Smart Contracts on the Ethereum Blockchain (2)

Instructor: Paruj Ratanaworabhan

Sources

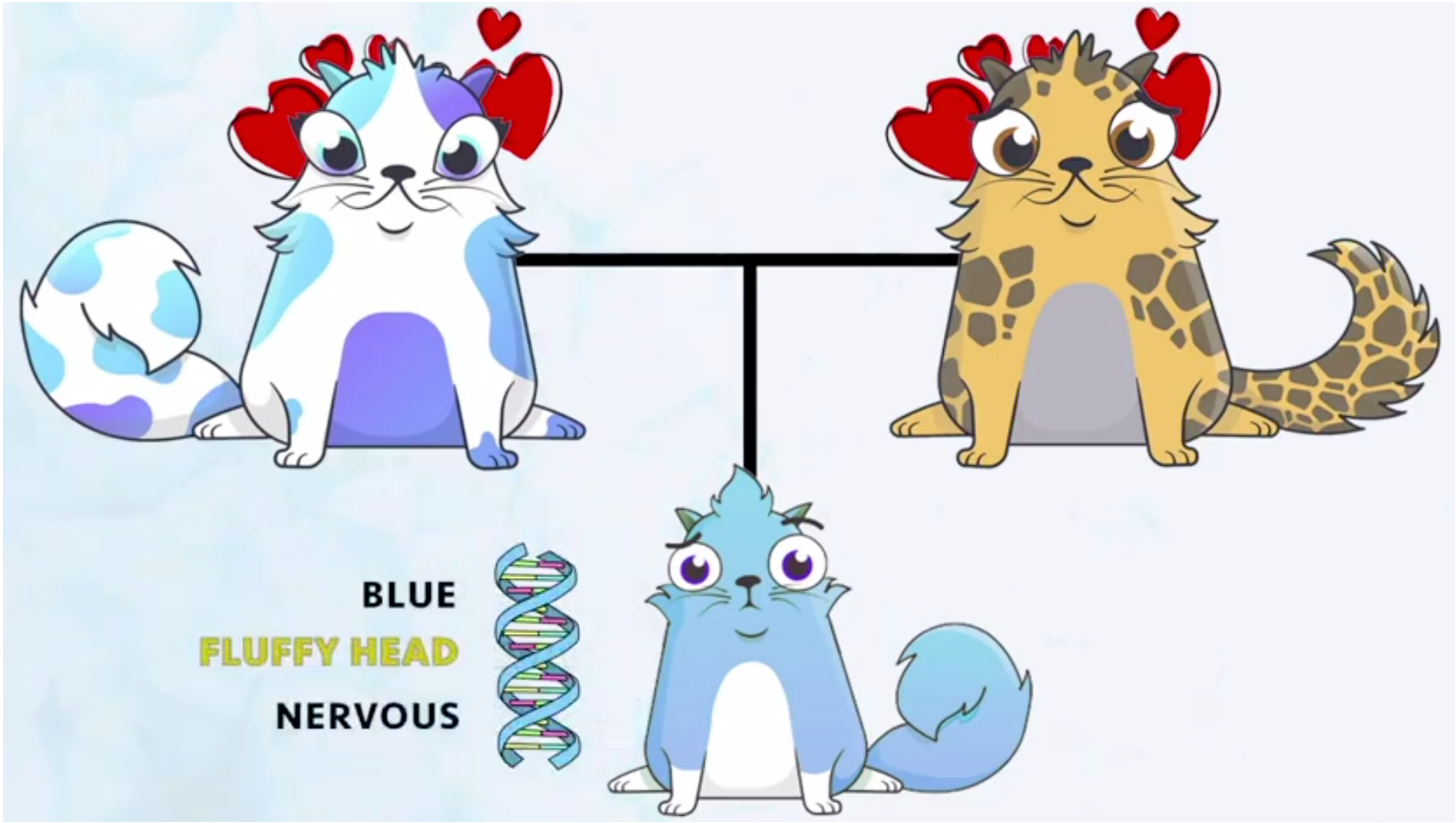
- “Introduction to Ethereum” presented at Swiss Blockchain Summer School by J. Bonneau
- D. Finlay (ConsenSys)
- “Introduction to Smart Contracts” presented at UCB DeFi MOOC by A. Miller

Contract: Launch Your Own Token

```
contract PhilipToken {

    /* Maps account addresses to token balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply
       tokens to the creator of the contract */
    function PhilipToken(uint256 initialSupply)
    {
        // Give the creator all initial tokens
        balanceOf[msg.sender] = initialSupply;
    }

    /* Send tokens to a recipient address */
    function transfer(address to, uint256 value)
    {
        if (balanceOf[msg.sender] < value) throw;    // Check if the sender has enough
        balanceOf[msg.sender] -= value;              // Subtract from the sender
        balanceOf[to] += value;                      // Add the same to the recipient
    }
}
```

Smart Contract Circa 2017 - 2018

Smart Contract	Transaction Count	Category
EtherDelta	10354398	Decentralized Exchange
IDEX_1	4590376	Decentralized Exchange
EOS Token	2952885	ICO
CryptoKitties Smart Contract	2568983	Collectible Token
Tron Token	1967331	ICO
Poloniex_3	1720771	Centralized Exchange
Bittrex_2	1527197	Centralized Exchange
Bittrex Wallet	1501350	Centralized Exchange
BTCM	1451763	ICO
OmiseGO	1350274	ICO



CryptoKitties are crypto collectibles; each CryptoKitty has a unique DNA whose string representation is stored on the Ethereum blockchain - this is an NFT (Non-Fungible Token).

Such an NFT can be sold on a smart contract platform.

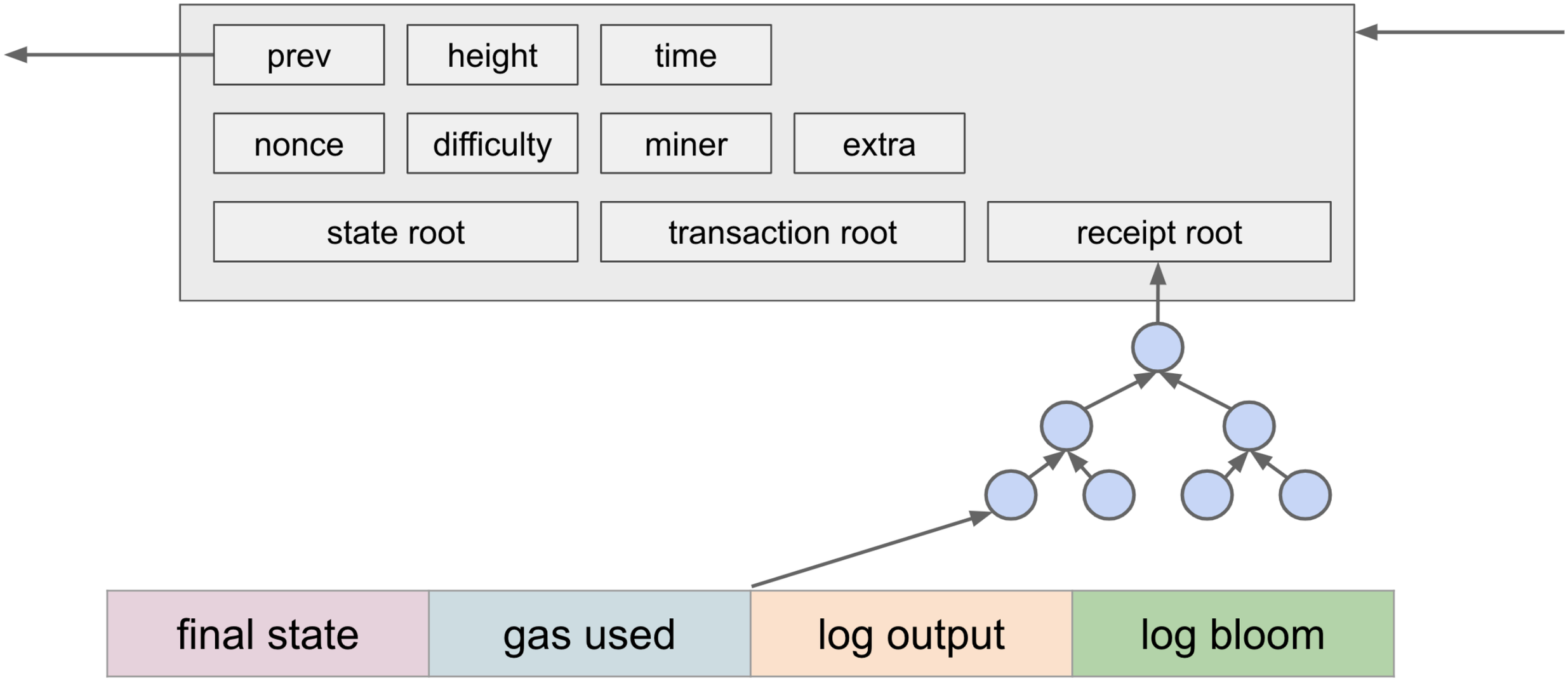
Cryptokitty Purchase with Dutch Auction

Dutch Auction in Solidity

```
contract DutchAuction {
    // Parameters
    uint public initialPrice; uint public biddingPeriod;
    uint public offerPriceDecrement; uint public startTime;
    KittyToken public kitty; address payable public seller;
    address payable winnerAddress;

    function buyNow() public payable {
        uint timeElapsed = block.timestamp - startTime;
        uint currPrice = initialPrice - (timeElapsed * offerPriceDecrement);
        uint userBid = msg.value;
        require (winnerAddress == address(0)); // Auction hasn't ended early
        require (timeElapsed < biddingPeriod); // Auction hasn't ended by time
        require (userBid >= currPrice); // Bid is big enough

        winnerAddress = payable(msg.sender);
        winnerAddress.transfer(userBid - currPrice);  // Refund the difference
        seller.transfer(currPrice);
        kitty.transferOwnership(winnerAddress);
    }

```

Event

```
event Registered(address registrant, string domain);

function registerDomain(string memory domain) public {
    // Can only reserve new unregistered domain names
    require(registry[domain] == address(0));

    // Update the owner of this domain
    registry[domain] = msg.sender;

    emit Registered(msg.sender, domain);
}
```



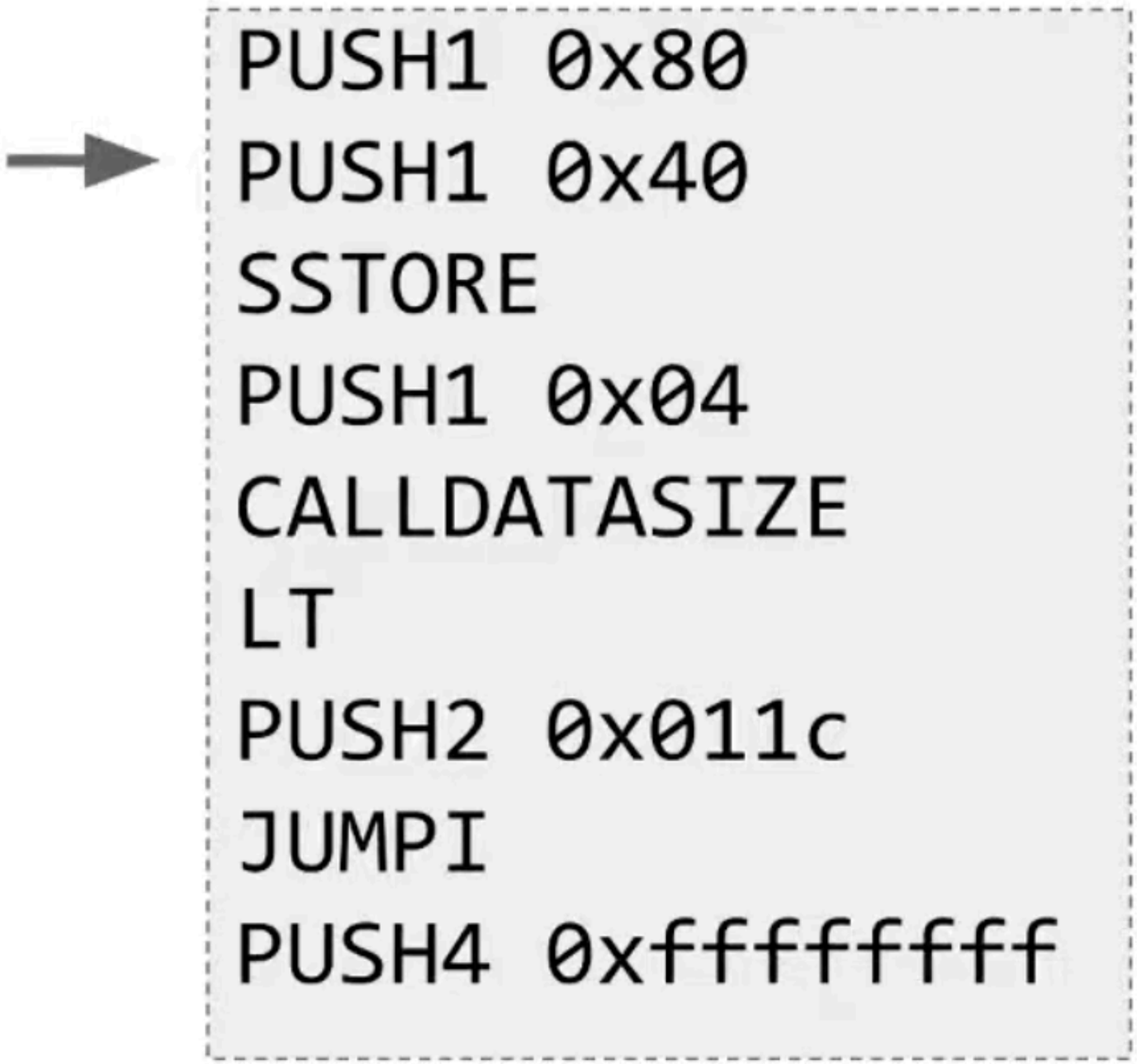
Emitted events stored as log output on the receipt tree

Gas in Ethereum

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x04
CALLDATASIZE
LT
PUSH2 0x011c
JUMPI
PUSH4 0xffffffff
```

Gas remaining

1000000



Gas remaining

999997

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x04
CALLDATASIZE
LT
PUSH2 0x011c
JUMPI
PUSH4 0xffffffff
```

Gas remaining

979994

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x04
CALLDATASIZE
LT
PUSH2 0x011c
JUMPI
PUSH4 0xffffffff
```

Gas remaining

971337

Why Do We Need Gas

- Users pay gas fees to miners to incentivize them to include complex transactions in a block
  - Such transactions consume lots of gas, so miners get paid more
    - Otherwise, miners would want to include only simple transactions (executed using less resources) in a block
- To prevent spams, the network asks users to pay gas fees to use its service, i.e., running smart contacts
- Deal with the halting problem

Provision for Gas Fees

- Transaction to be executed must specify START_GAS and GAS_PRICE
- START_GAS
  - Estimated maximum amount of gas to be used in processing this transaction
- GAS_PRICE
  - Per unit price of gas in wei
- START_GAS * GAS_PRICE = Money (in ETH) that you are willing to spend to run your transaction
  - This amount of ETH must be available in the account that originates the run of this transaction

Amount of Gas Consumed per Instruction

	opcodes	gas cost
Basic operations	ADD, MUL, PUSH, JUMP	2-10
Storage read	SLOAD	200
Storage write	SSTORE	5000
Storage write (from zero)	SSTORE	20000
Storage zeroize	SSTORE	-10000
Contract call	CALL, CODECALL, etc.	700
Transaction overhead	n/a	21000
Contract creation	n/a	32000
Contract destruction	SELFDESTRUCT	-19000

https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs/edit#gid=0

C.foo()

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x00
PUSH1 0x40
SSTORE
```

alice.balance

1000000

C.foo()

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x00
PUSH1 0x40
SSTORE
```

alice.balance

1000000

Alice wants to run C.foo() with START_GAS = 100000 and GAS_PRICE = 2

The first SSTORE does storage write from zero (consumes gas = 20000)
The second SSTORE does storage write to zero (consumes gas = -10000)

C.foo()

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x00
PUSH1 0x40
SSTORE
```

alice.balance

800000

Gas remaining

100000

Gas refund

0

C.foo()

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x00
PUSH1 0x40
SSTORE
```

alice.balance

800000

Gas remaining

99994

Gas refund

0

C.foo()

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x00
PUSH1 0x40
SSTORE
```

alice.balance

800000

Gas remaining

79994

Gas refund

0

C.foo()

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x00
PUSH1 0x40
SSTORE
```

alice.balance

800000

Gas remaining

79988

Gas refund

0

C.foo()

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x00
PUSH1 0x40
SSTORE
```

alice.balance

800000

Gas remaining

74988

Gas refund

15000

C.foo()

```
PUSH1 0x80
PUSH1 0x40
```

Total refund:
(74988+15000) × 2
=179976

alice.balance

800000

Gas remaining

74988

Gas refund

15000

C.foo()

```
PUSH1 0x80
PUSH1 0x40
SSTORE
PUSH1 0x00
PUSH1 0x40
SSTORE
```

alice.balance

979976

What Happen When You Run Out of Gas

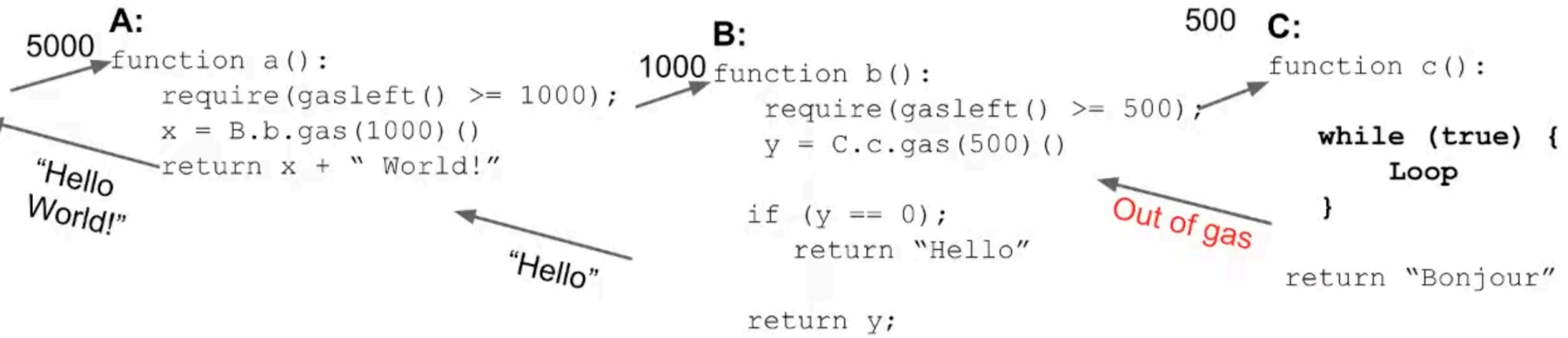
- All state changes nullified, reverting back to the original state before the transaction runs
- But, you need to pay ETH = START_GAS * GAS_PRICE even though you do not get any result

Preventing Out-of-Gas Exception

```
contract HelloWorld {
    Foo f;
    function bar(address a) public returns (int){
        if (gasleft() > 50000)
            f = new Foo();
        else
            f = Foo(a);

        return f.baz();
    }
}
```

Use gasleft() to check how much gas is available at a given execution point and choose appropriate operations accordingly

Limiting Gas Usage When Running a Contract

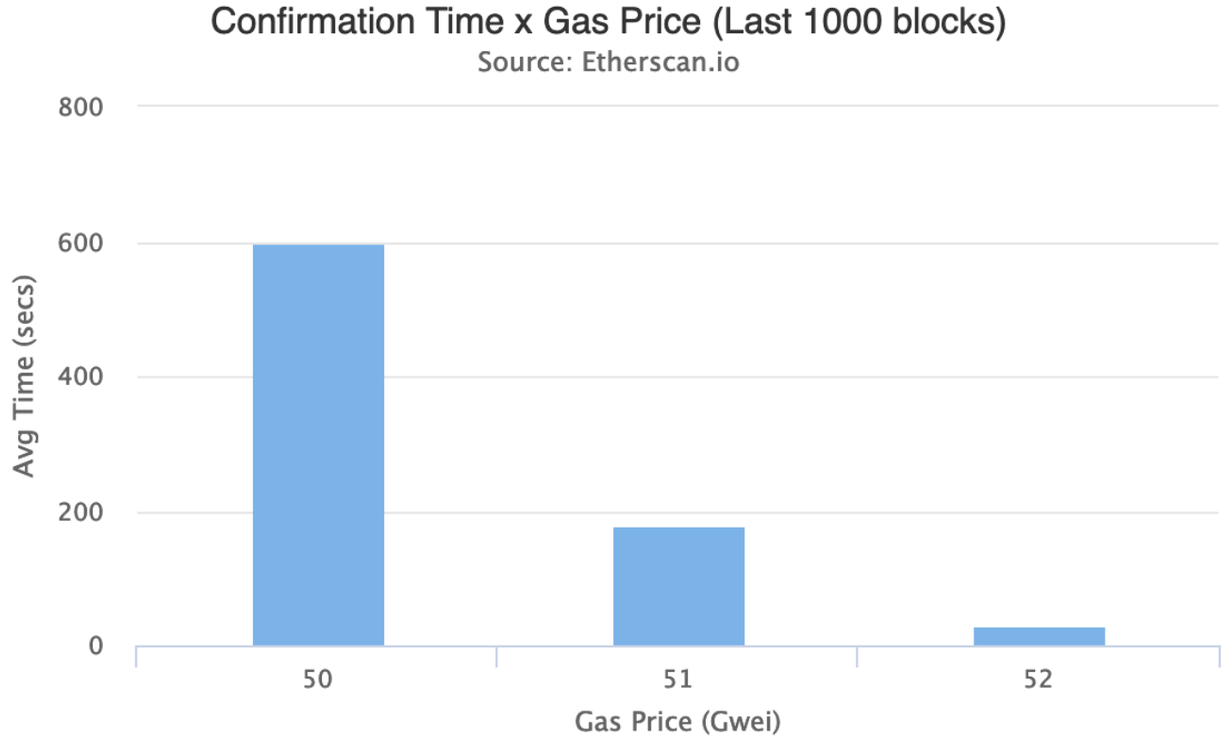


Miners Love Transactions with High Gas Fees

Next update in 1s

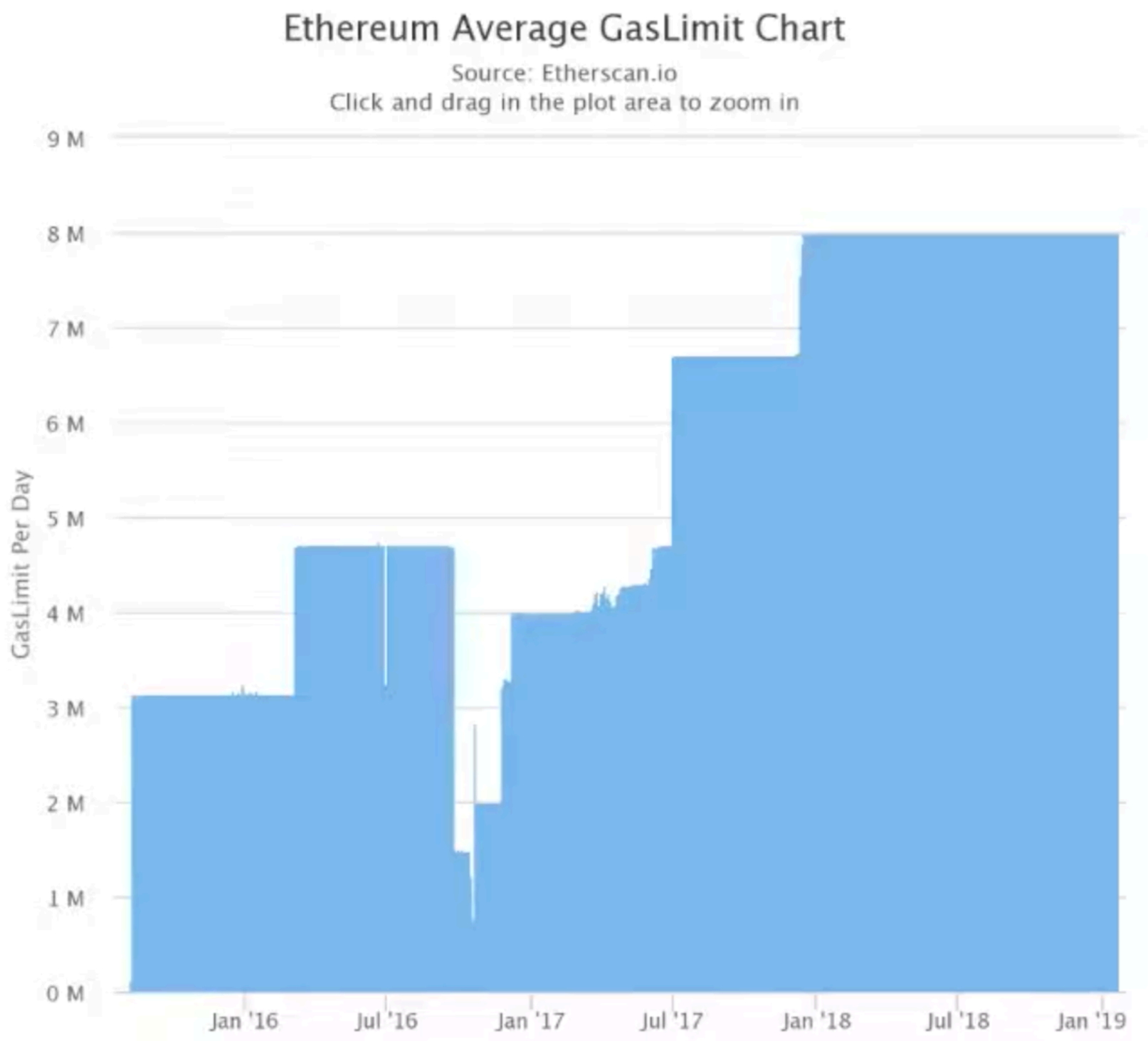
Wed, 09 Mar 2022 18:30:51 UTC

😁 Low	😀 Average	🙂 High
50 gwei	51 gwei	52 gwei
Base: 49 | Priority: 1	Base: 49 | Priority: 2	Base: 49 | Priority: 3
$2.57 | ~ 3 mins: 0 secs	$2.57 | ~ 3 mins: 0 secs	$2.69 | ~ 30 secs



etherscan.io/gastracker

Amount of Gas Allowed Per Block Is Limited



What We Have Learned

- Ethereum blockchain structures
  - Store account states, transactions, and receipts
- Ethereum network state is the states of all accounts in the network
  - Account state represented by a mapping from the account address to code, storage, balance, and nonce
- Basics of the Solidity programming language
- Gas and fees in Ethereum