

BITCOIN MECHANICS & OPTIMIZATIONS: A TECHNICAL OVERVIEW

Rustie Lin
Gloria Wang



BLOCKCHAIN
AT BERKELEY

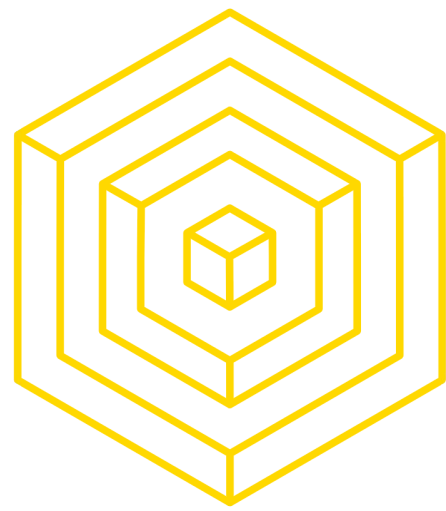


LECTURE OVERVIEW

- 1 CRYPTOGRAPHIC HASH FUNCTIONS
- 2 A TAMPER-EVIDENT DATABASE
- 3 SIGS, ECDSA, AND ADDRESSES
- 4 BITCOIN SCRIPT
- 5 P2PKH & P2SH



1 CRYPTOGRAPHIC HASH FUNCTIONS

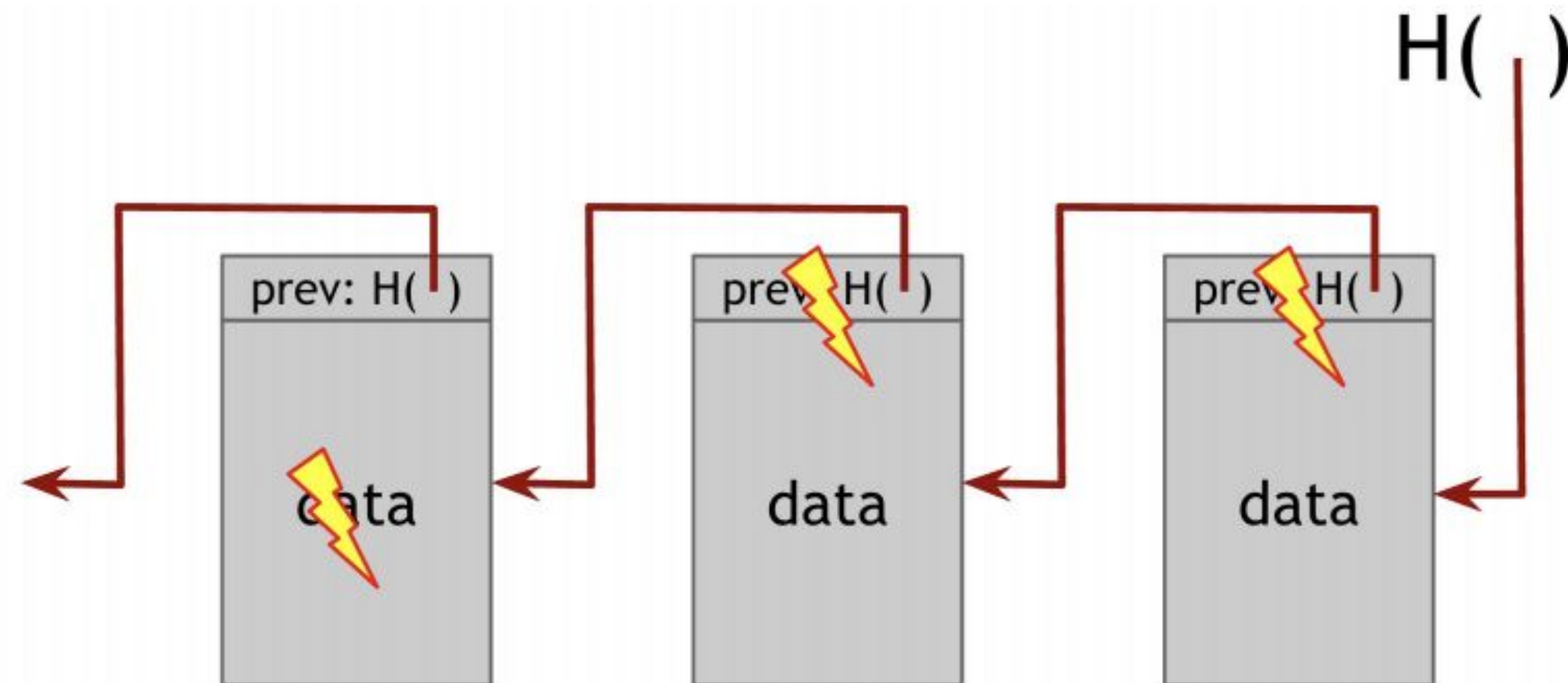


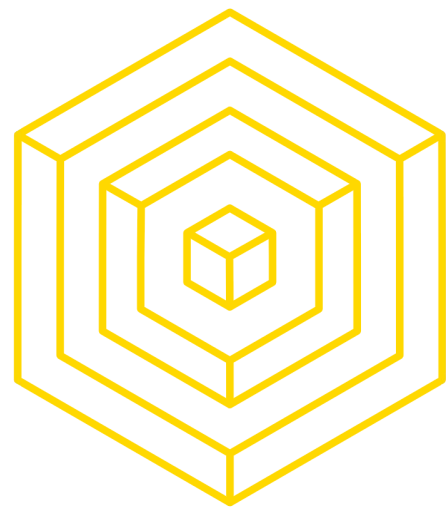
MOTIVATION

PROBLEMS IN BITCOIN

The problems Bitcoin has to face:

- Distrust of actors
- Transaction data being tampered with





CRYPTOGRAPHIC HASH FUNCTIONS

INTEGRITY OF INFORMATION

How do we ensure trust in communication in a trustless environment?

⇒ With **cryptographic hash functions**

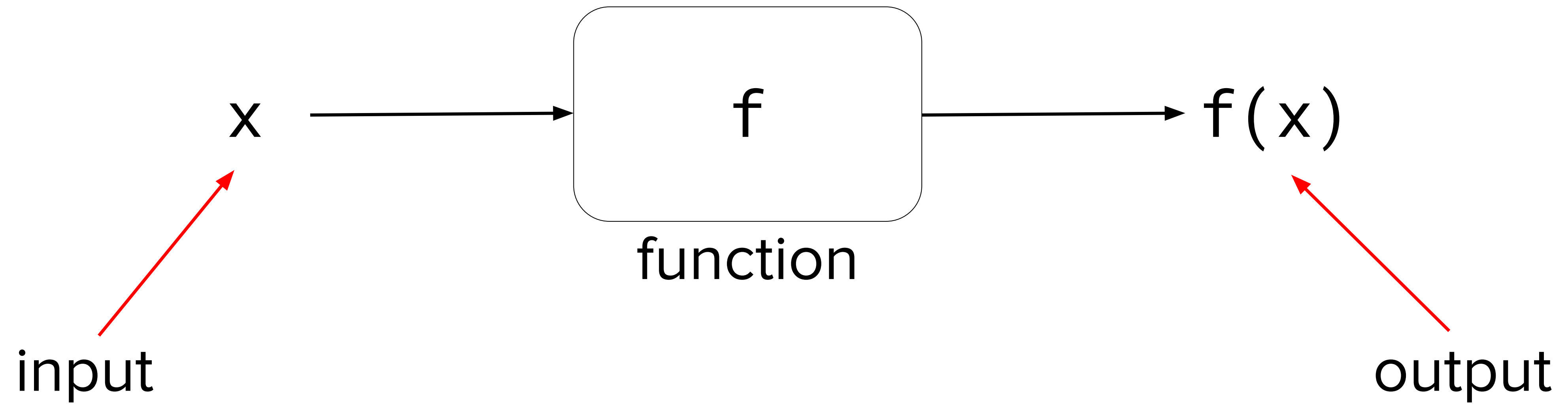


Image source: https://spiritegg.com/wp-content/uploads/2016/03/63180952_fingerprint_types624.jpg



CRYPTOGRAPHIC HASH FUNCTIONS

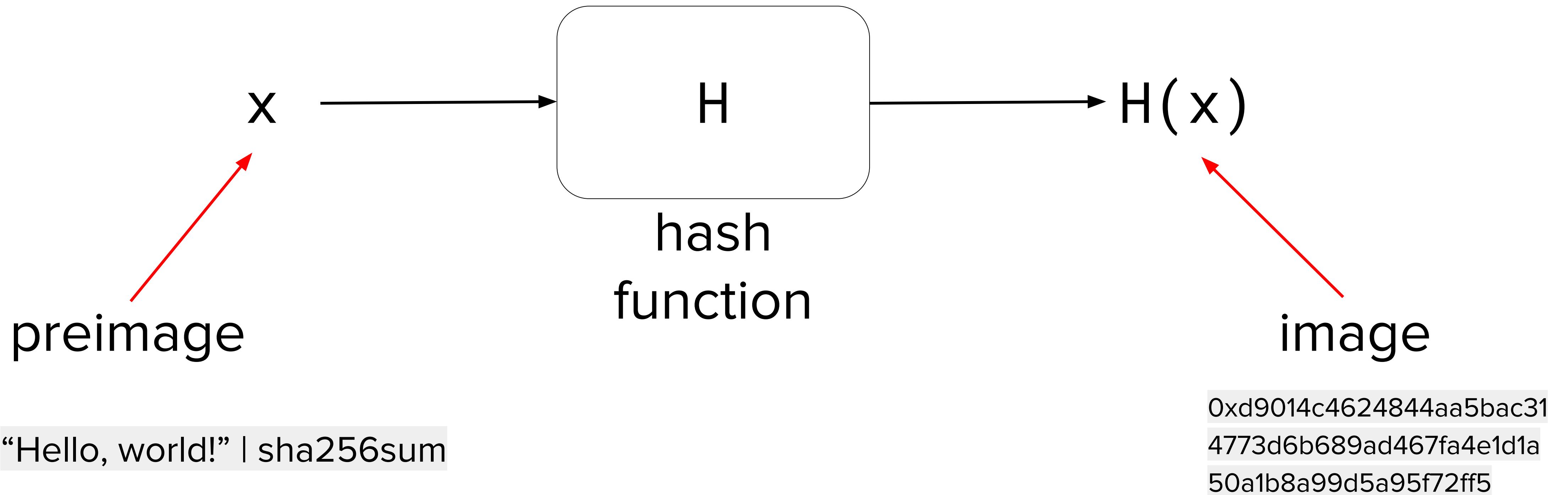
WHAT IS A HASH FUNCTION?





CRYPTOGRAPHIC HASH FUNCTIONS

WHAT IS A HASH FUNCTION?





CRYPTOGRAPHIC HASH FUNCTIONS

#CRYPTOGRAPHY

Cryptographic hash function:

A hash function with three special properties:

- Preimage resistance
- Second preimage resistance
- Collision resistance

The equivalent of **mathematical fingerprints/identifiers**

Image source:

http://chimera.labs.oreilly.com/books/1234000001802/ch08.html#_proof_of_work_algorithm

AUTHOR: NADIR AKHTAR

```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```




CRYPTOGRAPHIC HASH FUNCTIONS

PREIMAGE RESISTANCE

Preimage resistance:

Given $H(x)$, it is computationally difficult to determine x



Fingerprint analogy:

Whose fingerprint is this?



CRYPTOGRAPHIC HASH FUNCTIONS

SECOND PREIMAGE RESISTANCE

Second preimage resistance:

Given x , it is computationally difficult to find some value x' such that $H(x) == H(x')$



Fingerprint analogy:

Can you find someone with the same fingerprint as you?



CRYPTOGRAPHIC HASH FUNCTIONS

COLLISION RESISTANCE

Collision resistance:

It is computationally difficult to

find x and y such that

$$H(x) == H(y)$$

Fingerprint analogy:

Can you find two random people
with the same fingerprint?





CRYPTOGRAPHIC HASH FUNCTIONS

AVALANCHE EFFECT

Avalanche effect: a small change in the input produces a pseudorandom change in the output

- Often a significant difference from the first output
- Prevents “hot or cold” game with inputs to produce or predict outputs

```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```




CRYPTOGRAPHIC HASH FUNCTIONS

SHA-256²

SHA-256: A cryptographic hash function designed by the NSA

Bitcoin uses **SHA-256²** (“SHA-256 squared”), meaning that $H(x)$ actually means $\text{SHA256}(\text{SHA256}(x))$

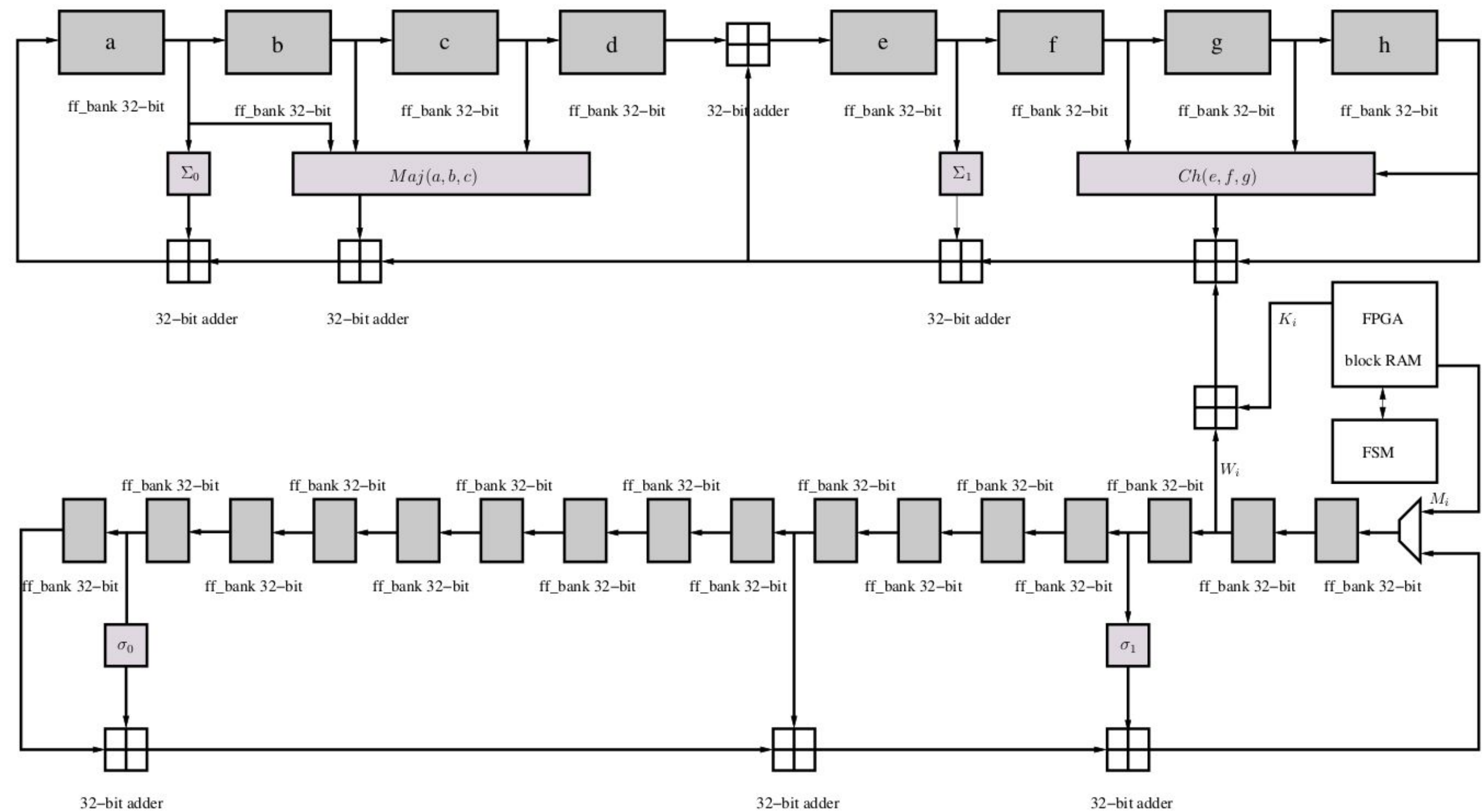


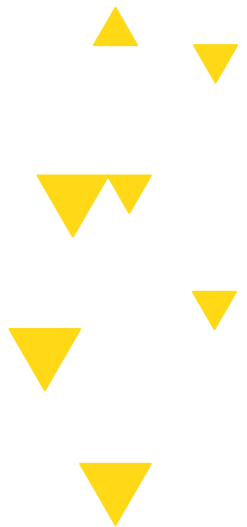
Image source:

<https://opencores.org/usercontent/img.1375985843>



2

A TAMPER-EVIDENT DATABASE





A TAMPER-EVIDENT DATABASE

EXAMPLE BLOCK

Block #485963

Summary		Hashes	
Number Of Transactions	2055	Hash	00000000000000000000000013942c4215cd92306bbce769cfc349d0b42f031c994eb
Output Total	4,819.27194588 BTC	Previous Block	0000000000000000000000004a5b64638b5d96d367a6d4e0a435fd460f972f1fb8f56b
Estimated Transaction Volume	1,770.2727223 BTC	Next Block(s)	
Transaction Fees	1.05055103 BTC	Merkle Root	ddb4970913d63bcb0c32a6d26fb9e792f8cd332ddf9c830a23c3e191608ce51a
Height	485963 (Main Chain)	Sponsored Link	
Timestamp	2017-09-19 02:11:37		
Received Time	2017-09-19 02:11:37		
Relayed By	BTC.TOP		
Difficulty	1,103,400,932,964.29		

Source: <https://blockchain.info/block/00000000000000000000000013942c4215cd92306bbce769cfc349d0b42f031c994eb>



A TAMPER-EVIDENT DATABASE

EXAMPLE BLOCK

```
{
  "hash": "00000000000000000000000013942c4215cd92306bbce769cfc349d0b42f031c994eb",
  "ver": 536870912,
  "prev_block": "0000000000000000000000004a5b64638b5d96d367a6d4e0a435fd460f972f1fb8f56b",
  "mrkl_root": "ddb4970913d63bcb0c32a6d26fb9e792f8cd332ddf9c830a23c3e191608ce51a",
  "time": 1505787097,
  "bits": 402718488,
  "fee": 105055103,
  "nonce": 600608926,
  "n_tx": 2055,
  "size": 999347,
  "block_index": 1625458,
  "main_chain": true,
  "height": 485963,
  "received_time": 1505787097,
  "relayed_by": "0.0.0.0",

  "tx": [
```

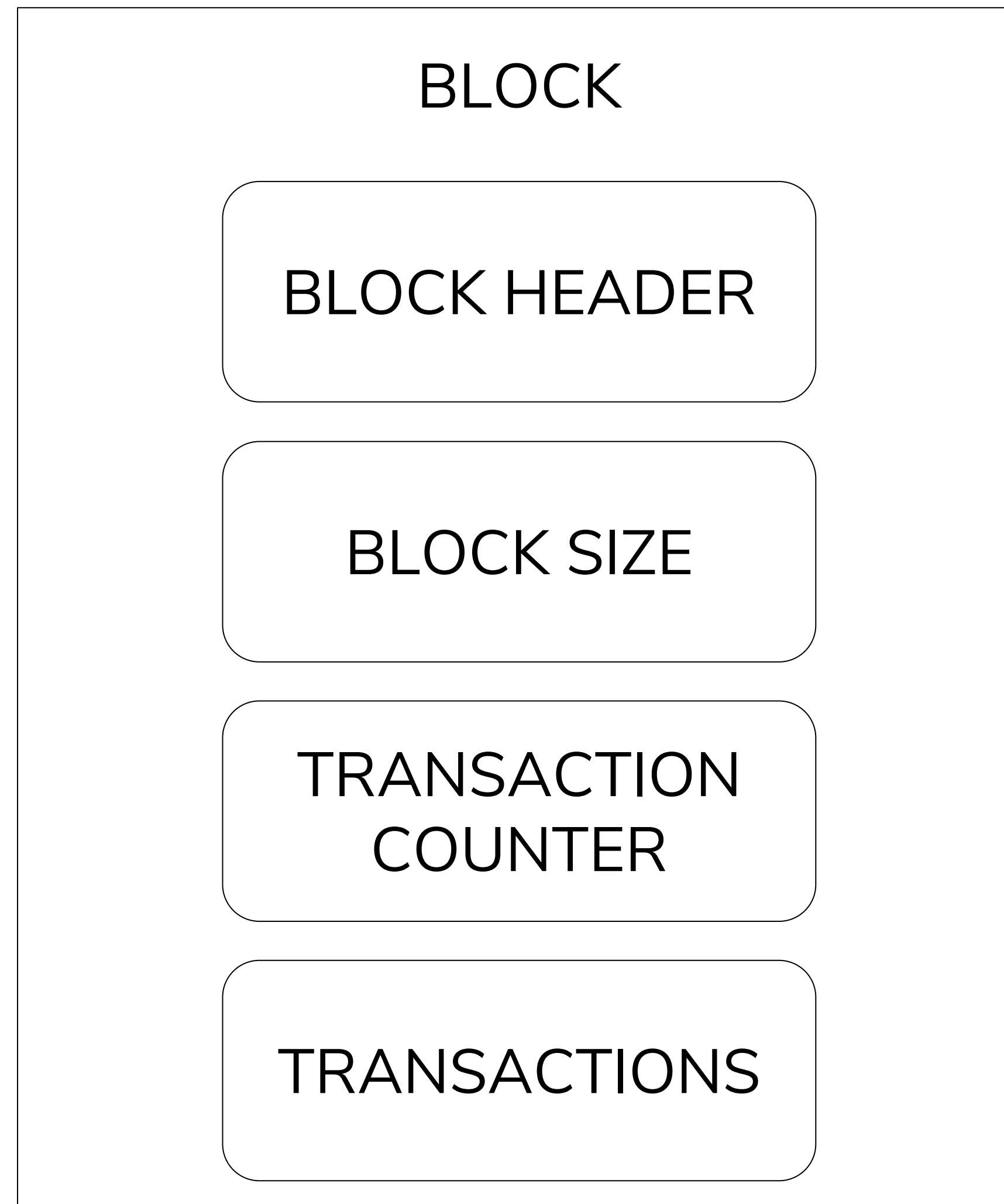
Source: <https://blockchain.info/block/00000000000000000000000013942c4215cd92306bbce769cfc349d0b42f031c994eb>





A TAMPER-EVIDENT DATABASE

DISSECTING THE BLOCKCHAIN



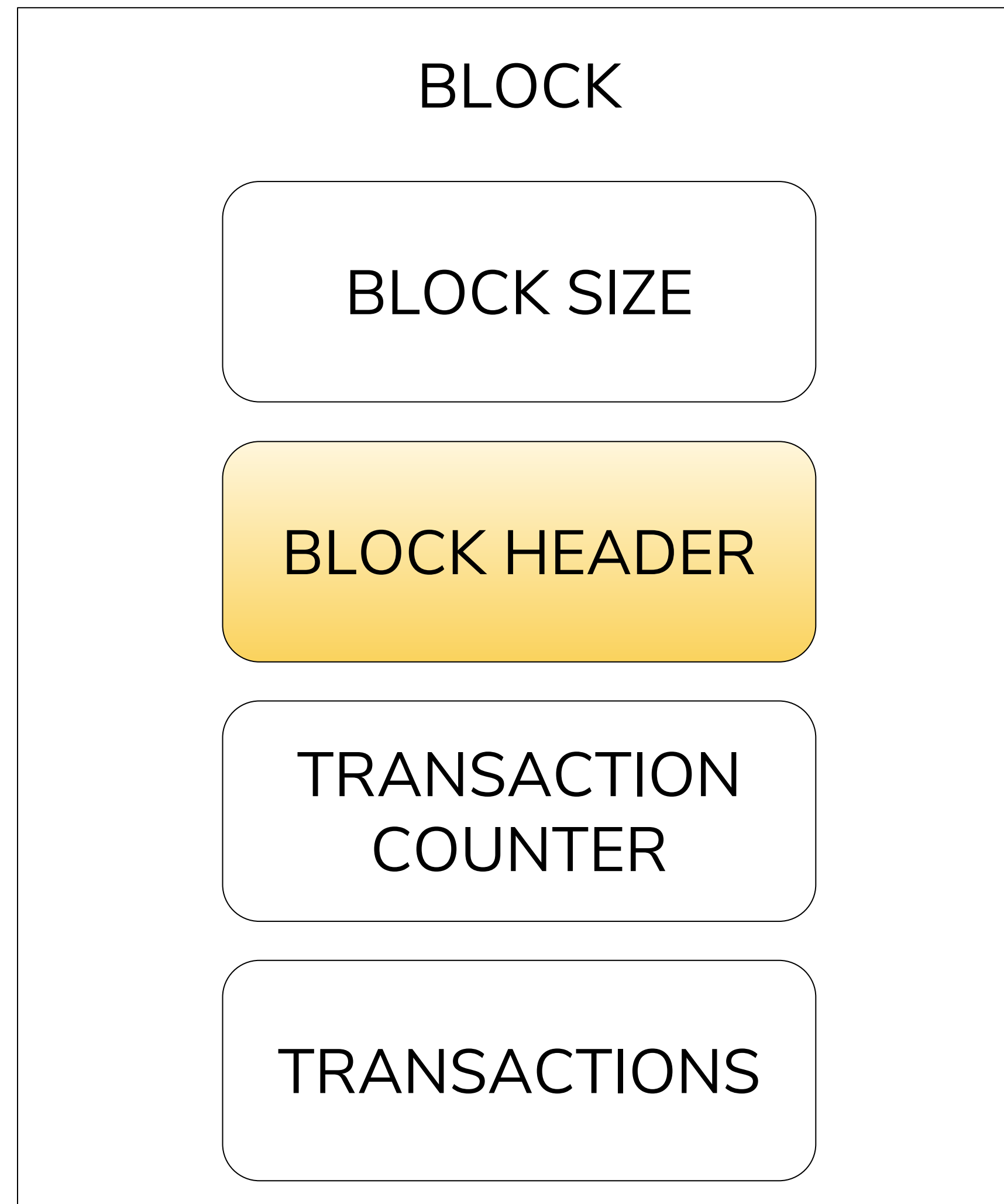
Source: Mastering Bitcoin

AUTHOR: MENGYI WANG



A TAMPER-EVIDENT DATABASE

DISSECTING THE BLOCKCHAIN



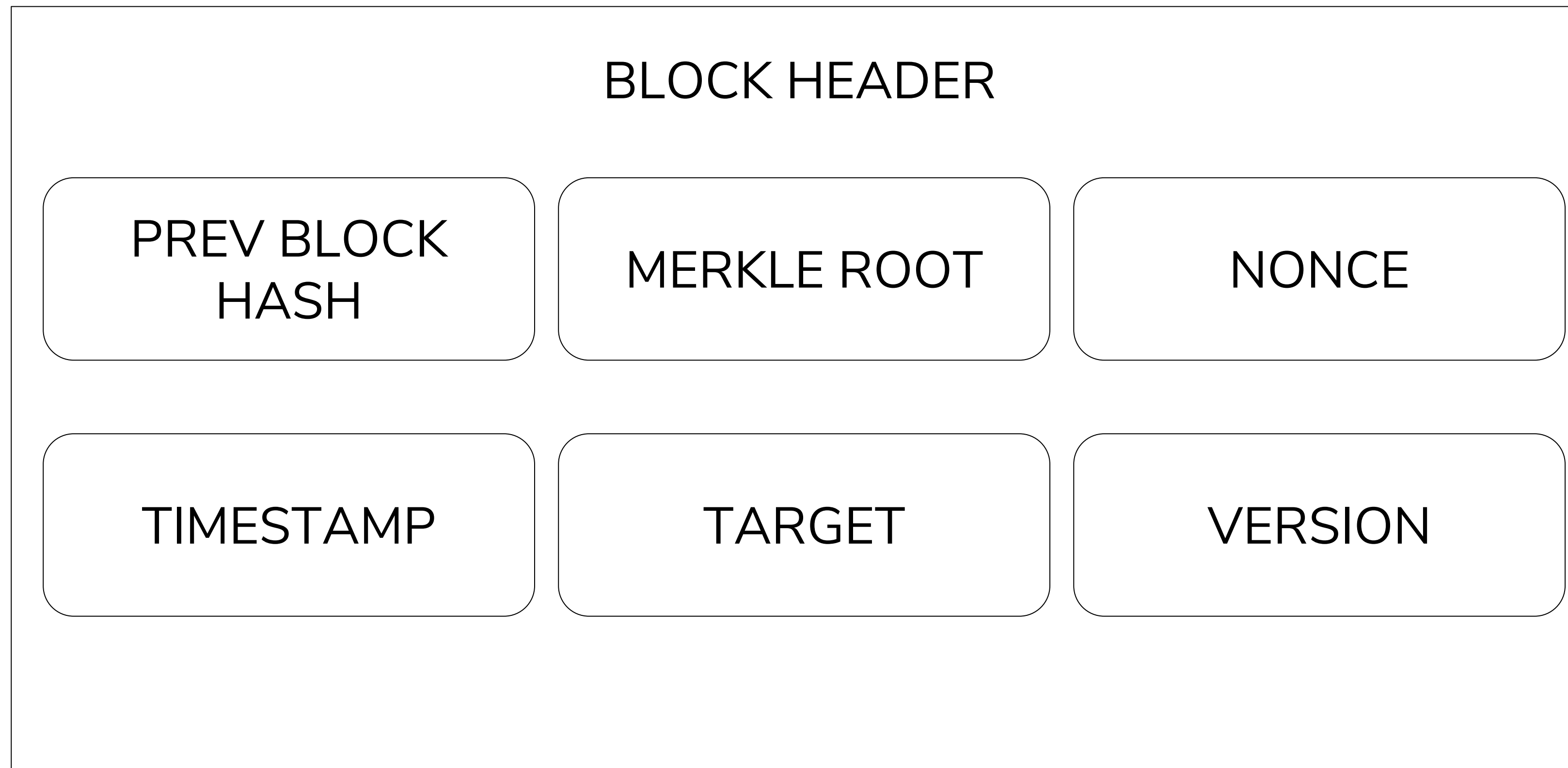
Source: Mastering Bitcoin

AUTHOR: MENGYI WANG



A TAMPER-EVIDENT DATABASE

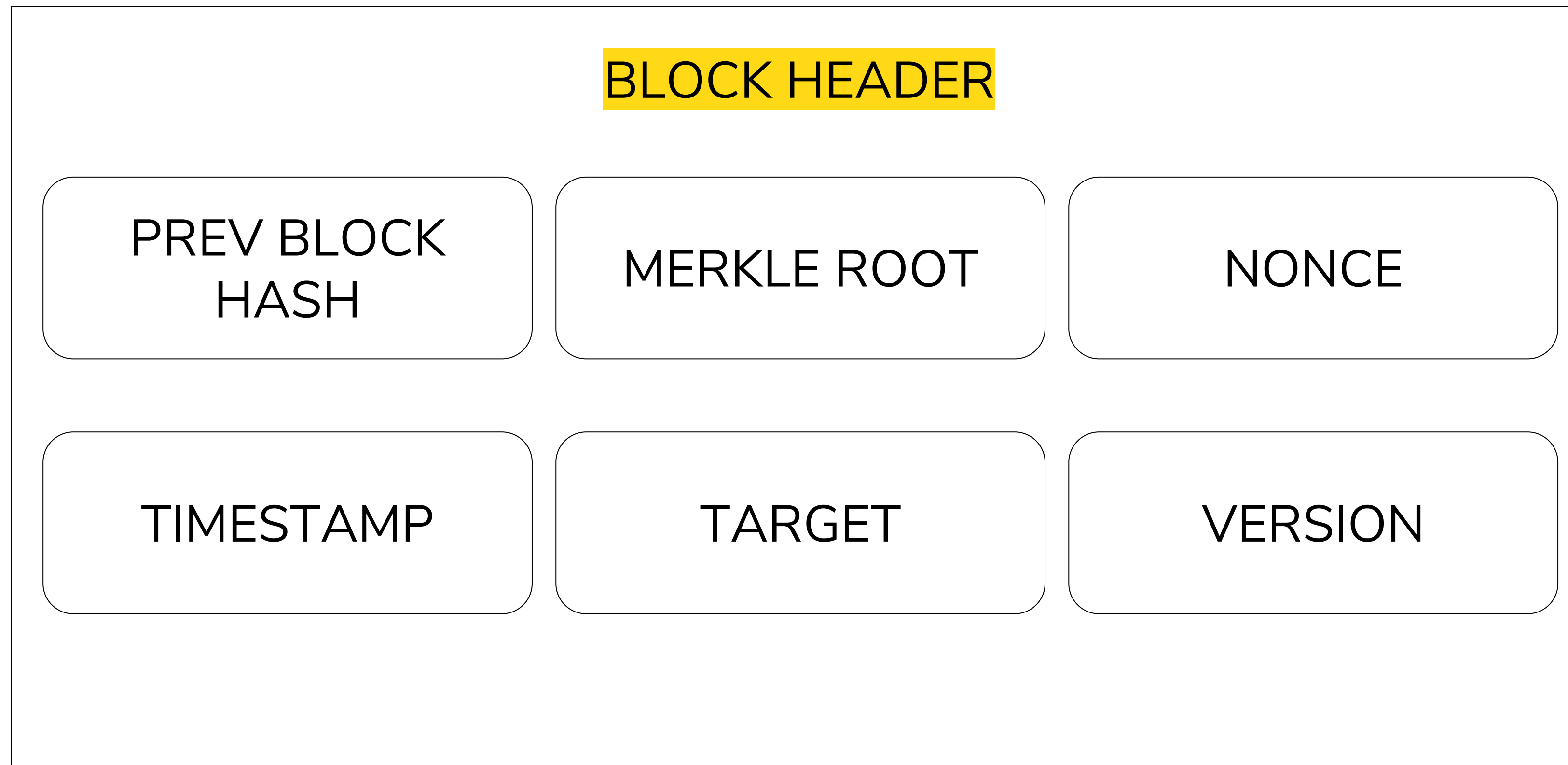
DISSECTING THE BLOCKCHAIN





A TAMPER-EVIDENT DATABASE

BLOCK HEADER

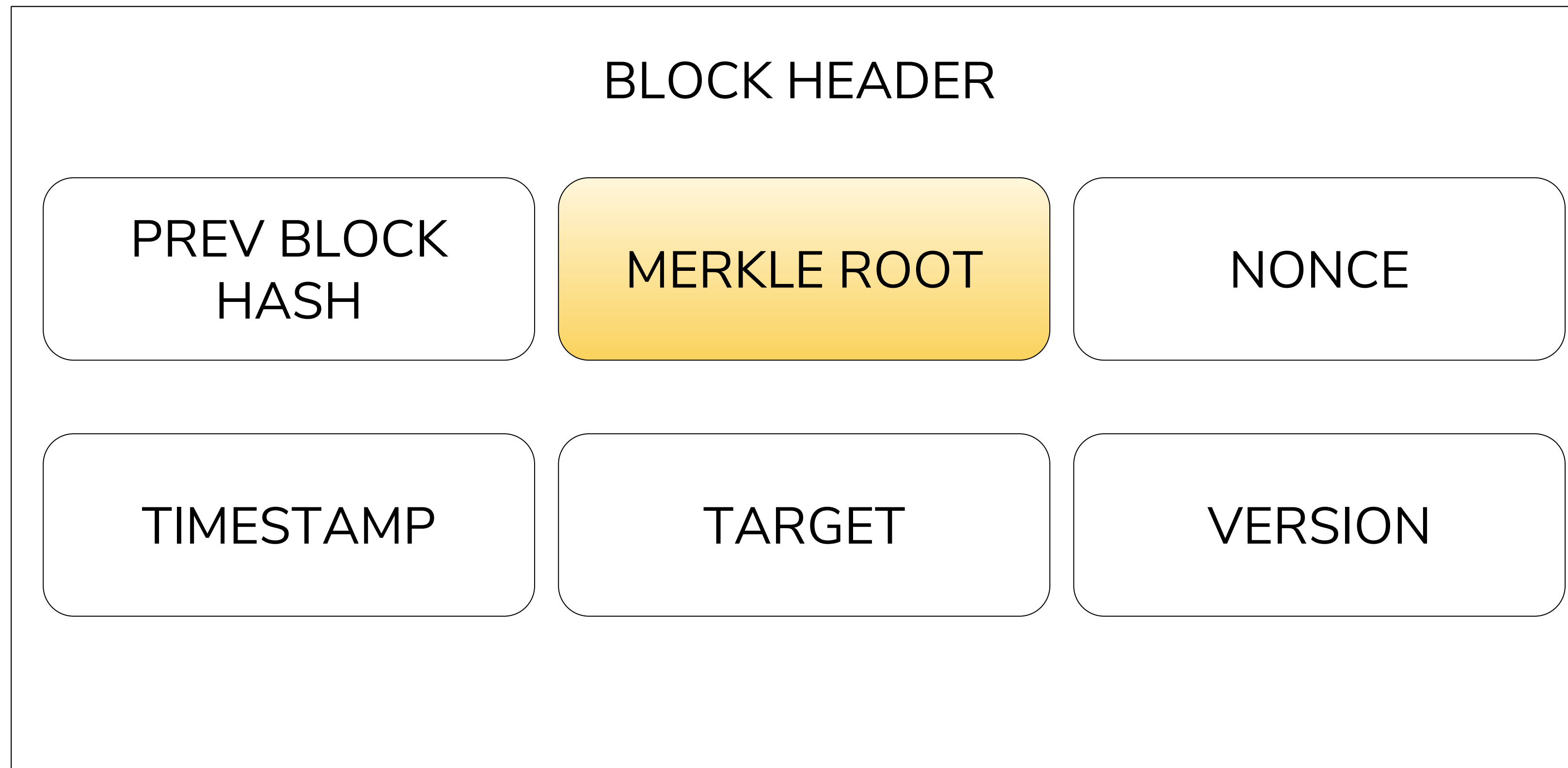


$\text{blockID} = H(\text{blockHeader}) = H(\text{prevBlockHash} || \text{merkleRoot} || \text{nonce} \dots)$



A TAMPER-EVIDENT DATABASE

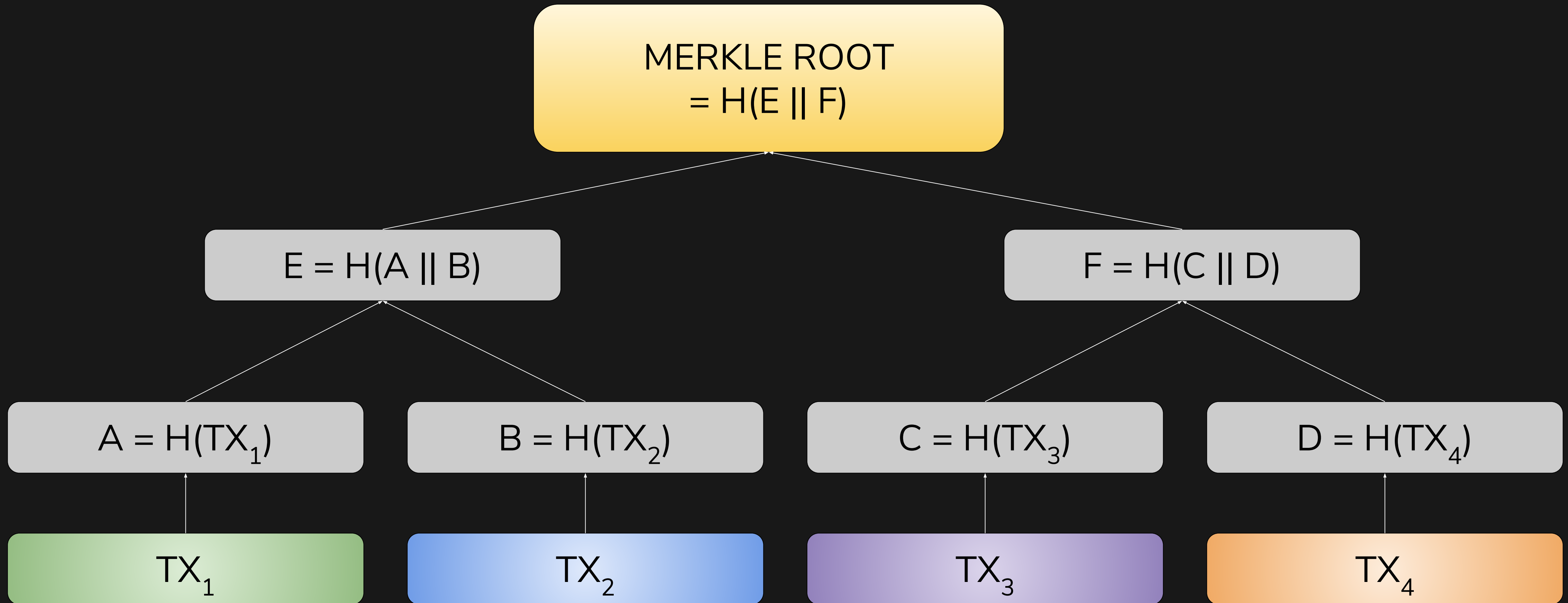
MERKLE ROOT





A TAMPER-EVIDENT DATABASE

MERKLE TREE





A TAMPER-EVIDENT DATABASE

GETTING TO THE ROOT OF THE PROBLEM

MERKLE ROOT (TAMPERED)
= $H(E \parallel F')$

$E = H(A \parallel B)$

$F' = H(C' \parallel D)$

$A = H(TX_1)$

$B = H(TX_2)$

$C' = H(TX_3')$

$D = H(TX_4)$

TX_1

TX_2

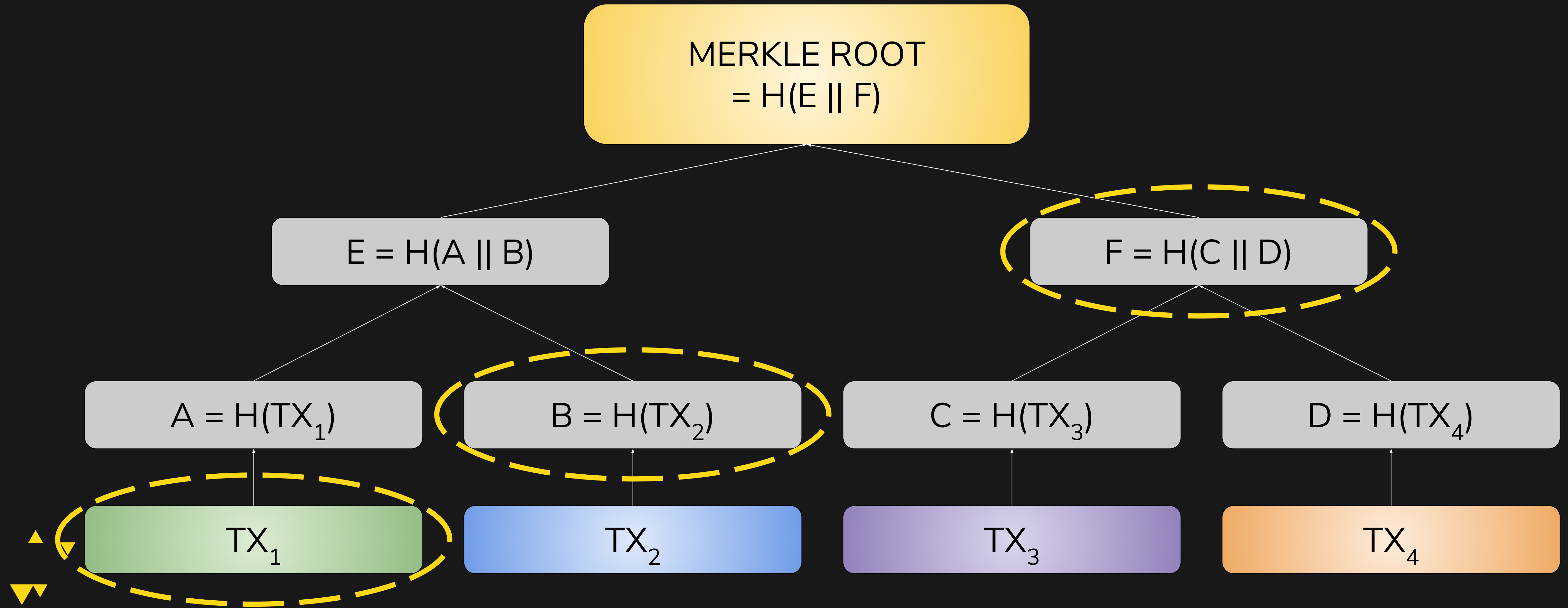
TX_3' (TAMPERED)

TX_4



A TAMPER-EVIDENT DATABASE

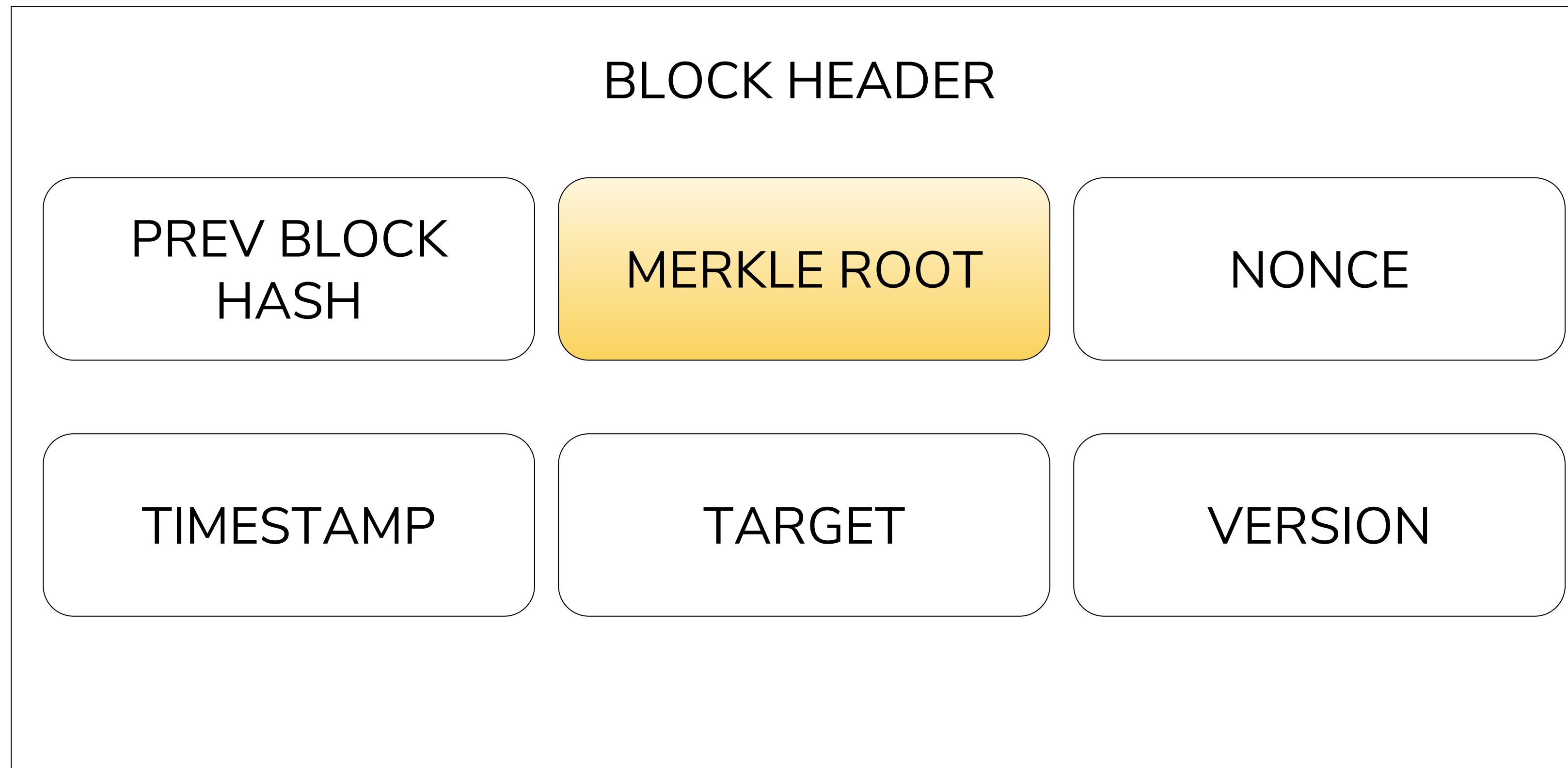
MERKLE BRANCH & PROOF OF INCLUSION





A TAMPER-EVIDENT DATABASE

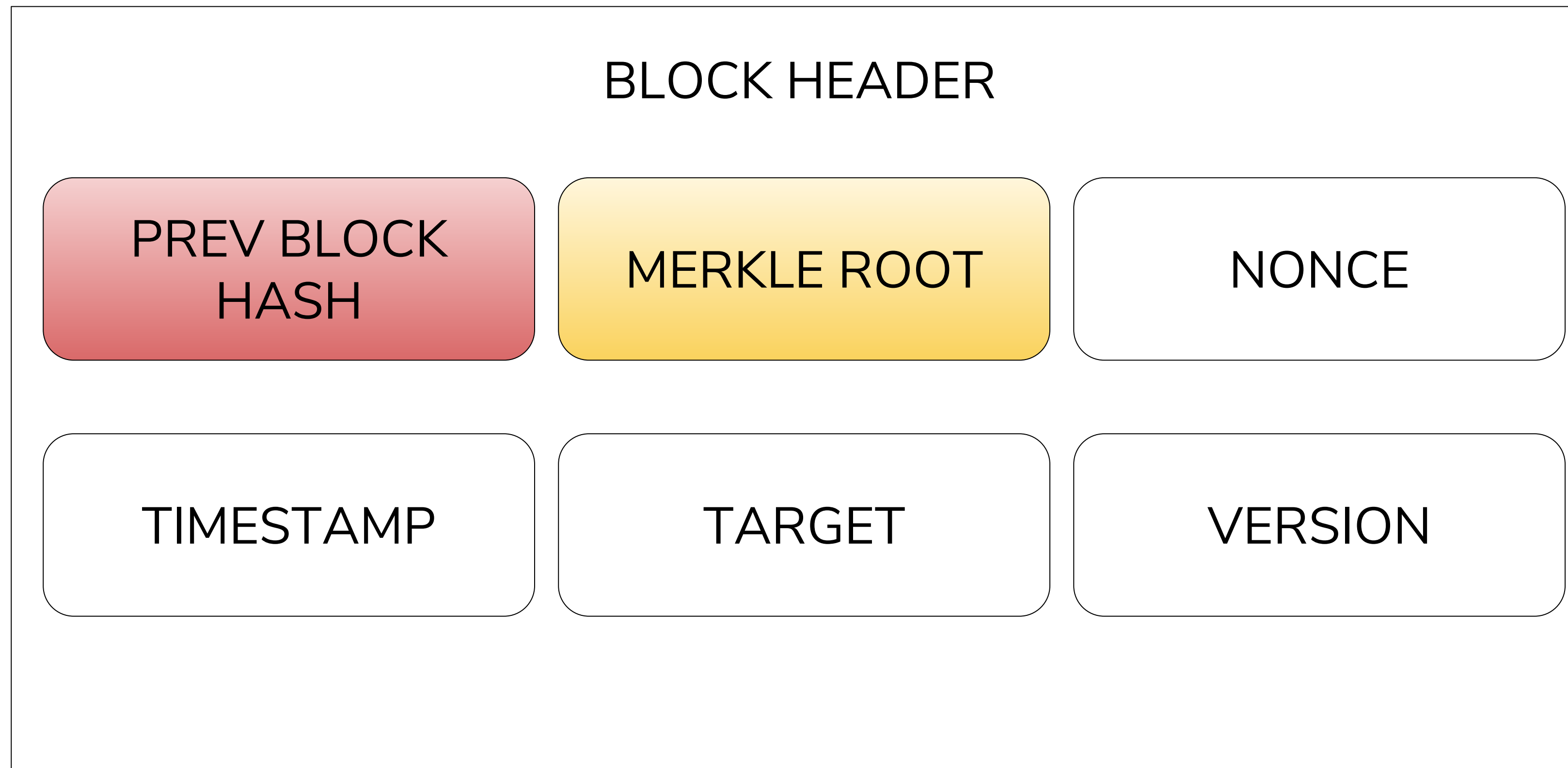
MERKLE ROOT





A TAMPER-EVIDENT DATABASE

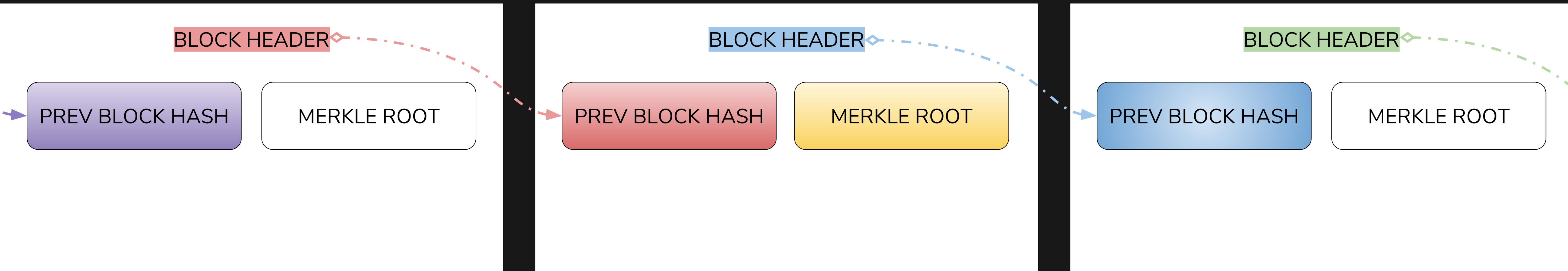
PREV BLOCK HASH



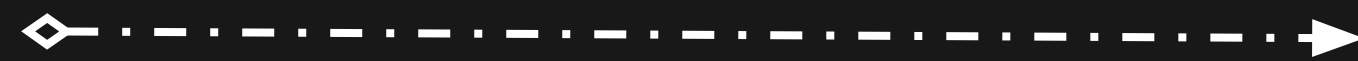


A TAMPER-EVIDENT DATABASE

PROTECTING THE CHAIN



SHA256(SHA256(x))

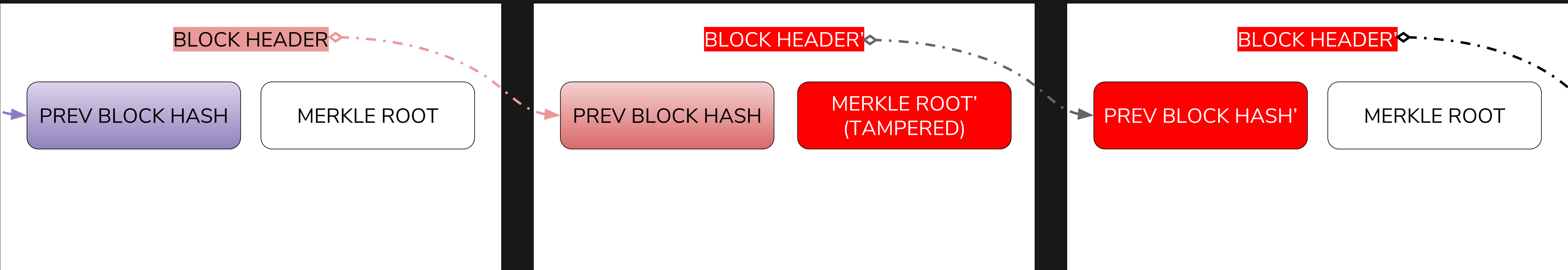


$$\text{prevBlockHash} = H(\text{prevBlockHeader})$$

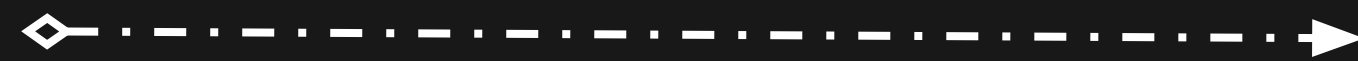


A TAMPER-EVIDENT DATABASE

PROTECTING THE CHAIN



SHA256(SHA256(x))

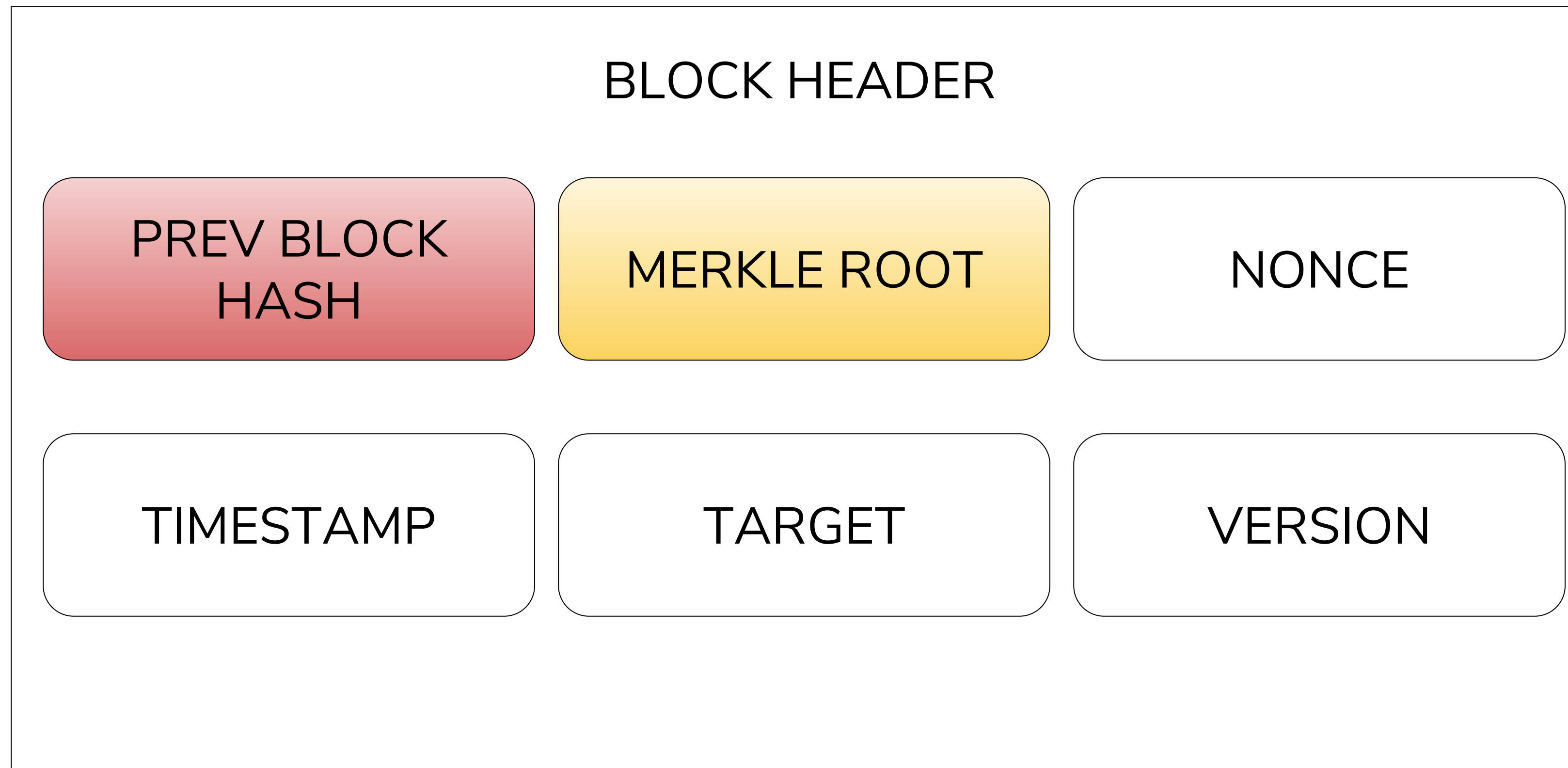


$$\text{prevBlockHash} = H(\text{prevBlockHeader})$$



A TAMPER-EVIDENT DATABASE

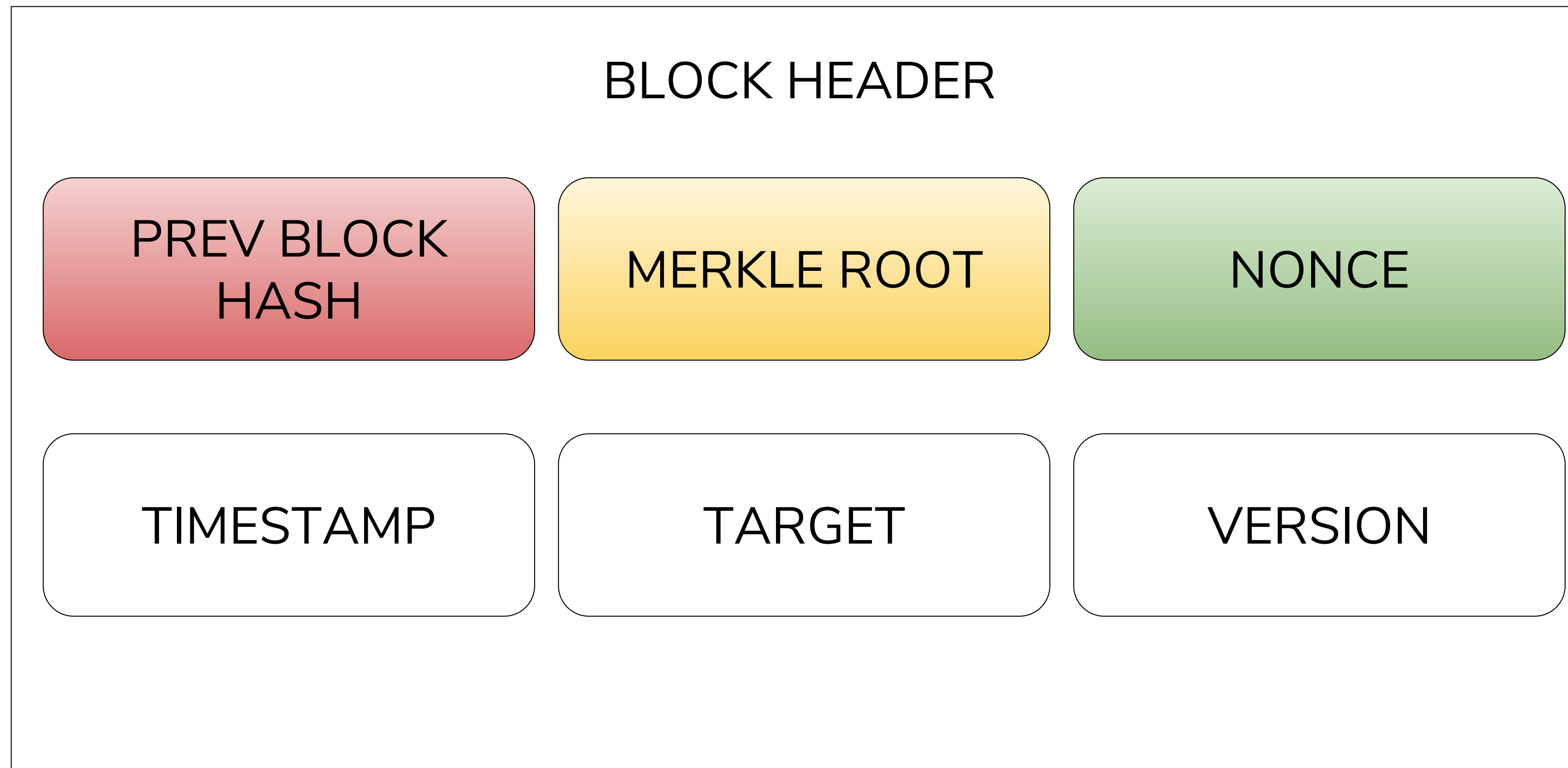
PREV BLOCK HASH





A TAMPER-EVIDENT DATABASE

NONCE





A TAMPER-EVIDENT DATABASE

PARTIAL PREIMAGE HASH PUZZLE

Bitcoin's partial preimage hash puzzle: A problem with a requirement to find a nonce that satisfies the following inequality:

$$H(\text{blockHeader}) < \text{target}$$

- Used to implement Proof-of-Work in Bitcoin (and every other PoW cryptocurrency)

Hash puzzles need to be:

1. Computationally difficult.
2. Parameterizable.
3. Easily verifiable.

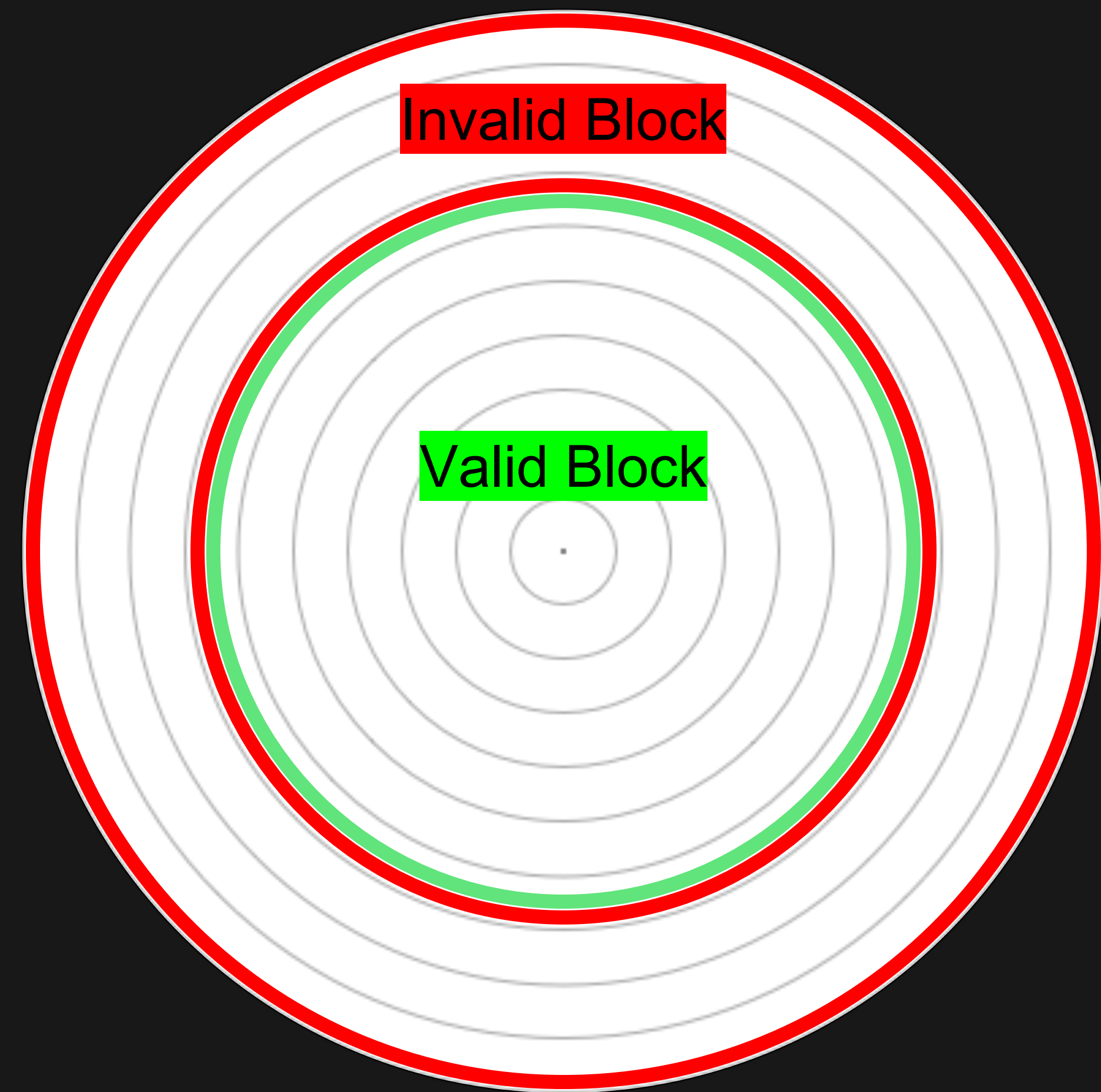




A TAMPER-EVIDENT DATABASE

MINING

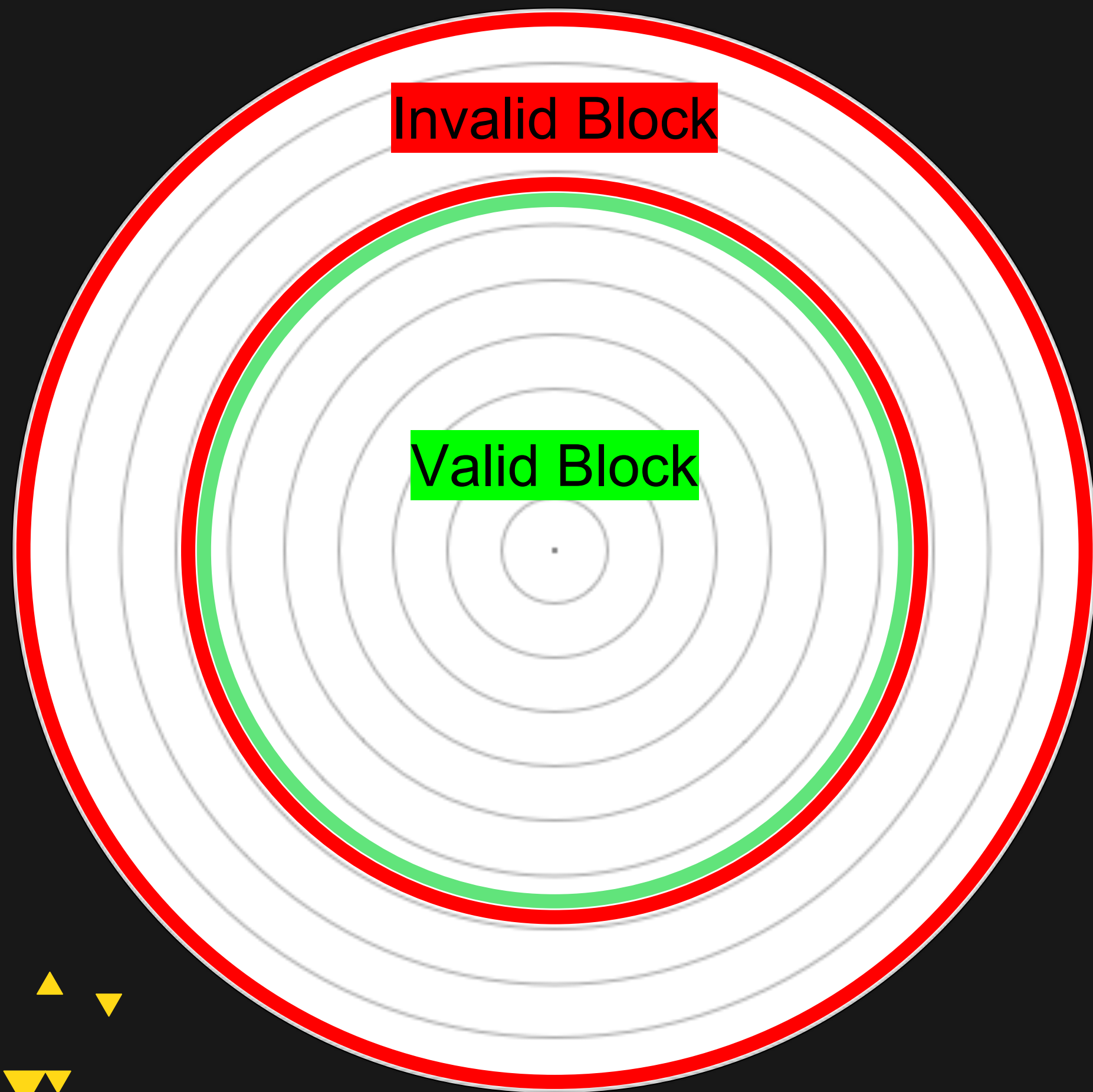
- **Mining** is like throwing darts at a target while blindfolded:
 - Equal likelihood of hitting any part of the target
 - Faster throwers \Rightarrow more hits / second
- Miners look for a hash below an algorithmically decided target





A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY



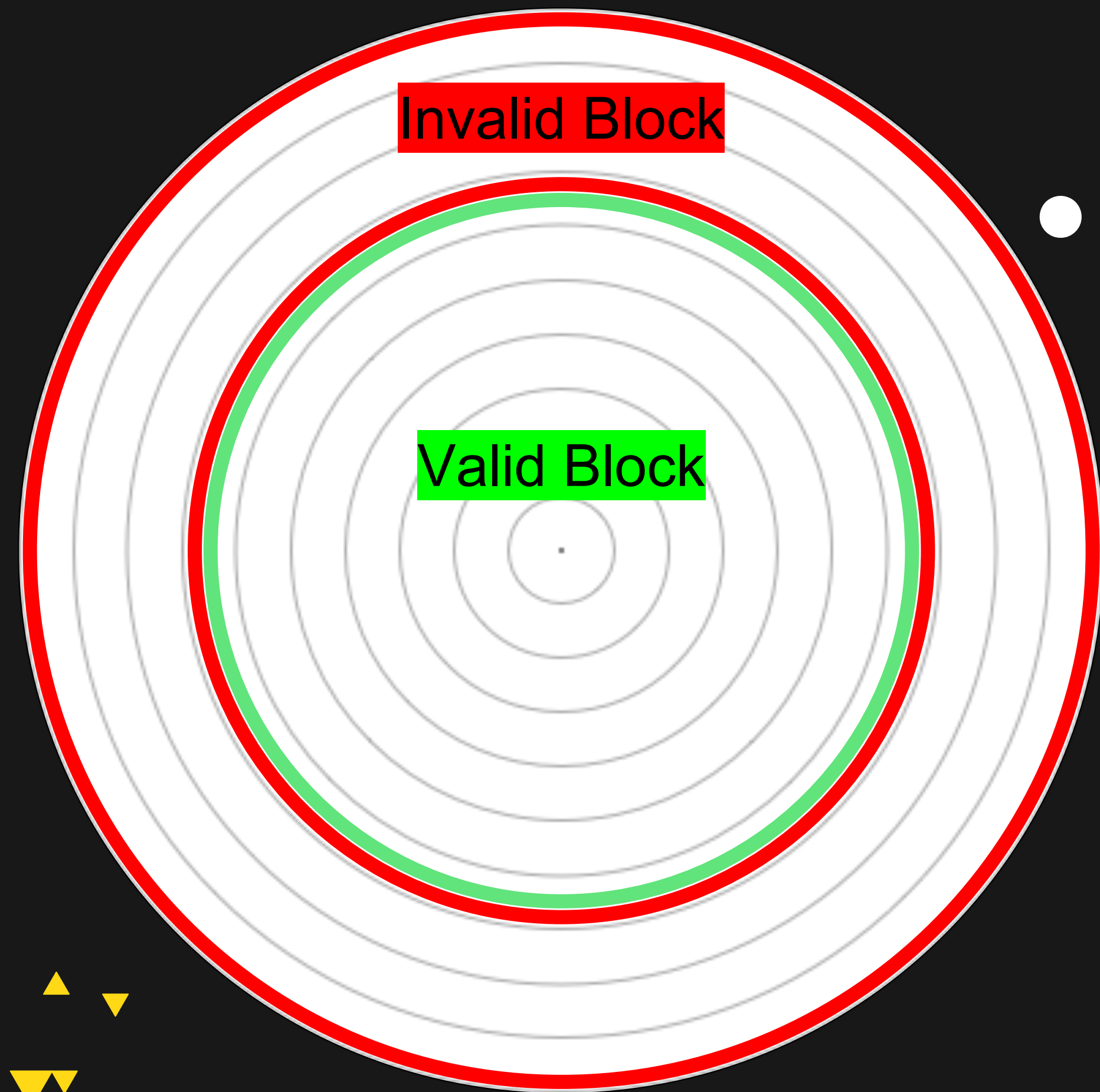
Difficulty: A representation of the expected number of computations required to find a block

- Implemented as requirement of leading number of 0s
- Adjusts with global hashrate
- $\text{difficulty} *= \text{two_weeks} / \text{time_to_mine_prev_2016_blocks}$
 - Technically every 2015 blocks



A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY



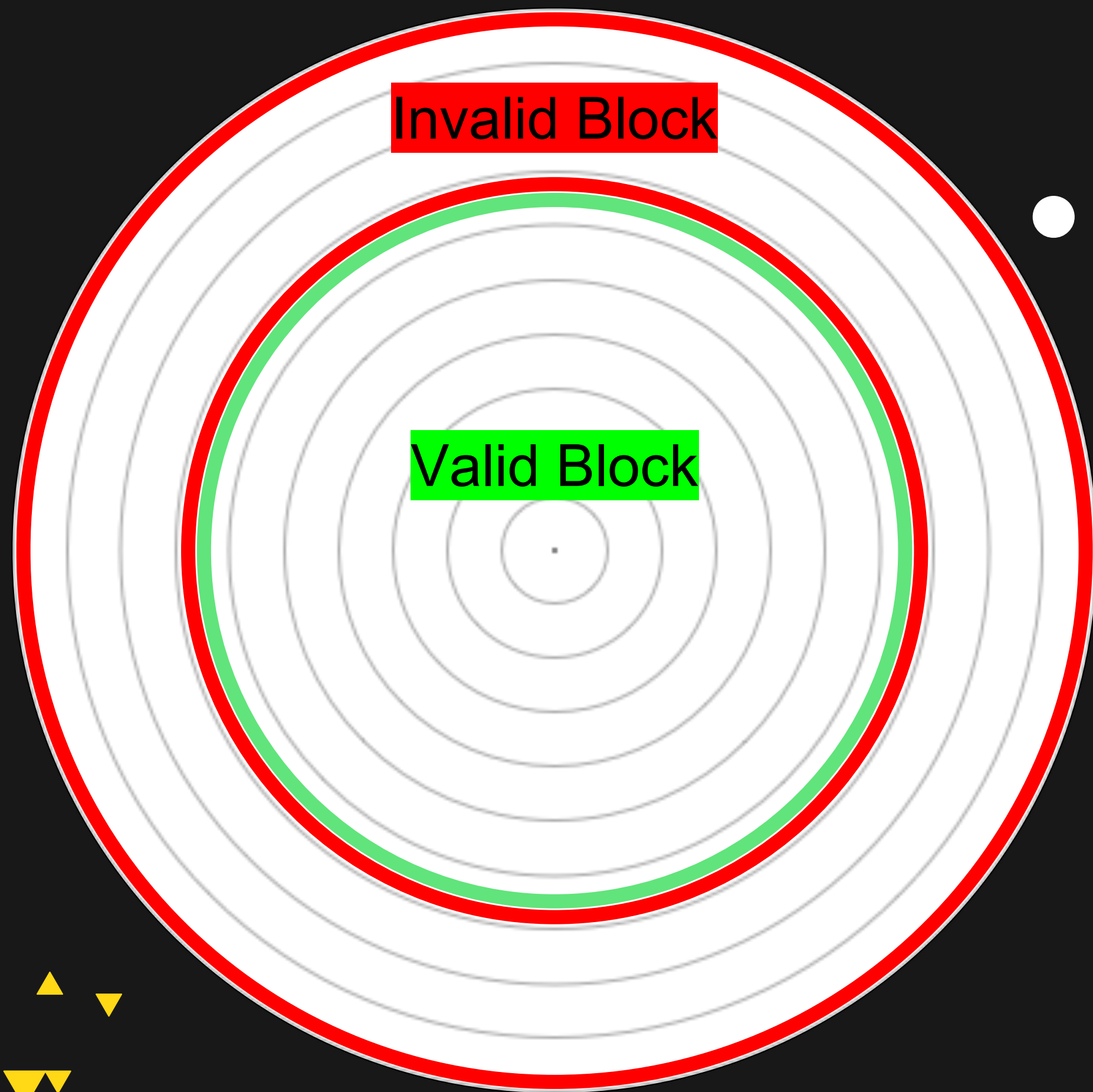
- **Sanity check** (difficulty = 10):
 - What is the new difficulty when `two_weeks =`
`time_to_mine_prev_2016_blocks?`





A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY



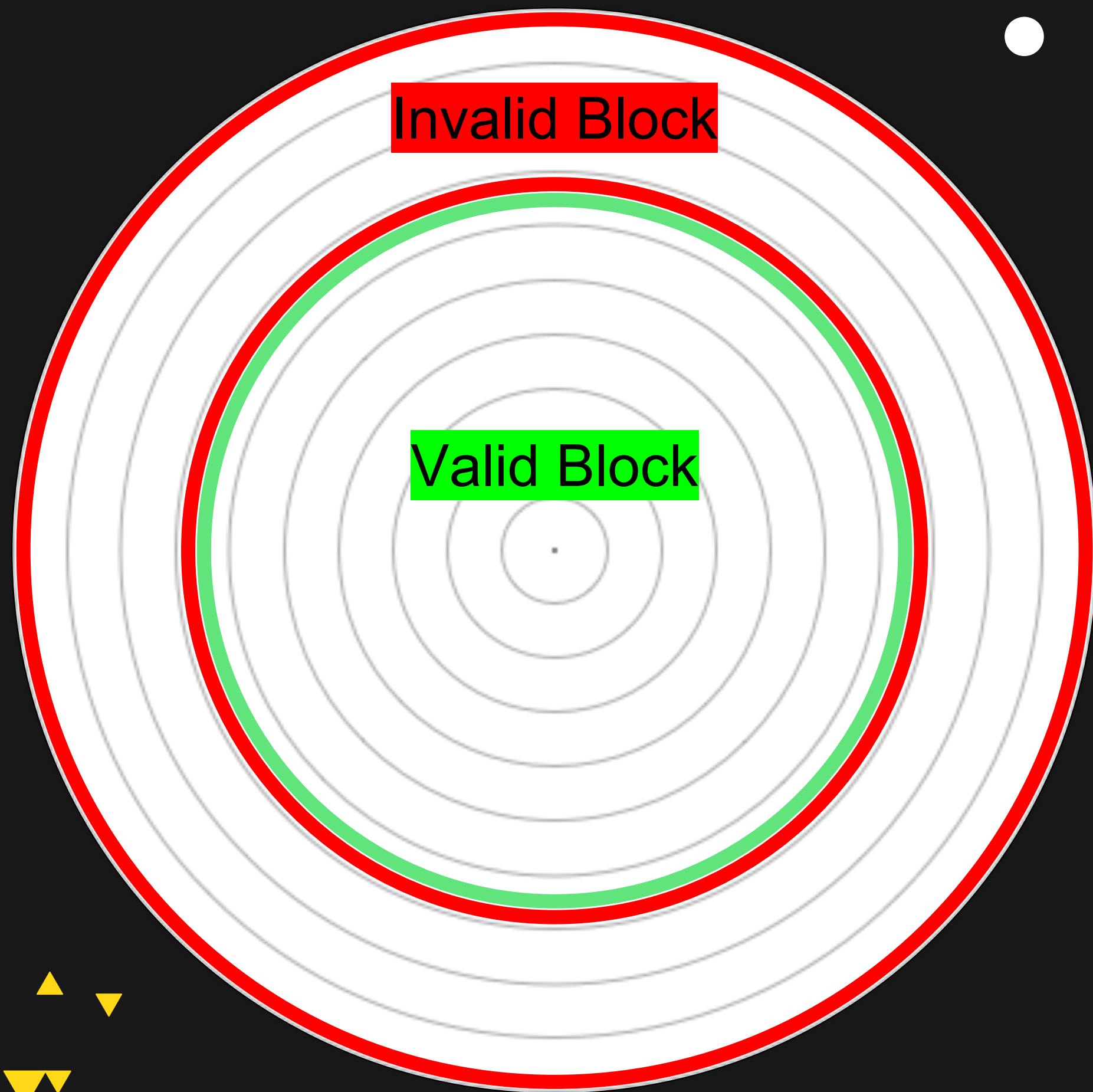
- **Sanity check** (difficulty = 10):
 - What is the new difficulty when `two_weeks = time_to_mine_prev_2016_blocks`?
(Answer: 10. Difficulty stays the same!)





A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY



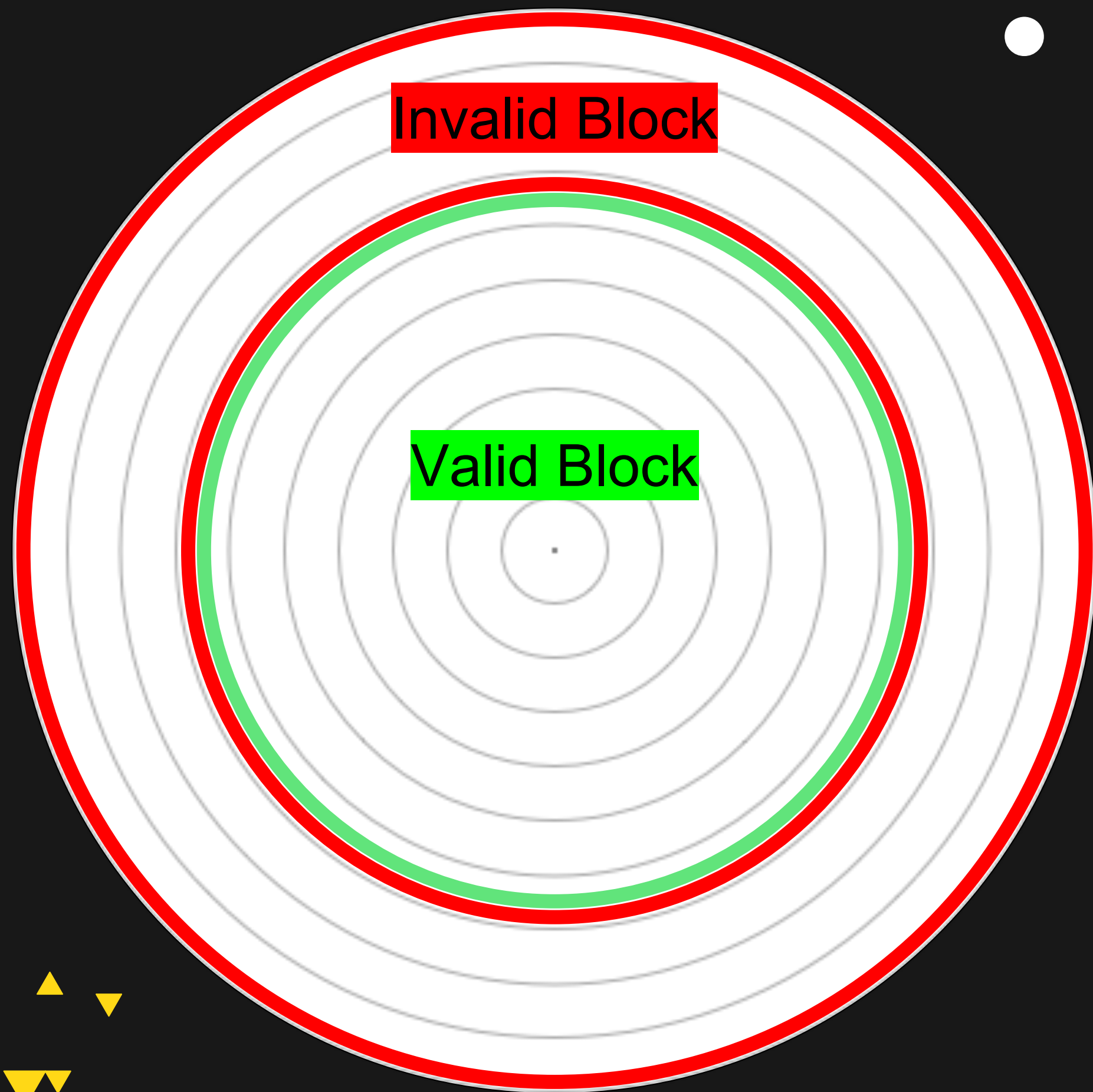
- **Sanity check** (difficulty = 10):
 - What does difficulty equal when:
 - `time_to_mine = one_week`?
 - `time_to_mine = four_weeks`?





A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY



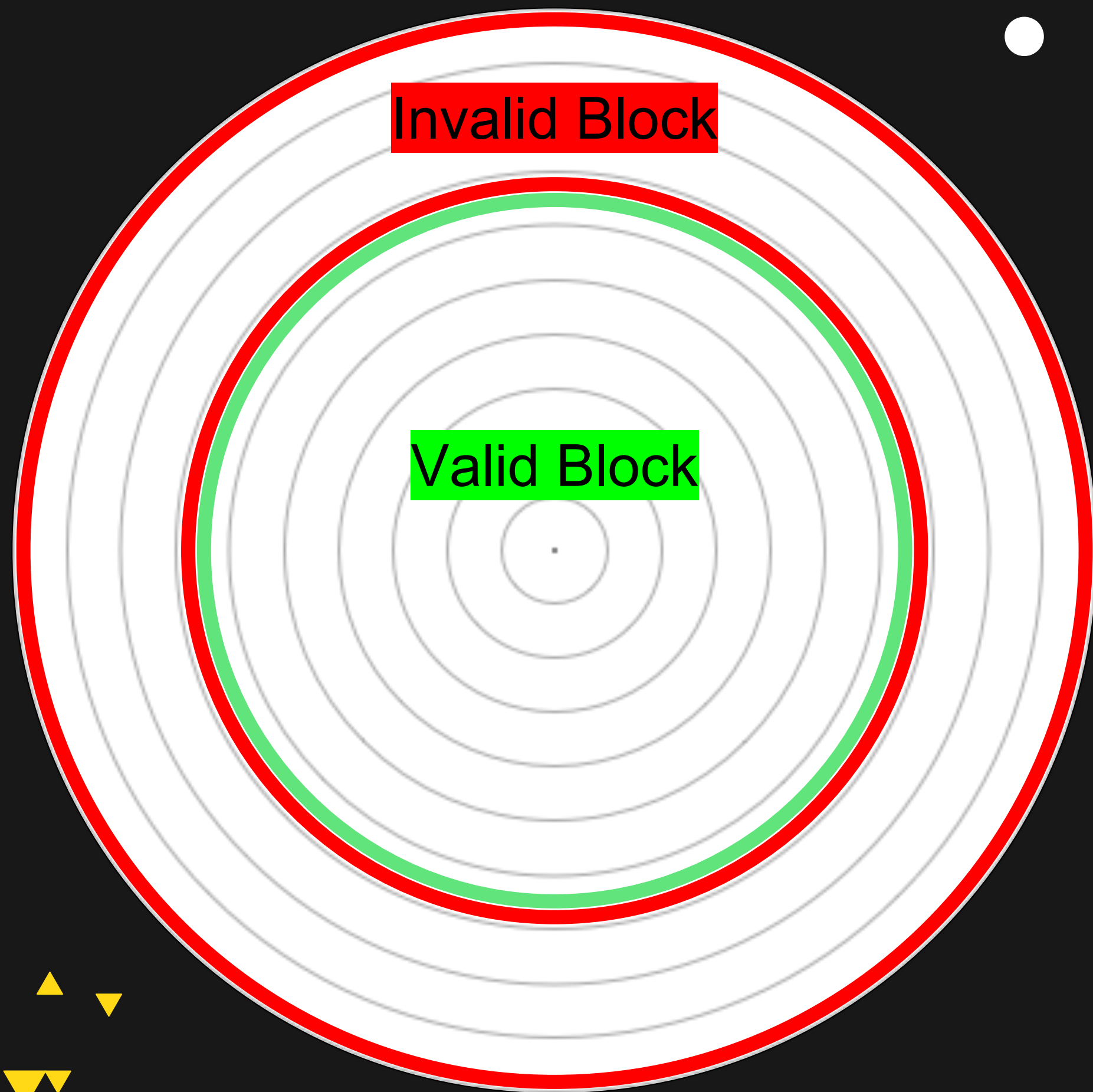
- **Sanity check** (difficulty = 10):
 - What does difficulty equal when:
 - `time_to_mine = one_week`?
(Answer: 20)
 - `time_to_mine = four_weeks`?





A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY



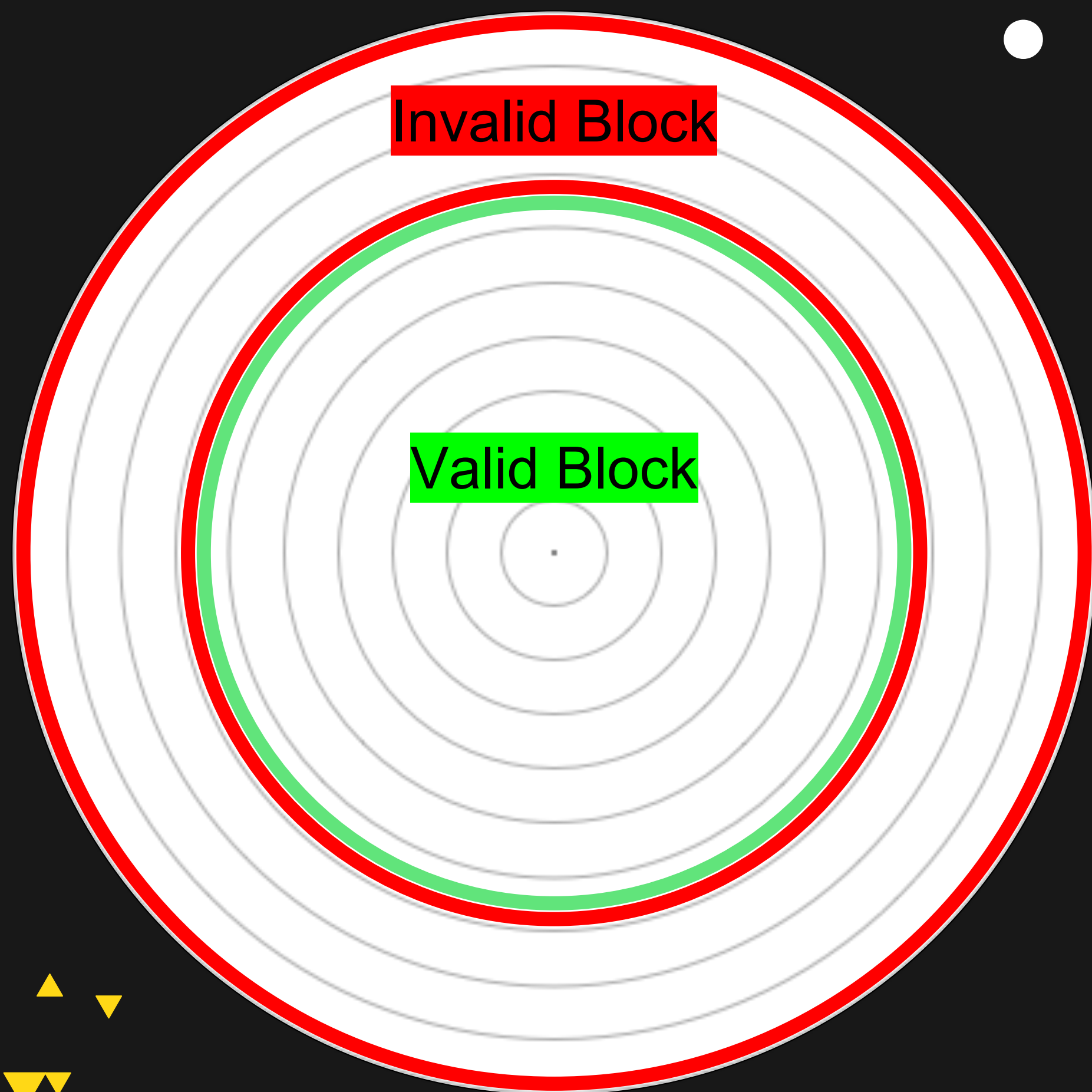
- **Sanity check** (difficulty = 10):
 - What does difficulty equal when:
 - `time_to_mine = one_week`?
(Answer: 20)
 - `time_to_mine = four_weeks`?
(Answer: 5)





A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY



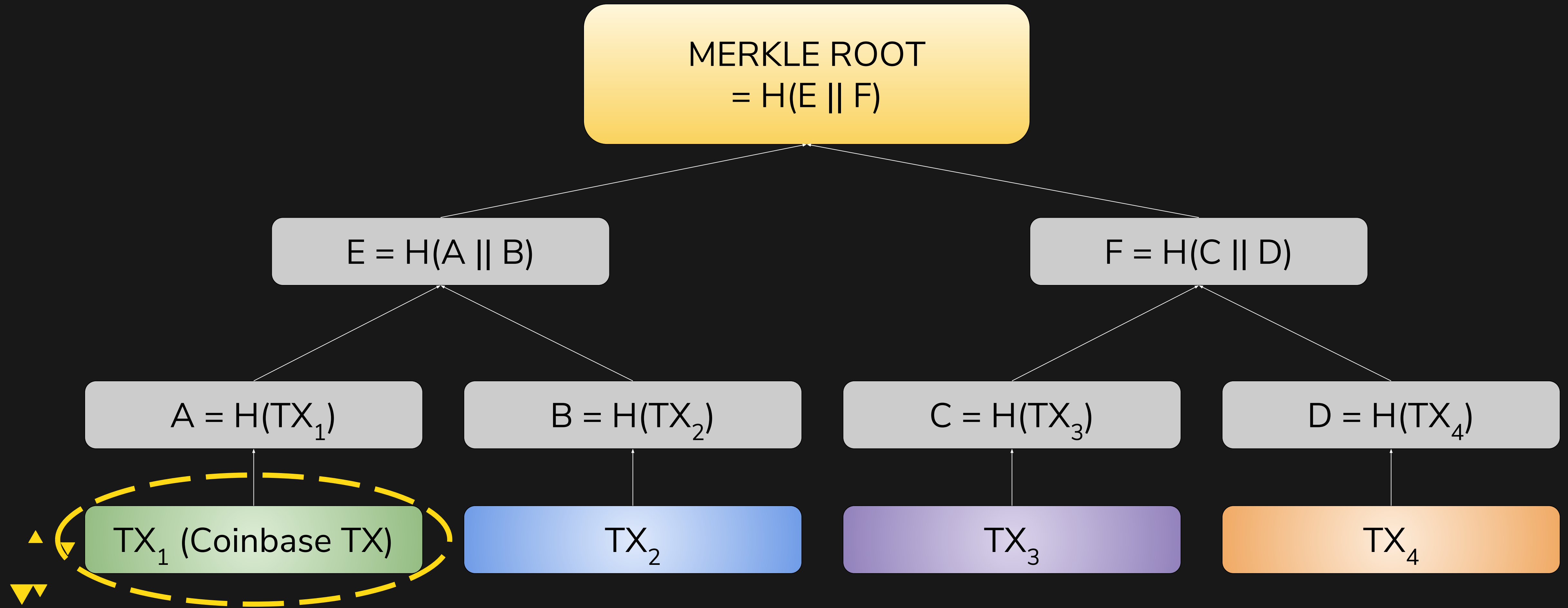
- **Sanity check** (difficulty = 10):
 - What does difficulty equal when:
 - `time_to_mine = one_week`?
(Answer: 20)
 - `time_to_mine = four_weeks`?
(Answer: 5)
 - Difficulty is inversely proportional to `time_to_mine`.)





A TAMPER-EVIDENT DATABASE

COINBASE TRANSACTION





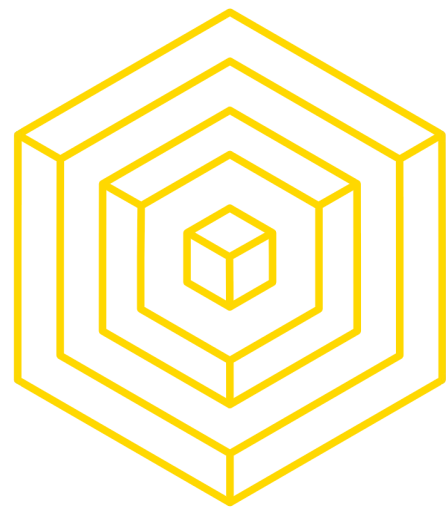
A TAMPER-EVIDENT DATABASE

MINING PSEUDOCODE

```
TARGET = (65535 << 208) / DIFFICULTY;
coinbase_nonce = 0;
while (1) {
    header = makeBlockHeader(transactions, coinbase_nonce);
    for (header_nonce = 0; header_nonce < (1 << 32); header_nonce++){
        if (SHA256(SHA256(makeBlock(header, header_nonce))) <
TARGET)
            break; //block found!
    }
    coinbase_nonce++;
}
```

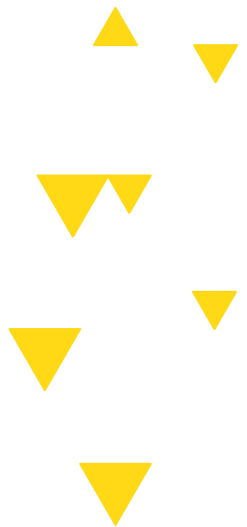
Figure 5.6 : CPU mining pseudocode.

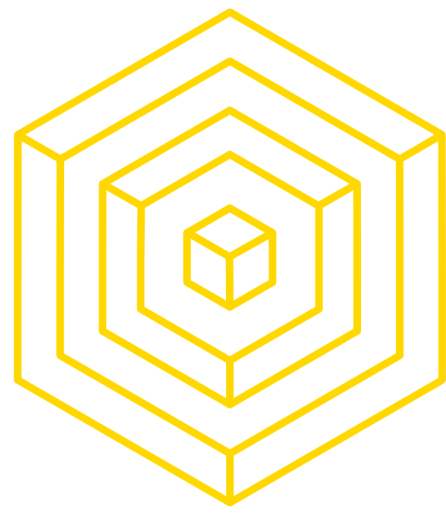
Source: (from Princeton Textbook, 5.2)



3

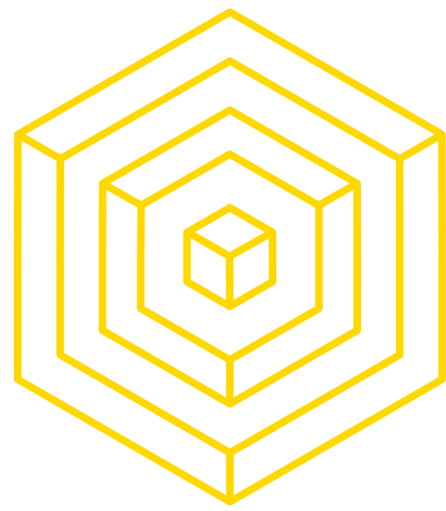
SIGS, ECDSA, AND ADDRESSES





3.1

SIGNATURES AND AUTHENTICATION



DIGITAL SIGNATURE SCHEMES (DSS)

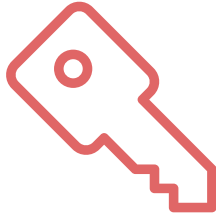
EXAMPLE



ALICE



BOB

private key: 

public key: 

message: 

Alice uses ECDSA to generate
private and public keys



DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE



ALICE



BOB

private key: 

public key: 

message: 

Bob needs Alice's public key

Alice's public key: 





DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE




ALICE



BOB


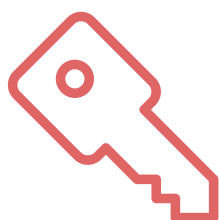

private key: 

public key: 

message: 

Alice signs her message

Alice's public key: 

signature:  =  + 



DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE





ALICE



BOB

Alice sends message + signature

private key: 

public key: 

message: 

signature: 

Alice's public key: 

Alice's message: 

Alice's signature: 





DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE



ALICE



BOB

private key:

public key:

message:

signature:

Bob can easily verify if Alice signed



+ = or



DIGITAL SIGNATURE SCHEMES (DSS)

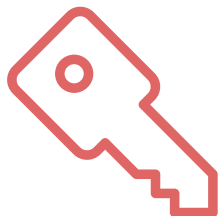
EXAMPLE



ALICE



BOB

private key: 





public key: 

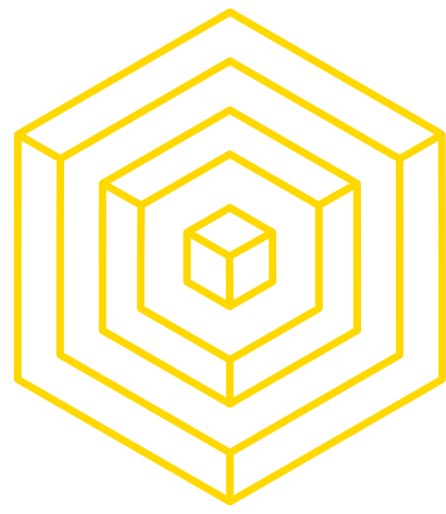
message: 

signature: 

Bob cannot easily guess Alice's private key



 +  =  or 



DIGITAL SIGNATURE SCHEMES (DSS)

DSS SECURITY DEFINITION

Recipients given the (message, signature) pair should be able to verify:

- **Message Origin:** original sender (owner of private key) has authorized this message/transaction
- **Non-repudiation:** original sender (owner of private key) cannot backtrack
- **Message Integrity:** message cannot have been modified since sending



3.2

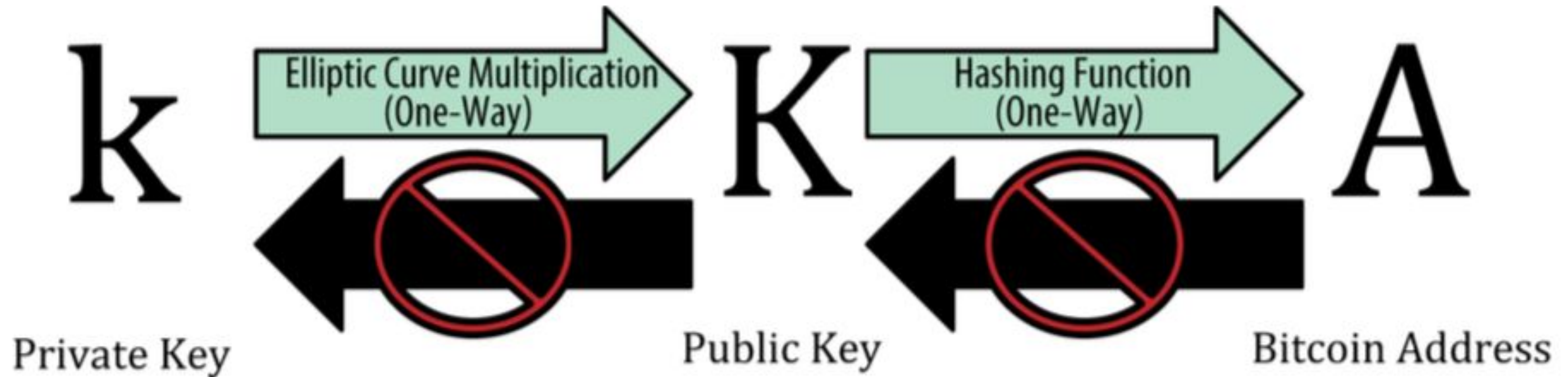
FROM PRIVATE KEYS TO ADDRESSES





KEYS AND ADDRESSES

BEFORE THE TECHNICAL STUFF



SOURCE: Mastering Bitcoin

AUTHOR: MENGYI WANG



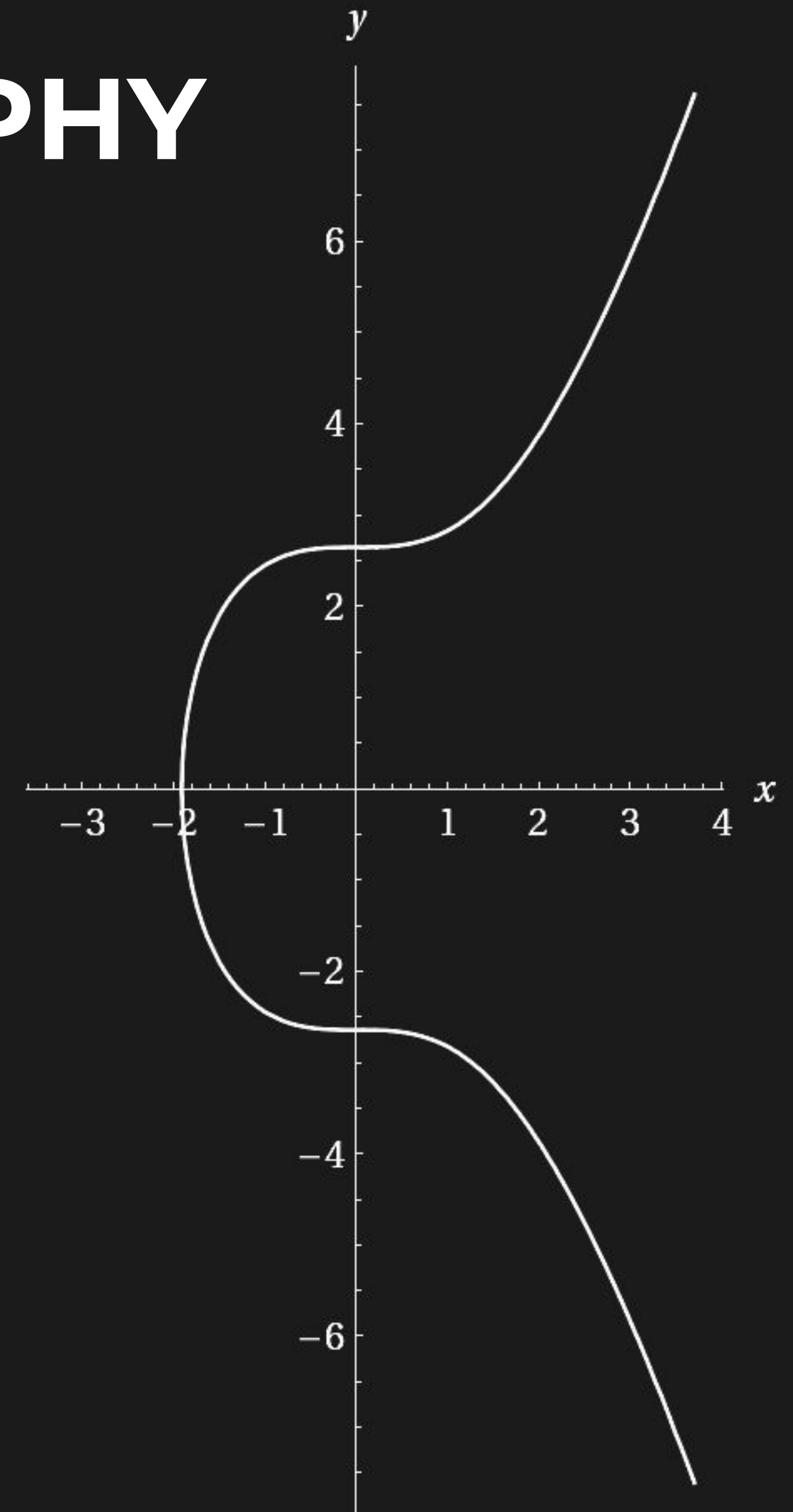
ELLIPTIC CURVE CRYPTOGRAPHY

PART 1: ELLIPTIC CURVE

- Bitcoin uses **ECDSA** (Elliptic Curve Digital Signature Algorithm) to produce private and public keys
- The **Elliptic Curve** is defined by general form

$$y^2 = x^3 + ax + b$$

- Over a **finite field** p (for key size)

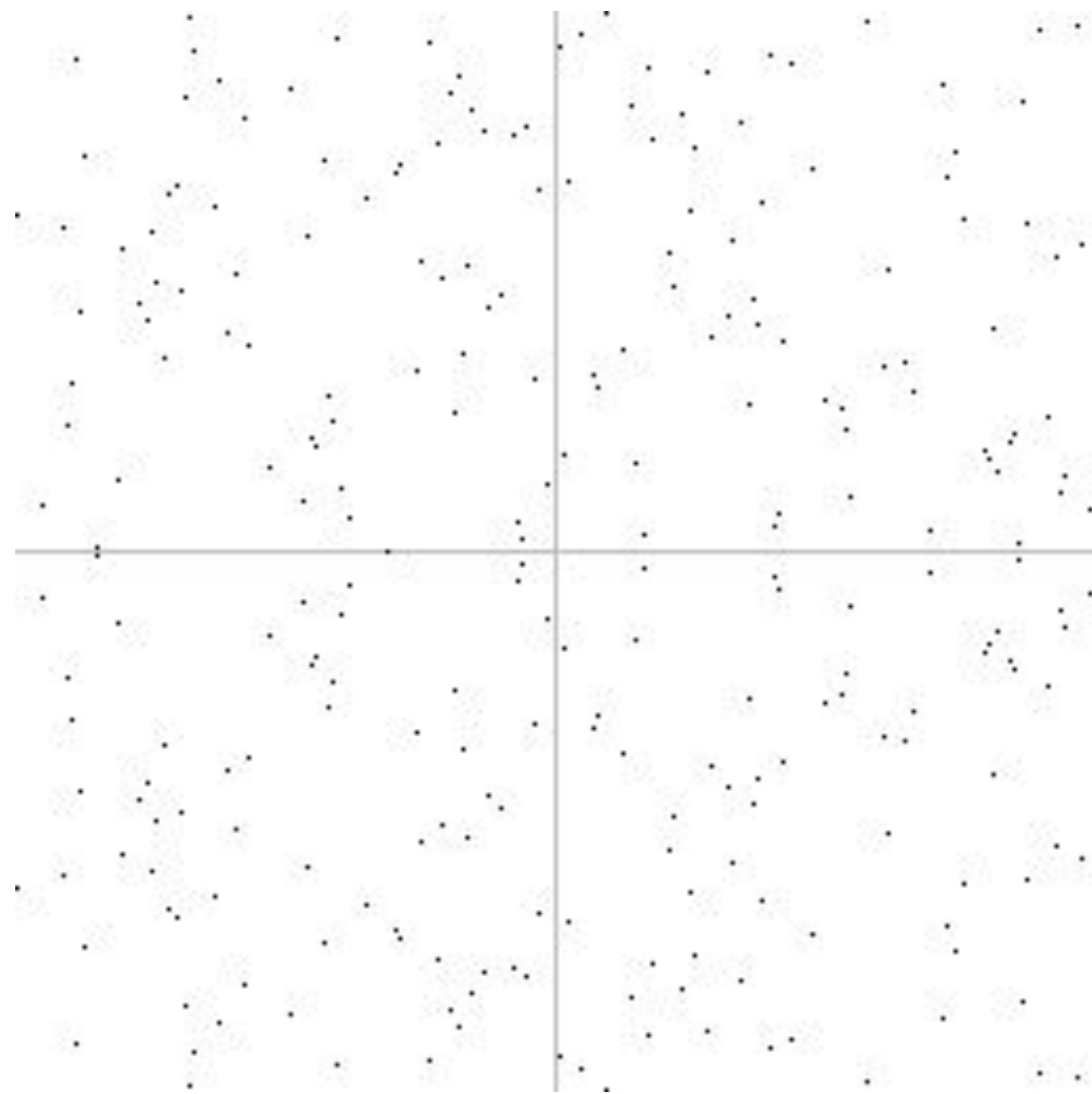


$y^2 = x^3 + 7$ | Computed by Wolfram|Alpha

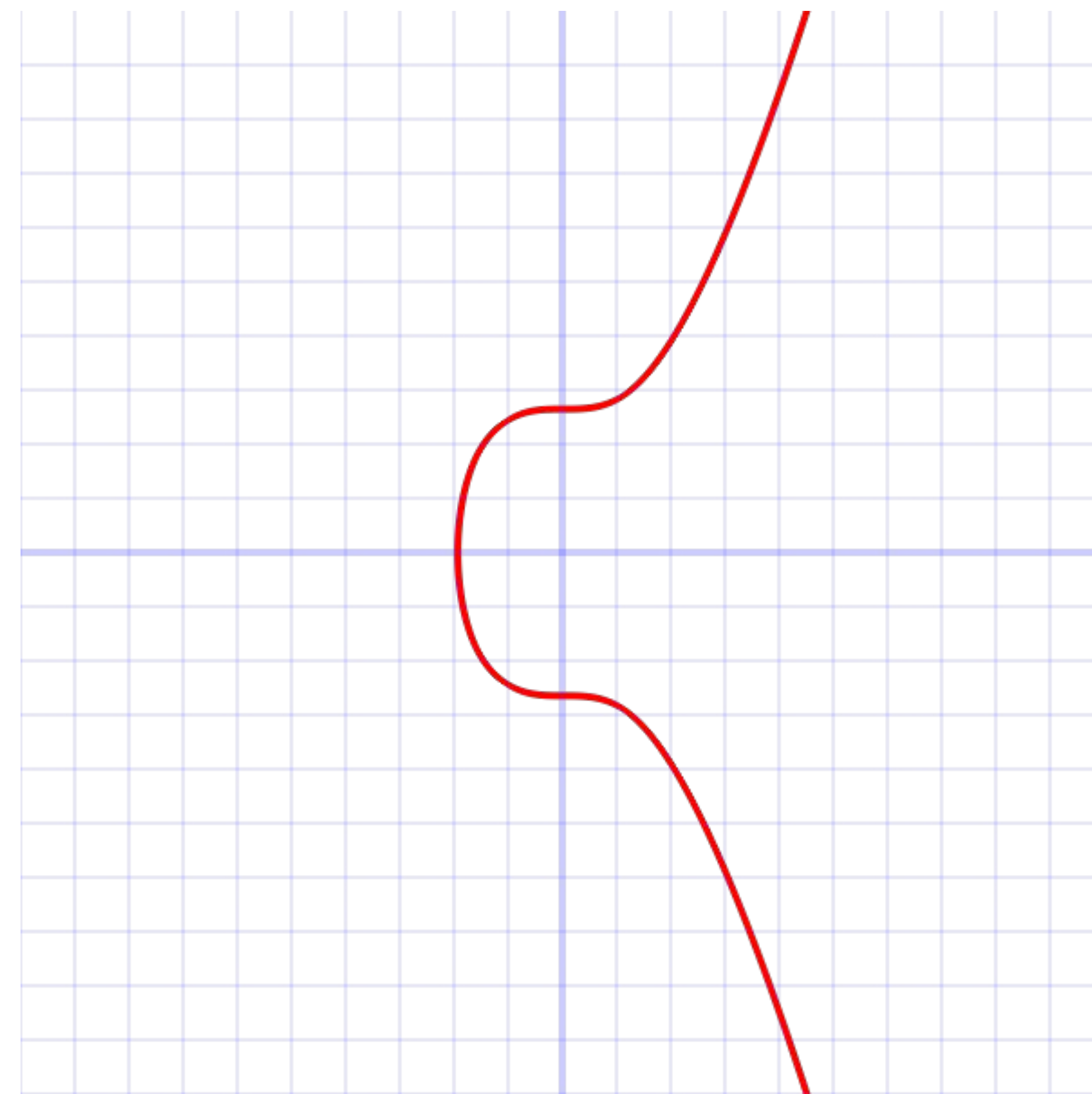


ELLIPTIC CURVE CRYPTOGRAPHY

PART 1: ELLIPTIC CURVE



\mathbb{Z}_{2^8+1}



$$\text{secp256k1} : Y^2 = (X^3 + 7)$$



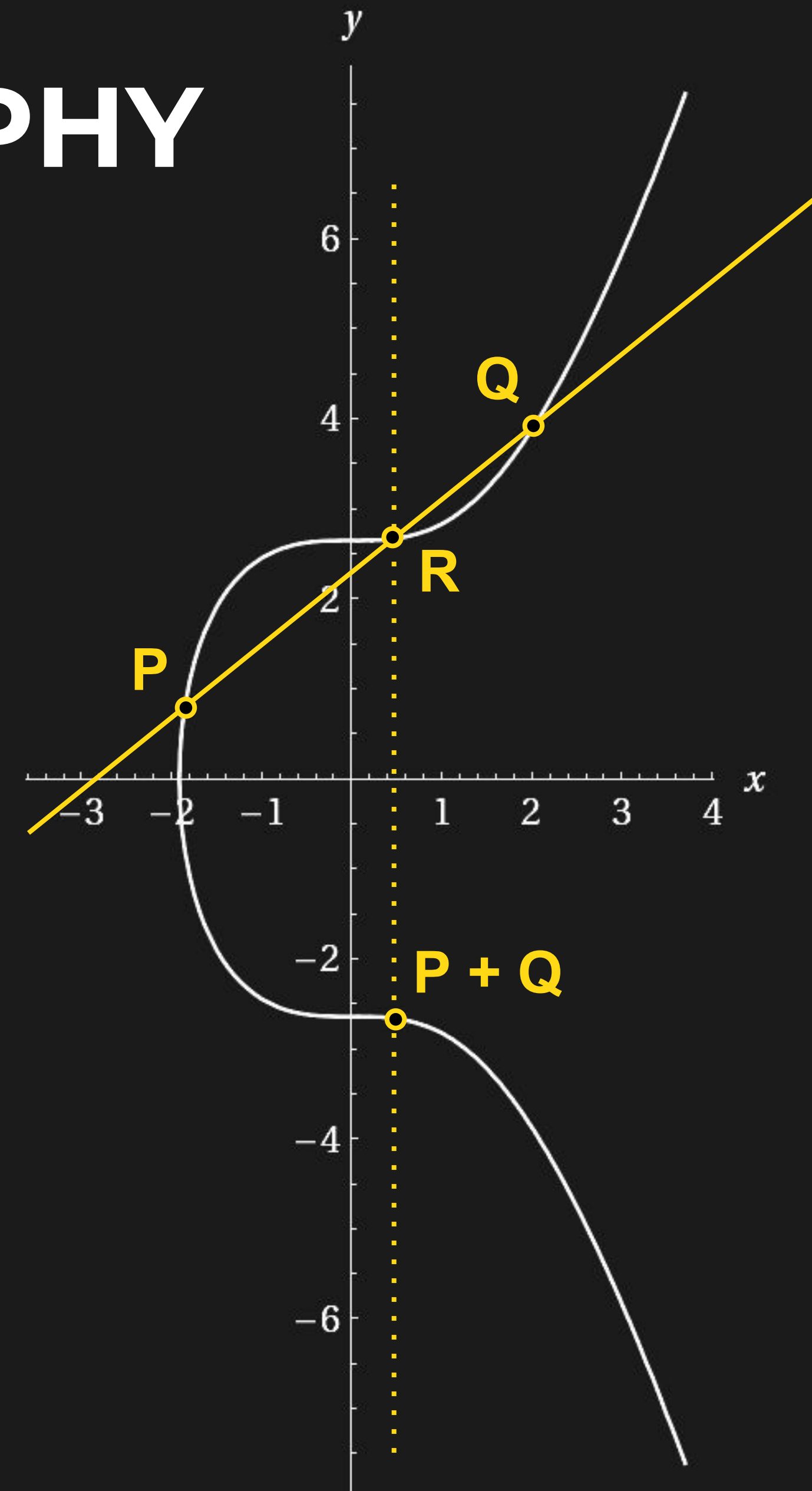


ELLIPTIC CURVE CRYPTOGRAPHY

PART 2: CHORD-TANGENT PROCESS

- Note symmetry
- Non-vertical lines intersect the curve in at most 3 points
- Elliptic curve scalar multiplication

- We can do “multiplication” repeatedly using lines and points on our elliptic curve, a **trapdoor function**
 - We define a group law on an elliptic curve using the **chord-tangent process** - “point multiplication”



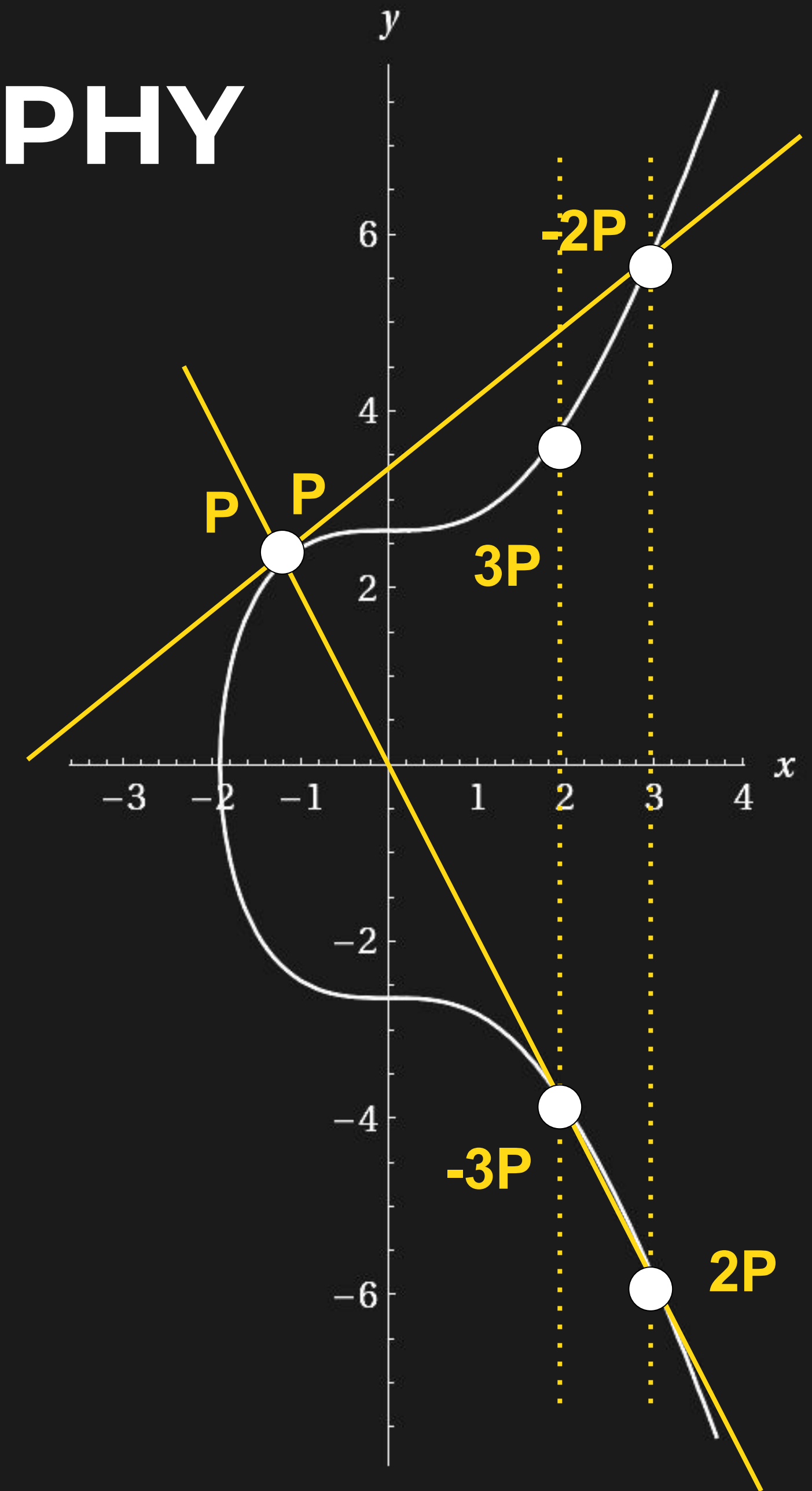
$y^2 = x^3 + 7$ | Computed by Wolfram|Alpha



ELLIPTIC CURVE CRYPTOGRAPHY

PART 2.5: POINT MULTIPLICATION DEMO

- Elliptic curve point scalar multiplication
- $Q = P + P + P + P + P + P \dots$
- Repeat n times
- $Q = nP$



$y^2 = x^3 + 7$ | Computed by Wolfram|Alpha



ELLIPTIC CURVE CRYPTOGRAPHY

PART 3: SECURITY OF ECC - TRAPDOOR FUNCTION

- ECDSA generates private and public keys in Bitcoin:

private key = n

public key = $nP = P + P + P \dots + P = K$

address = $RIPMD160(SHA256(nP))$

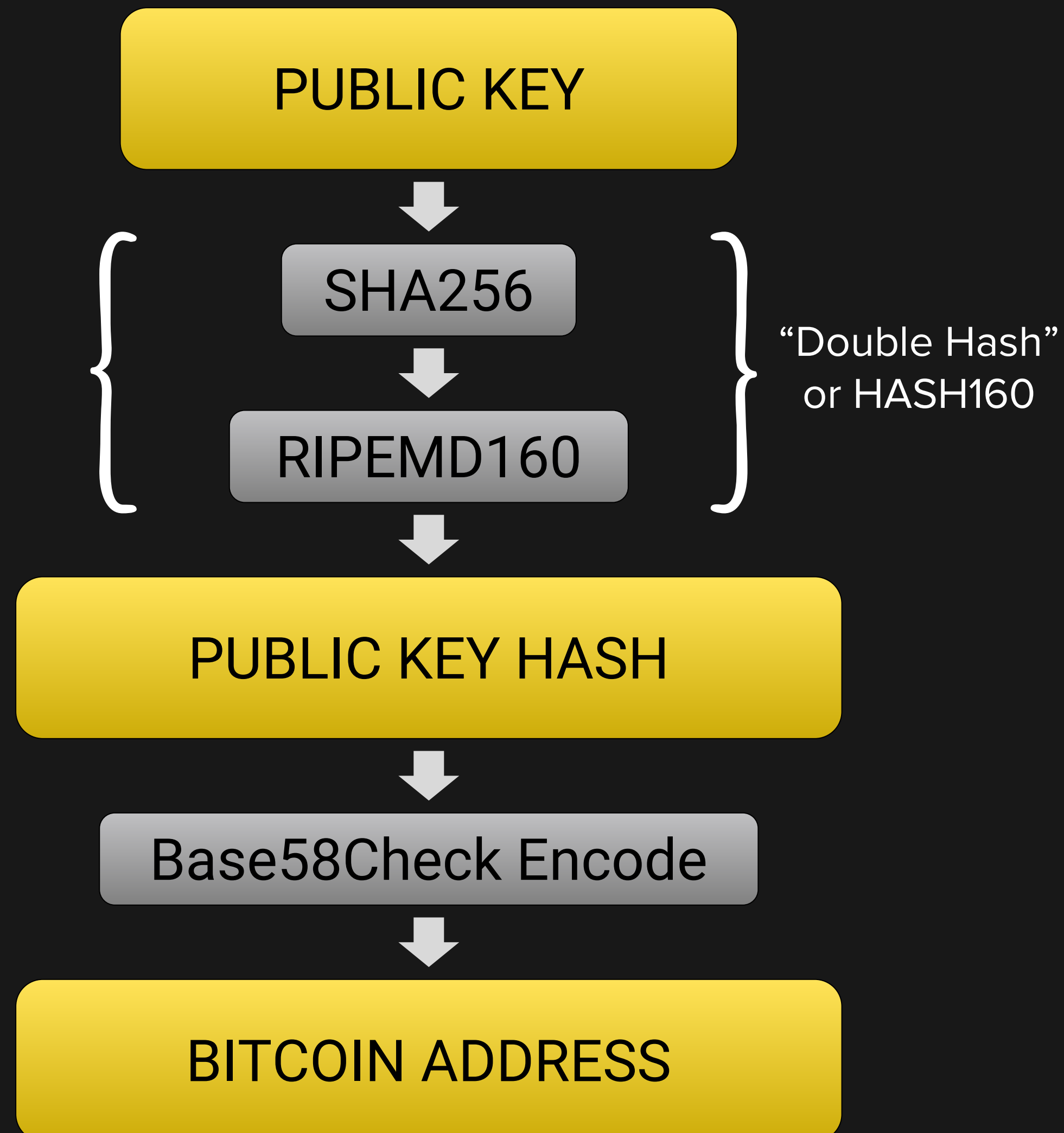
- How can we get n from nP ?
- \Rightarrow **Elliptic Curve Discrete Logarithm Problem** is **computationally infeasible** (over certain fields and curves), thus ECC is a “trapdoor function”
 - no sub-exponential time algorithm





PUBLIC KEY TO BITCOIN ADDRESS

PUBLIC KEY TO PUBKEYHASH



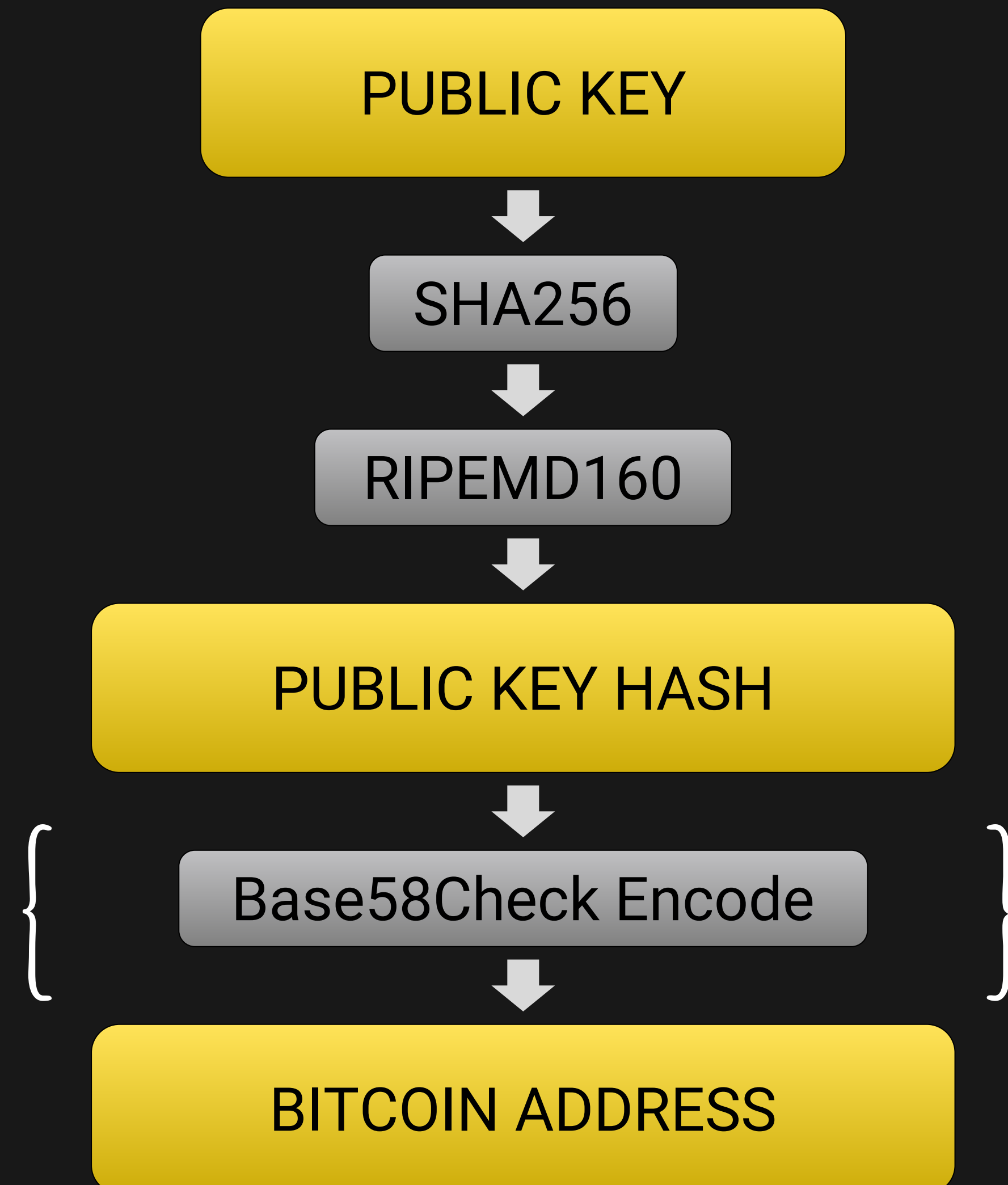
$$\text{PUBKEYHASH} = \text{RIPEMD160}(\text{SHA256}(K))$$

- **SHA-256** (Secure Hashing Algorithm)
 - Used extensively in bitcoin scripts and mining
- **RIPEMD** (RACE Integrity Primitives Evaluation Message Digest)
 - Produces 160-bit (20-byte) number



PUBLIC KEY TO BITCOIN ADDRESS

PUBLIC KEY HASH TO ADDRESS

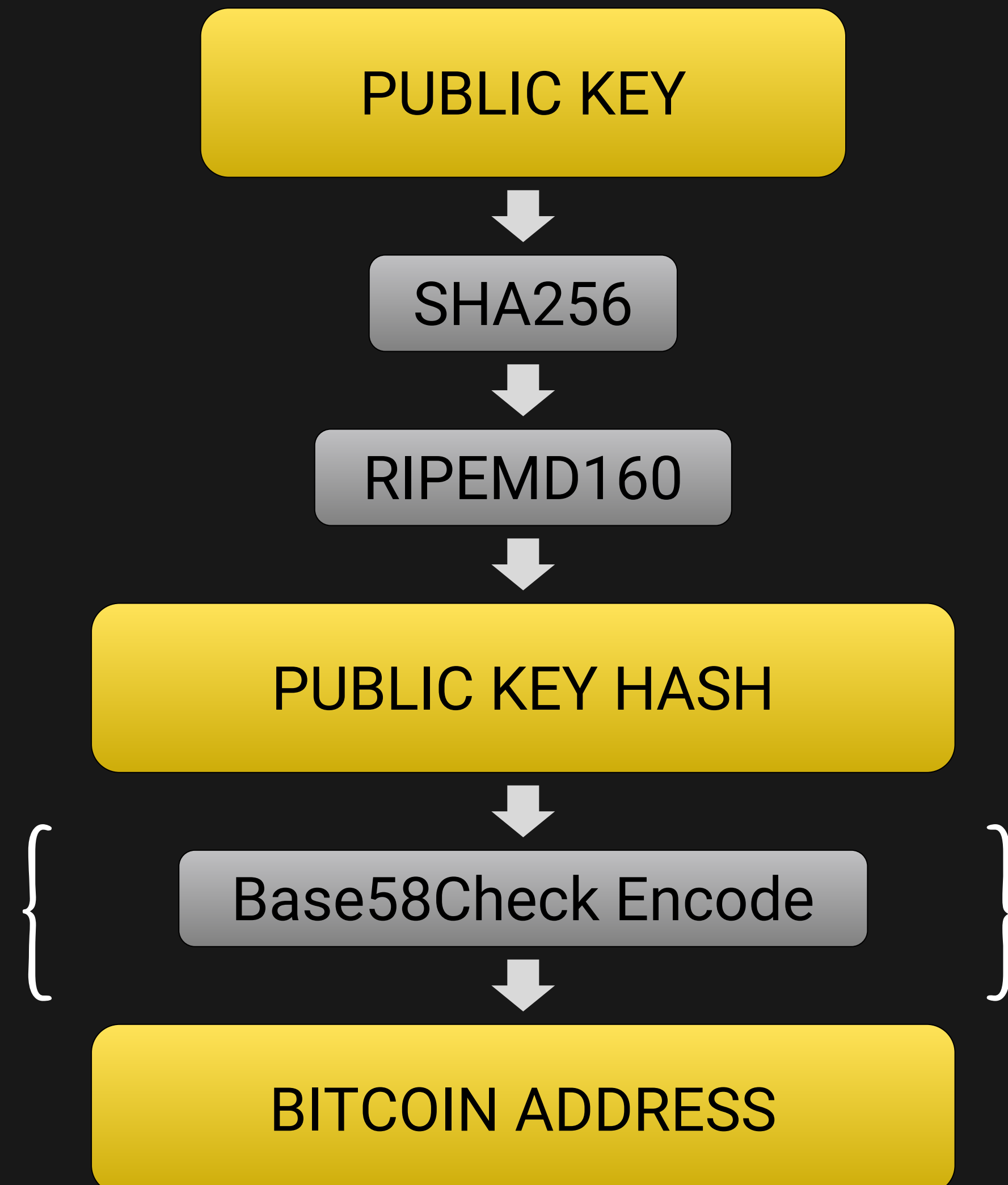


- **prefix**: “version byte” based on type of data
 - extended RIPEMD-160 hash
- Bitcoin Addresses are Base58Check Encoded
 - **Base-58** alphabet:
1234567890ABCDEFGHIJKLMN0PQRSTUVWXYZa
bcdefghijklmnopqrstuvwxyz
 - 58 characters (omits 0, O, l, I)



PUBLIC KEY TO BITCOIN ADDRESS

PUBLIC KEY HASH TO ADDRESS

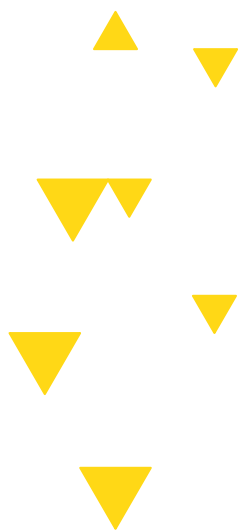


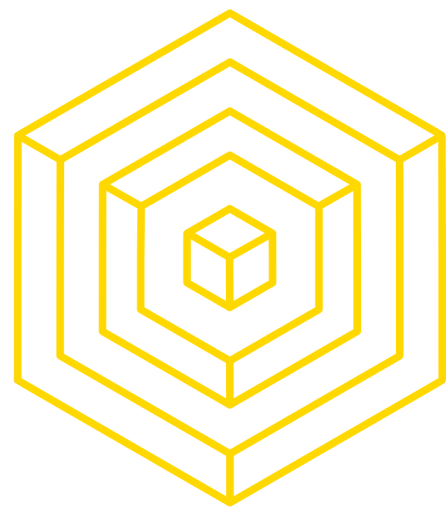
- **checksum**: 4-byte error-checking code appended to the end of an address
 - checksum = $\text{SHA256}(\text{SHA256}(\text{extended_RIPEMD160_hash}))$
 - first 4 bytes
- Decoding software uses checksum to validate address
 - Bitcoin address
 - 1 byte version
 - 20 bytes pub key hash
 - 4 bytes checksum



4

BITCOIN SCRIPT





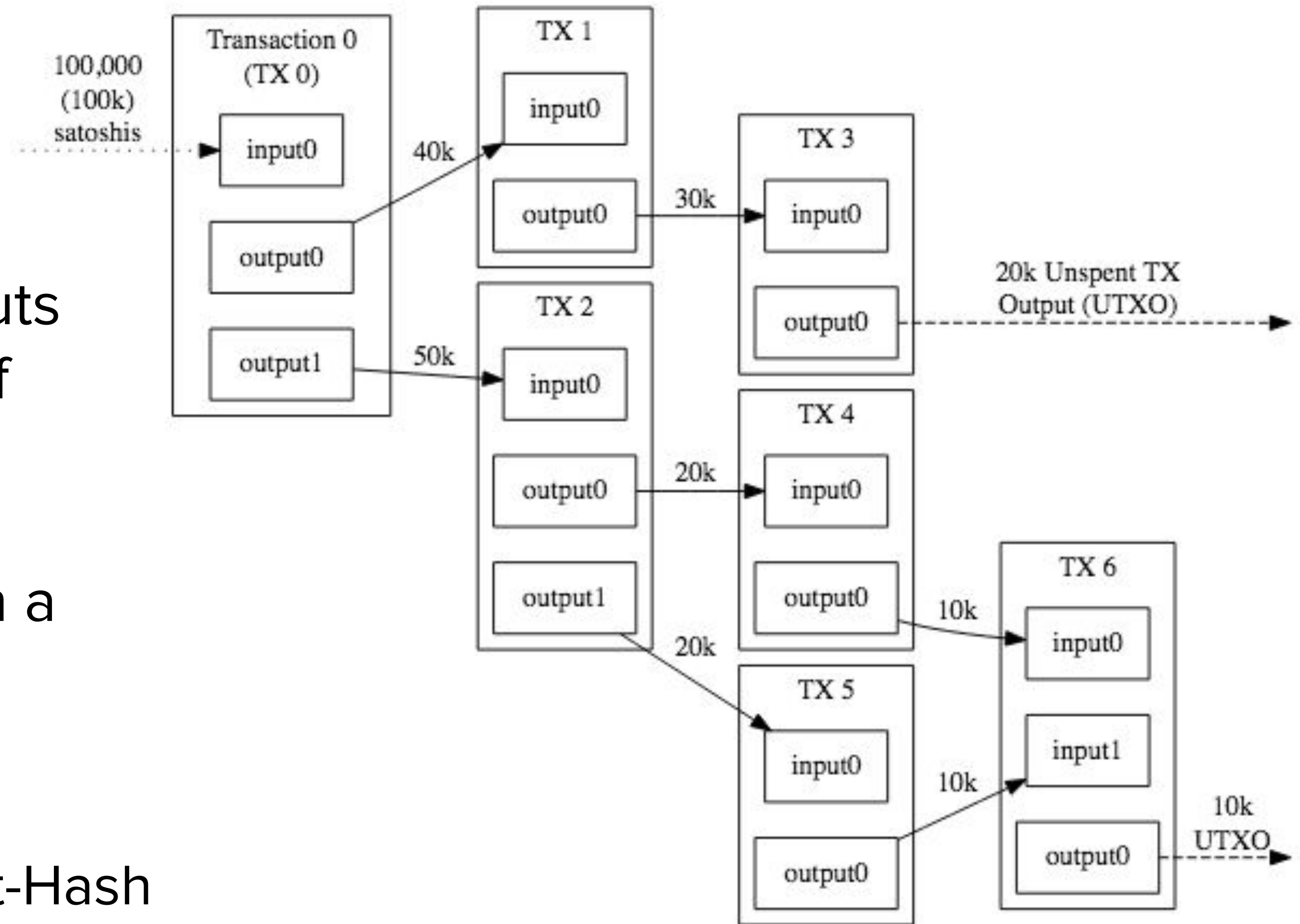
BITCOIN SCRIPTS

REMEMBER THE UTXO MODEL?

Source: [Bitcoin Developer Guide](#)

Reminders:

- Bitcoin uses a UTXO model
- Transactions map inputs to outputs
- Transactions contain signature of owner of funds
- Spending Bitcoin is **redeeming** previous transaction outputs with a **proof**
 - Public Key + Signature in in Pay-to-Pub-Key-Hash
 - Script + Signature in Pay-to-Script-Hash



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



CONTENTS OF A TRANSACTION



CONTENTS OF A TRANSACTION - METADATA

```
“hash”: “5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b”,
```

```
“ver”: 1,
```

```
“vin_sz”: 2,
```

```
“vout_sz”: 1,
```

```
“lock_time”: 0,
```

```
“size”: 404,
```

```
“in”: [
  {
    “prev_out”: {
      “hash”: “3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260”,
      “n”: 0
    },
    “scriptSig”: “30440...”
  },
  {
    “prev_out”: {
      “hash”: “7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e”,
      “n”: 0
    },
    “scriptSig”: “3f3a4ce81....”
  }
],
“out”: [
  {
    “value”: 1012287097”,
    “scriptPubKey”: “OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG”
  }
]
```

hash or “ID”
of this transaction

CONTENTS OF A TRANSACTION - METADATA

```
“hash”: “5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b”,
```

```
“ver”: 1,
```

```
“vin_sz”: 2, ← size (number) of inputs
```

```
“vout_sz”: 1, ← size (number) of outputs
```

```
“lock_time”: 0,
```

```
“size”: 404,
```

```
“in”: [
  {
    “prev_out”: {
      “hash”: “3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260”,
      “n”: 0
    },
    “scriptSig”: “30440...”
  },
  {
    “prev_out”: {
      “hash”: “7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e”,
      “n”: 0
    },
    “scriptSig”: “3f3a4ce81....”
  }
],
“out”: [
  {
    “value”: 1012287097”,
    “scriptPubKey”: “OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG”
  }
]
```

hash or “ID”
of this transaction

CONTENTS OF A TRANSACTION - METADATA

“hash”: “5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b”,

“ver”: 1, ← version

“vin_sz”: 2, ← size (number) of inputs

“vout_sz”: 1, ← size (number) of outputs

“lock_time”: 0, ← lock time (useful for scripting)

“size”: 404, ← size of transaction

hash or “ID”
of this transaction

```

“in”: [
  {
    “prev_out”: {
      “hash”: “3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260”,
      “n”: 0
    },
    “scriptSig”: “30440...”
  },
  {
    “prev_out”: {
      “hash”: “7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e”,
      “n”: 0
    },
    “scriptSig”: “3f3a4ce81....”
  }
],
“out”: [
  {
    “value”: 1012287097,
    “scriptPubKey”: “OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG”
  }
]

```

CONTENTS OF A TRANSACTION - INPUTS

```

{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404
  "in": [
    {
      "prev_out": {
        "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
        "n": 0
      },
      "scriptSig": "30440..."
    },
    {
      "prev_out": {
        "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
        "n": 0
      },
      "scriptSig": "3f3a4ce81....."
    }
  ],
  "out": [
    {
      "value": 1012287097,
      "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
}

```

remember these?

CONTENTS OF A TRANSACTION - INPUTS

```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",

```

```
  "ver": 1,
```

```
  "vin_sz": 2,
```

```
  "vout_sz": 1,
```

```
  "lock_time": 0,
```

```
  "size": 404
```

```
  "in": [
```

```
    {
```

```
      "prev_out": {
```

```
        "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
```

```
        "n": 0
```

```
      },
```

```
      "scriptSig": "30440..."
```

```
    },
```

```
    {
```

```
      "prev_out": {
```

```
        "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
```

```
        "n": 0
```

```
      },
```

```
input 2: "scriptSig": "3f3a4ce81...."
```

```
    }
  ],
```

```
  "out": [
```

```
    {
```

```
      "value": 1012287097,
```

```
      "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
```

remember these?

ID of previous transactions being referenced

CONTENTS OF A TRANSACTION - INPUTS

```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404
  "in": [
```

```
  {
    "prev_out": {
      "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
      "n": 0 ← index of input in previous transaction
```

input 1:

```
    "scriptSig": "30440..."
```

ID of previous transactions being referenced

```
  },
  {
    "prev_out": {
      "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
      "n": 0 ← index of input in previous transaction
```

input 2:

```
    "scriptSig": "3f3a4ce81...."
```

```
  ],
  "out": [
    {
      "value": 1012287097,
      "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
}
```

CONTENTS OF A TRANSACTION - INPUTS

```

{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404
  "in": [
    {
      "prev_out": {
        "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
        "n": 0 ← index of input in previous transaction
      },
      "scriptSig": "30440..." ← signature used to redeem previous transaction output
    },
    {
      "prev_out": {
        "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
        "n": 0 ← index of input in previous transaction
      },
      "scriptSig": "3f3a4ce81...." ← signature used to redeem previous transaction output
    }
  ],
  "out": [
    {
      "value": 1012287097,
      "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
}

```

CONTENTS OF A TRANSACTION - OUTPUTS

```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404,
  "in": [
    {
      "prev_out": {
        "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
        "n": 0
      },
      "scriptSig": "30440..."
    },
    {
      "prev_out": {
        "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
        "n": 0
      },
      "scriptSig": "3f3a4ce81...."
    }
  ]
},
```

output amount (how much satoshi is being sent)

```
"out": [
  {
    "value": 1012287097",
    "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
  }
]
```

type of script

output script



BITCOIN SCRIPTS

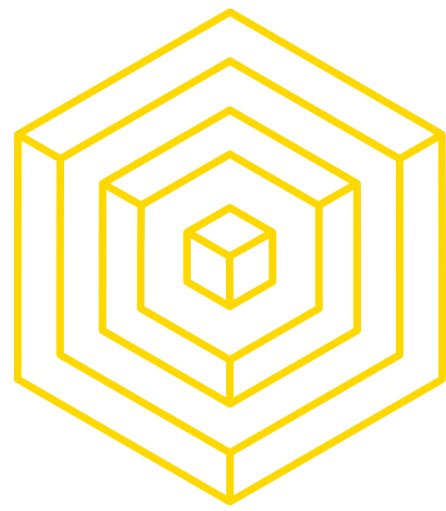
REMINDERS

Output “addresses” are actually scripts.

“scriptPubKey”: “OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG”

→ This particular Output Script: “This amount can be redeemed by the **public key** that hashes to address X, plus a **signature** from the owner of that public key”

- Inputs and outputs through scripting allows for future extensibility of Bitcoin.
- **Script or “Bitcoin Scripting Language”**: Language built specifically for Bitcoin
 - Stack based
 - Native support for cryptography
 - Simple, not turing complete (no loops)



BITCOIN SCRIPTS

P2PKH EXAMPLE

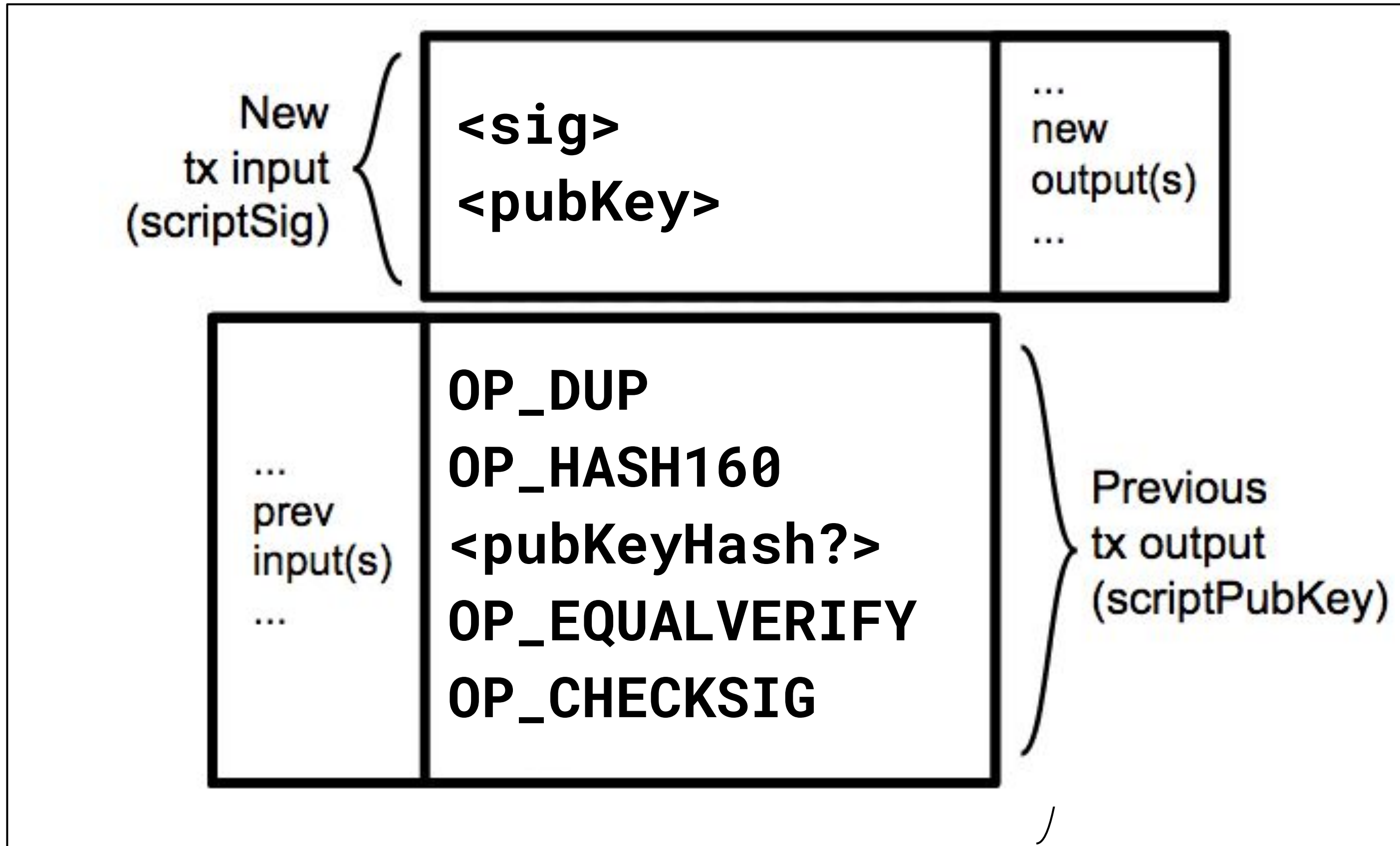


Figure: Two transactions along with their input and output scripts

AUTHOR: GLORIA ZHAO

Source: Princeton textbook

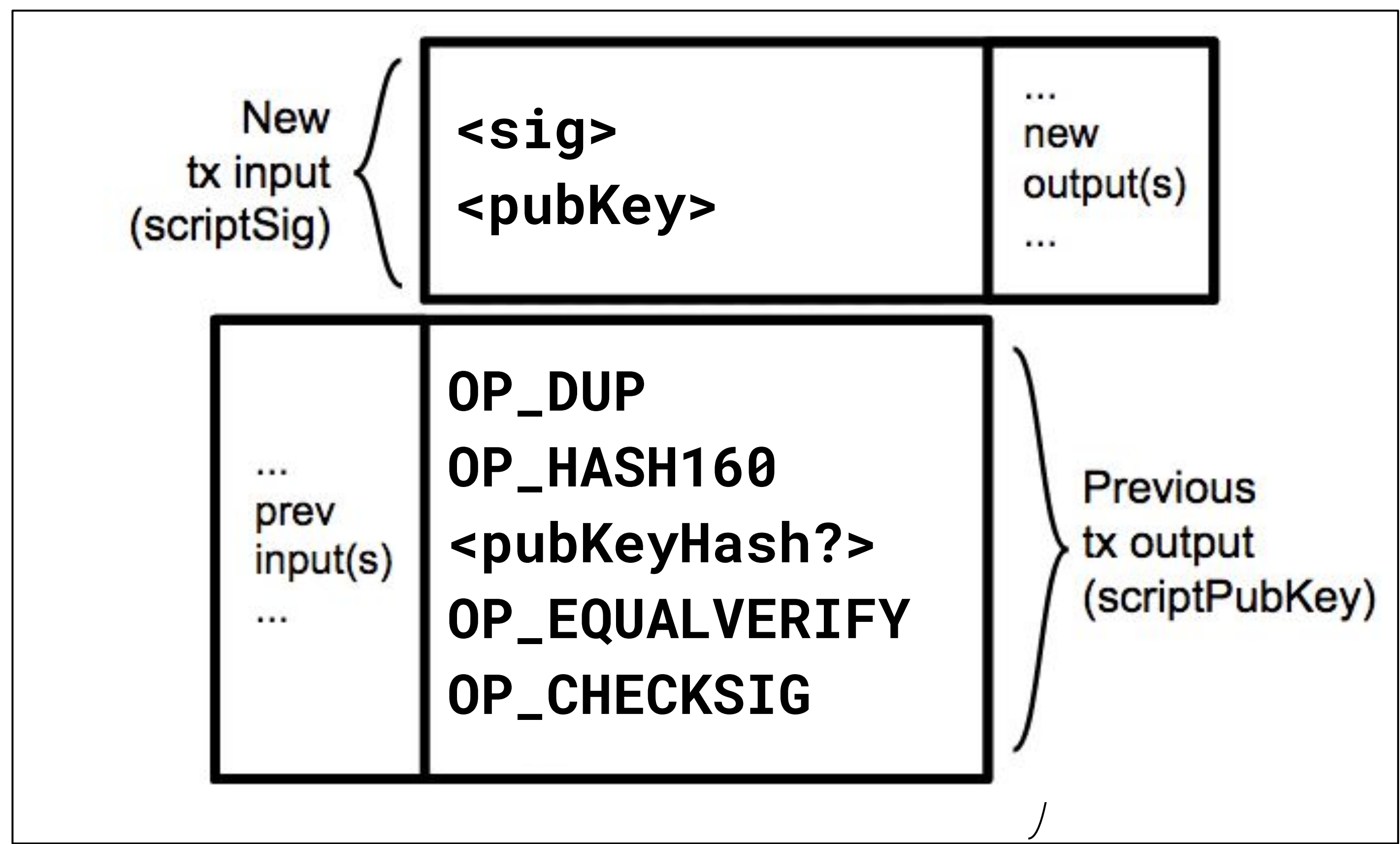
- **unlocking script (scriptSig):** found in **transaction input**, provided by the spender to redeem the output of a previous transaction
- **locking script (scriptPubKey):** found in **previous transaction output**, specifies requirements for redeeming transaction
- bitcoin validating node will execute the locking and unlocking scripts in sequence



BITCOIN SCRIPTS

P2PKH EXAMPLE

`"scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"`



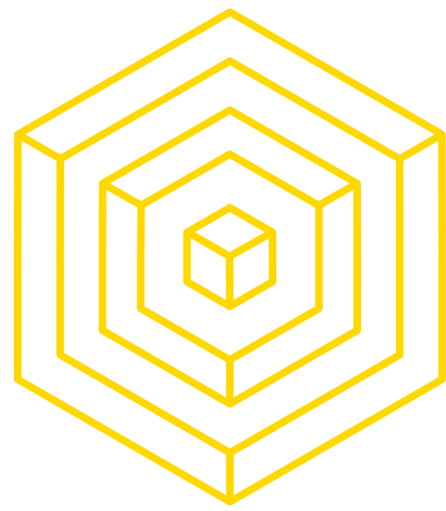
Code Execution

```

"scriptPubKey": "
OP_DUP
OP_HASH160
69e02e18b5705a05d.....
OP_EQUALVERIFY
OP_CHECKSIG"

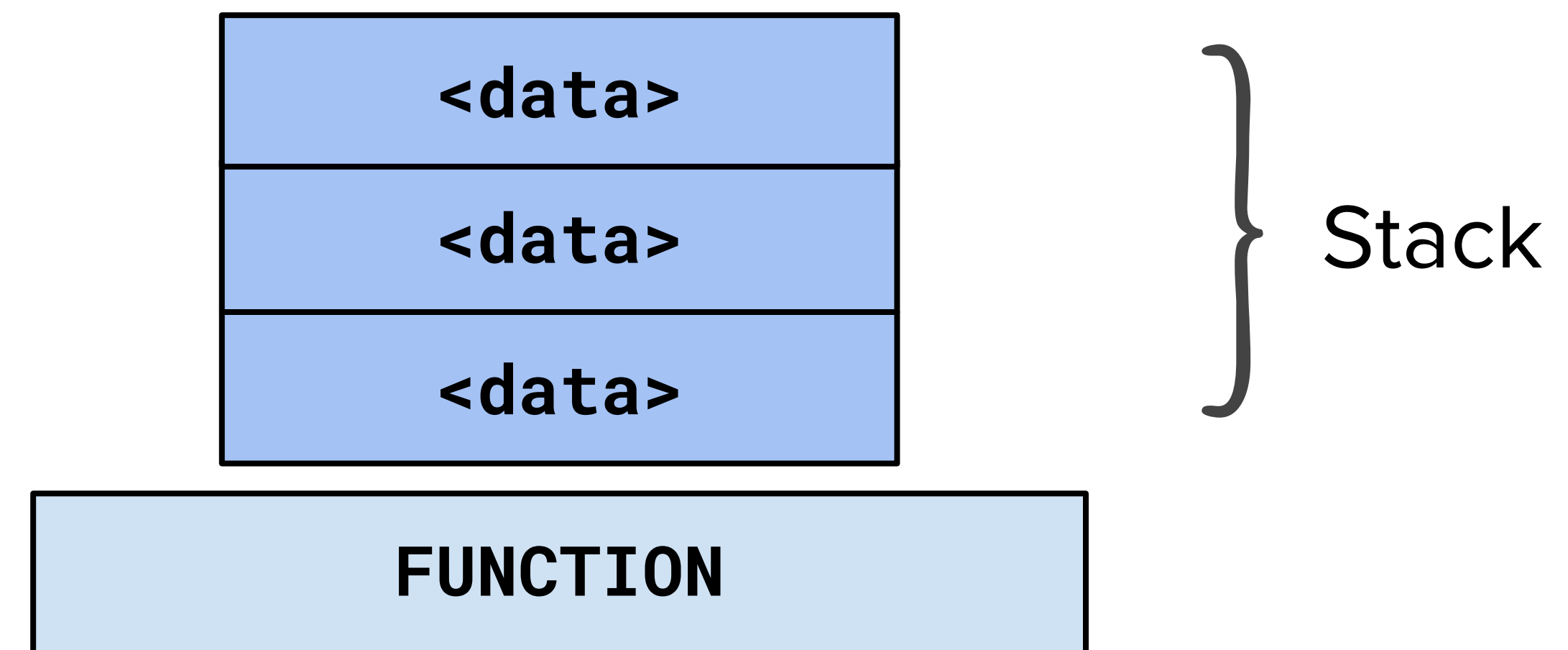
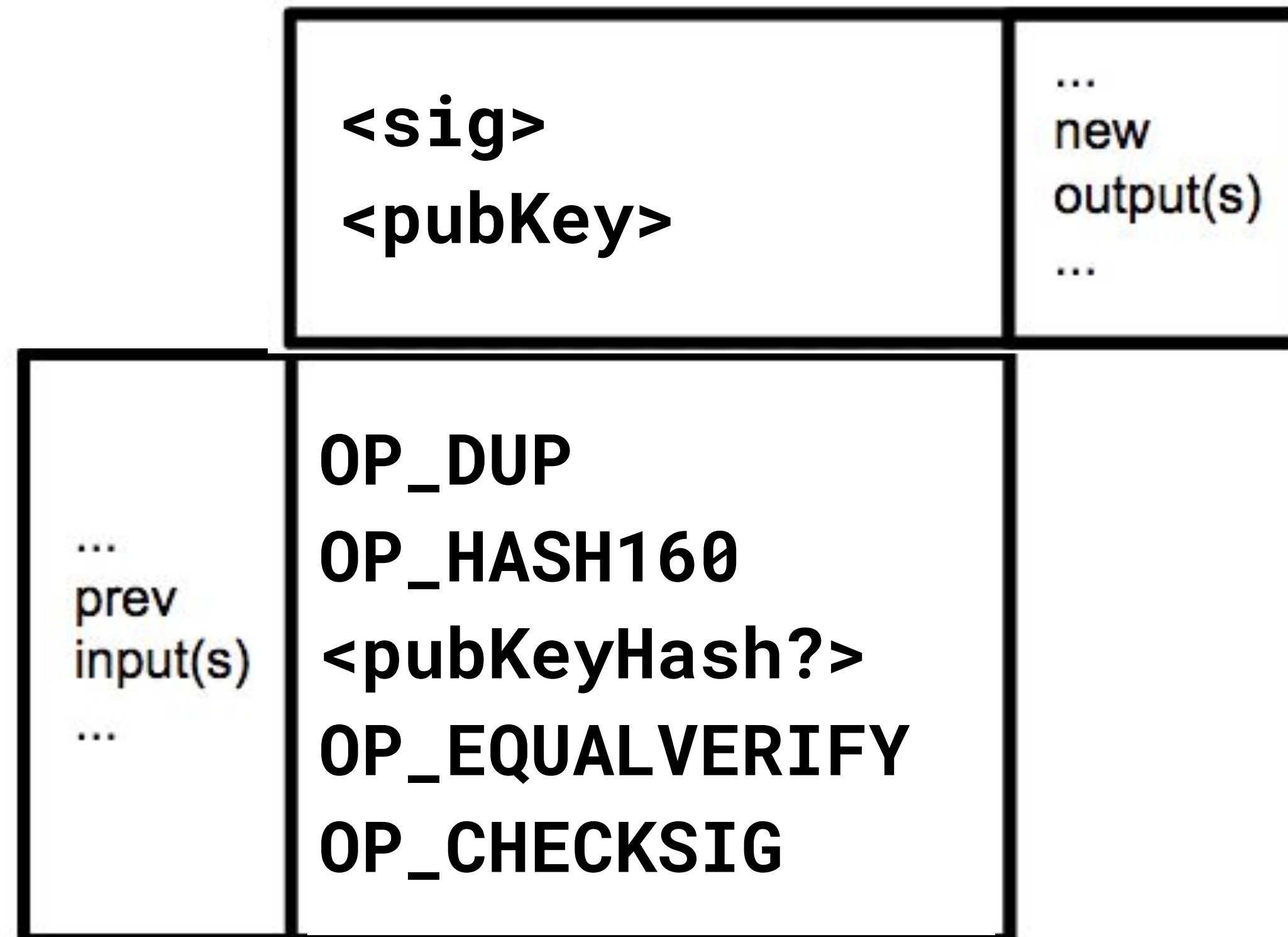
```

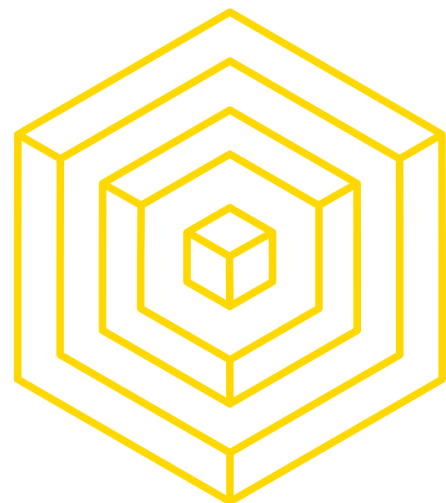
Figure: Two transactions along with their input and output scripts



BITCOIN SCRIPTS

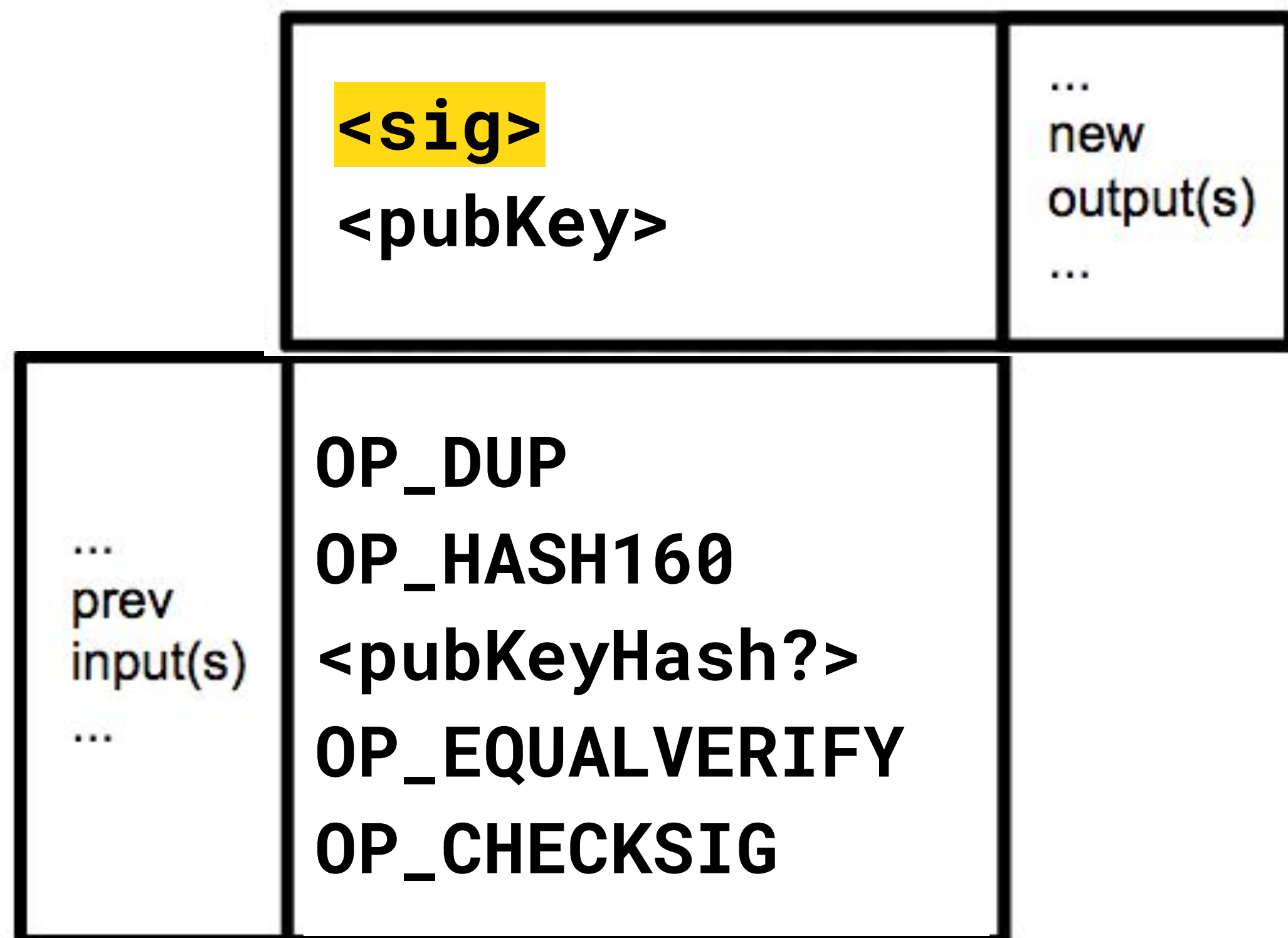
P2PKH EXAMPLE EXECUTION

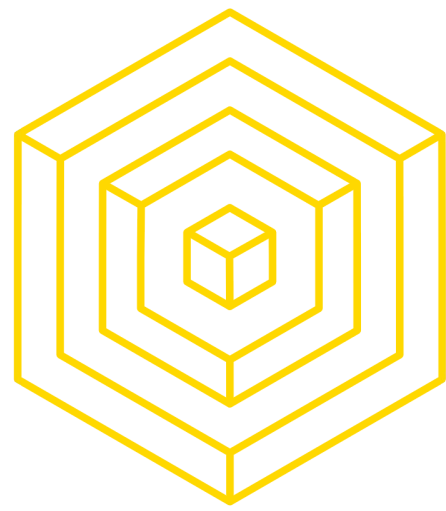




BITCOIN SCRIPTS

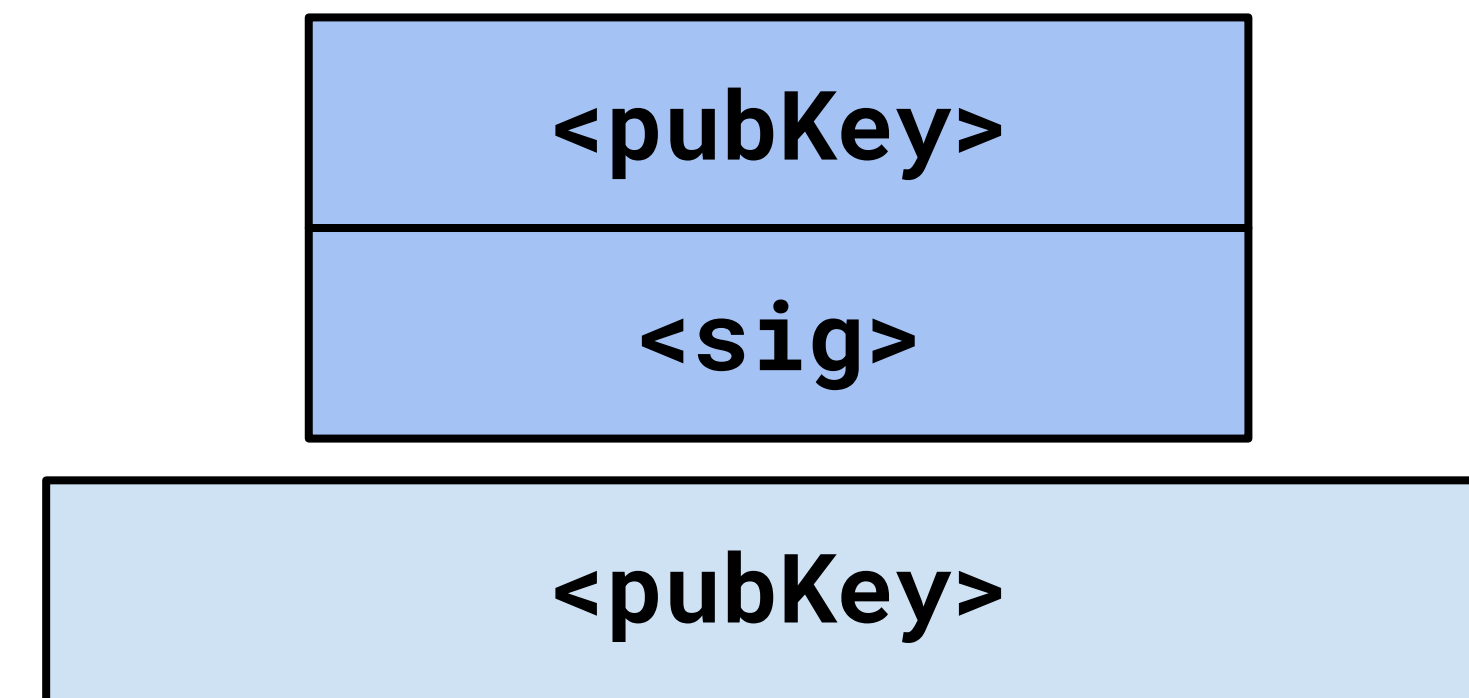
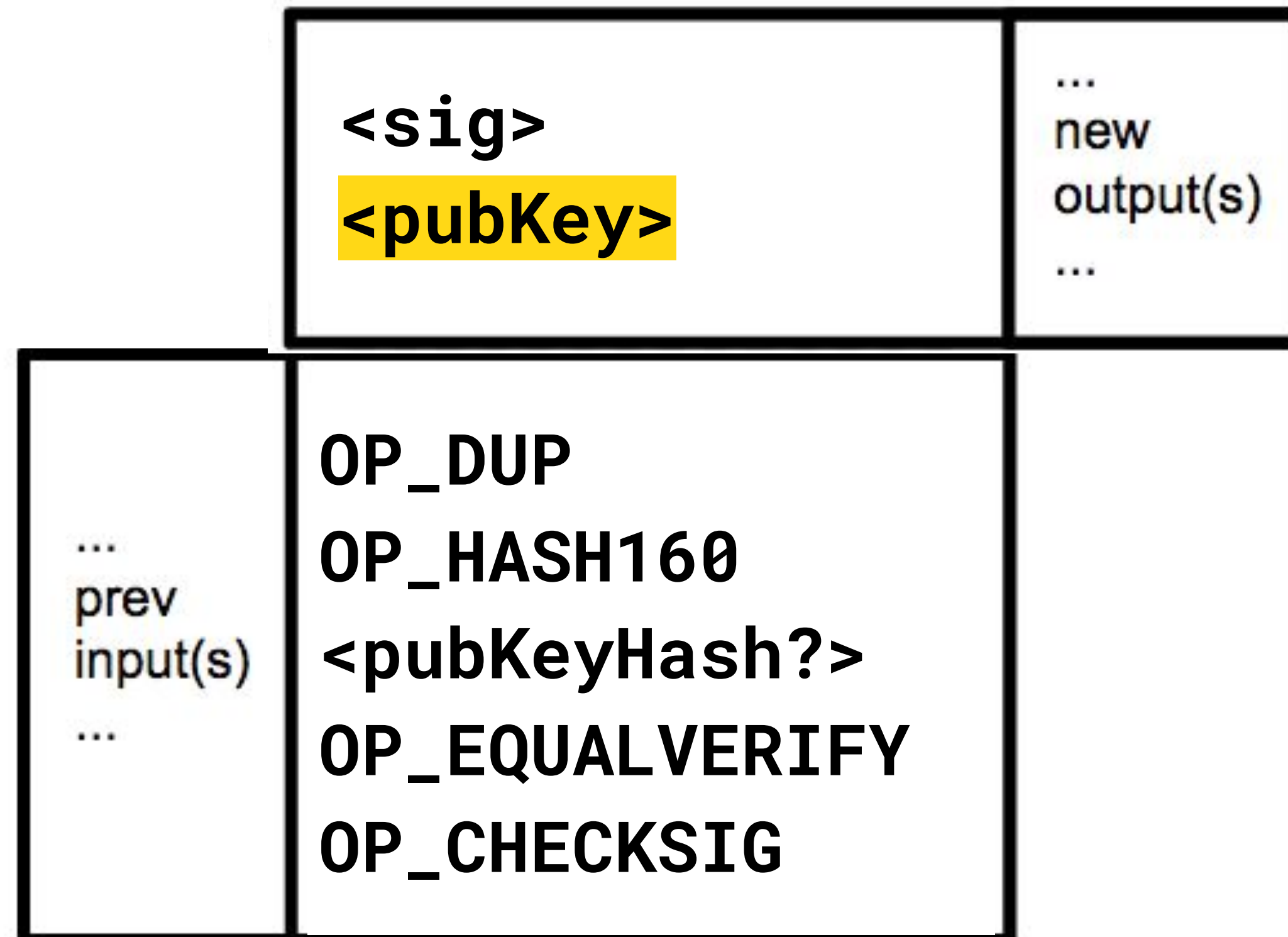
P2PKH EXAMPLE EXECUTION





BITCOIN SCRIPTS

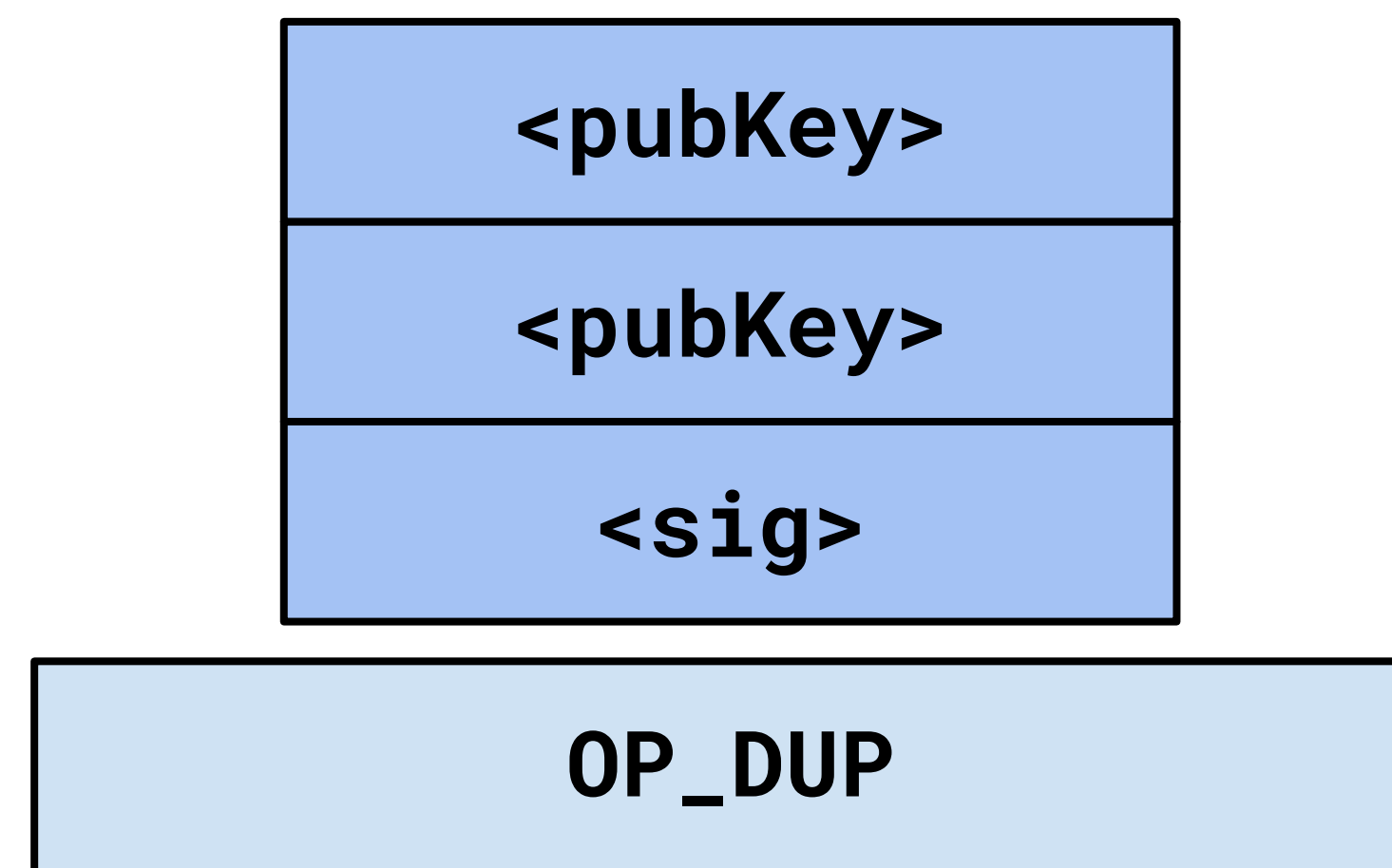
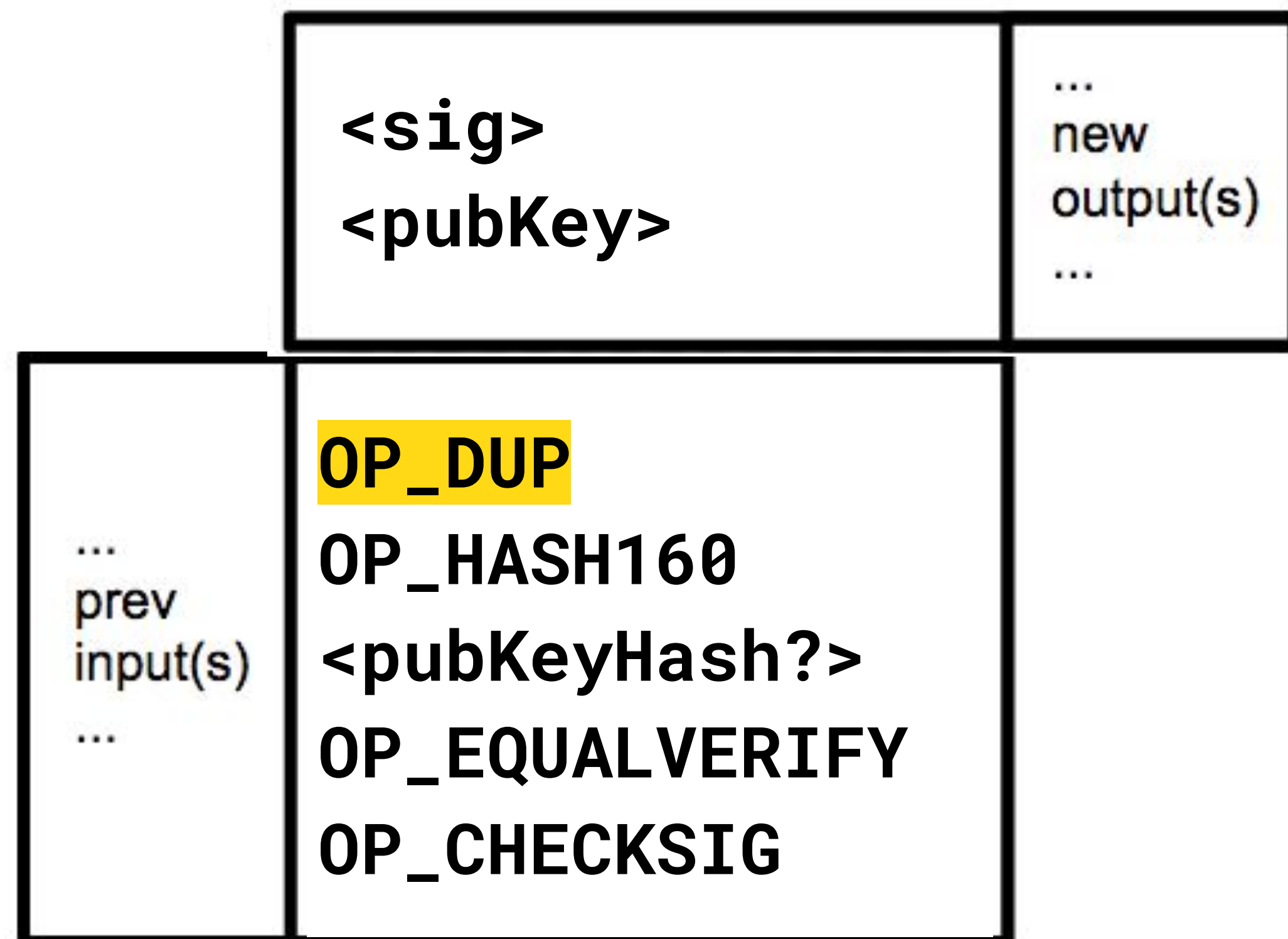
P2PKH EXAMPLE EXECUTION





BITCOIN SCRIPTS

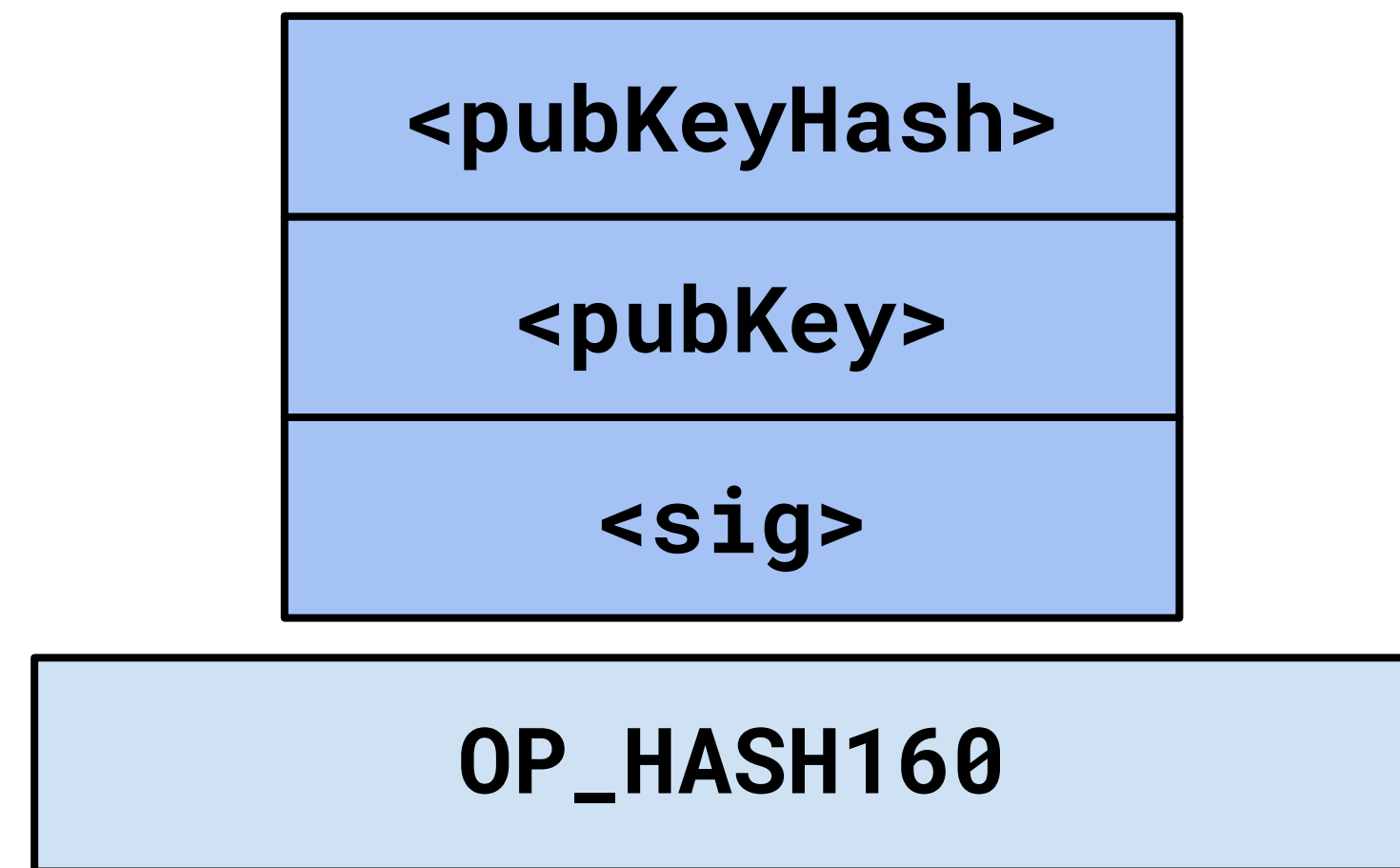
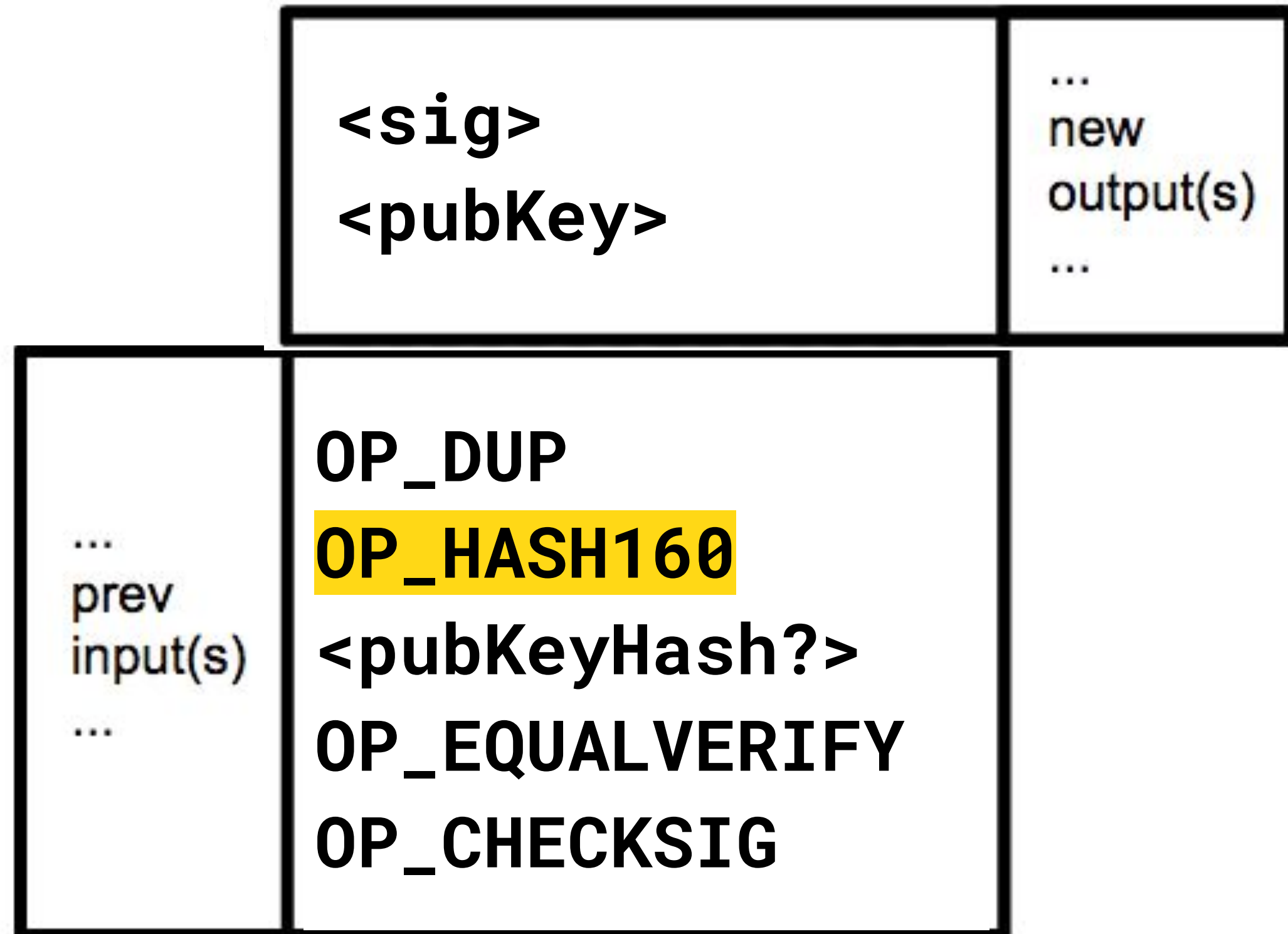
P2PKH EXAMPLE EXECUTION

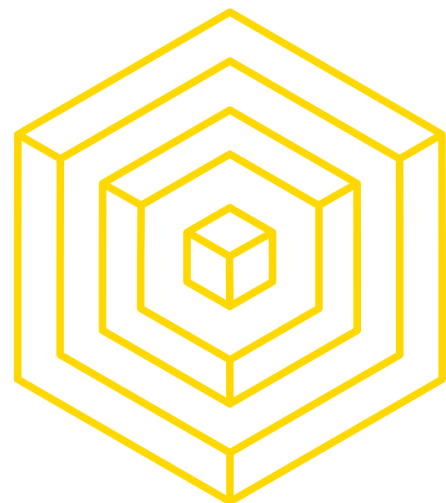




BITCOIN SCRIPTS

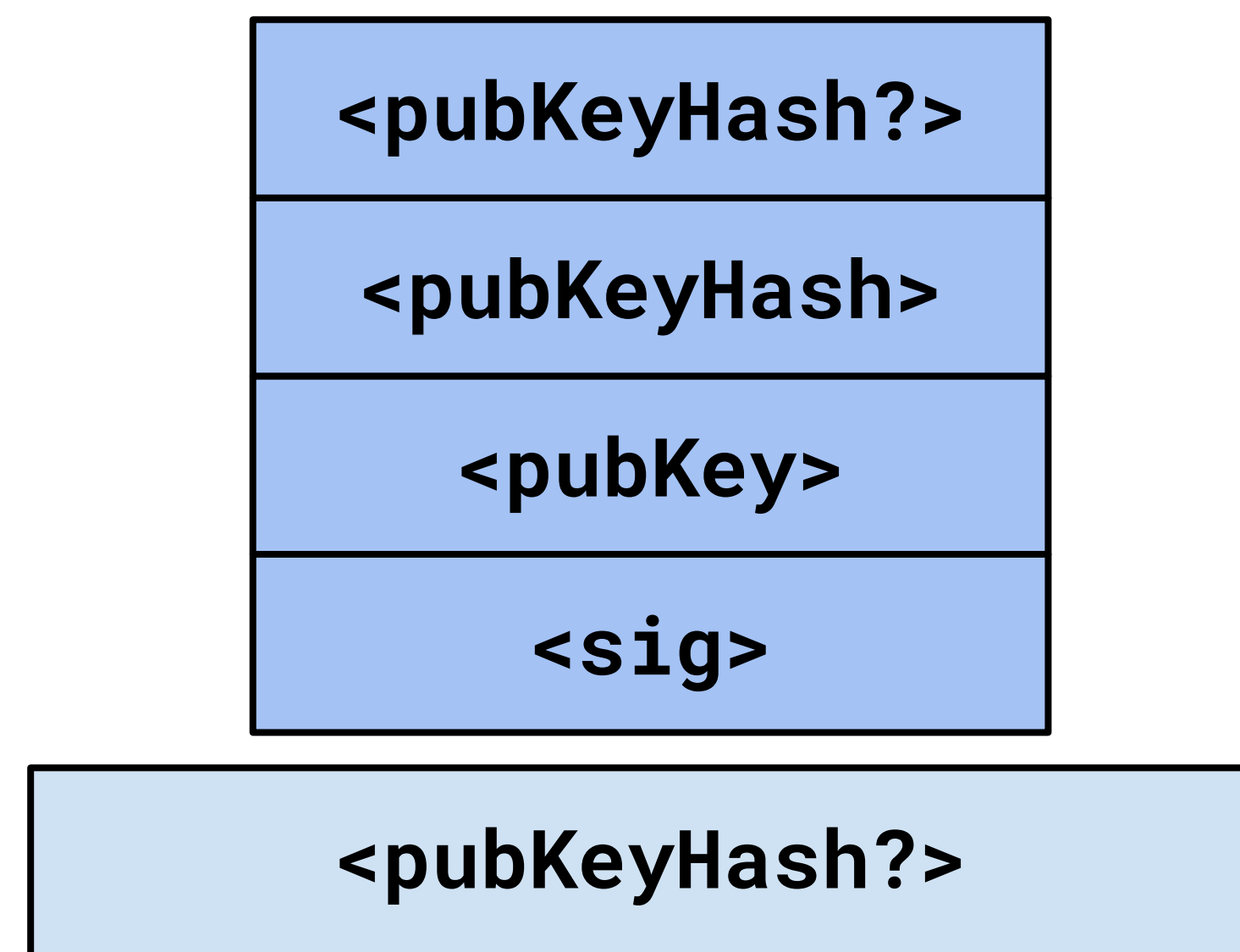
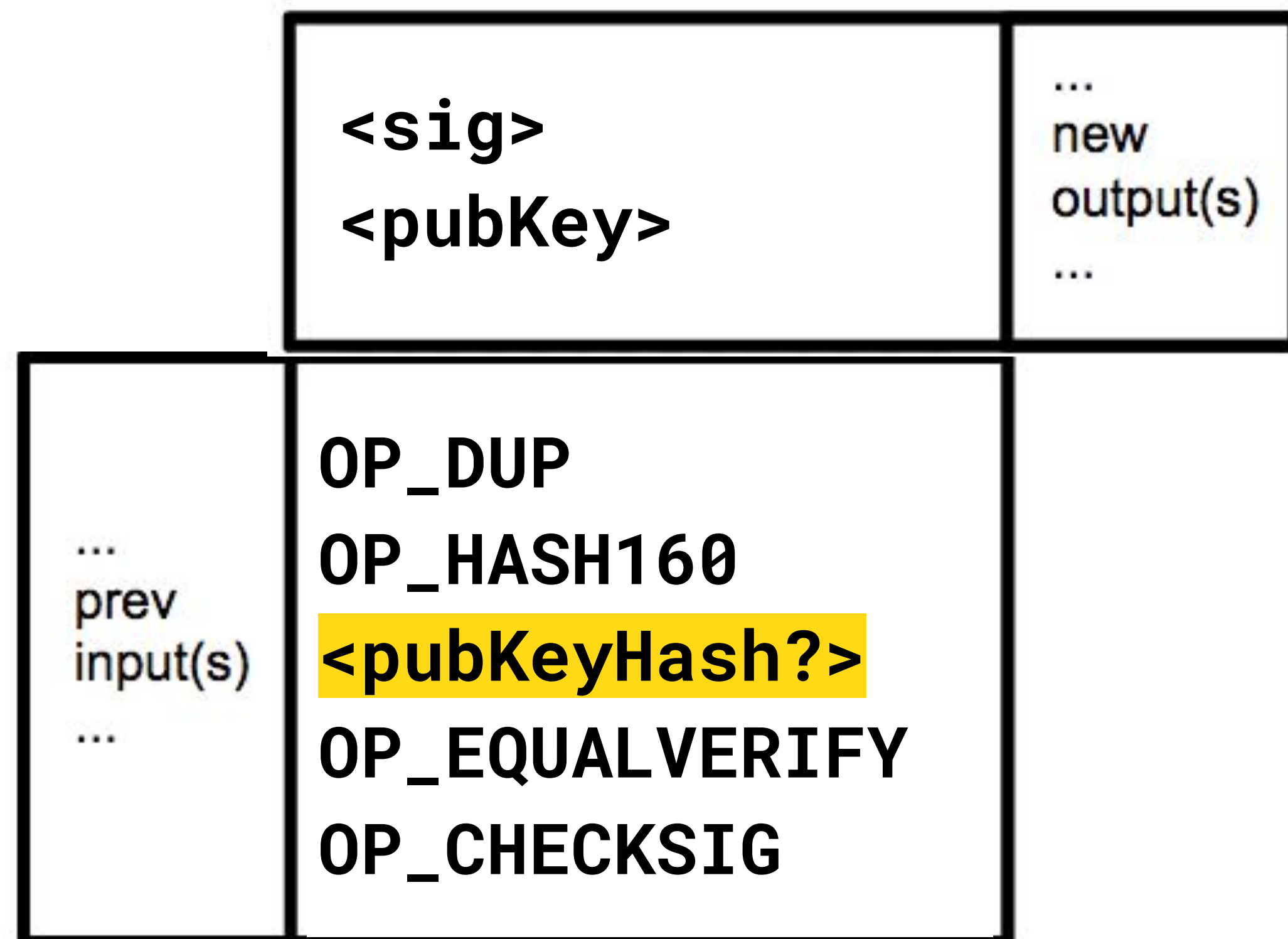
P2PKH EXAMPLE EXECUTION

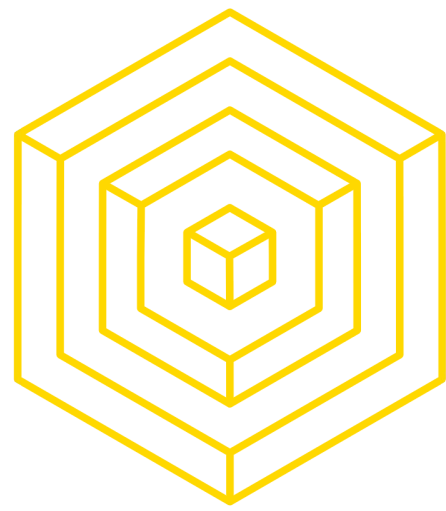




BITCOIN SCRIPTS

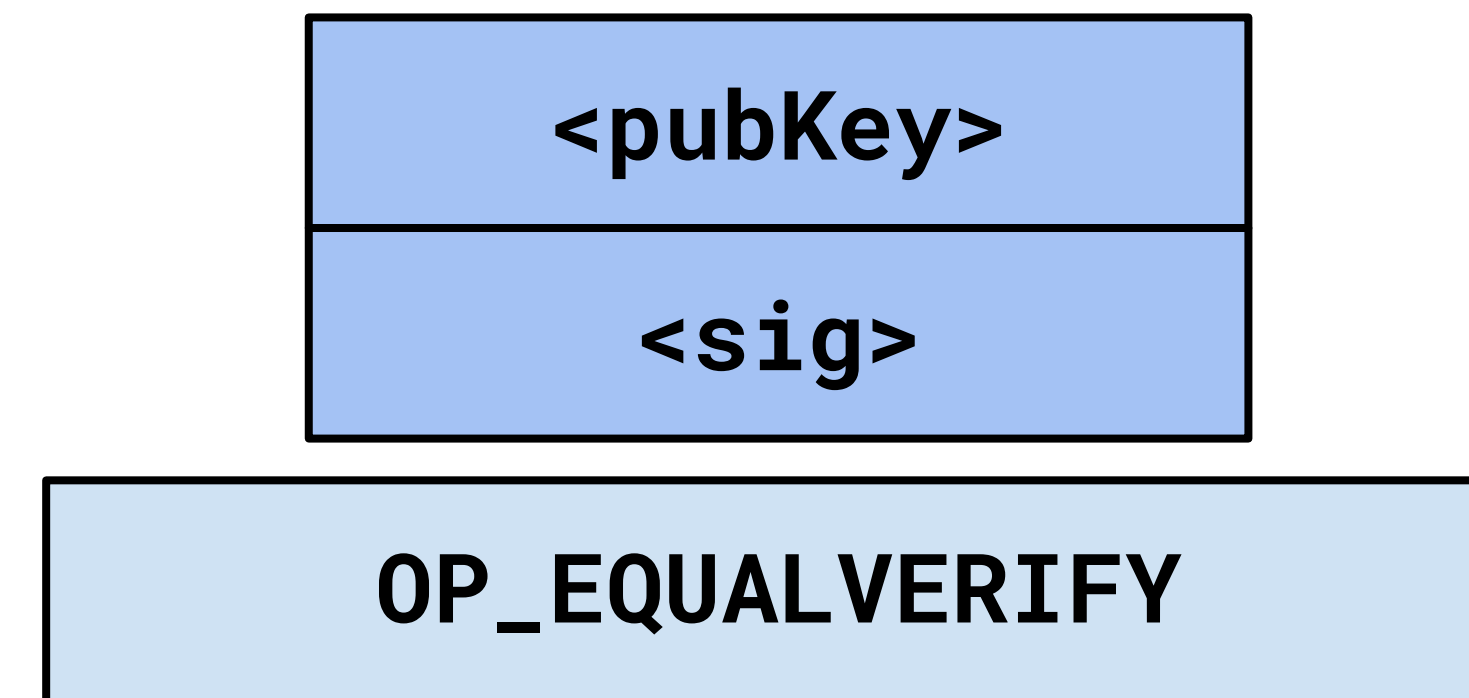
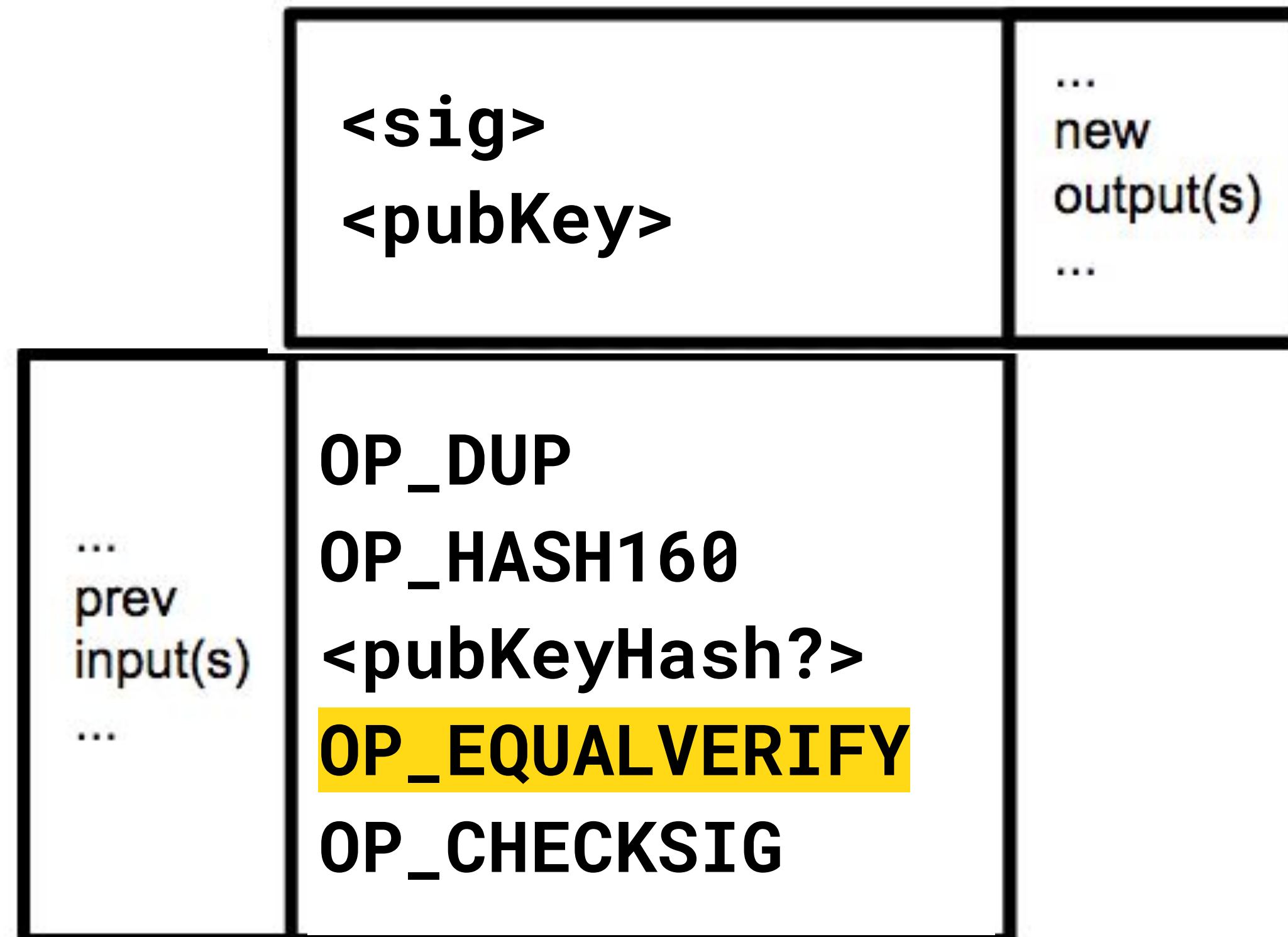
P2PKH EXAMPLE EXECUTION





BITCOIN SCRIPTS

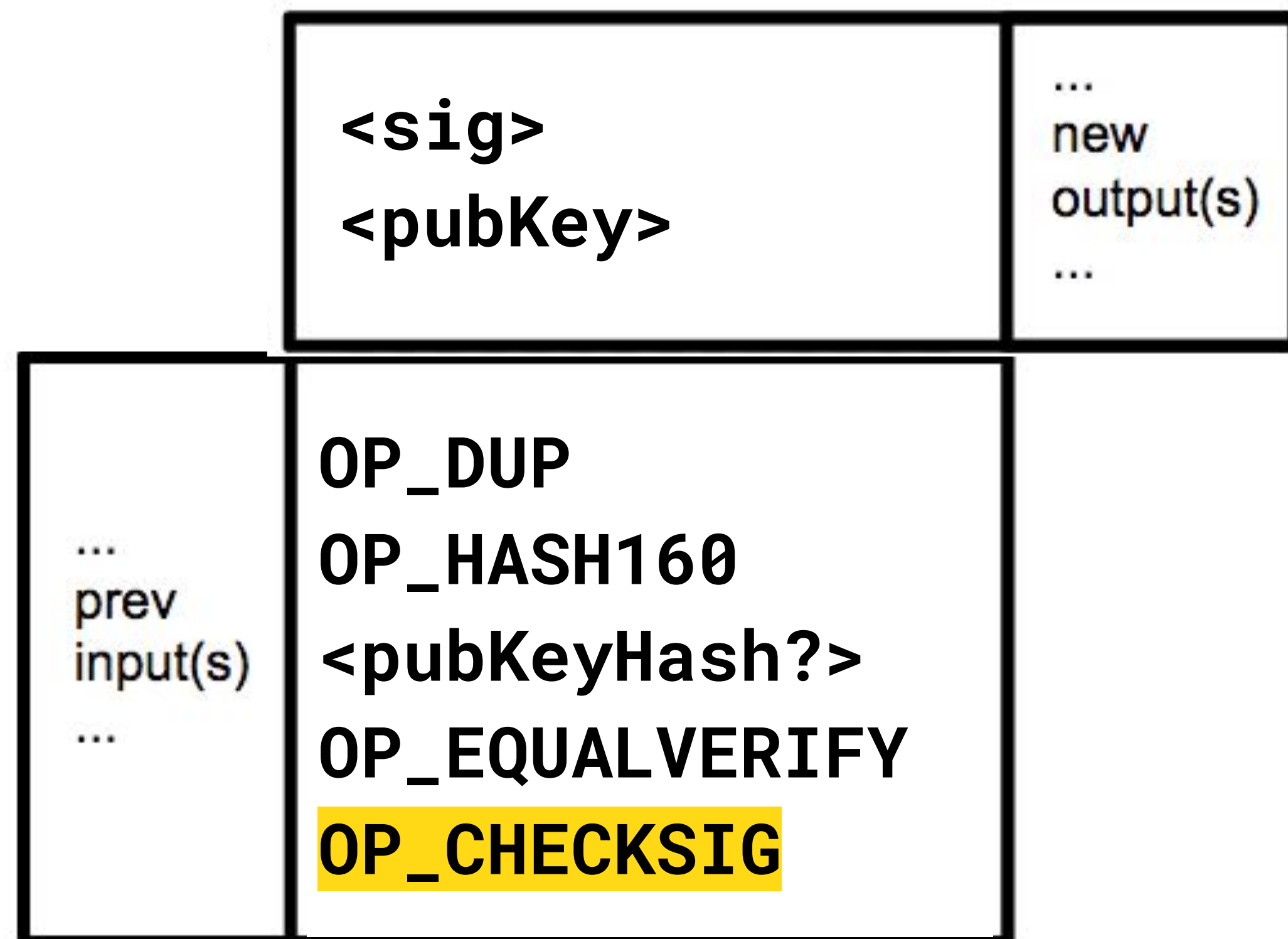
P2PKH EXAMPLE EXECUTION





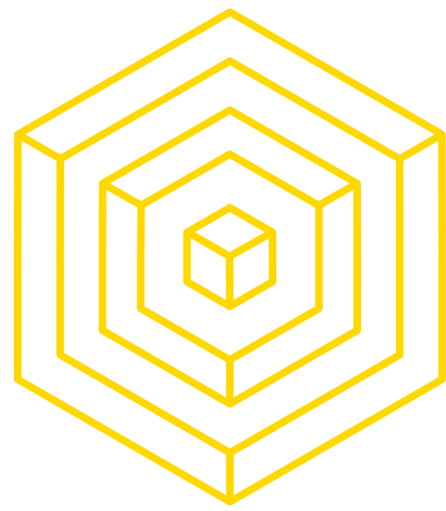
BITCOIN SCRIPTS

P2PKH EXAMPLE EXECUTION



true

OP_CHECKSIG



EXTRA: BITCOIN SCRIPTS

PROOF OF BURN

```

CryptoGraffiti.info v0.83  READ  WRITE  TOOLS  HELP  ABOUT
#4660 12. Sep 2016 20:18:38
I was here. I existed. I lived, loved, had good and bad times. Alastair Langwell - Unix Time 1473729285
1MVpQJA7FtcDrwKC6zATkZvZcxqma4JixS
2e0a91af45742908ebf7a5aeb3e36a2261e7a15c5336ca4a32605df866260908
  
```

Output script:

OP_RETURN

<arbitrary data>

How to write arbitrary data into the Bitcoin blockchain?

Proof of Burn

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

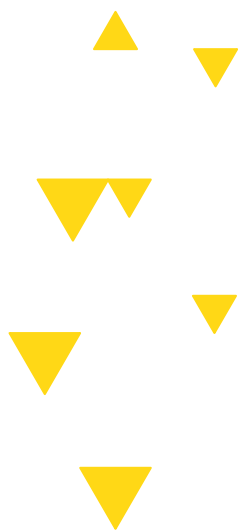
Use cases

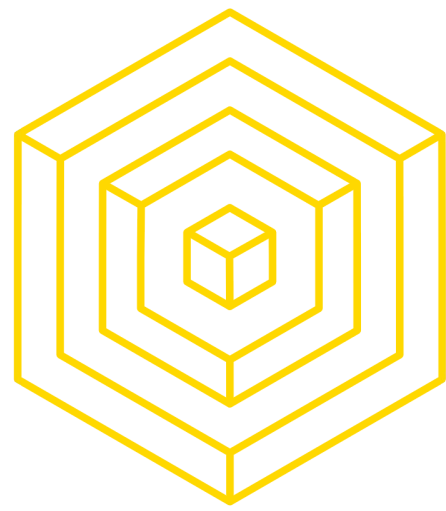
- Prove existence of something at a particular point in time
 - Ex. A word you coined, hash of a document/music/creative works
- Bootstrap altcoin by requiring that you destroy some Bitcoin to get altcoin



5

P2SH

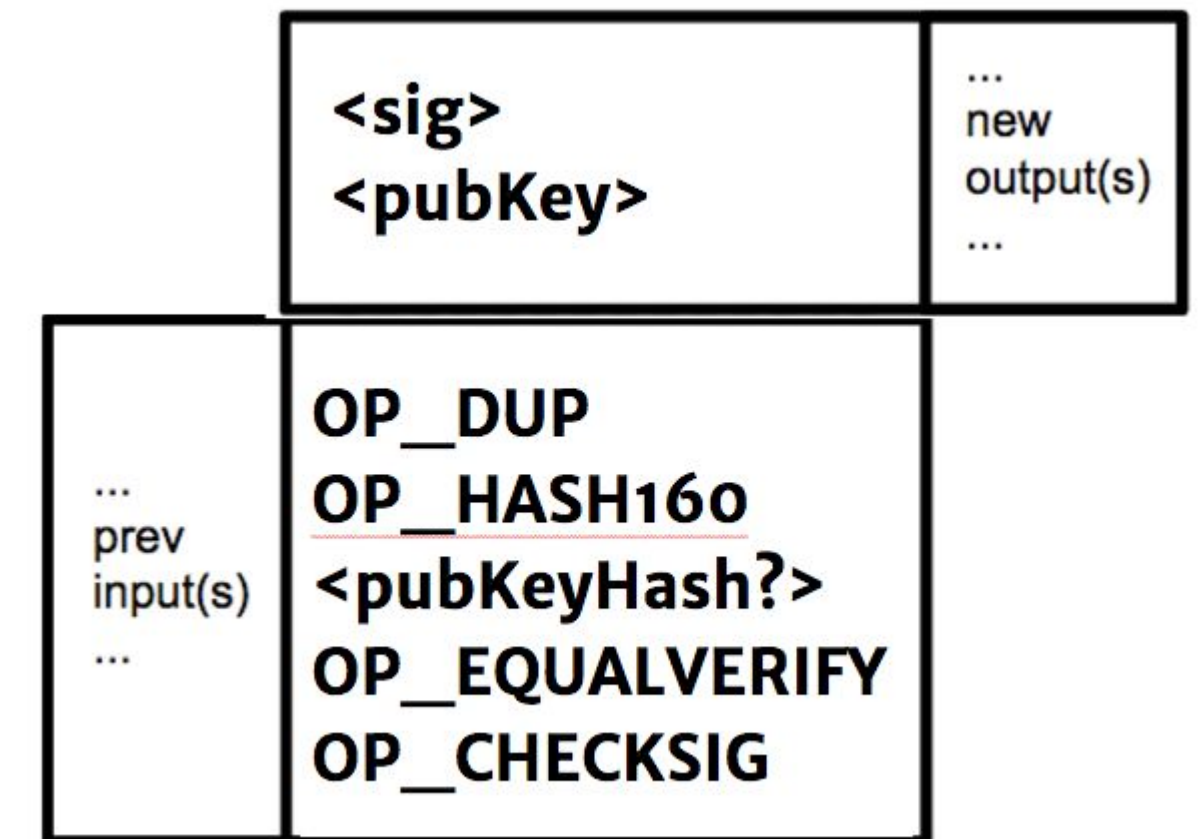




P2PKH VS P2SH

WHO SPECIFIES THE SCRIPT?

- In Bitcoin, senders specify a **locking script**, recipients provide an **unlocking script**
- **Pay-to-Pub-Key-Hash (P2PKH)**: Vendor (recipient of transaction) says “Send your coins to the hash of this **Public Key**.”
 - Simplest case
 - By far the most common case
- **Pay-to-Script-Hash (P2SH)**: Vendor says “Send your coins to the hash of this **Script**; **I will provide the script** and the data to make the script evaluate to true when I redeem the coins.”
 - A vendor cannot say, “To pay me, write a complicated output script that will allow me to spend using multiple signatures.”

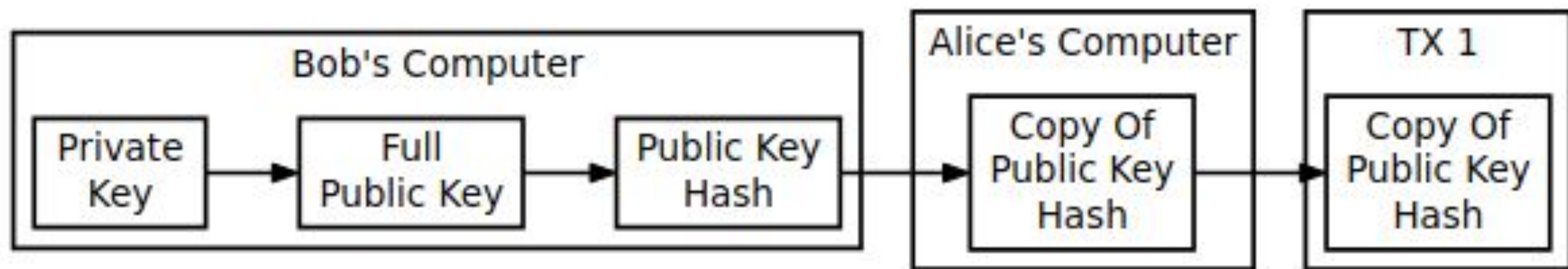


Simple P2PKH script - recipient only has to provide signature and public key



P2PKH VS P2SH

CREATING A P2PKH PUB KEY HASH



Creating A P2PKH Public Key Hash To Receive Payment



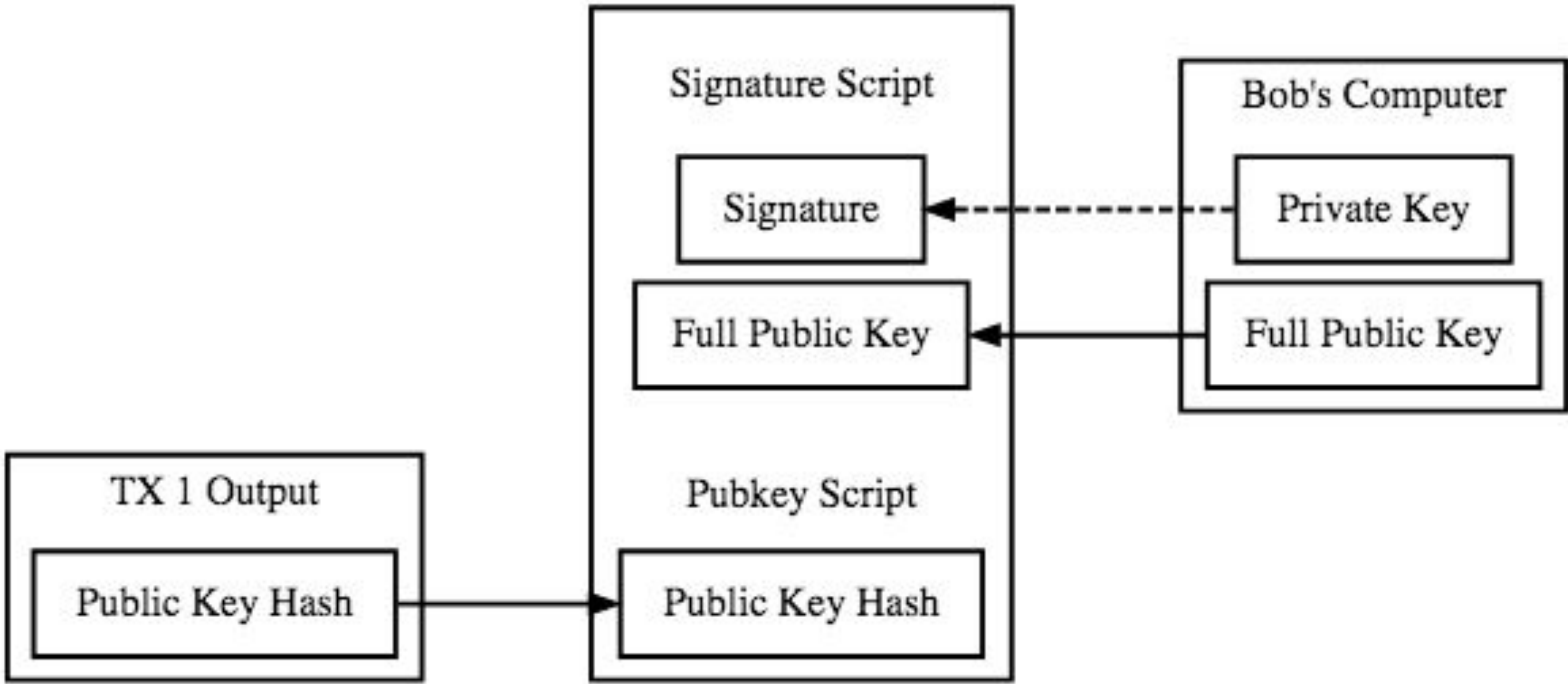
Source: Bitcoin Developer's Guide

AUTHOR: RUSTIE LIN



P2PKH VS P2SH

SPENDING A P2PKH OUTPUT



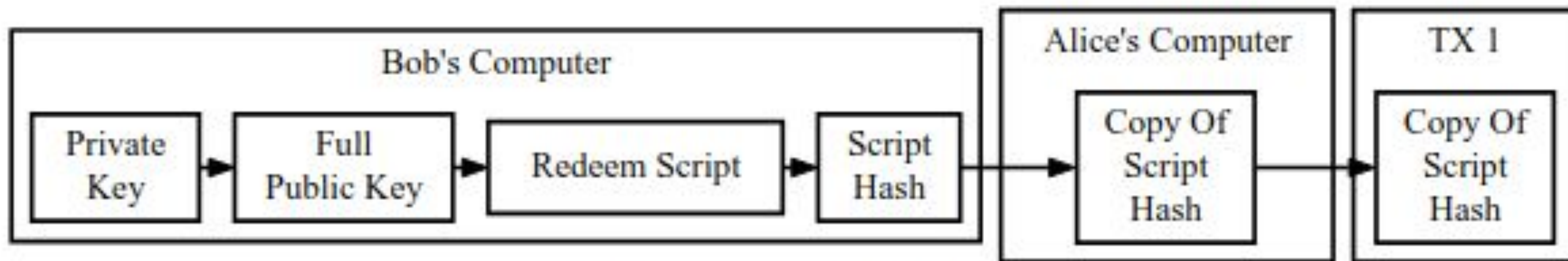
Spending A P2PKH Output

Source: Bitcoin Developer's Guide
AUTHOR: RUSTIE LIN



P2PKH VS P2SH

CREATING A P2SH REDEEM SCRIPT HASH



Creating A P2SH Redeem Script Hash To Receive Payment

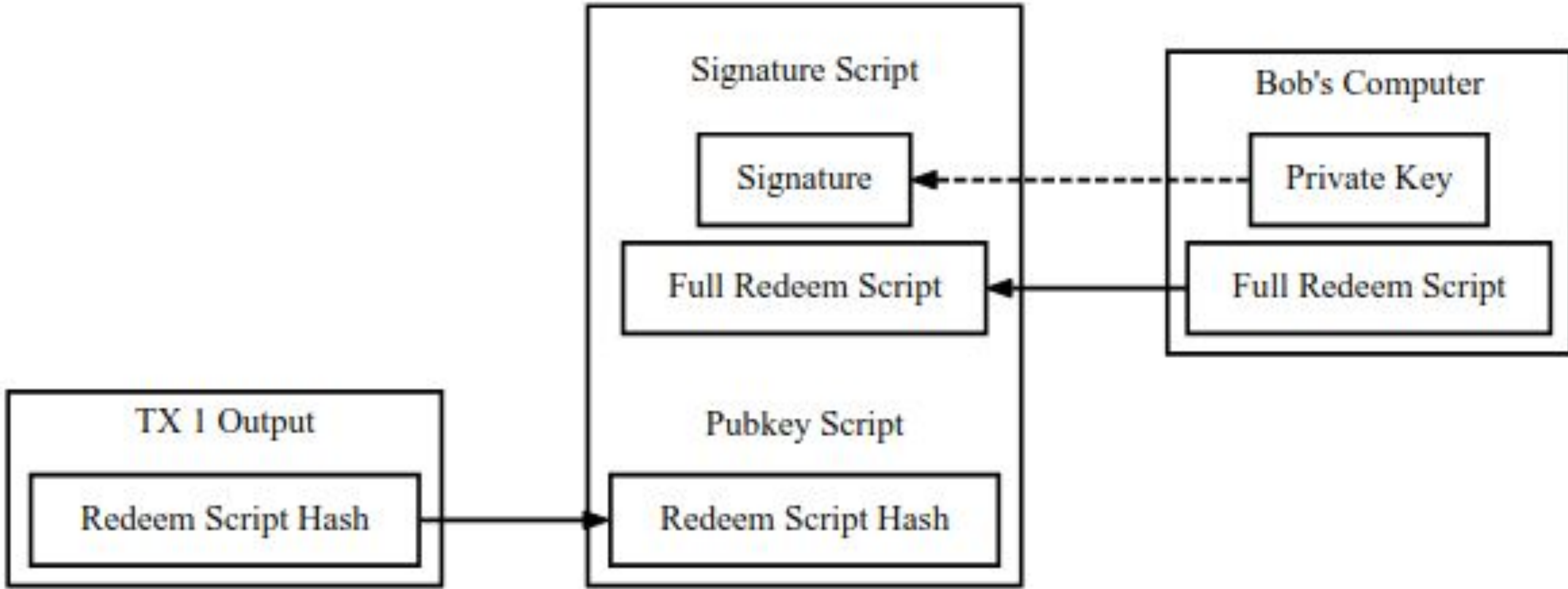
Source: Bitcoin Developer's Guide

AUTHOR: RUSTIE LIN



P2PKH VS P2SH

SPENDING A P2SH OUTPUT



Spending A P2SH Output

Source: Bitcoin Developer's Guide
AUTHOR: RUSTIE LIN



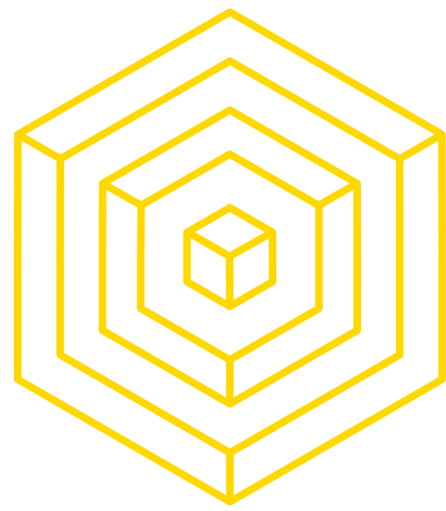
P2PKH VS P2SH

WHY P2SH?

Why Pay-to-Script-Hash?

- Offloads complicated script writing to recipients
- Makes more sense from a **vendor-customer** standpoint
 - Vendor (rather than customer) is responsible for writing correct and secure script
 - Customer doesn't care what the script actually is
- P2SH is the most important improvement to Bitcoin since inception
- Example: **MultiSig**
 - M of N specified signatures can redeem and spend the output of this transaction





P2PKH VS P2SH

MULTISIG EXAMPLE

e.g. Rustie, Gloria and Derrick are in charge of a joint account and make all spending decisions together:

2 of 3 MultiSig

**ScriptSig
(Unlocking Script)**

```
<sig1>
<sig2>
...
<sigm>
```

**ScriptPubKey
(Redeeming Script)**

```
m
<pubKey1>
<pubKey2>
...
N
OP_CHECKMULTISIG
```

**Prev. Output Script
(Locking Script)**

```
OP_HASH160
<hash?>
OP_EQUALVERIFY
```

```
sig1
sig2
```

```
2
PubKey1
PubKey2
PubKey3
3
OP_CHECKMULTISIG
```

```
OP_HASH160
<hash?>
OP_EQUALVERIFY
```

Rustie, Gloria and Derrick provide the script for multisig redemption of coins

Sender of transaction only has to provide this output script!



TRANSACTION LOCKTIME

TIMELOCKS

- Extend bitcoin scripting into the dimension of time.
- **Absolute and relative timelocks**
 - **Absolute timelocks** specify UNIX timestamp
 - **Relative timelocks** specify block height
- **Transaction-level and script-level timelocks**
 - **Transaction-level:** the transaction itself will be postponed until the specified time
 - **UTXO-level:** the locking script restricts use of specific UTXOs

