

BLOCK CHAIN TECHNOLOGY

II M.SC COMPUTER SCIENCE-

R.INDRA, ASSISTANT PROFESSOR,

DEPARTMENT OF COMPUTER SCIENCE, SIGC

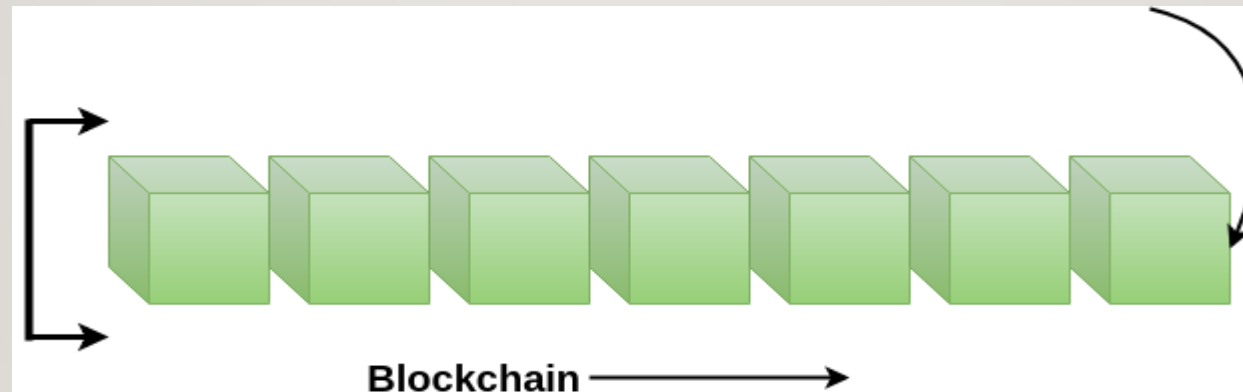
BLOCK CHAIN TECHNOLOGY

- Blockchain is a constantly growing **ledger** that keeps a **permanent** record of all the transactions that have taken place in a **secure, chronological, and immutable** way.
- It can be used for the secure transfer of money, property, contracts, etc. without requiring a third-party intermediary such as bank or government.
- Blockchain is a software protocol, but it could not be run without the Internet (like SMTP is for email).

WHAT IS BLOCKCHAIN?

- A blockchain is a constantly growing ledger which keeps a permanent record of all the transactions that have taken place in a secure, chronological, and immutable way.
- Let's breakdown the definition,
- **Ledger:** It is a file that is constantly growing.
- **Permanent:** It means once the transaction goes inside a blockchain, you can put up it permanently in the ledger.
- **Secure:** Blockchain placed information in a secure way. It uses very advanced cryptography to make sure that the information is locked inside the blockchain.
- **Chronological:** Chronological means every transaction happens after the previous one.
- **Immutable:** It means as you build all the transaction onto the blockchain, this ledger can never be changed.

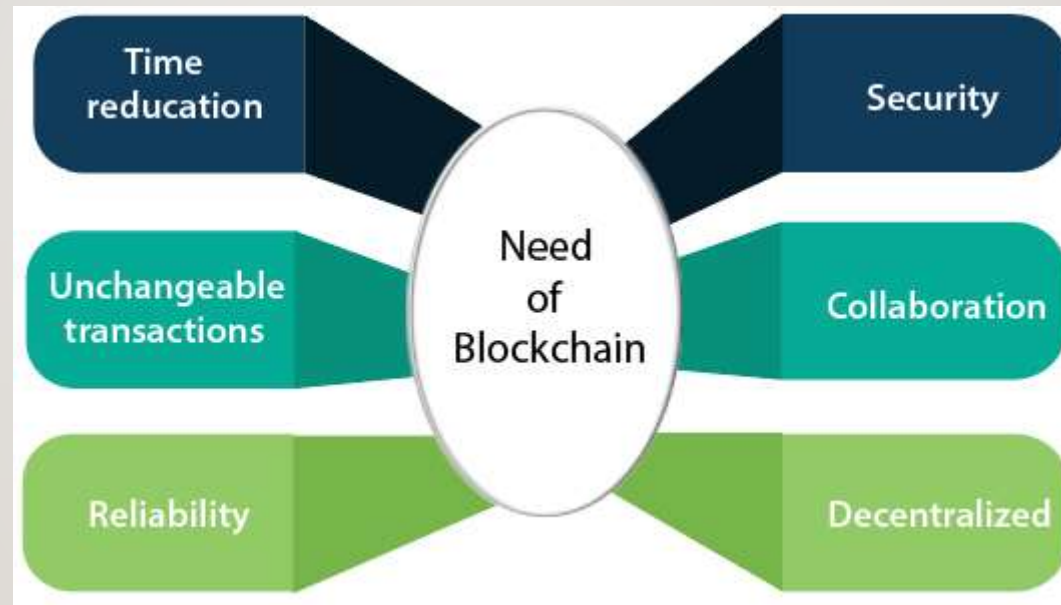
-
- A blockchain is a chain of blocks which contain information.
 - Each block records all of the recent transactions, and once completed goes into the blockchain as a permanent database.
 - Each time a block gets completed, a new block is generated.



WHO USES THE BLOCKCHAIN?

- Blockchain technology can be integrated into multiple areas.
- The primary use of blockchains is as a distributed ledger for cryptocurrencies.
- It shows great promise across a wide range of business applications like Banking, Finance, Government, Healthcare, Insurance, Media and Entertainment, Retail, etc

NEED OF BLOCKCHAIN



-
- Blockchain technology has become popular because of the following.
 - **Time reduction:** In the financial industry, blockchain can allow the quicker settlement of trades. It does not take a lengthy process for verification, settlement, and clearance. It is because of a single version of agreed-upon data available between all stakeholders.
 - **Unchangeable transactions:** Blockchain register transactions in a chronological order which certifies the unalterability of all operations, means when a new block is added to the chain of ledgers, it cannot be removed or modified.
 - **Reliability:** Blockchain certifies and verifies the identities of each interested parties. This removes double records, reducing rates and accelerates transactions.

-
- **Security:** Blockchain uses very advanced cryptography to make sure that the information is locked inside the blockchain. It uses Distributed Ledger Technology where each party holds a copy of the original chain, so the system remains operative, even the large number of other nodes fall.
 - **Collaboration:** It allows each party to transact directly with each other without requiring a third-party intermediary.
 - **Decentralized:** It is decentralized because there is no central authority supervising anything. There are standards rules on how every node exchanges the blockchain information. This method ensures that all transactions are validated, and all valid transactions are added one by one.
 - Prerequisite
 - Before learning blockchain in depth, you must have the basic knowledge of scripting languages such as HTML, JavaScript, and CSS.
 -

HISTORY OF BLOCKCHAIN



W. Scott Stornetta

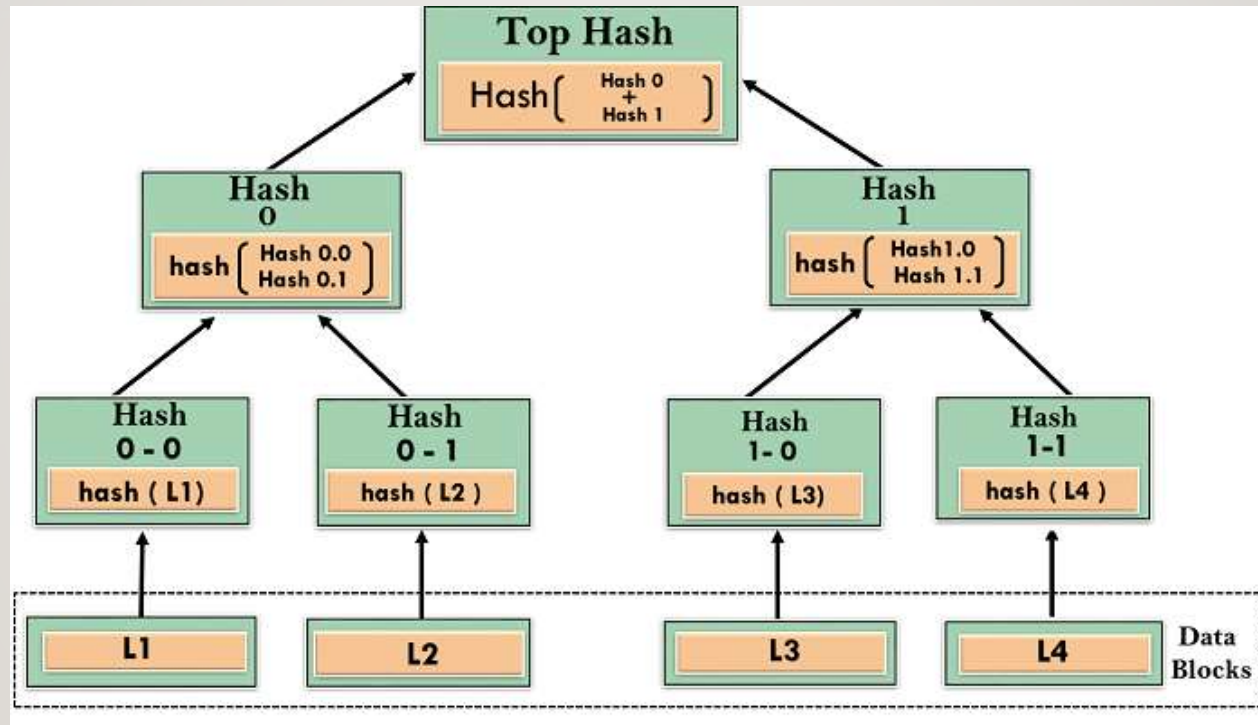


Stuart Haber

HISTORY OF BLOCKCHAIN

- The blockchain technology was described in **1991** by the research scientist **Stuart Haber** and **W. Scott Stornetta**.
- They wanted to introduce a computationally practical solution for time-stamping digital documents so that they could not be backdated or tampered.
- They develop a system using the concept of **cryptographically** secured chain of blocks to store the time-stamped documents.

-
- In **1992**, Merkle Trees were incorporated into the design, which makes [blockchain](#) more efficient by allowing several documents to be collected into one block.
 - **Merkle Trees** are used to create a 'secured chain of blocks.'
 - It stored a series of data records, and each data records connected to the one before it.
 - The newest record in this chain contains the history of the entire chain.
 - However, this technology went unused, and the patent lapsed in 2004.



-
- In **2004**, computer scientist and cryptographic activist **Hal Finney** introduced a system called **Reusable Proof Of Work (RPoW)** as a prototype for digital cash. It was a significant early step in the history of cryptocurrencies. The RPoW system worked by receiving a non-exchangeable or a non-fungible Hashcash based proof of work token in return, created an **RSA-signed** token that further could be transferred from person to person.



Hal Finney

-
- RPoW solved the double-spending problem by keeping the ownership of tokens registered on a trusted server. This server was designed to allow users throughout the world to verify its correctness and integrity in real-time.
 - Further, in **2008**, **Satoshi Nakamoto** conceptualized the theory of **distributed blockchains**. He improves the design in a unique way to add blocks to the initial chain without requiring them to be signed by trusted parties. The modified trees would contain a secure history of data exchanges.



Satoshi Nakamoto

-
- It could be managed autonomously without requiring a central authority.
 - These improvements were so beneficial that makes blockchains as the backbone of cryptocurrencies. [A cryptocurrency is an encrypted data string that denotes a unit of currency. It also serves as a secure ledger of transactions, e.g., buying, selling, and transferring.]
 - Today, the design serves as the public ledger for all transactions in the cryptocurrency space.
 - The evolution of blockchains has been steady and promising.
 - The words block and chain were used separately in Satoshi Nakamoto's original paper but were eventually popularized as a single word, the Blockchain, by **2016**.
 - In recent time, the file size of cryptocurrency blockchain containing records of all transactions occurred on the network has grown from **20 GB** to **100 GB**.

DISTRIBUTED SYSTEMS -SYLLABUS

- Distributed systems is essential to the understanding of blockchain technology, as blockchain is a distributed system at its core.
- It is a distributed ledger which can be centralized or decentralized.
- **In blockchain, decentralization refers to the transfer of control and decision-making from a centralized entity (individual, organization, or group thereof) to a distributed network.**
- [example McDonald's is a prime example of **centralized** management and standardization. The exact same number of pickles is put on each burger no matter where you are in the world.]
- [example one example of a **decentralized** social network. It is based on open-source software and functions a lot like Twitter.
- A blockchain is originally intended to be and is usually used as a decentralized platform.
- It can be thought of as a system that has properties of both decentralized and distributed paradigms.
- It is a decentralized-distributed system.

-
- **Distributed systems are a computing paradigm whereby two or more nodes work with each other in a** coordinated fashion to achieve a common outcome.
 - It is modeled in such a way that end users see it as a single logical platform. For example, Google's search engine is based on a large distributed system, but to a user, it looks like a single, coherent platform.
 - **A node can be defined as an individual player in a distributed system.**
 - **All nodes are capable of sending and** receiving messages to and from each other.
 - Nodes can be honest, faulty, or malicious, and they have memory and a processor

-
- A node that exhibits irrational behavior is also known as a **Byzantine node after the Byzantine Generals Problem**.
 - Byzantine fault tolerance refers to the ability of a network or system to continue functioning even when some components are faulty or have failed.
 - Byzantine Fault Tolerance [BFT]
 - With a BFT system, blockchain networks keep functioning or implementing planned actions as long as most network participants are reliable and genuine
 - A small-scale example of a distributed system is shown in the following diagram. This distributed system has six nodes out of which one (**N4**) is a **Byzantine node leading to possible data inconsistency**.
 - **L2 is a link that is broken or slow**, and this can lead to partition in the network

DESIGN OF A DISTRIBUTED SYSTEM: N4 IS A BYZANTINE NODE, L2 IS BROKEN OR A SLOW NETWORK LINK

A small-scale example of a distributed system is shown in the following diagram. This distributed system has six nodes out of which one (N4) is a Byzantine node leading to possible data inconsistency. L2 is a link that is broken or slow, and this can lead to partition in the network.

```
graph TD; N1((N1)) <-->|L1| N2((N2)); N2 <-->|L2| N3((N3)); N3 <-->|L3| N4((N4)); N3 <-->|L4| Nn((Nn)); N4 <-->|L5| N5((N5)); EndUsers[End users] <--> Nn;
```

Design of a distributed system: N4 is a Byzantine node, L2 is broken or a slow network link

The primary challenge in distributed system design is coordination between nodes and fault tolerance. Even if some of the nodes become faulty or network links break, the distributed system should be able to tolerate this and continue to work to achieve the desired result. This problem has been an active area of distributed system design research for many years, and several algorithms and mechanisms have been proposed to overcome these

38

Activate Windows
Go to Settings to activate Windows.

Type here to search

INR/USD -0.25%

10:23
20-06-2023

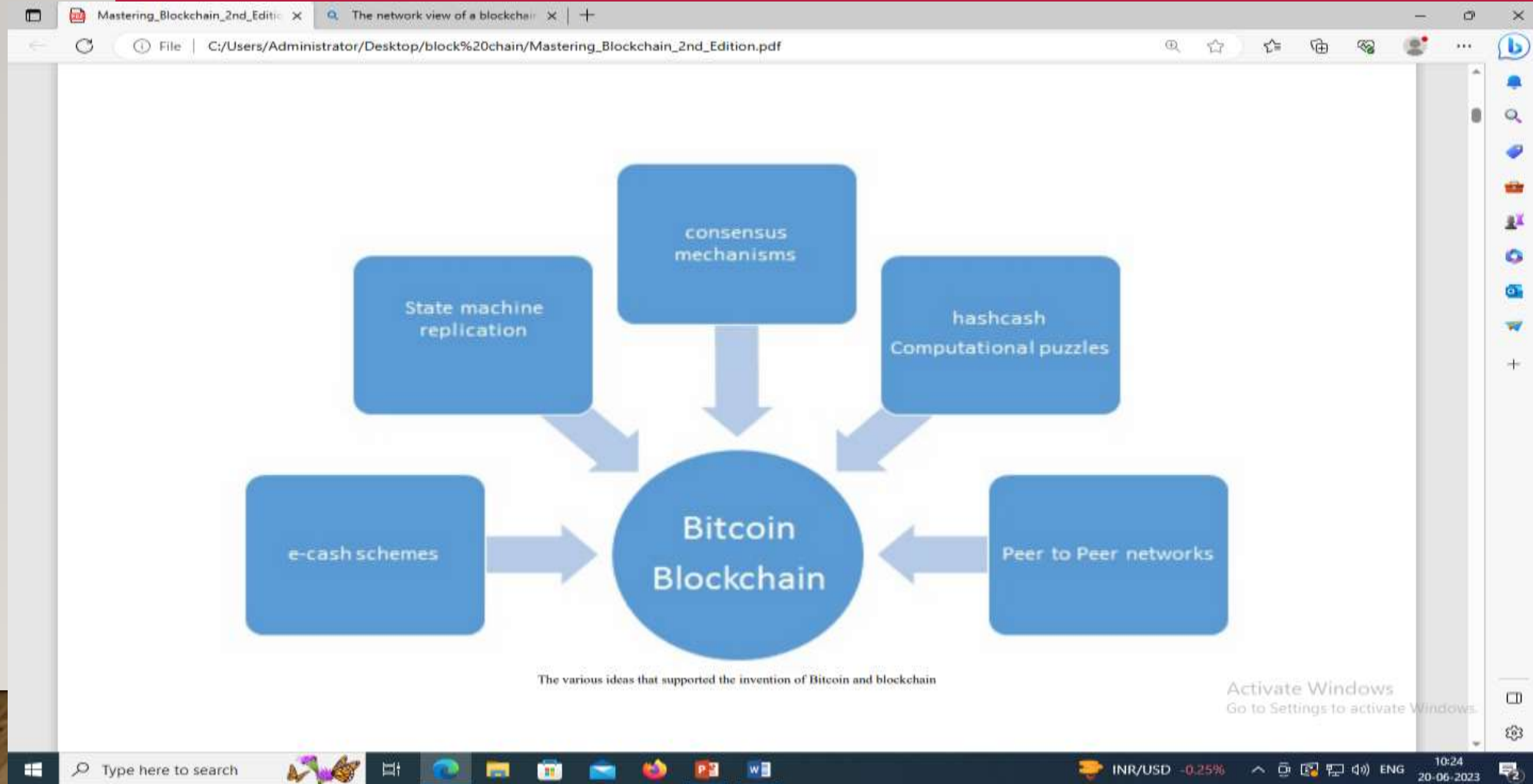
-
- The primary challenge in distributed system design is coordination between nodes and fault tolerance.
 - Even if some of the nodes become faulty or network links break, the distributed system should be able to tolerate this and continue to work to achieve the desired result.
 - This problem has been an active area of distributed system design research for many years, and several algorithms and mechanisms have been proposed to overcome these issues.

-
- Distributed systems are so challenging to design that a hypothesis known as the **CAP theorem has been proven, [CAP Theorem stands for Consistency, Availability, and Partition Tolerance.]** which states that a distributed system cannot have all three of the much-desired properties simultaneously; that is, consistency, availability, and partition tolerance.

THE HISTORY OF BLOCKCHAIN AND BITCOIN

- Blockchain was introduced with the invention of Bitcoin in 2008. Its practical implementation then occurred in 2009.
- it is essential to refer to Bitcoin because, without it, the history of blockchain is not complete.
- **Electronic cash**
- The concept of electronic cash or digital currency is not new. Since the 1980s, e-cash protocols have existed that are based on a model proposed by David Chaum

THE VARIOUS IDEAS THAT SUPPORTED THE INVENTION OF BITCOIN AND BLOCKCHAIN



-
- Just as understanding the concept of distributed systems is necessary to comprehend blockchain technology,
 - the idea of electronic cash is also essential in order to appreciate the first and astonishingly successful application of blockchain, Bitcoin, or more broadly cryptocurrencies in general.
 - Two fundamental e-cash system issues need to be addressed: accountability and anonymity

-
- **Accountability is required to ensure that cash is spendable only once (double-spend problem) and that it can** only be spent by its rightful owner.
 - Double spend problem arises when same money can be spent twice.
 - As it is quite easy to make copies of digital data, this becomes a big issue in digital currencies as you can make many copies of same digital cash.

-
- **Anonymity is required to protect users' privacy.**
 - **As with physical cash, it is almost** impossible to trace back spending to the individual who actually paid the money.
 - David Chaum solved both of these problems during his work in 1980s by using two cryptographic operations, namely **blind signatures and secret sharing.**

-
- In 2009, the first practical implementation of an electronic cash (e-cash) system named Bit coin appeared.
 - The term cryptocurrency emerged later. For the very first time, it solved the problem of distributed consensus in a trustless network.
 - It used **public key cryptography with a Proof of Work (PoW) mechanism to provide a** secure, controlled, and decentralized method of minting digital currency.
 - The key innovation was the idea of an ordered list of blocks composed of transactions and cryptographically secured by the PoW mechanism.

BLOCKCHAIN

- In 2008, a groundbreaking paper entitled *Bitcoin: A Peer-to-Peer Electronic Cash System* was written on the topic of peer-to-peer electronic cash under the pseudonym *Satoshi Nakamoto*.
- *It introduced the term **chain of blocks**.*
- **No one knows the actual identity of Satoshi Nakamoto.**
- **After introducing Bitcoin in 2009, he remained** active in the Bitcoin developer community until 2011. He then handed over Bitcoin development to its core developers and simply disappeared.
- Since then, there has been no communication from him whatsoever, and his existence and identity are shrouded in mystery.
- The term *chain of blocks* evolved over the years into the word *blockchain*.

-
- As stated earlier, blockchain technology incorporates a multitude of applications that can be implemented in various economic sectors.
 - Particularly in the finance sector, significant improvement in the performance of financial transactions and settlements is seen as resulting in desirable time and cost reductions.

BLOCKCHAIN DEFINED

- **Layman's definition:** *Blockchain is an ever-growing, secure, shared record keeping system in which each user of the data holds a copy of the records, which can only be updated if all parties involved in a transaction agree to update.*
- **Technical definition:** *Blockchain is a peer-to-peer, distributed ledger that is cryptographically-secure, append-only, immutable (extremely hard to change), and updateable only via consensus or agreement among peers.*

-
- We will look at all keywords in the definitions one by one.
 - **Peer-to-peer**
 - The first keyword in the technical definition is *peer-to-peer*.
 - *This means that there is no central controller in the network, and all participants talk to each other directly.*
 - This property allows for cash transactions to be exchanged directly among the peers without a third-party involvement, such as by a bank.

-

DISTRIBUTED LEDGER

- Dissecting the technical definition further reveals that blockchain is a *distributed ledger*, which simply means that a ledger is spread across the network among all peers in the network, and each peer holds a copy of the complete ledger.
- Cryptographically-secure
- Next, we see that this ledger is cryptographically-secure, which means that cryptography has been used to provide security services which make this ledger secure against tampering and misuse.
- These services include non-repudiation, data integrity, and data origin authentication

APPEND-ONLY

- Another property that we encounter is that blockchain is append-only, which means that data can only be added to the blockchain in time-ordered sequential order.
- This property implies that once data is added to the blockchain, it is almost impossible to change that data and can be considered practically immutable.
- Nonetheless, it can be changed in rare scenarios wherein collusion against the blockchain network succeeds in gaining more than 51 percent of the power.
- There may be some legitimate reasons to change data in the blockchain once it has been added, such as the right to be forgotten or right to erasure (also defined in General Data Protection (GDPR) ruling,

UPDATEABLE VIA CONSENSUS

- Finally, the most critical attribute of a blockchain is that it is updateable only via consensus.
- This is what gives it the power of decentralization. In this scenario, no central authority is in control of updating the ledger.
- Instead, any update made to the blockchain is validated against strict criteria defined by the blockchain protocol and added to the blockchain only after a consensus has been reached among all participating peers/nodes on the network.
- To achieve consensus, there are various consensus facilitation algorithms which ensure that all parties are in agreement about the final state of the data on the blockchain network and resolutely agree upon it to be true.

-
- Blockchain can be thought of as a layer of a distributed peer-to-peer network running on top of the internet, as can be seen in the following diagram.
 - It is analogous to SMTP, HTTP, or FTP running on top of TCP/IP

THE NETWORK VIEW OF A BLOCKCHAIN

can be seen in the following diagram. It is analogous to SMTP, HTTP, or FTP running on top of TCP/IP.

```
graph TD; U[Users / Nodes] <--> B1[Blockchain applications (smart contracts)]; U <--> B2[State machine]; U <--> B3[Consensus]; U <--> B4[Blocks]; U <--> B5[Transactions]; subgraph GreenOval; B1; B2; B3; B4; B5; end; B5 <--> P[Peer to Peer network]; P <--> I[The Internet];
```

The network view of a blockchain

At the bottom layer in the preceding diagram, there is the internet, which provides a basic communication layer for any network. In this case, a peer-to-peer network runs on top of the internet, which hosts another layer of

Activate Windows
Go to Settings to activate Windows

USD/INR +0.25%



- At the bottom layer in the preceding diagram, there is the internet, which provides a basic communication layer for any network.
- In this case, a peer-to-peer network runs on top of the internet, which hosts another layer of blockchain.
- That layer contains transactions, blocks, consensus mechanisms, state machines, and blockchain smart contracts.
- All of these components are shown as a single logical entity in a box, representing blockchain above the peer-to-peer network.
- Finally, at the top, there are users or nodes that connect to the blockchain and perform various operations such as consensus, transaction verification, and processing.

-
- From a business standpoint, a blockchain can be defined as a platform where peers can exchange value / electronic cash using transactions without the need for a centrally-trusted arbitrator.
 - For example, for cash transfers, banks act as a trusted third party.
 - In financial trading, a central clearing house acts as an arbitrator between two trading parties.
 - This concept is compelling, and once you absorb it, you will realize the enormous potential of blockchain technology.
 - This disintermediation allows blockchain to be a decentralized consensus mechanism where no single authority is in charge of the database.
 - Immediately, you'll see a significant benefit of decentralization here, because if no banks or central clearing houses are required, then it immediately leads to cost savings, faster transaction speeds, and trust.

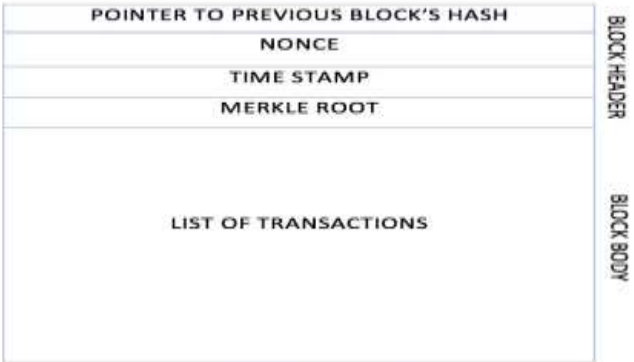
-
- A block is merely a selection of transactions bundled together and organized logically.
 - A transaction is a record of an event, for example, the event of transferring cash from a sender's account to a beneficiary's account.
 - A block is made up of transactions, and its size varies depending on the type and design of the blockchain in use.
 - A reference to a previous block is also included in the block unless it is a genesis block. A genesis block is the first block in the blockchain that is hardcoded at the time the blockchain was first started. The structure of a block is also dependent on the type and design of a blockchain.

-
- there are just a few attributes that are essential to the functionality of a block: the block header, which is composed of pointer to previous block, the timestamp, nonce, Merkle root, and the block body that contains transactions.
 - There are also other attributes in a block, but generally, the aforementioned components are always available in a block.

-
- A nonce is a number that is generated and used only once.
 - A nonce is used extensively in many cryptographic operations to provide replay protection, authentication, and encryption.
 - In blockchain, it's used in PoW consensus algorithms and for transaction replay protection.

-
- Merkle root is a hash of all of the nodes of a Merkle tree. Merkle trees are widely used to validate the large data structures securely and efficiently.
 - In the blockchain world, Merkle trees are commonly used to allow efficient verification of transactions. Merkle root in a blockchain is present in the block header section of a block, which is the hash of all transactions in a block.
 - This means that verifying only the Merkle root is required to verify all transactions present in the Merkle tree instead of verifying all transactions one by one.

THE GENERIC STRUCTURE OF A BLOCK.



The generic structure of a block.

This preceding structure is a simple block diagram that depicts a block. Specific block structures relative to their blockchain technologies will be discussed later in the book with greater in-depth technical detail.

Activate Windows
Go to Settings to activate Windows.

10:36
20-06-2023

CATEGORIES OF BLOCKCHAIN

- **I. Public/permissioned:**
- Allows applications to be deployed in production or removed without requiring anyone to be notified, reveal their identity, or meet any application criteria.
- The network's nodes, which make up the network and run the applications, must be invited to join.

- **2. Private/permissioned:**

- There is no decentralization in this type of network.
- Both the applications and the network nodes that run them must be invited to join the network and must meet certain requirements or provide proof of identity.
- Any party can be eliminated at any time without warning.

- **3. Public/permissionless:**

- This is the most decentralized type of network.

- Without having to notify anyone, reveal their identity, or meet any application criteria requirements, applications can be deployed in production or removed.

- In addition, the network's nodes can join and contribute freely and anonymously, usually in exchange for the network's native cryptocurrency

-
- 4. Private / Permissionless:
 - Requires that applications in production be invited to join the network and that they can be removed at any time without warning.
 - The nodes that make up the network and run the applications can join and contribute freely and anonymously, usually in exchange for the network's native cryptocurrency

CONSORTIUM BLOCKCHAIN:

- Consortium blockchains are managed and run by a number of organizations or entities.
- As a permissioned blockchain, users must be asked to join and have authorization before they can access the network.
- • The upkeep of the blockchain network and transaction verification are divided among the participating groups in a consortium blockchain.
- Instead of being under the control of a singular central authority, the member organizations manage the nodes that verify transactions and keep the blockchain.

-
- In sectors where multiple organizations must work together on a single platform while keeping control over their data and transactions, this kind of blockchain is frequently used.
 - • In general, consortium blockchains provide a balance between decentralization and control, making them appropriate for use cases where a number of well-known and trustworthy parties must collaborate on a common platform

BLOCKCHAIN NETWORK

- A block chain network is basically a technical network that is providing ledger and smart contract (chain code) services to the applications.
- Primarily, these smart contracts are used to generate transactions which are subsequently distributed among every peer node in the network where they are unalterably recorded on their copy of the ledger.
- The users of applications must be end users using client applications.

-
- Block chain underpins thousands of applications and networks that have some utility and are able to provide value in various industries like fashion, finance and gaming.
 - With the help of the decentralized nature of block chain networks industries like crypto currency and decentralized finance come into action.
 - The roots of the block chain network can be traced back to cryptographers from the early 1980's.
 - Although the modern block chain network began with Bit coin, digital payments began in January 2009 by Santos Nakamoto
 -

TYPES OF BLOCKCHAIN NETWORKS

- Blockchain network broadly are classified into 4 types and they are:
- Public blockchain, Private blockchain, Consortium blockchain, Hybrid blockchain.
- **Public Blockchain:** Public Blockchain network is accessible to the public without any limitations of validation and participants.
- As it does not have any central authority an individual is unable to manipulate the network that stores unchangeable data.

-
- **Private Blockchain:** It is a restrictive network, only authorised people have access to it.
 - The authorisers have the power to choose entities and blockchain developers and they are granted permission during the development phase.
 - **Consortium Blockchain:** It is a process where the consensus process is controlled by a pre- selected set of nodes.
 - a semi- decentralised network where multiple entities operate.

-
- **Hybrid Blockchain:** It is a network that possesses properties of both private and public blockchain networks.
 - It provides versatile nodes wherein the individual can set preferences to share or discrete data.

USES OF BLOCKCHAIN NETWORK

- Banking and Finance
- Real Estate
- **Authentication for Staff Credentials**
- Healthcare
- Media

WHAT ARE BLOCKCHAIN NODES?

- Blockchain nodes are network stakeholders and their devices are authorized to keep track of the distributed ledger and serve as communication hubs for various network tasks.
-
- A Blockchain node's primary job is to confirm the legality of each subsequent batch of network transactions, known as blocks.
- In addition, allocating a unique identifier to each node in the network helps to distinguish a node from other nodes easily.

-
- A Proof-of-Work (PoW) Blockchain, such as Bitcoin (BTC) or Monero (XMR), includes miners who are responsible for the following.
 -
 - Only “full nodes” must store all Blockchain transactions on their devices.
 - These nodes are in charge of validating blocks and transactions.
 -
 - On the other hand, lightweight nodes have low storage requirements because they just need to download block headers to verify transactions.
 - A block reward is not always included in either of these versions of a full node.

FUNCTIONS OF NODES

- A block broadcasts all the network nodes when a miner seeks to add a new block of transactions to the Blockchain.
- Based on the legitimacy of a block, nodes might accept or reject it (validity of signatures and transactions).
- When a node accepts a new block of transactions, it saves and stores it on top of the existing blocks.
- In a nutshell, nodes do the following:
-

-
- Nodes determine whether or not a block of transactions is legitimate and accept or reject it.
 - Nodes save and store transaction blocks (storing Blockchain transaction history).
 - This transaction history is broadcast and disseminated by nodes to other nodes that may need to synchronize with the Blockchain (updates on transaction history are important).

PEER-TO-PEER NETWORK

- In the simplest terms, a peer-to-peer network is a network created whenever two or more devices (usually a computer) are connected and share resources.
- But what creates the main difference here is that a peer-to-peer network, unlike conventional network systems, forms an ecosystem where the computers are connected through a single server computer.
- It can also be seen as a network where multiple computer systems are connected through a single server that enables the transfer of files from one end to the other

P2P (PEER-TO-PEER) NETWORK ARCHITECTURE?

- Peer-to-peer network architecture is a kind of network where there is absolutely no division of activities among various other sections.
- Every node performs the same task and set of actions where each device serves the purpose of both the server and the client.
- Response of network architecture established over the computer networking ecosystem under this model is such that each and every workstation is responsible for equal tasks but fewer devices are connected to the main server

-
- **Structured networks:** Unlike unstructured networks, structured ones allow each peer to look after a specific section of the content over the network.
 - These networks assign a specific value to each content and peer in the network which is then followed by a common protocol that determines which section is responsible for which part of the content.
 - In this way, whenever someone reaches out to a peer to search for content, the network uses the common protocol to determine the section responsible for data transfer and direct the search query towards the peer responsible for it.
 -
 - Examples – Tixati, Kademlia, P-Grid, etc.

-
- **Unstructured networks:** An unstructured peer-to-peer network is one where the links in the network are established randomly.
 - Such networks are easy to construct as any new peer that would like to join and contribute to the network can do so by copying the existing links of another section and then forming and spreading its own links.
 -
 - Examples – Napster, KaZaA, Gnutella, etc.

CHARACTERISTICS OF P2P NETWORK

- **Network topology**
- **Security**
- **Scalability**
- **Resource Sharing**
- **Decentralization**

TYPES OF P2P NETWORK

- **Centralized P2P Network**
- In a centralized P2P network, there is a central server or hub that manages all communication between nodes.
- Nodes in the network do not communicate directly with each other, but instead communicate through the central server.
- This type of P2P network is often used in applications
- such as online gaming and chat rooms, where a centralized server is needed to manage user authentication, content distribution, and other tasks.
-

DECENTRALIZED P2P NETWORK

- In a decentralized P2P network, there is no central point of control or authority.
- Nodes communicate directly with each other, and there is no hierarchy or structure to the network.
- This type of P2P network is often used in applications such as file sharing, where users share files directly with each other without the need for a central server.

HYBRID P2P NETWORK

- In a hybrid P2P network, there is a combination of centralized and decentralized elements.
- Some nodes in the network act as hubs or servers that facilitate communication between other nodes, while other nodes communicate directly with each other.
- This type of P2P network is often used in applications such as online marketplaces, where a central server is needed to manage transactions and user authentication, but direct communication between users is also desirable.

- **Advantages of P2P Network**

- **No Centralized Authority**

- **Cost-effective**

- **Scalable**

- **Privacy and Security**

-

- **Disadvantages of P2P Network**

- **Difficulty in Implementation**

- **Network Management**

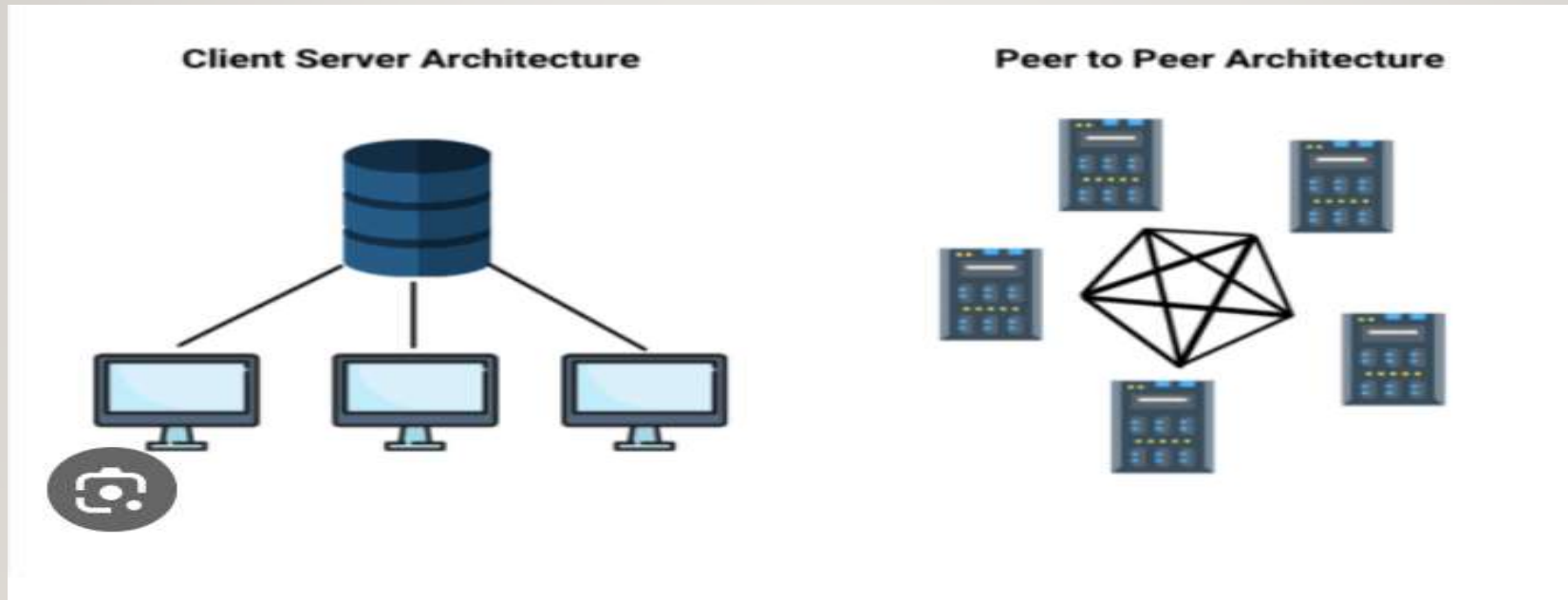
- **Security Risks**

- **Limited Control**

-
- **Applications of P2P Network**
 - **File Sharing**
 - **Blockchain**
 - **Gaming**
 - **Messaging and Communication**

WHAT EXACTLY IS BLOCKCHAIN MINING?

- A peer-to-peer computer process, Blockchain mining is used to secure and



-
- verify bitcoin transactions. Mining involves Blockchain miners who add bitcoin transaction data to Bitcoin's global public ledger of past transactions.
 - In the ledgers, blocks are secured by Blockchain miners and are connected to each other forming a chain.
 -
 - When we talk in-depth, as opposed to traditional financial services systems, **Bitcoins** have no central clearinghouse.
 - Bitcoin transactions are generally verified in decentralized clearing systems wherein people contribute computing resources to verify the same.
 - This process of verifying transactions is called mining.

TYPES OF MINING

- The process of mining can get really complex and a regular desktop or PC cannot cut it. Hence, it requires a unique set of hardware and software that works well for the user. It helps to have a custom set specific to mining certain blocks.
- The mining process undertaking can be divided into three categories:


- **I. Individual Mining**

- When mining is done by an individual, user registration as a miner is necessary.
- As soon as a transaction takes place, a mathematical problem is given to all the single users in the blockchain network to solve.
- The first one to solve it gets rewarded.
- Once the solution is found, all the other miners in the blockchain network will validate the decrypted value and then add it to the blockchain. Thus, verifying the transaction.
-

- **2. Pool Mining**

- In pool mining, a group of users works together to approve the transaction.
- Sometimes, the complexity of the data encrypted in the blocks makes it difficult for a user to decrypt the encoded data alone.
- So, a group of miners works as a team to solve it.
- After the validation of the result, the reward is then split between all users.


- **3. Cloud Mining**

- Cloud mining eliminates the need for computer hardware and software.
 - It's a hassle-free method to extract blocks.
 - With cloud mining, handling all the machinery, order timings, or selling profits is no longer a constant worry.
 - While it is hassle-free, it has its own set of disadvantages.
 - The operational functionality is limited with the limitations on bitcoin hashing in blockchain.
 - The operational expenses increase as the reward profits are low.
 - Software upgrades are restricted and so is the verification process
- 

GENERIC ELEMENTS OF A BLOCKCHAIN

- In this section, the generic elements of blockchain are presented. More precise elements will be discussed in the context of their respective
-
- blockchains in later chapters, for example, the Ethereum blockchain.
-

- **Addresses**

- Addresses are unique identifiers that are used in a transaction on the blockchain to denote senders and recipients.
 - An address is usually a public key or derived from a public key. While addresses can be reused by the same user, addresses themselves are unique.
 - In practice, however, a single user may not use the same address again and generate a new one for each transaction.
 - This newly generated address will be unique. Bitcoin is in fact a pseudonymous system. End users are usually not directly identifiable but some research in de-anonymizing bitcoin users have shown that users can be identified successfully.
 - As a good practice it is suggested that users generate a new address for each transaction in order to avoid linking transactions to the common owner, thus avoiding identification.
- 

- **Transaction**

- A transaction is the fundamental unit of a blockchain. A transaction represents a transfer of value from one address to another.

-

- **Block**

- A block is composed of multiple transactions and some other elements such as the previous block hash (hash pointer), timestamp, and nonce.

-

- **Peer-to-peer network**

- As the name implies, this is a network topology whereby all peers can communicate with each other and send and receive messages.

- **Scripting or programming language**

- This element performs various operations on a transaction.
- Transaction scripts are predefined sets of commands for nodes to transfer tokens from one address to another and perform various other functions.
- Turing complete programming language is a desirable feature of blockchains; however, the security of such languages is a key question and an area of important and ongoing research

-
- **Virtual machine**
 - This is an extension of a transaction script.
 - A virtual machine allows Turing complete code to be run on a blockchain (as smart contracts) whereas a transaction script can be limited in its operation.
 - Virtual machines are not available on all blockchains; however, various blockchains use virtual machines to run programs, for example **Ethereum Virtual Machine (EVM)** and **Chain Virtual Machine (CVM)**.

- **State machine**

- A blockchain can be viewed as a state transition mechanism whereby a state is modified from its initial form to the next and eventually to a final form as a result of a transaction execution and validation process by nodes.

-

- **Nodes**

- A node in a blockchain network performs various functions depending on the role it takes.
- A node can propose and validate transactions and perform mining to facilitate consensus and secure the blockchain.
- This is done by following a consensus protocol.
- (Most commonly this is PoW.)
- Nodes can also perform other functions such as simple payment verification (lightweight nodes), validators, and many others functions depending on the type of the blockchain used and the role assigned to the node.

- **Smart contracts**

- These programs run on top of the blockchain and encapsulate the business logic to be executed when certain conditions are met.

- The smart contract feature is not available in all blockchains but is now becoming a very desirable feature due to the flexibility and power it provides to the blockchain applications

- **Features of a blockchain**

- A blockchain performs various functions. These are described below in detail.

-

- **Distributed consensus**

- Distributed consensus is the major underpinning of a blockchain. This enables a blockchain to present a single version of truth that is agreed upon by all parties without the requirement of a central authority.

- **Transaction verification**

- Any transactions posted from nodes on the blockchain are verified based on a predetermined set of rules and only valid transactions are selected for inclusion in a block.

-

- **Platforms for smart contracts**

- A blockchain is a platform where programs can run that execute business logic on behalf of the users.

- As explained earlier, not all blockchains have a mechanism to execute smart contracts; however, this is now a very desirable feature.

-



- **Transferring value between peers**

- Blockchain enables the transfer of value between its users via tokens.
- Tokens can be thought of as a carrier of value.

-

- **Generating cryptocurrency**

- This is an optional feature depending on the type of blockchain used.
- A blockchain can generate cryptocurrency as an incentive to its miners who validate the transactions and spend resources in order to secure the blockchain.

-

- **Smart property**

- For the first time it is possible to link a digital or physical asset to the blockchain in an irrevocable manner, such that it cannot be claimed by anyone else; you are in full control of your asset and it cannot be double spent or double owned.
- Compare it with a digital music file, for example, which can be copied many times without any control; on a blockchain, however, if you own it no one else can claim it unless you decide to transfer it to someone.
- This feature has far-reaching implications especially in **Digital Rights Management (DRM)** and electronic cash systems where double spend detection is a key requirement. The double spend problem was first solved in bitcoin.

- **Immutability**

- This is another key feature of blockchain: records once added onto the blockchain are immutable.
- There is the possibility of rolling back the changes but this is considered almost impossible to do as it will require an unaffordable amount of computing resources.
- For example, in much desirable case of bitcoin if a malicious user wants to alter the previous blocks then it would require computing the PoW again for all those blocks that have already been added to the blockchain.
- This difficulty makes the records on a blockchain practically immutable

- **Uniqueness**

- This feature of blockchain ensures that every transaction is unique and has not been spent already.
- This is especially relevant in cryptocurrencies where much desirable detection and avoidance of double spending are a key requirement

- **Types of Blockchain**

-

- **There are 4 types of blockchain:**

-

- **Public Blockchain.**

-

- **Private Blockchain.**

-

- **Hybrid Blockchain.**

-

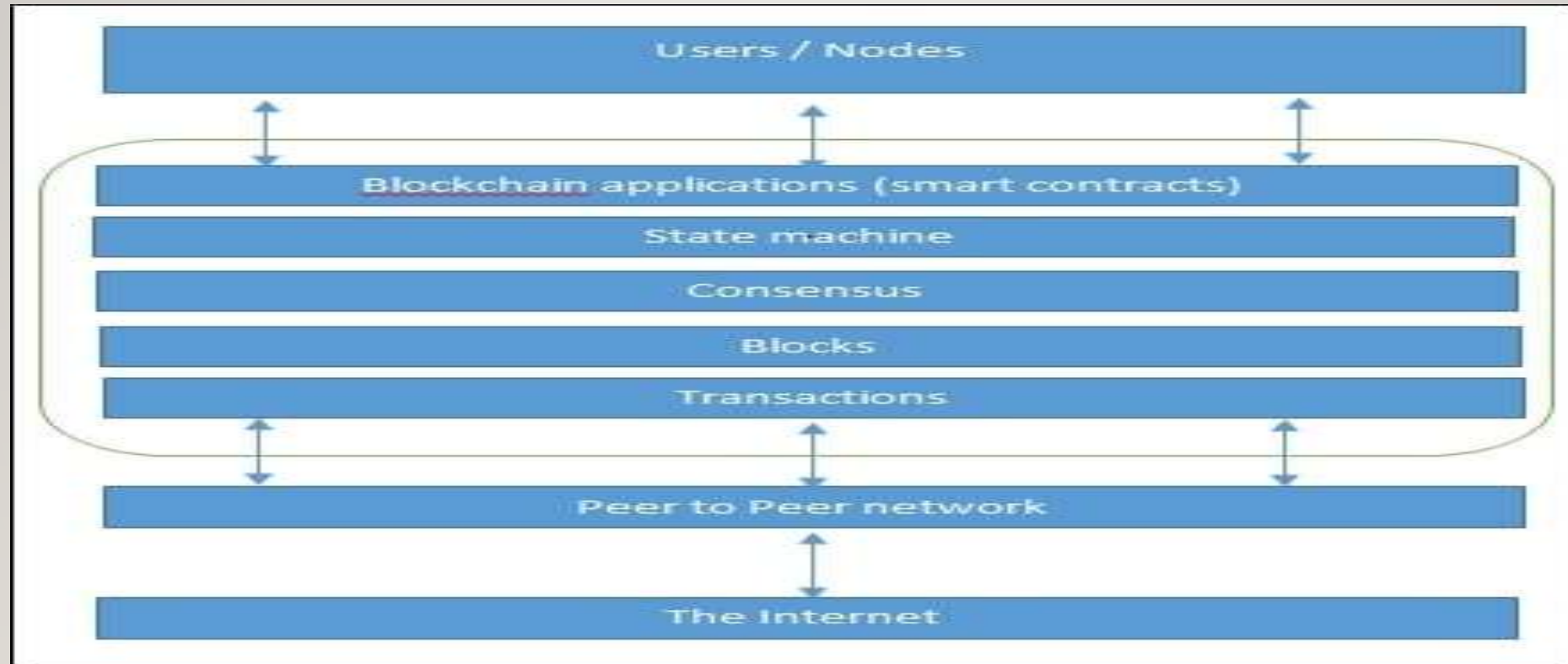
- **Consortium Blockchain.**

PUBLIC BLOCKCHAIN

- These blockchains are completely open to following the idea of decentralization.
- They don't have any restrictions, anyone having a computer and internet can participate in the network.
- As the name is public this blockchain is open to the public, which means it is not owned by anyone.
- Anyone having internet and a computer with good hardware can participate in this public blockchain.
- All the computer in the network hold the copy of other nodes or block present in the network
- In this public blockchain, we can also perform verification of transactions or records

ADVANTAGES:

- **Trustable:** There are algorithms to detect no fraud.
- Participants need not worry about the other nodes in the network
- **Secure:** This blockchain is large in size as it is open to the public.
- In a large size, there is greater distribution of records.
- **Anonymous Nature:** It is a secure platform to make your transaction properly at the same time, you are not required to reveal your name and identity in order to participate.



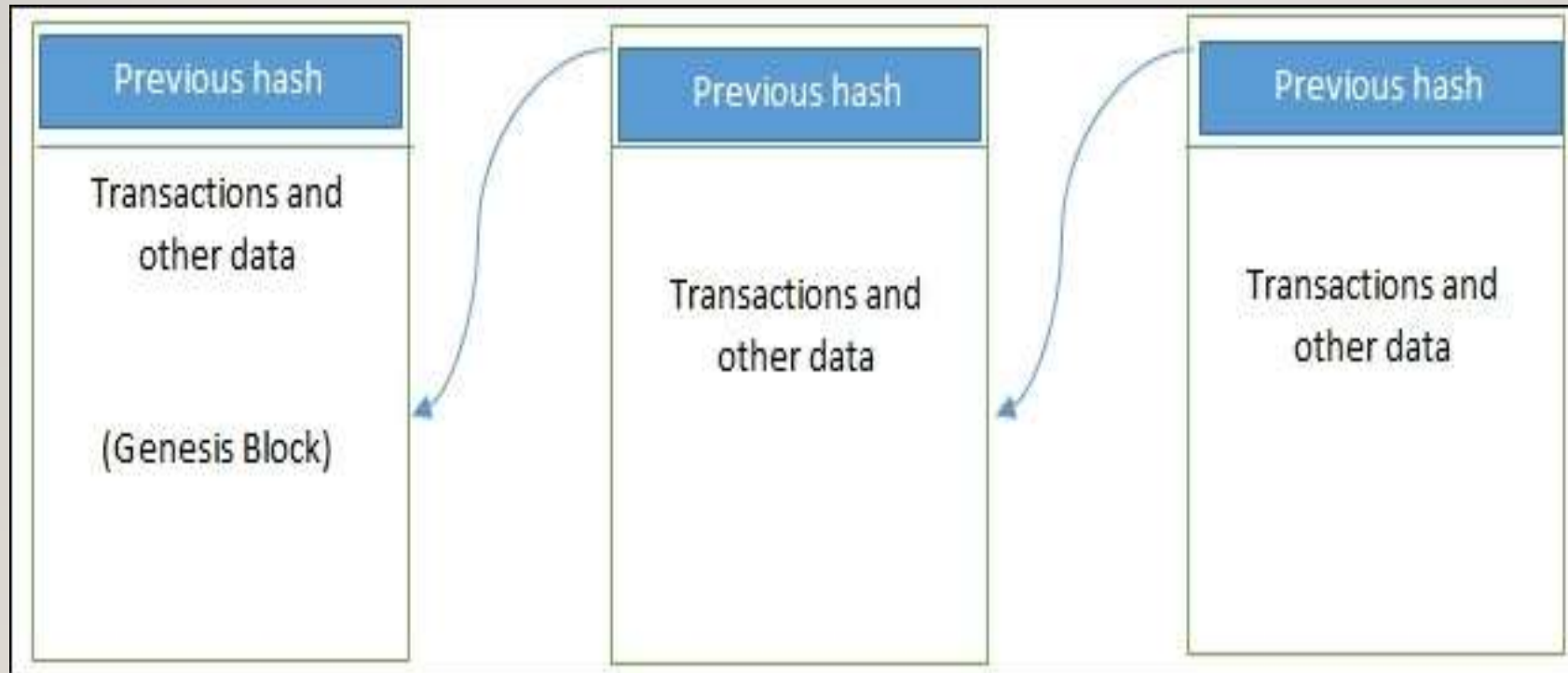
-
- **Decentralized:** There is no single platform that maintains the network, instead every user has a copy of the ledger.
 - **Disadvantages:**
 - **Processing:** The rate of the transaction process is very slow, due to its large size. Verification of each node is a very time-consuming process.
 - **Energy Consumption:** Proof of work is high energy-consuming. It requires good computer hardware to participate in the network.

Previous blocks hash

Nonce

Transactions

-
- **Acceptance:** No central authority is there so governments are facing the issue to implement the technology faster.
 - **2. Private Blockchain**
 - These blockchains are not as decentralized as the public blockchain only selected nodes can participate in the process, making it more secure than the others.
 - These are not as open as a public blockchain.



-
- They are open to some authorized users only.
 - These blockchains are operated in a closed network.
 - In this few people are allowed to participate in a network within a company/organization.
 - **Advantages:**
 - **Speed:** The rate of the transaction is high, due to its small size. Verification of each node is less time-consuming.

-
- **Scalability:** We can modify the scalability. The size of the network can be decided manually.
 - **Privacy:** It has increased the level of privacy for confidentiality reasons as the businesses required.
 - **Balanced:** It is more balanced as only some user has the access to the transaction which improves the performance of the network.

- **Disadvantages:**

- **Security-** The number of nodes in this type is limited so chances of manipulation are there. These blockchains are more vulnerable.
- **Centralized-** Trust building is one of the main disadvantages due to its central nature. Organizations can use this for malpractices.
- **Count-** Since there are few nodes if nodes go offline the entire system of blockchain can be endangered.

- **3. Hybrid Blockchain**

- It is the mixed content of the private and public blockchain, where some part is controlled by some organization and other makes are made visible as a public blockchain.
- It is a combination of both public and private blockchain.
- Permission-based and permissionless systems are used.
- User access information via smart contracts
- Even a primary entity owns a hybrid blockchain it cannot alter the transaction

- **Advantages:**

- **Ecosystem:** Most advantageous thing about this blockchain is its hybrid nature. It cannot be hacked as 51% of users don't have access to the network

- **Cost:** Transactions are cheap as only a few nodes verify the transaction. All the nodes don't carry the verification hence less computational cost.

- **Architecture:** It is highly customizable and still maintains integrity, security, and transparency.

- **Operations:** It can choose the participants in the blockchain and decide which transaction can be made public.

-



- **Disadvantages:**


- **Efficiency:** Not everyone is in the position to implement a hybrid Blockchain. The organization also faces some difficulty in terms of efficiency in maintenance.

- **Transparency:** There is a possibility that

- someone can hide information from the user. If someone wants to get access through a hybrid blockchain it depends on the organization whether they will give or not.

- **Ecosystem:** Due to its closed ecosystem this blockchain lacks the incentives for network participation.

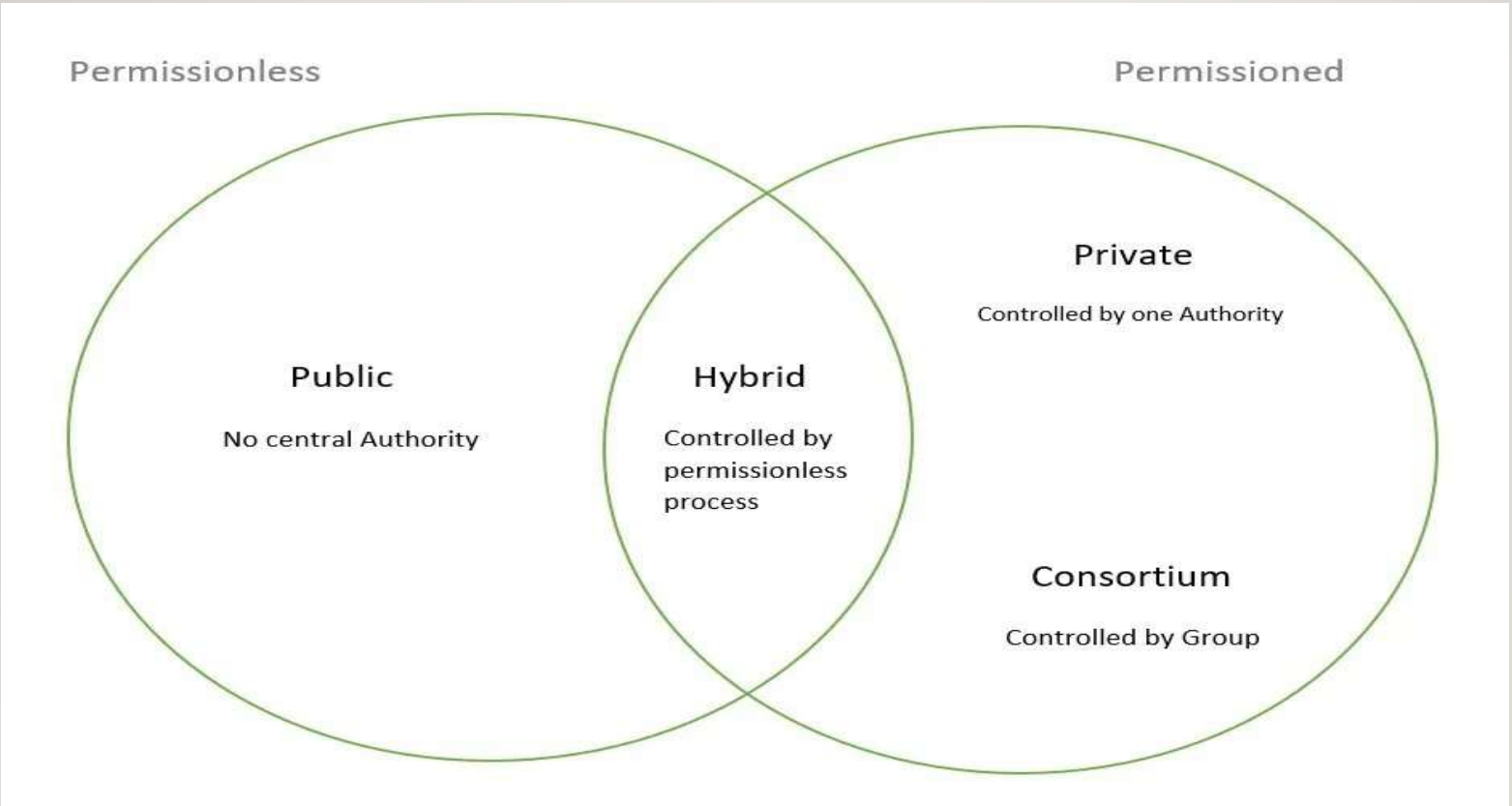
- **4. Consortium Blockchain**

- It is a creative approach that solves the needs of the organization. This blockchain validates the transaction and also initiates or receives transactions.
 - Also known as Federated Blockchain.
 - This is an innovative method to solve the organization's needs.
 - Some part is public and some part is private.
 - In this type, more than one organization manages the blockchain.
- 

- **Advantages:**

- **Speed:** A limited number of users make verification fast. The high speed makes this more usable for organizations.
- **Authority:** Multiple organizations can take part and make it decentralized at every level. Decentralized authority, makes it more secure.
- **Privacy:** The information of the checked blocks is unknown to the public view. but any member belonging to the blockchain can access it.
- **Flexible:** There is much divergence in the flexibility of the blockchain. Since it is not a very large decision can be taken faster.

-
- **Disadvantages:**
 - **Approval:** All the members approve the protocol making it less flexible. Since one or more organizations are involved there can be differences in the vision of interest.
 - **Transparency:** It can be hacked if the organization becomes corrupt. Organizations may hide information from the users.
 - **Vulnerability:** If few nodes are getting compromised there is a greater chance of vulnerability in this blockchain





● *****



BLOCKCHAIN TECHNOLOGY

UNIT-2 BLOCKCHAIN ARCHITECTURE

UNIT – II BLOCKCHAIN ARCHITECTURE:

- ▶ Operation of Bitcoin Blockchain, Blockchain Architecture – Block, Hash, Distributer P2P, Structure of Blockchain- Consensus mechanism: Proof of Work (PoW), Proof of Stake (PoS), Byzantine Fault Tolerance (BFT), Proof of Authority (PoA) and Proof of Elapsed Time (PoET)

OPERATION OF BITCOIN BLOCKCHAIN:

Bitcoin blockchain:

The bitcoin blockchain is a public ledger that records bitcoin transactions. It is implemented as a chain of blocks, each block containing a cryptographic hash of the previous block up to the genesis block in the chain. A network of communicating nodes running bitcoin software maintains the blockchain.

Wallets:

Bitcoin wallets are software applications or physical devices used to store, manage, and transact with Bitcoin. Wallets generate and store cryptographic keys, which are used to access and transfer ownership of Bitcoin. There are different types of wallets, including software wallets, hardware wallets, and paper wallets.

Mining:

Bitcoin mining involves the process of validating and adding new transactions to the blockchain. Miners use specialized hardware and computational power to solve complex mathematical puzzles, which allows them to add new blocks to the blockchain. Miners are rewarded with newly minted bitcoins and transaction fees for their mining efforts.

Transactions:

Bitcoin transactions involve the transfer of ownership of bitcoins from one wallet to another. Each transaction is broadcasted to the Bitcoin network, and miners include these transactions in blocks. Transactions require cryptographic signatures to ensure the authenticity and security of the transfers.

Security:

Bitcoin operations rely on cryptographic algorithms to ensure the security and integrity of the network. Public-key cryptography is used to generate and verify digital signatures, while hash functions are used to secure the blockchain. Additionally, users are encouraged to implement strong security measures to protect their wallets and private keys.

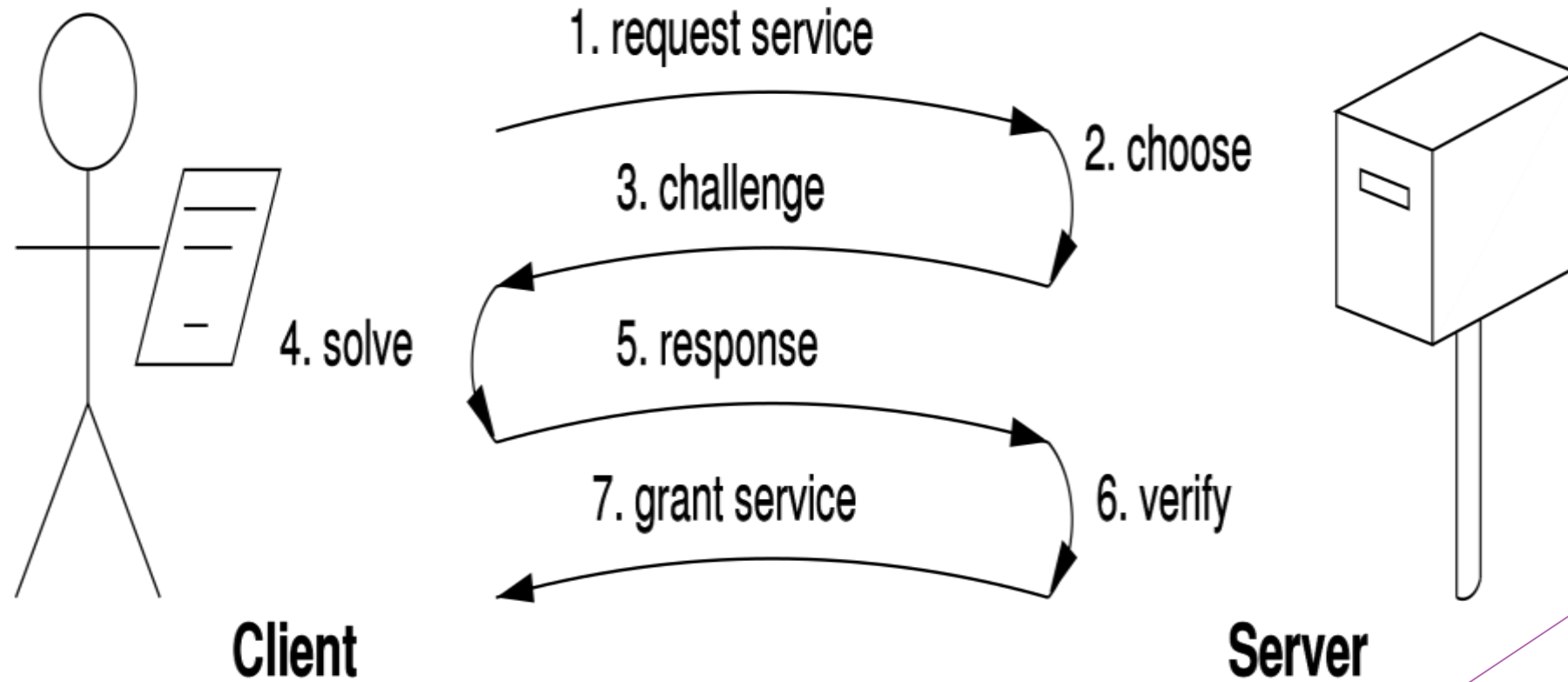
Exchanges:

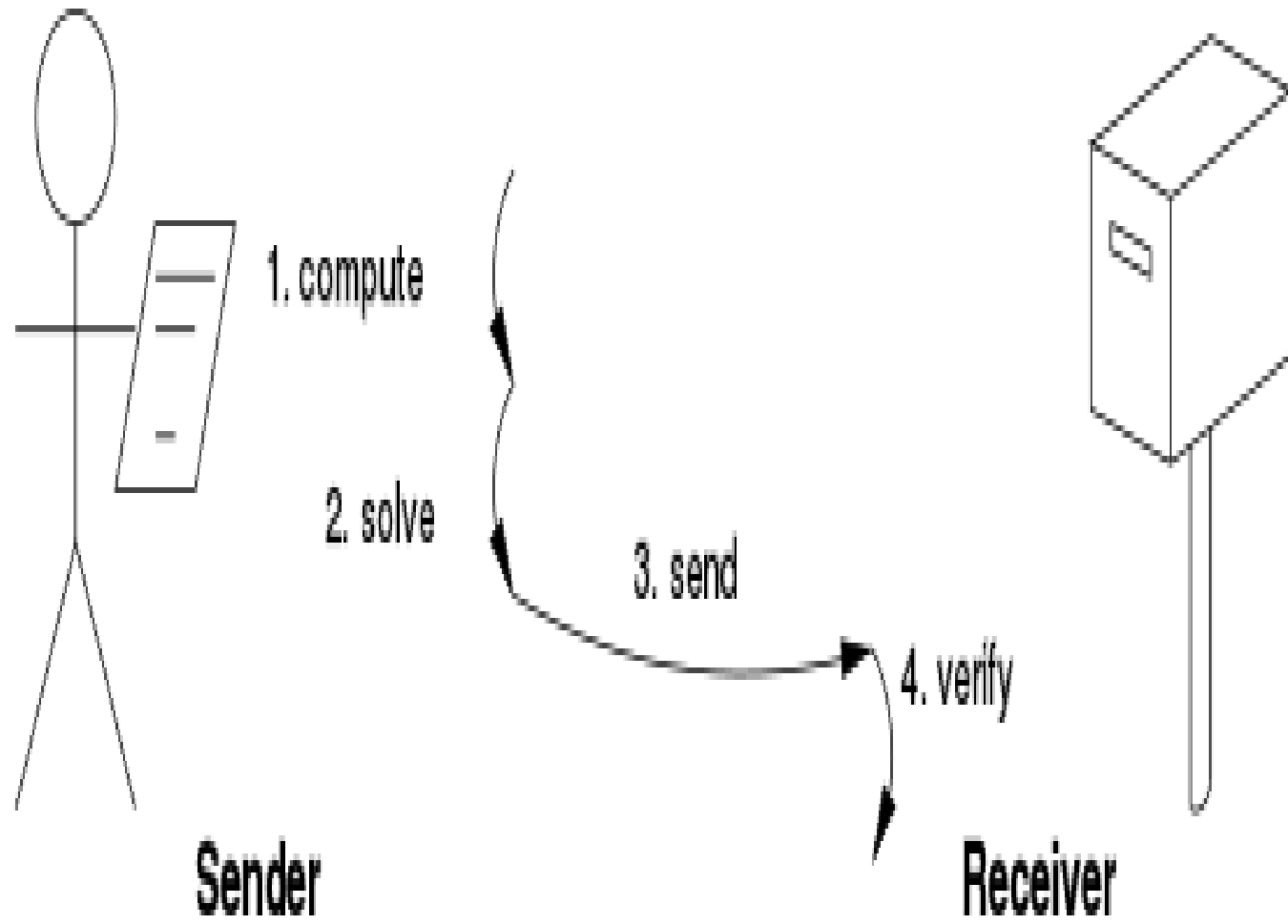
Bitcoin can be bought, sold, and traded on cryptocurrency exchanges. These exchanges act as intermediaries, matching buyers and sellers and facilitating the exchange of Bitcoin for fiat currencies or other cryptocurrencies. Exchanges provide trading platforms, wallets, and various tools to manage Bitcoin holdings.

Consensus:

Bitcoin operates on a consensus mechanism called proof-of-work (PoW). Miners compete to solve complex mathematical puzzles, and the first miner to solve the puzzle and validate a block is rewarded. This decentralized consensus mechanism ensures the security and immutability of the blockchain.

PROOF -OF-WORK(POW)





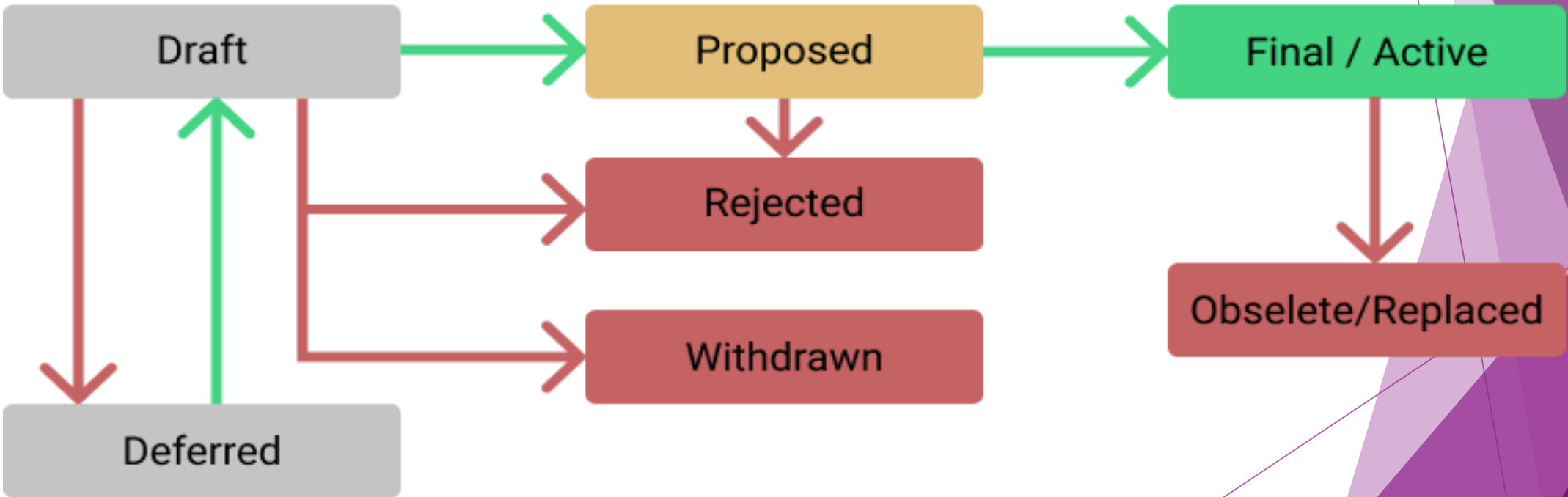
Governance:

Bitcoin's governance is decentralized, with decisions being made through a consensus of network participants. Changes to the Bitcoin protocol are proposed, debated, and implemented through a process called "Bitcoin Improvement Proposals" (BIPs). Developers, miners, and users contribute to the decision-making process.

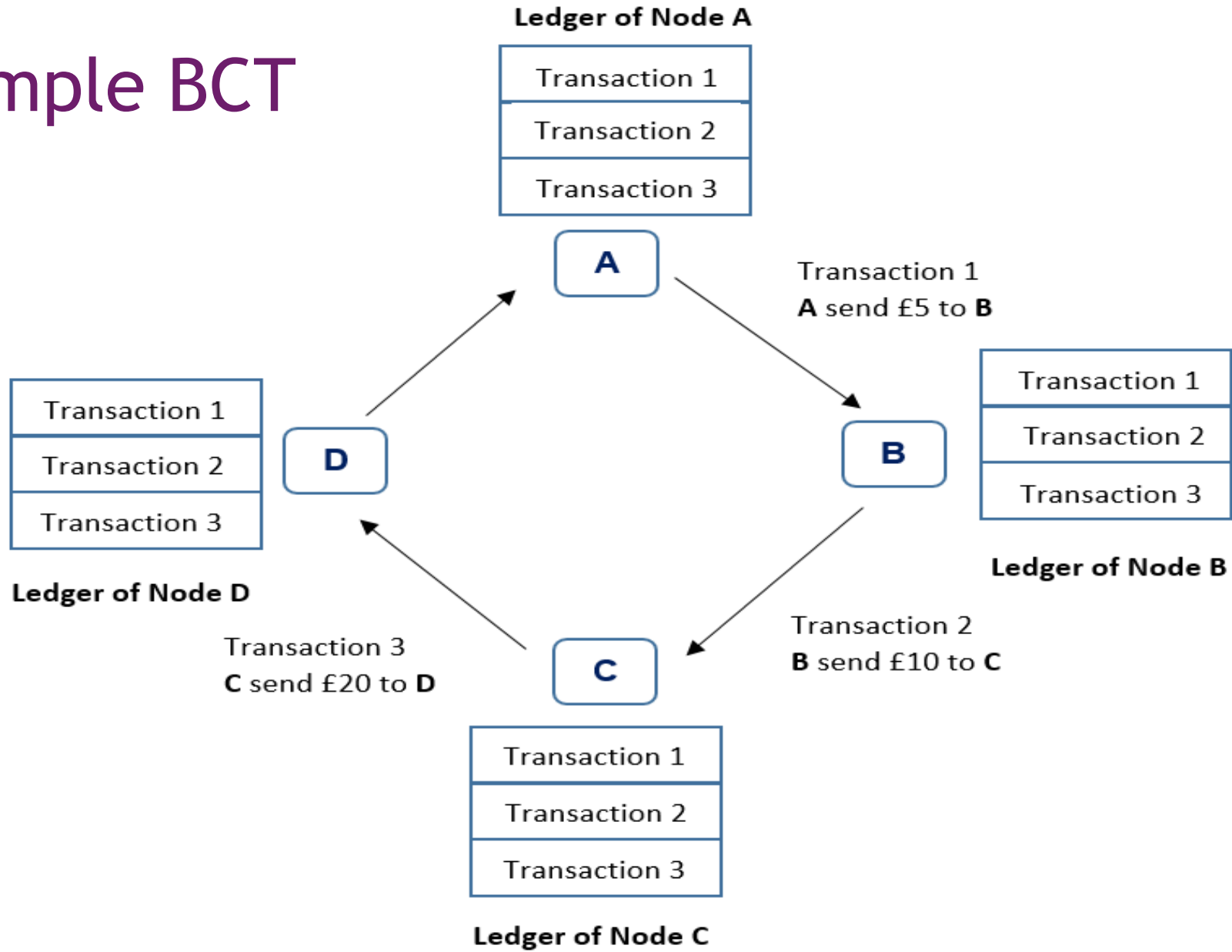
It's important to note that the operation of Bitcoin is a dynamic and evolving field. New technologies, upgrades, and developments continue to shape the ecosystem and improve the efficiency, scalability, and security of Bitcoin operations.

Bitcoin Improvement Proposal Process

■ Inactive ■ Optional / Being Activated ■ Active



Simple BCT

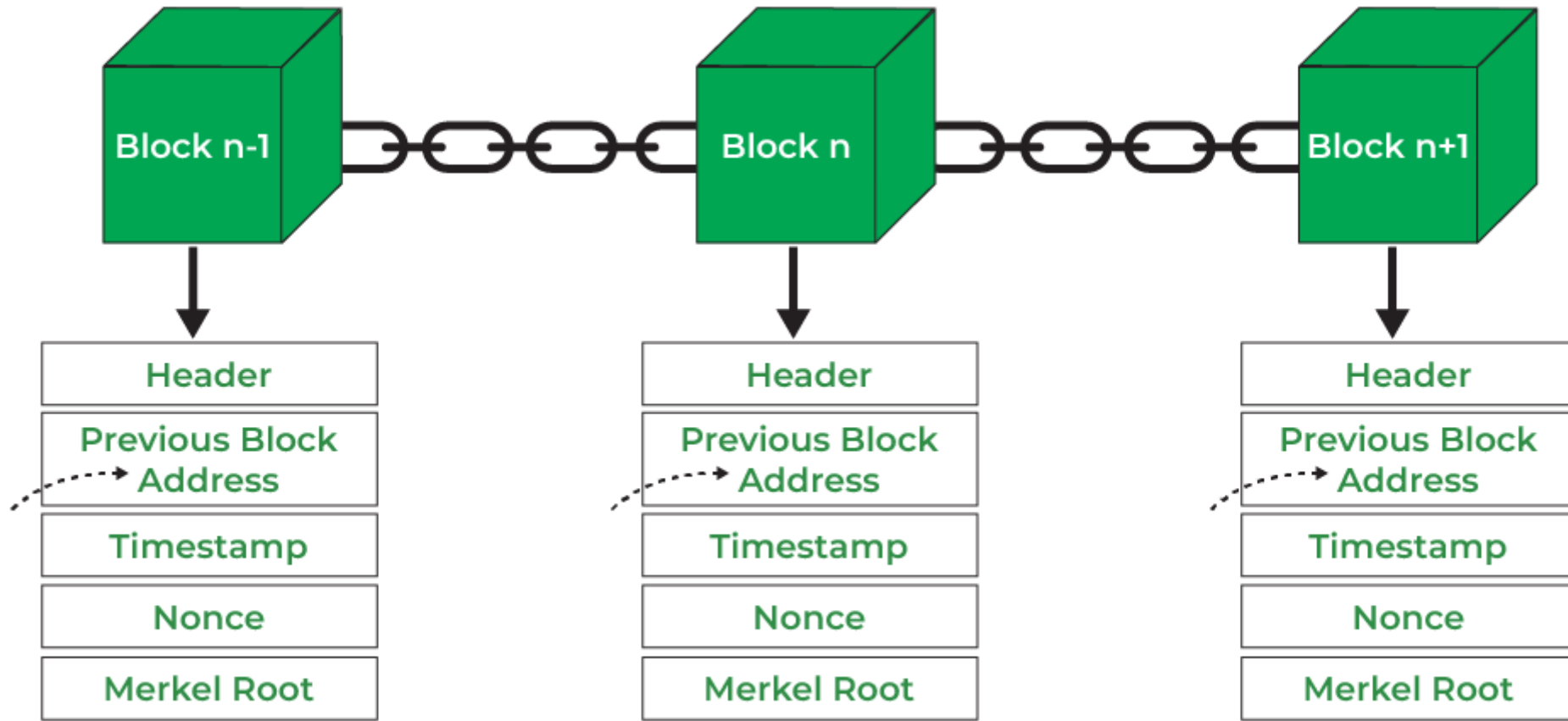


BLOCKCHAIN ARCHITECTURE:

Blockchain is a technology where multiple parties involved in communication can perform different transactions without third-party intervention. Verification and validation of these transactions are carried out by special kinds of nodes.

Benefits of Blockchain:

- It is safer than any other technology.
- To avoid possible legal issues, a trusted third party has to supervise the transactions and validate the transactions.
- There's no one central point of attack.
- Data cannot be changed or manipulated, it's immutable.



Header:

It is used to identify the particular block in the entire blockchain. It handles all blocks in the blockchain. A block header is hashed periodically by miners by changing the nonce value as part of normal mining activity, also Three sets of block metadata are contained in the block header.

Previous Block Address/ Hash:

It is used to connect the $i+1^{\text{th}}$ block to the i^{th} block using the hash. In short, it is a reference to the hash of the previous (parent) block in the chain.

Timestamp:

It is a system verify the data into the block and assigns a time or date of creation for digital documents. The timestamp is a string of characters that uniquely identifies the document or event and indicates when it was created.

Nonce:

A nonce number which uses only once. It is a central part of the proof of work in the block. It is compared to the live target if it is smaller or equal to the current target. People who mine, test, and eliminate many Nonce per second until they find that Valuable Nonce is valid.

Merkel Root:

It is a type of data structure frame of different blocks of data. A [Merkle Tree](#) stores all the transactions in a block by producing a digital fingerprint of the entire transaction. It allows the users to verify whether a transaction can be included in a block or not.

Types of Blockchain Architecture

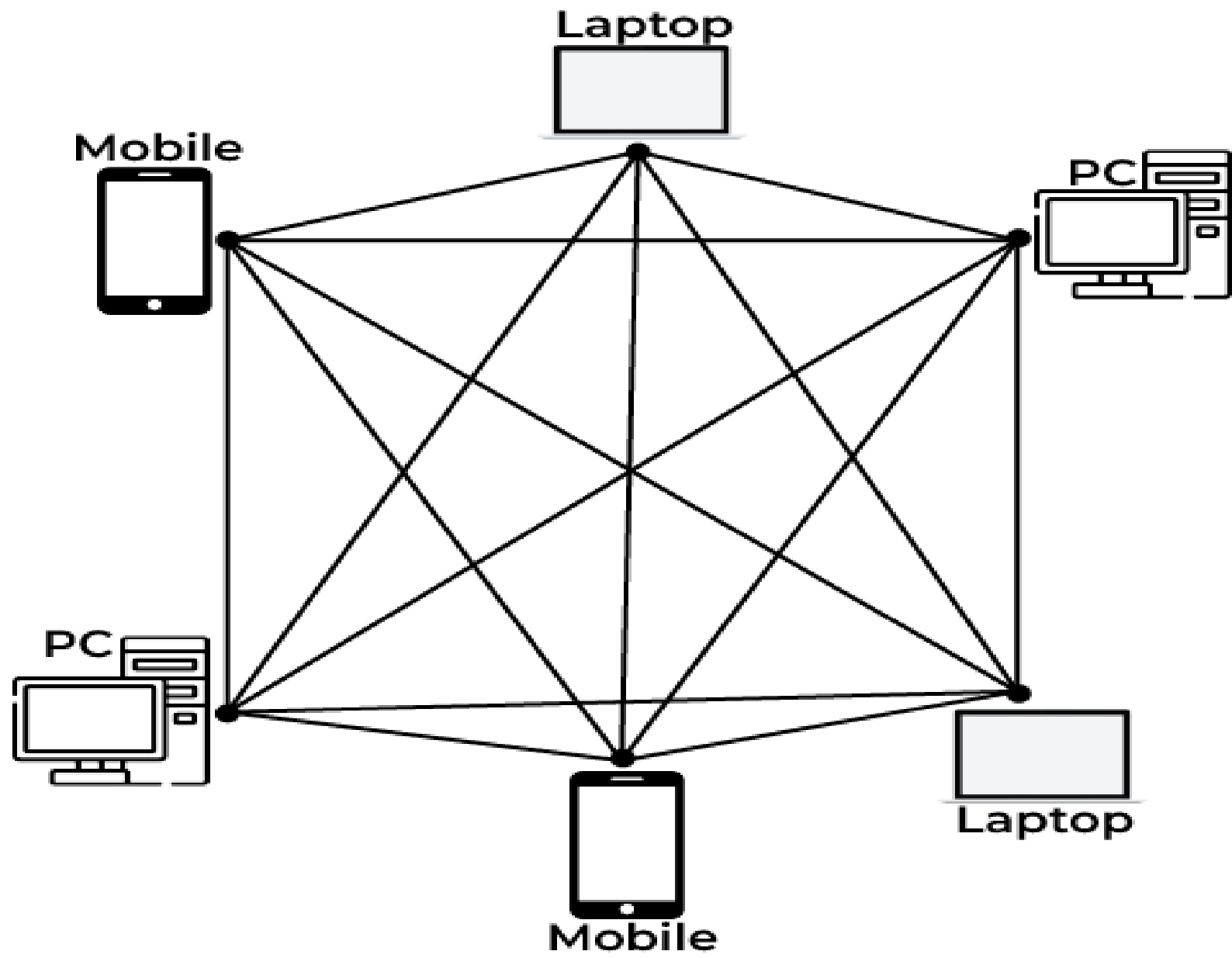
1. Public Blockchain:

- A public blockchain is a concept where anyone is free to join and take part in the core activities of the blockchain network.
- Anyone can read, write, and audit the ongoing activities on a public blockchain network, which helps to achieve the self-determining, decentralized nature often authorized when blockchain is discussed.
- Data on a public blockchain is secure as it is not possible to modify once they are validated.
- The public blockchain is fully decentralized, it has access and control over the ledger, and its data is not restricted to persons, is always available and the central authority manages all the blocks in the chain.

- There is publicly running all operations. Due to no one handling it singly then there is no need to get permission to access the public blockchain. Anyone can set his/her own node or block in the network/ chain.
- After a node or a block settled in the chain of the blocks, all the blocks are connected like peer-to-peer connections. If someone tries to attack the block then it forms a copy of that data and it is accessible only by the original author of the block.

Advantages:

1. A public network operates on an actuate scheme that encourages new persons to join and keep the network better.
2. There is no agreement in the public blockchain.
3. This means that a public blockchain network is immutable.
4. It has Rapid transactions.



2. Private Blockchain:

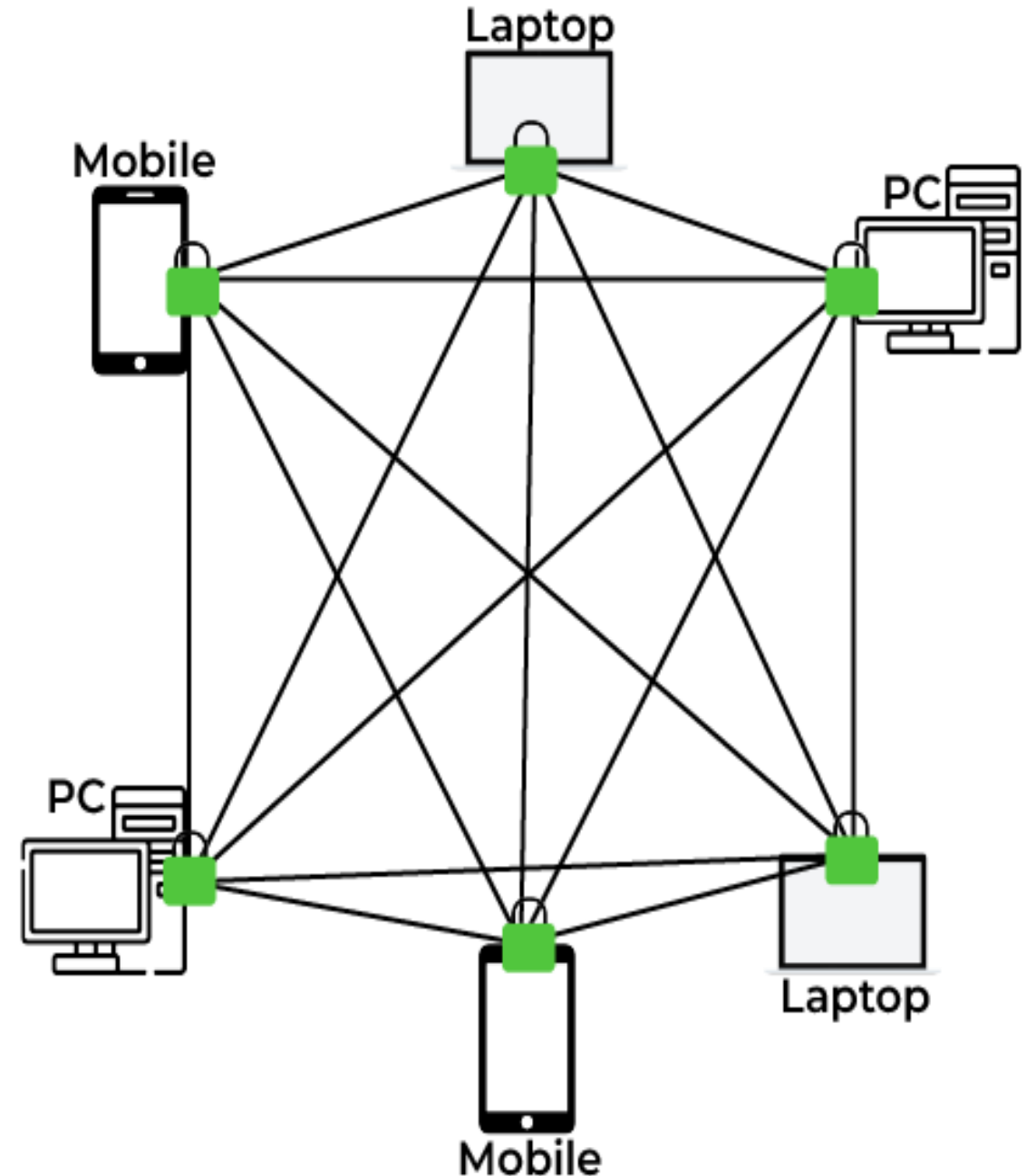
- Miners need permission to access a private blockchain. It works based on permissions and controls, which give limit participation in the network.
- Only the entities participating in a transaction will have knowledge about it and the other stakeholders not able to access it.
- By it works on the basis of permissions due to this it is also called a permission-based blockchain.
- Private blockchains are not like public blockchains it is managed by the entity that owns the network.
- A trusted person is in charge of the running of the blockchain it will control who can access the private blockchain and also controls the access rights of the private chain network.
- There may be a possibility of some restrictions while accessing the network of the private blockchain.

Advantages:

1. In a private blockchain, users join the network using the invitations and all are verified.
2. Only permitted users/ persons can join the network.
3. Private Blockchain is partially immutable.

Disadvantages:

1. A private blockchain has trust issues, due to exclusive information being difficult to access it.
2. As the number of participants increases, there is a possibility of an attack on the registered users.



3. Consortium Blockchain:

- A consortium blockchain is a concept where it is permissioned by the government and a group of organizations, not by one person like a private blockchain.
- Consortium blockchains are more decentralized than private blockchains, due to being more decentralized it increases the privacy and security of the blocks.
- Those like private blockchains connected with government organizations' blocks network.
- Consortium blockchains is lies between public and private blockchains.
- They are designed by organizations and no one person outside of the organizations can gain access.
- In Consortium blockchains all companies in between organizations collaborate equally.
- They do not give access from outside of the organizations/ consortium network.

Advantages:

1. Consortium blockchain providers will always try to give the fastest output as compared to public blockchains.
2. It is scalable.
3. A consortium blockchain is low transaction costs.

Disadvantages:

1. A consortium blockchain is unstable in relationships.
2. Consortium blockchain lacks an economic model.
3. It has flexibility issues.

CONSENSUS MECHANISM:

- The Blockchain consensus protocol consists of some specific objectives such as coming to an agreement, collaboration, cooperation, equal rights to every node, and mandatory participation of each node in the consensus process.
- Thus, a consensus algorithm aims at finding a common agreement that is a win for the entire network. Now, we will discuss various consensus algorithms and how they work.

PROOF OF WORK (PoW):

Proof of Work consensus is the mechanism of choice for the majority of cryptocurrencies currently in circulation. The algorithm is used to verify the transaction and create a new block in the blockchain.

The idea for Proof of Work(PoW) was first published in 1993 by Cynthia Dwork and Moni Naor and was later applied by Satoshi Nakamoto in the Bitcoin paper in 2008.

The term “proof of work” was first used by Markus Jakobsson and Ari Juels in a publication in 1999.

Cryptocurrencies like Litecoin. And Bitcoin are currently using PoW. Ethereum was using PoW mechanism, but now shifted to Proof of Stake (PoS).

PRINCIPLE:

A Solution that is difficult to find but is easy to verify.

Purpose of PoW:

The **purpose** of a consensus mechanism is to bring all the nodes in agreement, that is, trust one another, in an environment where the nodes don't trust each other.

- All the transactions in the new block are then validated and the new block is then added to the blockchain.
- The block will get added to the chain which has the longest block height(see [blockchain forks](#) to understand how multiple chains can exist at a point in time).
- Miners(special computers on the network) perform computation work in solving a complex mathematical problem to add the block to the network, hence named, Proof-of-Work.
- With time, the mathematical problem becomes more complex.

Features of PoW:

There are mainly two features that have contributed to the wide popularity of this consensus protocol and they are:

- It is hard to find a solution to a mathematical problem.
- It is easy to verify the correctness of that solution.

How Does PoW Work?

The PoW consensus algorithm involves verifying a transaction through the mining process. This section focuses on discussing the mining process and resource consumption during the mining process.

Mining:

The Proof of Work consensus algorithm involves solving a computationally challenging puzzle in order to create new blocks in the Bitcoin blockchain. The process is known as ‘mining’, and the nodes in the network that engages in mining are known as ‘miners’.

- The incentive for mining transactions lies in economic payoffs, where competing miners are rewarded with 6.25 bitcoins and a small transaction fee.
- This reward will get reduced by half its current value with time.

Energy and Time consumption in Mining:

The process of verifying the transactions in the block to be added, organizing these transactions in chronological order in the block, and announcing the newly mined block to the entire network does not take much energy and time.

- The energy-consuming part is solving the ‘hard mathematical problem’ to link the new block to the last block in the valid blockchain.
- When a miner finally finds the right solution, the node broadcasts it to the whole network at the same time, receiving a cryptocurrency prize (the reward) provided by the PoW protocol.

Mining reward:

- Currently, mining a block in the bitcoin network gives the winning miner 6.25 bitcoins.
- The amount of bitcoins won halves every four years. So, the next deduction in the amount of bitcoin is due at around 2024(with the current rate and growth).

- With more miners comes the inevitability of the time it takes to mine the new block getting shorter.
- This means that the new blocks are found faster. In order to consistently find 1 block every 10 minutes. (That is the amount of time that the bitcoin developers think is necessary for a steady and diminishing flow of new coins until the maximum number of 21 million is reached (expected some time with the current rate in around 2140)), the Bitcoin network regularly changes the difficulty level of mining a new block.

Proof of Stake (PoS):

- This is the most common alternative to PoW. Ethereum has shifted from PoW to PoS consensus.
- In this type of consensus algorithm, instead of investing in expensive hardware to solve a complex puzzle, validators invest in the coins of the system by locking up some of their coins as stakes.
- After that, all the validators will start validating the blocks. Validators will validate blocks by placing a bet on them if they discover a block that they think can be added to the chain.
- Based on the actual blocks added in the Blockchain, all the validators get a reward proportionate to their bets, and their stake increase accordingly.
- In the end, a validator is chosen to generate a new block based on its economic stake in the network.
- Thus, PoS encourages validators through an incentive mechanism to reach to an agreement.

- **Proof of Stake (PoS)** is a type of algorithm which aims to achieve distributed consensus in a [Blockchain](#).
- This way to achieve consensus was first suggested by Quantum Mechanic [here](#) and later Sunny King and his peer wrote a paper on it. This led to Proof-of-Stake (PoS) based Peercoin.
- A **stake** is value/money we bet on a certain outcome. The process is called staking.
- A more particular meaning of stake will be defined later on.

Why Proof-of-Stake:

- Before proof of stake, the most popular way to achieve distributed consensus was through Proof-of-Work (implemented in [Bitcoin](#)). But Proof-of-Work is quite energy(electrical energy in mining a bitcoin) intensive.

- So, a proof-of-work based consensus mechanism increases an entity's chances of mining a new block if it has more computation resources. Apart from the upper two points, there are other weaknesses of a PoW based consensus mechanism which we will discuss later on.
- In such a scenario, a Proof-of-Stake based mechanism holds merit.

What is Proof-of-Stake:

- As understandable from the name, nodes on a network stake an amount of [cryptocurrency](#) to become candidates to validate the new block and earn the fee from it.
- Then, an algorithm chooses from the pool of candidates the node which will validate the new block.
- This selection algorithm combines the quantity of stake (amount of cryptocurrency) with other factors (like coin-age based selection, randomization process) to make the selection fair to everyone on the network.

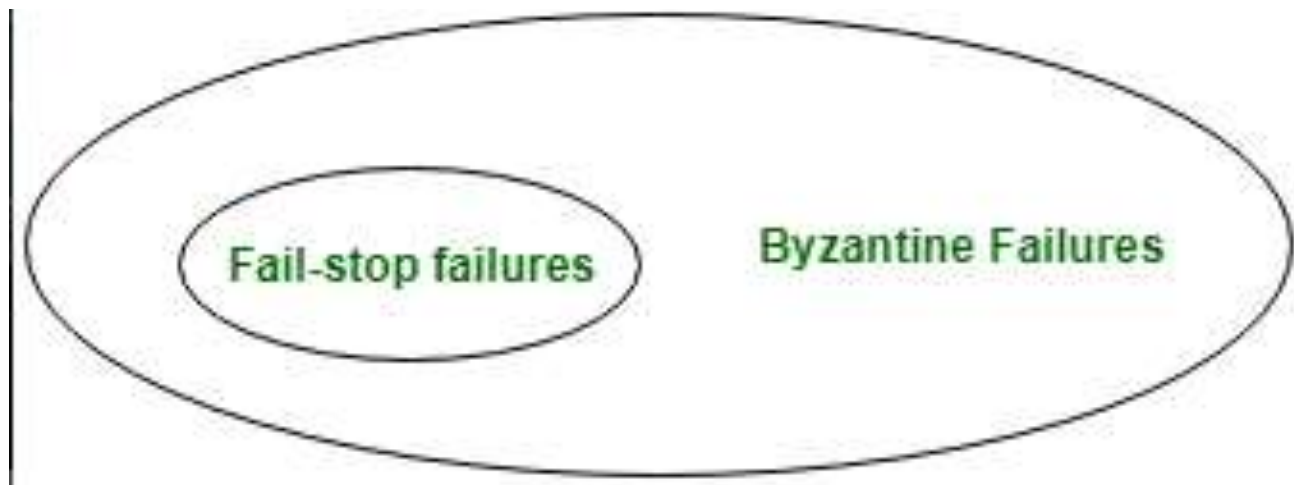
BYZANTINE FAULT TOLERANCE (BFT):

- Byzantine Fault Tolerance(BFT) is the feature of a distributed network to reach consensus(agreement on the same value) even when some of the nodes in the network fail to respond or respond with incorrect information.
- The objective of a BFT mechanism is to safeguard against the system failures by employing collective decision making(both – correct and faulty nodes) which aims to reduce to influence of the faulty nodes.
- BFT is derived from Byzantine Generals' Problem.
- Byzantine fault tolerance can be achieved if the correctly working nodes in the network reach an agreement on their values.
- There can be a default vote value given to missing messages i.e., we can assume that the message from a particular node is 'faulty' if the message is not received within a certain time limit.

Types of Byzantine Failures:

There are two categories of failures that are considered. One is fail-stop (in which the node fails and stops operating) and other is arbitrary-node failure. Some of the arbitrary node failures are given below :

- Failure to return a result
- Respond with an incorrect result
- Respond with a deliberately misleading result
- Respond with a different result to different parts of the system



PROOF OF AUTHORITY (PoA):

- In [blockchain](#) platforms, consensus mechanisms can be divided into permissionless (eg., Ethereum, [Bitcoin](#)) and permissioned (eg [Hyperledger](#), Ethereum Private).
- Unlike permissionless blockchain where anyone can become node, in permissioned blockchain all nodes are pre-selected.
- This allows to use consensus types with high scalability and bandwidth.
- One of these consensus types is **Proof-of-Authority (PoA) consensus** which provides high performance and fault tolerance. Term was proposed in 2017 by co-founder of Ethereum and Parity Technologies Gavin Wood.

Working of PoA :

- In PoA, rights to generate new blocks are awarded to nodes that have proven their authority to do so.
- These nodes are referred to as “**Validators**” and they run software allowing them to put transactions in blocks.
- Process is automated and does not require validators to be constantly monitoring their computers but does require maintaining the computer uncompromised.
- PoA is suited for both private networks and public networks, like POA Network, where trust is distributed.
- PoA consensus algorithm leverages value of identities, which means that block validators are not staking coins but their own reputation instead.
- PoA is secured by trust on the identities selected.

Advantages of PoA consensus :

- High risk tolerance as long as 51% of the nodes are not acting maliciously.
- Interval of time at which new blocks are generated is predictable. For PoW and PoS consensus, this time varies.
- High transaction rate.
- Far more sustainable than algorithms like Proof of Work which require computational power.

PROOF OF ELAPSED TIME (PoET):

Proof of Elapsed Time (PoET) is a network consensus algorithm that prevents high resource utilization and energy consumption. It implements a fair lottery system to keep the process more efficient.

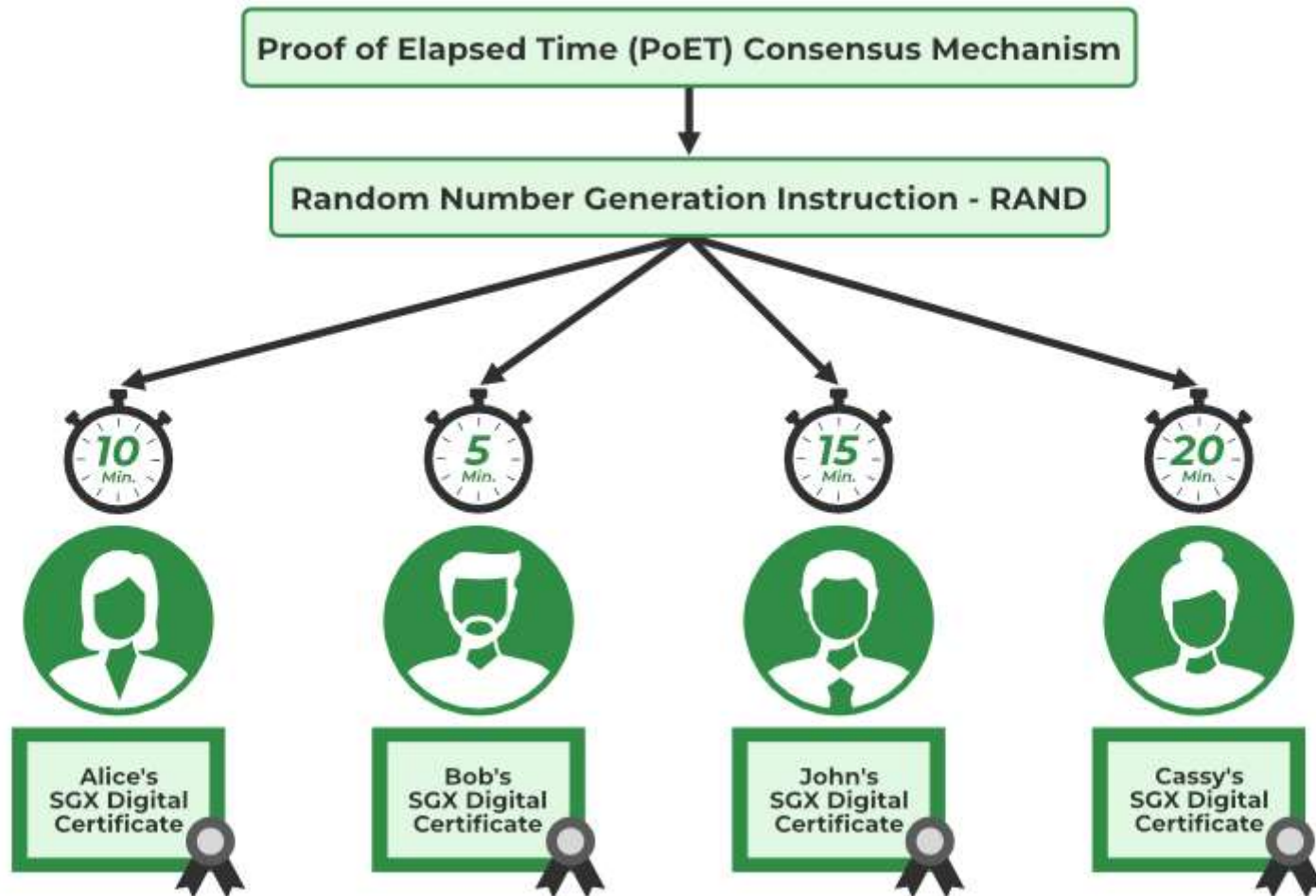
What is Proof of Elapsed Time(PoET)?

- Blockchain technology is a distributed ledger used to record any transactions that happen over the network and since Bitcoin's inception and the introduction of the **Proof of Work (PoW)** consensus model as the mechanism for a blockchain network's legitimacy.
- After numerous other consensus mechanisms have been experimented with and introduced, the Proof of Elapsed Time (PoET) concept is one such mechanism which was invented in early 2016 by Intel Corporation and is one of the fairest blockchain consensus algorithms that enables permissioned blockchain networks to determine who creates the next block.

- Intel in collaboration with other organizations such as the Linux Project and IBM sponsored the **Hyperledger Sawtooth** project, this open-source project uses the PoET consensus mechanism.
- Hyperledger Sawtooth is a distributed ledger that is both scalable and adaptable for many uses at the enterprise like supply chain and logistics.
- This mechanism is based on Byzantine Fault Tolerance and aims to reduce the energy consumption associated with proof of work's mining process.
- This algorithm assigns an amount of time to each node in the network randomly.
- The node must sleep or do another task for that random wait time.
- The node that gets the shortest waiting time wakes up and adds its block to the network.
- The newly updated information floods among the network participants.

3 Factors Need to Be in Favor for PoEt to Work

1. Ensure that the node gets the random waiting time instead.
2. Check if the nodes are not choosing the shortest wait time on purpose.
3. Verify if a node has completed the given waiting time or not.



The PoET consensus mechanism is divided into 2 phases:

1. Selection Process: This process involves the following activities:

- Every node in the network will share its certificate by Intel Software Guard Extension (SGX) which ensures the validity of the node to generate a new block in the network. After this, the node is eligible to get a timer object.
- The numbers are assigned to each node as a timer object by Intel's RAND i.e. random number generator. RAND generates difficult-to-detect random numbers.
- The time object given to each participating node activates.

2. Generation Process: This process involves the following activities:

- After the timer object expires and the node wakes up then the node is eligible to forge a new block to the network.
- The active node generates the hash of its block of transactions and submits it for acceptance.
- The update gets flooded to the network.

BLOCKCHAIN TECHNOLOGY

UNIT 3

REFERENCE BOOK:

Bellaj Badr ,Richard Horrocks,Xun (Brian) Wu, "BlockChain By Example:A developer's guide to creating decentralized applications using Bitcoin,Ethereum and Hyperledger" Packet Publishing Limited,2018

Blockchain-Based Futures System

In this unit,

- we will learn how to harness a powerful language, such as Java, to build an Ethereum desktop or mobile application
- calling third-party APIs in your smart contracts or libraries.

we will break into finance and build a financial project that manages a futures smart contract using Java (SE). Throughout this interesting use case, we will go over the following points:

- **Building futures solidity contract**
- **Introducing oracles**
- **Introducing the web3j API**

Building a futures smart contract will help you to understand the disruptive force of the blockchain and how smart contracts can alter the financial industry (for example, futures trading market) by automating payments and facilitating business processes.

Project Presentation

FUTURE CONTRACTS:

Futures contracts are legal agreements to buy or sell a given quantity of commodity or asset (barrel of oil, gold, and silver) at a predetermined price at a specified time in the future, hence the name futures.

EXAMPLE:

Let's suppose that an airline company wants to lock in the fuel price to manage their exposure to risk and to avoid any unpleasant surprises. Instead of buying the oil and storing it until they need it, they buy a futures contract for oil with a specific date of delivery and price per gallon. For example, an airline company can agree on a futures contract to buy 1,000,000 gallons of fuel, taking delivery 60 days in the future, at a price of \$3 per gallon.

- the airline company is considered as **hedger**
- while the seller might be considered as a **speculator**.
- As the futures contracts may become more valuable if the price goes up, the owner of that contract could sell that contract for more to someone else.

- i. That said, we will build a smart contract that implements the logic described here and
- ii. defines the futures contract's details (asset, quantity, unitary price, and delivery date) along with holding the funds.
- iii. We will also build a Java application enabling a third party (exchange market authority) to define a futures contract between the other two parties.

The smart contract will also enable the speculator to sell the futures contract to another speculator and calculate its investment offset by comparing the market price to the futures contract price.

Futures smart contract

To write the smart contract code

- First, create a directory called FuturesWeb3j
- initiate an empty Truffle project using truffle init.
- Afterward, in the contracts/ folder, create a Futures.sol file with the following code:

Example Coding For Future Smart Contract:

the starting point is to define a constructor that will determine the contract's details, namely: asset ID, agreed quantity to buy, unit price, buyer's address, seller's address, and delivery date.

We'll also define a few functions:

- `deposit()`: The hedger (airlines) will call this function to deposit the agreed funds.
- `sellcontract()`: The speculator can sell the contract to another investor.
- `getpaid()`: The contract will release the funds to the oil vendor at the delivery date.
- `onlyhedger()`: This is a modifier that restricts actions to the initial buyer (the hedger).

```
contract CFutures {
    address user;
    address hedger;
    address speculator;
    struct asset{
        uint id;
        uint Quantity;
        uint price;
        address seller;
        address buyer;
        uint date;
    }
    asset TradedAsset;

    event DepositEv(uint amount, address sender);
```

```
function CFutures(  
    uint assetID,  
    uint Quantity,  
    uint price,  
    address buyer,  
    address seller,  
    uint date) public {  
    TradedAsset.id = assetID;  
    TradedAsset.Quantity = Quantity;  
    TradedAsset.price = price;  
    TradedAsset.seller = seller;  
    speculator = seller;  
    hedger = buyer;  
    TradedAsset.buyer = buyer;  
    TradedAsset.date = date;  
}
```

```
function deposit() public payable returns(bool) {
    require(msg.value == TradedAsset.Quantity * TradedAsset.price);
    DepositEv(msg.value,msg.sender);
    return true;
}
```

```
modifier onlyhedger() {
    require(msg.sender == hedger);
    _;
}
```

```
function sellcontract(address newhedger) public onlyhedger returns(bool){
    hedger = newhedger;
    return true;
}
```

```
function getpaid() public returns(bool){
    require(now >= TradedAsset.date && address(this).balance > 0);
    speculator.transfer(address(this).balance);
    return true;
}
}
```

Blockchain oracles

once the contract expires, we want the contract to calculate the offset benefit (savings) or loss made by the futures deal – in other words, how much the hedger has economized or lost.

In this case, at settlement time, the smart contract needs a solid and reliable source of information to determine the underlying commodity (oil) price to calculate the offset.

The smart contract is deterministic code executed in an isolated VM without direct connection to an outside source of information!

However, there is a workaround for that situation: **oracles**

Oracles are trusted agents or data suppliers that read and send requested information to the smart contract from reliable external data sources. We have a few providers of this type of service in the blockchain space, such as Chainlink, Link, Blocksense, and Oraclize.

we will use Oraclize (<http://www.oraclize.it/>) as it is an industry leader in oracle servicing.

Let's see how we can connect our contract to an oracle contract. In our previous code, we have to bring in the following changes:

```
pragma solidity ^0.4.24;
```

```
import "./oraclizeAPI_0.5.sol";
```

```
contract CFutures is usingOraclize {
    uint public fuelPriceUSD;

    event NewOraclizeQuery(string description);
    event NewDieselPrice(string price);

    function CFutures(
        uint assetID,
        uint Quantity,
        uint price,
        address buyer,
        address seller,
        uint date) public {
... // keep it as defined previously
    }
```

```
function __callback(bytes32 myid, string result) public {  
    require(msg.sender == oraclize_cbAddress());  
    NewDieselPrice(result);  
    fuelPriceUSD = parseInt(result, 2);  
}
```

```
function updateprice() public payable {  
    NewOraclizeQuery("Oraclize query was sent, standing by for the answer..");  
    oraclize_query("URL","xml(https://www.fueleconomy.gov/ws/rest/fuelprices).fuelPrices.diesel");  
}  
}
```

In result we end up with a simple contract using the Oraclize service to request Fuelprice

- we start by importing the Oraclize API: `import "./oraclizeAPI_0.5.sol";`.
- Then, we inherit the `usingOraclize` contract to access the library methods.
- In the Oracle code part we defined, along with `fuelPriceUSD`, two new functions: `updateprice()` and `__callback`

`updateprice()`

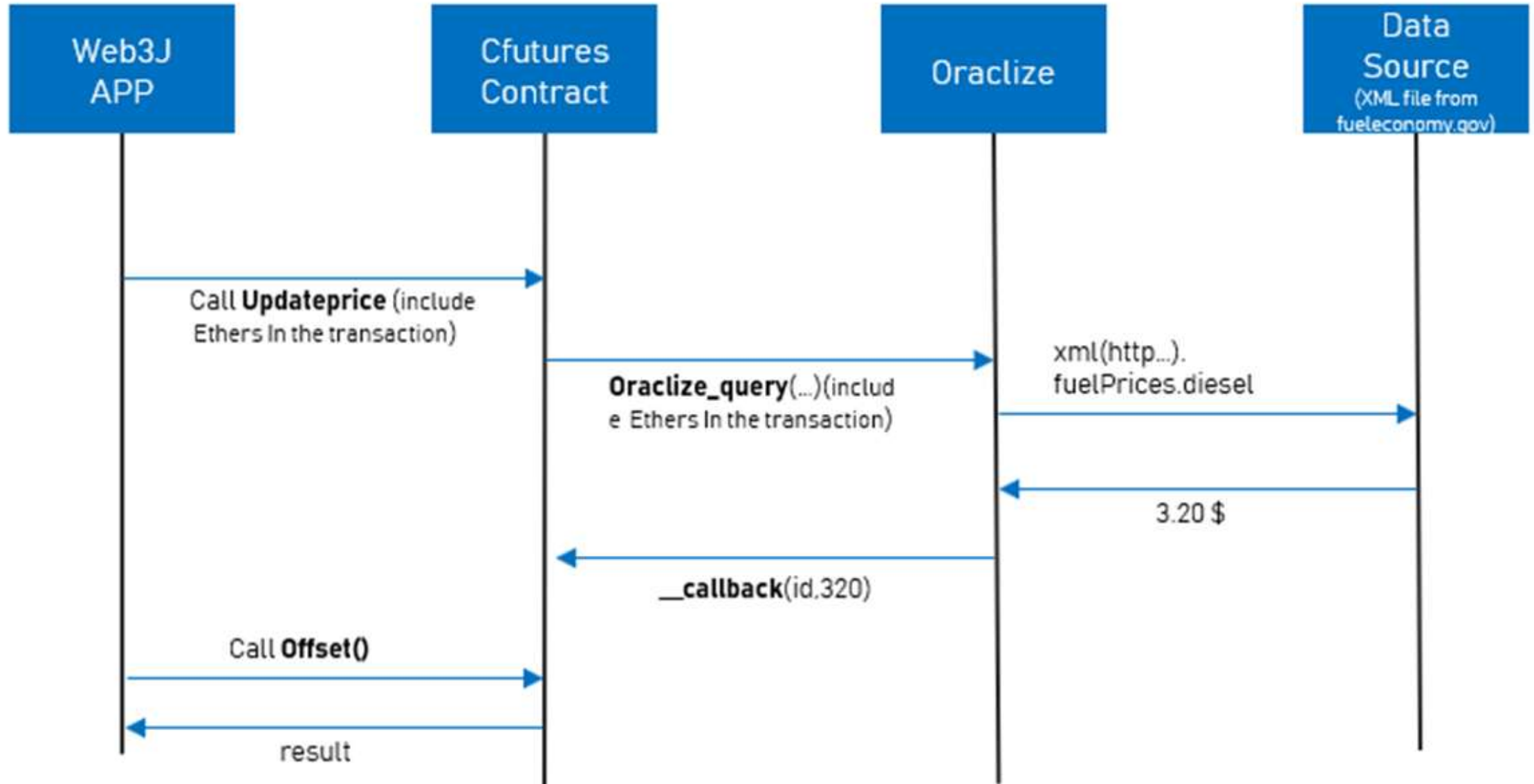
`oraclize_query` to send a query to the Oraclize's smart contract with two arguments: the type of request (URL, WolframAlpha, IPFS multihash) and the requested information (a specific property in the XML response). The `updateprice` function is declared as payable because the Oraclize service isn't free, and this function should accept ether to be able to pay for the Oraclize service.

However, the first query made by a contract is free, so we include a call for the `updateprice()` function without sending ether in our constructor.

`__callback()`

`__callback` is defined by the API and, as its name indicates, it's a callback function executed once Oracle sends back the response. Its first parameter is the id of the request, while the second parameter is the result of your request. In this example, we use the `parseInt` function, which is defined by the Oraclize API file, to set the number precision. In the preceding example, \$3.20 will be returned as 320 cents

The following diagram represents the interaction between the initial contract and the data source using Oraclize:



Web3j

Web3j is a highly modular, reactive, type safe java and Android library for working with Smart Contracts and integrating with clients(nodes) on Ethereum network

It represents a lightweight Java and Android API for integration with Ethereum clients. It enables you to build a decentralized Java application easily based on Ethereum.

You should also have the following elements installed to build the application:

- Java 8 JDK
- Eclipse and Maven – for coding and development

Setting up the Web3J Maven project

you have to create a new Maven project, as follows:

In the Eclipse IDE,

- navigate to File | New | Other to bring up the project-creation wizard.
- Scroll to the Maven folder, open it, and choose Maven Project. Then choose Next.

- Keep passing the dialogue forms by pressing next until you get the following form

New Maven Project

New Maven project

Specify Archetype parameters

Group Id:

Artifact Id:

Version:

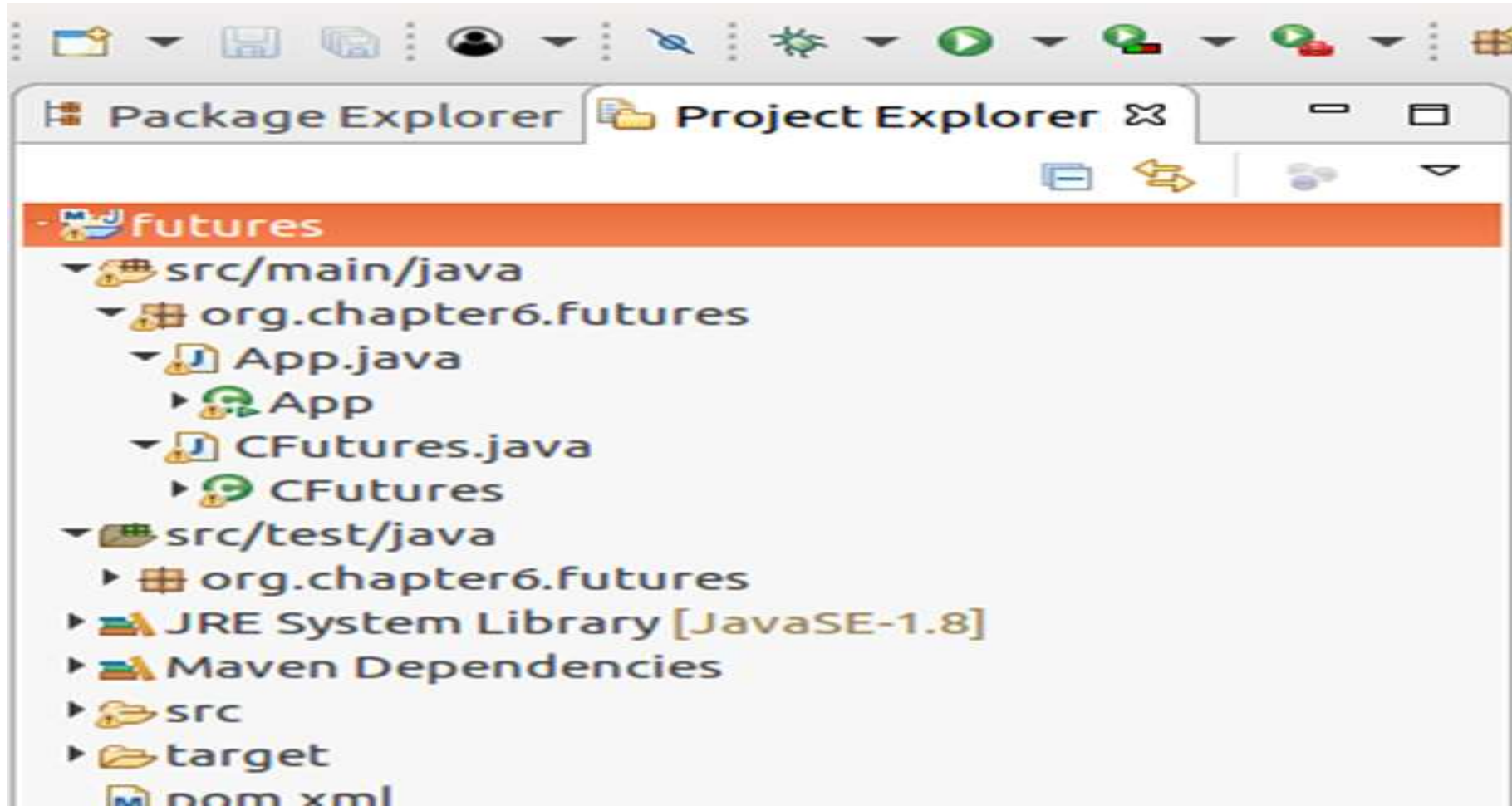
Package:

Properties available from archetype:

Name	Value
------	-------

▸ **Advanced**

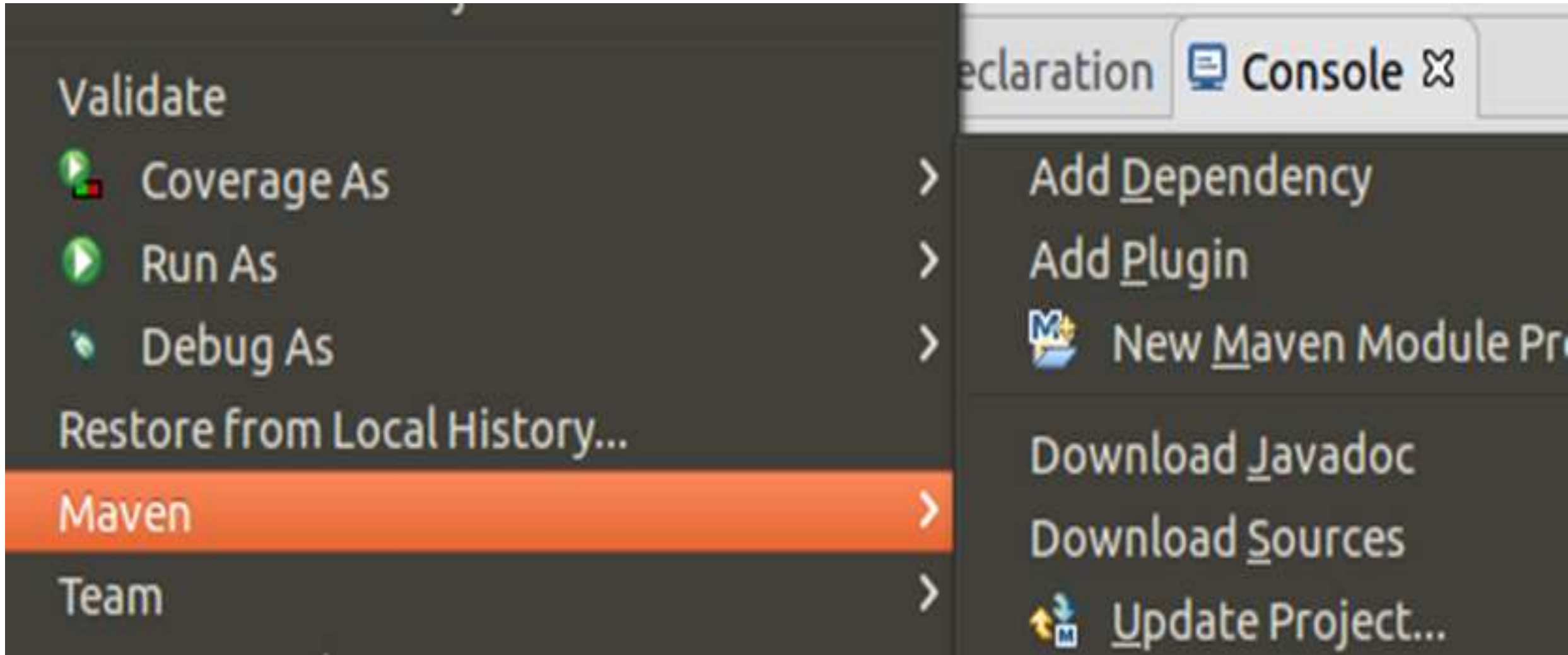
Here, you will need to enter information regarding the Maven project you are creating. Choose the names you want and press Finish. The following is what your sample project structure will look like at the end of the Maven project-creation process:



Then, in the Maven POM file (pom.xml), we'll need to add the following lines in the corresponding POM tags:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.7.0</version>
    <configuration>
      <source>1.8</source>
    <target>1.8</target>
    </configuration>
  </plugin>
</plugins>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.21</version>
</dependency>
```

After editing the Maven POM file, we have to update the project to download the necessary dependencies. Right-click on Project, click on Maven, then choose Update Project...:



Installing web3j

Web3j provides a command-line tool distributed as a portable .jar. In a few steps, you can start using the web3j command line, so no installation is required.

In your home folder.

<https://github.com/web3j/web3j/releases/download/v3.4.0/web3j-3.4.0.tar>.

```
Unzip the downloaded TAR file:  
tar xvf web3j-3.4.0
```

Then add the resultant folder to the system's path:

```
export PATH=$PATH:~/web3j-3.4.0/bi
```

Now, you can create Ethereum accounts or generate wrapper classes for your smart contracts.

Wallet creation

The web3j command-line tools allow us, among other features, to create wallets or send transactions.

To create a new Ethereum account, run `web3j wallet create`. You'll be asked to provide a password and then a location where you want to back up your account (wallet JSON file). At the end, you'll get a JSON file with a long name that carries lots of information, including the account address and private key

```
user@ByExample-node:~/eclipse-workspace/futures/src$ web3j wallet create
```



```
Please enter a wallet file password:
```

```
Please re-enter the password:
```

```
Please enter a destination directory location [/home/user/.ethereum/testnet/keystore]: .
```

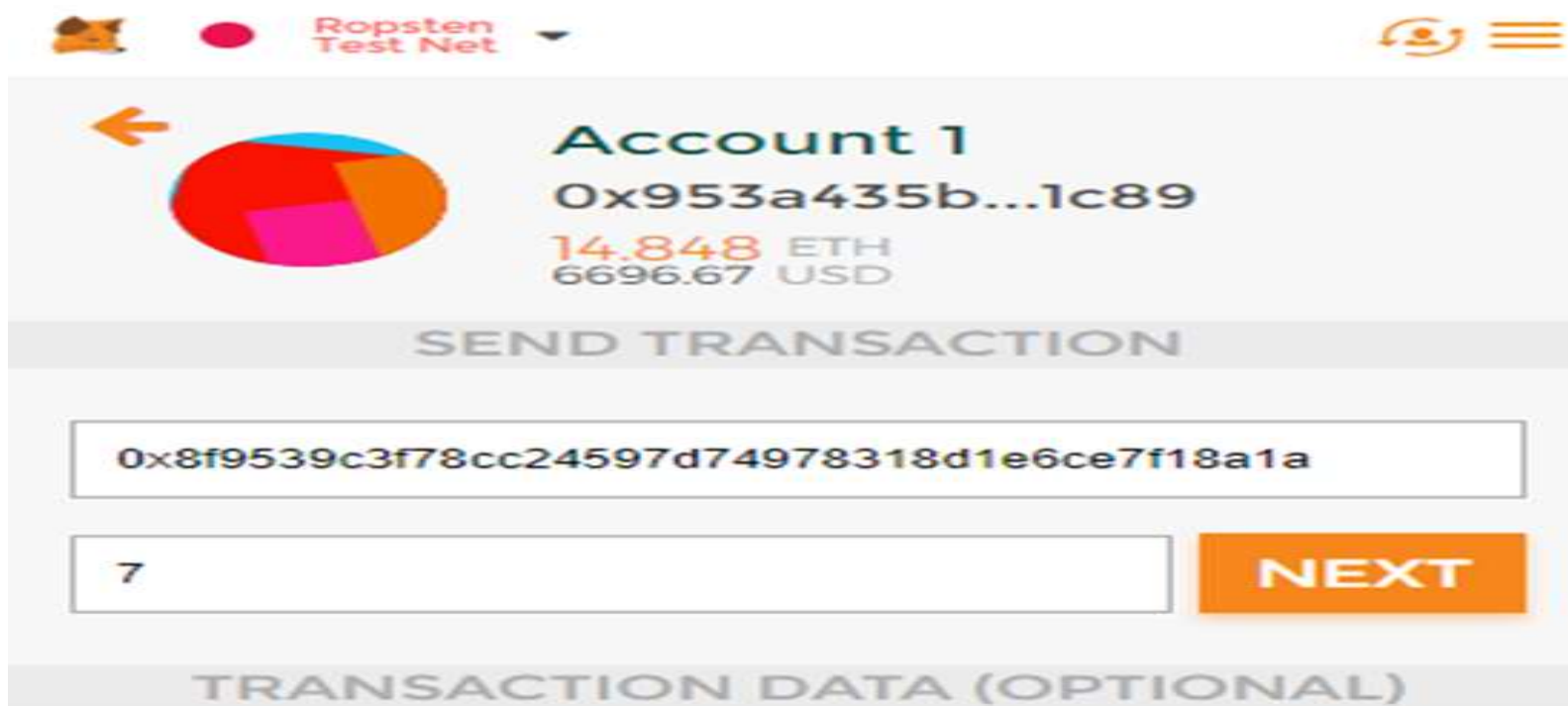
```
Wallet file UTC--2018-07-15T16-44-45.169000000Z--8f9539c3f78cc24597d74978318d1e6ce7f18a1a.json successfully created in:
```

```
user@ByExample-node:~/eclipse-workspace/futures/src$ ls
```

```
main test UTC--2018-07-15T16-44-45.169000000Z--8f9539c3f78cc24597d74978318d1e6ce7f18a1a.json
```

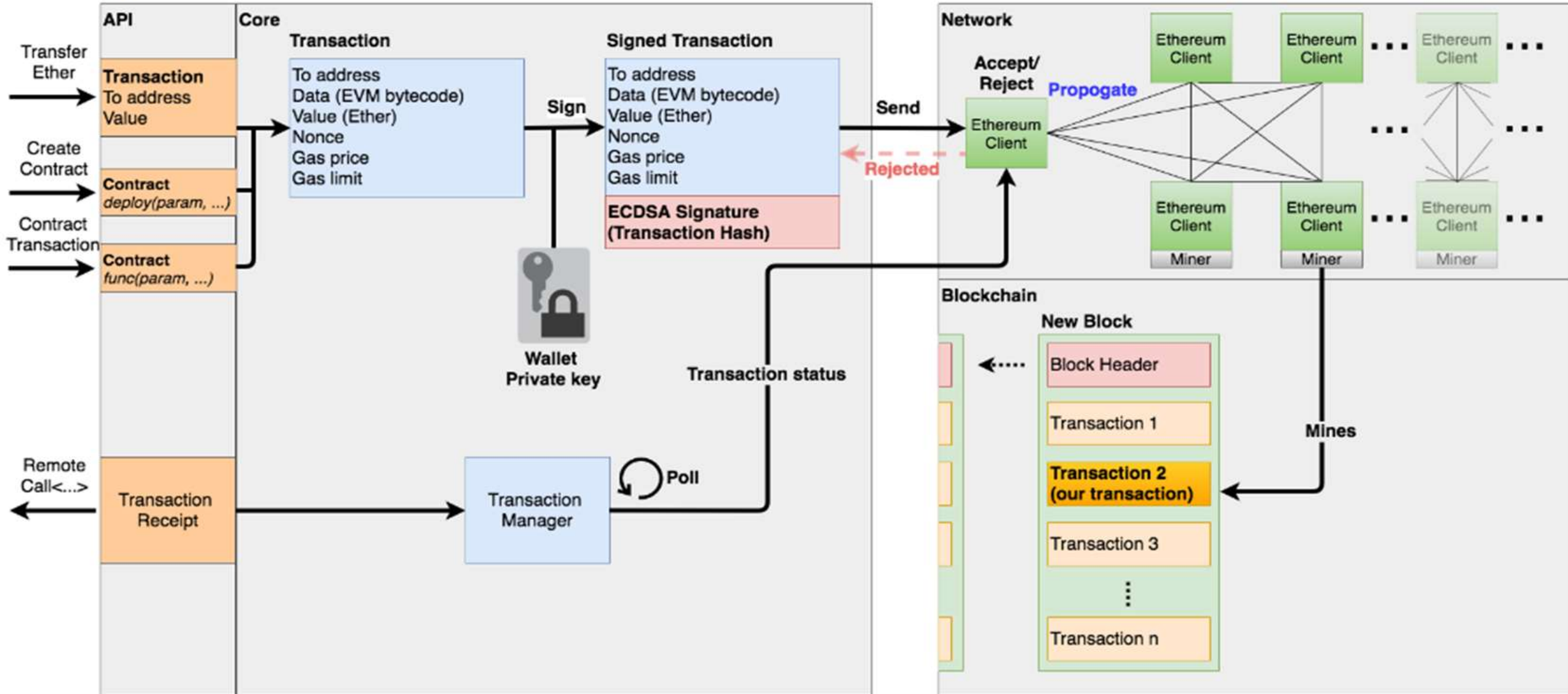
Alternatively, instead of using web3j, you can export one of your Testnet accounts created in MetaMask or Geth.

In this project, we will be running our contract in the Testnet, therefore we need to supply test ethers to our freshly created account. To do that, we can use MetaMask to get free ethers and send them to our new account's address. Be careful, you should add 0x and wait until the transaction is confirmed, as shown here



Java client

- Building a simple Java application using web3j.
- Web3j is the Java port of Ethereum's web3 API, which manages communication (JSON-RPC) between Ethereum clients and the network.
- For each smart contract, web3j generates a wrapper class, simplifying the access and smart contract interaction with Ethereum.
- Thus, all complexities will be hidden and you will not be forced to initiate RPC calls on your own. The following diagram (from the official documentation) depicts the interaction flow between web3j and Ethereum:



The wrapper generator

- Before using web3j, we need to build the Java wrapper class for our smart contract.
- The web3j command-line tool supports the autogeneration of Java smart contract function wrappers directly from Truffle's contract schema, as follows:

```
web3j truffle generate ~/FuturesWeb3j/build/contracts/CFutures.json -o ~/eclipse-workspace/futures/src/main/java/ -p  
org.chapter6.futures
```

- The first argument is the build file compiled by Truffle for the CFutures contract,
- The second is the path of your Java project destination where the class wrapper will be generated,
- The -p parameter should be the Java packaging structure for the generated class

This will result in creating a CFutures.java file with a new class with the same name as the smart contract:

```
public class CFutures extends Contract {  
    private static final String BINARY = "0x60606040523415600e576...";  
}
```

Initializing web3j

Now App.java file created in the Maven project.

The first step is to initialize a web3j object. We do that by passing in a provider (for example, the endpoint of a third-party or local Ethereum node) to the web3j build function:

```
Web3j web3j = Web3j.build(new HttpService("https://ropsten.infura.io"));
```

In this example, we will use the Infura gateway (<https://infura.io/>) to access the Testnet network.

In Ethereum, we always need an entry point, which is an RPC/HTTP provider (node) that carries the orders or actions dispatched by the user to the network. Normally, the end user should have a local RPC client connected to the network. However, we directly use an online provider (used also by MetaMask) to avoid needless instructions and to keep communication with the oracle straightforward.

Setting up Ethereum accounts

To interact with the blockchain network, we need to build transactions, thus we need to have a wallet. We can use an existing one, as follows:

```
Credentials credentials = WalletUtils.loadCredentials("PASSWORD", "/path/to/walletfile.json");
```

we can create a new wallet:

```
WalletUtils.generateNewWalletFile("PASSWORD", new File("/path/to/destination"), true);
```

Once loaded or created, you can check the current balance of your wallet using the following:

```
EthGetBalance AccountBalance = web3j.ethGetBalance(credentials.getAddress(),  
    DefaultBlockParameterName.LATEST).sendAsync().get();  
log.info("The accounts balance " + AccountBalance);
```

Deploying the contract

As the wallet credentials are defined and the web3 is initialized, we can construct and deploy our CFutures contract using the deploy method:

```
CFutures CFuture_ = CFutures.deploy(web3j, credentials, CUSTOM_GAS_PRICE, GAS_LIMIT, assetID,  
Quantity, price, buyer, seller, date).send();
```

- The first four arguments are needed to construct the deployment transaction, while the rest are the arguments needed by the contract's constructor.
- Among these four parameters, we need to define the gas limit and the gas price. In this example, I have defined them as global variables, as follows:

```
private static final BigInteger GAS_LIMIT = BigInteger.valueOf(4_700_000);  
private static final BigInteger CUSTOM_GAS_PRICE = Convert.toWei("140",  
Convert.Unit.GWEI).toBigInteger();
```

- Here, we define a gas limit of 47,00,000 units of gas and a gas price of 140 GWei. Web3j uses a default price of 22 GWei
- To choose the right value, you can check the current gas price for Ropsten at <https://ropsten.etherscan.io/chart/gasprice>.
- Once the `deploy()` function is executed, it will result in creating a new instance of the CFutures smart contract on the Ethereum blockchain using the supplied credentials and constructor parameter values.
- If, for any reason, the deployment transaction isn't validated within 600 seconds, an exception will be raised.
- At any time, you can visit the Ropsten explorer to examine whether the contract has been deployed using the link printed by the log here:

```
String CFuturesAddress=CFuture_.getContractAddress();
```

```
log.info("View contract at https://ropsten.etherscan.io/address/" + CFuturesAddress);
```

Interacting with smart contracts

If you wish to construct an instance of a smart contract wrapper with an existing smart contract, simply pass in its address to the predefined load function:

```
CFutures CFutureInstance = CFutures.load("contract_address", web3j, credentials,  
                                         CUSTOM_GAS_PRICE, GAS_LIMIT);
```

Once you instantiate a new CFuture object (using deploy or load), Eclipse will help you with the autocompletion feature to choose the functions to call or to access contract proprieties.

Calling a contract function

In our example, once the contract is loaded for the first time, we need to deposit the amount both parties agreed on. Hence, we need to execute the deposit function along with sending the contract value in ethers.

In web3j, all methods are defined identically to their equivalent Solidity methods and take the same parameters. To transact with a Cfutures function, we invoke the equivalent object's method using dot notation, as follows:

```
BigInteger Quantity = BigInteger.valueOf(120000);
```

```
BigInteger Price = BigInteger.valueOf(300);
```

```
TransactionReceipt DepositReceipt = CFutureInstance.deposit(Price.multiply(Quantity)).send();
```

the argument passed to the deposit function is a BigInteger (price * quantity), not an integer or int, and so all variables should be declared as BigInteger in web3j.

The transaction receipt is useful for two reasons:

- It provides details of the mined block that the transaction resides in
- Solidity events that are called will be logged as part of the transaction, which can then be extracted.

For example, we can get from the transaction receipt the transaction hash and examine the corresponding transaction on Etherscan as shown in the picture below:

[This is a Ropsten Testnet Transaction Only]


TxHash: 0x4ef437fce1a877c3b155a03b0ba11309695ac4d84d351f6f3aefee1cf

TxReceipt Status: Success

Block Height: 3648112 (1 block confirmation)

TimeStamp: 25 secs ago (Jul-16-2018 09:56:11 AM +UTC)

From: 0x8f9539c3f78cc24597d74978318d1e6ce7f18a1a

To: Contract 0x7cd884e217c638b04990af8adb5efec6e4aac015 

Value: 360 wei (\$0.00)

Gas Limit: 4700000

Gas Used By Txn: 21998

Gas Price: 0.00000014 Ether (140 Gwei)


Actual Tx Cost/Fee: 0.00307972 Ether (\$0.000000)

Nonce & {Position}: 30 | {0}

Input Data:

```
Function: deposit() ***
```

```
MethodID: 0xd0e30db0
```

View Input As 

Calling view methods

For view methods (which only read a value in a smart contract), they are invoked using an ordinary method call, except they will not return a TransactionReceipt as no transaction will be initiated (no cost associated).

In our smart contract, we have a view method, `offset()`, that can be invoked as follows:

```
BigInteger Offset=CFutureInstance.offset().send();
```

```
log.info("your offset is "+Offset.intValue());
```

In the same manner, it is possible to query the public states of a smart contract, for instance to read the last `fuelPriceUSD` state's value:

```
BigInteger LastfuelPrice = CFutureInstance.fuelPriceUSD().send();
```

```
Integer fuelPriceUSD = LastfuelPrice.intValue();
```

```
log.info("Last fuel Price Fuel price According to the Oracle is: " + fuelPriceUSD);
```


Web3j events

In the smart contract, we have defined a few events. Each one will be represented in the smart contract wrapper with a method named identically, which takes the transaction receipt and returns a decoded result (event parameters) in an instance of the EventValues object. Given that, here's how to get the event data and display its content in our console:

```
for (CFutures.DepositEvEventResponse event : CFutureInstance.getDepositEvEvents(DepositReceipt)) {  
    log.info("Depoist event detected:" + event.amount+"wei has been deposited");  
    log.info("The funds has been sent by: " + event.sender);  
}
```

Enhancement

The last thing to consider is adding a textual or a graphical menu to this application to help the user make their choices.

```
input = new Scanner(System.in);
CFutures CFutureInstance = null;
System.out.println("Please make your choice:");
System.out.println("1 – Deploy New Cfutures contract.");
System.out.println("2 – Load contract");
System.out.println("3 – Make deposit");
System.out.println("4 – current Fuel price");
System.out.println("5 – Update Fuel price");
System.out.println("6 – Investment offset");
System.out.println("0 - exit");
int selection;
choice: while(input.hasNextInt()) {
    selection = input.nextInt();
    switch (selection) {
        case 0:
            break choice;
        ...
    }
}
```

App (1) [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Jul 19, 2018, 1:17:20 PM)

[main] INFO org.chapter6.futures.App - 0x8f9539c3f78cc24597d74978318d1e6ce7f18a1a Credentials loaded

[main] INFO org.chapter6.futures.App - The accounts balance 3261863859999994600 wei

Please make your choice:

1 - Deploy New Cfutures contract.

2 - Load contract

3 - Make deposit

4 - current Fuel price

5 - Update Fuel price

6 - Investment offset

0 - exit

2

Please provide the contract's address to load:

0x4d8e3580f4929c2b4aaf8aaabde0c4e955802a01

[main] INFO org.chapter6.futures.App - Contract loaded: 0x4d8e3580f4929c2b4aaf8aaabde0c4e955802a01

4

[main] INFO org.chapter6.futures.App - Last fuel Price Fuel price According to the Oracle is: 324

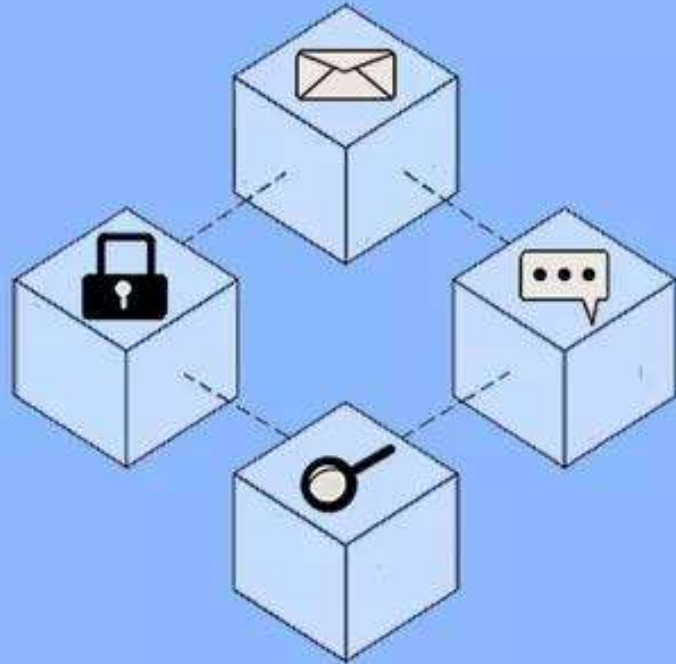
5

6

[main] INFO org.chapter6.futures.App - your offset is -2880000

What Is a Blockchain?

- A blockchain is a distributed database or ledger shared among a computer network's nodes. They are best known for their crucial role in cryptocurrency systems for maintaining a secure and decentralized record of transactions, but they are not limited to cryptocurrency uses. Blockchains can be used to make data in any industry immutable—the term used to describe the inability to be altered.



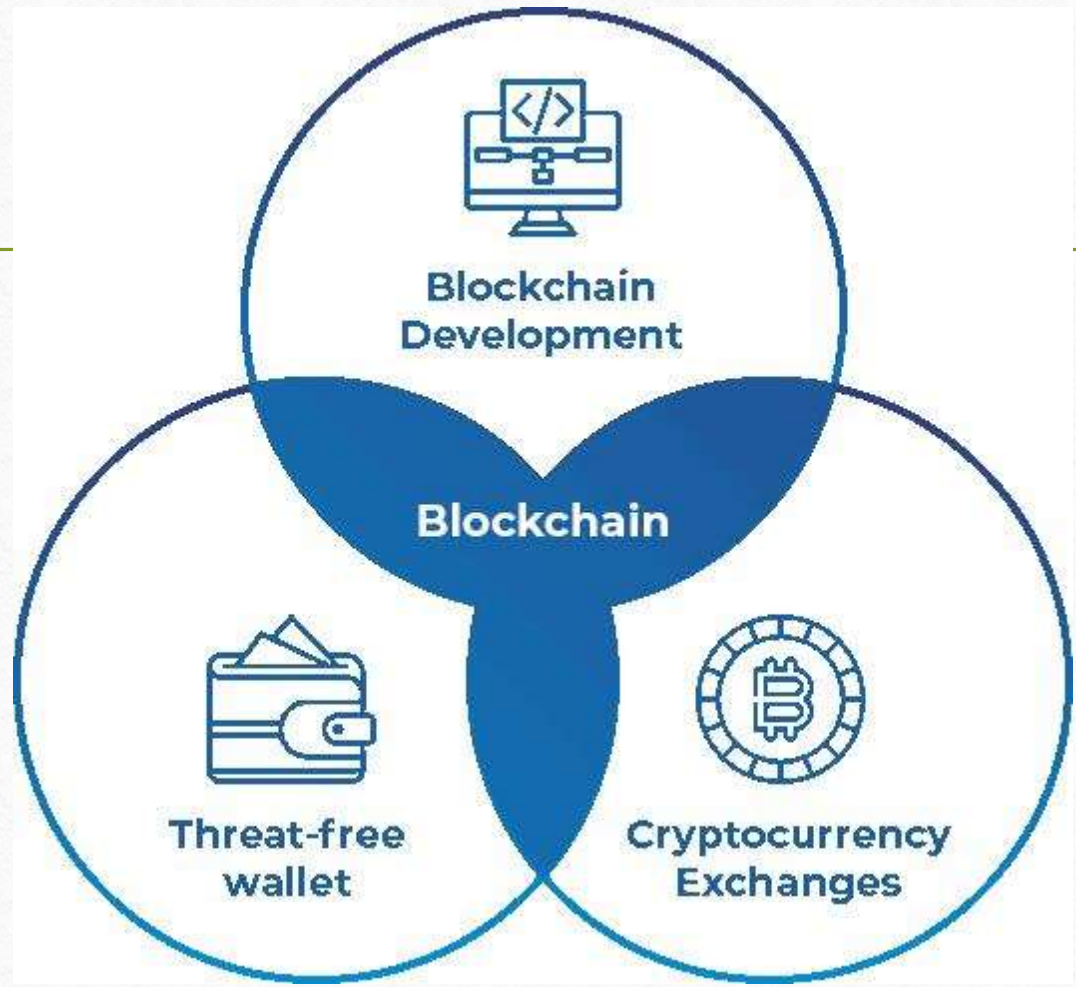
Blockchain

['bläk-,chān]

A digital database or ledger that is distributed among the nodes of a peer-to-peer network.

Blockchains in business and creating ICO

- Blockchain technology has gained significant attention in the business world due to its potential to revolutionize various industries.
- It offers benefits like increased transparency, enhanced security, and improved efficiency.
- Businesses are exploring blockchain for supply chain management, data sharing, identity verification, and more.



-
- Regarding Initial Coin Offerings (ICOs), they were popular fundraising methods in the past, where companies issued new cryptocurrencies or tokens to raise funds for their projects.
 - However, the ICO landscape has evolved, and regulations have become stricter in many countries due to the potential for fraud and scams.
 - As of my last update in September 2021, ICOs were declining in popularity, and other fundraising methods like Security Token Offerings (STOs) and Initial Exchange Offerings (IEOs) were gaining traction.

BLOCKCHAINS IN BUSINESS

The Ethereum main network, and its corresponding test networks, are all open for public use. All transactions made on the networks are 100% transparent, and anybody with access to the network—which is potentially everybody—can see and access all data.

- The users of these networks don't necessarily know each other: they are mutually distrustful parties who must assume that the other participants in the network are dishonest. For this reason, public blockchain networks require a way for such parties to reach an agreement without needing to trust each other, and this is where decentralized consensus algorithms, such as proof of work (PoW), come into the picture
- There are cases, however, where transacting parties in a network are more trusting of each other, and where the use of a blockchain isn't solely centered on allowing mutually distrustful parties to interact
- In this chapter, we will explore the main types of blockchain networks available – public and private—before looking in more detail at how blockchains can be used in business settings, and what options are currently available for keeping data private. We will then implement a very basic private network that makes use of certain privacy features.



Public versus private and permissioned versus permissionless blockchains

- The Ethereum main network is public, meaning anyone is free to join and utilize the network. There are no permissions involved: not only can users send and receive transactions, they can also take part in a consensus, as long as they have the appropriate hardware to mine blocks. All parties in the network are mutually distrustful, but are incentivized to remain honest by the mechanisms involved in the PoW consensus. This is an example of a public, permissionless network. These networks offer high resistance to censorship and good data persistence, but are less performant due to the decentralized nature of consensus.

-
- The idea of permission, when applied to blockchains, could take one of several forms: it could be explicit, as in the case of an access control list, or implicit, as in a requirement placed on users to enable them to join a network. An example of a public, permissioned blockchain, could take the form of a public proof of stake network, in which permission to participate in the network as a validator is granted in exchange for a deposit or stake.

-
- Of more concern for this chapter is the idea of networks run by companies, either for their own internal use, or as a shared resource used by a group of companies wanting to transact. Such networks are private: they are not open to the general public, but only to those companies authorized to join. These are sometimes also referred to as consortium blockchains, and as such, networks often also impose different types of permissions for different participants. For example, different members of the network may have read, write, or access permissions, while another subset of members may take on the job of validation.

-
- Such private networks can take advantage of permissioning to be more performant than their public and permissionless counterparts. If a set of nodes can be trusted as validators, then consensus can be reached more quickly. There is, however, a tradeoff: a set of permissioned validators must be trusted to correctly and honestly validate blocks. In the world of business, where time is money, such a tradeoff can often be worth making.

Privacy and anonymity in Ethereum

- We have so far discussed the differences between public and private networks, where it is the network itself, and access to it, which is either public or private. We now turn our attention to the data—both transaction and contract data—inside a given network.
- The Ethereum main network can be joined by anyone. Furthermore, all transaction and smart contract data is public, meaning all transactions between a to and from address can be seen by everybody using the network. There is no way to hide these transactions, or the addresses transacting, and as such, there's no way for a user on Ethereum to be truly anonymous. If a way were found to link an address with a real-world identity—either at the present time, or at a point in the future—then the identity of the transacting party would be known. This might seem obvious: on public networks, all data is public. What is less obvious is that even in a private Ethereum network, data within the network is visible to all participating nodes.

-
- Other cryptocurrencies, such as Zcash, Monero, and Dash, provide differing degrees of anonymity, and we will briefly discuss two of the techniques for doing so in this chapter, and look at how these could be applied to Ethereum.
 - Monero uses a type of digital signature called a ring signature, which helps anonymize the transacting addresses as well as the amount being sent. It's possible to use a similar technique in Ethereum by using mixing services based on ring signatures, but these generally aren't accepted as being robust and scalable methods that could be used in an enterprise.

-
- Zcash uses a different technique, leveraging Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) proofs. In this scheme, a user is able to prove possession of some data without needing to reveal what the data is, and without needing to interact with the user verifying the data. Some work has been done to incorporate zk-SNARKs with Ethereum, though at the time of writing this work is still experimental.

-
- So, there is currently no reliable method to make addresses and transactions anonymous in a guaranteed way – what about the contents of those transactions?
 - The contents of a transaction, as well as the code and data associated with a smart contract, are publicly viewable and cannot be obscured. Though a smart contract's code is compiled to bytecode, it should not be assumed that an adversary wouldn't be able to decompile and read the code. As such, sensitive information should neither be hardcoded into a contract nor sent to it as part of a transaction.

-
- - What can be done, however, is to encrypt any sensitive data off-chain before sending it to the network. Using public-key cryptography, one method would be as follows:
 - The sensitive data is encrypted with the recipient's public key, which could have been published either on- or off-chain
 - The encrypted data is sent either to a smart contract written for the purpose of receiving it, or in the data field of a normal transaction
 - The received data is decrypted using the recipient's private key

Why are privacy and anonymity important?

- If we consider the case of a private or consortium blockchain run by a group of companies, then certain aspects of privacy are clearly important.
- For example, companies A, B, and C create a consortium blockchain. Although the network is closed to the outside world, any transactions between A and B are visible to company C, which could be undesirable for a number of reasons, especially if the three companies are in competition with each other. In addition to any business-to-business (B2B) interactions between the companies themselves, it's possible the network would involve some form of business-to-customer (B2C) interaction, meaning further types of confidentiality and privacy would be required.
- Before we look in detail at this chapter's project, let's take a look at the currently available Ethereum-based business platforms.

ERC-20 token standard

- This is the most common type of token in use today, and the one we will use in our implementation. We'll go through each part of the standard and discuss its purpose. Note that there are certain parts of the interface that must be included to adhere to the standard, and certain parts that are optional.
- Using your editor of choice, create a new file in the `contracts/` directory of your project, named `PackToken.sol`. This file will become our ERC-20 token contract.
- At the top of the file, we must first declare our Solidity compiler version, which, at the time of writing, is 0.4.24:

```
pragma solidity ^0.4.24;
```
- Copy
-
- Explain
- Under this we can declare our contract, as follows:

```
contract PackToken {
```
-
- }
-
- Copy
-
- Explain
- Let's begin by taking a look at the initial parts of the ERC-20 standard.

The Ethereum Enterprise Alliance

- The Enterprise Ethereum Alliance (EEA) is a nonprofit working group whose aim is to define an open, standards-based architecture for business and enterprise use of Ethereum. Group members include many large companies from the world of software and finance, such as Microsoft, Accenture, and J.P.Morgan. The group is currently working toward defining the Ethereum Enterprise Architecture Stack, which itself will help guide the development of the EEA's overall standards-based specification.
- EEA's specification will help define a standard way in which Ethereum's blockchain can be used for business purposes. However, given the opensource nature of Ethereum's codebase, adhering to EEA's specification won't necessarily be required: Ethereum's codebase can be used in any way desired.

Ethereum's licensing

- The Ethereum Foundation has stated that Ethereum:
- “is both open-source software and Free software after the definition of the Free Software Foundation (so-called FLOSS).”
- In reality, different parts of the Ethereum technology stack are licensed – or will be licensed—in different ways.
- The core parts of the stack, including the consensus engine, networking code, and supporting libraries, haven't yet been licensed, but are expected to be covered by either the MIT, MPL, or LGPL license. It's important to understand that the first of these is a permissive license, while the latter two are more restrictive, and restrict a user's ability to distribute any modifications under commercial terms, therefore potentially restricting how businesses can use the code.
- Regardless of the potential future restrictions that may appear once licensing is confirmed, several enterprise-centric implementations of Ethereum have already been created, with two such implementations being Quorum, created by J.P.Morgan, and Monax. Later in the chapter we will be discussing and implementing a project based on Quorum.

Blockchain-as-a-Service

- Blockchain-as-a-Service (BaaS), is a service that allows customers to create and run their own client nodes on popular public blockchain networks, or to create their own private networks for their own use and testing. The provisioned services are based in the cloud, and are analogous to the Software-as-a-Service (SaaS) model. Several of the large cloud computing providers are now offering BaaS, such as the following:

-
- Microsoft Azure enables users to quickly deploy and manage applications in the cloud, and has templates available for Ethereum, Hyperledger (see later chapters), and R3's Corda (see <https://azure.microsoft.com/en-gb/solutions/blockchain/>).
 - AWS Blockchain Templates is a very similar offering, and again provides templates for Ethereum and Hyperledger (see <https://aws.amazon.com/blockchain/templates/>). Further services are offered by IBM, HP, and Oracle, all along the same lines.

Creating an ICO

- An Initial Coin Offering (ICO) is a fundraising tool that allows a project to raise equity by trading a cryptocurrency that has immediate, liquid value, for a new currency or tokens that might have value in the future. It is a quick and easy way to run a kickstarter campaign in a decentralized manner.
-
- As the name suggests, ICOs are akin to IPOs, but in which participants receive cryptocurrencies or tokens instead of stocks or shares. For the sake of simplicity, our tutorial will assume that an ICO is the same as a token sale, though it is often argued that there are differences in what these terms mean.

What is an ICO?

- During an ICO on Ethereum, participants send ether to a special smart contract, and, in return, receive tokens of equal value, depending on the price of the token set at the start of the sale.
- In Ethereum, a token is just a balance – itself just a number – stored in a smart contract and associated with an owner's wallet address. A minimal implementation of a token would therefore only require a way to store how many tokens a given address owns. In Solidity, this can be achieved simply by using a mapping:
- Mapping (address => uint256) balances;

-
- It's important to note that a token isn't something that resides in a user's wallet, nor is it something that can be moved around. When a user sees a token in their wallet, the wallet software is really making a call into the token contract and reading the balance associated with its address. Likewise, when a wallet sends a token to another address, the wallet is really calling into the contract to change the balances associated with the to and from addresses.
 - In addition to the contract that defines the new token, an ICO also requires a contract to define the terms of the sale itself. This contract only needs to exist while the sale is ongoing, and is used to receive ether in exchange for the tokens and to define variables such as the length of the sale and the price of the new tokens.
 - Finally, while it should be possible to interact with the ICO contracts directly, projects running an ICO almost always create a more user-friendly interface in the form of a web page.

Project setup

- Before we start to look in detail at the ICO implementation, it's a good idea to set up our working environment so we can incrementally add to the code as we walk through the explanation. For this project, we'll be using the Truffle framework and the Solidity language, Peer to Peer Auction in Ethereum, and Tontine Game with Truffle and Drizzle. The tutorial assumes you are working from a Linux-style terminal.
- If you haven't already installed Truffle, do so by running the following command on the command line:
- `npm install -g truffle`

-
- For complete details on dependencies, and to troubleshoot any Truffle-specific issues, please refer to the official documentation at <https://truffleframework.com/docs/truffle/getting-started/installation>.
 - Next, create a project directory:
 - `Mkdir PacktCoin`
 - `cd PacktCoin`
 - And finally, initialize the Truffle project, which will generate an empty Truffle project with the directory structure that was introduced in Chapter 5, Tontine Game with Truffle and Drizzle:
 - Our project environment is now ready for use. Before we jump into our implementation, let's first explore the types of tokens that are currently available.

Token contracts

- As mentioned in the introduction, a token contract can take any form, as long as there is a way to map between an address and a token balance. However, standard token implementations are available that contract creators can use. These standard implementations provide a simple and uniform interface through which interactions with contracts can occur, and allow wallets and exchanges to easily list a multitude of tokens while only implementing the integration logic once.

-
- These standards are defined as Ethereum Request for Comments (ERC) token standards, and are submitted through the Ethereum Improvement Proposal (EIP) process. The open source nature of Ethereum means that anybody can submit such a proposal to the community for discussion and consideration.
 - Various token standards exist, the most important of which we'll discuss here. It's important to note that the standards define an interface, which defines what a token contract can do, but not how it does it. The implementations are not defined by the standards.

Token contract in blockchain?

- What is token contract in blockchain?
- A token contract is simply an Ethereum smart contract. “Sending tokens” actually means “calling a method on a smart contract that someone wrote and deployed”.
- People also ask
- What is token sale contract?
- A token sale, sometimes called an Initial Coin Offering (ICO), is the first stage of a token offering, where a group of buyers become the first to reserve a portion of the project’s token supply.

-
- Token contracts are smart contracts deployed on blockchain platforms like Ethereum that define and manage the issuance, transfer, and ownership of tokens. These tokens can represent assets, digital assets, or even the native currency of a blockchain network.
 - Token contracts are crucial components of decentralized applications (dApps) and play a significant role in various blockchain use cases, including ICOs, decentralized finance (DeFi), and non-fungible tokens (NFTs).

There are two main types of tokens:

Fungible Tokens:

- Fungible Tokens: Fungible tokens are interchangeable with each other and have identical values. They follow a standard called ERC-20 (Ethereum Request for Comments 20) on the Ethereum blockchain. Examples of fungible tokens include stablecoins like USDT and cryptocurrencies like ETH.

Non-Fungible Tokens (NFTs):

- Non-Fungible Tokens (NFTs): NFTs are unique and indivisible tokens that represent ownership of a specific asset or item. Each NFT has a distinct value and cannot be exchanged on a one-to-one basis. The most commonly used standard for NFTs on Ethereum is ERC-721.

- Creating a token contract involves writing and deploying the smart contract code to the blockchain network.
-

- The contract will define functions for minting (creating), transferring, and managing tokens.
- When deploying the contract, the creator typically sets the initial supply of tokens, which can later be distributed, bought, or sold according to the contract's rules.
- When designing token contracts, security is of utmost importance to prevent potential vulnerabilities and attacks.

-
- Following the best practices mentioned earlier, such as code reviews, testing, and security audits, is essential to ensure the safety and reliability of token contracts.
 - It's worth noting that while Ethereum is the most popular platform for token contracts, other blockchain platforms may have their own token standards and contracts, each with its unique characteristics and purposes.

Token sale contracts

- Token sale contracts, also known as crowdsale contracts, are smart contracts that facilitate the distribution and sale of tokens during an Initial Coin Offering (ICO), Initial Exchange Offering (IEO), or any token sale event.
- These contracts play a critical role in raising funds for a project and distributing the issued tokens to contributors.

The main functionalities of token sale contracts typically include:

- **Token Distribution:** The contract manages the issuance and distribution of tokens to participants who contribute funds (usually in the form of cryptocurrency, like ETH or BTC).
- **Price and Exchange Rate:** The contract specifies the token price or exchange rate for the contributions, determining how much of the project's tokens contributors will receive per unit of the cryptocurrency they invest.

-
- **Contribution Cap:** Token sale contracts often include a cap on the total amount of funds that can be raised during the sale. This cap helps manage the sale's size and potential risks.
 - **Time Limit:** The contract usually sets a start and end time for the token sale, defining the period during which contributors can participate.

-
- **Token Refunds:** In some cases, the contract may include mechanisms for refunding contributions if certain conditions are not met, such as not reaching the minimum funding goal.
 - **Whitelisting and KYC:** To comply with regulatory requirements or control participation, token sale contracts may implement whitelisting or know-your-customer (KYC) procedures.

-
- It's essential to ensure that token sale contracts are secure and well-audited to prevent any vulnerabilities or potential attacks. Following the security best practices mentioned earlier, such as code review, testing, and external audits, is crucial for the success of a token sale and to protect the interests of participants.

-
- Since token sales involve financial transactions, legal compliance is also a critical consideration. Consultation with legal experts is recommended to ensure that the token sale adheres to relevant laws and regulations in the targeted jurisdictions.
 - Keep in mind that the popularity and regulatory landscape of token sales may have evolved since my last update in September 2021, so it's important to stay up-to-date with the latest developments in the blockchain and cryptocurrency space.

Contract security and testing the code

- Contract security is crucial in the blockchain world, especially when dealing with smart contracts, which are self-executing contracts with the terms of the agreement directly written into code. Properly securing smart contracts is essential to prevent vulnerabilities and potential exploits.

To enhance contract security, consider the following measures:

- **Code Review:** Conduct thorough code reviews by experienced developers to identify potential issues and vulnerabilities in the smart contract code.
- **Formal Verification:** Use formal verification methods to mathematically prove the correctness of the smart contract code and ensure it adheres to the intended specifications.

-
- **Testing:** Comprehensive testing is vital to uncover bugs and vulnerabilities. Test the smart contract in various scenarios, including edge cases, to ensure it functions as expected.
 - **External Audits:** Engage third-party security auditing firms to perform in-depth security audits of the smart contract code. They can provide an unbiased evaluation and identify potential security flaws.

-
- **Secure Development Practices:** Follow secure coding best practices to minimize the risk of introducing vulnerabilities during the development process.
 - **Penetration Testing:** Conduct penetration testing to simulate attacks on the smart contract and identify potential weaknesses in its security measures.
 - **Bug Bounty Programs:** Encourage the community to participate in finding vulnerabilities by offering bug bounty programs, which incentivize developers to report security issues.

-
- Upgradeability Considerations: If the smart contract allows for upgrades, ensure that upgrade mechanisms are secure and follow best practices to prevent unauthorized modifications.
 - By employing these security measures, you can significantly reduce the risk of security breaches and ensure the robustness of your smart contract code. Keep in mind that blockchain technologies and best practices may evolve over time, so it's essential to stay up-to-date with the latest security developments.

Ethereum

Ethereum tops the list as being the first blockchain to introduce a Turing-complete language and the concept of a virtual machine. This is in stark contrast to the limited scripting language in Bitcoin and many other cryptocurrencies. With the availability of its Turing-complete language called Solidity, endless possibilities have opened for the development of decentralized applications. This blockchain was first proposed in 2013 by Vitalik Buterin, and it provides a public blockchain to develop smart contracts and decentralized applications. Currency tokens on Ethereum are called Ethers.

- Ethereum is an Open Source Blockchain platform which allows anyone to develop and deploy Blockchain based Applications. Any kind of application including cryptocurrency, tokens, wallets, social apps etc.
- can be developed and deployed in a Distributed Environment of Ethereum. In other words, rather than sticking with the cryptocurrency alone, Ethereum opened the possibilities of the 'blockchain' and 'distributed ledger' technology to other application domains.
- Ethereum is not a single network rather it is more like a protocol for internode communication. Actually, in Ethereum many networks exist alongside.
- The community Ethereum Network, Community test network and other private Blockchain networks like
 1. Private network
 2. Public test network
 3. Main Ethereum network

Distributed Storage IPFS and Swarm

In our discussions of Ethereum so far, it should have become clear that using the Ethereum blockchain to store large amounts of data is neither cost-effective, nor what it was designed for. Storing information in the blockchain means storing it as state data, and doing so requires payment in the form of gas.

In this chapter, we will look at ways that data can be stored in a distributed manner by way of two different platforms: IPFS and Swarm. After introducing both platforms, we will describe how to install and make basic use of them, and demonstrate how both can be used to host the ICO DApp we created in [Chapter 8, *Creating an ICO*](#). Finally, we will create a small project to help us become familiar with how IPFS can be used programmatically in a website frontend.

Background

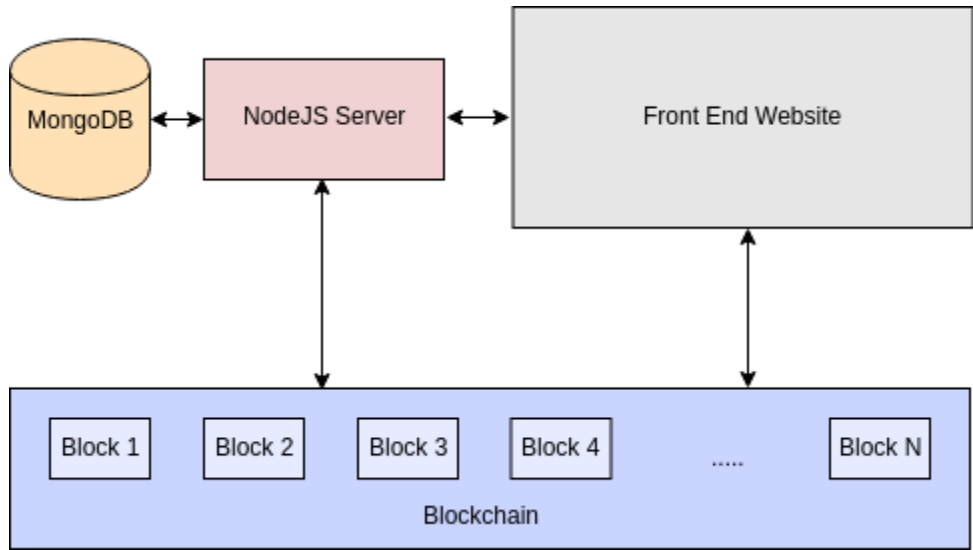
The Ethereum Virtual Machine (EVM) operates on words that are 256 bits, or 32 bytes, in size. Each 256-bit word of data costs 20,000 gas units to store, equating to 640,000 gas units per kilobyte. At the time of writing, the gas price is around 4.5 Gwei (0.0000000045 ETH), making a kilobyte of data cost 0.00288 ETH.

Scaling this up gives a cost of 2,880 ETH per GB of data, with the current price of \$220 per ETH giving each GB a price tag of \$621,000. This is an extremely high cost as compared to conventional centralized cloud storage, where the costs per GB are usually in the region of cents.

If we can't store large amounts of data in the blockchain itself, then the logical alternative would be to store it in a centralized storage layer, while making it available to the data layer located on a blockchain. An example of this would be a DApp that uses the blockchain as its

decentralized data layer backend, but with the storage layer and frontend hosted on a conventional, centralized server.

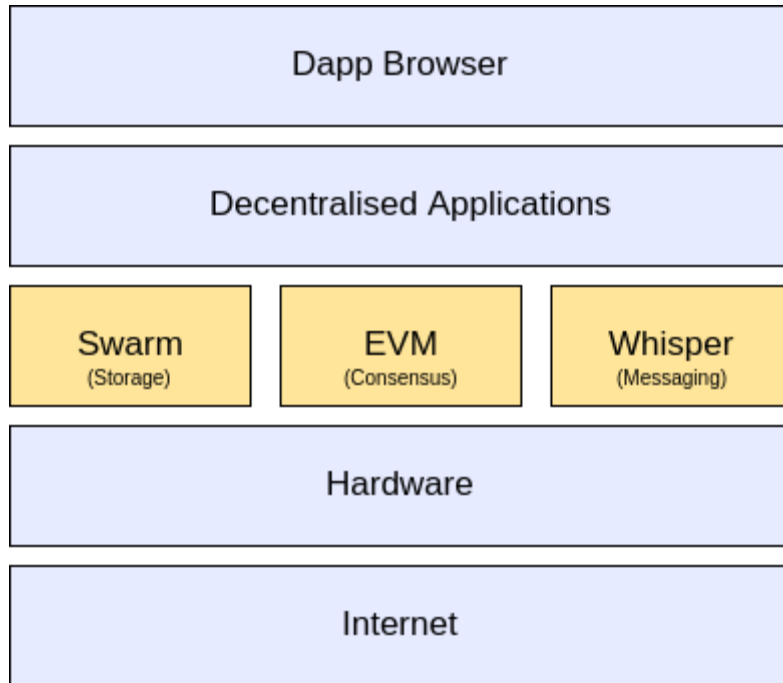
The following diagram shows how this is usually achieved:



A decentralized blockchain backend with a centralized frontend and storage

For some, this is an acceptable compromise between a decentralized data layer and a centralized storage layer, the latter being a necessary evil to get around the costs of storing data in the blockchain itself. It is, however, possible to do better, and make the storage layer decentralized, as well, without needing to use a blockchain directly.

In early versions of the Web3 technology stack, decentralized storage was provided by Ethereum's **Swarm** platform, which was designed to store and distribute a DApp's frontend code, as well as general blockchain data:



The Web3 technology stack

Another service, called **InterPlanetary File System (IPFS)**, provides an alternative to Swarm. The two platforms are similar, and to some extent, they can be used interchangeably, as will be discussed in more detail in subsequent sections.

Swarm and IPFS

Before looking at how we can make use of each of the two alternatives for decentralized storage in detail, we'll first briefly look at their main similarities and differences.

The aim of each project is to provide both a general decentralized storage layer and a content delivery protocol. To do so, both technologies use peer-to-peer networks composed of client nodes, which are able to store and retrieve content. The files that are stored on each of the platforms are addressed by the hashes of their content.

A result of being able to store files is that both IPFS and Swarm are able to store and serve the HTML, CSS, and JavaScript of applications built on top of them, and can therefore take the place of traditional server backends.

For files that are too large to be stored whole, both projects offer a model whereby larger files can be served in chunks, much the same as in the BitTorrent protocol. One of the main issues with the BitTorrent protocol is that users are not incentivized to host, or seed, content, creating a one-sided system in which many downloaders feed from a few hosts.

To mitigate similar issues, IPFS and Swarm are able to incentivize users to run clients by way of monetary rewards. For Swarm, the incentives are built in, as Swarm must be run in conjunction with an Ethereum Geth client.

For IPFS, a separate incentive layer must be applied, in the form of Filecoin (see <http://filecoin.io>).

Although the two platforms are similar in many ways, there are also differences. Firstly—and perhaps most importantly, from the perspective of a developer—the IPFS project is more mature, and has a higher level of adoption, despite Swarm's more integral position in the Ethereum ecosystem.

Further differences mainly involve the technologies from which each platform is built. For example, Swarm uses a **content-addressed chunkstore**, rather than the **distributed hash table (DHT)** used by IPFS. A further example is that, due to its close association with the rest of the Ethereum stack, Swarm is able to make use of Ethereum's DevP2P protocol (<https://github.com/ethereum/wiki/wiki/%C3%90%CE%9EVp2p-Wire-Protocol>), whereas IPFS uses the more generic libp2p network layer (<https://github.com/libp2p>).

For the purposes of our discussion, these comparisons should be sufficient. A much more detailed comparison can be found online, at <https://github.com/ethersphere/go-ethereum/wiki/IPFS-&-SWARM>.

Installing IPFS

We will start by installing IPFS locally on our machine, which will give us the tools required to upload and view content on the IPFS network. Installation processes will vary depending on your machine's architecture—full instructions can be found at <https://ipfs.io/docs/install/>.

Once IPFS has been installed, we can initialize our node as follows:

```
ipfs init
```

CopyExplain

Once this has completed correctly, the following output will displayed:

```
initializing ipfs node at /Users/jbenet/.go-ipfs
```

```
generating 2048-bit RSA keypair...done
```

```
peer identity: Qmcpo2iLBikrdfs1d6QU6vXuNb6P7hwrBNPW9kLAH8eG67z
```

```
to get started, enter:
```

```
ipfs cat /ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv/readme
```

CopyExplain

Run the suggested `ipfs cat` command to read a welcome file:

```
Hello and Welcome to IPFS!
```

```
███ 7 ██████████ 7 ██████████ 7 ██████████ 7
```

```
███ || ███ 7 ███ 7 ███ 7 ███ 7
```

```
███ || ██████████ 7 ██████████ 7 ██████████ 7
```

```
███ || ███ 7 ███ 7 ███ 7
```

```
███ || ███ || ██████████ ||
```

```
┌┌┌  ┌┌┌  ┌┌┌┌┌┌┌
```

```
If you're seeing this, you have successfully installed
```

```
IPFS and are now interfacing with the ipfs merkle dag!
```

```
-----
```

```
| Warning: |
```

```
| This is alpha software. Use at your own discretion! |
```

```
| Much is missing or lacking polish. There are bugs. |  
| Not yet secure. Read the security notes for more. |
```

```
Check out some of the other files in this directory:
```

```
./about
```

```
./help
```

```
./quick-start <-- usage examples
```

```
./readme <-- this file
```

```
./security-notes
```

CopyExplain

We've now initialized our node, but we haven't yet connected to the network. To do so, we can run the following command:

```
ipfs daemon
```

CopyExplain

To check that we are connected correctly, we can view the nodes in the network that we're directly connected to, as follows:

```
ipfs swarm peers
```

CopyExplain

Having connected to the network, it will now be possible to access the documents stored and distributed by it. To test whether we can do this, we'll use a test file that already resides on the network.

In this case, it's an image of a cat, and, because it's an image file, we'll first need to direct the data to a new file for viewing:

```
ipfs cat /ipfs/QmW2WQi7j6c7UgJTArActp7tDNikE4B2qXtFCfLPdsgaTQ/cat.jpg
```

```
>cat.jpg
```

Copy Explain

This is perhaps a confusing example: we're using the `ipfs cat` command to show an object stored in IPFS, where the object itself is a picture of a cat, named `cat.jpg`.

As well as accessing the file directly, the file can also be viewed by using any of the following options:

- A local browser-based userinterface (<http://localhost:5001/webui>):



Enter a hash

GO

Home

Connections

Files

DAG

Config

NODE INFO

Peer ID `QmZW1zPDKUtTbcbaXDYCD2jUodw9A2sNffJNeEy8eWK3bG`

Location Leeds, H3, United Kingdom, Earth

Agent Version `go-ipfs/0.4.17/`

Protocol Version `ipfs/0.1.0`

Public Key

```
CAASpgIwggEiMA0GCSqGSIB3DQEBAQUAA4IBDwAwggEKAoIBAQC5yEMM9snFy2
```

NETWORK ADDRESSES

```
/ip4/127.0.0.1/tcp/4001/ipfs/QmZW1zPDKUtTbcbaXDYCD2jUodw9A2sNffJN
```

```
/ip4/192.168.1.198/tcp/4001/ipfs/QmZW1zPDKUtTbcbaXDYCD2jUodw9A2sN
```

```
/ip6>:::1/tcp/4001/ipfs/QmZW1zPDKUtTbcbaXDYCD2jUodw9A2sNffJNeEy8eW
```

```
/ip4/95.150.46.139/tcp/4001/ipfs/QmZW1zPDKUtTbcbaXDYCD2jUodw9A2sN
```

```
/ip4/95.150.46.139/tcp/12727/ipfs/QmZW1zPDKUtTbcbaXDYCD2jUodw9A2s
```

- The local IPFS gateway that is run by our client, on port **8080**:

```
curl
```

```
"http://127.0.0.1:8080/ipfs/QmW2WQi7j6c7UgJTarActp7tDNike4B2qXtFCfLPdsgaTQ/
```

```
cat.jpg" > local_gateway_cat.jpg
```

Copy Explain

- A remote public IPFS gateway URL:

```
curl
```

```
"https://ipfs.io/ipfs/QmW2WQi7j6c7UgJTarActp7tDNike4B2qXtFCfLPdsgaTQ/cat.jp
```

```
g" > public_gateway_cat.jpg
```

CopyExplain

Now that our local client is running correctly, and we are able to interact with the network, we can move on to adding files and directories.

The simplest way to add a single file is to use the following command, passing it the path to the file that you want to upload:

```
ipfs add <file>
```

CopyExplain

Running this command returns a hash, which is used as the file's IPFS address. This address is based on the contents of the file, and can be used to access the file, as shown previously.

We'll return to adding files later, when we add the files associated with our ICO website to the network.

Installing Swarm

Installing Swarm is slightly more involved than IPFS, and it depends on a running Geth client. If you haven't already done so, ensure that Geth is installed by using the appropriate instructions for your OS (see <https://github.com/ethereum/go-ethereum/wiki/Installing-Geth>).

Once this is complete, install Swarm, also using the appropriate instructions for your OS, from <https://swarm-guide.readthedocs.io/en/latest/installation.html>.

Once it has been installed, we can check for a successful installation by using the following command:

```
$ swarm version
```

```
Swarm
```

```
Version: 0.3.2-stable
```

```
Git Commit: 316fc7ecfc10d06603f1358c1f4c1020ec36dd2a
```

```
Go Version: go1.10.1
```

```
OS: linux
```

CopyExplain

Before we run our local Swarm client, if Geth doesn't already have an account configured, we should create one by using the following command:

```
geth account new
```

CopyExplain

This account can then be used to connect to the Swarm network, as follows:

```
swarm --bzzaccount <your_account_address>
```

CopyExplain

Once the client is running, we can upload a test file from another Terminal. Here, we use the picture of the cat that we downloaded from IPFS:

```
$ swarm up cat.jpg
```

```
5f94304f82dacf595ff51ea0270b8f8ecd593ff2230c587d129717ec9bcbf920
```

CopyExplain

The returned hash is the hash of the Swarm manifest associated with the file. This is a JSON file containing the `cat.jpg` file as its only entry. When we uploaded the `cat.jpg` file, the manifest was uploaded with it, allowing for the main file to be returned with the correct MIME type.

To check that the file has uploaded, we can use one of several options. First, we can access it through our local Swarm client's HTTP API on port `8500`, using a web page that it serves at `http://localhost:8500`. From here, entering the hash returned by Swarm will display the picture of our cat.

The second option is to access the file through Swarm's public gateway, at http://swarm-gateways.net/bzz:<file_hash> (substituting your own hash at the end).

Hosting our frontend

In [Chapter 8](#), *Creating an ICO*, we created all of the necessary components of an ICO. Our backend consisted of the ERC-20 token contract, together with the token sale contract, both of which were deployed to the Rinkeby network.

We also created a frontend in the form of a standard HTML, CSS, and JavaScript website, hosted on a local web server. Our frontend interacted with the blockchain backend by way of the Web3 JavaScript library and the MetaMask browser add-on.

Now, we want to remove our reliance on the centralized frontend, and replace it with a frontend served by either IPFS or Swarm.

Serving your frontend using IPFS

Let's start by deploying our frontend to IPFS. First, we need to decide which files are required by the frontend, and add them to a directory that we can add to IPFS. With reference to [Chapter 8](#), *Creating an ICO*, the following files will be required:

- The frontend code that we created in the `src/` directory
- The contract abstractions, in the form of JSON files, in the `build/contract/` directory

From the root of our ICO project, add the contents of these directories to a new directory that we'll use for the deployment to IPFS:

```
$ mkdir dist
$ rsync -r src/ dist/
$ rsync -r build/contract/ dist/
```

CopyExplain

Then, add the new directory to IPFS by using the `add` command that we met earlier, passing it the recursive `-r` flag, as follows:

```
$ ipfs add -r dist/
added QmXnB22ZzXCw2g5AA1EnNT1hTxexjrYwSzDjJgg8iYnubQ dist/Migrations.json
added Qmbd5sTquZu4hXEvU3pQSUUsspL1vxoEZqiEkaJo5FG7sx dist/PacktToken.json
added QmeBQWLkTZDw84RdQtTtAG9mzyq39zZ8FvmsfAKt9tsruC
dist/PacktTokenSale.json
added QmNi3yyX7gGTCb68C37JykG6hkCvqxMjXcizY1fb1JsXv9 dist/SafeMath.json
added Qmbz7fF7h4QaRFabyPUc7ty2MN5RGg9gJkodq2b8UydkYP dist/index.html
added QmfKUdy8NGaYxsw5Tn641XGJcbJ1jQyRsfvkDKej56kcXy dist/js/app.js
added Qmc3xyRTJ2wNt2Ep5BFvGnSRyQcjk5FhYXkVfvobG26XAm dist/js
added QmQytFqNoyk8H1ZEaHe9wkGcbcUgaRbReEzurPBYRnbjNB dist
```

CopyExplain

With the files required for our site uploaded, we can access our fully decentralized ICO web page by taking the hash corresponding to the parent directory and viewing it via a public IPFS URL, as follows:

<https://ipfs.io/ipfs/QmQytFqNoyk8H1ZEaHe9wkGcbcUgaRbReEzurPBYRnbjNB/>.

This will bring up a fully decentralized version of our ICO website, which can be interacted with in the usual way, using the MetaMask browser add-on.

Note that the page may take longer to load than a centrally served frontend: this decrease in performance should perhaps be considered the price paid to make our site truly decentralized and censorship-resistant.

Using IPNS

Whenever we need to update any of our ICO frontend files, the associated IPFS file hashes will change, meaning that the IPFS path that we first generated for our project will no longer equate to the most recent version of the files. This will cause the public URL to change whenever file changes are made.

To solve this problem, we can use **IPNS** (short for **InterPlanetary Naming System**), which allows us to publish a link that won't change when the underlying files are changed.

Let's publish the first iteration of the site to IPNS, using the hash of the main directory:

```
ipfs name publish QmQytFqNoyk8H1ZEaHe9wkGcbcUgaRbReEzurPBYRnbjNB
```

CopyExplain

After a while, this will return a hash similar to the following:

```
Published to QmZW1zPDKUtTbcbaXDYCD2jUodw9A2sNffJNeEy8eWK3bG:
```

```
/ipfs/QmQytFqNoyk8H1ZEaHe9wkGcbcUgaRbReEzurPBYRnbjNB
```

CopyExplain

This shows a new hash, which is the peerID, along with the path that we are publishing to it. We can check that the peerID correctly resolves to our IPFS path, as follows:

```
$ ipfs name resolve QmZW1zPDKUtTbcbaXDYCD2jUodw9A2sNffJNeEy8eWK3bG
```

```
/ipfs/QmQytFqNoyk8H1ZEaHe9wkGcbcUgaRbReEzurPBYRnbjNB
```

CopyExplain

The most recent version of our site will now be available at the following URL (note that we are now using `ipns/` instead

of `ipfs/`): <https://ipfs.io/ipns/QmZW1zPDKUtTbcbaXDYCD2jUodw9A2sNffJNeEy8eWK3bG/>.

Whenever the files of the frontend are updated, they can be published to the same IPNS path:

```
$ ipfs add -r dist/
```

```
$ ipfs name publish <new_directory_hash>
```

CopyExplain

We can now access the most up-to-date version of our site without needing to know the updated hash associated with our changed `dist/` directory.

Our IPNS hash, however, is still not very user-friendly. Therefore, the next logical step would be to bind our IPNS file path to a static domain, something which can be done by purchasing a domain and making that domain publicly known. Doing this, would require us to alter the DNS TXT record associated with our domain, to point to our IPNS URL.

There is one problem with this: it increases our reliance on centralization. Using a centralized DNS server to resolve our site's address would defeat the object of using IPFS and IPNS to begin with. For now, this isn't something that we will do.

Our site is now fully decentralized, and we have published our files on IPNS. In the next section, we will explore the equivalent operations with Swarm.

Serving your frontend using Swarm

The method for adding our site to Swarm is almost identical to the previous method. With Swarm running, as outlined earlier, we will use our existing `dist/` directory, along with the `-recursive` flag.

In addition, we will pass a default path that specifies the file to render in the case of a request for a resource with an empty path, as follows:

```
$ swarm --defaultpath dist/index.html --recursive up dist/  
2a504aac8d02f7715bea19c6c19b5a2be8f7ab9442297b2b64bbb04736de9699
```

Copy Explain

Having uploaded our files, the site can then be viewed through our local Swarm client using the generated hash, as

follows: <http://localhost:8500/bzz:/2a504aac8d02f7715bea19c6c19b5a2be8f7ab9442297b2b64bbb04736de9699/>.

In the same way that we can use a public gateway to view our IPFS-hosted site, we can also use Swarm's public gateway: <http://swarm-gateways.net/bzz:/2a504aac8d02f7715bea19c6c19b5a2be8f7ab9442297b2b64bbb04736de9699/>.

Our frontend is now deployed to Swarm, but, as with the case for IPFS, our public-facing URL isn't very user-friendly. To solve this problem, we will use the **Ethereum Name**

Service (ENS), which is a distributed and extensible naming system based on the Ethereum blockchain.

IPFS file uploader project

In the second part of this chapter, we will take a more detailed look at how we can use IPFS's HTTP API programmatically. To do so, we will create a simple HTML and JavaScript web page from which we can upload files to IPFS directly, without the need to first upload the files to a backend server.

To do this, we will use the JS-IPFS-API JavaScript library, which will allow a browser-based frontend to communicate directly with the local IPFS node that we created earlier in the chapter.

Note that this API library should not be confused with JS-IPFS, which is a JavaScript implementation of the main IPFS protocol.

The aim here is not to explore the HTTP API completely, but to provide a basic example of how it can be used. To learn more about the methods made available by the API, the relevant documentation should be consulted, at <https://github.com/ipfs/js-ipfs-api>.

Project setup

First, we need to run IPFS by using the appropriate configuration, which will differ from the configuration that we used earlier in the chapter. If IPFS is still running from earlier, first, stop the process.

Next, we need to apply some configurations that will allow IPFS to perform cross-origin requests, or Cross-Origin Resource Sharing (CORS). This will allow our web page, which will be run by a local web server on its own local domain, to interact with the local IPFS gateway, itself hosted locally on a different domain:

```
$ ipfs config --json API.HTTPHeaders.Access-Control-Allow-Methods
'["GET", "POST", "PUT", "OPTIONS"]'
$ ipfs config --json API.HTTPHeaders.Access-Control-Allow-Origin '["*"]'
```

Having done this, we can start IPFS again by using the `ipfs daemon` command.

For the purposes of this project, we will run our web page on a local development web server. An exercise for the reader would be to follow the instructions earlier in this chapter to upload the site itself to IPFS, thereby giving us an IPFS-hosted IPFS file uploader!

First, let's create our project directory structure:

```
$ mkdir packtIpfsUploader
$ cd packtIpfsUploader
```

CopyExplain

We can initialize `npm` and walk through the suggested values:

```
npm init
```

CopyExplain

Then, we'll install the web server package:

```
npm install --save lite-server
```

CopyExplain

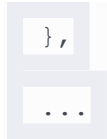
And finally, we'll edit the `package.json` file to include a way to easily start the server:

```
...
```

```
"scripts": {
```

```
  "test": "echo \"Error: no test specified\" && exit 1",
```

```
  "dev": "lite-server"
```



Copy Explain

The web page

Our web site will be a single page, created from two files: an `index.html` file and a `main.js` JavaScript file. It will have a single input for specifying the local file to upload, and a button to initiate the upload.

Once uploaded, a link to the file on IPFS will be shown on the page, as shown in the following screenshot:

IPFS File Uploader

Web-3.0-Tech-Stack.jpg	Browse	Upload
------------------------	--------	--------

File uploaded to:

<https://ipfs.io/ipfs/QmcbPuwUJPRSDKo7gajbTtXadbXxdYgA5dCdkimKMvfBFV>

User interface following a successful file upload