What Is a Smart Contract?

Smart contracts are computer programs that are hosted and executed on a blockchain network. Each smart contract consists of code specifying predetermined conditions that, when met, trigger outcomes. By running on a decentralized blockchain instead of a centralized server, smart contracts allow multiple parties to come to a shared result in an accurate, timely, and tamper-proof manner. Smart contracts are a powerful infrastructure for automation because they are not controlled by a central administrator and are not vulnerable to single points of attack by malicious entities. When applied to multi-party digital agreements, smart contract applications can reduce counterparty risk, increase efficiency, lower costs, and provide new levels of transparency into processes.

How Smart Contracts Work

Smart contracts are tamper-proof programs on blockchains with the following logic: "*if/when x event happens, then execute y action*." One smart contract can have multiple different conditions and one application can have multiple different smart contracts to support an interconnected set of processes. There are also multiple smart contract languages for programming, with Ethereum's Solidity being the most popular. Any developer can create a smart contract and deploy it on a public blockchain for their own purposes, e.g., a personal yield aggregator that automatically shifts their funds to the highest-earning application. However, many smart contracts involve multiple independent parties that may or may not know one another and don't necessarily trust one another. The smart contract defines exactly how users can interact with it, involving who can interact with the smart contract, at what times, and what inputs result in what outputs. The result is multi-party digital agreements that evolve from today's probabilistic state, where they will *probably* execute as desired, to a new deterministic state where they are *guaranteed* to execute according to their code.

Smart Contract Benefits

- **Security** – Running the contract on decentralized blockchain infrastructure ensures there is no central point of failure to attack, no centralized intermediary to bribe, and no mechanism for either party or a central admin to use to tamper with the outcome.
- **Reliability –** Having the contract logic redundantly processed and verified by a decentralized network of nodes provides strong tamper-proof, uptime, and correctness guarantees that the contract will execute on time according to its terms.
- **Equitable** – Using a decentralized network to host and enforce the terms of the agreement reduces the ability of a for-profit middleman to use their position of privilege to rent-seek and siphon off value.
- **Efficiency** – Automating the backend processes of the agreement—escrow, maintenance, execution, and/or settlement—means neither party has to wait for manual data to be entered, the counterparty to fulfill their obligations, or a middleman to process transaction

Smart Contract Limitations

One of the inherent limitations of smart contracts is that the underlying blockchains they run on are isolated networks, meaning blockchains have no built-in connection to the outside world. Without external connectivity, smart contracts cannot communicate with external systems to confirm the occurrence of real-world events nor can they access cost-efficient computational resources. Similar to a computer without the Internet, smart contracts are extremely limited without real-world connectivity. For example, they can't know the price of an asset before executing a trade, they can't check the average monthly rainfall before paying out a crop insurance claim, and they cannot verify that goods have arrived before settling with a supplier.

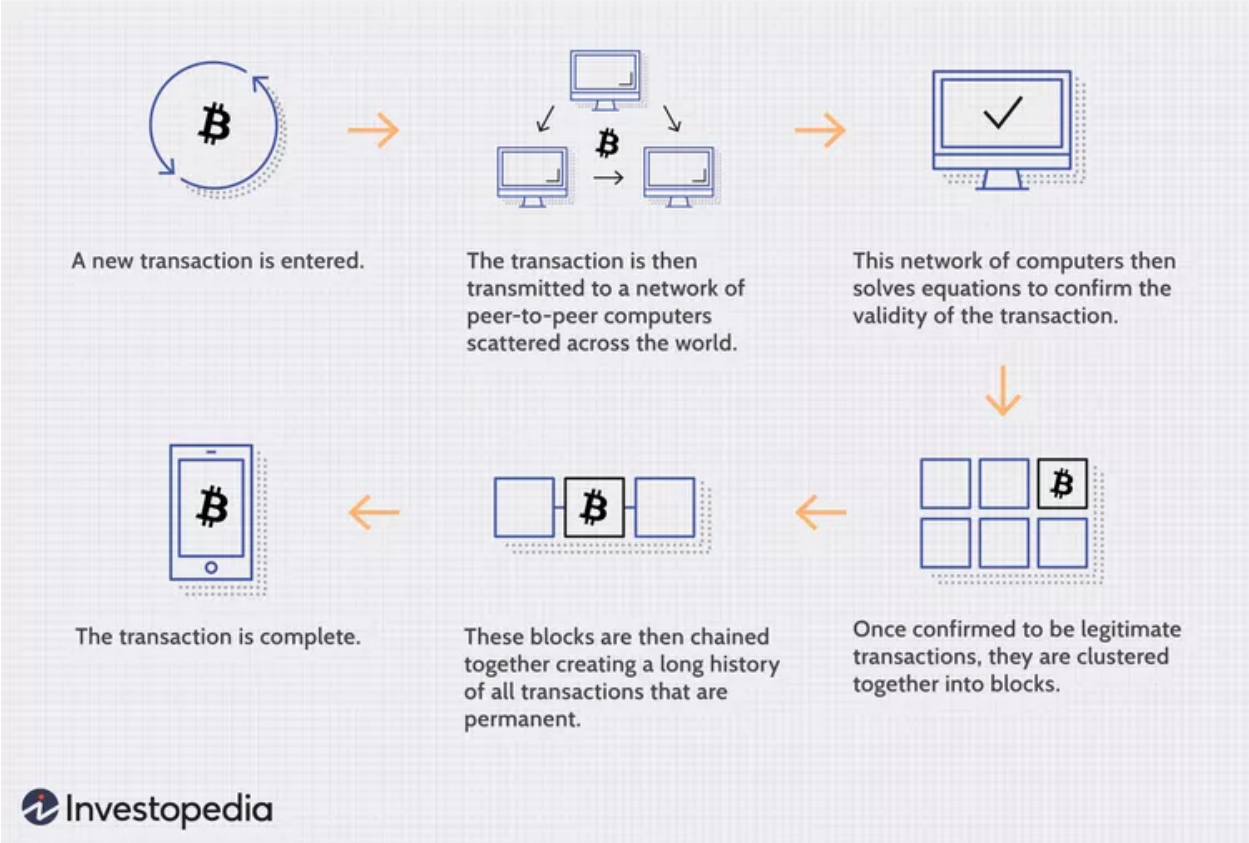
Thus, the major evolution underway in the blockchain industry is programmable smart contracts that connect with real-world data and traditional systems outside a blockchain, expanding the inputs and outputs used within smart contract logic. These hybrid smart contracts use secure middleware known as an oracle to combine on-chain code with off-chain infrastructure—e.g., trigger a smart contract with external data or settle a contract off-chain on a traditional payment rail.

What Is a Blockchain?

A blockchain is a distributed database or ledger that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format. Blockchains are best known for their crucial role in cryptocurrency systems, such as Bitcoin, for maintaining a secure and decentralized record of transactions. The innovation with a blockchain is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party.One key difference between a typical database and a blockchain is how the data is structured. A blockchain collects information together in groups, known as blocks, that hold sets of information. Blocks have certain storage capacities and, when filled, are closed and linked to the previously filled block, forming a chain of data known as the blockchain. All new information that follows that freshly added block is compiled into a newly formed block that will then also be added to the chain once filled. A database usually structures its data into tables, whereas a blockchain, as its name implies, structures its data into chunks (blocks) that are strung together. This data structure inherently makes an irreversible timeline of data when implemented in a decentralized nature. When a block is filled, it is set in stone and becomes a part of this timeline. Each block in the chain is given an exact timestamp when it is added to the chain.

How Does a Blockchain Work?

The goal of blockchain is to allow digital information to be recorded and distributed, but not edited. In this way, a blockchain is the foundation for immutable ledgers, or records of transactions that cannot be altered, deleted, or destroyed. This is why blockchains are also known as a distributed ledger technology (DLT). First proposed as a research project in 1991, the blockchain concept predated its first widespread application in use: Bitcoin, in 2009. In the years since, the use of blockchains has exploded via the creation of various cryptocurrencies.



Blockchain Decentralization

Imagine that a company owns a server farm with 10,000 computers used to maintain a database holding all of its client's account information. This company owns a warehouse building that contains all of these computers under one roof and has full control of each of these computers and all of the information contained within them. This, however, provides a single point of failure. What happens if the electricity at that location goes out? What if its Internet connection is severed? What if it burns to the ground? What if a bad actor erases everything with a single keystroke? In any case, the data is lost or corrupted.

What a blockchain does is to allow the data held in that database to be spread out among several network nodes at various locations. This not only creates redundancy but also maintains the fidelity of the data stored therein—if somebody tries to alter a record at one instance of the database, the other nodes would not be altered and thus would prevent a bad actor from doing so. If one user tampers with Bitcoin's record of transactions, all other nodes would cross-reference each other and easily pinpoint the node with the incorrect information. This system helps to establish an exact and transparent order of events. This way, no single node within the network can alter information held within it. Because of this, the information and history (such as of transactions of a cryptocurrency) are irreversible. Such a record could be a list of transactions (such as with a cryptocurrency), but it also is possible for a blockchain to hold a variety of other information like legal contracts, state identifications, or a company's product inventory.

Pros and Cons of Blockchain

For all of its complexity, blockchain's potential as a decentralized form of record-keeping is almost without limit. From greater user privacy and heightened security to lower processing fees and fewer errors, blockchain technology may very well see applications beyond those outlined above. But there are also some disadvantages.

PROS

- Improved accuracy by removing human involvement in verification
- Cost reductions by eliminating third-party verification
- Decentralization makes it harder to tamper with
- Transactions are secure, private, and efficient
- Transparent technology
- Provides a banking alternative and a way to secure personal information for citizens of countries with unstable or underdeveloped governments

CONS

- Significant technology cost associated with mining bitcoin
- Low transactions per second
- History of use in illicit activities, such as on the dark web
- Regulation varies by jurisdiction and remains uncertain
- Data storage limitations

The problem with blockchains

Since the blockchain has its distributed ledger nature, each node in the network has to be able to find the same end result given the same input. Otherwise, when a node looks to validate a transaction another node makes, it would end up with a different result. This architecture is intentional, and it's designed to be deterministic intentionally.

In blockchain, the mechanism for agreeing upon a data value is called consensus, and determinism is important so that nodes can come to a consensus. You might have heard of some of them, like Proof of Work (PoW) with Nakamoto Consensus or Proof of Stake (PoS) with Byzantine Consensus. Consensus is one of the key ingredients that make blockchain work in the first place. But we need the blockchain world to connect with the real world. We need to get the price of ETH and other cryptocurrencies into a contract so we can have DeFi. We need to get the weather data so we can have decentralized trustless insurance. We need data to use blockchain for one of its most important purposes, smart contracts. So how do we bridge the worlds with this constraint?

How oracles solve this

A blockchain oracle is any device or entity that connects a deterministic blockchain with off-chain data. These oracles enter every data input through an external transaction. This way, we can be sure that the blockchain itself contains all of the information required to verify itself. This is why oracles are known as blockchain middleware: They are the bridge between the two worlds.

What is a decentralized oracle?

A decentralized oracle or decentralized oracle network is a group of independent blockchain oracles that provide data to a blockchain. Every independent node or oracle in the decentralized oracle network independently retrieves data from an off-chain source and brings it on-chain. The data is then aggregated so the system can come to a deterministic value of truth for that data point. Decentralized oracles solve the oracle problem. Chainlink is a framework for choosing your independent network of nodes to connect the real world's data to the blockchain to enable smart contracts to reach their true potential. With this, we are leveraging the same reliable decentralized infrastructure concept the blockchain has, but for blockchain oracles. If

nodes/sources are hacked, depreciated, or deleted, the network of Chainlink will leverage the decentralized network and carry on.

PROOF OF WORK:

The proof of work (PoW) is a standard consensus algorithm used by the most popular cryptocurrency networks like bitcoin and litecoin. It requires a participant node to prove that the work done and submitted by them qualifies them to receive the right to add new transactions to the blockchain. However, this whole mining mechanism of bitcoin needs high energy consumption and a longer processing time. Bitcoin is a digital currency that is underpinned by a kind of distributed ledger known as a ‘blockchain.’ This ledger contains a record of all bitcoin transactions, arranged in sequential "blocks,’ so that no user is allowed to spend any of their holdings twice. To prevent tampering, the ledger is public, or "distributed"; an altered version would quickly be rejected by other users. Whatever the size of the original data set, the hash generated by a given function will be the same length. The hash is a one-way function: it cannot be used to obtain the original data, only to check that the data that generated the hash matches the original data. For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it. To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

PROOF OF STAKE:

The proof of work is another standard consensus algorithm that evolved as a low cost, low-energy-consuming alternative to the PoW algorithm. It involves the allocation of responsibility in maintaining the public ledger to a participant node in proportion to the number of virtual currency tokens held by it. However, this comes with the drawback that it incentivizes crypto coin hoarding instead of spending. POS reduces the amount of computational work needed to verify blocks and transactions that keep the blockchain, and thus a cryptocurrency, secure. Proof-of-stake changes the way blocks are verified using the machines of coin owners. The owners offer their coins as collateral for the chance to validate blocks. Coin owners with staked coins become “validators. Validators are then selected randomly to ‘mine, or validate the block. This system randomizes who gets to mine rather than using a competition-based mechanism like proof-of-work. To become a validator, a coin owner must ‘stake’ a specific amount of coins. Blocks are validated by more than one validator, and when a specific number of validators verify that the block is accurate, it is finalized and closed. Different proof-of-stake

mechanisms may use different methods for validating blocks—when Ethereum transitions to PoS, it will use shards for transaction submissions. A validator will verify the transactions and add them to a shard block, which requires at least 128 validators to attest to. Once shards are validated and the block created, two-thirds of the validators must agree that the transaction is valid. then the block is closed.

DIFFERENCE BETWEEN PoS AND PoW:

Both consensus mechanisms help blockchains synchronize data, validate information, and process transactions. Each method has proven to be successful at maintaining a blockchain, although there are pros and cons to each. However, the two algorithms have very different approaches.

1. Under PoS, block creators are called validators. A validator checks the transactions, verifies activity, votes on outcomes, and maintains records. Under PoW, the creators are called mines. Miners solve complex mathematical problems to verify transactions in return
2. To “buy into” the position of becoming a block creator, investors need only to purchase the sufficient limit of coins or tokens required to become a validator. For PoW, mines must invest in processing equipment and incur heavy energy charges to power the machines attempting to solve the computations.
3. The equipment and energy costs under PoW mechanisms are expensive, limiting access to mining and strengthening the security of the blockchain. However, PoS blockchains often allow for more scalability due to their energy efficiency.

EVM:

The EVM is a virtual stack that is embedded within every fully participating node in the network, that executes contract bytecode. The EVM is a turning complete system, which means it can perform any type of logical step associated with computational functions. While Bitcoin provides rewards for running a transaction, Ethereum charges fees for executing software instructions. The gas mechanism is Ethereum lets users pre-pay for the instructions they want to execute on the EVM using Ether. EVMs are pretty versatile in that they can be implemented using javascript, C++, Python, etc. The contracts that are run using the EVM are written using Solidity. So, once and for all, what is Solidity? It is a high-level programming language that is compatible with how humans express instructions using numbers and letters instead of binary code. Solidity smart contracts are instructions that are then compiled into the EVM's bytecode. The nodes in the Ethereum network, as mentioned, run EVM instances that permit them to agree on the execution of the same set of instructions.

DLT - DISTRIBUTED LEDGER TECHNOLOGY:

Properties:

1. Programmable
2. Distributed
3. Immutable
4. Time-Stamped
5. Unanimous
6. Anonymous
7. Secure

Blockchain is one type of distributed ledger. Distributed ledgers use independent computers (referred to as nodes) to record, share and synchronize transactions in their respective electronic ledgers (instead of keeping data centralized as in a traditional ledger). Distributed ledger technology (DLT) is a digital system for recording the transaction of assets in which the transactions and their details are recorded in multiple places simultaneously. Unlike traditional databases, distributed ledgers have no central data store or administration functionality. In a distributed ledger, each node processes and verifies every item, thereby generating a record of each item and creating a consensus on its veracity. A distributed ledger can be used to record static data, such as a registry, and dynamic data, such as financial transactions. Distributed ledger technology (DLT) refers specifically to the technological infrastructure and protocols that allow the simultaneous access, validation, and updating of records that characterize distributed ledgers. It works on a computer network spread over multiple entities or locations. DLT uses cryptography to securely store data, cryptographic signatures, and keys to allow access only to authorized users. The technology also creates an immutable database, which means information, once stored, cannot be deleted and any updates are permanently recorded for posterity. This architecture represents a significant change in how information is gathered and communicated by moving record-keeping from a single, authoritative location to a decentralized system in which all relevant entities can view and modify the ledger. As a result, all other entities can see who is using and modifying the ledger. This transparency of DLT provides a high level of trust among the participants and practically eliminates the chance of fraudulent activities occurring in the ledger.

SOLIDITY CONTRACTS & PROPERTIES:

Solidity Contracts are like a class in any other object-oriented programming language. They firmly contain data as state variables and functions which can modify these variables. When a function is called on a different instance (contract), the EVM function call happens and the context is switched in such a way that the state variables are inaccessible. A contract or its function needs to be called for anything to happen. Some basic properties of contracts are as

follows:

1. Constructor: Special method created using the constructor keyword, which is invoked only once when the contract is created.
2. State Variables: These are the variables that are used to store the state of the contract.
3. Functions: Functions are used to manipulate the state of the contracts by modifying the state variables.

ABI:

The **Contract Application Binary Interface** (ABI) is the standard way to interact with contracts in the Ethereum ecosystem, both from outside the blockchain and for contract-to-contract interaction. Data is encoded according to its type, as described in this specification. It specifies its interface and the set of functions accessed from outside the smart contract. The ABI is used only for defining the events of the contract and function signatures, such as names of functions, return types, and argument types. However, it does not define their implementation.

VISIBILITY MODIFIERS:

Solidity provides four types of visibilities for functions and state variables. Functions have to be specified by any of the four visibilities but for state variables external is not allowed.

1. External: External functions are can be called by other contracts via transactions. An external function cannot be called internally. For calling an external function within the contract the function name() method is used. Sometimes external functions are more efficient when they have large arrays of data.
2. Public: Public functions or variables can be called both externally or internally via messages. For public static variables, a getter method is created automatically in solidity.
3. Internal: These functions or variables can be accessed only internally ie within the contract or the derived contracts.

4. Private: These functions or variables can only be visible for the contracts in which they are defined. They are not accessible to derived contracts too.

CODE:

```
Solidity program to demonstrate
// visibility modifiers
pragma solidity ^0.5.0;

// Creating a contract
contract contract_ example

// Declaring private
// state variable uint private num2,

// Declaring public
// state variable uint public num

//Declaring Internal
// state variable string internal str;

// Defining a constructor
constructor) public
num2 = 10;
```

CONSTRUCTORS:

A constructor is a special method in any object-oriented programming language that gets called whenever an object of a class is initialized It is different in the case of Solidity, Solidity provides a constructor declaration inside the smart contract and it invokes only once when the contract is deployed and is used to initialize the contract state. A default constructor is created by the compiler if there is no explicitly defined constructor.

Creating a constructor:

A Constructor is defined using a constructor keyword without any function name followed by an access modifier. It's an optional function that initializes the state variables of the contract. A constructor can be either internal or public, an internal constructor marks a contract as abstract.

Syntax:

constructor() ‹Access Modifier>

```
// Solidity program to demonstrate
// creating a constructor
pragma solidity ^0.5.0;

// Creating a contract contract constructor Example |
// Declaring state variable
string str;

// Creating a constructo
// to set value of 'str'
constructor) public (
str = "GeeksForGeeks}

// Defining function to // return the value of 'str'
function getValue () public view returns ( string memory)} return str;}}
```

Variables

Declaration of Variables :

Solidity declaration of variables is a little bit different, to declare a variable the user has to specify the data type first followed by access modifier.

Syntax:
<type> <access modifier> <variable name> ;
Types of Variables
Solidity is a statically typed language i.e. each declared variable always has a default value based on its data type, which means there is no concept of ‘null’ or ‘undefined’.

Solidity supports three types of variables:
1. State Variables: Values of these variables are permanently stored in the contract storage. Each function has its own scope, and state variables should always be defined outside of that scope.
2. Local Variable: Values of these variables are present till the function executes and it cannot be accessed outside that function. This type of variable is usually used to store temporary Values.
3. Global Variables: These are some special variables that can be used globally and give information about the transactions and blockChain properties. Some of the global variables are listed below:

Arrays

Arrays are data structures that store the fixed collection of elements of the same data types in which each and every element has a specific location called index. Instead of creating numerous individual variables of the same type, we just declare one array of the required size and store the elements in the array and can be accessed using the index. In Solidity, an array can be of fixed size or dynamic size.
Arravs have a continuous memory location. where the lowest index corresponds to the first element while the highest represents the last

Creating an Array
To declare an array in Solidity, the data type of the elements and the number of elements should be specified. The size of the array must be a positive integer and data type should be a valid Solidity type

Syntax:
<data type> <array name> size = <initialization)

Types of Array

Fixed-size Arrays - The size of the array should be predenned. The total number of elements should not exceed the size of the array. If the size of the array is not specified then the array of enough size is created which is enough to hold the initialization.
Static arrays have their size or length determined when the array is created and/or allocated. For this reason, they may also be referred to as fixed-length arrays or fixed arrays

Dynamic array
A data structure consisting of a collection of elements that allows individual elements to be added or removed.
Dynamic arrays allow elements to be added and removed at runtime. Most current programming languages include built-in or standard library functions for creating and managing dynamic arrays.

// Solidity program to demonstrate Fixed-Array

```
// creating a fixed-size array
pragma solidity “0.5.0;
// Creating a contract
contract Types {
// Declaring state variables
// of type array
uint(6] data;
// Defining function to add
// values to an array
function array_example() public returns (
int[5] memory, uint[6] memory){
int[5] memory data
= [int(50), -63, 77, -28, 90];
data
=[uint(10), 20, 30, 40, 50, 60];
return (data, data);
}
}
```

// Solidity program to demonstrate Dynamic-Array

```
// creating a dynamic array
pragma solidity “0.5.0;
// Creating a contract
contract Types {
// Declaring state variable
// of type array. One is fixed-size
// and the other is dynamic array
uint[] data
=[10, 20, 30, 40, 50];
int[] datar;
// Defining function to
// assign values to dynamic array
function dynamic_array() public returns(
Deb eLiBeeloerteyamerelidbertoerte)aal
OPEB
= [int(-60), 70, -80, 90, -100, -120, 140];
return (data, datal);
}
}
```

Accessing Array Elements:

1. The elements of the array are accessed by using the index. If you want to access ith element then you have to access (i-1)th index.
2. Length of Array: Length of the array is used to check the number of elements present in an array. The size of the memory array is fixed when they are declared, while in case the dynamic array is defined at runtime so for manipulation length is required.
3. Push: Push is used when a new element is to be added in a dynamic array. The new element is always added at the last position of the array.
4. Pop: Pop is used when the last element of the array is to be removed in any dynamic array.

Enum and Struct:

Enums are the way of creating user-defined data types, it is usually used to provide names for integral constants which makes the contract better for maintenance and reading. Enums restrict the variable with one of a few predefined values, these values of the enumerated list are called enums. Options are represented with integer values starting from zero, a default value can also be given for the enum. By using enums it is possible to reduce the bugs in the code.
Syntax:
enum <enumerator_name> {
element I, element 2,....,element n
j
Restriction The explicit conversions check the value ranges at runtime and failure causes an exception. Enums needs at least one member.

Struct
Structs in Solidity allows you to create more complicated data types that have multiple properties. You can define your own type by creating a struct.
They are useful for grouping together related data.
Structs can be declared outside of a contract and imported in another contract.
Generally, it is used to represent a record. To define a structure struct keyword is used, which creates a new data type.

Web3.eth:

The web3-eth package allows you to interact with an Ethereum blockchain and Ethereum smart contracts.
Note on checksum addresses
All Ethereum addresses returned by functions of this package are returned as checksum addresses. This means some letters are
uppercase and some are lowercase. based on that it will calculate
a checksum for the address and prove its correctness Incorrect
checksum addresses will throw an error when passed into functions. If you want to circumvent the checksum check you can
make an address all lower- or uppercase

Example web3.eth.getAccounts(console.log);

Web2/ Web3

Many Web3 developers have chosen to build dapps because of Ethereum's inherent decentralization:

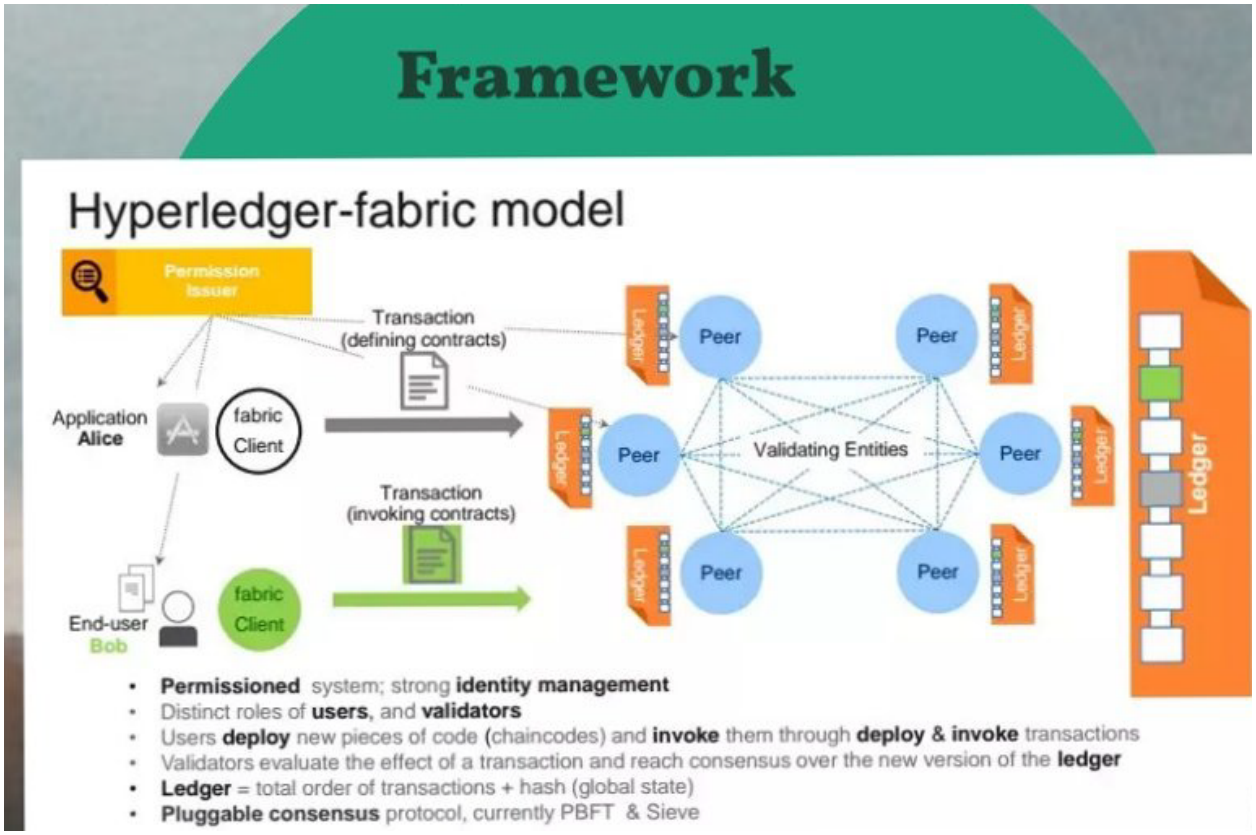
Anyone who is on the network has permission to use the service – or in other words, permission isn't required.
No one can block you or deny you access to the service.
Payments are built in via the native token, ether (ETH).
Ethereum is turing-complete, meaning you can program pretty much anything

Web3 has some limitations right now:

Scalability – transactions are slower on web3 because they're decentralized. Changes to state, like a payment, need to be processed by a node and propagated throughout the network.
UX – interacting with web3 applications can require extra steps, software, and education. This can be a hurdle to adoption.
Accessibility – the lack of integration in modern web browsers makes web3 less accessible to most users.
Cost – most successful dapps put very small portions of their code on the blockchain as it's expensive.

PRACTICAL COMPARISONS

Web2	Web3
Twitter can censor any account or tweet	Web3 tweets would be uncensorable because control is decentralized
Payment service may decide to not allow payments for certain types of work	Web3 payment apps require no personal data and can't prevent payments
Servers for gig-economy apps could go down and affect worker income	Web3 servers can't go down – they use Ethereum, a decentralized network of 1000s of computers as their backend



In a public blockchain, anyone is free to join and participate in the core activities of the blockchain network.
A private blockchain allows only selected and verified participants; the operator has the rights to override, edit, or delete entries on the blockchain.
A permissioned blockchain has properties of both private and public blockchains.
Permissioned have seen an increase in popularity thanks to their ability to allocate specific permissions to various users on the network.

Public Blockchain

A public blockchain is one where anyone is free to join and participate
in the core activities on the blockchain network .Anyone can read
write, and audit the ongoing activities on a public blockchain network, which helps achieve the self-governed, decentralized nature often touted when blockchain is discussed

Advantages -
A public network operates on an incentivizing scheme that encourages new participants to join and keep the network agile.

Public blockchains offer a particularly valuable solution from the point of view of truly decentralized democratized and authority-free operation.

Dis-advantages -
The primary disadv to secured public blockchains is the heavy energy consumption required to maintain them. The concern is a consensus mechanism that requires participants to compete to validate the information and receive reward for letting the network use their processing power. Not all blockchain networks use an energy intensive validation process, so not all use enormous amounts of electricity.

Private Blockchain

Participants can join a private blockchain network only through an invitation where their identity or other required information is authentic and verified. The validation is done by the network operators) or by a clearly defined set protocol implemented by the network through smart contracts or other automated approval methods
Private blockchains control who is allowed to participate in the network. If the network is capable of mining, its private nature can control which users can might maintain the shared lede blockchain as required or as
consensus protocol that decides the mining rights and rewards. Additionally, only select users or operator has the right to override, edit, or delete the necessary entries on the

Advantages
A private blockchain is not decentralized.
It is a distributed ledger that operates as a closed database secured with cryptographic concepts and the organization's need
• Only those with permission can run a full node, make transactions, or validate/authenticate blockchain changes.
By reducing the focus on protecting user identities and promoting transparency, private blockchains prioritize efficiency and immutability- the state of not being able to be changed. These are important features in supply, logistics, payroll, finances, accounting, and many other enterprise and business areas.
Disadvantages
While purposefully designed for enterprise applications, private blockchains lose out on many of the valuable attributes of permissionless systems functions.
simply because they are not widely applicable. They are instead built to accomplish specific tasks and
In this respect, private blockchains are susceptible to data breaches and other security threats. This is because there is generally a limited number of validators used to reach a consensus about transactions and data if there is a consensus mechanism.

In a private blockchain, there may not be consensus but only the immutability of entered data unless an operator or administrator make changes.

Decentralized application

A decentralized app (also known as a dApp or dapp) operates on a blockchin or peer-to-peer network of computers. It enables users to engage in transactions directly with one another as opposed to relying on a central authority.
Dapps have their backend code (smart contracts) running on a decentralized network and not a centralized server. They use the blockchain for data storage an smart contracts for their app logic
A smart contract is like a set of rules that live on-chain for all to see and run exactl according to those rules. Imagine a vending machine if you supply it with enough funds and the right selection, you'll get the item you want. And like vending machines, smart contracts can hold funds much like your blockhain account. This allows code to mediate agreements and transactions
Once dapps are deployed on the Blockchain network you can't change them. Dap can be decentralized because they are controlled by the logic written into the contract, not an individual or a company.

IPFS (Inter Planetary File System)

IPFS is a protocol and peer-to-peer network for storing and sharing data in a distributed file system.
A dapp can have frontend code and user interfaces written in any language (just like an app) to make calls to its backend. Furthermore, its frontend can get hosted on decentralized storage such as IPFS.
Decentralized - dapps operate on Ethereum, an open public decentralized platform where no one person or group has control
Deterministic - dapps perform the same function irrespective of the environment in which they get executed
Turing complete - dapps san perform any action given the required resources
Isolated - dapps are executed in a virtual environment known as Ethereum Virtual Machine so that if the smart contract has a bug, it won't hamper the normal functioning of the blockchain network
On smart contracts

A dapp can have frontend code and user interfaces written in any language (just like an app) to make calls to its backend. Furthermore, its frontend can get hosted on decentralized storage such as IPFS

Here's what happens when you add a file to IPFS - whether you're storing that file on your own local node or one operated by a pinning service or IPFS-enabled app.

1)When you add a file to IFS, your file is split into smaller chunks, cryptographically hashed, and given a unique fingerprint called a content identifier (CID). This CID acts as a permanent record of your file as it exists at that point in time.
2)When other nodes look up your file, they ask their peer nodes who's storing the content referenced by the file's CID. When they view or download your file, they cache a copy and become another provider of your content until their cache is cleared.
3)A node can pin content in order to keep (and provide) it forever, or discard content it hasn't used in a while to save space. This means each node in the network stores only content it is interested in, plus some indexing information that helps figure out which node is storing what.
4)If you add a new version of your file to IFS, its cryptographic hash is different, and so it gets a new CID. This means files stored on IFS are resistant to tampering and censorship - any changes to a file don't overwrite the original, and common chunks across files can be reused in order to minimize storage costs.
5)However, this doesn't mean you need to remember a long string of CIDs - IPFS can find the latest version of your file using the IPNS decentralized naming stem, and DNSLink can be used to map CIDs to human-readable DNS names.

Features
Zero downtime - Once the smart contract is deployed on the blockchain, the network as a whole will always be able to serve clients looking to interact with the contract. Malicious actors, therefore, cannot launch denial-of-service attacks targeted towards individual dapps.
Privacy - You don't need to provide real-world identity to deploy or interact with a dapp.
Resistance to censorship - No single entity on the network can block users from submitting transactions, deploying dapps, or reading data from the blockchain
Complete data integrity - Data stored on the blockchain is immutable and indisputable, thanks to cryptographic primitives. Malicious actors cannot forge transactions or other data that has already been made public.
Trustless computation/verifiable behavior - Smart contracts can be analyzed and are guaranteed to execute in predictable ways, without the need to trust a central authority. This is not true in traditional models; for example, when we use online banking systems, we must trust that financial institutions will not misuse our financial data, tamper with records, or get hacked.

Maintenance - Dapps can be harder to maintain because the code and data published to the blockchain are harder to modify
Performance overhead - There is a huge performance overhead, and scaling is really hard. To achieve the level of security, integrity, transparency, and reliability that Ethereum aspires to, every node runs and stores every transaction.

Network congestion - When one dapp uses too many
Currently, the network can only proci
Currently, the network can only pro Dot to-1s dad saerion.
per second.
User experience - It may be harder to engineer user-friendly experiences because the average end-user might find it too difficult to set up a tool stack necessary to interact with the blockchain in a truly secure fashion.
Centralization - User-friendly and developer-friendly solution built on top of the base layer of Ethereum might end up looking like centralized services anyways