Ethereum

Consensus & Execution Layers

Prof. James Won-Ki Hong

Distributed Processing & Network Management Lab.
Dept. of Computer Science and Engineering
POSTECH

http://dpnm.postech.ac.kr
jwkhong@postech.ac.kr

Table of Contents

Consensus Layer

- **Why do we need a consensus mechanism?**
  - To prevent certain nodes from arbitrarily manipulating the blockchain
  - To verify the validity of blocks and transactions
  - To make a rule to select one of blockchains in the network
    - When there are two or more valid blockchains

- **Proof of Work (PoW)**
  - The way to compensate the miner that most quickly finds the solution of the complex calculation in order to connect the block to the blockchain
  - Continue to change **Nonce**, iteratively calculating the hash of block header
    - Until the hash value is less than or equal to the target value

Consensus Layer

- **Chain Fork**
  - A way to upgrade system (software upgrade) of the blockchain in Ethereum
  - Also a way to create a new blockchain
  - Types of fork
    - **Soft Fork**:

      Users do not need but miners must upgrade in order to maintain validity of the new blockchain
    - **Hard Fork**:

      All miners and users must upgrade because of incompatibility between old and new blockchains
    - **Regular Fork**:

      A temporary collision that occurs when two blocks are created at the same time

It is important that there must be a single chain in the network

Consensus Layer

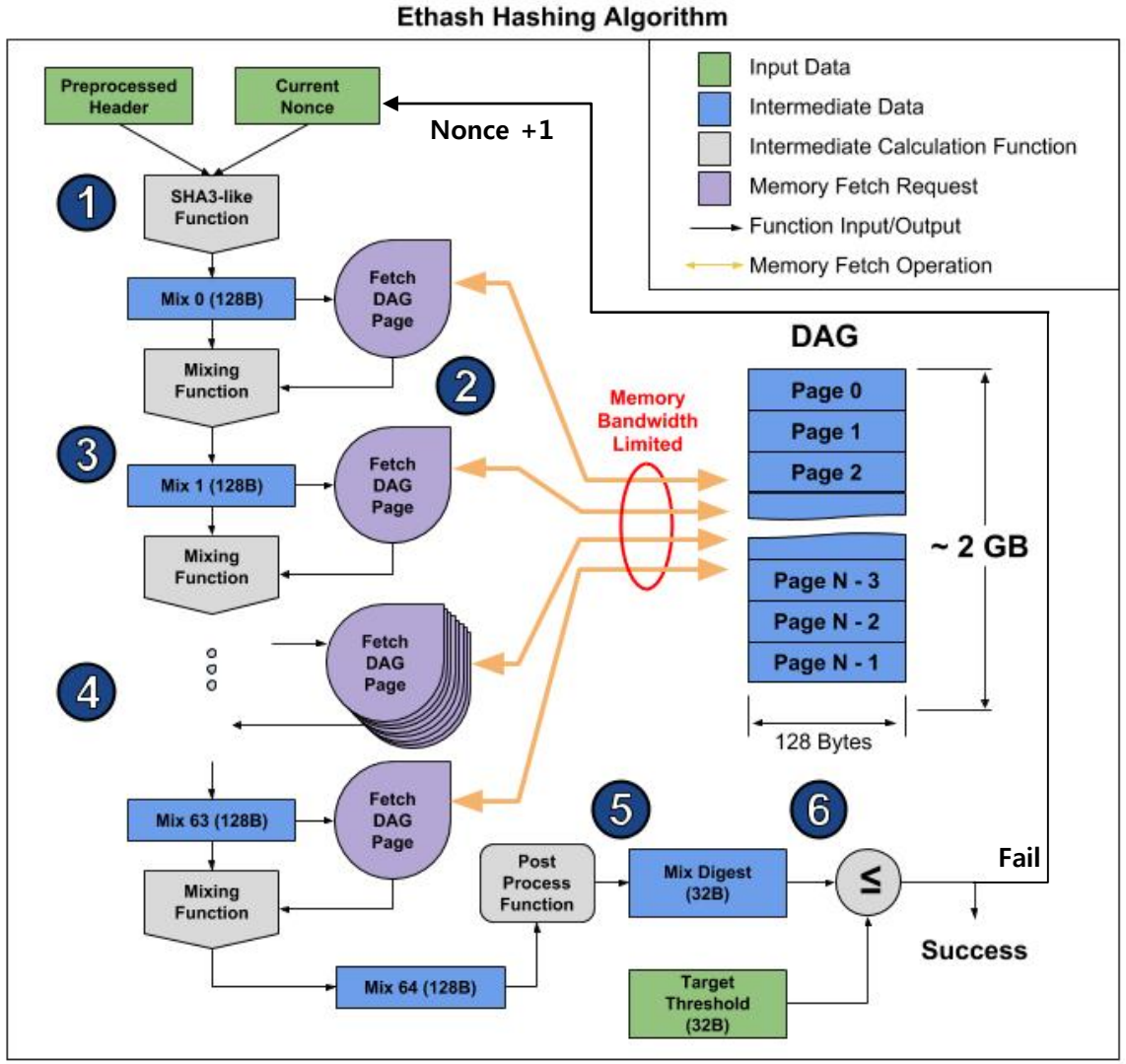
- **Ethash: Proof of Work in Ethereum**
  - Memory-based PoW consensus engine in Ethereum
    - A way to calculate a hash with nonce and a certain amount of data in **memory**
  - Use Directed acyclic graph (DAG) file for memory calculation
    - Create several GBs of Directed Acyclic Graph (DAG) for hash every 30,000 blocks
  - Discourage mining that uses ASIC chip
    - It is designed to make the proof of work function resistant to specialized hardware (ASIC)
  - Designed to generate a new block every 10 to 15 seconds

Consensus Layer

- **DAG (Directed Acyclic Graph) file**
  - Cache dataset of about 2GB generated by seed hash
    1. Extract seed hash by scanning the block's number
    2. Compute 16MB of pseudo random cache through the seed hash
    3. Create Full Dataset of 1GB or more through the cache
       - Full Dataset == DAG file
  - Recreate the entire DAG file at 30,000 block cycles
    - The size of DAG file increases linearly
      - 2018.09.07 ➔ About 2.6GB
    - 30,000 block units are called **Epoch**

Consensus Layer

- **Ethereum Mining: Find nonce and mixDigest**



https://www.vijaypradeep.com/blog/2017-04-28-ethereums-memory-hardness-explained/

(m, n) = PoW(H~~n~~, Hn, d)

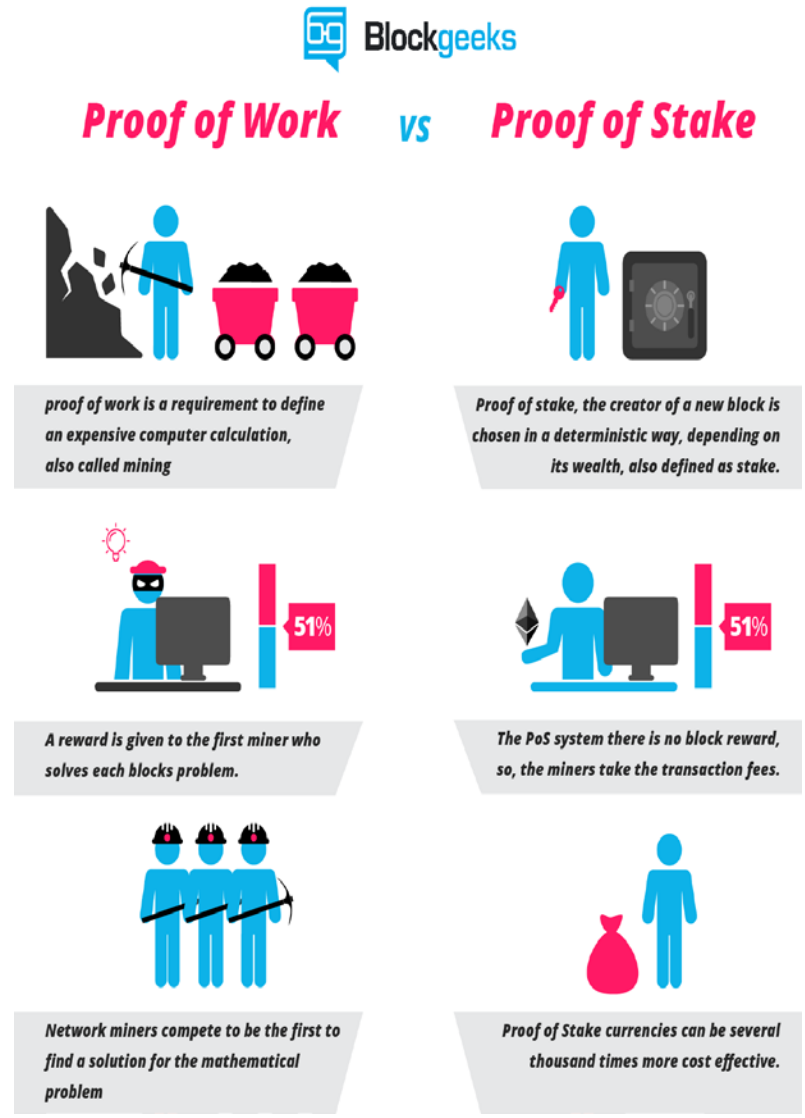
- m: mixDigest
- n: nonce
- H~~n~~: new block's header without nonce and mixDigest
- Hn: nonce of header
- d: dataset (DAG file)

Consensus Layer

- **Difficulty and Target**
  - Continue to be adjusted instead of fixed value
  - Calculated based on timestamp
  - Timestamp
    - Contained in the header of every block and used to validate the block
  - Relationship between Target and Difficulty
    - Target = 2^256 / Difficulty

Consensus Layer

- **Proof of Stake (PoS) (1)**
  - Problems of PoW
    - A lot of resources required and wasted
    - Maintaining block generation period
  - What is PoS?
    - **Validator** gets the **right to generate a block with a probability proportional to the stake**
    - Validator obtains compensation
      - After generating a block and attaching it to the desired chain
  - Advantages of PoS
    - Reduce energy costs
    - Strengthen security
      - Reduced risk of centralization
      - Finality
      - Penalty



https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/

Proof of Stake (PoS) (2)

- Basic concept of PoS in Ethereum
  - If the account deposits Ether to **smart contract**, it can be a **validator**
  - If the account becomes a validator, it can vote for a chain new block will be connected to
  - If the validator cannot follow the rule, it suffers disadvantage
    1. Vote on the wrong block
    2. Do not vote
- PoW → PoS: Finality
  - There is the checkpoint to finalize a blockchain every 100 blocks
    - **To be finalized, 2/3 or more votes must be obtained**
  - Once a blockchain is finalized, it cannot be turned back

Execution Layer

Smart Contract (1)

- A computer protocol intended to digitally facilitate, verify, or enforce a contract
- Allow the performance of credible transactions without third parties
  - These transactions are trackable and irreversible

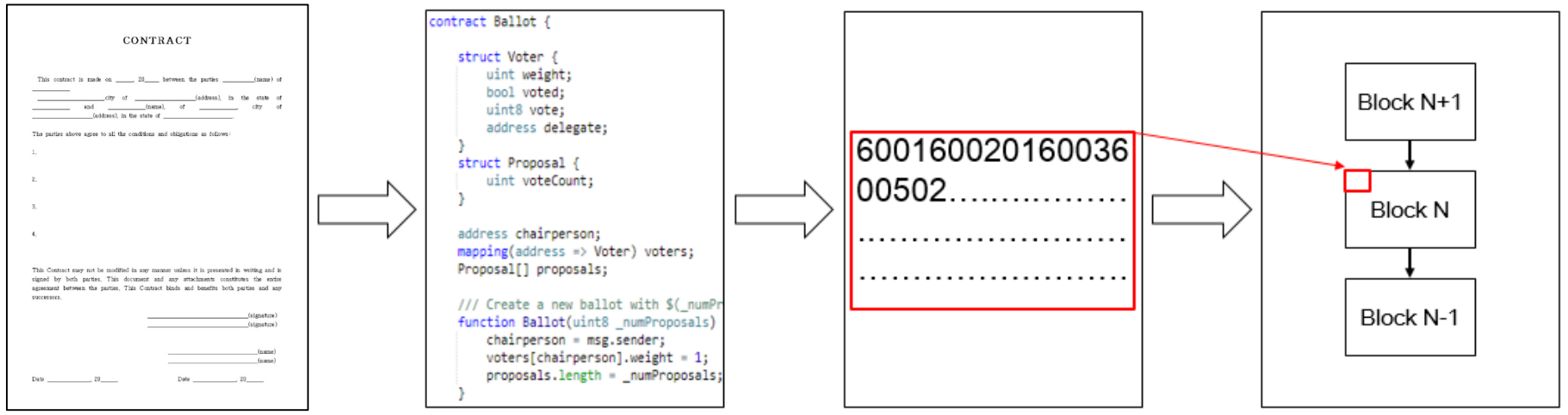
- **The first:** 1994 by Nick Szabo
  - "**Smart contract** is a protocol that enforces compliance with highly developed contracts in an unreliable computer Internet environment"
    - To provide better security than existing contract-related laws
    - To offer lower processing costs than traditional contract

- **Script** in Bitcoin
  - A simple function to verify the validity of a transaction by checking ownership of UTXO in the input value
    - Script (correct) → Transaction (Normal): The concept of a contract
  - limitation: 1. Can't use loop instruction, 2. Can only manage the balance

Execution Layer

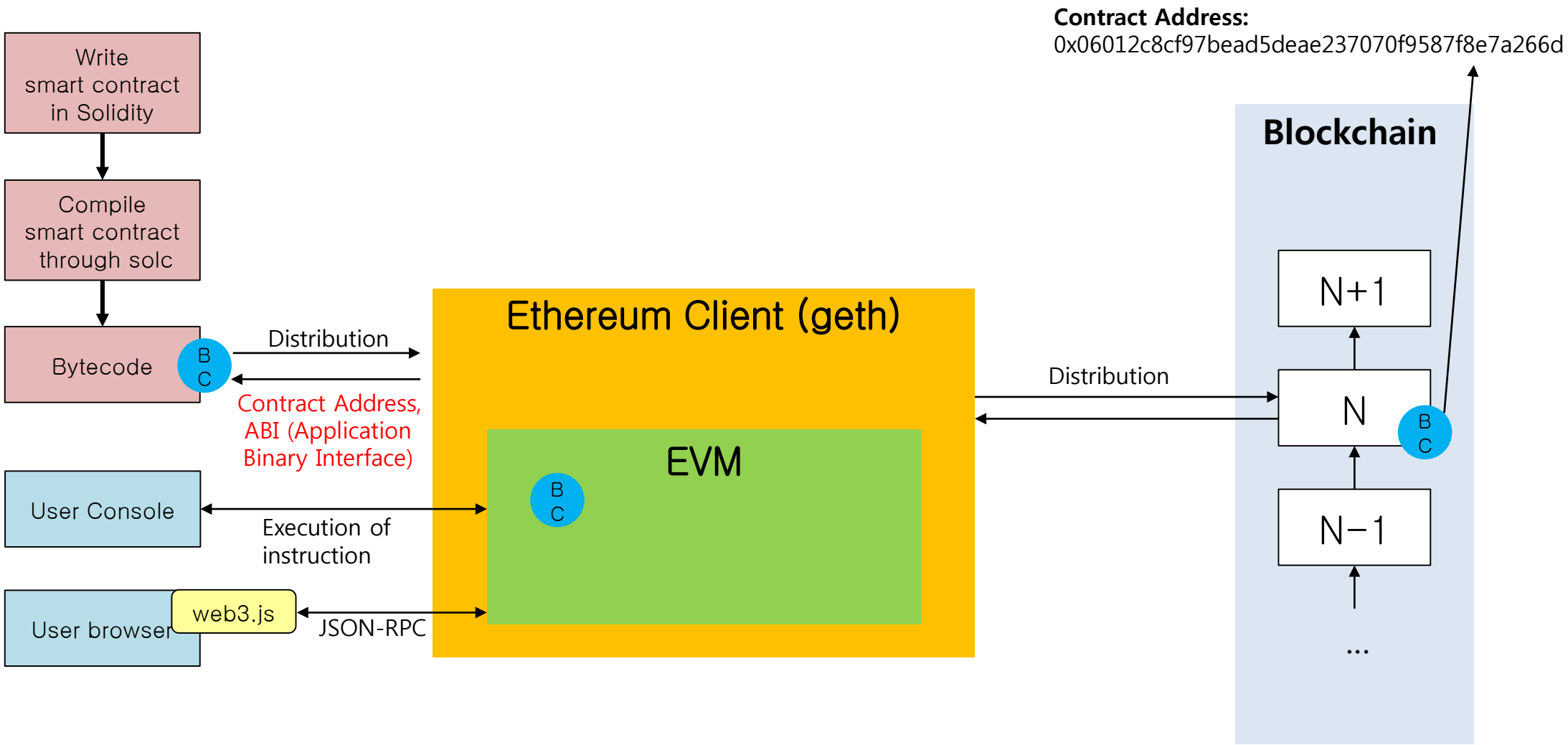
Smart Contract (2)

- **Smart Contract** in Ethereum
  - Make Etheruem as a blockchain computing platform
  - Program code that can change the state of an account (Turing Complete code)
  - Distributed on Ethereum P2P network and present as status information in blockchain
  - Run on the Ethereum Virtual Machine (EVM) to trigger state transitions
  - All content and input information is stored and shared in blockchain
    - prevent malicious manipulation of the contract
  - Smart Contract Languages: **Solidity**, Serpent, LLL, Mutan



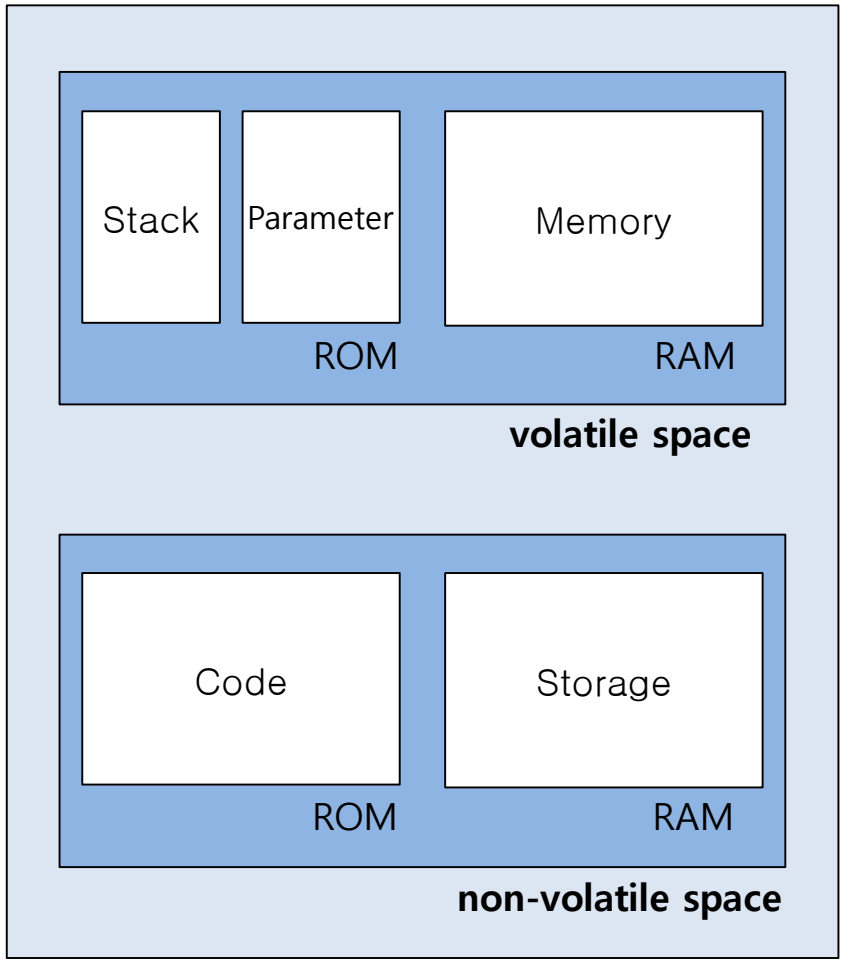
Execution Layer

- **Operating Process of Smart Contract**



Execution Layer

- **EVM (Etheruem Virtual Machine)**
  - A 32-byte stack-based execution environment that executes Bytecode of Smart Contract
  - Consist of volatile / nonvolatile memory, and store items of the stack in a byte array format
  - Consecutively executes opcodes while increasing the program counter from 0
    - **Stop condition**: Error, STOP and RETRUN command, Execution complete
  - Three spaces for running bytecode in EVM:
    1. Stack to push or pop values into LIFO container
    2. Memory that can hold infinitely expandable byte arrays
    3. Storage to persistently store value



EVM Architecture

Execution Layer

- **EVM Execution (1)**
  - Bytecode on “3+5”: 6003600501
  - Opcode: 0x60,0x03,0x60,0x05,0x01
    - 0x60: PUSH1
    - 0x03/0x05: Value
    - 0x01: ADD

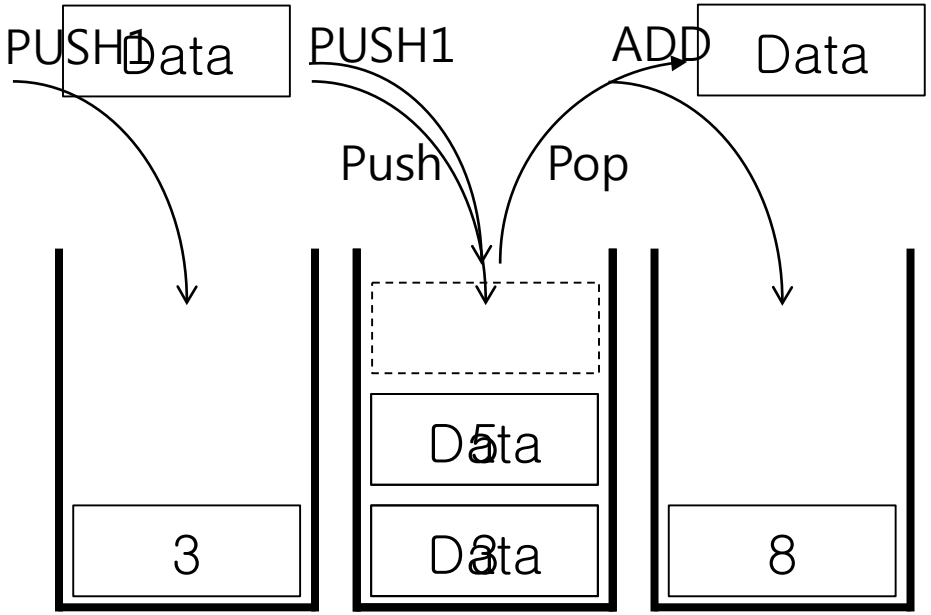
```
isac@isac06-PowerEdge-R610:~/go-ethereum/build/bin$ ./evm --debug --code 6003600501 run
0x
#### TRACE ####
PUSH1           pc=00000000 gas=10000000000 cost=3

PUSH1           pc=00000002 gas=9999999997 cost=3
Stack:
00000000  0000000000000000000000000000000000000000000000000000000000000003

ADD             pc=00000004 gas=9999999994 cost=3
Stack:
00000000  0000000000000000000000000000000000000000000000000000000000000005
00000001  0000000000000000000000000000000000000000000000000000000000000003

STOP            pc=00000005 gas=9999999991 cost=0
Stack:
00000000  0000000000000000000000000000000000000000000000000000000000000008

#### LOGS ####
```



Execution Layer

- **EVM Execution (2)**
  - Bytecode on “3x5”: 6003600502
  - Opcode: 0x60,0x03,0x60,0x05,0x02
    - 0x60: PUSH1
    - 0x03/0x05: Value
    - 0x02: MUL

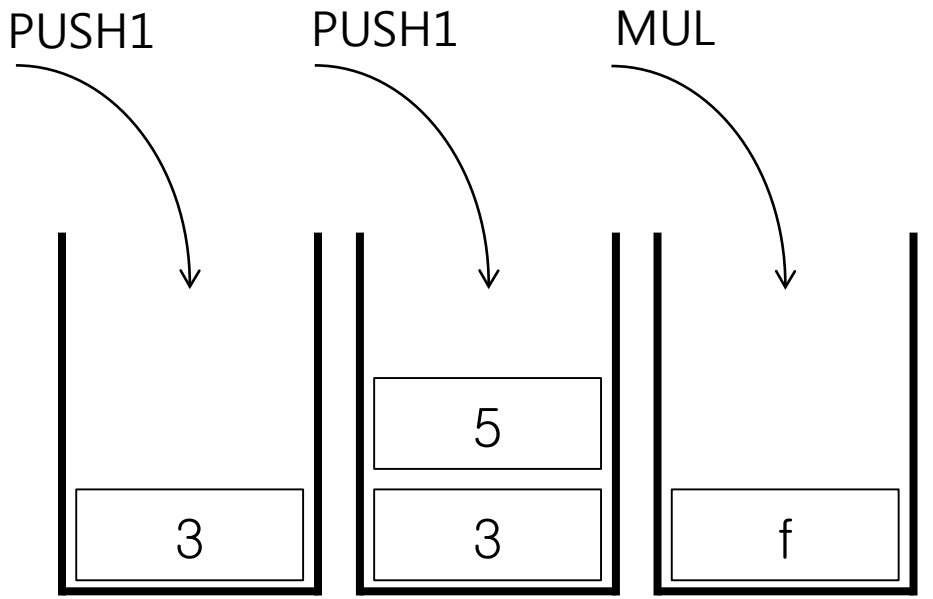
```
isac@isac06-PowerEdge-R610:~/go-ethereum/build/bin$ ./evm --debug --code 6003600502 run
0x
#### TRACE ####
PUSH1           pc=00000000 gas=10000000000 cost=3

PUSH1           pc=00000002 gas=9999999997 cost=3
Stack:
00000000  0000000000000000000000000000000000000000000000000000000000000003

MUL             pc=00000004 gas=9999999994 cost=5
Stack:
00000000  0000000000000000000000000000000000000000000000000000000000000005
00000001  0000000000000000000000000000000000000000000000000000000000000003

STOP            pc=00000005 gas=9999999989 cost=0
Stack:
00000000  000000000000000000000000000000000000000000000000000000000000000f

#### LOGS ####
```



Summary

- **Consensus Layer**
  - Ethash (PoW)
  - DAG
  - Proof of Stake (PoS)

- **Execution Layer**
  - Smart Contract
  - Operating Process of Smart Contract
  - EVM

References

- https://www.vijaypradeep.com/blog/2017-04-28-ethereums-memory-hardness-explained/
- https://github.com/ethereum/wiki/wiki/Ethash
- https://github.com/ethereum/wiki/wiki/Ethash-DAG
- https://github.com/trailofbits/evm-opcodes
- https://en.wikipedia.org/wiki/Smart_contract
- http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html
- http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html
- Jaehyun Park, **core ethereum programming**, Jpub, 2018
- DR. GAVIN WOOD, ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER, EIP-150 REVISION