Block Chain

Blockchain is a decentralized and distributed digital ledger that records transactions across a network of computers. Each transaction is stored in a "block," and these blocks are linked together in chronological order, forming a "chain."

Bitcoin is a one of the first protocols to utilize the revolutionary blockchain technology

Peer-to-peer (P2P) means that transactions occur directly between users without involving intermediaries like banks or payment processors.

Decentralized Network means there is no single authority (like a government or bank) controlling it. It provides a way for people to transfer value (money) securely and transparently without needing trust in a middleman. This is achieved through the blockchain, where every transaction is recorded and visible to all participants in the network.

Bitcoin is only for transactions but **Ethereum** was built for transactions, organizations, and agreements without a centralized intermediary

Smart Contracts

💡

1. Stack Overflow
2. Peeranha.io
3. Speed Run Ethereum
4. Web3education.dev

Smart Contracts - Trust Unbreakable Agreements i.e. there are agreements between 2 people. It is an agreement, contract or set of instructions that deployed on a decentralized blockchain

In the 80s and 90s, McDonald’s Monopoly game promised customers a chance to win money through game cards obtained with purchases. However, it turned out that the game was rigged by insiders who manipulated the system for their gain. Essentially, McDonald’s failed to keep its promise.

With smart contracts, we can eliminate the need for trust. A smart contract is an agreement or a set of instructions that are deployed on a decentralized blockchain. Once deployed, it cannot be altered, it automatically executes, and everyone can see its terms.

Imagine if McDonald’s Monopoly game was operated on a blockchain through a smart contract. The fraudulent activities would have been impossible due to the immutable, decentralized, and transparent nature of smart contracts.

This technology holds the potential to revolutionize industries and everyday agreements by ensuring honesty and fairness.

The Oracle Problem

Smart contracts face a significant limitation they cannot interact with or access data from the real world. Blockchains are deterministic system i.e. state of the blockchain at any given point in time is predictable and consistent based on the inputs and rules of the system.

To make smart contracts more useful and capable of handling real-world data, they need external data and computation.

Oracles serve this purpose. They are devices or services that provide data to blockchains or run external computation. To maintain decentralization, it's necessary to use a decentralized Oracle network rather than relying on a single source. This combination of on-chain logic with off-chain data leads to `hybrid smart contracts`

Chainlink is a popular decentralized Oracle network that enables smart contracts to access external data and computation. Chainlink is also blockchain agnostic - so it's going to work with any chain out there.

Security & Immutability

Once a smart contract is deployed, it cannot be altered or tampered with. This immutability ensures that the terms of the contract are set in stone. This is a stark contrast to centralized systems where a server or database can be hacked, and data can be altered. The decentralized nature of blockchain makes hacking nearly impossible since an attacker would have to take control of more than half the nodes, which is significantly more challenging than compromising a single centralized server.

Applications of Smart Contracts

Decentralized Finance (DeFi) - allows users to engage with financial markets without relying on centralized intermediaries

Decentralized Autonomous Organizations (DAOs) - DAOs are governed entirely by smart contracts and operate in a decentralized manner. This structure offers benefits such as transparent governance, efficient engagement, and clear rules.

DAOs are an evolution in politics and governance, and we will cover how to build and work with DAOs in future lessons.

Non-Fungible Tokens (NFTs) - NFTs, or Non-Fungible Tokens, can be thought of as digital art or unique assets. NFTs have created new avenues for artists and creators to monetize their work.

Decentralized Application - Combination of multiple smart contracts in an application

Transaction Details

Taking a brief look at some of the details of our transaction on Etherscan, we're given a lot of insight. Understanding these properties is a fundamental part of being a blockchain developer. Some of the basic details include:

- Transaction Hash - This is a unique identifier for our transaction
- From - The originating address of the transaction request
- To - The address a transaction was sent to
- Value - Any funds included with the transaction
- Gas - The cost of the transaction to execute - Each node referred to as miner or validator depending on the type of network gets paid a little bit of native currency of the blockchain. Gas is a measurement of computational measurement, more complex the computational thing more will be the gas

The transaction fee is the amount rewarded to the block producer for processing the transaction. It is paid in Ether or GWei. The gas price, also defined in either Ether or GWei, is the cost per unit of gas specified for the transaction. The higher the gas price, the greater the chance of the transaction being included in a block.

The total transaction fee can be calculated by multiplying the gas used with the gas price in Ether (not GWei). Therefore, `Transaction fee = gasPrice * gasUsed`.

Blockchains are run by a group of different nodes, sometimes referred to as miners or validators, depending on the network. These miners get incentivized for

running the blockchain by earning a fraction of the native blockchain currency for processing transactions. For instance, Ethereum miners get paid in Ether, while those in Polygon get rewarded in MATIC, the native token of Polygon. This remuneration encourages people to continue running these nodes.

In the context of transactions, gas signifies a unit of computational complexity.

The higher a transaction's complexity, the more gas it requires. For instance, common transactions like sending Ether are less complex and require relatively small amounts of gas. However, more sophisticated transactions like mining an NFT, deploying a smart contract, or depositing funds into a DeFi protocol, demand more gas due to their complexity.

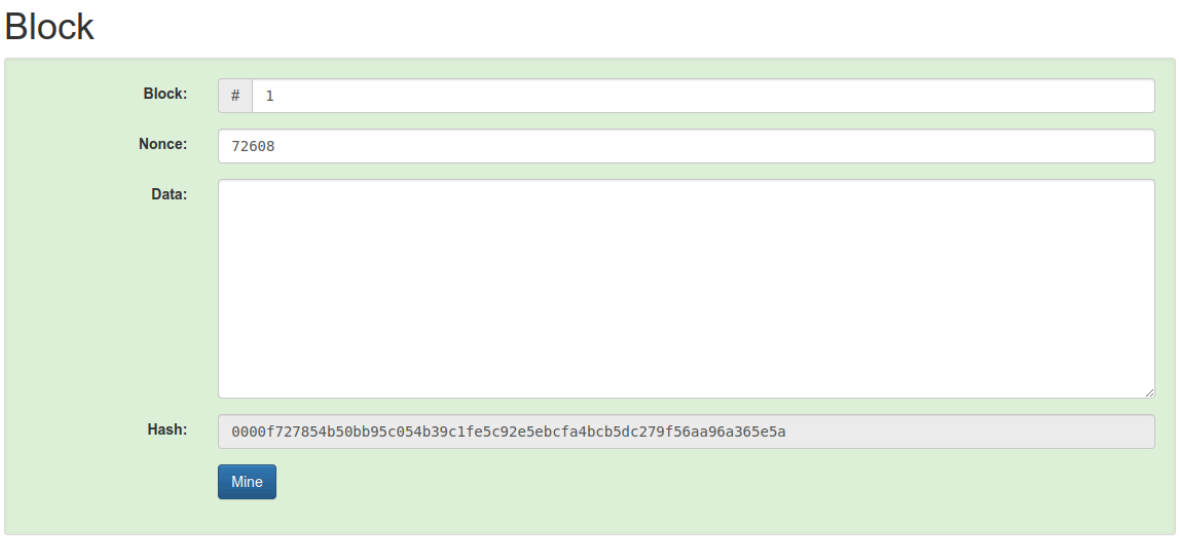
Block in a Blockchain

1. Block - Unit of data which contains collection of information, such as transactions or records, and metadata used to ensure the integrity and security of the block.
2. Block Number (#) - represents the position of the block in the chain. Block Number 1 is known as genesis block
3. Nonce - The **nonce** is a number that miners manipulate to solve the cryptographic puzzle required to validate the block.
4. Data - information stored inside the block
5. Hash - cryptographic representation of the entire block (including the block number, nonce, data, and other metadata). It acts as the block’s unique identifier.

Block:

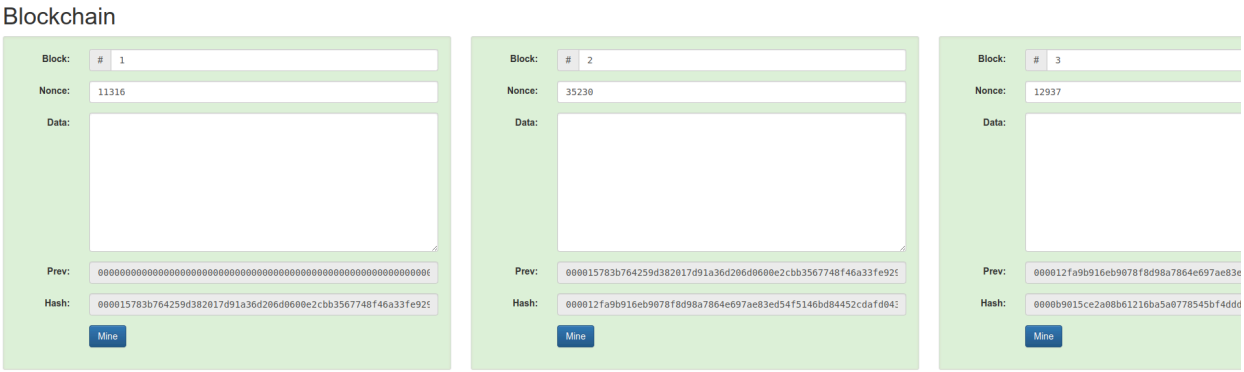
A block is a fundamental unit in a blockchain, containing a collection of data such as transactions or records. Each block also includes metadata, such as a timestamp and a cryptographic hash of the previous block. The hash ensures the

integrity of the block and links it to the chain, making it tamper-evident. A block's unique hash is generated based on its content, and even a small change in the data will result in a completely different hash.



Blockchain:

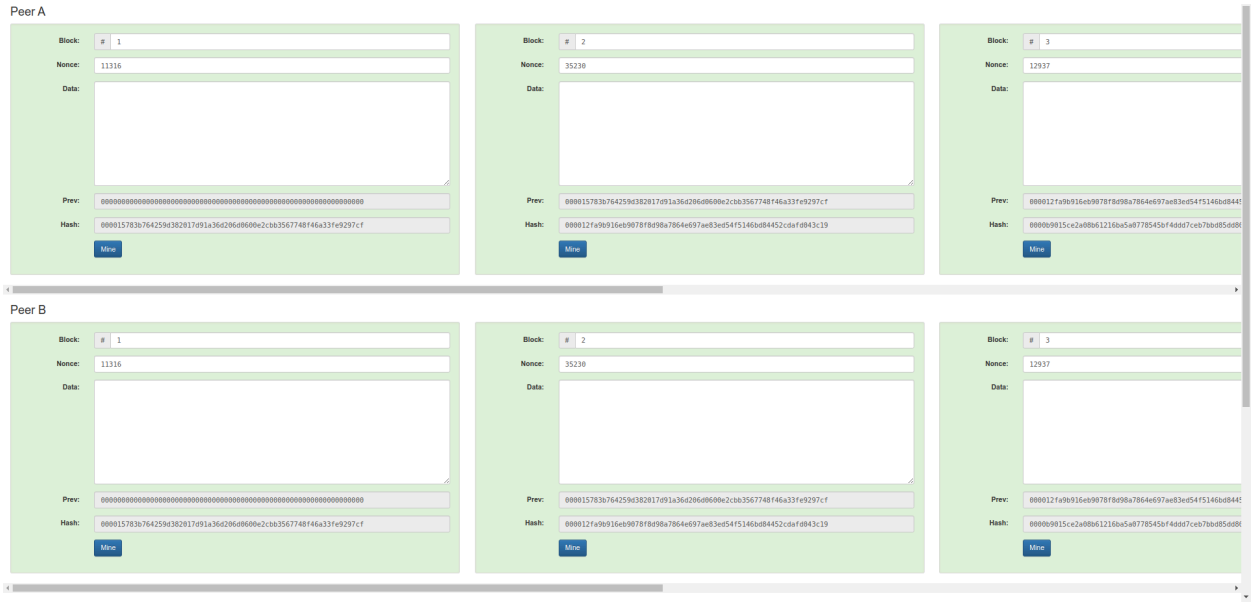
A blockchain is a decentralized ledger that organizes data into blocks, with each block linked to the previous one via cryptographic hashes. This chain of blocks ensures the security and immutability of the data, as altering one block would require altering all subsequent blocks, which is computationally infeasible. A blockchain operates across a distributed network of nodes, ensuring that no single entity has control over the data.



In this if we change data in block 2 then it will break the blockchain as the hash will change and we need to mine again, hash wont contain 0000 in the beginning (specific to a blockchain) and thus break the chain...

Distributed:

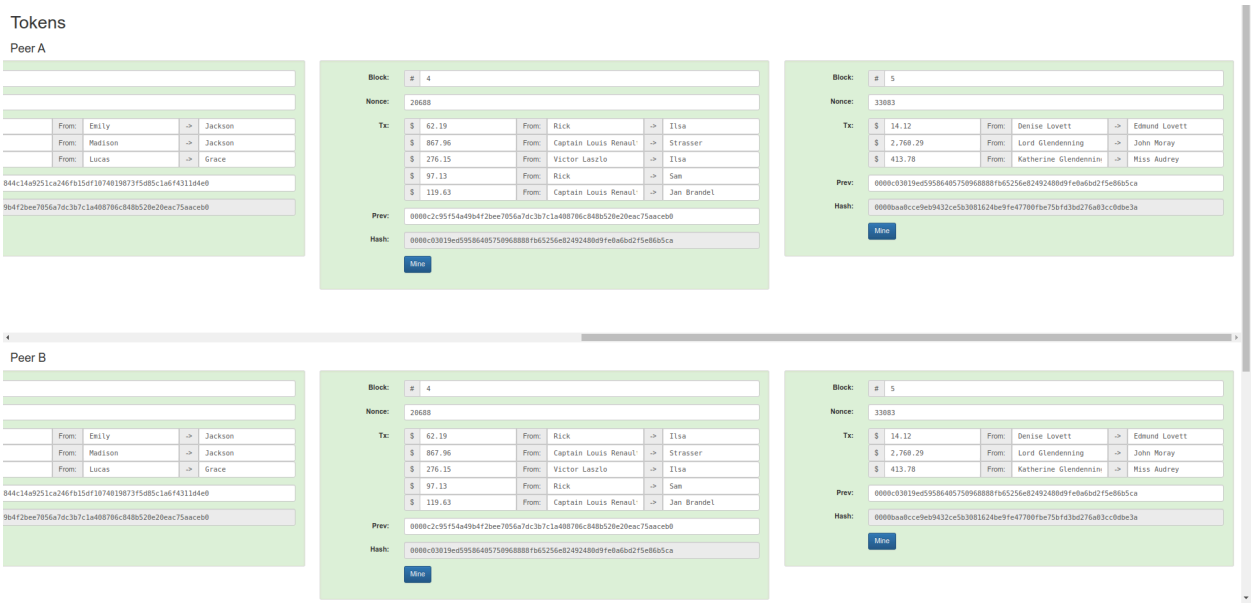
In the context of blockchain, "distributed" refers to the way the blockchain operates across a network of computers (nodes) rather than being stored and managed by a central authority. Each node maintains a copy of the entire blockchain, and all nodes work together to validate and agree on the state of the ledger. This distributed nature enhances security, transparency, and fault tolerance, as there is no single point of failure



There are multiple peers like A, B, C, ...
now for the end we match the hash like hash of end node of A, B, C like if A, C matches and B doesn't match then the node is considered to be malicious

Token:

A token is a digital representation of value or utility that exists on a blockchain. Tokens can represent various assets, such as currency, ownership rights, or access to services. They are created and managed through smart contracts and can be transferred between users on the blockchain. Tokens can be categorized as fungible (e.g., cryptocurrencies like Bitcoin) or non-fungible (e.g., NFTs). They play a crucial role in enabling decentralized applications and economies.



You can see hash of all the nodes are same, each node has track of all the transactions happening

Instead of the names we have address of the wallet

usually what is happening we sign the transaction with private key and obtain message that can be verified by anyone using our public key which can be distributed in the public

L2 (Layer 2) in blockchain

Refers to a set of scaling solutions built on Layer 1 (Bitcoin, Etherium) to enhance its performance, scalability, and usability. These solutions aim to process transactions more efficiently, reducing the burden on the main blockchain while maintaining security and decentralization.

Reduces gas fees by aggregating multiple transactions and submitting them as one to the Layer 1 chain. Offloads transaction processing, leading to quicker confirmation times. Leverages the security of the underlying Layer 1 blockchain.

Rollups:

- Types: Optimistic Rollups, ZK-Rollups (Zero-Knowledge Rollups).
- Process multiple transactions off-chain and submit a single compressed proof to the Layer 1 chain.

EIP 1559 (Ethereum Improvement Proposal)

Previously, Ethereum used an auction-based fee system where users bid for block space, leading to unpredictable gas prices. EIP-1559 replaced this with a base fee model.

Base Fee: A mandatory fee, determined algorithmically, based on network congestion.
Priority Tip: Users can add a small tip to incentivize miners for faster transaction processing.

Fee Burning Mechanism: The base fee paid in Ether (ETH) is burned (permanently removed from circulation), reducing the total supply of ETH over time.

This creates a deflationary pressure on ETH, potentially increasing its value.

Dynamic Block Sizes: Blocks can expand in size during periods of high demand (up to double their normal capacity), making fees more predictable and stabilizing the networ

Consensus

- Consensus is the mechanism used to reach agreement on the state/value in a blockchain
- It's particularly crucial in decentralized systems
- Uses majority rule principle - if one node differs from others, majority prevails

Consensus Protocol Components:

Two main pieces:
a) Chain Selection Algorithm
b) Sybil Resistance Mechanism

- Examples include Proof of Work (PoW) and Proof of Stake (PoS)

Ethereum's Historical Change:

- Significant event: "The Merge" in September 2022
- Transitioned from PoW to PoS
- Was a major network upgrade watched by many enthusiasts

Sybil Resistance:

- Definition: Blockchain's defense against users creating multiple fake identities
- Purpose: Prevents gaining disproportionate influence over the system
- Two main types discussed: PoW and PoS

Proof of Work (PoW):

- Requires computationally expensive mining process
- Each node must solve a complex mathematical puzzle
- Block time can be adjusted by changing puzzle difficulty
- Used by Bitcoin and previously by Ethereum
- Energy-intensive but effective against Sybil attacks

Nakamoto Consensus:

- Bitcoin's specific consensus algorithm
- Combines:
  - PoW for Sybil resistance
  - Longest chain rule for chain selection
- Common misconception: People often incorrectly refer to Bitcoin's consensus as just PoW

Transaction Fees and Rewards:

- Two types of rewards:
  a) Transaction fees (paid by users)
  b) Block rewards (issued by protocol)
- In PoW: Paid to miners
- In PoS: Paid to validators

Attack Types:

a) Sybil Attack:

- Creating multiple fake identities
- Prevented by PoW/PoS mechanisms

b) 51% Attack:

- When one entity controls majority of network
- Can manipulate the longest chain
- Harder on larger networks
- Has occurred on smaller chains like Ethereum Classic

Proof of Stake (PoS):

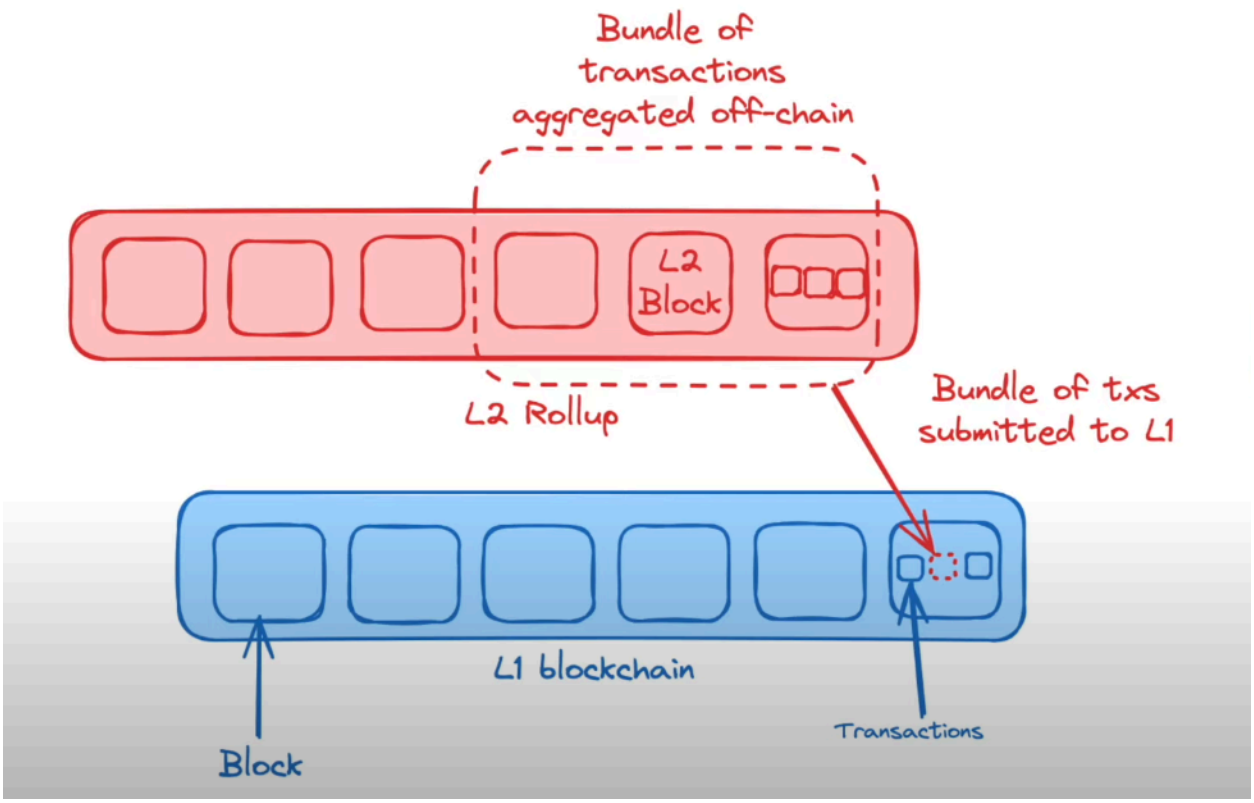
- Nodes stake collateral as security
- Validators randomly chosen to propose blocks
- More environmentally friendly than PoW
- Used by: Avalanche, Solana, Polygon, Polkadot
- Slashing mechanism for dishonest behavior
- Slightly less decentralized due to upfront costs

Layer 1 vs Layer 2:

- Layer 1: Base blockchain implementations (Bitcoin, Ethereum, Avalanche)
- Layer 2: Applications built on top of Layer 1
  - Examples: Chainlink, Arbitrum, Optimism
  - Rollups: Special L2s that inherit security from L1
  - Different from sidechains (which have their own security)

Rollups

Scaling solutions that increases the number of transactions on L1



Scalability:

- Major challenge in blockchain
- Solutions include sharding and rollups
- Limited block space leads to high gas prices
- Layer 2 solutions help address these issues

Blockchain Trilemma

The blockchain trilemma is a concept that highlights the challenge of achieving three key properties in blockchain systems simultaneously: decentralization, scalability, and security. Most blockchains prioritize two at the expense of the third, balancing these factors based on their use cases and goals.

1. **Decentralization:**

- Refers to the distribution of control across a network of nodes rather than relying on a single authority.
- Ensures trustlessness, transparency, and censorship resistance.
- Highly decentralized systems may face slower transaction speeds or reduced scalability due to the need for widespread consensus.

2. **Scalability:**
    - The ability of a blockchain to handle a large number of transactions efficiently without delays or high costs.
    - Scalability is often hindered in decentralized systems because consensus across many nodes can slow down the network.
    - Layer 2 solutions, like rollups, attempt to address this issue without sacrificing decentralization.
3. **Security:**
    - Ensures the network is resistant to attacks, fraud, and tampering.
    - A secure blockchain prevents unauthorized changes and protects user data and funds.
    - Prioritizing security often requires additional computational power and resources, which can limit scalability.

Trade-offs:

- Blockchain developers face trade-offs when designing systems. For example:
    - Bitcoin emphasizes security and decentralization but sacrifices scalability.
    - Layer 2 solutions, like Optimistic Rollups and ZK-Rollups, aim to improve scalability while leveraging the security of Layer 1.

The blockchain trilemma illustrates the difficulty of creating a perfect blockchain system, but ongoing innovations, such as sharding and hybrid models, continue to push the boundaries of what is achievable.

An operator is responsible for processing the transactions, job is to pick up the transactions and join with other people transactions and then bundle them and submit to L1, if Ethereum rolls back so is L2 thus have security of Ethereum, gas cost is reduced, thus leads to mass adoption

Optimistic Rollup vs ZK Rollup

- **Optimistic Rollup:**
  - Assumes transactions are valid by default and only checks validity if challenged.
  - Relies on fraud proofs to detect and penalize invalid transactions.
  - Longer finality time due to potential dispute resolution period.
  - More efficient for simple transactions but less secure without active monitoring.
  - Examples: Optimism, Arbitrum.
- **ZK Rollup (Zero-Knowledge Rollup):**
  - Uses zero-knowledge proofs to verify transaction validity off-chain.
  - Provides cryptographic proof to Layer 1, ensuring validity without the need for challenges.
  - Shorter finality time as there is no dispute period.
  - More complex to implement but provides stronger security guarantees.
  - Examples: zkSync, StarkNet.

Solidity Basics

```
pragma solidity 0.8.18;
// or
pragma solidity ^0.8.18;
// or
pragma solidity >=0.8.18 <0.9.0;
```

```
// On the top just add (in comment form only)
// SPDX-License-Identifier: MIT
```

```
contract SimpleStorage { // contract acts as a class

}


// Basic Types
bool
uint
int
address
bytes

// Variables can be formed like
bool hasFavNumber = true;
uint256 favNumber = 223;

// there are multiple like uint8, uint 48 ......
// similarly
int256 favNumberInt = 246;
string favNumberString = "234"

address adx = 0xf29DBB107D7b2b75B4d943CBFfA3272dD471cB14;

bytes32 favBytes32 = "cat";
// there are multiple bytes like bytes8 ....
```

```
contract SimpleStorage {
    uint256 favNumber; // by default it is 0

    function store(inuint256 _favNumber){
```

```
        favNumber = _favNumber;
    }
}

// The visibility of favNumber is defaulted to internal
// uint256 favNumber ====== uint256 internal favNumber

// if we need to get value we can have
uint256 public favNumber;

// Available visibility specifiers
public, private, external, internal

// public → visible externally and internally (good for storage/state variables)
// private → only visible in current contract
// external → only visible externally (only for functions)
// internal → only visible internally
```

Generally it takes around 21000 gas to send ether between 2 accounts

```
contract SimpleStorage {
    uint256 favNumber; // by default it is 0

    function store(inuint256 _favNumber) public {
        favNumber = _favNumber;
    }

    function retrieve() public view returns(uint256) {
        return favNumber;
    }
```

```
}

// view thing disallows any modification of state in the function
```

calling a view of pure function actually does cost gas only when a gas cost transaction is calling out like

`function store(uint256 _favNumber){`

`favNumber = _favNumber`

`retrieve()`

`}`

then retrieve will cost gas

```
// Array or struct

uint256[] listNum;

struct Person {
    uint256 favNumber;
    string name;
}

Person public pat1 = Perosn({
    favNumber: 7,
    name: "Pat"
})

// or

Person[] public listOfPeople; // Dynamic Array
Person[2] public listOfPeople_two; // Static Array
```

```
function addPerson(string memory _name, uint256 _favNumber) public {
    Person newPerson = Person(_favNumber, name);
    listOfPeople.push(newPerson);
}

// The EVM can read and write to several places
Write and Read:
1. Stack
2. Memory
3. Storage
4. Calldata

Write (not read):
1. Logs

Read (not write):
1. Chain Data

// calldata, memory, storage
// calldata and memory are temporary variables
// difference between both of them is memory variable can be changed
// if we pass calldata instead of memory we can not modify the variable

// we need to put memory and other keywords like behind which is array
// or struct or mapping
// it wont work with int, but works with string since string are array
// of bytes

// Storage
// we cant use storage for function parameters since solidity knows
// they are temporary
```

```
// MAPPING

mapping(string ⇒ uint256) public nameFavNumber;
```

```
function addPerson(string memory _name, uint256 _favNum) public {
    nameFavNumber[_name] = _favNum
}

// default value of all the keys is 0 , if key doesnt exist
// then also its value will come 0
```

💡 Whenever we compile the code, it gets compile down to something called the evm or the ethereum virtual machine

RECAP

the first thing you want to do in any smart contract or solidity code that you write be sure to write the version that you want to work with and above the version be sure to add the spdx license identifier if you're not sure what version to use for now just default to MIT then you have to create your contract object and name your contract a contract is similar to a class in other programming languages anything inside of the curly brackets for the contract is part of that contract there are many different types in solidity like uint256 string Boolean int etc if we want to create our own type in solidity we can use what's called a struct you can create arrays or lists in solidity you can create mappings or dictionaries or hash tables in solidity where if you give it a key it'll spit out the variable associated with that specific key we can create functions in solidity that modify the state of the blockchain we can also create functions in solidity that don't modify the state of the blockchain view and pure functions don't modify the state of the blockchain we can also specify different data locations in our parameters of our functions but we can only do that for special types like strings structs and arrays call data and memory mean that that variable is only temporary and will only exist for the duration of the function call Storage variables are permanent and stay in the contract forever function parameters can't be storage variables since they're only storage variables since they're only storage variables since they're only going to exist for the duration of the function call whenever we hit compile in our smart contracts it actually compiles our solidity code down to evm compatible our solidity code down to evm compatible bite code or machine readable code

💡 Contracts can interact with each other seamlessly this is one of the main features of blockchain also known as composibility. Used in Defi

```
contract StorageFactory {
    SimpleStorage public simpleStorage;

    function createSimpleStorageContract() public {
        simpleStorage = new SimpleStorage();
    }
}
```

```
// Inheritance

import { SimpleStorage } from "./SimpleStorage.sol"

contract AddFiveToStorage is SimpleStorage {
    function sayHello() public pure returns(string memory){
        return "Hello";
    }

    // override and virtual
    function store(uint256 _newNumber) public override {
        myFavNumber = _newNumber + 5;
    }

    // we need to add the keyword virtual to that function we
    // need to override
}
```

FUND ME

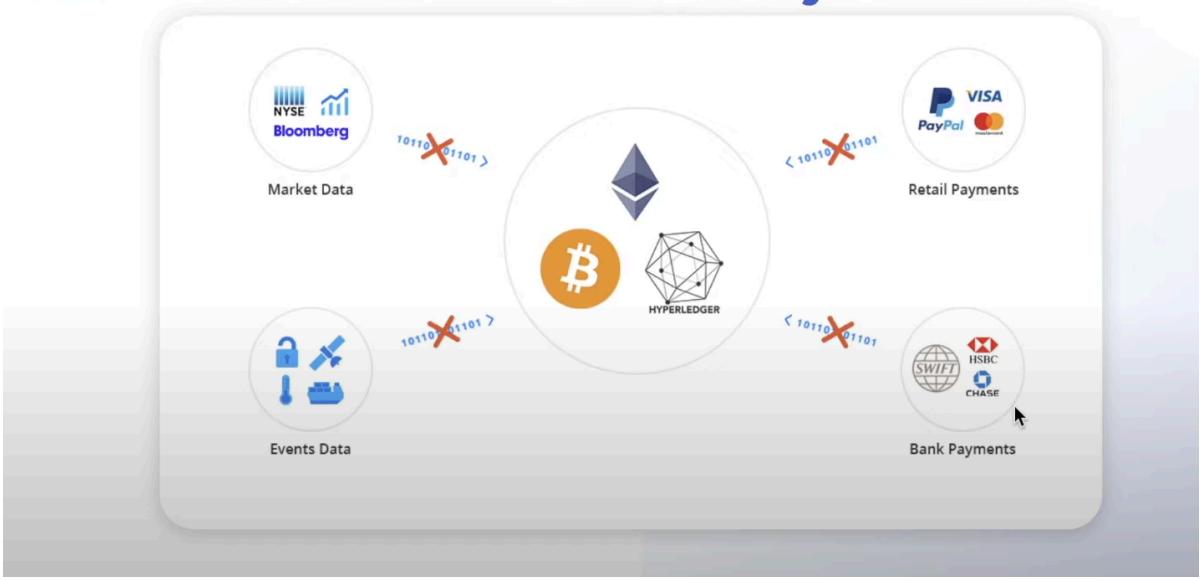
```
pragma solidity ^0.8.18

contract FundMe {
```

```
    function fund() public payable {
        // we can access the amount of transaction using one of the globals in solid
        require(msg.value > 1e18); // 1e18 = 1 ETH

        // revert
        // undo any actions that have been done, and send the remaining gas back



        // for e.g.
        // if someone sends below 1eth and there some computation above
        // then it will be reverted back

        // remaining gas will be given back to the owner
    }
}

// payable keyword makes the function payable
```

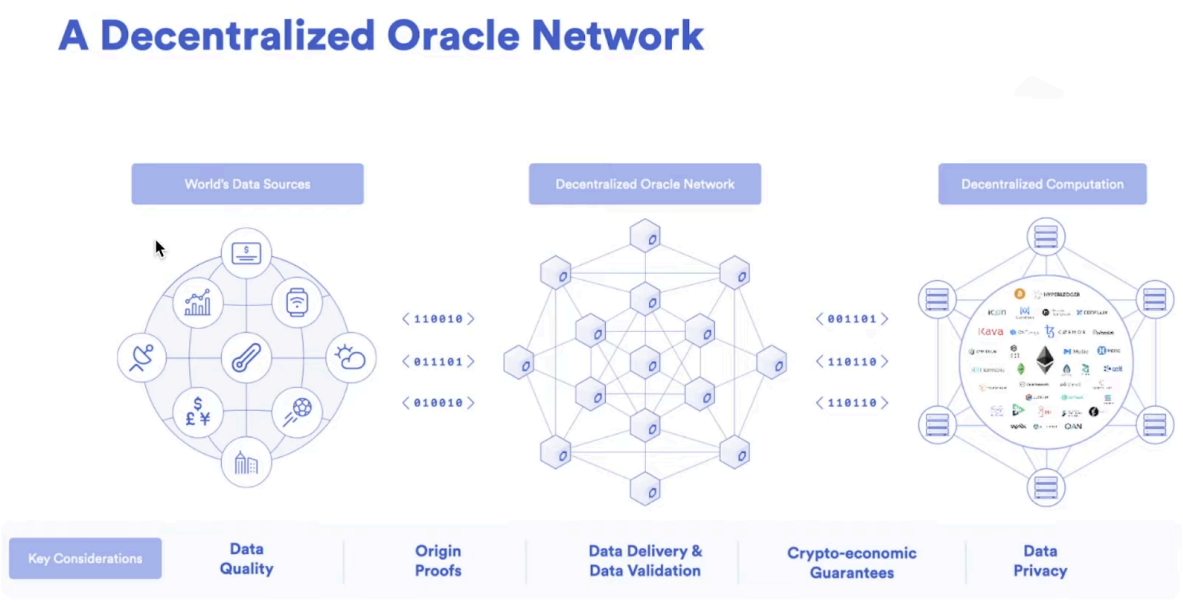
```
// Now if we want to take the amount in terms of dollars so we
// need to know the price but fundamentally blockchains dont have
// information outside, so we use oracles (hybrid smart contract)

// blockchains are determinsitic i.e. they cant actually interact
// with real world data and events
```

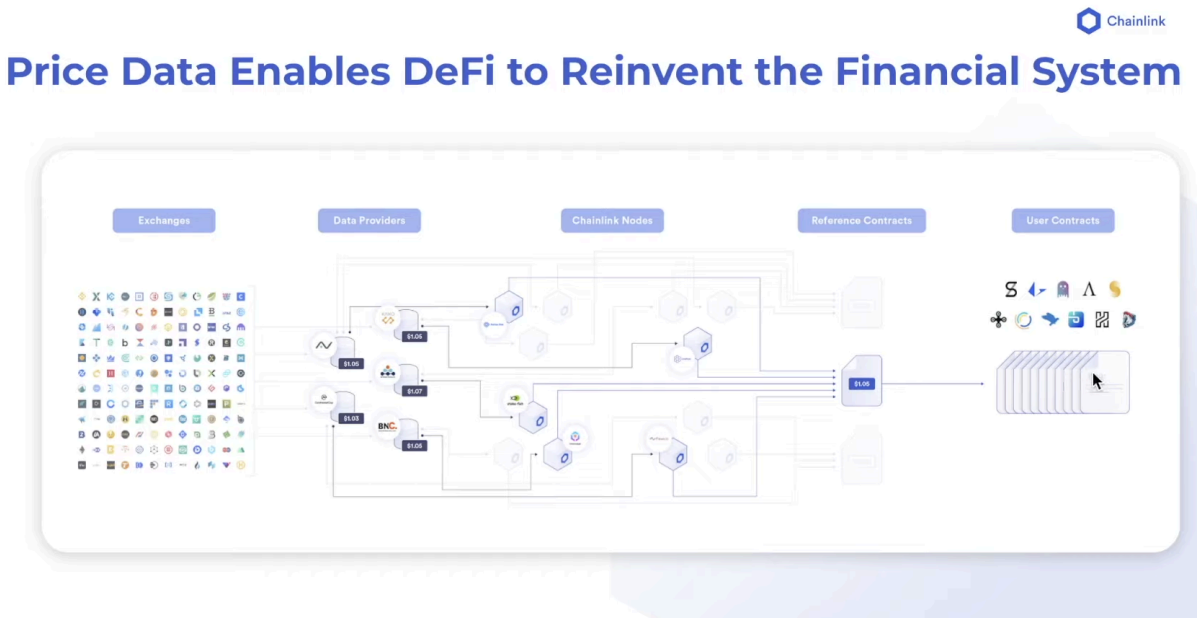
1. THE ORACLE PROBLEM

The Smart Contract Connectivity Problem



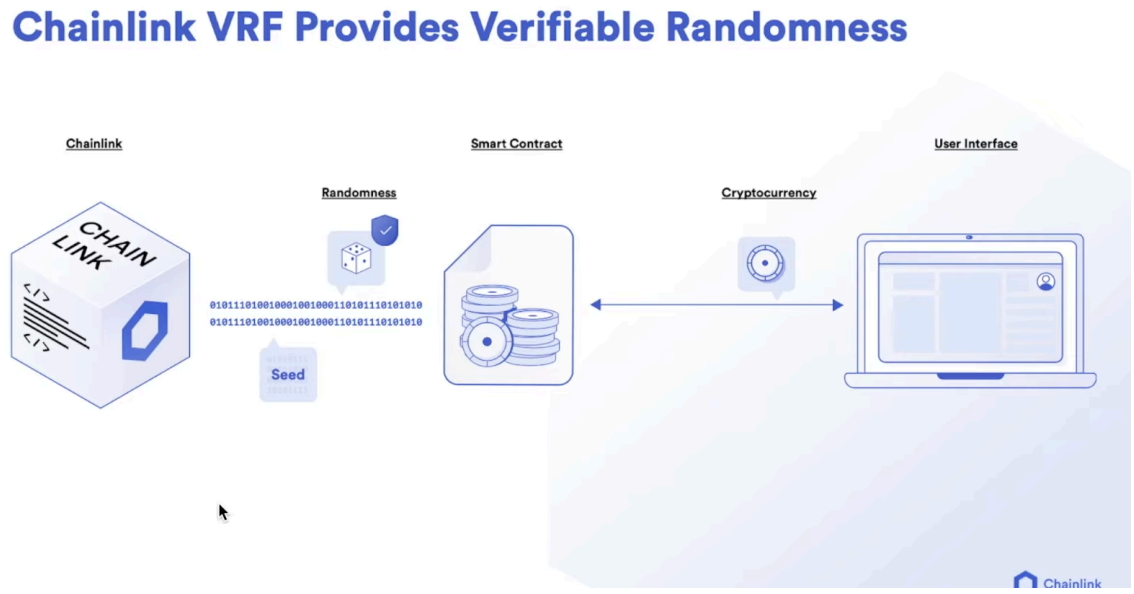

```
// Chainlink and blockchain oracle comes into play
// Centralized Node are a point of failure, like if theres one
// then it could be hacked or being a centralized if its gone
// down everything stops
```






```
// how is stuff getting done
// each node gets data from different exchanges and data providers
// and brings that data through network of decentralized chain link
// nodes
// then all these nodes using median to figure out what the actual price of asset i
// then all of this is stored in a single transaction called a
// reference contract or data contract that other smart contract uses
```

// Chainlink VRF (Verifiable Randomness Function)



Chainlink Keepers

Decentralized Event Driven Execution - so after specified interval of time they perform an activity like in a week do something or if the price of some asset hits some number, whatever event these thing listen whenever there is a trigger

Checkout Chainlink Functions

If we want to force a transaction to do something and want it to fail if that is not done then we use the ‘require’ keyword

ABI states the functions and tells hey you can interact with these functions

```
interface AggregatorV3Interface {
  function decimals() external view returns (uint8);

  function description() external view returns (string memory);

  function version() external view returns (uint256);

  function getRoundData(
    uint80 _roundId
  ) external view returns (uint80 roundId, int256 answer, uint256 startedAt, uint25

  function latestRoundData()
    external
    view
    returns (uint80 roundId, int256 answer, uint256 startedAt, uint256 updatedAt,
}
```

Now rather than using this contract in our code which makes it bulky we can do npm import

```
import { AggregatorV3Interface } from "@chainlink/contract/src/v0.8/shared/inte
```

```
function getPrice() public {
    AggregatorV3Interface priceFeed = AggregatorV3Interface("Address here");

    (,int256 price, , , ) = priceFeed.latestRoundData();

    return price * 1e10;
}
```

```
function getConversionRate(uint256 ethAmount) public view returns (uint 256) {
    uint256 ethPrice = getPrice();
    uint256 ethAmountInUSD = (ethPrice * ethAmount) / 1e18;
}
```

```
uint256 public minimumUsd = 5 * 1e18;

function fund() public payable {
    require(getConversionRate(msg.value) >= minimumUSD, "didnt send enough E
}
```

```
mapping(address funder => uint256 amountFunded) public addressToAmountFun

function fund() public payable {
    require(getConversionRate(msg.value) >= minimumUSD, "didnt send enough E
    funders.push(msg.sender);
```

```
    addressToAmountFunded[msg.sender]  addressToAmountFunded[msg.sender
}
```

Library are similar to contracts, but you cant declare any state variable and you can not send ether.

A library is embedded into the contract if all library functions are internal

Otherwise the library must be deployed and then linked before the contract is deployed

```
library PriceConverter {
    // We could make this public, but then we'd have to deploy it
    function getPrice() internal view returns (uint256) {
        // Sepolia ETH / USD Address
        // https://docs.chain.link/data-feeds/price-feeds/addresses#Sepolia%20Tes
        AggregatorV3Interface priceFeed = AggregatorV3Interface(
            0x694AA1769357215DE4FAC081bf1f309aDC325306
        );
        (, int256 answer, , , ) = priceFeed.latestRoundData();
        // ETH/USD rate in 18 digit
        return uint256(answer * 10000000000);
        // or (Both will do the same thing)
        // return uint256(answer * 1e10); // 1* 10 ** 10 == 10000000000
    }

    // 1000000000
    function getConversionRate(uint256 ethAmount)
        internal
        view
        returns (uint256)
    {
        uint256 ethPrice = getPrice();
        uint256 ethAmountInUsd = (ethPrice * ethAmount) / 1000000000000000000
        // or (Both will do the same thing)
        // uint256 ethAmountInUsd = (ethPrice * ethAmount) / 1e18; // 1 * 10 ** 18 ==
```

```
        // the actual ETH/USD conversion rate, after adjusting the extra 0s.
        return ethAmountInUsd;
    }
}
```

```
import { PriceConverter } from "./PriceConverter"

contract FundMe {
    use PriceConverter for uint256;

    function fund() public payable {
        msg.value.getConversionRate();
        funders.push(msg.sender);
        addressToAmountFunded[msg.sender]  addressToAmountFunded[msg.send
}
```

```
contract SafeMathTest {
    uint8 public BigNumber = 255;
    // Max number uint8 can accomadate is 255

    // function add() public {
    //     bigNumber = bigNumber + 1;
    // }

    function add() public {
        unchecked {bigNumber = bigNumber + 1;}
    }

}
```

```
// it automatically checkes and it automatically goes
// to next 0, again then 1, 2, ......
```

```
contract FundMe {
  function withdraw() public onlyOwner {
    for (uint256 funderIndex=0; funderIndex < funders.length; funderIndex++){
      address funder = funders[funderIndex];
      addressToAmountFunded[funder] = 0;
    }
    funders = new address[](0);


    // 3 ways → transfer, send, call

    // // transfer
    // payable(msg.sender).transfer(address(this).balance);

    // transfer(2300 gas, error())
    // If more than 2300 gas is used then it will send an error
    // but send will only revert the statement if we use require until transfer

    // // send
    // bool sendSuccess = payable(msg.sender).send(address(this).balance);
    // require(sendSuccess, "Send failed");

    // // call
    (bool callSuccess, ) = payable(msg.sender).call{value: address(this).balanc
    require(callSuccess, "Call failed");
  }
}
```

```
// Constructor

contract FundMe {
    require(msg.sender == owner, "Must be owner");
    address public owner;

    constructor() {
        owner = msg.sender;
    }
}
```

```
// Modifier
// Helps us to create a keyword that we can put right
// in the function declaration to add some functionality
// very quickly and easily to any function

modifier onlyOwner() {
    require(msg.sender == owner, "Must be owner");
    _; // This means we would execute the code inside the function first if placed c
    // now its at bottom so means this will be executed first then require
}

contract FundMe public onlyOwner{
    // require not required now
    address public owner;

    constructor() {
        owner = msg.sender;
    }
}
```

```
// Constant & Immutable

contract FundMe {
    uint256 public const minimumUsd = 5;
    // (This saves gas - using constant)

    address public immutable i_owner;
}
```

```
// Custom error (saves more gas)

modifier onlyOwner {
    // require(msg.sender == i_owner, NotOwner());
    if (msg.sender != i_owner) { revert NotOwner(); }
    _;
}
```

```
contract FallbackExample {
    uint256 public result;

    receive() external payable {
        // receive is a special function, whenever we make
        // transaction, as long as there's no data associated
        // with the transaction, this recieve function will trigger

        result = 1;
    }

    // similaryl we have fallback function similar to recieve function
    // works even when data is sent along with the transaction
```

```
    fallback() external payable {
        result = 2;
    }
}

// Explainer from: https://solidity-by-example.org/fallback/
// Ether is sent to contract
//      is msg.data empty?
//          /   \
//         yes  no
//         /     \
//    receive()?  fallback()
//     /   \
//   yes   no
//  /       \
//receive()  fallback()
```

Checkout solidity-1 blockchain code and concepts

Transaction Types

Legacy (0x0) → Ethereum transaction format used before the introduction of types transactions

EIP-2930 (0x1) → has access less thing, EIP-1559 (0x2) → has recording as the default transaction, EIP-712 (0x71) which is also 113, default, specific to zksync ecosystem