# Bitcoin Mechanics

Dan Boneh

Reminder: proj #1 is posted on the course web site.   Due Oct. 3
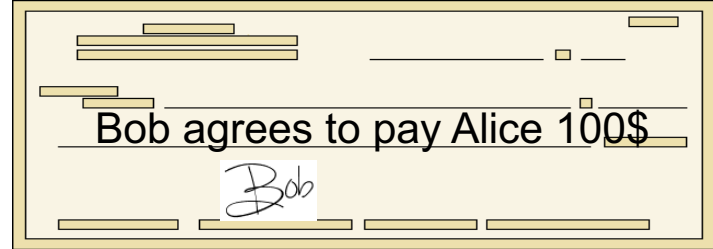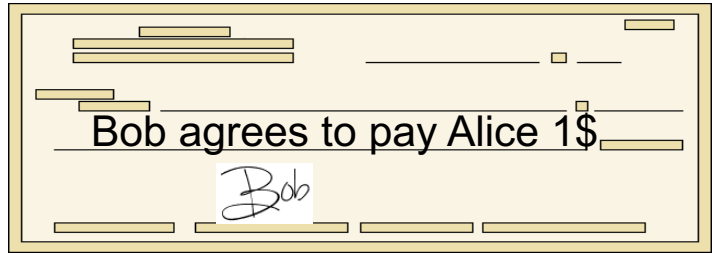
# Recap

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

**Applications**:

- a binding commitment to one value:   $\text{commit}(m) \twoheadrightarrow \text{H}(m)$
  or to a list of values:   $\text{commit}(m_1, \dots, m_n) \twoheadrightarrow \text{Merkle}(m_1, \dots, m_n)$

- Proof of work with difficulty D:
       given $x$ find $y$   s.t.   $H(x, y) < 2^{256}/D$   takes time $O(D)$

# Digital Signatures

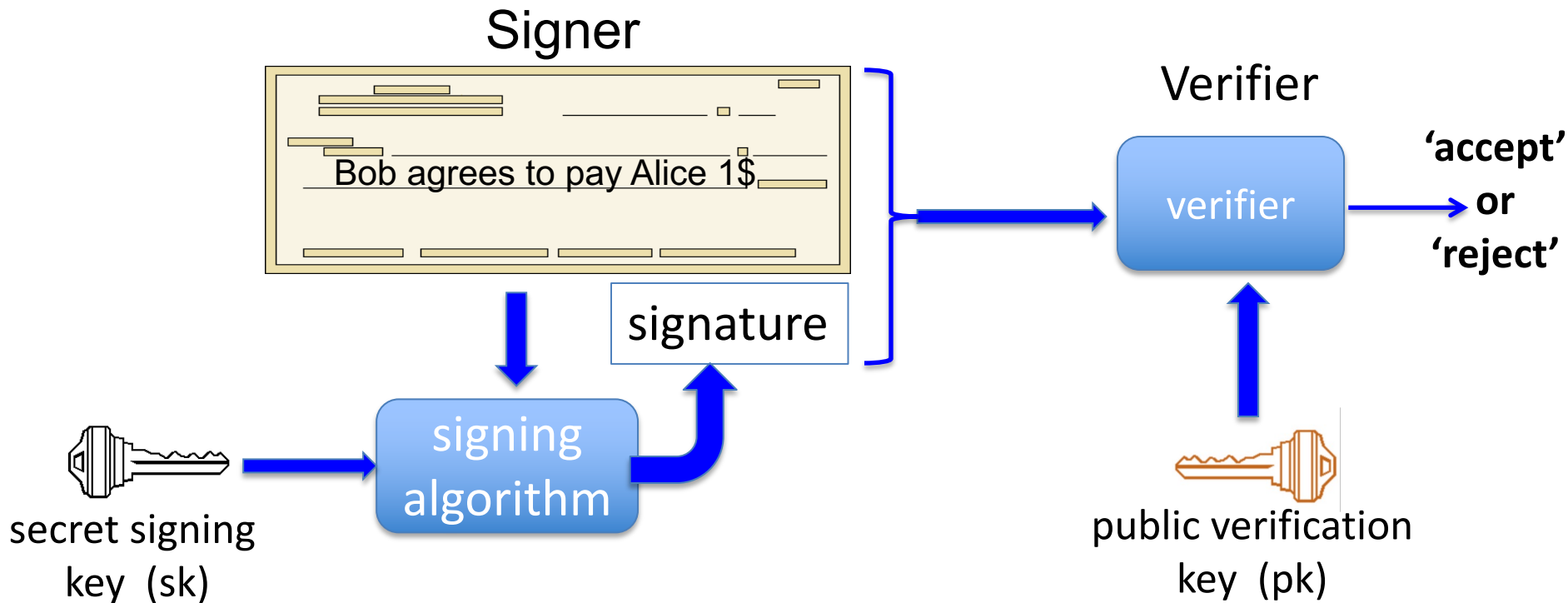Physical signatures: bind transaction to author



Problem in the digital world:

anyone can copy Bob's signature from one doc to another

# Digital signatures

Solution: make signature depend on document

# Digital signatures:   syntax

<u>**Def**</u>:    a signature scheme is a triple of algorithms:

- **Gen**():  outputs a key pair    (pk, sk)

- **Sign**(sk, msg)  outputs sig.  σ

- **Verify**(pk, msg, σ)  outputs 'accept'  or  'reject'

<u>**Secure signatures**</u>:   (informal)

Adversary who sees signatures **on many messages** of his choice, cannot forge a signature on a new message.

# Families of signature schemes

1.  <u>RSA signatures (old … not used in blockchains)</u>:

    - long sigs and public keys (≥256 bytes),    fast to verify

2.  <u>Discrete-log signatures</u>:   Schnorr and  ECDSA      (Bitcoin, Ethereum)

    - short sigs (48 or 64 bytes) and public key (32 bytes)

3.  <u>BLS signatures</u>:  48 bytes,   aggregatable,   easy threshold

    (Ethereum 2.0, Chia, Dfinity)

4.  <u>Post-quantum</u> signatures:    long  (≥600 bytes)

    details in CS255

# Signatures on the blockchain

Signatures are used everywhere:

- ensure Tx authorization,

- governance votes,

- consensus protocol votes.
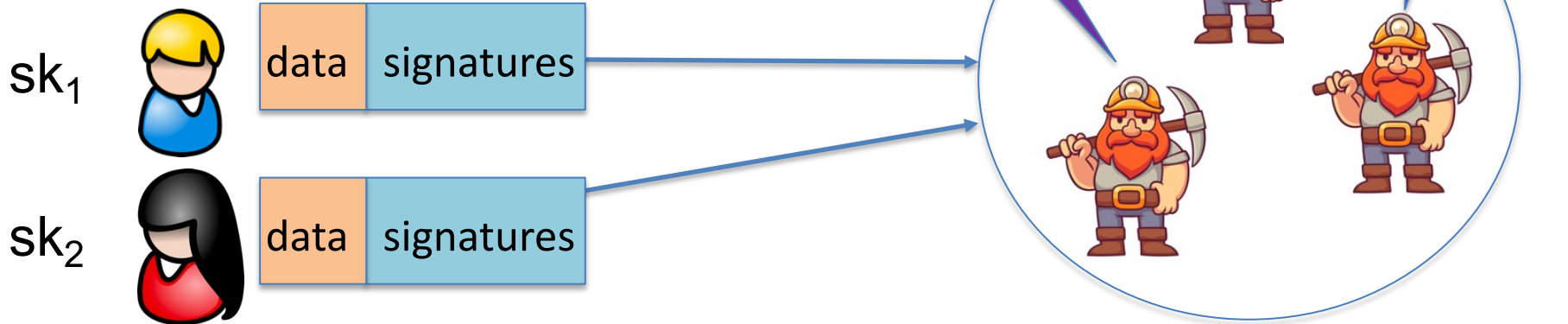
# In summary ...

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

Gen() $\rightarrow$ (pk, sk),

Sign(sk, m) $\rightarrow$ σ,       Verify(pk, m, σ) $\rightarrow$ accept/reject

signing key

verification key

# Bitcoin mechanics

# This lecture: Bitcoin mechanics

Oct. 2008: paper by Satoshi Nakamoto
Jan. 2009: Bitcoin network launched

Total market value:

Sep. 2022: $386B
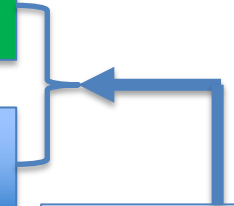
# First: overview of the Bitcoin consensus layer

mempool

miners broadcast received Tx to the P2P network

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

note: miners see all Tx before they are posted on chain

Bitcoin P2P network

blockchain

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

I am the leader

Bitcoin P2P network

blockchain

Selected miner is paid 6.25 BTC
in **coinbase Tx**  (first Tx in the block)

- only way new BTC is created

- block reward halves every four years

    ⇒ max 21M BTC  (currently 19.1M BTC)

6.25 BTC
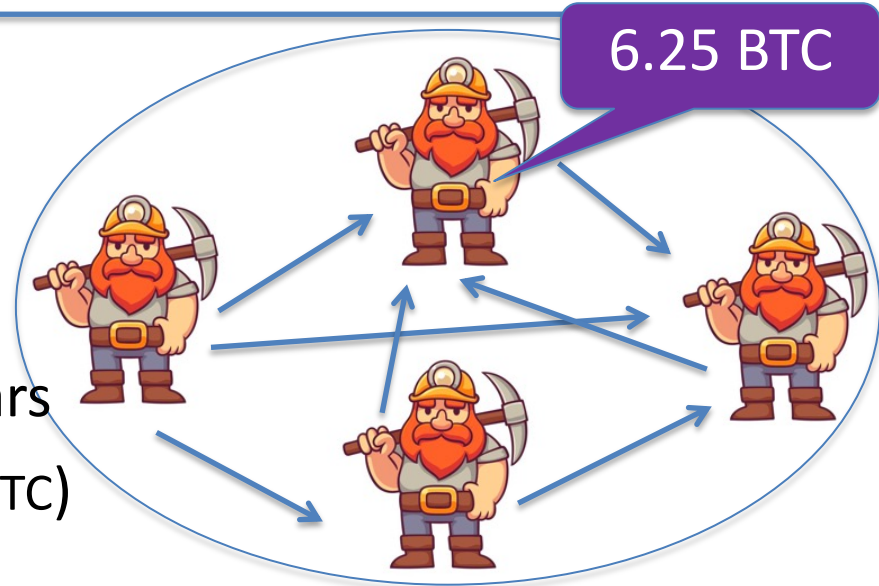
note:  miner chooses order of Tx in block

# Properties (very informal)

Next week:

**Safety / Persistence**:

- to remove a block, need to convince 51% of mining power *

**Liveness**:

- to block a Tx from being posted, need to convince 51% of mining power **

  (some sub 50% censorship attacks, such as feather forks)

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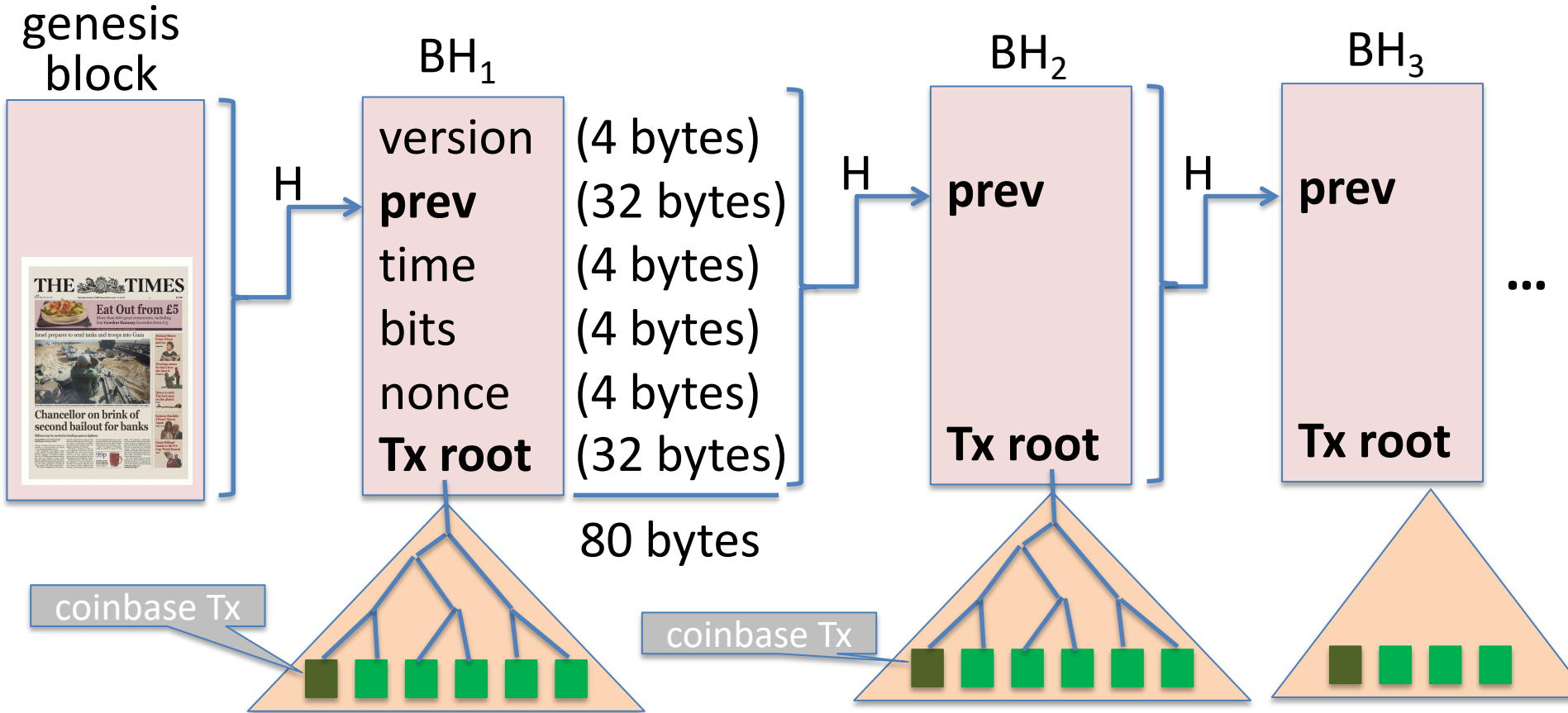
# Bitcoin blockchain:  a sequence of block headers, 80 bytes each

**time**:   time miner assembled the block.   Self reported.
                (block rejected if too far in past or future)

**bits**:  proof of work difficulty
**nonce**:  proof of work solution      } for choosing a leader (next week)

**Merkle tree**:  payer can give a short proof that Tx is in the block

new block every ≈10 minutes.

# An example (Sep. 2020)

Tx data

| Height | Mined | Miner | Size | #Tx |
|--------|-------|-------|------|-----|
| 648494 | 17 minutes | Unknown | 1,308,663 bytes | 1855 |
| 648493 | 20 minutes | SlushPool | 1,317,436 bytes | 2826 |
| 648492 | 59 minutes | Unknown | 1,186,609 bytes | 1128 |
| 648491 | 1 hour | Unknown | 1,310,554 bytes | 2774 |
| 648490 | 1 hour | Unknown | 1,145,491 bytes | 2075 |
| 648489 | 1 hour | Poolin | 1,359,224 bytes | 2622 |

# Block 648493

| | |
|---|---|
| Timestamp | 2020-09-15 17:25 |
| Height | 648493 |
| Miner | SlushPool    (from coinbase Tx) |
| Number of Transactions | 2,826 |
| Difficulty   (D) | 17,345,997,805,929.09    (adjusts every two weeks) |
| Merkle root | 350cbb917c918774c93e945b960a2b3ac1c8d448c2e67839223bbcf595baff89 |
| Transaction Volume | 11256.14250596 BTC |
| Block Reward | 6.25000000 BTC |
| Fee Reward | 0.89047154 BTC    (Tx fees given to miner in coinbase Tx) |

# This lecture

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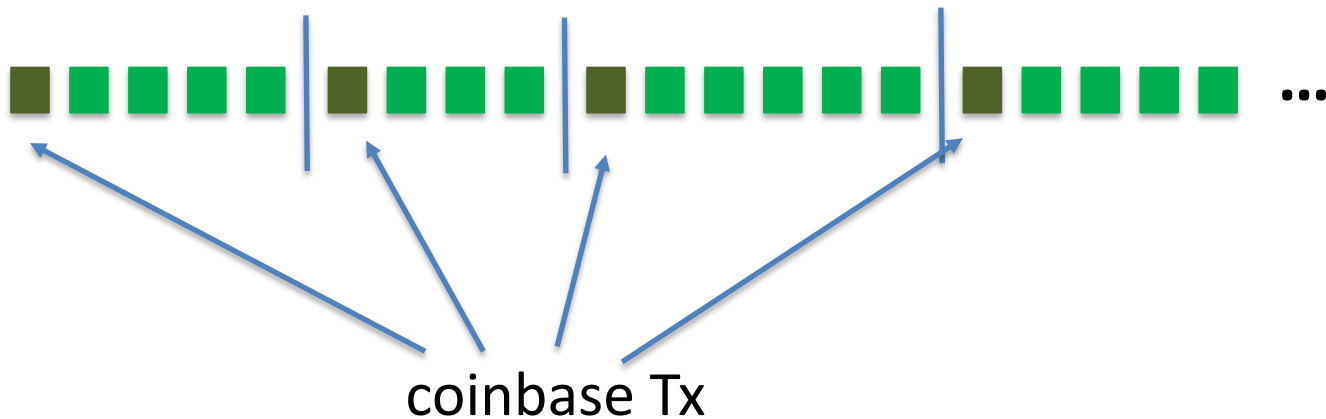


coinbase Tx

# Tx structure   (non-coinbase)

inputs

| input[0] |
| input[1] |
| input[2] |

outputs

| output[0] |
| output[1] |

(segwit)

witnesses
(part of input)

(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 |

#BTC = value/$10^8$

# Example

# Example



Tx1:
(funding Tx)

TxIn[0]  TxOut[0]  TxOut[1]  null locktime

input  2  ScriptPK  5  ScriptPK  0

$UTXO_1$  $UTXO_2$
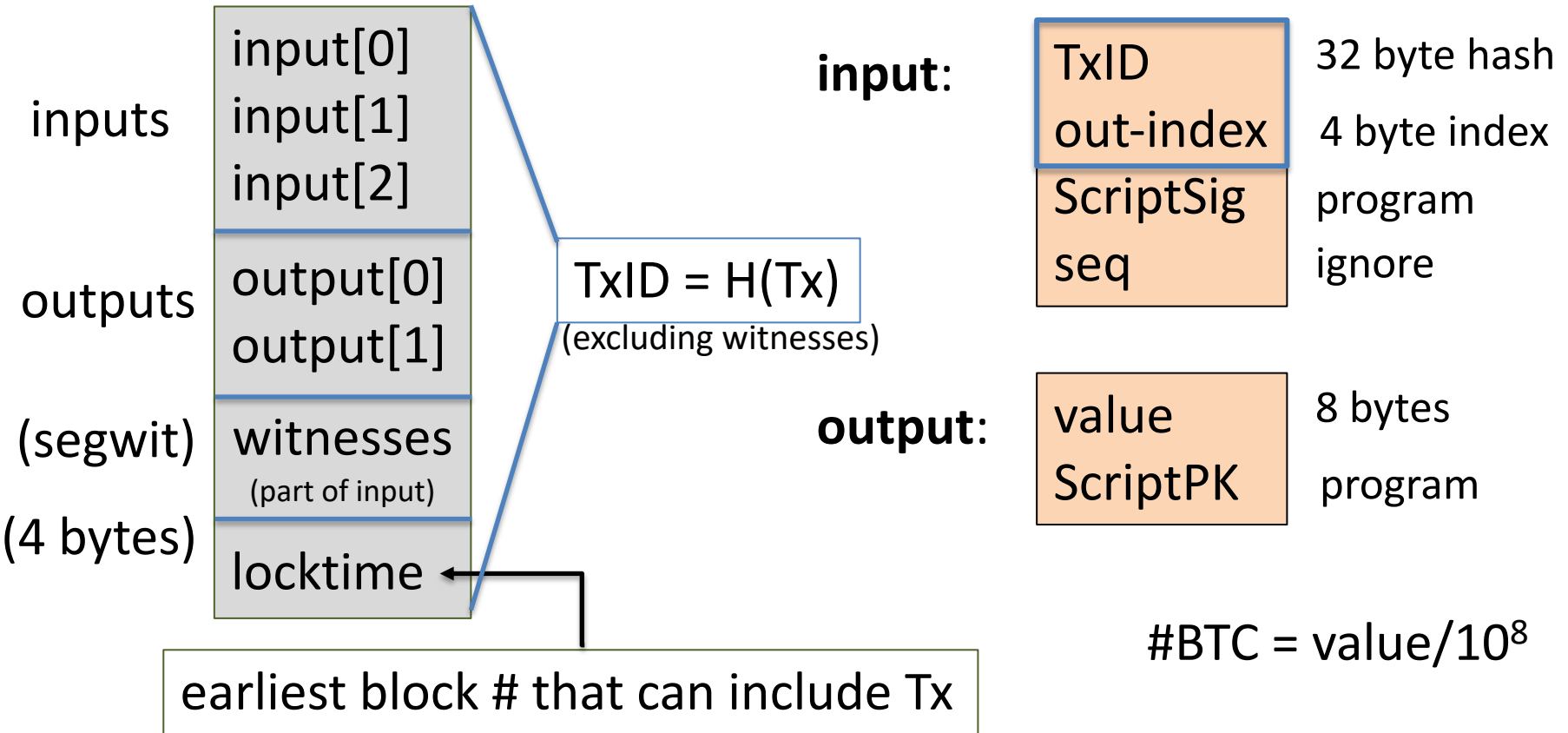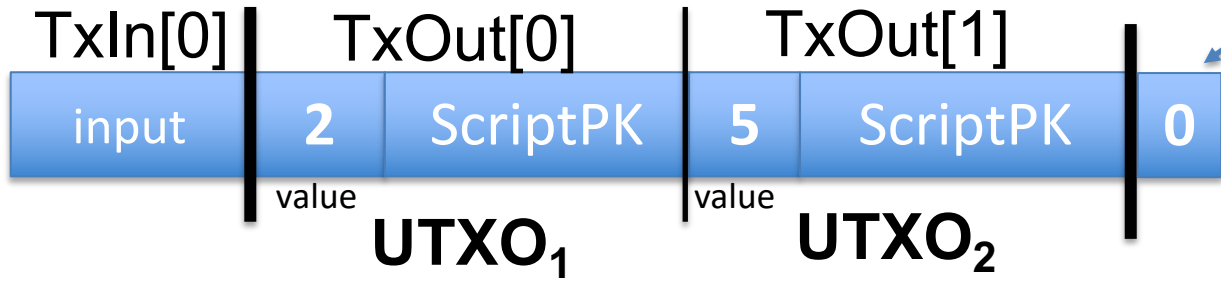
UTXO: unspent Tx output

Tx2:
(spending Tx)

TxIn[0]  TxOut[0]  TxOut[1]

TxID  1  ScriptSig  output  output  0

identifies a UTXO

$UTXO_3$  $UTXO_4$
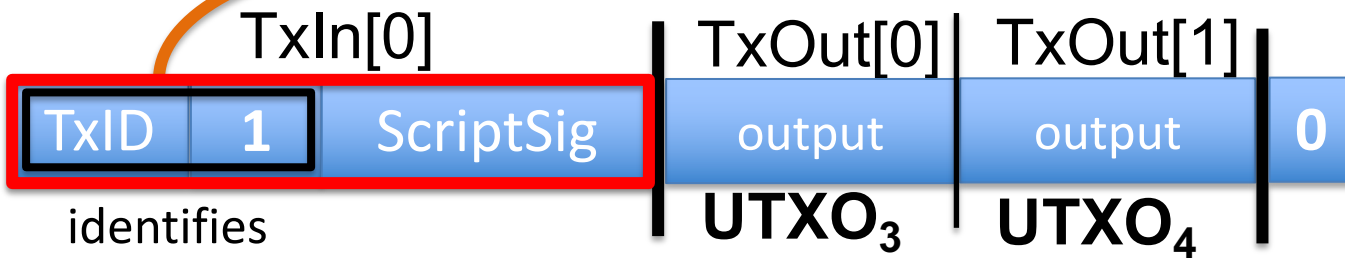
# Validating Tx2

Miners check (for each input):

1. The program   ScriptSig | ScriptPK   returns true

2. 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

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

# An example (block 648493) [2826 Tx]

COINBASE (Newly Generated Coins)

Tx0

1CK6KHY6MHgYvmRQ4PAafKYDrg1ejbH1cE   7.14047154 BTC 🌐
OP_RETURN   0.00000000 BTC
OP_RETURN   0.00000000 BTC

0.00000000 BTC

6.25 + Tx fees =   7.14047154 BTC

---

3PuJbxJS1pKxf8EdVR18yBkD1fPAbgUtyw

input

0.72333974 BTC 🌐
(input UTXO value)

Tx1

0.00005800 BTC   (Tx fee)

1E5Ao1VUnA5BhffvXf2Xmud6avUgwkFnJv   0.00917379 BTC 🌐
bc1qr8k3e0vx06lpu3j7m858pa2ak9tyr56ttwvefk   0.61504199 BTC 🌐
bc1qdrxve8kua3yz5dgx6wf3u95ngh0d3e648...   0.09290152 BTC 🌐
14ZhjuXpQ5jCDjtAy7ZMu3hfEQCWewzLw7   0.00616444 BTC 🌐

outputs

0.72328174 BTC

---

17MWze4Z1uP1jnvqvj7SAnGtxcoVq11H8A

0.05000000 BTC 🌐

Tx2

0.00192000 BTC   (Tx fee)

3G3C2RFQ8gsf77EQpdR4ZReChWFKEHhxVU   0.04808000 BTC 🔴

0.04808000 BTC

sum of fees in block added to coinbase Tx

# Tx fees

Bitcoin average Tx fees in USD  (last 60 days, sep. 2022)



Bitcoin average Tx fees in USD  (all time)
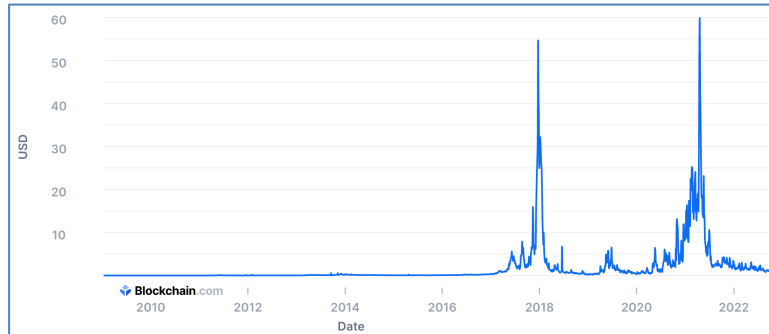
# All value in Bitcoin is held in UTXOs

## Unspent Transaction Outputs

The total number of valid unspent transaction outputs. This excludes invalid UTXOs with opcode OP_RETURN
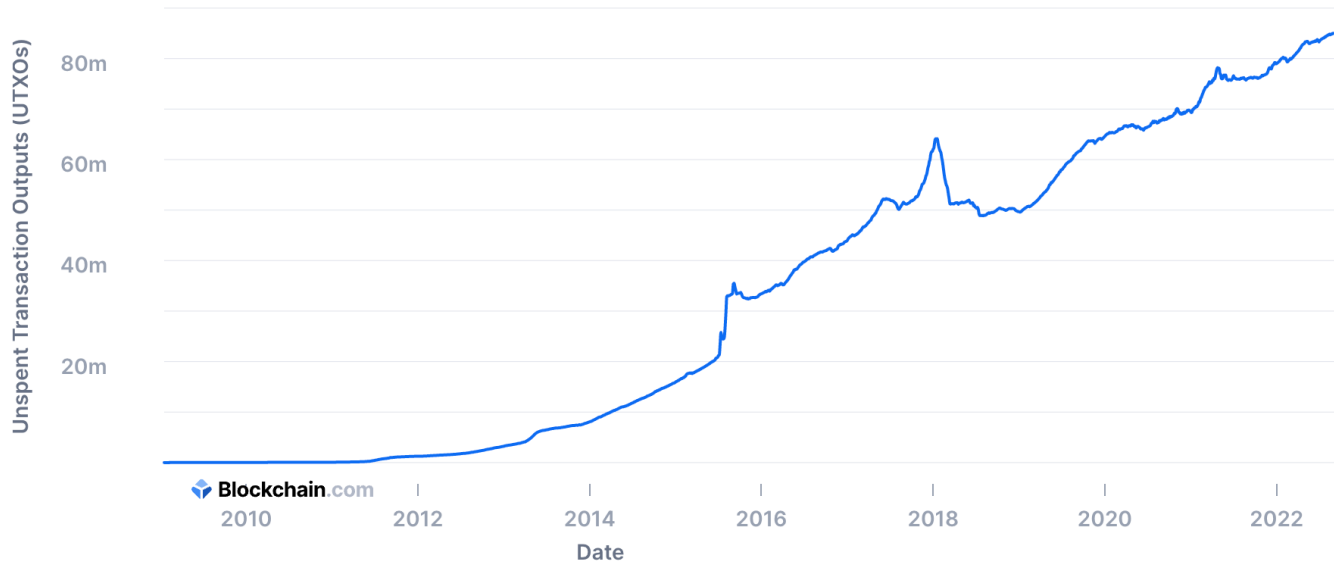


Sep. 2022:   miners need to store ≈85M UTXOs in memory

# Focusing on Tx2:    TxInp[0]

from UTXO
(Bitcoin script)

from TxInp[0]

| | |
|---|---|
| Value | 0.05000000 BTC |
| Pkscript | OP_DUP |
| | OP_HASH160 |
| | 45b21c8a0cb687d563342b6c729d31dab58e3a4e |
| | OP_EQUALVERIFY |
| | OP_CHECKSIG |
| Sigscript | 304402205846cace0d73de82dfbdeba4d65b9856d7c1b1730eb401cf4906b2401a69b |
| | dc90220589d36d36be64e774c8796b96c011f29768191abeb7f56ba20ffb0351280860 |
| | c01 |
| | 03557c228b080703d52d72ead1bd93fc72f45c4ddb4c2b7a20c458e2d069c8dd9e |

# 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 pk,   pop sig,   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

# Example: a common script

| &lt;sig&gt;  &lt;pk&gt; | **DUP  HASH256** &lt;pkhash&gt; **EQVERIFY  CHECKSIG** |
|---|---|

**stack**:  empty                                         init

    &lt;sig&gt; &lt;pk&gt;                              push values

    &lt;sig&gt; &lt;pk&gt; &lt;pk&gt;                      **DUP**

    &lt;sig&gt; &lt;pk&gt; &lt;hash&gt;                    **HASH256**

    &lt;sig&gt; &lt;pk&gt; &lt;hash&gt; &lt;pkhash&gt;         push value

    &lt;sig&gt; &lt;pk&gt;                              **EQVERIFY**

    1                                              **CHECKSIG**
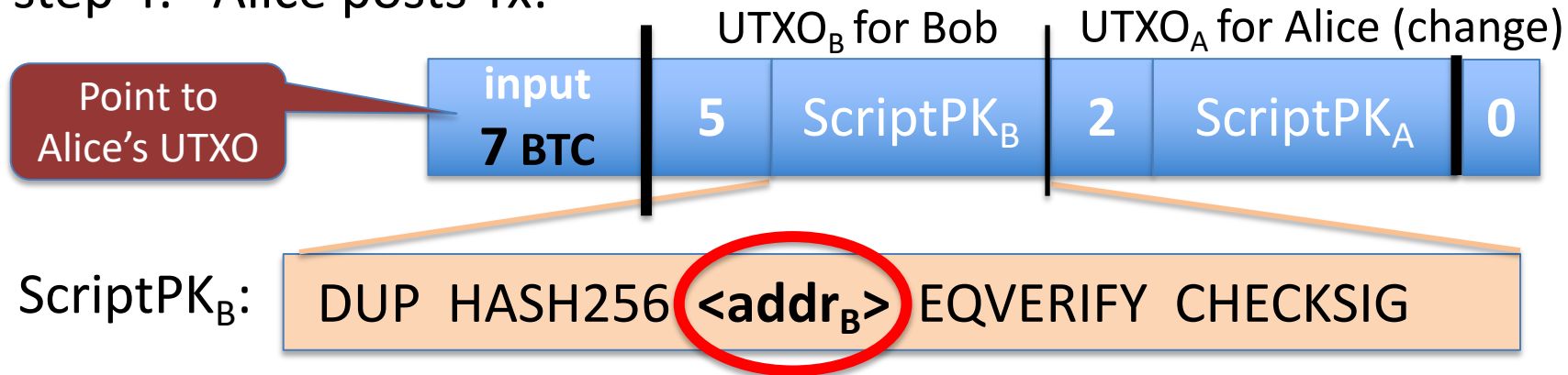
                                             verify(pk, Tx, sig)

  ⇒ successful termination

pay to public key hash

**Alice want to pay Bob 5 BTC**:

- step 1:  Bob generates sig key pair   $(pk_B, sk_B) \leftarrow Gen()$

- 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 posts Tx:

Point to Alice's UTXO

| input 7 BTC | | UTXO$_B$ for Bob | | UTXO$_A$ for Alice (change) | |
|---|---|---|---|---|---|
| | 5 | ScriptPK$_B$ | 2 | ScriptPK$_A$ | 0 |

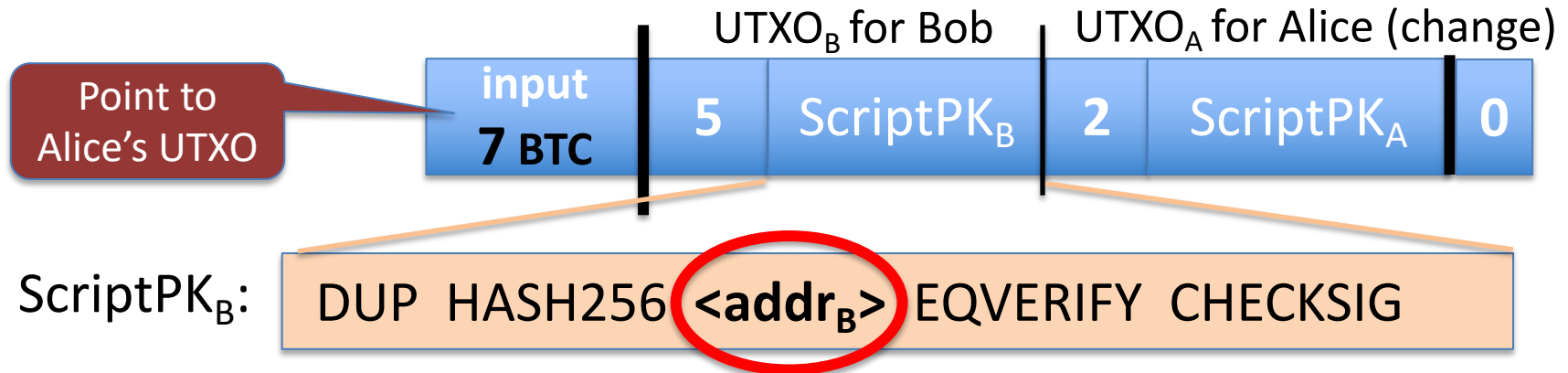ScriptPK$_B$:   DUP  HASH256  **<addr$_B$>**  EQVERIFY  CHECKSIG

# Transaction types:  (1) P2PKH

pay to public key hash

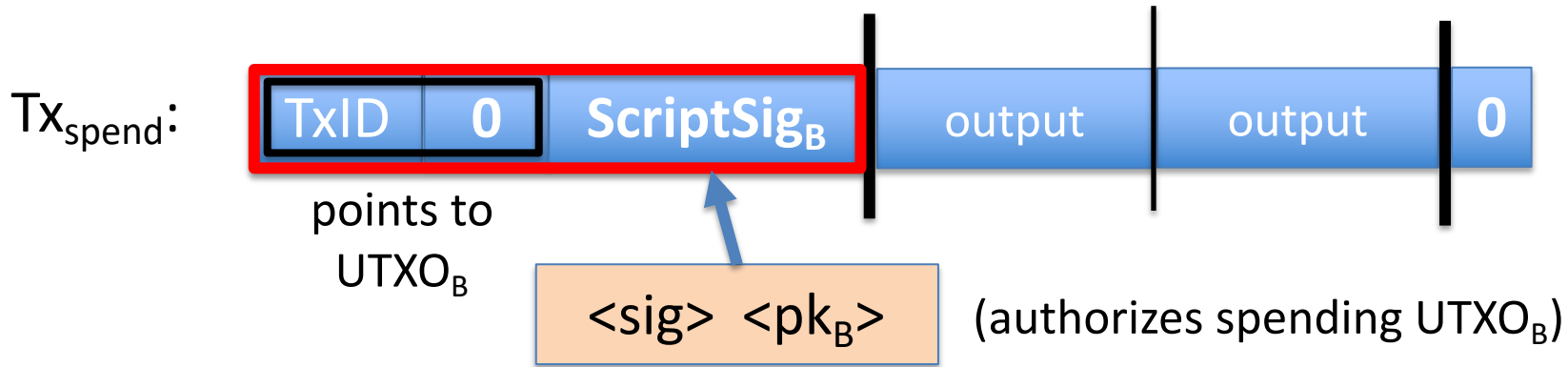"input" contains ScriptSig that authorizes spending Alice's UTXO

- example:  ScriptSig  contains Alice's signature on Tx

    $\implies$  miners cannot change $ScriptPK_B$    (will invalidate Alice's signature)



UTXO$_B$ for Bob | UTXO$_A$ for Alice (change)

Point to Alice's UTXO

| input 7 BTC | 5 | ScriptPK$_B$ | 2 | ScriptPK$_A$ | 0 |

ScriptPK$_B$:  DUP  HASH256 **\<addr$_B$\>** EQVERIFY  CHECKSIG

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

$Tx_{spend}$:

| TxID | 0 | **ScriptSig$_B$** | output | output | 0 |

points to UTXO$_B$

<sig>  <pk$_B$>   (authorizes spending UTXO$_B$)

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

Miners validate that   ScriptSig$_B$ | 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 Alice's signature that created $UTXO_B$

# 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)

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

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

Miner verifies:

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

(2)    ScriptSig = true           ⟵ script is satisfied

# Example P2SH:   multisig

**Goal**:  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

Next lecture:   interesting scripts,
        wallets, and how to manage crypto assets