A3a: Bitcoin & blockchain

* Many slides from Joseph Bonneau@NYU

outline

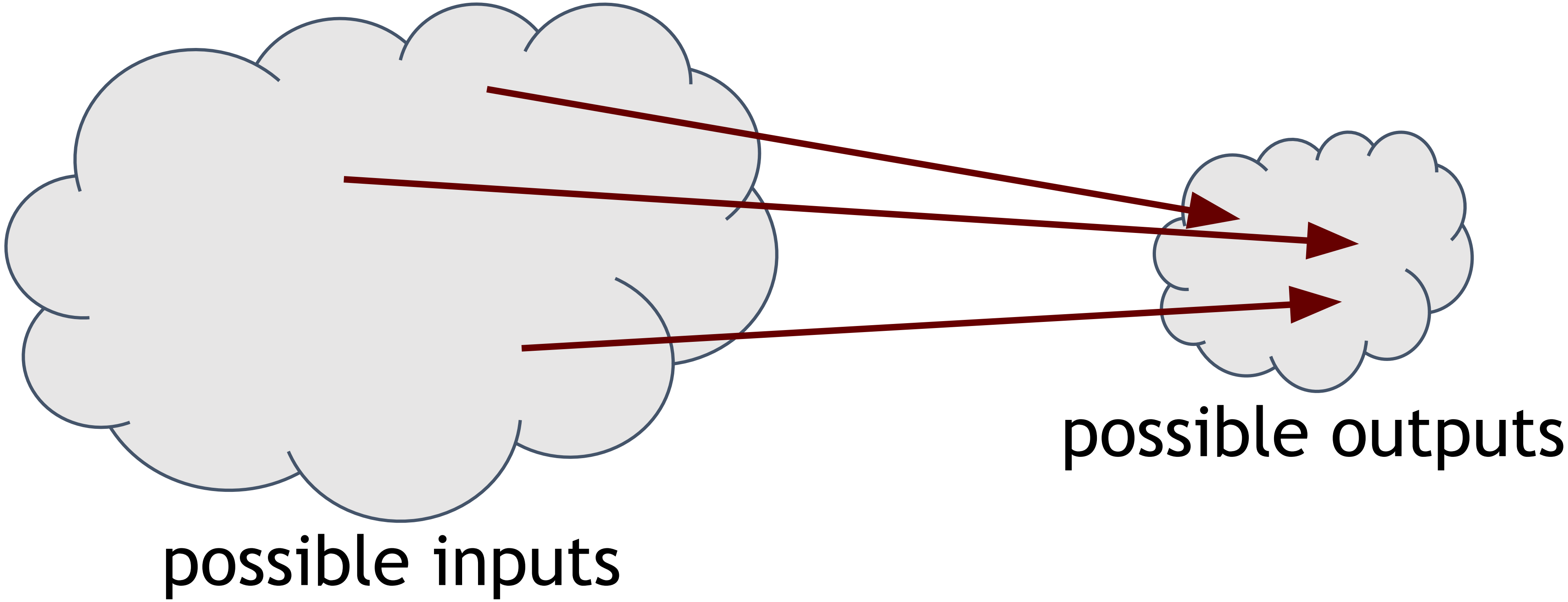
1. Cryptographic hash function
2. Hash pointer, Blockchain, Merkle tree
3. Digital signatures
4. Simple cryptocurrencies
5. Transaction semantics
6. A decentralized ledger: bitcoin
7. Bitcoin P2P networking
8. Finding a valid block
9. The rules of Bitcoin

1. Cryptographic Hash Functions: Review

Cryptographic Hash fn.

- Hash function:
    - Deterministic function H: $\{0,1\}^* \rightarrow \{0,1\}^k$
    - any string as input
    - fixed-size output (e.g. SHA-256: k=256 bits)
    - efficiently computable
- Security properties:
    - collision-resistant
    - one-way
    - puzzle-friendly

Hash property #1: collisions exit...



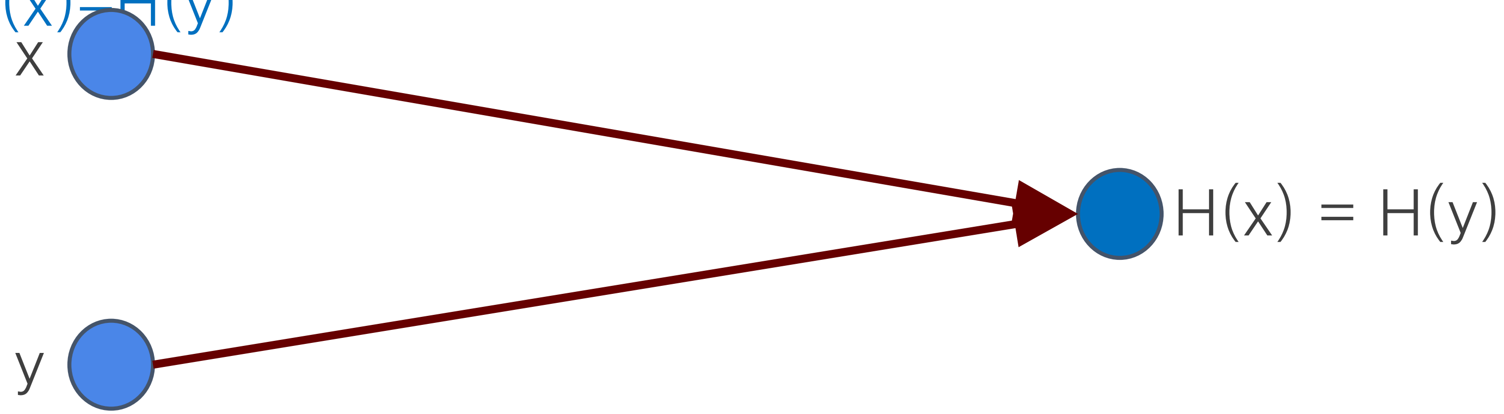
… but can anyone find them?

Birthday attack on any 256-bit hash **H**:

1. try 2^{130} randomly chosen inputs
2. >99.8% chance that two of them will collide

This works no matter what **H** is…, but it takes too long to matter

In reality, nobody can find x and y such that x != y and H(x)=H(y)



collision free?

collision resistant!

source: https://docs.google.com/presentation/d/13SOZqHzLRryyFXw26CCE89lKMa8X9YAhdTFgXdw0a1s/view#slide=id.p

There are faster ways to find collisions for some H

- MD5: 128 bits (collisions found in 2007)
- SHA-1: 160 bits (collisions found in 2017)

Others are currently *collision-resistant:*

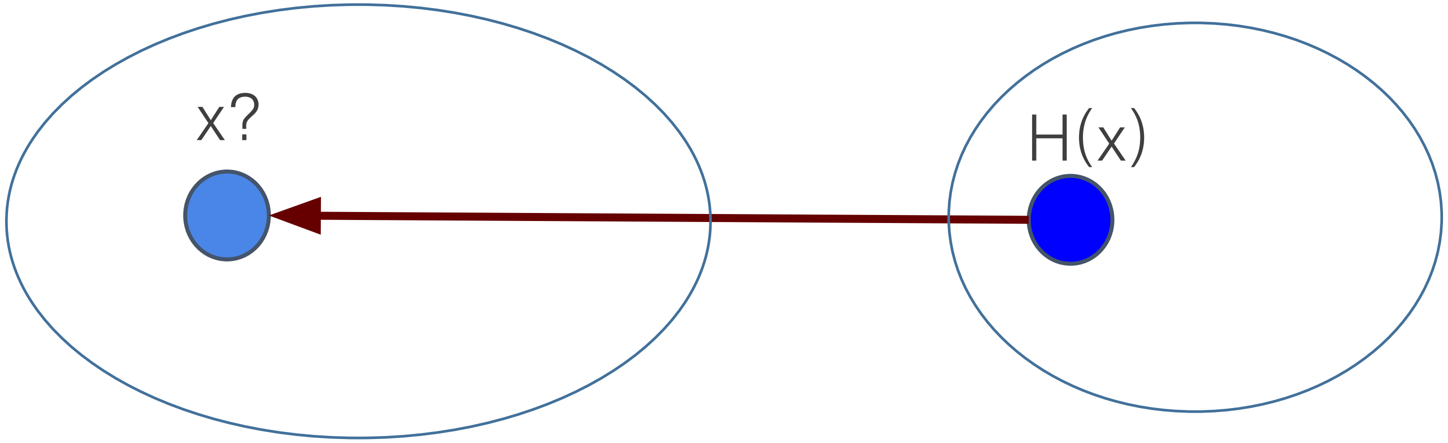
- SHA-256 (in SHA-2) (used heavily Bitcoin and others)
- KECCAK-256 (in SHA-3) (used in Ethereum)

Hash property #2: one-wayness

We want something like this:

"Given H(x), it is infeasible to find x"

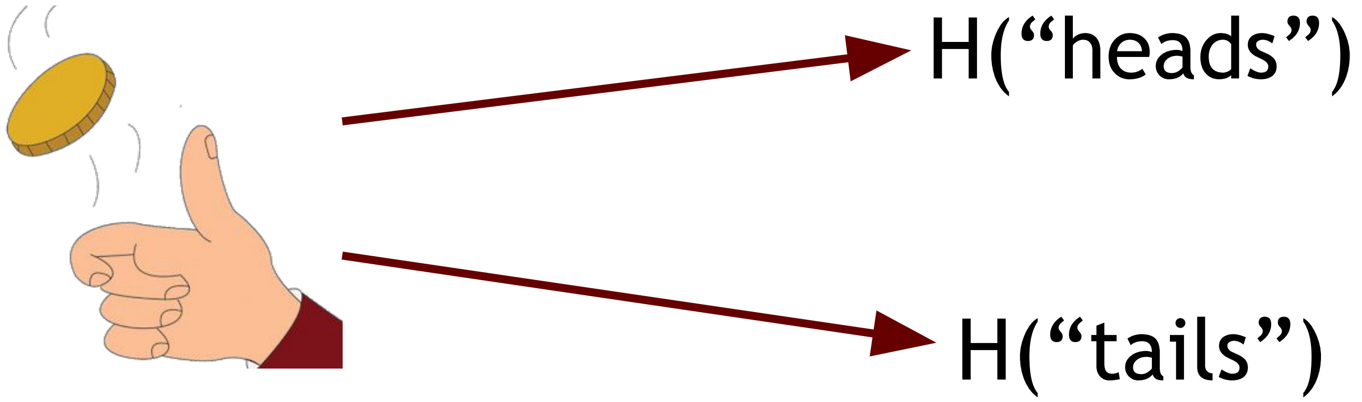
If H() has a k-bit image, we need to try $O(2^k)$ messages to find the pre-image for a given H(x)



aka pre-image attack resistant

Hash property #2: one-wayness

while H(x) is pre-image attack resistant….
But this breaks down if we know information about x:



easy to find x!

Hash property #2’: Hiding

If r is chosen from a probability distribution that has *high min-entropy*, then given $H(r \mid\mid x)$, it is infeasible to find x.

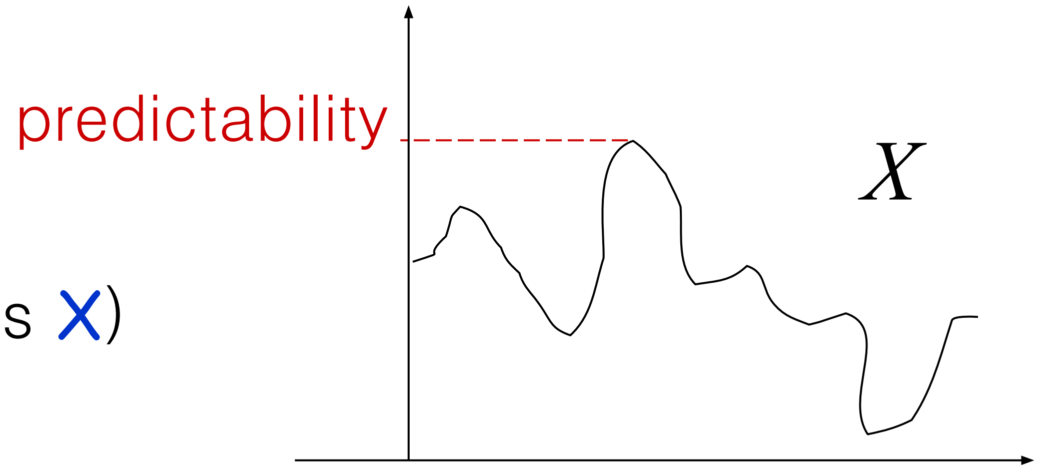
commit(x) := $H(r \mid\mid x)$ // com = commit(x)

verify(com, r, x) := [$H(r \mid\mid x) == com$]

High min-entropy means that the distribution has no particular value with probability above some low limit

entropy

- Let X be a random variable over alphabet Γ with distribution P
- Shannon Entropy
  - $H(X) = - \sum_{x \in \Gamma} P(x) \log_2 P(x)$
  - Quantifies uncertainty;
  - how to encode X on average (i.e. compress X)
- Min entropy
  - $H_{min}(X) = - \log_2 \max_{x \in \Gamma} P(x)$
  - How to encode the **most likely value** of X
- High min entropy means the most likely outcome has a small probability
  - $H_{min}(X) \geq m$, then the probability that Eve can guess the right outcome of X is at most 2^{-m}



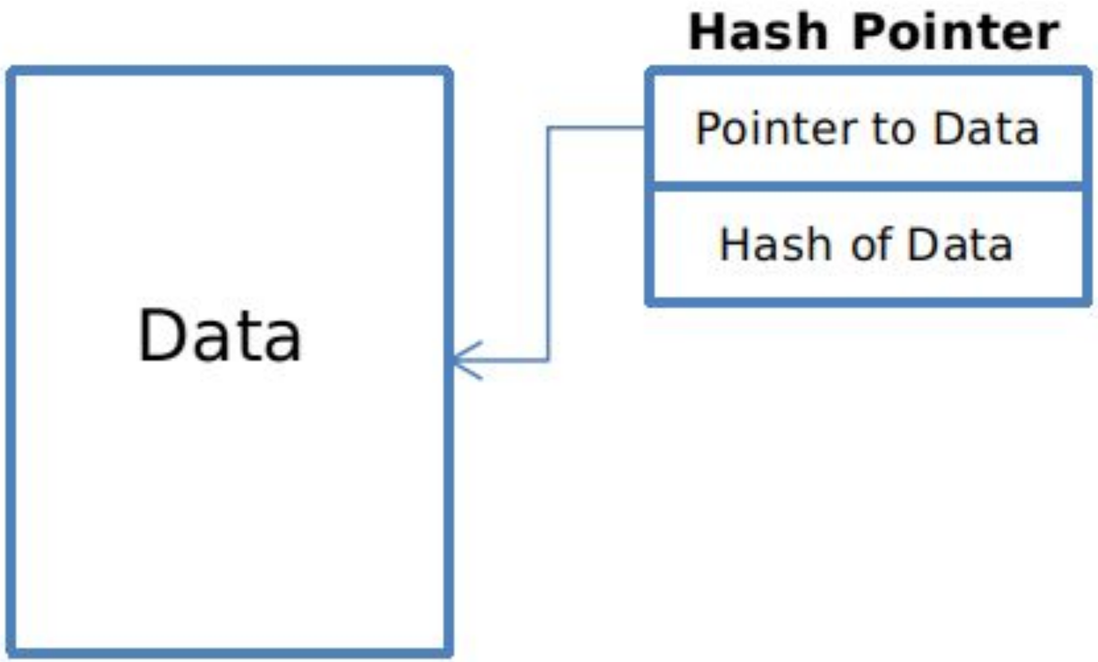
Hash property 3: puzzle friendliness

- A hash function $H(\cdot)$ is said to be puzzle friendly if for every possible k-bit output y, and if r is chosen from a uniform distribution, then it is infeasible to find x such that $H(r||x) = y$ in significantly less than $O(2^k)$ time
- Application:
  - A search puzzle that consists of
    - A hash fn. H
    - A value: id (called puzzle-id) chosen from a high min-entropy distri.
    - A target set Y
  - A solution is a value x such that
    - $H(id||x) \in Y$

2. Hash pointer, Blockchain, Merkle tree

Hash pointer

- a pointer to the place where some information is stored.
- Together with the pointer we store a cryptographic hash of the information

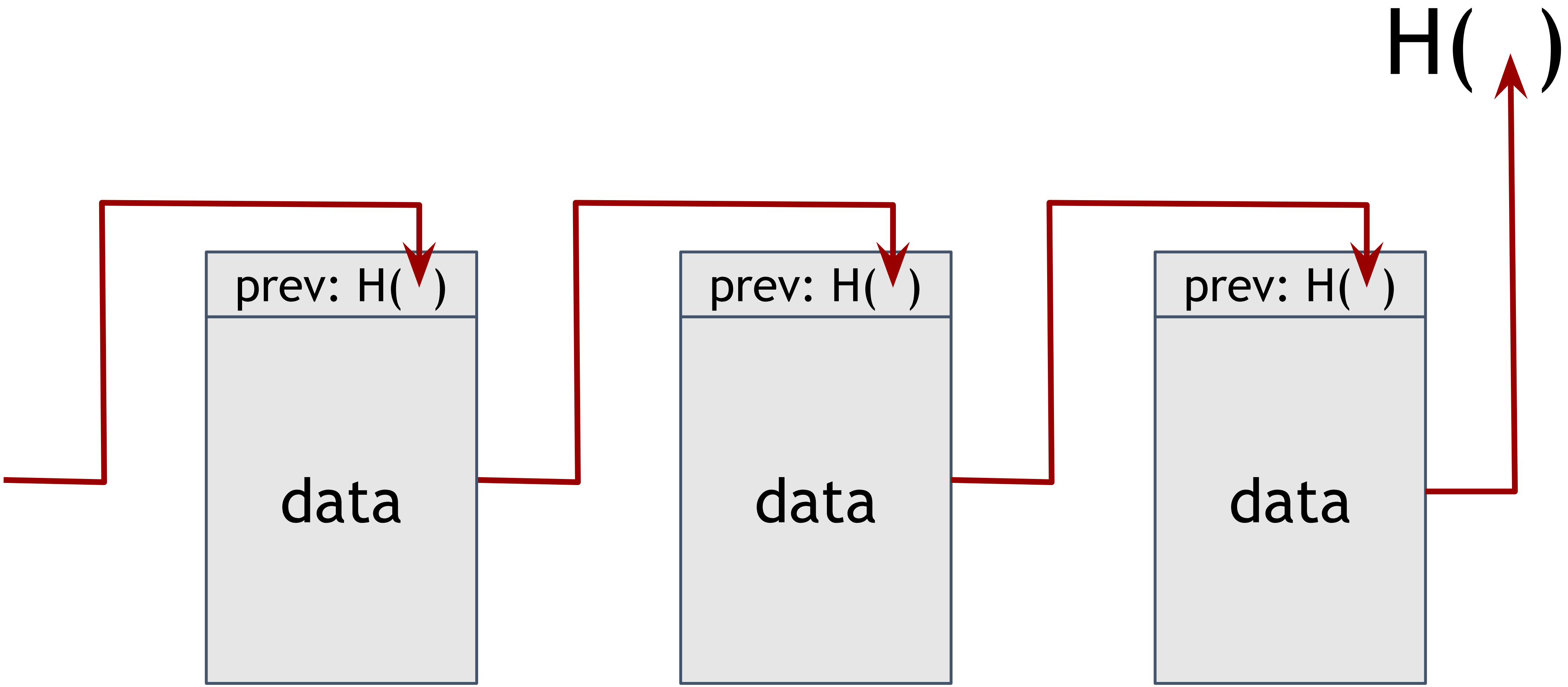


Source: Huabing Zhao@Medium

why do we need a hash pointer?

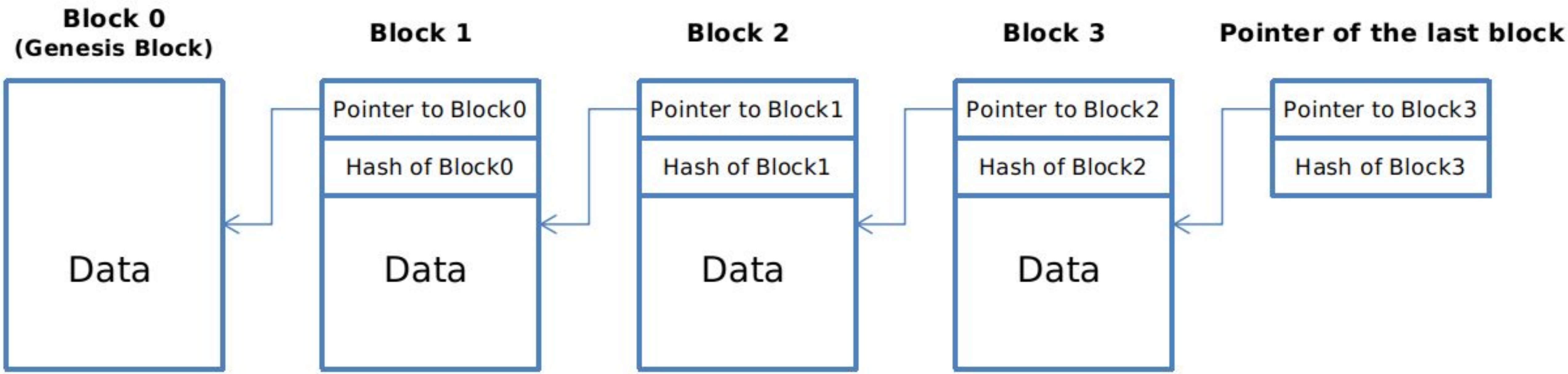
- if we have a hash pointer, what can we do?
  - ask to bring the original data back
  - verify that it hasn’t been changed

Blockchain: a linked list with hash pointers



Blockchain: a different picture

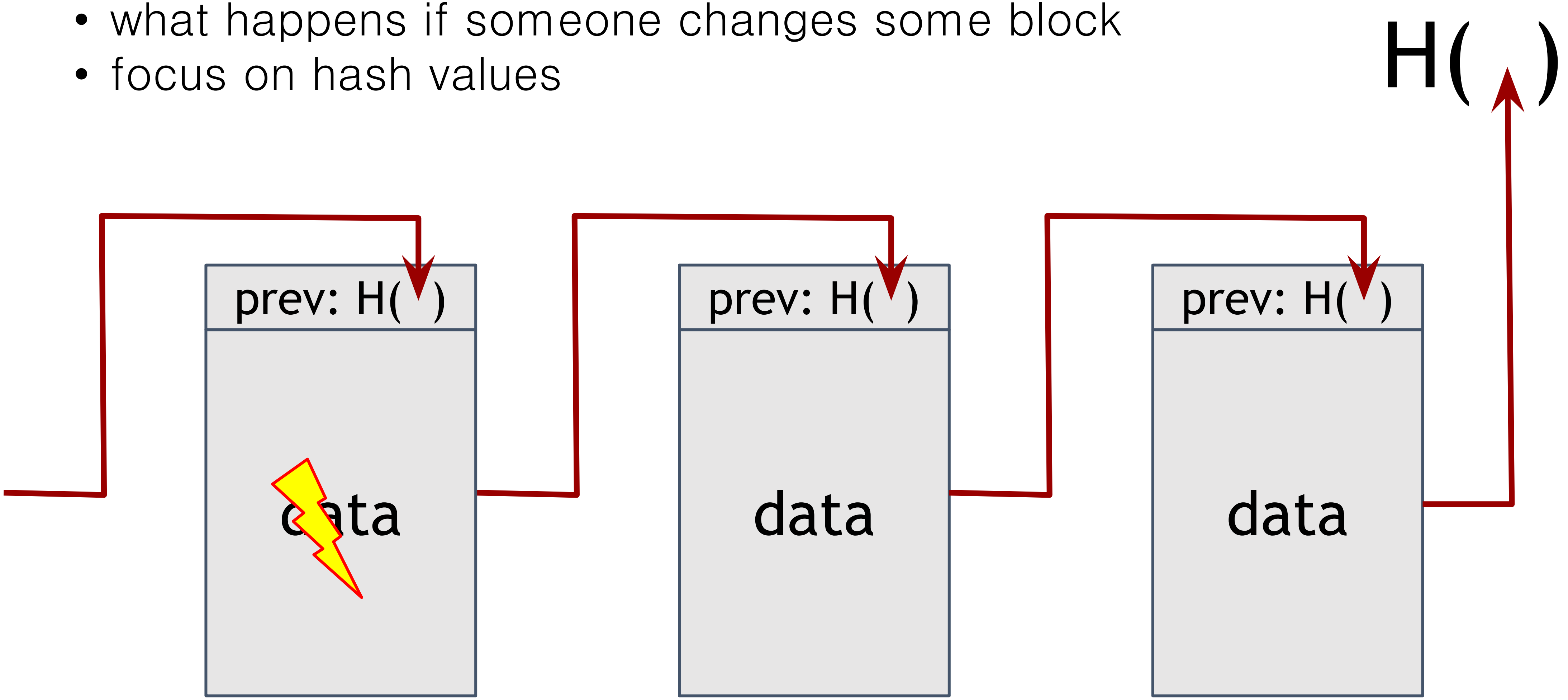
- a hash in a hash pointer is essentially the hash of all the previous blocks



Source: Huabing
Zhao@Medium

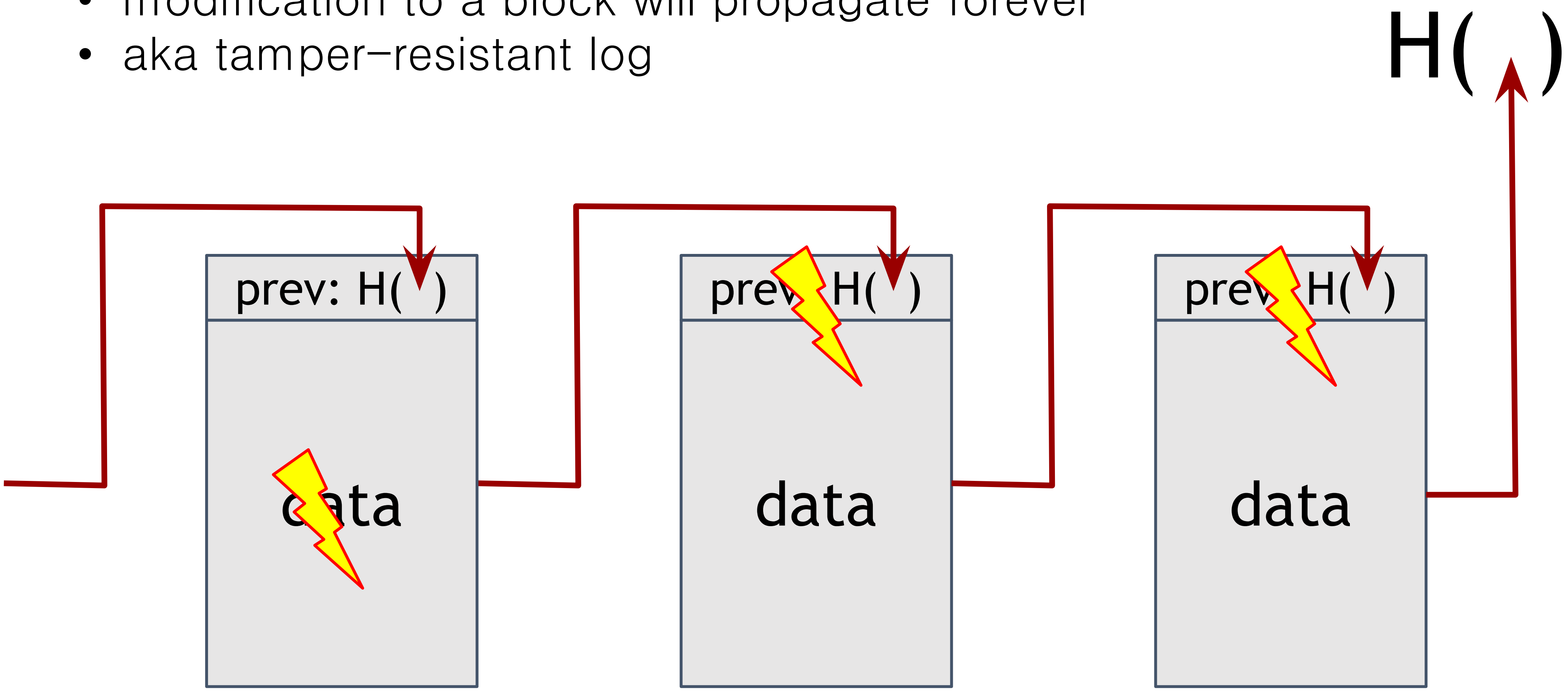
what is a blockchain for?

- what happens if someone changes some block
- focus on hash values



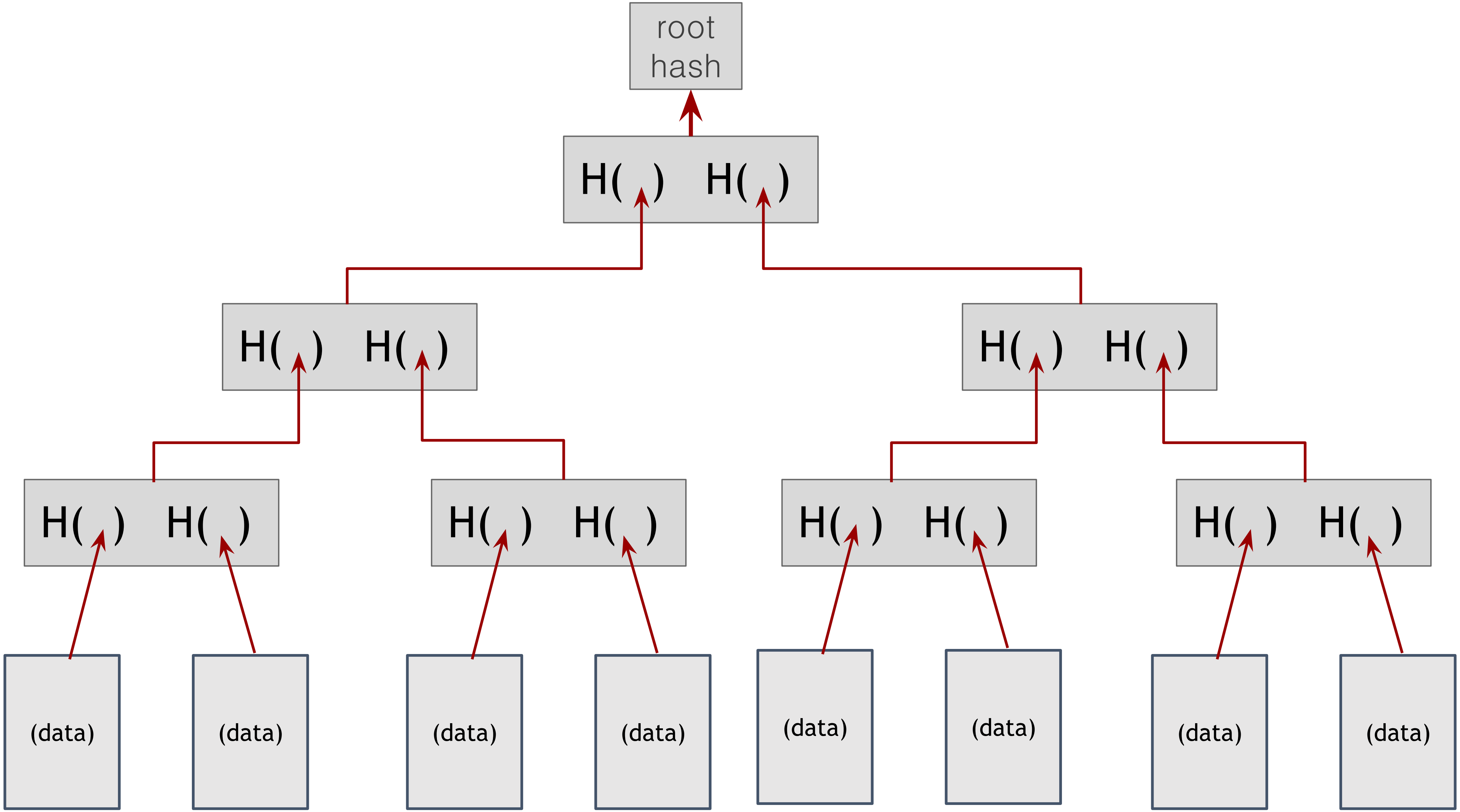
Modifications to any block will propagate forever

- modification to a block will propagate forever
- aka tamper-resistant log

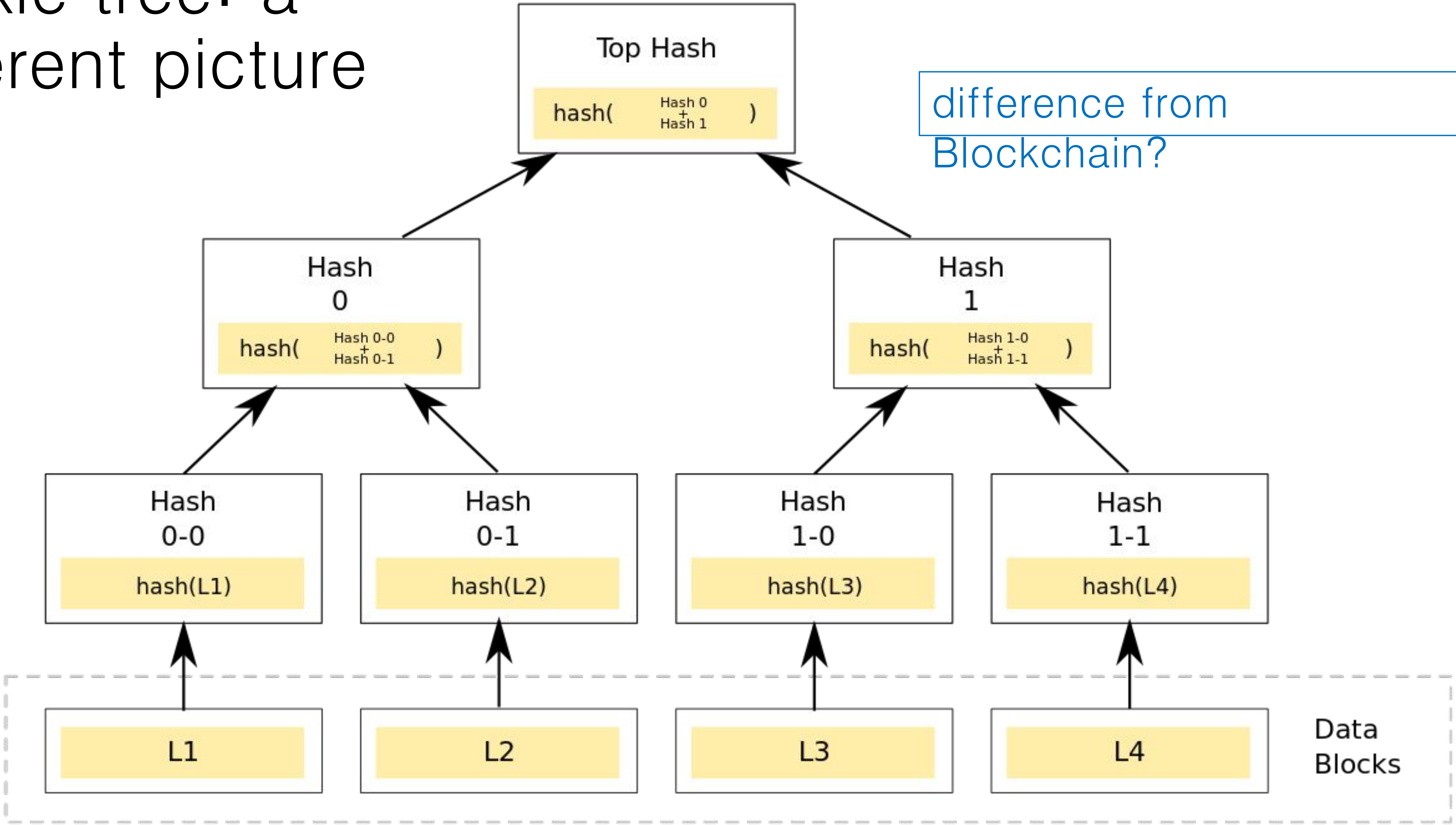


immutability

Merkle tree: binary tree with hash pointers



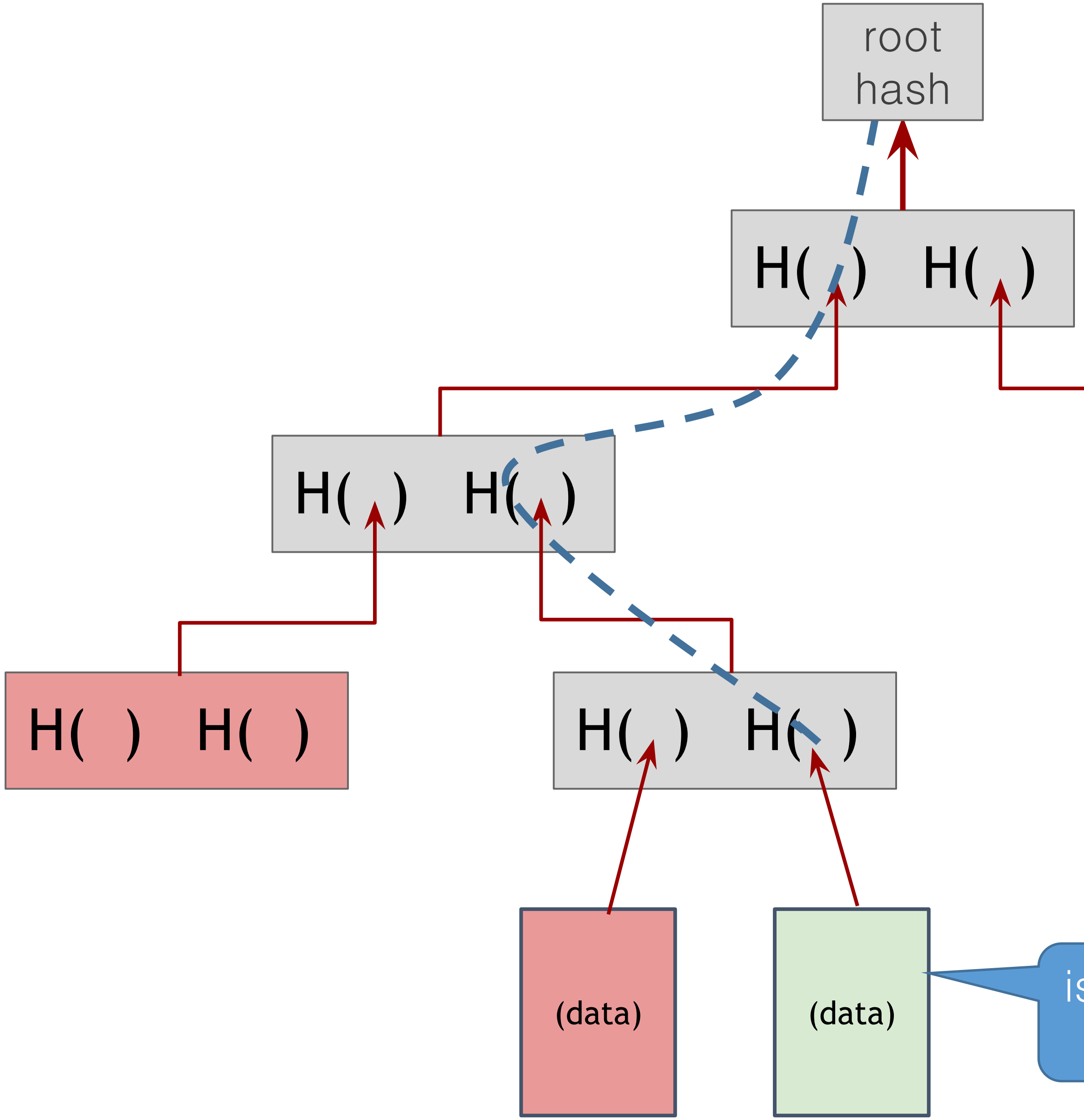
Merkle tree: a different picture



difference from Blockchain?

Source: Huabing Zhao@Medium

proving membership in a Merkle tree



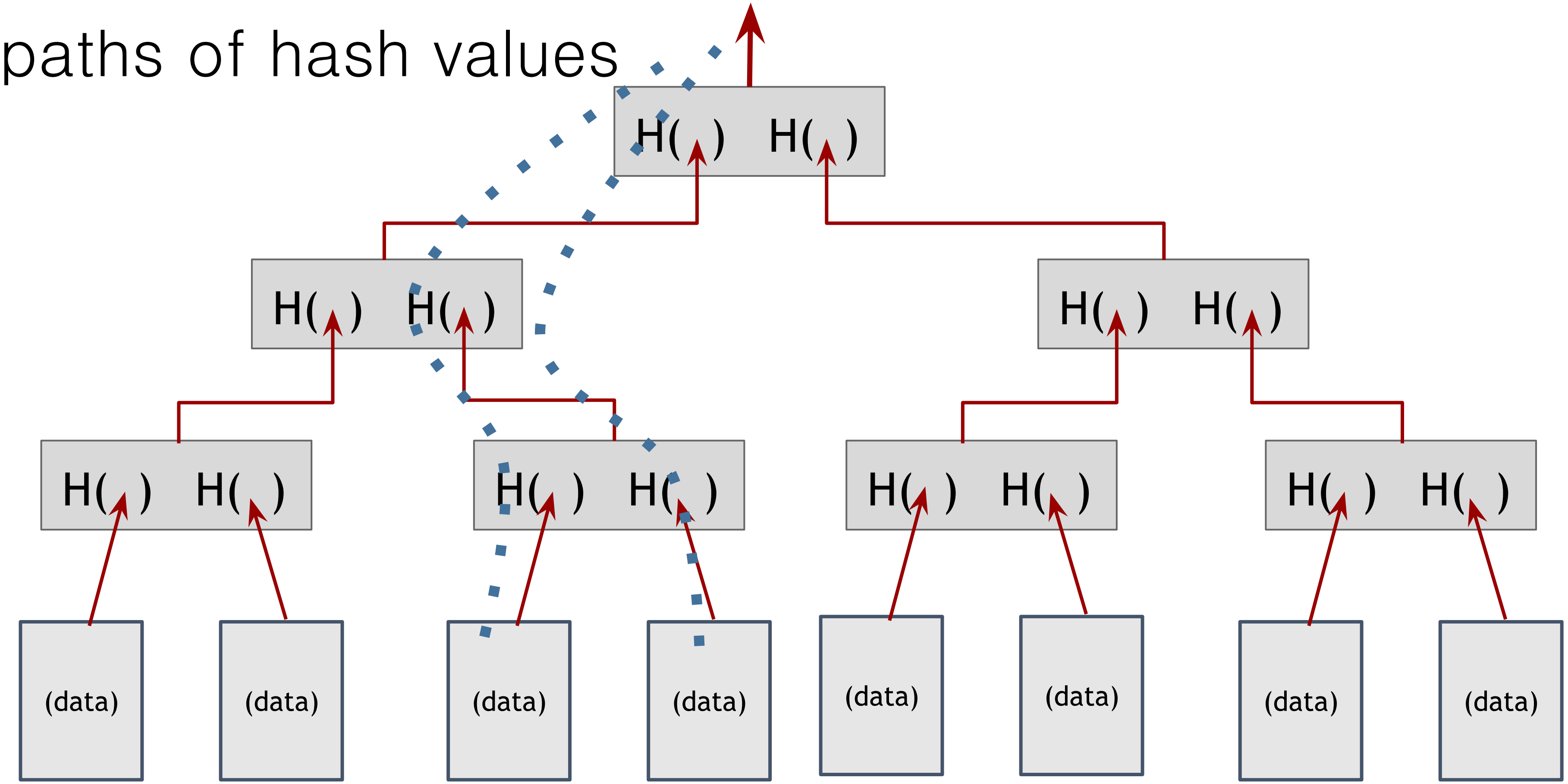
Show the hash values from data block to root

show O(log n) neighbors

is this data block included for the root hash value?

Proof of non-membership in Merkle tree

- Condition: Merkle tree is sorted
  - i.e., data blocks are sorted in order
- We can proof the non-existence of a particular item
- Show two paths of hash values



Comparison

	Blockchain	Merkle tree
Abstraction	list	set
Commitment size	O(1)	O(1)
Append	O(1)	O(lg n)
Update	O(n)	O(lg n)
Membership proof	O(n)	O(lg n)

3. Digital Signatures

Digital signatures

(sk, pk) := genKey(keysize)

sk: secret signing key

pk: public verification key

sig := sign(sk, message)

isValid := verify(pk, message, sig)

Requirements for signatures

1. correctness

   genKey(keysize) → (sk, pk)

   verify(pk, message, sign(sk, message)) == true

2. unforgeability

   adversary given pk

   may adaptively query an oracle for sign(m_i)

   cannot output a valid signature pair (σ, m') for any new message m'

* oracle always generates a valid signature σ_i for input message m_i

Unforgeability vs. non-malleability

unforgeability

adversary given pk

may adaptively query the oracle for sign(m_i)

cannot output a valid signature pair (σ, m’) for any new message m’

non-malleability

adversary given pk

may adaptively query oracle for sign(m_i)

cannot output a *new* valid signature pair (σ’, f(m_i))

Bitcoin uses ECDSA

- Elliptic Curve Digital Signature Algorithm
- curve used is `secp256k1`
- set of points $(x,y) \in \{F_p \times F_p \mid y^2 = x^3 + 7 \pmod p\}$
- $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
- Forms a group E, $|E| = q \approx p \approx 2^{256}$

	range	format	size (bits)
sk	$\mathbb{Z}_q$	random	256
pk	E	sk · G	512/257*
m	$\mathbb{Z}_q$	H(message)	256
sig	$\mathbb{Z}_q \times \mathbb{Z}_q$	(r, s)	512

Security level of Elliptic Curve Cryptography (ECC)

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

NIST Recommended Key Sizes

Identity, identifier, address

- identity 신원
    - who you are, the way you think about yourself, the way you are viewed by the world and the characteristics that define you
    - condition or character as to who a person or what an object is
    - name, social security #, thumbprint, DNA,…
- Identifier (ID) 식별자
    - a (unique) name/number that identifies (that is, labels the identity of) a unique object
    - Email address, Social Security #, passport #, domain name, URL,…
- Address
    - Location, information about some (real or virtual) place reachable
    - IP address, mail address, URL,…

In Bitcoin: a public key is an *identity*

- Anybody can get an identity with genKey
  - Collisions statistically negligible

- To “speak” as pk, sign using sk

- Keys are *pseudonyms*
  - real identities are anonymized

address in Bitcoin

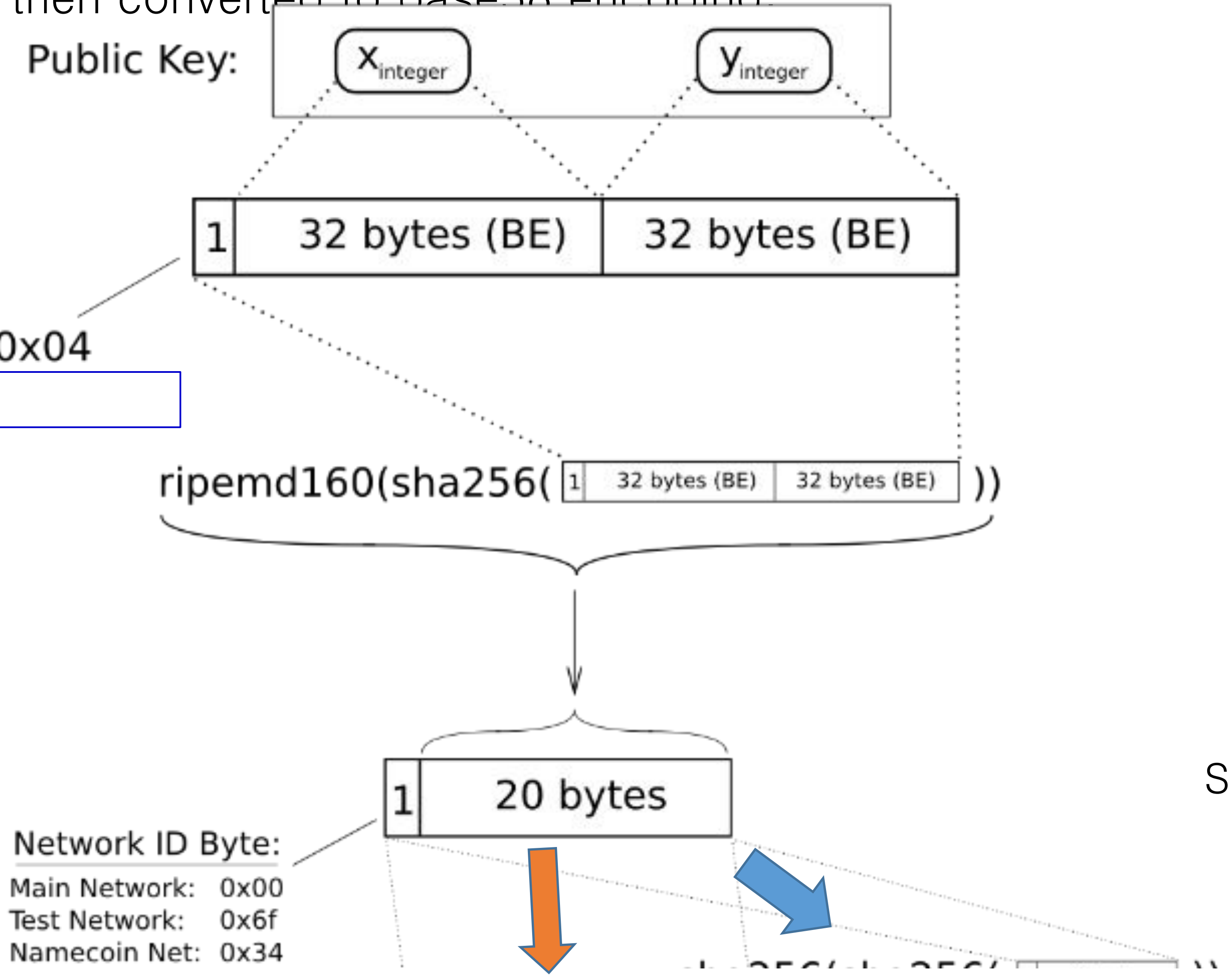
- a Bitcoin address is like a bank account number
- how about using a pk as an address?
  - pk is 512 bits... too long!
  - (x,y) in a curve, where each coordinate is 256 bit long
- Address
  - "Pay To PubKey Hash" (P2PKH)
  - Address ≈ H(pk), which is 160 bits + α
  - then converted to base58 encoding: 26–35 characters

base64 encoding: uppercase & lowercase alphabet, 0–9, +, /

base58 encoding: base64 encoding − `{I,l,0,O,+,/}`

Addresses in Bitcoin: “Pay To PubKey Hash” (P2PKH)

- pk is 2*256 bits (x,y)
- Address = H(pk) then converted to base58 encoding:



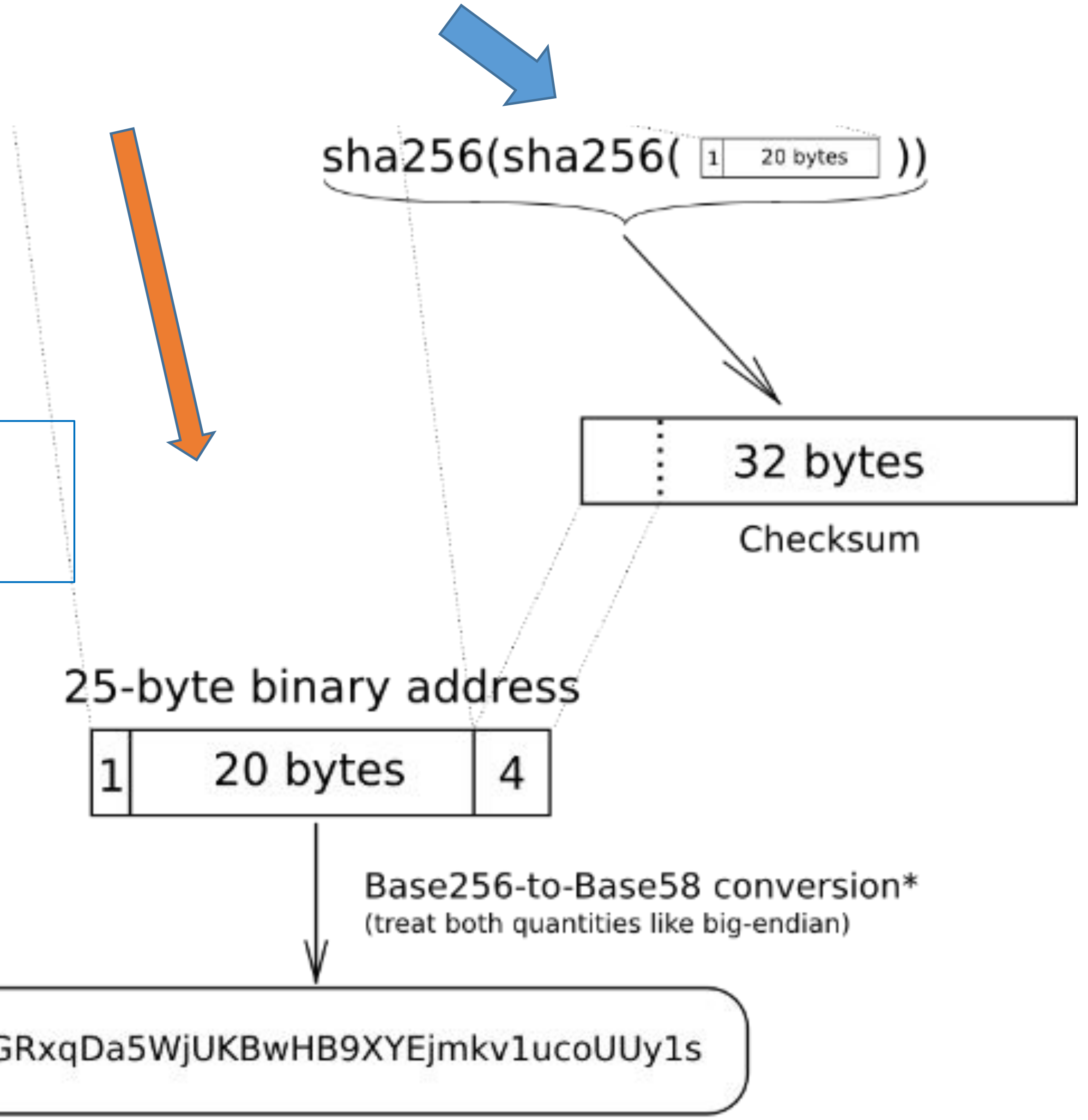
uncompressed vs compressed

*BE: binary encoding

Source: Wikipedia

Namecoin Net: 0x34

Base64: alphabet*2,digits,+,/
Base58: base64 – {0,O,I,l,+,/}



Source: Wikipedia

*In a standard base conversion, the 0x00 byte on the left would be irrelevant (like writing '052' instead of just '52'), but in the BTC network the left-most zero chars are carried through the conversion. So for every 0x00 byte on the left end of the binary address, we will attach one '1' character to the Base58 address. This is why main-network addresses all start with '1'

4. Simple cryptocurrencies

Alice transfers money to Bob

- receiver is Bob
  - Bob's address derived from his pk
- sender is Alice
  - Alice needs to sign the transaction (by her sk)
- two ways of changes
  - new coins are created (almost periodically)
  - existing coins are transferred among users

GoofyCoin

Goofy can create new coins

New coins belong to me.

Signed by sk, and its pk belongs to Goofy

signed by pk_{Goofy}
CreateCoin [uniqueCoinID]

only Goofy can mint coins!

source: https://docs.google.com/presentation/d/13SOZqHzLRryyFXw26CCE89lKMa8X9YAhdTFgXdw0a1s/view#slide=id.p

A coin’s owner can spend it.

I’m paying Alice

signed by pk_{Goofy}

Pay to pk_{Alice} : H( )

signed by pk_{Goofy}

CreateCoin [uniqueCoinID]

Goofy pays Alice to buy something

source: https://docs.google.com/presentation/d/13SOZqHzLRryyFXw26CCE89IKMa8X9YAhdTFgXdw0a1s/view#slide=id.p

The recipient can pass on the coin again.

signed by pk_{Alice}

Pay to pk_{Bob} : H( )

signed by pk_{Goofy}

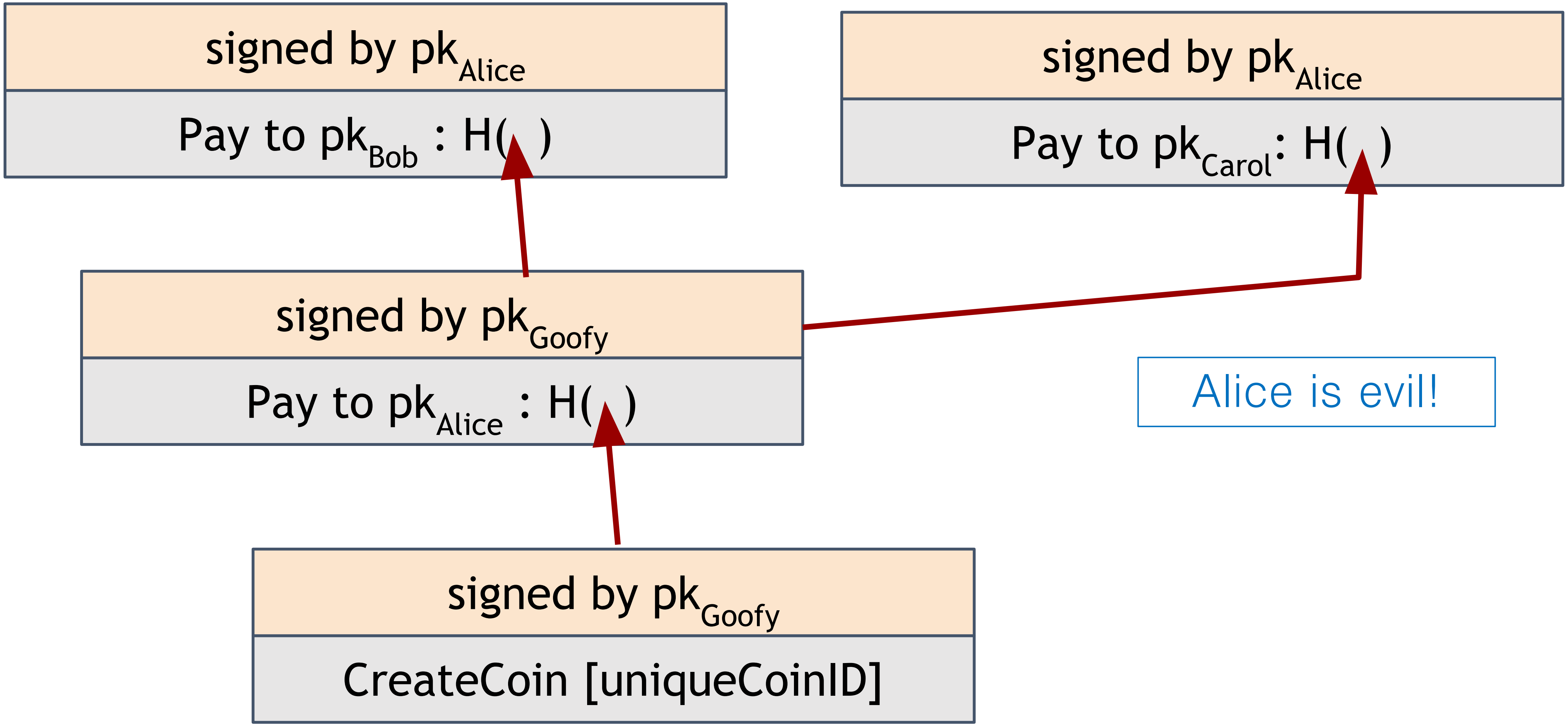
Pay to pk_{Alice} : H( )

signed by pk_{Goofy}

CreateCoin [uniqueCoinID]

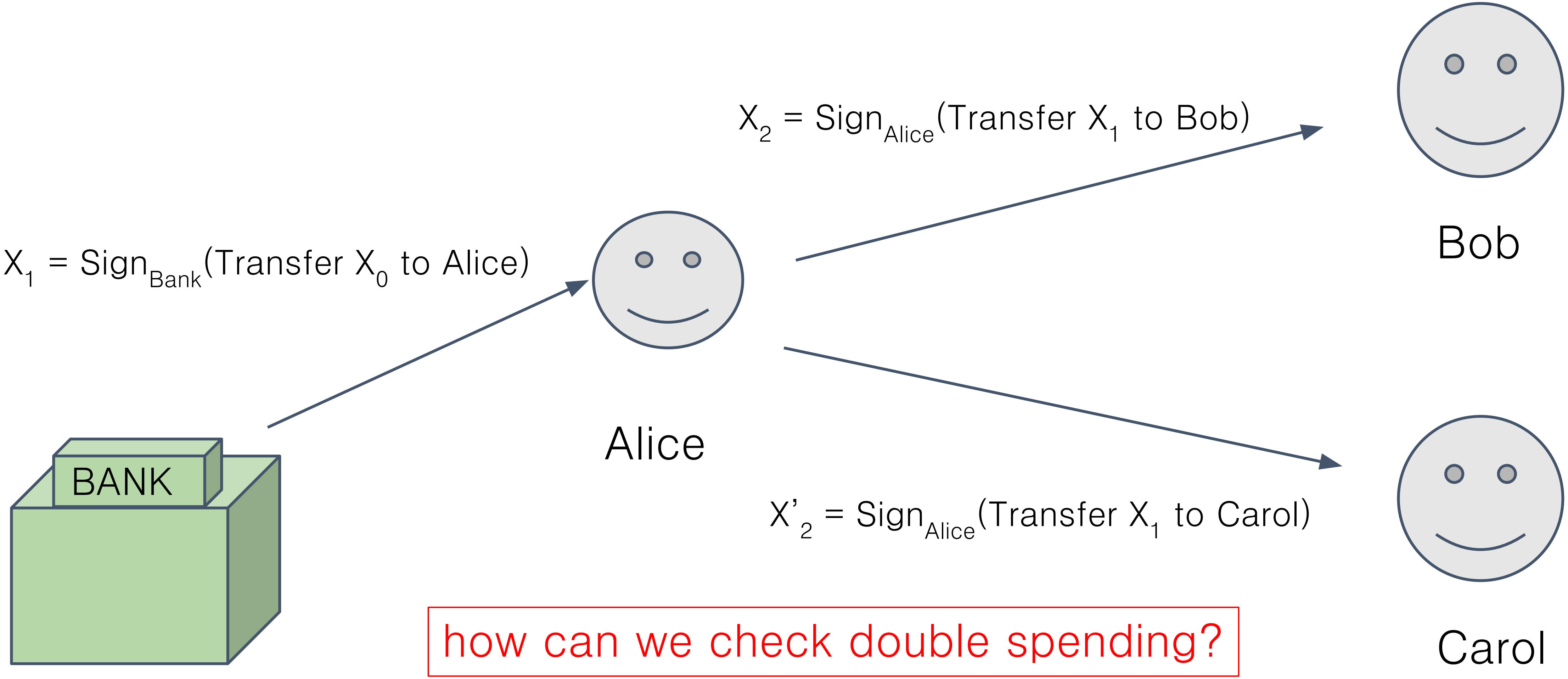
source: https://docs.google.com/presentation/d/13SOZqHzLRryyFXw26CCE89lKMa8X9YAhdTFgXdw0a1s/view#slide=id.p

double-spending attack



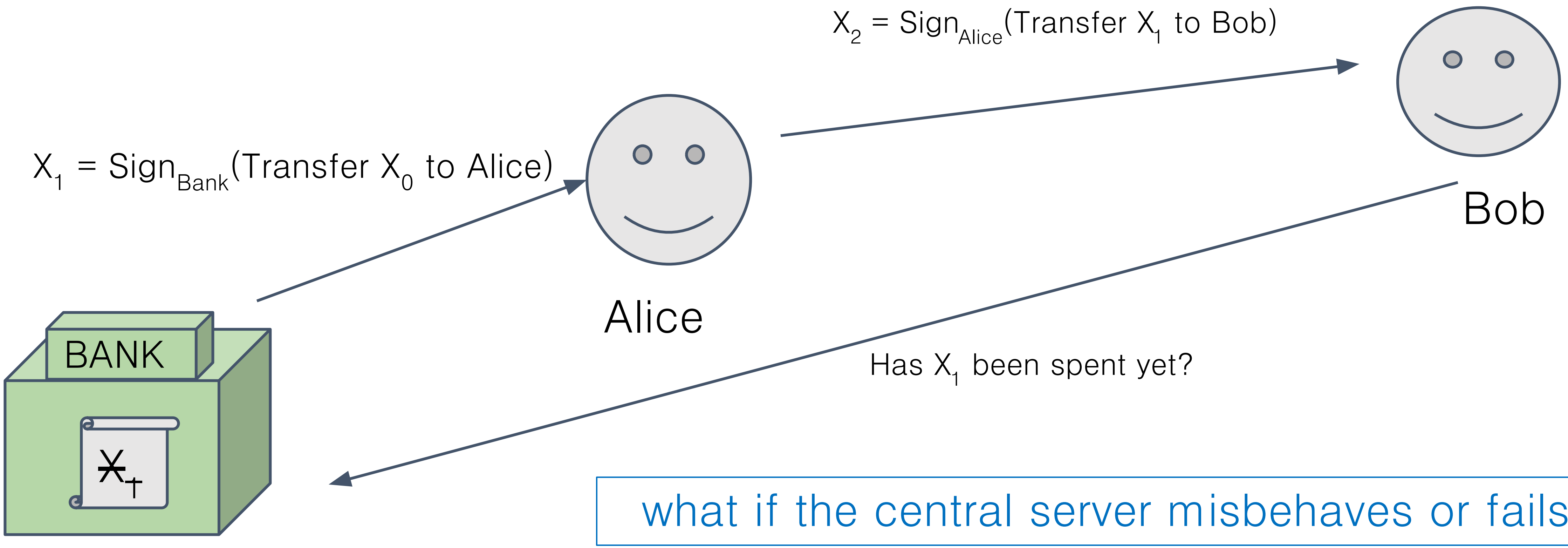
Alice is evil!

Double-spends must be prevented



how can we check double spending?

Centralized approach: talk to the issuer



what if the central server misbehaves or fails?

Bitcoin's approach: global ledger

Transfer X_0 Issuer→Alice, $Sign_{Issuer}$

Transfer X_1 Alice→Bob, $Sign_{Alice}$

Transfer X_2 Bob→Carol, $Sign_{Bob}$

...

time

every member keeps track of the ledger

Bitcoin's approach: global ledger

Transfer X_0 Issuer→Alice, $\text{Sign}_{\text{Issuer}}$

Transfer X_1 Alice→Bob, $\text{Sign}_{\text{Alice}}$

Transfer X_2 Bob→Carol, Sign_{Bob}

Transfer X_1 Alice→David, $\text{Sign}_{\text{Alice}}$

clearly invalid!

time

Ledger implementation: blockchain