Unit 3 Notes

What is Distributed Consensus?

- It is a procedure to reach a **common agreement** in a **distributed** and **decentralized** environment.
- Important for message passing system.(HTTP, RPC, or custom protocol built)
- It sets the rules that all participants must follow to process transaction.
- Common agreement about present state of ledger.
- A group of independent systems to agree on a **single version of the truth**

Types of Distributed Consensus

- Synchronous
- It is guaranteed that messages will be delivered within some **fixed, known amount of time.**
- Less complex(fixed upper bound of how long the message will take to reach its destination.)

- Asynchronous
- There's **no guarantee of a message being delivered**.
- It is assumed that a network may delay messages infinitely, duplicate them, or deliver them out of order. (no fixed upper bound)

One decision maker → no consensus required
multiple decision maker → consensus required

How does Consensus work?

- All the nodes need to agree the correctness of this transactions.
- Each node has set of outstanding transactions it's heard about.

Why consensus is hard?

- Nodes may **crash**
- Nodes may be **malicious**
- **Network is imperfect**
  - Not all pairs of nodes connected
  - Faults in network
  - Latency

Types of Consensus Algorithms

- Proof of Work (PoW)

- Proof of Stake (PoS)
- Delegated Proof of Stake (Dpos)
- Proof of Authority
- Proof of Elapsed Time
- Proof of Scope
- Proof of Space
- Proof of Burn
- RAFT
- PAXOS
- Byzantine Fault Tolerance System
- PBFT.

Fault Tolerant

Crash Fault Tolerance: **Paxos, RAFT**
Byzantine Fault Tolerance: **BFT, PBFT**

How can you choose the right consensus protocol for your Blockchain?

- The **speed** in which your blocks will need to be written into the Blockchain?
- What **type of network** will you be using?? (synchronous, partially synchronous, eventually synchronous or asynchronous)
- **How many miners, writers, or validators** do you think you will need?
- How "**final**" does a block need to be?
- To what degree do you put your **trust** in the nodes/operators?

Permissioned Blockchain

- Depends on factors like the extent of decentralization required, how much the participants in a network trust each other, the number of permissions that must be granted.
- Eg: **Practical byzantine fault tolerance** and its variants

Permissionless Blockchain

- It is crucial to choose the right consensus model for your specific Blockchain.
- **PoW, PoET**

Permissionless Model

- Open network
- Anyone can join in the network and initiate transactions
- Participants are free to leave the network, and can join later again
- Assumption: More than 50% of the participants are honest
- A society cannot run if majority of its participants are dishonest !!

Consensus challenges

1. **Participants do not know others**
   - Cannot use message passing !!
2. **Anyone can propose a new block**
   - Who is going to add the next block in the blockchain?
3. **The network is asynchronous**
   - We do not have any global clock
   - A node may see the blocks in different orders
4. **Any types of monopoly needs to be prevented**
   - A single user or a group of users should not gain the control – we don't trust anyone
5. **Others:**
   - No fixed ordering of transactions
   - No fixed number of transactions per block
   - Limit on the Block size

Safety vs Liveness

Safety-1: The next block should be "correct" in practice
• Transactions are verified, block contains correct Hash and Nonce

Safety-2: All the miners should agree on a single block
• The next block of the blockchain should be selected unanimously

Liveness: Add a block as long as it is correct
(contains valid transactions from the unconfirmed TX list and move further)

- Two (or more) different miners may add two (or more) different blocks
- Resolved using Forking

Proof of Work

- The idea for Proof of Work(PoW) was first published in 1993 by Cynthia Dwork and Moni Naor.
- Applied by Satoshi Nakamoto in the Bitcoin paper in 2008.
- **Competitive consensus algorithm.**
- Solving a computational challenging puzzle in order to create new- blocks.

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

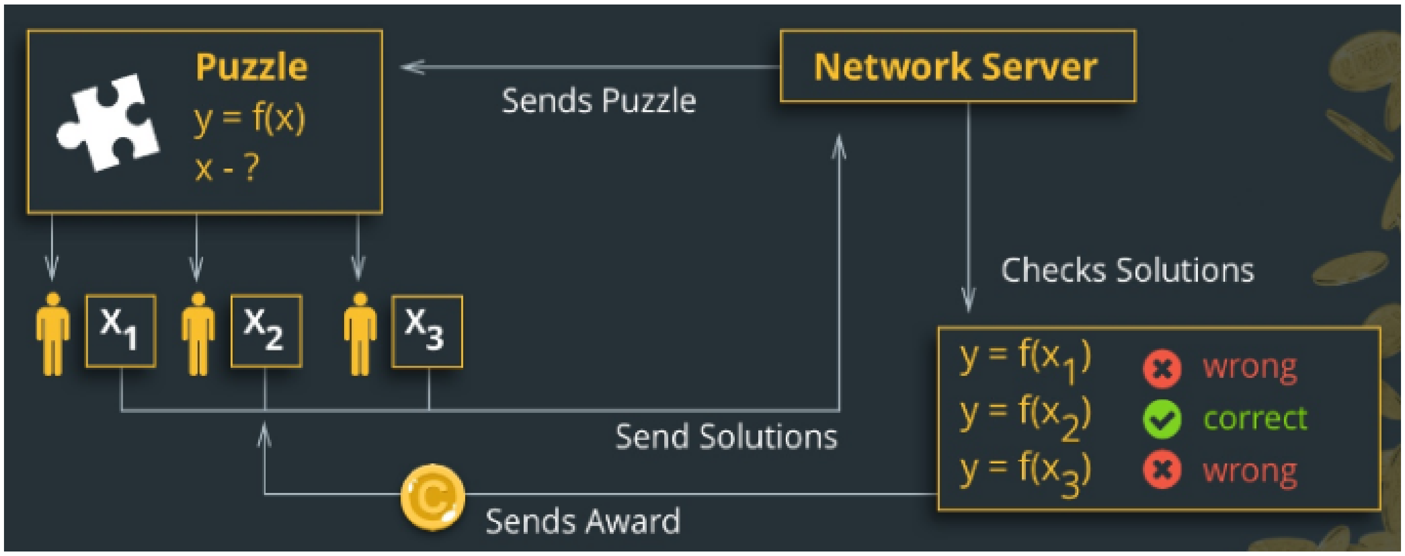
Consensus Algorithm – Principle

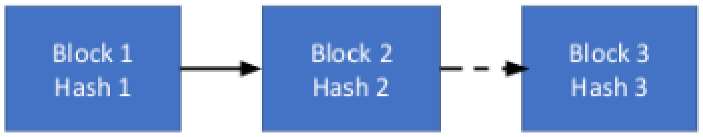
A solution that is difficult to generate but is easy to verify.

Example:
N leading zeros
If n=4, block hash should have 4 leading zeros
0000AC23EF32DD3422...
More the value of n ,difficulty increases(more complex, more power)

Working of POW

- Each node(miners) needs to solve math puzzle to propose transaction.
- Lot of **computation, power and time.**
- Finding nonce value which along with **input message + hash value of the previous block* gives the puzzle hash
- Math puzzle should be less than the difficulty level of the systems
- Once puzzle is solved, all other nodes should a**gree to solution.**
- The **node that is first to solve**, **rewarded for his work**





Example

- New Block, block 3 needs to be added to the existing chain.

- Calculate hash3
  H(Block 3,hash 2, Nonce) < Difficulty level(0000000....234)

Incrementing nonce helps in solving puzzle
51% more of the miners agree with the proposed nonce, the transactions on the winner's block are considered to be correct -(system won't waste time waiting for 49% to send their verification)

- **Bitcoin transfers to all 100%** for agreement
- **Other blockchains** may use the **51% rule**(financial sector)(incentive tokens)
  This percentage can vary depending on the Blockchain in question
- The miner with the correct answer will be rewarded
  - Usually the reward is given in **cryptocurrencies/tokens**
- If the majority of miners **do not agree with the nonce , no reward is given**
- Miners have no incentive to cheat - **strong incentive towards honesty**

N = number of nodes on network

- Adding a node to the **network increases security by 1/N**
- **Increases transacting time by 1/N**

Cryptocurrencies that use POW

- Litecoin
- Ethereum
- Monero coin
- Dodge coin
- Bitcoin

Currently, Miners receive 6.25 bitcoins as reward.

Do all nodes need to validate?

- Broadcasting - 2 ways

1. Transaction generated - passed to peer nodes, and so on, they **verify valid user of network and transaction**, then pass to neighboring nodes(everyone has to verify)
2. 51% is enough to **validate the block**

Bitcoin's Proof of Work System

- Bitcoin uses the **Hashcash** Proof of Work system as the mining basis.
- The ‘hard mathematical problem’ can be written in an abstract way like below :
  Given data A, find a number x such as that the hash of A appended to x results in a number equal to B.

Hashcash is a proof-of-concept system proposed in 1997 that demonstrates an implementation of this idea.

- Hashcash is designed to **combat email spam.**

Situations (Email spam):

- Sender A sends 10000 fake messages → makes money as soon as a user reads it
- You are trying to send a message → but the receiver is not opening it

Idea is :

- Sender adds some value to the message (that is difficult to be attacked)
- Receiver sees the attached value and convinced that the message is not fake.

Spammers, whose business model relies on their ability to send large numbers of emails with very little cost per message.
For email uses, **a textual encoding of a hashcash stamp is added to the header of an email** to prove the sender has expended a modest **amount of CPU time calculating** the stamp prior to sending the email. As the sender has taken a certain amount of time **(computationally expensive)** to generate the stamp and send the email, it is unlikely that they are a spammer.

The header contains :

- The **recipient's email address**
- The **date of the message**
- **Information** proving that the **required computation** has been **performed.**

The presence of the recipient's email address requires that a **different header be computed** for each recipient.
The **date allows the recipient to record headers received recently** and to ensure that the header is unique to the email message.

Hashcash adds the following header line to the email:
X-Hashcash:
1:20:1303030600:michelle@cypherspace.org::McMybZIhxKXu57jd:ckvi

1→ **ver**: Hashcash format version, 1(which supersedes version 0)
20→ **bits** : Number of "partial pre-image"(zero) bits in the hashed code
130303 0600→ **date**: The time that the message was sent, in the format YYMMDD[hhmm[ss]].
michelle@cypherspace.org → **resource** : Resource data string being transmitted, e.g., an IP address or email address.
ext: Extension (optional; ignored in version 1)
McMybZIhxKXu57jd →**rand**: String of random characters , encoded in base 64 format
ckvi→**counter**: Binary Counter, encoded in base 64 format

Proof of Work(PoW)

Pros:

- It has been tested since 2009 and still works great
- It is slower(you need to mine the block) and safer(majority agreement)- you know your transaction will not be rolled back
- It is trustless - no one can block your transaction from processing

Cons:

- 51% risk
- **Slow**(10 mins to wait for your transaction to be confirmed)
- **Costly**(transaction cost/time can go up with the number of users)
- It is susceptible to **centralization** over time- those with the most resources can pool together their efforts in mining(high computation power)→ They can beat other miners(hence gain more, solve the problem first and add blocks to the blockchain)

What is wrong with proof of work?
The higher computational resources people will always solve the problem
OFFENSE: wasting electricity with the constant computation power(damaging environment)
DEFENSE: We are finally solving the problem(defense of proof of work). Change in difficulty(nonce), makes it more difficult for us to mine a block.

Proof of Stake

- **Competitive consensus algorithm**- Debate for PoW
- Alternative to PoW
- **Peercoin** was the first currency to utilise the PoS

Why Proof of Stake claims to better:

- Attempt to **overcome scalability concerns** imposed by PoW consensus
    - Removes the guessing game from consensus
    - Mining no longer requires specialized and powerful hardware
- Many feel that for PoW, specialized **hardware requirements lead to centralization**
    - Blockchain is about de-centralization
- **Less energy intensive** form of consensus
    - Addresses concerns about “green” mining

- **Mining can be performed from anywhere**
    - Reduces concerns around geo-political centralization

How does proof of Stake work?

- Block(pool of transactions created)
- When it's time for group consensus, all who wish to participate **lock up funds as stake**
- Participants must lock a certain amount of its currencies, called stake, into an **escrow account** in order to participate in the block creation process
- A **random player is selected**
- That **players' block data is shown to all other participants**
- **Other players stake on the validity of the block transactions**
- **Forging/validators**

How is a 'random' player is selected?

1. A person with more money, more centralization. more chance
2. A person who holds the stake for a longer time(person in the network for very long, are more trustworthy, **coinage**)
3. randomly

Consensus Algorithm – Proof of Stake(PoS)

- Nodes that have given the stake, are **validators, now can mine blocks**
- **Broadcasting will be only given to validators** not to all **for verification**
- Proof of stake- **leads to forging**
- Here after the miner(forger) makes the block, broadcasting the node is only to the other remaining validators

If the **majority agree with the proposed block**, the random player is **rewarded** *(in some implementations, those who voted for the winner are rewarded as well)*

If the **majority disagree**, the random player gets **no reward AND loses their stake!** *(In some implementations, those who voted for the loser may loose a portion of their stake as well.)*

- Then a new player is randomly selected to share their block data.
- Bad validators and forgers are removed from network
- No “computing” is ever performed during consensus, only staking/ wagering
- Any kind of device can stake, regardless of computing power
- **CRITIC:** Some argue that this also leads to **centralization** as only players who can afford to stake are able to participate in consensus

Ethereum 2.0 (is proof of stake)

- **1.0 Ethereum is POW**
- Ethereum requires **32 ETH** to be staked before a user can become a validator
- Blocks are validated by more than one validator, and when a specific number of validators verify that the block is accurate, it is finalized & closed
- Different proof-of-stake mechanisms may use different methods for validating blocks—when Ethereum transitions to PoS, it will use **shards** for transaction submissions.
- A validator will verify the transactions and add them to a shard block, which requires at least **128 validators** to attest to.
- Once shards are validated and block created, **two-thirds** of the validators must agree that the transaction is valid, then the block is closed.

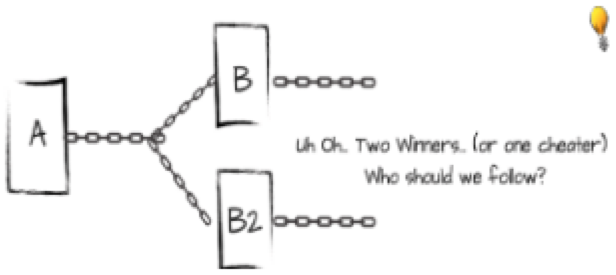
Is it possible to consensus in Bitcoin can be change to PoS???

Yes.
Etherum began its existence using PoW and is transitioning to
PoS, but the process can take years to implement in an
already established cryptocurrency.

Nothing at stake

- Theoretical **security hole in proof-of-stake systems.**
- The nothing-at-stake problem describes a scenario where block creators on generic proof-of-stake protocols have **nothing to lose when the network forks.**
- The nothing-at-stake-problem can be explained by opportunity cost. Miners can follow both chains and **reap the rewards at no additional cost** to their original deposit.

forking- two or more competing blocks
Two branches of blockchain ledger



- Stake your crypto on both blocks
- Little cost and economic incentive
- Pow not economically viable on both chains

Proof of Stake

Pros:

- It is energy efficient and **does not burn electricity when mining**

- It can be **more expensive to attack than PoW**- hackers need to purchase a large percentage of the native cryptocurrency
- It **scales easily** to handle transaction **load and size**

Cons:

- Rewards are weighted to those who stake their cryptocurrency the longest. **The longer a miner stakes, the greater the reward.**
- The network structure allows wealthy stakes to control more of the network and this may **centralization and censorship**.
- PoS is used by **Ethereum, Peercoin and Nxt**

Delegated Proof of Stake(DPoS)

- Reputation scores, better website and social media accounts are used to select the set of validators .
- Stakeholders elect **"witnesses"** who will **validate transactions and create blocks**
- EOS one of the most popular DPoS Blockchains, only has **21 witnesses.**
- Paid fees for producing blocks
- Produce blocks one at a time in a **round-robin** fashion or by **random selection.**

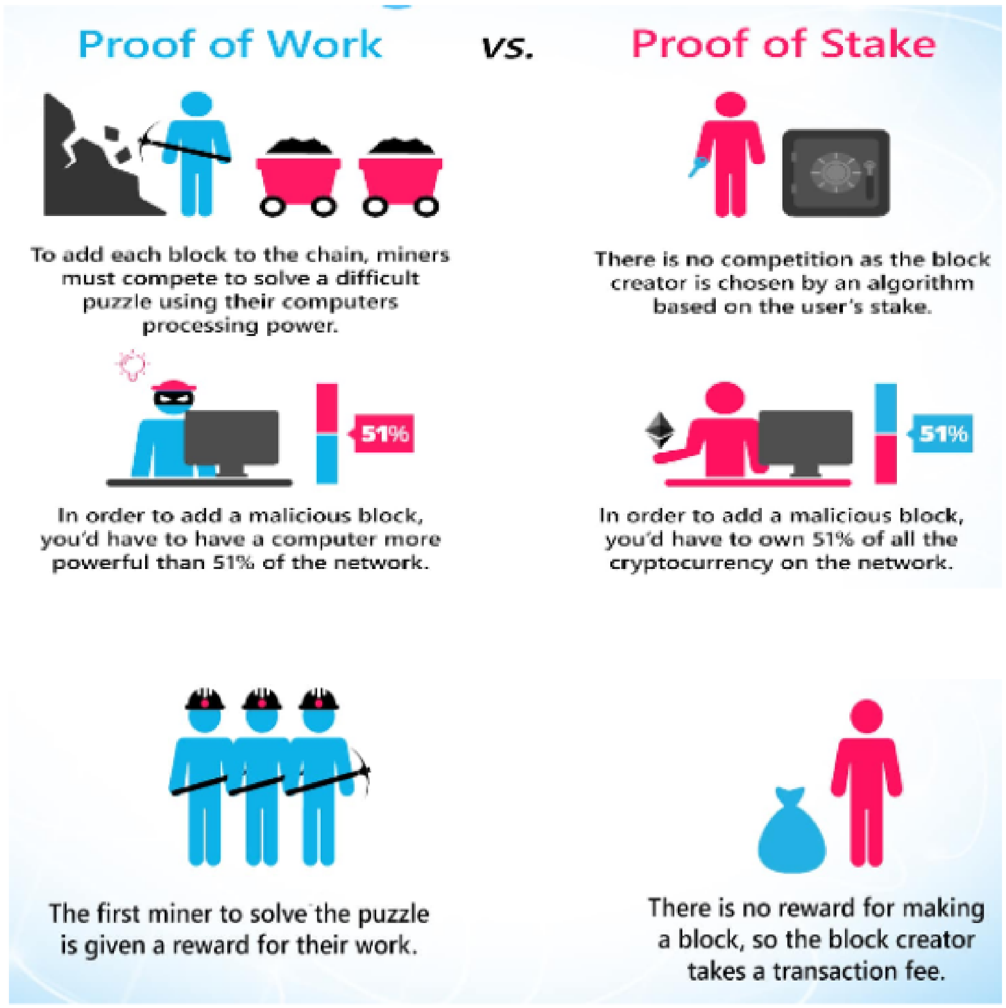
Pros:

- Like PoS, It is energy efficient
- It is fast- for example, EOS has a block time of 0.5 second.

Cons:

- It is a centralized system- this may make it prone to corruption

PoW vs PoS


Proof of Work
vs.
Proof of Stake
To add each block to the chain, miners must compete to solve a difficult puzzle using their computers processing power.
There is no competition as the block creator is chosen by an algorithm based on the user's stake.
51%
51%
In order to add a malicious block, you'd have to have a computer more powerful than 51% of the network.
In order to add a malicious block, you'd have to own 51% of all the cryptocurrency on the network.
The first miner to solve the puzzle is given a reward for their work.
There is no reward for making a block, so the block creator takes a transaction fee.

Proof of Authority

- Proof of authority asks you to **put your identity on stake**
- Best suited for **private, permissioned Blockchain**(because of Dos attack)
- Found in 2017, by CTO Gavin Wood
- Everyone in the network knows who you are
- Anyone can be miner, **no need a specialized hardware,** algorithms are the same, no cryptocurrencies need to be put as stake(only reveal your identity)
- As long, your identity is good(not malicious), you will be given reward
- **Collaborative consensus algorithm**
- Transactions and blocks are validated by approved accounts(revealed identities)(they have authorization process which decide whether you should be given license by the network)
- In permissioned blockchain you already know the identities of the person[is not constrained to only these blockchains]
- Validator nodes run consensus software(licensed participants)
- Joining (combinations) with other consensus algorithms is allowed
- **Used for test networks** that are used to test applications before they are deployed on the public network

Conditions for Proof of Authority Consensus

The conditions may vary from system to system, the PoA consensus
algorithm is usually reliant upon:

1. Valid and trustworthy identities
2. Difficulty to become a validator (on DDoS attack, can your system handle it, does it provide same speed)
3. A standard for validator approval

- Participant will get **reward, for the period you remain in the network**(longer you are there the better) as long as no one claims the participant is malicious.

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**”
- **Validators 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.
- **less expensive than PoW**

PoA consensus and common attacks :

Distributed Denial of service attacks(DDos):

100 nodes in network 10 validator nodes

- speed of validator would reduce. when **overwhelmed with false messages from other nodes**. unnecessary traffic. **Delaying the speed of creating a block**
- An attempt to make an online service unavailable by **overwhelming it with traffic** from multiple sources.
- An attacker sends a large number of transactions and blocks to a targeted network node in an attempt to disrupt its operation and make it unavailable.
- PoA mechanism makes it possible to defend against this attack because network nodes are pre-authenticated, block generation rights can be granted only to nodes that can withstand DoS attacks.
- Initially system aim is to become a validator, however, because of Dos attack, you will need a **good system to handle it, with updated software**

- In **private blockchain, the identities of other nodes is known**, so it is easier to remove them out of the network
- In **public blockchain, address are not known**, which makes it harder to remove them

51% attack

- Obtaining control of the nodes in a permissioned blockchain network is much harder than obtaining computational power.
- With PoA, individuals earn right to become validators, so there is an incentive to retain position that they have gained.
- Validators are **incentivized with reputation** which lets them retain their authority as a node.
- PoA only allows non-consecutive block approval from any one validator, meaning that the risk of serious damage is centralized to the authority node.

Proof of Authority

Pros:

- **Energy efficient** (not many servers needed)
- **Fast** (already know who is going to be validator node(randomly chooses the participant to mine the block))
- **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. (That particular node, if you already knows how many transaction it will put in a block, we can estimate time)**For PoW and PoS consensuses, this time varies**
- High transaction rate(speed in the system)

Cons :

- PoA is **not totally decentralized** but is just an effort to make centralized systems more efficient
    - the guys with the greater setup to get licensing will have the more preference
- PoA validators are visible to anyone.(Prone to attack) Knowing validators identities could potentially lead to third-party manipulation
- Third party manipulation(you put malicious stuff ,we will give you stake)

Application of PoA consensus

PoA consensus algorithm may be applied in variety of scenarios and is deemed great option for **logistical applications such as supply chains.**

Proof of Authority model enables companies to maintain their privacy while availing benefits of blockchain technology.
Microsoft Azure is another example where PoA is being implemented. Azure platform provides

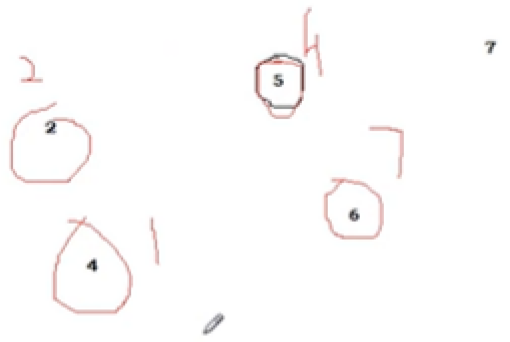
solutions for private networks, with system that does not require native currency like ether 'gas' on Ethereum, since there is no need for mining. Azure nodes are pre-selected.

Proof of elapsed Time(PoET)

- **Competitive consensus algorithm**
- Designed to improve upon proof of work consensus and provide a fresh alternative for permissioned blockchain networks
- Participant Identify themselves before join (Identity known)
- **Sawtooth lake project** on Hyperledger
- Built to run in a secure area of the central processor of Computer called a **trusted execution environment(TEE)** so that your code is not tempered. no malicious activity can happen. Entire network is secured
- Lottery based System(don't need to reveal identity in public, but need to private) if one wants to become miner
- To get license - A random node is selected to mine the block

How is it different from POA ?

- PoET the consensus **software runs on every miner node**, and some **timer is provided** on every miner node



Randomly values are added to all the miners available in the pool at time=0;
Here, node 4 finishes first.

- So then , whoever finishes the timer first, smallest value, wins and gets the chance to mine the block

Lottery based system that randomly selects a node from pool of validating nodes.

- The probability of a node being selected increases in line with how much processing resource the node has contributed to that Blockchain. (not necessary)
- Control cost of consensus process(no higher server needs)
- Internal project or all participants are known.
- Used to build decentralized applications.

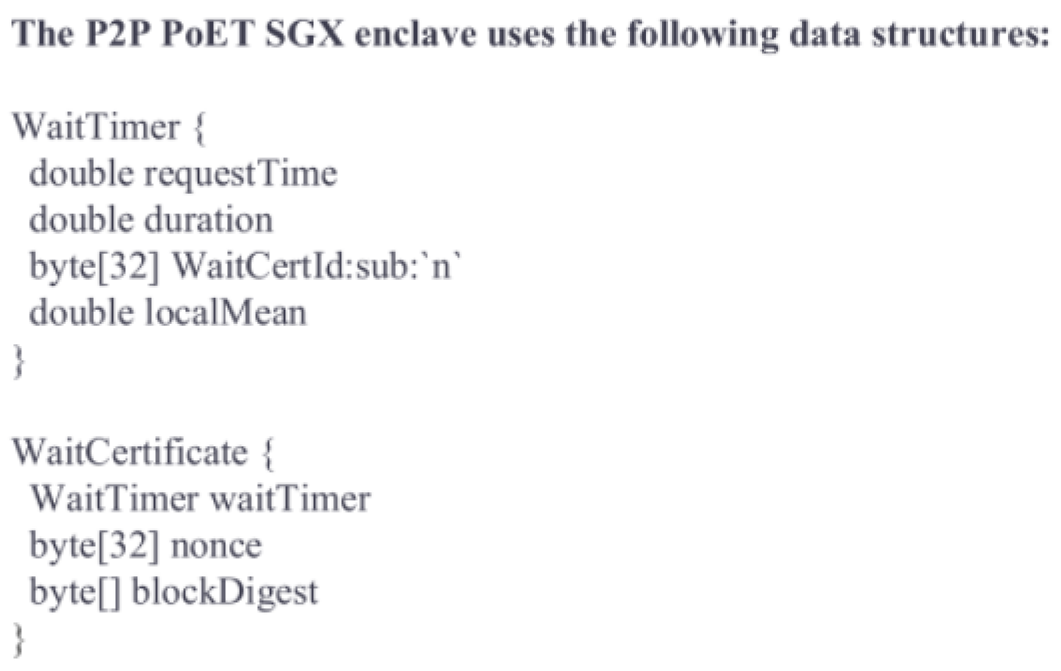
How will one verify that the propose has really waited for a random amount of time?

If a particular node is finishing their timer properly and not acting maliciously- **Software Gaurd Extension**

- The PoET algorithm is for **permissioned** blockchain networks.
- A special verification is required from a node when it tries to join the network.
- This verification is achieved using Intel's **Software Guard Extension** (SGX) technology which was first introduced in 2015. (acts as a trusted execution environment, put code here, so no one can alter it) Independent of application being worked on)
- It creates an **attestation** for a piece of code and **protects the code from external access.**

Working of PoET

- Join to a network
- A node downloads POET code for consensus mechanism
- Get attestation from SGX(became valid node)→ generates attestation(key)
- Forward this key when requesting to join the network
- The new node now has its own **timer object** which is initialized to a random value
- All nodes are initialized with a random time



The P2P PoET SGX enclave uses the following data structures:

```
WaitTimer {
  double requestTime
  double duration
  byte[32] WaitCertId:sub:`n`
  double localMean
}

WaitCertificate {
  WaitTimer waitTimer
  byte[32] nonce
  byte[] blockDigest
}
```

Proof of Elapsed Time

Pros

- Low cost of upkeep
- Scalable to operation

Cons

- Needs specialized hardware
- Must know participants in the network
- PoET is used by Hyperledger- Sawtooth lake project.
- Can't trust the intel third party, Software guard extension

Proof of Capacity / Proof of Space

- Proof-of-Capacity, aka Proof-of-Space, was first proposed in 2013 in the Proofs of Space white paper.
- Proof of Capacity (PoC) is a consensus algorithm used in Blockchains that allows miners in the **network to mine for tokens with the available space they have on their hard drive.**

PoSpace in Cloud

- PoSpace is also being used in cloud storage technologies in which the peers contribute their **free disk space** and get service proportionately.
- Peers can also get paid if they let their free space to be used by peers that need more space. Eg: **Storj is a PoSpace** based cloud storage

Collaborative Consensus

Farmers

- Nodes that are securing transaction.
- Allocate non-trivial amount of memory or disk space
- Use leftover memory
- Farmers not fighting to secure blocks.
- Green and scalable

PoC miners use a 2-step system

Plotting

Plotting consists of creating a random solution, known as a plot, through the **Shabal cryptographic algorithm** and storing it on a miner's hard drive.

- for different nonces, find hash value and save it in different hard disk spaces, before starting your puzzle

Mining

Mining consists of miners reaching the solution, and whoever reaches it first, gets to mine the next block.

Proof of Capacity

Pros

- Uses wasted hardware space.
- Environmentally friendly

Cons

- New and not tested.
- PoSpace and PoC are used by **Burstcoin, Chia and SpaceMint**

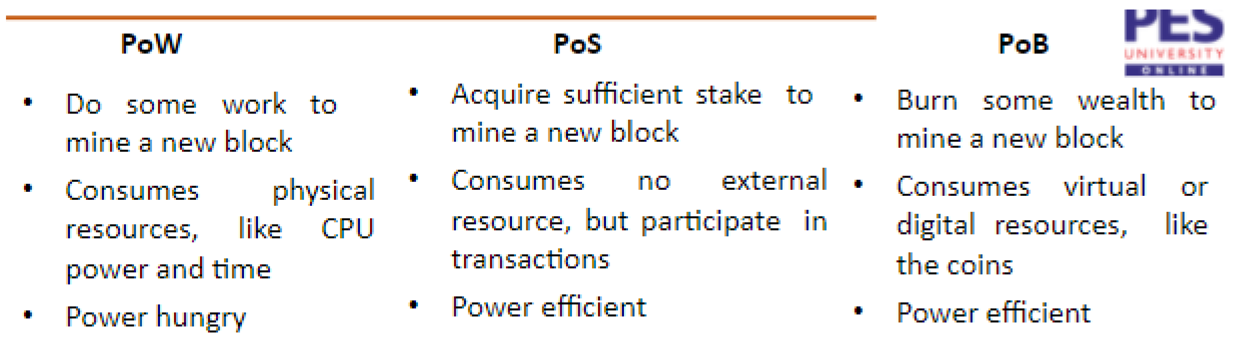
Proof of Burn

- Hypothesis: If a node is ready to burn the crypto that he has bought, it is a serious node
- **Sending crypto to an address where it can never be retrieved**
- More the crypto you burn, more chance of being a validator-(becomes centralized)
- Incentive: Eg. If you give 5 crypto- system will reward you with 20 crypto
- Non-competitive
- Validated by elected nodes.
- User pays the fee by burning some cryptocurrency
- To secure things like **land titles, birth records and data feeds on IoT devices.**

Coinburn

- Miners should show proof that they have burned some coins
- Sent them to a **verifiably un-spendable address**

Proof of Burn is used by Slimcoin, TGCoin and Factom.



PoW	PoS	PoB
• Do some work to mine a new block	• Acquire sufficient stake to mine a new block	• Burn some wealth to mine a new block
• Consumes physical resources, like CPU power and time	• Consumes no external resource, but participate in transactions	• Consumes virtual or digital resources, like the coins
• Power hungry	• Power efficient	• Power efficient

Proof of Burn

Pros

- Cheap to use

- Scalable to all types of applications

Cons

- Waste of mined crypto
- Limited functionality
- Centralized, more you burn more chance to be a miner(add new blocks)

Consensus Algorithms

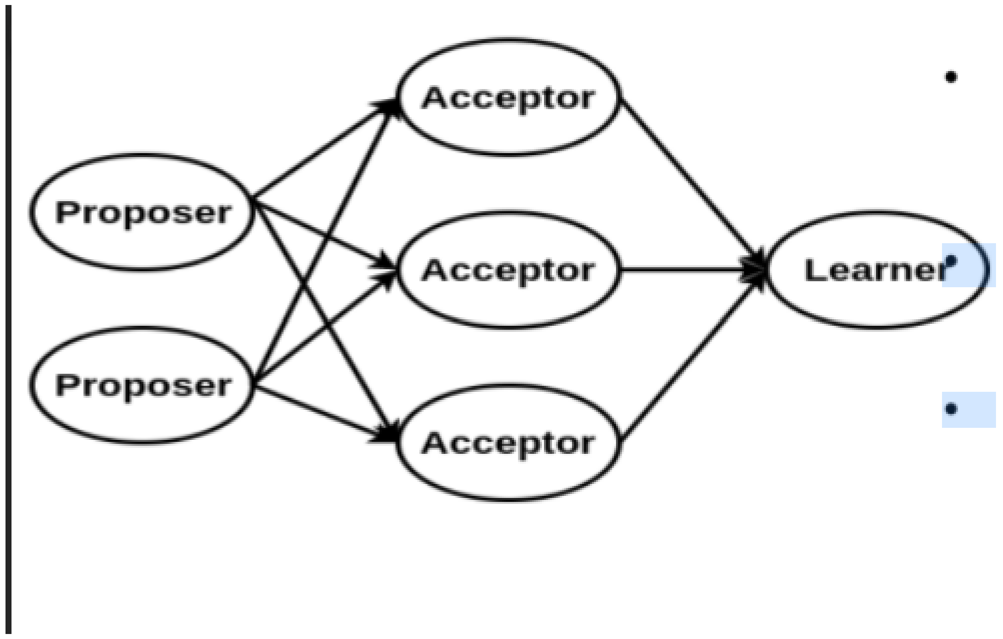
Classical Distributed Consensus Algorithms(Paxos, RAFT, Byzantine Agreement) are based on State Machine Replication

PAXOS

First Consensus Algorithm proposed by L. Lamport in 1989

Objective: choosing a single value under crash or network fault **
• System process
– Making a proposal
– Accepting a value
– Handling Failures

Proposer: propose values that should be chosen by the consensus(can have multiple proposers)
Acceptor: form the consensus and accept values
Learner: learn which value was chosen by each acceptor



Paxos is based on **state-machine replication**

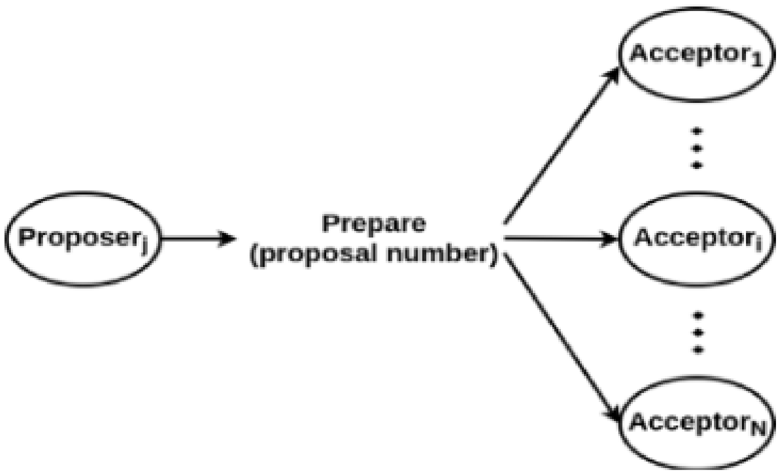
- Proposers and Acceptors maintain a state of the **running epochs**
- Uses a **variable IDp** where p is an epoch number maintains the state

A Paxos run aims at reaching a **single consensus**

- Once a consensus is reached, Paxos cannot progress to another consensus
- To reach multiple consensus, you need to run Paxos in rounds(Multi-Paxos)

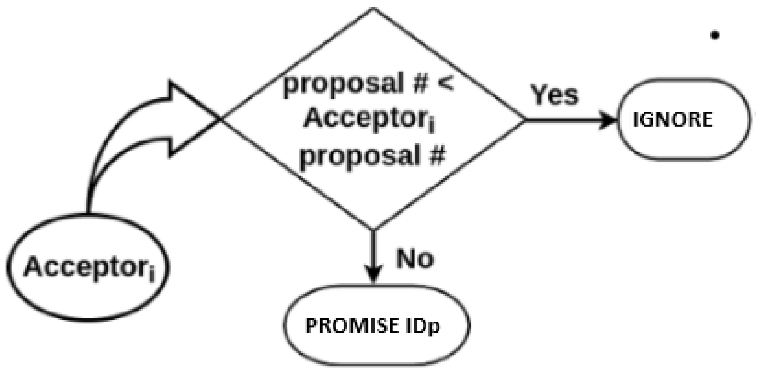
STEPS:

1. Making a Proposal: **Proposer Process**
   proposal number: form a timeline, biggest number considered up-to-date
   Proposer wants to propose its value(IDp = proposal number)
   - Sends **PREPARE IDp** to a majority of the acceptors
   - IDp must be unique across proposers for each PREPARE message e.g. HASH(timestamp+Proposer ID) to generate p



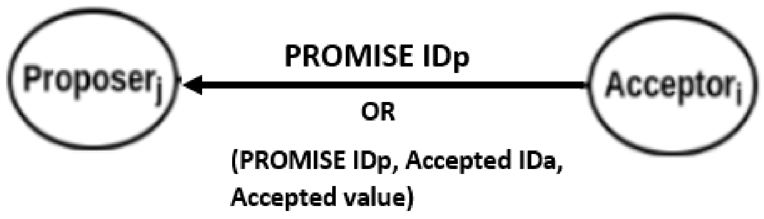
2. Making a Proposal: **Acceptor's Decision Making**
   Acceptor received a PREPARE message with IDp

- Each acceptor compares received proposal number with the current known values for all proposer's prepare message
- Did it promise to ignore requests with this IDp?
- **YES:** Ignore
- **NO:** Will promise to ignore any request lower than incoming IDp(i.e. incoming proposal value is bigger than previously promised value[Acceptor(i) proposal #)]→
  - Has it ever accepted anything?
  - **YES:** Reply with **PROMISE IDp, accepted IDa, VALUE**
  - **NO:** Reply with **PROMISE IDp**



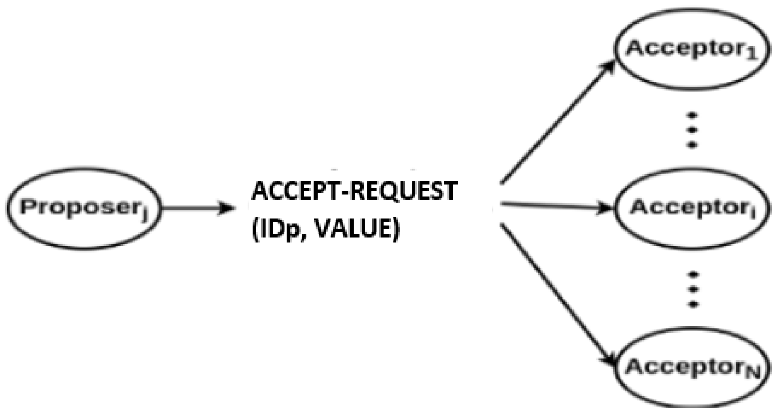
3. Making a Proposal: **Acceptor's Message**

- **IDa→ previously accepted proposal number**
- Acceptor replies with a **PROMISE IDp**(if never accepted a value)
- or **PROMISE IDp, accepted IDa, VALUE**(if accepted a value before)



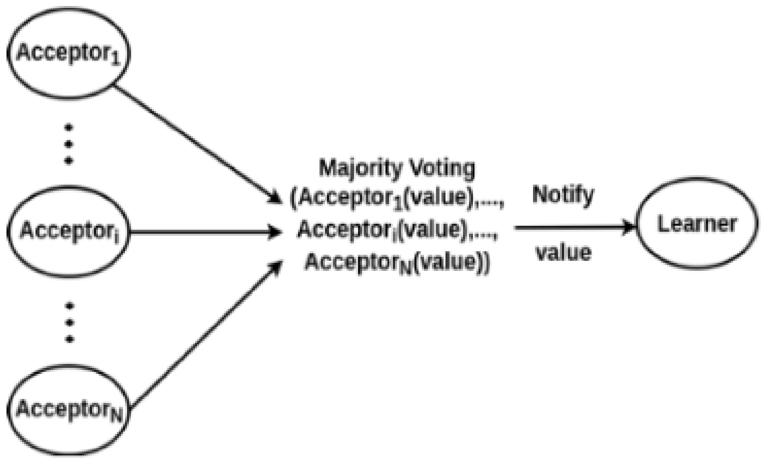
4. Accepting a Value: **Proposer's Decision Making**

- Proposer receive a response from majority of acceptors before proceeding
- Propose **gets majority of PROMISE messages** for a specific IDp
- It sends **ACCEPT-REQUEST IDp, VALUE** to a majority(or all) Acceptors
  - Has it got any already accepted value from it's promises?
  - **YES:** Picks the value with the highest IDa
  - **NO:** Picks the value of it's choice



5. Accepting a Value: **Accept Message**

- Acceptor **receives an ACCEPT-REQUEST IDp, VALUE**:
  - Did it promised to ignore request with this IDp?
  - **YES:** Ignore
  - **NO:** Reply with **ACCEPT IDp, VALUE**. Also
    - **Send it to all the learners**



NOTE:

1. If majority of acceptors **Accept IDp, VALUE→ consensus is reached**
2. Consensus is reached on the **VALUE, not on IDp**
3. PAXOS works in **two rounds**
   - Agreement on the state(ID)
   - Agreement on the value

Q) What happens if the acceptor acts as a malicious node or fails?

- If one of the acceptor nodes crashes? other acceptor can hear the proposal and vote
- If more than n/2-1 acceptors fails, then the condition will fail(system will be faulty)
  - no proposer get a reply, no values accepted

Q)Proposer fails during prepare phase?

• Acceptors wait, wait, wait, and then someone else become the proposer

Q)Proposer fails during accept phase?

- Acceptors have already agreed upon whether to choose or not to choose the proposal

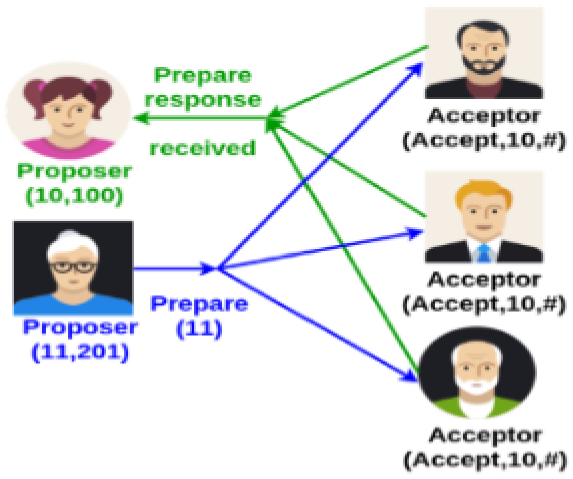
Q) Single Proposer: No Rejection

- Proposer always have proposal with biggest number
- No proposal rejected

Handling Failure: Dueling Proposer

Proposer received confirmations to her prepare message from majority – yet to send accept messages
• Another proposer sends prepare message with higher proposal number
• Block the first proposer's proposal from being accepted

Use leader election - select one of the proposer as leader
• Paxos can be used for leader election !!



Limitations of PAXOS

- Cannot handle malicious proposers
- Concept from Distributed systems, wasn't designed for blockchain
- Blockchain uses Paxos algorithm

RAFT

Designed as an **alternative to Paxos**
• A generic way to **distribute a state machine** among a set of **servers**

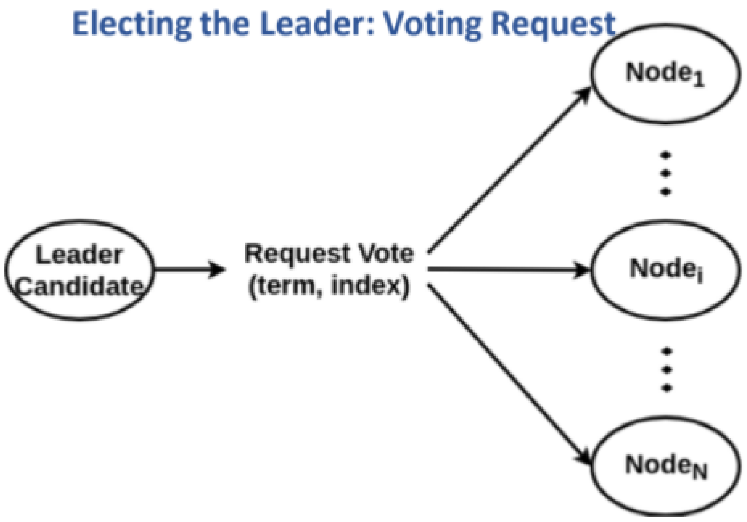
- Ensures that every server agrees upon same series of state transitions

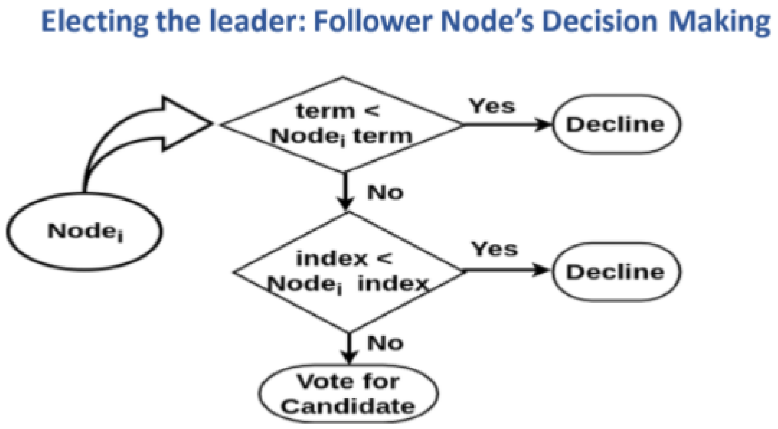
• **Basic idea:**
– The **nodes collectively selects a leader**; others become followers
– The leader is responsible for **state transition log replication** across the followers, deals with **replicas failing**

- **One leader is selected priorly**, and that decision is taken
- The leader remains the leader till the **entire system stops**, or the **leader crashes**
- **Initially** all the nodes are called **followers**
- There are **candidates, that pitch** to be a leader(majority vote from the acceptor side)

term is the round value(which round is going on) node is going to pitch for to be a leader(till which round they participated) $\longrightarrow$ **(last calculated # known to candidate) + 1**

index upto current round how many transactions have been executed
committed transaction available to the candidate

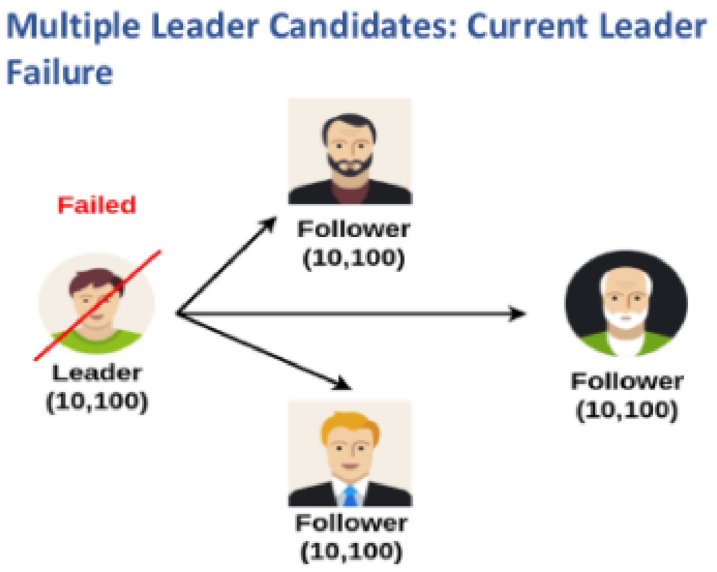

Electing the Leader: Voting Request
Leader Candidate
Request Vote (term, index)
Node1
Nodei
NodeN

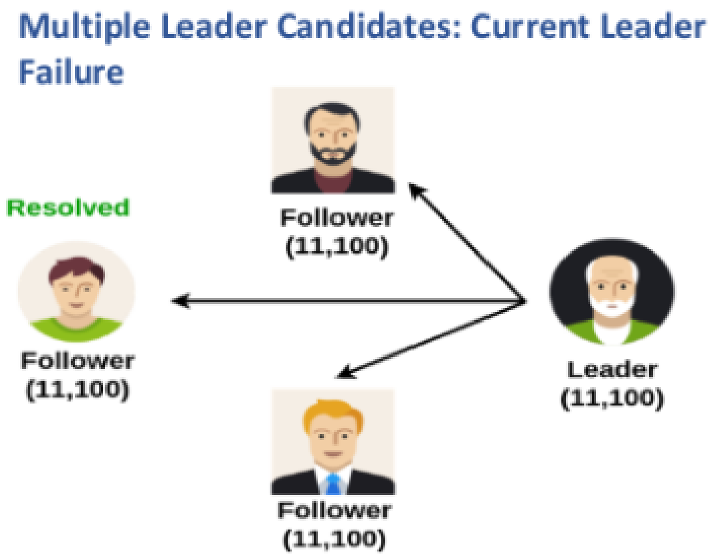


- Each node **compares received term and index with corresponding current known values**
- If term & index **value is less than candidate values → decline** else **vote for candidate**

Nodes have(term and index)
node term index
n1, 30, 15
n2, 50,20
n3 55, 30

- The algorithm for selecting a leader is different
- term and index is not the same for every round





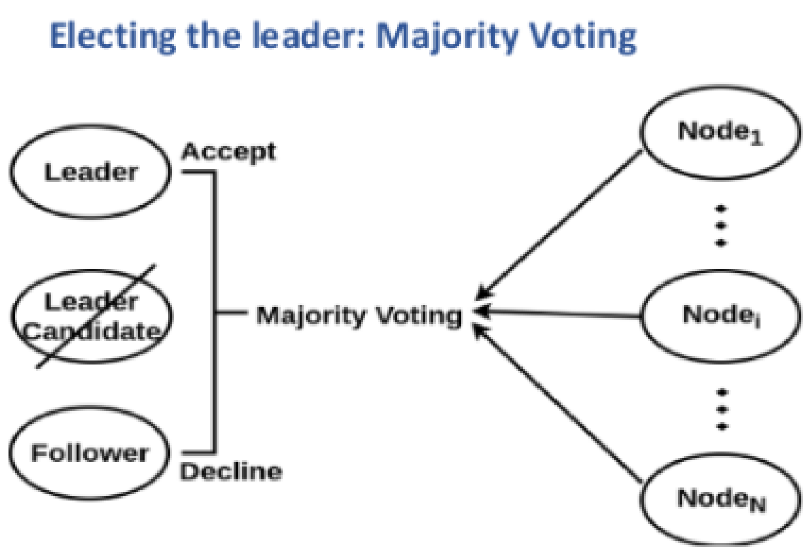
A leader with three followers

- **term: 10**
- **commit index: 100**

1. The **leader node failed**
2. **New leader elected** with term 11
3. Old leader **recovered**
4. Old leader receive **heartbeat message** from new leader with greater term
5. Old leader **drops to follower state**

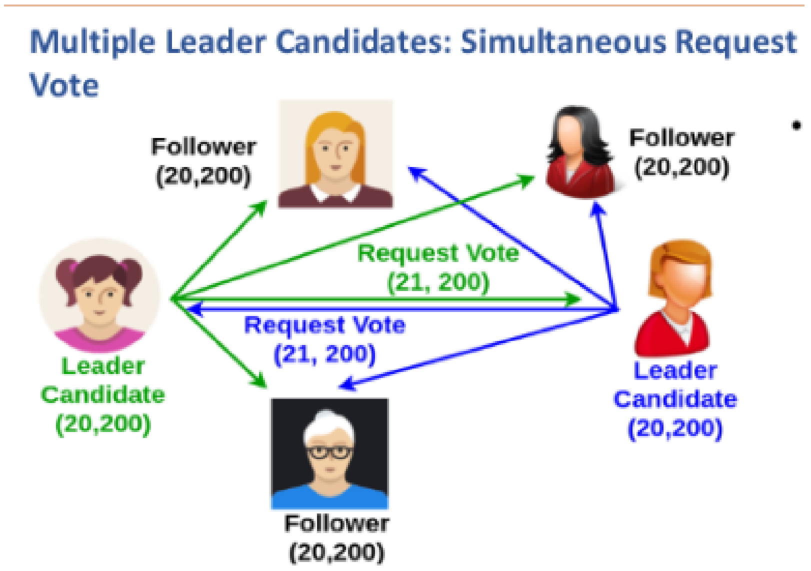
Use of Majority voting

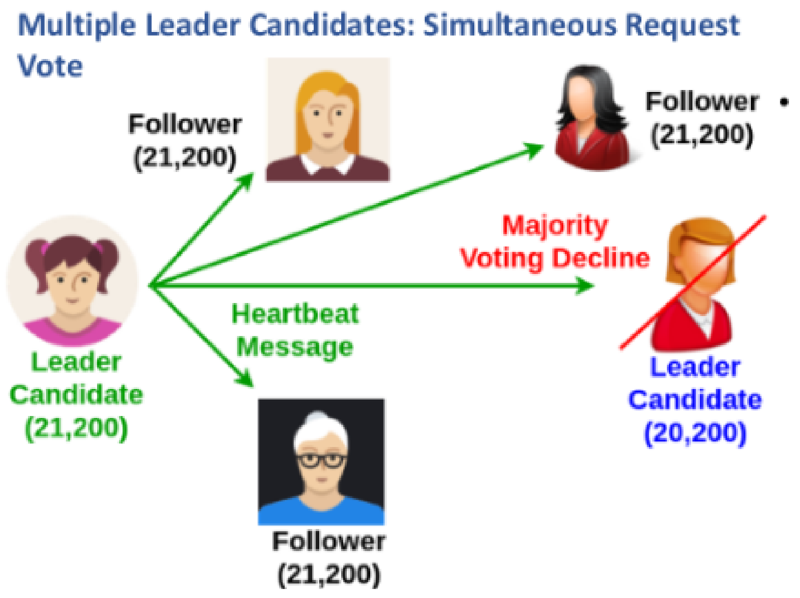
- **leader selection**
- **commit the log entry**



SCENARIO 1:

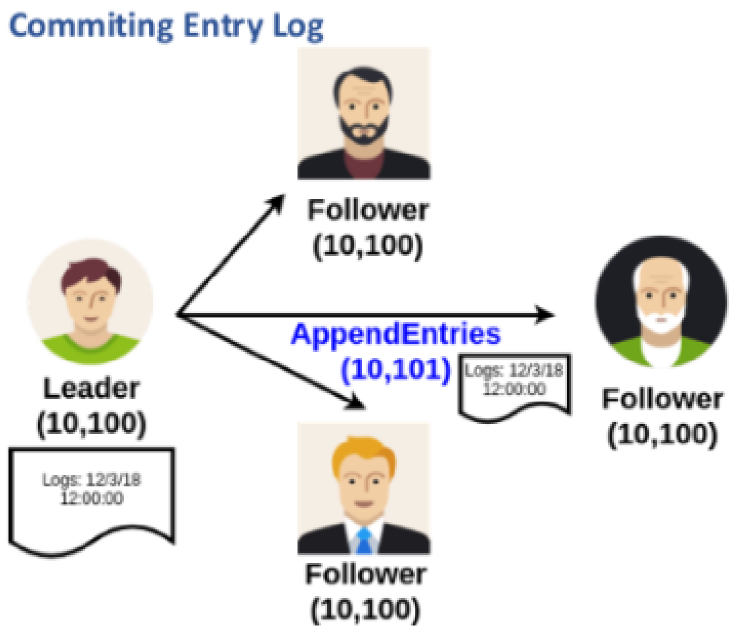
- Two nodes send **Request vote message with term 21** at the **same time**
- One of them gets **majority voting**
- **Winner sends heartbeat message**
- **Other leader candidate** switches to **follower state**

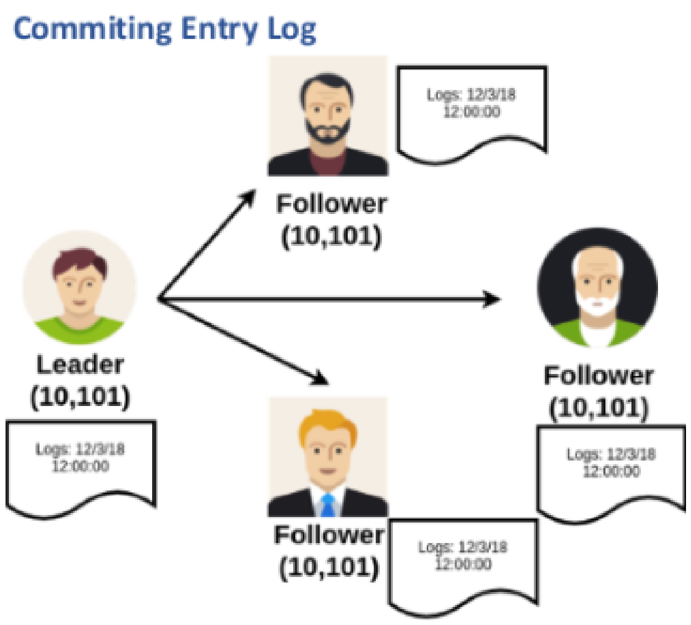

Multiple Leader Candidates: Simultaneous Request Vote
Follower (20,200)
Follower (20,200)
Request Vote (21, 200)
Request Vote (21, 200)
Leader Candidate (20,200)
Leader Candidate (20,200)
Follower (20,200)


Multiple Leader Candidates: Simultaneous Request Vote
Follower (21,200)
Follower (21,200)
Majority Voting Decline
Heartbeat Message
Leader Candidate (21,200)
Leader Candidate (20,200)
Follower (21,200)

SCENARIO 2:

1. Leader adds entry to **log with term 10 and index 101**
2. Leader **sends AppendEntries message** to followers with index 101
3. **Majority voting** decides to **accept or reject** the entry log
4. Successfully accept entry log– All leader and followers **update committed index to 101**





NOTE:

Failure of up to **N/2 - 1 nodes** does not affect the system due to majority voting

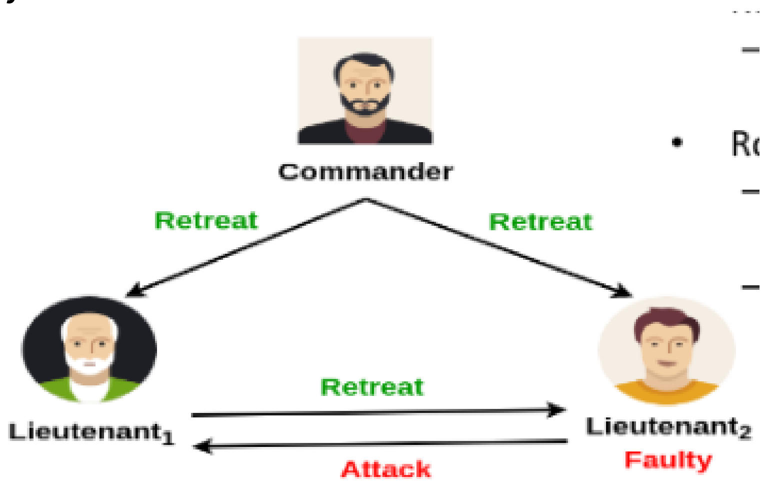
Consensus Algorithms: Byzantine Fault Tolerance System

- **Paxos and Raft** can tolerate up to **N/2− 1** number of **crash faults**
- What if the nodes behave maliciously?

Three Byzantine Generals Problem:

SCENARIO 1:

What if **Lieutenant Faulty?**

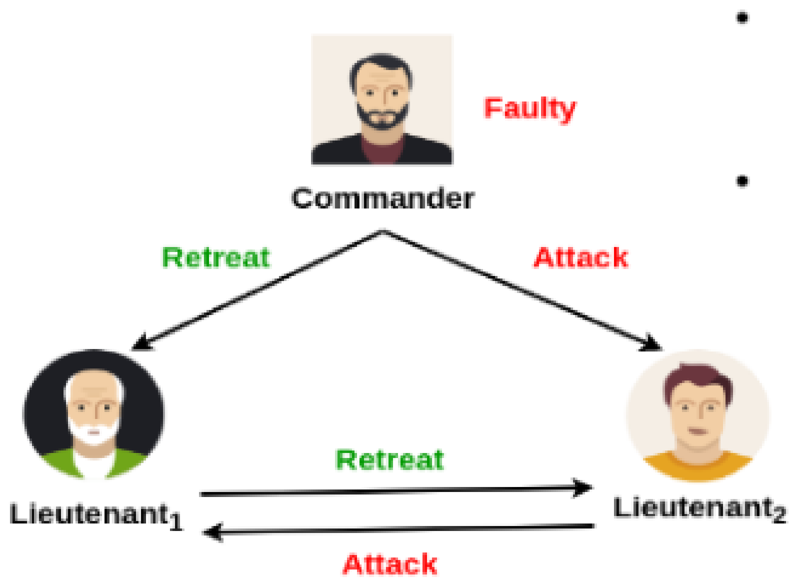


1. **Commander** correctly sends same message to **Lieutenants**
2. **Lieutenant1 correctly** echoes to **Lieutenant2**(FAULTY)
3. **Lieutenant2**(FAULTY) **incorrectly** echoes to **Lieutenant1**

- **Lieutenant1** received **differing message**
  - Finally **decides "retreat"** because **listens to commander**(integrity condition)

SCENARIO 2:

What if Commander is faulty?



1. **Commander**(FAULTY) **sends differing message** to **Lieutenants**
2. **Lieutenant1 correctly** echoes to **Lieutenant2**
3. **Lieutenant2 correctly** echoes to **Lieutenant1**

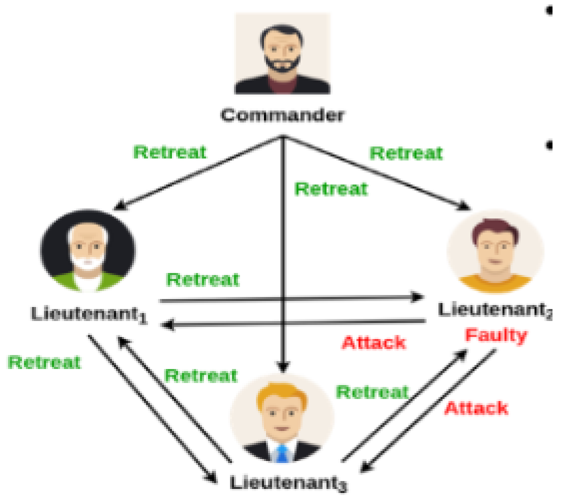
- **Lieutenant1** received **differing message**
- By integrity condition, both **Lieutenants conclude with Commander's message**
  - This **contradicts the agreement condition**
  - Therefore: **No solution** possible for **three generals including one faulty**

Four Byzantine Generals Problem:

Follows Majority Voting

SCENARIO 3:

What if **Lieutenant Faulty?**



1. **Commander** sends a message to each of the **Lieutenants**
2. **Lieutenant1** and **Lieutenant3 correctly** echo the message to others
3. **Lieutenant2**(FAULTY) **incorrectly** echoes to others

- **Lieutenant1** : majority(Retreat,Attack,Retreat)= **Retreat**
- **Lieutenant3** : majority(Retreat,Retreat,Attack)= **Retreat**
- **Lieutenant2** : majority(Retreat,Retreat,Retreat)= **Retreat**

Therefore,

Consensus can be reached.

SCENARIO 4:

What if **Commander Faulty?**

1. **Commander** sends **differing message** to Lieutenants
2. **Lieutenant1, Lieutenant2** and **Lieutenant3 correctly** echo the message to others

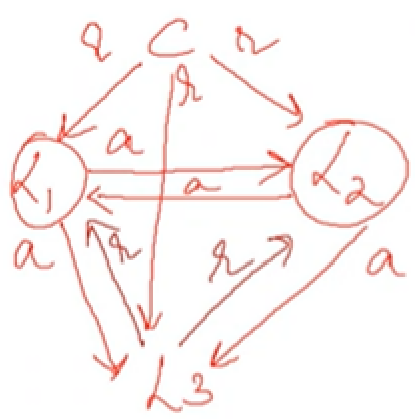
- **Lieutenant1**: majority(Retreat,Attack,Retreat)= **Retreat**
- **Lieutenant2**: majority(Attack,Retreat,Retreat)= **Retreat**
- **Lieutenant3**: majority(Retreat,Retreat,Attack)= **Retreat**

Therefore,

Consensus can be reached.

SCENARIO 5:

Lieutenant1 and Lieutenant2 are FAULTY?



$L_1: C(r); L_2(a); L_3(r)$
$\hookrightarrow r$

$L_2: C(r): L_1(a); L_3(r)$
$\hookrightarrow r$

$L_3: C(r); L_1(a); L_2(a)$
$\hookrightarrow a$

Can't reach consensus because L3 goes by, majority voting⇒ attack

Note: Being it a commander or anyone else who is a faulty node**(1 of them faulty)** you can still reach consensus in 4

- If you have **2f+1 lieutenant nodes**, it will **reach consensus**
- f=1: nodes=4(3+1 commander)

f - # of traitor nodes.
2f+1 Correct following nodes

Drawback: bft
Works only for synchronous nodes

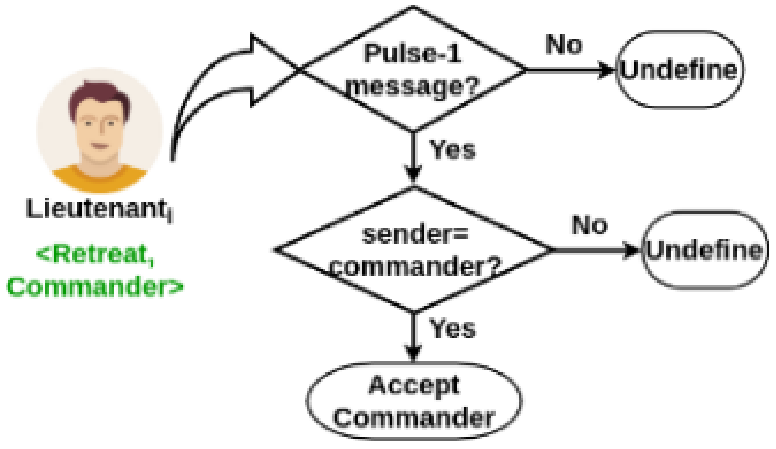
BFT:

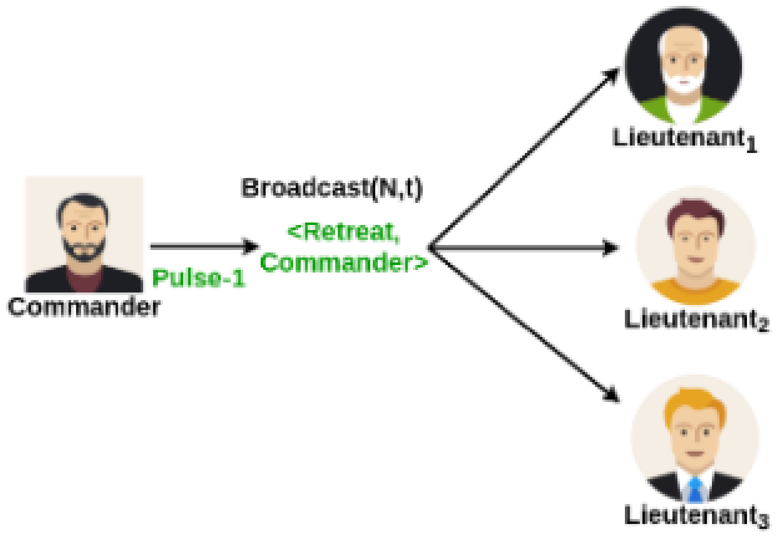
- **N number of process** with **at most f faulty**
- Receiver always knows the identity of the sender
- Fully connected
- Reliable communication medium
- Synchronous system

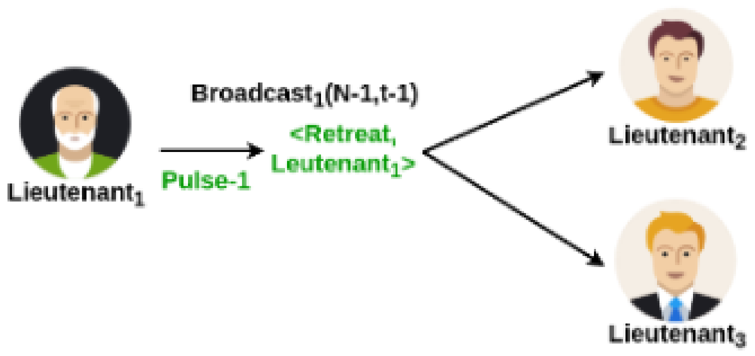
Lamport-Shostak-Pease Algorithm

Base Condition:
Broadcast(N,t=0)
N: number of processes
t: algorithm parameter

1. **Commander** decides on its **own value**
2. **Lieutenants decision** by **sender matching**
3. Only **commander sends to all lieutenants**
4. All **lieutenants broadcast** their values to the **other lieutenants except the senders**


Lieutenant$_i$
<Retreat, Commander>
Pulse-1 message?
No
Undefine
Yes
sender= commander?
No
Undefine
Yes
Accept Commander


Commander
Pulse-1
Broadcast(N,t)
<Retreat, Commander>
Lieutenant$_1$
Lieutenant$_2$
Lieutenant$_3$



Practical Byzantine Fault Tolerant

Lamport-Shostak-Pease Algorithm
Why Practical?

- **Ensures safety** over an **asynchronous network** (not liveness!)
- **Byzantine Failure**
- **Low overhead**

Real Applications

- Tendermint
- IBM's Openchain
- ErisDB
- Hyperledger

Properties:

Works on asynchronous
Problem with bft: all the nodes need to return an answer to reach consensus, more overhead

Asynchronous distributed system
- delay, out of order message
Byzantine failure handling
- deals with arbitrary node behavior
Privacy
- tamper-proof message, authentication

PBFT Model

- A **state machine** is **replicated across different nodes**

- The replicas move through a **successions of configurations**, known as **views**
- One replica in a view is **primary** and others are **backups**
- Views are changed when **primary** is detected as **faulty**
- **View is a collection of nodes**, where one is assigned as **primary node** and others are **secondary nodes**
- Every view is identified by a **unique integer** number vv
- **Only the messages from the current views are accepted**

As faulty nodes

NOTE: **3f+1 replicas**(included the commander) to **reach consensus**

3f+1 replicas

- If primary fails, anyone can be leader

Client **waits for f+1 messages** that are the same result(forget about the faulty node and reach individual consensus)
–f is the **maximum number of faulty replicas that can be tolerated**

Request: client to primary
pre-pepare: message to secondary, that client has asked to do this query
prepare: secondary sends message to primary, we are ready with the resources to execute request(acknowledge)
commit: execute
reply send to client

client check: f+1 message, and whether they are **same**

Limitations of pBFT:

Scaling: Finalizing a leader in all these nodes, taking it to a public node. One leader can't send message to all so many nodes. Prefer permissioned rather than public.

Sybil attacks:

- **One** entity(party) **controls many identities**.
- As the number of nodes in the network increase, **sybil attacks become increasingly difficult** to carry out.
- pBFT mechanism is used in combination with other mechanism(s).
  A Sybil Attack refers to **when a system is violated by an entity that controls two or more different identities in a network**.

Forks

In blockchain, a fork is defined variously as:

- "what happens when a **blockchain diverges into two potential paths forward**"
- "a **change in protocol**", or

- “a situation that "occurs when **two or more blocks have the same block height**"

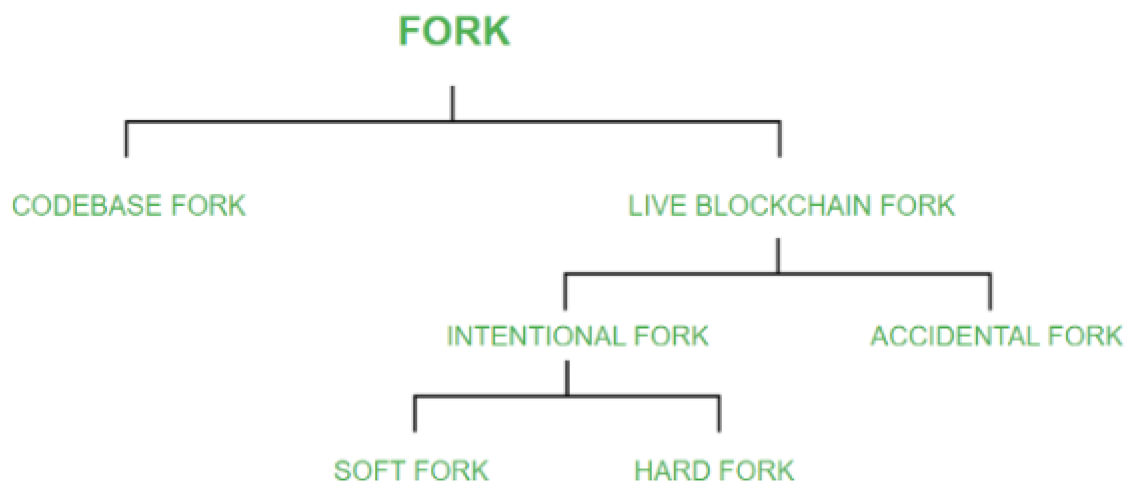
Reasons for the occurrence of a blockchain fork:

- **Add new functionality:** The Blockchain code is **upgraded regularly**. The improvements, issues are created, resolved and new versions are released when the time is suitable.

- **Fix security issues:** versions are bumped and updates are released to fix the security issues that arise in the way.

- **Reverse transactions:** The community can actually void all the transaction of a specific period if they are found to be breached and malicious.

Forks in blockchain



CODEBASE FORK:

- In codebase blockchain fork, you can **copy the entire code of a particular software**.

Let us take BITCOIN as an example, so suppose you copied the whole blockchain code and modified it according to your need:
Eg:

- decreased the **block creation time**
- made some crucial changes and created a f**aster software than BITCOIN**
- publish / launch it has a new whole software named against you, by completing the whole **white paper work process.**

So in these way a new BLOCKCHAIN will be created from an **empty blank ledger.**
There are many coins have made little up and down changes in the code of BITCOIN and created their **whole new ALT COIN**.

LIVE BLOCKCHAIN FORK:

- Live Blockchain fork means a running blockchain is been **divided further into two parts or two ways.**
- Specific page the software is same and from that specific point the chain is divided into two parts.

ACCIDENTAL FORK / TEMPORARY FORK/ SOFT FORKS:

- When multiple **miners mine a new block at nearly the same time**, the entire network may not agree on the choice of the new block.

- Such a situation arises because it takes some **finite time for the information to propagate** in the entire blockchain network and hence **conflicted opinions** can exist regarding the **chronological order of events.**

- **Temporary forks resolve themselves eventually** when one of the **chain dies out** (gets orphaned) because **majority of the full nodes choose the other chain** to add new blocks to and sync with.

INTENTIONAL FORK:

- In intentional fork the **rules of the blockchain are been changed**, knowing the code of the software and by modifying it intentionally.

This gives rise to **two types of forks** which can occur based on the **backwards-compatibility** of the blockchain protocol and the **time instant** at which a new block is mined.

1. **Soft Forks**
2. **Hard Forks**

SOFT FORK:

- When the blockchain protocol is altered in a **backwards-compatible** way.
- In soft fork you tend to **add new rules** such that they **do not clash with the old rules.**
- **Rules** in soft fork are **tightened.**
- **New blocks mined on the basis of new rules** (in the Blockchain protocol) are also considered **valid** by the **old version of the software**. This feature is also called as backwards-compatibility.

HARD FORK:

- When the blockchain protocol is altered in a **non backwards-compatible way**.
- Here the **rules are loosened.**
- When there is a change in the software is such that the new blocks mined on the basis of new rules (in the Blockchain protocol) are **not considered valid by the old version of the software.**
- When hard forks occur, new currency come into existence
  - original : **Ethereum**, new : **Ethereum Classic**
  - original :**Bitcoin**, new : **Bitcoin cash**
- **Equivalent quantity of currency is distributed** to the **full nodes** who choose to upgrade their software so that **no material loss occurs.**
- Such hard forks are often **contentious** (generating conflicts in the community).
- If the full node chooses to join the new chain, the software has to be upgraded to make newer transactions valid while the nodes who do not choose to upgrade their software continue working the same.

Example (HARD FORK): The new **Casper update in the Ethereum** → Proof of Work (PoS) to a type of Proof of Stake (PoS). **Full nodes that do not choose to install the Casper update** will become **incompatible with the full nodes** that do.