Ethereum

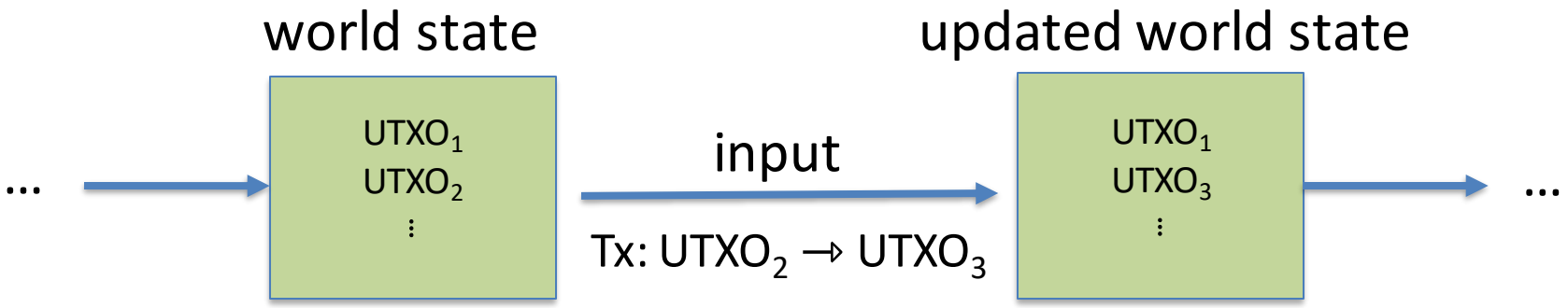
Slides from Dan Boneh

Ethereum: on-chain Turing machine

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)

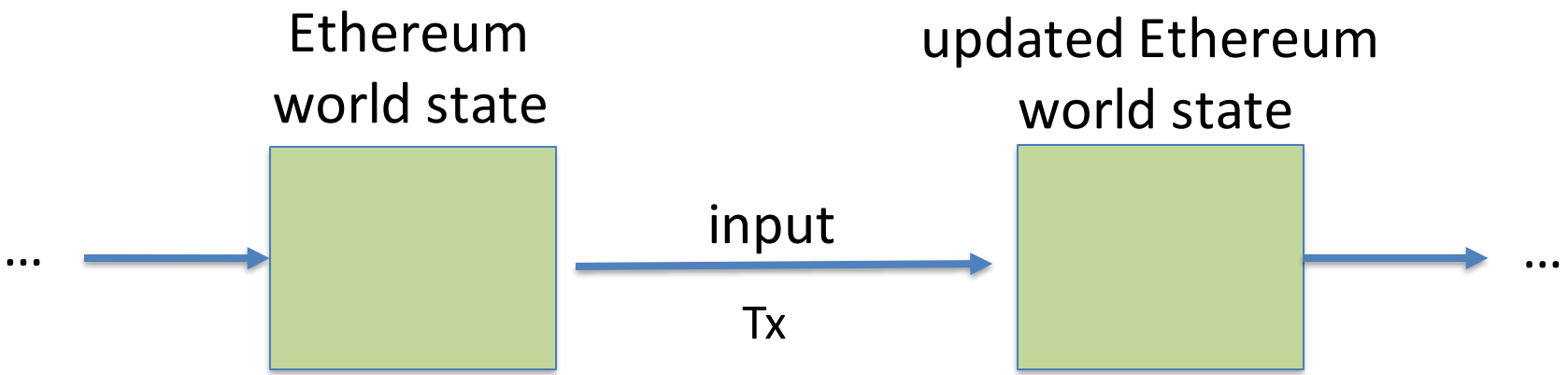
Bitcoin as a state transition system



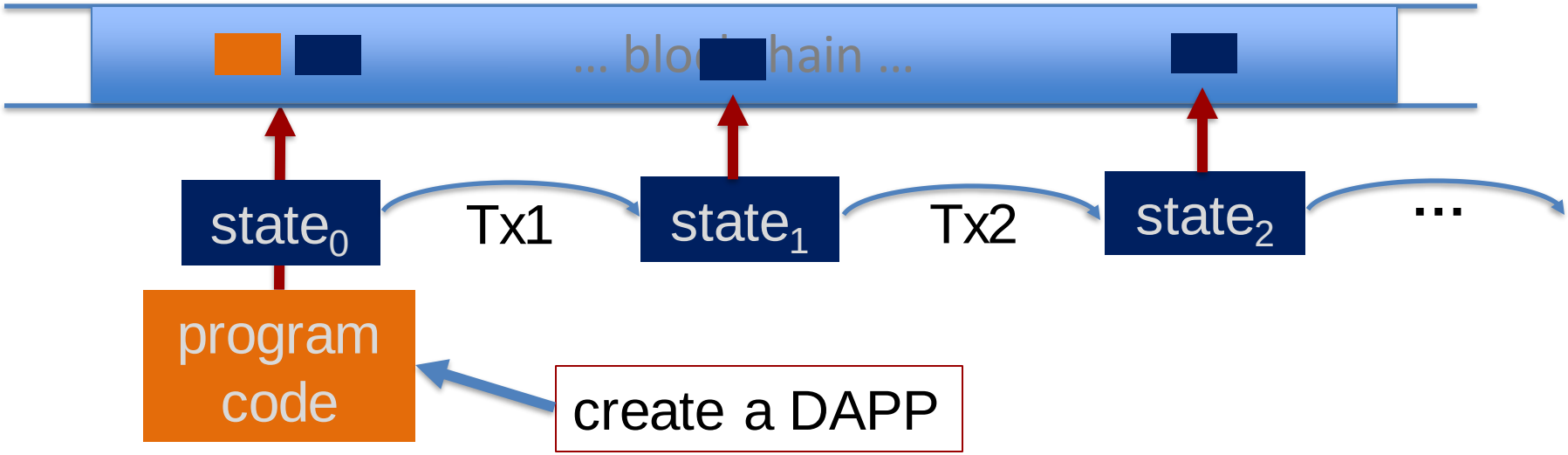
Ethereum as a state transition system

Much richer state transition functions

⇒ one transition executes an entire program



Running a program on a blockchain (DAPP)



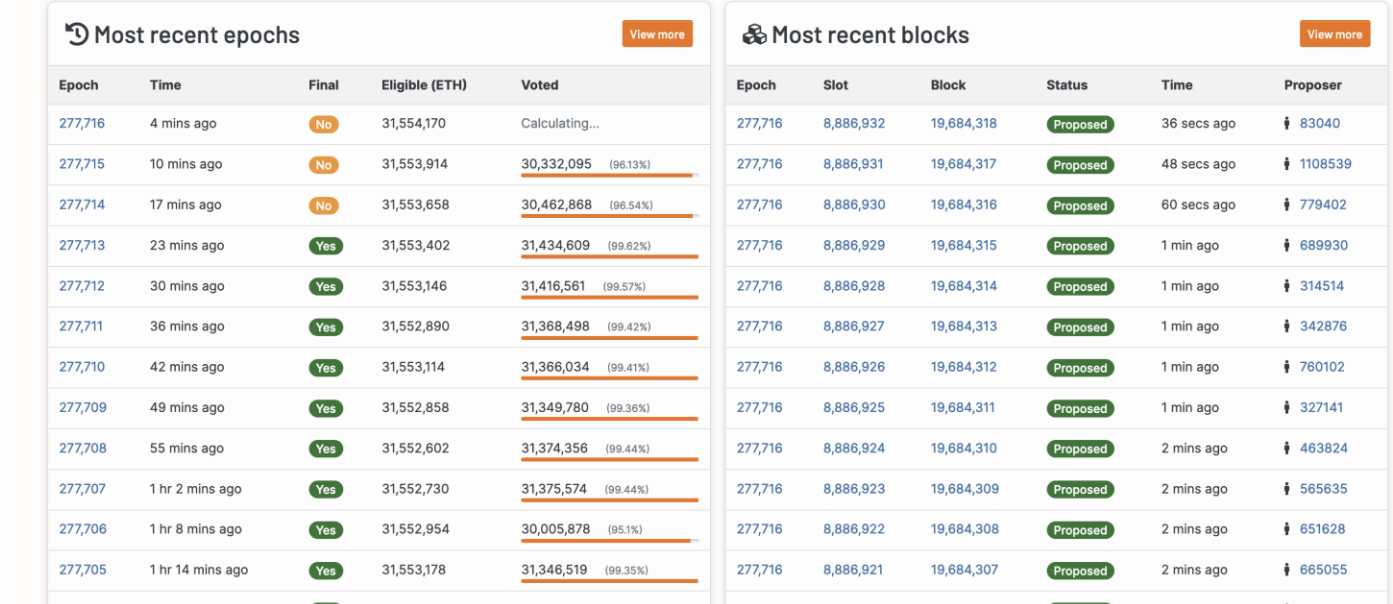
compute layer (execution chain): The EVM

consensus layer (Beacon chain)

The Ethereum system

One block every 12 seconds (~150 Tx per block

Block proposer receives Tx fees for block (+tips)



Most recent epochs

View more

Epoch	Time	Final	Eligible (ETH)	Voted
277,716	4 mins ago	No	31,554,170	Calculating...
277,715	10 mins ago	No	31,553,914	30,332,095 (96.13%)
277,714	17 mins ago	No	31,553,658	30,462,868 (96.54%)
277,713	23 mins ago	Yes	31,553,402	31,434,609 (99.62%)
277,712	30 mins ago	Yes	31,553,146	31,416,561 (99.57%)
277,711	36 mins ago	Yes	31,552,890	31,368,498 (99.42%)
277,710	42 mins ago	Yes	31,553,114	31,366,034 (99.41%)
277,709	49 mins ago	Yes	31,552,858	31,349,780 (99.36%)
277,708	55 mins ago	Yes	31,552,602	31,374,356 (99.44%)
277,707	1 hr 2 mins ago	Yes	31,552,730	31,375,574 (99.44%)
277,706	1 hr 8 mins ago	Yes	31,552,954	30,005,878 (95.1%)
277,705	1 hr 14 mins ago	Yes	31,553,178	31,346,519 (99.35%)

Most recent blocks

View more

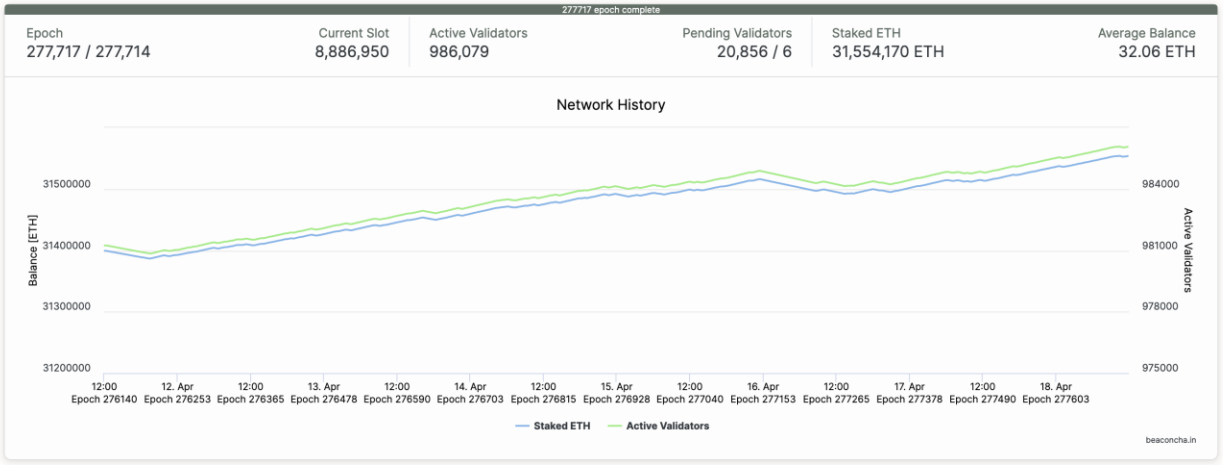
Epoch	Slot	Block	Status	Time	Proposer
277,716	8,886,932	19,684,318	Proposed	36 secs ago	83040
277,716	8,886,931	19,684,317	Proposed	48 secs ago	1108539
277,716	8,886,930	19,684,316	Proposed	60 secs ago	779402
277,716	8,886,929	19,684,315	Proposed	1 min ago	689930
277,716	8,886,928	19,684,314	Proposed	1 min ago	314514
277,716	8,886,927	19,684,313	Proposed	1 min ago	342876
277,716	8,886,926	19,684,312	Proposed	1 min ago	760102
277,716	8,886,925	19,684,311	Proposed	1 min ago	327141
277,716	8,886,924	19,684,310	Proposed	2 mins ago	463824
277,716	8,886,923	19,684,309	Proposed	2 mins ago	565635
277,716	8,886,922	19,684,308	Proposed	2 mins ago	651628
277,716	8,886,921	19,684,307	Proposed	2 mins ago	665055

A bit about the Beacon chain (Eth2 consensus layer)

To become a validator: stake (lock up) 32 ETH … soon more..

Validators: - sign blocks to express correctness (finalized once enough sigs)

- occasionally act as ***block proposer*** (chosen at random)
- correct behavior ⇒ issued **new ETH** every epoch (32 blocks)



The economics of staking

Validator locks up 32 ETH.

Annual validator income (an example):

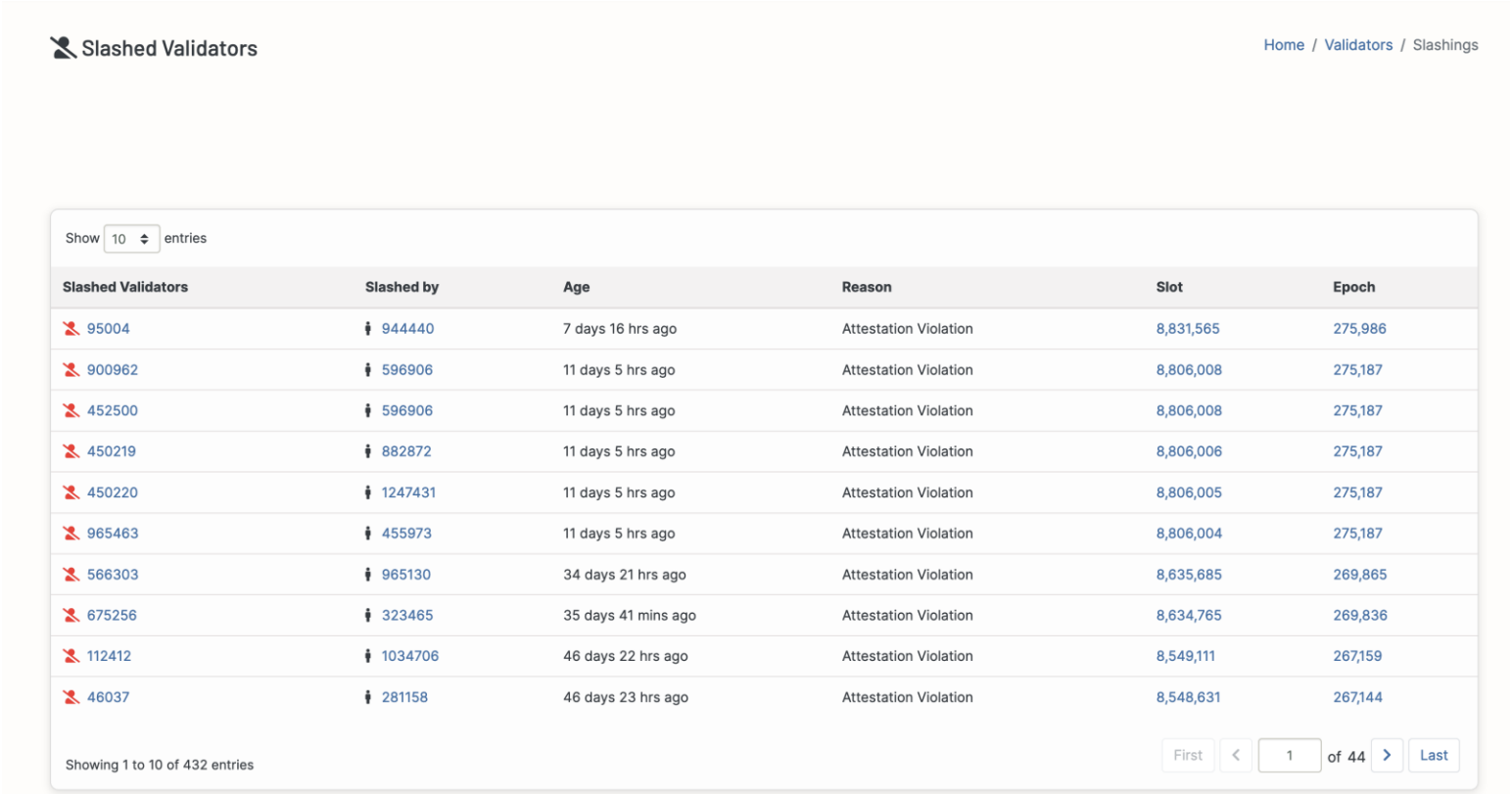
- Issuance: 1.0 ETH
- Tx fees: 0.4 ETH
- MEV: 0.4 ETH
- Total: 1.8 ETH (5.6% return on 32 ETH staked)

Can be adjusted (BASE_REWARD_FACTOR)

A function of congestion

In practice: staking provider (e.g., Ankr or LIOD) takes a cut

What about malicious behavior?



Slashed Validators

Home / Validators / Slashings

Show 10 entries

Slashed Validators	Slashed by	Age	Reason	Slot	Epoch
95004	944440	7 days 16 hrs ago	Attestation Violation	8,831,565	275,986
900962	596906	11 days 5 hrs ago	Attestation Violation	8,806,008	275,187
452500	596906	11 days 5 hrs ago	Attestation Violation	8,806,008	275,187
450219	882872	11 days 5 hrs ago	Attestation Violation	8,806,006	275,187
450220	1247431	11 days 5 hrs ago	Attestation Violation	8,806,005	275,187
965463	455973	11 days 5 hrs ago	Attestation Violation	8,806,004	275,187
566303	965130	34 days 21 hrs ago	Attestation Violation	8,635,685	269,865
675256	323465	35 days 41 mins ago	Attestation Violation	8,634,765	269,836
112412	1034706	46 days 22 hrs ago	Attestation Violation	8,549,111	267,159
46037	281158	46 days 23 hrs ago	Attestation Violation	8,548,631	267,144

Showing 1 to 10 of 432 entries

First 1 of 44 Last

How does slashing work?

- Slashed for breaking protocol rules
  - Double sign
  - Surround vote
- Penalty:
  - Exited from the beacon chain + lose % of staked ETH
  - When many validators are slashed: you lose more
- Incentive for slashing:
  - Receive rewards for reporting evidence of slashable offences.

EVM

Recap: Blocks

Validators collect Tx from users:

⇒ run Tx sequentially on current world state

⇒ new block contains **updated world state**, Tx list, log msgs

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(CreatorAddr, 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

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 (1 Wei = 10^{-18} ETH)
- 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)
- chain_id: ensures Tx can only be submitted to the intended chain

State transitions: Tx and messages

Transaction types:

EOA→ EOA: transfer ETH between users

EOA→ contract: call contract with ETH & data

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

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

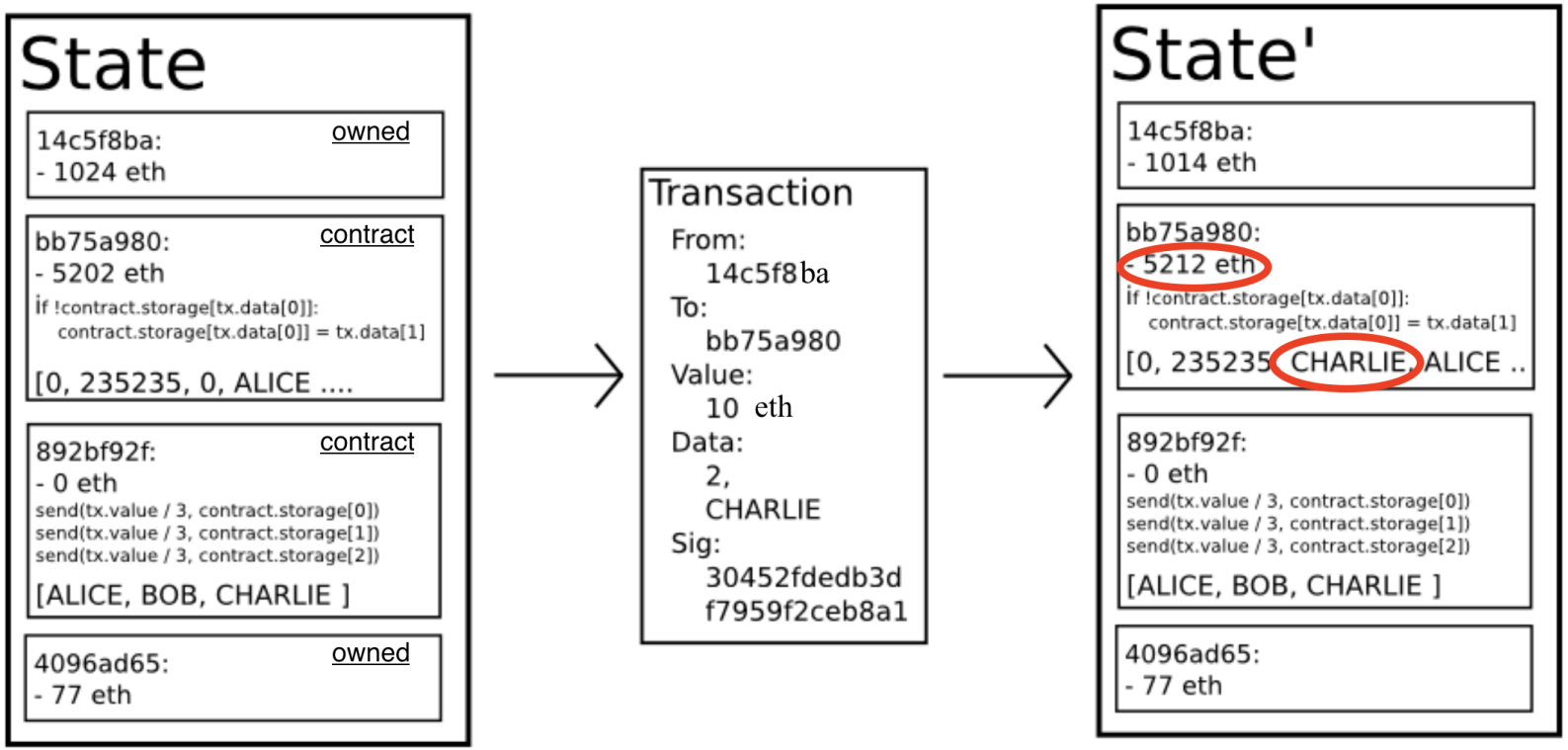
contract → contract: one program calls another (and sends funds)

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

Tx from EOA → contract → another contract

another contract → different EOA

Example Tx



world state (four accounts)

updated world state

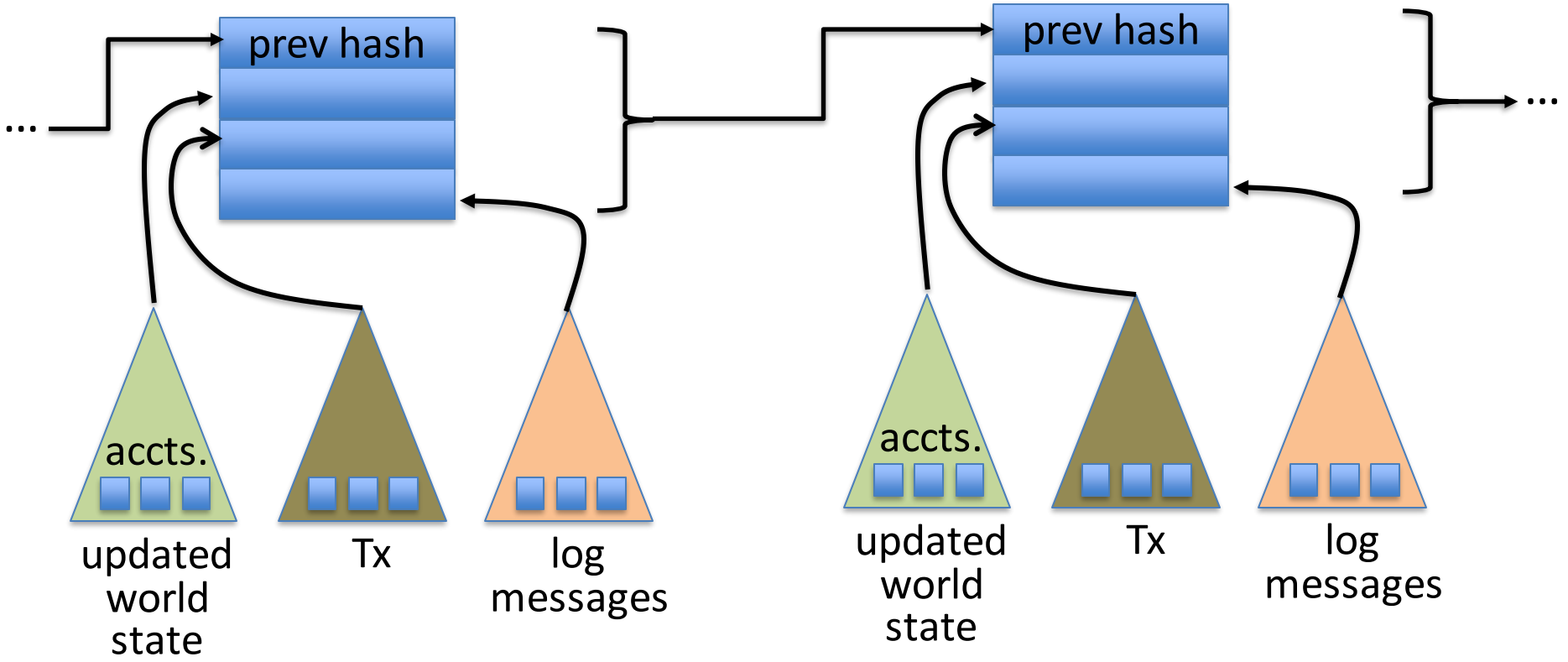
An Ethereum Block

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

- To produce a block do:
  - for i=1,…,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 ⇒
sign block if valid ⇒ enough sigs, epoch is finalized.

The Ethereum blockchain: abstractly



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

see https://www.evm.codes

Stack machine (like Bitcoin) but with JUMP

- contract can create or call another contract ⇒ composability

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 tree (not readable by EVM)
- Tx Calldata (16 gas/byte): readable by EVM in current Tx

(near future: support for cheap 128KB blobs)

Every instruction costs gas

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.

if **gasUsed ≥ gasLimit**: block proposer keeps gas fees (from Tx originator)

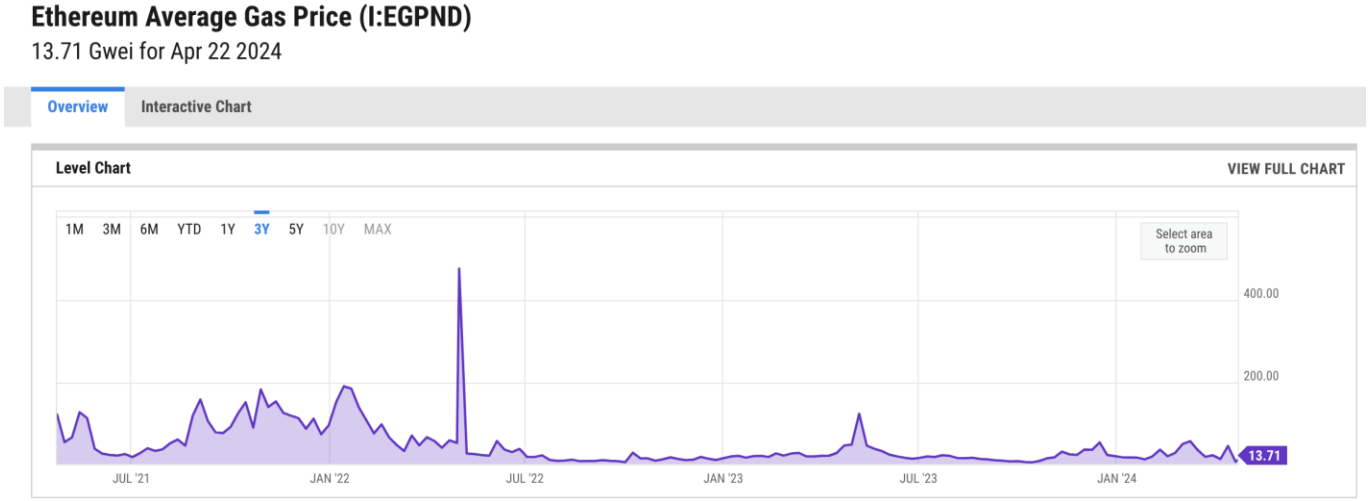
calculated by EVM

specified in Tx

Every instruction costs gas

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.



Gas calculation: EIP1559

Every block has a "baseFee": the **minimum** gasPrice for Tx in the block

baseFee is computed from total gas in earlier blocks:

- earlier blocks at gas limit (30M gas) $\Longrightarrow$ base fee goes up 12.5%
- earlier blocks empty $\Longrightarrow$ base fee decreases by 12.5%

} interpolate in between

If earlier blocks at "target size" (15M gas) $\Longrightarrow$ baseFee does not change

Gas calculation

A transaction specifies three parameters:

- **gasLimit**: max total gas allowed for Tx

bid

- **maxFee:** maximum allowed gas price
- **maxPriorityFee**: additional "tip" to be paid to block proposer

Computed **gasPrice** bid (in Wei = 10^{-18} ETH):

gasPrice ⇦ min(**maxFee**, **baseFee** + **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 (proposer keeps **gasLimit** × **gasPrice**)
(6) Refund **Gas** × **gasPrice** to msg.sender.balance (leftover change)

---

(7) **gasUsed** ⇦ **gasLimit** – **Gas**
(7a) BURN **gasUsed** × **baseFee**

(7b) Send **gasUsed** × (**gasPrice** – **baseFee**) to block producer

Solidity

Contract structure

```
interface IERC20 {
    function transfer(address _to,  uint256 _value)  external  returns (bool);
    function totalSupply()  external  view  returns (uint256);
    …
}

contract ERC20 is IERC20  {          // inheritance
    address owner;
    constructor() public { owner = msg.sender; }
    function transfer(address _to, uint256 _value)  external returns (bool)  {
       …  implementation …
}     }
```

Value types

- uint256
- address (bytes32)
  - _address.**balance**, _address.**send**(value), _address.**transfer**(value)
  - call: send Tx to another contract

    bool success = _address.**call**{value: msg.value/2, gas: 1000}(args);
  - delegatecall: load code from another contract into current context
- bytes32
- bool

> **Warning**
>
> There are some dangers in using `send`: The transfer fails if the call stack depth is at 1024 (this can always be forced by the caller) and it also fails if the recipient runs out of gas. So in order to make safe Ether transfers, always check the return value of `send`, use `transfer` or even better: use a pattern where the recipient withdraws the Ether.

Reference types

- structs
- arrays
- bytes
- strings
- mappings:
  - Declaration: mapping (address => unit256) **balances**;
  - Assignment: balances[addr] = value;

```
struct Person {
      uint128 age;
      uint128 balance;
      address addr;
  }
Person[10] public people;
```

Globally available variables

- **block**: .blockhash, .gaslimit, .number, .timestamp, .coinbase
- gasLeft()
- **msg**: .data, .sender, .sig, .value
- **tx**: .gasprice, .origin
- abi: encode, encodePacked, encodeWithSelector, encodeWithSignature
- Keccak256(), sha256(), sha3()
- **require**, **assert** e.g.: require(msg.value > 100, “insufficient funds sent”)

A → B → C → D:
at D: msg.sender == C
tx.origin == A

Function visibilities

- **external**: function can only be called from outside contract.

  Arguments read from calldata
- **public**: function can be called externally and internally.

  if called externally: arguments copied from calldata to memory
- **private**: only visible inside contract
- **internal**: only visible in this contract and contracts deriving from it
- **view**: only read storage (no writes to storage)
- **pure**: does not touch storage

```
function f(uint a) private pure returns (uint b) { return a + 1; }
```

ERC20 tokens

- A standard API for fungible tokens that provides basic functionality to transfer tokens or allow the tokens to be spent by a third party.
- An ERC20 token is itself a smart contract that maintains all user balances
- A standard interface allows other contracts to interact with every ERC20 token. No need for special logic for each token.

ERC20 tokens

Let's look at:

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol

And...

https://etherscan.io/address/0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48#code

Calling other contracts

- Addresses can be cast to contract types.

```
address  _token;
IERC20Token  tokenContract = IERC20Token(_token);
ERC20Token  tokenContract = ERC20Token(_token);
```

- When calling a function on an external contract, Solidity will automatically handle ABI encoding, copying to memory, and copying return values.
  - **tokenContract**.transfer(_to, _value);

Stack variables

- Stack variables generally cost the least gas
  - can be used for any simple types (anything that is <= 32 bytes).
    - uint256 a = 123;
- All simple types are represented as bytes32 at the EVM level.
- Only 16 stack variables can exist within a single scope.

Calldata

- Calldata is a read-only byte array.
- Every byte of a transaction's calldata costs gas

  (16 gas per non-zero byte, 4 gas per zero byte).
- It is cheaper to load variables directly from calldata, rather than copying them to memory.
  - This can be accomplished by marking a function as `external`.

Memory (compiled to MSTORE, MLOAD)

- Memory is a byte array.
- Complex types (anything > 32 bytes such as structs, arrays, and strings) must be stored in memory or in storage.

  string memory **name** = “Alice”;

- Memory is cheap, but the cost of memory grows quadratically.

Storage array (compiled to SSTORE, SLOAD)

- Using storage is very expensive and should be used sparingly.
- Writing to storage is most expensive.

  Reading from storage is cheaper, but still relatively expensive.
- mappings and state variables are always in storage.
- Some gas is refunded when storage is deleted or set to 0
- Trick for saving has: variables < 32 bytes can be packed into 32 byte slots.

Event logs

- Event logs are a cheap way of storing data that does not need to be accessed by any contracts.

- Events are stored in transaction receipts, rather than in storage.

Security considerations

- Are we checking math calculations for overflows and underflows?(developers sometimes explicitly opt out)
- What assertions should be made about function inputs, return values, and contract state?
- Who is allowed to call each function?
- Are we making any assumptions about the functionality of external contracts that are being called?

What's the problem here?

```
1  contract Vulnerable {
2      ...
3
4      function withdraw() external {
5          uint256 amount = balanceOf[msg.sender];
6          (bool success, ) = msg.sender.call.value(amount)("");
7          require(success, "Transfer failed.");
8          balanceOf[msg.sender] = 0;
9      }
10 }
```

https://consensys.io/diligence/blog/2019/09/stop-using-soliditys-transfer-now/