Blockchain/Bitcoin Fundamentals

Jonathan Katz

Blockchain

Outline: Part 1

- What is consensus?
- Classical results on consensus
- Cryptographic hash functions
- Blockchain: Nakamoto consensus

Bitcoin

Outline: Part 2

- Digital signature schemes
- From consensus to cryptocurrency
- Bitcoin

Consensus/Byzantine agreement

- Network of distributed processors
  - Can communicate with each other over point-to-point links
- Want to maintain a *consistent view* of the state of the system
  - I.e., to reach *agreement*
- Challenge: some processors may fail, or be compromised and behave arbitrarily
- A *consensus protocol* allows the processors to reach agreement even in the presence of (a bounded number of) faults

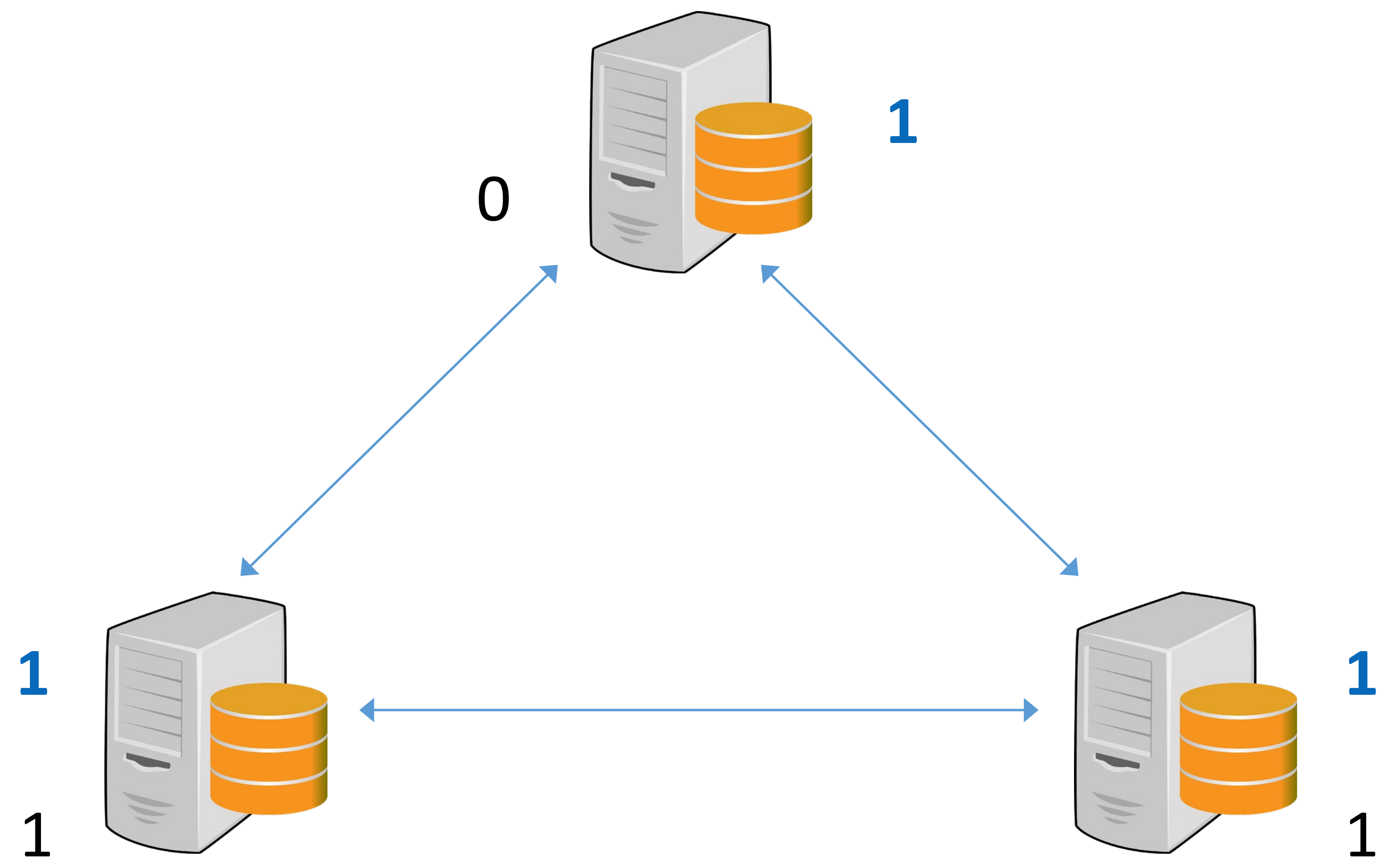
Consensus

- Many different ways to formulate the problem
  - I.e., different ways to define the requirements
- Many different settings can be considered
  - Communication model
  - Fault model
  - Prior setup
  - Cryptographic assumptions
- Subtle changes in the requirements or the setting can have a significant impact on feasibility or efficiency of consensus

Consensus

- Begin by considering a simple setting
  - Parties fixed in advance
  - Synchronous communication network
  - No prior setup
- Begin by considering simple requirements
  - All processors begin holding some (possibly different) input – just a bit!
  - All processors must terminate with the same (nontrivial) output

Consensus



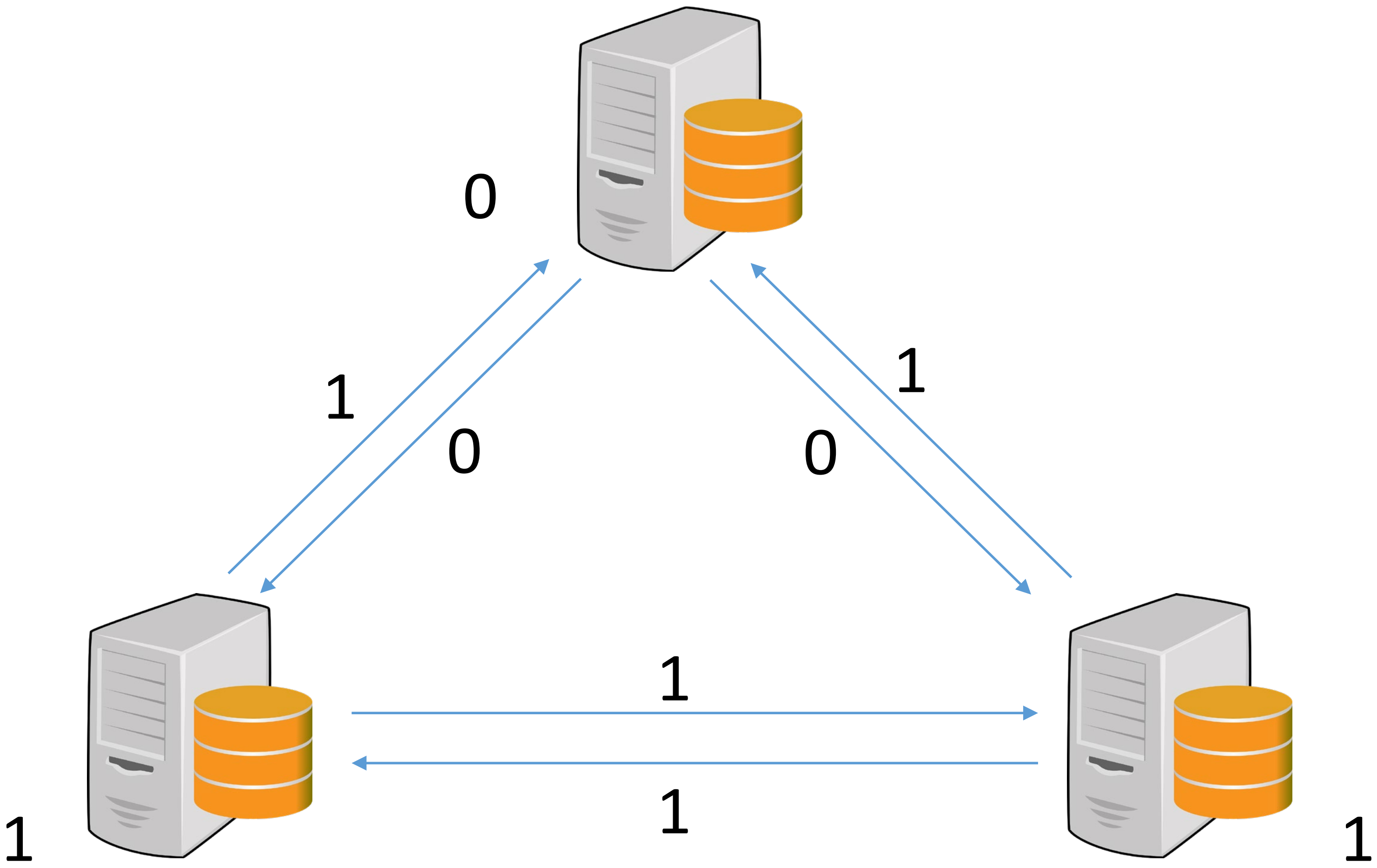
Consensus requirements formalized

- Each processor P_i begins with an input x_i
- After running the protocol, each processor must terminate with an output y_i
- **Agreement** (consensus):
  - Every correct processor must output the same value
  - I.e., if P_i and P_j are both correct, then $y_i = y_j$
- **Validity** (non-triviality):
  - If all correct processors begin holding the same input value, then they should all output that value
  - I.e., if all correct P_i hold $x_i = v$, then all P_i must output $y_i = v$

Strawman protocol

- Each processor sends the others its input
- Each processor outputs the majority value of all inputs
  - Output default value (say, 0) if no majority

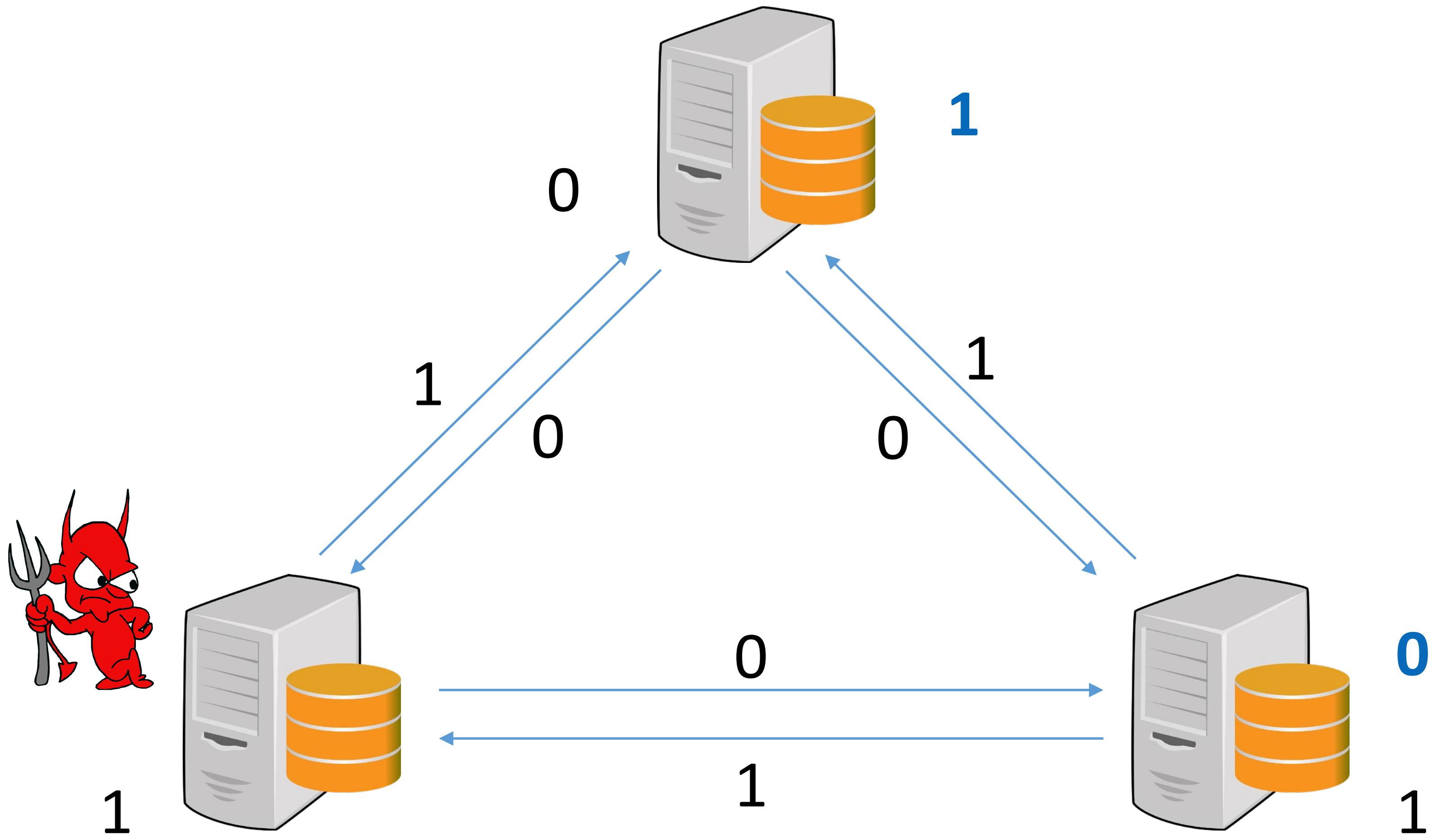
Consensus



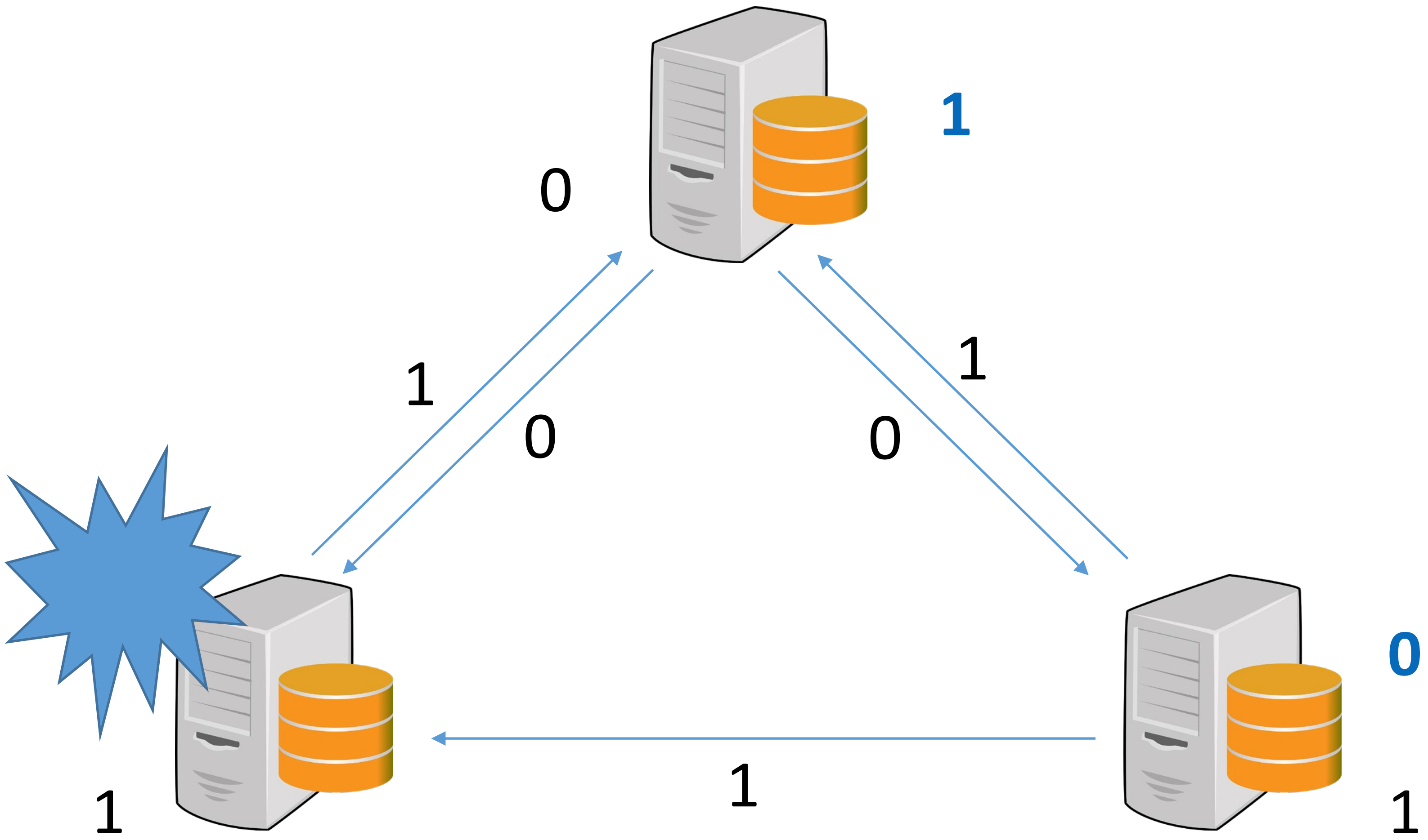
Faults?

- What does it mean for a processor to fail?
- Two common models
  - **Fail-stop model**: processor crashes at an arbitrary point in its execution
  - **Byzantine (adversarial) model**: processor behaves arbitrarily; actions of all faulty processors may be coordinated by a single adversary

Byzantine case



Fail-stop case

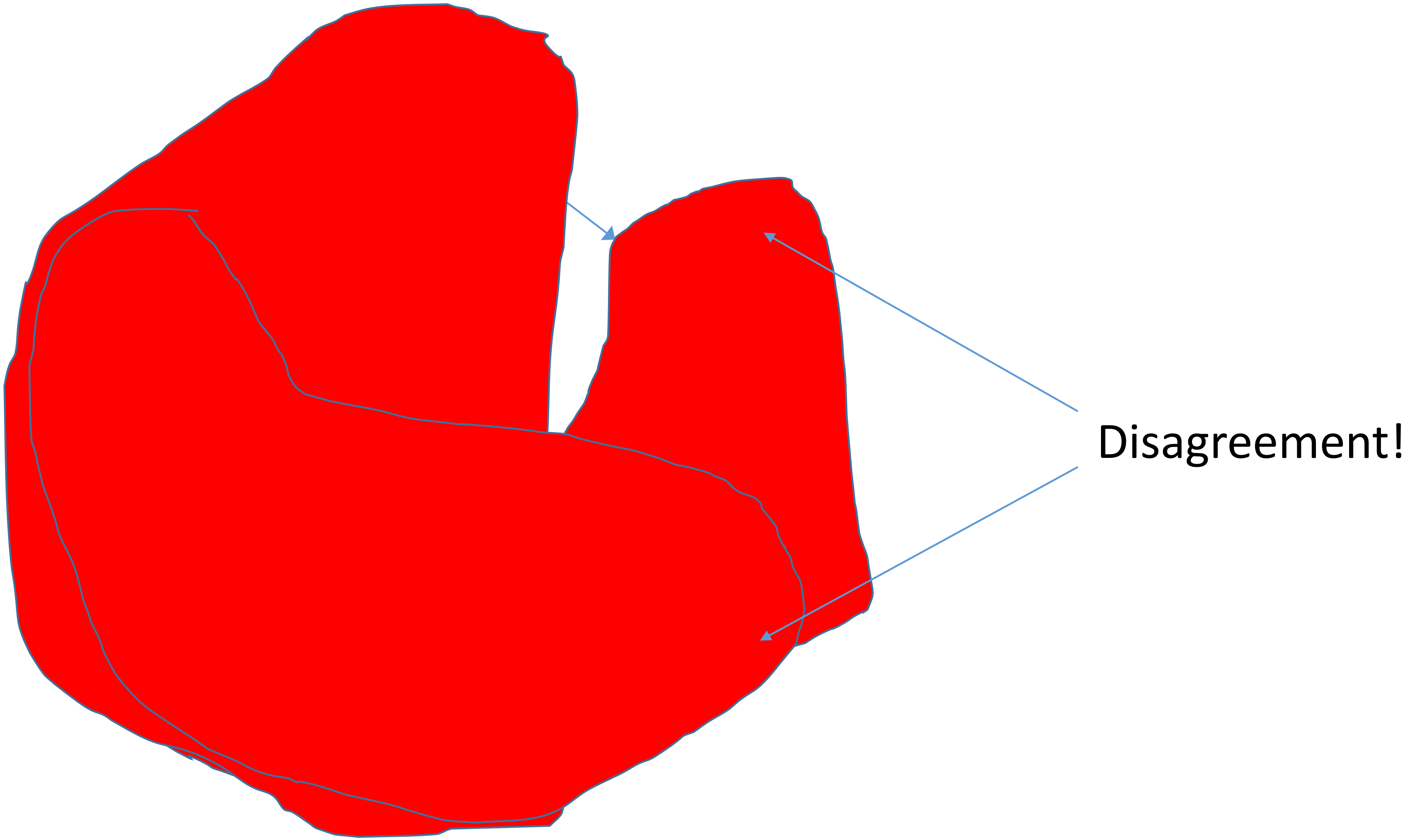


Feasibility?

- Consensus is possible* iff $t < n/3$ of the processors may be adversarial

*Assuming synchronous communication and no prior setup

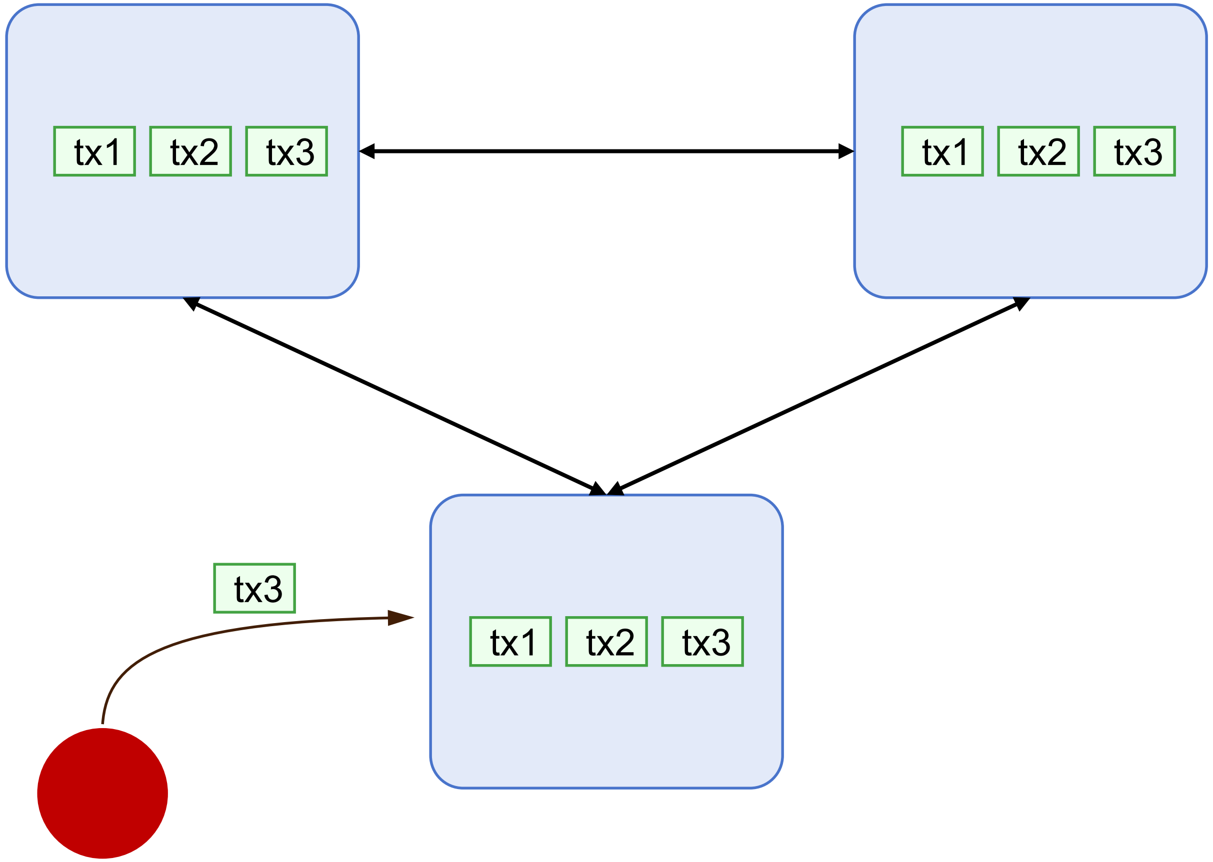
Proof of impossibility



Replicated state machines

- So far, we have viewed consensus as a "one-shot" mechanism
- In real-world systems, processors must repeatedly agree on values over time
  - No "termination"
- More generally, think in terms of maintaining agreement *on an ordered list of values*
  - Commands to be executed
  - Transactions
- Refer to the list as a "log" or "ledger"

Replicated state machines



Requirements (informal)

- Processors receive transactions over time, and *commit* to an ordered list of transactions over time
  - Let $t_i[j]$ denote the transaction committed by processor i and index j
  - $t_i[j]$ must be committed before $t_j[j+1]$
  - Once $t_i[j]$ is committed, it cannot be changed
- **Agreement**:
  - Correct processors agree on committed values
  - I.e., P_i and $P_{i'}$ are correct, and $t_i[j]$ and $t_{i'}[j]$ are both committed, then $t_i[j]=t_{i'}[j]$
- **Liveness**:
  - A transaction received by a correct processor is eventually committed by all other correct processors

Notable protocols

- Paxos (1998) / Raft (2014)
  - Tolerates < n/2 fail-stop faults in an asynchronous network
- PBFT (1999)
  - Tolerates < n/3 Byzantine faults in an asynchronous network
  - Relies on digital signatures, with necessary keys distributed in advance

What is a blockchain?

- Blockchain = protocol for realizing consensus

- Centralized consensus (with one processor/database) is trivial…
- Decentralized consensus has been studied since the 1970s…

- Why the recent surge in interest?

Why the interest in Blockchain (1)?

- Nakamoto introduced a *permissionless* consensus protocol
  - All prior protocols we have discussed (and almost all prior work) assumes a *permissioned* setting
- Permissioned:
  - Set of processors *fixed*; all processors aware of each other, and can be provisioned with other processors' cryptographic keys, etc.
- Permissionless:
  - Processors can come and go as they like!
  - No one manages membership
  - Processors not aware of all other processors

Permissionless setting

- One of the main challenges in the permissionless setting is handling **Sybil attacks**
  - Will see later how Nakamoto consensus prevents this using *proofs of work*
- Sybil attacks:
  - A compromised processor pretends to be multiple processors
  - Even a single compromised processor can become the majority!
- Note some blockchain protocols assume the permissioned setting…

Why the interest in Blockchain (2)?

- Nakamoto consensus can theoretically exceed the classical security threshold!
  - In network of fixed processors, all with same computational power, can tolerate $t < n/2$ faults
  - Uses hash functions/computational assumptions

- Exercise: how does it circumvent the impossibility result?

Why the interest in Blockchain (3)?

- Add an additional property (tamper-proofness):
  - Cannot easily tamper with the log maintained by the processors – *even if all processors are malicious!*
- This can be added on to existing protocols using *hash functions*, but (for the most part) was not explicitly considered before

(Cryptographic) hash functions

Hash functions

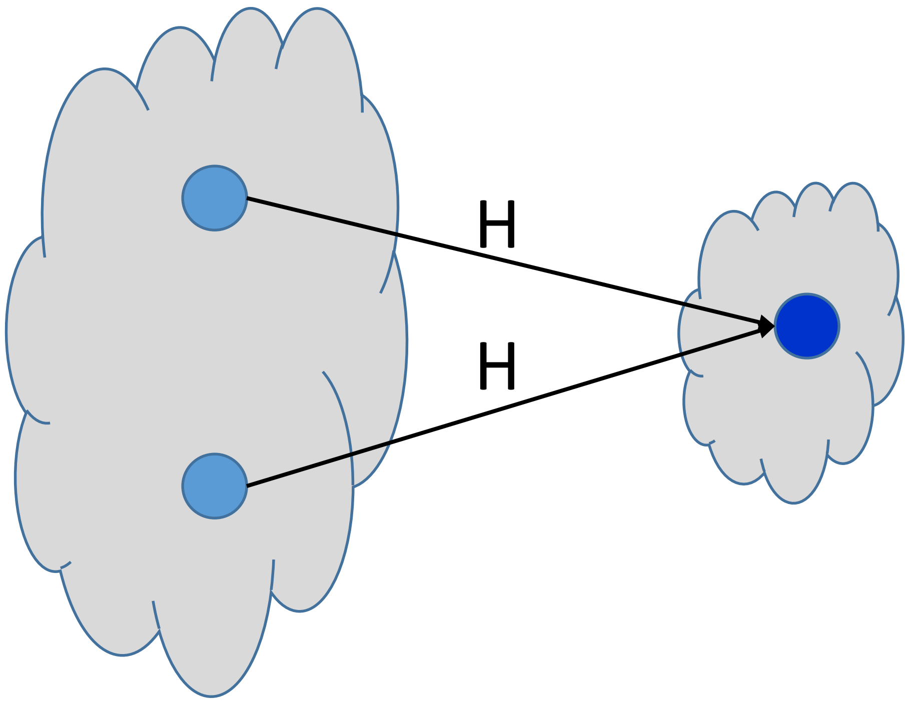
- Deterministic function H: $\{0,1\}^* \rightarrow \{0,1\}^n$
  - Arbitrary length inputs
  - Fixed-length (short) output
  - Efficient

Security properties

- Collision-resistant
- “Random behavior”
  - Proofs of work

Collision resistance

- Computationally infeasible to find two distinct inputs mapping to the same output



Generic collision attacks

- What can we say about the hardness of finding collisions (in general) as a function of the output length n?
- Naïve attack
  - Compute $H(x_1), \ldots, H(x_k)$ for $k = 2^n + 1$
  - **Guaranteed** to find a collision!
  - Is this the best possible attack?
- "Birthday" attack
  - Compute $H(x_1), \ldots, H(x_k)$ for *random* inputs and hope to find a collision
  - For what value of k is a collision expected with high probability?
  - Related to the *birthday problem*

Birthday problem

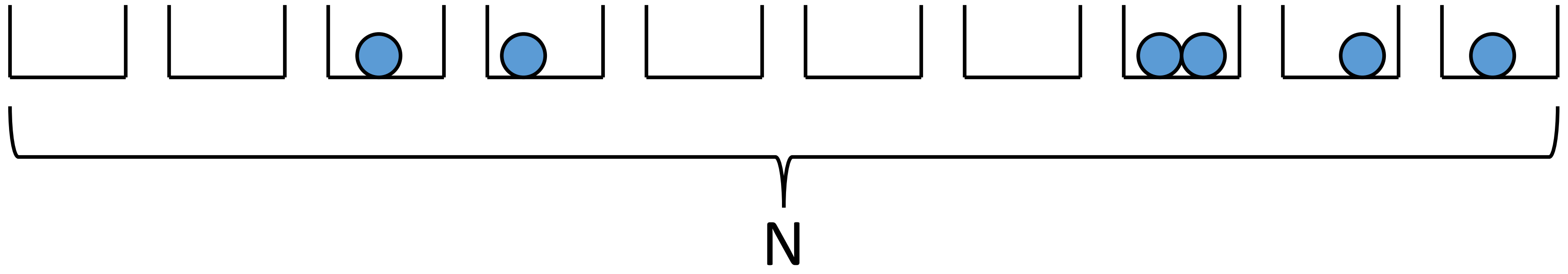
Bins: days of the year (N=365)

Balls: k people

Bins: values in $\{0,1\}^n$ $(N = 2^n)$

Balls: k hash-function computations

How many balls do we need
to have a 50% chance of a collision?



Birthday bound

- **Theorem**: the probability of a collision is $O(k^2/N)$
- When $k \approx N^{1/2}$, probability of a collision is $\approx$ 50%
  - Birthdays: 23 people suffice!
  - Hash functions: $O(2^{n/2})$ hash-function evaluations

- Need $n = 2k$ to get security against attackers running in time 2^k
  - I.e., 256-bit output to get "128-bit" security
  - Note that this is a lower bound; hash functions must be carefully designed!

Hashes as digests

- Collision resistance implies that H(x) can “stand in” for x
  - I.e., someone who obtains a reliable copy of H(x) cannot later be fooled into accepting a different value x’
  - Refer to H(x) as a **digest** of x
  - Note that H(x) is much smaller than x

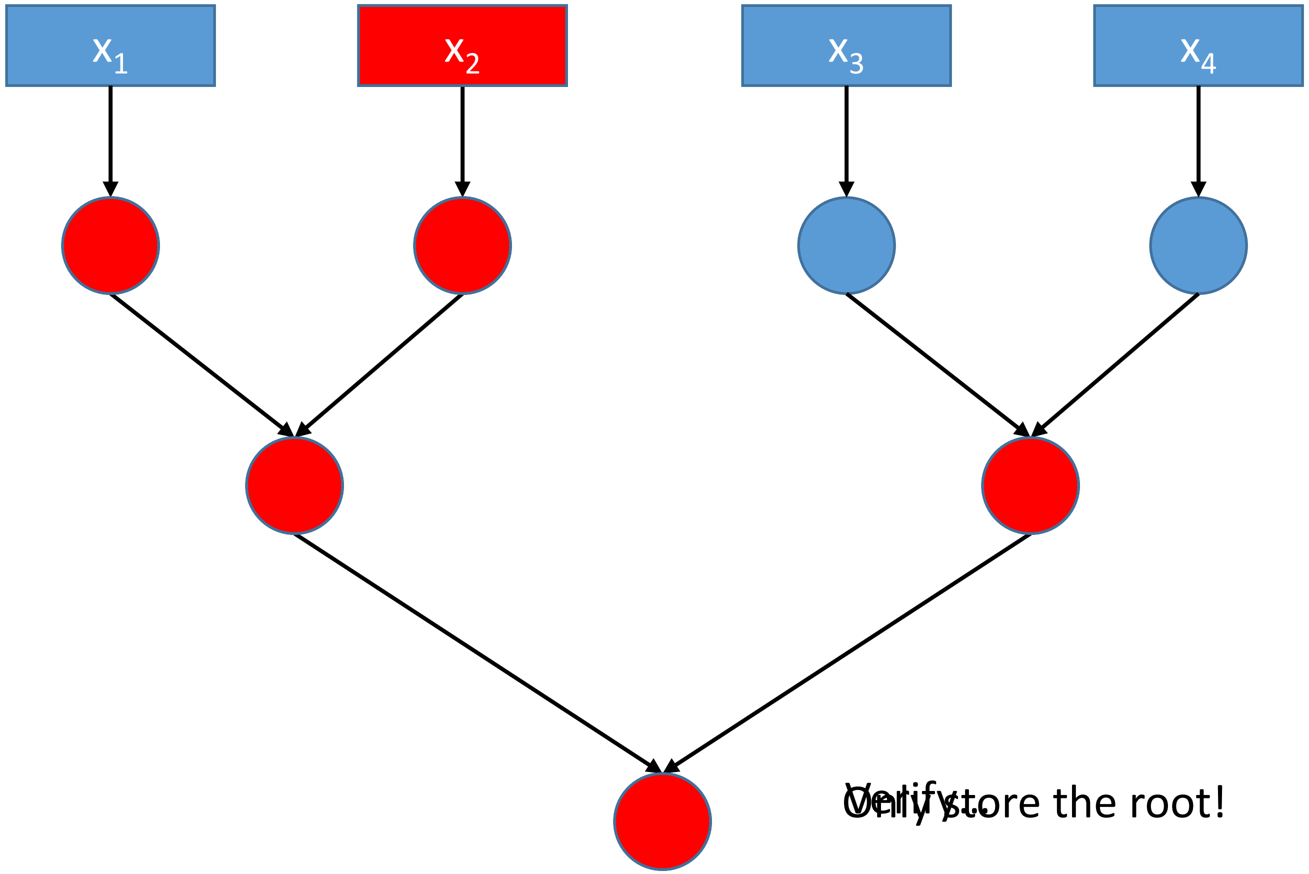
- What if we want a digest for multiple values?

Digests for multiple values

- Say we want to provide digests for $x_1, \ldots, x_k$
- Approach 1: compute $H(x_1), \ldots, H(x_k)$
  - Verifying x_i requires only x_i
  - Drawback: storage grows linearly in k

- Approach 2: compute $H(x_1, \ldots, x_k)$
  - This has constant storage
  - Drawback: verifying x_i requires all values

Merkle trees

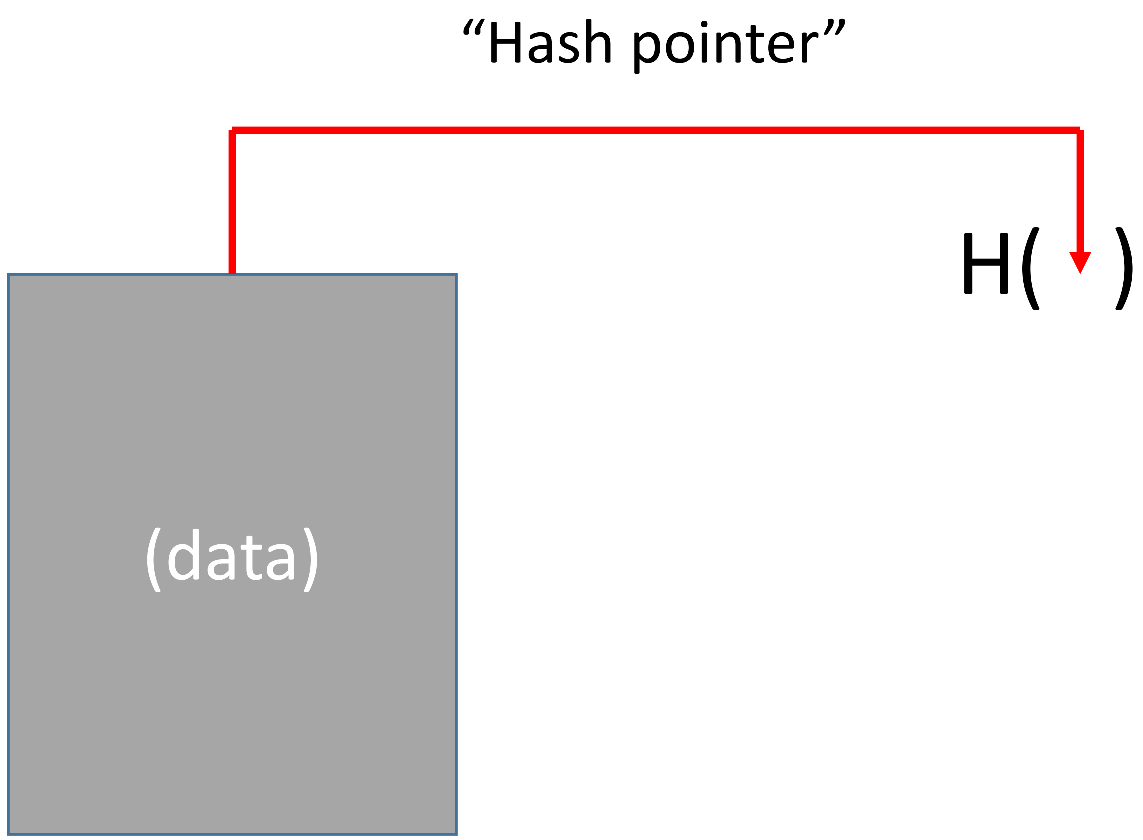


Verify…

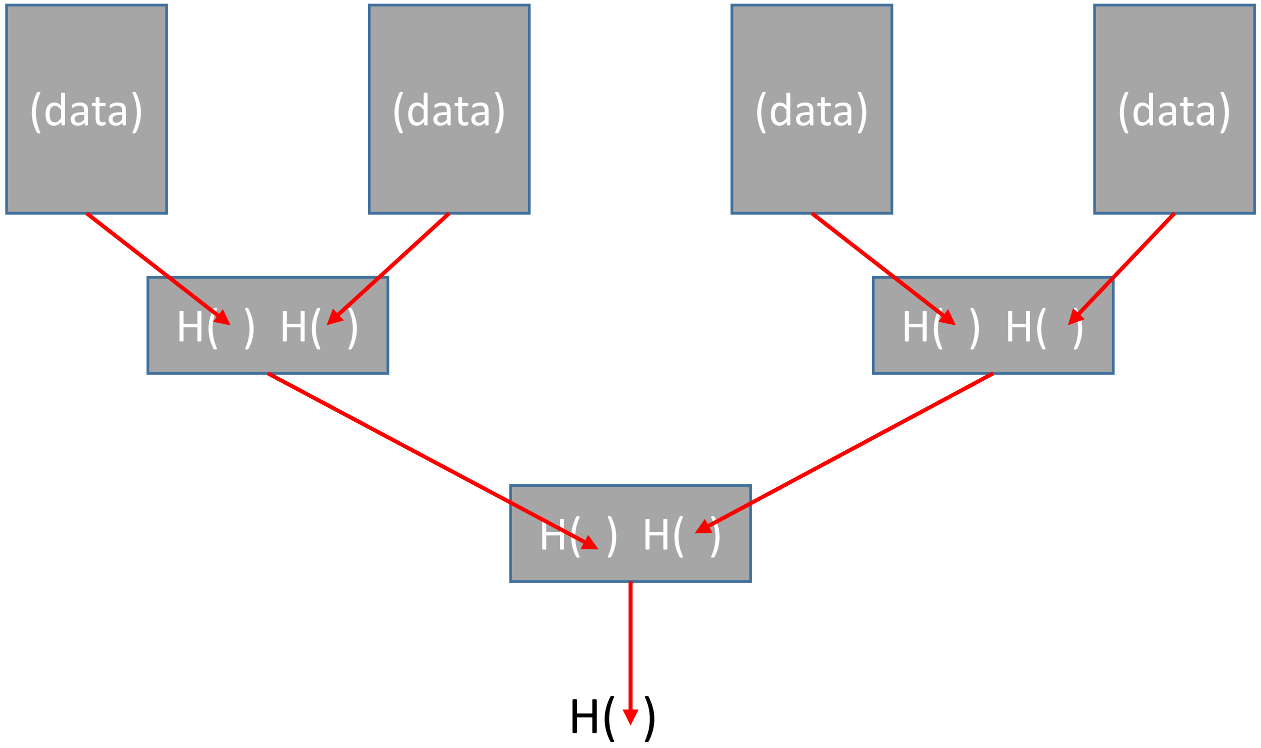
Only store the root!

O(log n) communication/computation!

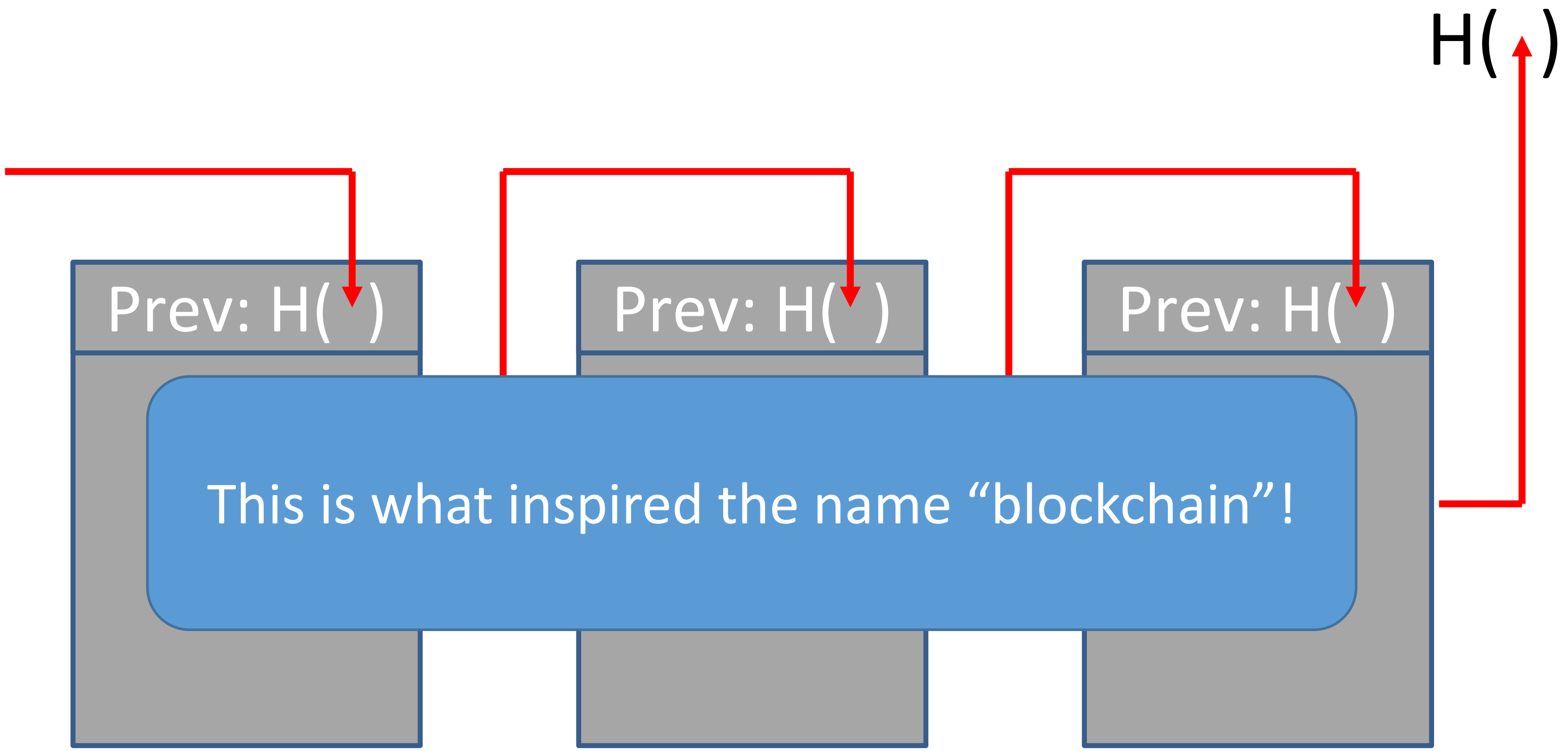
Digests, abstractly…



Merkle trees



"Linked list"



Modifying a block affects all subsequent blocks
(and the root digest)

“Random behavior”

- Roughly speaking, every evaluation of H on a new input should result in a “completely unpredictable” value H(x)

- In particular, finding x such that H(x) satisfies some property should take as long as choosing random strings until the property holds

Proofs of work (PoW)

- Puzzle instance defined by a (random) value r
- Solving a puzzle means finding an x such that H(r, x) has some property
  - E.g., t most-significant bits all equal to 0
- If H is "random," then solving a puzzle is hard
  - E.g., expected time 2^t
- Verifying a puzzle solution is easy!
  - Just one hash evaluation

PoW properties

- No better strategy than trying random values
- Progress-free: don’t get closer to a solution the more work you have already done
- Parameterizable: easy to adjust puzzle difficulty

Hash functions in practice

- MD5: 128-bit output length
  - Too short by current standards
  - Collision-resistance broken in 2005
- SHA-1: 160-bit output length
  - Collisions found (using almost 2^{80} work) in 2017
- SHA-256: 256-bit output length
  - Other output lengths also possible
- SHA-3: variable output lengths supported

Nakamoto consensus

Caveats

- Nakamoto consensus is only one example of a blockchain protocol
  - Though it was the one to start the craze…
- Nakamoto’s whitepaper proposed a cryptocurrency (Bitcoin)
  - Useful to conceptually separate the blockchain layer and the cryptocurrency, though technically there is not a clean separation
- Some details have been simplified for the presentation
  - Do not rely on this presentation for low-level details

Nakamoto consensus (setting)

- Completely permissionless!
  - Processors can join or leave the protocol at any time
  - Processors do not need to know identities of all other participants, or even how many there are

- Synchronous communication

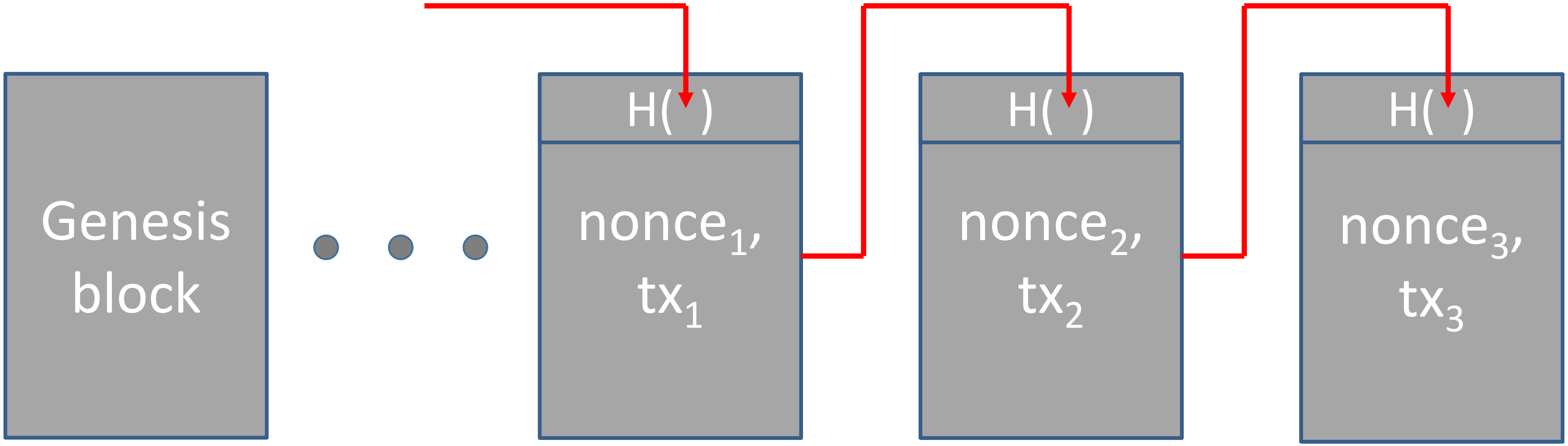
Nakamoto consensus (network)

- Not a fully-connected network
  - Messages are propagated by flooding
- Peer-to-peer network with random topology
  - Low degree
- Nodes can join/leave at any time
  - Drop non-responding nodes after timeout

Nakamoto consensus (protocol overview)

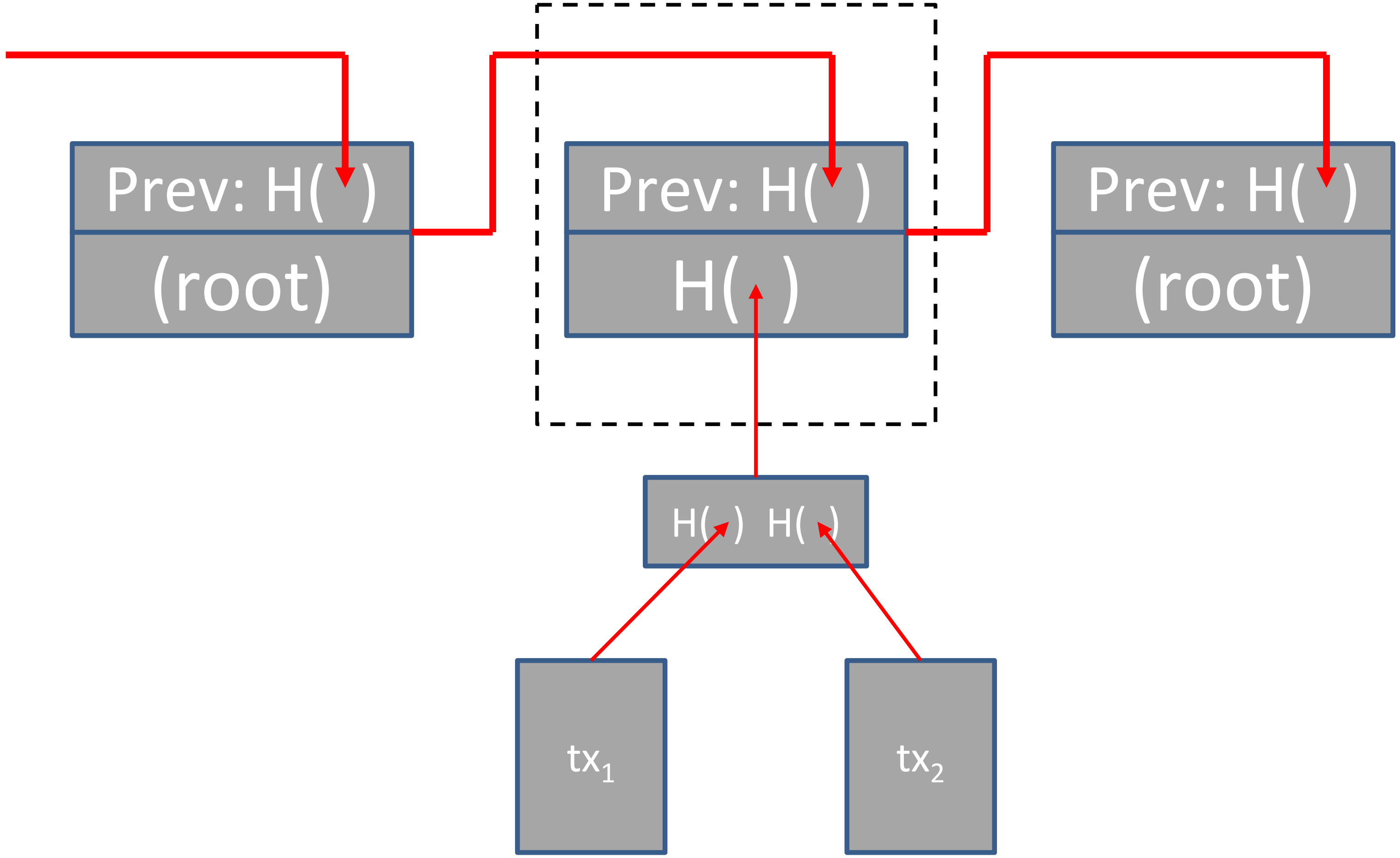
- All processors maintain a linked list data structure (“blockchain”)
  - Initial block is a publicly known “genesis block”
- Processors continually exchange their copies of the blockchain
  - Rule: switch to the longest (valid) blockchain
- When a processor hears about a new transaction, it tries to append a block containing that transaction to its local copy of the blockchain
  - To do so, it must solve a proof of work
  - This is called “mining” a new block
- If successful, it then broadcasts the updated blockchain

Blockchain



Blocks

- In fact, a block can incorporate *multiple* transactions
  - Increased rate for accepting transactions
  - Reduced length of hash chain
- Transactions arranged in a Merkle tree!
- Merkle tree root included in the block
  - Transactions sent separately

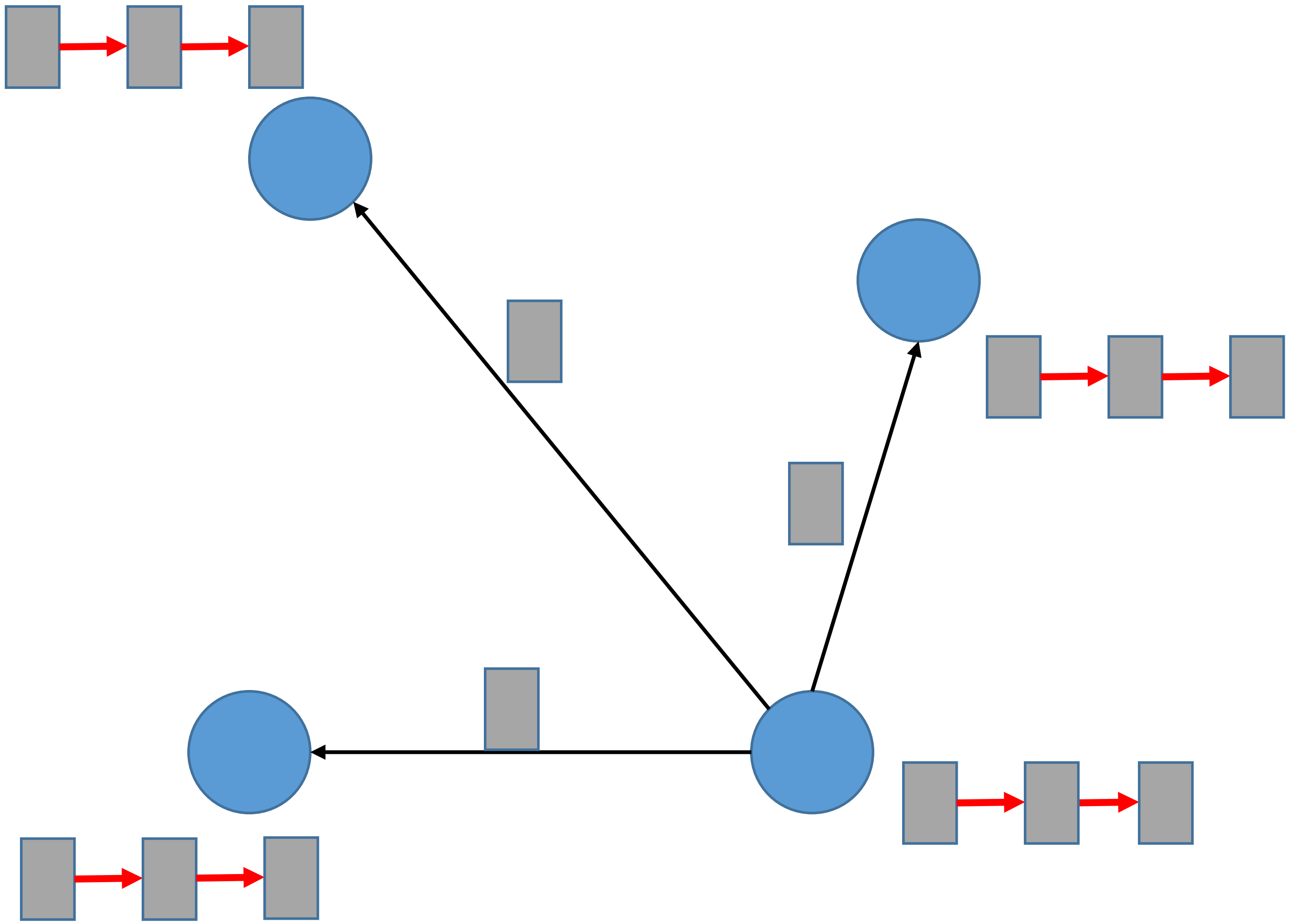

Prev: H( )
(root)
Prev: H( )
H( )
Prev: H( )
(root)
H( ) H( )
tx1
tx2

Blockchain

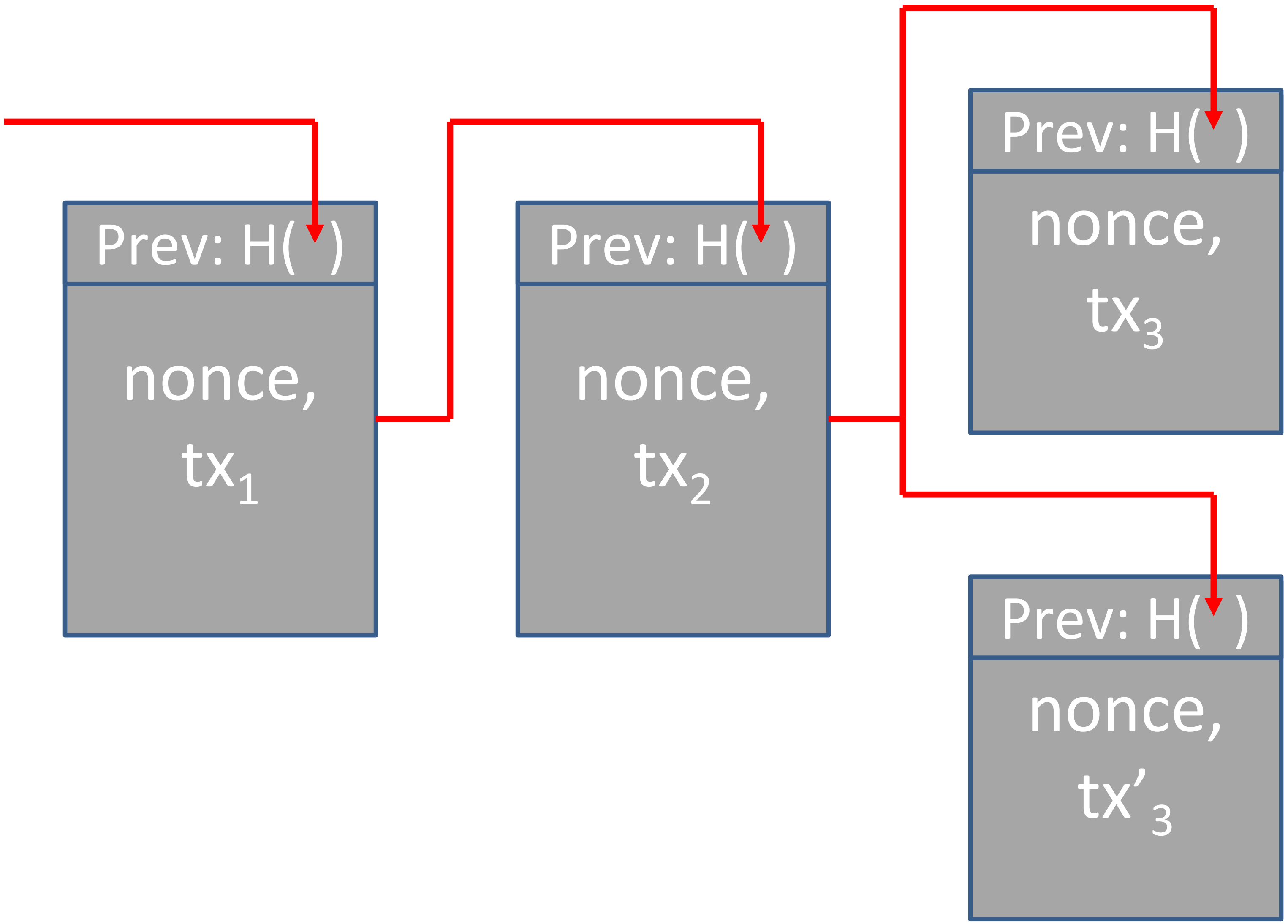
- A blockchain is only *valid* if:
  - The initial block is the genesis block
  - Each subsequent block contains a hash of the previous block
  - Each block contains only valid transactions
    - "Valid" here is application dependent
  - Each block satisfies the "proof of work" criterion described next

Proofs of work

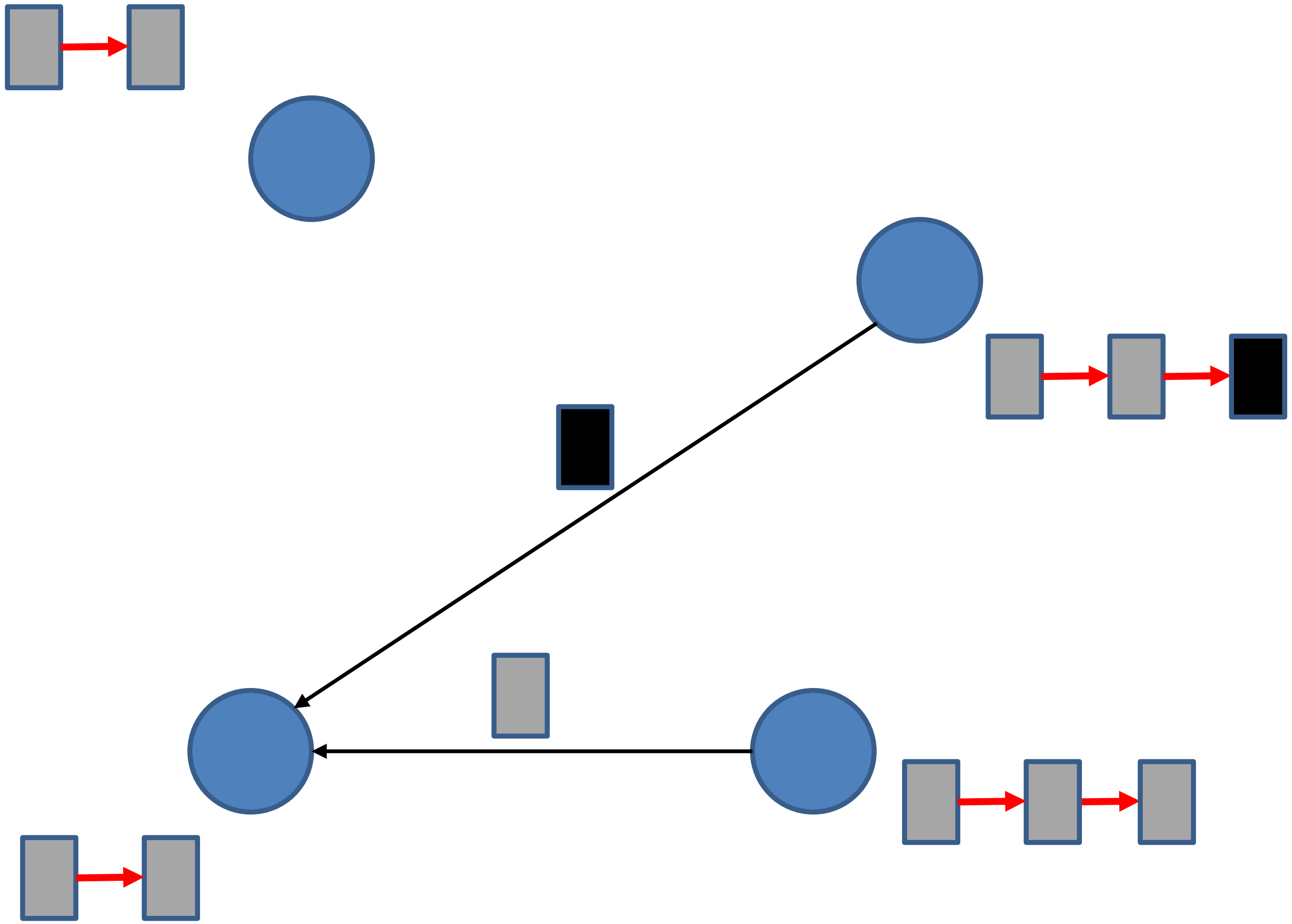
- A block (prev, nonce, data) is *valid* only if H(prev, nonce, data) has *t* leading zeros
  - *t* is a parameter…more later
- Easy to verify validity!
- Note that prev is fixed by the previous block, and data is fixed by the set of transactions a processor wants to include in the current block
  - Repeatedly choose nonce until satisfying the above
  - Expected work 2^t (on behalf of the entire network!) to mine the next block
  - A given processor (or set of processors) mines the next block with probability proportional to its hash power!



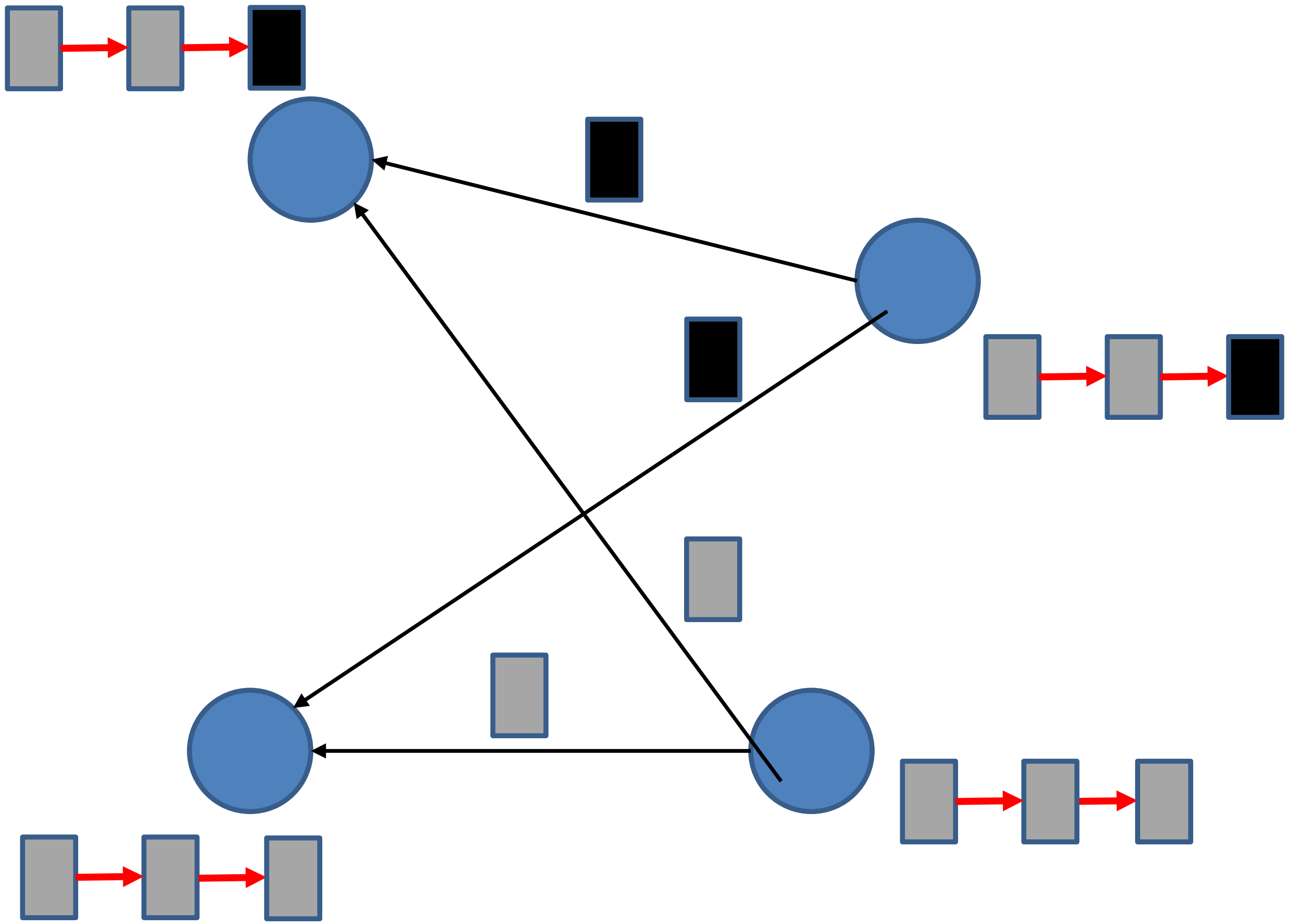
“Forks”



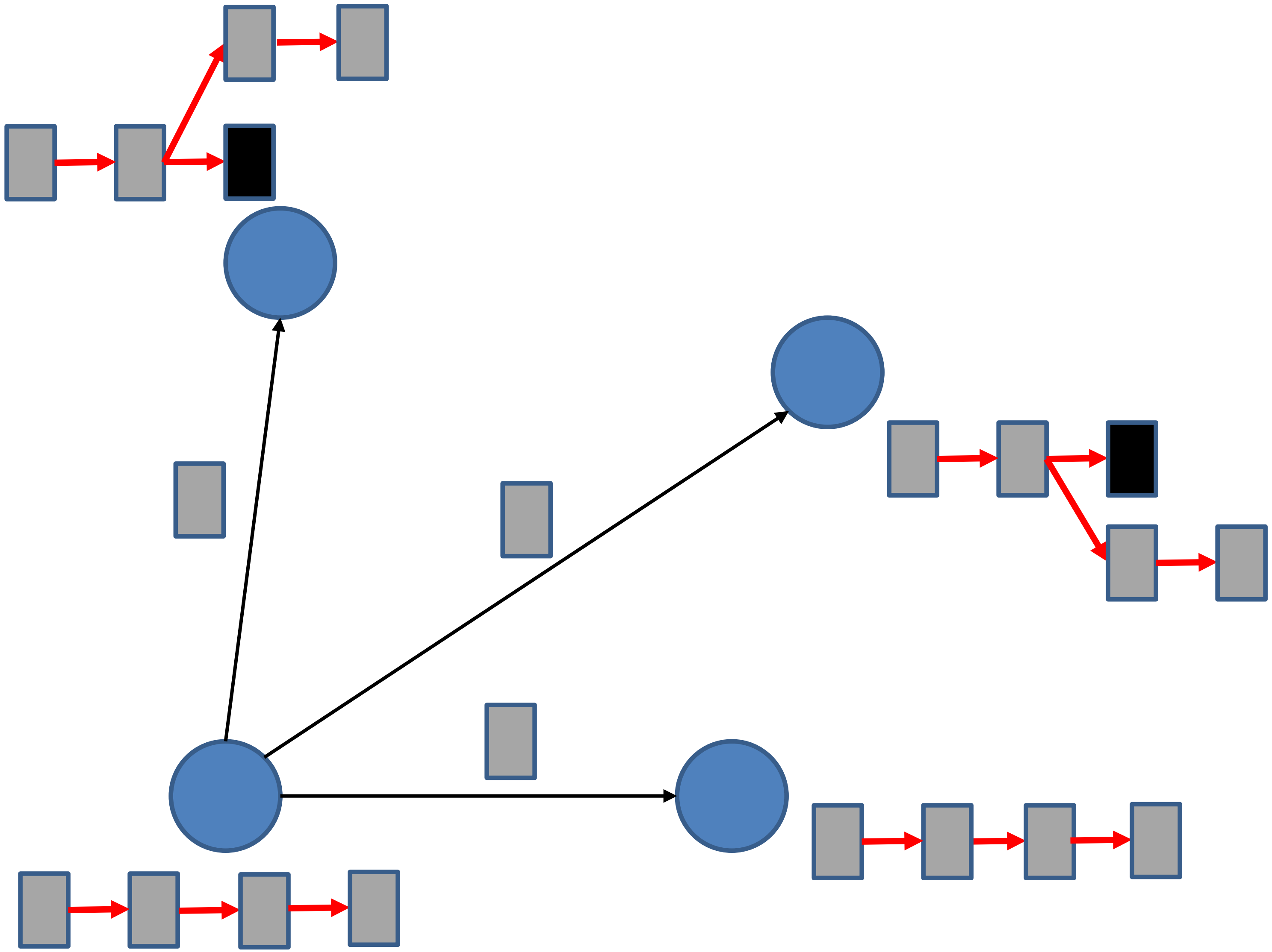
Prev: H( )
nonce,
tx1
Prev: H( )
nonce,
tx2
Prev: H( )
nonce,
tx3
Prev: H( )
nonce,
tx’3



Break ties arbitrarily

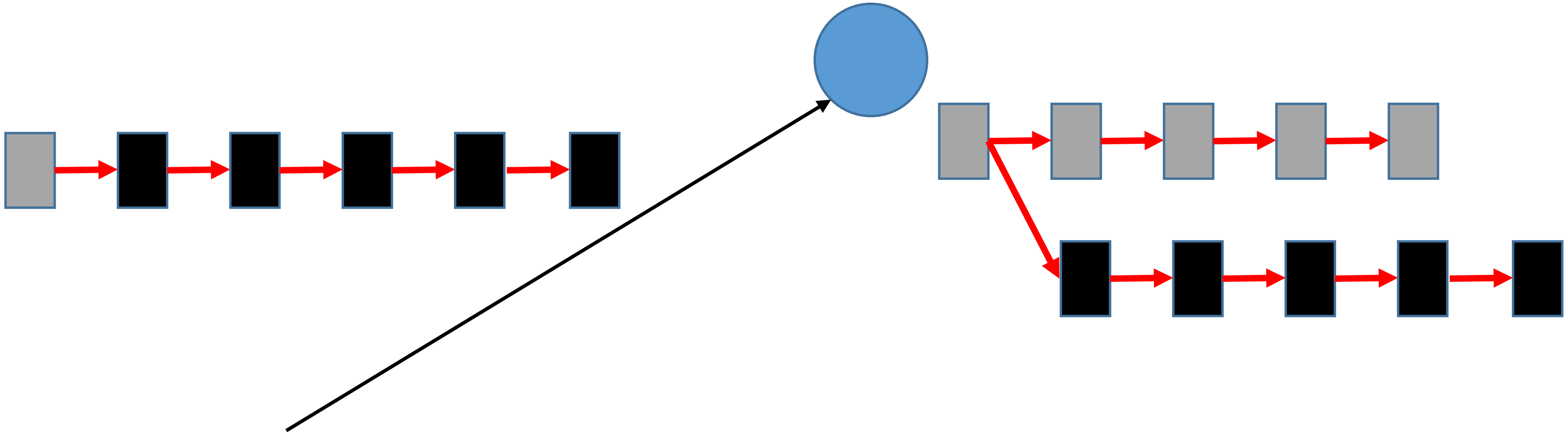


There may be temporary forks (disagreement)!



Committed values

- Blocks in the blockchain can change!
- Values are never truly committed
  - Always possible for there to be a fork, or for the current chain to be overtaken by a subsequent chain



Committed values

- Idea: set parameters such that a long fork is exceedingly unlikely
  - In particular, ensure that the block-mining rate (which depends on the available hash power) is much slower than the block-propagation rate (which depends on the network)

- This ensures that blocks that are sufficiently deep in the blockchain are exceedingly unlikely to ever change

Block-mining rate

- Balance two competing goals
  - Faster mining rate $\Rightarrow$ txs incorporated faster
  - Slower mining rate $\Rightarrow$ better security

- In bitcoin, parameters set so a block is mined every $\approx$10 minutes
  - Much slower than the network propagation rate

- Transactions that are 6 blocks deep are assumed to be committed

Block-mining rate

- Potential problem: when more processors join the network, the hash power increases and so the block-mining rate will increase!
- Solution: recalibrate the PoW difficulty every 2016 blocks (≈2 weeks)
- Network currently performs ~2^{65} hashes per second…

Why does this protocol work? (Informal)

- Sybil attacks are prevented!
  - An attacker can “pretend” to be 1000 different processors…
  - …but its total hash power is fixed
- Changing (transactions sufficiently deep in) the blockchain is difficult!
  - Changing a transaction in a block at depth N requires mining N+1 new blocks

Why does this protocol work? (Informal)

- Assume attacker controls δ-fraction of hash power in the network
- **Liveness:**
  - Assume a transaction is propagated to all (correct) processors
  - Intuition: A transaction will be included in the blockchain when a correct processor successfully mines a block
  - This happens (on average) every $1/(1-\delta)$ blocks
- **Agreement:**
  - Extremely unlikely for two forks to grow at the same rate forever
    - Eventually, one overtakes the other
  - Longer chain eventually adopted by everyone

Why does this protocol work? (Formally)

- Garay, Kiayias, and Leonardos: “The Bitcoin Backbone Protocol: Analysis and Applications” (2014)
- Assumptions:
  - Fixed hash power, PoW difficulty
  - Synchronous network
  - Assumed upper bound on fraction of hash power controlled by an attacker (precise bound depends on various factors)
- These assumptions have been relaxed in subsequent work

Why does this protocol work? (Formally)

- **Agreement:**
  - Say a correct processor has some block B at depth n
  - When any correct processor ever has a block at that position, at depth n, then it will be the *same* block, except with probability $2^{-O(n)}$
- **Liveness:**
  - If all correct processors learn about some transaction, then it is eventually incorporated into the blockchain of every honest processor (at depth n)

Drawbacks of the protocol

- Permissionless protocol
  - Do you trust the majority or not?

- Computationally wasteful
  - Proofs of work are expensive, environmentally unfriendly
  - Storage requirements also a concern

Outline: Part 2

- Digital signature schemes
- From consensus to cryptocurrency
- Bitcoin

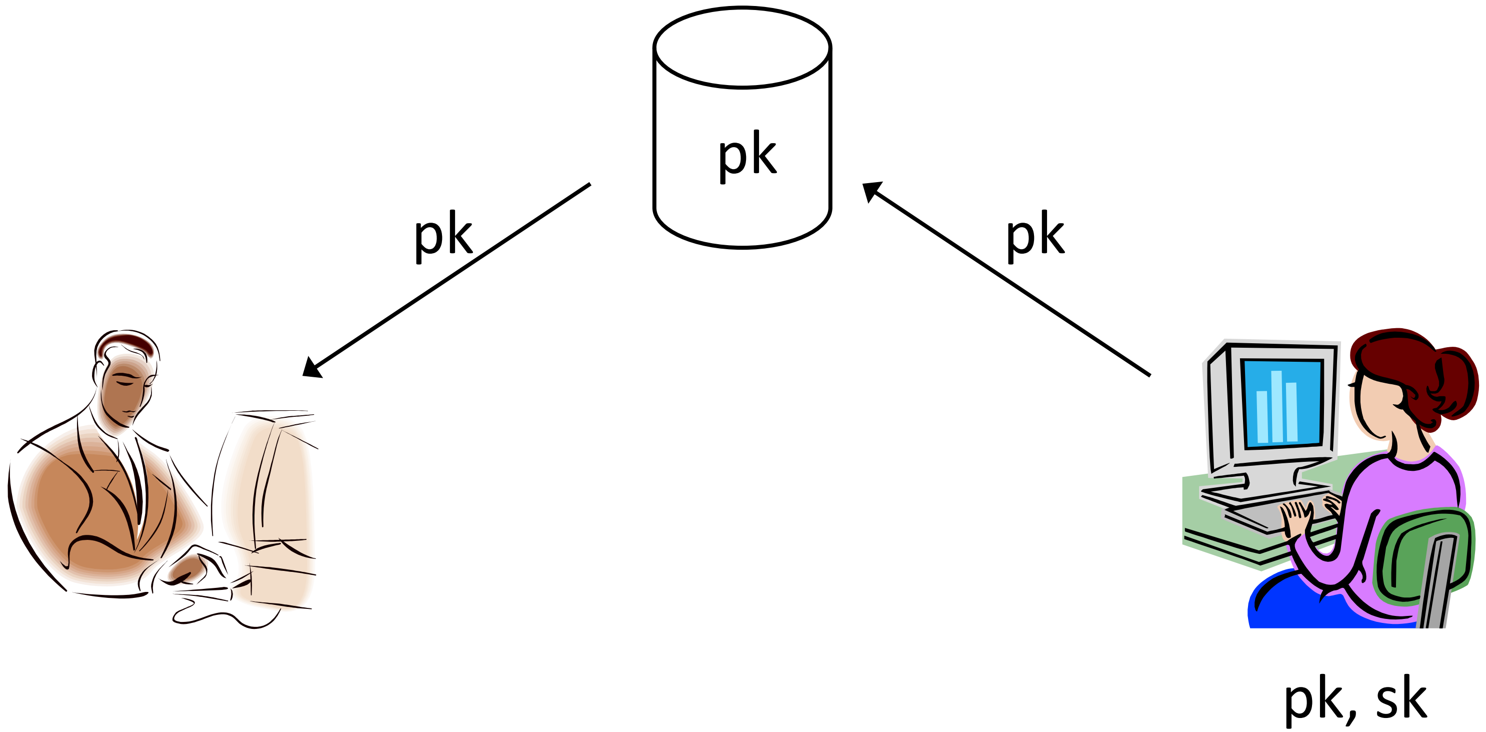
Signature schemes

Signature schemes

- Signature schemes provide *message integrity* in the public-key setting

The public-key setting

- One party generates a *pair* of keys: public key pk and private key sk
  - Public key is widely disseminated
  - Private key is kept secret, and shared with no one
- Private key used by the party who generated it; public key can be used by anyone else
- Security must hold even if an attacker knows pk



Assume it is possible to get a reliable copy of pk

Message integrity

- Ensure that a message originated from the claimed party
- Ensure that a message was not modified along the way

Security

- Even after observing signatures on multiple messages, an attacker should be unable to *forge* a valid signature on any *new* message

Replay attacks

- Note that *replay attacks* are not prevented by signature schemes
  - No stateless mechanism can prevent them
- Replay attacks are a potential real-world concern
  - Must be dealt with at the application level

Signature scheme in Bitcoin

- ECDSA signatures used
  - Intended to provide 256-bit security
- These are based on *elliptic curves*, and are relatively short
  - Public keys are 256-bits long
  - Signatures are 512-bits long

From blockchain to cryptocurrency

Cryptocurrency?

- Assume a replicated state machine protocol that allows processors to maintain a distributed ledger
  - For now, exact details of the consensus protocol do not matter
  - Later, we will assume Nakamoto consensus is used

- How can that be leveraged to build a cryptocurrency?
  - With Bitcoin as the running example

Key ideas…

- Use the ledger to keep track of account balances
- Transactions are used to transfer funds from one account to the other

- Need to ensure that only authorized parties can initiate a transaction
- Use digital signatures!
  - Accounts are identified with public keys
  - Owner of account knows the associated private key
  - Transactions are signed statements transferring funds

Example…

time

pk_A: 20, pk_B: 0, pk_C: 0

Transfer 10 from pk_A to pk_B, s_A

Transfer 5 from pk_B to pk_C, s_B

Transfer 5 from pk_B to pk_A, s_B

Drawbacks

- Need to handle replay attacks
  - Could detect by searching through the entire previous state…
  - …but this would be inefficient

Alternate approach

- Track coins, not accounts
- More precisely, track **unspent transaction outputs** (UTXOs)
  - Identified by some index
  - Associated with a public key
  - Owner of the UTXO knows the corresponding private key
  - Only someone who knows the private key can use the UTXO
- When used, a UTXO must be spent in its entirety
  - Any given UTXO is used only once
  - But different UTXOs may be associated with the same public key

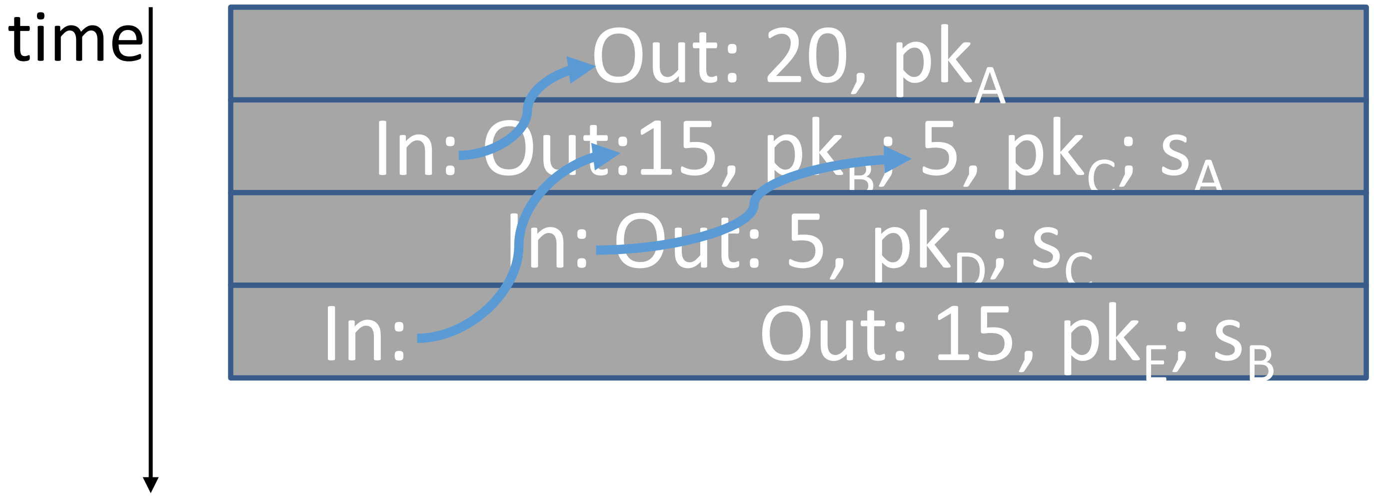
Note…

- Public keys need not be associated with any real-world entity
- Rather, a public key is associated with a UTXO; only someone who knows the associated private key can spend that UTXO
- Keys can be disposable or long term…

Transactions

- A transaction uses old UTXOs and creates new ones
  - Old UTXOs = “inputs”
  - New UTXOs = “outputs”
  - Require

$$\text{sum of inputs} \geq \text{sum of outputs}$$


time
Out: 20, pkA
In: Out:15, pkB; 5, pkC; sA
In: Out: 5, pkD; sC
In: Out: 15, pkE; sB

“Change”

- Since a UTXO must be used in its entirety, can pay any balance back to the same public key
  - New UTXO, just same public key
- Can equivalently be a fresh public key with private key known by the same person

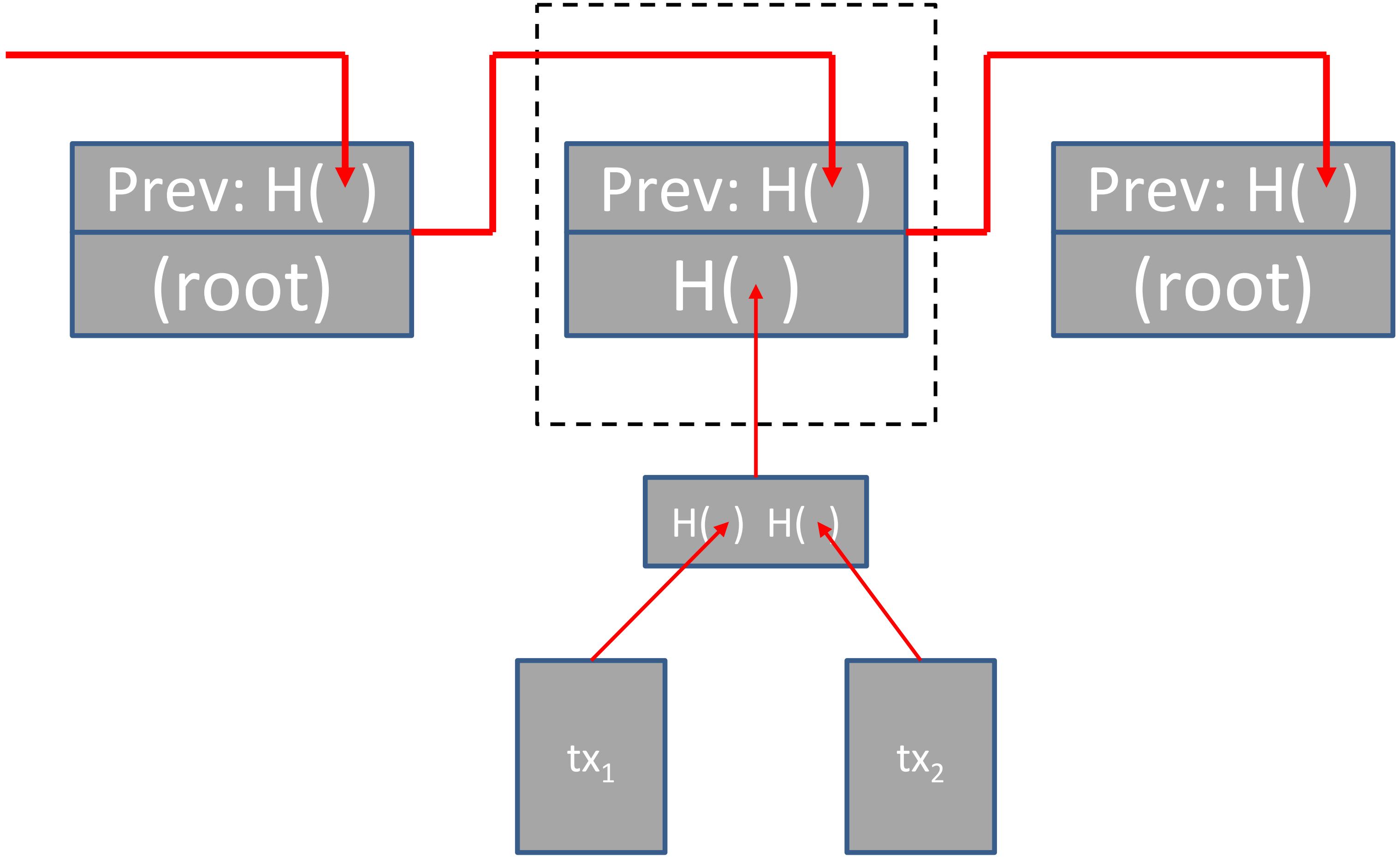
Transaction validity

- All processors keep track of current set of UTXOs at all tmes
- Verify that inputs to a transaction all correspond to a (different) UTXO
- Verify correctness of signatures
- Verify that sum of outputs ≤ sum of inputs

- Delete UTXOs used as inputs; create UTXO for each output

Blocks

- Blocks incorporate multiple transactions
- Transactions arranged in a Merkle tree
- Validity of transactions determined one-by-one, in order


Prev: H( )
(root)
Prev: H( )
H( )
Prev: H( )
(root)
H( ) H( )
tx1
tx2

Transactions

- Transactions in Bitcoin can be more complex than described so far
- Bitcoin provides a simple, stack-based scripting language; input/ouput UTXOs specify scripts
- Verifying a signature with respect to public key is just one example of a script

Where does money come from?

- How are funds initially allocated in the system?
- How can new money be created?

Bitcoin’s solution

- New coins created when new blocks are mined!
- Miners incorporate a special “coinbase” transaction in each block they mine
    - Single input, pointing to nothing
    - Single output (nominally miner’s public key)
    - Value is the current block reward

Block reward

- Determined as part of the Bitcoin protocol
- Started at 50 BTC; halves every 210,000 blocks (~4 years)
- Finite supply of 21 million BTC!

Transaction fees

- Transactions can also specify transaction fees
  - Using funds from the input UTXOs
  - In fact, the fee is just (sum of the inputs –sum of the outputs)

- When a miner mines a block, the value of the coinbase transaction also includes the fees for all the transactions included in that block

Bitcoin incentives

- For the blockchain to be secure, need hashing power in the network to be much greater than hashing power of any attacker
- But why should people participate in the protocol at all?

- Block reward encourages participation!
  - Improves agreement guarantee
- Block reward encourages increased hash power
  - E.g., more investment in computational resources

Bitcoin incentives

- What incentivizes miners to include transactions in blocks?
  - Effort involved in learning about transactions
  - Effort involved in incorporating transactions into blocks
- Transaction fees provide the incentive!
  - Improves liveness guarantee

Game-theoretic considerations

- Some work showing ways of “gaming” the Bitcoin protocol
  - E.g., selfish mining
  - Mining pools
  - See my talk tomorrow

What did I leave out?

- The Bitcoin ecosystem
  - Mining hardware
  - Mining pools
  - Bitcoin wallets
  - Efficiency aspects (e.g., off-chain transactions)

What did I leave out?

- I have only described *one* blockchain protocol and *one* cryptocurrency
  - There are many, many more out there!
- From a blockchain to a global computer ("smart contracts")
- Privacy aspects
- Alternate mechanisms (e.g., proof of storage, proof of stake, …)

Thank You!