BIU Winter School on cryptography

Day 1 Lecture 3

Ethereum Mechanics

Dan Boneh

Stanford University

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)

⋮

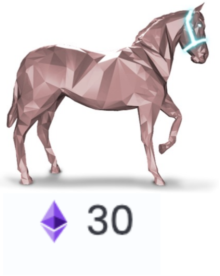
STEPH CURRY
Assist · Nov 28 2021
Metallic Gold LE (Series 3)
Rare /749

SOLD BY
55 collectors

55
FOR SALE

$330.00 USD
Lowest Ask

ENS

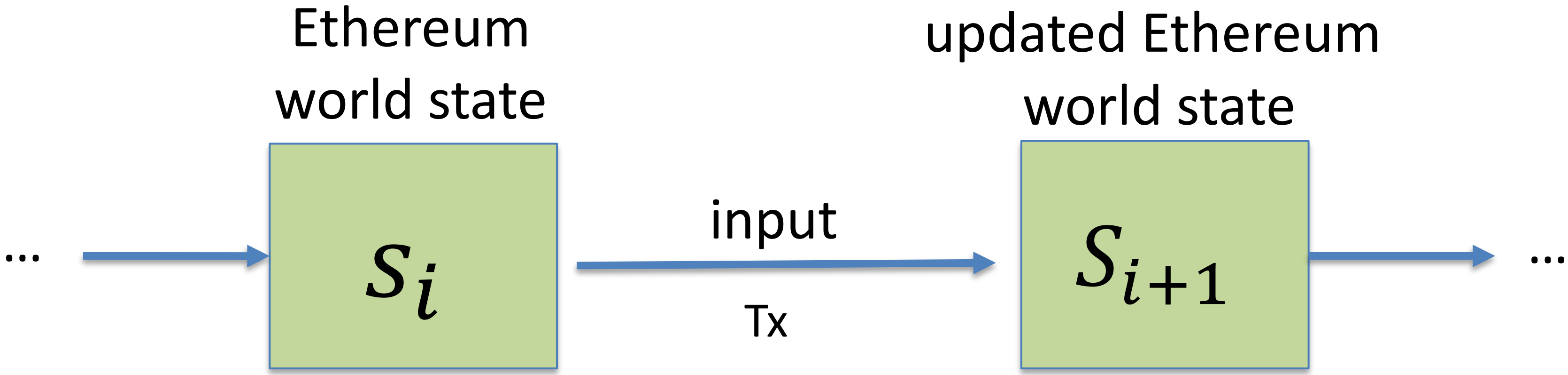


stateofthedapps.com, dapp.review

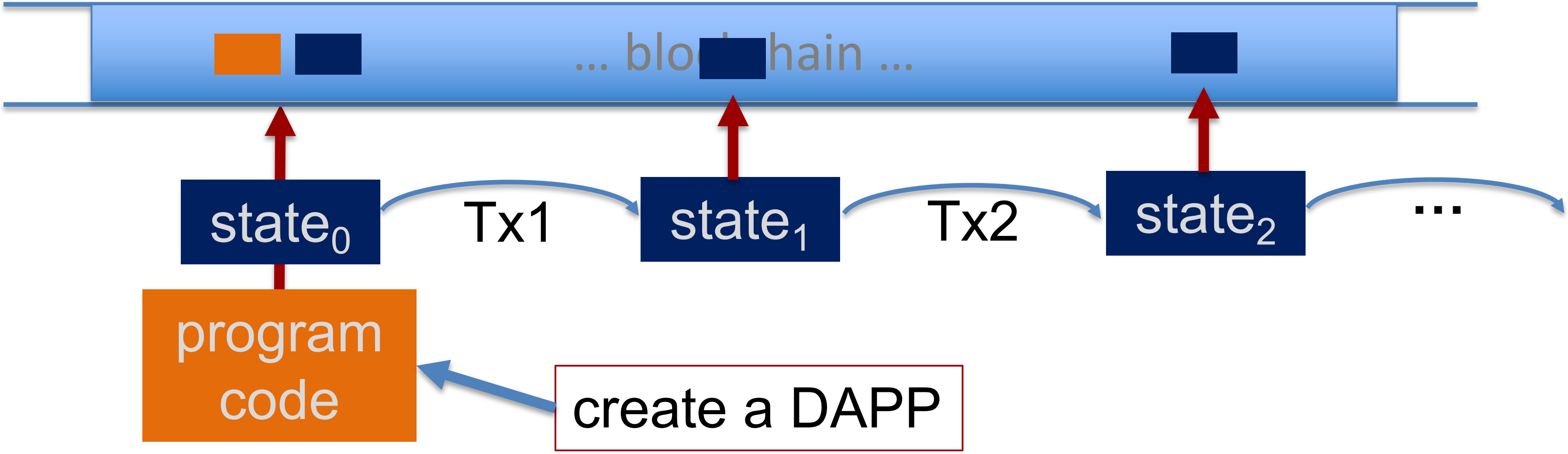
Ethereum as a state transition system

A rich state transition function

⇒ one transition executes an entire program



Running a program on a blockchain (DAPP)

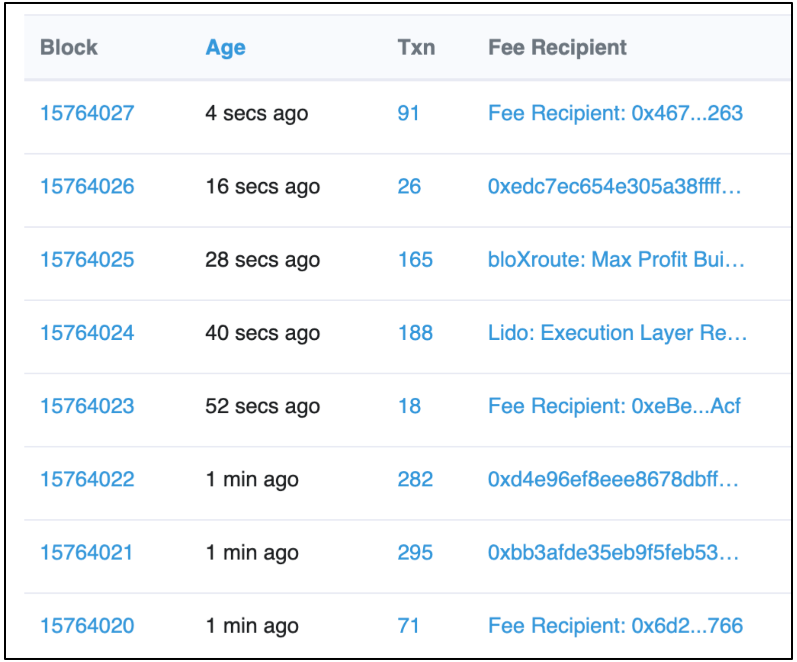


compute layer (execution chain): The EVM

consensus layer (beacon chain)

The Ethereum system

Proof-of-Stake consensus



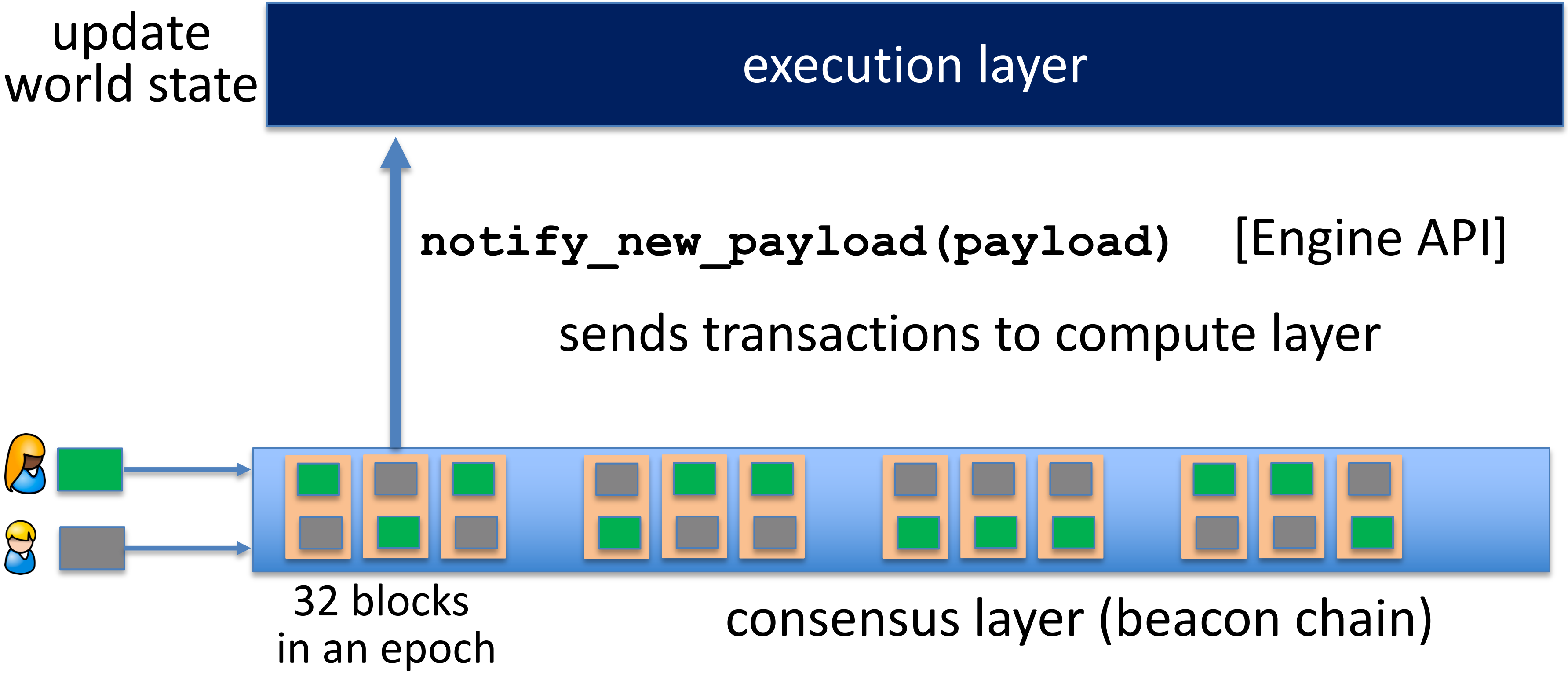
Block	Age	Txn	Fee Recipient
15764027	4 secs ago	91	Fee Recipient: 0x467...263
15764026	16 secs ago	26	0xedc7ec654e305a38ffff…
15764025	28 secs ago	165	bloXroute: Max Profit Bui…
15764024	40 secs ago	188	Lido: Execution Layer Re…
15764023	52 secs ago	18	Fee Recipient: 0xeBe...Acf
15764022	1 min ago	282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

The Ethereum system (post merge)



update world state

execution layer

`notify_new_payload(payload)` [Engine API]

sends transactions to compute layer

32 blocks in an epoch

consensus layer (beacon chain)

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) owned accounts (EOA): controlled by a signing key pair (pk,sk).

sk: owned by account owner

(2) **contracts**: controlled by code (set by creator)

Data associated with an account

Account data	Owned	Contracts
address (computed):	H(pk)	(different with CREATE2) H(CreatorAddr, CreatorNonce)
balance (in Wei):	balance	balance (10^{18} Wei = 1 ETH)
code:	$\perp$	CodeHash
storage root (state):	$\perp$	StorageRoot
nonce:	nonce	nonce

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

Account state: persistent storage

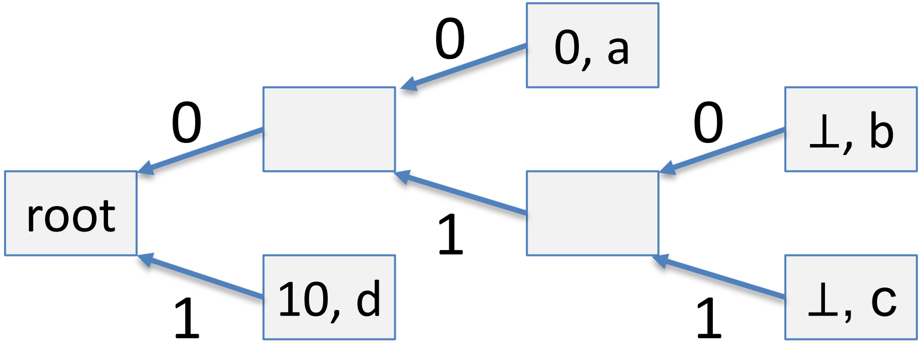
Every contract has an associated **storage array S**[]:

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

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

- 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:
≤ 2 × |S|

|S| = # non-zero cells

State transitions: Tx and messages

Transaction types:

- **owned** → owned: transfer ETH between users
- **owned** → contract: call contract with ETH & data

After a contract is called:

- **contract** → contract: one program calls another (composability)
- **contract** → owned: contract sends funds to user

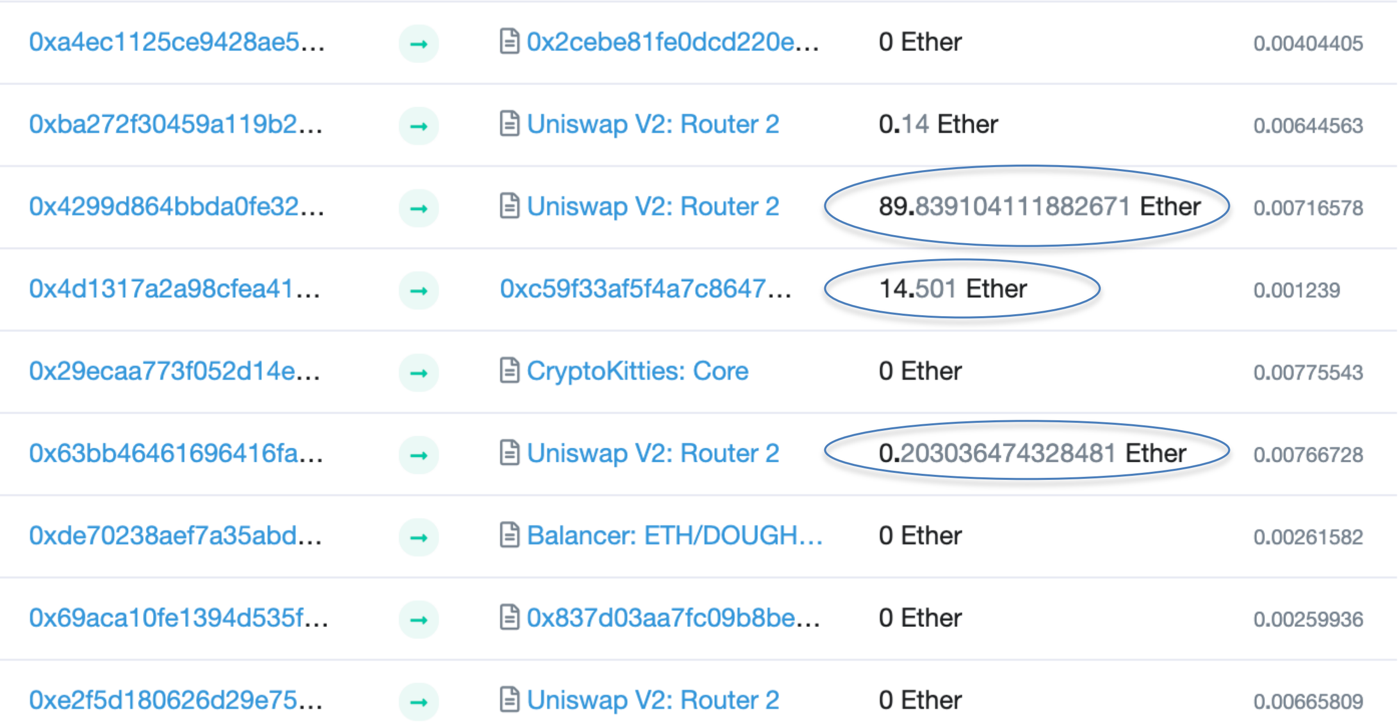
Calling a contract can start a chain of transactions: A → B → C → D

State transitions: Tx and messages

Transactions: signed data by initiator

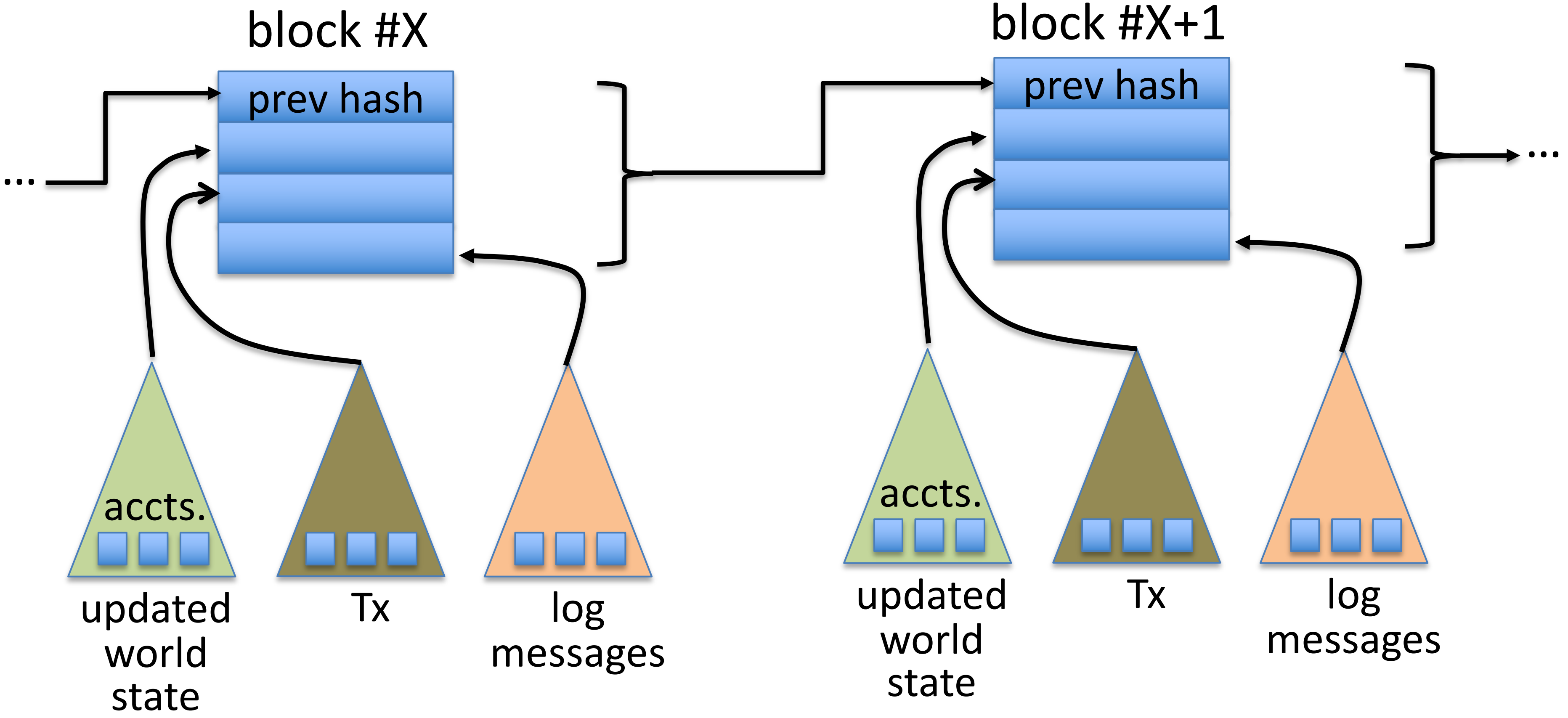
- **To:** 32-byte address of target (0 → create new account)
- **From**, **[Signature]**: initiator address and signature on Tx (if owned)
- **Value**: # Wei being sent with Tx
- Tx fees (EIP 1559)**: gasLimit, maxFee, maxPriorityFee** (later)
- if To ≠ 0: **data** (what function to call & arguments)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

Example (block #10993504)

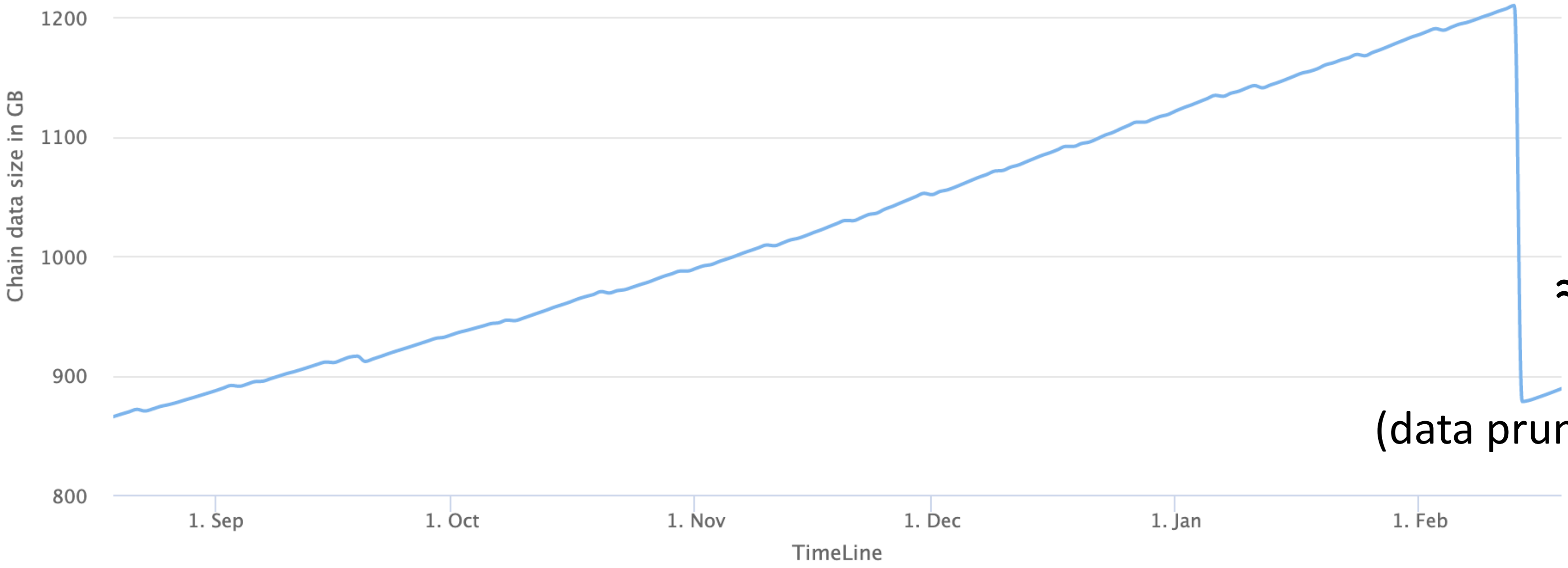


From		To	msg.value	Tx fee (ETH)
0xa4ec1125ce9428ae5…	→	0x2cebe81fe0dcd220e…	0 Ether	0.00404405
0xba272f30459a119b2…	→	Uniswap V2: Router 2	0.14 Ether	0.00644563
0x4299d864bbda0fe32…	→	Uniswap V2: Router 2	89.839104111882671 Ether	0.00716578
0x4d1317a2a98cfea41…	→	0xc59f33af5f4a7c8647…	14.501 Ether	0.001239
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

The Ethereum blockchain: abstractly



Amount of memory to run a node



≈900 GB

(data pruned)

ETH total blockchain size (archival): 13 TB (Feb. 2023)

An example contract: NameSystem

A name system on Ethereum: [uniswap → addr]

(a simplified ENS)

Need to support three operations:

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

An example contract: NameSystem

```
contract nameSys {          // Solidity code

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

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

data
in contract
storage

An example contract: NameSystem

```
function nameNew(bytes32 name) {

    // registration fee is 100 Wei

    if (data[name] == 0   &&   msg.value >= 100) {
        data[name].owner = msg.sender       // record 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 commit-reveal.

An example contract: NameSystem

```
function nameUpdate(
                bytes32 name, bytes32 newValue, address newOwner) {

   // check if message is from owner, and fee 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: NameSystem

```
    function nameLookup(bytes32 name) {
        return data[name];
    }

}  // end of contract
```

EVM contracts cannot keep secrets
(we need practical iO)

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 (https://www.evm.codes)

Stack machine

- code can CREATE or CALL another contract

on chain storage is expensive

In addition: several types of memory

- Persistent storage (on blockchain): SLOAD, SSTORE (expensive)
- Volatile memory (for single Tx): MLOAD, MSTORE (cheap)
- LOG0(data): write data to log
- CallData: arguments in Tx (persistent, but only readable by current Tx)

Every instruction costs gas, examples:

MLOAD, **MSTORE**: 3 gas (cheap)

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)

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

CALL **gas**, addr, **value**, args

Gas calculation

Why charge gas?

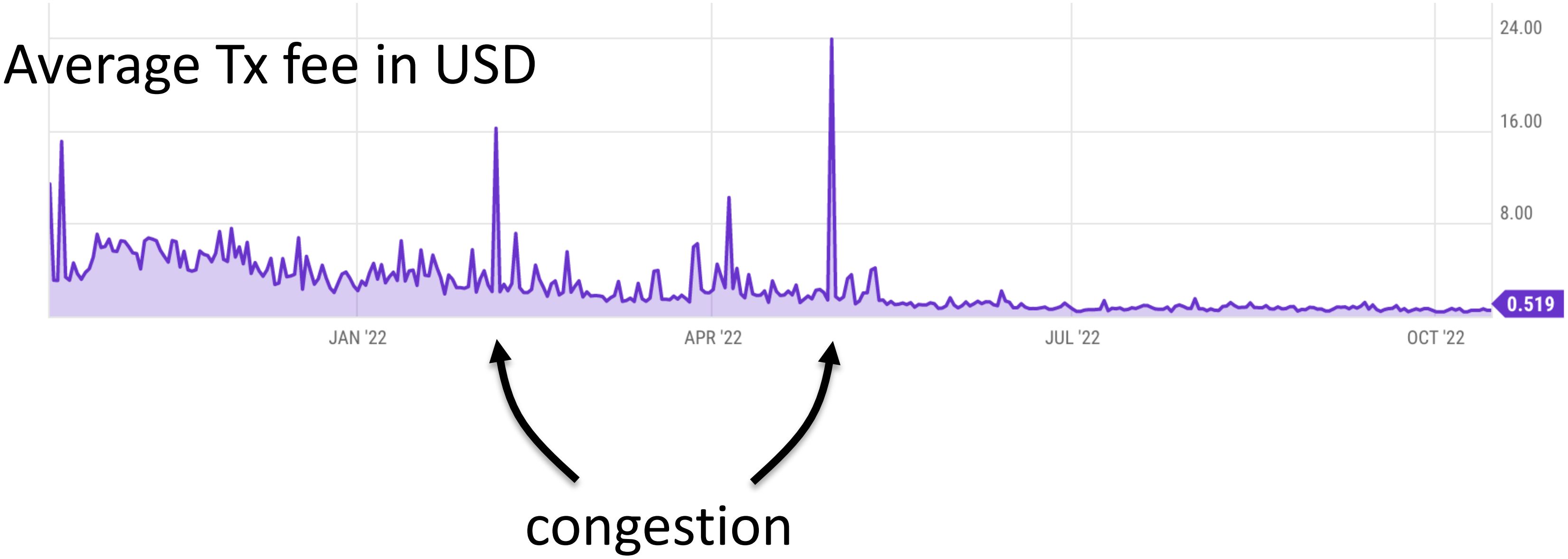
- Tx fees (gas) prevents submitting Tx that runs for many steps.
- During high load: block proposer chooses set of Tx from mempool that maximize its income.

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 sum(gasPrice × gasUsed))

$\Longrightarrow$ not an efficient auction mechanism (first price auction)

Gas prices spike during congestion



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 (since 8/2021)

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) ⟹ base fee goes up 12.5%
- earlier blocks empty ⟹ base fee decreases by 12.5%

interpolate in between

If earlier blocks at “target size” (15M gas) ⟹ 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

gasUsed ← gas used by Tx

Send **gasUsed** × (**gasPrice – baseFee**) to block proposer

BURN **gasUsed** × **baseFee**

⇒ total supply of ETH can decrease

END OF LECTURE