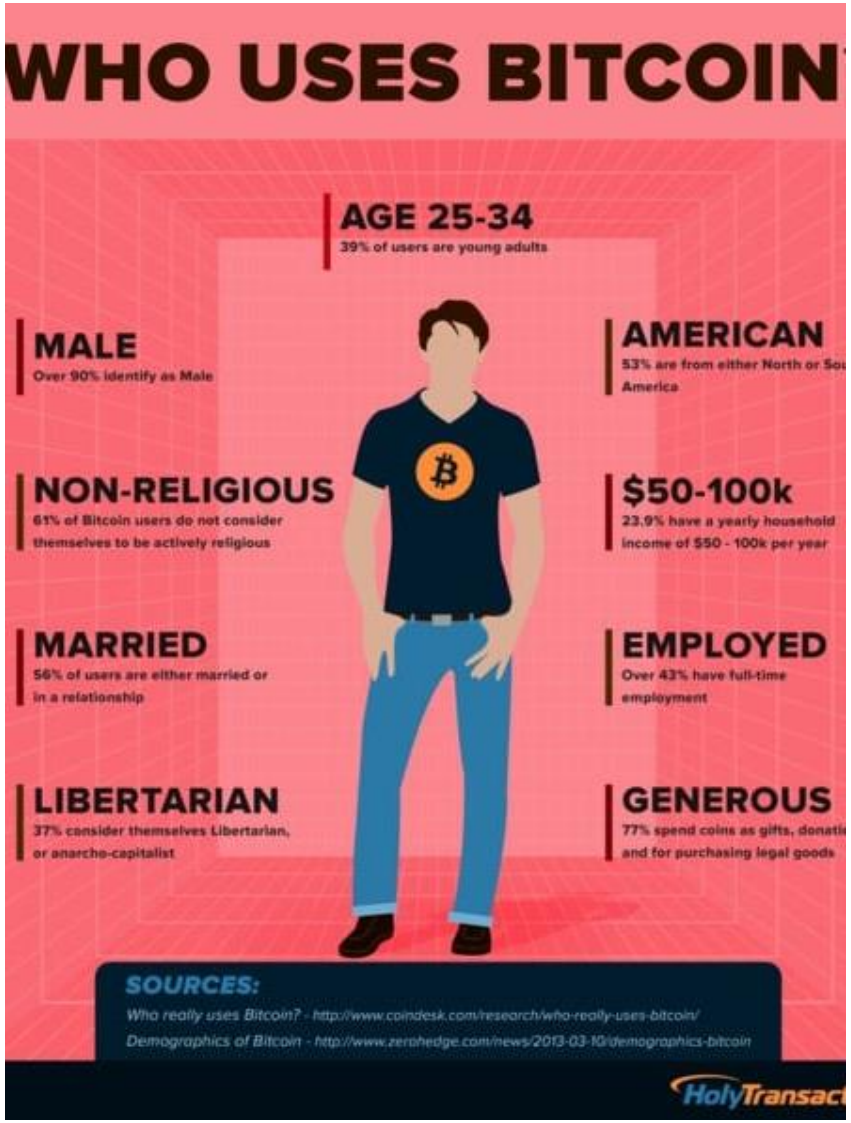
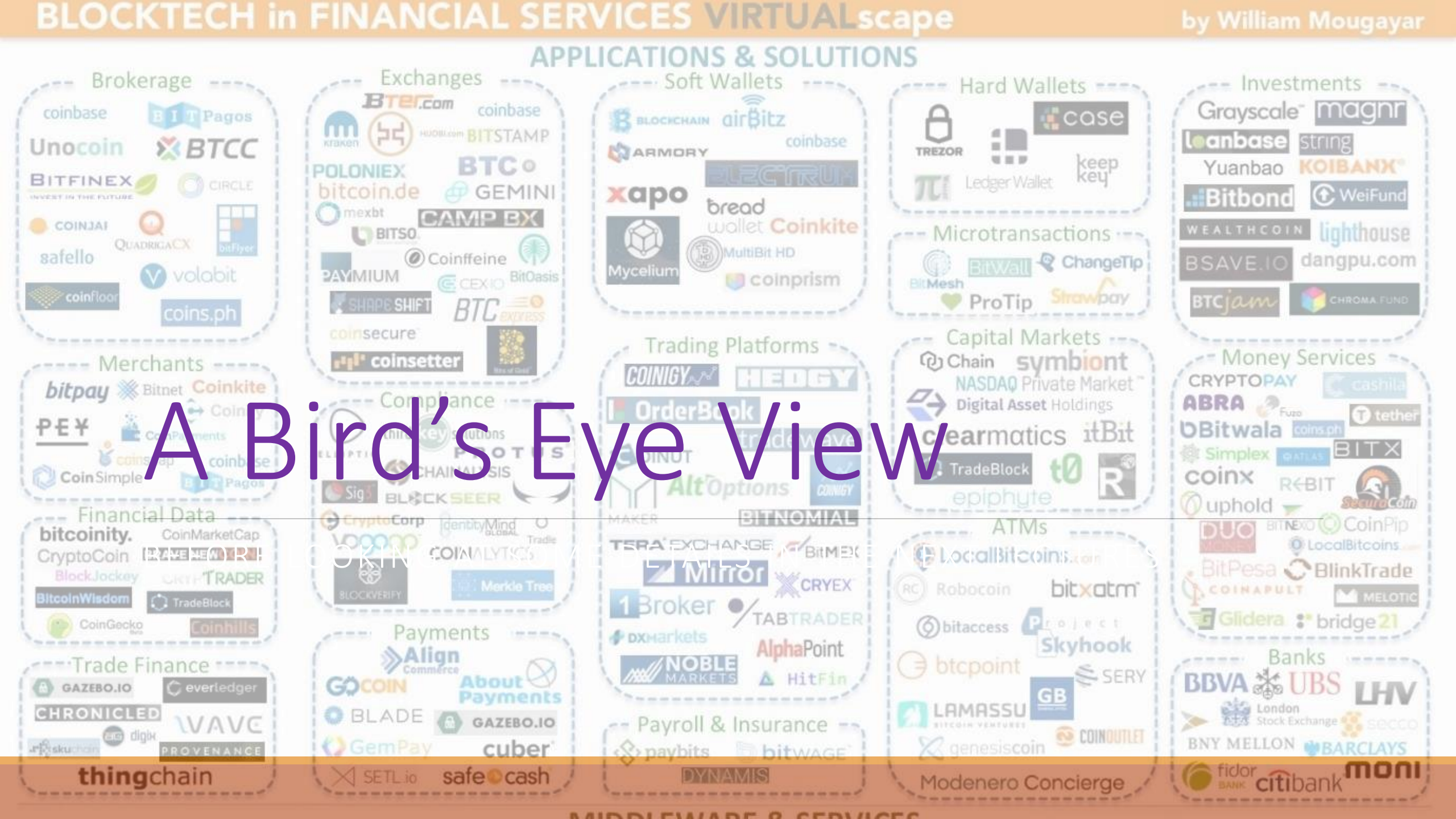


Blockchains & Bitcoin & …

SOME SLIDES ARE TAKEN FROM:

1. S. CHAKRABORTY, S. SURAL
2. P. VISWANATH

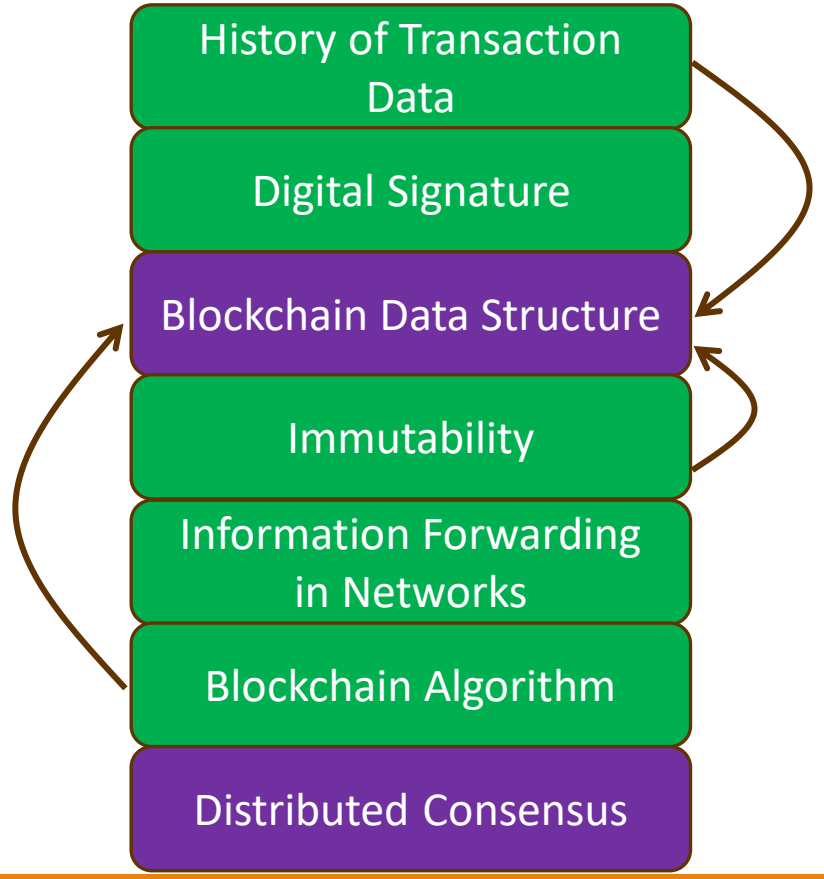

BLOCKTECH in FINANCIAL SERVICES VIRTUALscape
by William Mougayar
APPLICATIONS & SOLUTIONS
Brokerage
Exchanges
Soft Wallets
Hard Wallets
Investments
Microtransactions
Merchants
Compliance
Trading Platforms
Capital Markets
Money Services
Financial Data
ATMs
Trade Finance
Payments
Banks
Payroll & Insurance
A Bird's Eye View
MIDDLEWARE & SERVICES

Tasks for Designing a P2P System for Managing Ownership

Goal

- Describing Ownership
- Protecting Ownership
- Storing Transaction Data
- Preparing Ledgers for being Distributed
- Distributing Ledgers
- Adding New Transactions
- Deciding which Ledger Represents the Truth

Major Concept



Technical Concepts of the Blockchain and their Purpose (1)

Technical Concept	Purpose
Transaction Data	Describing Transfer of Ownership
Transaction History	Proving the Current State of Ownership
Cryptographic Hash Value	Identifying any kind of Data Uniquely
Asymmetric Cryptography	Encrypting and Decrypting Data
Digital Signature	Stating Agreement with the Content of Transaction Data
Hash Reference	A Reference that becomes Invalid once the Data being Referred are Changed
Change-Sensitive Data Structures	Storing Data in a way that Makes any Manipulation Stand out Immediately

Technical Concepts of the Blockchain and their Purpose (2)

Technical Concept	Purpose
Hash Puzzle	Imposing a Computational Expensive Task
Blockchain Data Structure	Storing Transaction Data in a Change-Sensitive way and Maintaining their Order
Immutability	Making it impossible to Change the History of Transaction Data
P2P Network	Sharing the Transaction History Among all Nodes in Network
Message Passing	Ensure that all Nodes of the System Eventually Receive all Information
BlockChain Algorithm	Ensure that only Valid Transaction Data are added to the Blockchain Data Structure
Distributed Consensus	Ensure that all Nodes of the System use the Identical History of Transaction Data
Compensation	Giving Nodes an Incentive to Maintain Integrity

Purpose of BlockChain

1. Clarifying Ownership

2. Transferring Ownership

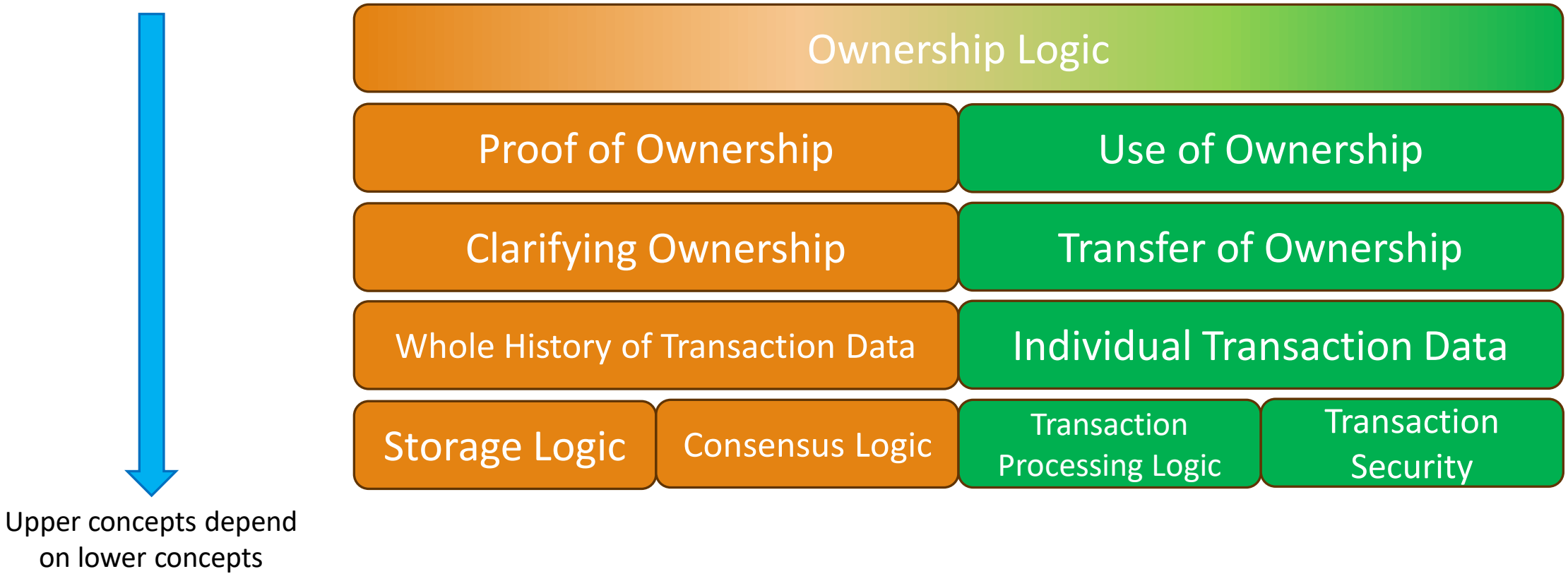
Properties of BlockChain

- Highly Available
- Censorship Proof
- Reliable
- Open
- Pseudoanonymous
- Secure
- Resilient
- Eventually Consistent
- Keeping Integrity

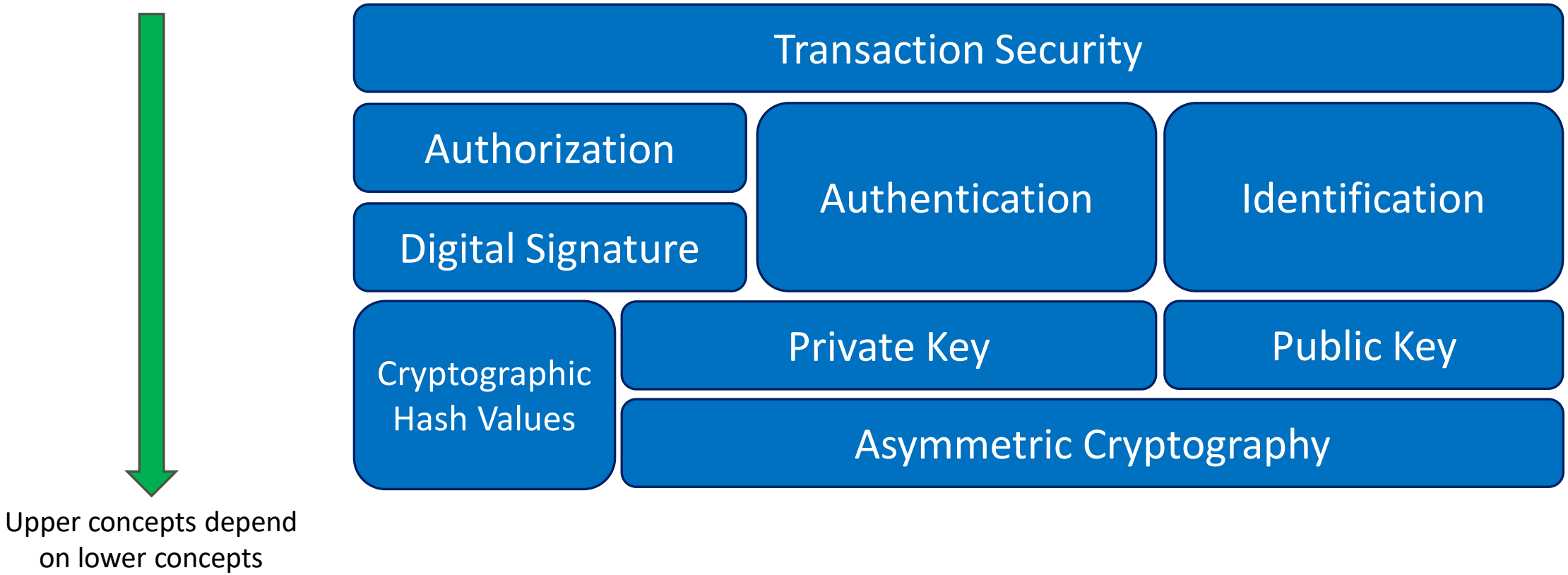
Internal Functioning of BlockChain

- Ownership Logic
- Transaction Security
- Transaction Processing Logic
- Storage Logic
- P2P Architecture
- Consensus Logic

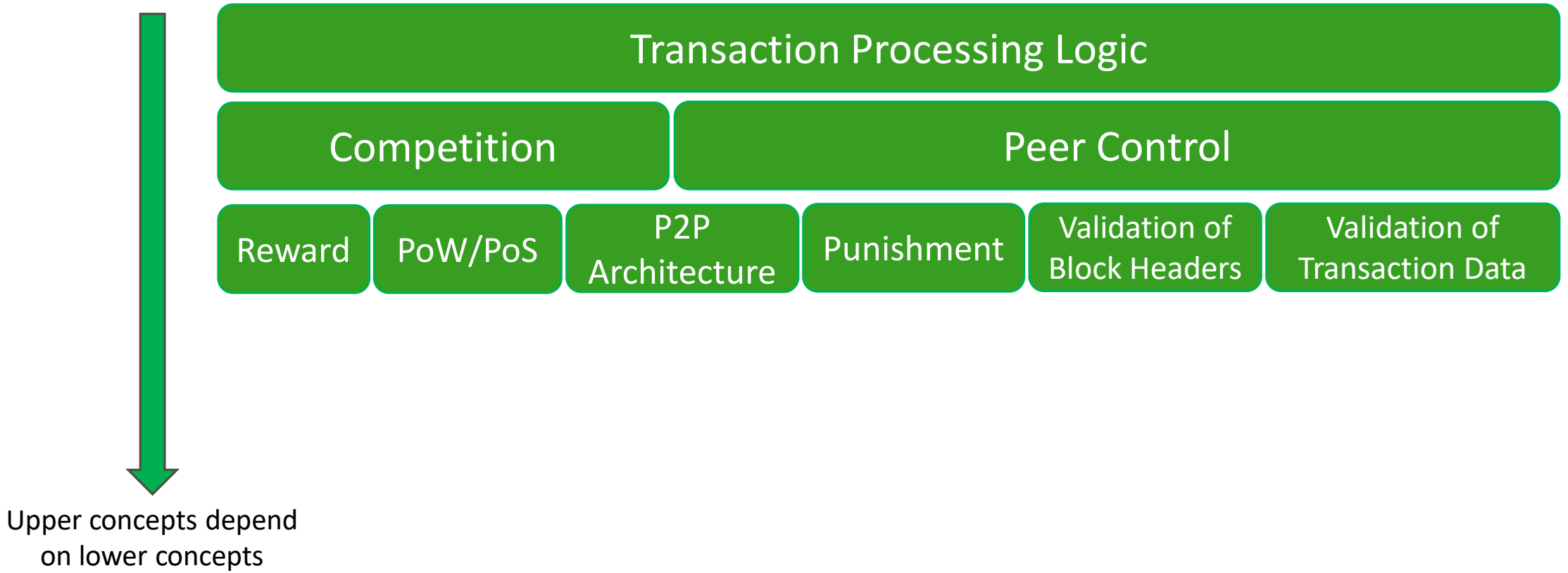
Ownership Logic



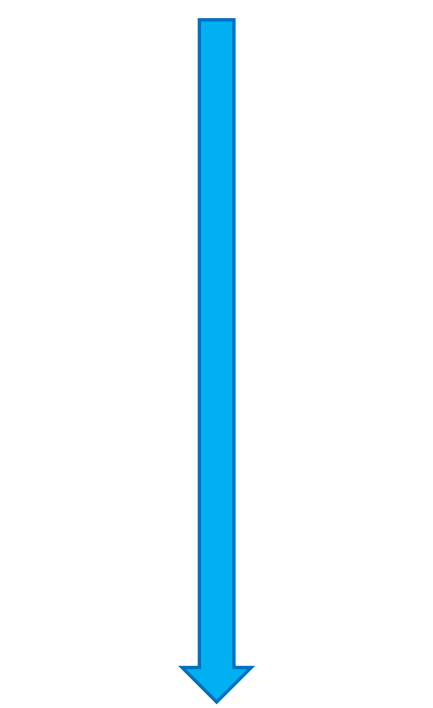
Transaction Security

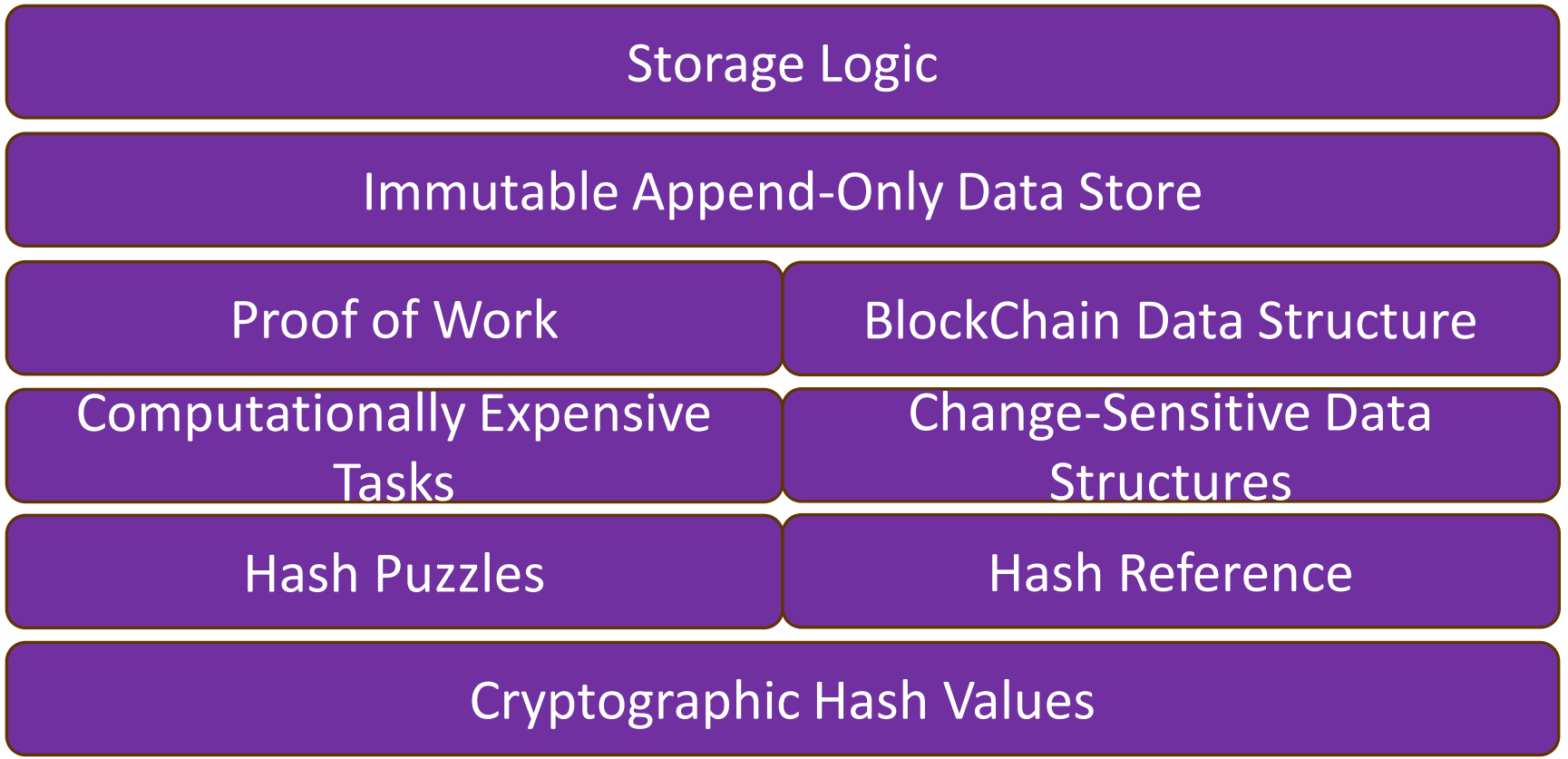


Transaction Processing Logic

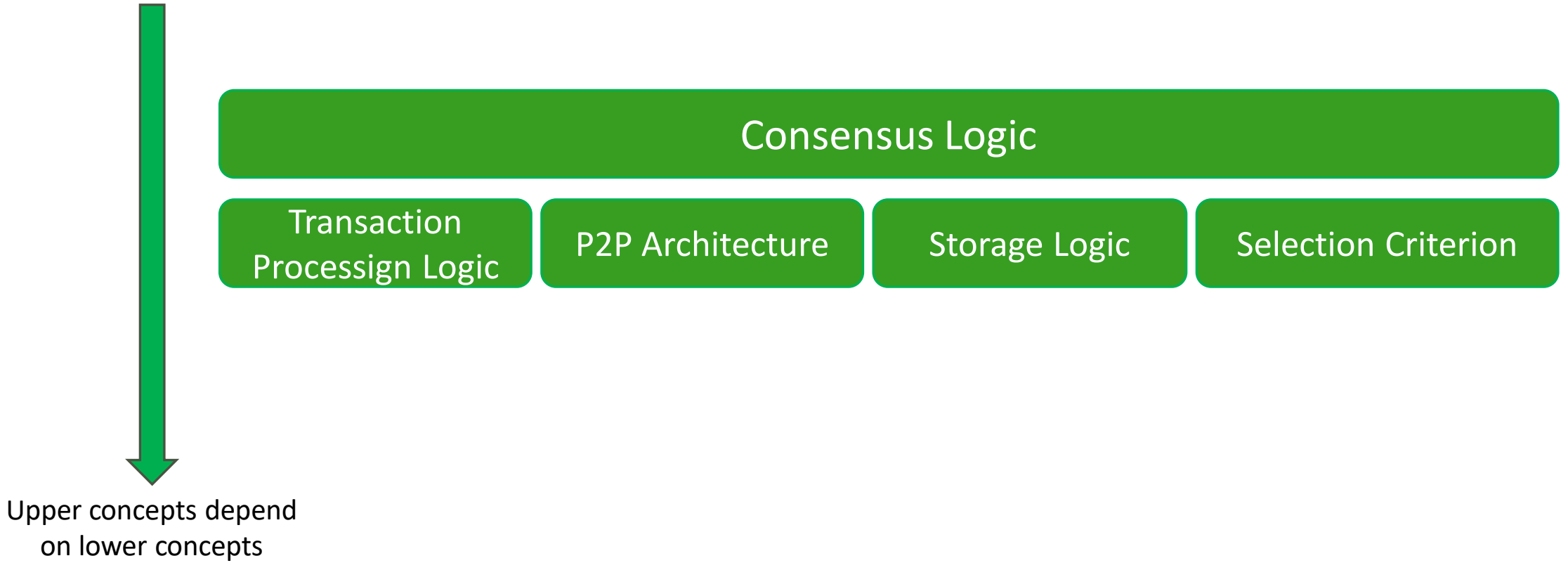


Storage Logic

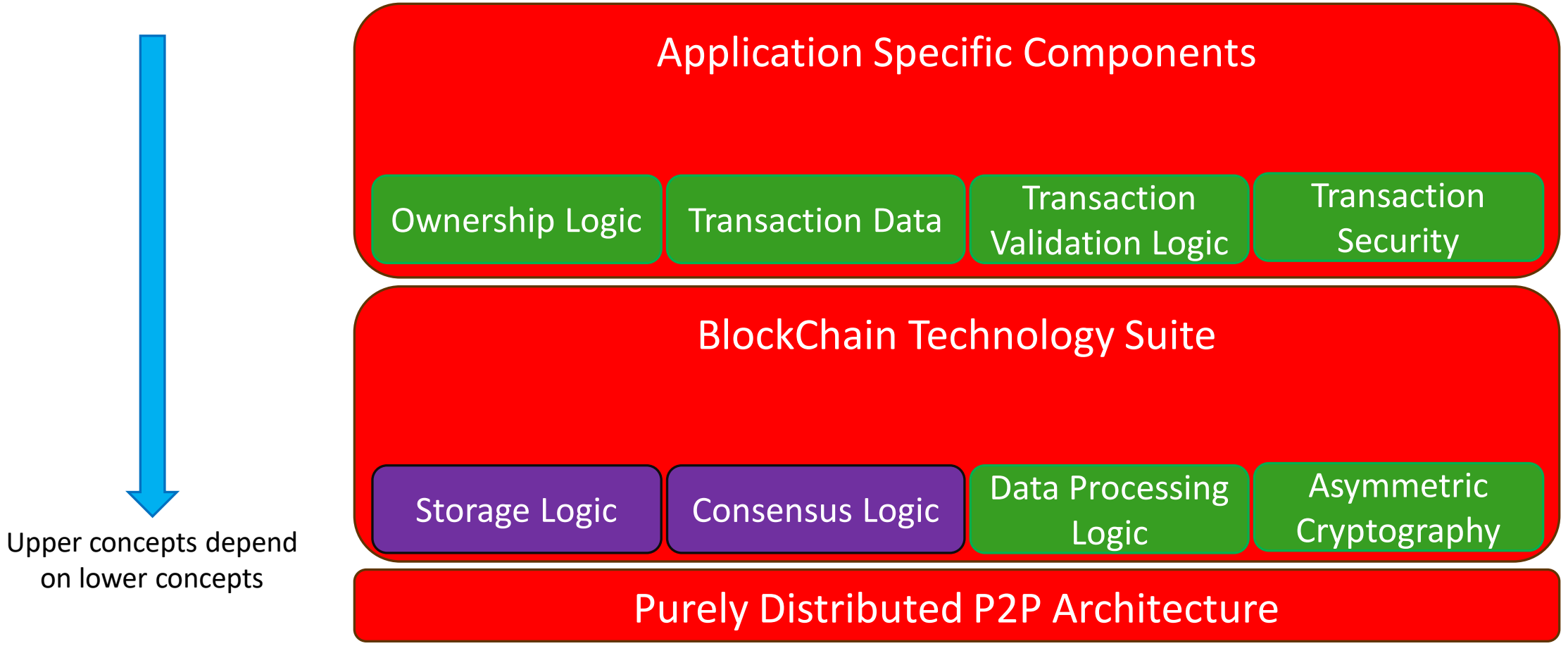




Consensus Logic

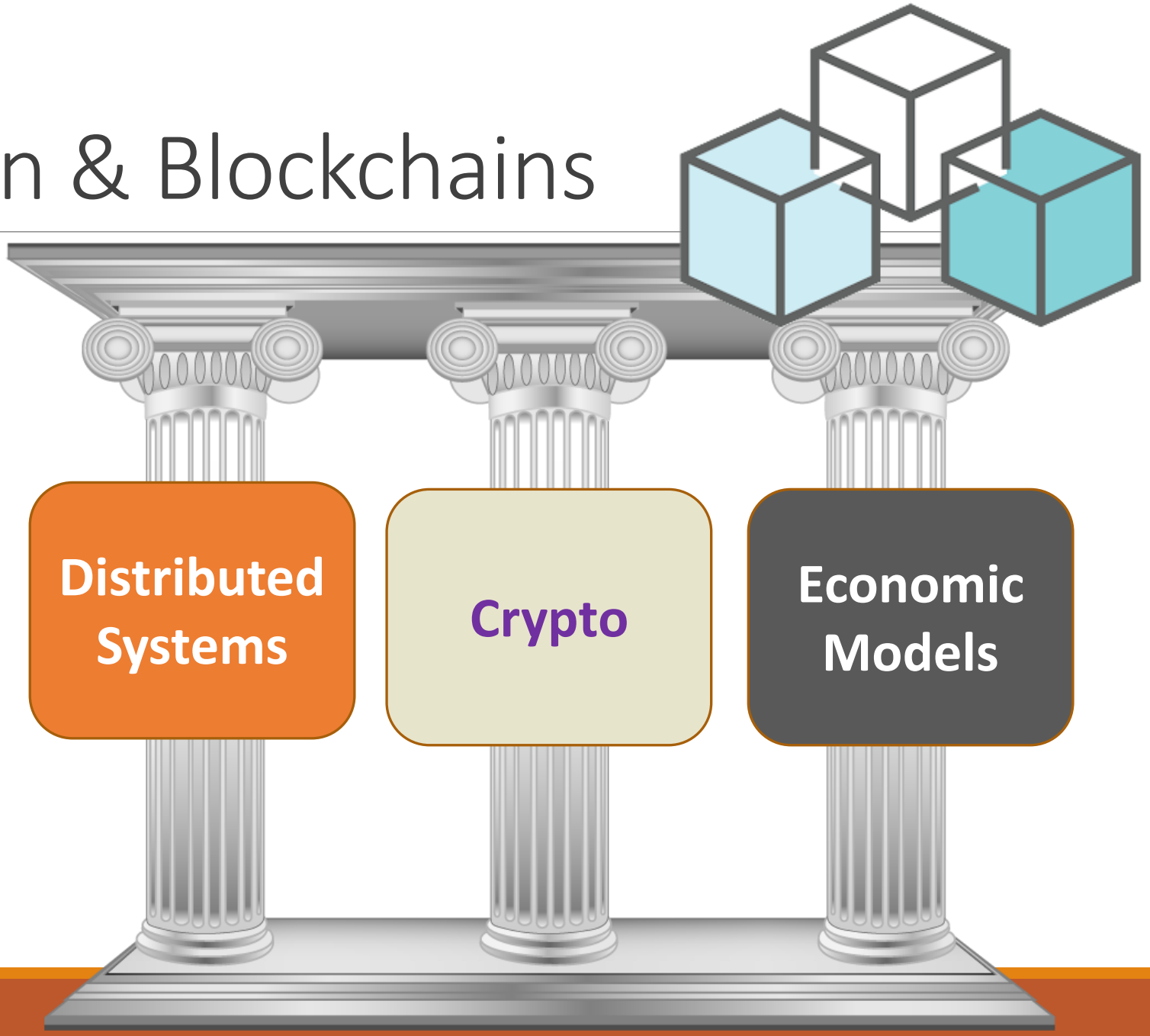


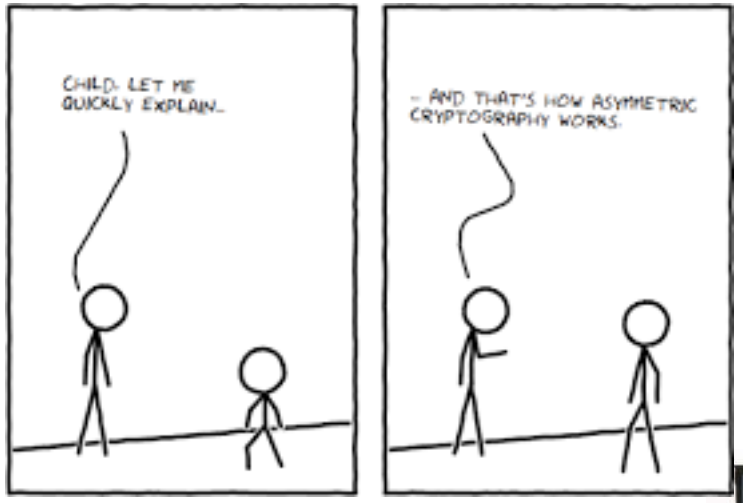
Abstraction



Decentralization & Blockchains

The Three Pillars





Secure Hash Functions

Cryptographic Hash Functions

Takes any arbitrarily sized string as input

- Input M: The message

Fixed size output (usually 256 bits are used in Blockchain)

- Output H(M): We call this the message digest

Efficiently computable

Cryptographic Hash Function: Properties

Deterministic

- Always yield identical hash value for identical input data

Collision-Free

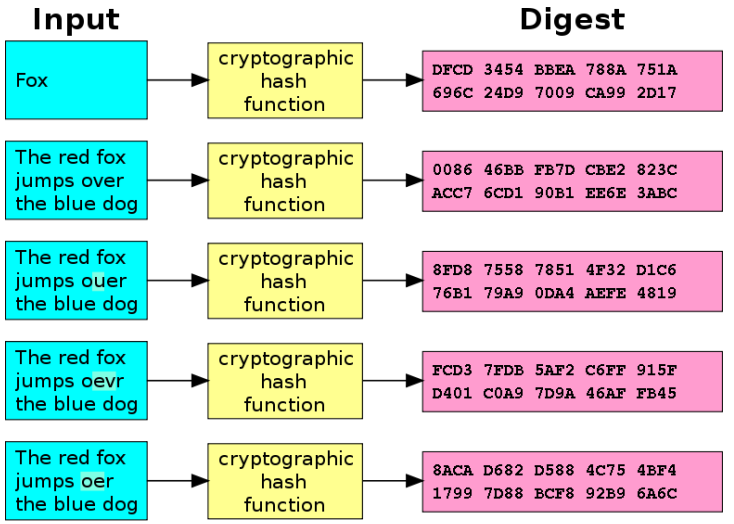
- If two messages are different, then their digests also differ (with high probability ☺)

Hiding

- Hide the original message (the avalanche effect)

Puzzle-friendly

- Given X and Y, find out k such that $Y = H(X||k)$ - used to solve the mining puzzle in Bitcoin *Proof of Work (PoW)*



Collision Free

Hash functions are one-way; Given a x, it is easy to find $H(x)$. However, given an $H(x)$, **no efficient deterministic/probabilistic algorithm** can find x

It is **difficult to find** x and y, where $x \neq y$, but $H(x) = H(y)$

Note the phrase **difficult to find**, collision is **not impossible**

Try with randomly chosen inputs to find out a collision – but it takes too long

Collision Free – How Do We Guarantee?

It may be relatively easy to find collision for some hash functions

Birthday Paradox: Find the probability that in a set of n **randomly chosen persons,** some of them will have the same birthday

- By *Pigeonhole Principle*, the probability reaches 1 when number of people reaches 366 (not a leap year) or 367 (a leap year)
- 0.999 probability is reached with just ~70 people, and 0.5 probability is reached with only ~23 people

If a hash function produces N bits of output, an attacker need to compute only $2^{\frac{N}{2}}$ hash operations on a random input to find two matching outputs with probability > 0.98

For a 256-bit hash function, the attacker needs to compute 2^{128} hash operations – this is significantly time consuming

- If every hash computation takes only 1µsec, it will need $\sim 10^{25}$ years

Hash as A Message Digest

If we observe $H(x) = H(y)$, it is safe to assume $x = y$

We need to remember just the hash value rather than the entire message – we call this the **message digest**

To check if two messages x and y are same, i. e. , whether $x = y$, simply check if $H(x) = H(y)$

- This is efficient because the size of the digest is significantly less than the size of the original messages

Information Hiding through Hash

- Given an $H(x)$, it is “computationally difficult” to find x
- The difficulty depends on the size of the message digests
- Hiding helps to commit a value and then check it later
  - Compute the message digest and store it in a digest store – commit
  - To check whether a message has been committed, match the message digest at the digest store

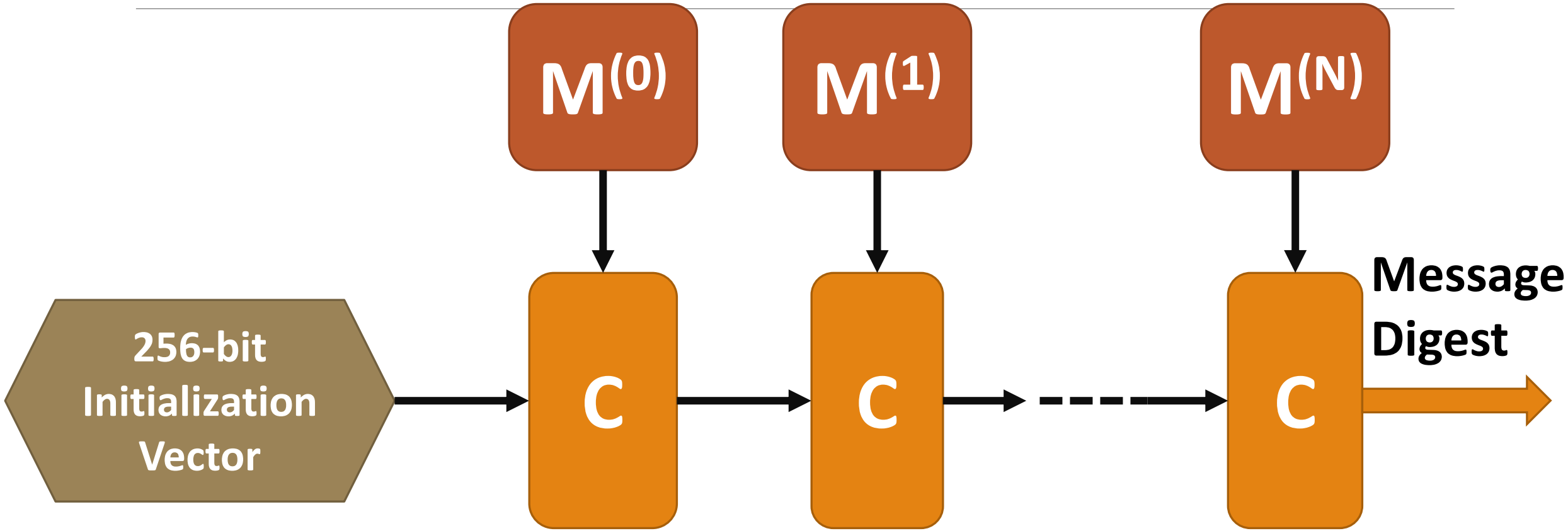
Hash Function – SHA256

SHA256 is used in Bitcoin mining – to construct the Bitcoin blockchain

Secure Hash Algorithm (SHA) that generates 256 bit message digest

A part of SHA-2, a set of cryptographic hash functions designed by United States National Security Agency (NSA)

SHA-256 Algorithm from afar



Puzzle Friendly

Say M is chosen from a widely spread distribution; it is computationally difficult to compute k, such that $Z = H(M||k)$, where M and Z are known a priori.

A Search Puzzle (Used in Bitcoin Mining)

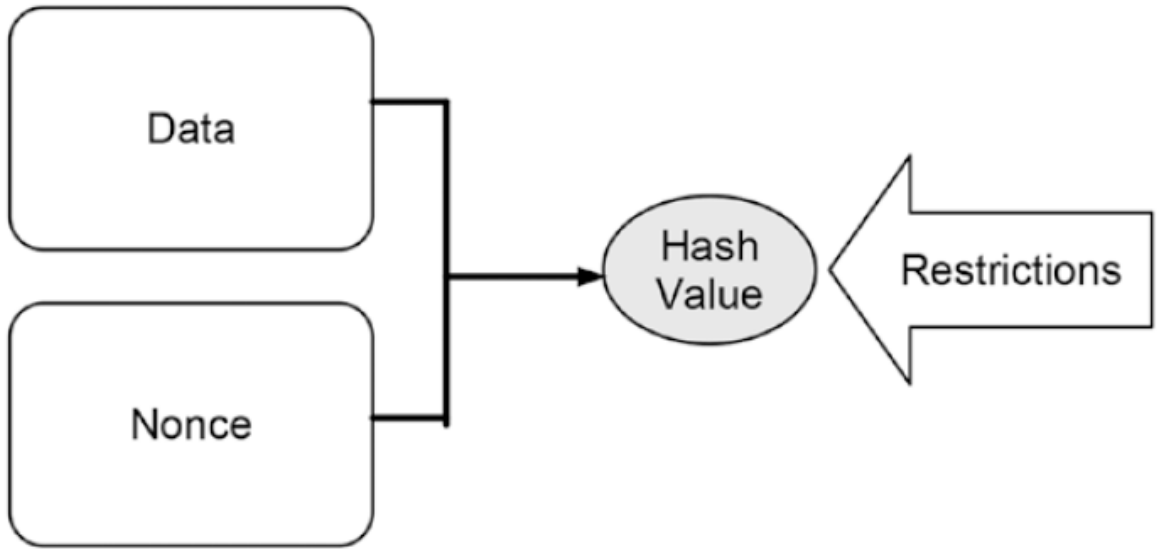
- M and Z are given, k is the search solution
- Note: It might be not exactly a particular value Z, but some properties that Z satisfies, i.e., Z could be a set of possible values

Puzzle friendly property implies that random searching is the best strategy to solve the above puzzle

On Computational Puzzles

Elements of a hash puzzle:

1. Data that must be kept unchanged
2. Data that can be freely changed (nonce)
3. The hash function
4. Restrictions on the hash value of the combined hashing (1) and (2) – the difficulty level



Causing Time Consuming Computations

Hash puzzles can be only solved by trial and error:

1. Guess a nonce
2. Calculate the hash value of data+nonce
3. If the hash value satisfy restrictions (solution) end, else repeat from 1

The solution is easy to check given the nonce.

Nonces for Solving a Hash Puzzle

Nonce	Text to Be Hashed	Output
0	Hello World! 0	4EE4B774
1	Hello World! 1	3345B9A3
2	Hello World! 2	72040842
3	Hello World! 3	02307D5F
	…	
613	Hello World! 613	E861901E
614	Hello World! 614	**00068A3C**
615	Hello World! 615	5EB7483F

Basic Cryptography

Asymmetric vs Symmetric Cryptography

Symmetric: Key for Encryption and Decryption is the same.

Asymmetric: Key for Encryption and Decryption are different.

Symmetric vs Asymmetric

Differences	Symmetric Cryptography	Asymμetric Cryptography
Data Size	Use for Sending Large Data	Use for Sending Small Data
Resources	Low	High
Key Size	128-256 bits	RSA key: ≥2048 bits
Number of Keys	One Key for Encryption/Decryption	Two Keys: One for Encryption and one for Decryption
Security	Less because of One Key	More because of Two Keys
History	Old Technique	Newer Technique
Dangers	The Use of One Key	The Loss of the Private Key
Speed	Fast	Slow

Public Key Cryptography

Also known as **asymmetrical cryptography** or **asymmetric key cryptography**

Key: A parameter that determines the functional output of a cryptography algorithm

- **Encryption:** The key is used to convert a plain-text to a cypher-text; $M' = E(M, k)$
- **Decryption:** The key is used to convert the cypher-text to the original plain text; $M = D(M', k)$

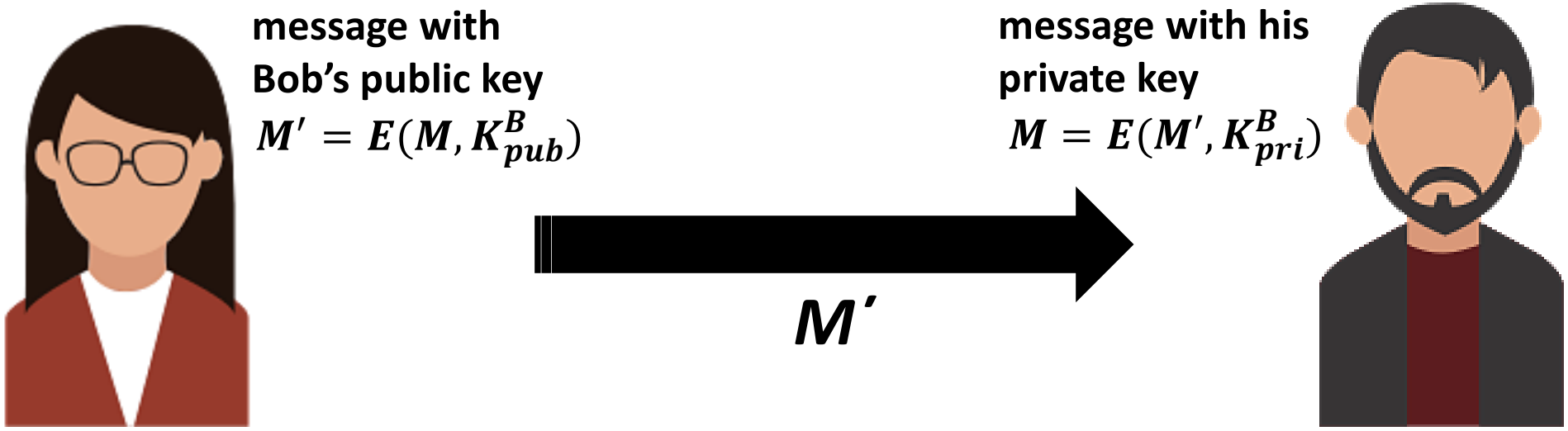
Properties of a cryptographic key (you need to prevent it from being guessed)

- Generate the key truly randomly so that the attacker cannot guess it
- The key should be of sufficient length – increasing the length makes the key difficult to guess
- The key should contain sufficient entropy, all the bits in the key should be equally random

Public Key Cryptography

Two keys are used

- **Private key**: Only Alice has her private key
- **Public key:** "Public" to everyone – everyone knows Alice's public key



Public Key Encryption - RSA

Named over (Ron) Rivest – (Adi) Shamir – (Leonard) Adleman – inventors of the public key cryptosystem

The encryption key is public and decryption key is kept secret (private key)

- Anyone can encrypt the data
- Only the intended receiver can decrypt the data

Digital Signatures

Digital Signature

A **digital code**, which can be included with an electronically transmitted document to

- Verify the identity of the sender
- Authenticate the content of the document
- Prevent *non-repudiation* – sender will not be able to deny about the origin of the document

Purpose of Digital Signature:

- Only the **signing authority** can sign a document, but everyone can verify the signature
- Signature is **associated with** the particular document
  - Signature of one document cannot be transferred to another document

Digital Signature using Public Key Cryptography

Sign the message using the Private key

- Only Alice can know her private key

Verify the signature using the Public key

- Everyone has Alice's public key and they can verify the signature

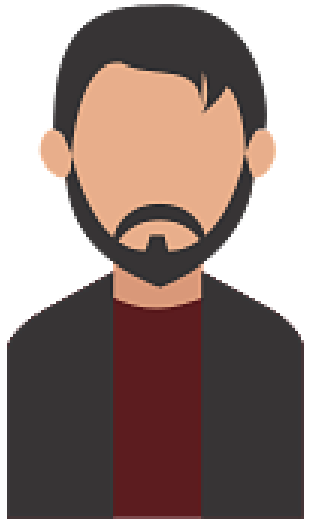
Sign the message with her private key

$$M' = E(M, K^A_{pri})$$

M, M'

Verify the signature using Alice's public key

$$M = E(M', K^A_{pub})$$



Reduce the Signature Size

Use the message digest to sign, instead of the original message

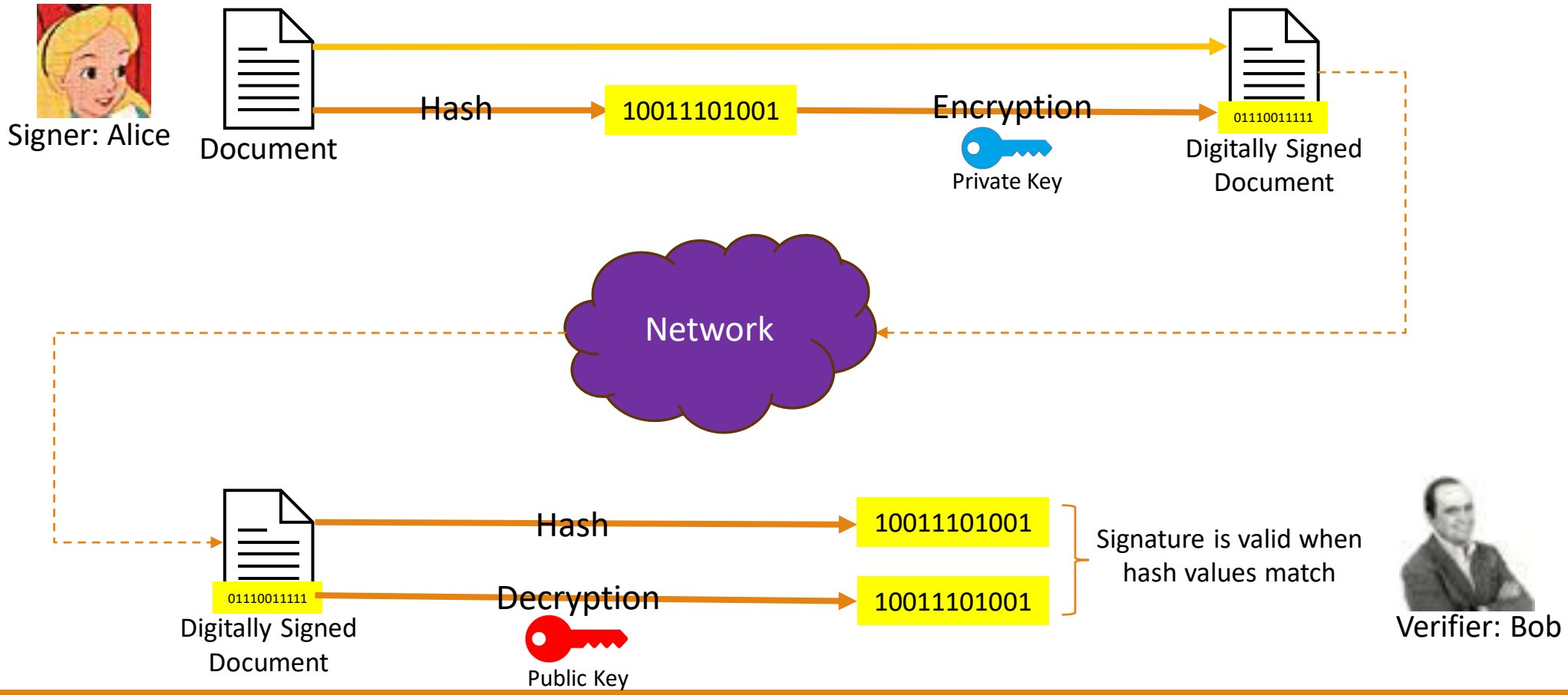
Sign the message with her private key

$$S = E(H(M), K^A_{pri})$$

Verify the signature using Alice's public key

$$H(M) = E(S, K^A_{pub})$$

In a Figure…


Signer: Alice
Document
Hash
10011101001
Encryption
Private Key
01110011111
Digitally Signed Document
Network
01110011111
Digitally Signed Document
Hash
Decryption
Public Key
10011101001
10011101001
Signature is valid when hash values match
Verifier: Bob

Importance of User Keys

- Get a blockchain address

- Make transactions sending or receiving in her address

- Sign transactions to prove that she is the owner of the transferred goods

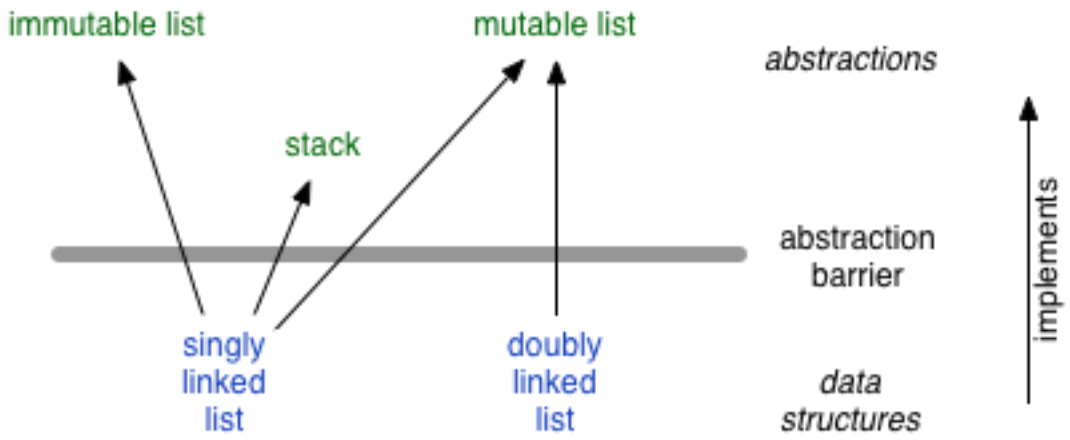
Digital Signature in Blockchain

Used to validate the origin of a transaction

- Prevent non-repudiation
  - **Alice cannot deny her own transactions**
  - **No one else can claim Alice's transaction as his/her own transaction**

Bitcoin uses *Elliptic Curve Digital Signature Algorithm* ***(ECDSA)***

- Based on elliptic curve cryptography
- Supports good randomness in key generation



Immutable Linked Structures

Hash Pointer

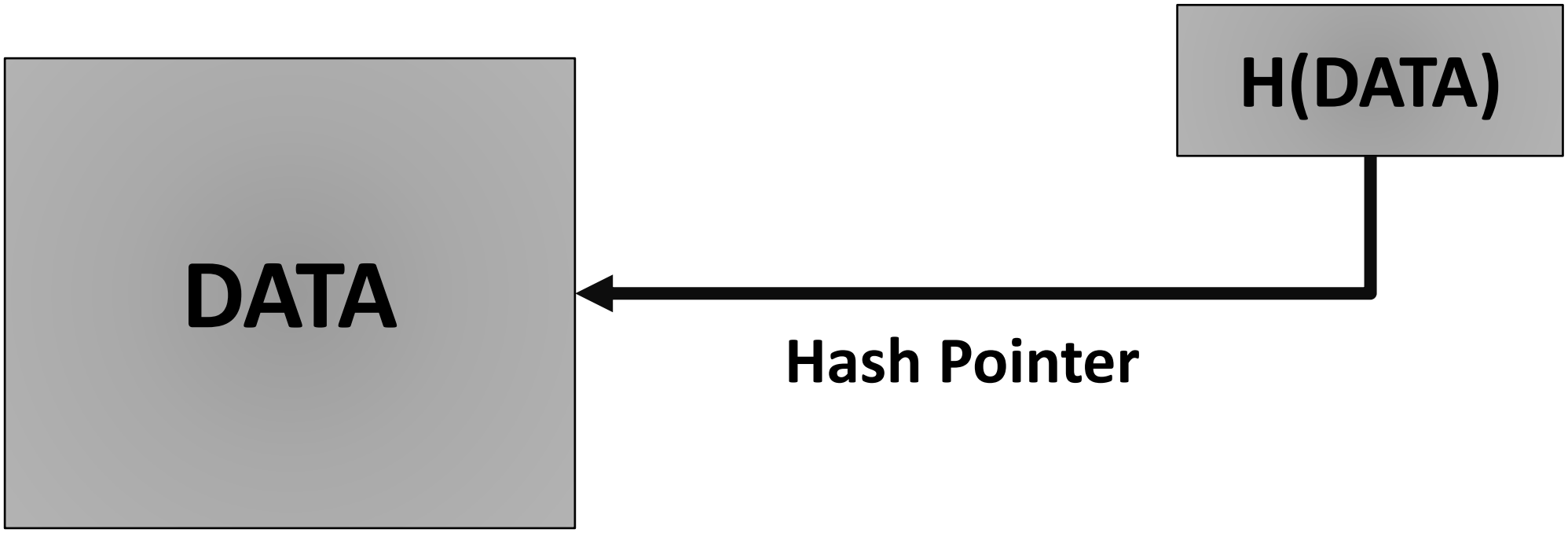
A **Cryptographic Hash Pointer** (Often called Hash Reference) is a pointer to a location:

- The location stores some information
- Hash of this information is stored in the pointer

With the hash pointer, we can

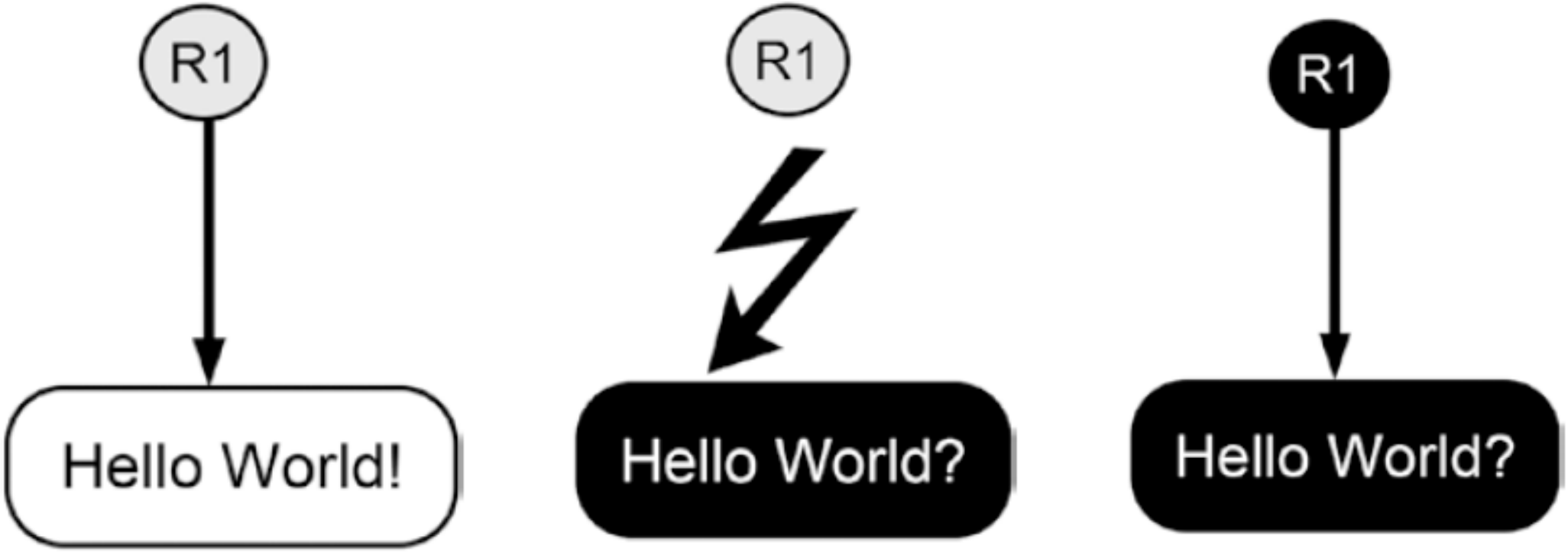
- Retrieve the information
- Check that the information has not been modified (by computing the message digest and then matching the digest with the stored hash value)

Hash Pointer



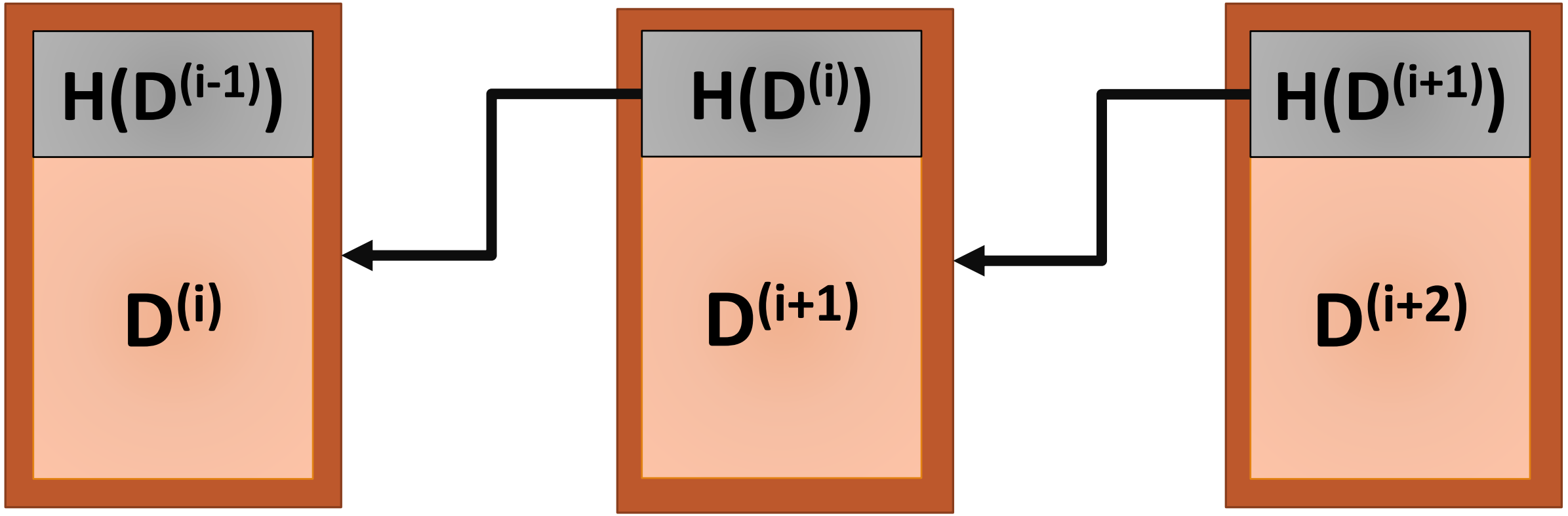
Reminds you of a linked list??

Tamper Detection using Hash Pointer



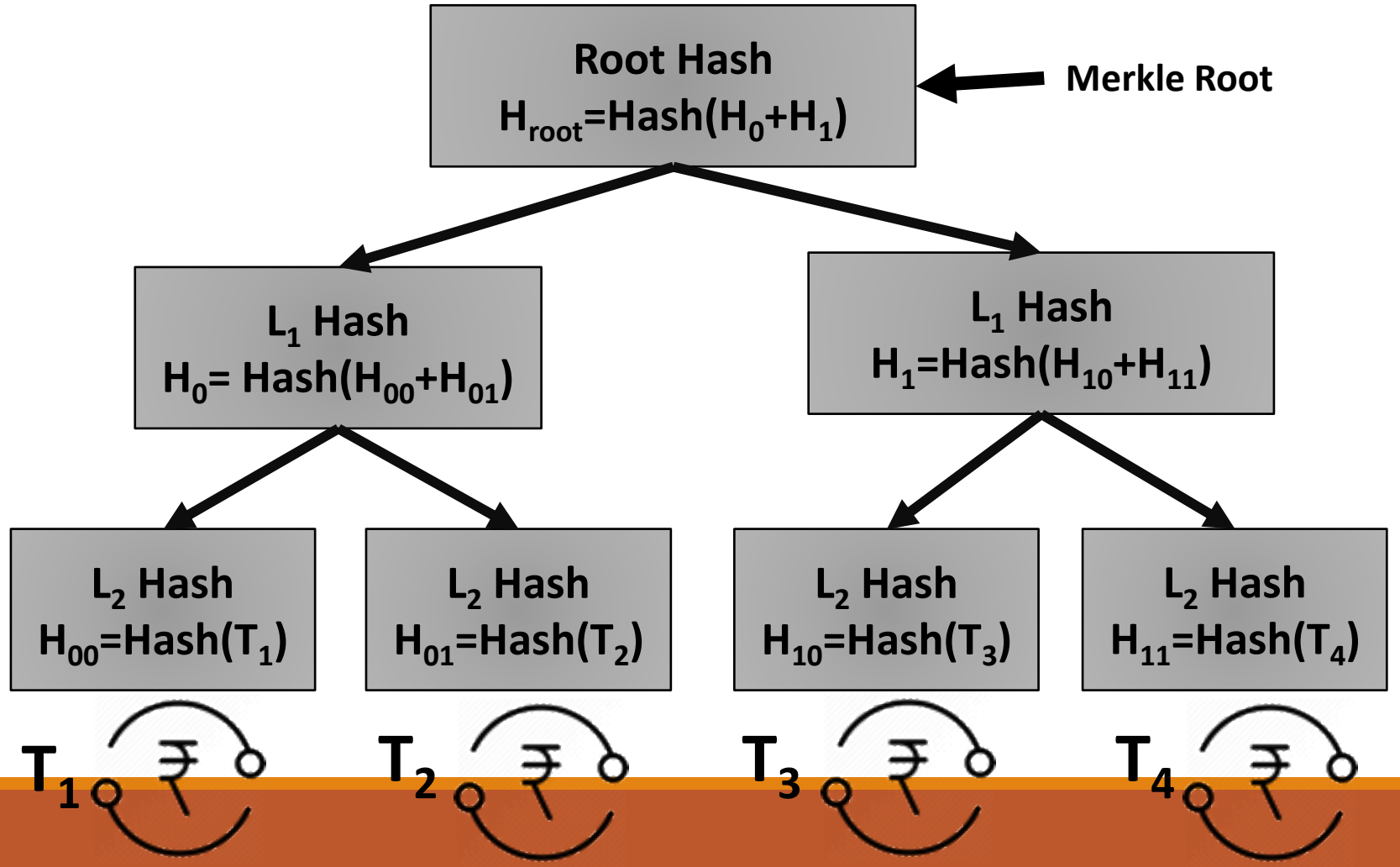
Detect Tampering from Hash Pointers

Hashchain: A Change-Sensitive Linked List



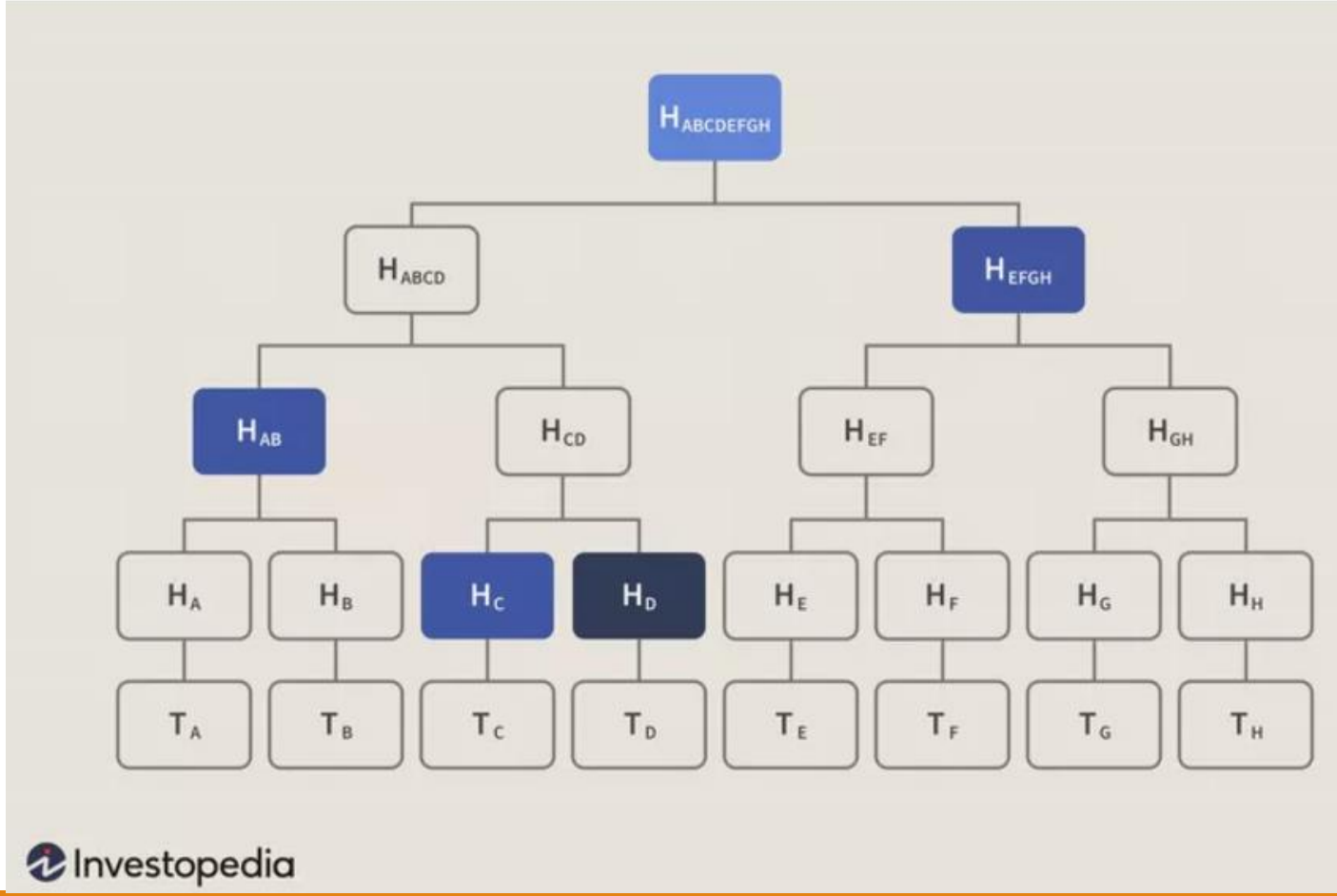
Organization of Hash Pointers in a Tree

Merkle Tree: A Change-Sensitive Tree

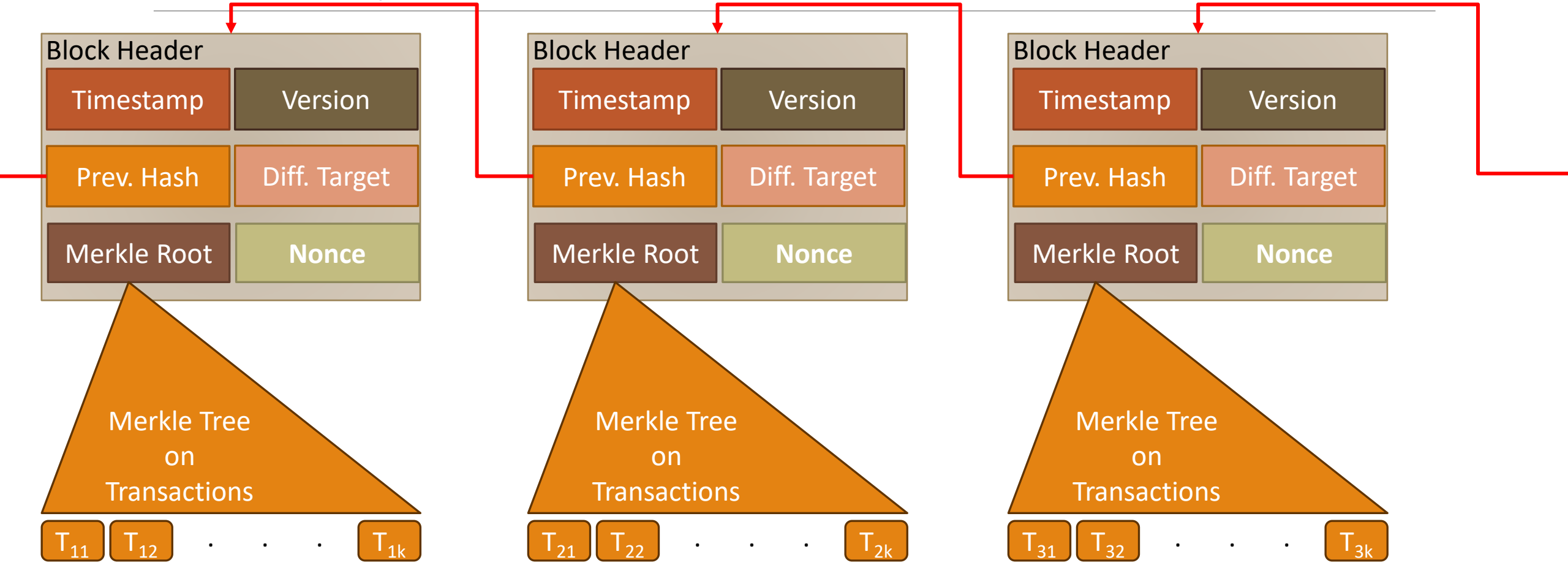


Efficient Verification of a Transaction

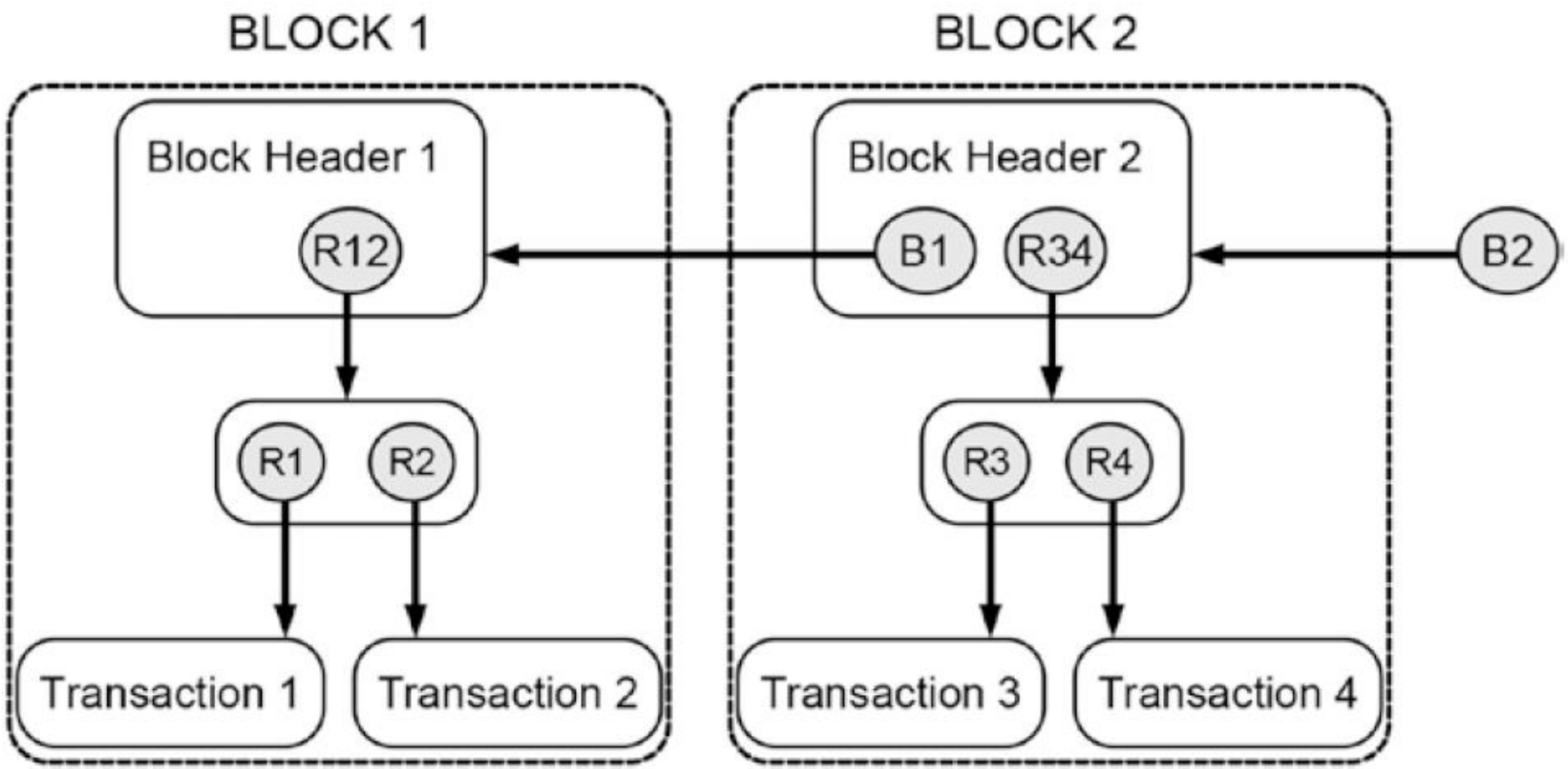
Verify efficiently transaction T_D



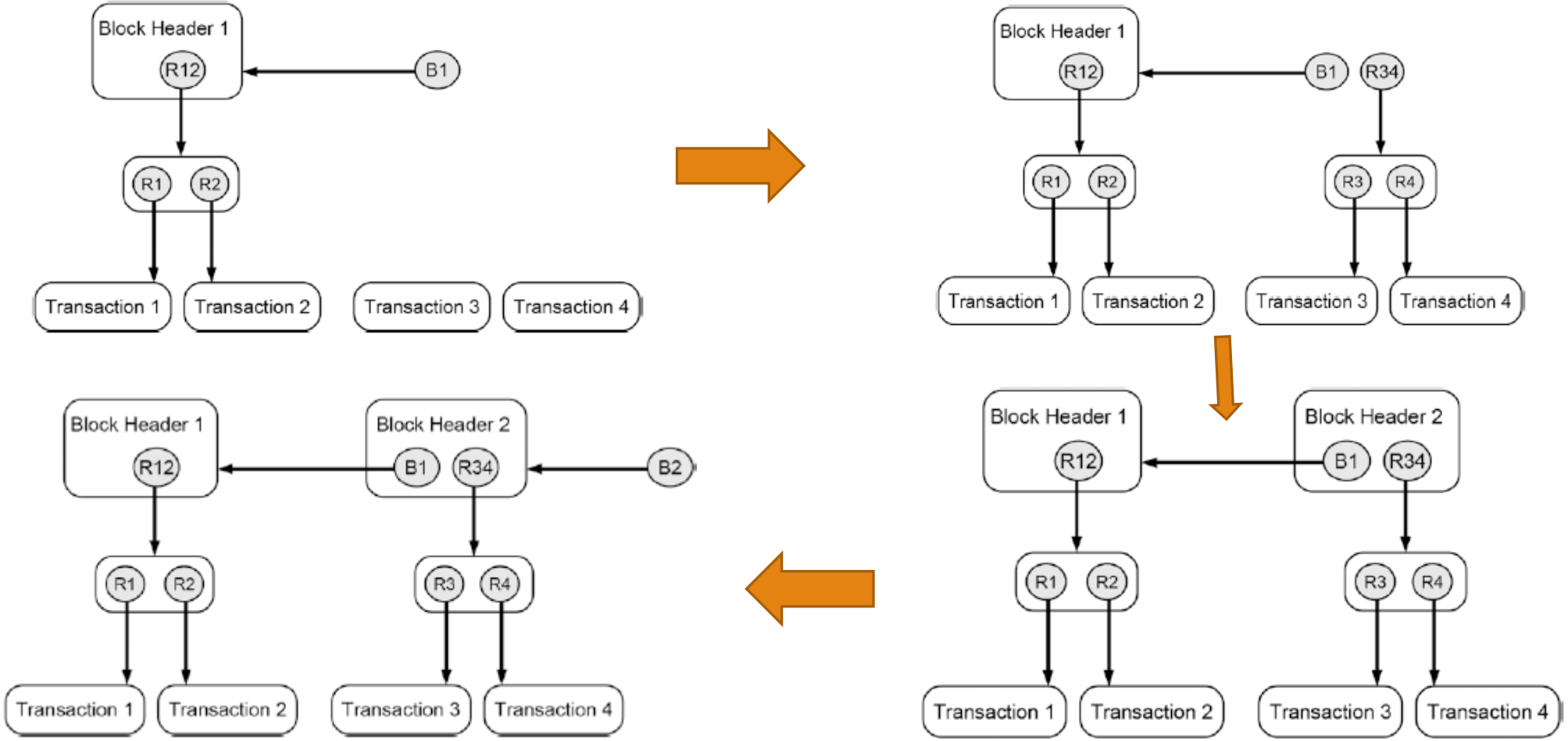
Blockchain is a Hashchain
(a bird's eye view)



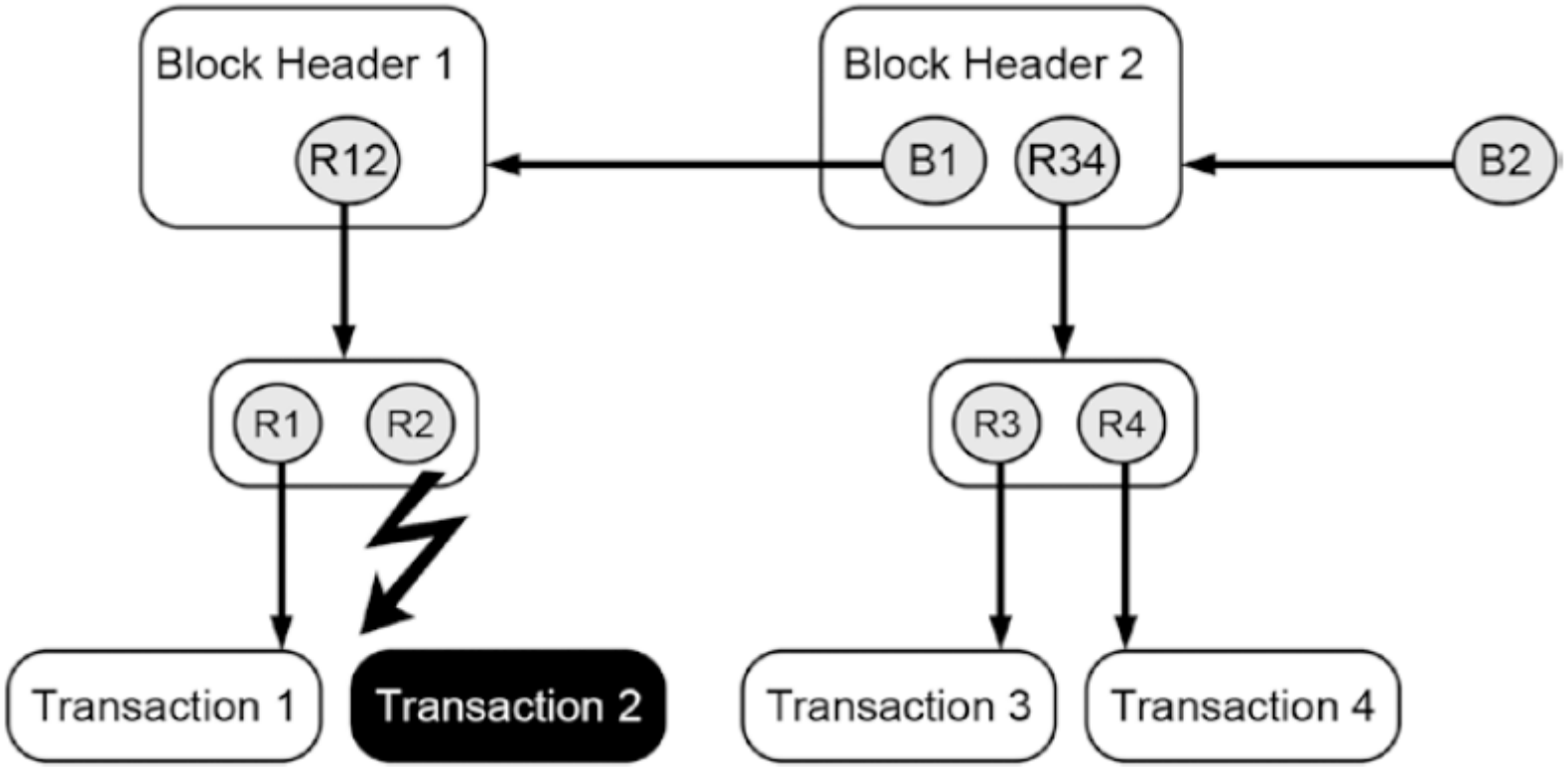
The BlockChain Data Structure



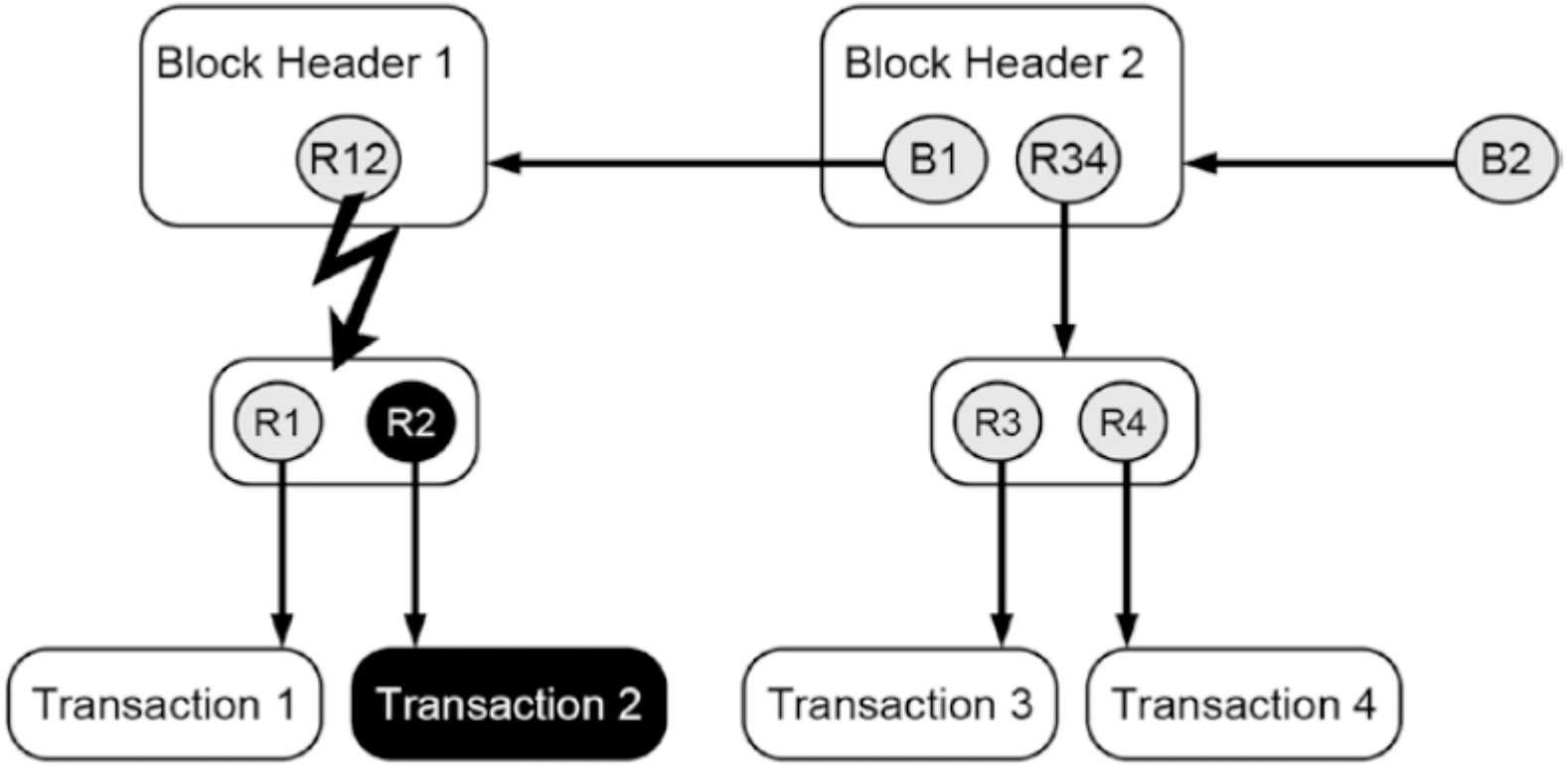
Adding New Transactions



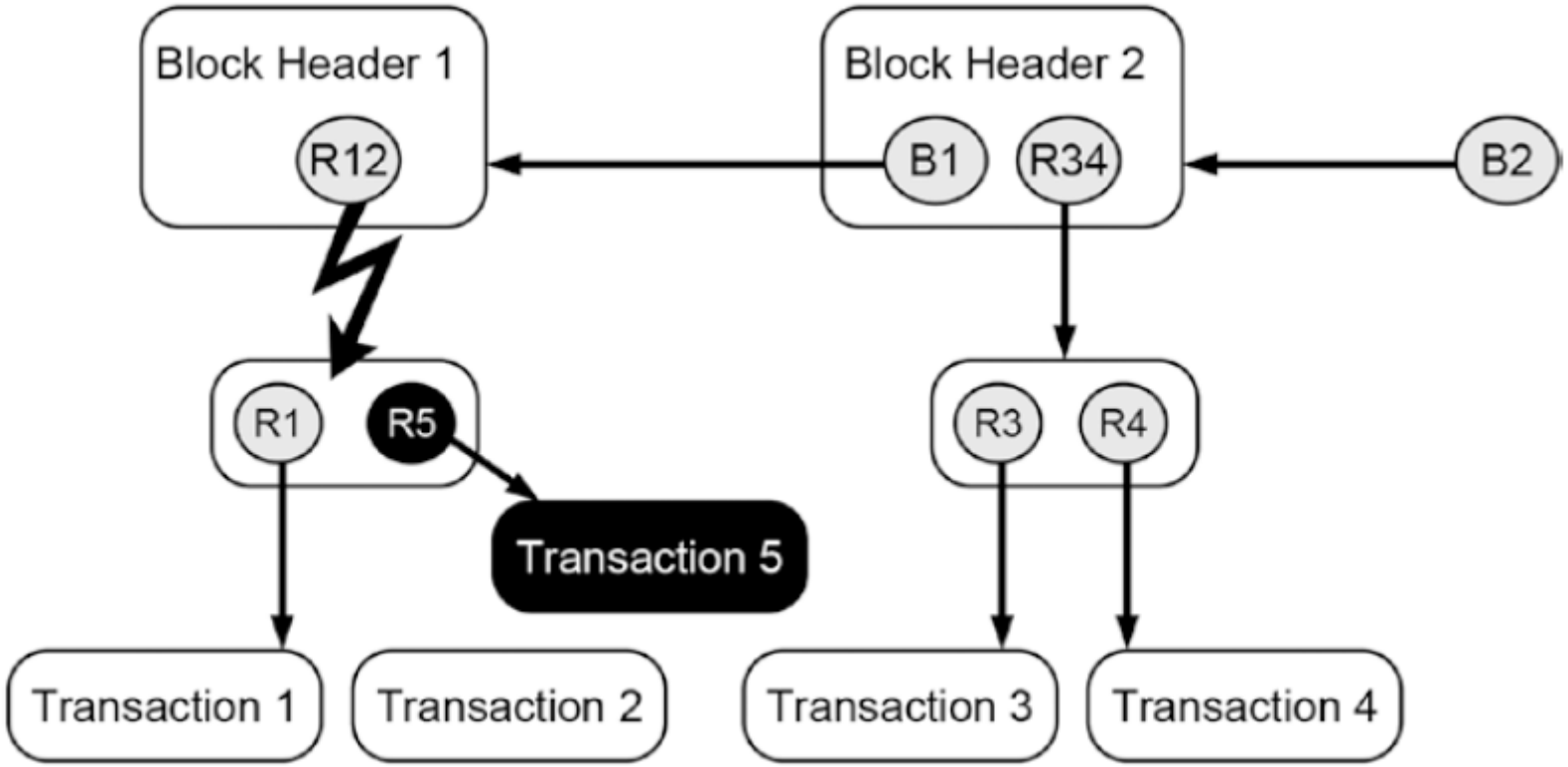
Detecting Changes
Changing the Content of a Transaction



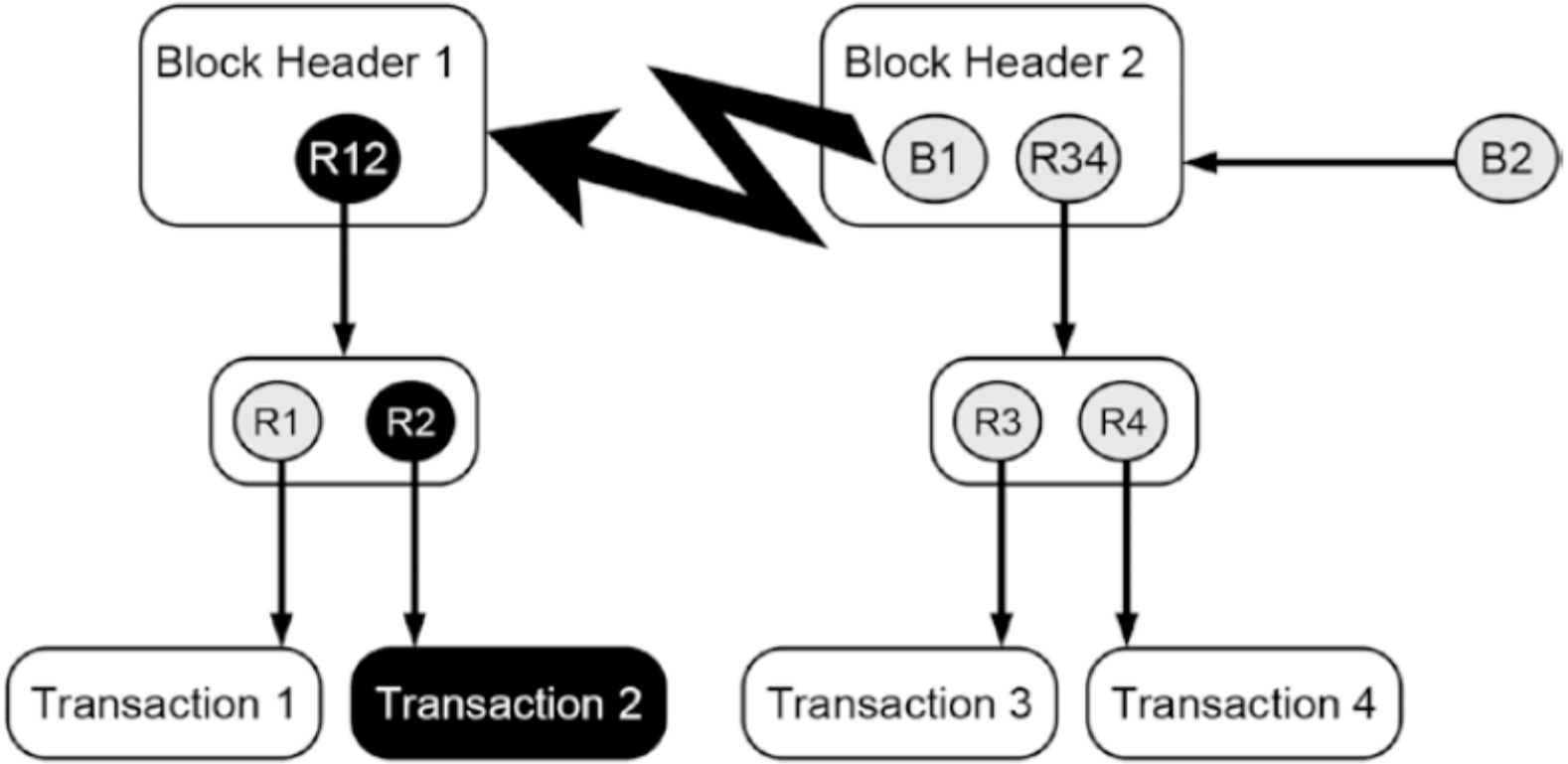
Detecting Changes
Changing a Reference in the Merkle Tree



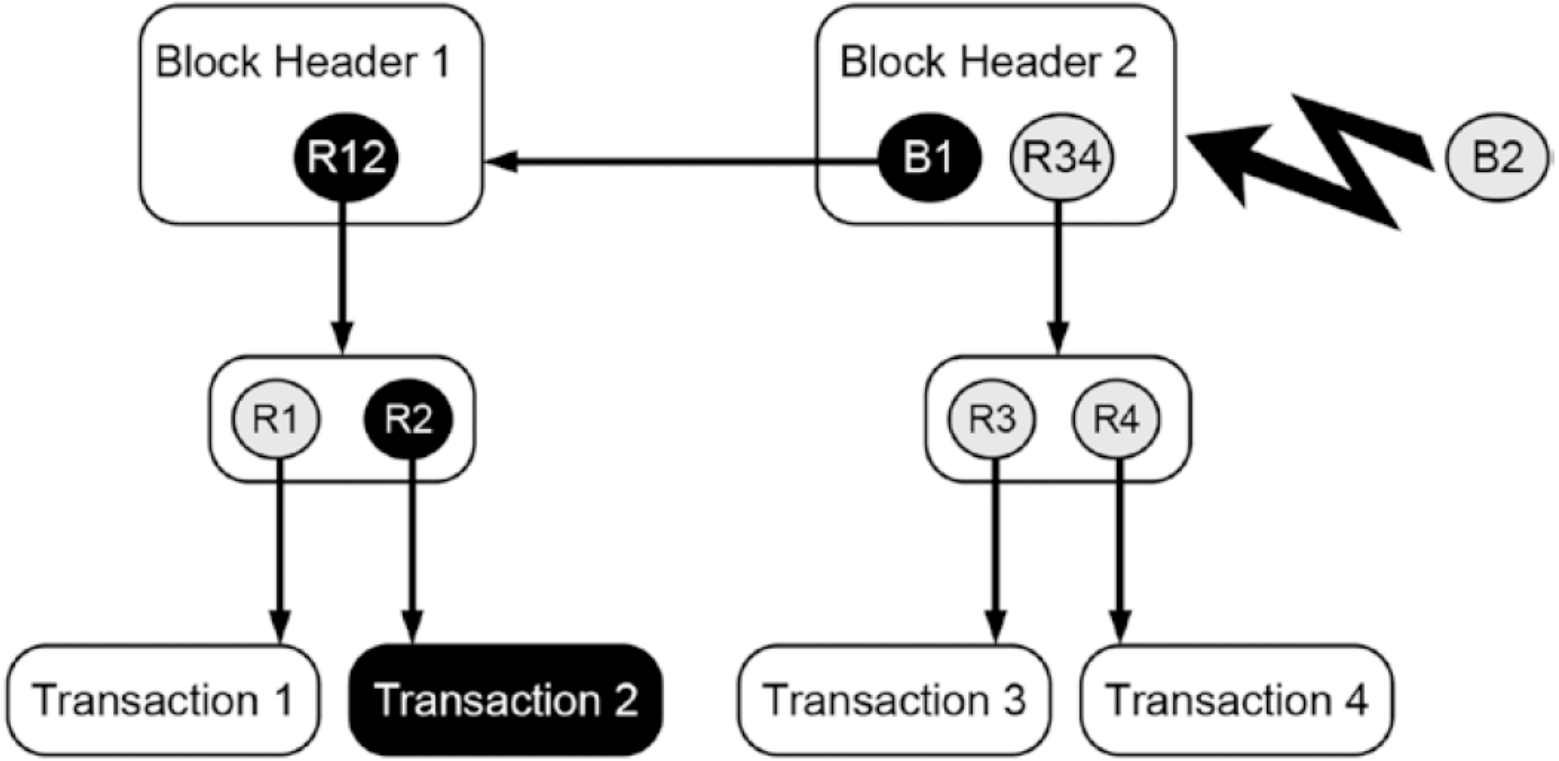
Detecting Changes
Replacing a Transaction



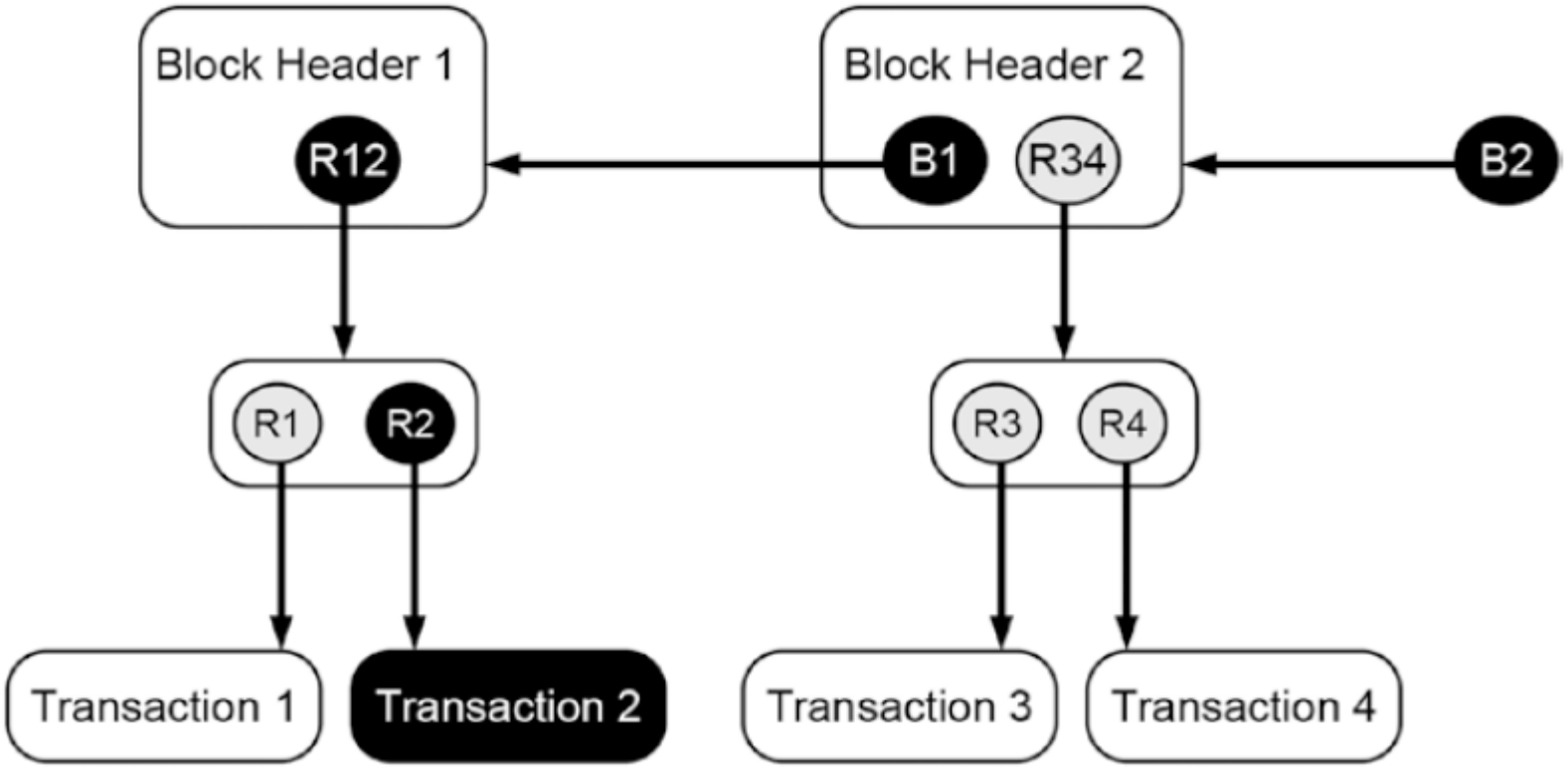
Detecting Changes
Changing the Merkle Root



Changing the Hash Pointer to Previous Block



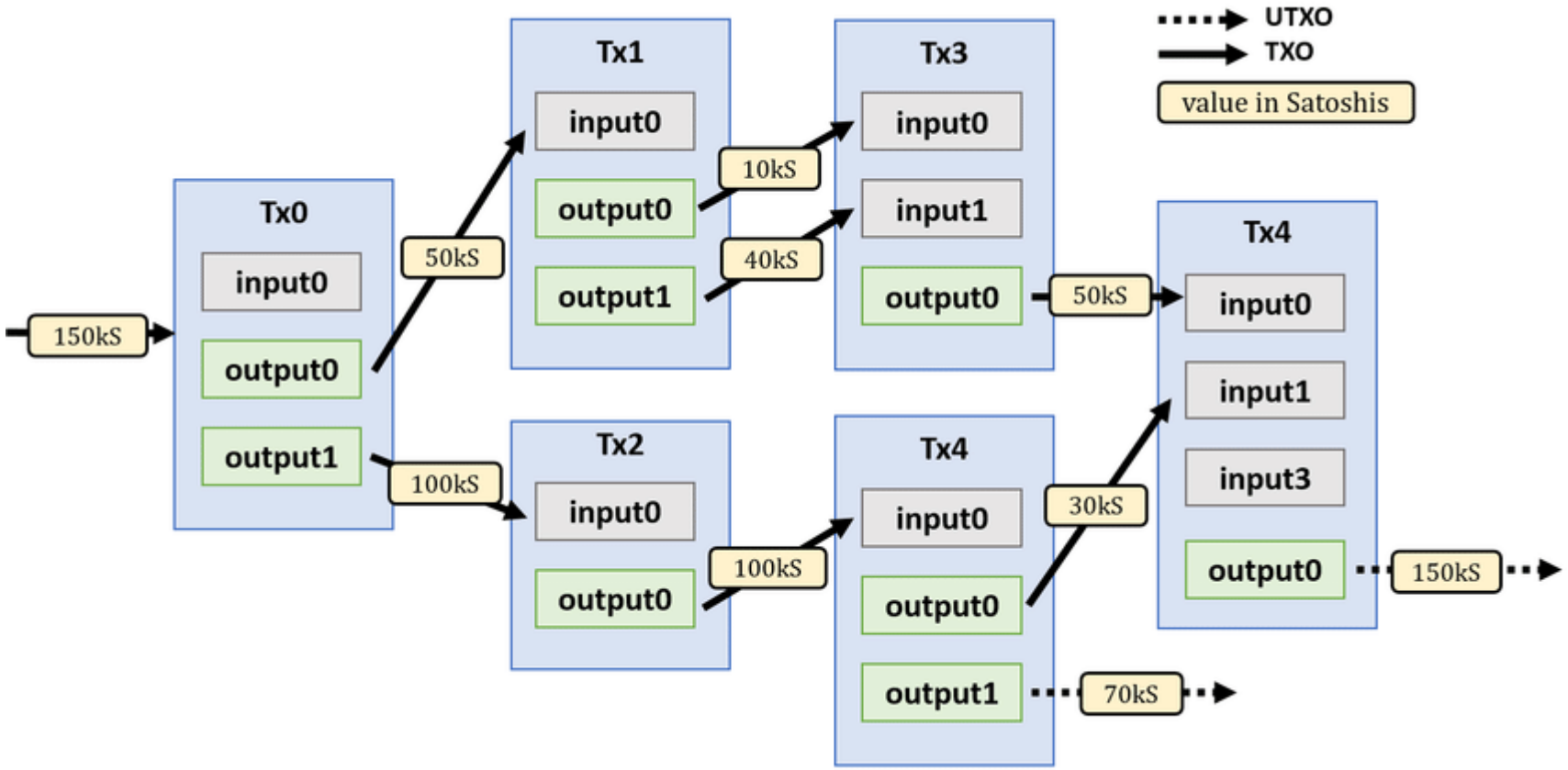
Detecting Changes
Making a Correct Change



TRANSACTIONS IN BLOCKCHAIN

Unspent Transaction Output (UTXO)

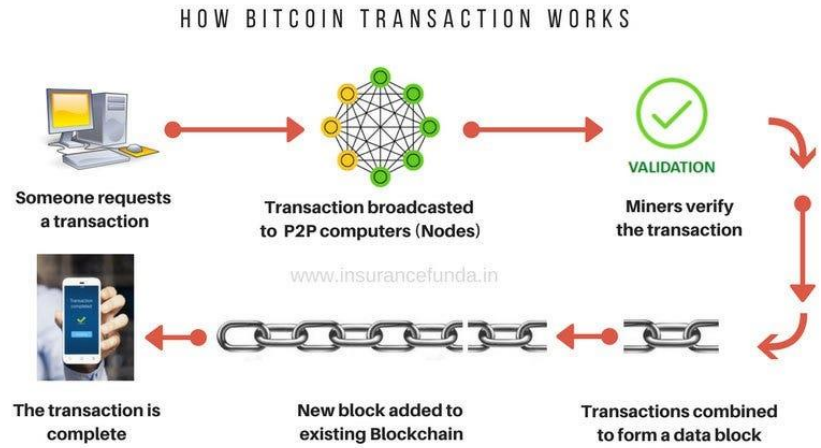
(Example: Bitcoin)



Verification of a Transaction
(Before Broadcast – this is for Bitcoin)

1. Transaction Format and Structure Check
2. Duplicate Transactions Check (check whether you have already acquired this transaction)
3. Digital Signature Verification
4. Inputs Existence (looking up that the UTXOs exist)
5. No Double-Spending (look whether a UTXO has been already spent)
6. Transaction Outputs Validity (the sum of the outputs must not exceed inputs)
7. Sufficient Transaction Fees (fees=Outputs-Inputs)
8. Script Executions, Locktime and Sequence Numbers

Bitcoin Network



Each P2P node runs the following algorithm:

- New transactions are broadcast to all nodes.
- Each node (miners) collects new transactions into a block.
- Each node works on finding a proof-of-work for its block. (Hard to do. Probabilistic. The one to finish early will probably win.)
- When a node finds a proof-of-work, it broadcasts the block to all nodes.
- Nodes accept the block only if all transactions in it are valid (e.g., digital signature checking) and not already spent (check all the transactions).
- Nodes express their acceptance by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

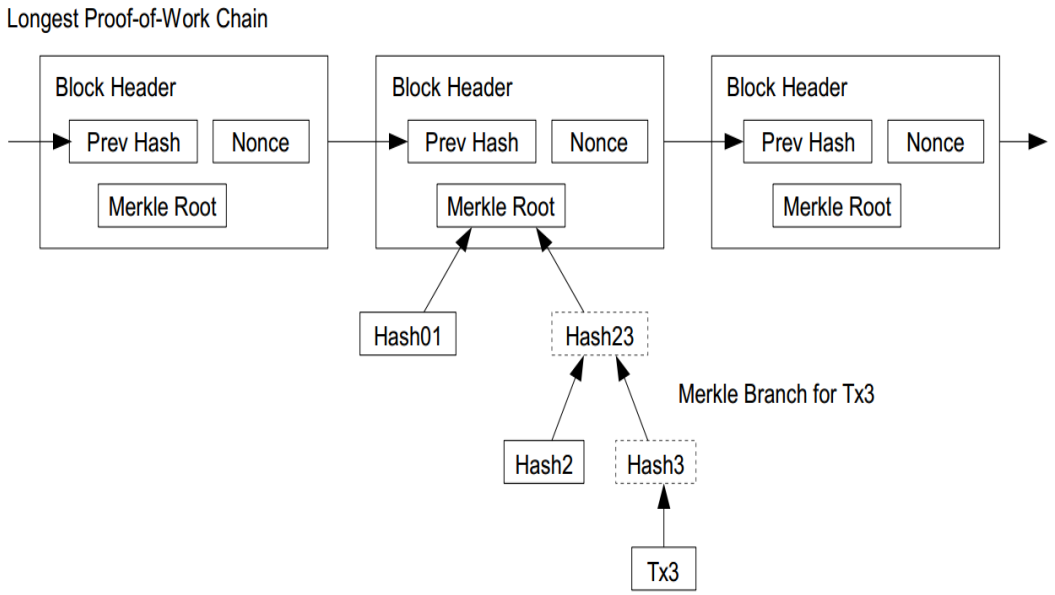
Simplified Transaction Verification

Any user can verify a transaction easily by asking a node.

First, get the longest proof-of-work chain

Query the block that the transaction to be verified (tx3) is in.

Only need Hash01 and Hash2 to verify; not the entire Tx's.



Account-Based
(Example: Ethereum)

State *n*:

Alice's Account	10ETH
Contract's Account	1ETH
Vader's Account	10000 ETH

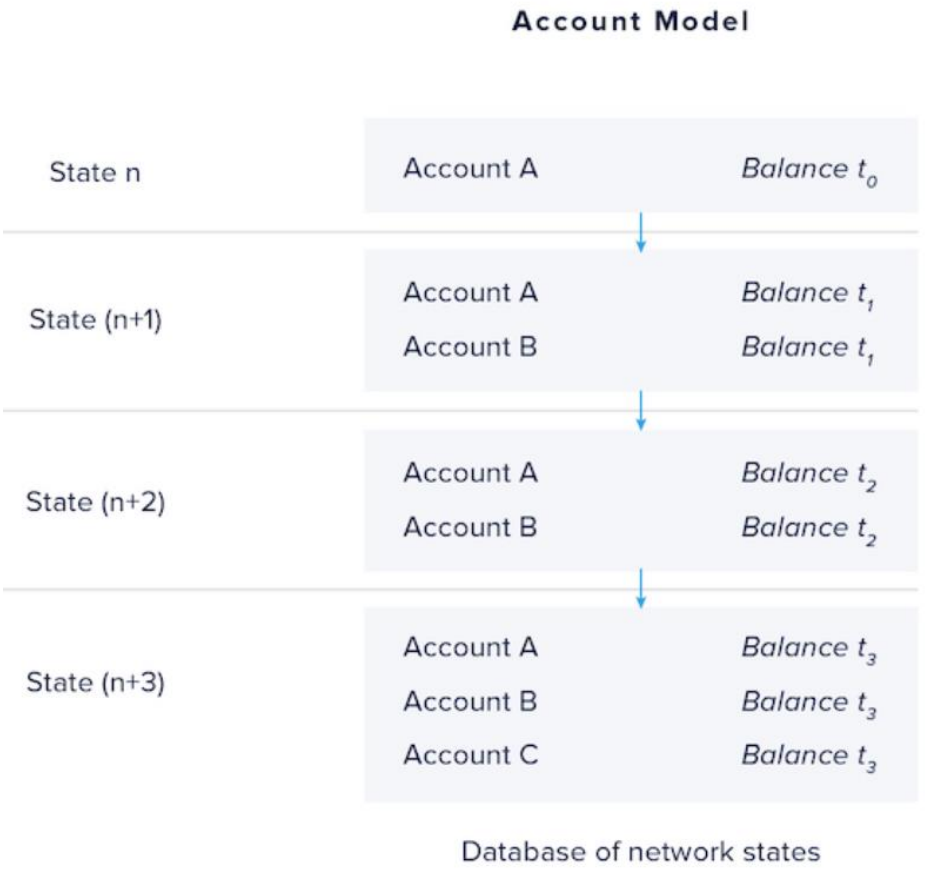
From: Alice's Address

Value: 1 ETH

To: Vader's Address

State *n+1*:

Alice's Account	9 ETH
Contract's Account	1 ETH
Vader's Account	10001 ETH



Verification of a Transaction
(Before Broadcast – for Ethereum)

The validator:

1. Checks transaction format and structure
2. Verifies signature
3. Checks for sufficient balance
4. Checks gas price and gas limit
5. Executes the transaction (even if it is a smart contract)
6. Applies state transition if successful execution
7. Packages the transaction(s) in a block.

Ethereum 2.0

The validator:

1. Broadcast the block to other validators
2. The block is checked by other validators and signal about its correctness (attestation – vote)
3. Aggregation of votes (by aggregators – special validators)
4. The aggregated votes are included in the blockchain
5. Finalization of a block (through consensus)
6. Validators get rewards or penalties for their work
7. Epoch processing

On Simple CryptoCurrency

GOOFY AND SCROOGE

The GoofyCoin

Creation of Coins

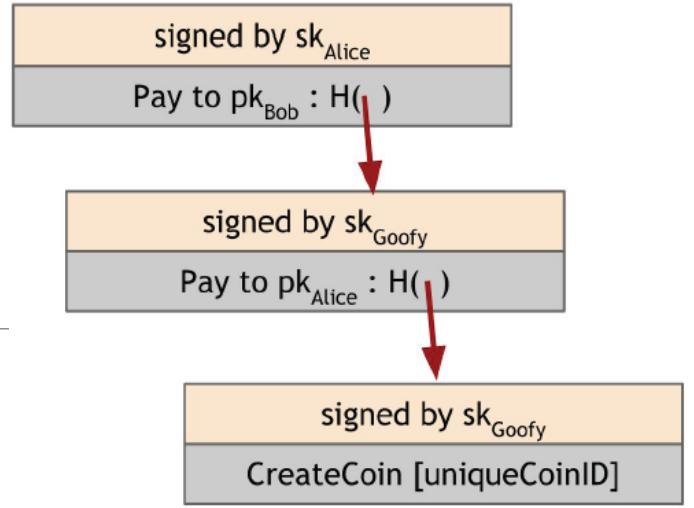
Goofy can create coins whenever he wants.

- These coins belong to him

How?

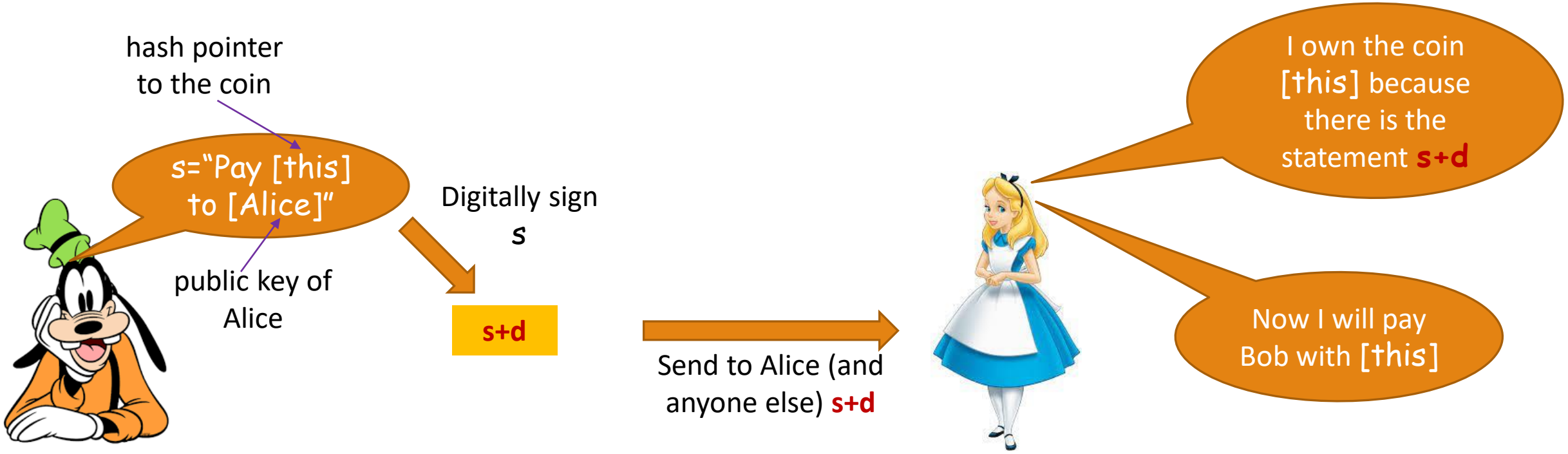
1. Creates a unique coin ID [uniqueCoidID] constructing the string: **s**="CreateCoin [uniqueCoinID]"
2. Computation of digital signature **d** of string **s**
3. **d+s** is a valid digital coin, and anyone can validate it through the public key of Goofy

Transfer of Coins



Whoever owns a coin can transfer it to someone else.

- To transfer a coin, cryptographic operations must be used



The Goofy Coin has a Critical Security Problem: Double Spending Attack

- The witch sends the coin to Alice but does not tell it anyone else.
- She sends the same coin at almost the same time to the dwarf as well.
- She has used the same coin twice. Who owns the coin?

Send to Dwarf s_D+d_D

Send to Alice s_A+d_A

The ScroogeCoin

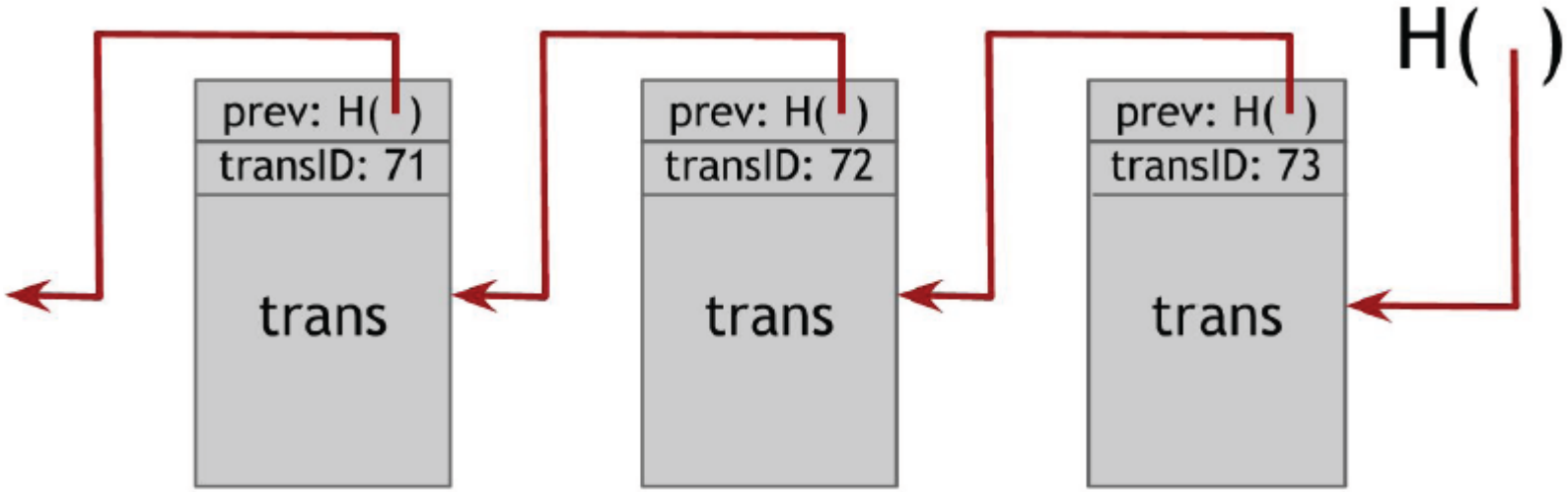
SOLVING THE DOUBLE SPENDING ATTACK IN THE GOOFYCOIN

MORE COMPLICATED DATA STRUCTURES ☺

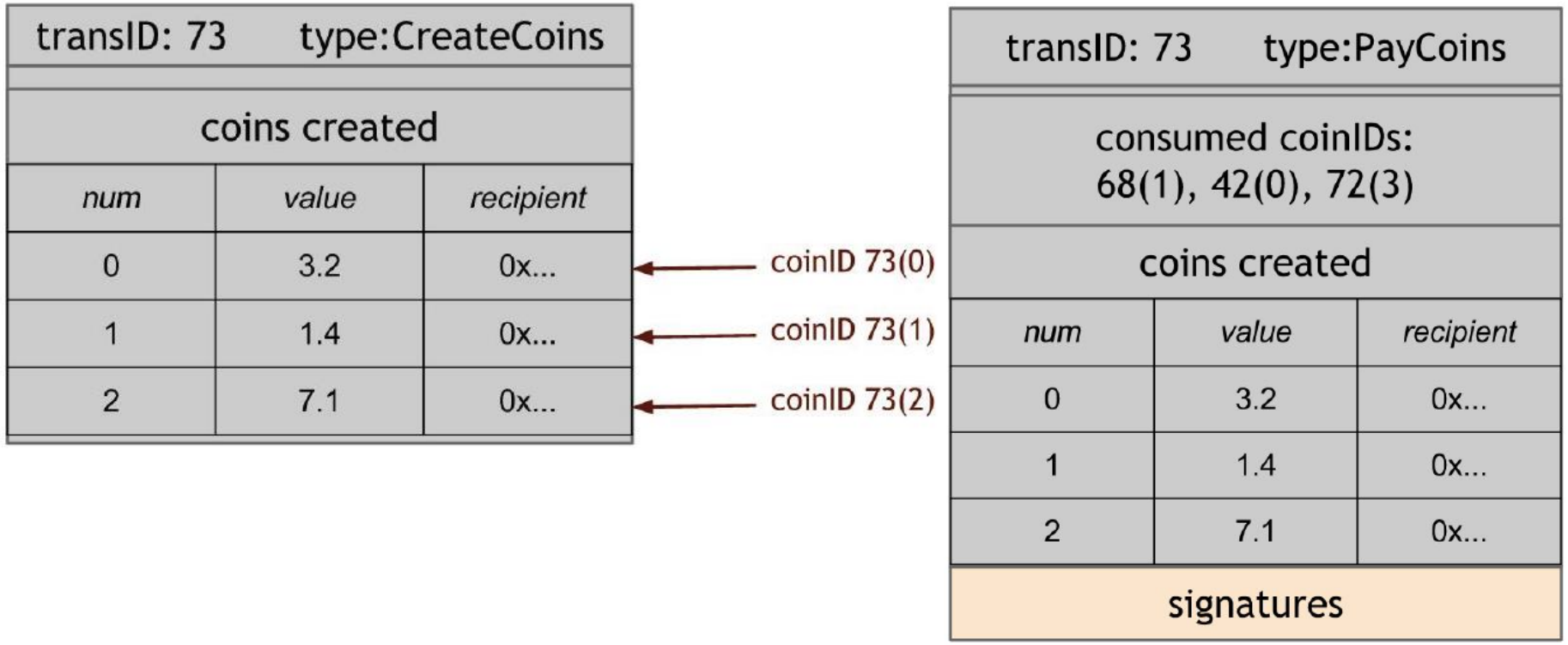
Using the BlockChain

- Scrooge create coins like Goofy **BUT**
- he publishes an append-only ledger with the history of transactions that have happened.
  - Append-only: the transaction cannot change/be deleted.
  - All transactions are written to the ledger before accepted.

Use a BlockChain:



Two Types of Transactions



PayCoins is valid if:

- The consumed coins are valid
- not already consumed
- total value out = total value in
- Signed by all owners of spent coins

Coins are Immutable

Coins cannot be transferred, subdivided or combined

Solution: Use Transactions!!!

To Subdivide:

1. Create a new transaction
    1. Consume your coin
    2. Pay out two new coins to yourself (of same total value)

The Double-Spending Attack

- A transaction is valid if in a block in the blockchain signed by Scrooge
- Scrooge makes sure that no double-spending transactions are registered
- All can check the validity of the blocks digitally signed by Scrooge

ScroogeCoin Problems

Can Scrooge change a transaction in the history (already registered)?

- No, the others will understand it because the hash pointers will be invalid
  - Of course, one can say: "*Who cares? He is Scrooge. He is doing whatever he likes*"

- Highly Available ✗
- Censorship Proof ✗
- Reliable ✓
- Open ✗
- Pseudoanonymous ✓
- Secure ✓
- Resilient ?
- Eventually Consistent ✓
- Keeping Integrity ✓

Question:

Can we descroogify the currency?
Can we operate without a central trusted party?

Smart Contracts

Smart Contracts
(in the case of Ethereum)

- Executable code
- Turing Complete
- Function like an external account
  - Hold funds
  - Can interact with other accounts and smart contracts
  - Contain code
- Can be called through transactions

Code Execution
(in the case of Ethereum)

- Every node contains a virtual machine (similar to Java)
  - Called the Ethereum Virtual Machine (EVM)
  - **Compiles** code from high-level language to bytecode
  - Executes smart contract code and broadcasts state

- ***Every full-node on the blockchain processes every transaction and stores the entire state***

Gas
(in the case of Ethereum)

- Halting problem (infinite loop) – reason for Gas
  - Problem: Cannot tell whether or not a program will run infinitely from compiled code
  - Solution: charge fee per computational step to limit infinite loops and stop flawed code from executing
- Every transaction needs to specify an estimate of the amount of gas it will spend
- Essentially a measure of how much one is willing to spend on a transaction, even if buggy

Summary of Advantages and Disadvantages of Smart Contracts

Advantages	Disadvantages
Agent neutrality in signing deals	Difficult to make changes
Automation in signing deals, time saving: excludes human participation in transactions, everything is done by the prescribed program code	The third party agents do not disappear but starts playing a different role. The need for lawyers experienced in IT increases in the future because the programmers of smart contracts will need consultations for making new kinds of contracts
Safety: data in the decentralized registry cannot be lost and cyber attacked	The consumers are quite suspicious because it is a new technology and they do not understand it yet
Precision: no mistakes can be made due to the absence of hand-filled forms	One can keep and save data in smart contracts safely and it is void of any distortions, only if the code is written perfectly and precisely

Βιβλιογραφία

Αλυσίδες Συστοιχιών (BlockChain). Κάλλιπος.

Blockchain Basics: A Non-Technical Introduction in 25 Steps