Blockchain

EK → 2008 FC

fundamentals of Blockchain

Blockchain

Blockchain is a distributive Immutable ledger which is completly transparent.

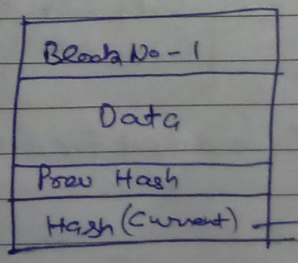
[ledger Book → खाता किताब]

Smart Contract A smart contract is a computer program or a transaction protocal which is intended to automatically execute, control, or document legally relevant events and actions accordiny to the terms of a contract or agreement.

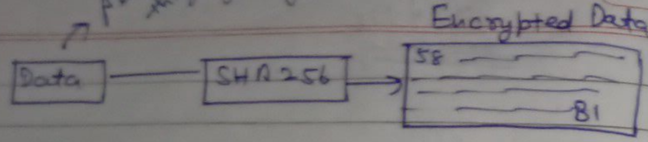
[Kind of like a if else program]

Hashing Algorithm →

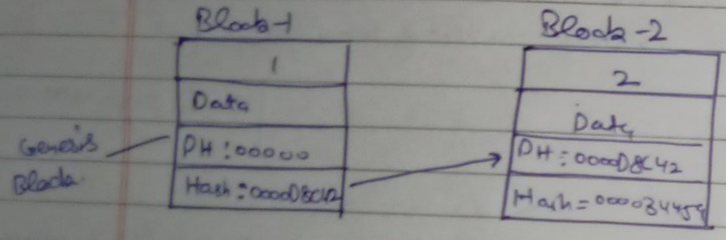
Bloackin Blockchain



→ Alot of other things are also which you will learn later



This has 64 hexadecimal characters. Each char is of 4 bit. So in total it has 64*4 bit i.e → 256 bit.



Remember one things that SHA256 will generate the same encrypted data when a specific value is added like for ABC it would be always (556) every time it execute. just example
But even one char change will lead to complete change in output

Hashing Algorithm
→ Requirements

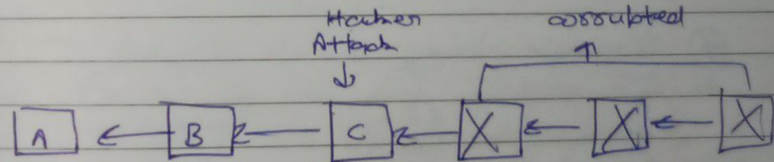
① One way i.e → Data → Encryption [Possible
[Data ← X Encrypt [Not Possible]]
[kind of confused here]
② Deterministic [Same output for same data]
③ Fast Computation.
④ withstand collision [Diff to Hack]

⑤ Avalanche Effect → Just a minor change in input will change the whole output completely

Concept of immutable ledger

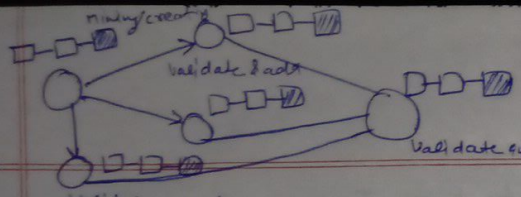
→ how each dependent Now as we know due to the distributed system each block is dependent on other i.e → if hacker tries to attack at least one computer then whole foremost blockchain will change which will fail the hack. cuz due to Avalanche effect

So it is theoritically possible to hack blockchain But we don't have that much enough power or not even the access of 51% at least of computers being used for blockchain around the world.



Distributed P2P Network Data is distributed over all the system or network. and one can access files from its adjacent system network.

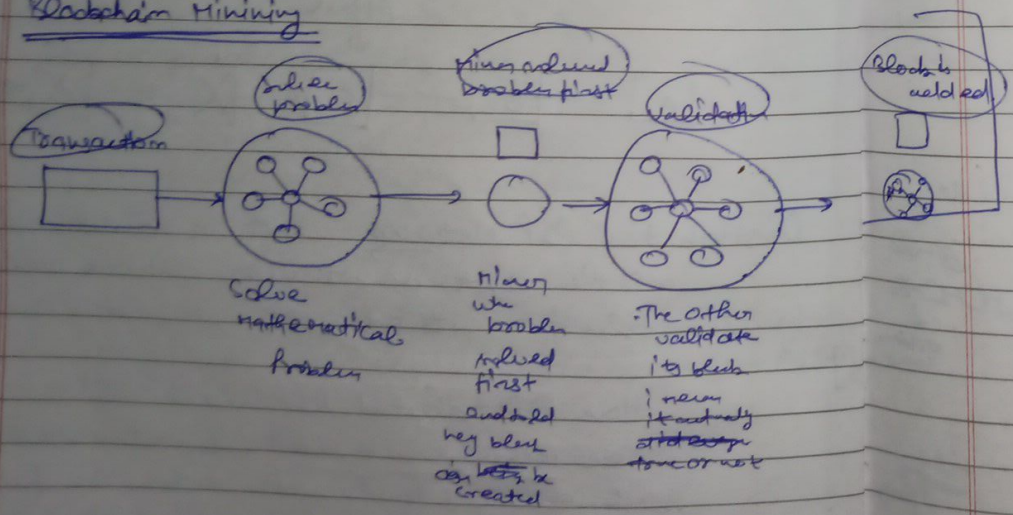
How Distributed P2P work.
These actually work like one initial node mine a Block and told to foremost member. The foremost been taken network checkout that its valid block or not then they simply pass it further to another peer network and the rest process is same.



Now how blockchain can't be hacked. Let's suppose hacker attacks on the blockchain But if he does that one block then the further will corrupt. So he hacks all the foremost one's also on a particular system in network.

But when here the next or fore most system will validate all the blocks and they don't find similar then correct so they asks him to drop and this way the corrupted data is recovered due to the distributed P2P Networks.

⊛ Blockchain Mining



double spending is attacks when a blockchain is hacked.

General attacks nowhere problem approved 66%. Majority for the decision taking. only if 66% majority they will finalize the decision.

Now this one is amazing. Before previously we understand that what if a hacker attacks in somewhere mid in the blockchain then the other system will validate through their blockchain and recover or solve every problem.

But this can be possible when attacked in somewhere before last node.

But what if the attack is on the mining node just i.e. added block.

How do the other validate then

for this there is consensus protocol. i.e. a algorithm which validates that the solved problem and of added blocks by the another peer system is valid.

Just think that the system which solved the problem and said "Hey I have solved and mined a block" has actually invested so much power and money if he found corrupting as malicious he won't even be rewarded/gets fee.

It like a lose-lose for him, so why he will do it. Its like really heavily costly for him.

Yes the Gas fees is the reward given to system's computing power and they solved the problem.

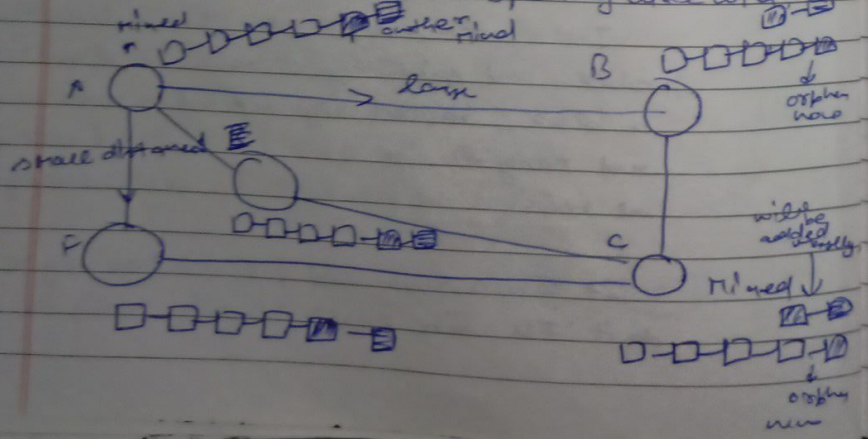
So yes the other system use PoW algorithm to validate the blocks added by another system

Actually the problem solving part is very time consuming. But this validation part is not.

Now what if at the same time two system solved the problem then whose block will be accepted.

Now to solve this the portion of the system which will create the longest blockchain will be considered as the right one and other have to accept it. Yes there will be some orphan blocks generate which will be discarded at that point. and they are not rewarded for them too.

One more thing that the large area will create another block. So ofc they will win



users like BCProblem this consensus protocol needs 51% majority to finalize decision.

Due to orphan blocks problem it is said that wait for the 6 confirmation befor assuming payment to be succesful.

Technology [Bitcoin Blockchain]

Protocol/Coin	Waves	Bitcoin	Ethereum

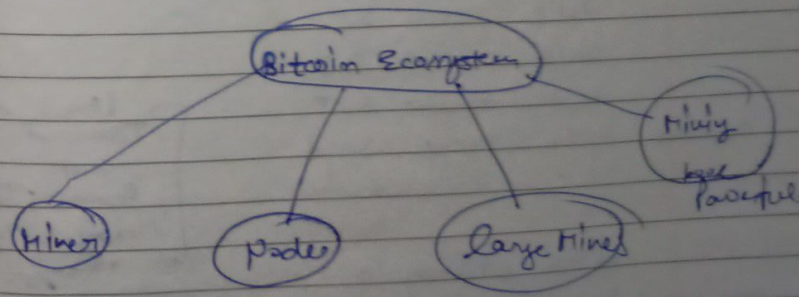
Tokens

Cool!

These are basically don't have their own blockchain and depend on coin's blockchain.

SHA-256 is used to encrypt blocks of blockchain for bitcoin or to currency.

EthHash is used for Ethereum.



The data in the blocks is immutable

Learn realistically * [i.e → To Manage supply demand]

[Bitcoin's Monetary Policy]

Two variables

The Halves

Each block that gets accepted to the main chain results in generation of new coins, or in other words a Block reward and to the miner to compensate them for the effort they have put in mining blocks.

and this reward gets halves after almost every 210,000 accepted blocks

		Block Number	Reward
Example	2009	0	50 new BT
	2012	210000	25
	2016	420000	12.5
	2020	630000	6.25
Expected	2024	740000	3.125
	⋮		
	2140	6930000	0

Now you might think, if no reward then why they will do mining [Transaction fees]

Block frequency

Average time to create Block is 10 mins.

[Gas] Transaction fees and Rewards are completely diff things.

→ i.e. for short computing power

Go explore Trustreet.com for visualization

" " Btc.com for block explorer what is happening on this blocks & the data.

Q→ what is a Block in Blockchain.

→ A Block is a container data structure. In the bitcoin world, a block contains more than 500 transaction on an average. The average size of block seems to be 1MB. The size can go upto 8MB.

Each blocks contain takes roughly average 10 minutes to be created.

* [only 21 Million Bitcoins are possible] *

Every thing that is being controlled and organised is handling through the algorithm of then Bitcoin etc so there is no centralization. Completely decentralized.

[How does Mining actually works]

a lot of Transactions → Mempool. → Solve Mathematical Problem

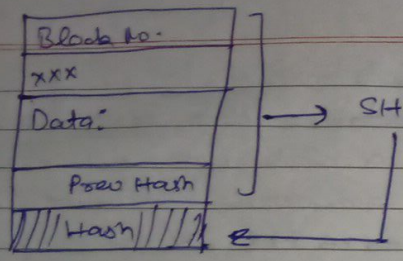
The first who solve will get reward of new generated bitcoin.

Block created by the miner → other Miners verify the validity → Block is added

reward is given now.

PoW

Block



Q→ What is Proof of work?

Ans→ In this miners compete to be the first to unlock a cryptographic puzzle encoding the order of transactions.

who
one solve the problem first and gets validated by other miners is rewarded with bitcoin.

Q→ what is proof of stake?

→ Participent miners stake 32 ethers.

→ Algorithm assign one member of the pool to order transactions into a block.

→ The verifier gets free ethereum if the block is approved but loose the staked coins if the block is rejected or found to be in error.

Q→ what is the Problem that is solved by miners. and what is Nonce. what's its role.

Ans→ Do you remember when we were talking that if some blocks have same value than the generated hash would always be same.

So here the concept of Nonce came

SHA-256

Block contents + Algorithm Process = Hash Value → will be same as in the block content.

Nonce It is a completly unique value which is added to the block content before giving it to the miners.

Now the miners have to process a large hit-trial calculation to find the particular nonce value which gives the value equal to given hash. and when someone solve the problem and find the particular nonce.

Then the other miners simply verifies it by adding into Block content and hashing the same hash given which makes clear that validation is quite easy which and takes less than second maybe that makes sense of .

Now I get what was the problem.

Block Content + SHA-256 Algorithm = Hash value

Nonce

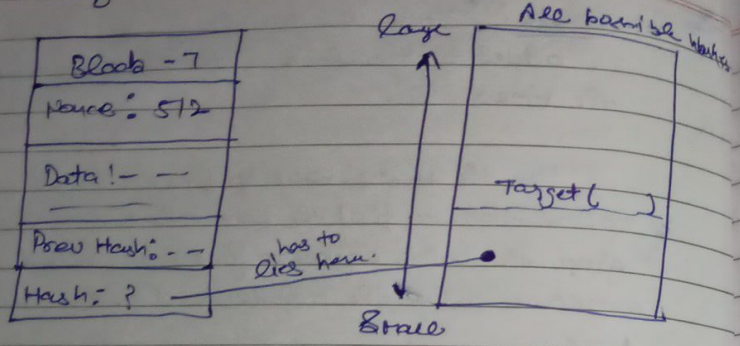
(How Mining works upper one is a bit mistaken)

Important Here understand one thing the given hash is actually a target hash. where miners have to randomly use a Nonce value and then combine it up with Block data & SHA to create a hash less than the target hash. That is the main goal.

and when the certain Nonce value creates a hash less than the target hash then that is the verified by other miners also to get it validated

Hash is [illegible]

Visually



And Basically SHA-256 produces a 256-bit (32 bytes) hash value. It's usually represented as a hexadecimal number of 64 digits.

Satoshi Nakamoto intentionally creates an algorithm which keeps increasing or decreasing the complexity of target hash value. So that the average time that miners take to generate a block keeps at a point of 10 minute at least. otherwise blocks will be generated at higher rate with the increase in technology. So the complexity gets increased with the time and requirment (as tech of miners rises) by the algorithm. It automatically precise that how much complexity needs to increased.

Heavy Hardwares consisting of

Mining pools consists of Heavy tech housed GPU and CPU's.

CPU's VS GPU's VS ASIC's

CPU < 10 MH (Mega Hashes) / s
GPU < 1 GH (Giga Hashes) / s
only ← ASIC > 1000 GHashes/S (2014)
for Hash creation.

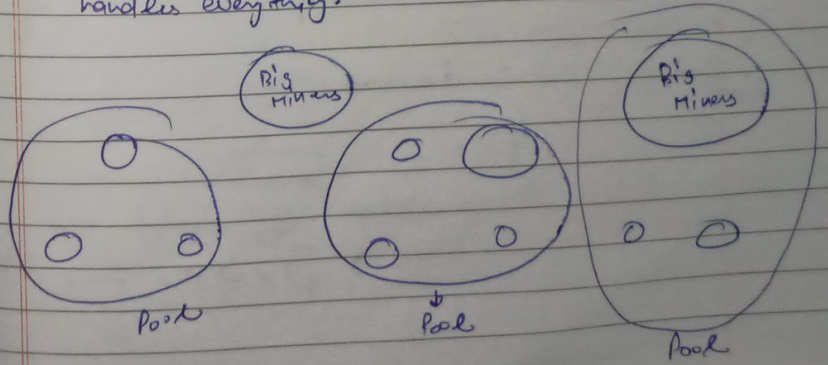
[Ethereum mostly]
These days ASIC has More powerful Hashrates
↓
Very costly

Mining Pools

Now as you know there is/are some community who put a ton of money only for this Mining. But they create so large Miners that small miners gets only loses in that case.

So what they do is rather than working individually on the same problem. These small miners create their mining pool by combining their powers together.

They are provided with an software which handles everything.



Permutation formula → $^nP_r = \frac{n!}{(n-r)!}$ → when order matters

Combination formula → $^nC_{r2} = \frac{n!}{r!(n-r)!}$ → when order not matters

→ both without replacement.

Nonce Range

Nonce Range :→ Nonce is a 32 bit number.

& the range of Nonce

0 to $2^{32} - 1 = 4 \times 10^9$

approx 4 billion.

Total no. of possible hashes $16^{64} = 10^{77}$

→

SHA 256 → 256 bit

i.e → 64 byte

where each place can have

16 value (Hexadecimal 0-F)

$16 \times 16 \times 16$ —— 64 times $\approx 16^{64}$

when you wrote 4 billion

i.e was from $2^{32} = 4 \times 10^9$

Blockchain

i.e was 0 1

↓

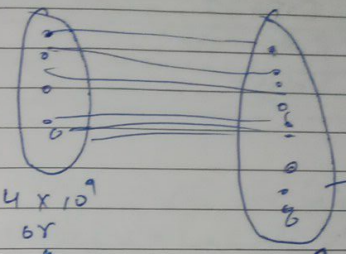
2 value possible

&

$[2 \times 2 \times$ —— 32 times$]$

So that shows that the total no. of hashes possible are far more than the hashes produced by the possible Nonces right.

like

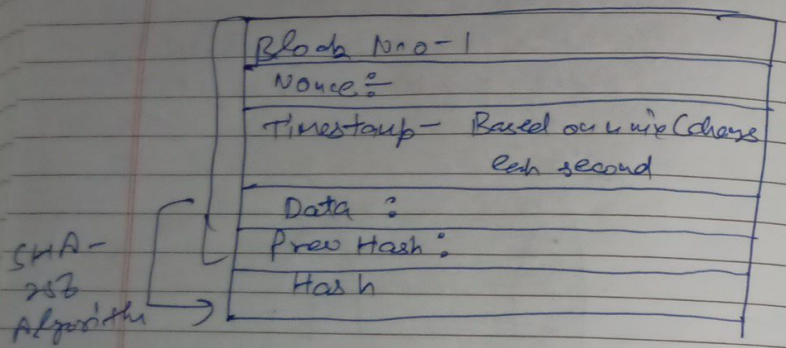


→ To get the hashes from the given target hashes could be in this answer i mean the required one.

& to hardware enough hashes there comes Timestamp data even though if you on Blockchain explorer then you will find that the speed of producing hash is faster than the with the timestamp as well so for this the method increase more complexity.

As by wiki 25 Feb 2022 the hash rate is 199 million hashes per second.

Timestamp is basically



Mining

Unix standard time which chooses every 1 sec. But this is very few for more hash rate

This high rate hash is due to mining pool (Combined mining)

But now comes a question will the mining work cuz it counts the possible hashes by the values he has in less than a second. So how it will reach to the hash i.e is out of its scope.

Tulp

So now they will pick diff transaction data from mempool by filtered by their gas prices. So they will be back roughly then to reach to the neg hash i.e less than the target hash.

Satoshi Nakamoto

'Si' Founder

Hello like Portal

Transactions & UTXO's

So UTXO are basically the amount of digital currency someone has left during a transaction. Like while spending for something.

↓

to complete the exact amount.

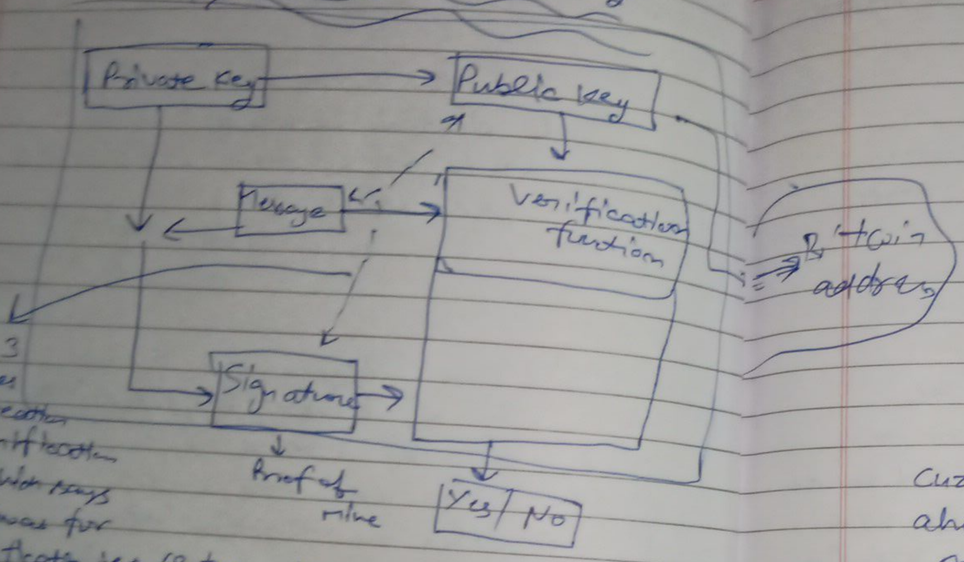
Transactions in Wallets

Remember one thing that there is not like Balance or remained money like in Banks.

There is only UTXO's which actually tells what exactly is your crypto balance i.e → wallets fetch UTXO's from blocks of blockchain i.e Someone → Me type transaction and select the required ones and by calculating them wallets tells your balance

* So its just virtual, i.e is not even in the hands of wallets too.

Private key & Public key



There 3 gives the creation of verification proof which says response for verification for the transaction sent by you is really by you.

Also the Signature is first generate with respect Private key and then the 3 [Public key, Message, Signature] are passed in v.f to get verified.

Now to add a extra level of security Public keys are not shared But its the Bitcoin address which is open. So that even if someone would trace Public key by some how then he needs also Private key also.

Segregated witness

So Basically the size of a Block is set to near about 1 MB But while the Past year transaction increased. So to solve this issue to fill more Transaction in a single block. What developers did is they just removed or seperate the signature & Public key from blocks. Cuz they are the witness of a transaction. also they took about 60-65% of A block's space.

Hierarchial Deterministic Wallets

Now in Private/Public section you see that there are kind of 3 layers i.e.→ Private key, Public key & then Bitcoin address.

But the developers [including Satoshi Nakamoto] thought of some more fact that if someone determine the pattern of Bitcoin address then they can possibly.

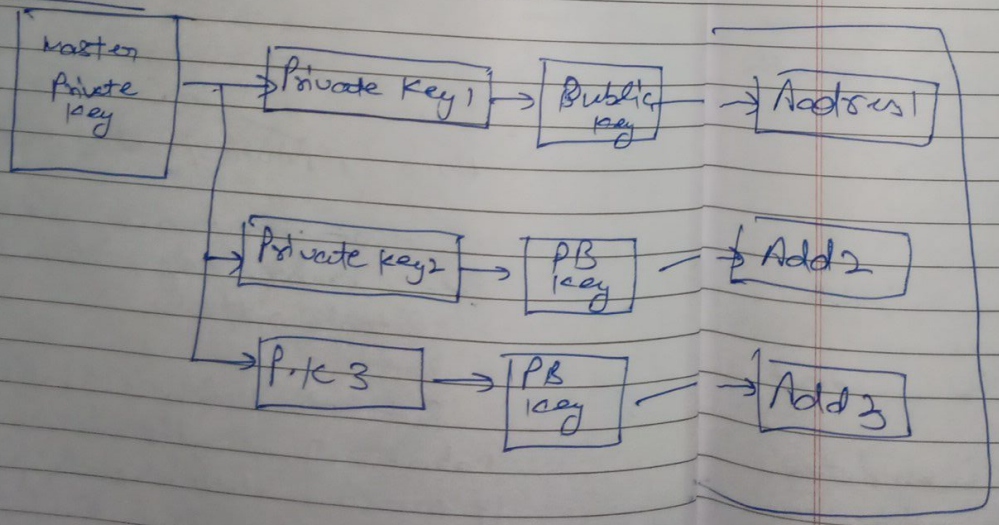
Date / /
Page No.

reach to the private key as the private key is more all the time.

So a new concept came where they add a another layer of creating Private key which help them to create as many normal private keys. So it is almost very very difficult to reach to private key.

Also if diff private keys can be divided into diff. people to get the do more transaction as controlled power. By this the Business owners can have an eye on when the money is spending By its derived private keys to the other company employees for his company.

Main Owner



Vitalik Buterin

Ethereum

He made the concept of running program on Blockchain itself. ies in Blocks.

Now both Bitcoin & Ethereum are protocols But bitcoin doesn't have tokens. i.e → bitcoin have its blockchain but no another token can use it

But Eth has token which can use its blockchain, why they will use that, simply bcz its more complicated/hard to hack a large blockchain you know that. So many token use Eth blockchain.

Eth is open source, blockchain based platform.

Types of Nodes

1. Full Node
2. Light Node
3. Archive Node.

full node :→

1. Locally stores a copy of the entire Blockchain
2. Verifies & validates all the blocks

G Host

Light Node
Only used for transactions

1. Stores only the block header. Depends on full node
2. For low capacity devices - which cannot afford to store gigabytes of data

Archive Node

1. Stores everything kept in the full node an archive of historical data.
2. Requires terabytes of diskspace.

Etherum Accounts

1. Externally owned Account [EOA] → It's kind of bitcoin wallet which is used for eth transaction
2. Contract Account → It is created when a smart contract is deployed on blockchain.

EOA	CA
1. Private key is needed	1. No Private or public key is needed.
2. Controlled by human	2. Controlled by contract code.
3. No gas is associated	3. Gas is associated.
4. Has a unique address	4. Unique address req
5. Hold Eth balance	5. Hold Eth balance

Smart Contract

Smart Contract is basically a program which works on blockchain.

Bitcoin Script	Solidity
Not Turing Complete	Turing complete
doesn't contain loops i.e. need to write code again & again	contains loops But they can be infinite so blockchain can be interrupted or have broken due to it.

Now the only diff b/w a bitcoin's blockchain block & Etherum blockchain block is simple, i.e. the block on Etherum can also run a program on itself i.e. the block on complete blockchain network

So Vitalik simply apply gas fees to run program on blockchain. So infinite not possible for someone to do

Awesome

Decentralized Application (DAPPS)

Centralized Apps	Decentralized Apps
1. Not trustworthy	1. Trustworthy/open source
2. Censorship	2. No censorship
3. You pay Cour attention to the	3. They pay you for your attention
4. Go down anytime	4. Can never go down

In centralized application there is a server or database which sends information. But on DAPPs they run on P2P Network. They are Decentralized.

#) Ethereum Virtual Machine [EVM]

Now as DAPPs run on a Network which may include our own system. So if someone upload a virus on Network then it's a big problem.

So for this we have EVM where our DAPPs run on. The DAPPs run on EVM so that no one can hack into it. i.e -> no hacking.

Etherum Gas

Now to run a program on Etherum block ~~of application~~ there is some gas fees which is already decided or measured according to operation you are doing on them.

for example for multiply -> 5 gas
subtr -> 3 gas
equal to -> 3 gas etc.

Imp Points

① Any transaction that modifies the blockchain costs gas
② The user that generated the transaction pays for gas fees

Gas prices -> ① It is the amount that the sender wants to pay per unit of gas to get the transaction mined. Gas prices is set by sender.

② Gas prices are denoted in Gwei.

[1 Gwei = 10^-9 eth] (10^{-9} ether)

1 Gas price = 1000 gwei [It varies as acc to the user wants to give]

But the higher the Gas price the fast will be the transaction by the miner due to prefer while picking from mempool.

#) Gas limit

Gas limit is basically used so that the infinite loop or Hackers attack, or heavy operations can be stopped which are sign of attacks.

Max Gas limit is Basically = 21000 units.

So if someone set per unit gas = 100 gwei

where 1 gwei = 10^{-9} ether

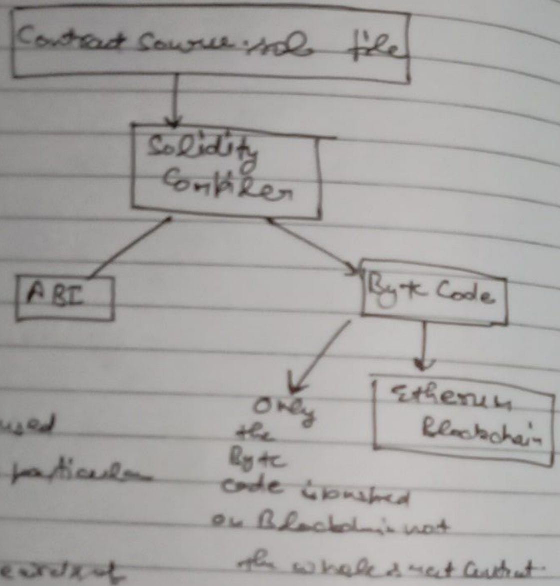
~~[illegible]~~

In case of less gas the eth is not back due to miner already checked

Solidity

Remix ide support Solidity & Vyper
~~written~~ written in → JS

[Smart Contract Compilation]



\# (ABI) #

ABI is the file (medium) which is used to communicate with this particular smart contract.

You can say it make readable data, function names to the code so that some other can call them or set or get values.

\# (Byte code) #

Byte code is only used to generate of code then this BC is pushed on Blockchain for process

An opcode is the first byte of an instruction in machine language which tells the hardware what operation needs to be performed with this instruction.

Opcode is the instruction given to EVM.

\# The pragma directive

pragma ~~...~~ solidity >=0.7.0 < 0.9.0

//used for telling the version bw the solidity can be used for here compilation

also pragma solidity ^0.8.0

// for specific version telling

Some Important points →

* Contract byte in public in readable form.
* Contract doesn't have to be public
* Bytecode is immutable
* ABI act as a bridge between application and smart contract (kind of API)
* ABI and Bytecode cannot be generated without source code

* Mainnet VS Testnet *

Mainnet	Testnet
(1) Used for actual transaction	(1) Used for testing smart contracts and decentralized application
(2) Mainnet network ID is 1	(2) Testnet have network ID of 3, 4 and 42
(3) Ethereum mainnet Eg	(3) Eg → Rinkeby test Network

Storage → ① Key value binder study
① expensive ② Store and Persist on blockchain [immutable]
③ Stores data in field of 256 bit size i.e. 256 bit is must allocated [i.e. 32 byte word]

Feature to get freq but data type and memory to be in Solidity

Memory ① Byte array with slot size of 256 bits. Data is stored in it only during function execution then deleted out after function execution.
② stored not on blockchain.
③ less expensive

Stack ① Internal place where numbers are stored in 32 bit slots
② Don't cause 256 bit

Metamask

Used for

- Store Ether
- Send Ether
- Receive Ether
- Run DAPPS
- Swap Tokens

Read the Blue pen carefully Imp

[Contract Deployment]

JS VM Berlin
3 points extra] → Are the Deployment bases

(#)

Solidity

High level statically typed programming language.
↓
need to tell everything at compilation time

Case Sensitive.

State variables

State variable are those which are stored on the Blockchain and as they are expensive and needs some cost.
for e.g. → uint age;

Three Methods by which state variable can be initialized,

Q. How do you mean the 256 bit slot arranged for storing values in "storage". i.e. 256 bit slot is fixed even if you save a 8 bit value only. They kind of set to save storage [8 bit 8 bit 16 bit ... etc 256 bit]

This happens when you store storage

① Using the contracts Constructor
② Initializing the variable at the declaration itself
③ Using the set function (creating manually)

→ in state var they have to be initialized while declaration

Imp when memory keyword is possible → [How memory keyword can only be specified for array, struct & maps] uint[] not [uint]

Other properties →
① Permanently stored in contract storage
② Cost gas required [Expensive]
③ storage not dynamically allocated.
④ Instance of contract cannot have other state variables besides those already declared.

Local variables of struct, array or mapping type ref are stored in storage by default. Use memory in local for temp memory

Also One more thing when you use (public) then a set function is automatically created for that particular variable.

State var	stored in	Storage (always)
fun arguments	"	Memory
Local var of value type (uint, int etc)		Stack

(#) Local variables

When we use the [string] we explicitly

These are stored on stack not on blockchain.

But there are some type in solidity which are by default stored into some memory i.e. on blockchain or on other kind of storage are defined. So you can't just simply write the variable name and initialize in their case

you need to use memory to get them enter into not into stack, not even contract storage
But in memory [another storage]

That doesn't happens with uint
it happens with string, struct etc,

(#) Pure keyword with function shows/tells that the function only contains local variable not static variables

(#) Local variable Properties (Imp)

1. Declared inside functions are kept on the stack, not on the storage/i.e. on contract.
2. It don't cost gas,
3. There are some types that reference the storage/contract storage by default.
   like → string, struct etc.
4. Memory keyword can't be used at contract level,

(#) function syntax

```
                      pure   ↑ already talked about it before      Type
function store() pure public returns(uint)
{                            ↓ set function created     ↓ * if default storage var then use type memory
   uint age=11;    //local
   return age;
}
```

* * * [returns (string memory)]

(#) function (setters & getters)

(#) Per everything in solidity works in the contract code

It state not static OK

(#) view vs pure in functions

Just remember one thing that when you want to read a particular state var inside a function then use view with the function.

But if you are not gonna read/write the state variable then use pure.

Pure can't be used when you need to read or write a state var by function.

{what if not even one's used [illegible] warning. But you don't use both of them in case of writing state var value cuz view only reads not write}

(#) Getter vs setter

⇒ finally. No need to create a function (getter, setter) in case of creating state public var.

→ You need to write only public if writing/changing the state var value. which requires cost also. But not getter function cost. *(with input)

```
function setter (uint newage) public
{
   age = newage;
}
```

functions properties

1. Setter function needs to be mined i.e. changing value of state var on ~~storage~~ storage/blockchain.

2. In case of public state var a getter function is automatically created.

3. By default variable visibility is private.

(#) constructor

Uses — most common ones
1. To initalize ~~[illegible]~~ state variable
2. To tell the owner of smart contract

Properties
1. Executed only once
2. You can create only one constructor and i.e. optional
3. A default constructor is created by compiler if there is no explicitly defined constructor.

(#) Integers (int) (contain both +ve -ve) (uint) (only +ve)

1. default int is int256 i.e. int / int256 same
2. can be written int8 and can be written on the basis of 8 intervals

By default value is (0)

1. (Same) uint256 Default
2. By default is 0 value

int8

Overflow is detected at compile the

Range of both

int8 = -128 to 127	uint8: 0 to 255
int16 = -32768 to 32767	uint = 0 to 65535
formula → $-2^{(n-1)}$ to $2^{(n-1)}-1$	0 to $(2^{n})-1$

(#) Overflow

In case of new solidity if a uint8 var = 255 //state

and you try to increase it by 1 (x++) by a function then it won't increase and will be reverted back to 255 (transaction revert)

But in old solidity it became zero means 0 money (lol)

(#) Array

(#) Fixed size
(#) dynamic array

Fixed Size

```
uint[4] public arr = [2,4,6,8];

function setter(uint index, uint value) public
{
    arr[index] = value;
}
```

(for users input)

Dynamic Array

```
uint[] public arr;    //dynamic Now

function pushElement(uint item) public
{
    arr.push(item);
}

function len() public view returns(uint)
{
    return arr.length;
}

function popElement() public
{
    arr.pop();
}
```

Mapping is solidity

Syntax → mapping(uint => string) public name;

(Key type, value type)

```
function setter(uint key, string memory
        value) public
{
    name[key] = value;
}
```

##

Mapping are always stored on storage always.

(#) Byte Array [This array is immutable]

1 byte = 8 bits.
1 hex digit = 4 bits

Everything stored in byte array will be in the form of hexadecimal digit

```
Contract Array
{
    bytes3 public b3;    // 3 bytes array
    bytes2 public b2;    // 2 bytes array

    function setter() public
    {
        b3 = 'abc';
        b2 = 'a';
    }
```

Property =>

1. Padding of 0 is added if lesser value than size.
2. Immutable.

~~(#) Array~~

By default bool type value is false

Structure in solidity

```
struct Student {
    uint roll;
    string name;
}

Contract demo
{
    Student public s1;

    // assigning values by constructor
    constructor(uint _roll, string memory _name)
    {
        s1.roll = _roll;
        s1.name = _name;
    }

    function change(uint _roll, string memory _name) public
    {
        // Very Imp // Creating new object and then assigning it to someone else
        Student memory new_student = Student({
            roll: _roll,
            name: _name
        });
        s1 = new_student;
    }
}
```

Justellig it you write

also this is fine
// student memory Person = Student({
}); // wide work without ...
student(...) Product = Student(

Important

```
struct student {
    string name;
    uint id;
}
```

//Two ways to quickly initialize a struct object

fastest/quicker ① `student memory obj = student("Rahul", 25);`

for using with user input function ②

function ~~somethig~~ (string memory val, uint id (Passed argument for value) {
    student memory obj = student({
        name: val,
        id: _id
    });
}

(#) Enumeration in General (Not just in solidity)

An enumeration is a user-defined data type that consists of integral constants. To define an enumeration keyword enum is used.

example (This is type i.e. are actually)

```
enum season { spring, summer, autumn, winter };
                0       1        2       3
```

You can give customize values to them also by assigning values to them explicitly.

Also, to create variable use

```
enum boolean { false, true };
```

Now to create variables

ways

①
```
enum boolean {
    false, true
} check;
```

Also ②
```
enum week { Sunday, Monday, Tue ----, Saturday }
int main() {
    week today; var.
```

Example: Code

```
#include <iostream>
using ____

enum week { Sunday, ----, Saturday };

int main() {
    week today;
    today = Wednesday;
    cout << today << endl;
    return 0;
}
```

Output: ④

~~Now for solidity~~

~~Solidity example~~

Just new enum are for better readability

1-3

Storage vs memory

```
contract demo {
    string[] public student = ['Ravi','Rita','Aman'];

    function mem() public view
    {
        string[] memory s1 = student;
        s1[0] = 'Akash';
    }

    function sto() public
    {
        string[] storage s1 = student;
        s1[0] = 'Akash';
    }
}
```

Imp

Global variables

Example [look in documentation for more.]

```
Contract demo
{
    function getter() public view returns (uint block.No, uint timestamp, address msg.sender)
    {
        return (block.number, block.timestamp, msg.sender)
    }
```

(block.number → when the block was mined; msg.sender → who sends the amount / who calls)

Global variables. Who sends the request to mine a block the person address.

V.V.V.Imp

Contract Balance & Payable

Payable is keyword which is only used with user's addresses which are going to be paid or receive amount.

Note: Payable is not used with the account add who they will send money

Code

```
pragma solidity ^0.8.6;

contract Pay
{
    address payable user = payable(0xCA---);

    function payether() public payable
    {
    }

    function getbalance() public view returns (uint)
    {
        return address(this).balance
    }

    function sendetherAccount() public
    {
        user.transfer(1 ether);
    }
}
```

[0xCA--] has 100 ethers already, who receive (letter)

This is simply used to add more ether from the sender's account into contract so that they can be send to this one here.

lets suppose we are adding ether from (0x7c...)

address(this) → inbuilt

will return the balance of this i.e. contractor's balance

letsay 2 ether that i.e. ether amount send from [0x72--] with its msg.value

will send these 2 ether over to the [0x6A] from this acc

[# Visibility in Solidity]

Accessible in

Public	Private	Internal	External
Outside	X	X	Outside
Within	Within the contract	Within	X
Derived	X	Derived	Derived
Others	X	X	Others

→ Only means communication b/w two separate contracts # Not inherited strictly

To inherit a contract use is

```
Contract A {
}

Contract B is A {

}
```

Other is by creating objects
(creating contract A)

```
Contract C
{
  A obj = new A();
  uint public cn = obj.f1();
}
```

Const variables takes less gases

Events in solidity

One very important point is when you don't use (pure & view) in a function then remember they never return anything.

Events in Solidity

~~This/event This/pen/is/still~~

Now as you already know event works when something defined comes into work / specific play.

So just similar to that almost here —

(4) indexed keyword is used in event for indexing and events can only take upto 3 arguments.

Events are something which are used to deploy some data on blockchain when a specific function is called.

and that is done by emit() inside the function.

That emit works as a values passed to the event function itself. which works/can be implemented when that function is working and emit is started automatically.

Everything that is stored on blockchain can be seen under Transaction logs section.

Example

```
Contract Event_wine {

    event balance(address account,
                  string message, uint val);
    function setdata(uint _val) public {
        emit balance(msg.sender, "hello",
                     97);
    }
}
```

(call him and deploy on blockchain.)

Transaction log is a special data structure in the blockchain where data is stored.

They cannot be accessed by smart contracts and give information about what goes on in transaction and blocks. It's their inaccessibility to smart contracts that makes them cheaper to emit.

[V.V.V. Important]

Types of location of data in solidity

(1) Storage → Variable is a state variable and is stored on the blockchain.

(2) Memory → Variable is in memory and it exists while a function is being called.

(3) calldata → Special data location that contain function arguments, only available for external functions.

Call data is something which is used on the place of memory, so that the gas fee can be saved i.e → gas fees.

But why it takes less gas.

As the variable which are calldata variable can not be changed after assigning.

So that's why they require less gas fee.

Also while passing values from a calldata value containing function to an another calldata function (i.e → Calldata → calldata) with it doesn't require extra memory. i.e → usually when you pass arguments or a new memory is allocated to a variable passed in function.

But when calldata → calldata - No extra location space is taken. The data is used.

Using memory and calldata etc they don't need store values on blockchain as like all state variables.
But yes both takes more gas fees.

But its like this

state/storage variables > Memory variables > calldata var.
(Imp)

for example

```
contract Datafoc {
    function Memory(string memory str, uint[] memory arr) public {   // we used M capital so that it won't work as a keyword.
        getInMem(arr);    // Way to call functions.
        // getInCall(arr)   // will produce error
        # // our calling or passing value from memory to calldata is not valid #
    }

    function Calldata(string calldata str, uint[] calldata arr) public {   // capital, so no keyword working
        getInMem(arr);    // Both are valid & possible
        getInCall(arr);   // Here the same location will passed
    }

    function getInMem(uint[] memory arr) public {
    }

    function getInCall(uint[] calldata arr) public {
        // and after getting value from calldata (stored in memory based) no new memory is created to pay values
    }
}
```

Ternary operator in Solidity

similar as in C++;

Rem, In do while first do is executed then while is checked or condition is checked.

Continue & Break are also there.

// # Require in Solidity //

[Require, Revert, assert]

Used in error handling.

Its task
1. Input Validation.
2. Access Control.

Advantages
1. can be used to revert back the leftover gas which is still not used. But the transaction is failed. So to revert it to the user.
2. to revert/reset the values of state variables.

Require in Solidity

Syntax for Require

(#) You can use require without error msg

```
function CheckRequire(uint num) public {
    require(num == 2, "Here Error MSG");   (else Print error)
}
```
(The condition if true then move forward)

That an idea you can even check the owner who is calling the function

```
require(address == msg.sender, "Not Owner");
```

↓ it checked the owner, move forward in function, else → it stopped & give this error.

Revert & Assert

Revert is almost similar to the require. You can say the diff is that you can have custom errors in revert. But require not gives you this functionality.

Example

```
contract Mycont {

address public owner = msg.sender;
uint public age = 25;

error throwerror(string, address);

function checkRevert(uint _x) public {
    age = age + 5;
    // must need a condi explicitly
    if (_x < 2) {
        revert throwerror("value not valid", msg.sender);
    }
}
```

→ also possible this also
revert("This is error");

Assigment 3/10

when revert to initial state happens then it revert whatever the change is happened to state variable throughout the function to revert back to the state.

Assert

1. Used for Bug check.
2. " " Security.

Mostly assert is used for Bug check.

```
function checkOwnership() public view {
    assert(owner == 0x787-----);
}
```

only fimrly

fallback & receive

fallback is more superior than receive

fallback is capable of doing anything by receive. But receive is not enough capable.

// About fallback

1. It is executed when a non-existent function is called on the contract.
2. It is required to be marked external.
3. It has no name.
4. It has no argument.
5. It takes two things data & ether.
6. It doesn't return anything.
7. It can be defined one per contract.
8. If not marked payable, it will throw exception if contract receives ether.
9. Its main use is to directly send the ETH to contract.

Doubt # The ether that are received, are stored where
In block or smart contract.

(through function Payable etc., fallback/receive
Abe bad bhi to sab contract hi BC par jab execute hota hai I mean ether ... the block so what do you expect now
msg.sender, block.number, return, blocknumber
it need more info)

Difference b/w Receive & Payable
① Receive needs to be payable.
② It only receives ether not data.
Otherwise its completely similar to fallback.

1 ether = 10^{18} wei

Example

① contract Mycont {

(will receive ether and data and add to block.)

```
fallback external payable {

}

receive() external payable {

}

function checkbal() public view
    returns(uint) {
    return address(this).balance;
}
}
```

(If both are in the code then if (data + ether) give with calldata input then fallback work.
if only ether added as input calldata the receive will work.
otherwise in simple you know receive can't take data and is simple.)

You can check out their working when by adding emit so that when fallback/receive works event will execute and tell which one was working for receiving.

Important Point

→ msg.sender is the address of the contract caller.

→ address(this) is the address of the smart contract itself.
i.e. you use [address(this).balance;]

Also
(msg.sender) In it the owner can be a contract sender,
also it changes when a function is called every time you got it truly not.

(Avoid it) (tx.origin) The owner can never be a contract
It is actually the user who started the transaction

for example:
In a simple call chain A→B→C→D
msg.sender will be C and
tx.origin will be A.

Very Important Point
(Def) declare
* When you create an address as Payable then while initializing anywhere or passing value to it use Payable typecasting as it only accept payable(var) now.
Rem: address & address payable are all different

`msg.value` is a member of the msg (message) object when sending transactions on the Ethereum Networks.

`msg.value` contains the amount of (ether) wei i.e → (1 ether = 1e18) sent in the transaction

** HardHAT **

#) NPM is basically a Node package Manager (Million of packages)
which manages all the Packages which you can install also on your machine for test code.
You can't create things from basic right so here is the solution Million solution for all problem you can think of better search once if need more info.

Some Commands

```
Truffle compile
Truffle migrate
Truffle migrate --reset
Truffle migrate --network ganache
```

Integrating frontend with Project

To start react → [npm start] || [Truffle migrate for migration]

BTW W3.JS is a libraries which allows to create UI to communicate