ETH Debug Notes

EVM (ETH Virtual Machine)

- contains two types of accounts: External (Humans) and Contract (Code)
  - Treated equally by the EVM
  - Both have a balance, address, etc

Transactions

- Message sent from one account to another (**Reference**: look at 'Message Calls, these are closely related')
- Contains Binary Data/Ether
- If target is 0 acc, the transaction creates new contract
- GAS: depleted during the execution of a contract (**Reference:** see 'Halting Problem' for why we need GAS in the ETH network)
  - if gas is totally depleted, the transaction is reversed, and state is returned to before the contract even began execution
  - triggers 'out-of-gas' exception

Storage, Memory, and Stack

- Storage: a key-Value store mapping **256 bit** words to 256 bit words.
  - cannot enumerate storage from within contract
  - costly to read, write, or modify storage
  - contract cannot read or write to any storage apart from it's own storage

EVM

- Stack Machine where all computations are performed
- **MAX_SIZE = 1024 elements (each element is a 256bit word)**

```
Aside: since this may contain nuances, here is the official paragraph describing the
EVM:


"The EVM is not a register machine but a stack machine, so all computations are
performed on an area called the stack. It has a maximum size of 1024 elements and
contains words of 256 bits. Access to the stack is limited to the top end in the
following way: It is possible to copy one of the topmost 16 elements to the top of the
stack or swap the topmost element with one of the 16 elements below it. All other
operations take the topmost two (or one, or more, depending on the operation) elements
from the stack and push the result onto the stack. Of course it is possible to move
stack elements to storage or memory, but it is not possible to just access arbitrary
elements deeper in the stack without first removing the top of the stack."
```

ACCESS

- Copy one of 16 topmost elements
- swap topmost element with one of 16 elements below it
- all other operations take topmost two elements from the stack, and pushes the result of the operation back onto the stack
- can move stack elements to storage or memory, but cannot arbitrarily access all stack elements (only the topmost 16, in accordance with the rules above)

Instruction Set

- instruction set kept minimal in order to avoid consensus issues
- All instructions operate on 256 bit words
- usual arithmetic operations present (adding, binary &, OR's, etc)
- conditional and unconditional jumps possible
- contracts can access props like number + timestamp

Message Calls (Kind of like the EVM IPC Mechanism, but for contracts)

- send Ether to other contracts or non-contract accounts
- basically like a transaction, except more low-level. Transactions are built on top of Message Calls
    - Both transactions and message calls have a source, target, data payload, Ether, gas, and return data
- Every Transaction consists of a top-level Message Call
    - can create more message calls
- contract(IE: the code) decides how much of it's remaining gas should be sent with inner-message call
- 'Out-of-Gas' exception signaled by an error value put onto the stack
    - only gas sent with call used up
- **in solidity, errors bubble up**
- calls limited to a depth of 1024 (since that is the maximum size of the stack, that only makes sense)

Delegatecalls / Callcodes and Libraries

- Delegatecalls are identical to message calls apart from:
  - code at target executed in context of the calling contract
  - `msg.sender` and `msg.value` do not change their values
- **Because of the above, this means contracts can dynamically load code from different addresses at runtime** (IE: libraries)
  - Storage, current address, and balance still refer to the calling (original) contract
    - only the code is taken from the called address, nothing else.
    - makes it possible to implement libraries in solidity

Logs (Event Implementation)

- Possible to store data in an indexed data-structure that maps to the block level
- contracts cannot access log data after its been created. Accessed from outside the blockchain

```
Aside: Bloom Filters
Probabilistic data structure used to test whether an element is part of a set. False
positives are possible, but false negatives are NOT.

IE: If the data struct says something is apart of a set, it might not actually be apart
of that set.However, if the data structure says that something is NOT part of a set, it
is definitely NOT part of that set.

Look at Wikipedia page for more information,
```

- Bloom Filters make it possible to search data in an efficient and cryptographically secure way
  - in this way, light clients can still access info on the blockchain

Self Destruct (Removing Contracts from Blockchain)

- When contract performs this operation, it is removed from the blockchain. ETH is sent to target, and code removed from state. (Implemented after the DAO contract fiasco, and is the reason ETC exists.)

```
History Aside
This self-destruct operation was implemented after the DAO contract fiasco, and is the
reason ETC exists.

ETC fans are staunch supporters of a blockchain being totally and irreversably
permanent. The DAO resulted in the loss of millions of ETH because of a faulty contract
allowing hackers to take advantage of it, and so the fork was done to allow for this
self destruct behavior.
```

- Even when a contract does not contain a call to `selfdestruct` it can still perform that operation using `delegatecall` or `callcode`
- The pruning of old contracts may or may not be implemented by Ethereum clients. Additionally, archive nodes could choose to keep the contract storage

and code indefinitely.

- Currently **external accounts** (human accounts) cannot be removed from the state. (So our precious money+wallet is safe from skynet)