Overview : Bitcoin and existing concept

Bitcoin can be thought of as a **state transition system**.

Lets us consider a state transition function APPLY, so that it is something like this:

APPLY(S, Tx) -> S'

Where,
S (state): is the collection of all coins (technically, "unspent transaction outputs" or UTXO) that have been mined and not yet spent. Each UTXO value and address (owner) associated with it. Set to input UTXO.

Tx (transaction) : a transaction contains one or more input where, each input is referencing to existing UTXO and signature, produced by private keys of owner associated with those UTXOs.

S' (new state) : state with all input UTXOs removed and output UTXOs added. Set of output UTXO.

Now, let's defined APPLY

1. If referenced UTXOs, present in Tx does not belong to set S, then error
2. If provided signature does not match to the owners in set S, then error
3. If sum of values of input UTXO < sum of values of output UTXO, then error
4. Return S' with all input UTXO removed and output UTXO added.

So from point no 1, we can prevent a user for spending money that does not exist. From point no 2, we are preventing user from spending someone else's money. 3rd point enforces changing ownership of value. And the last point tells a new state have been achieved.

Centralize to Decentralize

If we had to implement system something like this, we could have done it using centralized manner, but as we know Bitcoin is decentralized, and in a decentralized network, control does not exists in the hand of single entity.

But how would be achieve decentralization, by adding **Consensys System** along with **State Transition System**

Consensys system is required, in order to ensure everyone in the network agrees on order of transaction. In such system, no single entity will take decision, rather the network(or special

nodes, called miners) will come to an agreement.
Lets try to understand how we can put Consensys on top of State transition system that we talked earlier.
Suppose, **A** sent $100 to **B** and the transaction gets added to the ledger. and then again, he tries to sent $100 to himself. **A** broadcasted this message to the network. But the miner will not consider the transaction this time as, when calculating **APPLY(S,Tx)** miners will notice that, UTXO consumed in Tx is not present in S and hence discard. However, A can create fork of blockchain, by mining another version of block (say, 1000 to same parent block 999) but with different transaction.
By the time he added this block, in the actual chain, few more block would have been added till time (say 1000,1001,1002). And by the rule of longest chain, the network will accept the actual chain, not his chain.

Merkle Tree

One of the scalability feature of Blockchain is that blocks are stored in multilevel data structure. The hash of the block is actually has of block header, which contains, timestamp, nounce, previous hash and root hash of data st. called, merkle tree storing all transactions in a block. A Merkle tree is a type of binary tree, composed of a set of nodes with a large number of leaf nodes at the bottom of the tree containing the underlying data, a set of intermediate nodes where each node is the hash of its two children, and finally a single root node, also formed from the hash of its two children, representing the "top" of the tree.

The Merkle tree is useful because it allows users to verify a specific transaction without downloading the whole blockchain

The reason why this works is that hashes propagate upward: if a malicious user attempts to swap in a fake transaction into the bottom of a Merkle tree, this change will cause a change in the node above, and then a change in the node above that, finally changing the root of the tree and therefore the hash of the block

Account in Ethereum

As we were discussing about state, we will see how these states have been built up. State is made up of objects called, accounts. There are two types of accounts in ethereum, EOA and CA. Externally owned accounts and Contract accounts. Lets us first understand what are they and then we will discuss why there is need to two types of accounts in Ethereum.

Accounts represent identities of external agents (e.g., human personas, mining nodes or automated agents). Accounts use public key cryptography to sign transaction so that the EVM can securely validate the identity of a transaction sender.

Externally owned accounts : EOA are accounts, which are controlled by private key. And since a human can hold private key, EOA is controlled by human. They have balance.

Contract Account : CA are accounts that are controlled by code. This code is also called contracts in ethereum (we will discuss about contracts in details later). As of now just think that they are codes which are wirtthen on Ethereum blockchain. Since code can not hold/handle private key, CA are not controlled by private key. They have balance as well as storage to store code. Every time a contract account receives a transaction, its code is executed as instructed by the input parameters sent as part of the transaction.

Why two types of accounts?

Because it is different from bitcoin and altcoin :) You might have understood so far that, state management is the key concept of blockchain. So you dont have CA, then you only have limited feature of sending and receiving cryptos like bitcoin. CA helps to track state of code. Hope its clear now.

Ether and Gas

Lets us understand first, what Gas is? then we will talk about why Gas is actually needed in Ethereum blockchain?

Its a cost of network utilization. Meaning, it is a special unit used in Ethereum which tells how much work is required to perform an action or set of action. Let me tell you in detail a bit. So where ever you write some code (called smart contract) in Ethereum blockchain, it gets converted into OPCODE (set of instruction that is to be executed on machine). Every OPCODE is associated with one number, for example, for ADD gas 3, MUL it is 5 and so on and so forth.

So when ever you try to execute this piece of code, you basically trigger a transaction (signed by your private key) and then that transaction gets executed by all the miners in the node (as we talked about how mining works)

Keeping this in mind, let us understand why they have introduced the concept of Gas in Ethereum.

Gas - Why?

As we know when these piece of code gets triggered, miners will spend their time and resource to execute so in return they should get back something right? Hence Gas, transaction cost given by caller to the miners.
It also helps to secure the network by restricting network flooded with unwanted transactions.

Moreover, this transaction fee is different from transaction fee in Bitcoin network. in Ethereum the fees are paid in gas and calculated based on contract code execution complexity, in bitcoin the're based solely on transaction size. I will not go talk about Bitcoin Tx. size as its off topic.

Still we have left with one question, why there is need of another unit for paying transaction fee ? Why not use Ether itself?

Gas prices of all the operation are hard-codes in Ethereum protocol and if the code listed them in ether, then we would have to update the code every time Ether’s value fluctuate to keep the price of computing normal and keep the system usable.
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.

Gas Terms

Gas Limit: It is the maximum amount of Gas, we are willing to spend on a Transaction.
Gas Cost: Is the static value that is associated with OPCODE.
Gas Price: While gas is fixed per operation, the amount a user pays per gas—*gas price*—is dynamic and dictated by market conditions. Gas price is a value representing how much Ether the user is willing to pay per gas.

Gas Fee: Effective amount of transaction fee that user pays. It is calculated by multiplying Gas used to the Gas price. This fee is paid to miners.

Note:
We, as a user, can modify the amount of gas we want to spend on a transaction and reduce it, but if the transaction runs out of gas during execution, we lose the gas we sent in. It's been spent and the transaction is rejected. On the other hand, if we provide more gas than is needed, the rest is refunded to us. Hence, it's always better to send more gas than you might need to execute a transaction.

EVM & Solc

EVM is Ethereum virtual machine, a stack based virtual machine that executes bytecodes which is generated by compilers, called solc. EVM has two components, **stack** and **store**. When the code gets compiled, bytecode is generated and EVM starts reading each instruction and executed it one by one and add the result into the stack.

If a any instruction comes for storage, like SSTORE, it store the value in into the memory. In this way after executing all the instructions, stack gets empty and machine stops execution.

Example,
If you compile this code

```
pragma solidity ^0.4.11;
contract C {
   uint256 a;
   function C() {
     a = 1;
   }
}
```

You will get this bytecode :

```
60606040523415600e57600080fd5b5b60016000819055505b5b60368060266000396000f3006
```

```
0606040525b600080fd00a165627a7a72305820af3193f6fd31031a0e0d2de1ad2c27352b1ce08
1b4f3c92b5650ca4dd542bb770029
```

`6001600081905550` is binary for `a=1`

Which will translate into the following instructions

```
PUSH 1
PUSH 0
DUP2
SWAP1
SSTORE
POP
```

So you have a source code, solc compiles this code (.sol file) into bytecode which gets executed on the EVM. You can install solc from here and compile files like this,

```
solc --bin test.sol
```

`--bin` option will output the binary/bytecode of the solidity code. You can also use `--asm` option to see the opcodes or assemblies. More options can be find using option `--help`.

You can also use node js library called, **solc-js** for your node project to compiler the solidity code. Look for npm package for the same from here.

References

EVM Instructions : https://gist.github.com/hayeah/bd37a123c02fecffbe629bf98a8391df
Detailed discussion on EVM : https://blog.qtum.org/diving-into-the-ethereum-vm-6e8d5d2f3c30
EVM : https://github.com/CoinCulture/evm-tools/blob/master/analysis/guide.md

Yellow Paper : https://ethereum.github.io/yellowpaper/paper.pdf
White Paper : https://github.com/ethereum/wiki/wiki/White-Paper
Gas concept : https://bitfalls.com/2017/12/05/ethereum-gas-and-transaction-fees-explained/
Gas concept : https://hackernoon.com/ether-purchase-power-df40a38c5a2f