BLOCKCHAIN

AT BERKELEY

Workshop 5: Data Structures, Consensus Algorithms, Bitcoin Scripting

Tobias Boelter and Max Fang

IoT

BLOCKCHAIN
AT BERKELEY

IOTA and Ripple

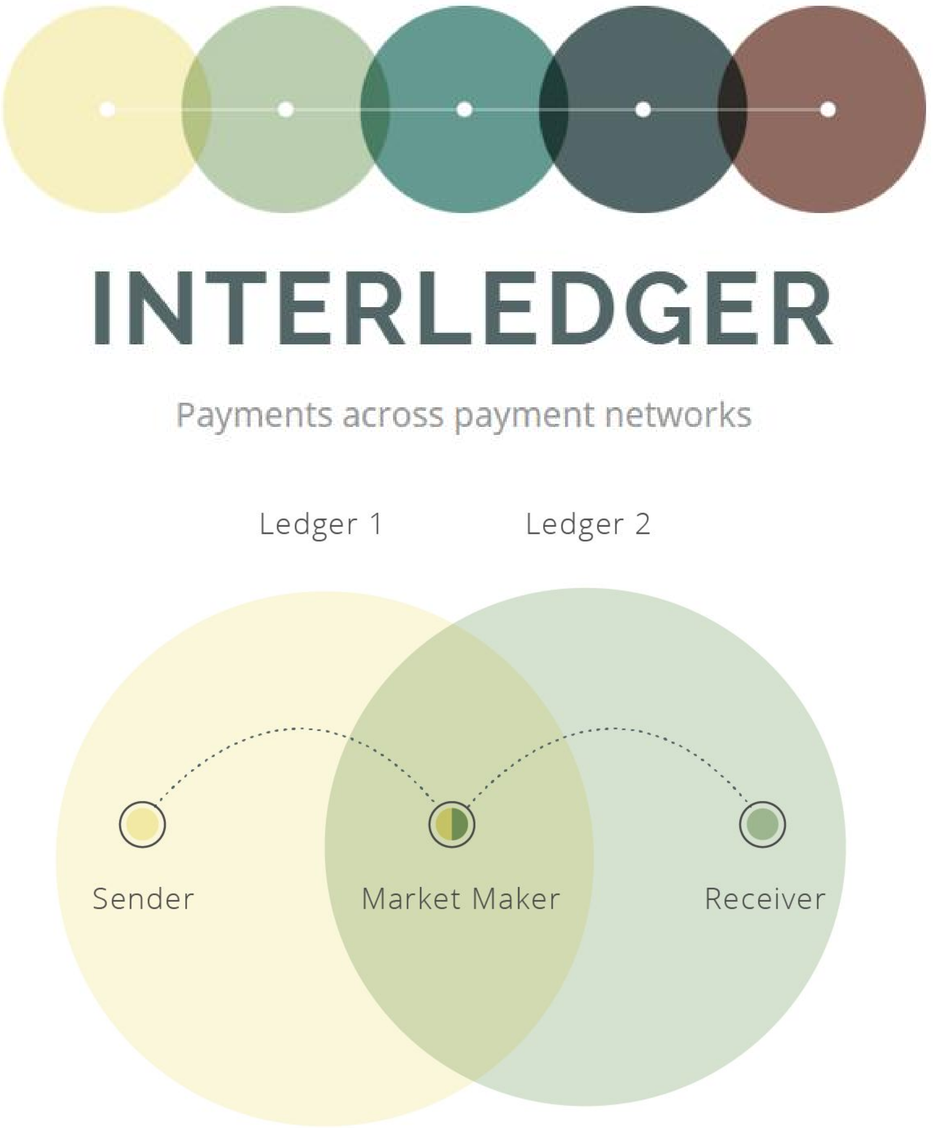
ripple

What is Ripple Trying to Do

- Exchange money like we exchange information
- Efficient Bank Settlements
- Distributed Currency Exchange
    - Currencies are debt instruments that exist as balances in accounts
- Payments can be made through existing fiat currencies or the native cryptocurrency - XRP
    - XRP is the third largest cryptocurrency behind Bitcoin and Ether
    - XRP is not a debt instrument, so it has no counterparty risk

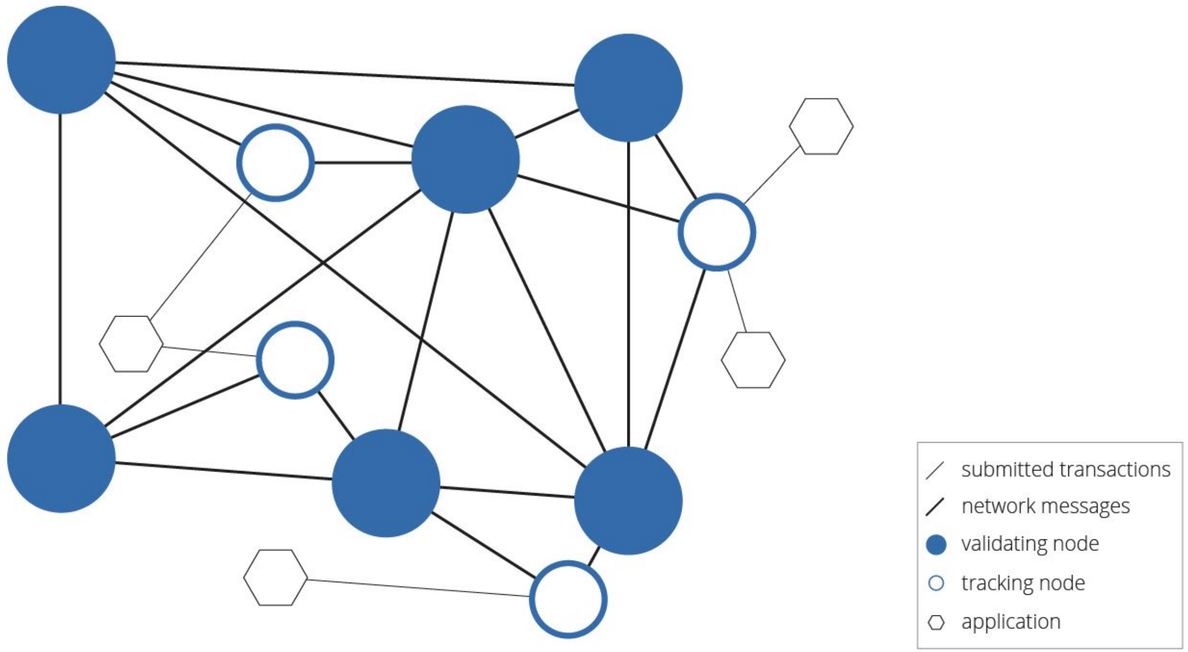
Interledger

- Secure transfers between different ledgers
- Anyone with an account on two different ledgers can link them together
- No need for shared trust
- Ripple vs Interledger:
  - Ripple is a ledger for currency transfers and bank settlements
  - Interledger is a protocol that connects existing ledgers
- Metaphor using email



How are they unique

- Shared ledger
  - Account settings and balances
  - Offers to buy and sell currencies
  - Time stamp and settings of network
  - New ledger produced every few seconds with processed transactions
- Iterative consensus
  - Nodes track and validate transactions
  - Nodes go through multiple *iterative* rounds of of proposals
  - When a supermajority (80%) agree on validity, the transaction is added to the ledger

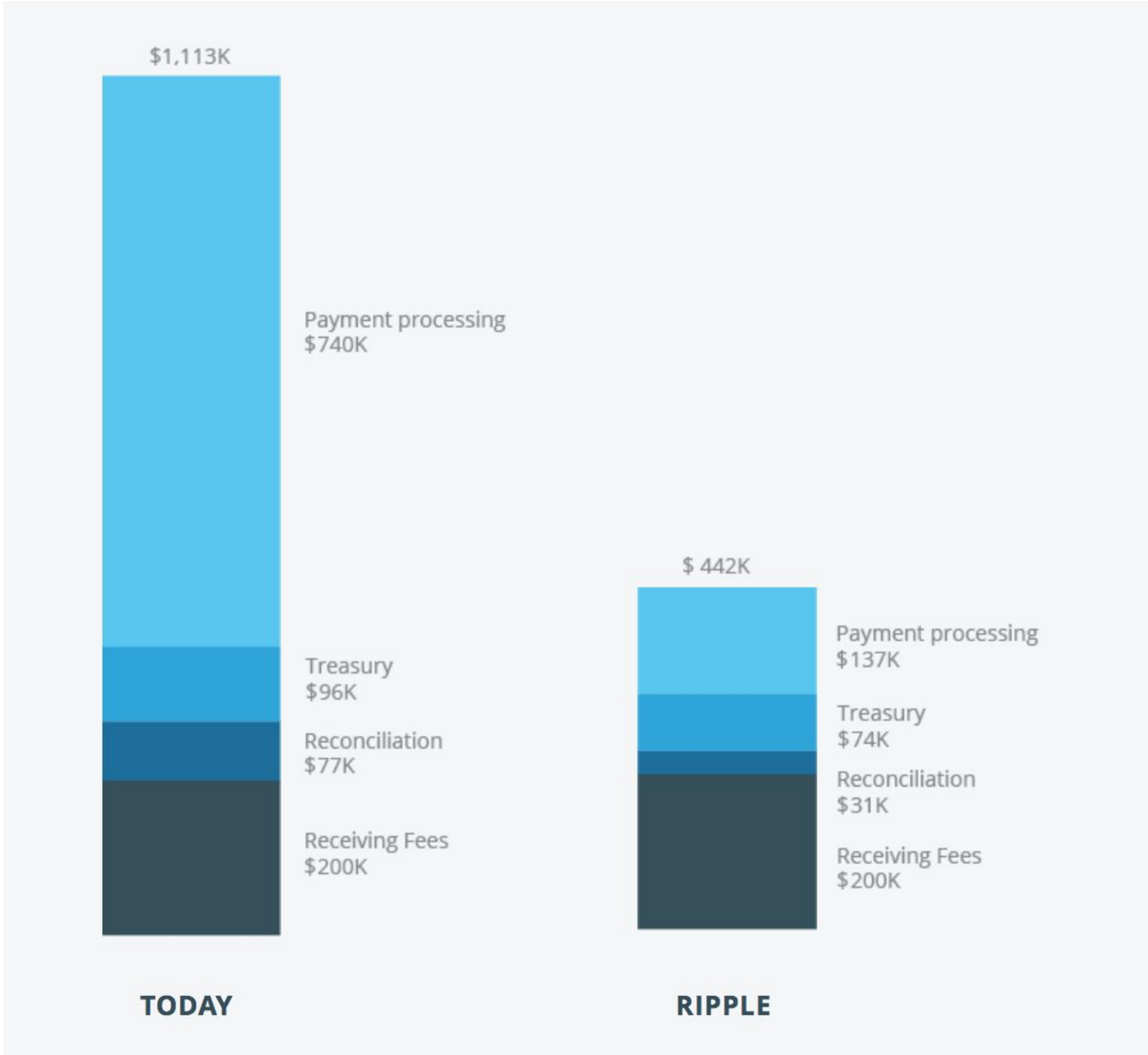


Some Major Investors

- Andreessen Horowitz
- Google Ventures
- Accenture
- Standard Chartered
- Santander InnoVentures

Concrete Use Cases

- Corporate Disbursements
  - Ripple acts as a middleman
  - Allows banks to distribute high volumes of low value payments
- Low Cost Remittances
- Efficiency over existing systems that rely on long chains of banks
  - 3-5 seconds




IOTA

IOTA’s Mission

- Economic backbone of IoT
  - ➔ Zero-fee transactions and microtransactions utilizing “Iota Tokens”

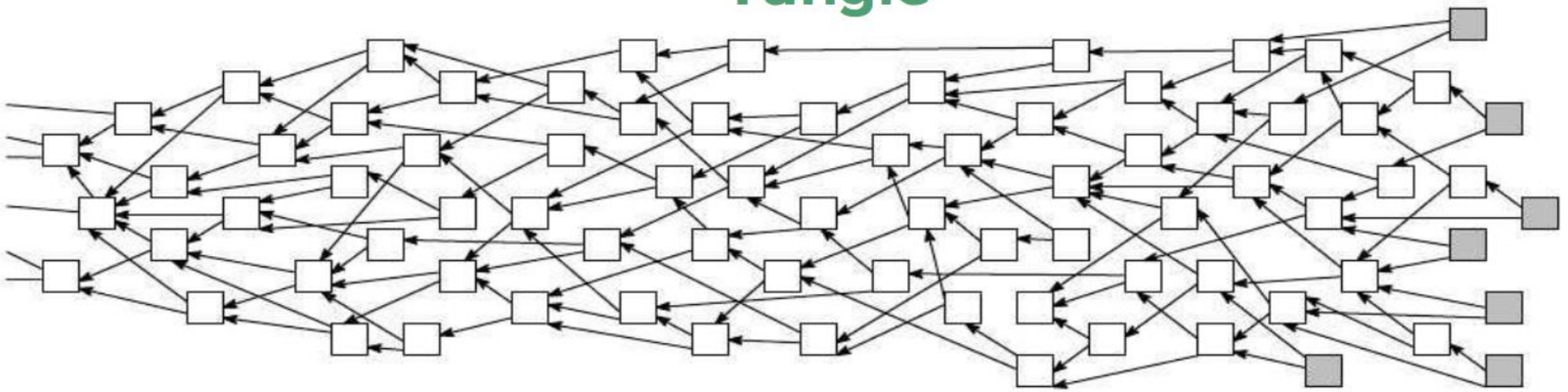
- IOTA wants to be the premier real-time payment system for IoT devices
- IOTA relies on interoperability
- Utilizing “the Tangle”

How Does IOTA work?

What is Tangle?

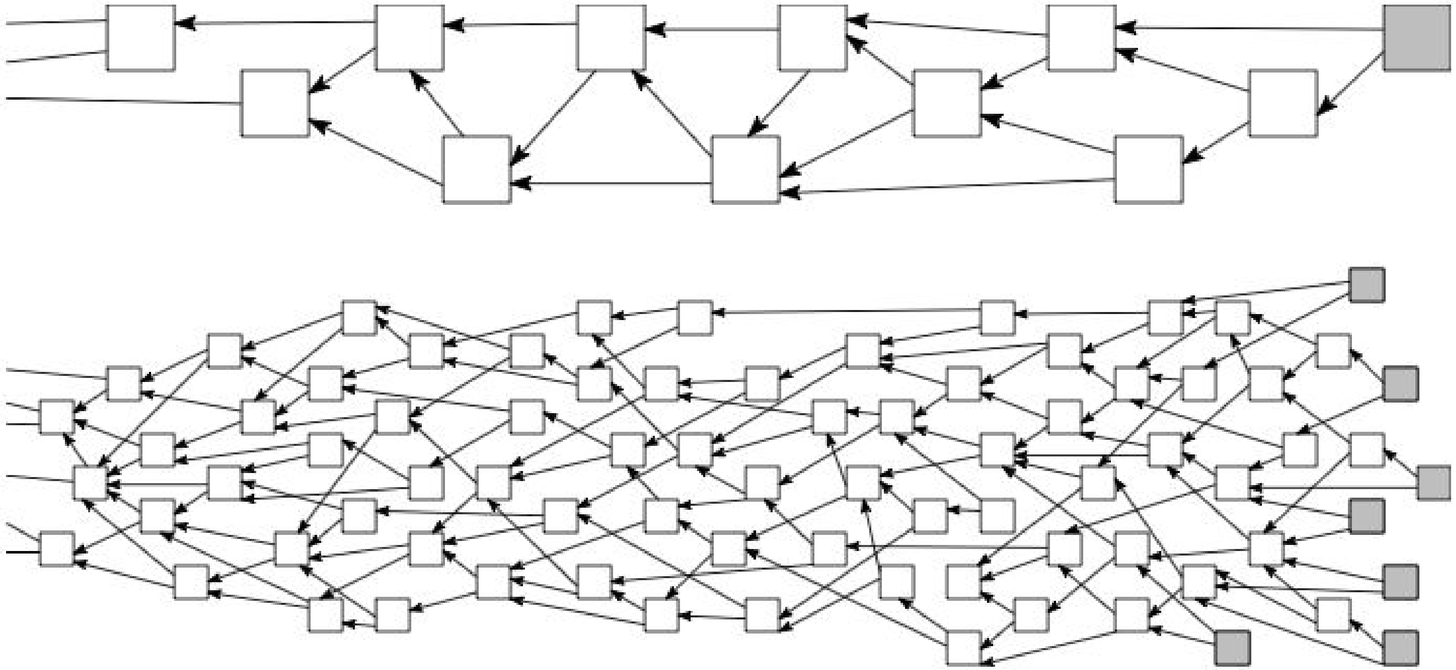
- Bundles all transactions in a tangle
- Validates transactions only on-demand
- Completely self-regulating, consensus no longer decoupled
- Uses low overhead Proof-of-Work to prevent spam

Tangle



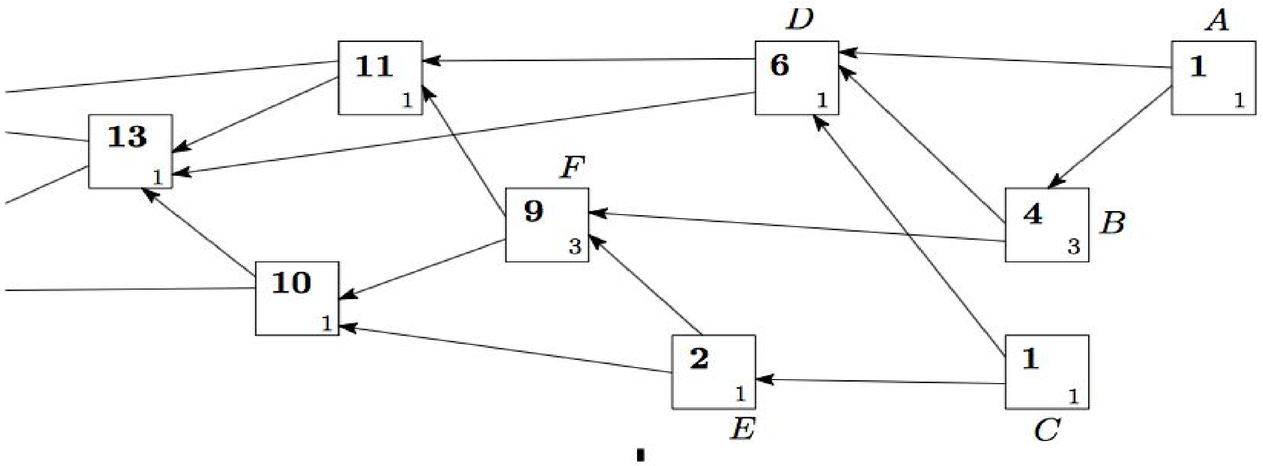
How Tangle Works

- Instead of the blockchain there is a DAG (directed acyclic graph) called tangle.
- Nodes in this graph are transactions and have directed edges to represent approvals of other transactions. Each new transaction approves two previous transactions.



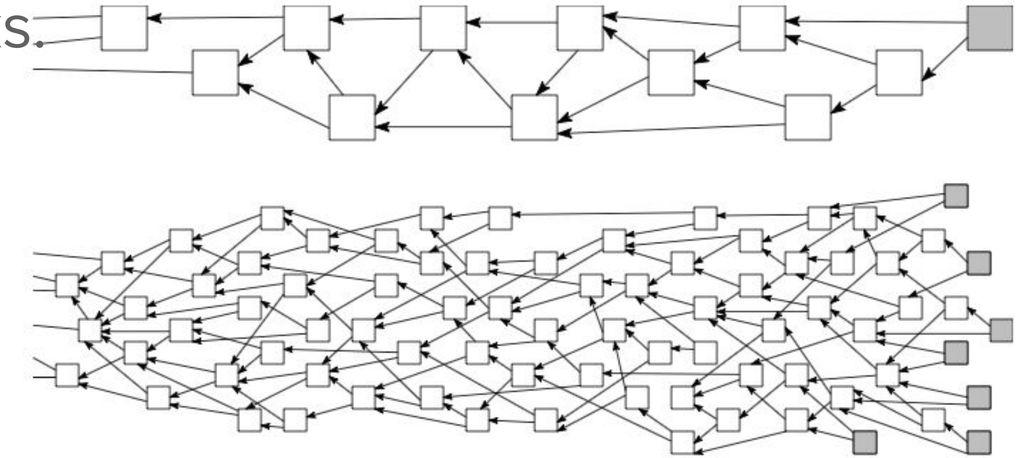
How Tangle Works

- Every transaction has weights proportional to work done.
- Set of completed transactions in the network are transactions in the main/largest tangle
- Other terms
  - Cumulative weight: weight of transaction plus sum of weights of all approving transactions
  - Tips: unapproved transactions i.e. A and C below



How Tangle Deals with confirmations/consensus

- It is assumed that the nodes check if the approved transactions are not conflicting and do not approve (directly or indirectly) conflicting transactions. The idea is that, as a transaction gets more and more (direct or indirect) approvals, it becomes more accepted by the system
- Naive tip selection algorithm: choose two random tips and approve them
- Issue: Nodes aren’t encouraged to approve newer transactions which we want. Also susceptible to different attacks.



How Tangle Deals with confirmations/consensus

- Final algorithm: MCMC tip selection algorithm
- Uses random walks in the tangle and tends to tips/regions with high cumulative weight
- Implications: uses the assumption that the main tangle has more hashing power/average transaction cumulative weight.
- Result: new tips are tended to for approval and parasitic chains are rarely jumped to/worked on.
- Quantum resistant?

Interoperability from a protocol standpoint

- IOTA is NOT a Blockchain-killer IoT swiss army knife
- IOTA is flexibly designed so that it can communicate with other blockchains: Ethereum, Bitshares, Nxt, Bitcoin
- IOTA as an Oracle, checkpointing transactions for other platforms.

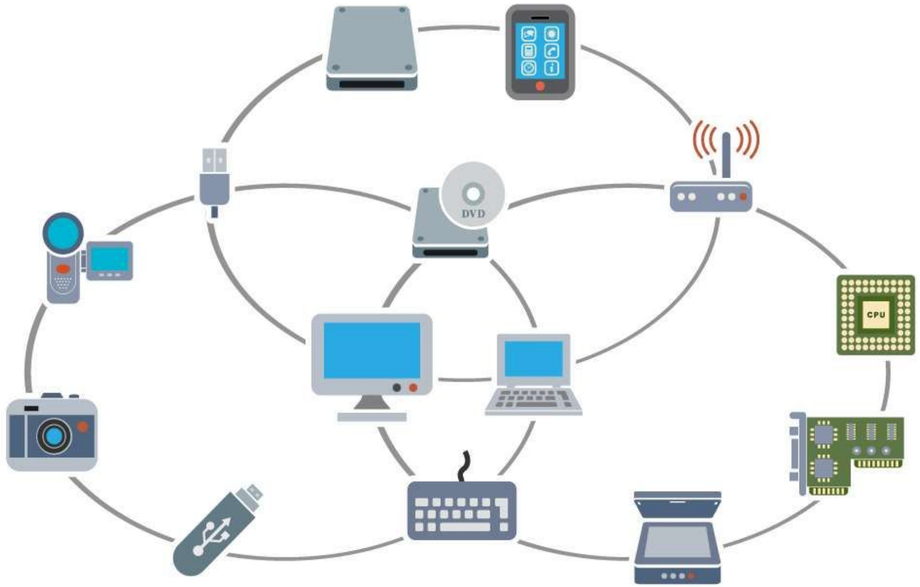
“We believe that there is no ‘universal solution’ to all the problems that ledgers can solve, and thus interoperability is required, much like it is in IoT.” -David Sønstebø, IOTA co-founder




Interoperability from a connection standpoint

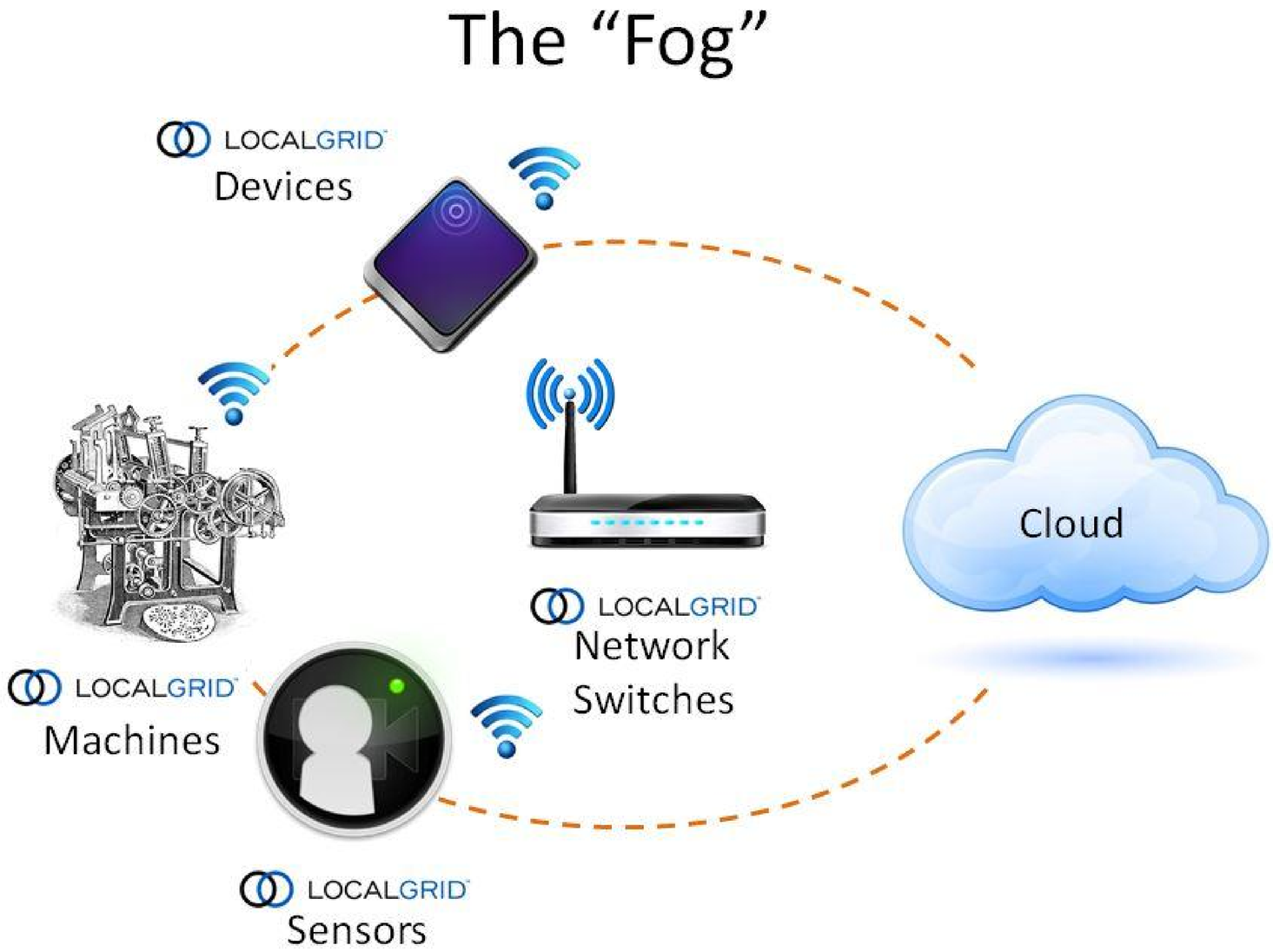
-Taking advantage of tangle/blockchain distributed nature allows transactions to flow even if main net goes down.

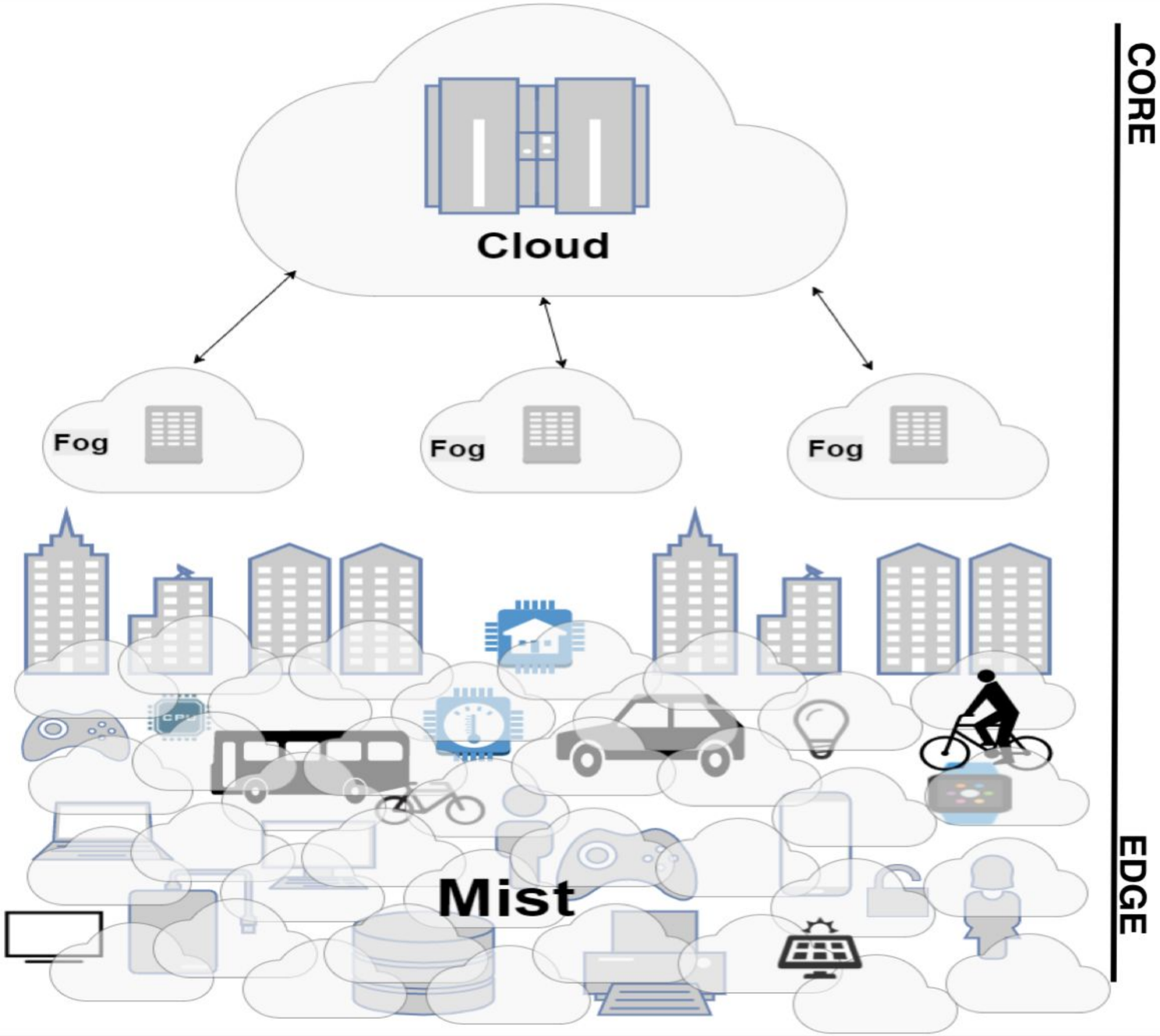
- How: IOTA Tangle- In the event of the main internet being unavailable, communication via wifi/bluetooth/dash7/zigbee/lifi/ keeps transactions flowing uninterrupted.
- While down, local nodes create local tangles.
- All nodes re-tangle without being dependent on a central node to validate them.



Fog Versus Mist

- As IoT scales, we will see the emergence of what is called the ‘Fog,’ which is essentially a distributed ‘Cloud.’
- When it comes to the absolute edge, we’ll approach what we call ‘Mist,’ where you literally have the computational/storage resource either embed into the device itself or in the extreme proximity of the device that needs this technological resource.



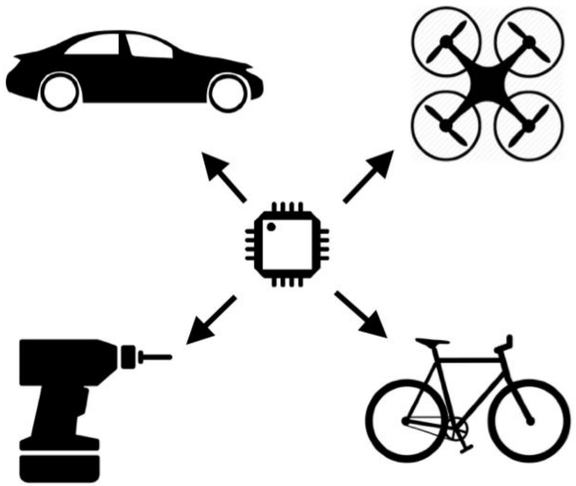


- All those **Mist** clouds can communicate and transact directly through **IOTA**

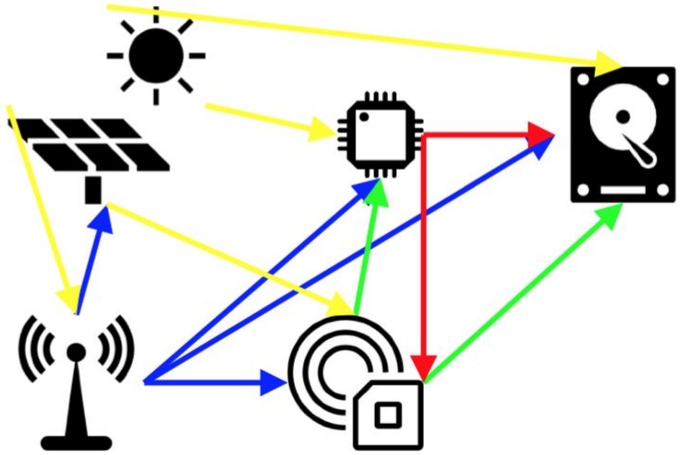
Why IOTA

- Micro and Nano transactions are possible opening up new business possibilities for companies. (NO FEES!)
- Data transfer through Tangle is fully authenticated and tamper proof.
- Everything as a service
- Anything that needs a scalable ledger

Anything with a chip in it can be leased



Devices trade resources among each other



USE CASES

- Other use cases include:
- Compensated storage sharing
- Compensated bandwidth sharing
- Real-time data trade from any kind of sensors and even wearables
- Autonomous ordering (fridge ordering groceries or machine ordering new parts)
- Automated insurance claims (e.g. based on the weather)
- Automated parking meters
- Automated charging stations for EV cars
- Self-discipline / Rewarding wearables
- voting in terms of E-Governance

BLOCKCHAIN
AT BERKELEY

Supply Chain Team Presents: Chain (Platform)

What is Chain?

> Chain is a blockchain platform aimed to streamline the issuance, ownership and flow of digital assets.

Digital Assets on Chain Core

- Financial Instruments
  - Asset ID
  - Corresponds to an Issuance Programs
- Cryptographic Keys
  - Issuance Key is linked to public Asset

1

Issuer Signs an Issuance transaction

2

Issuer Specifies Details of Transaction

3

Transaction is Posted

How Chain Works - Asset Creation

- Create custom assets, defined by:
  - Alias
  - Quorum
  - Definition
  - Tag

- Write “issuance program” and “control program” for the asset
  - Rules for issuing and spending asset
  - Multi party
  - Smart contract
  - Set *who, when,* and *how* of spending

How Chain Works - Transactions

- There are two ways to save global state
  - Account balances
  - Track all changes (transactions!)

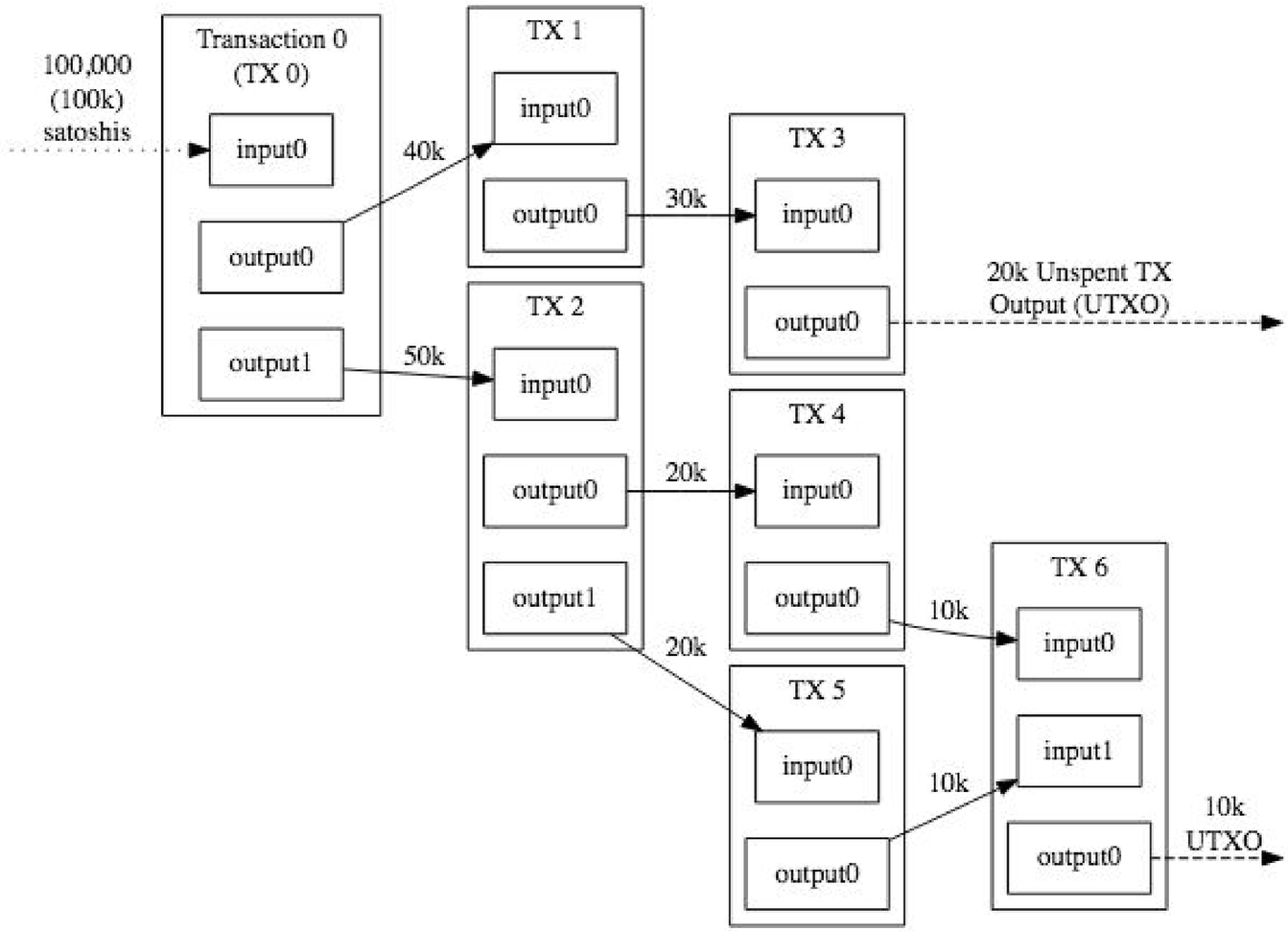
- Ethereum tracks account balances
  - e.g. Max has $10 in his account
- Chain (and Bitcoin) track transaction outputs
  - Asdf e.g. Max has $5 from Charlie and $5 from David

- Transactions: the result of some modification to the blockchain
  - Issuing new units of asset
  - Transferring assets

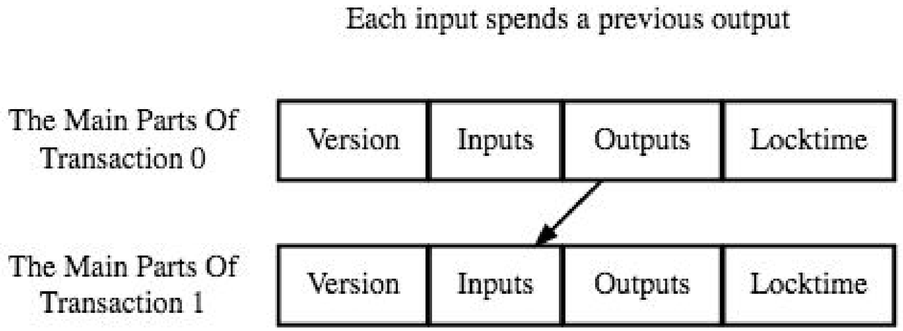
Basic Concepts - Transactions

Source: Bitcoin Developer Guide

- Maps inputs addresses to output addresses
  - Outputs can only be spent once
- Typical tx: one input, two outputs
- Fees are implicit

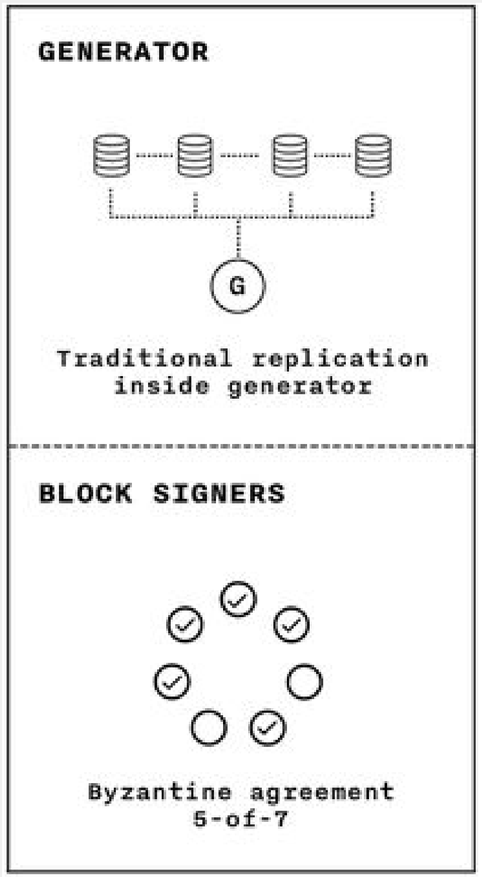


Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin



Each output waits as an Unspent TX Output (UTXO) until a later input spends it

How Chain Works - Consensus



- Program arbitrary consensus programs (e.g. proof-of-work)
- Default: federation of block signers
- Sign blocks based on singular block generator
- Higher trust level necessary
  - Block generator can lock network
  - Produce artificially slow timestamps
  - Signatories created by creators of the blockchain

How Chain Works - Security

Direct Asset Ownership

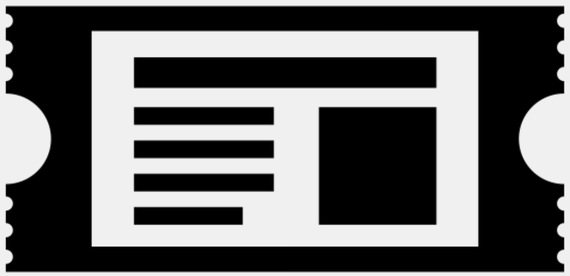
Privacy

Consensus Security

Local Policy

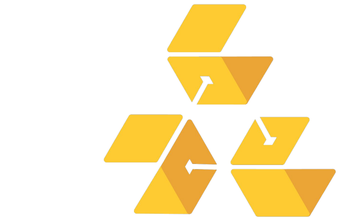
Compact Proofs

How the Team can Implement Chain



How Chain is a Viable Solution

- Asset creation
  - Easy to create new ‘currencies’ (tickets)
- Much of supply chain is based on money movement
  - Facilitated through Chain
- Easy tracking of assets
  - Opposed to ethereum
  - Easy auditing
- Well documented/tested
  - Visa B2B connect

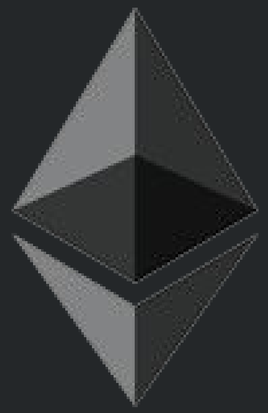


Blockchain Platforms

Reimagining Ethereum

Ethereum:

A decentralized platform that runs smart contracts



ethereum

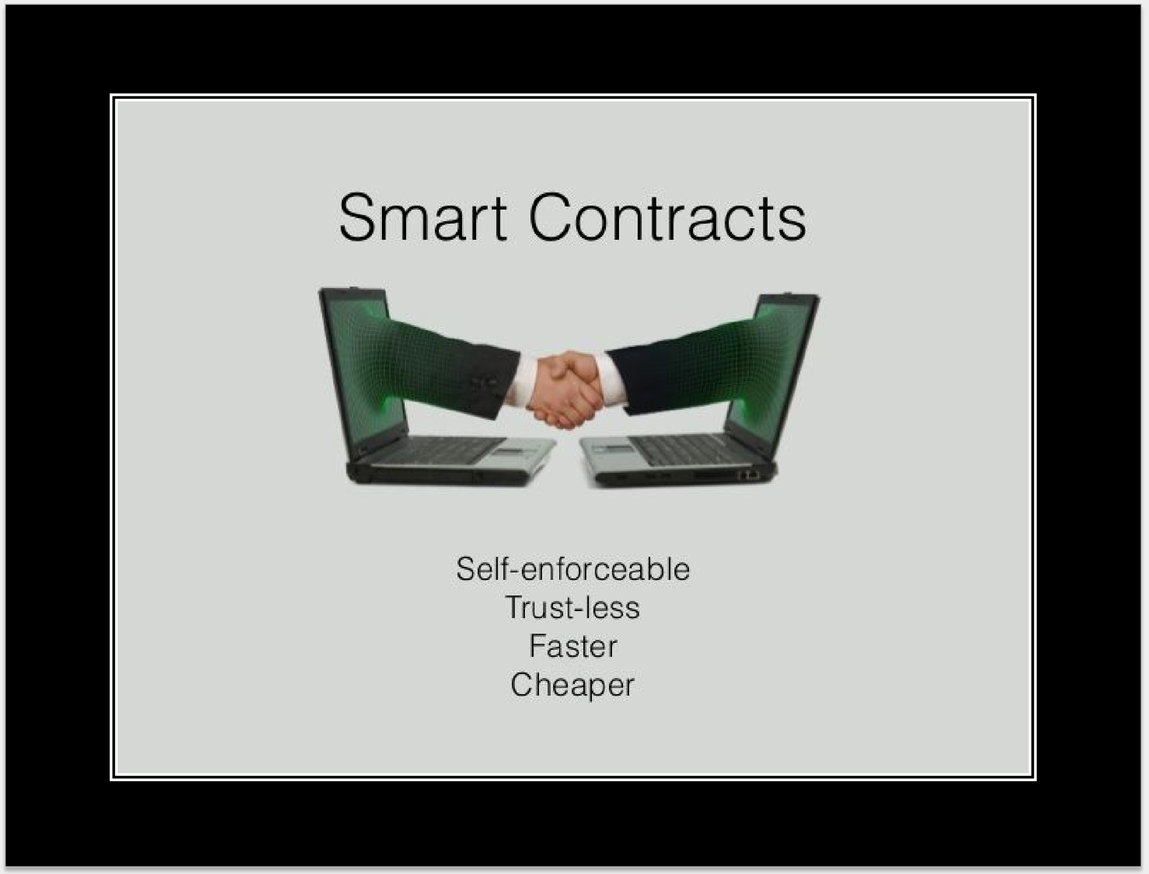
Homestead Documentation

What is Ethereum?

- Cryptocurrency popular for **smart contract** development
- Fast block time verification
- Allows users to deploy own “tokens/cryptocurrencies” on Ethereum blockchain
- Account system rather than UTXO system as in Bitcoin

Why Prefer Ethereum?

1) Bitcoin has very simple scripting language
2) Large public community to interact with
3) Versatility, permanence, and automation of transactions
4) Fast block confirmation times



Possible Problems

- Bugs in contract allow for direct monetary theft (rather than merely information)
- Account-based system loses serialization, makes tracking ether more difficult
- Fast block confirmation time leads to more uncles/aunts
- Malicious/buggy contracts
  - Can we entirely trust humans to write perfect and secure code? Ex: The DAO

Proof of Work vs Proof of Stake

- Why proof of stake is better:
  - Doesn’t waste electricity and computation cycles
  - Keeps the miners costs low and sustainable
    - PoW: miners waste electricity to mine blocks → would rather include tx with high fees → higher average tx fee → users look to side chains to avoid high fees → miners get screwed
  - Puts the power to the stakeholders, not the miners
  - Harder to execute a double spend attack
    - Would need 51% of funds rather than hashing power. Solves the problem of having mining pools that approach a centralized system.

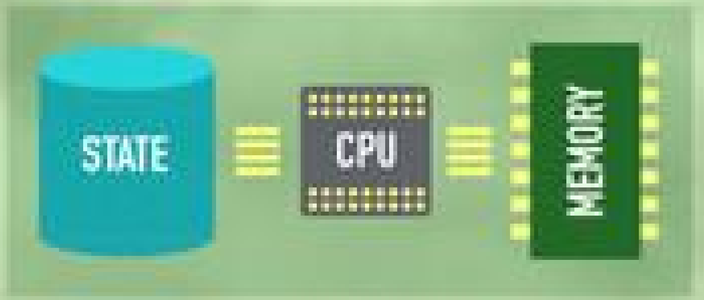
Dfinity:

The Infinite Trust Machine

The Next Cloud Is Decentralized

DFINITY

Ethereum Virtual Machine Compatible



GLOBALLY ACCESSIBLE VIRTUAL COMPUTER

WORLD COMPUTE PLATFORM

Use Cases

Tezos:

A self-amending cryptographic ledger

What is Tezos?

- Currently, blockchains only establish consensus *on the next block* in the chain
- Tezos' platform establishes consensus *on the protocol* of the blockchain (including changes to amendments)
- Tezos tries to solve common problems, including:
  - Hard forks
  - Cost and Centralization issues via Proof of Work
  - Limited Expressiveness
- Scheduled release: 1st quarter 2017

Why was the platform created?

- Anticipation that code would not solve all social problems
- Future innovation is impossible to anticipate
- Concerns on consensus rules emerged regarding:
  - *Miners* who controlled future protocol changes
  - *Developers* who might leverage their position to exert their influence over protocol changes (even if well-intentioned)
  - Examples: Bitcoin blocksize debate, Ethereum DAO hard fork
- Tezos was created to become the *last* cryptocurrency

“Good” vs. “Poor” Governance

What Happens Today

- Informal approach
- Future rules influenced by:
  - Who shouts loudest
  - Miners
  - Developers
    - Bitcoin Improvement Proposals
    - Ethereum Foundation

Tezos' platform:

- Formal approach
- Future rules influenced by:
  - Stakeholders (proof of stake)
  - Representatives (when votes are delegated)
- Proposals marketed to stakeholders *in the system* (i.e., not out-of-band)

What makes the platform unique?

- **Abstraction**
  - All blockchains track state, and changes (via operations) to those states
  - Generic methods for creating blocks, handling transactions, and establishing consensus
  - Way to adopt new blockchain protocols via consensus
  - Tracks *Contexts* a blockchain in three areas: network, transaction, and consensus protocols

What makes the platform unique?

- **Formal verification/proofs**
  - Strongly-typed program language; removes ambiguity at compilation
  - Operations are deterministic and can be mathematically proven (unlike Ethereum's Solidity); stronger, more general guarantees (than unit tests)

How does the platform work?

- A blockchain begins with a **seed protocol**: a conservative set of rules that changes over time

- Two-phase voting process on new protocol changes

  (1) Approval voting system: stakeholders propose protocol changes; voted yea/nay on each proposal

  (2) Vote taken on most popular proposal; 60% approval needed

How does the platform work?

- Separate private key for voting
- Can be delegated to another representative

- This process only pertains to governance proposals (“patches” not covered)
- Votes taken on quarterly basis

What are some use cases?

- More resilient blockchains
  - Tezos mimics Ethereum's smart contract functionality but provides endogenous protocol changes
- Constitutionalism
  - Can ensure that changes to rules abide to properties of the seed protocol (or currently-approved protocol)
- Futarchy
  - *"Vote Values, But Bet Beliefs"*

Tendermint

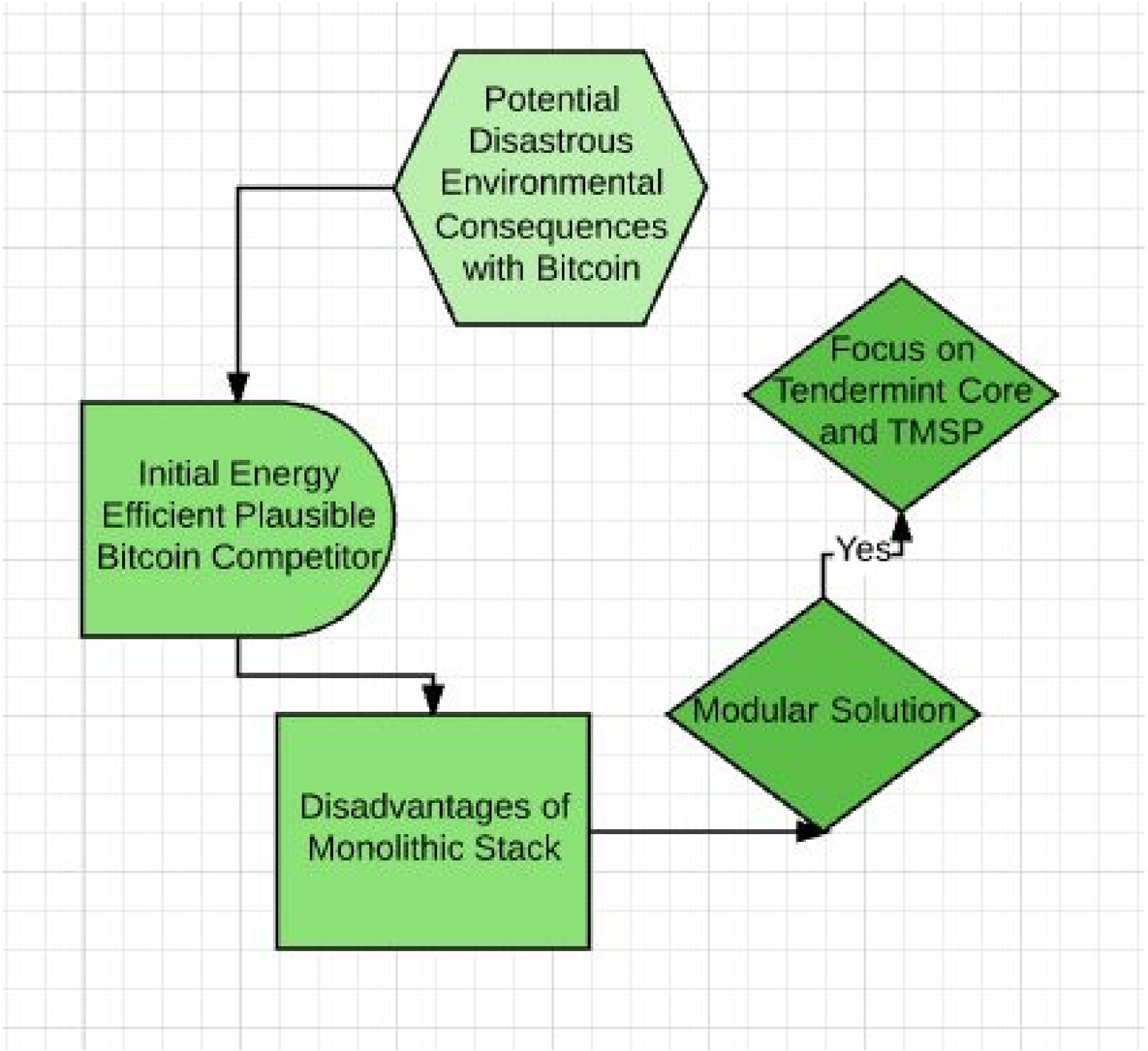
What is tendermint?

- Tendermint consists of a blockchain consensus engine (**Tendermint Core**) and a generic application interface (**TMSP = Tendermint Socket Protocol**).

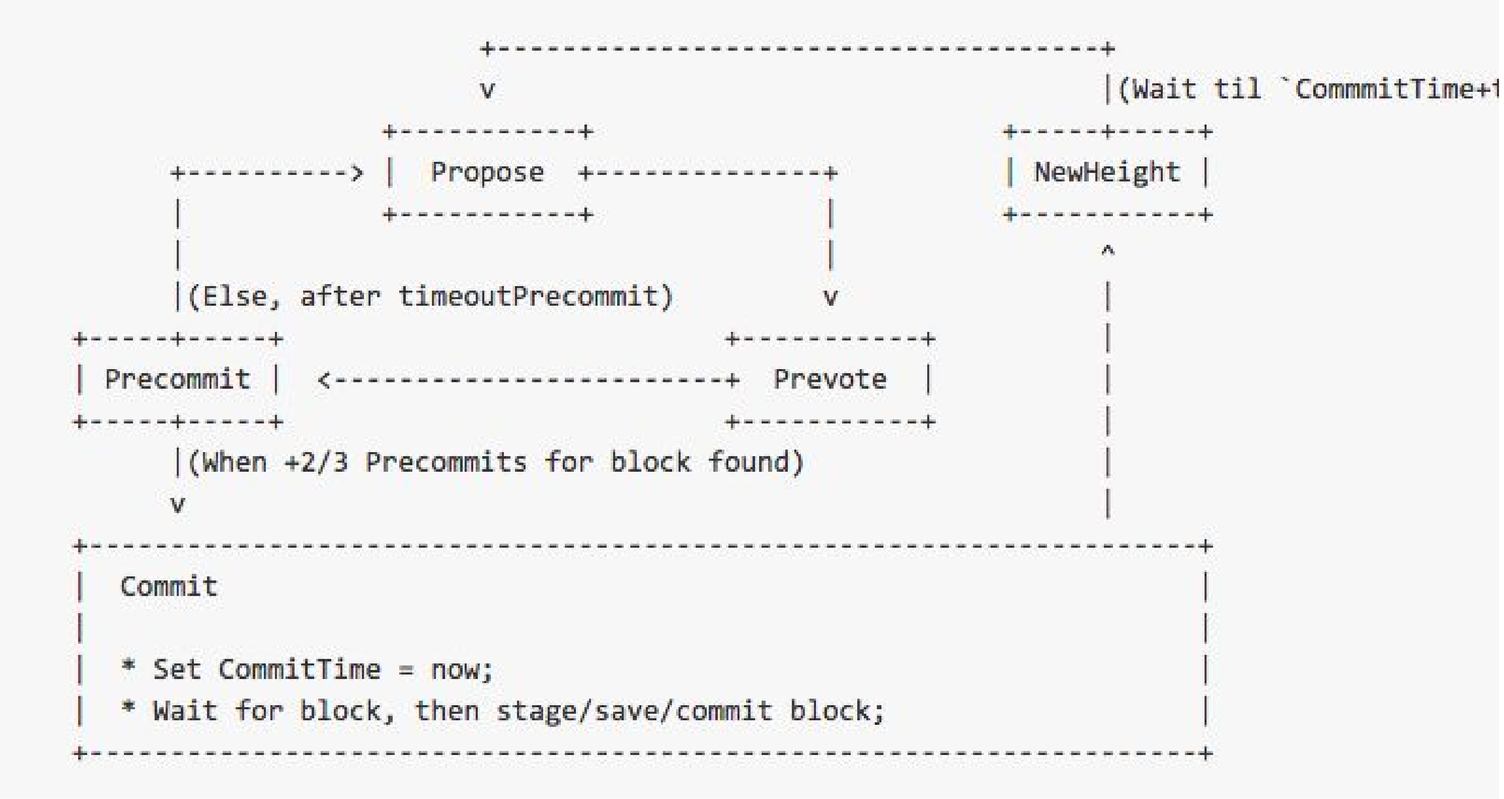
TC ensures that the same transactions are recorded on every machine in the same order, and consists of the BFT consensus alg

TMSP enables the transactions to be processed in any programming language. *

Why was it created?



How does it work? - Tendermint Core



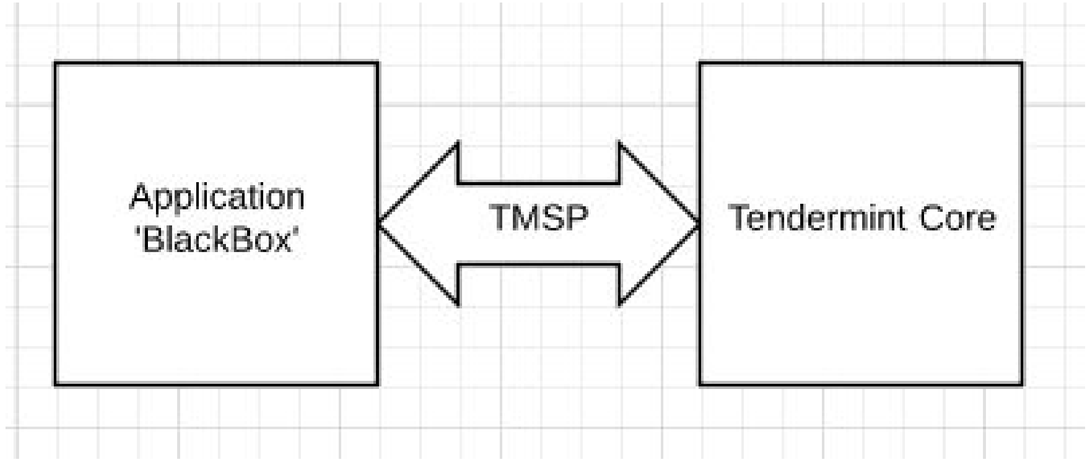
Tendermint is currently used as a consensus layer by dev platforms

Byzantine Generals' Problem:

- Group of generals surrounding a city
- Attack vs retreat
- Traitorous generals

How does it work? - Tendermint Socket Protocol

01. Maintaining “unspent transaction database,” *
02. Validating signatures,
03. Preventing invalid transactions and
04. Allowing clients to query the unspent database.



Quick Note: Scalability

- 10,000 non-processed transactions per blockchain second (before any optimization)
- *easily hitting the bft transaction minimum*
- bottleneck likely to be on the application side

How is it unique?

- No distributed key-value stores w classical, other softwares have non byzantine consensus alg:
  - TMSP allows for applications to be written in any language.
  - Tendermint Core handles Byzantine Fault Tolerance*

Use Cases

- “Blockstreet companies” Forex Exchange use the blockchain as a triple entry account
- Stocks: Autonomical Digital Bare Assets eliminate Clearing & Settlement *far-off
- GovernMint: Basic voting platform on Tendermint

Appendix

Hard forks

- Hard forks are a result of a failure to reach protocol consensus
- Tezos solves this by:
  - Putting the stakeholders in charge
  - Rallying a consensus before the protocol changes
  - Requiring approvement of such changes
  - Ensures everyone is compliant with changes and uses a single protocol

Blockchain Platforms

IBM and Microsoft

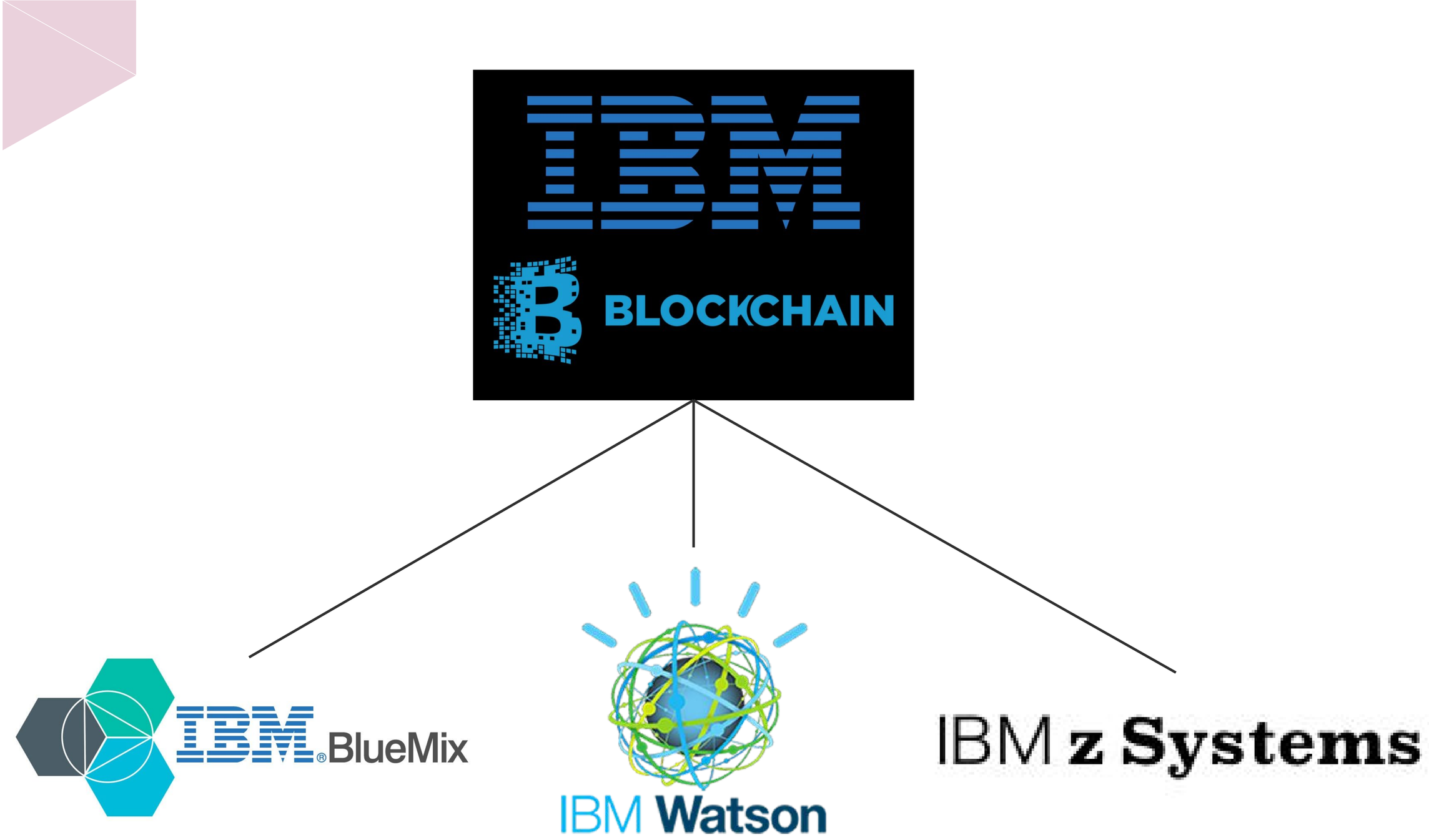
IBM Blockchain

Existing Problem:

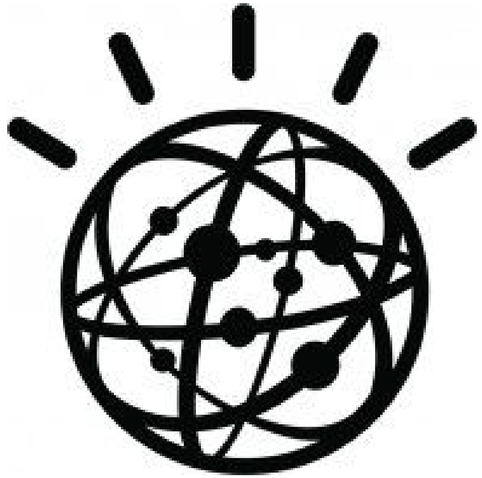
Companies have no easy way to develop enterprise-level blockchain solutions quickly and efficiently.

IBM Goals:

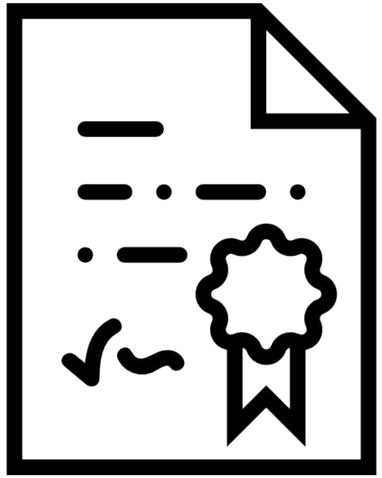
The Blockchain-as-a-service (BaaS) offering will incorporate existing company assets such as IBM z Systems, its core IT system for top 100 global banks; the Watson Internet of Things (IoT) platform; and Bluemix Garage, its development workshop initiative to provide tailormade, scalable solutions to businesses.


IBM
BLOCKCHAIN
IBM BlueMix
IBM Watson
IBM z Systems

IBM Use Cases:



Device-to-device communication using Watson API



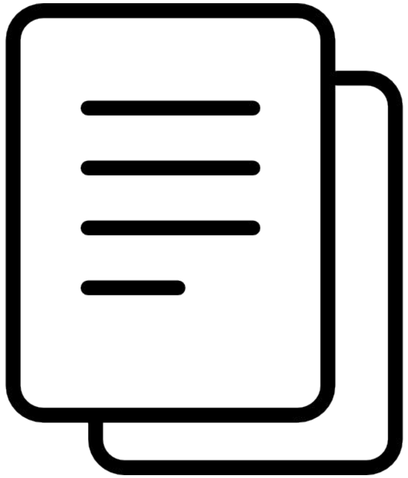
Dispute settlement within organizations

Provenance in government

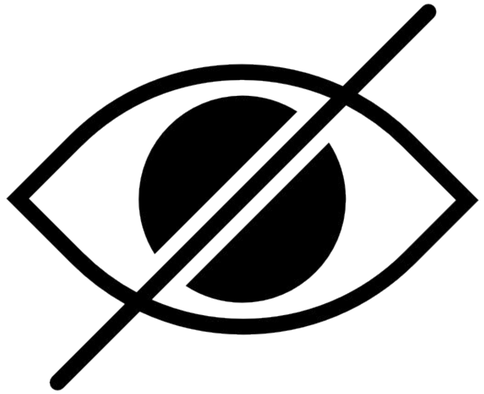
How is Hyperledger Fabric different from other blockchain implementations?

	Bitcoin	Ethereum	Hyperledger
Cryptocurrency required	bitcoin	ether, user-created cryptocurrencies	none
Network	public	public or permissioned	permissioned
Transactions	anonymous	anonymous or private	public or confidential
Consensus	proof of work	proof of work	PBFT
Smart contracts (business logic)	none	yes (Solidity, Serpent, LLL)	yes (chaincode)
Language	C++	Golang, C++, Python	Golang, Java

IBM:
Key Features



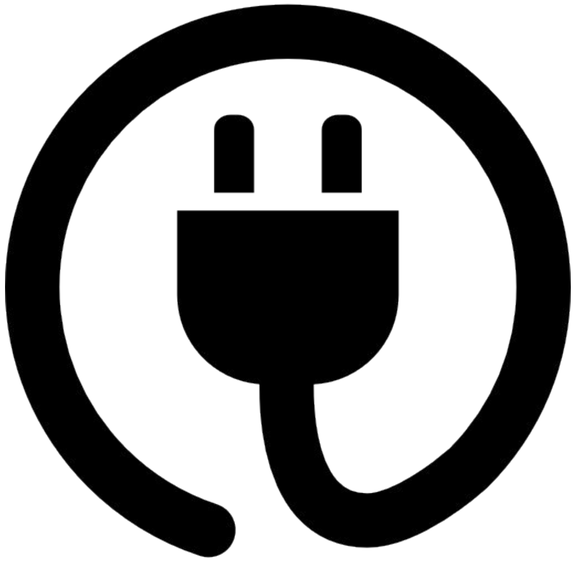
Replication: entries proposed to all nodes



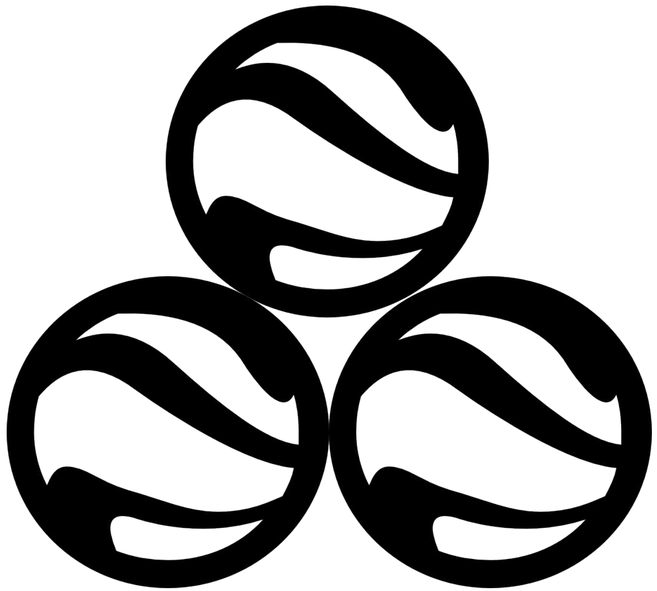
Permissioned: only see what you are allowed to see

Perform Business Logic

IBM:
Unique Selling Prop



Replication: entries proposed to all nodes

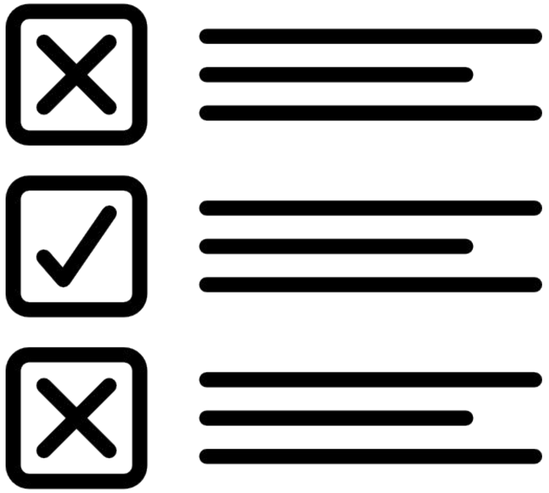


“Marbles App”--Exchange marbles

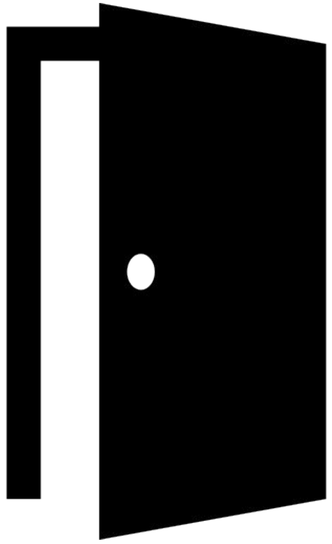
Microsoft Blockchain as a Service

Existing Problem:

Companies have no easy way of securely doing ‘experimental transactions’.



What if it fails?

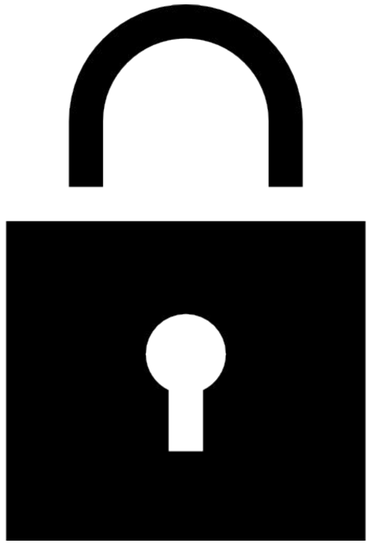


What if Someone Backs Out?

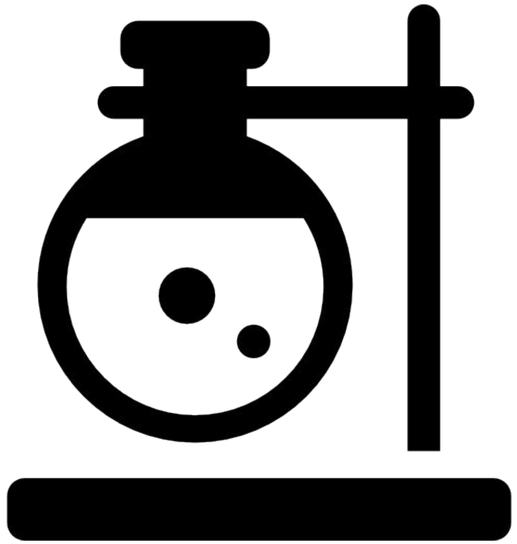
What if it’s too Expensive?

Microsoft: Goals

Rapid + low cost platform

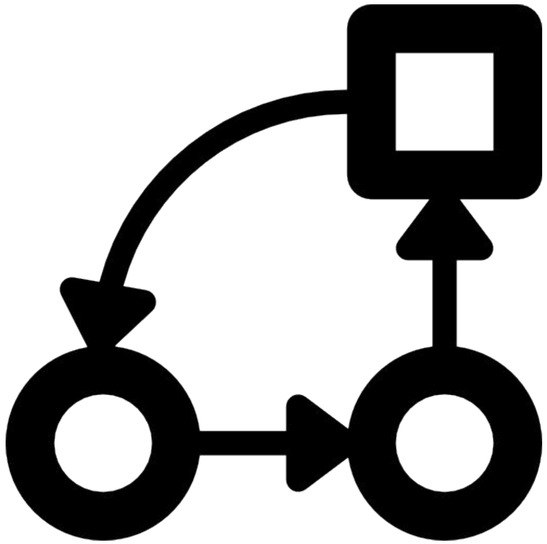


Fail fast, extremely secure

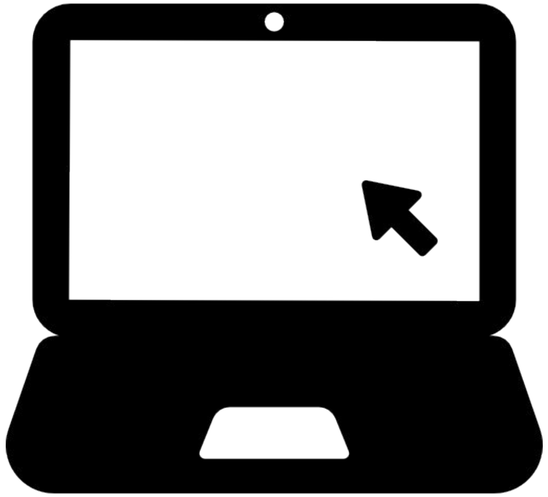


Bold Experimentation

Microsoft: Use Cases

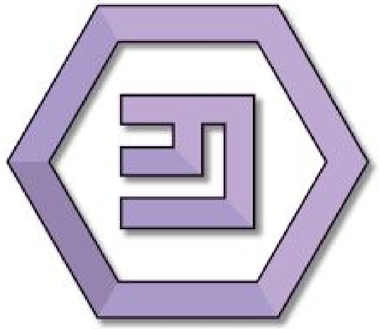


Business to business



For developers too!

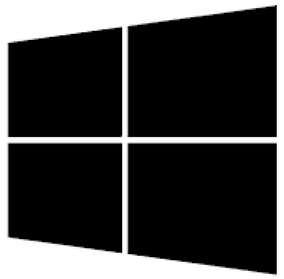
BlockApps



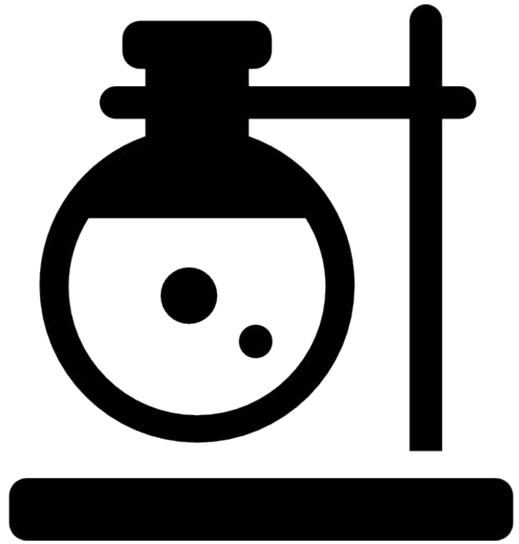
Chain

< ether.camp >

Microsoft: USP



Microsoft
Azure

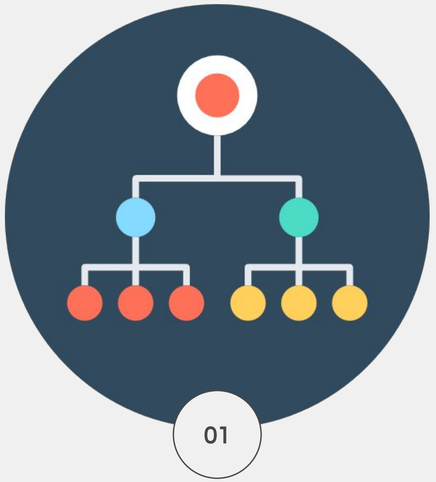


Integration with Azure

Services like Ethereum Studio

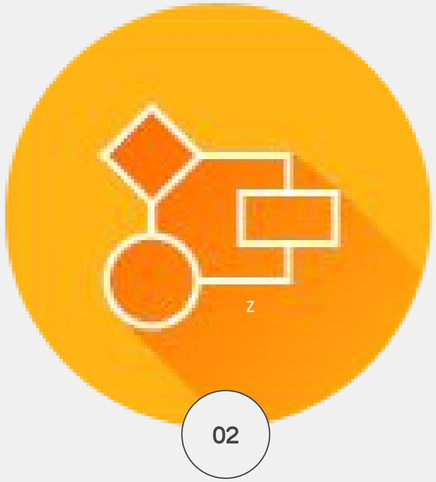
Low risk experimentation

Content



Blockchain Data Structures

Max Fang



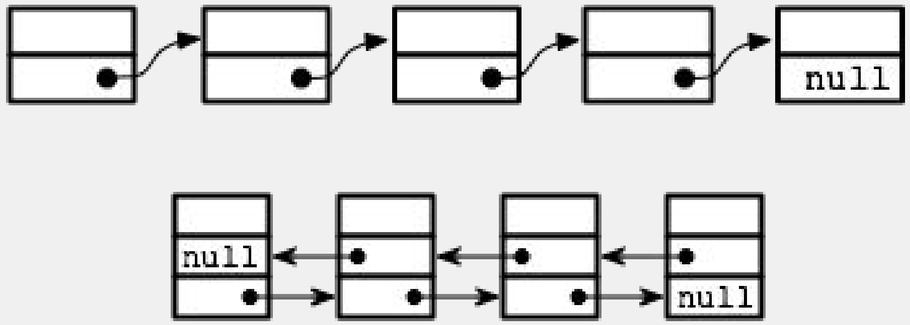
Consensus Algorithms

Tobias Boelter

Bitcoin Scripting

Max Fang

Blockchain Data Structures



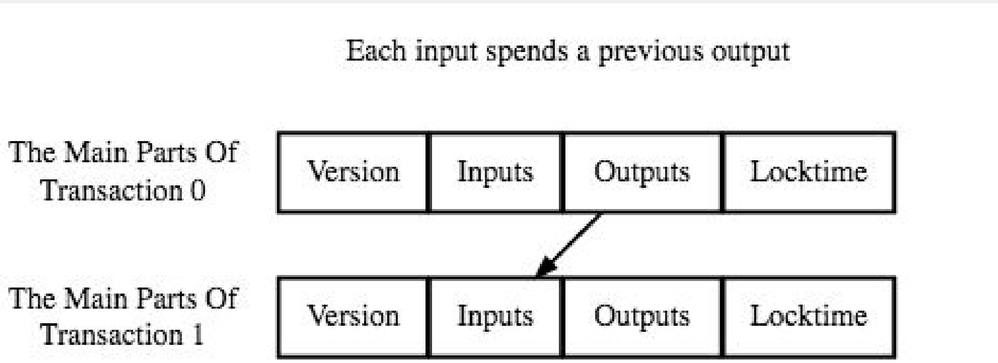
Basic Concepts - Identity in Bitcoin

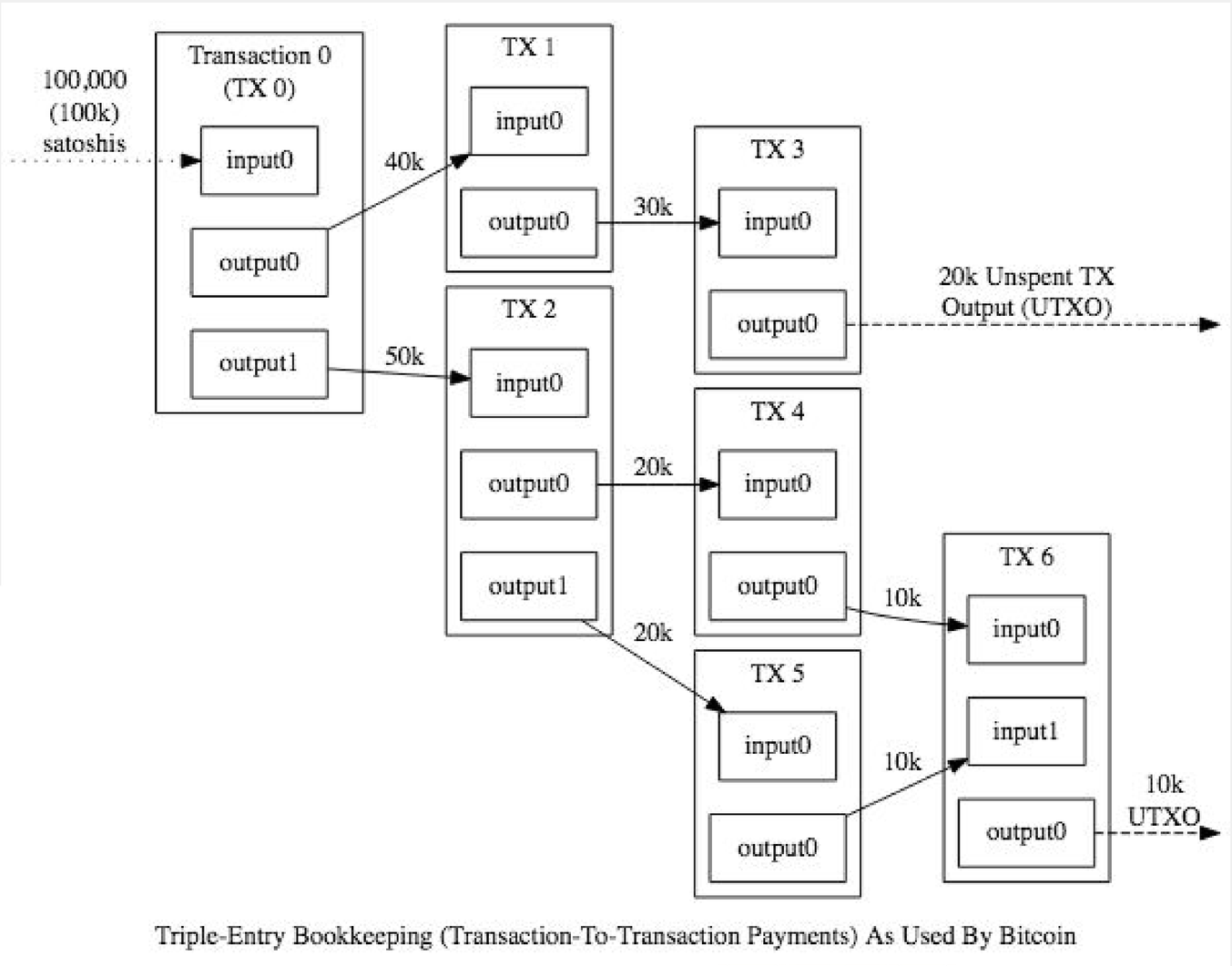
- Send money between pseudonyms
  - pseudonym == address == public key
  - address = hash(public key)
- Cryptographic primitives
  - digital signature scheme (ECDSA: Elliptic Curve Digital Signature Algorithm)
    - public key/private key pair; like email address + password
  - one-way hash function (SHA-256)
- Bitcoin is hidden in the large amount of public keys
  - Users can generate arbitrarily many key pairs
  - Example Address: 1FtQU9X78hdshngJiCBw9tbE2MYpx87eLT
  - 2^160 possible addresses
    (1,461,501,637,330,902,918,203,684,832,716,283,019,655,932,542,976 addresses)
  - Grains of sand on earth: 2^63
  - 2^126 is actually only 0.0000000058% of 2^160

Basic Concepts - Transactions

Source: Bitcoin Developer Guide

- Maps inputs addresses to output addresses
  - Outputs can only be spent once
- Typical tx: one input, two outputs
- Fees are implicit





Basic Concepts - UTXO analogy

UTXOs stands for "Unspent Transaction Outputs"

- Global set of unspent bitcoins
- "I'm spending THIS bitcoin," not "I'm spending A bitcoin."

Analogous to Rai Stones of the Yap Islands

- Rai Stones never moved
- Instead: Agreed on change of ownership

Source: Wikipedia

Merkle Tree

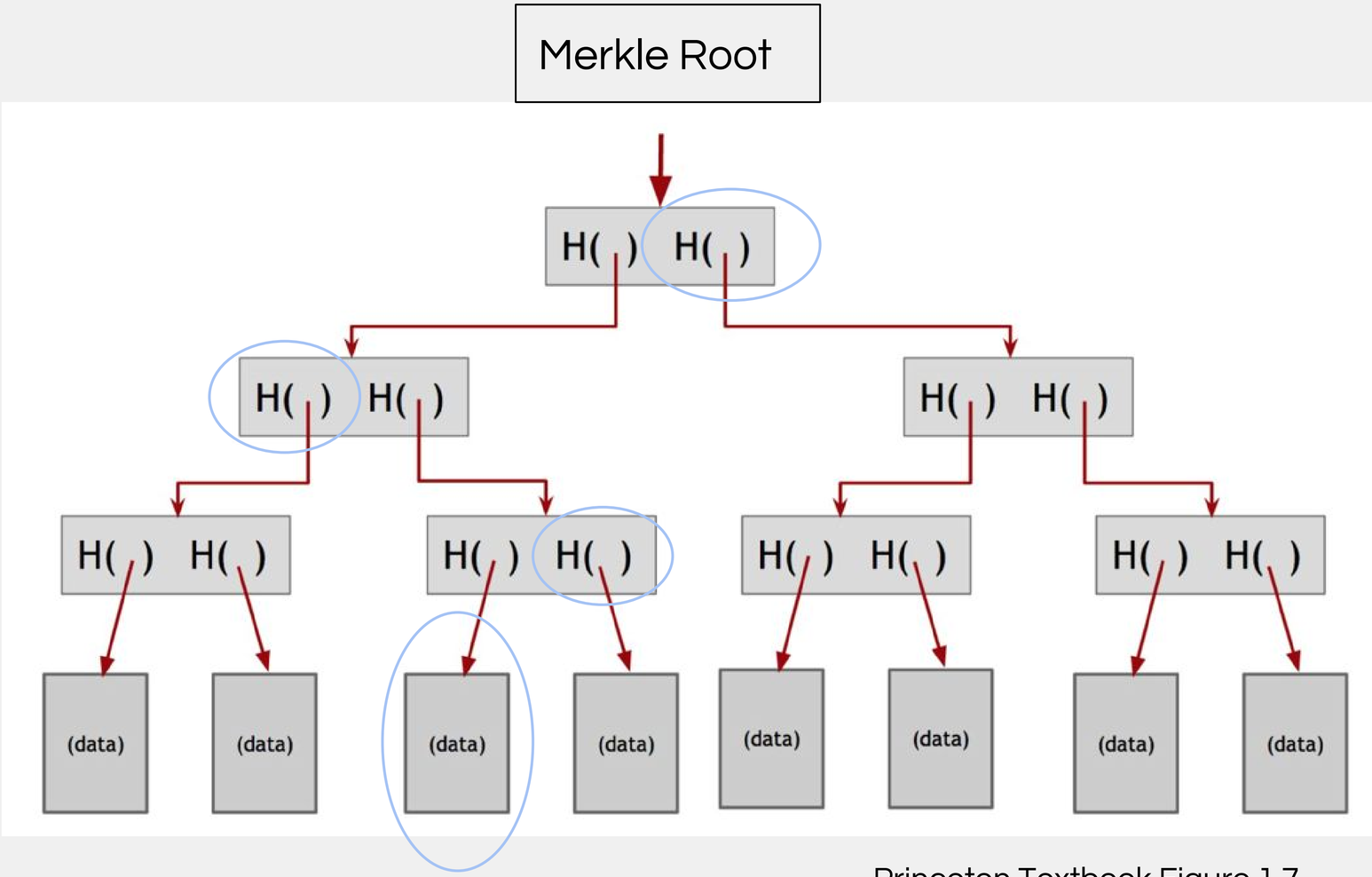
A binary tree of hash pointers

- Blobs of data are hashed
- Hashes are hashed together

Merkle trees are a way to very efficiently **commit** to a large string of data and later prove that this string contains certain substrings.

To prove inclusion of data in the Merkle tree, provide root data and intermediate hashes

- To fake the proof, one would need to find hash preimages
  - Second preimage resistance meets this qualification



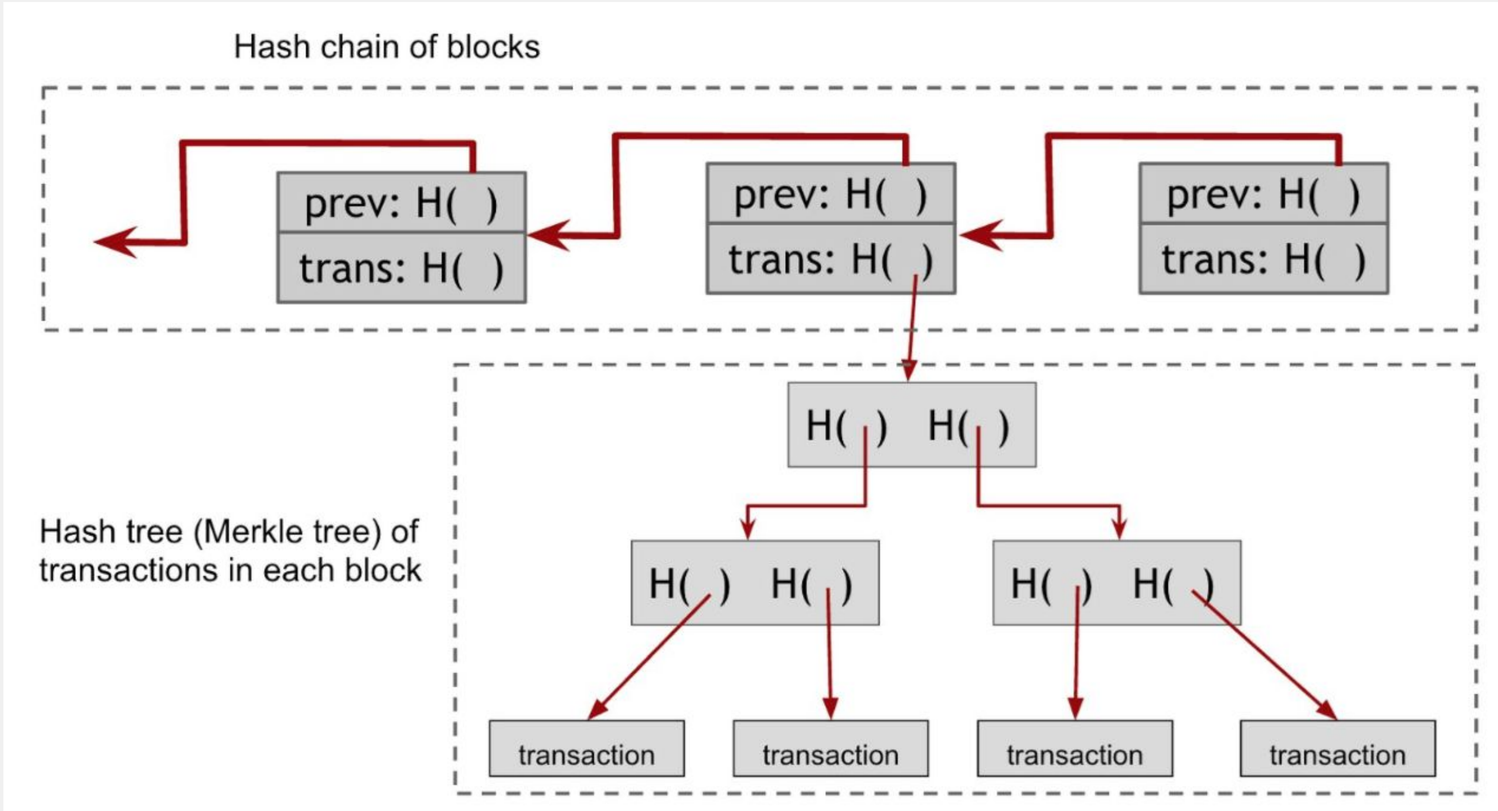
Princeton Textbook Figure 1.7

Merkle Tree - Bitcoin construction

Transactions are leaves in the Merkle tree, includes a coinbase transaction

Two hash structures

1. Hash chain of blocks
   a. These blocks are linked together and based off of each other
      i. tamper evident
2. A Merkle tree of txs, internal to each block
   a. Detail: Merkle tree is always full - duplicate the last tx to fill in gaps



Princeton Textbook Figure 3.7/3.8

Merkle Tree - PoW Mining, in more detail

Previously, hash of:

- Merkle Root
- PrevBlockHash
- Nonce (varied value)

below some target value.

Actually two nonces:

1. In the block header
2. In the coinbase tx

Hash of

- PrevBlockHash
- Coinbase nonce (varied value)
  - Affects the Merkle Root
- Block header nonce (varied value)

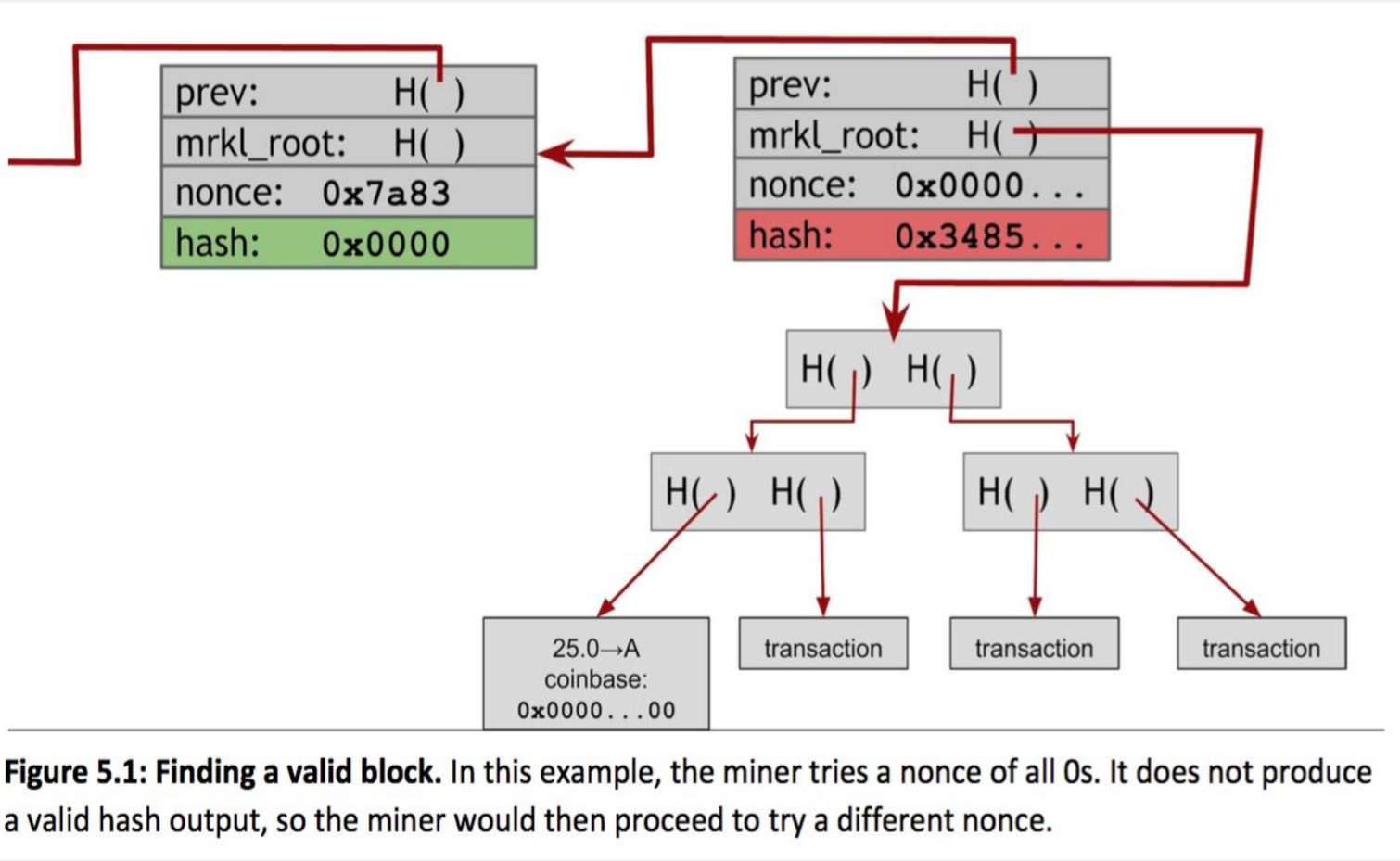


Figure 5.1: Finding a valid block. In this example, the miner tries a nonce of all 0s. It does not produce a valid hash output, so the miner would then proceed to try a different nonce.

Princeton Textbook

Merkle Tree - Bitcoin construction

What if there is no solution?

- Block header nonce is 32 bits
  - Antminer S9 hashes 14 TH/s
  - How long does it take to try all combinations?
  - 2^32 / 14,000,000,000,000 = 0.00031 seconds
  - Exhausted 3260 times per second
- Therefore, must change Merkle root
  - Increment coinbase nonce, then run through block header nonce again
  - Incrementing coinbase nonce less efficient because it must propagate up the tree

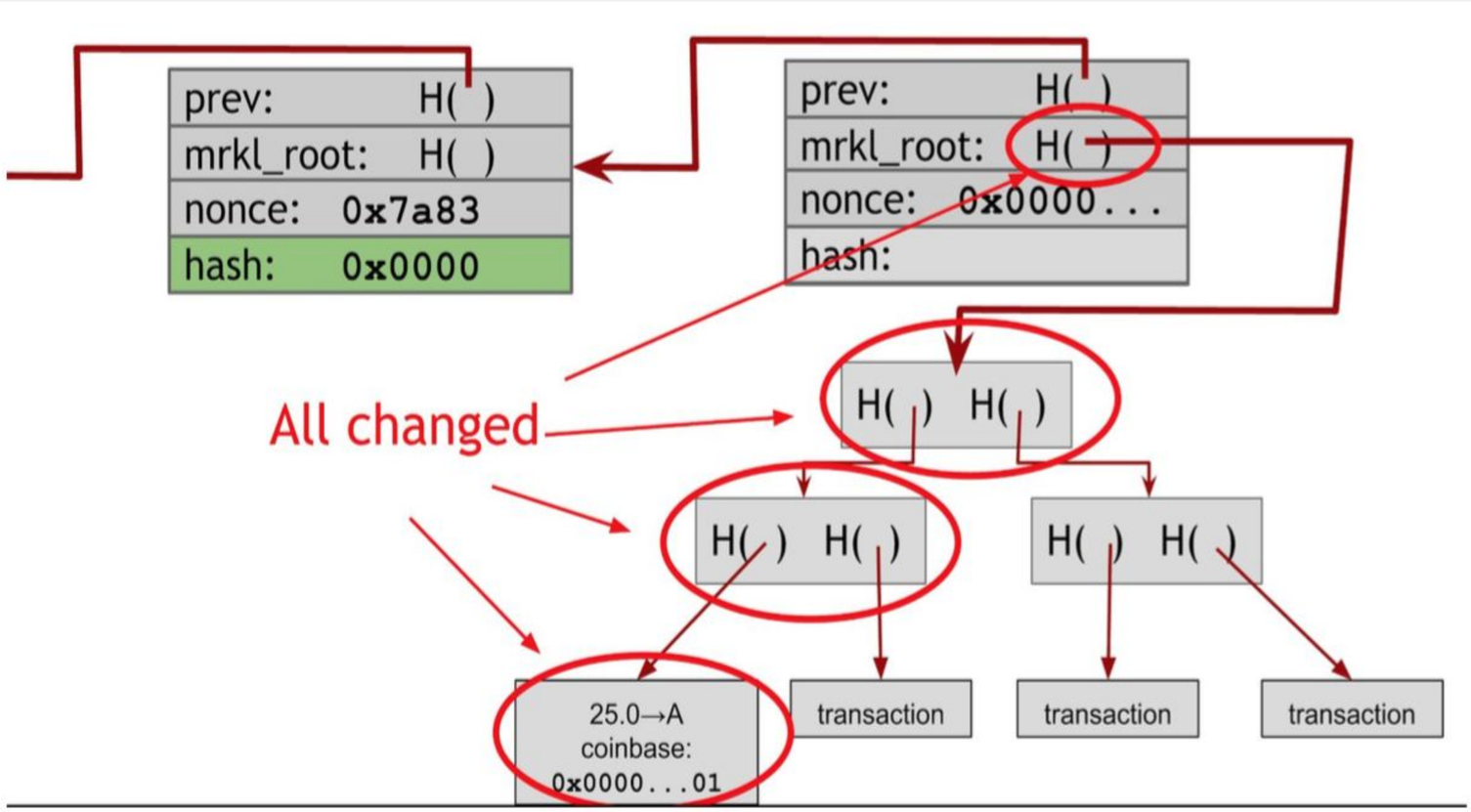


Figure 5.2: Changing a nonce in the coinbase transaction propagates all the way up the Merkle tree.

Princeton Textbook

Account-based vs. Transaction-based ledger

Account-based

- must track every transaction affecting Alice
- Requires additional maintenance, error-prone

Bitcoin is a transaction-based ledger (triple-entry accounting).

Features:

- Change addresses - Required since tx outputs only spent once
- Efficient verification - only read recent history
- Joint payments - Alice + Bob form 1 tx

(Credit for content organization and figures goes to Princeton textbook)

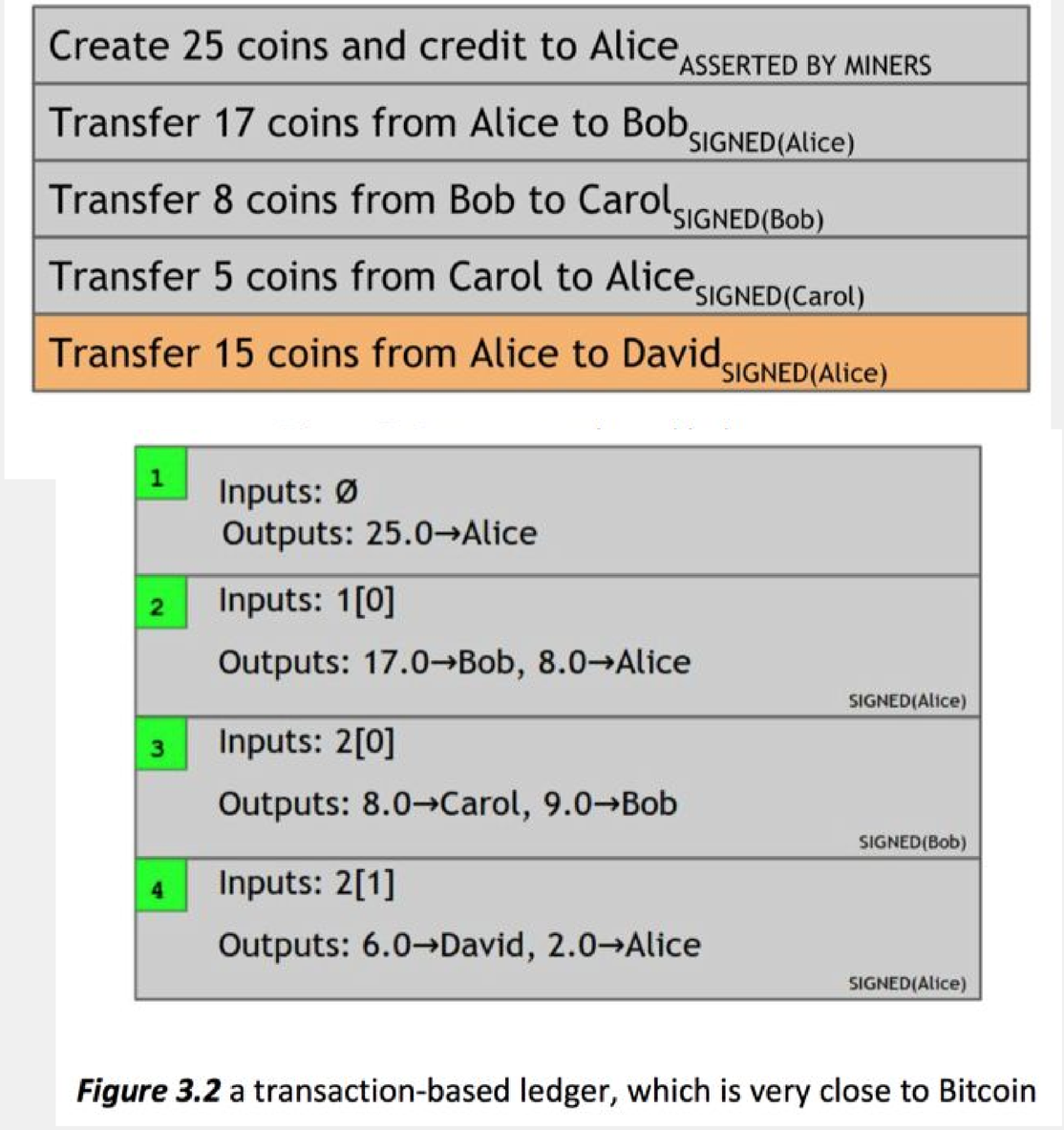


Figure 3.2 a transaction-based ledger, which is very close to Bitcoin

SPV - Simplified Payment Verification

Current size of Bitcoin blockchain: 91 Gigabytes and growing

- 350% of what it was 2 years ago
- Storing the full Bitcoin blockchain will likely not be feasible for the average user, so how do we address this?

SPV - Simplified Payment Verification

- "SPV nodes" or "thin" client, as opposed to "full nodes"
  - Don't store the full blockchain
  - Only store the pieces of data needed to verify transactions that concern them
- Nearly all nodes on network are SPV nodes

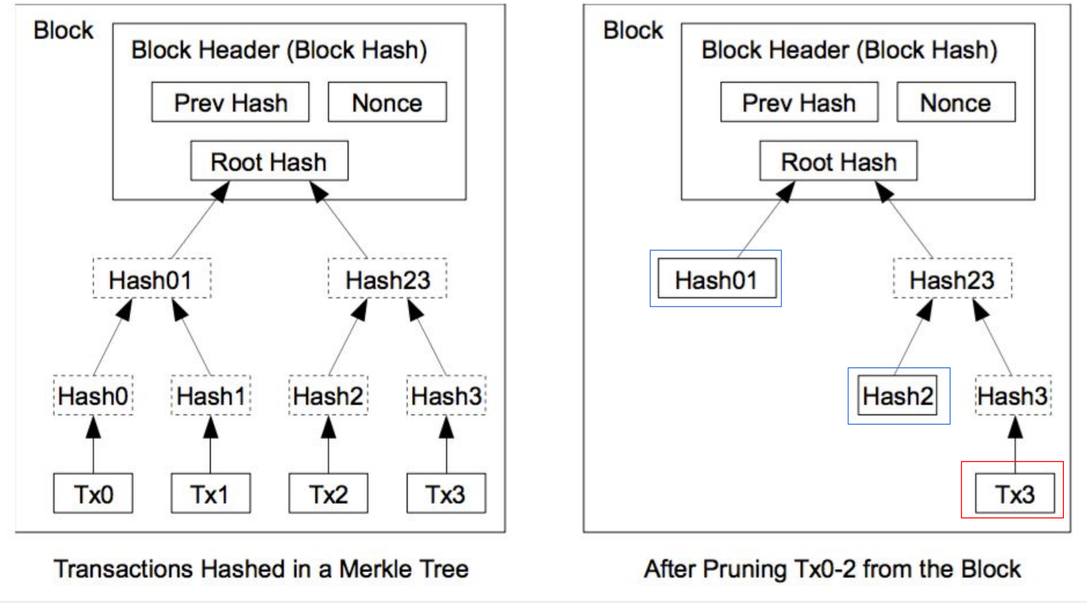
SPV - Structure

SPV nodes only keep block headers of the blockchain

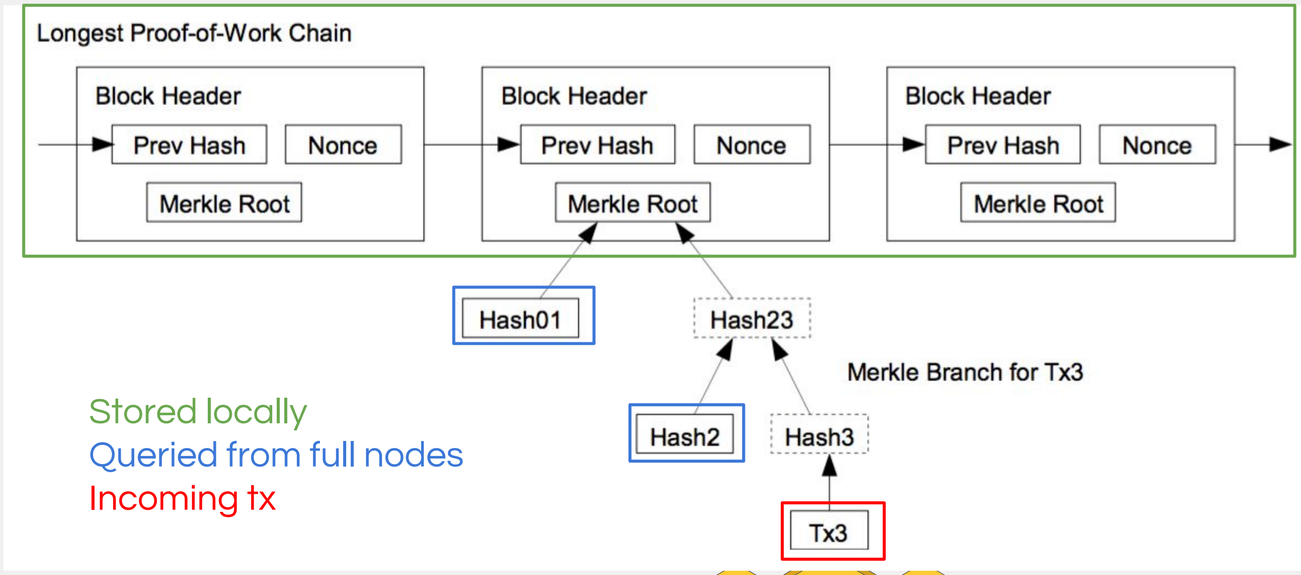
- Stores the proof of work information
- Obtained by querying different full nodes until the node is convinced it has the longest chain

How to validate an incoming tx?

- Query full nodes to get the Merkle branch for that transaction
- Hash tx together with intermediate hashes to obtain a Merkle root, check that it matches the one saved locally.
- Wait for this tx to have enough confirmations before delivering good

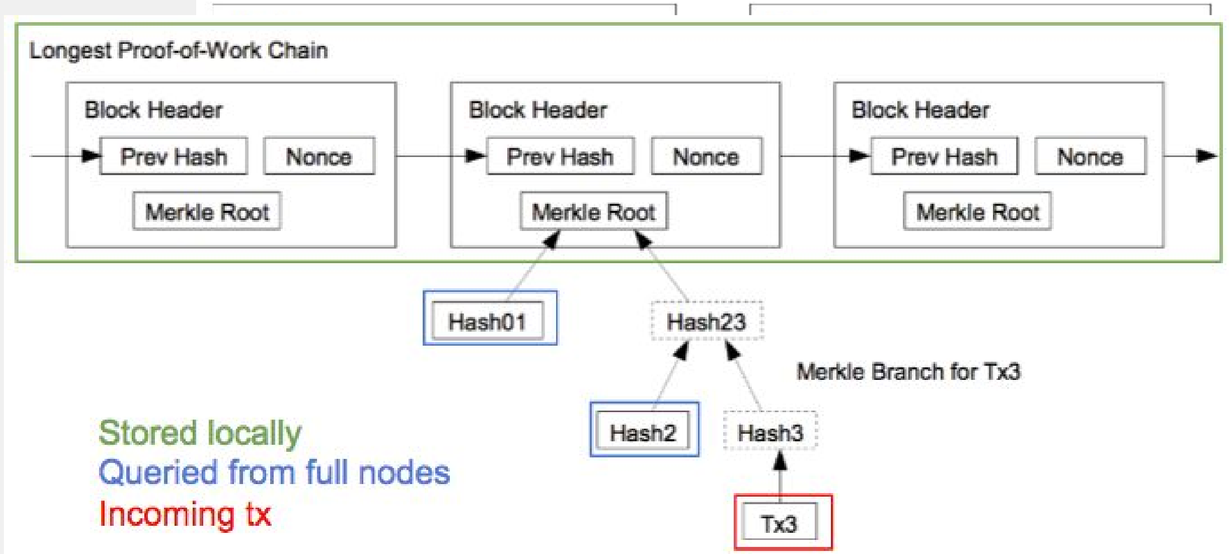


Nakamoto, 2009



Nakamoto, 2009

SPV - Security Analysis



Nakamoto, 2009

SPV nodes:

- Don't have full tx history, don't know UTXO set
- Don't have same level of security of full nodes
  - Can't check if every tx included in a block is actually valid

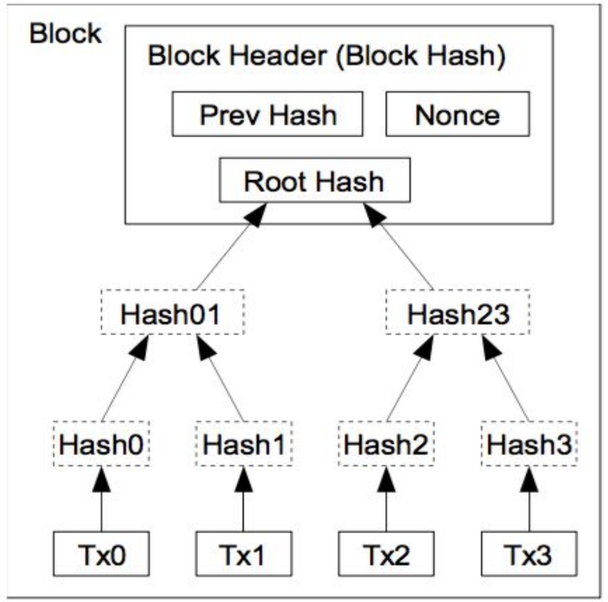
SPV nodes assume:

- ...that incoming block headers aren't a false chain
  - Very expensive for attackers (or anyone) to create blocks
  - Not sustainable over the long term
- ...that there ARE full nodes out there validating all transactions
  - There are efficiency benefits and incentives to doing so (e.g. large merchants)
- ...that miners ensure that the transactions they include in their blocks are valid
  - Otherwise their blocks would be rejected by full nodes (very expensive mistake!)

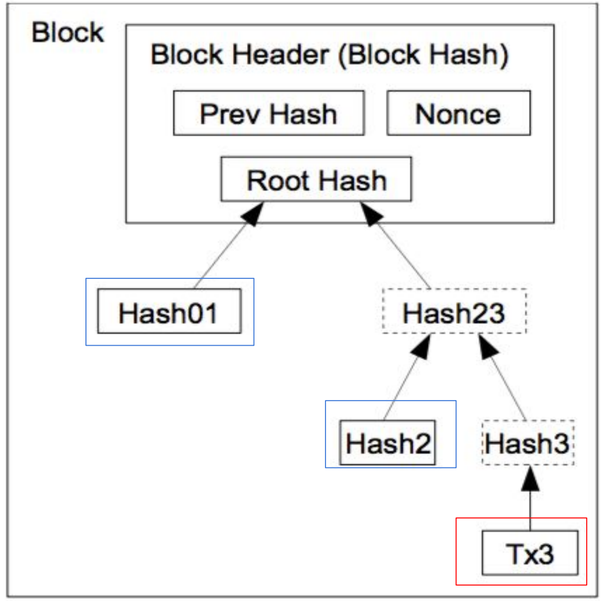
SPV - Cost savings

Huge cost savings

- Block headers are only ~1/1000 the size of the full blockchain
  - 91 MB vs. 91 Gigabytes



Transactions Hashed in a Merkle Tree



After Pruning Tx0-2 from the Block

Nakamoto, 2009

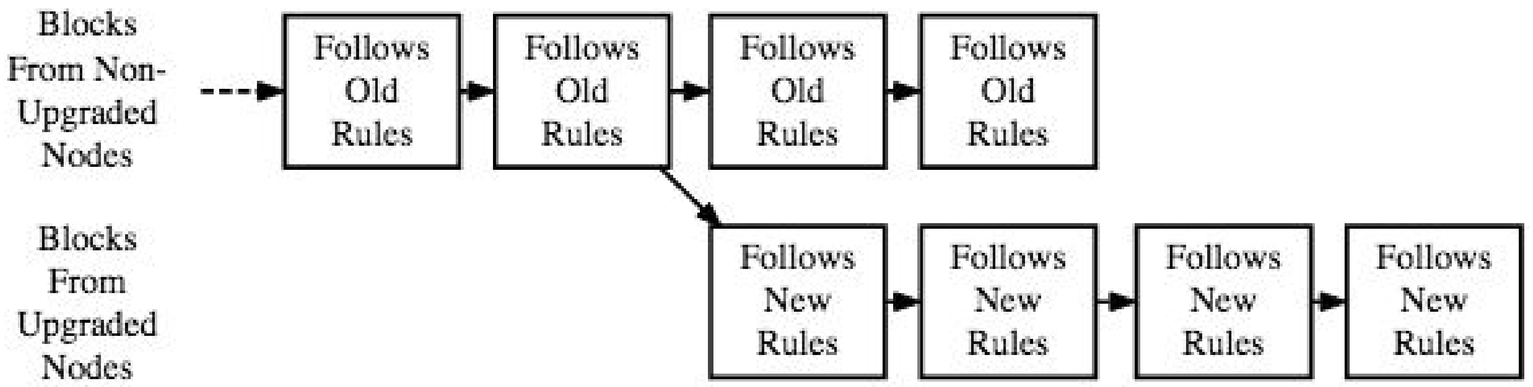
SPV nodes capitalize on obtaining data for verifying transactions lazily

- Recall that it could be an issue for big merchants who must verify many txs

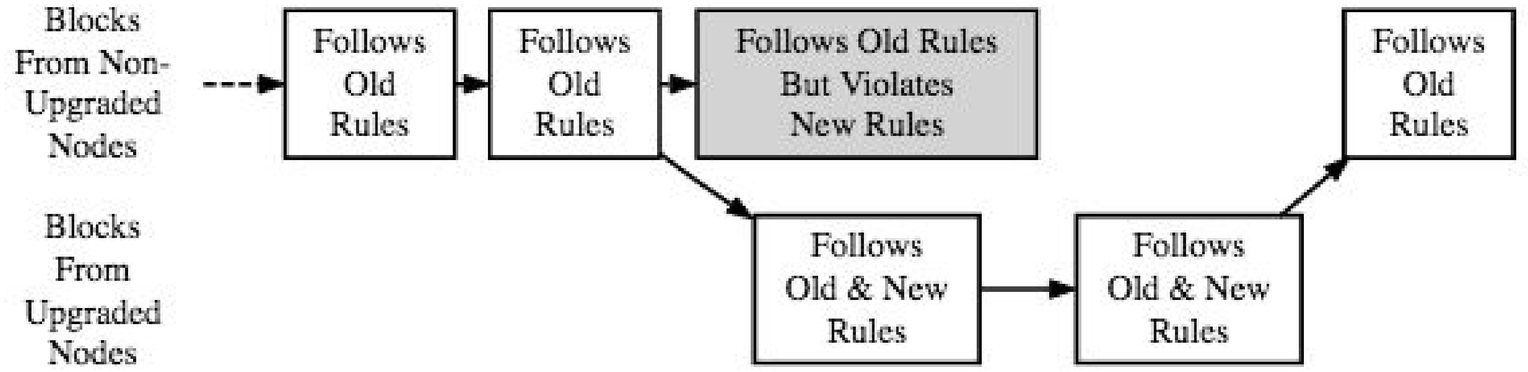
For most consumers and users of Bitcoin, SPV is a decent tradeoff.

Forking + Consensus Updates

Source: Bitcoin.org Developer Guide

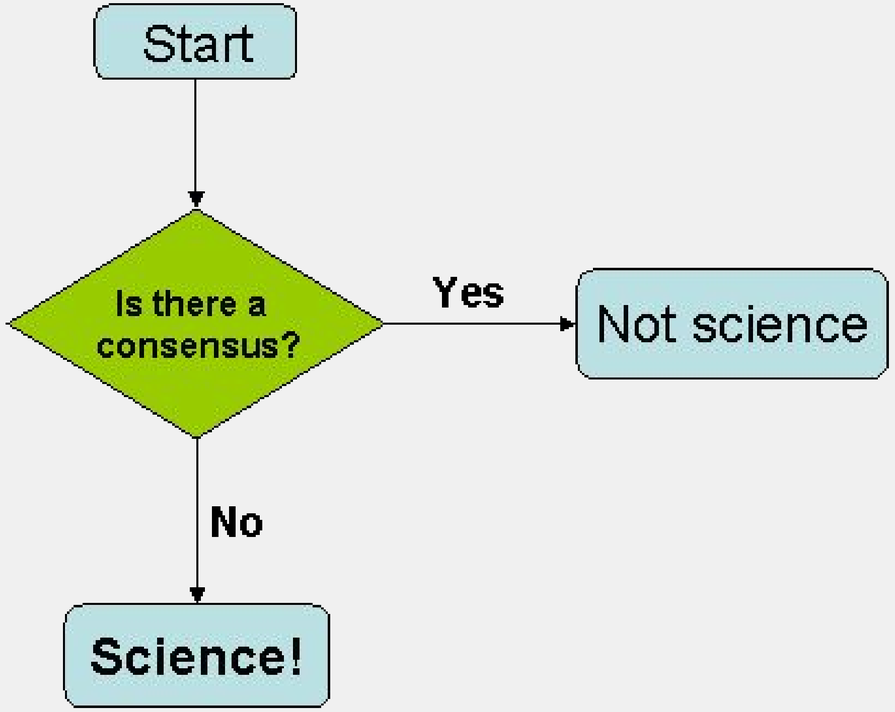


A Hard Fork: Non-Upgraded Nodes Reject The New Rules, Diverging The Chain



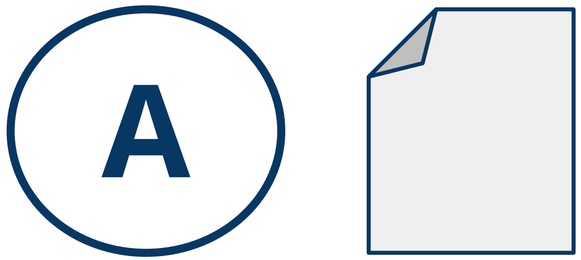
A Soft Fork: Blocks Violating New Rules Are Made Stale By The Upgraded Mining Majority

Consensus Algorithms



v1

Alice writes and signs a message describing her transaction

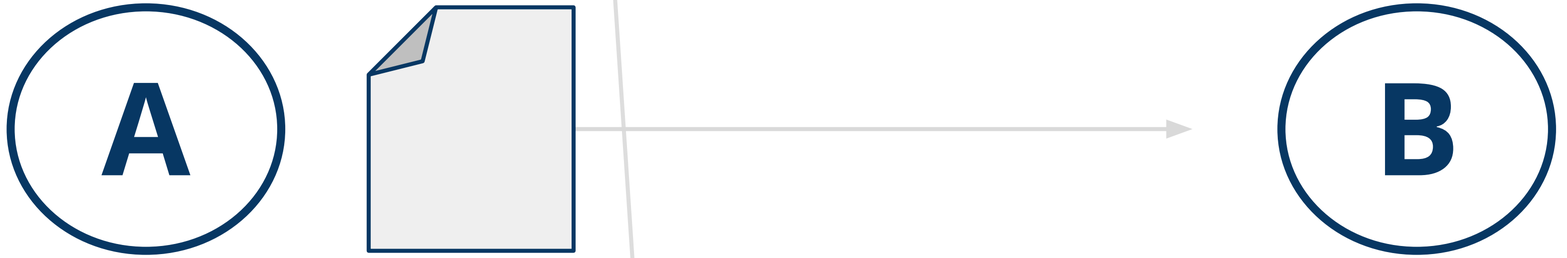


“I, Alice, am giving Bob one bitcoin.”

Slide by Viget

v1

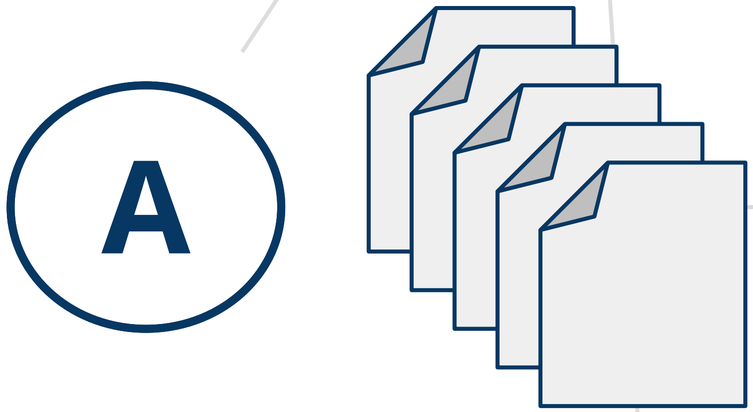
Alice sends her message to the world

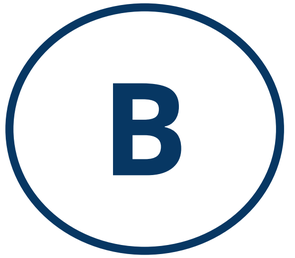


Slide by Viget

v1

Alice sends five identical messages



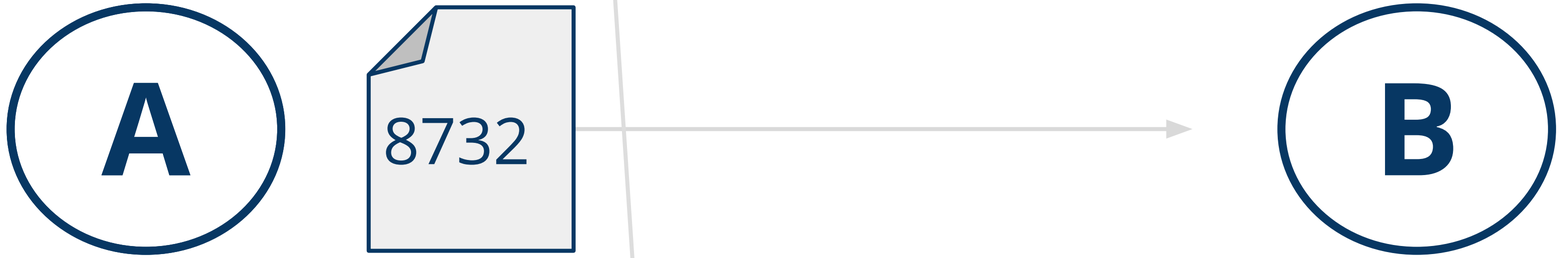


?

Slide by Viget

v2

Introducing uniquely identifiable serial numbers



Slide by Viget

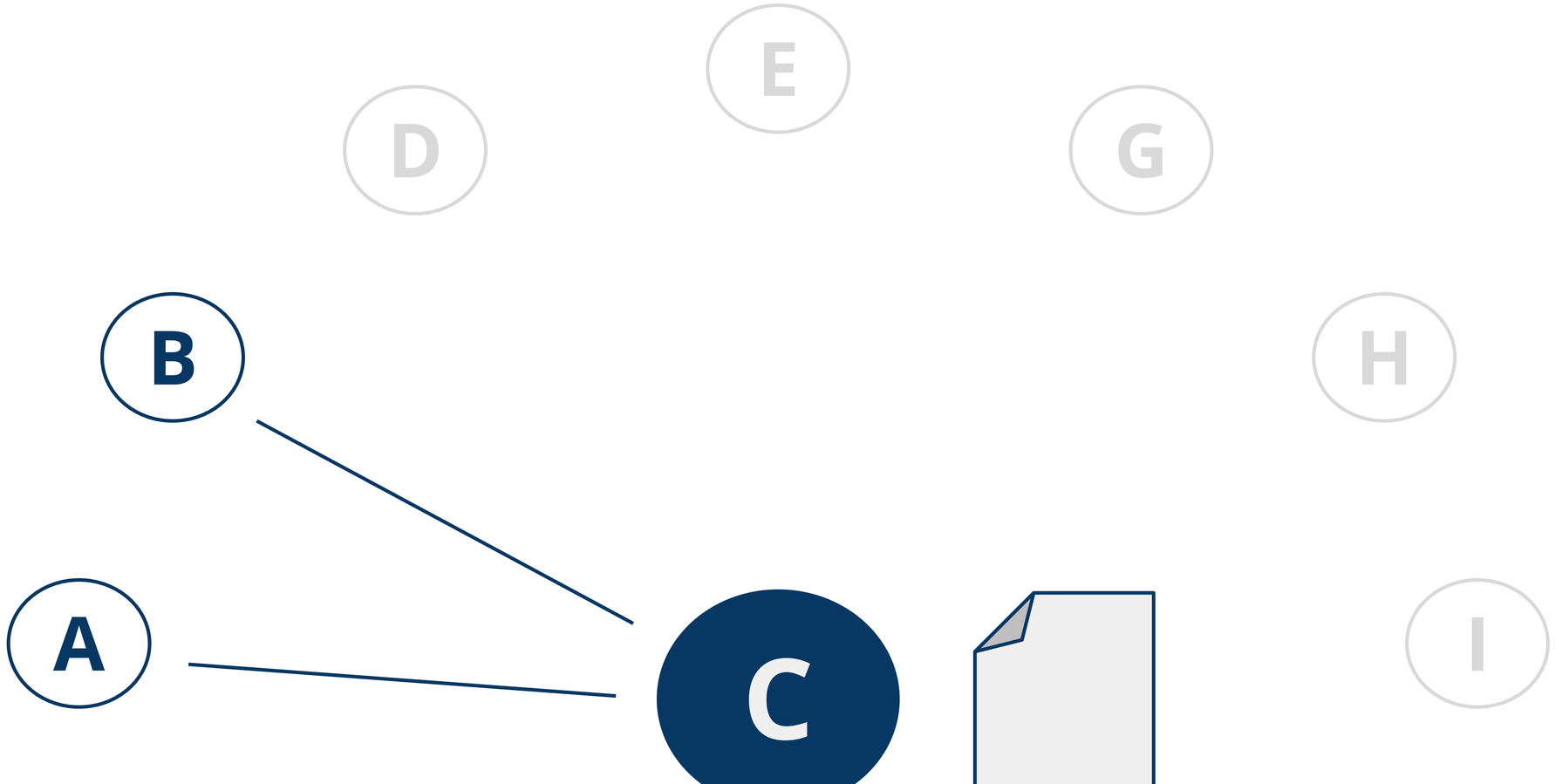
v2

Where do serial numbers come from?

?

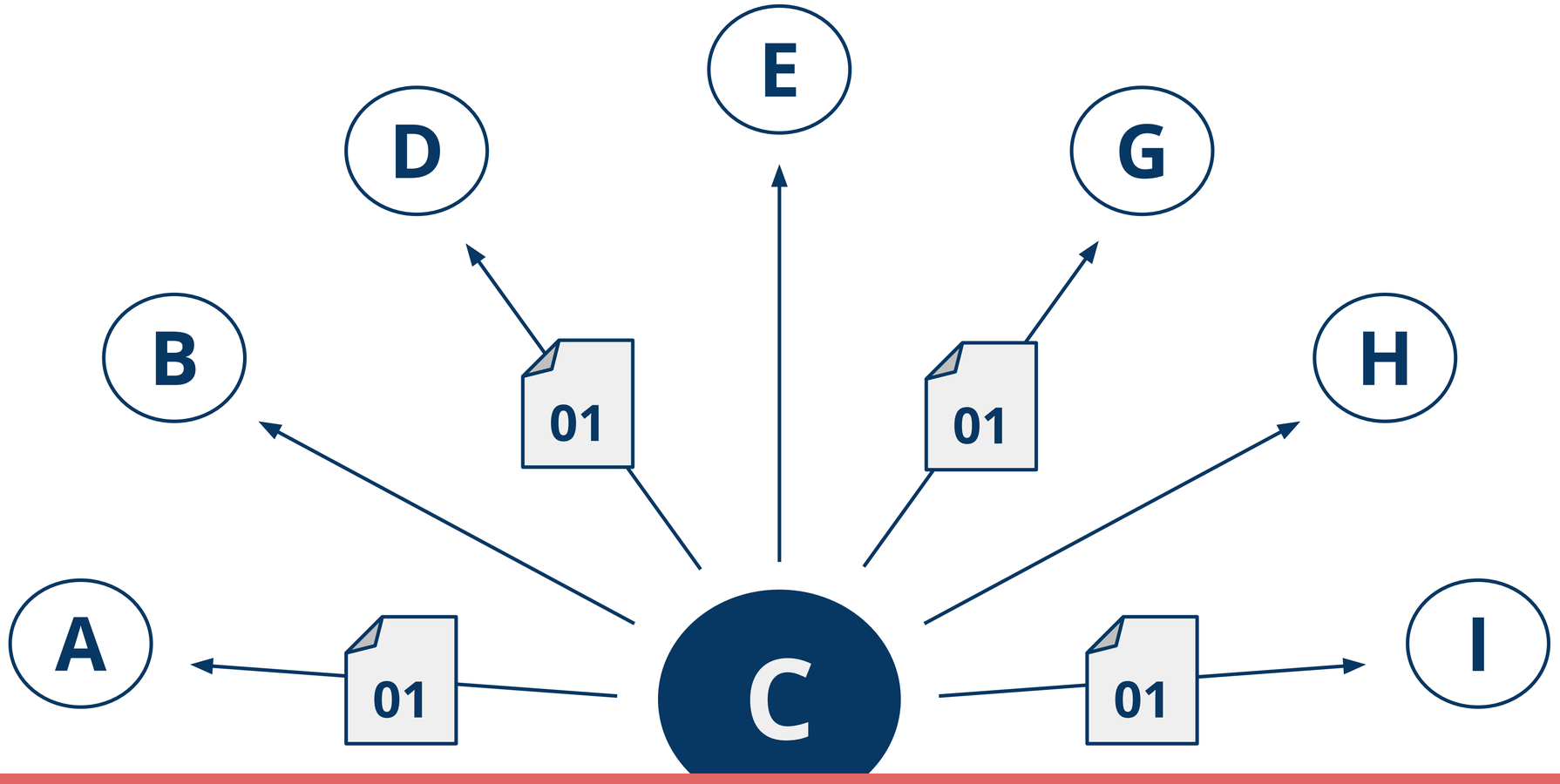
v2

A central bank manages transactions and balances



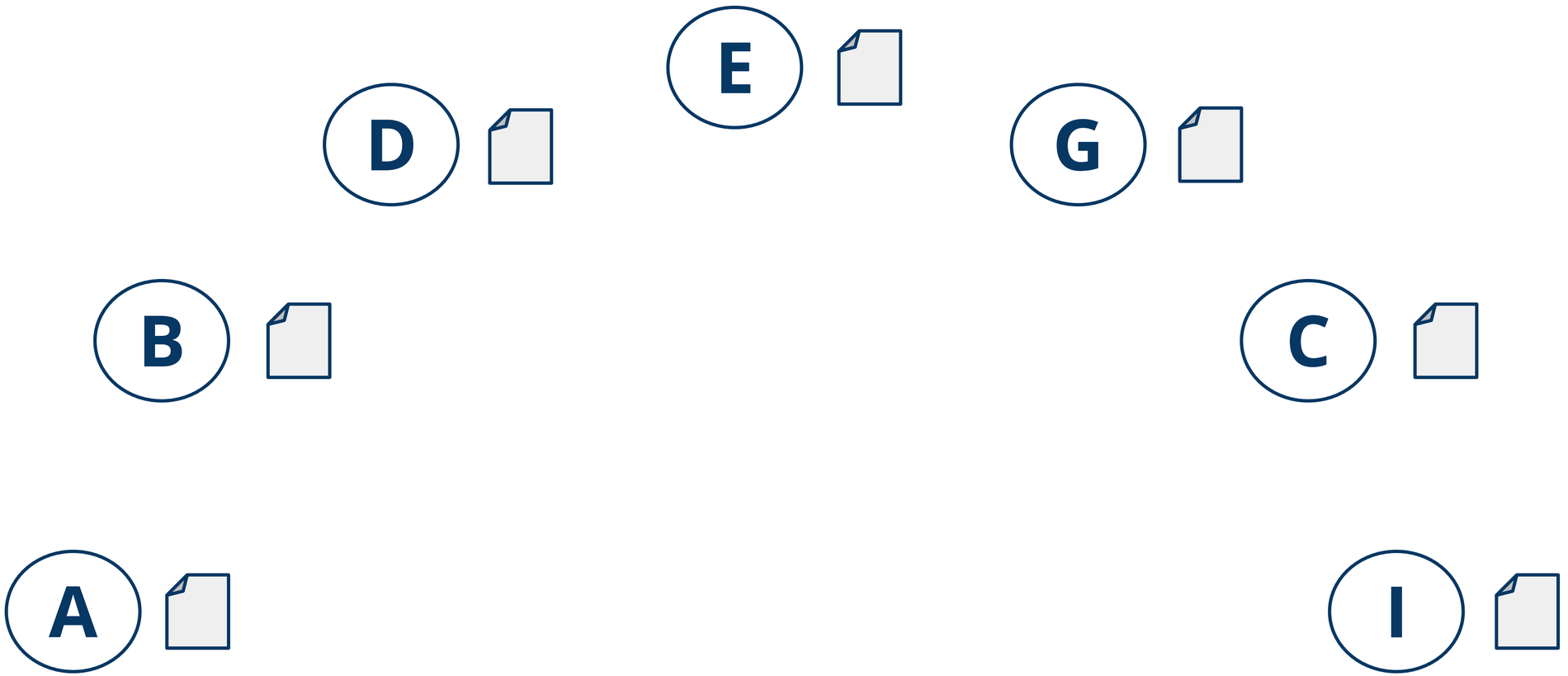
v2

Centralization



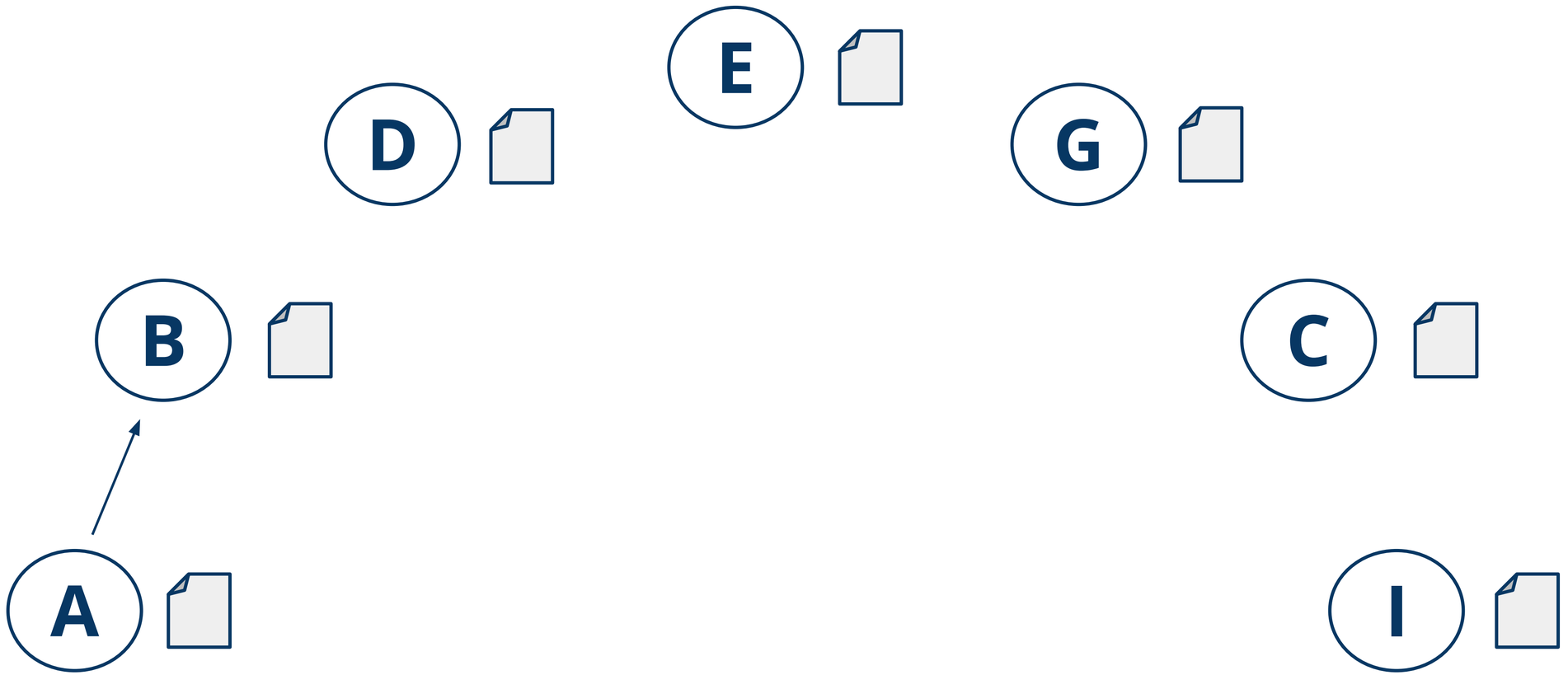
v3

Making everyone the bank



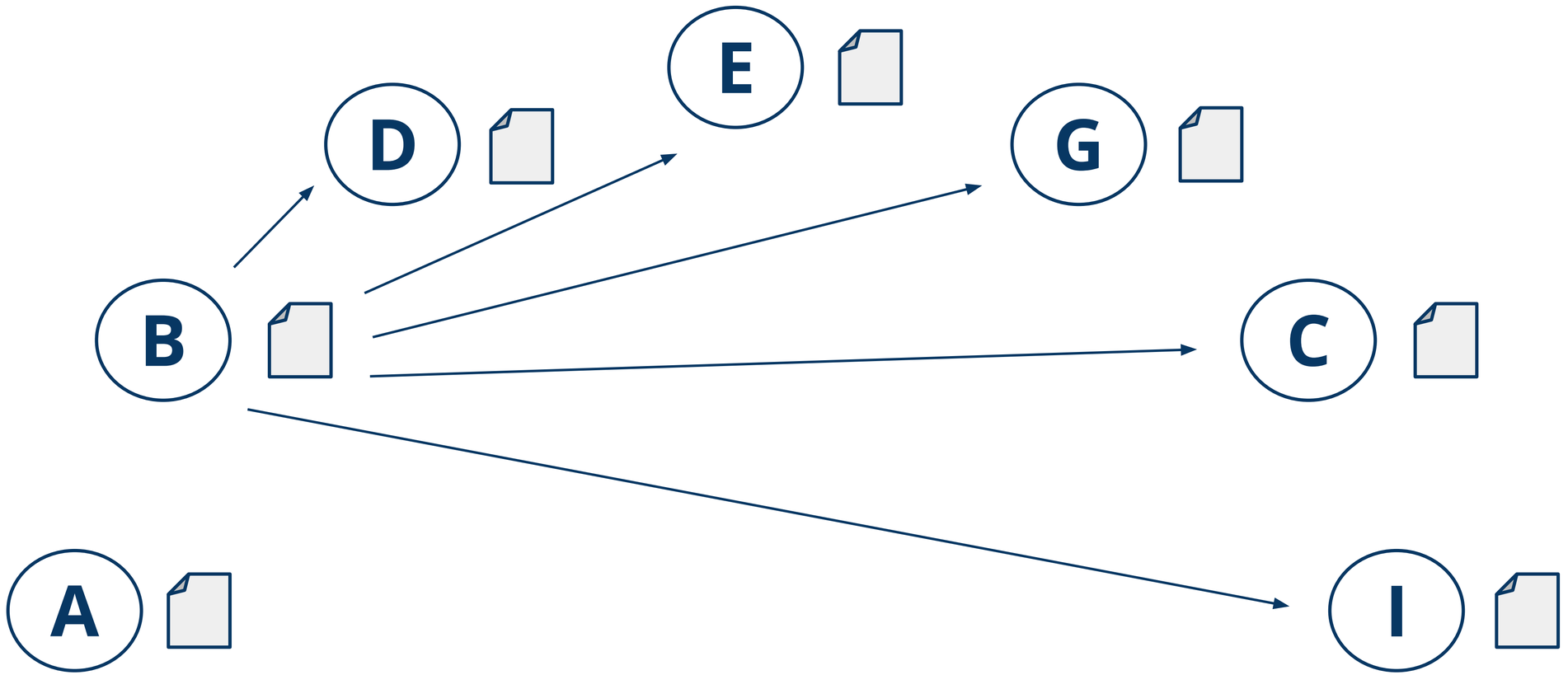
v3

Alice sends her transaction to Bob



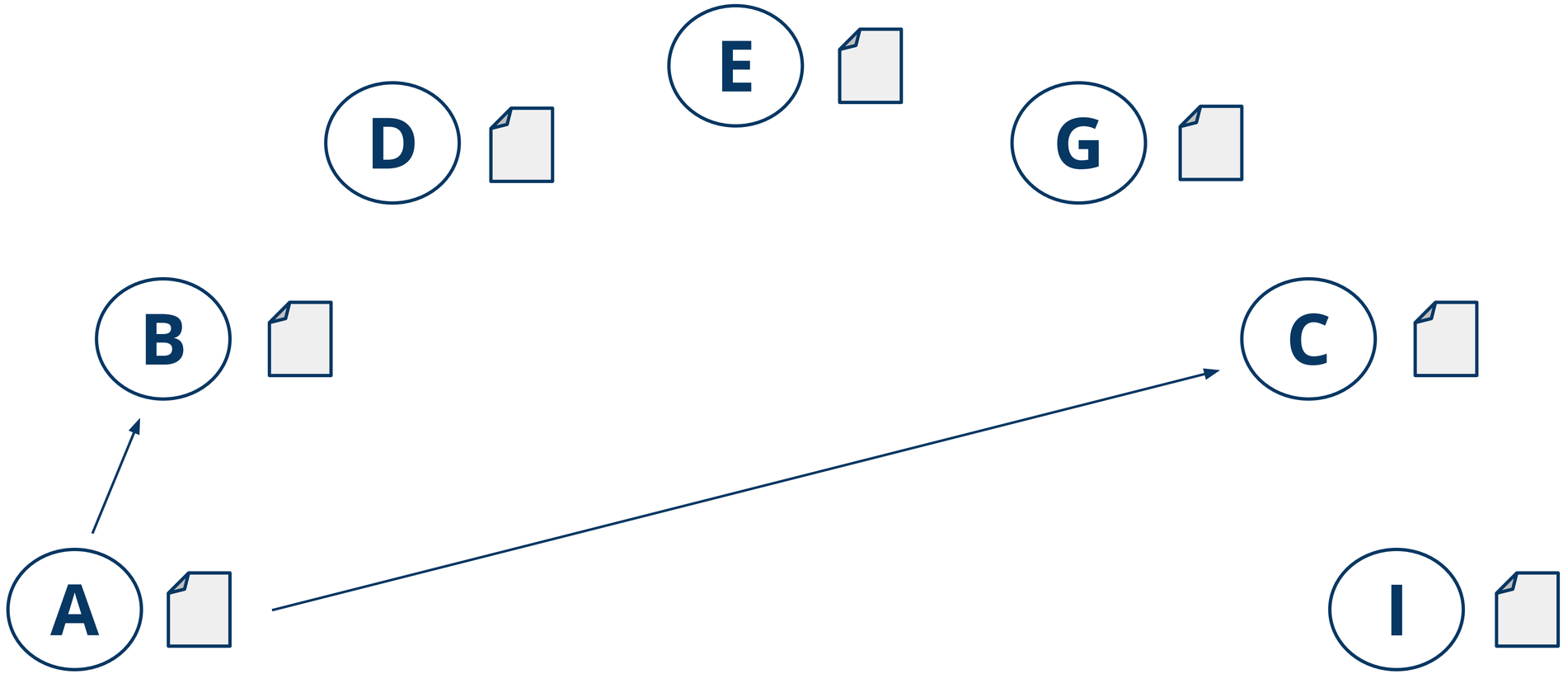
v3

Bob announces the transaction to the world



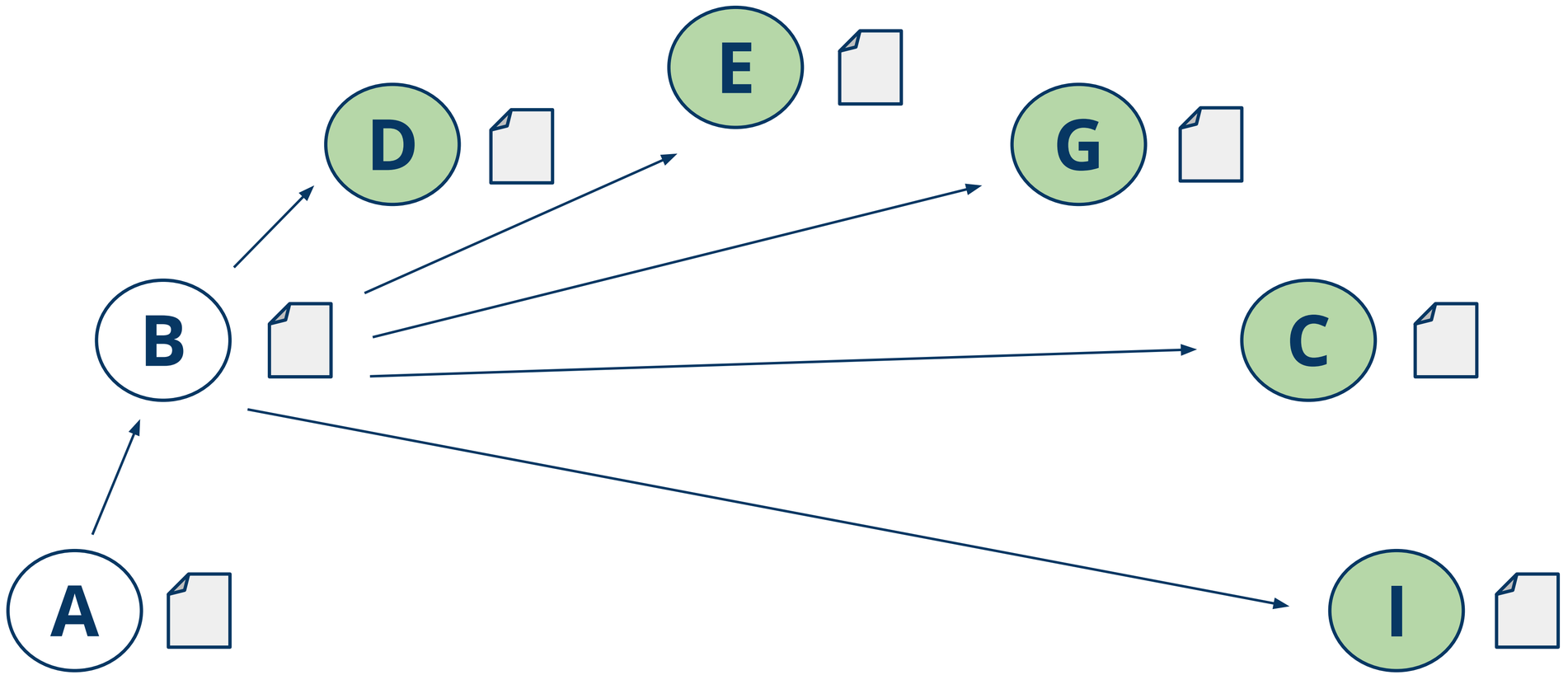
v3

Alice double spends on Bob and Charlie



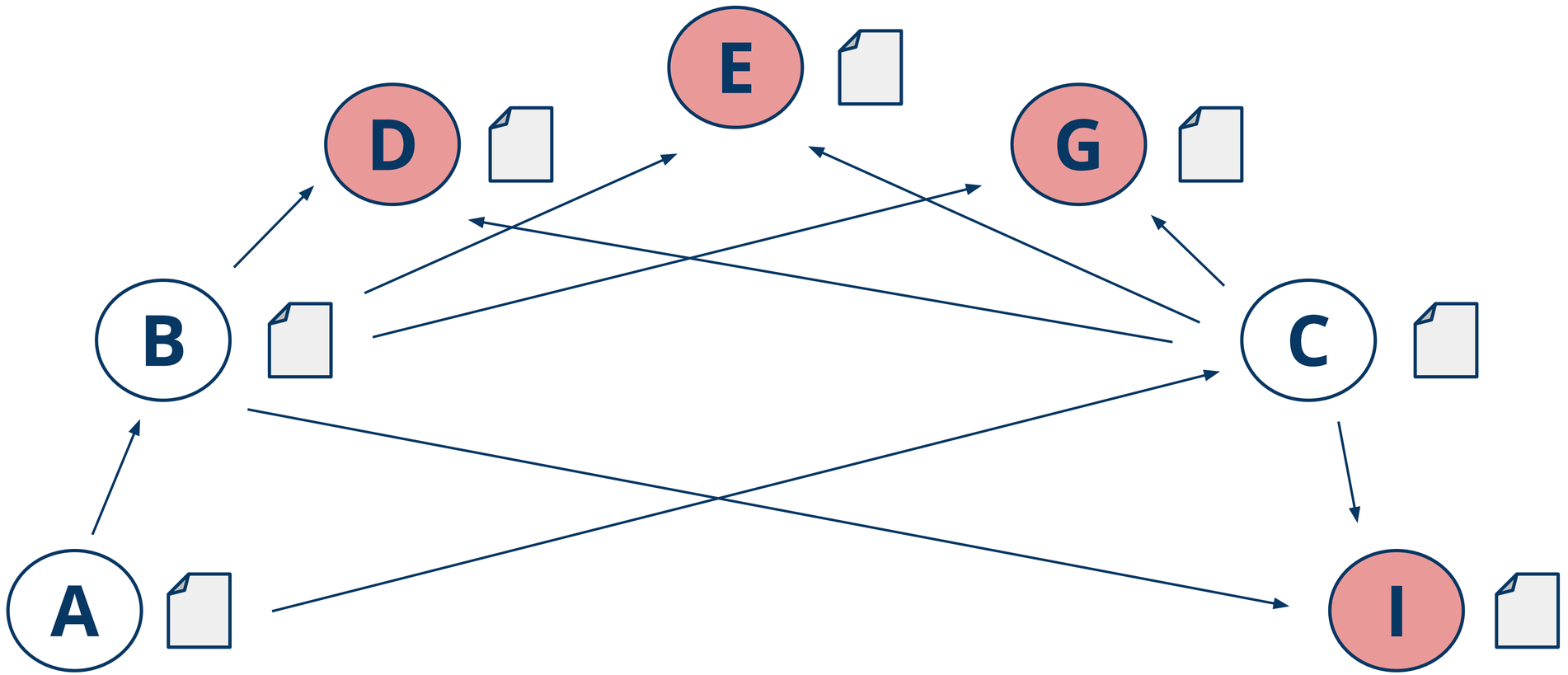
v4

Everyone verifies transactions



v4

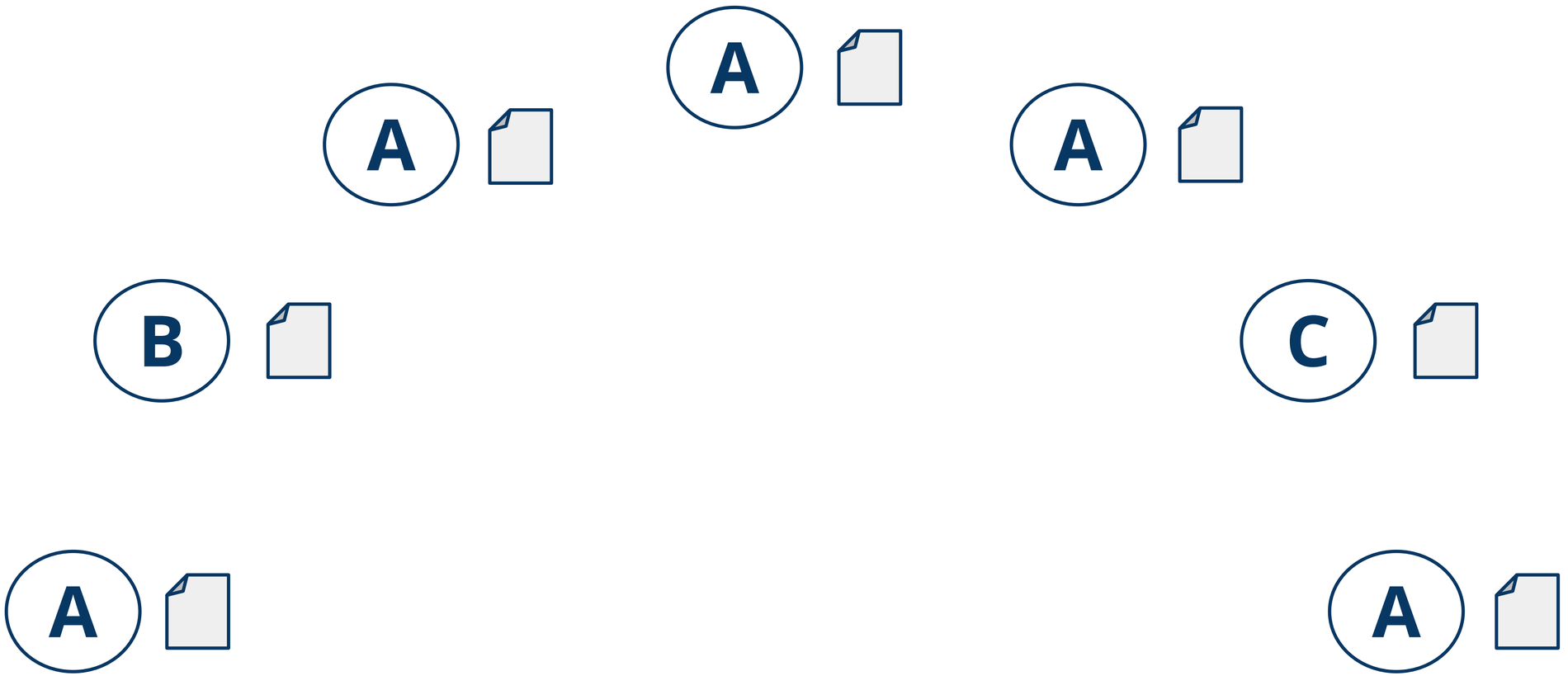
Alice is prevented from double spending



Slide by Viget

v4

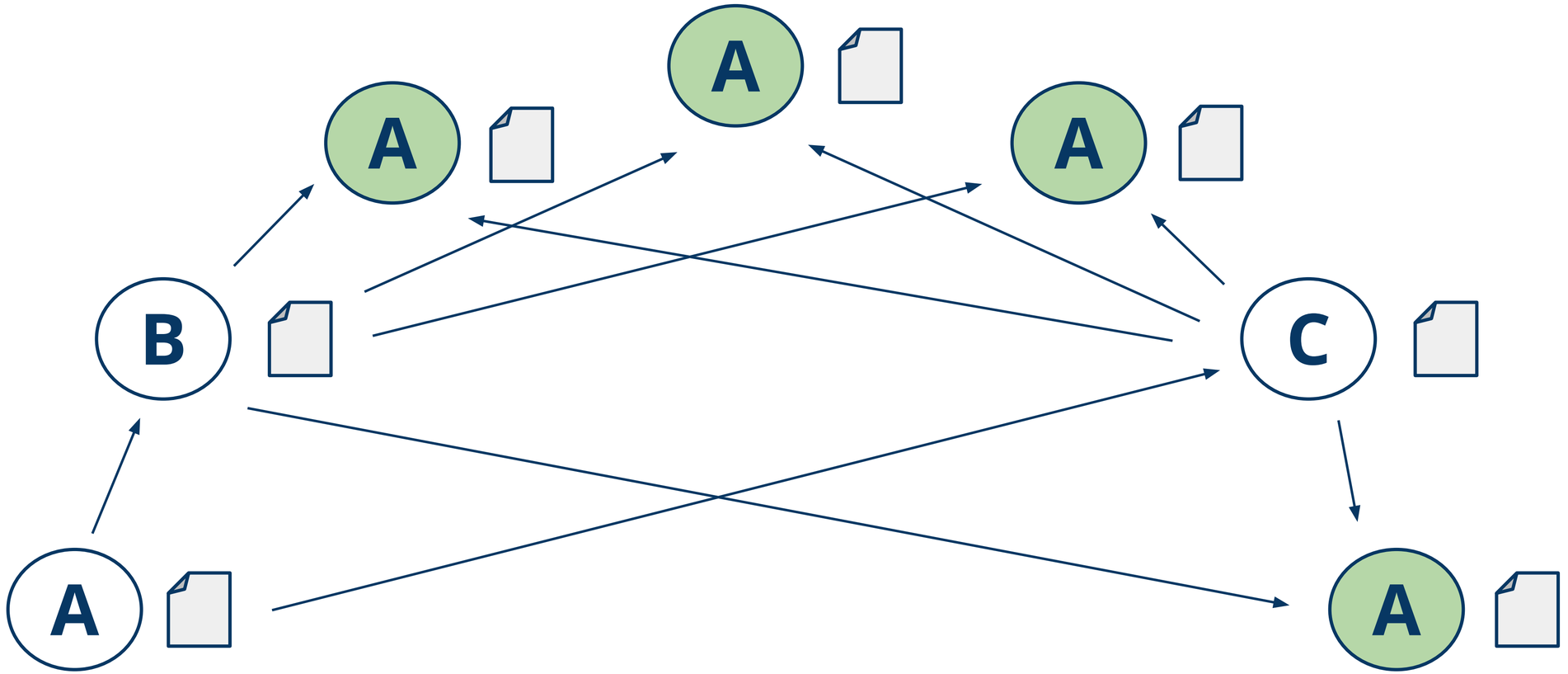
Alice sets up multiple identities



v4

Alice double spends with her multiple identities

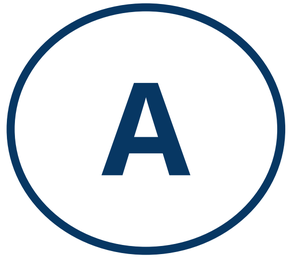
Sybil Attack: Done by creating many fake identities

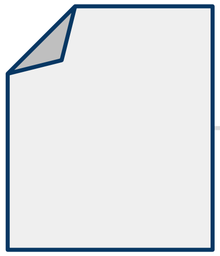


Slide by Viget

v5

Proof-of-work





v5

Pending transactions

1. I, Tom, am giving Sue one bitcoin, with serial number 3920.
2. I, Sydney, am giving Cynthia one bitcoin, with serial number 1325.
3. I, Alice, am giving Bob one bitcoin, with serial number 1234.

Slide by Viget

v5

Verifying transactions

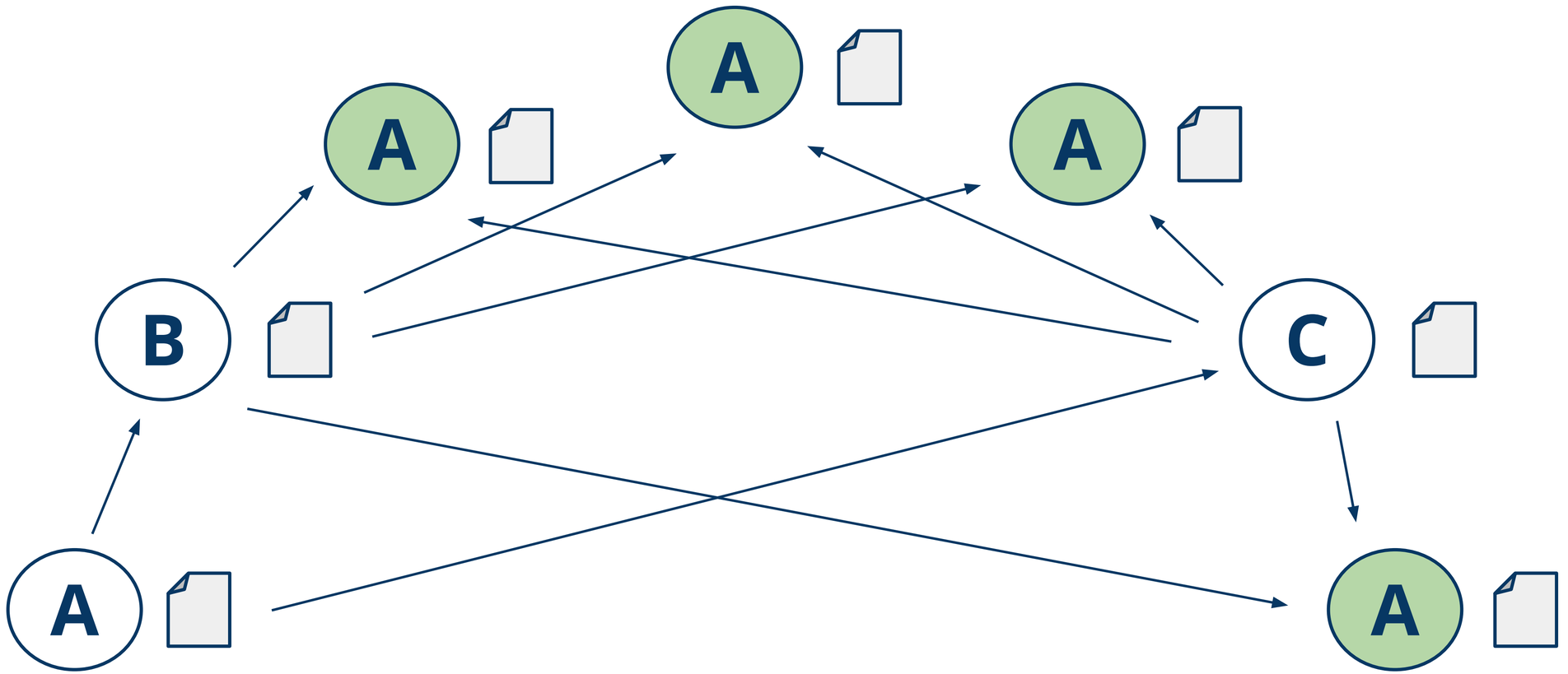
1 2 3

v5

Why the math?

v4

Alice double spends with her multiple identities



Slide by Viget

v5

Proof-of-work as a competition

Summary

Version	Major feature	Value added
1	Signed messages announced to the network	Basis of entire system
2	Serial numbers	Uniquely identifiable transactions
3	The block chain	Shared record of transactions
4	Everyone verifies transactions	Increased security
5	Proof-of-work	Prevents double spending

Consensus Algorithms: Outline

- What is Consensus?
- Alternatives to proof of work?
- Consensus + ? = Decentralized Payment Network
- Byzantine Fault Tolerance (BFT)
- Federated Byzantine Agreement (FBA)
- The Stellar Consensus Protocol (SCP)

What is Consensus?

- In order to prevent double-spending, the bitcoin network needs to reach consensus on the ledger
- The consensus mechanism needs to be “stable”
  - what does that actually mean?
- “system will continue to behave in a way that facilitates a functional currency as it grows and participants attempt novel attacks”

An Attempt of Defining Stability of Bitcoin

- Requirements for Stability:
  - Eventual Consensus
  - Exponential convergence
  - Liveness
  - Correctness (?)
  - Fairness

Is Bitcoin “stable”?

- Open question
- Some say: “Bitcoin is broken in theory but works in practice”
- Some thoughts to question your trust in Bitcoin:
  - Temporary block withholding attack
  - Collusion
  - Diminishing block rewards
  - Seizable external shorting markets → “Goldfinger attack”
  - Cloud mining
  - Mining pools
  - Blackmailing
  - Liquidity

Alternatives to proof of work

- PoW destroys valuable resources (energy + computation).
  - really just like mining gold.
- Idea: Move to PoW that solves useful problems (e.g. finding prime numbers, protein folding, …) instead of SHA-256
- Other concept: virtual mining / proof of stake
  - proof of coin age
  - proof of deposit
  - proof of burn
  - proof of activity
- But: Nothing-at-stake problem
  - workarounds?

Consensus + ? = Decentralized Payment Network

- On top of consensus we need:
  - Way to distribute currency fairly. Options:
    - Mining: Wasting resources to get some of a limited resource (similar to gold)
    - Distribution through founders: Organizations that benefit the project get paid in currency. Like crowdfunding.
  - Provide intrinsic incentives for good behavior
    - need to loop back to real world
  - Tell you whom to trust
    - “Majority” doesn’t work - sybil attacks
    - intrinsic incentives could be the solution

Alternatives to the Bitcoin Consensus Mechanism

Some slides adopted form David Mazières. Errors are my own.

Byzantine fault tolerance

- BFT has its roots in systems design: Networks of unreliable processors
  - Word root: Illustration through the “Byzantine Generals' Problem”
- Byzantine failure: Most nasty type of failure, i.e. arbitrary misbehaving, including pretending to behave correctly.
- Contrast with “fail-stop”

Properties of a Consensus Mechanism

- Safety
  - Agreement – All outputs produced have the same value
  - Validity – The output value equals one of the agent's' inputs
- Liveness
  - Termination – Eventually non-faulty agents output a value
- Fault tolerance
  - It can recover the failure of an agent at any point
  - Fail-stop protocols handle agent crashes
  - Byzantine-fault-tolerant protocols handle compromised agents
- Impossibility result
  - (safety, liveness, fault tolerance) - choose two

Some terminology

- Univalent state: When only one remaining output is possible. If that value is **a**, we call it **a**-valent
- Bivalent state: Network can influence outcome of protocol.
- Stuck state: When a non-faulty node can never output a value.

Straw Man

- Straw man: Vote on consensus value
  - pick quorum size T > N/2
  - any two quorums intersect, votes cannot change => agreement
- Problem:
  - node failure:
  - outage → stuck or
  - byzantine misbehavior → violation of safety

So that doesn't work. What to do next?

- How to avoid getting stuck?
  - 1. Have irrefutable statements: An irrefutable statement is one that correct nodes never vote against.
  - 2. Have statements whose hold on consensus question can be broken.
- Important because fault tolerance requires neutralizing deciding messages (theorem)
- A neutralizable statement is one that can be rendered irrelevant to the consensus question.

Ballot-based neutralization (Paxos)

- Solution to (1): Ballot-based neutralization (Paxos)
- Vote to commit or abort ballots
- Each ballot is <n,x> where n is a counter and x a candidate output value
- If a quorum votes to commit <n,x> for any n, it is safe to output x.
- Invariant: all commiteted and stuck ballots must have the same x.
- To preserve: Cannot vote to commit a ballot before preparing it.
- - Prepare <n,x> by aborting all <n',x'> with n' < n and x' != x.
- - Concisely encode whole set of abort votes with PREPARE message
- If ballot <n,x> is stuck, neutralize by restarting with <n+1, x>

Byzantine agreement

- In fail-stop case it was sufficient that any two quorums share a node.
- Now, it needs to be ensured that any two quorums share a **non-faulty** node

- Safety requires f_S <= 2T - N -1
- Liveness requires f_L <= N - T

- i.e. at least one entirely non-faulty quorum exists

- Solution: Castros '99:
  - N = 3f + 1, T = 2f + 1 to tolerate f_S = f_L = f failures

When has a vote succeeded?

- When has a vote succeeded?
- System reaches a-valent state when T votes for a are seen. But how does it reach a-agreed state?

- f_S + 1 notes all claim to have seen T votes for a
- → Can assume system is a-valent with no loss of safety.
- i.e. the remaining nodes able to convince rest for liveness.

Federated Byzantine Agreement (FBA)

- Remember: Sybil attack; so quorum by size doesn't work.
- → participants need to specify their trusted quorums

- That's FBA: BA without magically blessing N nodes

- Each node v picks one or more quorum slices where v trusts in all its slices
- v requires any quorum to contain at least one of its slices

FBSA

- **Def**: A **FBSA** (Federated Byzantine Agreement System) is a set of nodes V and a quorum function Q where Q(v) is the set slice chosen by node v.

- **Def**: A **quorum** $U \subseteq V$ is a set of nodes that contains at least one slice of each of its members, i.e. $\forall\ v \in V\ \exists\ S \in Q(v)$ s.th. $S \subseteq U$

- What is necessary to guarantee safety?
  - If there are two entirely disjoint quorums, they can make progress without communication from other
  - Therefore, safety requires quorum intersection.

FBSA cont.

- **Def**: An FBAS enjoys quorum intersection when every two quorums share at least one node.
- This is not sufficient with byzantine failure. Why?
- We need quorum intersection despite ill-behaved nodes, i.e. after deleting all ill-behaved nodes, the network continues to provide quorum intersection.

FBSA cont.

- What about liveness?
- **Def**: A v-blocking set contains at least one node from each of v's slices. It is impossible to guarantee liveness for v, if a v-blocking set has crashed.

- Hence, for system-wide liveness, all failed nodes cannot be v-blocking for any correct node v.

The Stellar Consensus Protocol (SCP)

- First general FBA protocol
  - Guarantees safety for well-behaved nodes that enjoy quorum intersection despite ill-behaved nodes
  - → i.e. best possible protocol
  - Guarantees that intact nodes will not get stuck
- Core idea: federated voting
  - Nodes exchange vote messages to agree on statements
  - Every message also specifies the voter's quorum slices
  - Allows dynamic quorum discovery while assembling votes

SCP cont.

- Problems compared to BFT:
  - Failure is per node, not system-wide
  - The "pass on" function we used in fixed systems doesn't work because nodes might not care about ratifications.
- Solution: Introduce notion of accepting (weaker that ratifying).
  - A node v accepts a statement a consistent with history if each member of a v-blocking set claims to accept a.
- Problems:
  - No guarantee that all intact nodes can accept a statement
  - Suboptimal safety for non-intact nodes enjoying quorum intersection

SCP cont.

- Confirming
  - Idea: Hold a second vote on the fact that the first vote succeeded
  - A quorum confirms a statement **a** by ratifying the satement "we accepted **a**".
- Solves both problems

- As soon as a node confirms **a**, all other intact nodes will eventually confirm **a**.

SCP cont.

- What SCP is not: SCP does not
  - Offer a rate-limited way to distribute new digital coins
  - Provide intrinisc incentives for good behavior
  - Tell you whom to trust
- i.e. SCP is not a Decentralized Payment Network

Other Properties of Consensus Mechanisms to think about

- Trust in participants
  - why should I trust an entity? Rational incentives?
- Mutability
  - Can a decision be reverted or is it set in stone?
- Latency, Throughput
- Stability
  - what happens in unexpected circumstances
- Flexibility / updatability
  - is it future proof?

Bitcoin Scripting

Contents of a Bitcoin Transaction

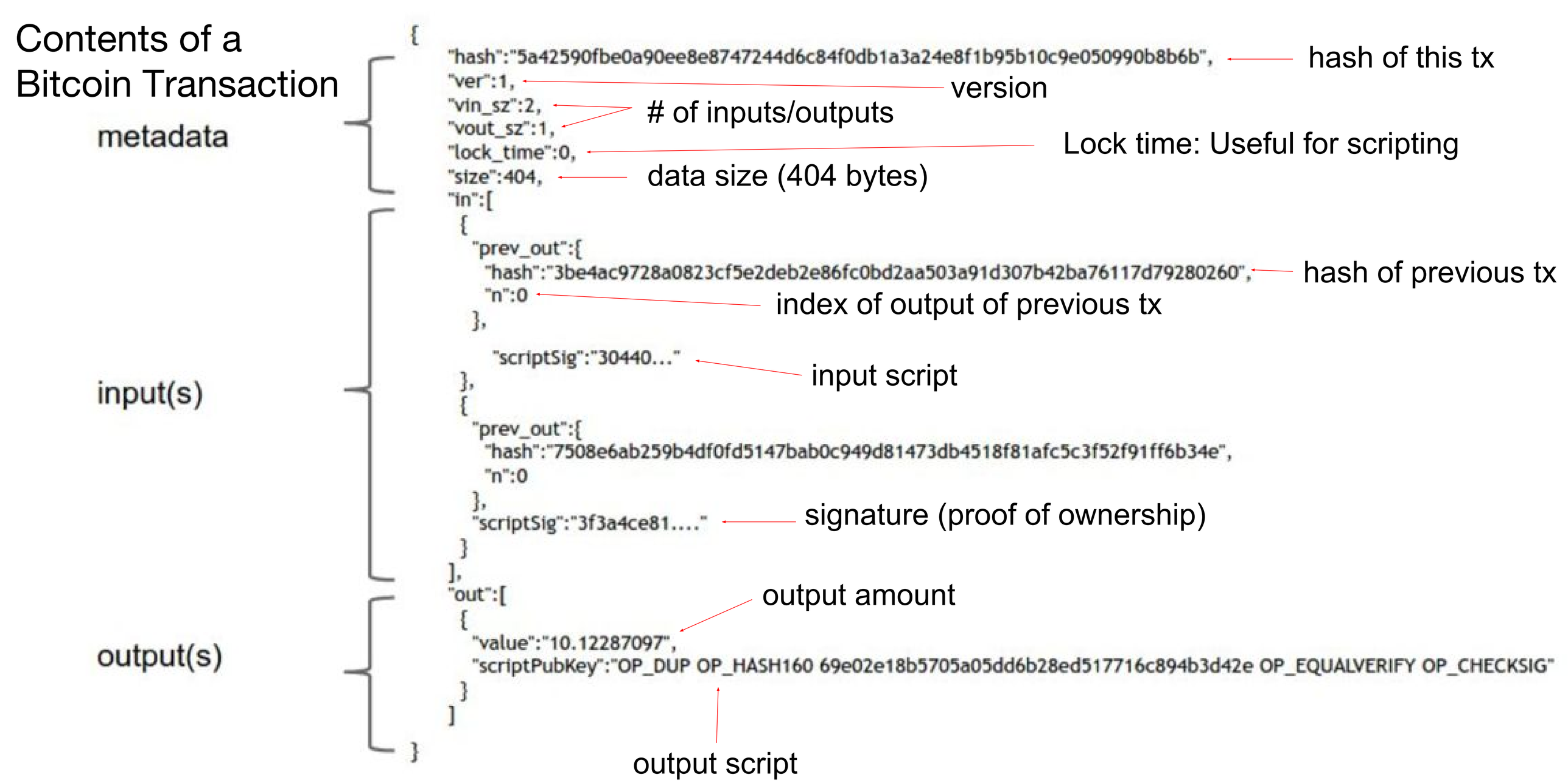
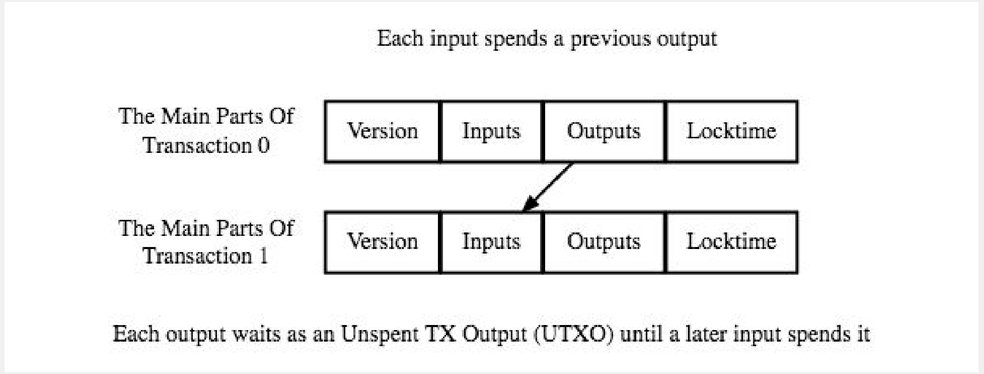


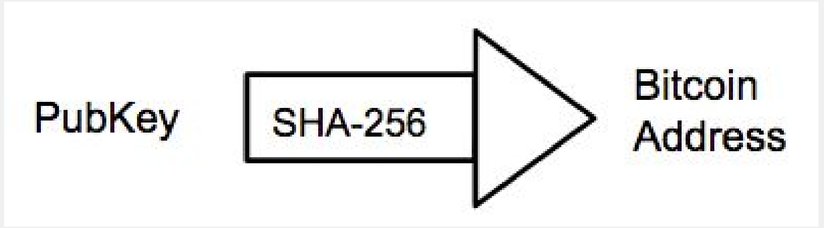
Figure 3.3 **An actual Bitcoin transaction.**

Bitcoin Scripting

(Bitcoin.org developer guide)



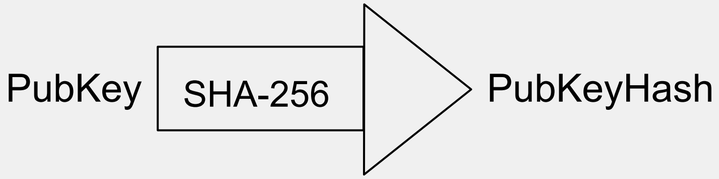
Output "addresses" are really scripts.
Inputs and outputs through scripting allows for future extensibility of Bitcoin.

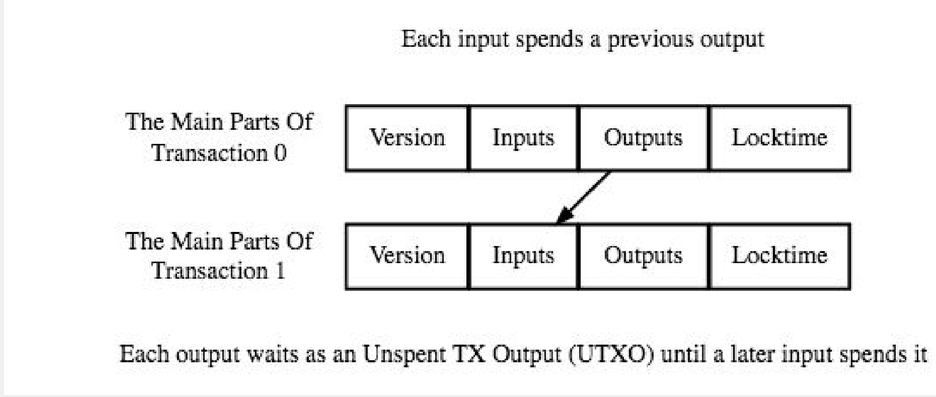


- Remember:
  - **Signatures** prove ownership of a public key
  - **Bitcoin Address == Hash(PubKey)**
  - Transaction composed of input and output **addresses**
  - Spending Bitcoin is **redeeming** previous transaction outputs
- "This amount can be redeemed by a **signature** from owner of address X"
  - No way to get the public key from the address!
- "This amount can be redeemed by the **public key** that hashes to address X, plus a **signature** from the owner of that public key"

(Quotes from Princeton textbook)

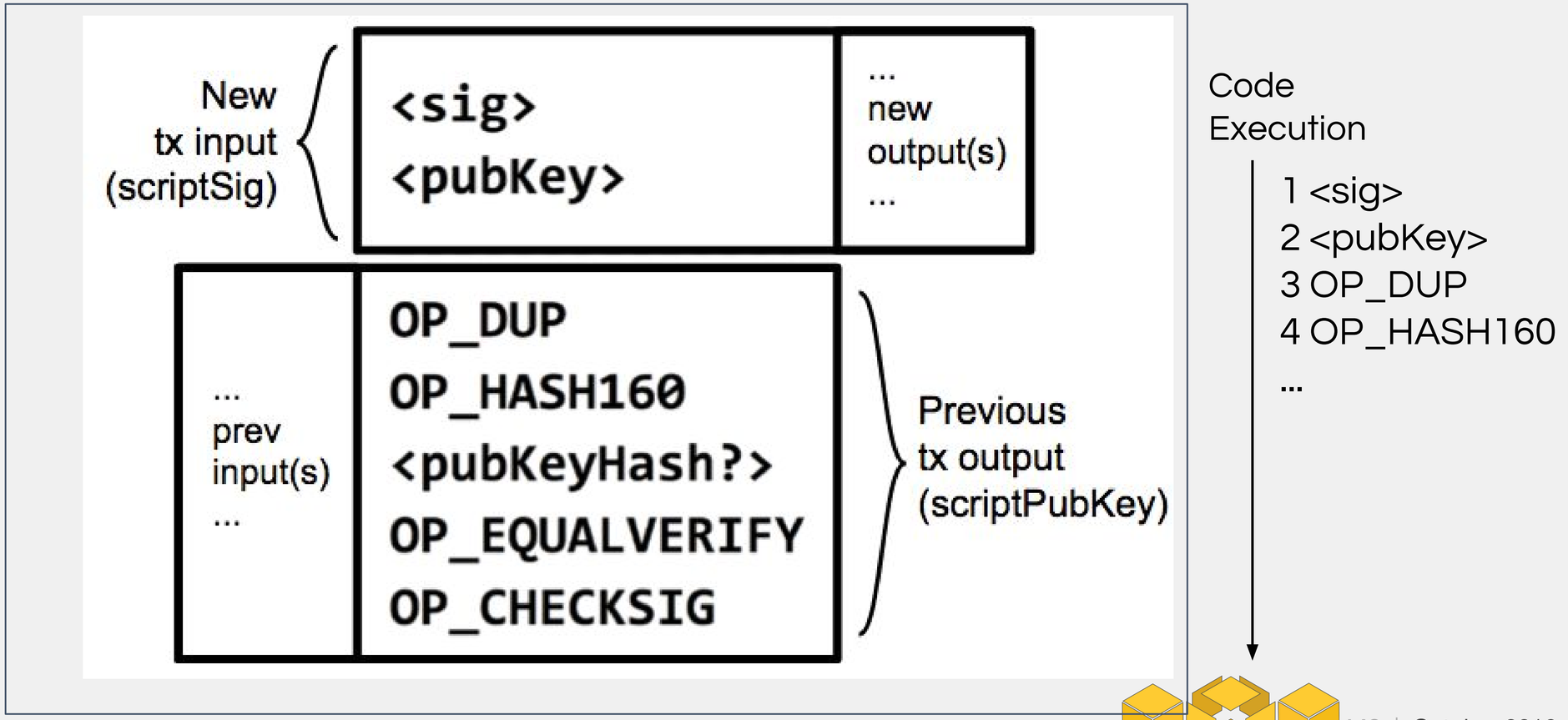
Bitcoin Scripting





Remember: Hash(PubKey) == Address == "PubKeyHash"

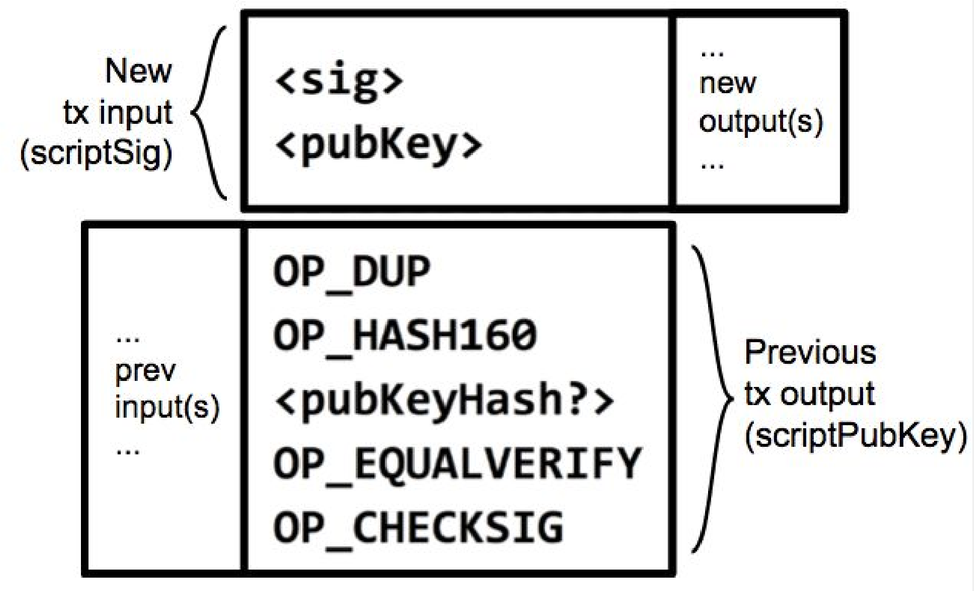
Figure:
Two transactions along with their input and output scripts



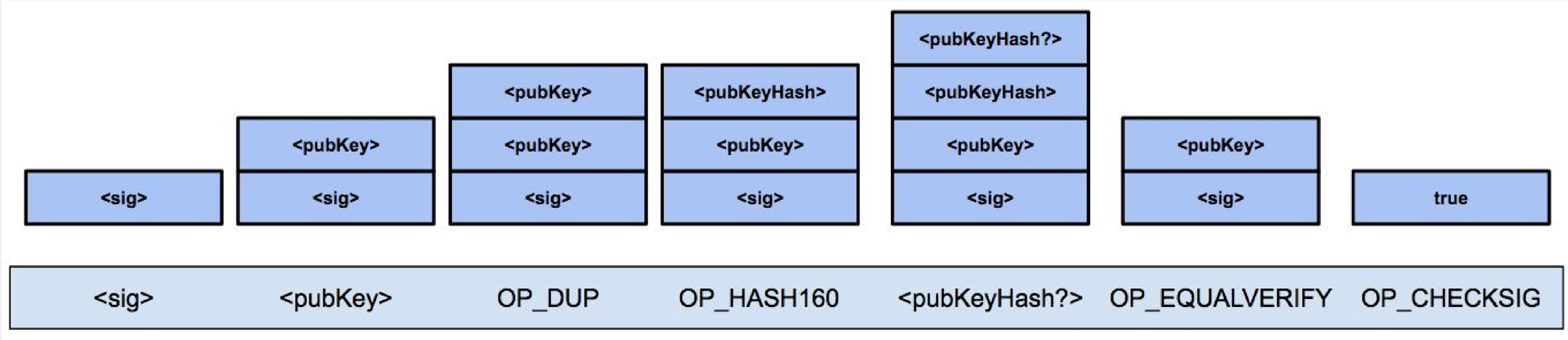
Bitcoin Scripting

Language built specifically for Bitcoin called "Script" or simply "the Bitcoin scripting language"

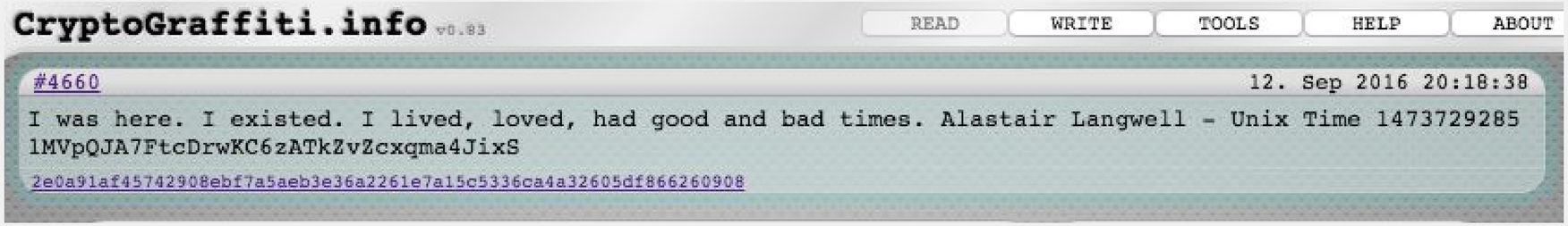
- Stack based
- Native support for cryptography
- Simple
  - No loops



Output says: "This amount can be redeemed by
1) the **<pubKey>** that hashes to address **<pubKeyHash?>**
2) plus a **<sig>** from the owner of that **<pubKey>**
...that will make this script evaluate to **true**."



Proof of Burn



How to write arbitrary data into the Bitcoin blockchain?

Output script:

OP_RETURN
<arbitrary data>
<dirks>

Proof of Burn

- OP_RETURN throws an error if reached
- Output script can't be spent, therefore you prove that you destroyed some currency
- Anything after OP_RETURN is not processed

Use cases

- Prove existence of something at a particular point in time
  - Ex. A word you coined, hash of a document/music/creative works
- Bootstrap TobiasCoin based off of Bitcoin by requiring that you destroy some Bitcoin to get TobiasCoin

Pay-to-PubkeyHash vs. Pay-to-Script-Hash

Need to clarify important detail (useful for developers): In Bitcoin, senders specify the script.

Previous example was **P2PKH**: Vendor says "Send your coins to the hash of this **PubKey**."

- Simplest case
- Is by far the most common case

But for complicated scripts (such as multisig), this can be a problem.

Ex. Vendor says "To pay me, write a complicated output script that will allow me to spend using multiple signatures."

How would the sender know?

Pay-to-PubkeyHash vs. Pay-to-Script-Hash

Solution? Pay-to-Script-Hash (P2SH)

P2PKH: "Send your coins to the hash of this **PubKey**."

P2SH: "Send your coins to the hash of this **script**. To redeem those coins, you must reveal the script that has the given hash and provide **data** that will make the script evaluate to true."

- Offloads complicated script writing to recipients
  - Makes more sense from a payer-payee standpoint
    - Merchant doesn't burden customer with writing a complicated script
    - Customer doesn't care what the script actually is
      - Merchant responsible for writing correct and secure script
    - Customer shouldn't need to know anything about how company stores funds
- P2SH is the most important improvement to Bitcoin since inception

BLOCKCHAIN

AT BERKELEY

Thanks!

We're the world's first university-based blockchain consulting firm.

Like us on Facebook:
@Berkeleyblockchain
https://www.facebook.com/berkeleyblockchain/