Ethererum currency

- Ether (ETH) is the cryptocurrency generated by the Ethereum protocol as a reward to miners in a proof-of-work system for adding blocks to the blockchain
- It is the only currency accepted in the payment of transaction fees, which also go to miners
- The block reward together with the transaction fees provide the incentive to miners to keep the blockchain growing (i.e. to keep processing new transactions)
- Therefore, ETH is fundamental to the operation of the network
- Each Ethereum account has an ETH balance and may send ETH to any other account
- The smallest subunit of ETH is known as a Wei and is equal to 10^{-18} ETH
- Ether is often erroneously referred to as "Ethereum"
- Ether is listed on exchanges under the currency code ETH. The Greek uppercase Xi character Ξ is sometimes used for its currency symbol

Ethereum Accounts

two types of accounts on Ethereum:

1. user accounts (also known as externally-owned accounts)
2. Contracts

✓ *Both types have an ETH balance, may send ETH to any account, may call any public function of a contract or create a new contract, and are identified on the blockchain and in the state by their address*

User Accounts

- User accounts are the only type which may create transactions
- For a transaction to be valid, it must be signed using the sending account's private key, a 64-character hexadecimal string that should only be known to the account's owner
- The signature algorithm used is ECDSA
- this algorithm allows one to derive the signer's address from the signature without knowing the private key

Contracts

- Contracts are the only type of account which has associated code (a set of functions and variable declarations) and contract storage (the values of the variables at any given time)
- A contract function may take arguments and may have return values
- Within the body of a function, in addition to control flow statements, a contract's code may include:
- instructions to send ETH
- read from and write to its storage
- create temporary storage (memory) that dies at the end of the function
- perform arithmetic and hashing operations
- call its own functions
- call public functions of other contracts
- create new contracts
- query information about the current transaction or the blockchain

Ethereum addresses

- Ethereum addresses are composed of the prefix "0x", a common identifier for hexadecimal, concatenated with the rightmost 20 bytes of the Keccak-256 hash of the ECDSA public key
- In hexadecimal, two digits represent a byte, meaning addresses contain 40 hexadecimal digits, e.g. 0xb794f5ea0ba39494ce839613fffba74279579268
- Contract addresses are in the same format -- they are determined by sender and creation transaction nonce

Ethereum VM

- The Ethereum Virtual Machine (EVM) is the runtime environment for transaction execution in Ethereum
- It is a 256-bit register stack that is sandboxed from the node's other files and processes to ensure that for a given pre-transaction state and transaction, every node produces the same post-transaction state, thereby enabling network consensus

Ethereum Forks

- Changes to the rules of the Ethereum protocol which often include planned technical upgrades are called **Ethereum forks**
- Forks are when major technical upgrades or changes need to be made to the network
- Typically stem from Ethereum Improvement Proposals (EIPs) and change the "rules" of the protocol

Ethereum Forks Contd..

- Blockchains work differently because there is no central ownership
- Ethereum clients therefore must update their software to implement the new fork rules
- Also, block creators (miners in a proof-of-work world, validators in a proof-of-stake world) and nodes must create blocks and validate against the new rules

Ethereum Forks Contd..

- These rule changes may create a temporary split in the network
- New blocks could be produced according to the new rules or the old ones
- Forks are usually agreed upon ahead of time so that clients adopt the changes in unison and the fork with the upgrades becomes the main chain
- In rare cases, disagreements over forks can cause the network to permanently split
- Ex: creation of Ethereum Classic with the DAO fork

DAO fork

- The Altair upgrade is the first scheduled upgrade for the Beacon Chain
- It will add support for "sync committees", which can enable light clients, and will bring inactivity and slashing penalties up to their full values
- Altair is the first major network upgrade that will have an exact rollout time
- Every upgrade so far has been based on a declared block number on the proof-of-work chain, where block times vary
- The Beacon Chain does not require solving for proof-of-work
- Instead it works on a time-based epoch system consisting of 32 twelve-second "slots" of time where **validators** can propose blocks

https://ethereum.org/en/upgrades/beacon-chain/

Ethereum Gas

- Gas is a unit of account within the EVM used in the calculation of a transaction fee, which is the amount of ETH a transaction's sender must pay to the miner who includes the transaction in the blockchain
- Each type of operation which may be performed by the EVM is hardcoded with a certain gas cost, which is intended to be roughly proportional to the amount of resources (computation and storage) a node must expend to perform that operation
- When creating a transaction, the sender must specify a *gas limit* and *gas price*

Ethereum Gas

- All transactions on the Ethereum blockchain are required to cover the cost of computation they are performing
- The cost is covered by something called *gas* or *crypto fuel*, which is a new concept introduced by Ethereum
- This gas as *execution fee* is paid upfront by the transaction originators

Gas limit and Gas price

- The *gas limit* is the maximum amount of gas the sender is willing to use in the transaction
- The *gas price* is the amount of ETH the sender wishes to pay to the miner per unit of gas used
- The higher the *gas price*, the more incentive a miner has to include the transaction in their block, and thus the quicker the transaction will be included in the blockchain

Gas limit and Gas price Contd..

- The sender buys the full amount of gas (i.e. the *gas limit*) up-front, at the start of the execution of the transaction, and is refunded at the end for any gas not used
- If at any point the transaction does not have enough gas to perform the next operation, the transaction is reverted but the sender still pays for the gas used
- Gas prices are typically denominated in Gwei, a subunit of ETH equal to 10^{-9} ETH

Advantages of Ethereum fee mechanism

- ✓ to mitigate transaction spam
- ✓ prevent infinite loops during contract execution
- ✓ provide for a market-based allocation of network resources

Confirmed transactions per day

EIP

Ethereum Improvement Proposal (EIP)

Difficulty bomb

- The difficulty bomb is an Ethereum protocol feature that causes the difficulty of mining a block to increase exponentially over time after a certain block is reached, with the intended purpose being to incentivize upgrades to the protocol and prevent miners from having too much control over upgrades
- As the protocol is upgraded, the difficulty bomb is typically pushed further out in time
- The protocol has included a difficulty bomb from the beginning, and the bomb has been pushed back several times
- originally placed primarily to ensure a successful upgrade from proof of work to proof of stake, an upgrade which removes miners entirely from the design of the network
- The period during which the mining difficulty is increasing is known as the **"Ice Age"**

Consensus Mechanisms in Ethereum

- The consensus mechanism in Ethereum is based on the GHOST protocol originally proposed by *Zohar* and *Sompolinsky* in December 2013
- The Ghost protocol in Ethereum stands for (Greedy Heaviest Observed Subtree)
- To combat the way that fast block time blockchains suffer from a high number of stale blocks
  - i.e. blocks that were propagated to the network and verified by some nodes as being correct but eventually being cast off as a longer chain achieved dominance, or forking

GHOST protocol

- In GHOST, stale blocks are added in calculations to figure out the longest and heaviest chain of blocks
- An orphan, or stale block, is created when two nodes find a block at the same time
- The protocol also combats the issue of **centralization bias**
  - – the larger the pool the less time the more often they are going to get a head start on other miners by producing the block themselves and immediately start the race for the next block

GHOST protocol

- GHOST includes stale blocks – or Uncles as Ethereum calls them –
- These are included in the calculation of which chain is ongest or has the highest cumulative difficulty
- Centralization is solved by giving block rewards to stales of 87.5%
- the nephew (child of the Uncle block) also receives a reward of 12.5% of the block reward

Levels of GHOST protocol in Ethereum

The Ethereum version of Ghost only goes down seven levels – or back seven levels in the height of the block chain

- ✓ A block must specify its parents and its number of Uncles.
- ✓ An Uncle included in a block must be a direct child of the new block and less than seven blocks below it in terms of height
- ✓ It cannot be the direct ancestor of the block being formed.
- ✓ An Uncle must have a valid block header.
- ✓ An Uncle must be different from all other Uncles in previous blocks and the block being formed.
- ✓ For every Uncle included in the block the miner gets an additional 3.125% and the miner of the Uncle receives 93.75% of a standard block reward

Longest vs Heaviest Chain

The World State

- The world state in Ethereum represents the global state of the Ethereum blockchain
- It is basically a mapping between Ethereum addresses and account states
- The addresses are 20 bytes long
- This mapping is a data structure that is serialized using **Recursive Length Prefix**
- (RLP)
- RLP is a specially developed encoding scheme that is used in Ethereum to serialize binary data for storage or transmission over the network and also to save the state in a Patricia tree
- The RLP function takes an item as an input, which can be a string or a list of items, and produces raw bytes that are suitable for storage and transmission over the network
- RLP does not encode data; instead, its main purpose is to encode structures

The Account State

The account state consists of four fields

- ✓Nonce
- ✓Balance
- ✓Storageroot
- ✓Codehash

Nonce

- This is a value that is incremented every time a transaction is sent from the address
- In case of contract accounts, it represents the number of contracts created by the account

Balance

- ✓ This value represents the number of Weis which is the smallest unit of the currency (Ether) in Ethereum held by the address

Storageroot

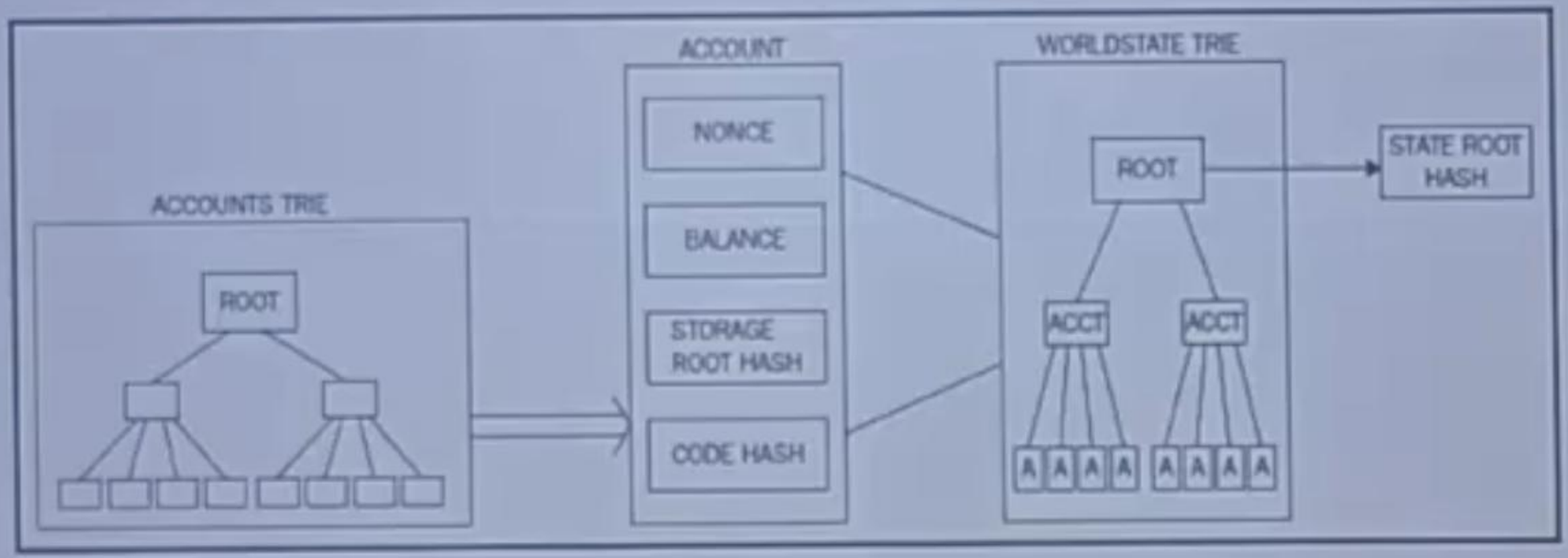
- This field represents the root node of a Merkle Patricia tree that encodes the storage contents of the account

Merkle Patricia tree example

Codehash

- This is an immutable field that contains the hash of the smart contract code that is associated with the account
- In the case of normal accounts, this field contains the Keccak 256-bit hash of an empty string
- This code is invoked via a message call

Accounts trie (storage contents of account), account tuple, world state trie, and state root hash and their relationship



Transactions in Ethereum

- A transaction in Ethereum is a digitally signed data packet using a private key that contains the instructions that, when completed, either result in a message call or contract creation
- Transactions can be divided into two types based on the output they produce:

1. **Message call transactions**: This transaction simply produces a message call that is used to pass messages from one account to another.
2. **Contract creation transactions**: As the name suggests, these transactions result in the creation of a new contract. This means that when this transaction is executed successfully, it creates an account with the associated code.

Fields in Ethereum Transactions

- Nonce
- gasPrice
- gasLimit
- To
- Value
- Signature

  Signature is composed of three fields, namely *v*, *r*, and *s*. These values represent the digital signature (*R*, *S*) and some information that can be used to recover the public key (*V*). Also of the transaction from which the sender of the transaction can also be determined. The signature is based on ECDSA scheme and makes use of the SECP256k1 curve
- Init (only for contracts)
- Data (message call)

Contract creation transaction

1. Sender
2. Original transactor
3. Available gas
4. Gas price
5. Endowment, which is the amount of ether allocated initially
6. A byte array of arbitrary length
7. Initialization EVM code
8. Current depth of the message call/contract-creation stack (current depth means the number of items that are already there in the stack)

The current version of Ethereum (homestead) specifies that the result of contract transaction is either a new contract with its balance, or no new contract is created with no transfer of value

Message call transaction

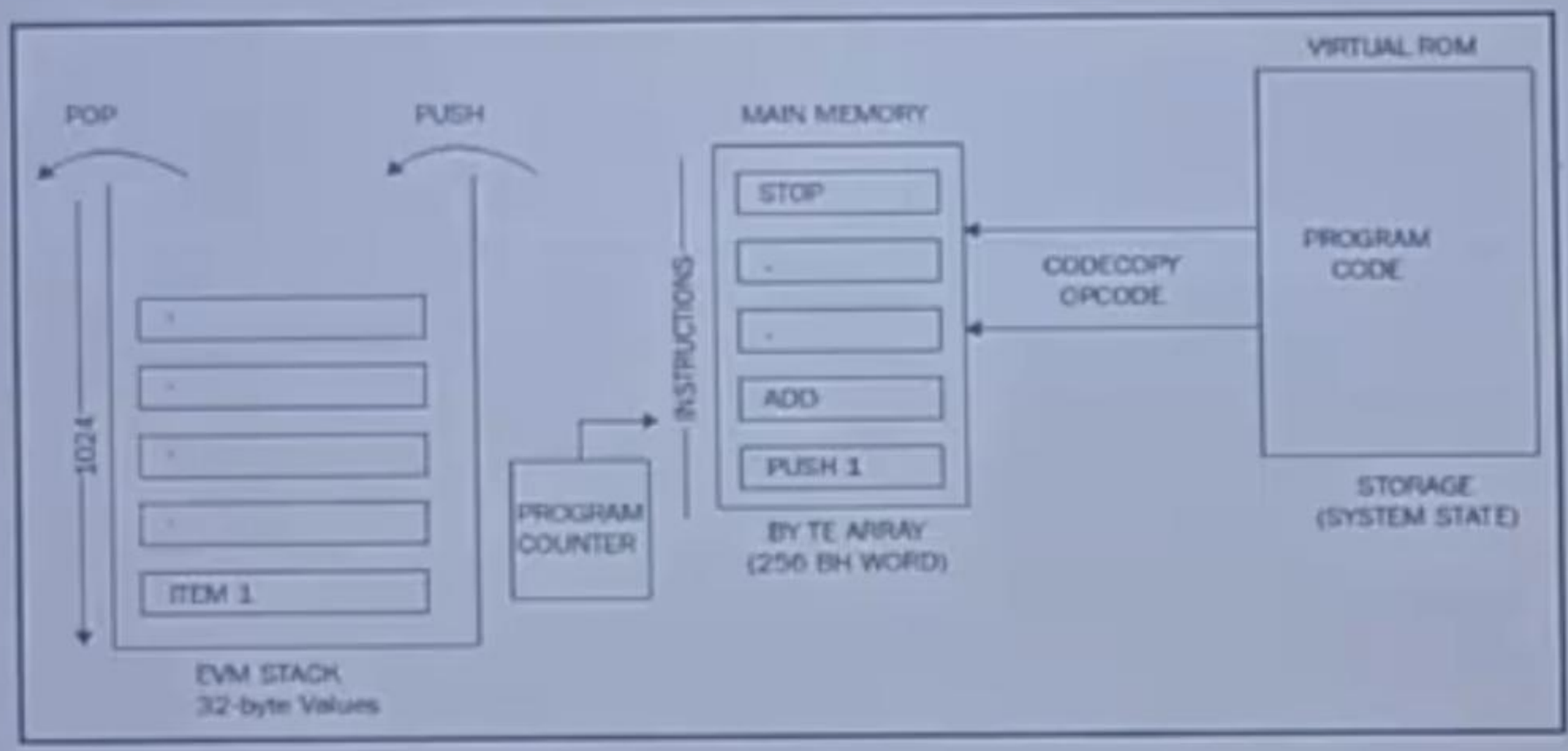
1. Sender
2. The transaction originator
3. Recipient
4. The account whose code is to be executed
5. Available gas
6. Value
7. Gas price
8. Arbitrary length byte array
9. Input data of the call
10. Current depth of the message call/contract creation stack

Message calls result in state transition. Message calls also produce output data, which is not used if transactions are executed

Ethereum Transactions

Elements of Ethereum Blockchain

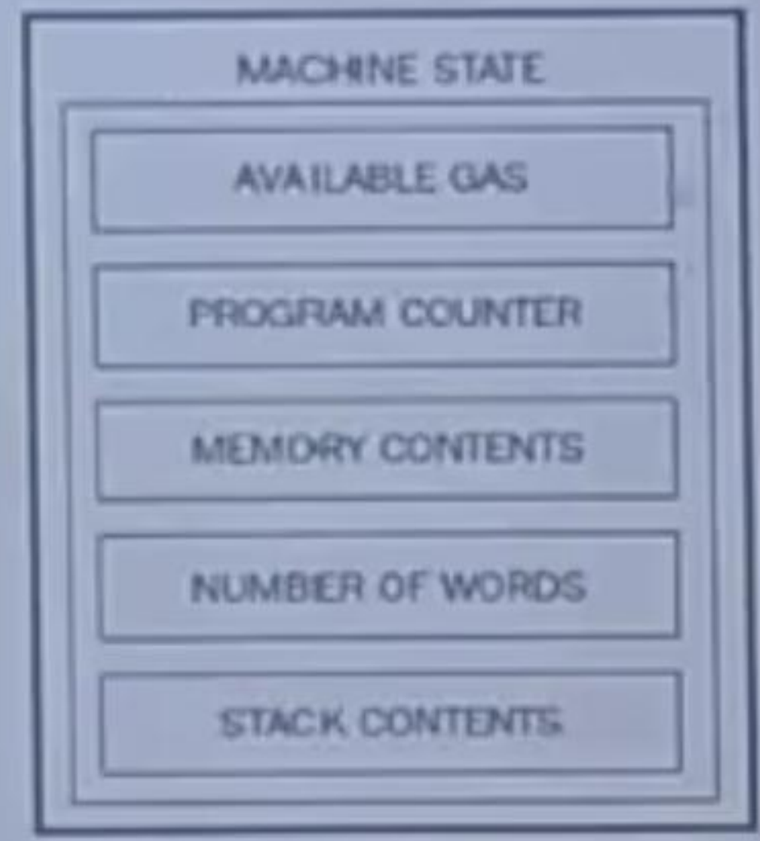
EVM



Execution Environment Tuple

Machine state

- Available gas
- The program counter, which is a positive integer up to 256
- Memory contents
- Active number of words in memory
- Contents of the stack



EVM exceptions

- Not having enough gas required for execution
- Invalid instructions
- Insufficient stack items
- Invalid destination of jump op codes
- Invalid stack size (greater than 1024)

The iterator function

- It fetches the next instruction from a byte array where the machine code is stored in the execution environment
- It adds/removes (PUSH/POP) items from the stack accordingly
- Gas is reduced according to the gas cost of the instructions/Opcodes
- It increments the **program counter** (**PC**)

LLL, Solidity, Serpent

- Code written in a high-level language such as Serpent, LLL, or Solidity is converted into the byte code that EVM understands in order for it to be executed by the EVM
- Solidity is the high-level language that has been developed for Ethereum with JavaScript such as syntax to write code for smart contracts
- Once the code is written, it is compiled into byte code that is understandable by the EVM using the Solidity compiler called solc

https://soliditylang.org/

EVM - Opcodes and their meaning

Arithmetic operations (modulo 2^{256})

Mnemonic	Value	POP	PUSH	Gas	Description
STOP	0x00	0	0	0	Halts execution
ADD	0x01	2	1	3	Adds two values
MUL	0x02	2	1	5	Multiplies two values
SUB	0x03	2	1	3	Subtraction operation
DIV	0x04	2	1	5	Integer division operation
SDIV	0x05	2	1	5	Signed integer division operation
MOD	0x06	2	1	5	Modulo remainder operation
SMOD	0x07	2	1	5	Signed modulo remainder operation
ADDMOD	0x08	3	1	8	Modulo addition operation
MULMOD	0x09	3	1	8	Module multiplication operation
EXP	0x0a	2	1	10	Exponential operation (repeated multiplication of the base)
SIGNEXTEND	0x0b	2	1	5	Extends the length of 2s complement signed integer

Logical operations

Mnemonic	Value	POP	PUSH	Gas	Description
LT	0x10	2	1	3	Less than
GT	0x11	2	1	3	Greater than
SLT	0x12	2	1	3	Signed less than comparison
SGT	0x13	2	1	3	Signed greater than comparison
EQ	0x14	2	1	3	Equal comparison
ISZERO	0x15	1	1	3	Not operator
AND	0x16	2	1	3	Bitwise AND operation
OR	0x17	2	1	3	Bitwise OR operation
XOR	0x18	2	1	3	Bitwise exclusive OR (XOR) operation
NOT	0x19	1	1	3	Bitwise NOT operation
BYTE	0x1a	2	1	3	Retrieve single byte from word

Cryptographic operations

Original Keccak implementation

Mnemonic	Value	POP	PUSH	Gas	Description
SHA3	0x20	2	1	30	Used to calculate Keccak 256-bit hash.

Contd...

Environmental information

Nomenclature	Value	POP	PUSH	Gas	Description
ADDRESS	0x30	0	1	2	Used to get the address of the currently executing account
BALANCE	0x31	1	1	20	Used to get the balance of the given account
ORIGIN	0x32	0	1	2	Used to get the address of the sender of the original transaction
CALLER	0x33	0	1	2	Used to get the address of the account that initiated the execution
CALLVALUE	0x34	0	1	2	Retrieves the value deposited by the instruction or transaction
CALLDATALOAD	0x35	1	1	3	Retrieves the input data that was passed a parameter with the message call
CALLDATASIZE	0x36	0	1	2	Used to retrieve the size of the input data passed with the message call
CALLDATACOPY	0x37	3	0	3	Used to copy input data passed with the message call from the current environment to the memory.
CODESIZE	0x38	0	1	2	Retrieves the size of running the code in the current environment
CODECOPY	0x39	3	0	3	Copies the running code from current environment to the memory
GASPRICE	0x3a	0	1	2	Retrieves the gas price specified by the initiating transaction.
EXTCODESIZE	0x3b	1	1	20	Gets the size of the specified account code
EXTCODECOPY	0x3c	4	0	20	Used to copy the account code to the memory.

Contd..

Block Information

Mnemonic	Value	POP	PUSH	Gas	Description
BLOCKHASH	0x40	1	1	20	Gets the hash of one of the 256 most recently completed blocks
COINBASE	0x41	0	1	2	Retrieves the address of the beneficiary set in the block
TIMESTAMP	0x42	0	1	2	Retrieves the time stamp set in the blocks
NUMBER	0x43	0	1	2	Gets the block's number
DIFFICULTY	0x44	0	1	2	Retrieves the block difficulty
GASLIMIT	0x45	0	1	2	Gets the gas limit value of the block

Stack, memory, storage and flow operations

Mnemonic	Value	POP	PUSH	Gas	Description
POP	0x50	1	0	2	Removes items from the stack
MLOAD	0x51	1	1	3	Used to load a word from the memory.
MSTORE	0x52	2	0	3	Used to store a word to the memory.
MSTORE8	0x53	2	0	3	Used to save a byte to the memory
SLOAD	0x54	1	1	50	Used to load a word from the storage
SSTORE	0x55	2	0	0	Saves a word to the storage
JUMP	0x56	1	0	8	Alters the program counter
JUMPI	0x57	2	0	10	Alters the program counter based on a condition
PC	0x58	0	1	2	Used to retrieve the value in the program counter before the increment.
MSIZE	0x59	0	1	2	Retrieves the size of the active memory in bytes.
GAS	0x5a	0	1	2	Retrieves the available gas amount
JUMPDEST	0x5b	0	0	1	Used to mark a valid destination for jumps with no effect on the machine state during the execution.

Contd...

Push operations

Mnemonic	Value	POP	PUSH	Gas	Description
PUSH0 ... PUSH32	0x60 ... 0x7f	0	1	3	Used to place *N* right-aligned big-endian byte item(s) on the the stack. *N* is a value that ranges from 1 byte to 32 bytes (full word) based on the mnemonic used.

Dupliation operations

Mnemonic	Value	POP	PUSH	Gas	Description
DUP1 ... DUP16	0x80 ... 0x8f	X	Y	3	Used to duplicate the nth stack item, where *N* is the number corresponding to the DUP instruction used. *X* and *Y* are the items removed and placed on the stack, respectively.

Swap operations

Mnemonic	Value	POP	PUSH	Gas	Description
SWAP1 ... SWAP16	0x90 ... 0x9f	X	Y	3	Used to swap the nth stack item, where *N* is the number corresponding to the SWAP instruction used. *X* and *Y* are the items removed and placed on the stack, respectively.

Contd..

Logging operations

Mnemonic	Value	POP	PUSH	Gas	Description
LOG0 ..... LOG4	0xa0 ..... 0xa4	X	Y (0)	375, 750, 1125, 1500, 1875	Used to append log record with *N* topics, where *N* is the number corresponding to the LOG Opcode used. For example, LOG0 means a log record with no topics, and LOG4 means a log record with four topics. *X* and *Y* represent the items removed and placed on the stack, respectively. *X* and *Y* change incrementally, starting from 2, 0 up to 6, 0 according to the LOG operation used.

System operations

Mnemonic	Value	POP	PUSH	Gas	Description
CREATE	0xf0	3	1	32000	Used to create a new account with the associated code.
CALL	0xf1	7	1	40	Used to initiate a message call into an account.
CALLCODE	0xf2	7	1	40	Used to initiate a message call into this account with an alternative account's code.
RETURN	0xf3	2	0	0	Stops the execution and returns output data.
DELEGATECALL	0xf4	6	1	40	The same as CALLCODE but does not change the current values of the sender and the value.
SUICIDE	0xff	1	0	0	Stops (halts) the execution and the account is registered for deletion later

Birla Institute of Technology & Science, Pilani
Hyderabad Campus

BLOCKCHAIN TECHNOLOGY
BITS F452
1st Sem 2022-23
Lecture 11
Ethereum Blockchain contd..

Click to add notes

Precompiled contracts in Ethereum

- **The elliptic curve public key recovery function (addr1)**
  - The ECDSA recovery function is shown as follows:

ECDSARECOVER(H, V, R, S) = Public Key

- **The SHA-256 bit hash function (addr2)**
- **The RIPEMD-160 bit hash function (addr3)**
- **The identity function (addr4)**
  - It simply defines output as input; in other words, whatever input is given to the ID function, it will output the same value.

Gas requirement is calculated by a simple formula: *15 + 3 [Id/32]* where *Id* is the input data.

Ethereum Accounts and Ethereum transactions

1. Confirm the **transaction validity** by checking the syntax, signature validity and nonce
2. **Transaction fee** is calculated and the sending address is resolved using the signature
3. Provide enough **ether (gas price)** to cover the cost of the transaction
4. In this step, the actual **transfer of value** occurs
5. In cases of **transaction failure** due to insufficient account balance or gas, all state changes are rolled back with the exception of fee payment, which is paid to the miners
6. Finally, the remainder (if any) of the fee is sent back to the sender as change and fee is paid to the miners accordingly. At this point, **the function returns the resulting state**

Ethrereum Block

- Blocks are the main building blocks of a blockchain
- Ethereum blocks consist of various components, which are described as follows:
  - ✓ The block header
  - ✓ The transactions list
  - ✓ The list of headers of Ommers or Uncles

Block header info::

- Parent hash
- Ommers hash
- Beneficiary
- State root (Keccak 256-bit hash of the root node of the state trie)
- Transactions root (Keccak 256-bit hash of the root node of the transaction trie)
- Receipts root
- Logs bloom
- Difficulty
- Number
- Gas limit
- Gas used
- Time stamp
- Extra data
- Mixhash
- Nonce

Sturcture of block and block header in Ethereum

Ethereum Genesis Block

Element	Description
Timestamp	(Jul-30-2015 03:26:13 PM +UTC)
Transactions	8893 transactions and 0 contract internal transactions in this block
Hash	0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3
Parent hash	0x0000000000000000000000000000000000000000000000000000000000000000
Sha3Uncles	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347
Mined by	0x0000000000000000000000000000000000000000 IN 15 secs
Difficulty	17,179,869,184
Total Difficulty	17,179,869,184
Size	540 bytes
Gas Limit	5,000
Gas Used	0
Nonce	0x0000000000000042
Block Reward	5 Ether
Uncles Reward	0
Extra Data	ëÛN4(NCE'!ƒpàµú3'ÛiEÛz8àà ,ù (Hex:0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa)

More info

Transaction receipts
The post-transaction state
Gas used
Set of logs

Bloom filter

A bloom filter is created from the information contained in the set of logs; Log entry is composed of the logger's address and log topics and log data. Log topics are encoded as a series of 32 byte data structures. Log data is made up of a few bytes of data.

Transaction validation and execution

- A transaction must be well-formed and RLP-encoded without any additional trailing bytes
- The digital signature used to sign the transaction is valid
- Transaction nonce must be equal to the sender's account's current nonce
- Gas limit must not be less than the gas used by the transaction
- The sender's account contains enough balance to cover the execution cost

The purpose of RLP (Recursive Length Prefix) is to encode arbitrarily nested arrays of binary data, and RLP is the main encoding method used to serialize objects in Ethereum. The only purpose of RLP is to encode structure; encoding specific data types.

Transaction substate

- **Log series**
  - This is an indexed series of checkpoints that allow the monitoring and notification of contract calls to the entities external to the Ethereum environment, such as application frontends
- **Suicide set**
  - This element contains the list of accounts that are disposed of after the transaction is executed
- **Refund balance**
  - This is the total price of gas in the transaction that initiated the execution. Refunds are not immediately executed; instead, they are used to partially offset the total execution cost.

The block validation mechanism

- Consistent with Uncles and transactions. This means that all Ommers (Uncles) satisfy the property that they are indeed Uncles and also if the Proof of Work for Uncles is valid.
- If the previous block (parent) exists and is valid.
- If the timestamp of the block is valid.

Block finalization steps

- ✓ Ommers validation
- ✓ Transaction validation
- ✓ Reward application
- ✓ State and nonce validation

senders nonce = txn nonce

Block difficulty

- Block difficulty is increased if the time between two blocks decreases, whereas it increases if the block time between two blocks increases
- This is required to maintain a roughly consistent block generation time.
- The difficulty adjustment algorithm in Ethereum's homestead release is shown as follows:

decreases

increases

timestamp-difference-based difficulty adjustment

```
block_diff = parent_diff + parent_diff // 2048 *
max(1 - (block_timestamp - parent_timestamp) // 10, -99) +
int(2**((block.number // 100000) - 2))
```

** denotes power

// denotes - integer division

https://ethereum.stackexchange.com/questions/5913/how-does-the-ethereum-homestead-difficulty-adjustment-algorithm-work

https://ethereum.stackexchange.com/questions/1880/how-is-the-mining-difficulty-calculated-on-ethereum/1910#1910

Interpretation

- The preceding algorithm means that, if the time difference between the generation of the parent block and the current block is less than 10 seconds, the difficulty goes up
- If the time difference is between 10 to 19 seconds, the difficulty level remains the same
- Finally, if the time difference is 20 seconds or more, the difficulty level decreases
- This decrease is proportional to the time difference

In addition to timestamp-difference-based difficulty adjustment, there is also another part that increases the difficulty exponentially after every 100,000 blocks
This is the so called *difficulty time bomb* or *Ice age* introduced in the Ethereum network, which will make it very hard to mine on the Ethereum blockchain at some point in the future

Ether denomination table

Unit	Wei Value	Weis
Wei	1 Wei	1
Babbage	1e3 Wei	1,000
Lovelace	1e6 Wei	1,000,000
Shannon	1e9 Wei	1,000,000,000
Szabo	1e12 Wei	1,000,000,000,000
Finney	1e15 Wei	1,000,000,000,000,000
Ether	1e18 Wei	1,000,000,000,000,000,000

Transaction cost

*Total cost = gasUsed * gasPrice*

Operation Name	Gas Cost
step	1
stop	0
suicide	0
sha3	30
sload	20
txdata	5
transaction	500
contract creation	53000

Fee schedule

- Gas is charged in three scenarios as a prerequisite to the execution of an operation

1. The computation of an operation
2. For contract creation or message call
3. Increase in the usage of memory

Message components

1. Sender of the message
2. Recipient of the message
3. Amount of Wei to transfer and message to the contract address
4. Optional data field (Input data for the contract)
5. Maximum amount of gas that can be consumed

Mining functions

1. Listens for the transactions broadcasted on the Ethereum network and determines the transactions to be processed.
2. Determines stale blocks called Uncles or Ommers and includes them in the block.
3. Updates the account balance with the reward earned from successfully mining the block.
4. Finally, a valid state is computed and block is finalized, which defines the result of all state transitions.

Ethash

- **Ethash** is the name of the Proof of Work algorithm used in Ethereum

Mining types

- ✓ CPU mining
- ✓ GPU mining
- ✓ Mining Rigs
- ✓ Mining pools

Ethererum clients

- Geth
- Eth
- Pyethapp
- Parity
- SPV light clients

Birla Institute of Technology & Science, Pilani
Hyderabad Campus

BLOCKCHAIN TECHNOLOGY
BITS F452
1st Sem 2022-23
Lecture 12
Hyperledgers

Introduction to Hyperledgers

- Hyperledger is not a blockchain, but it is a project that was initiated by Linux foundation in December 2015 to advance blockchain technology
- Collaborative effort by its members to build an open source distributed ledger framework that can be used to develop and implement cross-industry blockchain applications and systems
- The key focus is to build and run platforms that support global business transactions
- The project also focuses on improving the reliability and performance of blockchain systems

Projects under Hyperledger umbrella

- Fabric
- Iroha
- Sawtooth lake
- Blockchain explorer
- Fabric chaintool
- Fabric SDK Py
- Corda (most recent addition)

Definition of Hyperledger

- Hyperledger is a global enterprise blockchain project that offers the necessary framework, standards, guidelines, and tools to build open-source blockchains and related applications for use across various industries
- Building enterprise blockchain ecosystems through global, open source collaboration

- https://www.hyperledger.org/

Design philosophy behind Hyperledger projects

→ Modular

→ Highly Secure

→ Interoperable

→ Cryptocurrency-Agnostic

→ Complete with APIs

Enterprise blockchain and Hyperledger

- Blockchain solves the core problem of many organizations wanting to share data in a distributed database
- Blockchain technologies enable direct transactions in a secure, transparent way, baking trust into systems that operate with the efficiency of a peer-to-peer network
- Hyperledger technologies are open source code bases built with collaborative design and governance
- So enterprises have embraced them as trusted infrastructure for building blockchain solutions

Hyperledger Foundation Hosted Projects

HYPERLEDGER ARIES

HYPERLEDGER BESU

HYPERLEDGER FABRIC

HYPERLEDGER INDY

HYPERLEDGER IROHA

HYPERLEDGER SAWTOOTH

Graduated Hyperledger Projects

Incubating Hyperledger Projects

HYPERLEDGER AVALON

HYPERLEDGER BURROW

HYPERLEDGER CACTUS

HYPERLEDGER CALIPER

HYPERLEDGER CELLO

HYPERLEDGER EXPLORER

HYPERLEDGER FIREFLY

HYPERLEDGER GRID

HYPERLEDGER QUILT

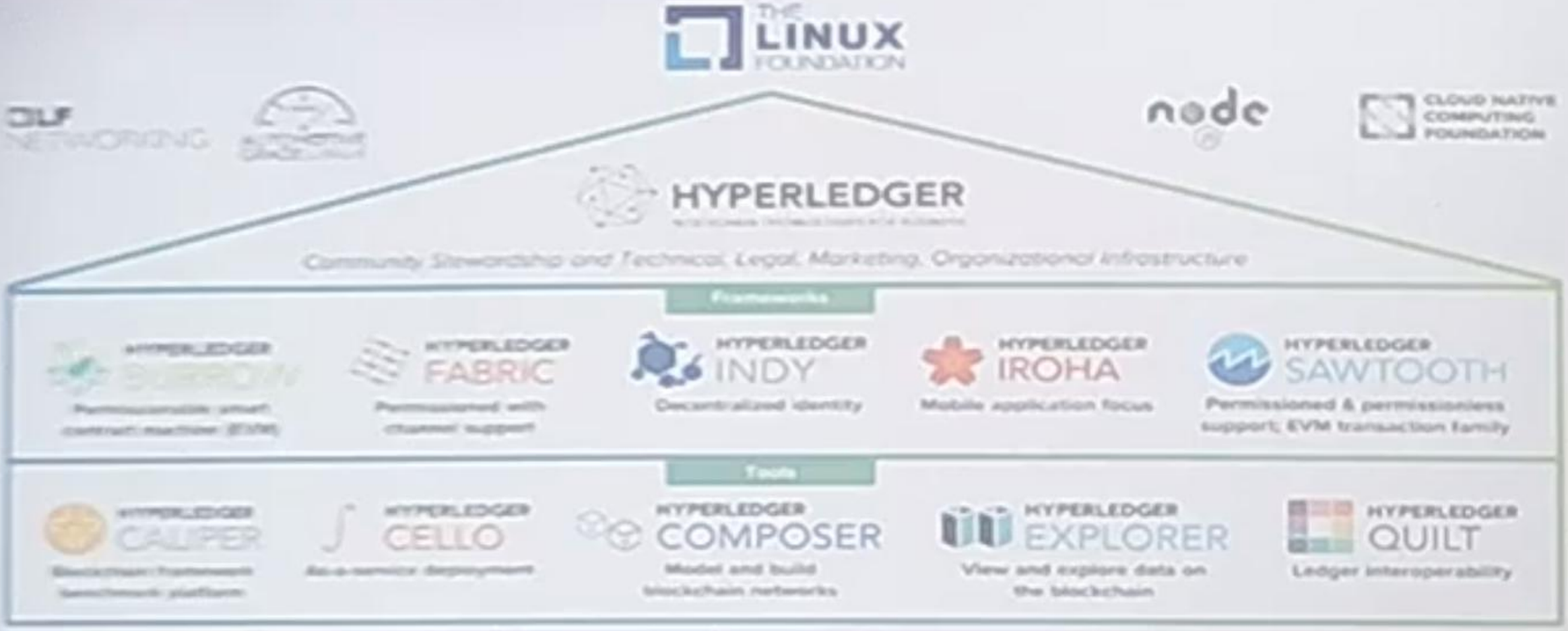
HYPERLEDGER TRANSACT

Hyperledger as a protocol

- Hyperledger is aiming to build a new blockchain platform that is driven by industry use cases
- Hyperledger blockchain platform is evolving into a protocol for business transactions
- Hyperledger is also evolving into a specification that can be used as a reference to build blockchain platforms as compared to earlier blockchain solutions that address only a specific type of industry or requirement

https://www.hyperledger.org/wp-content/uploads/2018/07/HL_Whitepaper_IntroductiontoHyperledger.pdf

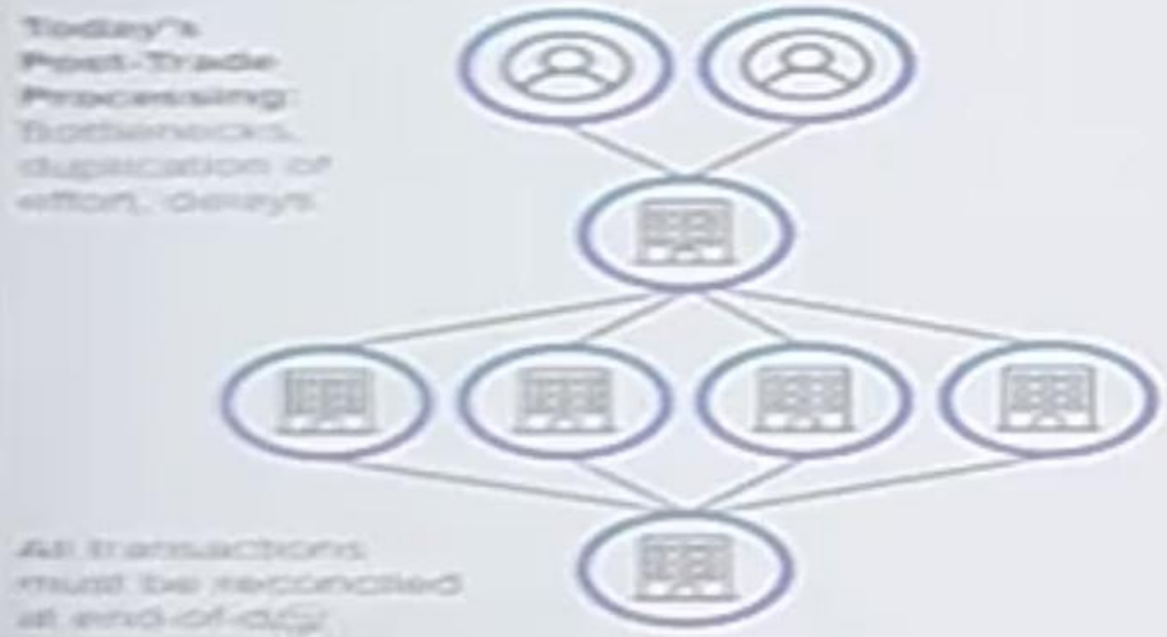
Hyperledger Greenhouse for Enterprise Blockchain



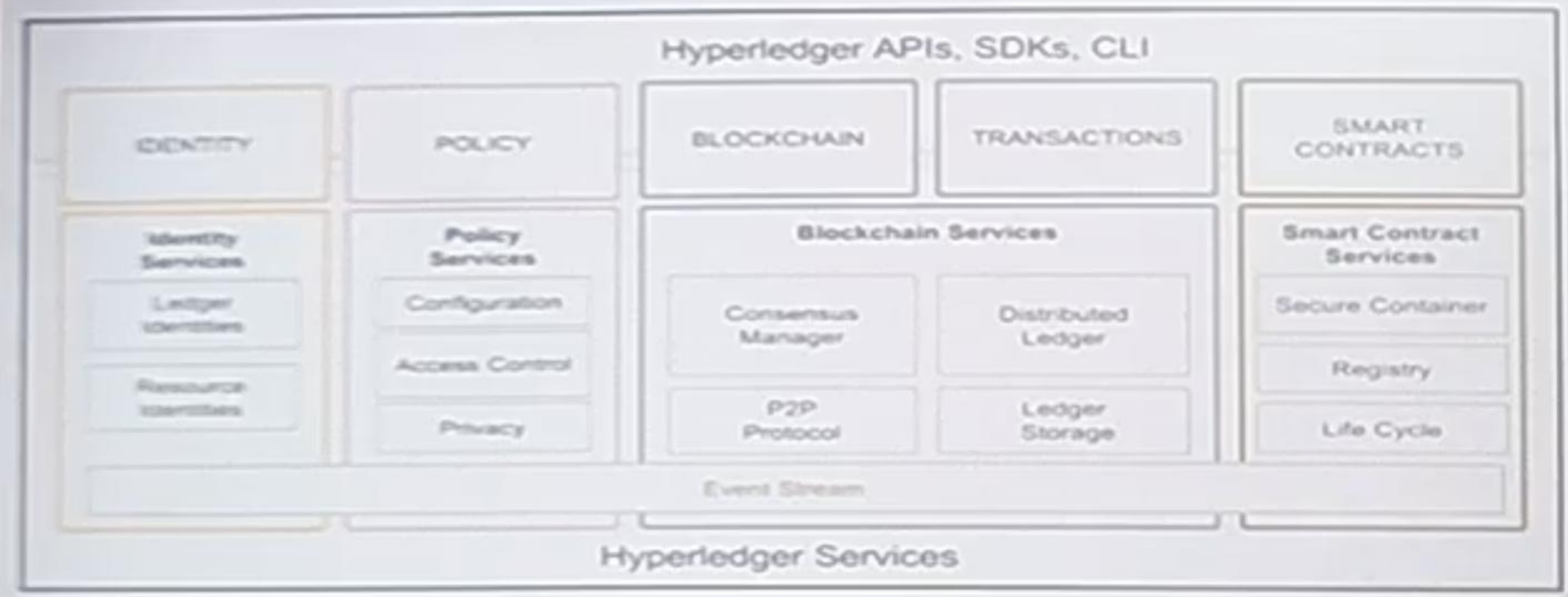
Use cases

- ✓ Banking—applying for a loan
- ✓ Financial services—post-trade processing
- ✓ Healthcare—credentialing physicians
- ✓ IT—managing portable identities
- ✓ Supply chain management—tracking fish from ocean to table

Hyperledger has useful tools available; in some cases, a proof a proof-of-concept has already been developed



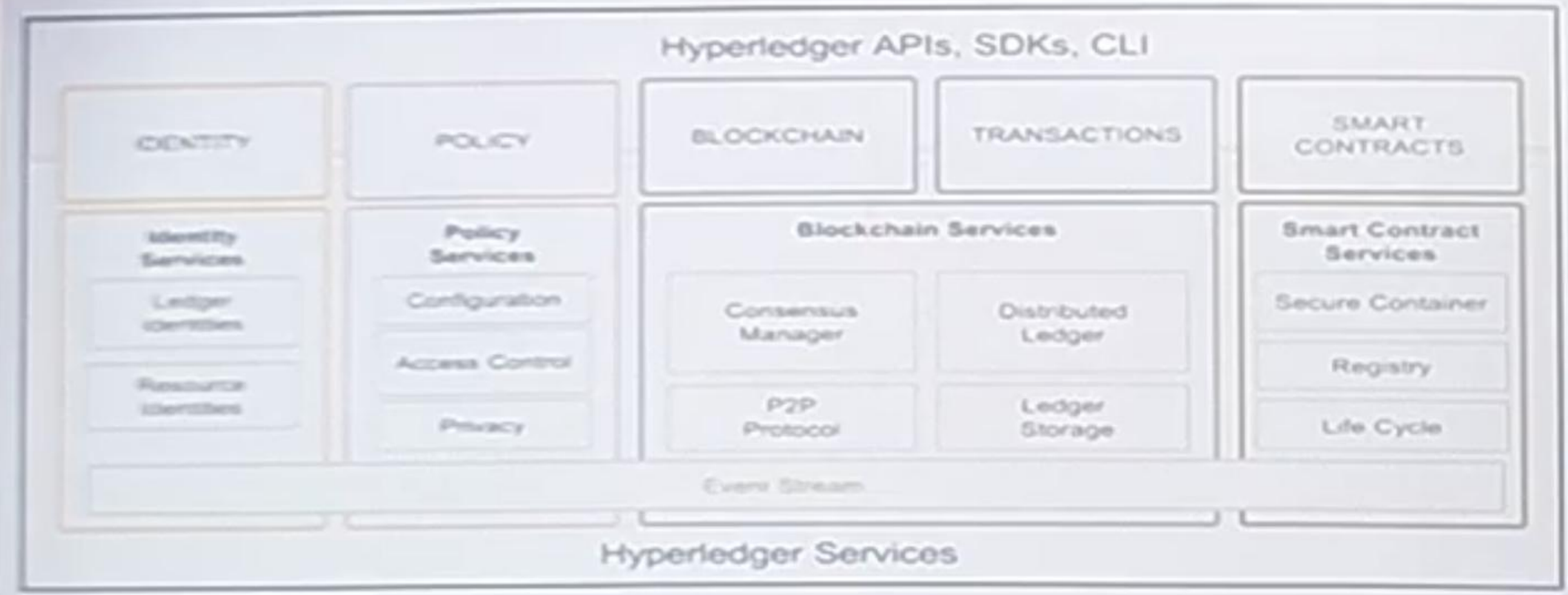
Hyperledger Architecture



Current Projects: Frameworks

- Distributed ledger frameworks
- Smart contract engines
- Client libraries
- Graphical interfaces
- Utility libraries
- Sample applications

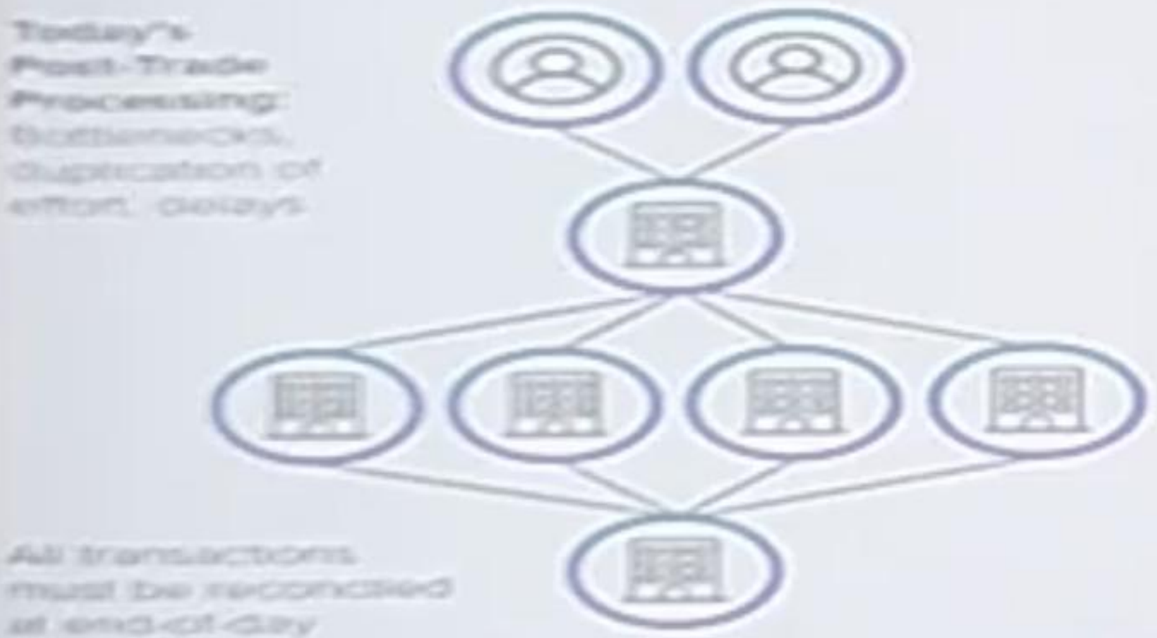
Hyperledger Architecture



Use cases

- ✓ Banking—applying for a loan
- ✓ Financial services—post-trade processing
- ✓ Healthcare—credentialing physicians
- ✓ IT—managing portable identities
- ✓ Supply chain management—tracking fish from ocean to table

Hyperledger has useful tools available; in some cases, a proof a proof-of-concept has already been developed



Current Projects: Frameworks

- Distributed ledger frameworks
- Smart contract engines
- Client libraries
- Graphical interfaces
- Utility libraries
- Sample applications

Summary of Hyperledger frameworks

HYPERLEDGER BURROW	A modular blockchain client with a permissioned smart contract interpreter developed in part to the specifications of the Ethereum Virtual Machine (EVM). bro-EVM
HYPERLEDGER FABRIC	A platform for building distributed ledger solutions with a modular architecture that delivers a high degree of confidentiality, flexibility, resiliency, and scalability. This enables solutions developed with Fabric to be adapted for any industry.
HYPERLEDGER INDY	A distributed ledger that provides tools, libraries, and reusable components purpose-built for decentralized identity.
HYPERLEDGER IROHA	A blockchain framework designed to be simple and easy to incorporate into enterprise infrastructure projects. mobile applications
HYPERLEDGER SAWTOOTH	A modular platform for building, deploying, and running distributed ledgers. Sawtooth features a new type of consensus, proof of elapsed time (PoET) which consumes far fewer resources than proof of work (PoW).

Hyperledger Fabric

- Fabric enables a modular, open and flexible approach towards building blockchain networks
- Various functions in the fabric are pluggable, and it also allows use of any language to develop smart contracts
- This is based on container technology which can host any language
- Chaincode (smart contract) is sandboxed into a secure container which includes a secure operating system, chaincode language, runtime environment and SDKs for Go, Java, and Node.js
- Smart contracts are called chaincode in the Fabric

Fabric Architecture Services

- membership services
- blockchain services
- chaincode services

1. Membership services

for access control

1. User identity validation
2. User registration
3. Assign appropriate permissions to the users depending on their roles

- Used PKI
- Components of membership services:
  - Registration authority (RA)
  - Enrolment certificate authority
  - Transaction certificate authority
  - TLS certificate authority

II. Blockchain services

Blockchain services are at the core of the Hyperledger Fabric

❑ Consensus manager

- Consensus manager is responsible for providing the interface to the consensus algorithm
- This serves as an adapter that receives the transaction from other Hyperledger entities and executes them under criteria according to the type of algorithm chosen

❑ Distributed ledger

- Blockchain and world state are two main elements of the distributed ledger. Blockchain is simply a linked list of blocks and world ledger is a key value database
- This database is used by smart contracts to store relevant states during execution by the transactions
- The blockchain consists of blocks that contain transactions.
- These transactions contain chaincode, which runs transactions that can result in updating the world state. Each node saves the world state on disk in RocksDB.

Block in the Hyperledger Fabric

- ✓ version
- ✓ timestamp
- ✓ Transaction hash
- ✓ State hash
- ✓ Previous hash
- ✓ Consensus metadata
- ✓ Non hash data

Version: Used for keeping track of changes in the protocol.
Timestamp: Timestamp in UTC epoch time, updated by block proposer.
Transaction hash: This field contains the Merkle root hash of the transactions in the block.
State hash: This is the Merkle root hash of the world state.
Previous hash: This is the previous block's hash, which is calculated after serializing the block message and then creating the message digest by applying the SHA3 SHAKE256 algorithm.
Consensus metadata: This is an optional field that can be used by the consensus protocol to provide some relevant information about the consensus.
Non-Hash data: This is some metadata that is stored with the block but is not hashed.

Peer to Peer protocol

- ✓ P2P protocol in the Hyperledger Fabric is built using google RPC (gRPC)
- ✓ It uses protocol buffers to define the structure of the messages
- ✓ Messages are passed between nodes in order to perform various functions

Four main types of messages in Hyperledger Fabric:

- Discovery
- Transaction
- Synchronization
- Consensus

iii. Chaincode services

These services allow the creation of secure containers

Secure container

- Chaincode is deployed in Docker containers that provide a locked down sandboxed environment for smart contract execution
- Currently Golang is supported as the main smart contract language, but any other main stream language can be added and enabled if required

Secure registry

- This provides a record of all images containing smart contracts

III. Chaincode services

These services allow the creation of secure containers

Secure container

- Chaincode is deployed in Docker containers that provide a locked down sandboxed environment for smart contract execution
- Currently Golang is supported as the main smart contract language, but any other main stream language can be added and enabled if required

Secure registry

- This provides a record of all images containing smart contracts

Summary of Hyperledger frameworks

HYPERLEDGER BURROW	A modular blockchain client with a permissioned smart contract interpreter developed in part to the specifications of the Ethereum Virtual Machine (EVM).
HYPERLEDGER FABRIC	A platform for building distributed ledger solutions with a modular architecture that delivers a high degree of confidentiality, flexibility, resiliency, and scalability. This enables solutions developed with Fabric to be adapted for any industry.
HYPERLEDGER INDY	A distributed ledger that provides tools, libraries, and reusable components purpose-built for decentralized identity.
HYPERLEDGER IROHA	A blockchain framework designed to be simple and easy to incorporate into enterprise infrastructure projects.
HYPERLEDGER SAWTOOTH	A modular platform for building, deploying, and running distributed ledgers. Sawtooth features a new type of consensus, proof of elapsed time (PoET) which consumes far fewer resources than proof of work (PoW).

Hyperledger Fabric

- Fabric enables a modular, open and flexible approach towards building blockchain networks
- Various functions in the fabric are pluggable, and it also allows use of any language to develop smart contracts
- This is based on container technology which can host any language
- Chaincode (smart contract) is sandboxed into a secure container which includes a secure operating system, chaincode language, runtime environment and SDKs for Go, Java, and Node.js
- Smart contracts are called chaincode in the Fabric

Fabric Architecture Services

- membership services
- blockchain services
- chaincode services

I. Membership services

for access control

1. User identity validation
2. User registration
3. Assign appropriate permissions to the users depending on their roles

- Used PKI
- Components of membership services:
  - Registration authority (RA)
  - Enrolment certificate authority
  - Transaction certificate authority
  - TLS certificate authority

II. Blockchain services

Blockchain services are at the core of the Hyperledger Fabric

❑ Consensus manager

- Consensus manager is responsible for providing the interface to the consensus algorithm
- This serves as an adapter that receives the transaction from other Hyperledger entities and executes them under criteria according to the type of algorithm chosen

❑ Distributed ledger

- Blockchain and world state are two main elements of the distributed ledger. Blockchain is simply a linked list of blocks and world ledger is a key value database
- This database is used by smart contracts to store relevant states during execution by the transactions
- The blockchain consists of blocks that contain transactions.
- These transactions contain chaincode, which runs transactions that can result in updating the world state. Each node saves the world state on disk in RocksDB.

III. Chaincode services

These services allow the creation of secure containers

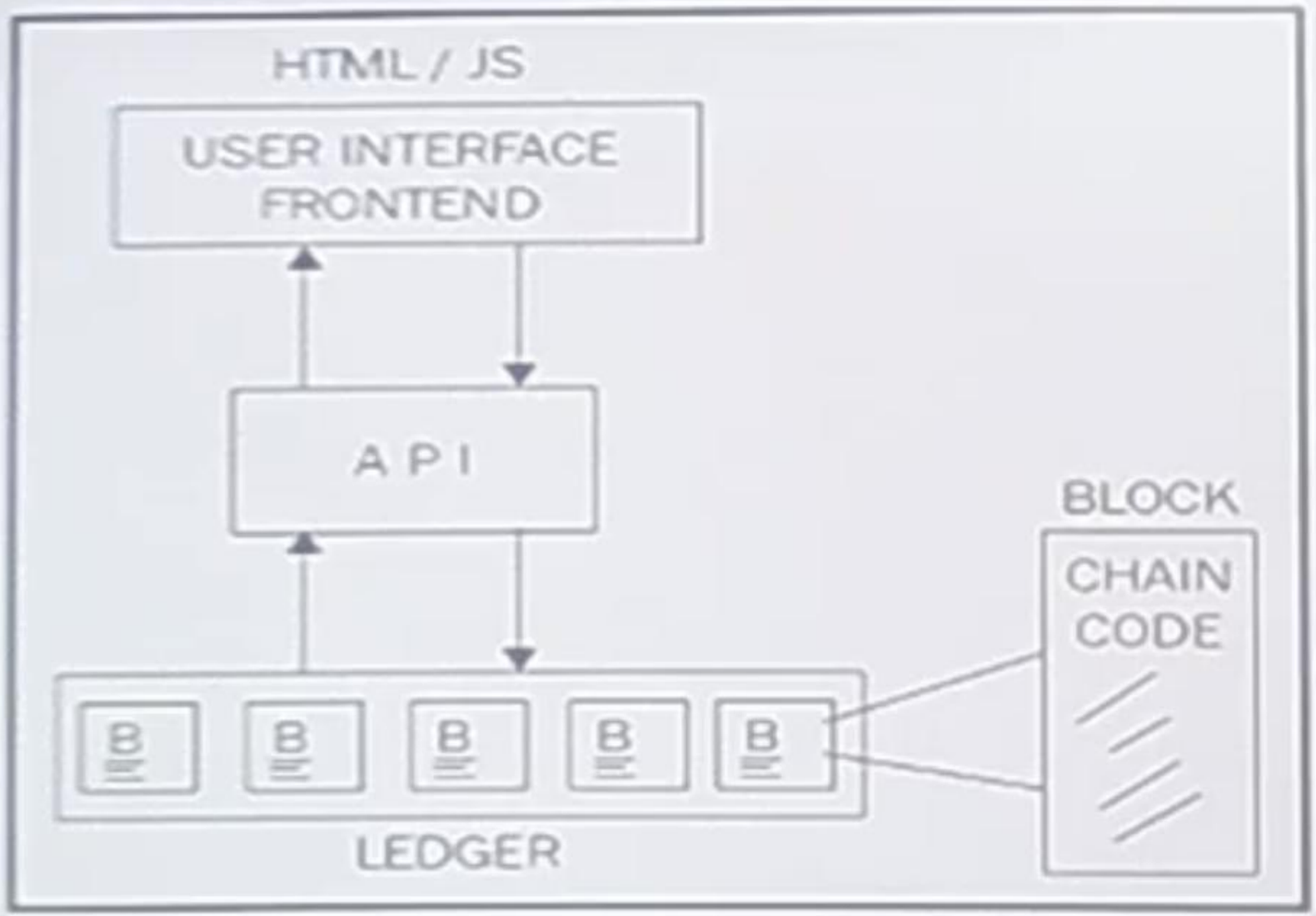
Secure container

- Chaincode is deployed in Docker containers that provide a locked down sandboxed environment for smart contract execution
- Currently Golang is supported as the main smart contract language, but any other main stream language can be added and enabled if required

Secure registry

- This provides a record of all images containing smart contracts

Applications on blockchain with Hyperledger



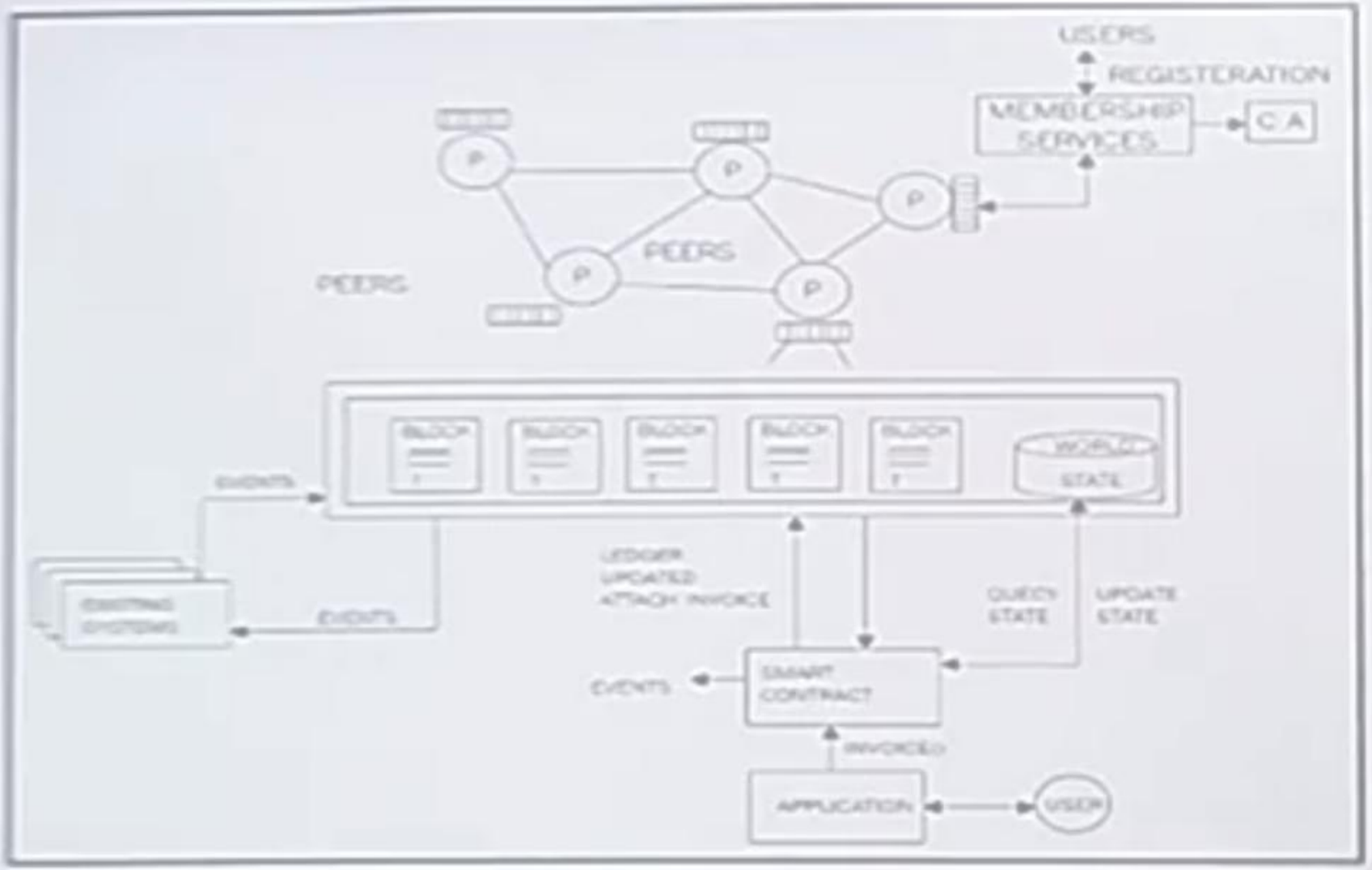
Chaincode implementation

- https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode4ade.html
- Functions required to implement Chaincode:

1. Init()
2. Invoke()
3. Query()
4. Main()

- Chaincode is a program, written in Go, Node.js, or Java that implements a prescribed interface
- Chaincode runs in a separate process from the peer and initializes and manages the ledger state through transactions submitted by applications

High-level overview of Hyperledger Fabric



Blockchain application model for Hyperledger Fabric

MVC based approach:

- View logic
- Control logic
- Data model
- Blockchain logic

Significance of Sawtooth lake

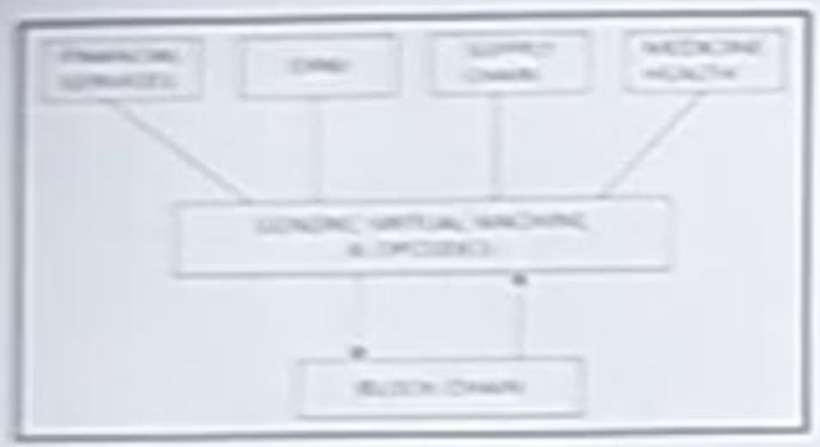
- Sawtooth lake can run in both permissioned and non-permissioned modes
- It is a distributed ledger that proposes two novel concepts

1. new consensus algorithm called **Proof of Elapsed Time (PoET)**
2. idea of transaction families

PoET

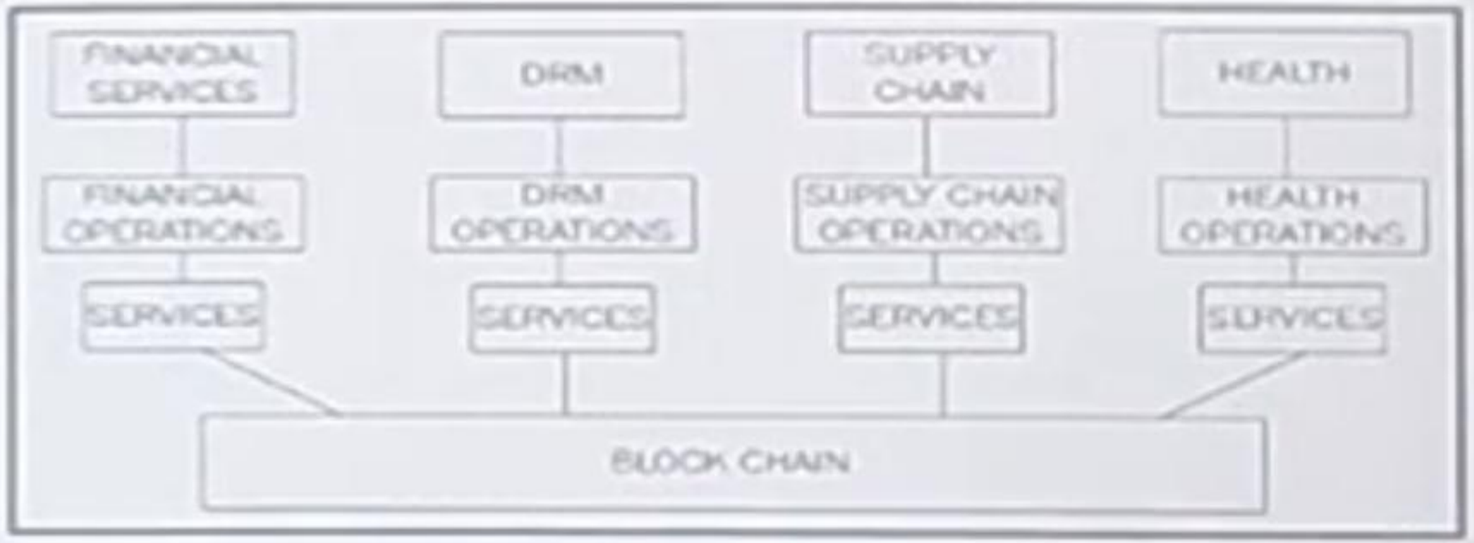
- PoET is a novel consensus algorithm that allows a node to be selected randomly based on the time that the node has waited before proposing a block
- This is in contrast to other leader election and lottery based proof of work algorithms, where an enormous amount of electricity and computer resources are used in order be elected as a block proposer, for example in the case of bitcoin
- PoET is a type of Proof of Work algorithm
- Instead of spending computer resources, it uses a trusted computing model to provide a mechanism to fulfill Proof of Work requirements.

Transaction families



Traditional Smart Contract paradigm

Sawtooth Transaction families
Smart Contract paradigm



Corda

- https://www.corda.net/
- Corda is not a blockchain!
- Traditional blockchain solutions, concept of transactions that are bundled together in a block and each block is linked back cryptographically to its parent block, which provides an immutable record of transactions
- Corda has been designed entirely from scratch with a new model for providing all blockchain benefits, but without a traditional blockchain
- It has been developed purely for the financial industry to solve issues arising from the fact that each organization manages their own ledgers and thus have their own view of *truth*, which leads to contradictions and operational risk
- Data is also duplicated at each organization which results in an increased cost of managing individual infrastructures and complexity
- These are the types of problems within the financial industry that Corda aims to resolve by building a decentralized database platform

Corda Architecture

- The main components of the Corda platform include state objects, contract code, legal prose, transactions, consensus, and flows

Advanced Message Queuing Protocol

Node 1 is communicating with **Node 2** over a TLS communication channel using the AMQP protocol, and the nodes have a local relational database for storage

Corda Permissioning service

- Network map service
- Notary service
- Oracle service
- Transactions (in a semi private network)

Transactions in a Corda network: - the components

1. Input references
2. Output states
3. Attachments (hash zip files)
4. Commands
5. Signatures
6. Type (Normal or Notary changing)
7. Timestamp
8. Summaries (text description on transactions)

More on Corda

Vaults: Vaults run on a node and are akin to the concept of wallets in bitcoin. As the transactions are not globally broadcast, each node will have only that part of data in their vaults that is considered relevant to them. Vaults store their data in a standard relational database and as such can be queried by using standard SQL. Vaults can contain both on ledger and off ledger data, meaning that it can also have some part of data that is not on ledger.

CorDapp: **Corda distributed application (CorDapp).**

Corda smart contracts:

1. Executable code that defines the validation logic to validate changes to the state objects.
2. State objects represent the current state of a contract and either can be consumed by a transaction or produced (created) by a transaction.
3. Commands are used to describe the operational and verification data that defines how a transaction can be verified.

Birla Institute of Technology & Science, Pilani
Hyderabad Campus

BLOCKCHAIN TECHNOLOGY
BITS F452
1st Sem 2022-23
Lecture 13
Blockchain:-
Enabling Technologies and Applications

Blockchain Technology Applications:

- Blockchain technology can be utilized in multiple industries including **Financial Services, Healthcare, Government, Travel and Hospitality, Retail and CPG**
- Ex:: Financial Services: In the financial services sector, Blockchain technology has already been implemented in many innovative ways.

Blockchain Enabling Technologies

- Blockchain as a key enabling technology for Decentralized Cyber-Physical Production Systems
- Blockchain as an enabling technology in the COVID-19 pandemic
- Blockchain as Key Enabling Technology for Future Electric Energy Exchange
- Blockchain and Internet of Things
- E-governance
- National security
- Elections and Voting
- Citizen identification (Aadhar in the Indian context)
- Media.....
- And the list is endless.....

Blockchain-based IoT model

Application Layer
transportation, financial, insurance and many others
Management Layer
data processing, analytics
Blockchain Layer
Security, P2P (MCM) autonomous transactions, decentralization, smart contracts
Network Layer
LAN, WSN, PAN, Routers
Device Layer
Sensors , Actuators, smart devices
Physical Objects
People, cars, boxes etc. etc

Blockchain-based direct communication model

Prominent Applications

- Secure sharing of medical data.
- NFT marketplaces.
- Music royalties tracking.
- Cross-border payments.
- Real-time IoT operating systems.
- Personal identity security.
- Anti-money laundering tracking system.
- Supply chain and logistics monitoring.

Scalability and other challenges

- Block size increase
- Block interval reduction
- Invertible Bloom lookup tables
- Sharding
- State channels (side channels)
- Private blockchains
- PoS
- Sidechains
- Subchains
- treechains

Other challenges

- Privacy
- Homomorphic encryption
- Zero knowledge proof
- Secure multiparty challenges
- Usage of hardware to provide confidentiality
- Confidential transactions
- Coinjoin

- ✓ Coinjoin is a technique which is used to anonymize the bitcoin transactions by mixing them interactively.
- ✓ The idea is based on forming a single transaction from multiple entities without causing any change in inputs and outputs.
- ✓ It removes the direct link between senders and receivers, which means that a single address can no longer be associated with transactions, which could lead to identification of the users.

Indistinguishability Obfuscation (IO)

- This cryptographic technique can address all privacy and confidentiality issues in blockchains
- The key idea behind IO is a *multilinear jigsaw puzzle*, which basically obfuscates program code by mixing it with random elements
- If the program is run as intended, it will produce expected output but any other way of executing would render the program look random and garbage

Smart Contract Security

- Formal verification of smart contract code

http://why3.lri.fr/

- Formal verification of Solidity code

Solidity implements a formal verification approach based on SMT (Satisfiability Modulo Theories) and Horn solving

The SMTChecker module automatically tries to prove that the code satisfies the specification given by require and assert statements

https://docs.soliditylang.org/en/v0.8.10/smtchecker.html

- ❑ Arithmetic underflow and overflow.
- ❑ Division by zero.
- ❑ Trivial conditions and unreachable code.
- ❑ Popping an empty array.
- ❑ Out of bounds index access.
- ❑ Insufficient funds for a transfer.

Birla Institute of Technology & Science, Pilani
Hyderabad Campus

BLOCKCHAIN TECHNOLOGY
BITS F452
1st Sem 2022-23
Lecture 14
Blockchain Technology
Case studies
&
Demo

Taxonomy of Blockchain based applications

Demo of Blockchain Implementation

https://ethereum.org/en/developers/docs/dapps/

Demo:

- dApp
  - Ganache (local ethereum blockchain)
  - Metamask (wallet manager for the personal blockchain)
  - Truffle Framework (for deployment and use of smart contracts)