Blockchains & Cryptocurrencies

Smart Contracts & Ethereum II

Instructors: Matthew Green & Abhishek Jain
Johns Hopkins University - Spring 2019

Many slides adapted from NBFMG

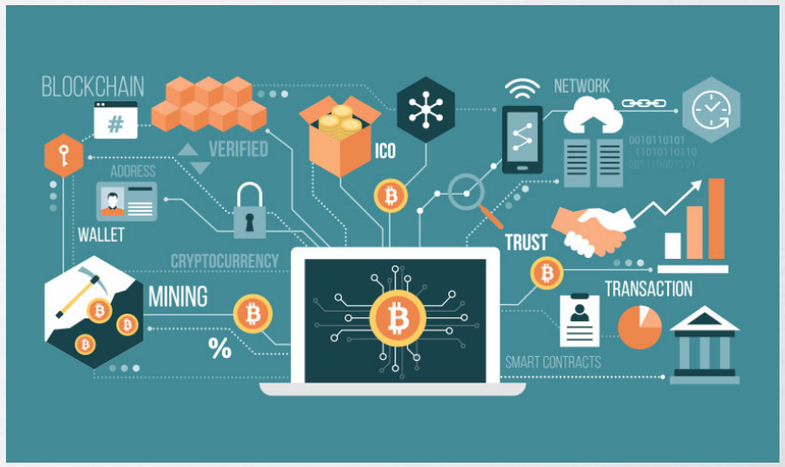
Housekeeping

- **Assignment 2 is out**
- Project proposals are extended to 3/1 (Friday)

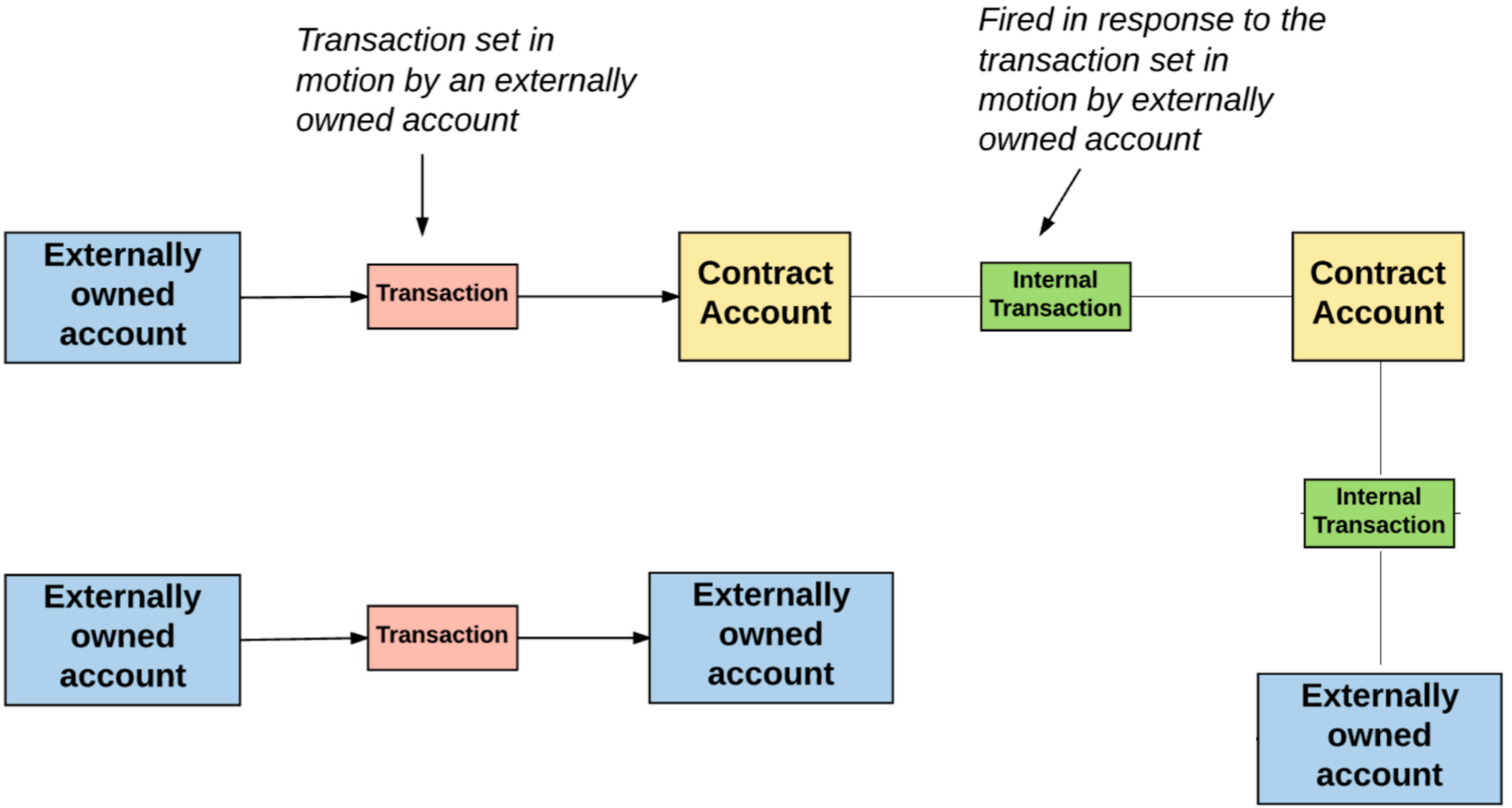
News?

Today

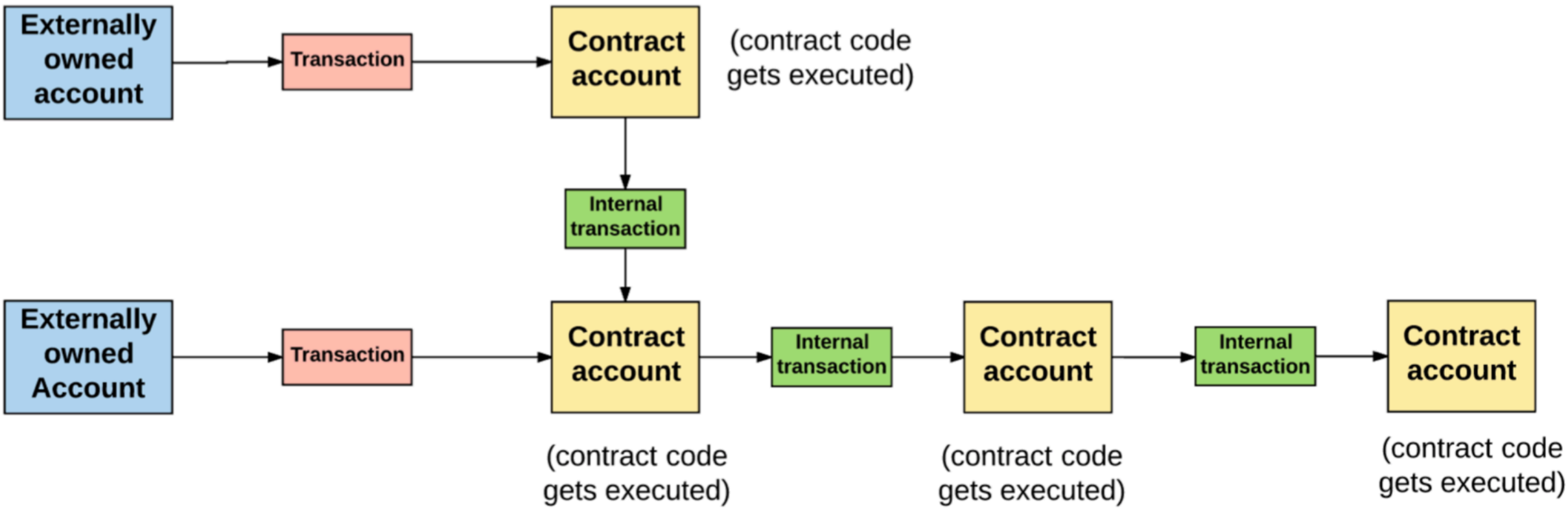
- Ethereum cont'd
- Applications of smart contracts



Review: Ethereum transactions



Review: Ethereum transactions



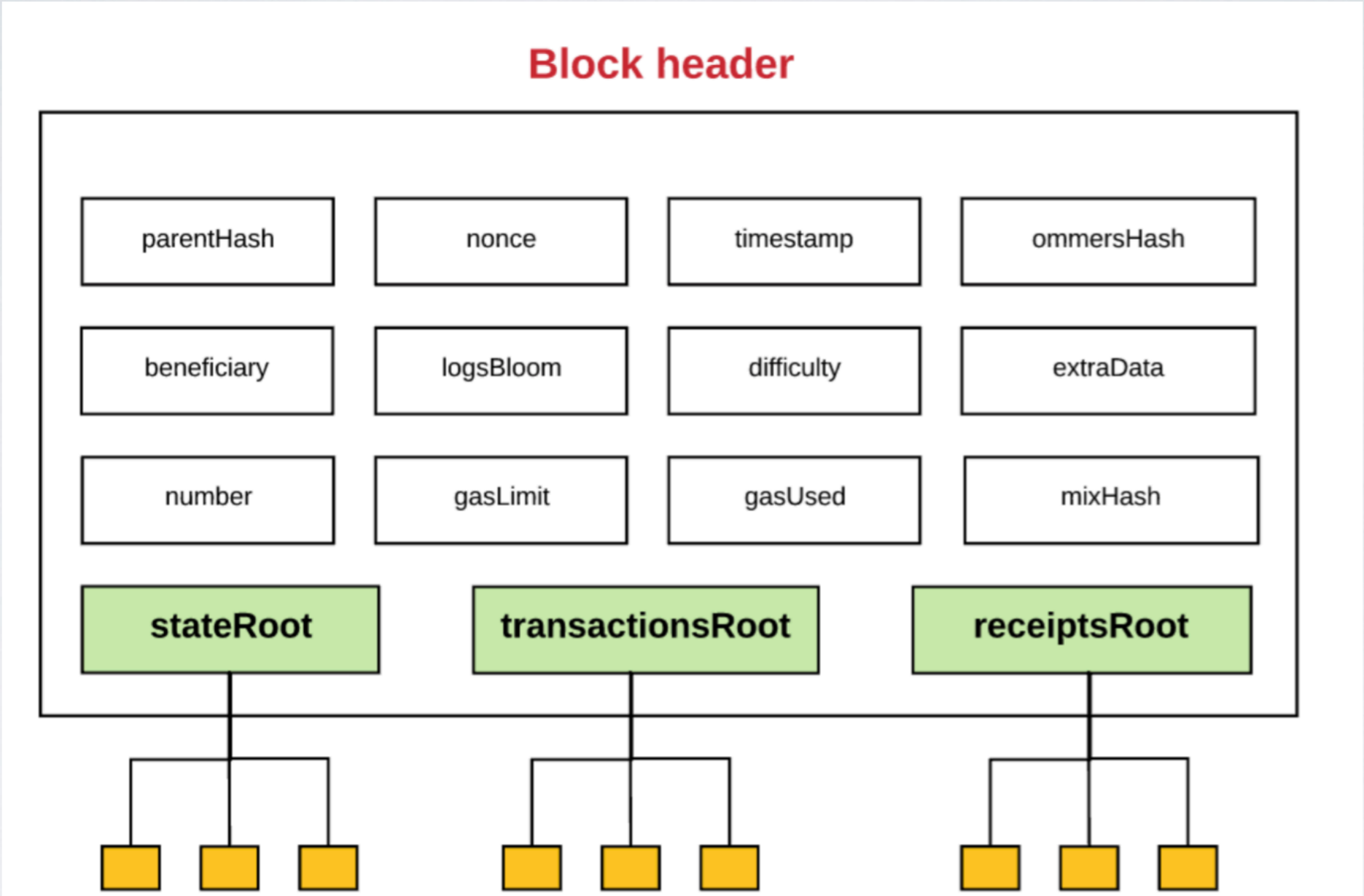
Contracts

- Smart contracts are often written in a high-level object-oriented language (e.g., Solidity)
  - The "contract" object has "methods", which can be public or private (internal)
  - Public methods can be called by anyone, private methods can only be called from other methods within that contract
  - External transactions contain the contract address + the data (arguments) for a method call

Review: Gas

- EVM code is "Turing complete", e.g., has loops and may not halt
  - Users pay fees to submit TXes to the network
    - (Pays for both computation and storage of data)
      - Contract computation/storage costs are denominated in "gas" (e.g., one hash call requires 30 "gas")
      - Each transaction contains a "gas limit", and a "gas price" price the sender will pay per unit of gas, denominated in ETH (currency)
        - Along with sufficient ETH to pay the fee

Merkle Trees



Binary Merkle Trees:

- Good data structure for authenticating information.
- Any edits/insertions/deletions are costly.

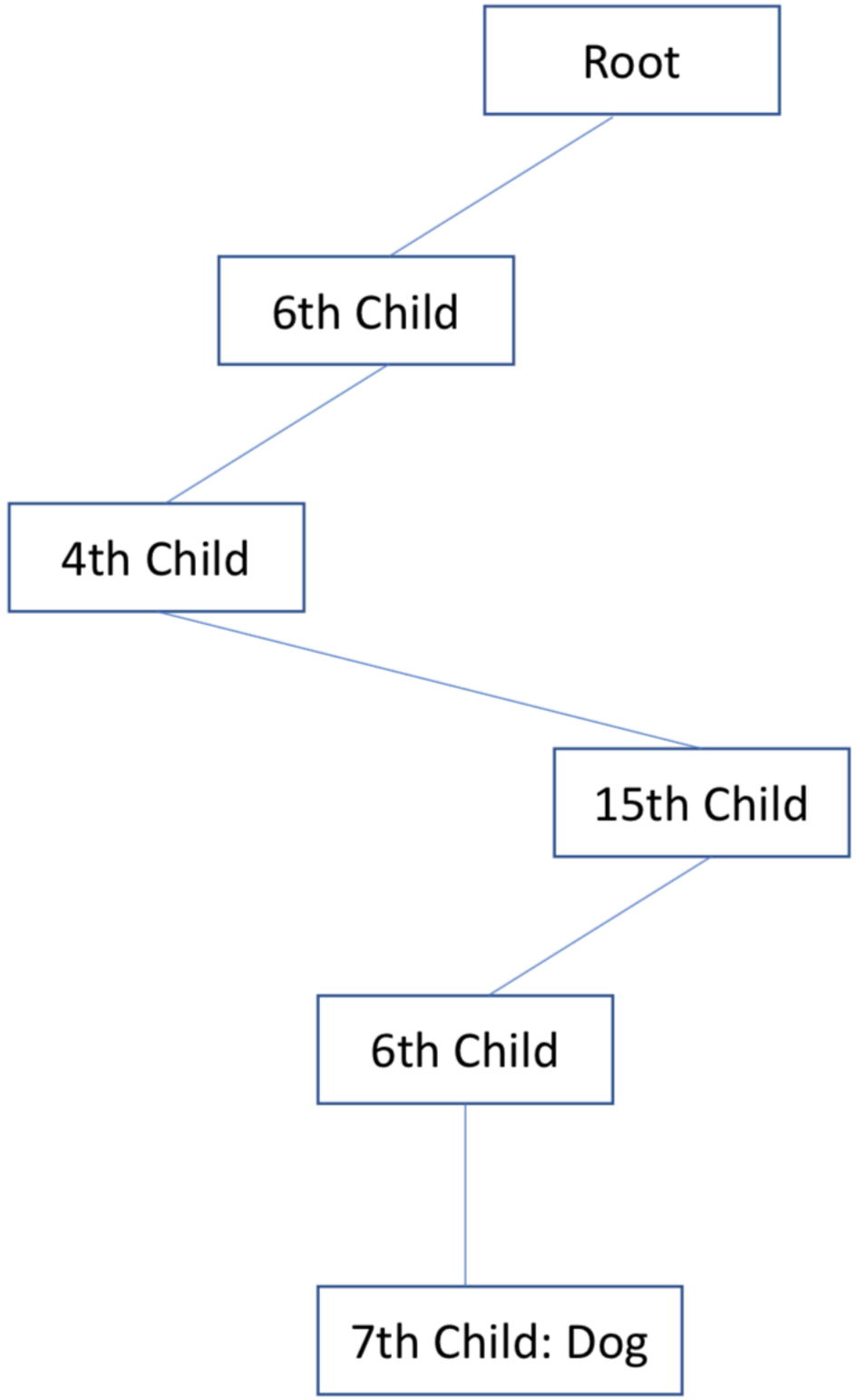
Merkle Patrica Trees:

- New tree root can be quickly calculated after an insert, update edit or delete operation in O(log n) time.
- Key-Value Pairs: Each value has a key associated with it.
- Key under which a value is stored is encoded into the path that you have to take down the tree.

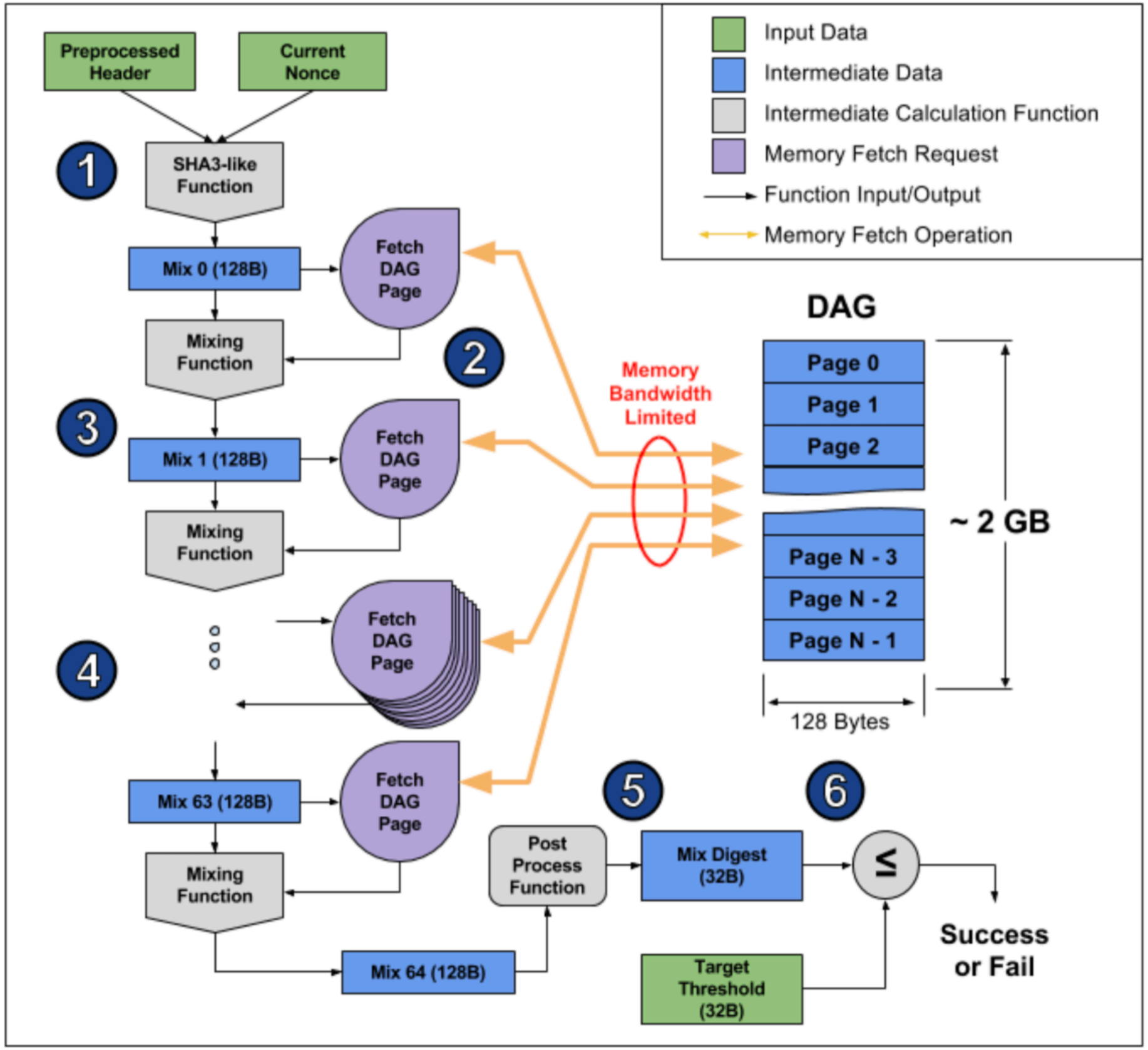
Why Trees?

Merkle Patricia Trees

- Each node has 16 children.
- eg: Hex(dog)= 6 4 6 15 6 7



Ethash Hashing Algorithm



Constructing the Chain

Ommers/Uncles

- An ommer is a block whose parent is equal to the current block's parent's parent.
- Block times in Ethereum are around 15 sec. This is much lower than that in Bitcoin (10 min).
- This enables faster transaction. But there are more competing blocks, hence a higher number of orphaned blocks

Constructing the Chain

Ommers/Uncles

- An ommer is a block whose parent is equal to the current block's parent's parent.
- Block times in Ethereum are around 15 sec. This is much lower than that in Bitcoin (10 min).
- This enables faster transaction. But there are more competing blocks, hence a higher number of orphaned blocks
- The purpose of ommers is to help reward miners for including these orphaned blocks.
- The ommers that miners include must be within the sixth generation or smaller of the present block.

There are plans to replace Ethereum PoW with "proof of stake", first through a "finality gadget" and later through a full PoS protocol.

This keeps getting delayed.

Contract examples

- Standard "custom token" contract (ERC20 interface)

```
1  // ----------------------------------------------------------------------
2  // ERC Token Standard #20 Interface
3  // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
4  // ----------------------------------------------------------------------
5  contract ERC20Interface {
6      function totalSupply() public view returns (uint);
7      function balanceOf(address tokenOwner) public view returns (uint balance);
8      function allowance(address tokenOwner, address spender) public view returns (uint remaining);
9      function transfer(address to, uint tokens) public returns (bool success);
10     function approve(address spender, uint tokens) public returns (bool success);
11     function transferFrom(address from, address to, uint tokens) public returns (bool success);
12
13     event Transfer(address indexed from, address indexed to, uint tokens);
14     event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
15 }
```

```
contract FixedSupplyToken is ERC20Interface {
    string public constant symbol = "FIXED";
    string public constant name = "Example Fixed Supply Token";
    uint8 public constant decimals = 18;
    uint256 _totalSupply = 1000000;

    // Owner of this contract
    address public owner;

    // Balances for each account
    mapping(address => uint256) balances;

// Owner of account approves the transfer of an amount to another account
mapping(address => mapping (address => uint256)) allowed;

// Functions with this modifier can only be executed by the owner
modifier onlyOwner() {
    if (msg.sender != owner) {
        require(msg.sender == owner);
    }
    _;
}
```

- Stanc

```
// Constructor
function FixedSupplyToken() {
    owner = msg.sender;
    balances[owner] = _totalSupply;
}

function totalSupply() constant returns (uint256) {
    return _totalSupply;
}

// What is the balance of a particular account?
function balanceOf(address _owner) constant returns (uint256 balance) {
    return balances[_owner];
}

// Transfer the balance from owner's account to another account
function transfer(address _to, uint256 _amount) returns (bool success) {
    if (balances[msg.sender] >= _amount
        && _amount > 0
        && balances[_to] + _amount > balances[_to]) {
        balances[msg.sender] -= _amount;
        balances[_to] += _amount;
        Transfer(msg.sender, _to, _amount);
        return true;
    } else {
        return false;
    }
}
```

- Stand

```
// Send _value amount of tokens from address _from to address _to
// The transferFrom method is used for a withdraw workflow, allowing contracts to send
// tokens on your behalf, for example to "deposit" to a contract address and/or to charge
// fees in sub-currencies; the command should fail unless the _from account has
// deliberately authorized the sender of the message via some mechanism; we propose
// these standardized APIs for approval:
function transferFrom(
    address _from,
    address _to,
    uint256 _amount
) returns (bool success) {
    if (balances[_from] >= _amount
        && allowed[_from][msg.sender] >= _amount
        && _amount > 0
        && balances[_to] + _amount > balances[_to]) {
        balances[_from] -= _amount;
        allowed[_from][msg.sender] -= _amount;
        balances[_to] += _amount;
        Transfer(_from, _to, _amount);
        return true;
    } else {
        return false;
    }
}

// Allow _spender to withdraw from your account, multiple times, up to the _value amount.
// If this function is called again it overwrites the current allowance with _value.
function approve(address _spender, uint256 _amount) returns (bool success) {
    allowed[msg.sender][_spender] = _amount;
    Approval(msg.sender, _spender, _amount);
    return true;
}

function allowance(address _owner, address _spender) constant returns (uint256 remaining) {
    return allowed[_owner][_spender];
```

Solidity

- Every function (method) in a contract has a "selector", which consists of the first four bytes of the hash of the function signature
  - E.g., **bytes4(keccak256("transfer(address,uint256)");**
- Arguments are encoded as a series of left-padded 32-byte values
- Contracts also have a "fallback" function that receives inputs when no matching selector is found

Solidity/EVM

- You can use casting to turn an address into an interface:
  - **SillyContract silly = SillyContract(_to)**
- Solidity handles all aspects of contract calls, etc. (including formulating Ethereum transactions)
- EVM has a limited stack (16 stack variables per scope)
  - Each is 32 bytes wide
  - Standard Solidity types include e.g., uint256, address, etc.

Some slides adapted from: http://cs251crypto.stanford.edu/18au-cs251/lectures/lecture10.pdf

Solidity/EVM

- Calldata (input arguments)
  - A read-only array that functions can access
  - Each byte costs 68 gas
    - Copying these to memory costs more gas, so not worth doing
- Memory:
  - Just an array of bytes
  - Things that aren't 32-byte values need to go there

Some slides adapted from: http://cs251crypto.stanford.edu/18au-cs251/lectures/lecture10.pdf

Solidity/EVM

- Storage
  - This is permanent storage in the Eth database
  - Most expensive: though you may get some gas back when this stuff is deleted

Some slides adapted from: http://cs251crypto.stanford.edu/18au-cs251/lectures/lecture10.pdf

Solidity/EVM

operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{selfdestruct}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{\cdots}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.

Some slides adapted from: http://cs251crypto.stanford.edu/18au-cs251/lectures/lecture10.pdf

Review: EVM

To prevent cheating, the network works like Bitcoin:

Every single node in the network must also run the EVM machine for each transaction in a received block,
and only accepts the block if the EVM outputs (in the block) match their local computations.

Verification through repeated computing:

Each contract execution is "replicated" across the entire Ethereum network!

- What if that node cheats?

"NameCoin" in Ethereum

```
contract Namespace {

    struct NameEntry {
        address owner;
        bytes32 value;
    }

    uint32 constant REGISTRATION_COST = 100;
    uint32 constant UPDATE_COST = 10;
    mapping(bytes32 => NameEntry) data;

    function nameNew(bytes32 hash){
        if (msg.value >= REGISTRATION_COST){
            data[hash].owner = msg.sender;
        }
    }

    function nameUpdate(bytes32 name, bytes32 newValue, address newOwner){
        bytes32 hash = sha3(name);
        if (data[hash].owner == msg.sender && msg.value >= UPDATE_COST){
            data[hash].value = newValue;
            if (newOwner != 0){
                data[hash].owner = newOwner;
            }
        }
    }

    function nameLookup(bytes32 name){
        return data[sha3(name)];
    }

}
```

Credit: Andrew Miller and Joe Bonneau for this code

Multisig and filters

- Can create "filter" contracts that execute another contract and/or pay out money if complex conditions are satisfied
  - E.g., If k-out-of-N signers sign (in Bitcoin this is called "multisig")
  - Verify that a certain number of blocks have elapsed
  - Check that another contract executed

- Can c
pay o
  - E.g.,
  - Ver
- Che

```
contract SimpleMultiSig {

  uint public nonce;                // (only) mutable state
  uint public threshold;            // immutable state
  mapping (address => bool) isOwner; // immutable state
  address[] public ownersArr;        // immutable state

  function SimpleMultiSig(uint threshold_, address[] owners_) {
    if (owners_.length > 10 || threshold_ > owners_.length ||
threshold_ == 0) {throw;}

    for (uint i=0; i<owners_.length; i++) {
      isOwner[owners_[i]] = true;
    }
    ownersArr = owners_;
    threshold = threshold_;
  }

  // Note that address recovered from signatures must be strictly
increasing
  function execute(uint8[] sigV, bytes32[] sigR, bytes32[] sigS,
address destination, uint value, bytes data) {
    if (sigR.length != threshold) {throw;}
    if (sigR.length != sigS.length || sigR.length != sigV.length)
{throw;}
```

Prediction Markets

- Remember that contracts can "control" a balance (in ETH)
  - (They can control balances in e.g., ERC20 tokens as well)
  - Can make payouts of ETH conditional on certain events — e.g., signed by a notary
    - Requires: method to "place bet"
    - Method to "claim bet" (verify sig/conditions), pay to an address
    - Relies on a centralized notary! See Augur….

DAO

- Decentralized Autonomous Organization
  - "Like a VC fund" but decentralized
  - Implementation: a contract that controls money, and directs its disbursement according to "shareholder votes"
- Shareholders buy in, pool their ETH (sending to contract)
- Then vote on investments, which are made together
- Users can "split" a DAO

“The DAO”

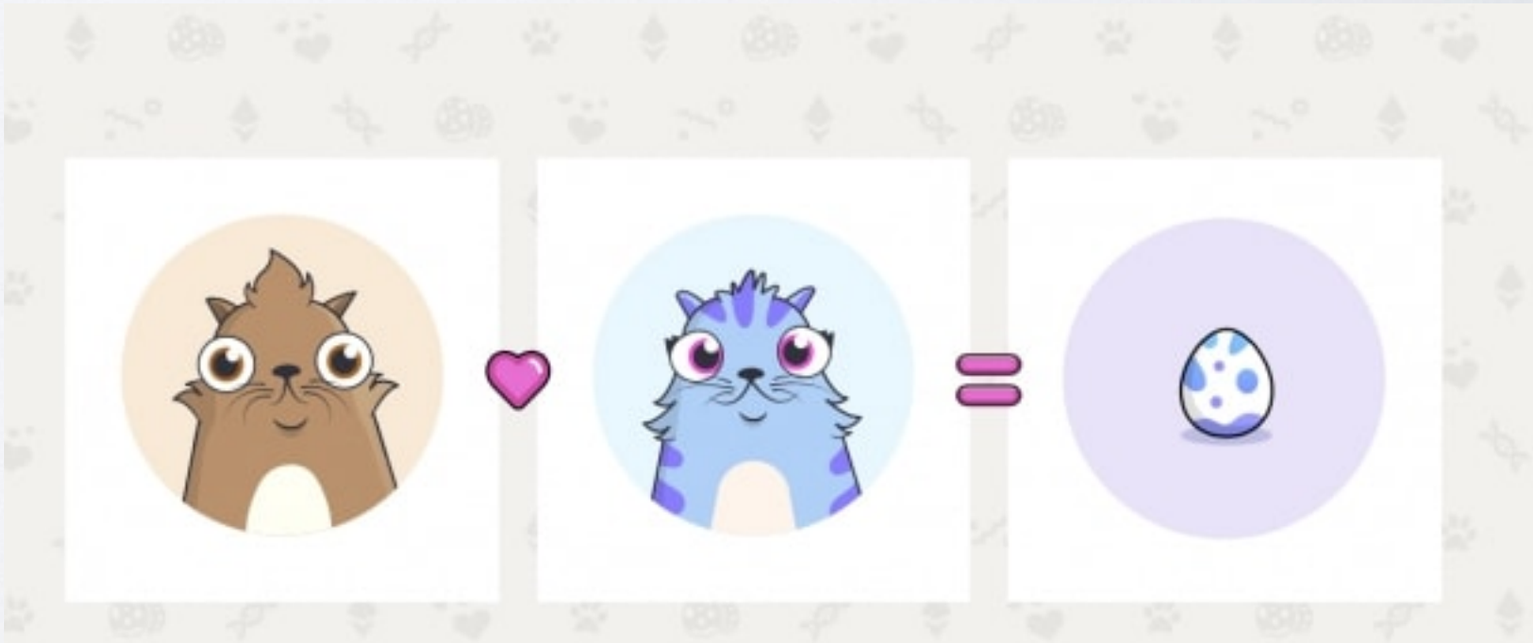
```
function splitDAO(
  uint _proposalID,
  address _newCurator
) noEther onlyTokenholders returns (bool _success) {

  ...
  // XXXXX Move ether and assign new Tokens.  Notice how this is done first!
  uint fundsToBeMoved =
      (balances[msg.sender] * p.splitData[0].splitBalance) /
      p.splitData[0].totalSupply;
  if (p.splitData[0].newDAO.createTokenProxy.value(fundsToBeMoved)(msg.sender)
== false) // XXXXX This is the line the attacker wants to run more than once
      throw;

  ...
  // Burn DAO Tokens
  Transfer(msg.sender, 0, balances[msg.sender]);
  withdrawRewardFor(msg.sender); // be nice, and get his rewards
  // XXXXX Notice the preceding line is critically before the next few
  totalSupply -= balances[msg.sender]; // XXXXX AND THIS IS DONE LAST
  balances[msg.sender] = 0; // XXXXX AND THIS IS DONE LAST TOO
  paidOut[msg.sender] = 0;
  return true;
}
```

How to upgrade a contract?

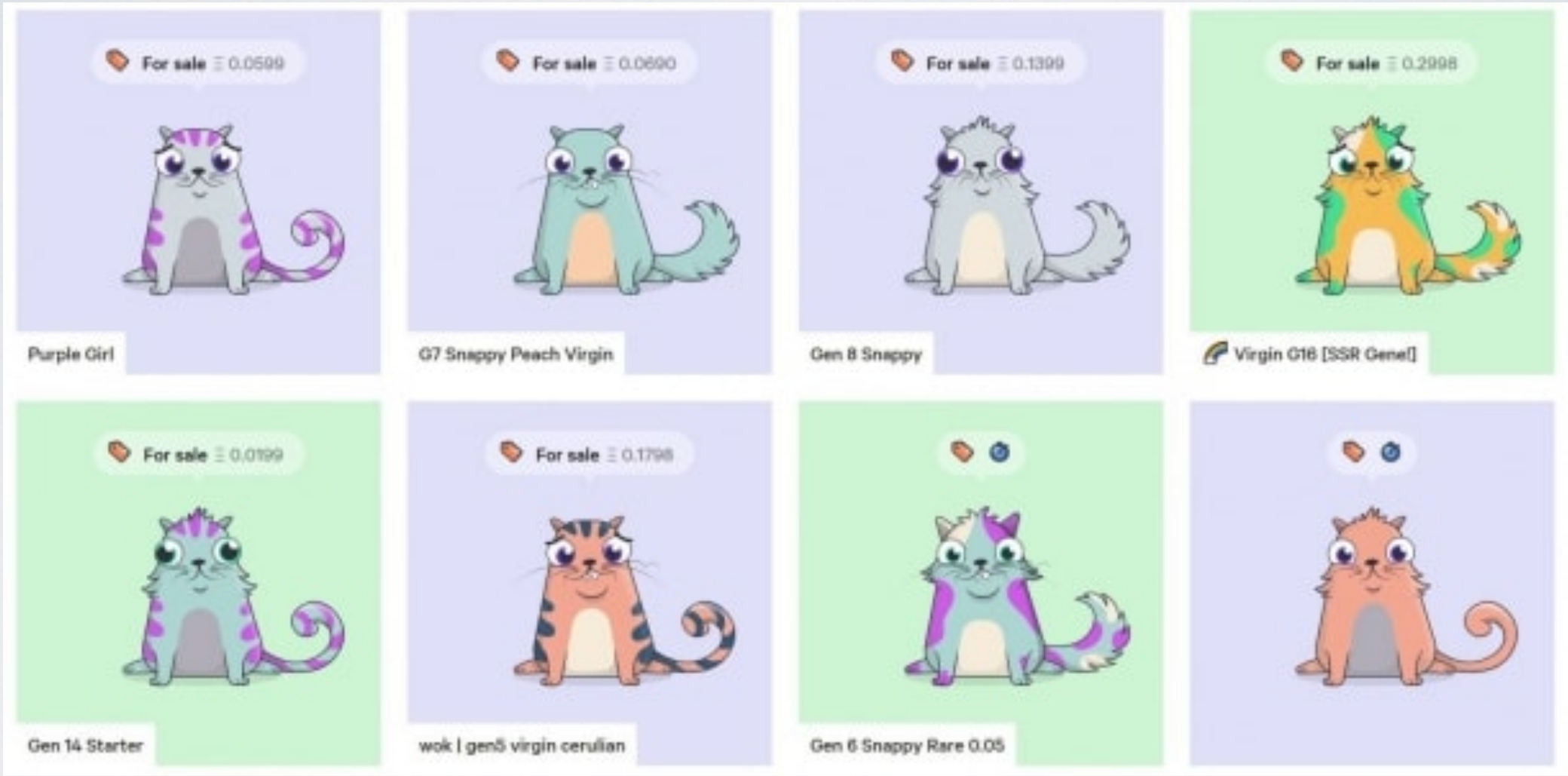
- If there is no useful ongoing state, don't: just replace it
- If there is ongoing state (e.g., account balances) then:
  - Don't allow upgrades — and pray you got the code right
  - Call upgradeable/replaceable library code
  - Create a complex mechanism to transfer state from an old contract instance to a new contract instance

Some applications

- Cryptokitties
  - These are non-fungible tokens, each with a unique "DNA signature"
  - They can "get pregnant" and "give birth"
  - The resulting process combines "DNA" from the sirer and matron to produce a new "kitty"! Yay!



- Cryptokitties



- Cryptokitties

Etherscan

Search by Address / Txhash / Block / Token / Ens

Eth: $138.19 (+1.03%)

Home | Blockchain | Tokens | Resources | More | Sign In

Token CryptoKitties

CryptoKitties | NFT | Collectibles

Crypto Loan

Overview [ERC-721]

Total Supply:	1,421,167 CK
Holders:	72,463 addresses
Transfers:	3,633,295

Profile Summary

Contract:	0x06012c8cf97bead5deae237070f9587f8e7a266d
Official Site:	https://www.cryptokitties.co/
Social Profiles:	

Transfers | Holders | Inventory | Info | DEX Trades | Read Contract | Write Contract | Comments (7)

Read Contract Information [Reset]

- Cryp

```
function giveBirth(uint256 _matronId)
        external
        whenNotPaused
        returns(uint256)
    {
        Kitty storage matron = kitties[_matronId];

// Check that the matron is a valid cat.
        require(matron.birthTime != 0);

// Check that the matron is pregnant, and that its time has come!
        require(_isReadyToGiveBirth(matron));

// Grab a reference to the sire in storage.
        uint256 sireId = matron.siringWithId;
        Kitty storage sire = kitties[sireId];

// Determine the higher generation number of the two parents
        uint16 parentGen = matron.generation;
        if (sire.generation > matron.generation) {
            parentGen = sire.generation;
        }

// Call the sooper-sekret gene mixing operation.
        uint256 childGenes = geneScience.mixGenes(matron.genes,
sire.genes, matron.cooldownEndBlock - 1);
```

Annotated code from: https://medium.com/@montedong/towards-cracking-crypto-kitties-genetic-code-629fcd37b09b

- Cryptokitties
  -


```
0x60606040526004361061006c5763ffffffff7c0100000000000000000000000000
00000000000000000000000000000060003504166630d9f5aed811461007157806315
97ee441461009f57806354c15b82146100ee57806361a769001461011557806377a7
4a201461017e575b600080fd5b341561007c57600080fd5b61008d60043560243560
44356101cd565b6040519081526020016040518090........
```


Annotated code from: https://medium.com/@montedong/towards-cracking-crypto-kitties-genetic-code-629fcd37b09b

- Reverse engineering Cryptokitties (see link below):
  - Kitty gets "pregnant", I.e., commit to DNA of sirer and matron
  - Then kitty "gives birth" sometime later, and this is when DNA is computed
- The DNA combination is randomized by the Ethereum block hash at birth time
- Are there attacks on this type of randomness?

Annotated code from: https://medium.com/@montedong/towards-cracking-crypto-kitties-genetic-code-629fcd37b09b

- Decentralized exchanges
  - Allow you to trade Ethereum-native tokens (e.g., ERC20 tokens)
  - If you could perform a "swap" from another blockchain to an Ethereum token, then you could trade that currency too
- What problems come up here?