CS251 Fall 2025

(cs251.stanford.edu)

Bitcoin Mechanics

Dan Boneh

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

Recap

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

Applications:

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

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

Digital signatures: syntax

Digital signatures: (Gen, Sign, Verify)

$\text{Gen}() \rightarrow (\text{pk}, \text{sk})$

$\text{Sign}(\text{sk}, m) \rightarrow \sigma$, $\text{Verify}(\text{pk}, m, \sigma) \rightarrow \text{accept/reject}$

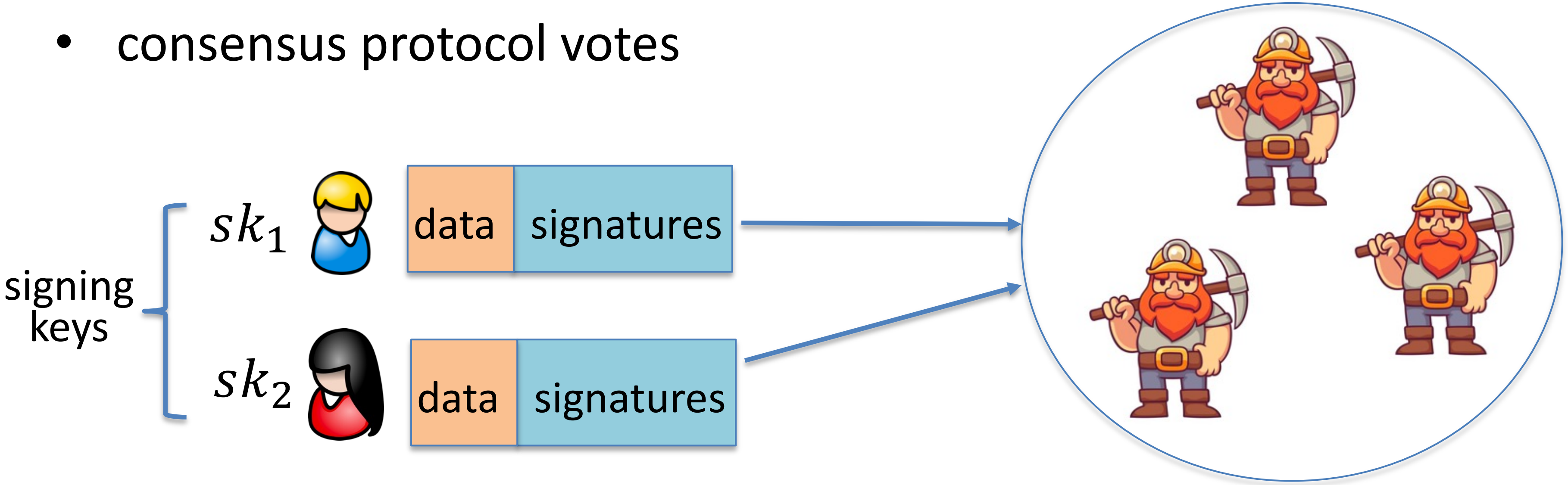
signing key

verification key

Signatures on the blockchain

Signatures are used everywhere:

- ensure Tx authorization
- governance votes
- consensus protocol votes

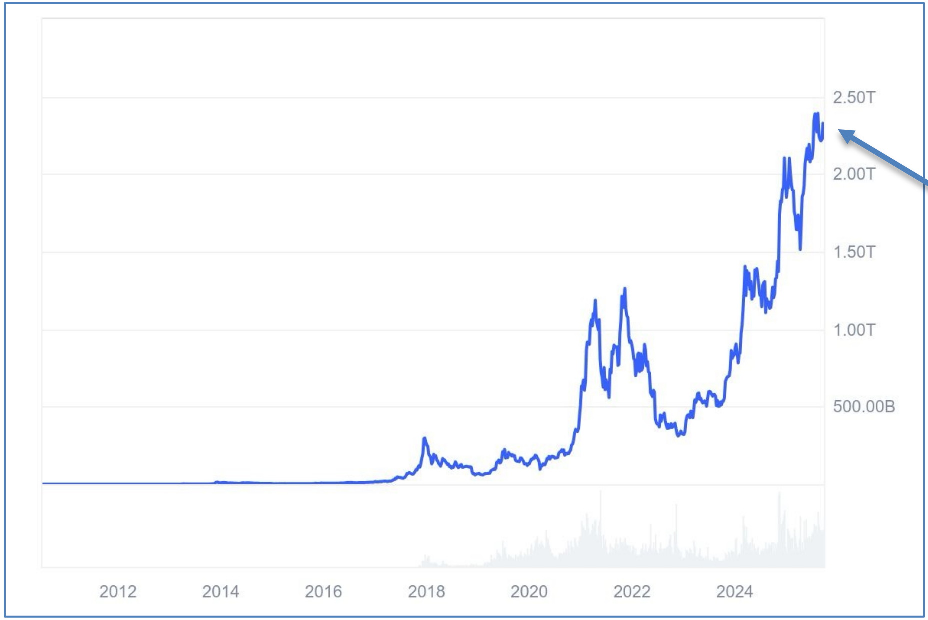


Bitcoin mechanics

This lecture: Bitcoin mechanics

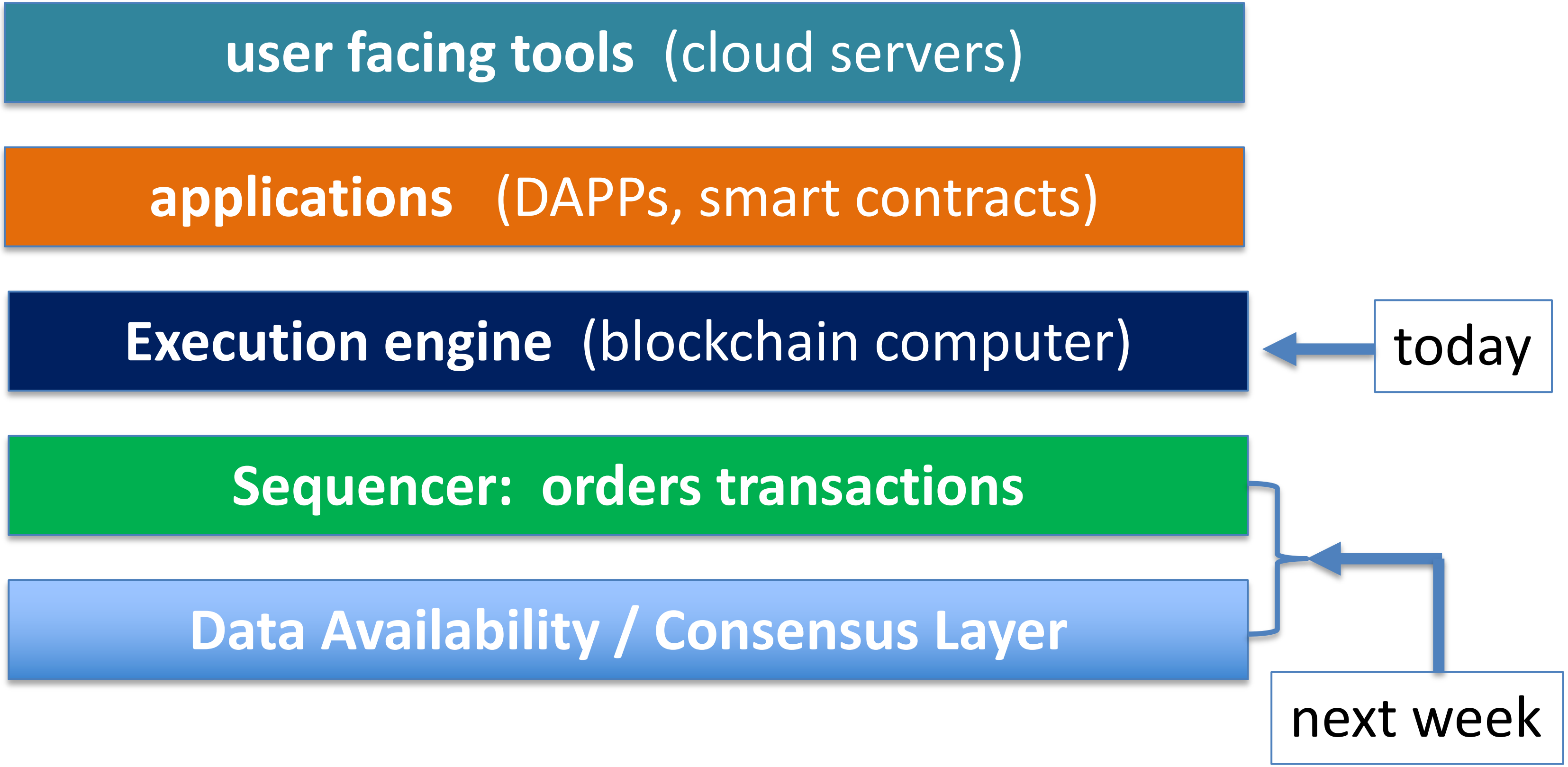
Oct. 2008: paper by Satoshi Nakamoto
Jan. 2009: Bitcoin network launched

Total market value:

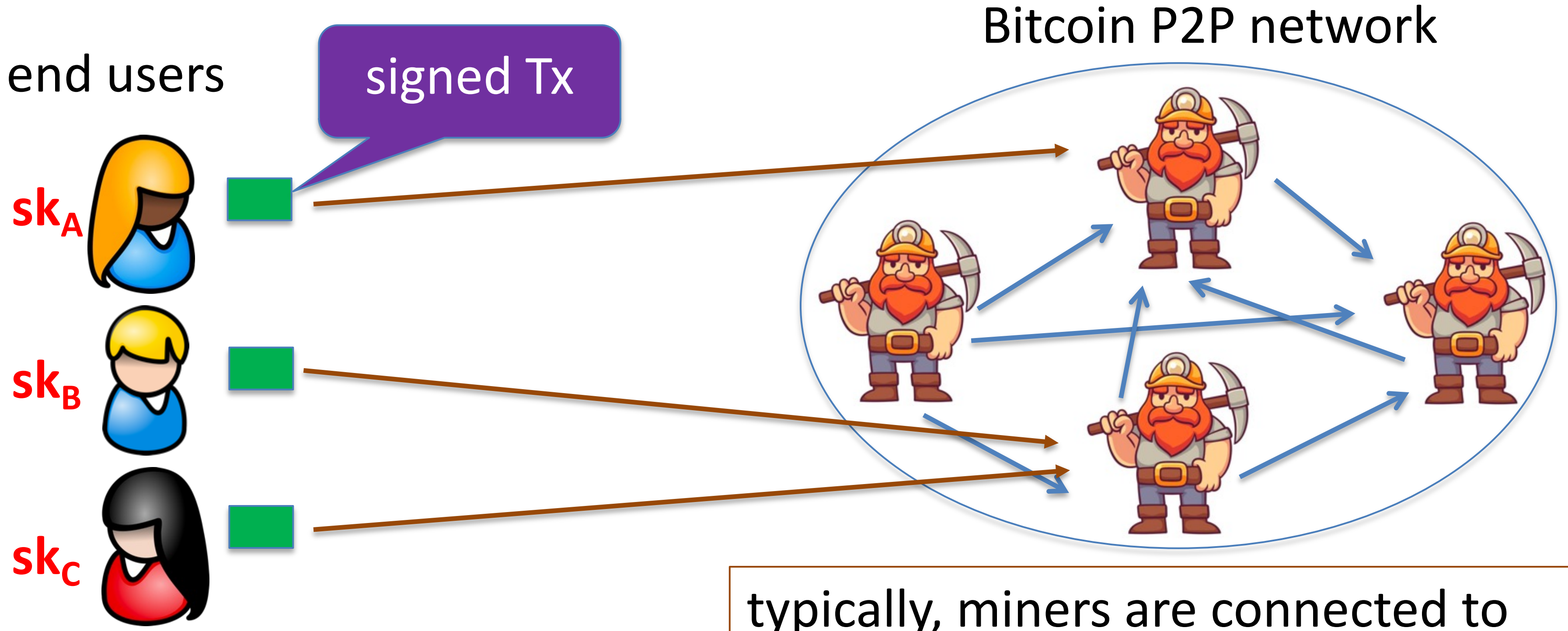


Sep. 2025: $2.23T

This lecture: Bitcoin mechanics



First: overview of the Bitcoin consensus layer



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

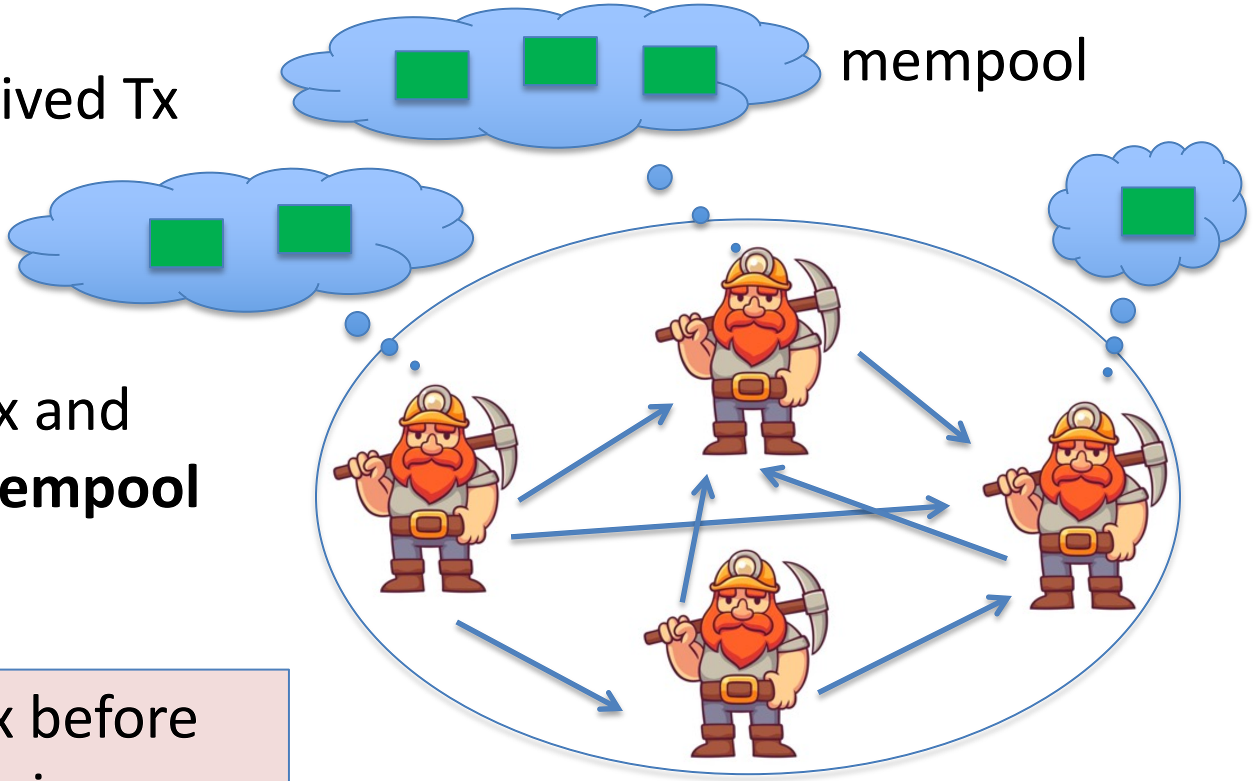
First: overview of the Bitcoin consensus layer

miners broadcast received Tx to the P2P network

every miner:

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

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



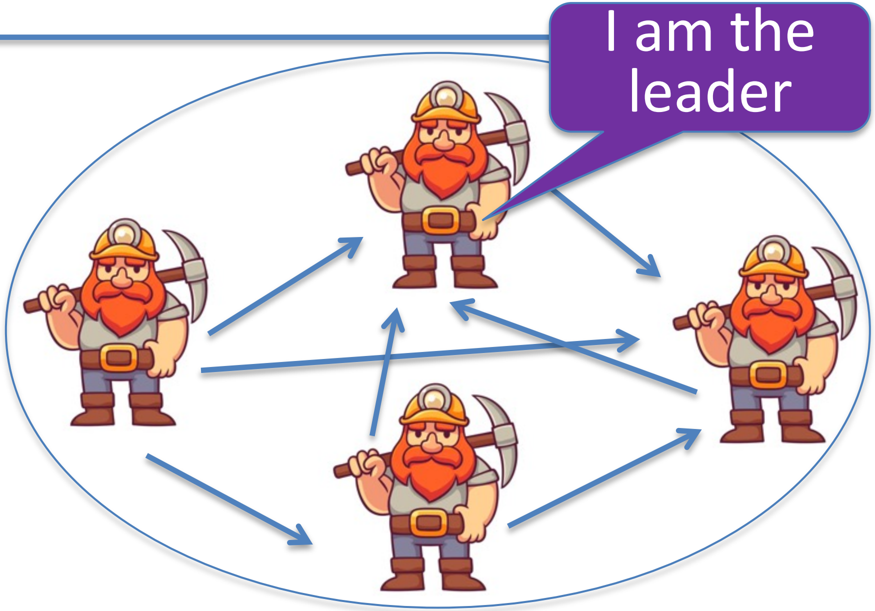
Bitcoin P2P network

First: overview of the Bitcoin consensus layer

blockchain

<u>Every ≈**10 minutes**</u>:

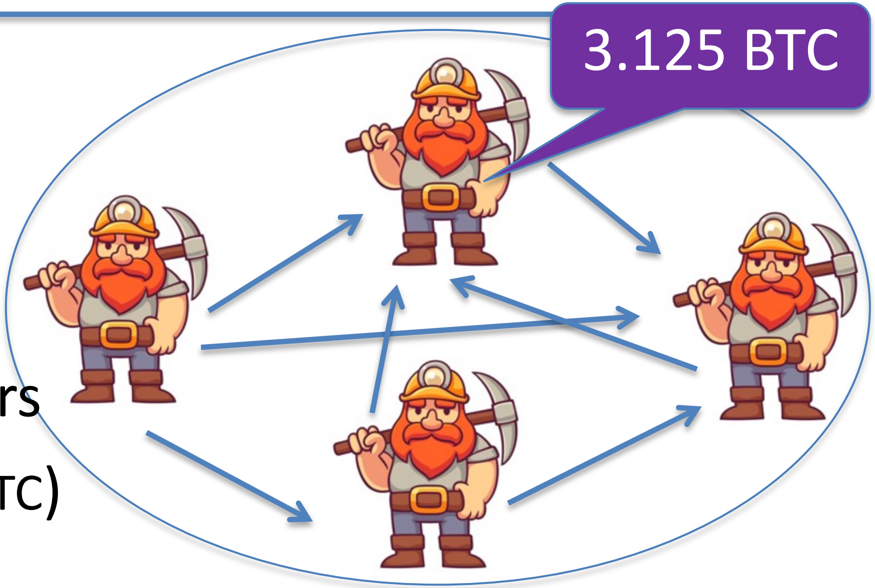
- Every miner creates a candidate block from Tx in its mempool
- a “random” miner is selected (how: next week), and broadcasts its block to P2P network
- all miners validate the new block



Bitcoin P2P network

First: overview of the Bitcoin consensus layer

blockchain



Selected miner is paid 3.125 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 19.9M BTC)

note: miner chooses order of Tx in block

Properties (very informal)

Next week:

Safety / Persistence:

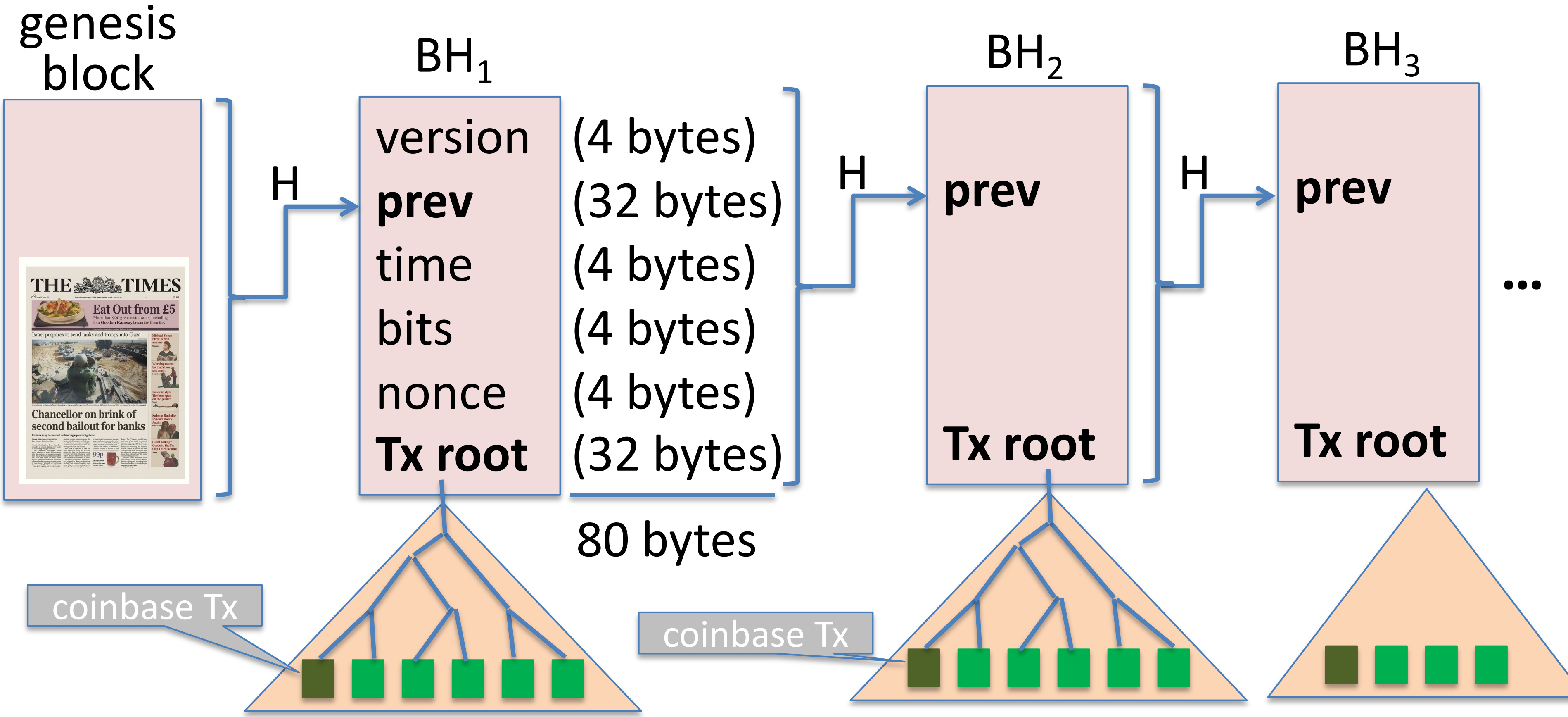
- to remove a block, need to convince 51% of mining power *

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

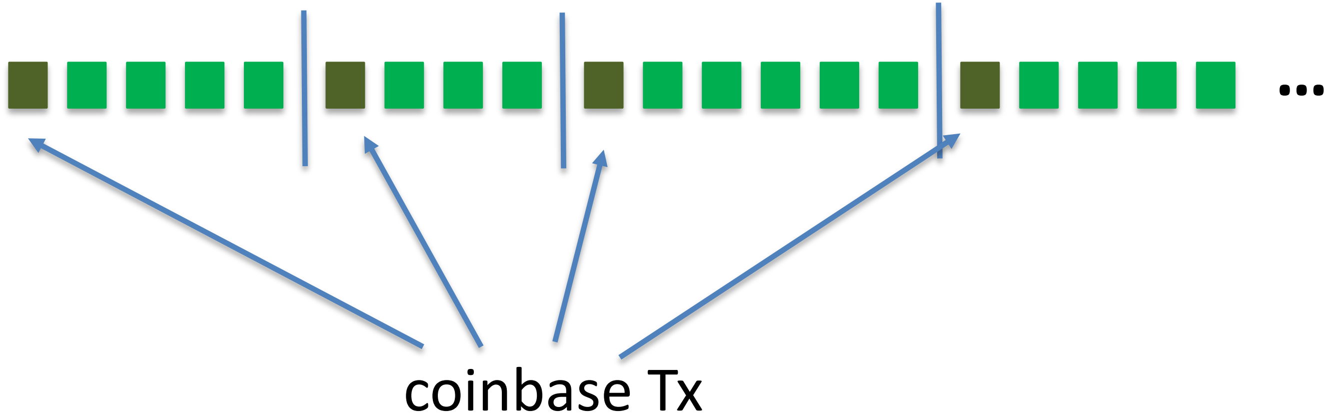
(block height) Number	Mined	Tx Count	Nonce	Tx Data Size	Total Sent	Total Fees
916098	3m 50s	5,176	1,937,572,100	1,770,074 Bytes	2,100 BTC	0.01BTC
916097	8m 43s	4,351	890,157,676	1,648,069 Bytes	3,555 BTC	0.02BTC
916096	23m 50s	4,811	3,213,781,271	1,644,000 Bytes	1,378 BTC	0.01BTC
916095	30m 56s	3,183	2,375,946,542	1,617,500 Bytes	6,653 BTC	0.03BTC
916094	50m 14s	5,201	1,519,773,008	1,637,737 Bytes	1,883 BTC	0.01BTC
916093	58m 24s	5,314	2,358,052,060	1,679,652 Bytes	5,851 BTC	0.01BTC

Block 916096 (Sep. 23, 2025, 15:54:50)

Miner:	**AntPool**	(from coinbase Tx)
Num. of Tx:	**4811**	(1377.59 BTC transferred, ≈$154M)
Difficulty (D):	**142,342,602,928,674.94**	(adjusts every two weeks)
Merkle root:	4cef12cf0d7d116a60ea1ec35e5325b17c28c83c84a6846251d4364de1e233d3	
Block Reward:	**3.125 BTC**	
Fee:	**0.01335727 BTC**	(Tx fee given to miner in coinbase Tx)

This lecture

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



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:

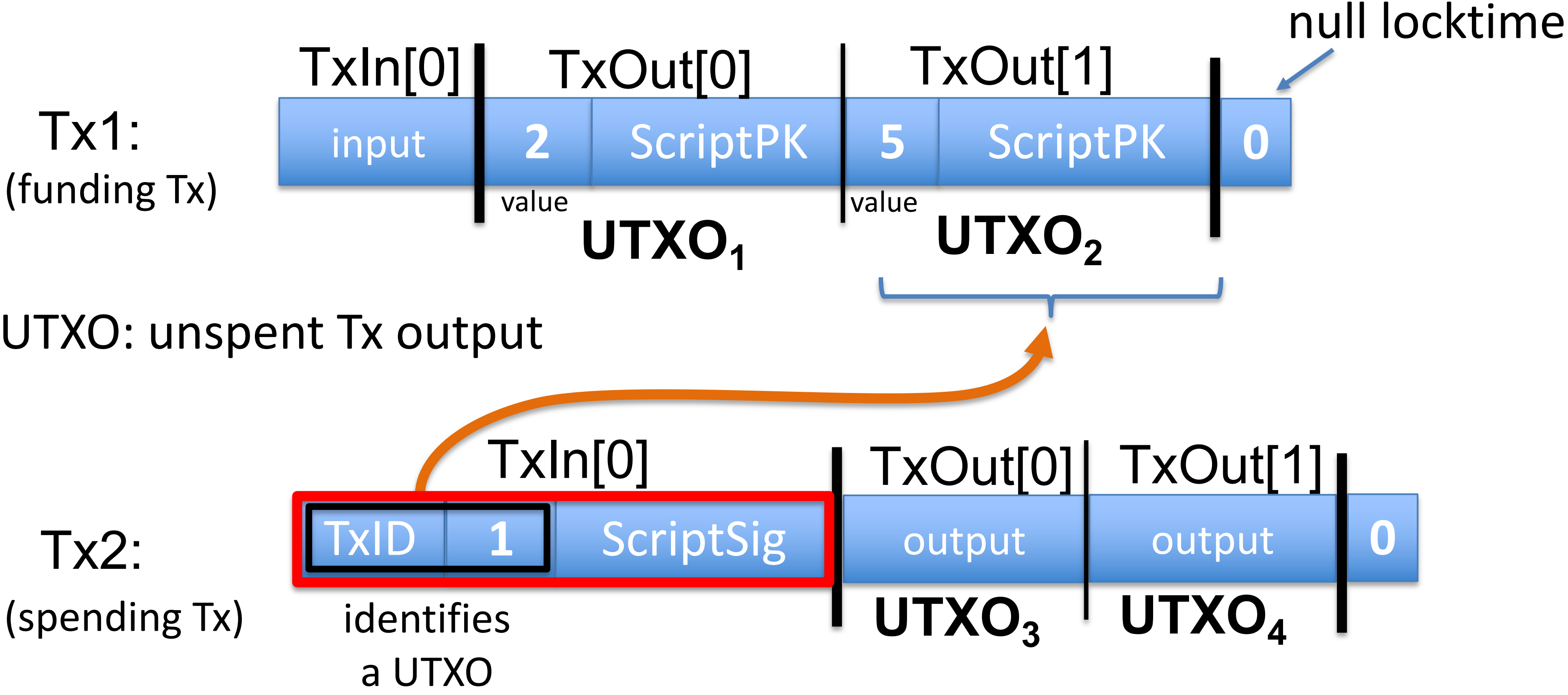
TxID	32 byte hash
out-index	4 byte index
ScriptSig	program
seq	ignore

output:

value	8 bytes
ScriptPK	program

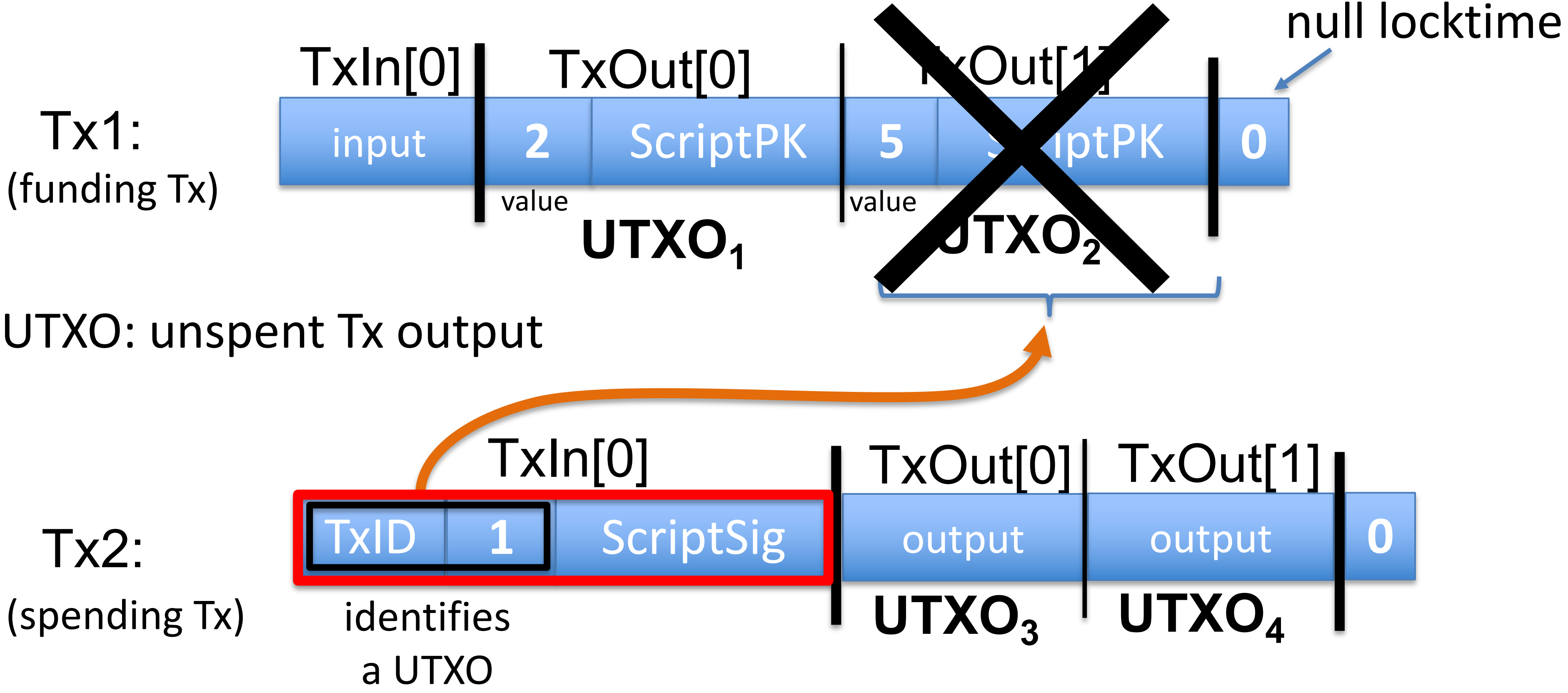
$\#BTC = value/10^8$

Example



UTXO: unspent Tx output

Example


Tx1:
(funding Tx)
TxIn[0]
TxOut[0]
null locktime
input
2
ScriptPK
5
0
value
value
UTXO1
UTXO: unspent Tx output
Tx2:
(spending Tx)
TxIn[0]
TxOut[0]
TxOut[1]
TxID
1
ScriptSig
output
output
0
identifies
a UTXO
UTXO3
UTXO4

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 $\geq$ sum output values

program from spending Tx: proof that conditions are met

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

An example (block 916096) [4811 Tx in block]

Tx0 (coinbase)

total out = 3.138357 BTC

From	To
1 Block Reward 0.00 BTC • $0.00	1 37jKPSmbEGwgfacCr2nayn1wTaqMAbA94Z 0.00000546 BTC • $0.61
	2 39C7fxSzEACPjM78Z7xdPxhf7mKxJwvfMJ 3.13835181 BTC • $352,038

Tx5

Fee = 0.000114 BTC

From	To
1 bc1qwqdg6squsna38e46795at95yu9atm8azzmyvckulcc7kytlcckxswvvzej 0.00470098 BTC • $527.32	1 3CxzwSY2p9ue5wuE4dVja3LpgywHt96qCP 0.00142340 BTC • $159.67
	2 bc1qwqdg6squsna38e46795at95yu9atm8azzmyvckulcc7kytlcckxswvvzej 0.00316358 BTC • $354.87

Tx6

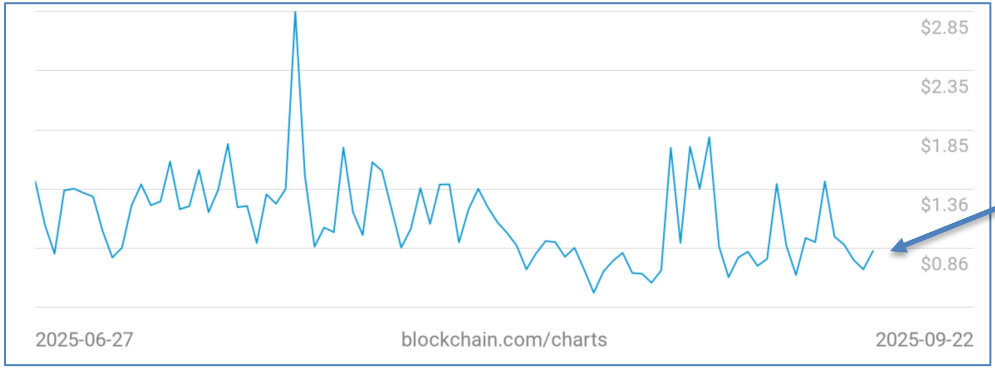
Fee = 0.000198 BTC

From	To
1 1KwCJtSys65ZLMZDfZiHfbd7UG1993i5X4 0.00336287 BTC • $377.22	1 bc1que69yuylzyqnsa2frhqfg8d3duyy2fwdz3l3g4 0.02983985 BTC • $3,347.22
2 13jDp6TouDQV6DNMnj2sezusfCCugM3T5s 0.02940749 BTC • $3,298.72	2 1KwCJtSys65ZLMZDfZiHfbd7UG1993i5X4 0.00273251 BTC • $306.51

- Fee is chosen by Tx creator; if too low, Tx will wait in mempool
- Sum of fees in block added to coinbase Tx

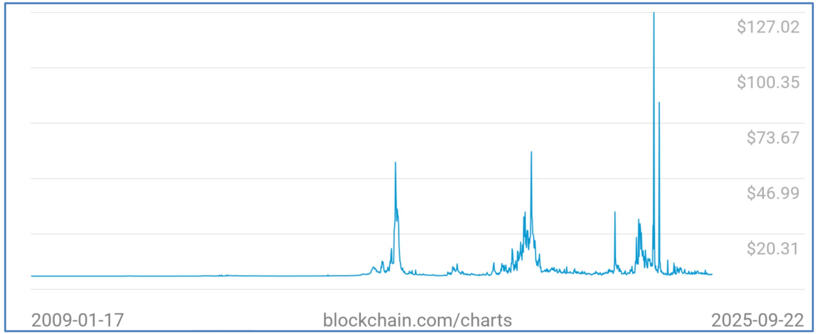
Tx fees

Bitcoin daily average Tx fees in USD (last 90 days, sep. 2025)



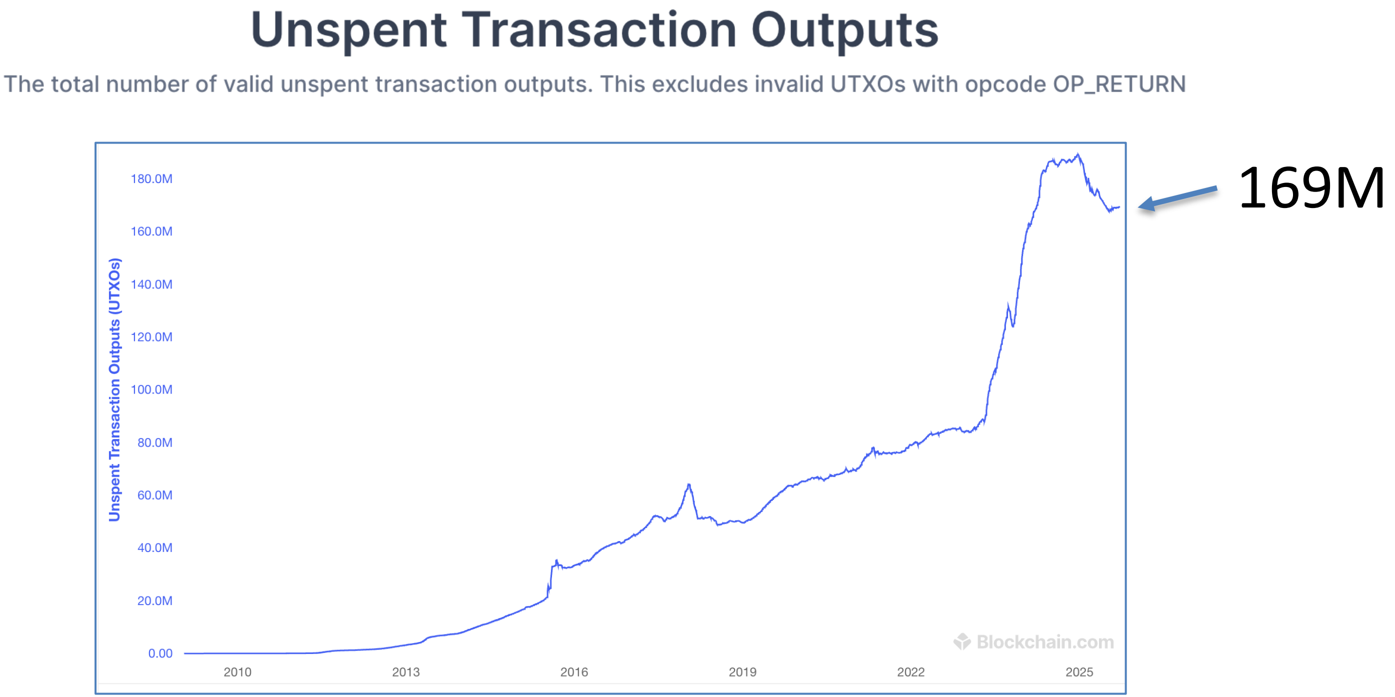
$0.83/Tx

Bitcoin average Tx fees in USD (all time)



Fees depend on network congestion

All value in Bitcoin is held in UTXOs



Sep. 2025: miners need to store ≈169M UTXOs in memory

Focusing on one Tx input

from UTXO
(Bitcoin script)

Pkscript: a program that specifies the conditions needed to spend the UTXO

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 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

<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

What's up with OP_CAT

OP_CAT: stack = <abc> <def> ⇒ stack = <abcdef>

The problem script: ⇒ Miner would run out of memory

OP_DUP OP_CAT OP_DUP OP_CAT … (repeat 50 times)

Satoshi's solution (2010): disable OP_CAT

⇒ quite sad: Bitcoin cannot verify Merkle proofs

Today: stack cell is limited to 520 bytes (script fails if exceeded)

… but OP_CAT still not re-enabled.

Transaction types: (1) P2PKH

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**	5	$ScriptPK_B$	2	$ScriptPK_A$	0
	$UTXO_B$ for Bob		$UTXO_A$ for Alice (change)		

$ScriptPK_B$: DUP HASH256 $\mathbf{<addr_B>}$ EQVERIFY CHECKSIG

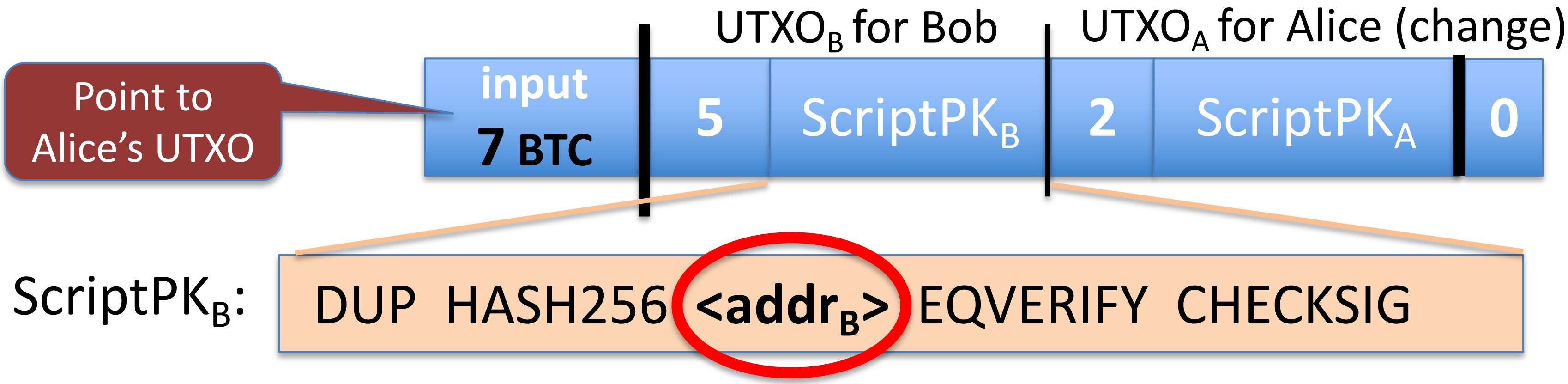
Transaction types: (1) P2PKH

pay to public key hash

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

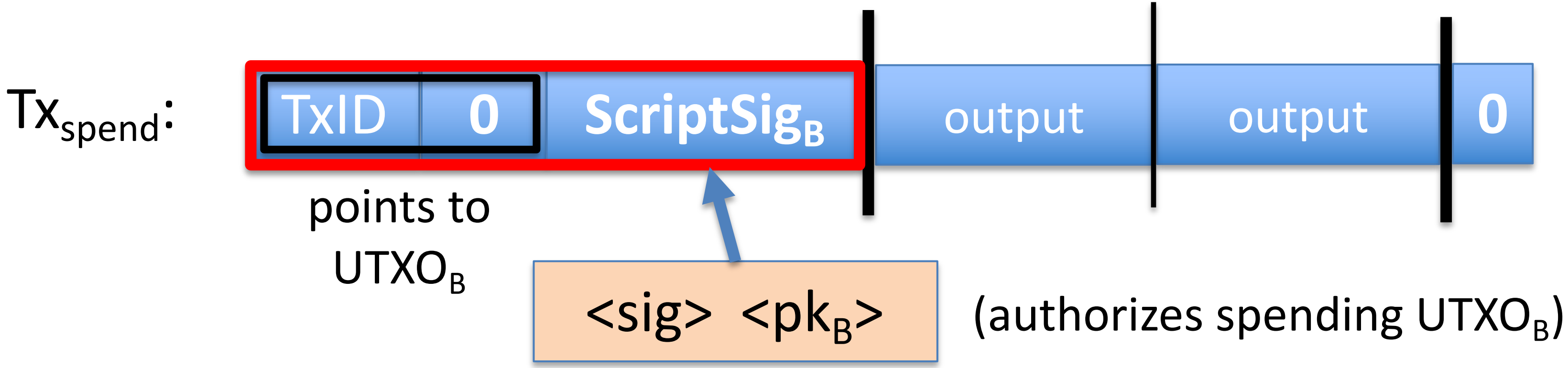
- example: ScriptSig contains Alice's signature on Tx

  $\Longrightarrow$ miners cannot change ScriptPK$_B$ (will invalidate Alice's signature)



Transaction types: (1) P2PKH

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



<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 quantum attacks on pk
  - ⇒ Unfortunately, first 200K coinbase Tx are P2PK (no hash)

- 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

(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: $<sig_1> <sig_2> \dots <sig_n>$ <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

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