

Ethereum: mechanics

Yu Feng
University of California, Santa Barbara



Slides adapted from Stanford CS251

New topic: limitations of Bitcoin

Recall: UTXO contains (hash of) ScriptPK

- simple script: indicates conditions when UTXO can be spent

Limitations:

- Difficult to maintain state in multi-stage contracts
- Difficult to enforce global rules on assets

A simple example: rate limiting. My wallet manages 100 UTXOs.

- Desired policy: can only transfer 2BTC per day out of my wallet

An example: DNS

Domain name system on the blockchain: [google.com → IP addr]

Need support for three operations:

- **Name.new(OwnerAddr, DomainName)**: intent to register
- **Name.update(DomainName, newVal, newOwner, OwnerSig)**
- **Name.lookup(DomainName)**

Note: also need to ensure no front-running on **Name.new()**

A broken implementation

Name.new() and Name.update() create a UTXO with ScriptPK:

```
DUP HASH256 <OwnerAddr> EQVERIFY CHECKSIG VERIFY  
<DNS> <DomainName> <IPAddr> <1>
```

only owner can “spend” this UTXO to update domain data

Contract: (should be enforced by miners)

if domain google.com is registered,
no one else can register that domain

Problem: this contract cannot be enforced using Bitcoin script

verify
sig is valid

ensure top
of stack is 1

What to do?

NameCoin: a fork of Bitcoin that implements this contract
(see also the Ethereum Name Service -- ENS)

Can we build a blockchain that natively supports generic contracts like this?

⇒ Ethereum



Ethereum: enables a world of applications

A world of Ethereum Decentralized apps (DAPPs)

- New coins: ERC-20 standard interface
- DeFi: exchanges, lending, stablecoins, derivatives, etc.
- Insurance
- DAOs: decentralized organizations
- NFTs: Managing asset ownership (ERC-721 interface)

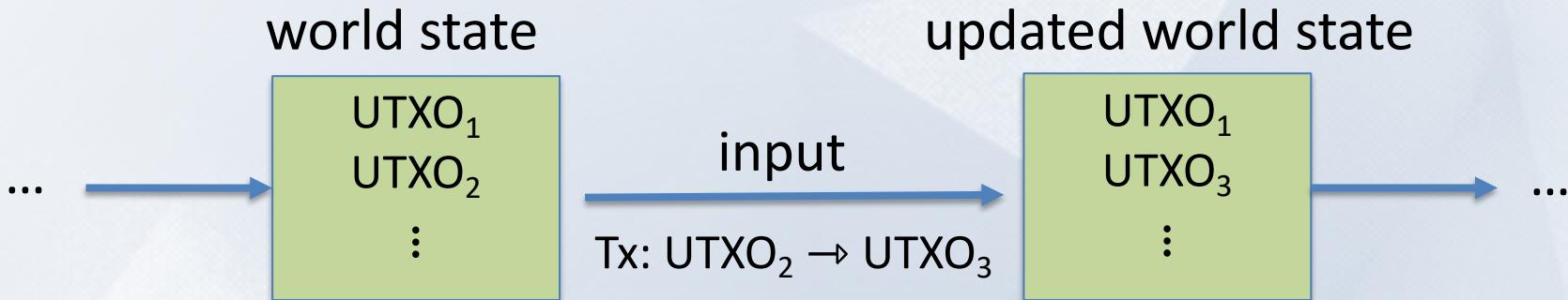
⋮



30

stateofthedapps.com, dapp.review

Bitcoin as a state transition system



Bitcoin rules:

$$F_{\text{bitcoin}} : S \times I \rightarrow S$$

S: set of all possible world states, $s_0 \in S$ genesis state

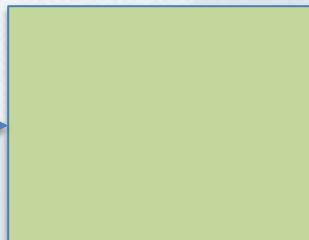
I: set of all possible inputs

Ethereum as a state transition system

Much richer state transition functions

⇒ one transition executes an entire program

Ethereum
world state



updated Ethereum
world state



...

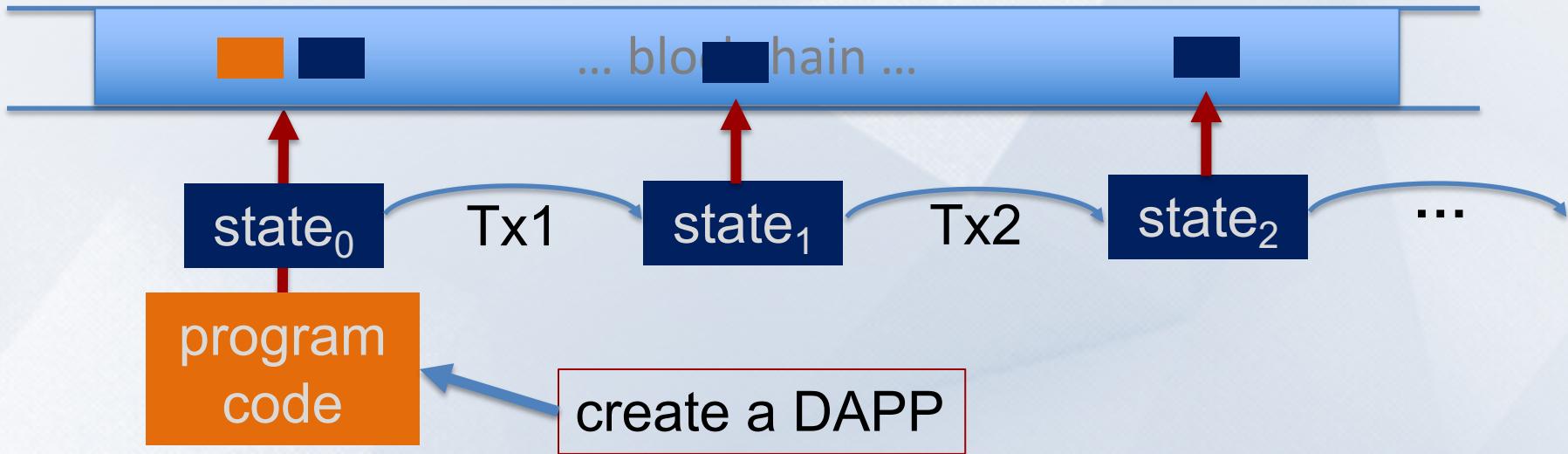


input



Tx

Running a program on a blockchain (DAPP)



compute layer (execution chain): The EVM

consensus layer (beacon chain)

The Ethereum system

Proof-of-Stake consensus

Block	Age	Txn	Fee Recipient
15764027	4 secs ago	91	Fee Recipient: 0x467...263
15764026	16 secs ago	26	0xedc7ec654e305a38ffff...
15764025	28 secs ago	165	bloXroute: Max Profit Bui...
15764024	40 secs ago	188	Lido: Execution Layer Re...
15764023	52 secs ago	18	Fee Recipient: 0xeBe...Acf
15764022	1 min ago	282	0xd4e96ef8eee8678dbff...
15764021	1 min ago	295	0xbb3afde35eb9f5feb53...
15764020	1 min ago	71	Fee Recipient: 0x6d2...766

One block every 12 seconds.
about 150 Tx per block.

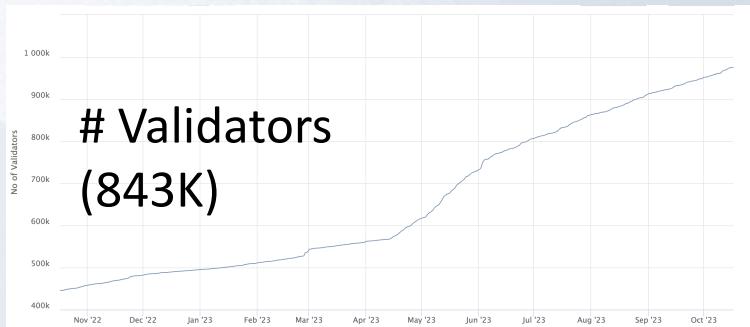
Block proposer receives
Tx fees for block
(along with other rewards)

A bit about the beacon chain (Eth2 consensus)

To become a validator: stake (lock up) 32 ETH ... or use Lido.

Validators:

- sign blocks to express correctness (finalized once enough sigs)
- occasionally act as *block proposer* (chosen at random)
- correct behavior \Rightarrow issued new ETH every epoch (32 blocks)
- incorrect behavior \Rightarrow slashed (lots of details)



The economics of staking

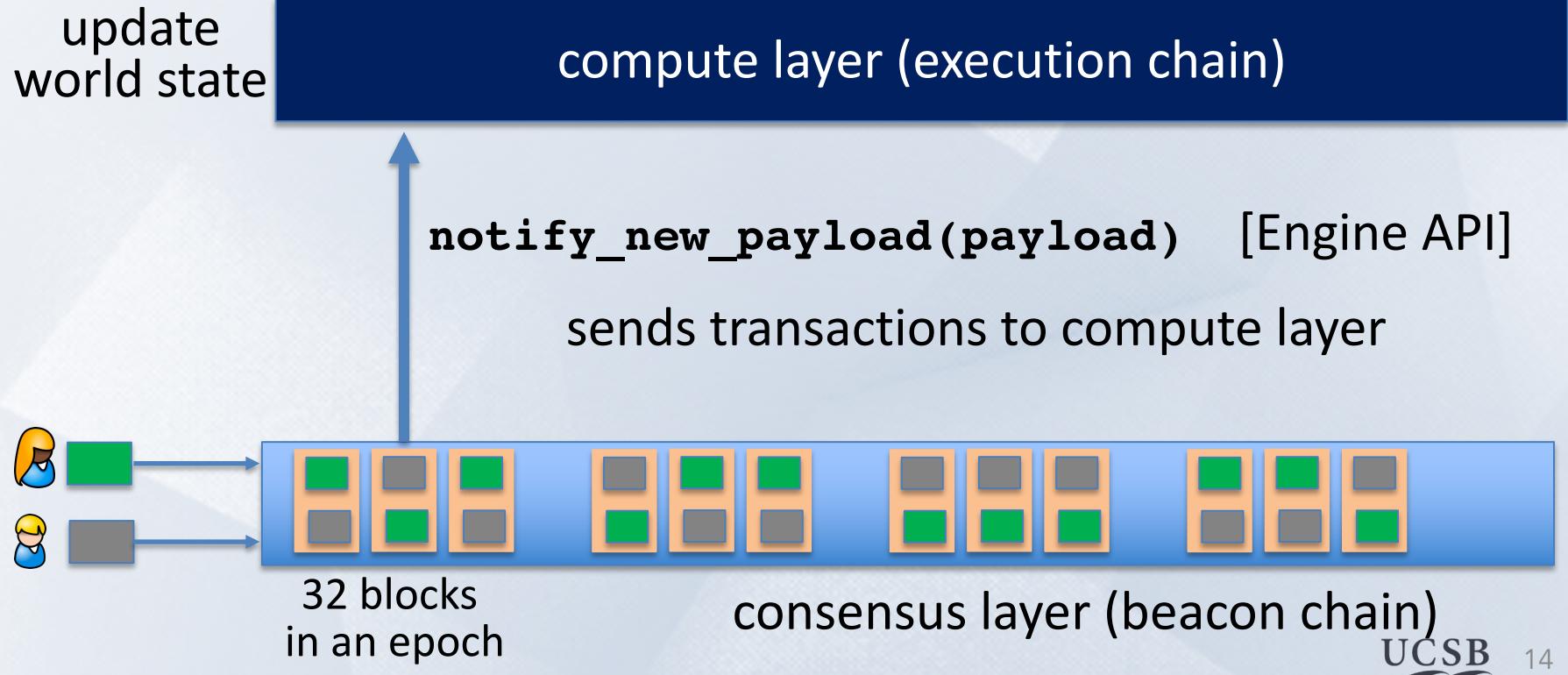
Validator locks up 32 ETH. Oct 2023: 27M ETH staked
(total)

Annual validator income (an example):

- Issuance: 1.0 ETH
 - Tx fees: 0.4 ETH
 - MEV: 0.4 ETH
 - Total: 1.8 ETH (5.6% return on 32 ETH staked)
-
- Can be adjusted
(BASE_REWARD_FACTOR)
- A function of
congestion

In practice: staking provider (e.g., Lido) takes a cut of the returns

The Ethereum system



The Ethereum Compute Layer: The EVM



Ethereum compute layer: the EVM

World state: set of accounts identified by 32-byte address.

Two types of accounts:

(1) externally owned accounts (EOA):

controlled by ECDSA signing key pair (pk, sk).

sk : signing key known only to account owner

(2) contracts: controlled by code.

code set at account creation time, does not change

Data associated with an account

Account data

Owned (EOA)

Contracts

address (computed): $H(pk)$ $H(\text{CreatorAddr}, \text{CreatorNonce})$

code: \perp CodeHash

storage root (state): \perp StorageRoot

balance (in Wei): balance balance (1 Wei = 10^{-18} ETH)

nonce: nonce nonce

(#Tx sent) + (#accounts created): anti-replay mechanism

Account state: persistent storage

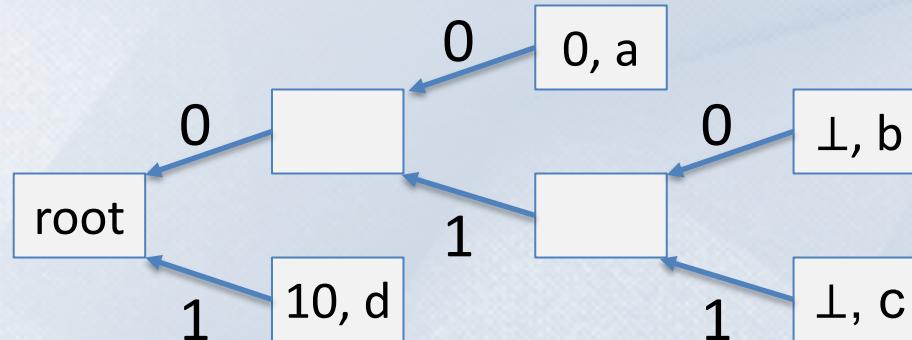
Every contract has an associated **storage array** $S[]$:

$S[0], S[1], \dots, S[2^{256}-1]$: each cell holds 32 bytes, init to 0.

Account storage root: Merkle Patricia Tree hash of $S[]$

- Cannot compute full Merkle tree hash: 2^{256} leaves

$S[000] = a$
 $S[010] = b$
 $S[011] = c$
 $S[110] = d$



time to compute root hash:
 $\leq 2 \times |S|$
 $|S| = \# \text{ non-zero cells}$

State transitions: Tx and messages

Transactions: signed data by initiator

- **To:** 32-byte address of target ($0 \rightarrow$ create new account)
- **From, [Signature]:** initiator address and signature on Tx (if owned)
- **Value:** # Wei being sent with Tx ($1 \text{ Wei} = 10^{-18} \text{ ETH}$)
- Tx fees (EIP 1559): **gasLimit, maxFee, maxPriorityFee** (later)
- if $\text{To} = 0$: create new contract **code = (init, body)**
- if $\text{To} \neq 0$: **data** (what function to call & arguments)
- **nonce**: must match current nonce of sender (prevents Tx replay)
- **chain_id**: ensures Tx can only be submitted to the intended chain

State transitions: Tx and messages

Transaction types:

owned → owned: transfer ETH between users

owned → contract: call contract with ETH & data

Example (block #10993504)

<u>From</u>	<u>To</u>	<u>msg.value</u>	<u>Tx fee (ETH)</u>
0xa4ec1125ce9428ae5...	0x2cebe81fe0dcd220e...	0 Ether	0.00404405
0xba272f30459a119b2...	Uniswap V2: Router 2	0.14 Ether	0.00644563
0x4299d864bbda0fe32...	Uniswap V2: Router 2	89.839104111882671 Ether	0.00716578
0x4d1317a2a98fea41...	0xc59f33af5f4a7c8647...	14.501 Ether	0.001239
0x29ecaa773f052d14e...	CryptoKitties: Core	0 Ether	0.00775543
0x63bb46461696416fa...	Uniswap V2: Router 2	0.203036474328481 Ether	0.00766728
0xde70238aef7a35abd...	Balancer: ETH/DOUGH...	0 Ether	0.00261582
0x69aca10fe1394d535f...	0x837d03aa7fc09b8be...	0 Ether	0.00259936
0xe2f5d180626d29e75...	Uniswap V2: Router 2	0 Ether	0.00665809

Messages: virtual Tx initiated by a contract

Same as Tx, but no signature (contract has no signing key)

contract → owned: contract sends funds to user

contract → contract: one program calls another (and sends funds)

One Tx from user: can lead to many Tx processed. Composability!

Tx from owned addr → contract → another contract



another contract → different owned

Example Tx

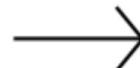
State

14c5f8ba: owned
- 1024 eth

bb75a980: contract
- 5202 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, 0, ALICE ...]

892bf92f: contract
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]

4096ad65: owned
- 77 eth



Transaction

From: 14c5f8ba
To: bb75a980
Value: 10 eth
Data: 2,
CHARLIE
Sig: 30452fdedb3d
f7959f2ceb8a1



State'

14c5f8ba:
- 1014 eth

bb75a980:
- 5212 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, CHARLIE, ALICE ...]

892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])

[ALICE, BOB, CHARLIE]

4096ad65:
- 77 eth

world state (four accounts)

updated world state

An Ethereum Block

Block proposer creates a block of n Tx: (from Tx submitted by users)

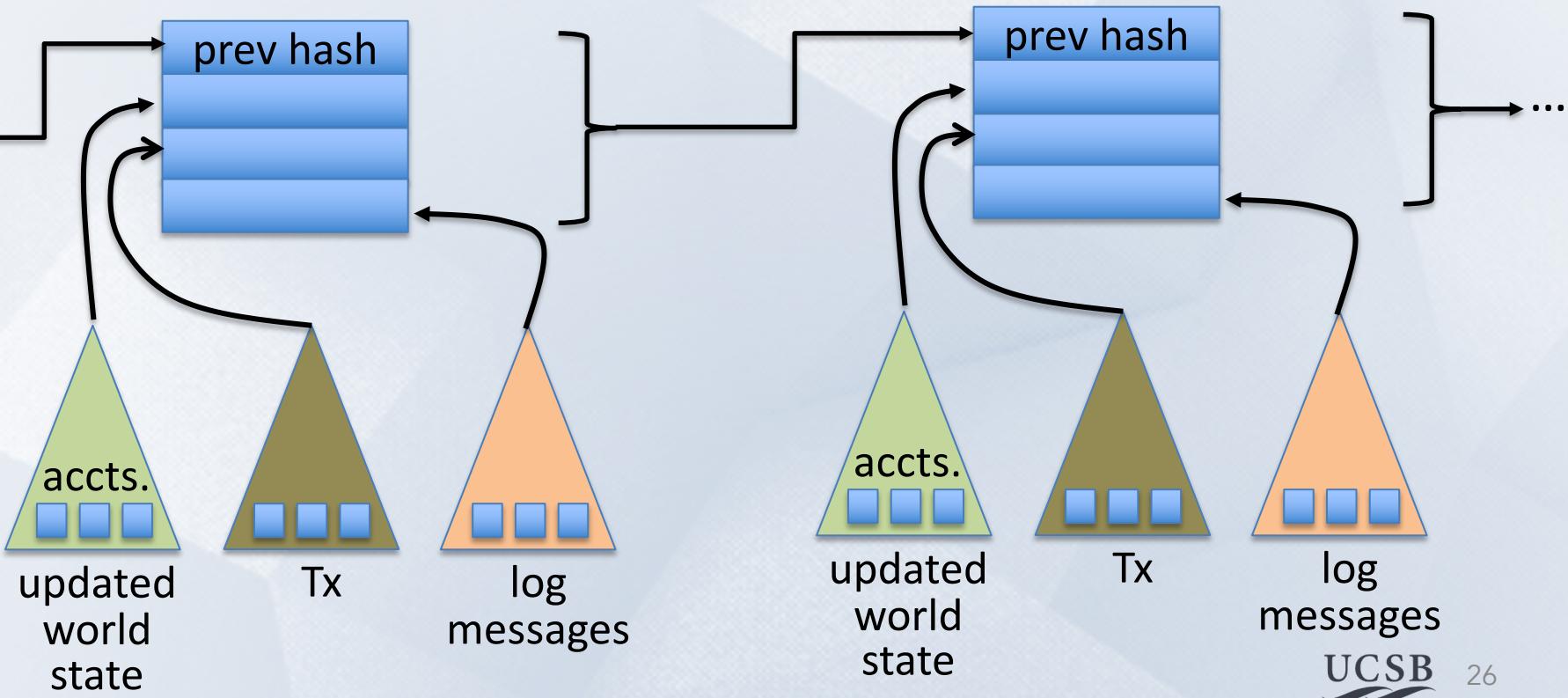
- To produce a block do:
 - for $i=1, \dots, n$: execute state change of Tx_i sequentially
(can change state of $>n$ accounts)
 - record updated world state in block

Other validators re-execute all Tx to verify block \Rightarrow
sign block if valid \Rightarrow enough sigs, epoch is finalized.

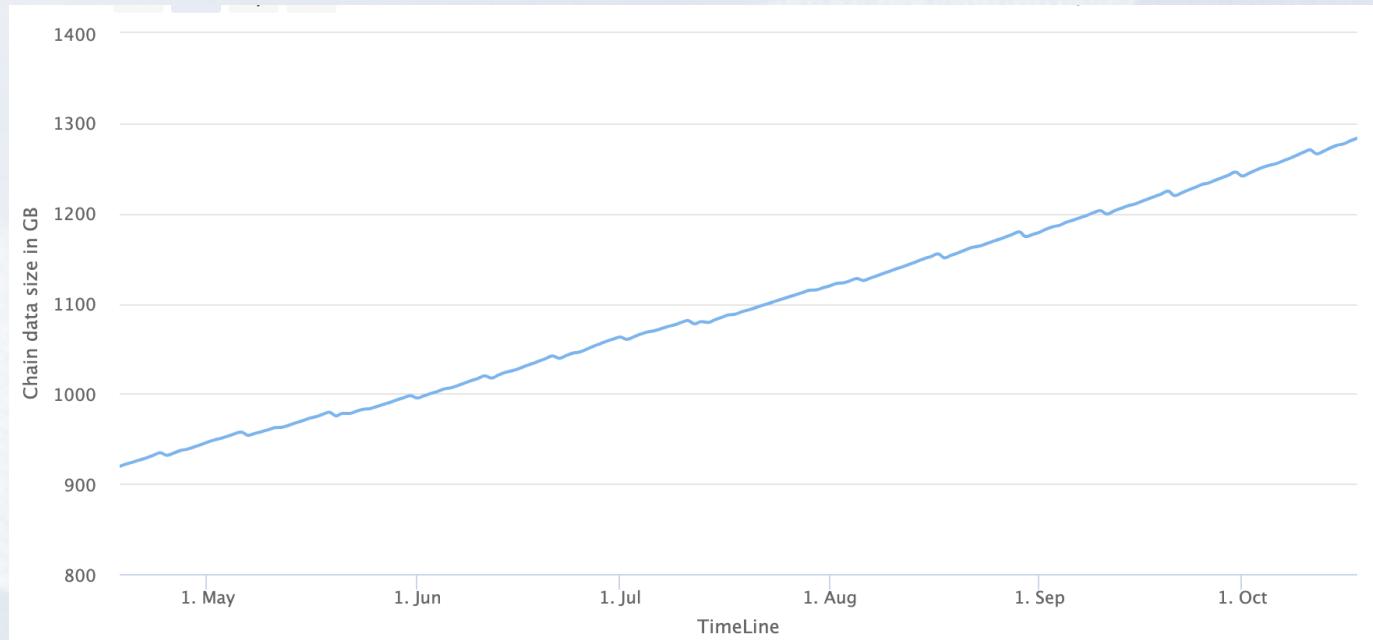
Block header data (simplified)

- (1) consensus data: proposer ID, parent hash, votes, etc.
- (2) address of gas beneficiary: where Tx fees will go
- (3) world state root: updated world state
 - Merkle Patricia Tree hash of all accounts in the system
- (4) Tx root: Merkle hash of all Tx processed in block
- (5) Tx receipt root: Merkle hash of log messages generated in block
- (5) Gas used: used to adjust gas price (target 15M gas per block)

The Ethereum blockchain: abstractly



Amount of memory to run a node



≈1.3 TB

ETH total blockchain size (archival): 16 TB (Oct. 2023)

An example contract: NameCoin

```
contract nameCoin {      // Solidity code (next lecture)

    struct nameEntry {
        address owner;    // address of domain owner
        bytes32 value;    // IP address
    }

    // array of all registered domains
    mapping (bytes32 => nameEntry) data;
```

An example contract: NameCoin

```
function nameNew(bytes32 name) {  
    // registration costs is 100 Wei  
  
    if (data[name] == 0 && msg.value >= 100) {  
        data[name].owner = msg.sender // record domain owner  
        emit Register(msg.sender, name) // log event  
    }  
}
```



Code ensures that no one can take over a registered name

Serious bug in this code! Front running. Solved using commitments.

An example contract: NameCoin

```
function nameUpdate(  
    bytes32 name, bytes32 newValue, address newOwner) {  
  
    // check if message is from domain owner,  
    //         and update cost of 10 Wei is paid  
  
    if (data[name].owner == msg.sender && msg.value >= 10) {  
  
        data[name].value = newValue;          // record new value  
        data[name].owner = newOwner;          // record new owner  
    }  
}
```

An example contract: NameCoin

```
function nameLookup(bytes32 name) {  
    return data[name];  
}  
  
} // end of contract
```

Used by other contracts
Humans do not need this
(use etherscan.io)

EVM mechanics: execution environment

Write code in Solidity (or another front-end language)

- ⇒ compile to EVM bytecode
 - (some projects use WASM or BPF bytecode)
- ⇒ validators use the EVM to execute contract bytecode in response to a Tx

The EVM

Stack machine (like Bitcoin) but with JUMP

- max stack depth = 1024
- program aborts if stack size exceeded; block proposer keeps gas
- contract can create or call another contract

In addition: two types of zero initialized memory

- Persistent storage (on blockchain): SLOAD, SSTORE (expensive)
- Volatile memory (for single Tx): MLOAD, MSTORE (cheap)
- LOG0(data): write data to log

see <https://www.evm.codes>

Every instruction costs gas, examples:

SSTORE `addr` (32 bytes), `value` (32 bytes)

- zero → non-zero: 20,000 gas
- non-zero → non-zero: 5,000 gas (for a cold slot)
- non-zero → zero: 15,000 gas refund (example)

Refund is given for reducing size of blockchain state

CREATE : 32,000 + 200 × (code size) gas;

CALL `gas`, `addr`, `value`, `args`

SELFDESTRUCT `addr`: kill current contract (5000 gas)

Gas calculation

Why charge gas?

- Tx fees (gas) prevents submitting Tx that runs for many steps.
- During high load: block proposer chooses Tx from mempool that maximize its income.

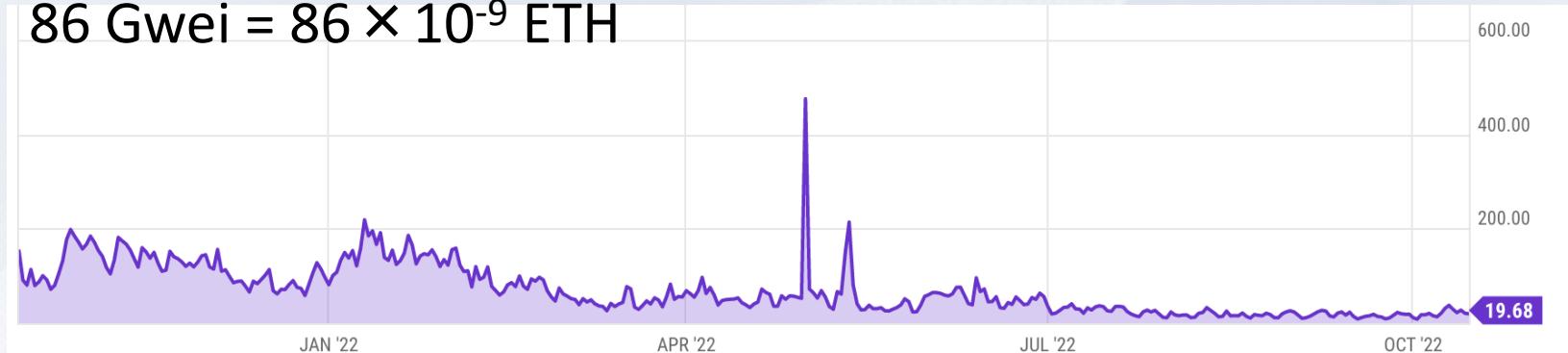
Old EVM: (prior to EIP1559, live on 8/2021)

- Every Tx contains a gasPrice “bid” (gas → Wei conversion price)
- Producer chooses Tx with highest gasPrice (max $\text{sum}(\text{gasPrice} \times \text{gasLimit})$)
⇒ not an efficient auction mechanism (first price auction)

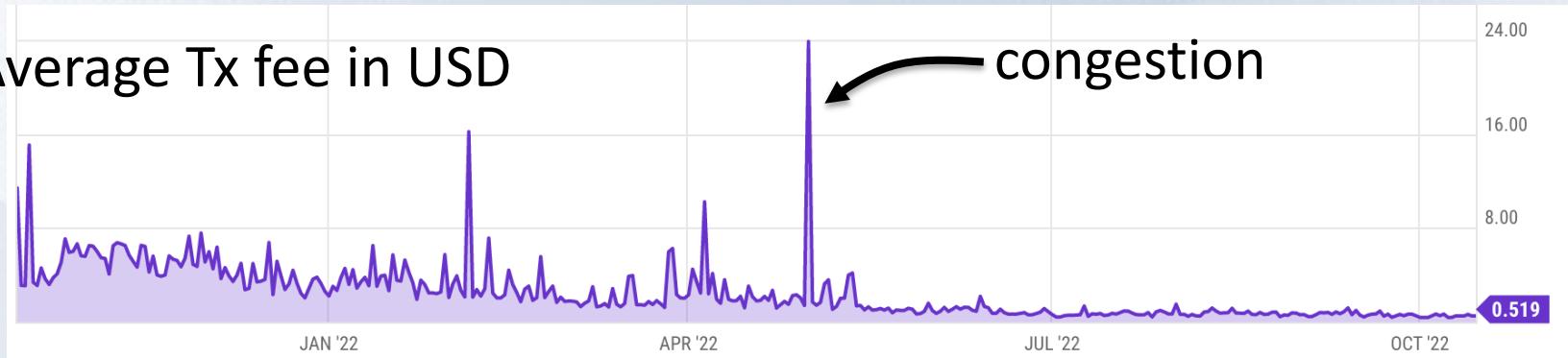
Gas prices spike during congestion

GasPrice in Gwei:

$$86 \text{ Gwei} = 86 \times 10^{-9} \text{ ETH}$$



Average Tx fee in USD



Gas calculation: EIP1559 (since 8/2021)

EIP1559 goals (informal):

- users incentivized to bid their true utility for posting Tx,
- block proposer incentivized to not create fake Tx, and
- disincentivize off chain agreements.

[Transaction Fee Mechanism Design, by T. Roughgarden, 2021]

Gas calculation: EIP1559

Every block has a “baseFee”:

the minimum gasPrice for all Tx in the block

baseFee is computed from total gas in earlier blocks:

- earlier blocks at gas limit (30M gas) \Rightarrow base fee goes up 12.5%
- earlier blocks empty \Rightarrow base fee decreases by 12.5% } interpolate in between

If earlier blocks at “target size” (15M gas) \Rightarrow base fee does not change

Gas calculation

EIP1559 Tx specifies three parameters:

- **gasLimit**: max total gas allowed for Tx
- **maxFee**: maximum allowed gas price (max gas → Wei conversion)
- **maxPriorityFee**: additional “tip” to be paid to block proposer

Computed **gasPrice** bid:

```
gasPrice ← min(maxFee, baseFee + maxPriorityFee)
```

Max Tx fee: **gasLimit** × **gasPrice**

Gas calculation (informal)

`gasUsed` \leftarrow gas used by Tx

Send $\text{gasUsed} \times (\text{gasPrice} - \text{baseFee})$ to block proposer

BURN $\text{gasUsed} \times \text{baseFee}$



\Rightarrow total supply of ETH can decrease

Gas calculation

- (1) if $\text{gasPrice} < \text{baseFee}$: abort
 - (2) If $\text{gasLimit} \times \text{gasPrice} < \text{msg.sender.balance}$: abort
 - (3) deduct $\text{gasLimit} \times \text{gasPrice}$ from $\text{msg.sender.balance}$
-
- (4) set $\text{Gas} \leftarrow \text{gasLimit}$
 - (5) execute Tx: deduct gas from Gas for each instruction
if at end ($\text{Gas} < 0$): abort, Tx is invalid (proposer keeps $\text{gasLimit} \times \text{gasPrice}$)
 - (6) Refund $\text{Gas} \times \text{gasPrice}$ to $\text{msg.sender.balance}$
-
- (7) $\text{gasUsed} \leftarrow \text{gasLimit} - \text{Gas}$
 - (7a) BURN $\text{gasUsed} \times \text{baseFee}$
 - (7b) Send $\text{gasUsed} \times (\text{gasPrice} - \text{baseFee})$ to block producer



Example baseFee and effect of burn

block #	gasUsed	baseFee (Gwei)	ETH burned
15763570	21,486,058	16.92 ↓	0.363
15763569	14,609,185 (<15M)	16.97	0.248
15763568	25,239,720	15.64	0.394
15763567	29,976,215	13.90 ↓	0.416
15763566	14,926,172 (<15M)	13.91 ↓	0.207
15763565	1,985,580 (<15M)	15.60	0.031

beacon chain  $\approx \text{gasUsed} \times \text{baseFee}$

baseFee < 16Gwei \Rightarrow new issuance > burn \Rightarrow ETH inflates
baseFee > 16Gwei \Rightarrow new issuance < burn \Rightarrow ETH deflates

Why burn ETH ???

Recall: EIP1559 goals (informal)

- users incentivized to bid their true utility for posting Tx,
- block proposer incentivized to not create fake Tx, and
- disincentivize off chain agreements.

Suppose no burn (i.e., baseFee given to block producer):

⇒ in periods of low Tx volume proposer would try to increase volume by offering to refund the baseFee *off chain* to users.

Note: transactions are becoming more complex

Total Gas Usage

Evolution of the total gas used by the Ethereum network per day



Gas usage is increasing ⇒ each Tx takes more instructions to execute

END OF LECTURE

Next lecture: writing Solidity contracts

