Announcement

We're going to use Piazza!

https://piazza.com/class/m8zibfzpqsi115/

Find a team by end of week.

Introduction to Bitcoin

Deian Stefan

Slides from Dan Boneh

Today

1. What is a blockchain?
2. Basic introduction to cryptography
3. Start looking at Bitcoin (and finish next week)

What is a blockchain?

user facing tools (cloud servers)

applications (DAPPs, smart contracts)

Execution engine (blockchain computer)

Sequencer: orders transactions

Data Availability / Consensus Layer

Consensus layer (informal)

achieved by replication

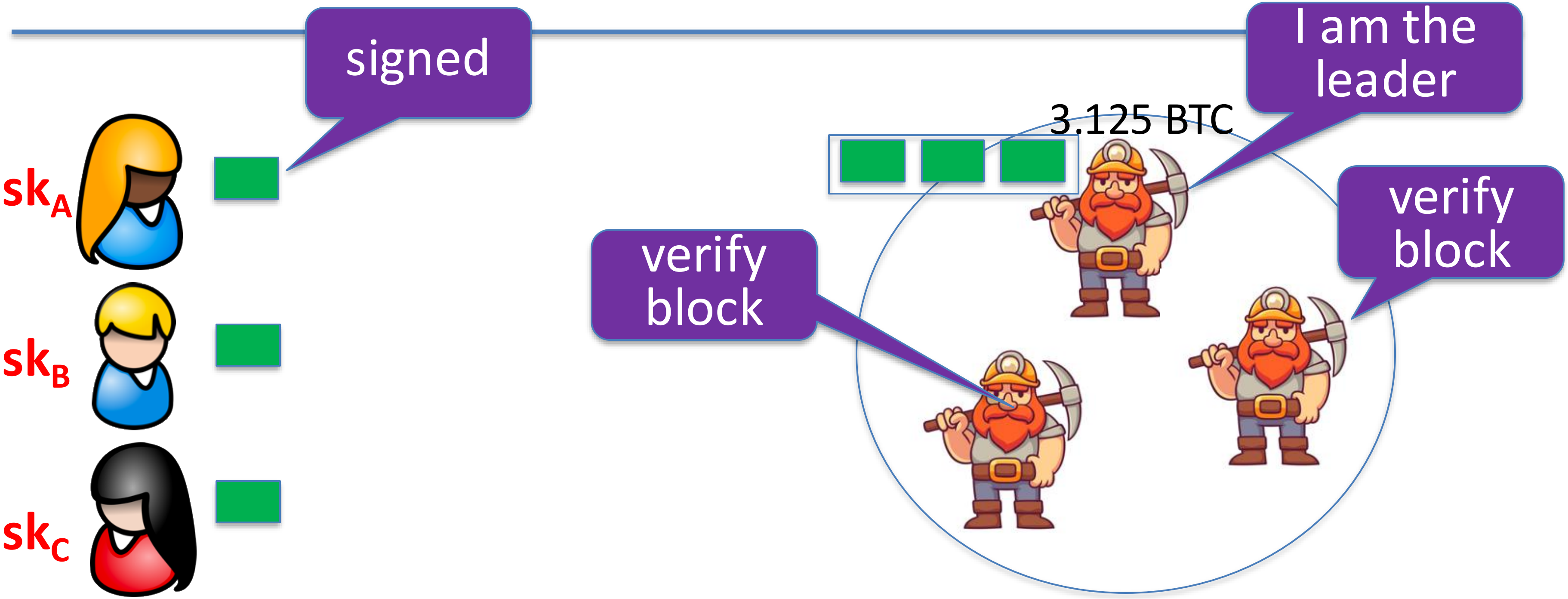
A **public** append-only data structure:

- **Persistence**: once added, data can never be removed*
- **Safety**: all honest participants have the same data**
- **Liveness:** honest participants can add new transactions
- **Open(?)**: anyone can add data (no authentication)

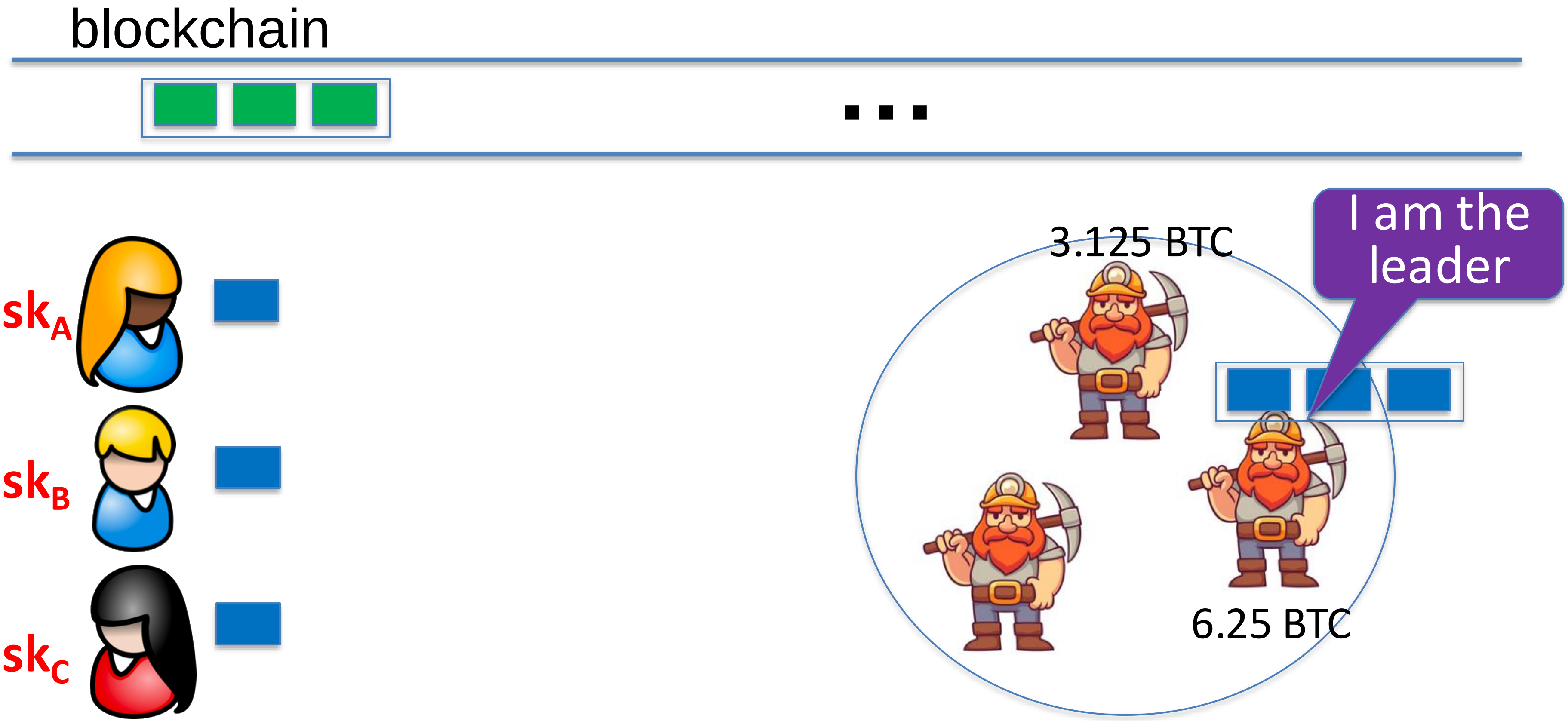
Data Availability / Consensus layer

How are blocks added to chain?

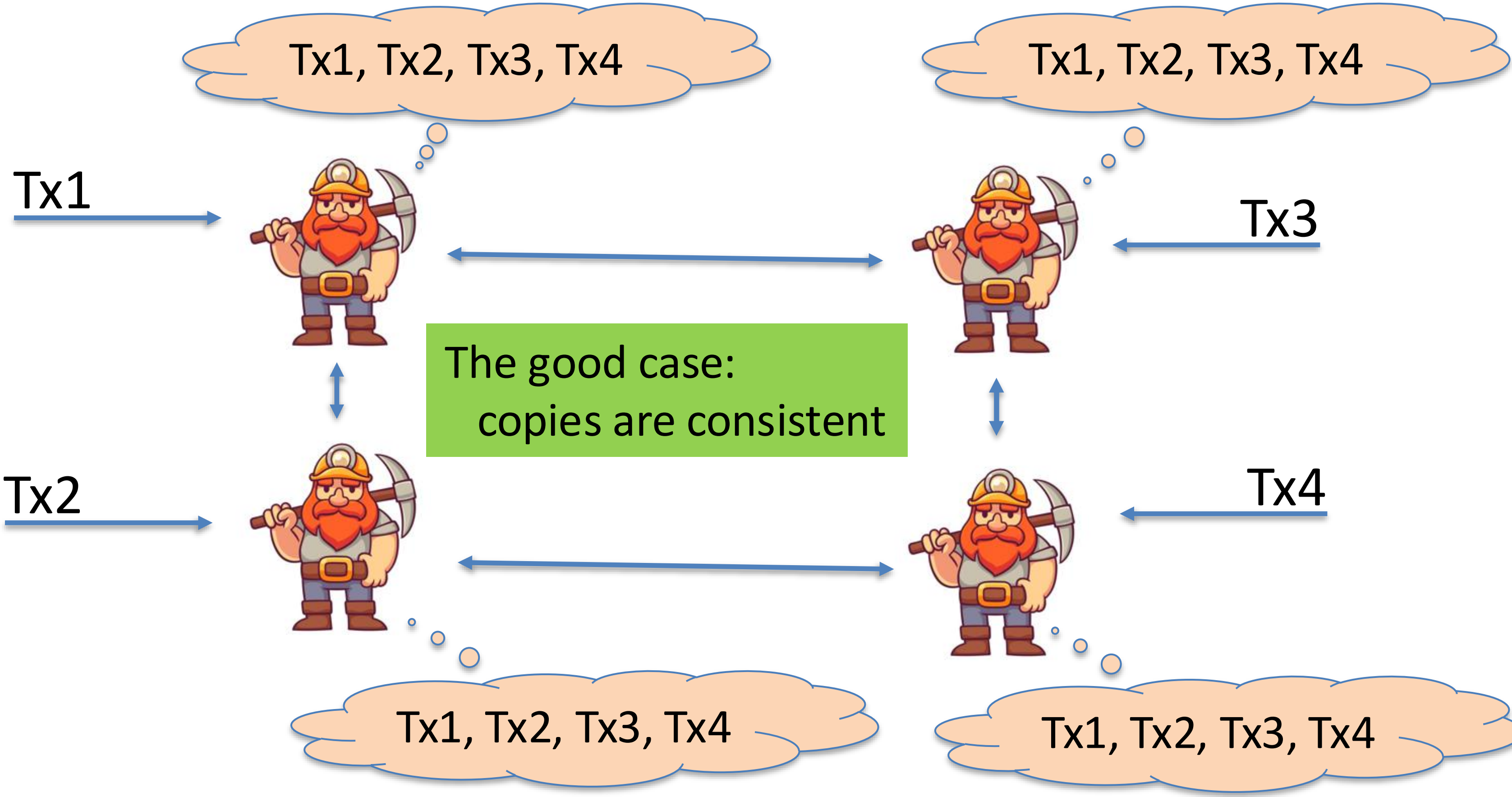
blockchain



How are blocks added to chain?



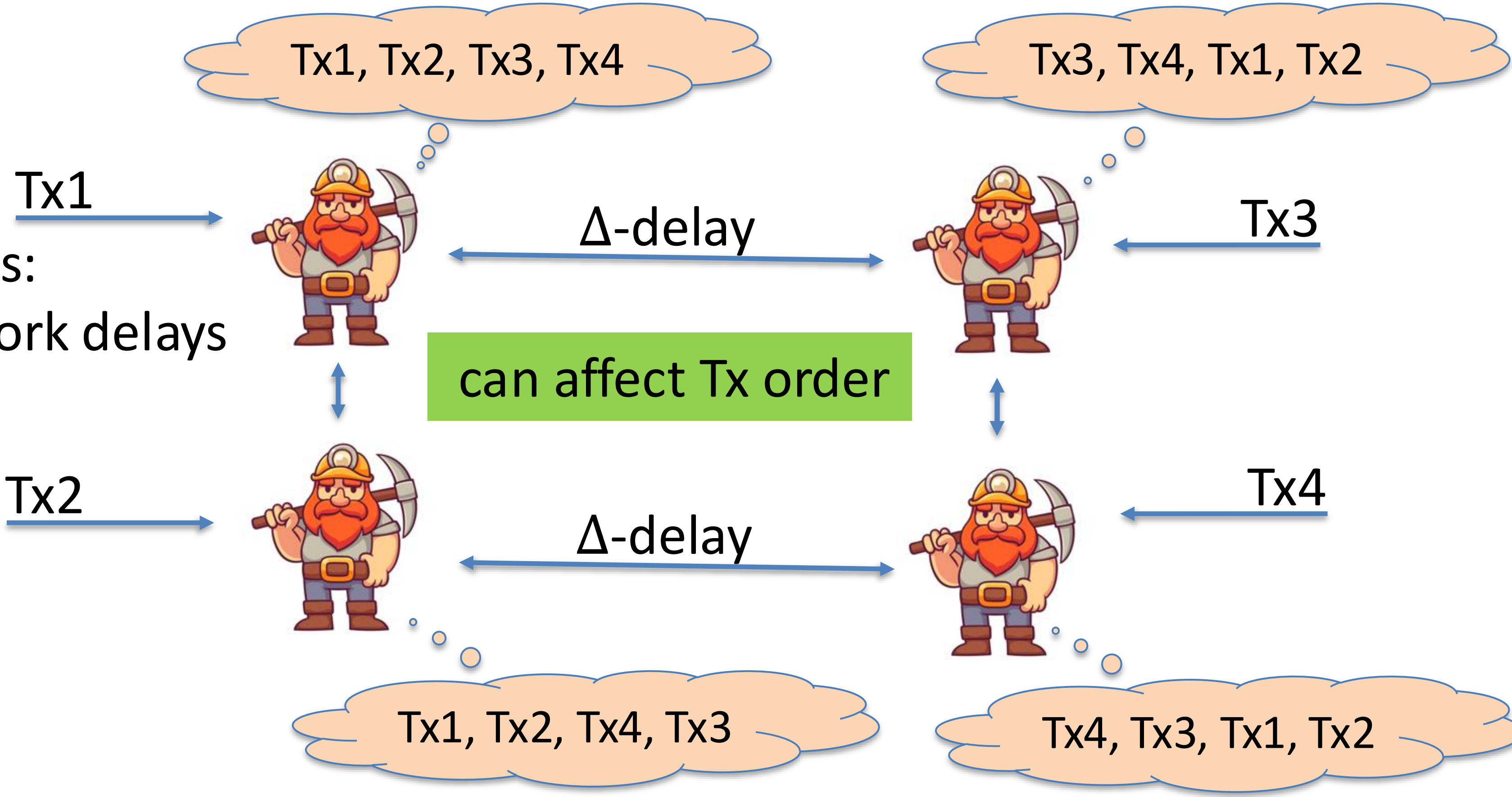
Why is consensus a hard problem?



Why is consensus a hard problem?

Problems:

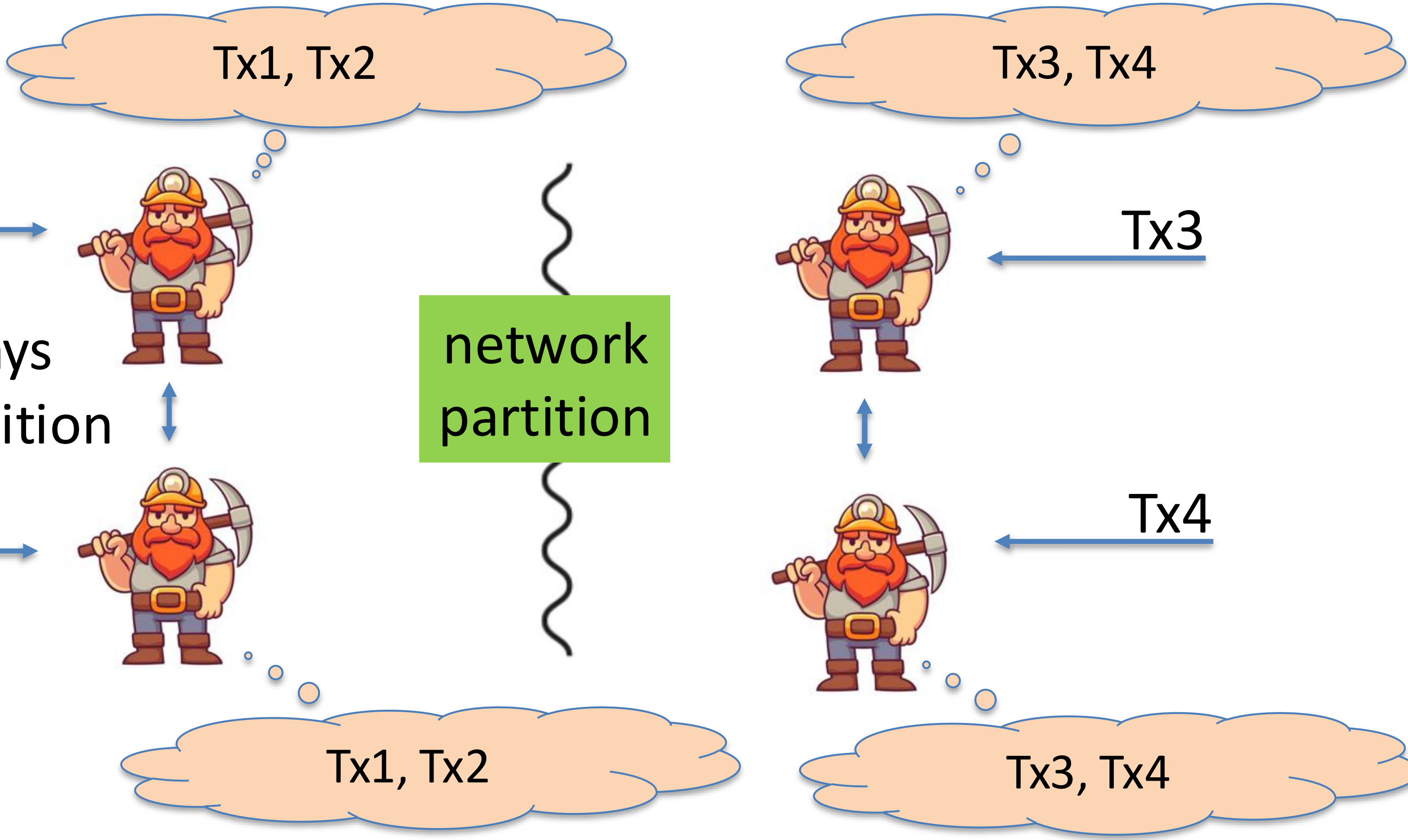
- Network delays



Why is consensus a hard problem?

Problems:

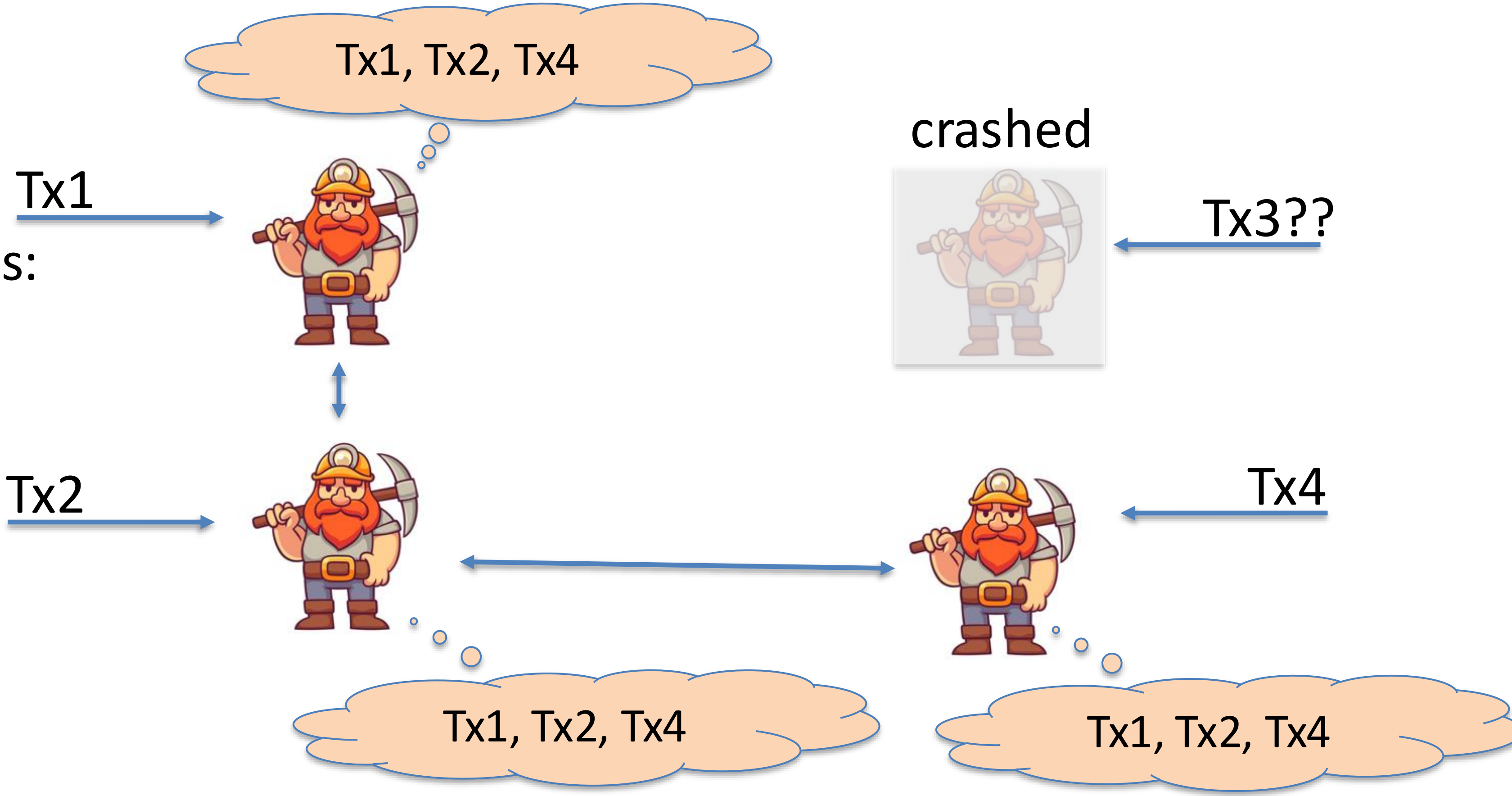
- Network delays
- Network partition



Why is consensus a hard problem?

Problems:

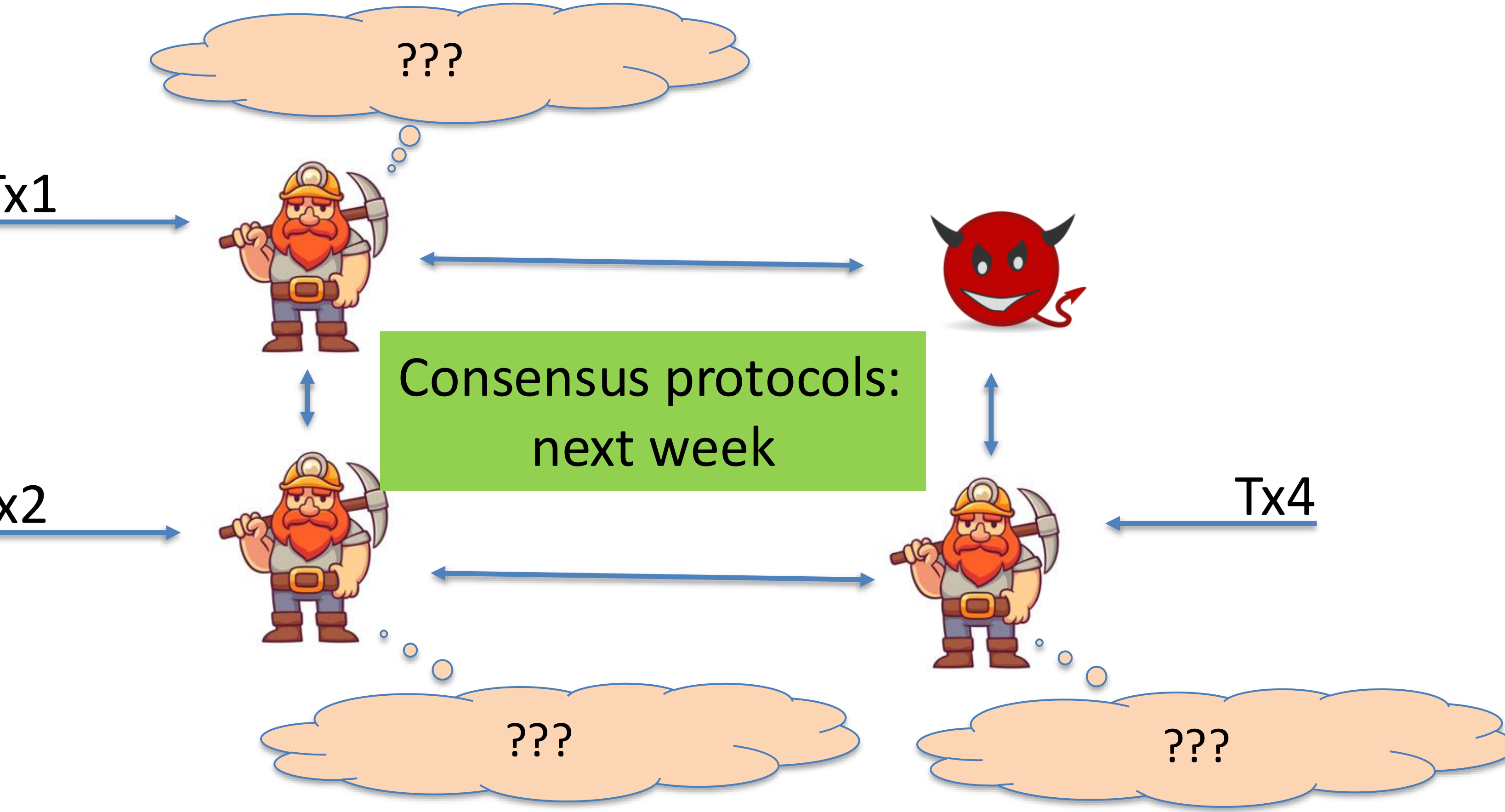
- crash



Why is consensus a hard problem?

Problems:

- crash
- malice

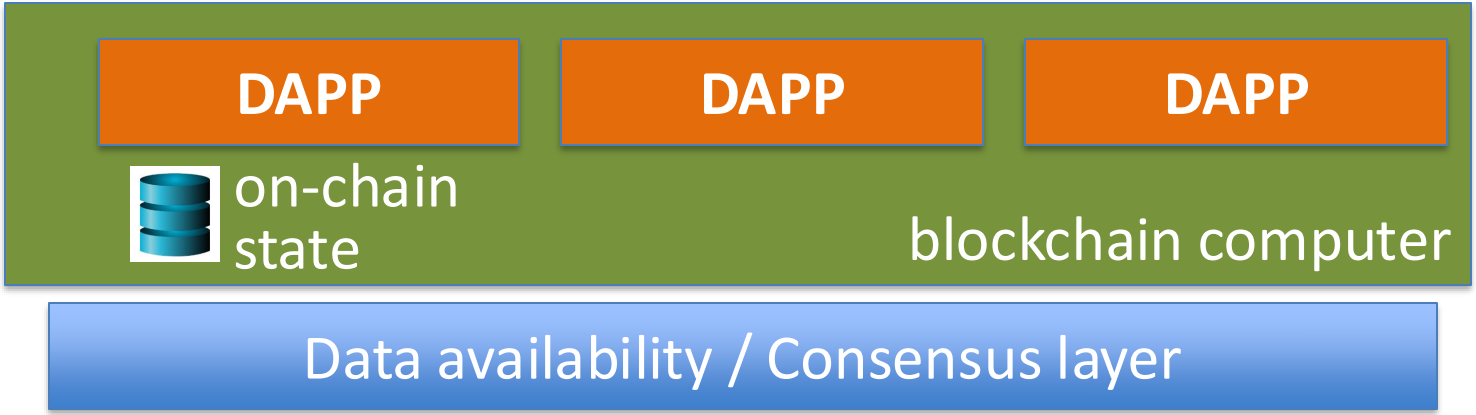


Consensus protocols:
next week

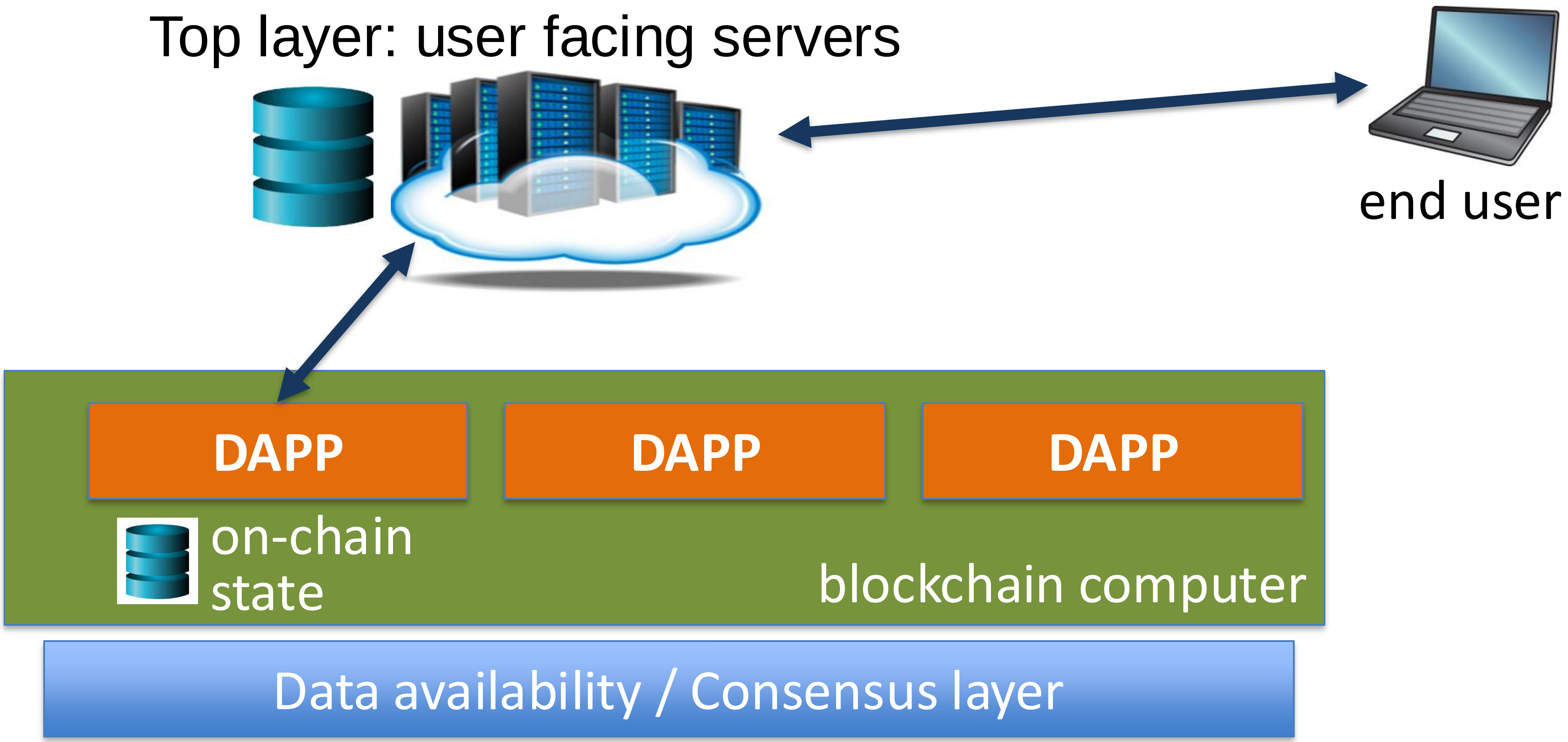
Next layer: the blockchain computer

Decentralized applications (DAPPs):

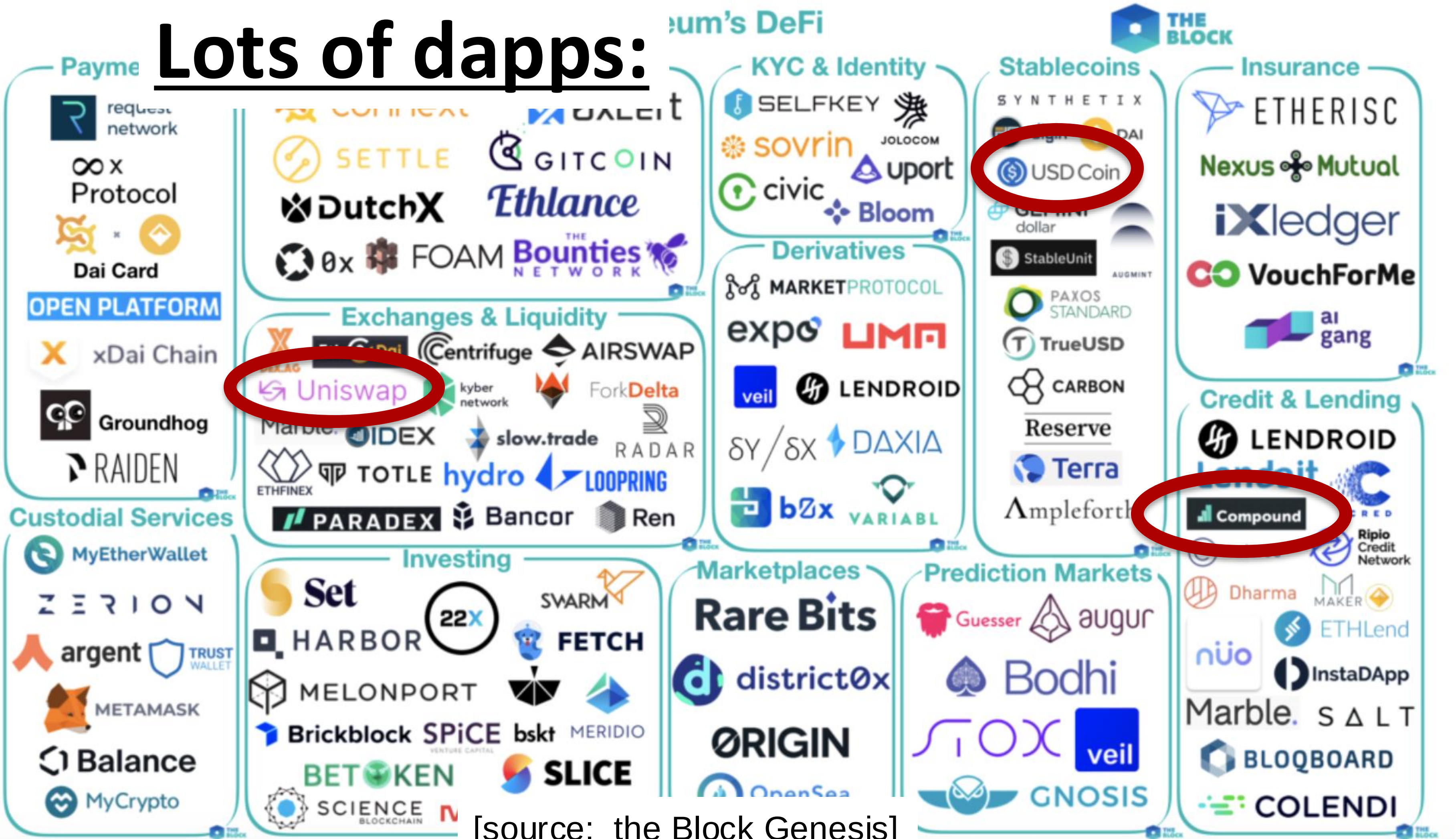
- Run on blockchain: code and state are written on chain
- Accept Tx from users $\Rightarrow$ state transitions are recorded on chain



Next layer: the blockchain computer



Lots of dapps:



[source: the Block Genesis]

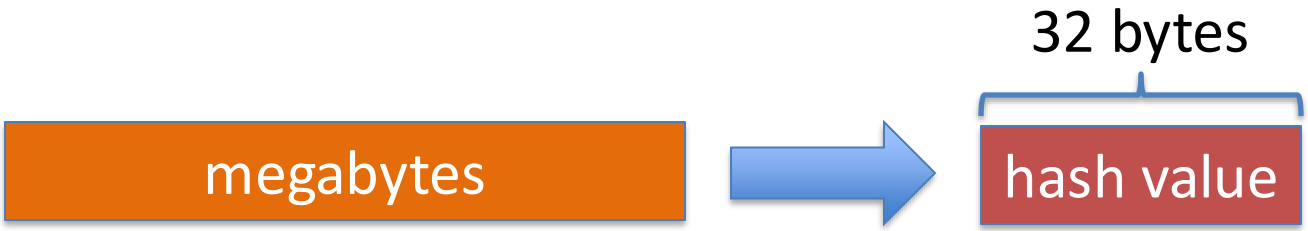
Today

1. What is a blockchain?
2. Basic introduction to cryptography
    - Hash functions
    - Commitments
    - Merkle trees
    - Signatures
3. Start looking at Bitcoin

Hash functions

An efficiently computable function $H: M \to T$

where $|M| \gg |T|$



$T = \{0,1\}^{256}$

Collision resistance

<u>Def</u>: a **<u>collision</u>** for $H: M \to T$ is pair $x \neq y \in M$ s.t. $\boxed{H(x) = H(y)}$

$|M| \gg |T|$ implies that <u>many</u> collisions exist

<u>Def:</u> a function $H: M \to T$ is **<u>collision resistant</u>** if it is "hard" to find even a single collision for H (we say H is a CRF)

Example: **SHA256**: $\{x : \text{len}(x) < 2^{64} \text{ bytes}\} \to \{0,1\}^{256}$

(output is 32 bytes)

Application: committing to data on a blockchain

Alice has a large file m. She posts $h = H(m)$ (32 bytes)

Bob reads h. Later he learns m' s.t. $H(m') = h$

H is a CRF $\Rightarrow$ Bob is convinced that $m' = m$

(otherwise, m and m' are a collision for H)

We say that $h = H(m)$ is a **binding commitment** to m

(note: not hiding, h may leak information about m)

Committing to a list (of transactions)

32 bytes

Alice has $S = (m_1, m_2, \ldots, m_n)$

Goal:

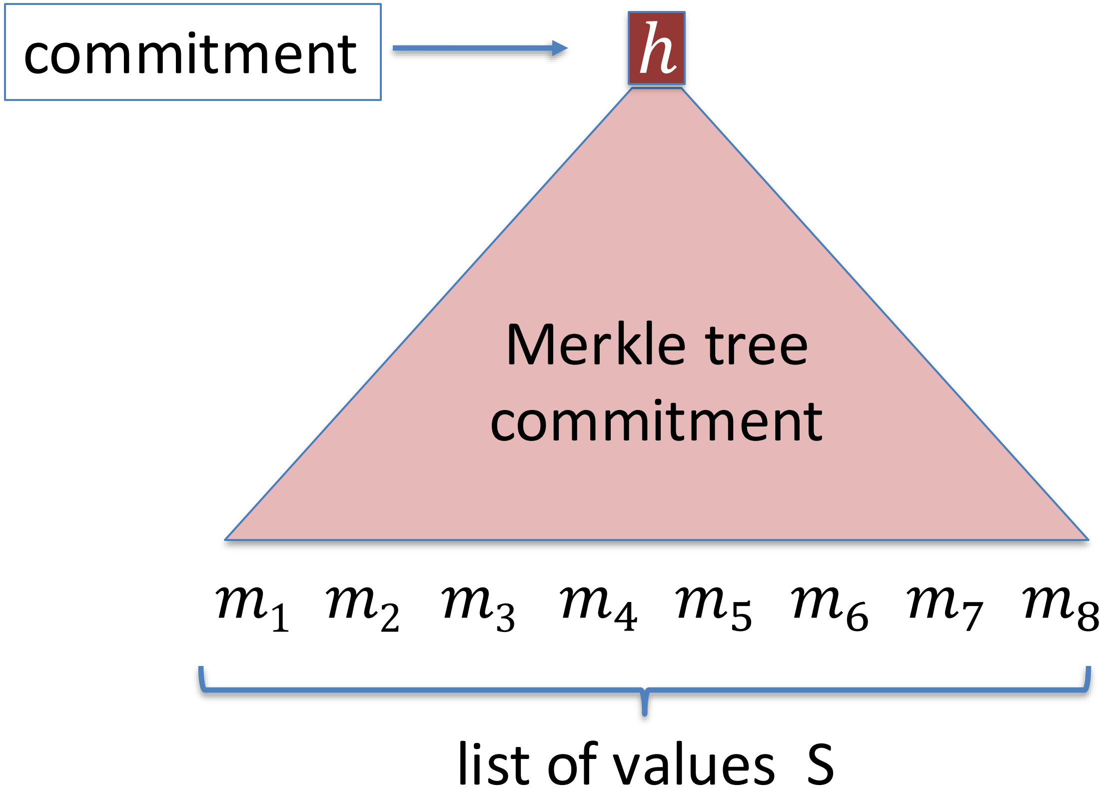
- Alice posts a short binding commitment to S, $h = \text{commit}(S)$
- Bob reads h. Given $(m_i,\ \text{proof}\ \pi_i)$ can check that $S[i] = m_i$

  Bob runs $\text{verify}(h, i, m_i, \pi_i) \rightarrow$ accept/reject

security: adv. cannot find (S, i, m, π) s.t. $m \neq S[i]$ and

$\text{verify}(h, i, m, \pi) = \text{accept}$ where $h = \text{commit}(S)$

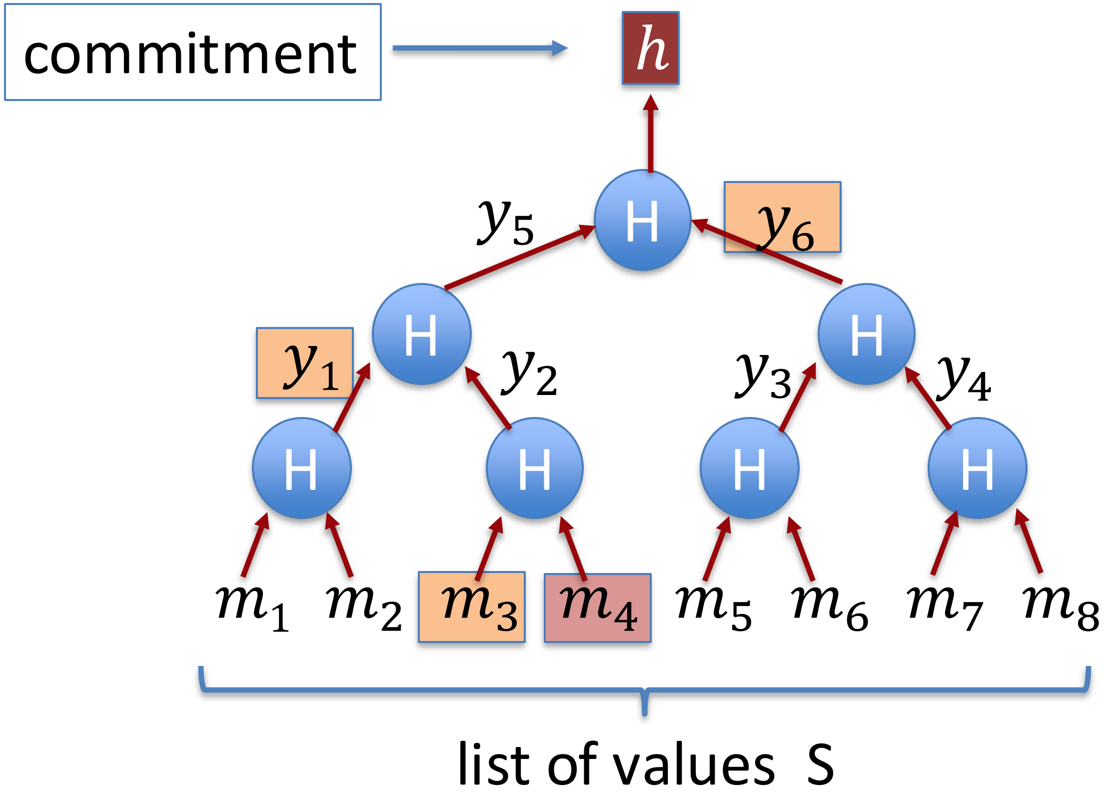
Merkle tree (Merkle 1989)



Goal:

- commit to list S of size n
- Later prove $S[i] = m_i$

Merkle tree (Merkle 1989) [simplified]



Goal:

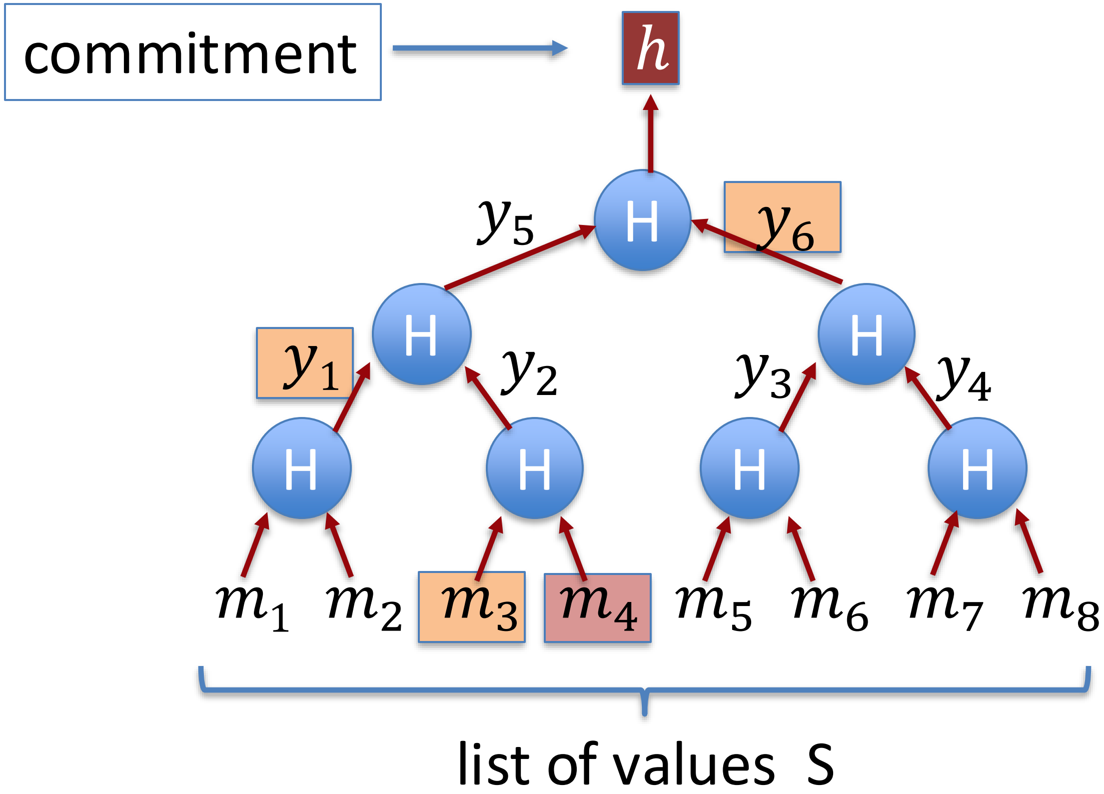
- commit to list S of size n
- Later prove $S[i] = m_i$

To prove $S[4] = m_4$,

proof π $= (m_3, y_1, y_6)$

length of proof: $\log_2 n$

Merkle tree (Merkle 1989) [simplified]



To prove $S[4] = m_4$,

proof $\pi = (m_3, y_1, y_6)$

Bob does:

$y_2 \leftarrow H(m_3, m_4)$

$y_5 \leftarrow H(y_1, y_2)$

$h' \leftarrow H(y_5, y_6)$

accept if $h = h'$

Merkle tree (Merkle 1989)

Thm: For a given n: if H is a CRF then

adv. cannot find (S, i, m, π) s.t. $|\mathrm{S}| = \mathrm{n}$, $m \neq S[i]$,

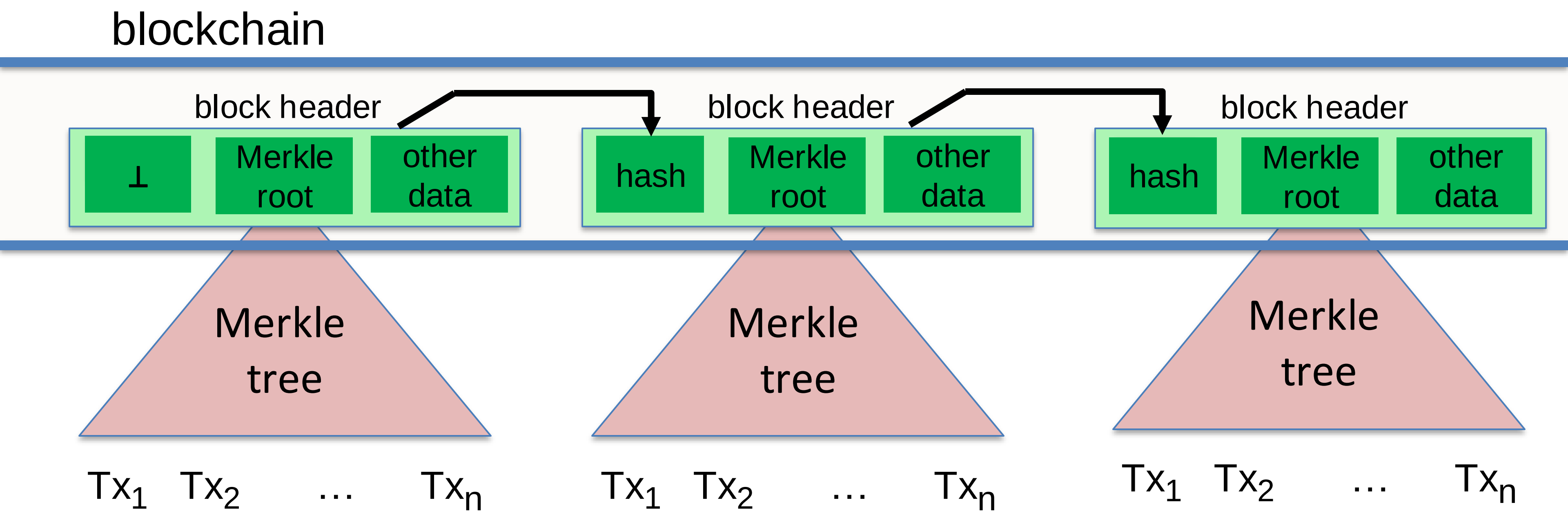
$h = \text{commit}(S)$, and $\text{verify}(h, i, m, \pi) = \text{accept}$

(to prove, prove the contra-positive)

How is this useful? To post a block of transactions S on chain suffices to only write commit(S) to chain. Keeps chain small.

$\Rightarrow$ Later, can prove contents of every Tx.

Abstract block chain



Merkle proofs are used to prove that a Tx is “on the block chain”

Another application: proof of work

Goal: computational problem that

- takes time $\Omega(D)$ to solve, but
- solution takes time O(1) to verify

(D is called the **difficulty**)

How? $H: X \times Y \rightarrow \{0,1,2,\ldots,2^n - 1\}$ e.g. $n = 256$

- puzzle: input $x \in X$, output $y \in Y$ s.t. $H(x,y) < 2^n/D$
- verify(x,y): accept if $\boxed{H(x,y) < 2^n/D}$

Another application: proof of work

<u>Thm</u>: if H is a "random function" then the best algorithm requires D evaluations of H in expectation.

Note: this is a parallel algorithm

$\Rightarrow$ the more machines I have, the faster I solve the puzzle.

Proof of work is used in some consensus protocols (e.g., Bitcoin)

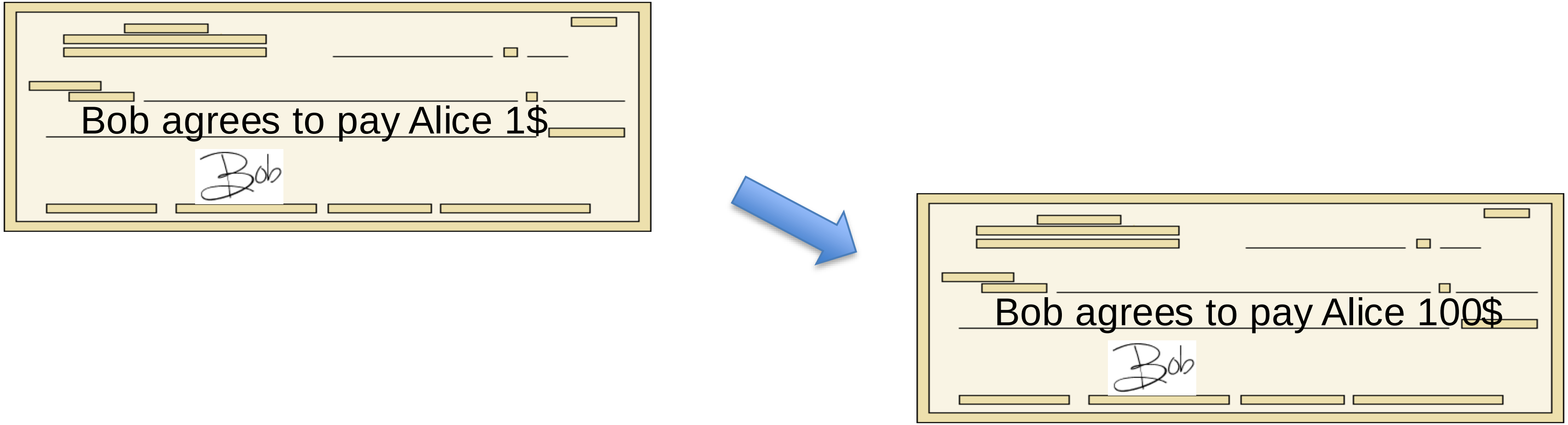
Bitcoin uses $H(x, y) = \mathrm{SHA256}(\mathrm{SHA256}(x.y))$

Today

1. What is a blockchain?
2. Basic introduction to cryptography
    - Hash functions
    - Commitments
    - Merkle trees
    - Signatures
3. Start looking at Bitcoin

Signatures

Physical signatures: bind transaction to author

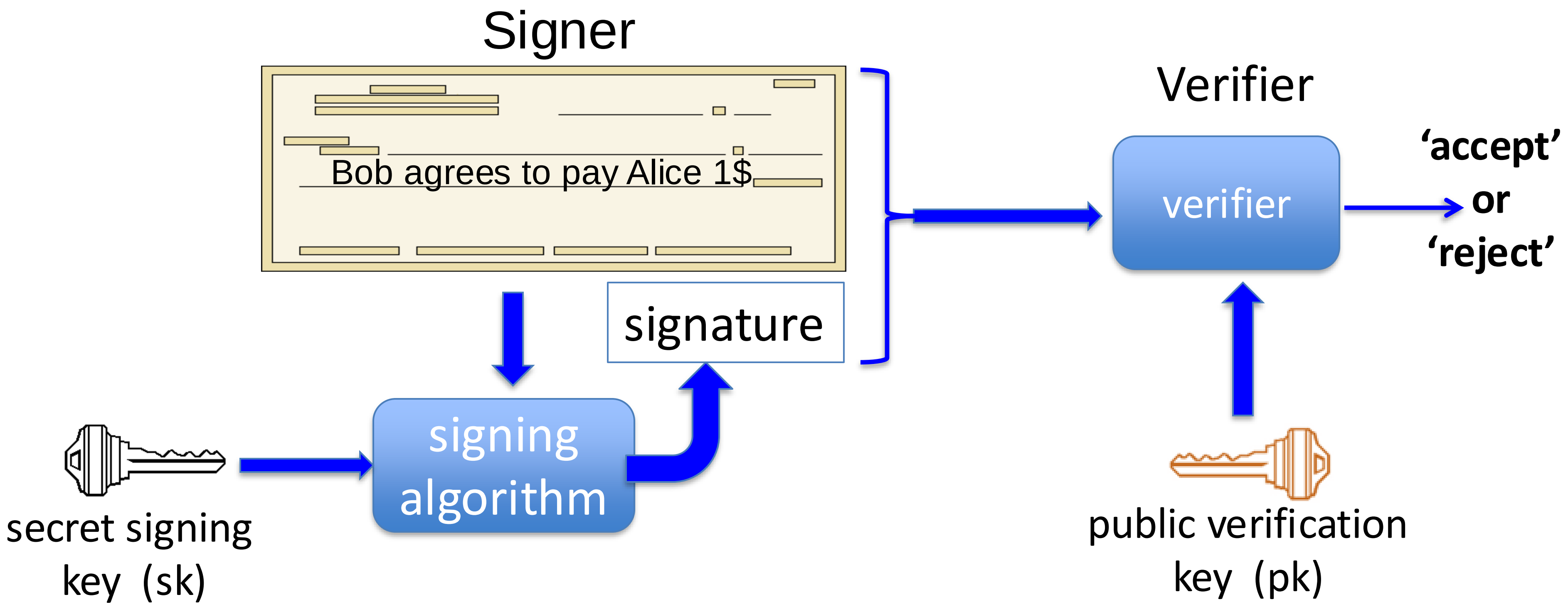


Problem in the digital world:

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 their 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)

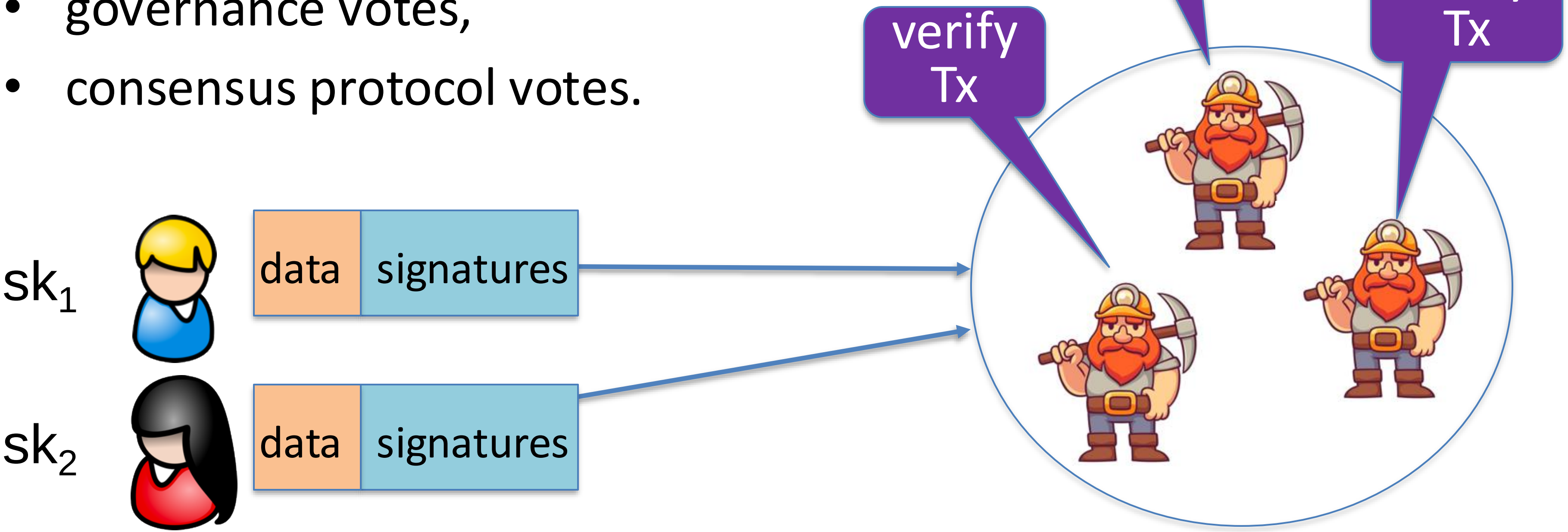
3. <u>BLS signatures</u>: 48 bytes, aggregatable, easy threshold

   (Ethereum 2.0, Chia, Dfinity)

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

Today

1. What is a blockchain?
2. Basic introduction to cryptography
   - Hash functions
   - Commitments
   - Merkle trees
   - Signatures
3. **Start looking at Bitcoin**

Bitcoin mechanics

This lecture: Bitcoin mechanics

user facing tools (cloud servers)

applications (DAPPs, smart contracts)

Execution engine (blockchain computer)

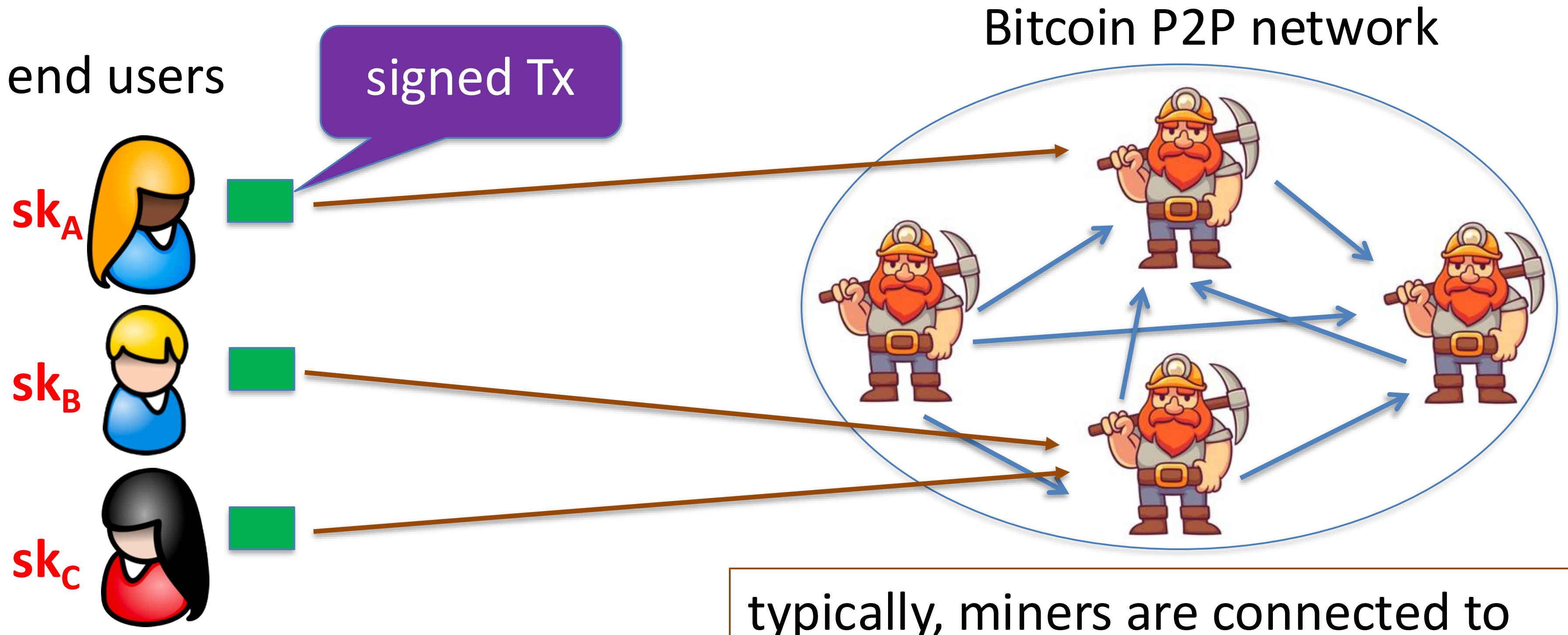
Sequencer: orders transactions

Data Availability / Consensus Layer

start today

next week

First: overview of the Bitcoin consensus layer



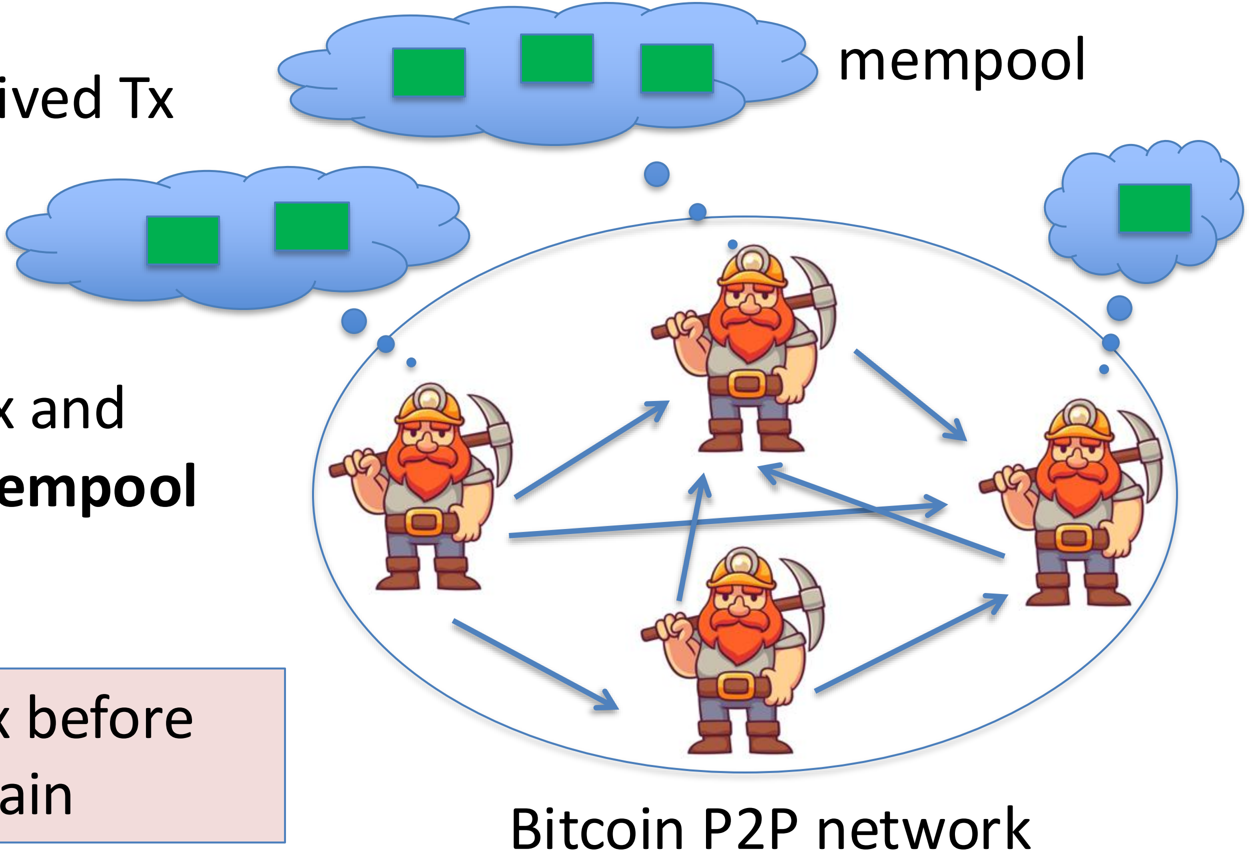
typically, miners are connected to eight other peers (anyone can join)

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

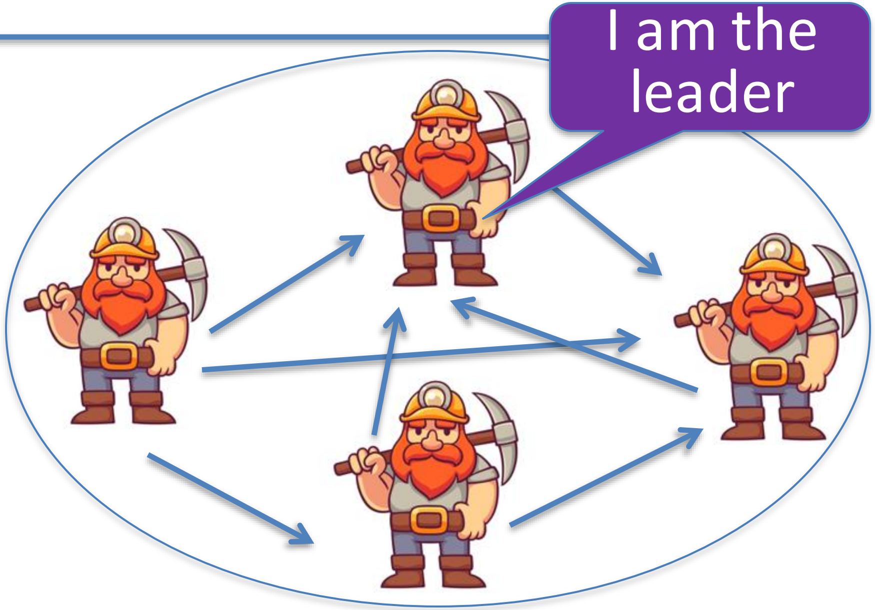


First: overview of the Bitcoin consensus layer

blockchain

<u>Every ≈**10 minutes**</u>:

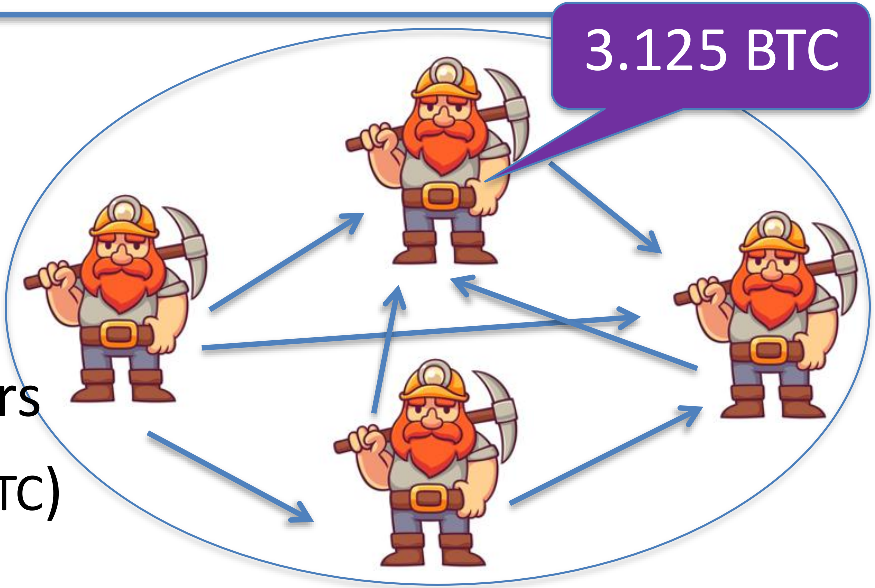
- Each miner creates a candidate block from Tx in its mempool
- a “random” miner is selected (how: next week), and broadcasts its block to P2P network
- all miners validate new block



Bitcoin 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:

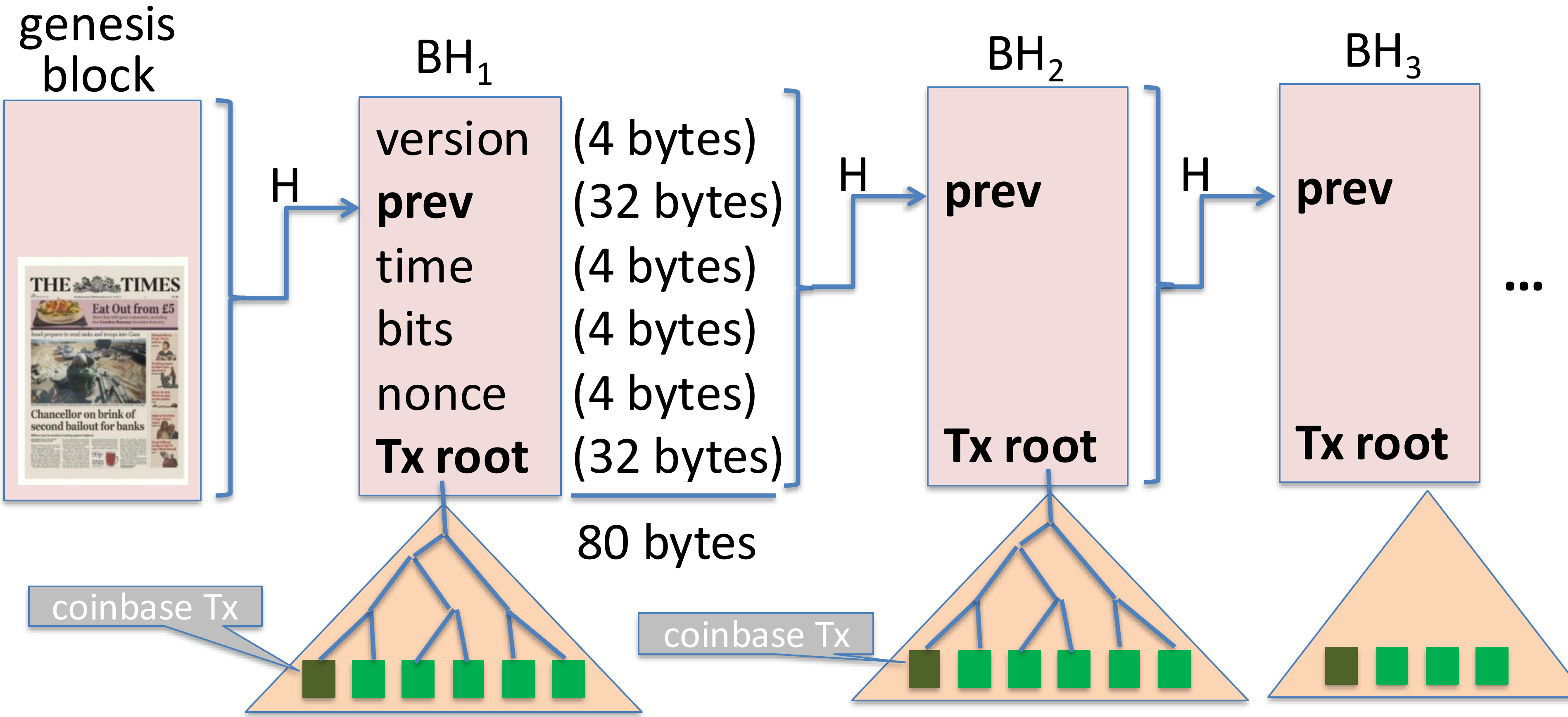
- to remove a block, need to convince 51% of mining power *

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

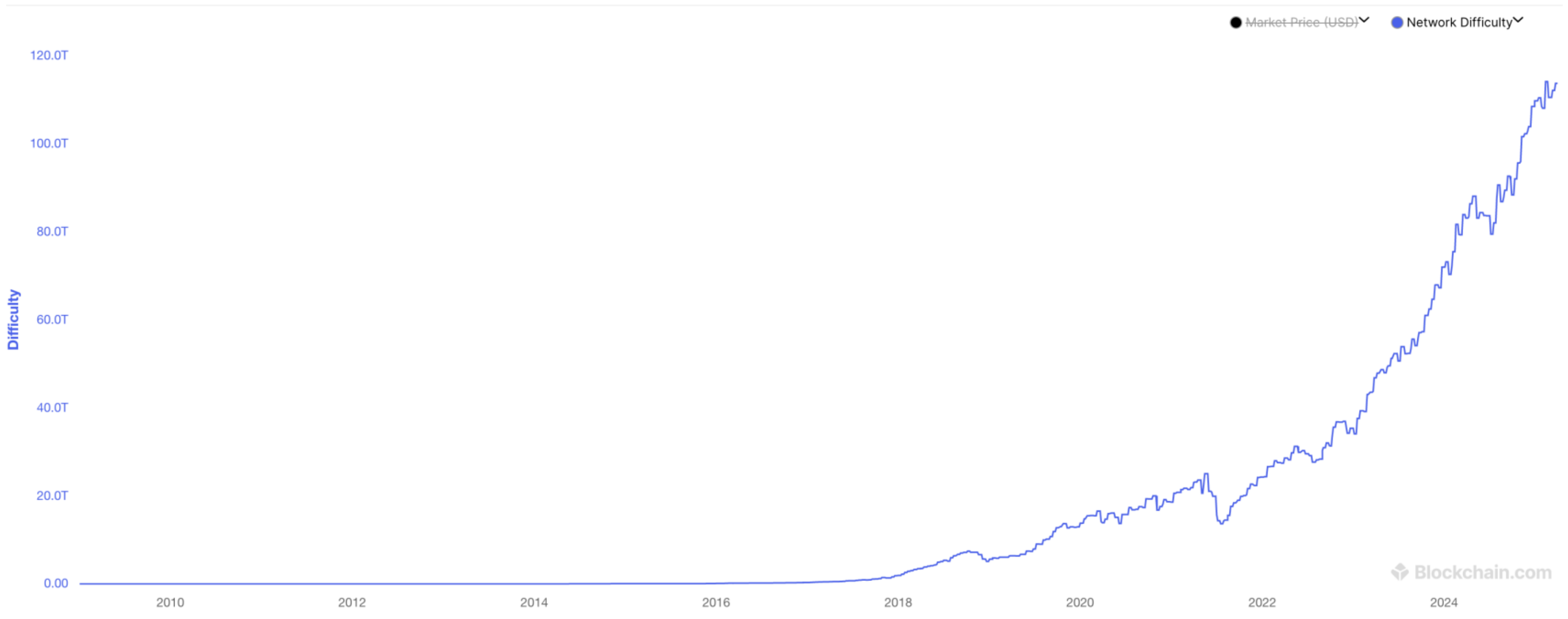
time: time miner assembled the block. Self reported.
(block rejected if too far in past or future)

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.

Difficulty over time



An example

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

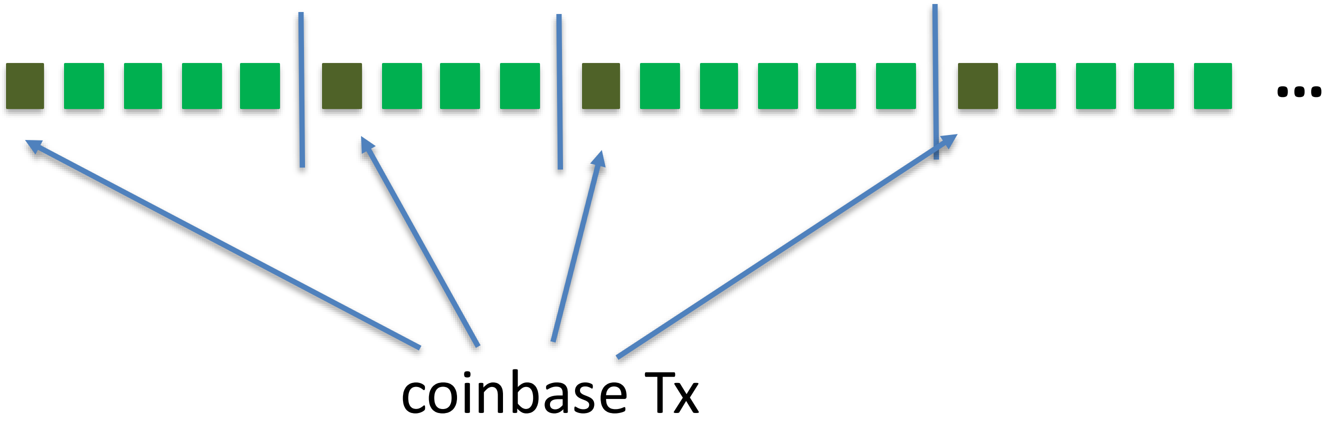
Block 648493

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 was 2020

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:

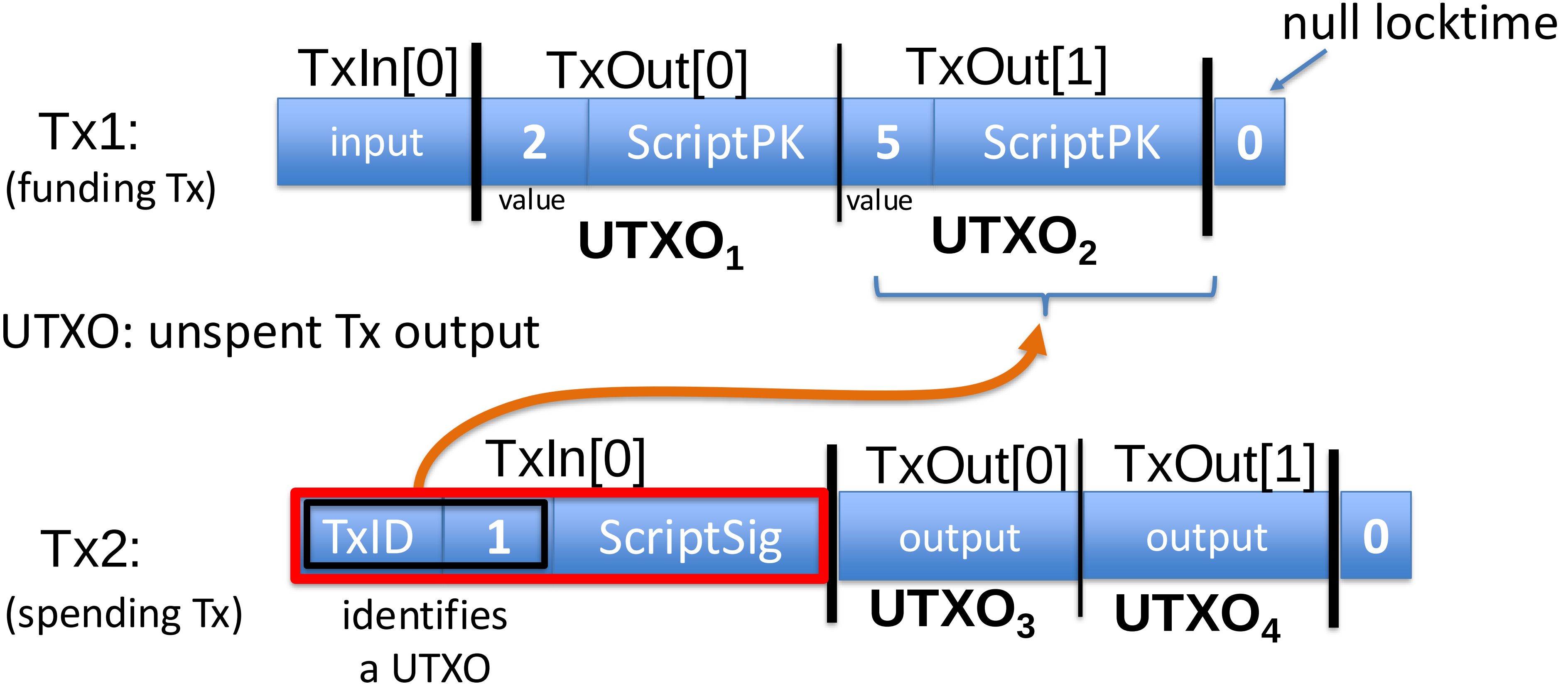
TxID	32 byte hash
out-index	4 byte index
ScriptSig	program
seq	ignore

output:

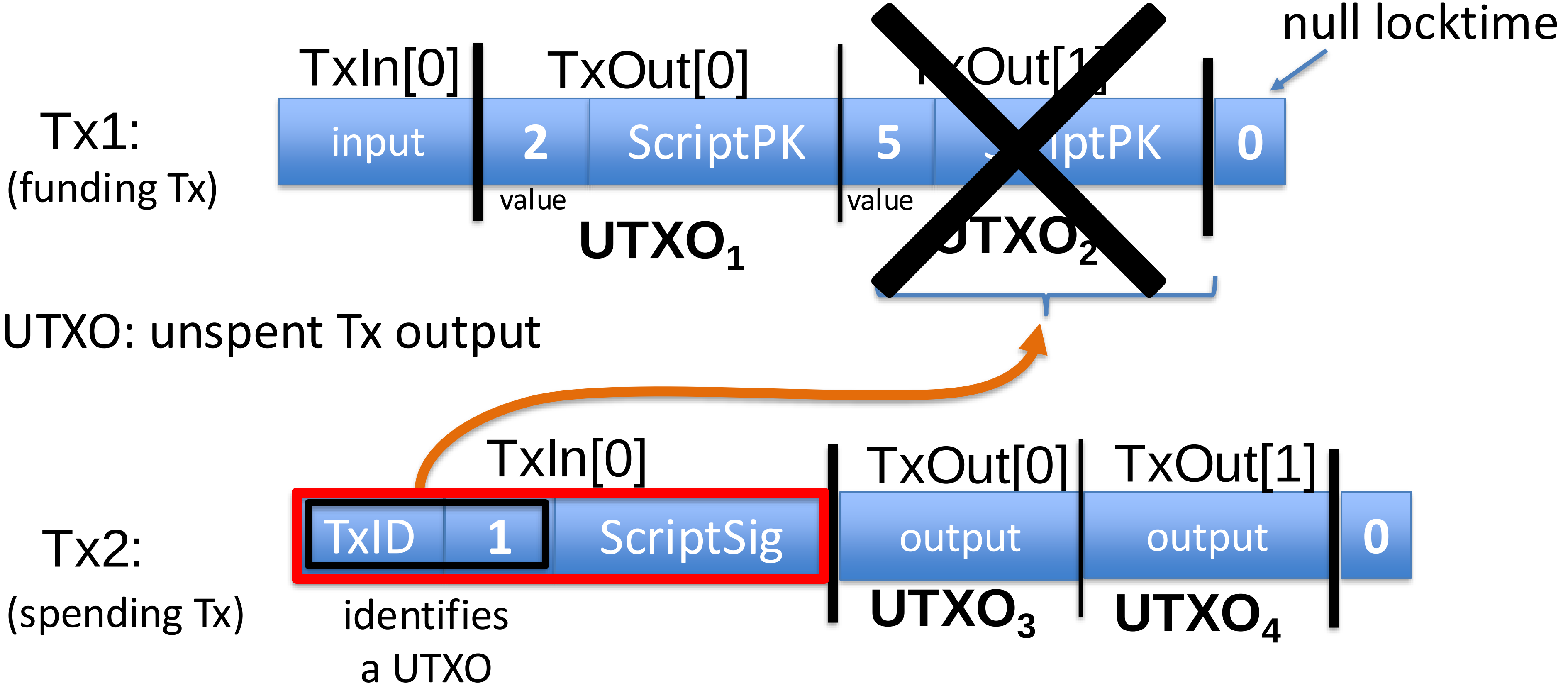
value	8 bytes
ScriptPK	program

#BTC = value/10^8

Example


null locktime
TxIn[0]
TxOut[0]
TxOut[1]
Tx1:
(funding Tx)
input
2
ScriptPK
5
ScriptPK
0
value
value
$UTXO_1$
$UTXO_2$
UTXO: unspent Tx output
TxIn[0]
TxOut[0]
TxOut[1]
Tx2:
(spending Tx)
TxID
1
ScriptSig
output
output
0
identifies
a UTXO
$UTXO_3$
$UTXO_4$

Example



null locktime
Tx1:
(funding Tx)
TxIn[0]
TxOut[0]
input
2
ScriptPK
5
0
value
value
UTXO$_1$
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

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

Miners check (for each input):

1. The program ScriptSig | ScriptPK returns true

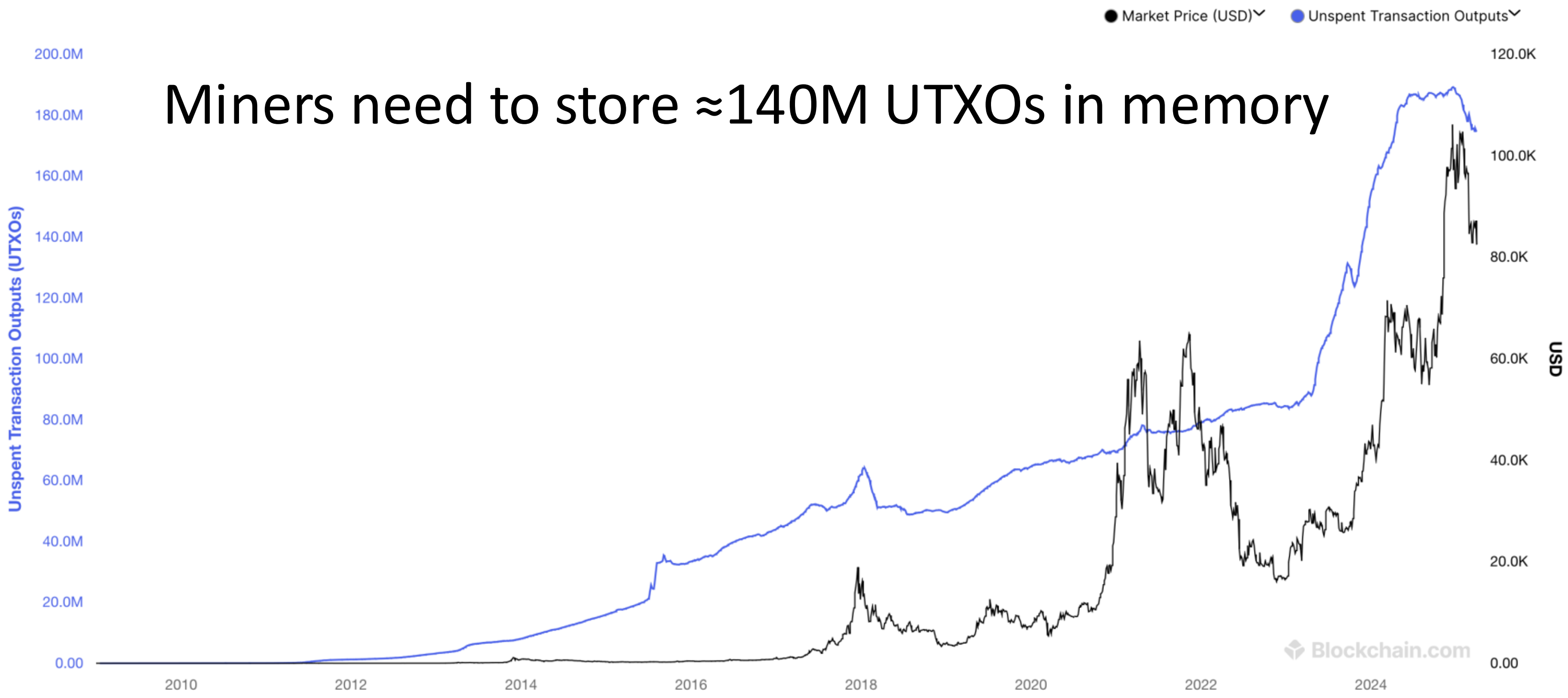
2. TxID | index is in the current UTXO set

3. sum input values $\geq$ sum output values

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

All value in Bitcoin is held in UTXOs

Miners need to store ≈140M UTXOs in memory



An example (block 648493) [2826 Tx]

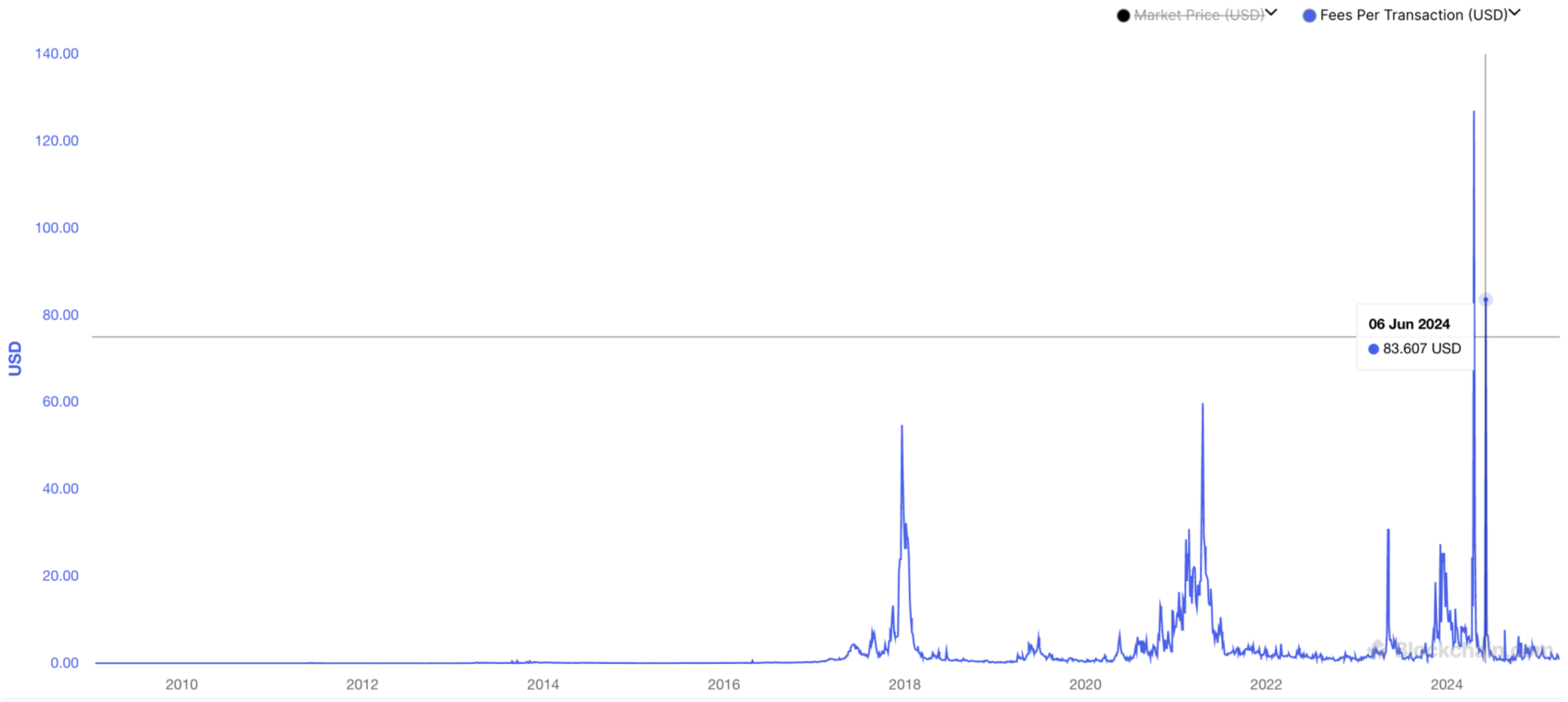
COINBASE (Newly Generated Coins)		→	1CK6KHY6MHgYvmRQ4PAafKYDrg1ejbH1cE	7.14047154 BTC
			OP_RETURN	0.00000000 BTC
Tx0			OP_RETURN	0.00000000 BTC
0.00000000 BTC			6.25 + Tx fees =	7.14047154 BTC

3PuJbxJS1pKxf8EdVR18yBkD1fPAbgUtyw	0.72333974 BTC	→	1E5Ao1VUnA5BhffvXf2Xmud6avUgwkFnJv	0.00917379 BTC
input	(input UTXO value)		bc1qr8k3e0vx06lpu3j7m858pa2ak9tyr56ttwvefk	0.61504199 BTC
			bc1qdrxve8kua3yz5dgx6wf3u95ngh0d3e648...	0.09290152 BTC
Tx1			14ZhjuXpQ5jCDjtAy7ZMu3hfEQCWewzLw7	0.00616444 BTC
0.00005800 BTC (Tx fee)			outputs	0.72328174 BTC

17MWze4Z1uP1jnvqvj7SAnGtxcoVq11H8A	0.05000000 BTC	→	3G3C2RFQ8gsf77EQpdR4ZReChWFKEHhxVU	0.04808000 BTC
Tx2				
0.00192000 BTC (Tx fee)				0.04808000 BTC

sum of fees in block added to coinbase Tx

Tx fees (all time)



Focusing on Tx2: TxIn[0]

from UTXO
(Bitcoin script)

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

from TxIn[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)

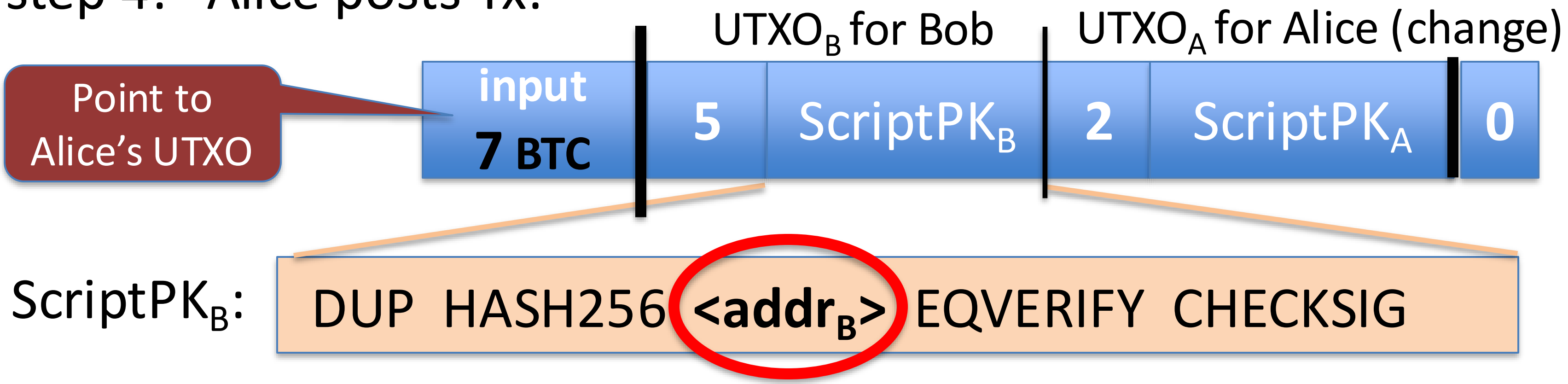
$\Rightarrow$ successful termination

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:



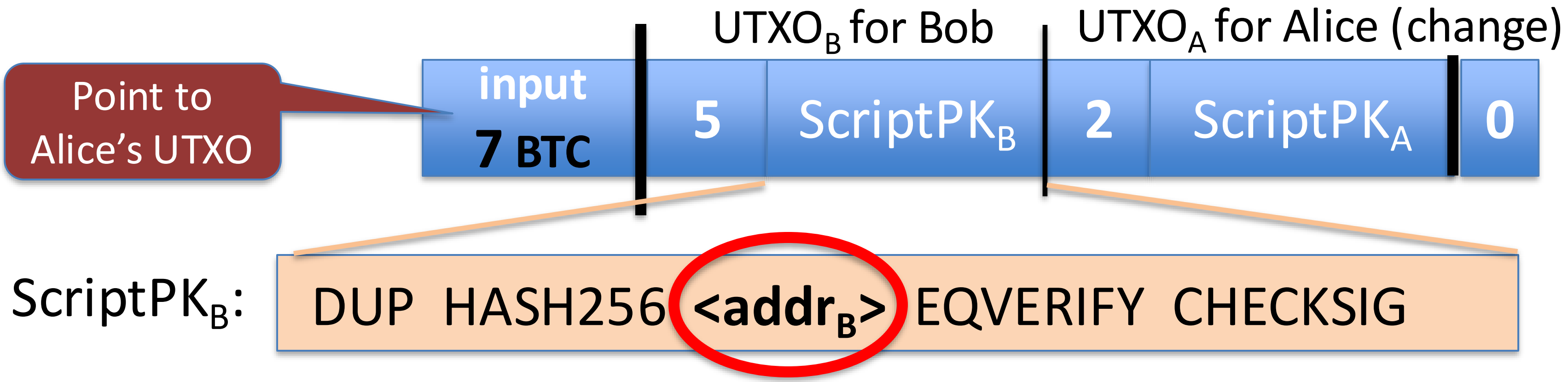
Transaction types: (1) P2PKH

pay to public key hash

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

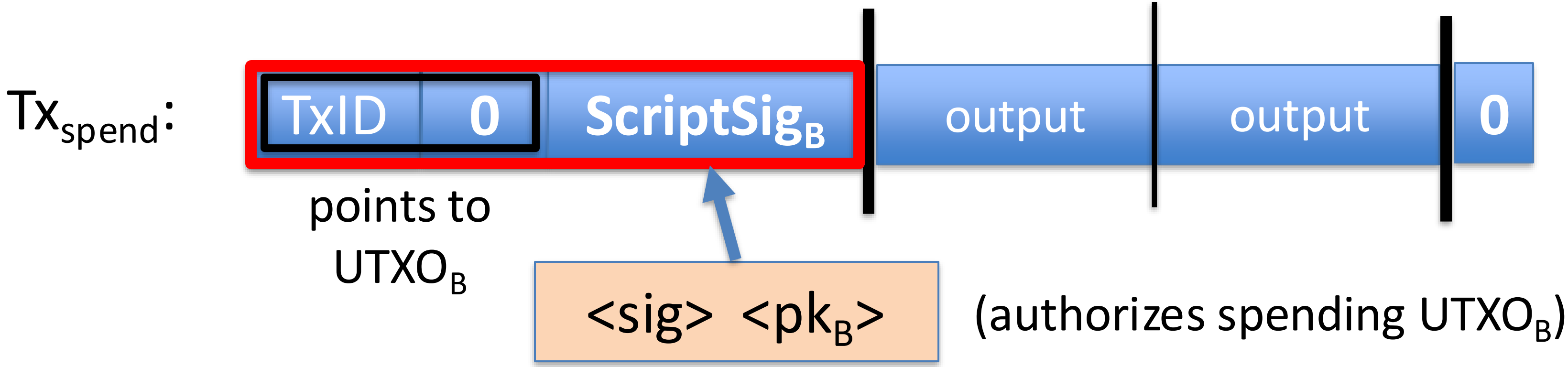
- example: ScriptSig contains Alice's signature on Tx

  $\Longrightarrow$ 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 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

(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 $\longleftarrow$ payee gave correct script

(2) ScriptSig = true $\longleftarrow$ 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>