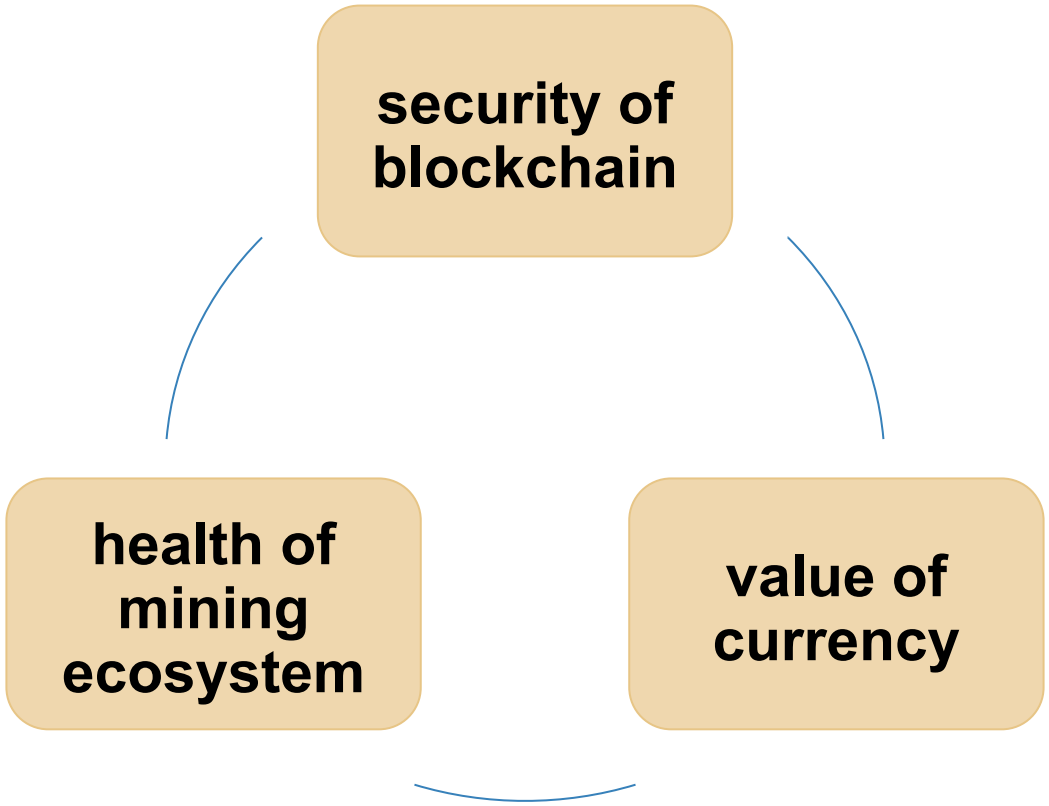
Hash: h
Inputs: h_x
Outputs: 1 -> pk_{Bob}, 2 -> pk_{Alice}
Signature: Sig_{Alice}

Real Transaction Format

- The outputs amount is not necessarily equal to inputs
  - Unspent portion of inputs is transaction fee to miner
  - This acts as another incentive

Hash: h
Inputs: h_x
Outputs: 1 -> pk_{Bob}, 2 -> pk_{Alice}
Signature: Sig_{Alice}

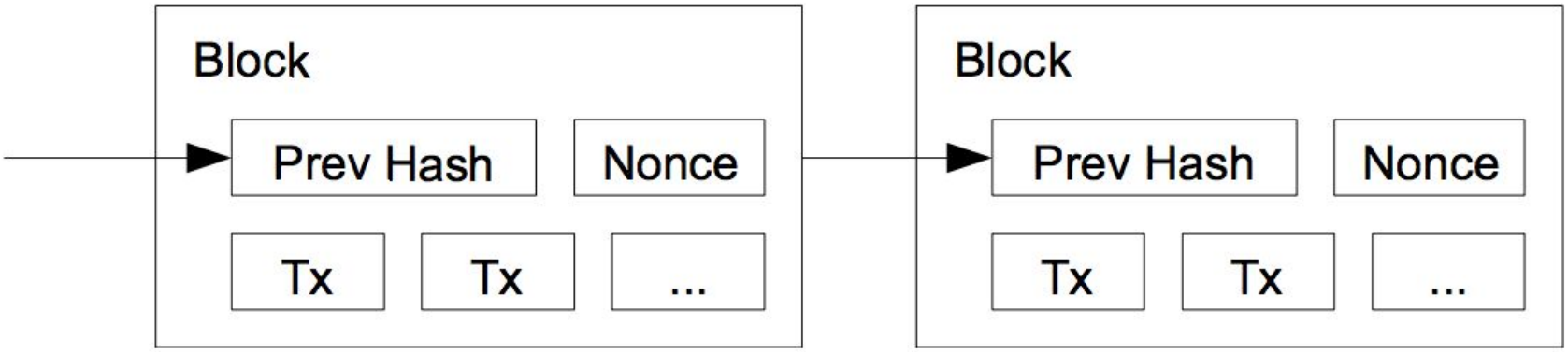
Bitcoin & Blockchain Intrinsically Linked



Performance Issue

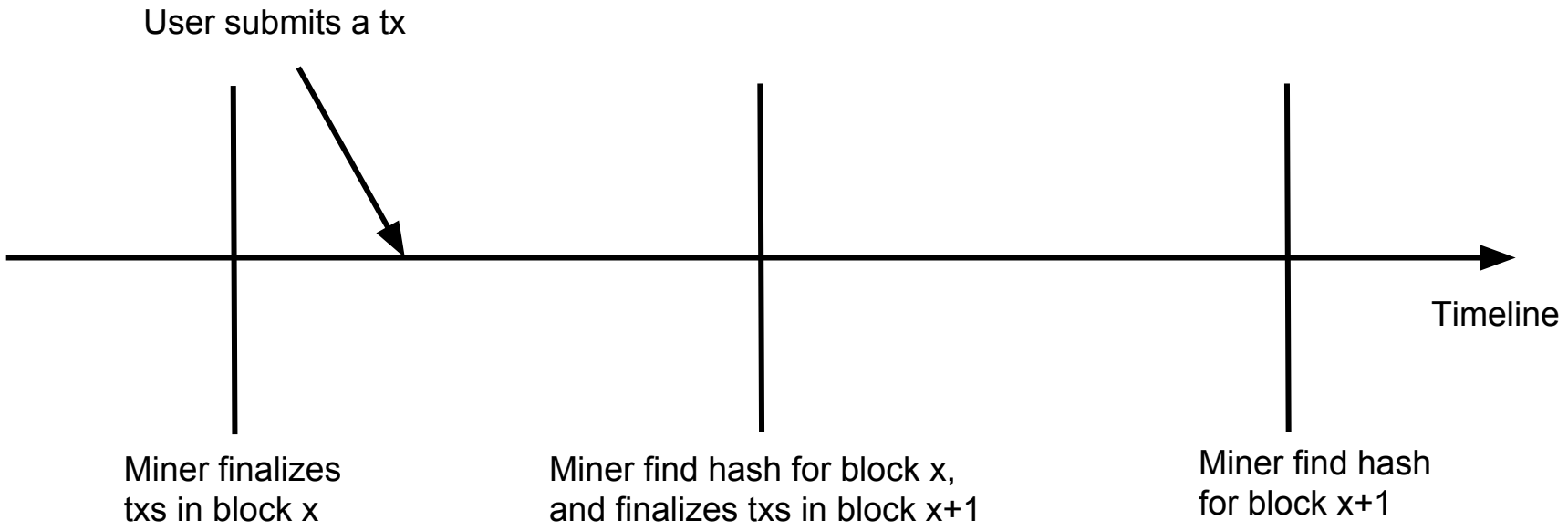
- Our current version processes 1 transaction every 10 min
  - Extremely slow!
- How to improve this?

Batching Transactions into Blocks



- Miner picks the set of transactions
- Builds block header: prevhash, version, timestamp, txns, etc
- Until it wins OR another node wins

Transactions Are Delayed

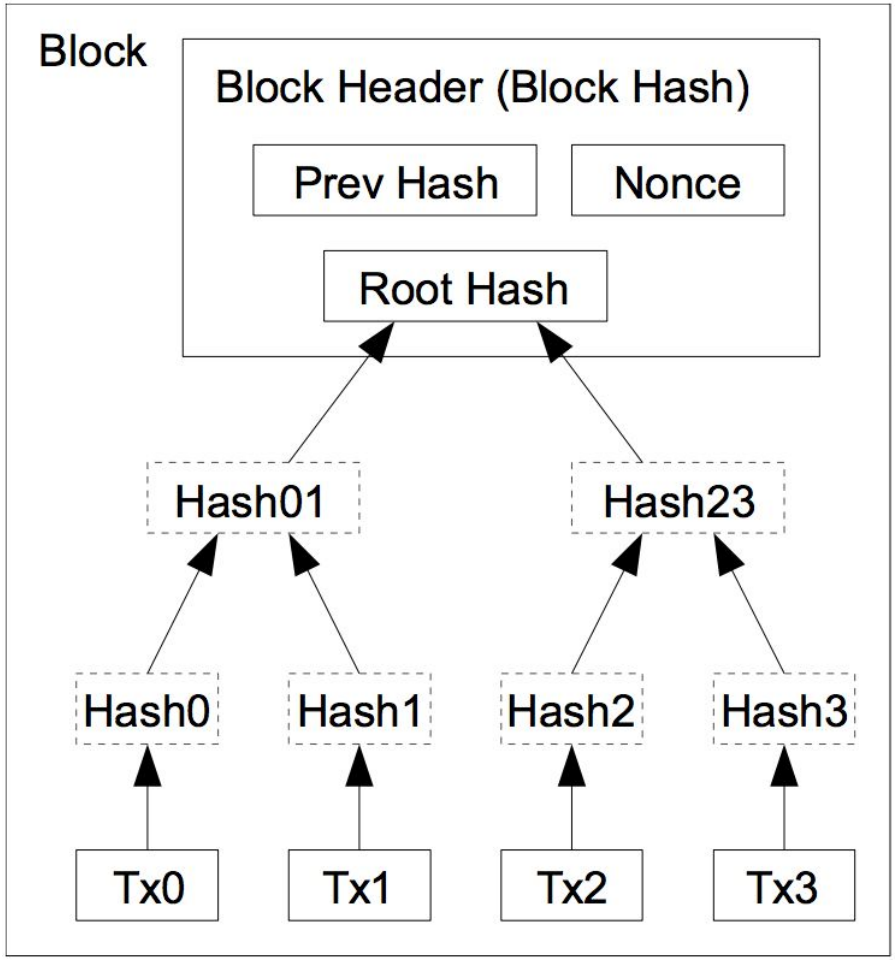


- After a user submit a tx to miner, it will not be included in the block computing right now
- So transactions are from 10 - 20 min before block creation
- Can be much longer if many pending transactions

Commit Further Delayed

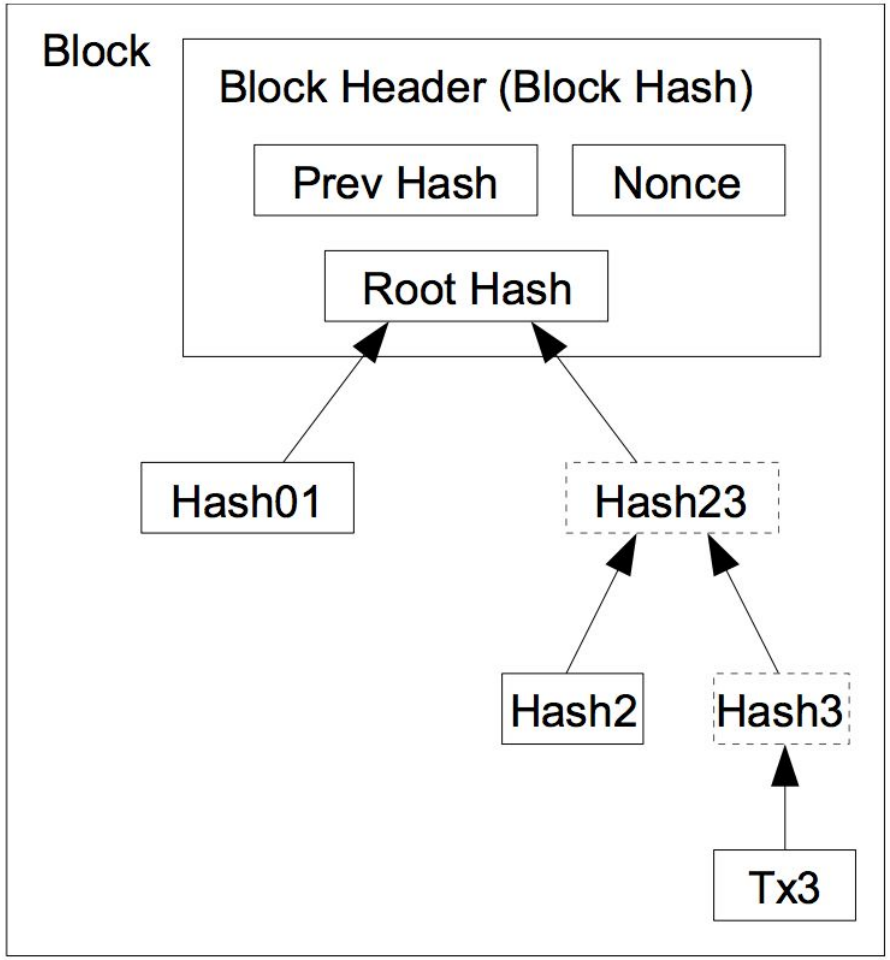
- When do you trust a transaction?
  - After we know it is “stable” on the hash chain
  - Recall that the longer the chain, the harder to “revert”
- Common practice: transaction “committed” when 6 blocks deep
  - i.e., takes another ~1 hour for txn to become committed

Storage / Verification Efficiency



- Merkle tree
  - Binary tree of hashes
  - Root hash “binds” leaves given collision resistance
- Using a root hash
  - Block header now constant size for hashing
  - Can prune tree to reduce storage needs over time

Storage / Verification Efficiency



- Merkle tree
  - Binary tree of hashes
  - Root hash “binds” leaves given collision resistance
- Using a root hash
  - Block header now constant size for hashing
  - Can prune tree to reduce storage needs over time
    - Can prune when all txn outputs are spent

Bitcoin Protocol Analysis

- Safety:
  - No. Only probabilistic. The deeper block, the safer
- Liveness:
  - Yes. You can always compute a hash in a finite number of steps

Limitation of Scaling

- Scaling limitations
  - 1 block = 1 MB max, ~ 2000 txns, ~ 10 min
    - 3-4 txns / sec
    - Visa payment system: typically 2,000 txns / sec
  - The fundamental limitation on sequential consistency
    - Remember in Lecture 12 we talked about PRAM
      - read time + write time ≥ max delay
      - Blockchain is designed to read on one node, write on all nodes
      - Each block needs to propagate to all nodes around the world
      - We cannot make packet travel faster than light

Summary

- Payment system
  - Coins transfer/split between “addresses” (public keys)
- Blockchain: globally-ordered, append-only log of txns
  - Reached through decentralized consensus
  - Nodes incentivized to perform work and behave correctly
    - When “solving” a block, get block rewards + txn fees
    - Only “keep” reward if block persists on main chain

What's Going on This Area

- Bitcoin happened ~10 years ago
- There is exciting progress in the area in recent years
  - Ethereum: Turing-complete Blockchain to support more complex state transitions
    - Run program (smart contract) on blockchain
    - ICO, Stable coin (Tether, USDC, GUSD, PAX)
    - Can implement financial derivatives like option contract, future contract…
    - More main-chains are developing
  - Layer 2: scale the blockchain by releasing sequential consistency:
    - State channel and side chain
  - See optional slides for more information about state channel