Bitcoin Mechanics

Transaction and script

Recap

(1) <u>SHA256</u>: a collision resistant hash function that outputs 32-byte hash values

Applications:

- a binding commitment to one value: commit$(m) \to$ H(m)
  or to a list of values: commit$(m_1, \ldots, m_n) \to$ Merkle$(m_1, \ldots, m_n)$
- Proof of work with difficulty D

Recap

(2) Digital signatures: (Gen, Sign, Verify)

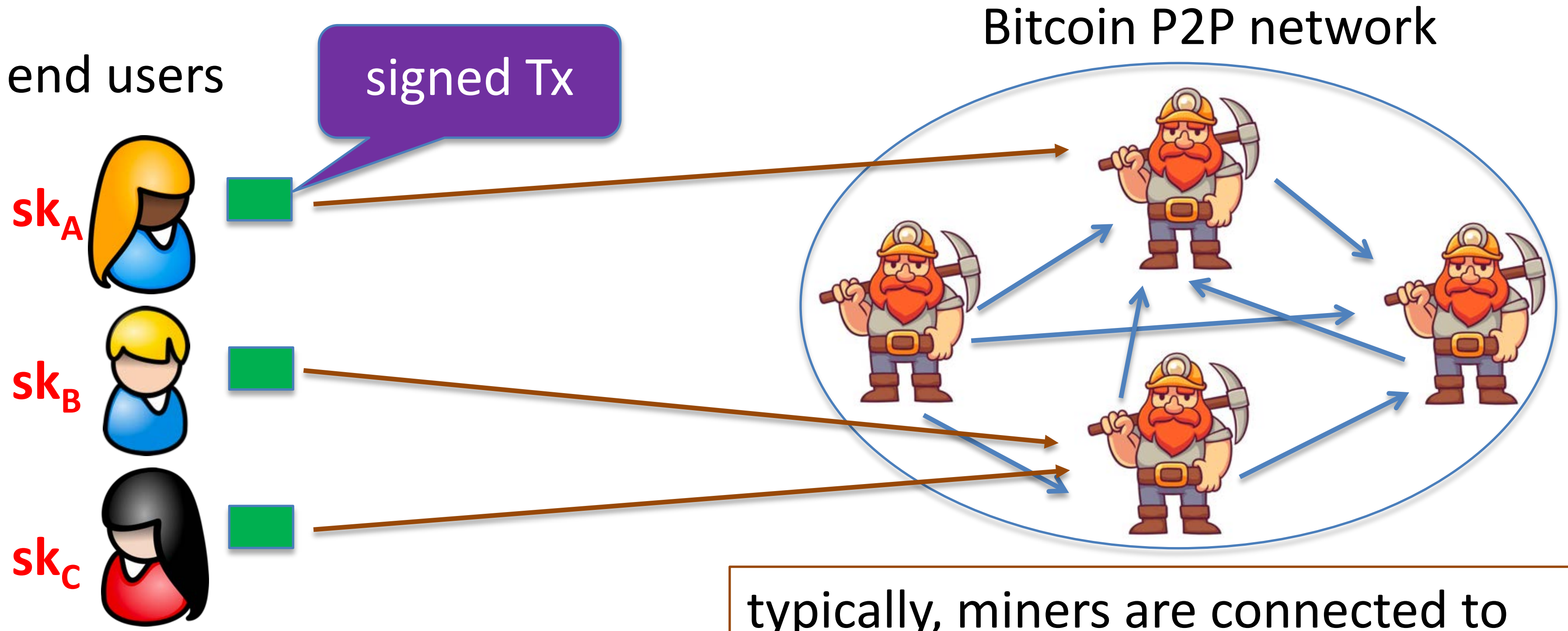
Gen() → (pk, sk),

Sign(sk, m) → σ, Verify(pk, m, σ) → accept/reject

signing key

verification key

First: overview of the Bitcoin consensus layer



typically, miners are connected to eight other peers (anyone can join)

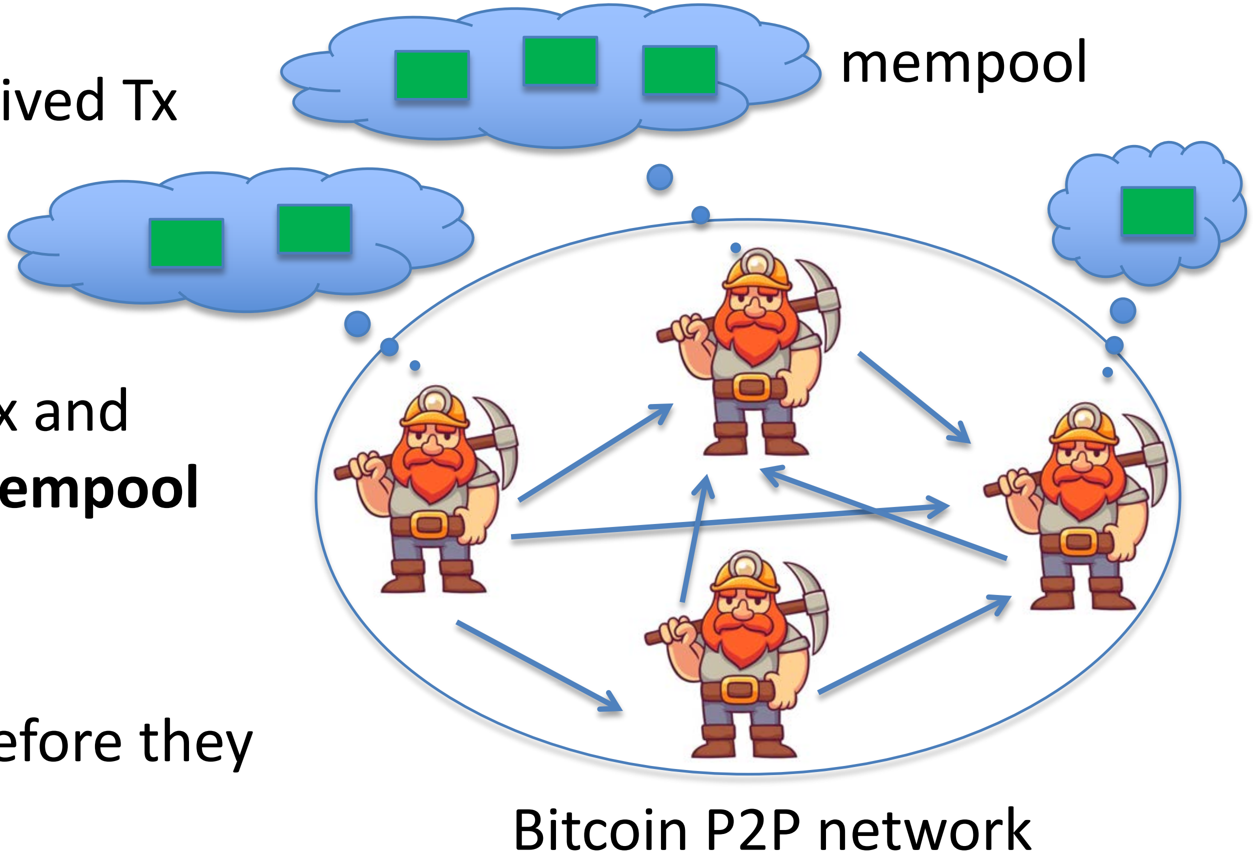
First: overview of the Bitcoin consensus layer

miners broadcast received Tx to the P2P network

every miner:

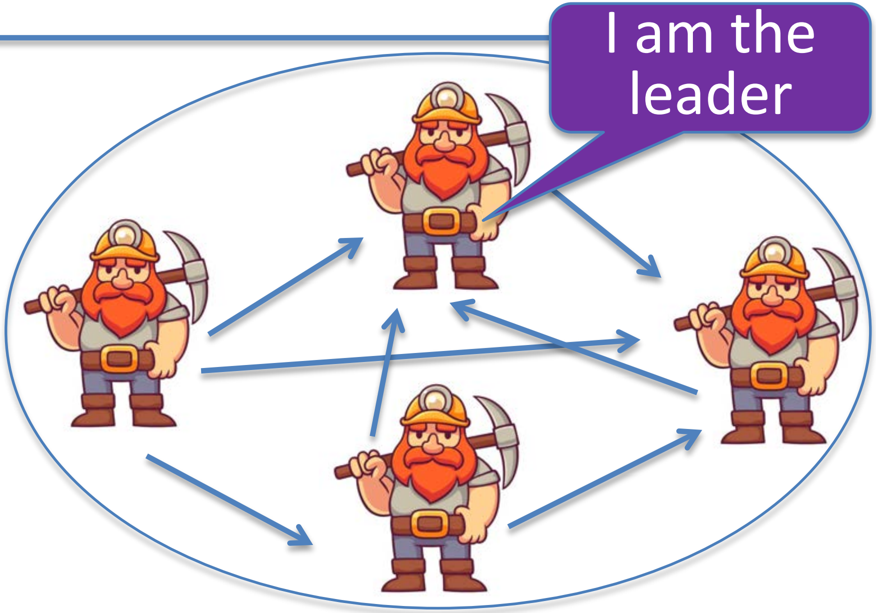
  validates received Tx and stores them in its **mempool** (unconfirmed Tx)

note: miners see Tx before they are posted on chain



First: overview of the Bitcoin consensus layer

blockchain

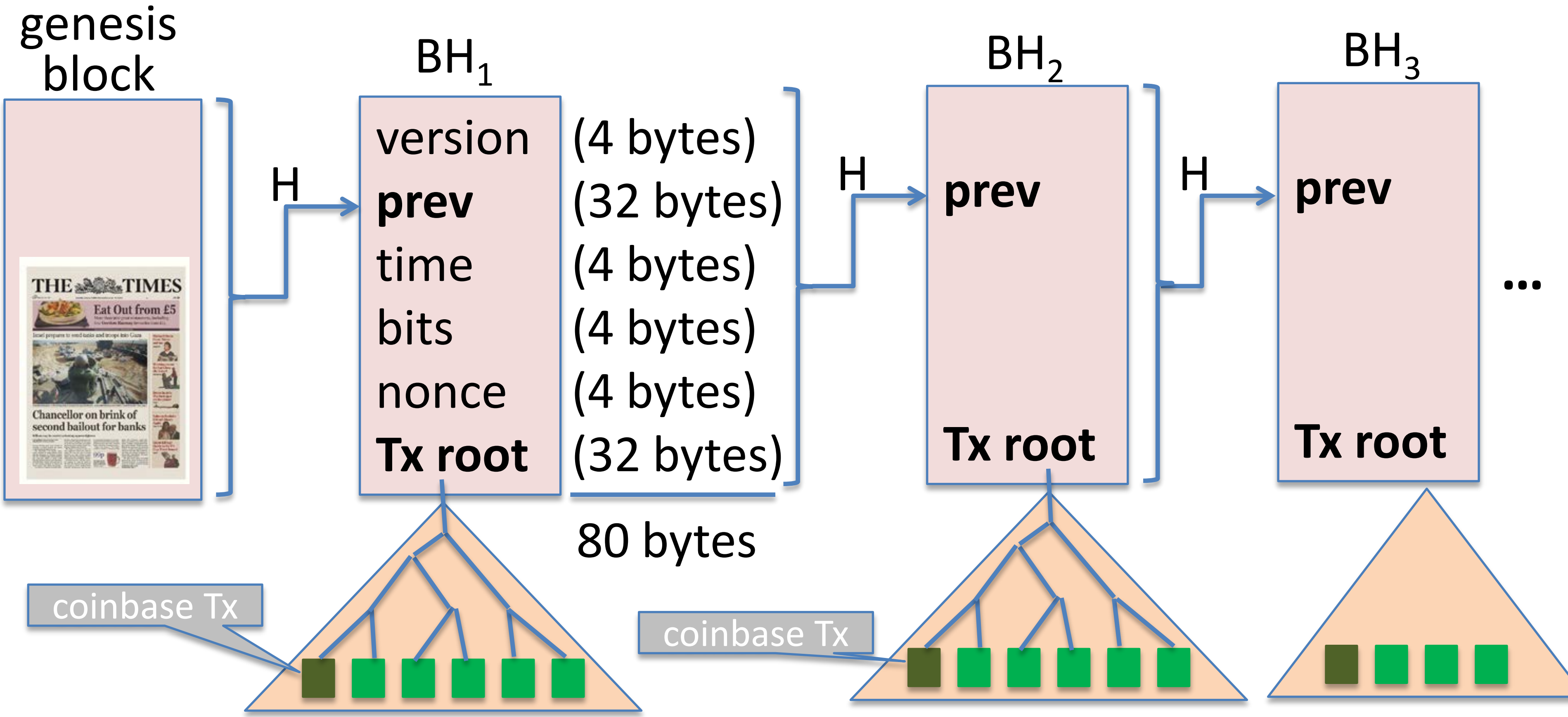


Bitcoin P2P network

Every **10 minutes**:

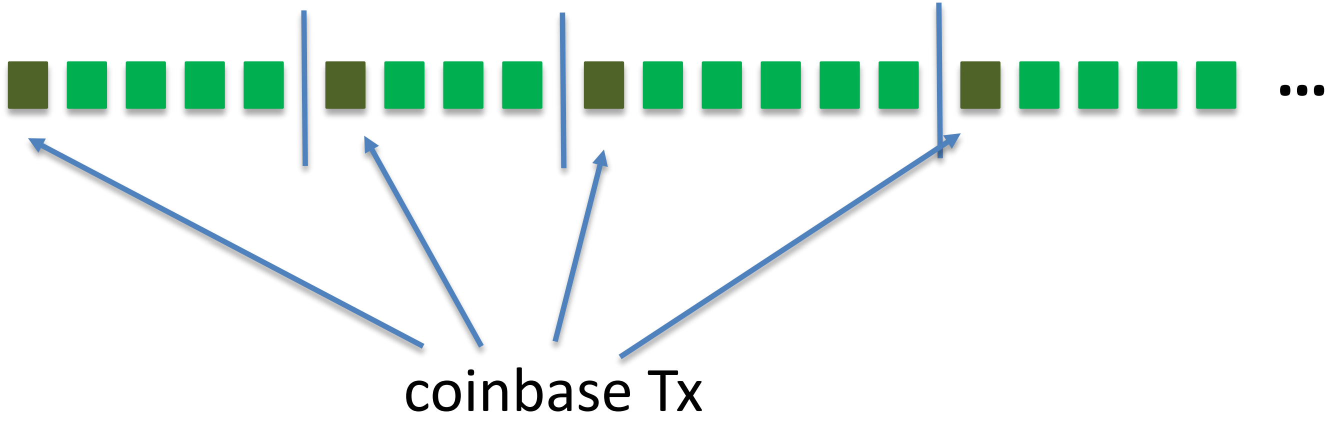
- Each miner creates a candidate block from Tx in its mempool
- a “random” miner is selected (how: next week), and broadcasts its block to P2P network
- all miners validate new block

Bitcoin blockchain: a sequence of block headers, 80 bytes each



This lecture

View the blockchain as a sequence of Tx (append-only)



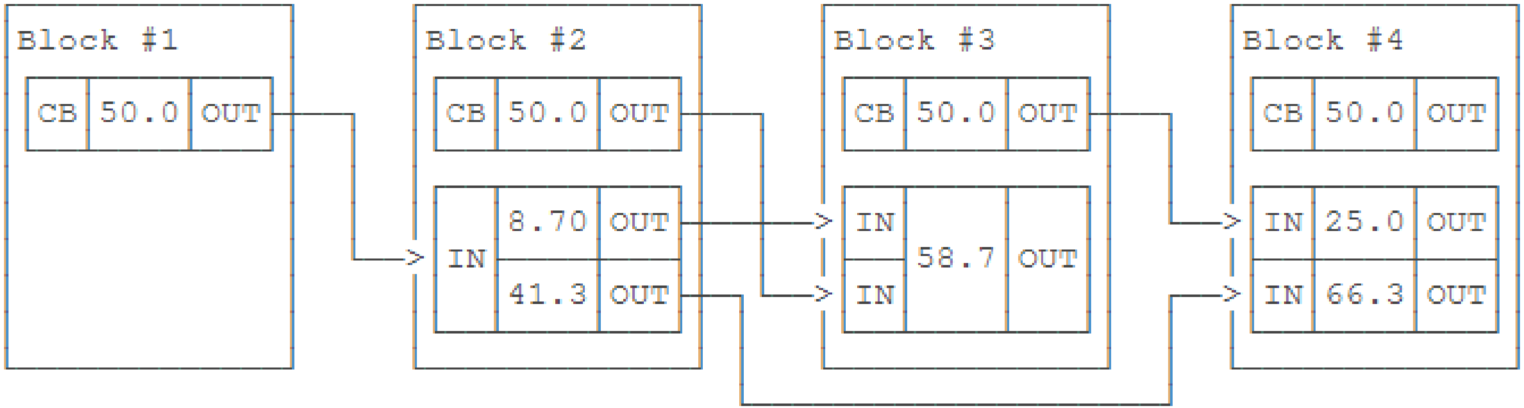
Bitcoin Transaction

- An account-based ledger
  - Like bank, Alipay

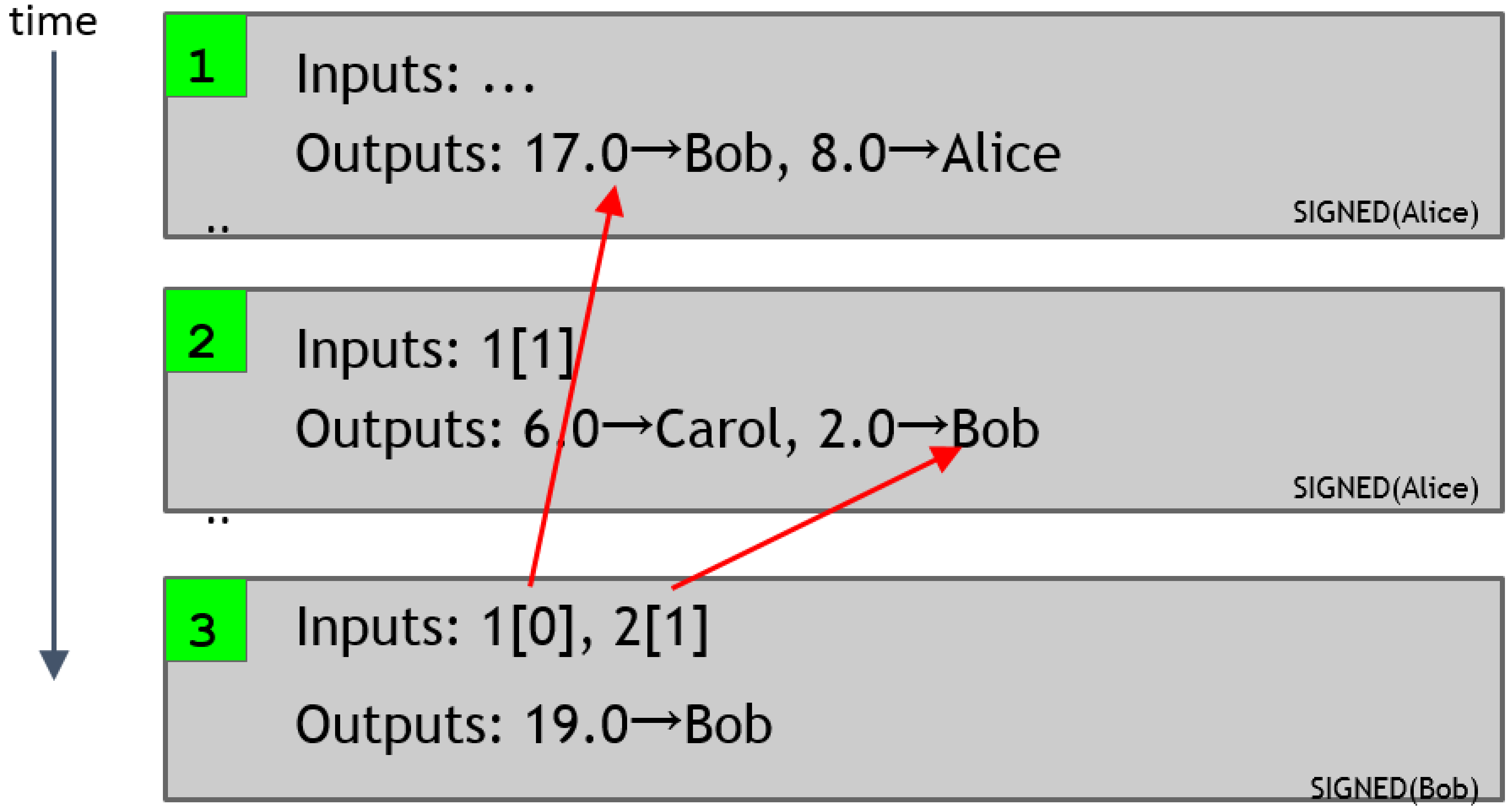
Create 25 coins and credit to Alice$_{\text{ASSERTED BY MINERS}}$
Transfer 17 coins from Alice to Bob$_{\text{SIGNED(Alice)}}$
Transfer 8 coins from Bob to Carol$_{\text{SIGNED(Bob)}}$
Transfer 5 coins from Carol to Alice$_{\text{SIGNED(Carol)}}$
Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$

Bitcoin Transaction

- Input & Output



1	Inputs: Ø Outputs: 25.0→Alice
2	Inputs: 1[0] Outputs: 17.0→Bob, 8.0→Alice SIGNED(Alice)
3	Inputs: 2[0] Outputs: 8.0→Carol, 9.0→Bob SIGNED(Bob)
4	Inputs: 2[1] Outputs: 6.0→David, 2.0→Alice SIGNED(Alice)


time
1
Inputs: ...
Outputs: 17.0→Bob, 8.0→Alice
SIGNED(Alice)
2
Inputs: 1[1]
Outputs: 6.0→Carol, 2.0→Bob
SIGNED(Alice)
3
Inputs: 1[0], 2[1]
Outputs: 19.0→Bob
SIGNED(Bob)

Tx structure (non-coinbase)

inputs	input[0] input[1] input[2]
outputs	output[0] output[1]
(segwit)	witnesses
(4 bytes)	locktime

TxID = H(Tx)

(excluding witnesses)

earliest block # that can include Tx

input:

TxID	32 byte hash
out-index	4 byte index
ScriptSig	program
seq	ignore

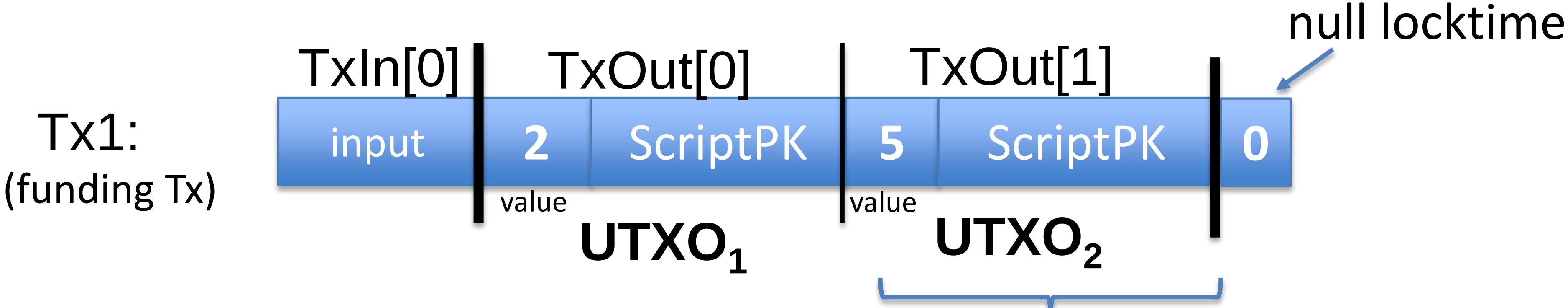
output:

value	8 bytes
ScriptPK	program

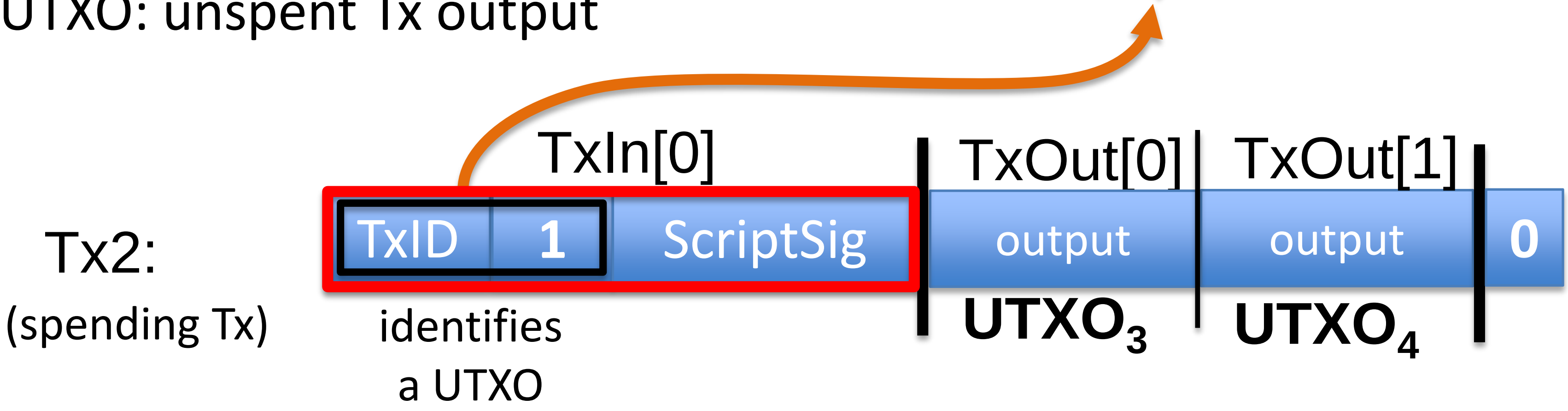
value = #BTC/10^8

$[10^{-8}, \ldots, 2^{37}]$

Example



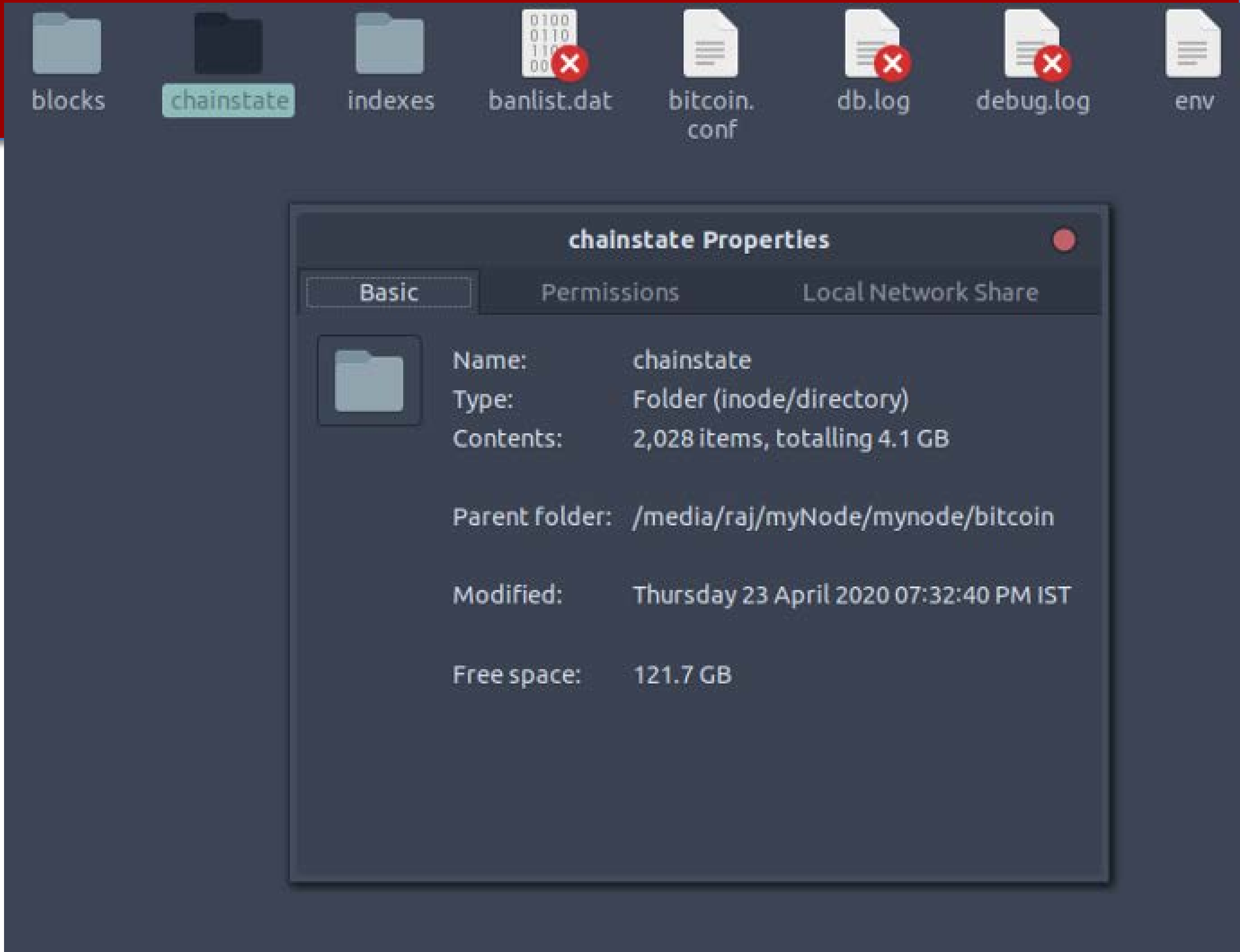
UTXO: unspent Tx output



UTXO Set

- UTXO set and is being constantly maintained by every Bitcoin node
- Technically they are known as the *chainstate* and are stored in the chainstate data directory of a node
- The chainstate updates every time a new block is accepted in the blockchain

- UTXOs size



blocks
chainstate
indexes
banlist.dat
bitcoin.conf
db.log
debug.log
env
chainstate Properties
Basic
Permissions
Local Network Share
Name: chainstate
Type: Folder (inode/directory)
Contents: 2,028 items, totalling 4.1 GB
Parent folder: /media/raj/myNode/mynode/bitcoin
Modified: Thursday 23 April 2020 07:32:40 PM IST
Free space: 121.7 GB

UTXO

- Advantage:
  - Scalability: process multiple UTXOs in parallel
  - Private: as long as a user uses new address for each transaction
- Disadvantage:
  - Not intuitive
  - Not efficient

Transaction Script

- *Metadata .*
  - *the size of the transaction,*
  - *the number* of inputs, and the number of outputs. T
  - the hash of the entire transaction which serves as a unique ID for the transaction
- *Inputs.*
  - *The transaction inputs form an array, and each input has the same form.*
  - User needs to sign to show he/she has the ability to claim those previous transaction outputs.
- *Outputs.*
  - *The outputs are again an array. Each output has just two fields.* the sum of all the output values has to be less than or equal to the sum of all the input values
  - the difference is a transaction fee to the miner who publishes this transaction.

metadata

input(s)

output(s)


```
{
    "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":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
        "n":0
      },
      "scriptSig":"3f3a4ce81...."
     }
    ],
    "out":[
     {
      "value":"10.12287097",
      "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
     }
    ]
}
```

Validating Tx2

program from funding Tx: under what conditions can UTXO be spent

Miners check (for each input):

1. The program $\boxed{\text{ScriptSig | ScriptPK}}$ returns true
2. $\boxed{\text{TxID | index}}$ is in the current UTXO set
3. sum input values ≥ sum output values

After Tx2 is posted, miners remove $UTXO_2$ from UTXO set

Bitcoin Script

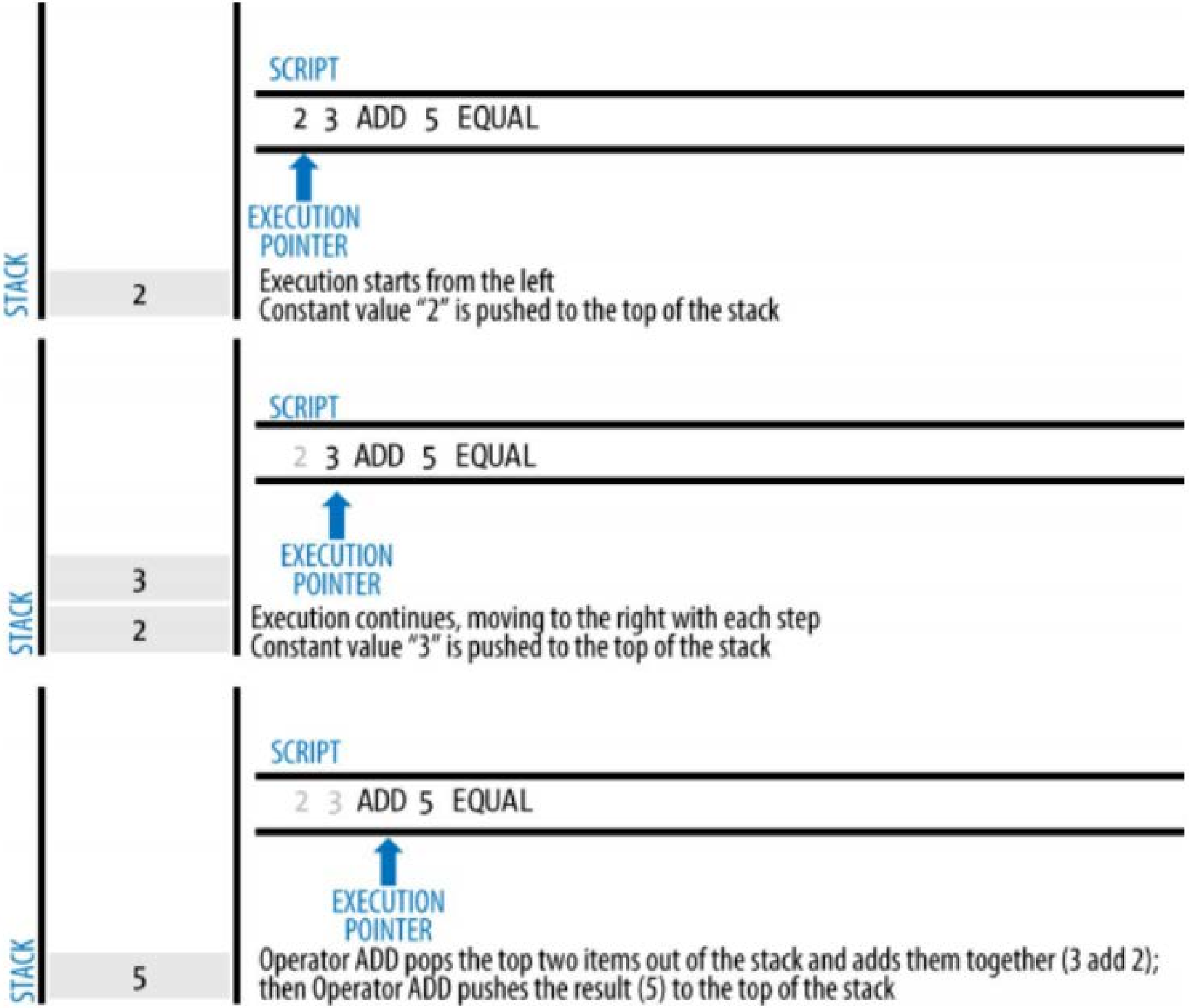
- scriptPubkey is served as a lock
  - Output needs to specify who can use the output; and ensures that only the legitimate user can actually use it
  - Scriptsig in fact is the key to the lock

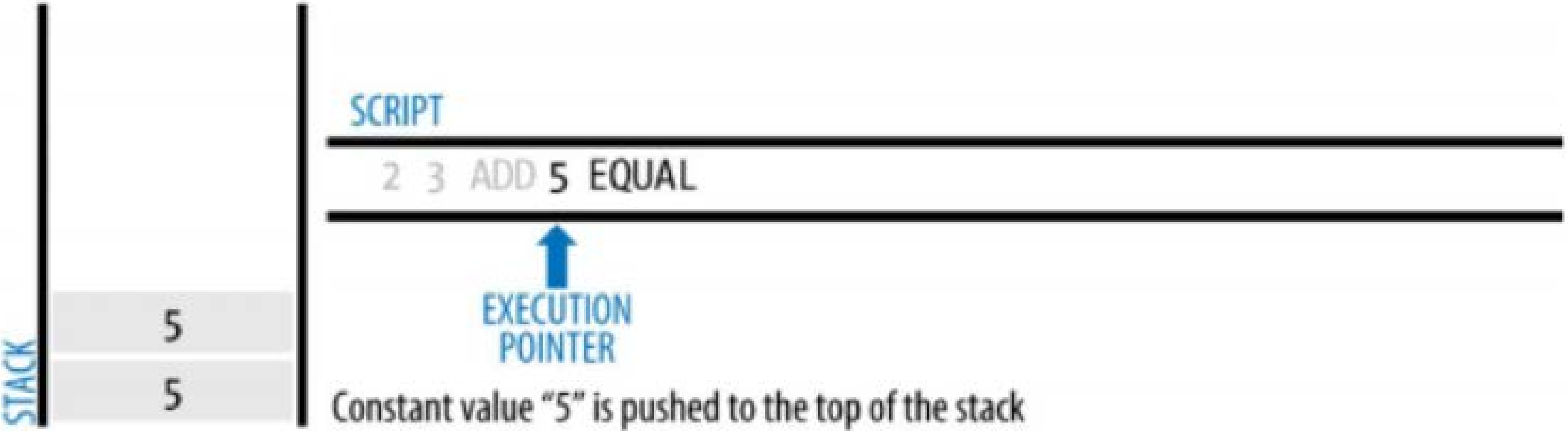
Bitcoin script

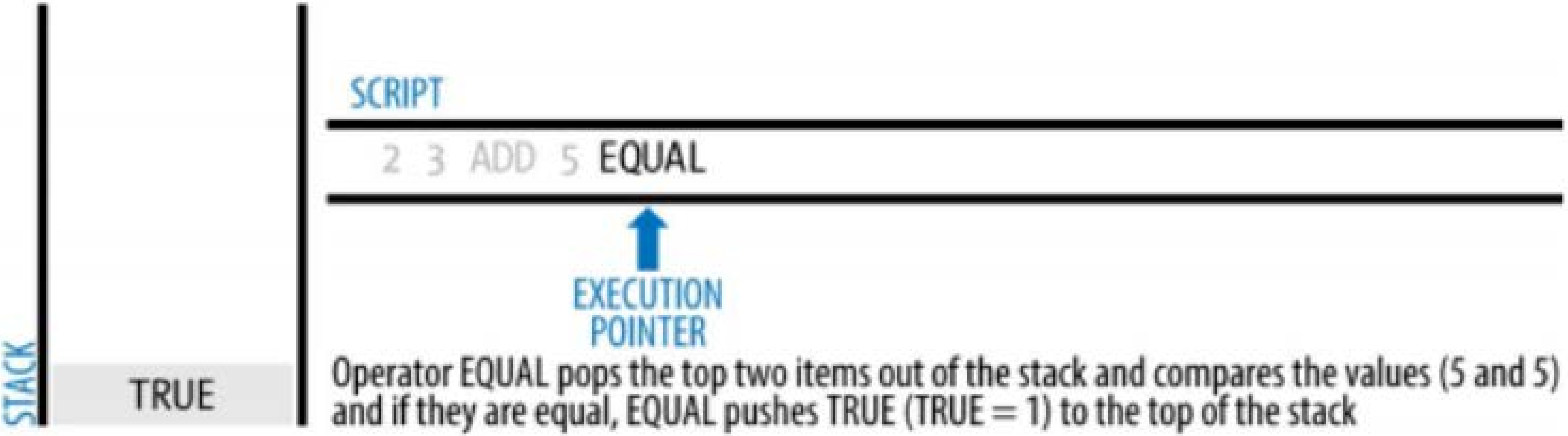
- Stack machine
  - An example: the calculator
  - Infix expression: 1+((2+3)*4)-5
  - To use a stack, postfix expression
    - 123+4*+5-

Bitcoin Script

- Anna will give 1 btc to any one who knows the answer to 5-3
  - She will create a transaction, sets a challenge in the output script
  - 3 OP_ADD 5 OP_EQUAL
  - Anyone can use the output by supplying 2 in the scriptsig


SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
2
Execution starts from the left
Constant value "2" is pushed to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
3
2
Execution continues, moving to the right with each step
Constant value "3" is pushed to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
5
Operator ADD pops the top two items out of the stack and adds them together (3 add 2);
then Operator ADD pushes the result (5) to the top of the stack


SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
5
5
Constant value "5" is pushed to the top of the stack


SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
TRUE
Operator EQUAL pops the top two items out of the stack and compares the values (5 and 5) and if they are equal, EQUAL pushes TRUE (TRUE = 1) to the top of the stack

Bitcoin Script

- Practice
  - 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
  - What would be the sigscript?

Bitcoin Script

- Practice
- OP_HASH256 6fe28c0ab6f1b372c1a6a246ae63f74f931e8365e15a089c68d6190000000000 OP_EQUAL

Focusing on Tx2: TxInp[0]

from UTXO
(Bitcoin script)

Value	0.05000000 BTC
Pkscript	OP_DUP OP_HASH160 45b21c8a0cb687d563342b6c729d31dab58e3a4e OP_EQUALVERIFY OP_CHECKSIG
Sigscript	304402205846cace0d73de82dfbdeba4d65b9856d7c1b1730eb401cf4906b2401a69bdc90220589d36d36be64e774c8796b96c011f29768191abeb7f56ba20ffb0351280860c01 03557c228b080703d52d72ead1bd93fc72f45c4ddb4c2b7a20c458e2d069c8dd9e

from TxInp[0]

Bitcoin Script

A stack machine. Not Turing Complete: no loops.

Quick survey of op codes:

1. **OP_TRUE** (OP_1), **OP_2**, …, **OP_16**: push value onto stack

   81 82 96

2. **OP_DUP**: push top of stack onto stack

   118

Bitcoin Script

3. control:

99 **OP_IF** <statements> **OP_ELSE** <statements> **OP_ENDIF**

105 **OP_VERIFY**: abort fail if top = false

106 **OP_RETURN**: abort and fail

what is this for? ScriptPK = [OP_RETURN, <data>]

136 **OP_EQVERIFY**: pop, pop, abort fail if not equal

Bitcoin Script

4. arithmetic:

   OP_ADD, **OP_SUB**, **OP_AND**, …: pop two items, add, push

5. crypto:

   OP_SHA256: pop, hash, push

   OP_CHECKSIG: pop sig, pop pk, verify sig. on Tx, push 0 or 1

6. Time: **OP_CheckLockTimeVerify** (CLTV):

   fail if value at the top of stack > Tx locktime value.

   usage: UTXO can specify min-time when it can be spent

Bitcoin Script

```
<sig>
<pubKey>
---------------
OP_DUP
OP_HASH160
<pubKeyHash?>
OP_EQUALVERIFY
OP_CHECKSIG
```

Example: a common script

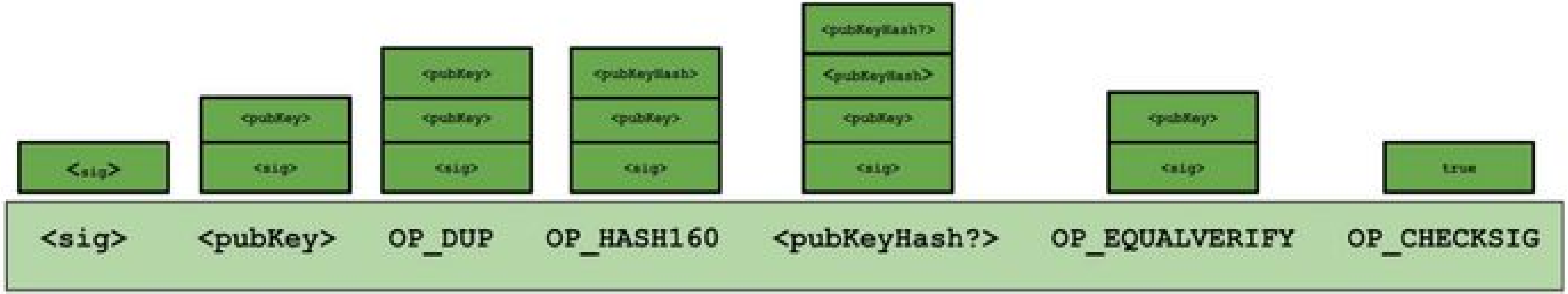
<sig> <pk> **DUP HASH256** <pkhash> **EQVERIFY CHECKSIG**

stack:	empty	init
	<sig> <pk>	push values
	<sig> <pk> <pk>	**DUP**
	<sig> <pk> <hash>	**HASH256**
	<sig> <pk> <hash> <pkhash>	push value
	<sig> <pk>	**EQVERIFY**
	1	**CHECKSIG**
		verify(pk, Tx, sig)

⇒ successful termination

Bitcoin Script

- The validation



Transaction types: (1) P2PKH

pay to public key hash

Alice wants to pay Bob 5 BTC:

- step 1: Bob generates sig key pair $(pk_B, sk_B) \leftarrow \text{Gen}()$
- step 2: Bob computes his Bitcoin address as $Addr_B \leftarrow H(pk_B)$
- step 3: Bob sends $Addr_B$ to Alice
- step 4: Alice creates Tx:

input	5	$ScriptPK_B$	2	$ScriptPK_A$	0
7 BTC	UTXO$_B$ for Bob		UTXO$_A$ for Alice (change)		

$ScriptPK_B$: DUP HASH256 **<$Addr_B$>** EQVERIFY CHECKSIG

Transaction types: (1) P2PKH

Later, when Bob wants to spend his UTXO: create a Tx_{spend}

$ScriptSig_B$: <sig> <pk_B>

<sig> = Sign(sk_B, Tx) where Tx = (Tx_{spend} excluding all ScriptSigs) (SIGHASH_ALL)

Miners validate that $\boxed{ScriptSig_B \mid ScriptPK_B}$ returns true

P2PKH: comments

- Alice specifies recipient's pk in $UTXO_B$

- Recipient's pk is not revealed until UTXO is spent

  (some security against attacks on pk)

- Miner cannot change <$Addr_B$> and steal funds:

  invalidates the signature that created the $UTXO_B$

Segregated Witness

ECDSA malleability:

- given (m, sig) anyone can create (m, sig') with sig ≠ sig'

⇒ miner can change sig in Tx and change TxID = SHA256(Tx)

⇒ Tx issuer cannot tell what TxID is, until Tx is posted

⇒ leads to problems and attacks

Segregated witness: signature is moved to witness field in Tx

TxID = Hash(Tx without witnesses)

Segregated Witness

- **Malleability**
  - **A gold coin got hammered, so it is not round any more; will this gold coin be used later?**
- Transaction Malleability
  - The signature of the transaction is modified a little; however, it is still a valid signature
  - Without accessing private key
  - Due to many reasons:
    - One example, OpenSSL verifies the signature not strictly
- The consequence
  - Txid will be changed

Segregated Witness

- mtgox attack:
  - An attacker applies an account in an exchange center; and deposit bitcoins in it
  - The attacker then apply a withdraw; the exchange center will initiate a transaction
  - The transaction will be broadcast to the network; but before the transaction is confirmed in the network, the attacker received the transaction and slightly modifies the scriptsig, generate a new transaction(still valid); and broadcast to the network

Segregated Witness

- After the hacker’s new transaction is in the blockchain (the hacker can use the bitcoin now and the original transaction will be regarded as a double-spending), he would file a complain to the exchange center, saying he hasn’t received the bitcoin yet
- The exchange center will check the blockchain with the original txid, which indeed is not included, so the exchange center will repay the hacker

Segragated witness

- **Segregated Witness**, or **SegWit**, is the process by which the block size limit on a blockchain is increased by removing signature data from transactions that are included in each block.
  - Originally, there was no limit to the size of blocks. However, this allowed malicious actors to make up fake "block" data that was very long as a form of DoS attack
  - Block is constrained to a max size of one megabyte

Segragated witness

- Digital signature accounts for 65% of the space in a given transaction
- Segwit ignores the signature, therefore increase the one MB limit for block sizes to a little under four MB

Transaction types: (2) P2SH: pay to script hash

(pre SegWit in 2017)

Let’s payer specify a redeem script (instead of just pkhash)

Usage: payee publishes hash(redeem script) ⟵ Bitcoint addr.

payer sends funds to that address

ScriptPK in UTXO: HASH160 <H(redeem script)> EQUAL

ScriptSig to spend: <sig$_1$> <sig$_2$> … <sig$_n$> <redeem script>

payer can specify complex conditions for when UTXO can be spent

P2SH

Miner verifies:

(1) <ScriptSig> ScriptPK = true ⟵ payee gave correct script

(2) ScriptSig = true ⟵ script is satisfied

Example P2SH: multisig

<u>Goal</u>: spending a UTXO requires t-out-of-n signatures

Redeem script for 2-out-of-3: (set by payer)

<2> <PK$_1$> <PK$_2$> <PK$_3$> <3> CHECKMULTISIG

hash gives P2SH address

ScriptSig to spend: (by payee) <0> <sig1> <sig3> <redeem script>

END OF LECTURE