

Ethereum: mechanics

以太坊: 机制

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: NameCoin

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.upate() create a UTXO with ScriptPK:

```
DUP HASH256 <OwnerAddr> EQVERIFY CHECKSIG VERIFY  
<NAMECOIN> <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

verify
sig is valid

ensure top
of stack is 1

Problem: this contract cannot be enforced using Bitcoin script

What to do?

NameCoin: fork of Bitcoin that implements this contract
(see also the Handshake, Chia projects)

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 interface to DAPP
- **DeFi**: exchanges, lending, stablecoins, derivatives, etc.
- Insurance
- **DAOs**: decentralized organizations
- **NFTs**: Managing distinguished assets (ERC-721 interface)
- **Games, metaverse**: assets managed on chain

stateofthedapps.com, dapp.review

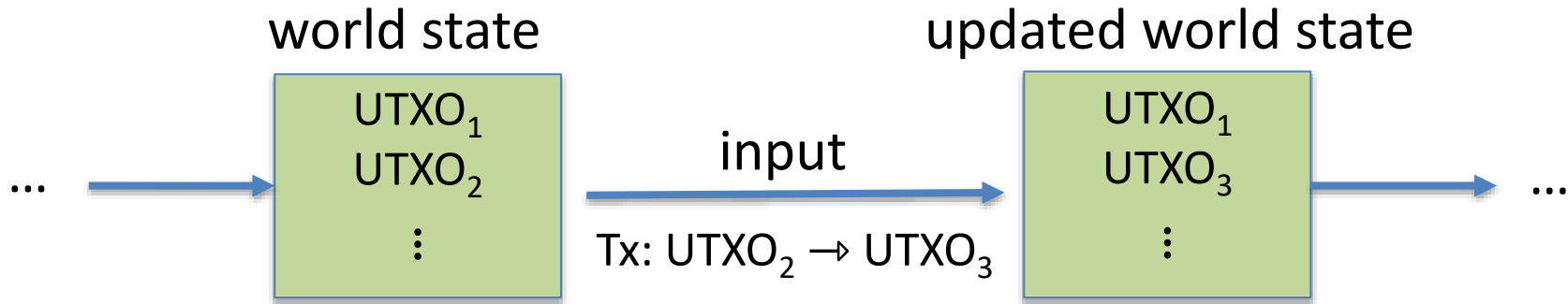


CryptoPunk #2890



◆ 30

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

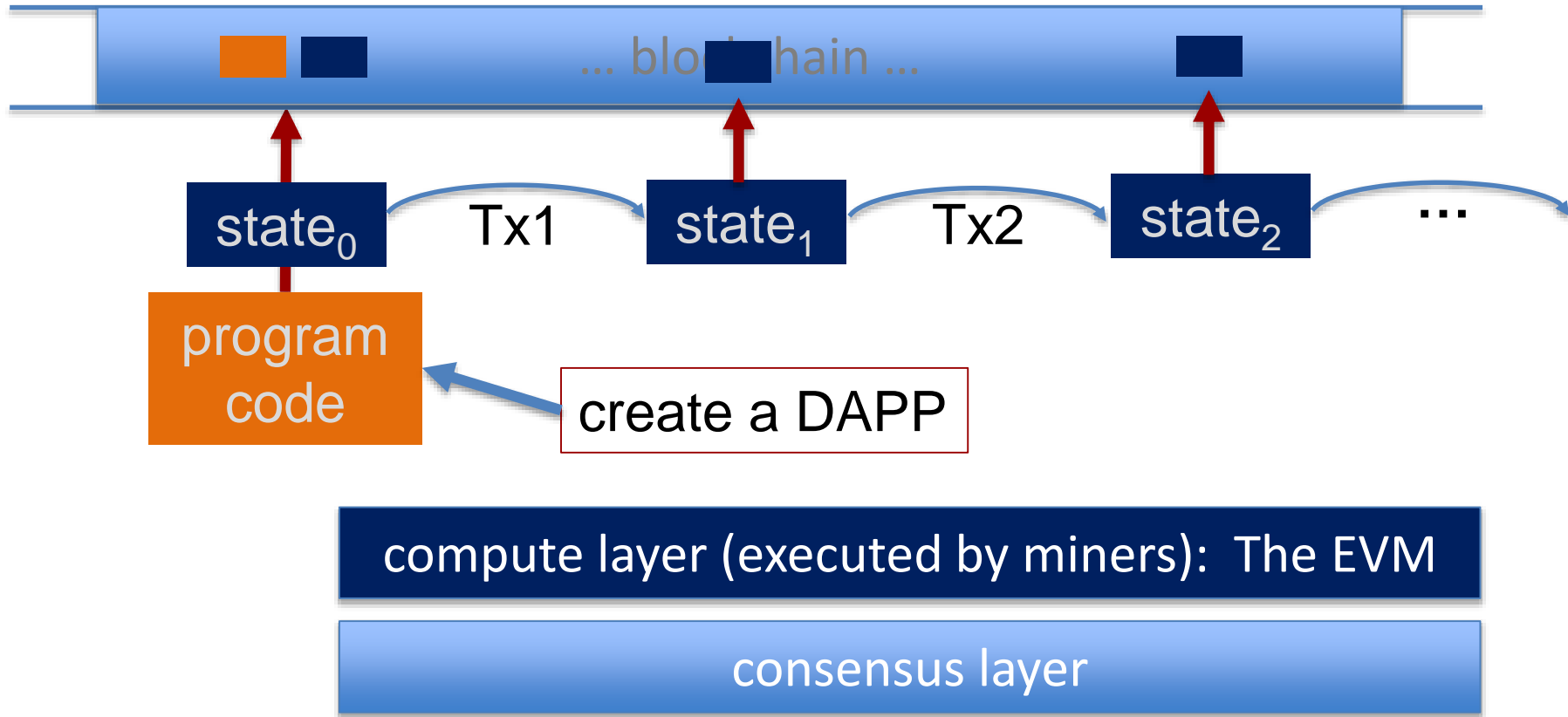
I: set of all possible inputs

Ethereum as a state transition system

Much richer state transition functions

⇒ one transition executes an entire program

Running a program on a blockchain (DAPP)



The Ethereum system

Layer 1 (ETHv1): PoW consensus. Block reward = 2 ETH + Tx fees (gas)

Latest Blocks

Bk	10991728 43 secs ago	Miner BTC.com Pool 168 txns in 18 secs
Bk	10991727 1 min ago	Miner zhizhu.top 152 txns in 30 secs
Bk	10991726 1 min ago	Miner Spark Pool 203 txns in 14 secs
Bk	10991725 1 min ago	Miner F2Pool 131 txns in 6 secs
Bk	10991724 1 min ago	Miner 0x6ebaf477f83e05558... 119 txns in 0 secs
Bk	10991723 2 mins ago	Miner Ethermine 131 txns in 48 secs

avg. block rate = 15 seconds.

ETHv1: variant of Nakamoto PoW

ETHv2: proof of stake consensus

about 150 Tx per block.

Ethereum compute layer: the EVM

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

Two types of accounts:

(1) owned accounts: 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

<u>Account data</u>	<u>Owned</u>	<u>Contracts</u>
address (computed):	H(PK)	H(CreatorAddr, CreatorNonce)
code :	⊥	CodeHash
storage root (state):	⊥	StorageRoot
balance (in Wei):	balance	balance (10 ¹⁸ Wei = 1 ETH)
nonce :	nonce	nonce

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

Account state: persistent storage

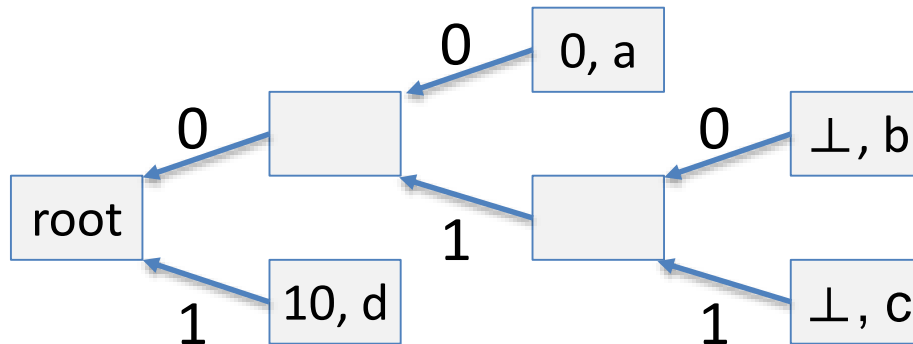
Every contract has an associated **storage array S[]**:

S[0], S[1], ... , S[2²⁵⁶-1]: each cell holds 32 bytes, init to 0.

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

- Cannot compute full Merkle tree hash: 2²⁵⁶ leaves

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



time to compute
root hash:
 $\leq 2 \times |S|$

$|S|$ = # non-zero cells

State transitions: Tx and messages

Transactions: signed data by initiator

- **To:** 32-byte address of target (0 → create new account)
- **From, [Signature]:** initiator address and signature on Tx (if owned)
- **Value:** # Wei being sent with Tx
- TX fees (EIP 1559): **gasLimit, maxFee, maxPriorityFee** (later)
- if To = 0: create new contract **code = (init, body)**
- if To ≠ 0: **data** (what function to call & arguments)
- **nonce:** must match current nonce of sender (prevents Tx replay)

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
0x4d1317a2a98cfea41...	→	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

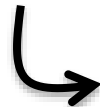
Same as Tx, but no signature (contract has no signing key)

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: - 1024 eth	<u>owned</u>
bb75a980: - 5202 eth if !contract.storage[tx.data[0]]: contract.storage[tx.data[0]] = tx.data[1] [0, 235235, 0, ALICE	<u>contract</u>
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]	<u>contract</u>
4096ad65: - 77 eth	<u>owned</u>

world state (four accounts)

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

State'	
14c5f8ba: - 1014 eth	
bb75a980: <u>- 5212 eth</u> if !contract.storage[tx.data[0]]: contract.storage[tx.data[0]] = tx.data[1] [0, 235235, <u>CHARLIE</u> , 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	

updated world state

An Ethereum Block

Miners collect Txs from users \Rightarrow leader creates a block of n Tx

- Miner does:
 - 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 miners re-execute all Tx to verify block

- Miners should only build on a valid block
- Miners are not paid for verifying block (note: verifier's dilemma)

Block header data (simplified)

(1) consensus data: parent hash, difficulty, PoW solution, etc.

(2) address of gas beneficiary: where Tx fees will go

(3) world state root: updated world state

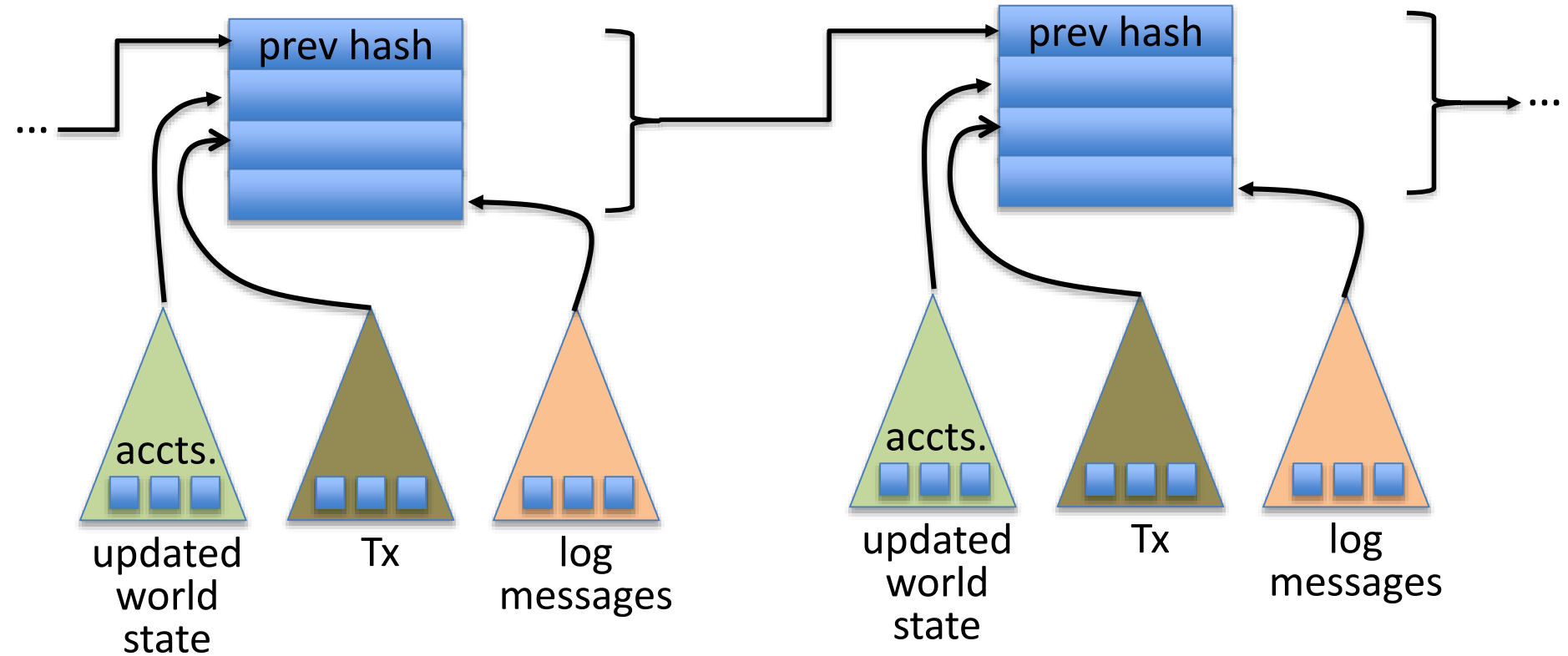
Merkle Patricia Tree hash of all accounts in the system

(4) **Tx root:** Merkle hash of all Tx processed in block

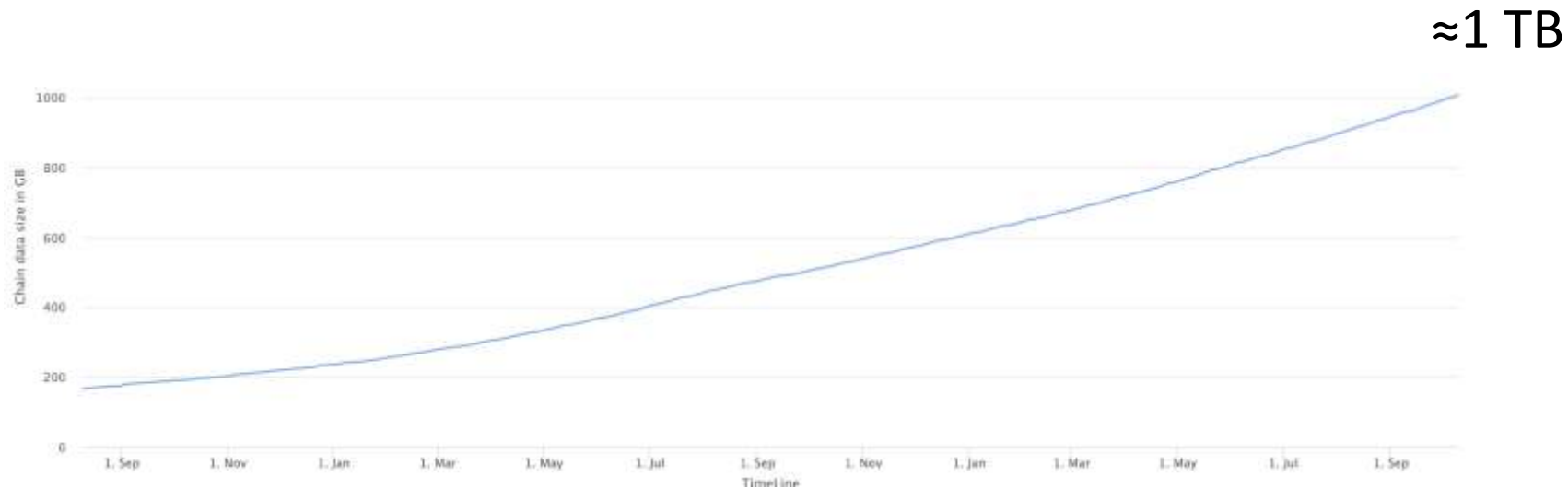
(5) **Tx receipt root:** Merkle hash of log messages generated in block

(5) Gas used: tells verifier how much work to verify block

The Ethereum blockchain: abstractly



Amount of memory to run a node (in GB)



ETH total blockchain size: 8.6 TB (Oct. 2021)

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
```

EVM mechanics: execution environment

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

⇒ compile to EVM bytecode

(some projects use WASM or BPF bytecode)

⇒ miners 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; miner 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://ethervm.io/>

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

Refund is given for reducing size of blockchain state

SELFDESTRUCT **addr**: kill current contract (in the past: 24,000 gas refund)

CREATE : 32,000 + 200 × (code size) gas **CALL** **gas**, **addr**, **value**, **args**

Gas calculation

Why charge gas?

- Tx fees (gas) prevents submitting Tx that runs for many steps.
- During high load: miners choose Tx from the mempool that maximize their income.

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

- Every Tx contains a gasPrice ``bid'' (gas \rightarrow Wei conversion price)
- Miners choose Tx with highest gasPrice (max $\sum(\text{gasPrice} \times \text{gasLimit})$)
 \Rightarrow not an efficient auction mechanism (first price auction)

Gas prices spike during congestion

GasPrice in Gwei:

86 Gwei = 86×10^{-9} ETH



Average Tx fee in USD



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 miner

Computed **gasPrice** bid:

$$\mathbf{gasPrice} \leftarrow \min(\mathbf{maxFee}, \mathbf{baseFee} + \mathbf{maxPriorityFee})$$

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

Gas calculation

- (1) if **gasPrice** < **baseFee**: abort
- (2) If **gasLimit** × **gasPrice** < msg.sender.balance: abort
- (3) deduct **gasLimit** × **gasPrice** from msg.sender.balance

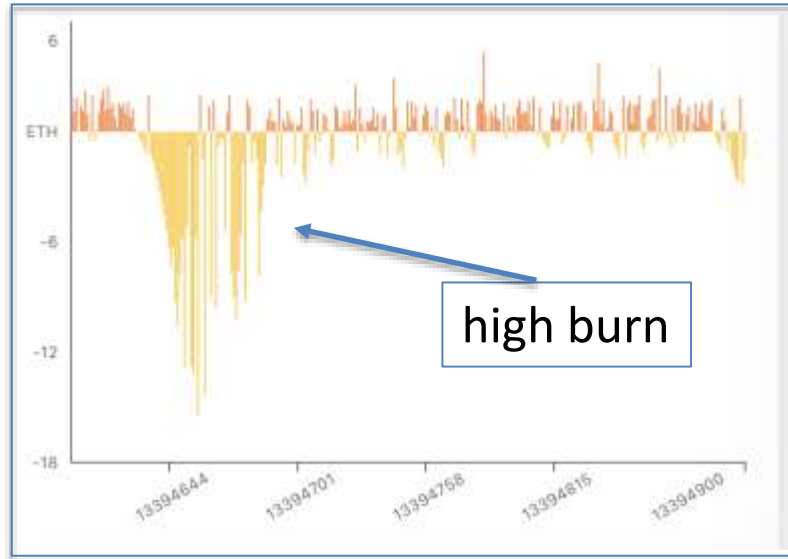
- (4) set **Gas** ← **gasLimit**
- (5) execute Tx: deduct gas from **Gas** for each instruction
if at end (**Gas** < 0): abort, Tx is invalid (miner keeps **gasLimit** × **gasPrice**)
- (6) Refund **Gas** × **gasPrice** to msg.sender.balance

- (7) **gasUsed** ← **gasLimit** – **Gas**
 - (7a) BURN **gasUsed** × **baseFee**
 - (7b) Send **gasUsed** × (**gasPrice** – **baseFee**) to miner

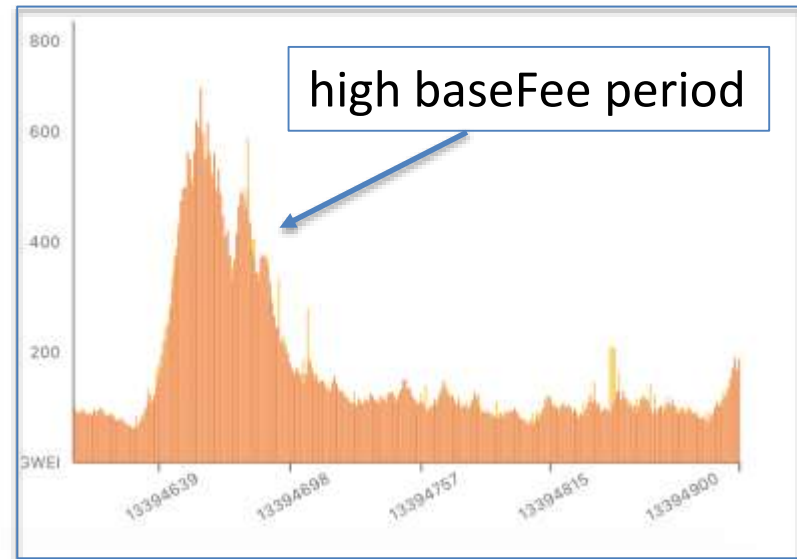


Burn results in practice

block reward (2ETH) –
Total baseFee burned in block



baseFee for block (Wei)



... sometimes burn exceeds block rewards \Rightarrow ETH deflation

Why burn ETH ???

EIP1559 goals (informal):

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

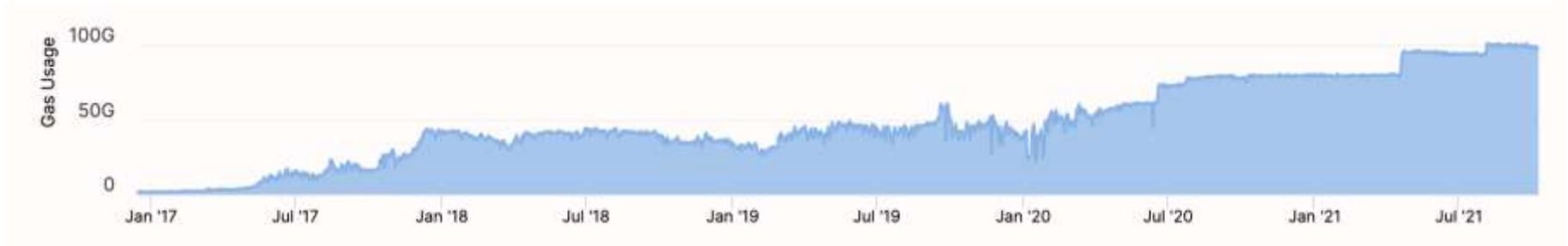
Suppose no burn (i.e., baseFee given to miners):

⇒ in periods of low Tx volume miners 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 \Rightarrow each Tx takes more instructions to execute

END OF LECTURE

Next lecture: writing Solidity contracts