Lecture 7:
Ethereum and Smart Contracts

Course instructors: Alexey Frolov and Yury Yanovich

Teaching assistant: Stanislav Kruglik

Nobember 20, 2018

«Traditional» contract

Execution

Interaction between human and objects.
Possible application of punishments
Definitions can be interpreted (trial)

Contract

Paper

Definition

Audit
(Control)

(semi-) **manual**
data recollection

Smart contract

Execution

Program execution
(Only one interpretation)
Not reversible, autonomous

Software
program

Contract

Definition

Audit
(Control)

Real time
Immutable

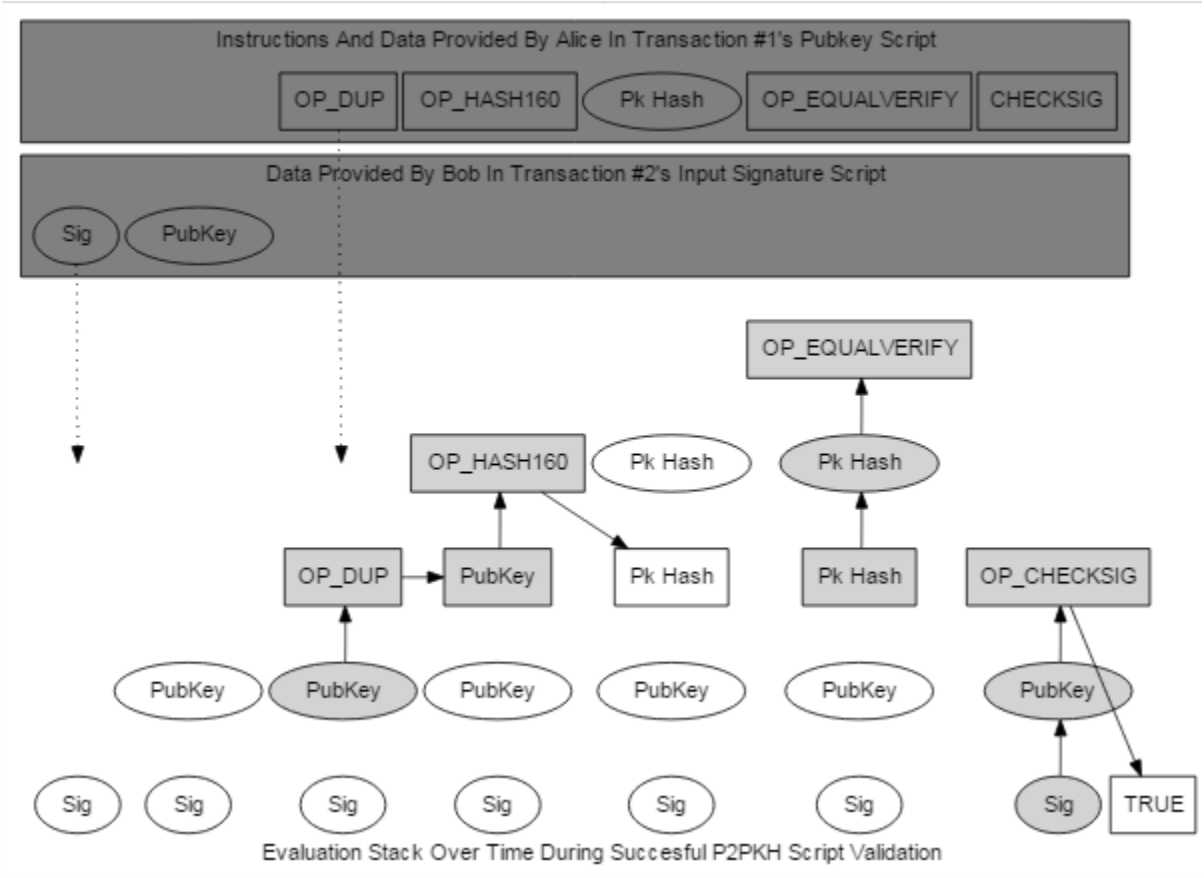
Bitcoin is a
smart contract!

Bitcoin is a smart contract

- It is a program
- Its execution is autonomous
  - because of the decentralized network
- Every transactions are public
- It is not possible to modify the history of transactions
  - The execution cannot be reverted

- A few **clauses/statements** of this contract
  - No more than 21.000.000 de bitcoins
  - A new block every 10 minutes
  - Mining difficulty is ajusted to the power of the network
  - Only a subset of possible transactions are allowed
  - …

Bitcoin transaction

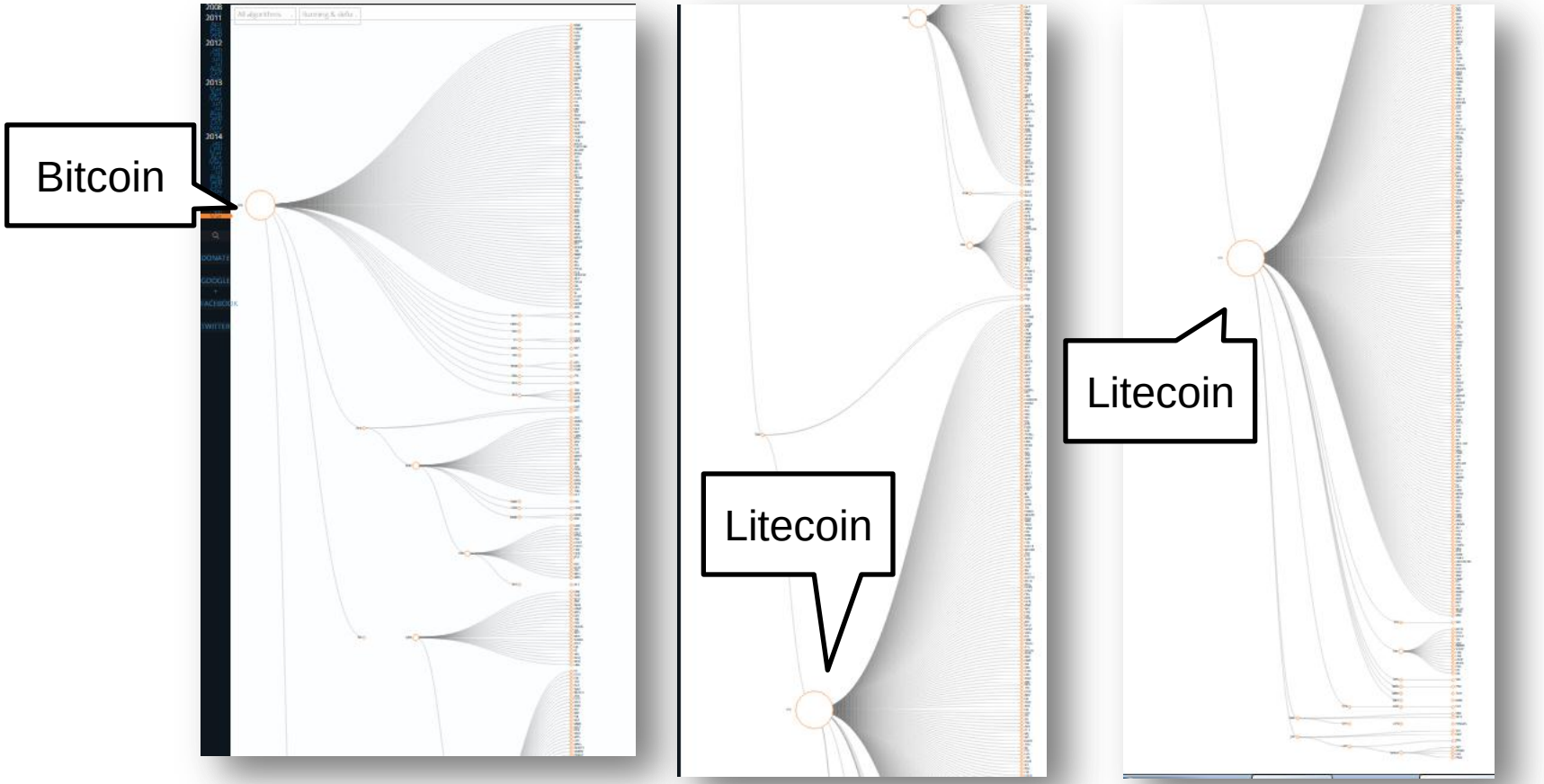


Much more complex than a simple signed message…

It is a smart contract!

https://bitcoin.org/en/developer-guide#signature-hash-types

Innovation v/s Fragmentation



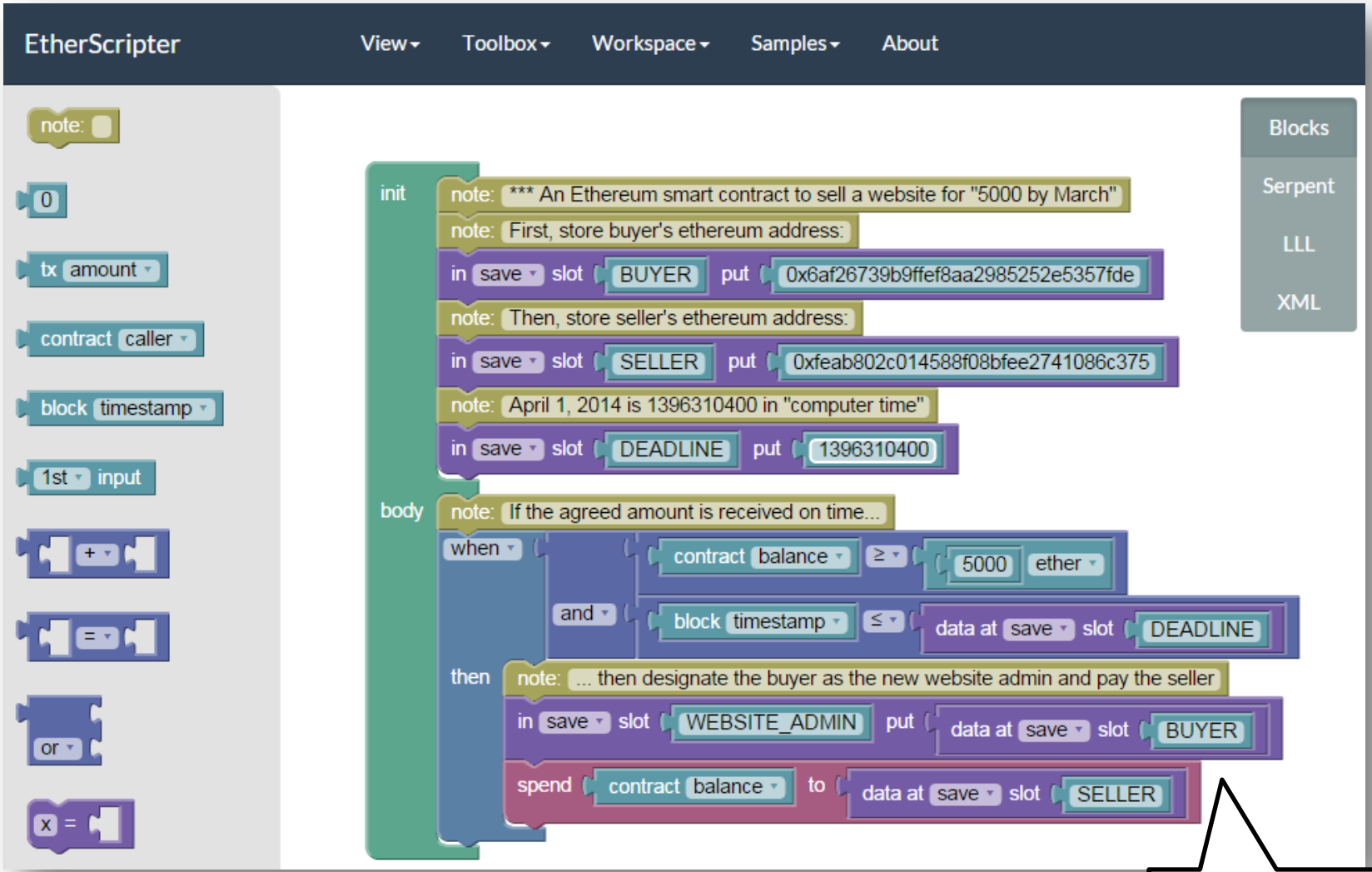
http://mapofcoins.com/bitcoin#

Ethereum

- Platform similar to Bitcoin but**:**
  - The language for writing smart contracts is more expressive (**Turing-Complete**)
  - Avoids to reinvent the wheel by forking an existing cryptocurrency
  - All the smart contracts use the same blockchain
- Crowdfunding (sept. 2014)
  - 31531 BTC = US$18,439,086
- Launching:
  - **it's live** since July 30th 2015!

Vitalik Buterin
Ethereum founder

Sales contract



http://etherscripter.com/0-5-1/

Hard problem solved: who pays/sends the product first?

More examples of smart contracts

- Decentralized DNS
- Autonomous companies
  - Define the shares at the beginning
  - Dividends can be distributed automatically
  - One could buy and sell stock instantly
- Insurance
- Heritance
- Direct democracy
- IOT (IBM+ Samsung using Ethereum => https://www.youtube.com/watch?v=U1XOPIqyP7A)

Great video, only 8 minutes

Using smart contracts for crime

- Enable to do business without relying on trust
  - => perfect for cybercrime
- Example of evil businesses
  - Selling secrets
  - DoS
  - Assassination
  - Defacement
- Relies on very sophisticated cryptography on top of smart contracts

http://www.arijuels.com/wp-content/uploads/2013/09/public_gyges.pdf

Ethereum Enterprise Vision

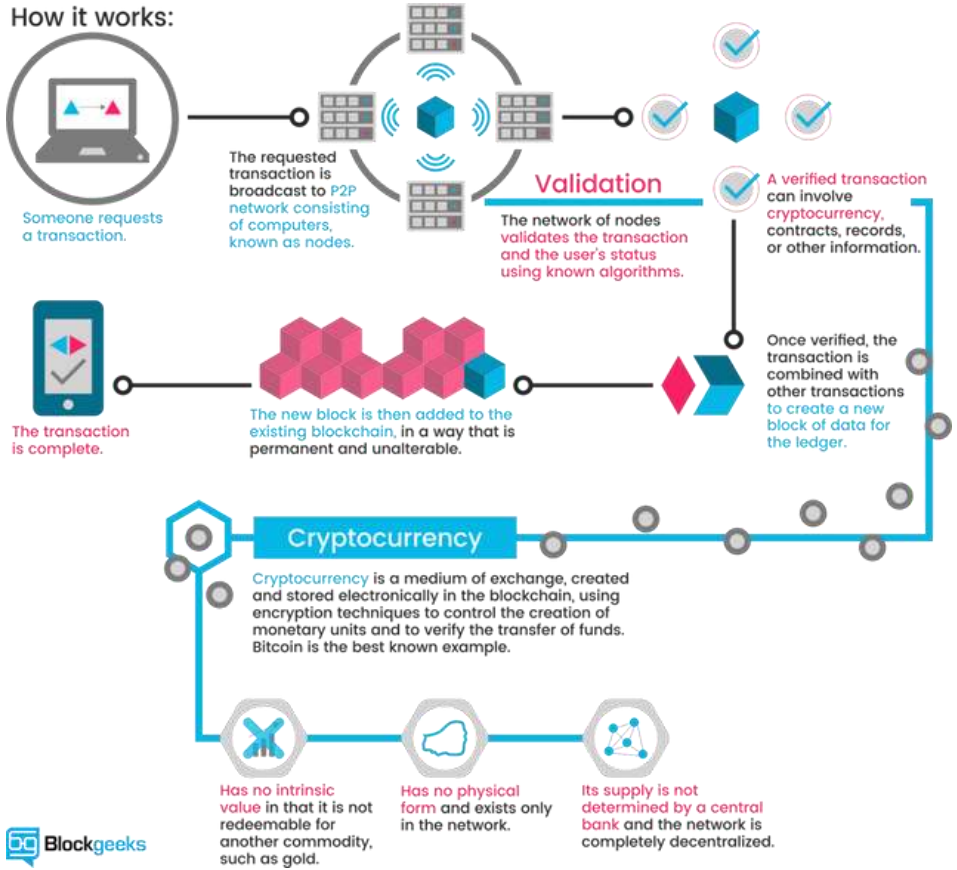
https://www.infoq.com/news/2017/03/Enterprise-Ethereum-Vision

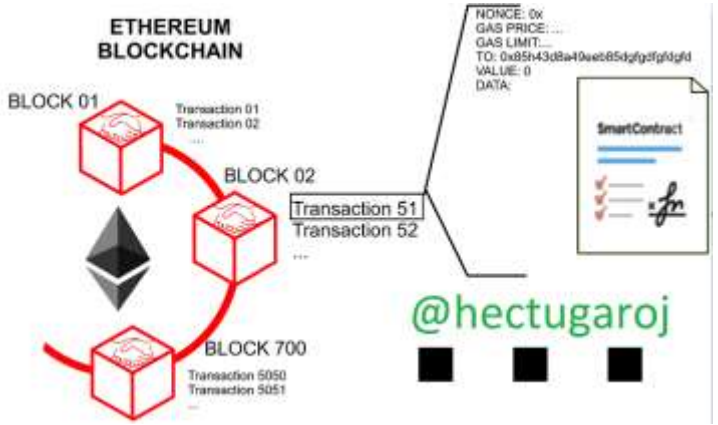
1. Develop a sufficiently modular Ethereum implementation to separate and define clear interfaces between networking and storage layers - that is a prototype for pluggable consensus that minimizes the code changes required to switch consensus algorithms.
2. Experiment with potential consensus algorithms, along with data privacy and permissioning frameworks.
3. Develop a clear set of capabilities and performance characteristics that suit the needs of enterprises, including:
    1. 100 transactions per second, across a 10 party network
    2. High volume and value use cases
    3. High availability/reliability
    4. Parallelization and horizontal scaling
4. Develop a Version 1 specification for Enterprise Ethereum, based on the learnings from the above plus the roadmap and requirements gathered from members, i.e., produce a reference implementation.
5. Leverage a robust governance process to ensure alignment and agreement on approaches

Pluggable Concensus

A common, modularized implementation will provide a code base for developing the Enterprise Ethereum specification and also experimenting with consortia consensus algorithms. Pluggable consensus needs a modularized client with a clean interface for networking and between the Ethereum Virtual Machine and the consensus algorithm - it is really these interfaces that make the consensus layer pluggable.

Crypto Currency General Flow





Account Types, Gas and Transactions

- http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html

- **"Gas" is the name for a special unit used in Ethereum.** It measures how much "work" an action or set of actions takes to perform: for example, to calculate one Keccak256 cryptographic hash it will take 30 gas each time a hash is calculated, plus a cost of 6 more gas for every 256 bits of data being hashed. **Every operation that can be performed by a transaction or contract on the Ethereum platform costs a certain number of gas**, with operations that require more computational resources costing more gas than operations that require few computational resources.

- The reason gas is important is that **it helps to ensure an appropriate fee is being paid** by transactions submitted to the network. By requiring that a transaction pay for each operation it performs (or causes a contract to perform), we ensure that network doesn't become bogged down with performing a lot of intensive work that isn't valuable to anyone. This is a different strategy than the Bitcoin transaction fee, which is based only on the size in kilobytes of a transaction. Since Ethereum allows arbitrarily complex computer code to be run, a short length of code can actually result in a lot of computational work being done. So **it's important to measure the work done directly instead of just choosing a fee based on the length of a transaction or contract**.

- So if gas is basically a transaction fee, how do you pay it? This is where it gets a little tricky. **Although gas is a unit that things can be measured in, there isn't any actual *token* for gas. That is, you can't own 1000 gas.** Instead, gas exists only inside of the Ethereum virtual machine as a count of how much work is being performed. **When it comes to actually *paying* for the gas, the transaction fee is charged as a certain number of *ether***, the built-in token on the Ethereum network and the token with which miners are rewarded for producing blocks.

- This might seem odd at first. **Why don't operations just have a cost measured in ether directly?** The answer is that ether, like bitcoins, have a market price that can change rapidly! But the cost of computation doesn't go up or down just because the price of ether changes. So **it's helpful to separate out the price of computation from the price of the ether token**, so that the cost of an operation doesn't have to be changed every time the market moves.

- The terminology here gets a little messy. Operations in the EVM have gas **cost**, but gas itself also has a gas **price** measured in terms of ether. Every transaction specifies the gas *price* it is willing to pay in ether for each unit of gas, allowing the market to decide the relationship between the price of ether and the cost of computing operations (as measured in gas). **It's the combination of the two, total gas used multiplied by gas price paid, that results in the total fee paid by a transaction**.

- As tricky as it is, it's important to understand this distinction, because it results in one of the most confusing things about Ethereum transactions to the initial learner: **there is a difference between your transaction running out of gas and your transaction not having a high enough fee**. If the **gas price** I set in my transaction is too low, no one will even bother to run my transaction in the first place. It will simply not be included in the blockchain by miners. But if I provide an acceptable gas *price*, and then my transaction results in so much computational work that the combined **gas costs** go past the amount I attached as a fee, **that gas counts as "spent"** and I don't get it back. The miner will stop processing the transaction, revert any changes it made, **but still include it in the blockchain as a "failed transaction", collecting the fees for it**. This may seem harsh, but when you realise that the real work for the miner was in performing the computation, you can see that they will never get those resources back either. So **it's only fair that you pay them for the work they did**, even though your badly designed transaction ran out of gas.

- Providing too big of a fee is also different than providing too much ether. **If you set a very high gas price, you will end up paying lots of ether for only a few operations**, just like setting a super high transaction fee in bitcoin. You'll definitely be prioritised to the front of the line, but your money is gone. If you provided a normal gas price, however, and just **attached more ether than was needed to pay for the gas that your transaction consumed, the excess amount will be refunded back to you**. Miners only charge you for the work that they actually do. **You can think of the gas price as the hourly wage for the miner, and the gas cost as their timesheet of work performed**.

- There are a lot of other subtleties to gas, but that should give you the basics! **Gas is the key mechanism that makes the complex computations in Ethereum "safe" for the network to work on**, because any programs that run out of control will only last as long as the money provided by the people who requested they be run. When the money stops, the miners stop working on it. And **the mistakes you make in your program will only affect the people who pay to use it**--the rest of the network can't suffer performance issues due to your error. They will simply get a big payday when the performance issues consume all of your ether! Without this critical technique, the idea of a general-purpose blockchain would have been completely impossible.

Gas

- https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas
- Gas is the way that fees are calculated
- The fees are still paid in ether, though, which is different from gas
- The gas **cost** is the amount of work that goes into something, like the number of hours of labour, whereas the gas **price** is like the hourly wage you pay for the work to be done. The combination of the two determines your total transaction fee.
- If your gas *price* is too low, no one will process your transaction
- If your gas *price* is fine but the gas *cost* of your transaction runs "over budget" the transaction fails but still goes into the blockchain, and you don't get the money back for the work that the labourers did.
- **This makes sure that nothing runs forever, and that people will be careful about the code that they run. It keeps both miners and users safe from bad code!**

Gas

- https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas

- **"Gas" is the name for a special unit used in Ethereum.** It measures how much "work" an action or set of actions takes to perform: for example, to calculate one Keccak256 cryptographic hash it will take 30 gas each time a hash is calculated, plus a cost of 6 more gas for every 256 bits of data being hashed. **Every operation that can be performed by a transaction or contract on the Ethereum platform costs a certain number of gas**, with operations that require more computational resources costing more gas than operations that require few computational resources.

- The reason gas is important is that **it helps to ensure an appropriate fee is being paid** by transactions submitted to the network. By requiring that a transaction pay for each operation it performs (or causes a contract to perform), we ensure that network doesn't become bogged down with performing a lot of intensive work that isn't valuable to anyone. This is a different strategy than the Bitcoin transaction fee, which is based only on the size in kilobytes of a transaction. Since Ethereum allows arbitrarily complex computer code to be run, a short length of code can actually result in a lot of computational work being done. So **it's important to measure the work done directly instead of just choosing a fee based on the length of a transaction or contract**.

- So if gas is basically a transaction fee, how do you pay it? This is where it gets a little tricky. **Although gas is a unit that things can be measured in, there isn't any actual *token* for gas. That is, you can't own 1000 gas.** Instead, gas exists only inside of the Ethereum virtual machine as a count of how much work is being performed. **When it comes to actually *paying* for the gas, the transaction fee is charged as a certain number of *ether***, the built-in token on the Ethereum network and the token with which miners are rewarded for producing blocks.

- This might seem odd at first. **Why don't operations just have a cost measured in ether directly?** The answer is that ether, like bitcoins, have a market price that can change rapidly! But the cost of computation doesn't go up or down just because the price of ether changes. So **it's helpful to separate out the price of computation from the price of the ether token**, so that the cost of an operation doesn't have to be changed every time the market moves.

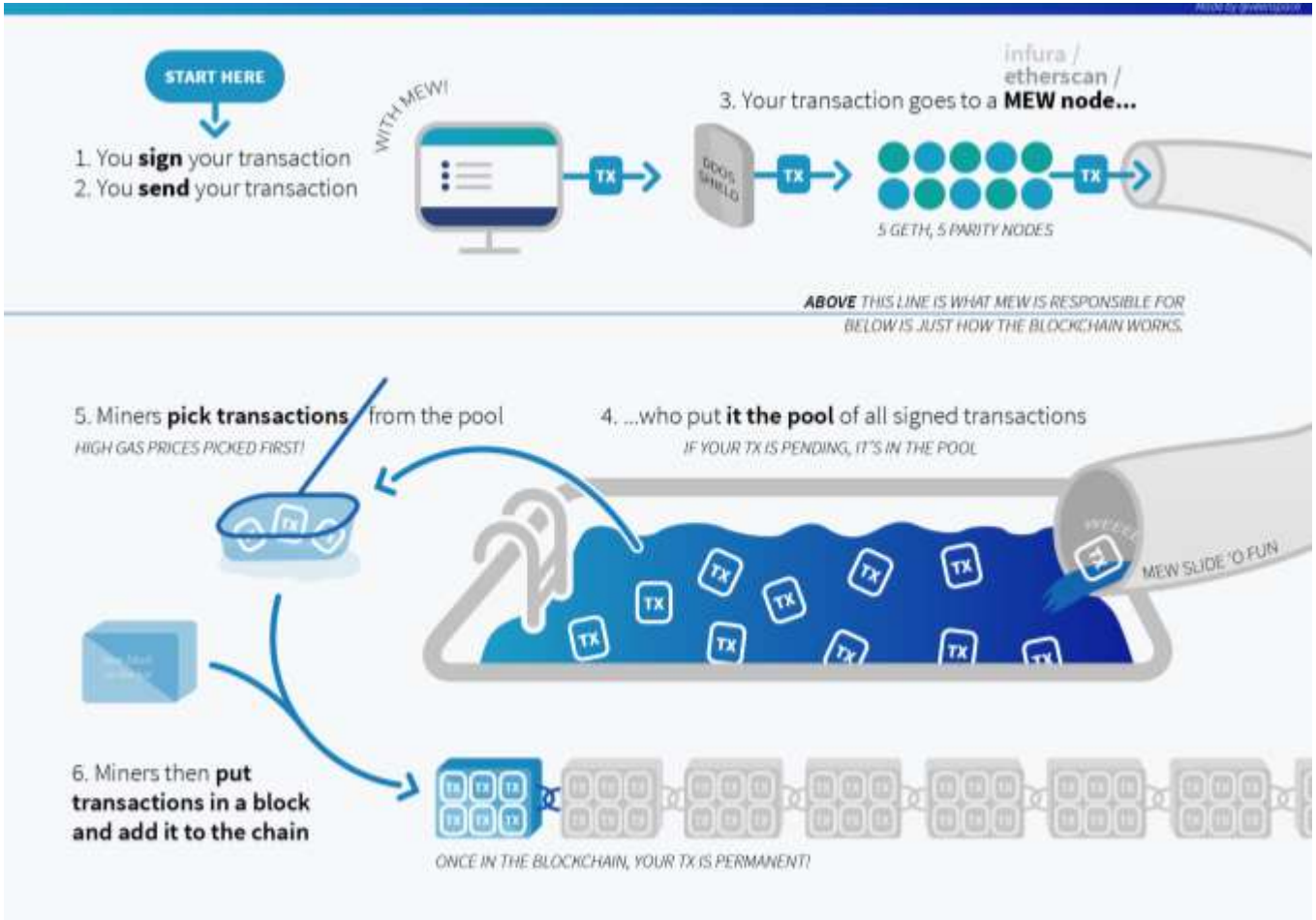
- The terminology here gets a little messy. Operations in the EVM have gas **cost**, but gas itself also has a gas **price** measured in terms of ether. Every transaction specifies the gas *price* it is willing to pay in ether for each unit of gas, allowing the market to decide the relationship between the price of ether and the cost of computing operations (as measured in gas). **It's the combination of the two, total gas used multiplied by gas price paid, that results in the total fee paid by a transaction**.

- As tricky as it is, it's important to understand this distinction, because it results in one of the most confusing things about Ethereum transactions to the initial learner: **there is a difference between your transaction running out of gas and your transaction not having a high enough fee**. If the **gas price** I set in my transaction is too low, no one will even bother to run my transaction in the first place. It will simply not be included in the blockchain by miners. But if I provide an acceptable gas *price*, and then my transaction results in so much computational work that the combined **gas costs** go past the amount I attached as a fee, **that gas counts as "spent"** and I don't get it back. The miner will stop processing the transaction, revert any changes it made, **but still include it in the blockchain as a "failed transaction", collecting the fees for it**. This may seem harsh, but when you realise that the real work for the miner was in performing the computation, you can see that they will never get those resources back either. So **it's only fair that you pay them for the work they did**, even though your badly designed transaction ran out of gas.

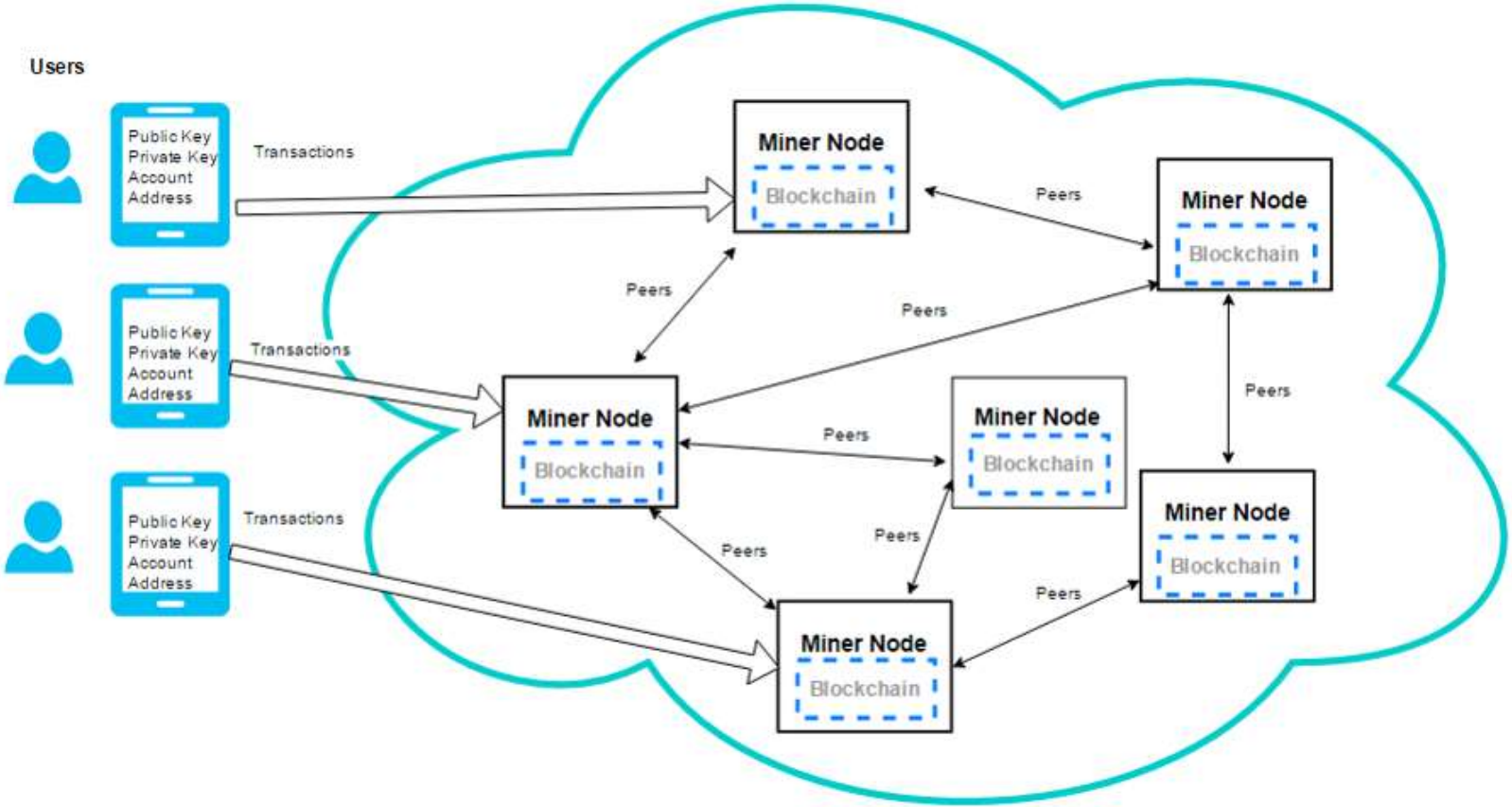
- Providing too big of a fee is also different than providing too much ether. **If you set a very high gas price, you will end up paying lots of ether for only a few operations**, just like setting a super high transaction fee in bitcoin. You'll definitely be prioritised to the front of the line, but your money is gone. If you provided a normal gas price, however, and just **attached more ether than was needed to pay for the gas that your transaction consumed, the excess amount will be refunded back to you**. Miners only charge you for the work that they actually do. **You can think of the gas price as the hourly wage for the miner, and the gas cost as their timesheet of work performed**.

- There are a lot of other subtleties to gas, but that should give you the basics! **Gas is the key mechanism that makes the complex computations in Ethereum "safe" for the network to work on**, because any programs that run out of control will only last as long as the money provided by the people who requested they be run. When the money stops, the miners stop working on it. And **the mistakes you make in your program will only affect the people who pay to use it**--the rest of the network can't suffer performance issues due to your error. They will simply get a big payday when the performance issues consume all of your ether! Without this critical technique, the idea of a general-purpose blockchain would have been completely impossible.

Gas

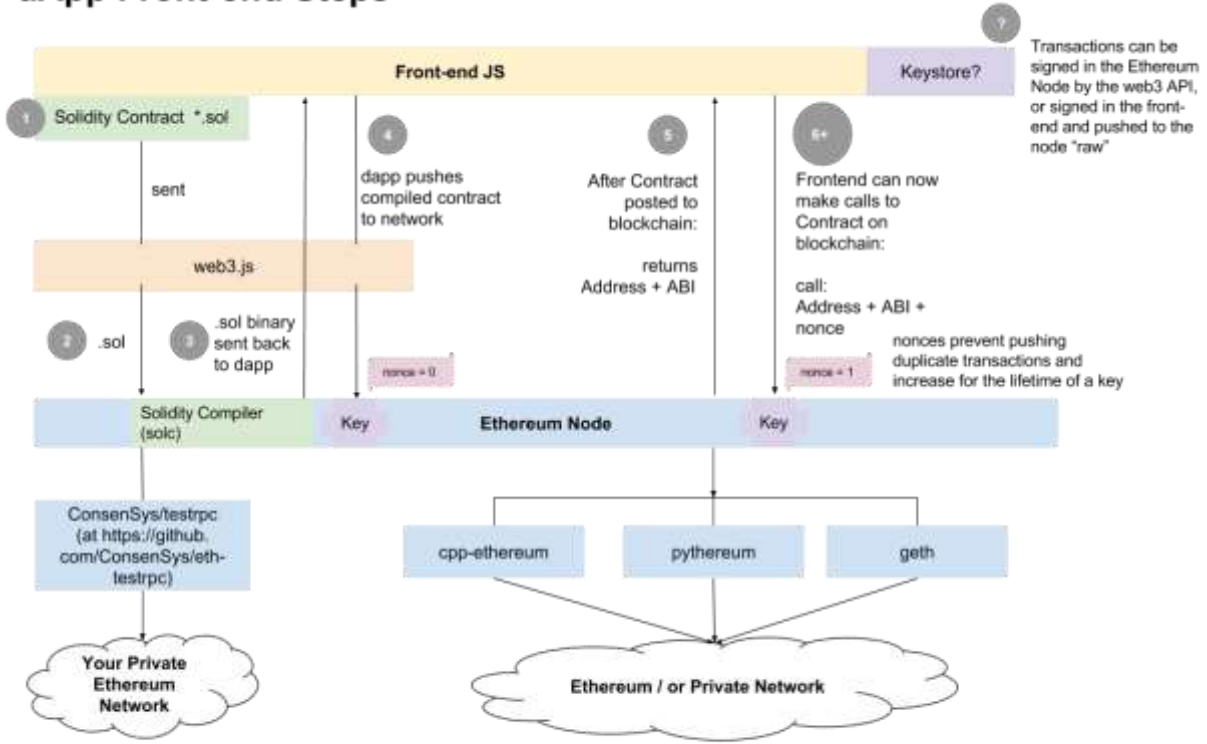

START HERE
1. You **sign** your transaction
2. You **send** your transaction
WITH MEW!
TX
DDOS SHIELD
infura / etherscan /
3. Your transaction goes to a **MEW node...**
5 GETH, 5 PARITY NODES
ABOVE THIS LINE IS WHAT MEW IS RESPONSIBLE FOR
BELOW IS JUST HOW THE BLOCKCHAIN WORKS.
5. Miners **pick transactions** from the pool
HIGH GAS PRICES PICKED FIRST!
4. ...who put **it the pool** of all signed transactions
IF YOUR TX IS PENDING, IT'S IN THE POOL
MEW SLIDE 'O FUN
6. Miners then **put transactions in a block and add it to the chain**
ONCE IN THE BLOCKCHAIN, YOUR TX IS PERMANENT!

Gas


Users
Public Key
Private Key
Account
Address
Transactions
Public Key
Private Key
Account
Address
Transactions
Public Key
Private Key
Account
Address
Transactions
Miner Node
Blockchain
Miner Node
Blockchain
Miner Node
Blockchain
Miner Node
Blockchain
Miner Node
Blockchain
Miner Node
Blockchain
Peers
Peers
Peers
Peers
Peers
Peers
Peers
Peers
Peers

Gas

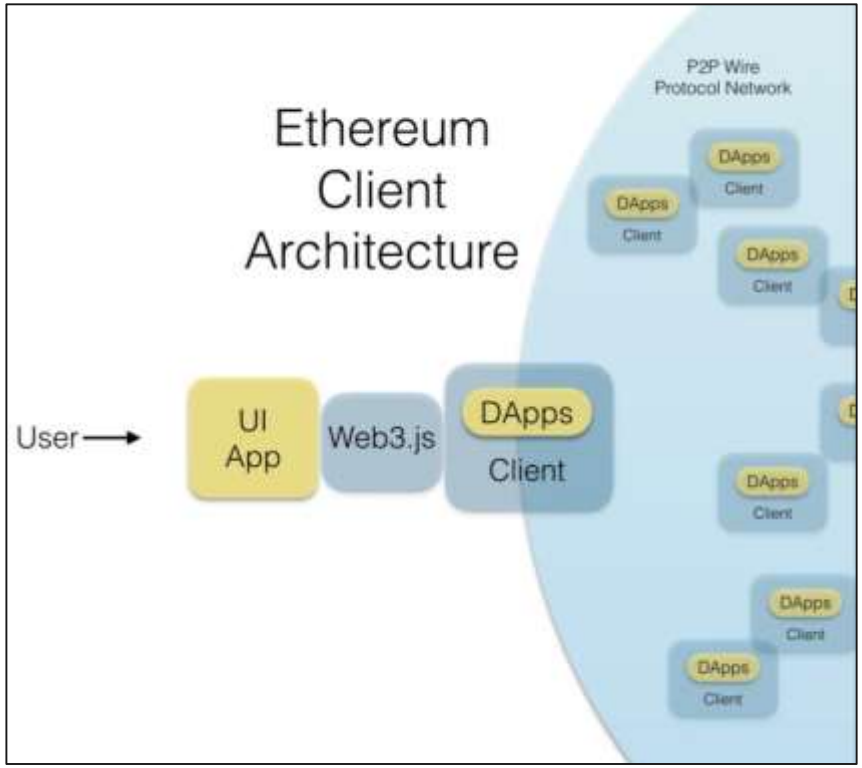
dApp Front-end Steps

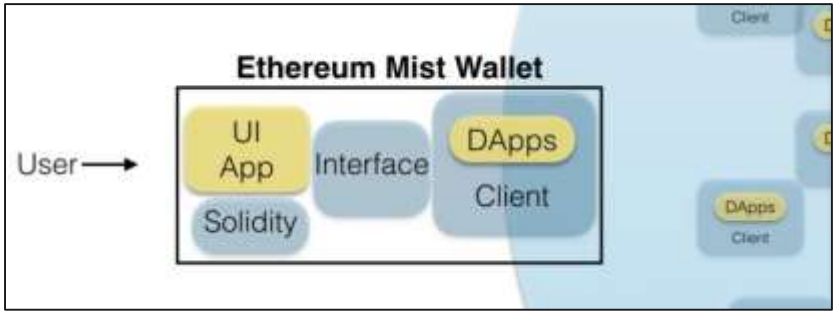


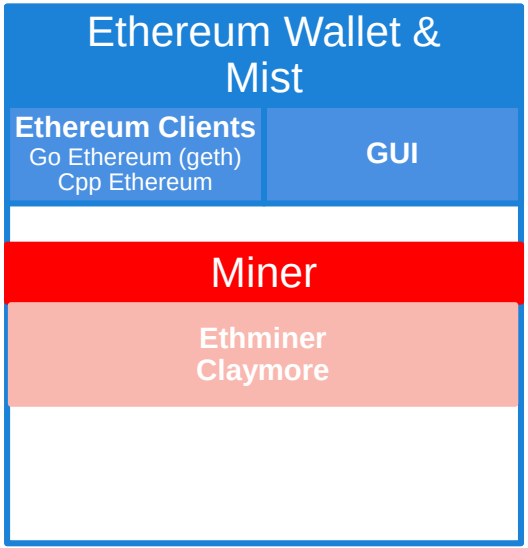
A **Contract Creation Transaction** is shown in steps 1-5 at above.

An **Ether Transfer** or **Function Call Transaction** is assumed in step 6.

Ethereum Client Architecture







http://blockchainers.org/index.php/author/alex/
https://www.coingecko.com/buzz/how-to-mine-ethereum-basic-guide
https://steemit.com/money/@robert11/ethereum-windows-pc-mining-tutorial

Pool Mining

- https://eth.nanopool.org/help
- https://ethermine.org/
- https://ethereumpool.co/
- http://ethpool.org/
- https://ethereum.miningpoolhub.com/

Check Balance

https://eth.nanopool.org/account/0x7fe42d10049f746c25b39e4cfc2d12ab571d351f

Check Ethereum Price

https://www.coingecko.com/en/price_charts/ethereum/usd

Solidity: Contract Oriented Programming Language

Ethereum Solidity

Solidity is designed to compile to code – for the Ethereum Virtual Machine.

https://ethereum.github.io/browser-solidity/
https://ethereumbuilders.gitbooks.io/guide/content/en/solidity_tutorials.html
https://solidity.readthedocs.io/en/develop/

```
contract Coin {
    address minter;
    mapping (address => uint
    function Coin() {
        minter = msg.sender;
    }
    function mint(address ow
        if (msg.sender != mi
        balances[owner] += a
    }
    function send(address re
        if (balances[msg.sen
        balances[msg.sender]
        balances[receiver] +
    }
    function queryBalance(ad
        return balances[addr
    }
}
```

Glossary

No	Terminology	Description
1	Đapp	Ethereum Decentralized Apps, Ethereum based Applications
2	Wallet	
3	geth, eth, pyethapp	Ethereum Clients, written in different languages
4	Satoshi Nakamoto	
5	DAG	https://github.com/ethereum/wiki/wiki/Ethash-DAG Ethash is the PoW system. It requires a ~1GB dataset known as the DAG (see Dagger Hashimoto). This typically takes hours to generate so we tend to memorise it. Clients wishing to store the DAG in a cache should conform to this spec in order to share the cache with other clients: https://ethereum.stackexchange.com/questions/1993/what-actually-is-a-dag
	ETH	Ethereum built I native cryptocurrency, used for paying for smart contracts to run
	Ethereum Virtual Machine, Swarm and Whisper	Decentralised computation, file storage and communication protocols
	Solidity, Serpent, LLL	Smart contract programming languages
	Frontier, Homestead, Metropolis, Serenity	Friendly names for different software releases
	Ethereum Vision	software running on a network of computers that ensures that data and small computer programs called smart contracts are replicated and processed on all the computers on the network, without a central coordinator. The vision is to create an unstoppable censorship

Solidity

Useful Tools - Opportunities for contribution

ethereumjs-testrpc
- Node.js based Ethereum client for testing and development
- Simulates full client behavior
- https://github.com/ethereumjs/testrpc

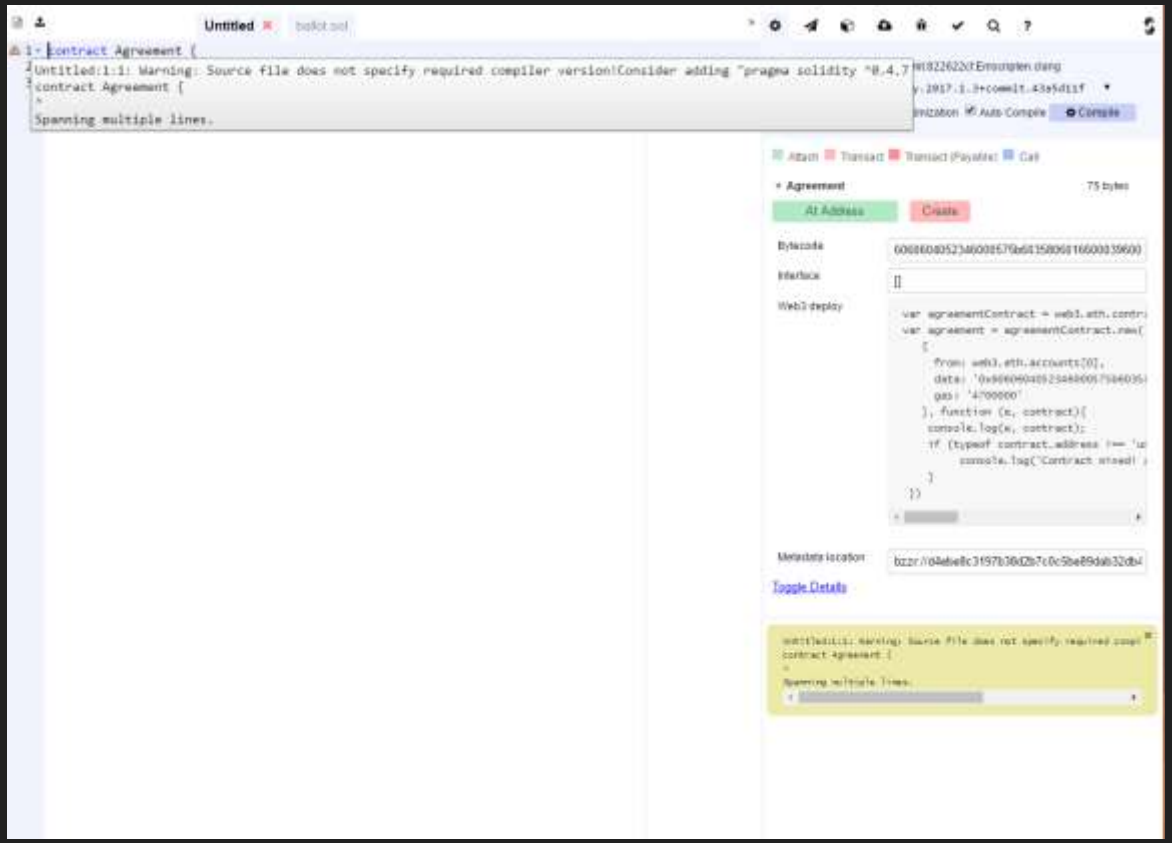
dapple
- developer multitool for managing the growing complexity of interconnected smart contract systems
- https://github.com/nexusdev/dapple

truffle
- A development environment, testing framework and asset pipeline for Ethereum

Browser-Solidity

http://remix.ethereum.org

Shows errors



Creating Contracts

What is the ABI?

Application Binary Interface

Deploying a Contract with Web3

1. Write your contract

```
contract SimpleStorage {
    uint storedData;

    function set(uint x) {
        storedData = x;
    }

    function get() constant returns (uint) {
        return storedData;
    }
}
```

2. Make sure you have the Solidity compiler

```
> eth.getCompilers()
[]
```

No Solidity compiler? Proceed to next step

Download https://github.com/ethereum/solidity/releases

```
> admin.setSolc("C://Solc/solc.exe")
"solc, the solidity compiler commandline interface\r\nVersion: 0.4.7+commit.822622cf.Windows.msvc\r\n"
> eth.getCompilers()
["Solidity"]
```

3. Compile your contract with web3.eth.compile.solidity(source) to get an object with the contract and compiler info

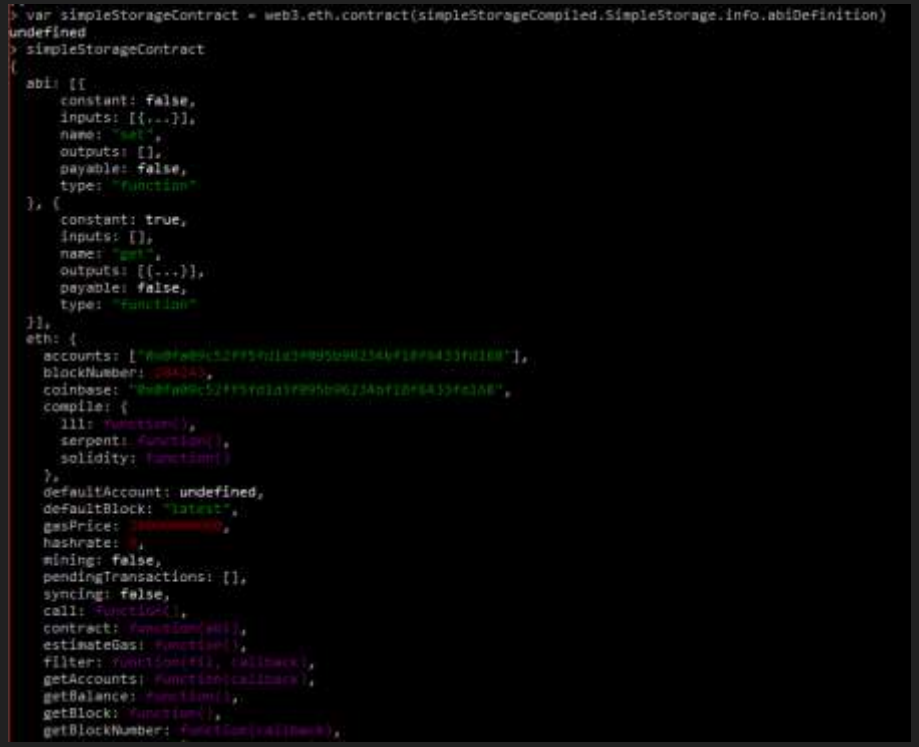
```
> var simpleStorageSource = 'contract SimpleStorage{uint storedData;function set(uint x){storedData=x;}function get()constant returns(uint){return storedData;}}'
undefined
> var simpleStorageCompiled = web3.eth.compile.solidity(simpleStorageSource)

undefined
> simpleStorageCompiled
{
  SimpleStorage: {
    code: "0x6060604052346000575b60938061001760003960006000f300606060405263ffffffff60e060020a6000350416360fe47b18114602c5780636d4ce63c14603b575b6000565b3460005760396004356057
565b005b346000576045606056565b60408051918252519081900360200190f35b60008190555b50565b6000545b905600a165627a7a723058203b41adafb3468285da196fc4ed72c4ee4b4e87799be92ef95d8209bac
f3f766e0029",
    info: {
      abiDefinition: [{...}, {...}],
      compilerOptions: "--combined-json bin,abi,userdoc,devdoc --add-std --optimize",
      compilerVersion: "0.4.7",
      developerDoc: {
        methods: {}
      },
      language: "Solidity",
      languageVersion: "0.4.7",
      source: "contract SimpleStorage{uint storedData;function set(uint x){storedData=x;}function get()constant returns(uint){return storedData;}}",
      userDoc: {
        methods: {}
      }
    }
  }
}
>
>
```

You need the abiDefinition

```
> simpleStorageCompiled.SimpleStorage.info.abiDefinition
[{
    constant: false,
    inputs: [{
        name: "x",
        type: "uint256"
    }],
    name: "set",
    outputs: [],
    payable: false,
    type: "function"
}, {
    constant: true,
    inputs: [],
    name: "get",
    outputs: [{
        name: "",
        type: "uint256"
    }],
    payable: false,
    type: "function"
}]
```

Create a contract object which can be used to instantiate contracts on an address



Deploy the Contract

```
var contractInstance = MyContract.new([contructorParam1] [, contructorParam2], {data: '0x12345...', from: myAccount,
gas: 1000000});
```

Make sure to unlock the account first with personal.unlockAccount(eth.accounts[0])

```
var simplestorage = simpleStorageContract.new(
   {
     from: web3.eth.accounts[0],
     data: '0x6060604052346000575b60938061001760003960000f300606060405263ffffffff60e060020a6000350416360fe47b18114602c578063
     gas: '4700000'
   }, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
         console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' + contract.transactionHash);
    }
 })
```

Contract mined!

```
> I0104 14:52:00.014631 core/blockchain.go:1047] imported 1 blocks,     8 txs (  0.176 Mg) in   5.514ms (31.881 Mg/s). #284446 [2fbd8db9…]
I0104 14:52:32.221565 core/blockchain.go:1047] imported 1 blocks,     5 txs (  0.275 Mg) in  19.582ms (14.055 Mg/s). #284447 [73c95303…]
I0104 14:52:45.056695 core/blockchain.go:1047] imported 1 blocks,     9 txs (  0.199 Mg) in  18.570ms (10.738 Mg/s). #284448 [b83c316c…]
I0104 14:52:59.838297 core/blockchain.go:1047] imported 1 blocks,     1 txs (  0.034 Mg) in   7.551ms ( 4.444 Mg/s). #284449 [1b9a2852…]
I0104 14:53:44.394541 core/blockchain.go:1047] imported 1 blocks,     1 txs (  0.024 Mg) in   5.013ms ( 4.708 Mg/s). #284450 [b206498e…]
I0104 14:54:09.250764 core/blockchain.go:1047] imported 1 blocks,     6 txs (  0.129 Mg) in  23.076ms ( 5.573 Mg/s). #284451 [87b4a8db…]
I0104 14:54:15.131207 core/blockchain.go:1047] imported 1 blocks,     4 txs (  0.161 Mg) in   4.511ms (35.745 Mg/s). #284452 [7b1b8155…]
null [object Object]
Contract mined! address: 0xeb2186b9f667796497f0fc3c824d0bb57a548c25 transactionHash: 0x87e42f5a325e13736342ce0d8e505e332b3ee20a5aa0c6f1080
f8a0f1ab
I0104 14:54:17.734428 core/blockchain.go:1047] imported 1 blocks,     0 txs (  0.000 Mg) in   3.510ms ( 0.000 Mg/s). #284453 [e5f2dcfd…]
I0104 14:54:25.148281 core/blockchain.go:1047] imported 1 blocks,     7 txs (  0.152 Mg) in  20.554ms ( 7.405 Mg/s). #284454 [4391573f…]
I0104 14:55:21.419146 core/blockchain.go:1047] imported 1 blocks,     0 txs (  0.000 Mg) in   5.994ms ( 0.000 Mg/s). #284455 [cdfdc0be…]
I0104 14:55:38.312323 core/blockchain.go:1047] imported 1 blocks,    16 txs (  2.334 Mg) in  47.126ms (49.529 Mg/s). #284456 [e35ec7ad…]
I0104 14:55:56.994674 core/blockchain.go:1047] imported 1 blocks,     7 txs (  0.334 Mg) in  39.603ms ( 8.438 Mg/s). #284457 [f38bf726…]
```

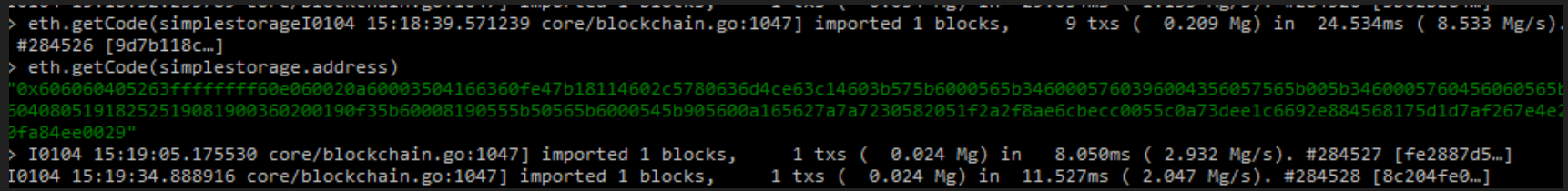
Check transactionHash

```
> var simplestorage=simpleStorageContract.new({from:web3.eth.accounts[0],data:'0x61726e6f6c642077617320686572652121',gas:'4700000'},function(e,contract){console.log(e,contra
ct);if(typeof contract.address!=='undefined'){console.log('Contract mined! address: '+contract.address+' transactionHash: '+contract.transactionHash)}})
null [object Object]
undefined
> simplestorage
{
  abi: [{
      constant: false,
      inputs: [{...}],
      name: "set",
      outputs: [],
      payable: false,
      type: "function"
  }, {
      constant: true,
      inputs: [],
      name: "get",
      outputs: [{...}],
      payable: false,
      type: "function"
  }],
  address: undefined,
  transactionHash: "0xa34f685f8f955ce705bd56480f2cfcf198c471a4b0993569907219e490a58073"
}
```

transactionHash: 0x87e42f5a325e13736342ce0d8e505e332b3ee20a5aa0c6f1080c0d5df8a0f1ab

You can also check that the contract was deployed at the address with: eth.getCode(simplestorage.address)

Contract Address: "0xeb2186b9f667796497f0fc3c824d0bb57a548c25"



1. Write your contract

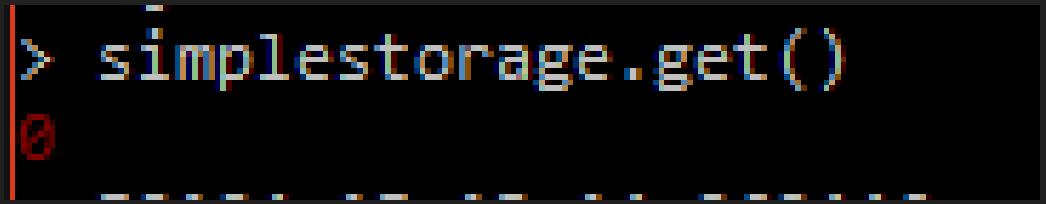
```
contract SimpleStorage {
    uint storedData;

    function set(uint x) {
        storedData = x;
    }

    function get() constant returns (uint) {
        return storedData;
    }
}
```

Retrieving Data from the simplestorage Contract instance

> simplestorage.get ()



Method calls with sendTransaction

contractInstance.method.sendTransaction([parameter], { from: eth.accounts[0])

Change storedData to 20. simplestorage.get() to view change

> simplestorage.set.sendTransaction(20, {from: eth.accounts[0]})

```
> simplestorage.set.sendTransaction(20, {from: eth.accounts[0]}I0104 15:36:56.523825 core/blockchain.go:1047] imported 1 blocks,     3 txs (  0
.078 Mg) in   6.016ms (13.036 Mg/s). #284597 [a33e7f9a…]
> personal.unlockAccount(eth.coinbase)
Unlock account 0x0fa09c52ff5fd1d3f095b96234bf18f6433fd168
Passphrase:
true
> simplestorage.set.sendTransaction(20, {from: eth.accounts[0]})I0104 15:37:15.537136 core/blockchain.go:1047] imported 1 blocks,     3 txs (
0.250 Mg) in   6.016ms (41.477 Mg/s). #284598 [4607f1c3…]

I0104 15:37:16.107231 internal/ethapi/api.go:1047] Tx(0xa507655dff7417c381ff2c222297b9cae262043f62d69529f90a96751c16f0ae) to: 0xeb2186b9f667796
497f0fc3c824d0bb57a548c25
"0xa507655dff7417c381ff2c222297b9cae262043f62d69529f90a96751c16f0ae"
> I0104 15:37:27.489649 core/blockchain.go:1047] imported 1 blocks,     6 txs (  0.129 Mg) in   6.015ms (21.378 Mg/s). #284599 [71adb150…]
I0104 15:37:57.263284 core/blockchain.go:1047] imported 1 blocks,     3 txs (  0.698 Mg) in  12.028ms (58.072 Mg/s). #284600 [96744bbe…]
I0104 15:37:57.427702 core/blockchain.go:1047] imported 1 blocks,     3 txs (  0.698 Mg) in   6.043ms (115.569 Mg/s). #284600 [16b0f16e…]
I0104 15:37:57.665153 core/blockchain.go:1047] imported 1 blocks,     0 txs (  0.000 Mg) in   3.509ms ( 0.000 Mg/s). #284596 [feea89ac…]
I0104 15:38:02.174368 core/blockchain.go:1047] imported 1 blocks,     2 txs (  0.272 Mg) in  13.034ms (20.869 Mg/s). #284601 [83de5b3c…]
> simplestorage.get()
20
> I0104 15:38:58.350399 core/blockchain.go:1047] imported 1 blocks,     4 txs (  0.290 Mg) in  13.558ms (21.408 Mg/s). #284602 [2464a134…]
I0104 15:39:05.887029 core/blockchain.go:1047] imported 1 blocks,    10 txs (  0.213 Mg) in  12.033ms (17.667 Mg/s). #284603 [ba2bdc87…]
I0104 15:39:10.139656 core/blockchain.go:1047] imported 1 blocks,     1 txs (  0.083 Mg) in  13.034ms ( 6.398 Mg/s). #284604 [0618f7ce…]
```

Getting other people to interact with your code

Requires the ABI

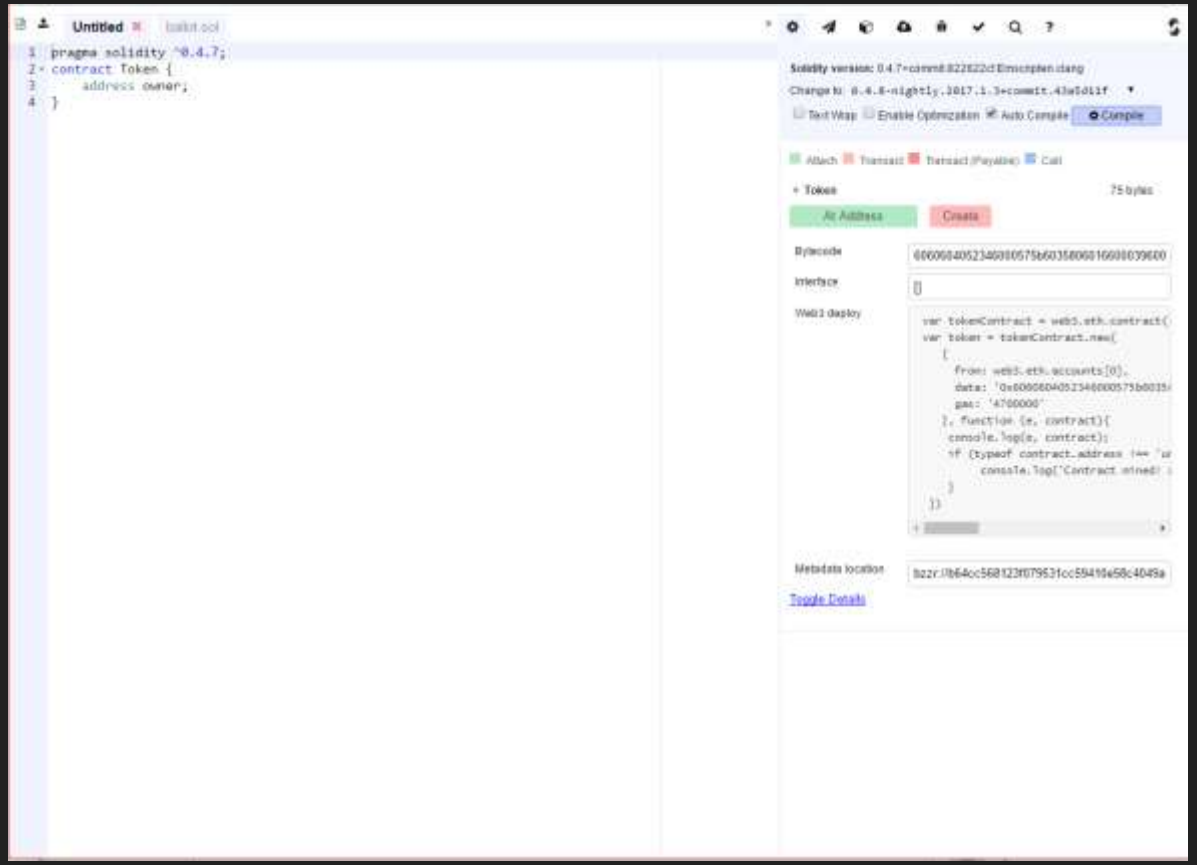
Then they can instantiate a Javascript object which can be used to call the contract from any node on the network

var newInstance = eth.contract(ABI).at(Address);

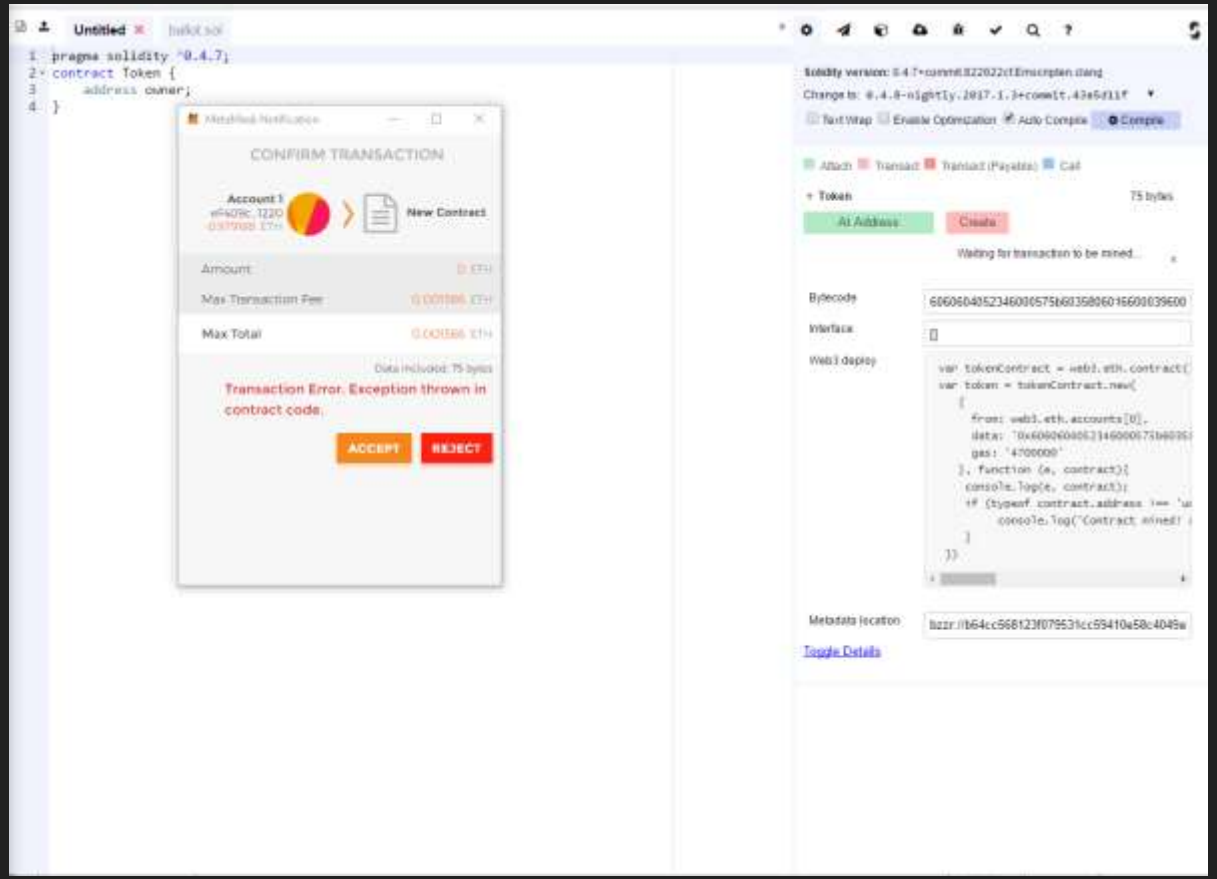
```
> var simplestorage2 = eth.contract(simplestorage.abi).at(simplestorage.address)
undefined
> simplestorage2.get()
30
>
```

Deploying a Contract with Browser-Solidity

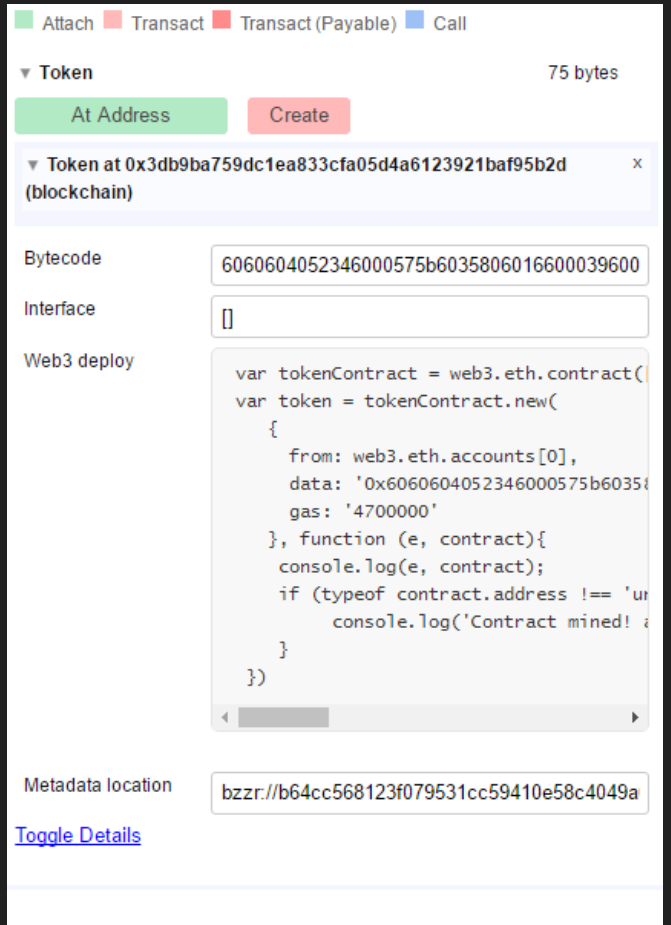
Write the contract in Browser-Solidity



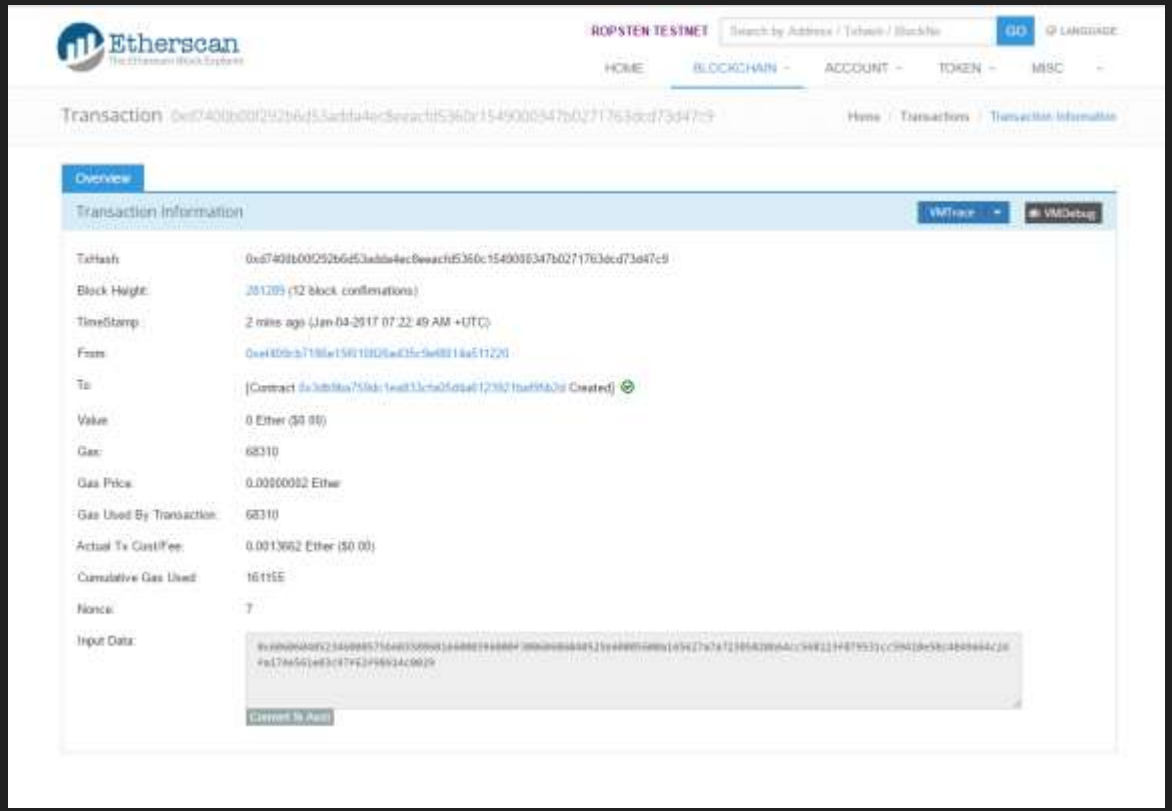
Uses Metamask. Spend Ether to deploy it



Now on blockchain



Etherscan Block Explorer shows the contract



Creating a simple token contract

(Warning: Very simplified Example)

The Token Contract

```
contract Token {
    //Define owner
    address public owner;

    // Define balances
    mapping (address => uint) balances;

    // Constructor function that executes once and sets the owner of the contract
    // with 100 tokens
    function Token() {
        owner = msg.sender;
        balances[owner] = 100;
    }

    // Send _value amount of tokens to address _to
    function transfer(address _to, uint _value) returns (bool success) {
        if (balances[msg.sender] < _value) {
            return false;
        }

        balances[msg.sender] -= _value;
        balances[_to] += _value;
        return true;
    }

    // Shows the token balance of address _user
    function getBalance(address _user) constant returns (uint _balance) {
        return balances[_user];
    }
}
```

Steps to deploy the contract

> var tokenSource = 'contract Token{address public owner;mapping(address=>uint)balances;function Token(){owner=msg.sender;balances[owner]=100;}function transfer(address _to,uint _value)returns(bool success){if(balances[msg.sender]<_value){return false;}balances[msg.sender]-=_value;balances[_to]+=_value;return true;}function getBalance(address _user)constant returns(uint _balance){return balances[_user];}}'

```
> tokenSource
"contract Token{address public owner;mapping(address=>uint)balances;function Token(){owner=m
sg.sender;balances[owner]=100;}function transfer(address _to,uint _value)returns(bool succes
s){if(balances[msg.sender]<_value){return false;}balances[msg.sender]-=_value;balances[_to]+
=_value;return true;}function getBalance(address _user)constant returns(uint _balance){retur
n balances[_user];}}"
```

> var tokenCompiled = web3.eth.compile.solidity(tokenSource)

```
> var tokenCompiled = web3.eth.compile.solidity(tokenSource)
undefined
> tokenCompiled
{
  Token: {
    code: "0x6060604052346100005756b60008054600160a060020a03191633600160a060020a0390811691909
1178083551681526001602052604090206006490555b5b61017a8061004c6000396000f300606060405263ffffffff
f60e060020a6000350416638da5cb5b811461003a578063a9059cbb14610063578063f8b2cb4f14610093575b610
000565b34610000576100476100be565b60408051600160a060020a039092168252519081900360200190f35b346
100005761007f600160a060020a0360043516602435610cd565b60408051911515825251908190036020019 0f35
b34610000576100ac600160a060020a0360043516610 12f565b60408051918252519081900360200190f35b60005
4600160a060020a031681565b600160a060020a033316600090815260016020526040812054829010156100f6575
0600061012956 5b50600160a060020a033381166000908152600160208190526040808320805486900390559285 1
68252919020805483019055 5b92915050565b600160a060020a038116600090815260016020526040902054 5b919
0505600a165627a7a72305820cbf2cb48092ad3283f725493e28f4338b5ac3d2d128339a0f9965cf5c5a8cdbe002
9",
    info: {
      abiDefinition: [{...}, {...}, {...}, {...}],
      compilerOptions: "--combined-json bin,abi,userdoc,devdoc --add-std --optimize",
      compilerVersion: "0.4.7",
      developerDoc: {
        methods: {}
      },
      language: "Solidity",
      languageVersion: "0.4.7",
      source: "contract Token{address public owner;mapping(address=>uint)balances;function T
oken(){owner=msg.sender;balances[owner]=100;}function transfer(address _to,uint _value)retur
ns(bool success){if(balances[msg.sender]<_value){return false;}balances[msg.sender]-=_value;
balances[_to]+=_value;return true;}function getBalance(address _user)constant returns(uint _
balance){return balances[_user];}}",
      userDoc: {
        methods: {}
      }
    }
  }
}
```

> var tokenContract = web3.eth.contract(tokenCompiled.Token.info.abiDefinition)

```
> var tokenContract = web3.eth.contract(tokenCompiled.Token.info.abiDefinition)
undefined
> tokenContract
{
  abi: [{
      constant: true,
      inputs: [],
      name: "owner",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      constant: false,
      inputs: [{...}, {...}],
      name: "transfer",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      constant: true,
      inputs: [{...}],
      name: "getBalance",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      inputs: [],
      payable: false,
      type: "constructor"
  }],
  eth: {
    accounts: ["0x0fa09c52ff5fd1d3f095b96234bf18f6433fd168"],
    blockNumber: 284939,
    coinbase: "0x0fa09c52ff5fd1d3f095b96234bf18f6433fd168",
    compile: {
      lll: function(),
      serpent: function(),
      solidity: function()
```

Create new instance

```
var token = tokenContract.new(
   {
     from: web3.eth.accounts[0],
     data: tokenCompiled.Token.code,
     gas: '4700000'
   }, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
         console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' + contract.transactionHash);
    }
 })
```

```
> toknull [object Object]
Contract mined! address: 0xde90aeeefb5205fce1b824cca4de550ecd4d4881 transactionHash: 0xd99f4
b9e0e39e85cc74de0c26cccbd8951f260372ba54499db24fb9caa490b2c
```

Check the owner

```
> token.owner()
"0x0fa09c52ff5fd1d3f095b96234bf18f6433fd168"
> eth.coinbase
"0x0fa09c52ff5fd1d3f095b96234bf18f6433fd168"
>
```

GetBalance

```
> token.getBalance(eth.coinbase)
100
>
```

Transfer tokens to another address

```
token.transfer.sendTransaction(eth.accounts[1], 20, {from: eth.accounts[0]})
```

```
> token.transfer.sendTransaction(eth.accounts[1], 20, {from: eth.accounts[0]})
"0x4d029490ec6a92df9a1756bbcf095c966544ab15b59fac94eb57bb53ac989e51"
> token.getBalance(eth.coinbase)
80
> token.getBalance(eth.accounts[1])
20
>
```

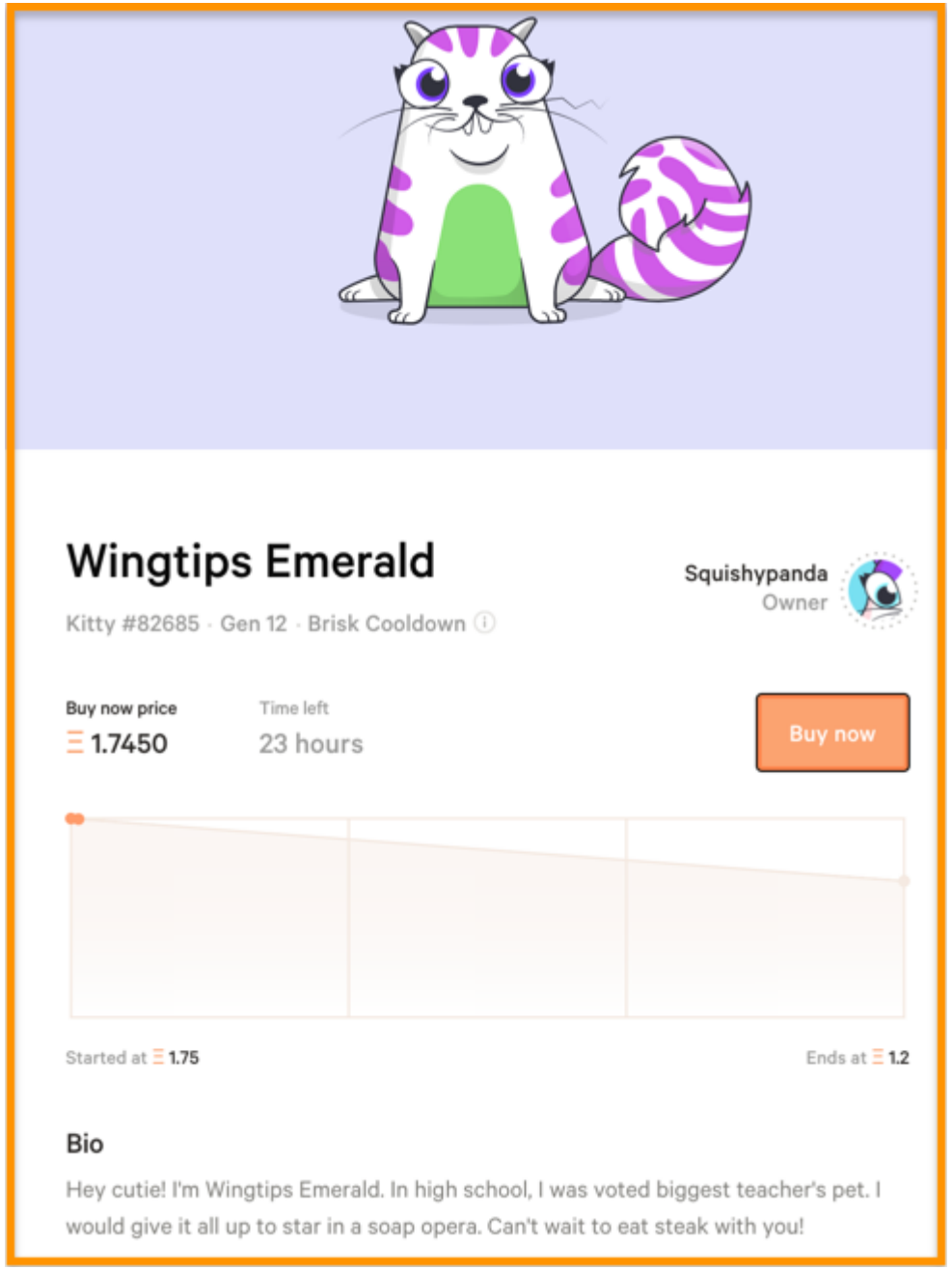
ERC20 Specification

Token Methods:

- totalSupply
- balanceOf
- transfer
- transferFrom
- approve
- allowance

Cryptokitties

- Genetic algorithm
  - 5000 initial kitties
  - others are result of smart contract execution
- Smart contract https://etherscan.io/address/0x06012c8cf97bead5deae237070f9587f8e7a266d#code ~2000 lines
  - KittiBase, KittyOwnership, KittyBreeding, KittyAuctions, KittyMinting
- Stats
  - Kitties: 1198056
  - Owners: 83318
  - Sales: 466533 / 52717.9727 ETH



Initial Coin Offering (ICO)

Top 10 ICOs 2018

ICO Name	Amount Raised	Description	Start Date	End Date
EOS	$4,197,956,136	EOS raised the most amount of funds ever in its ICO. The reason for this is mostly likely the combination of Dan Larimer the founder of BitShars and Steemit as CTO as well as being the most ambition blockchain project for decentralised apps ever attempted.	Jun 26, 2017	Jun 01, 2018
Telegram	$1,700,000,000	Telegram's ICO aimed to use the funds generated from the token sale to expand its functionality via Blockchain technology beyond simple messaging services. It is hoped that this extra functionality to increase telegrams user base beyond its current 200 million user number.	Oct 15, 2017	Feb 15, 2018
Ruby-X	$1,196,000,000	Ruby- X will be releasing its final exchange in 2 phases. Phase 1 will be a conventional exchange and in phase 2 the team promise will allow users to exchange, buy and sell any tangible or intangible asset.	Aug 10, 2018	Sep 17, 2018
Petro	$735,000,000	The Venezuelan government have attempted to circumnavigate economic sanctions by creating their own cryptocurrency and using it as a payment method for oil. This is the first officially government backed cryptocurrency.	Feb 20, 2018	Mar 19, 2018
TaTaTu	$575,000,000	The TaTaTu platform aims to revolutionise social media and entertainment via their Blockchain platform. Consumers of content will be paid for their viewership as well as for supplying it. Further features include using the accumulated TaTaTu token to pay for/unlock paid content: movies, games etc.	Jun 11, 2018	Jun 30, 2018
Dragon	$420,000,000	Dragon Coin an ERC 20 token will be used to grant access at Casinos powered by Dragon's Blockchain. All the casinos facilities will be managed by Dragon parter junkets.	Feb 15, 2018	Mar 15, 2018
Huobi token	$300,000,000	The Huobi Token (HT) rewards exchange users for their loyalty with lowered transaction fees while also carrying its own value in tradable pairs against popular currencies.	Jan 24, 2018	Feb 28, 2018

***

Crypto zombies

https://cryptozombies.io/