Consensus and Blockchain

Foundation of Cryptocurrency

Dr. Laltu Sardar

CREST CRYPTO SUMMER SCHOOL (CCSS) 2025
Institute for Advancing Intelligence (IAI),
TCG Centres for Research and Education in Science and Technology (TCG Crest)

June 26, 2025

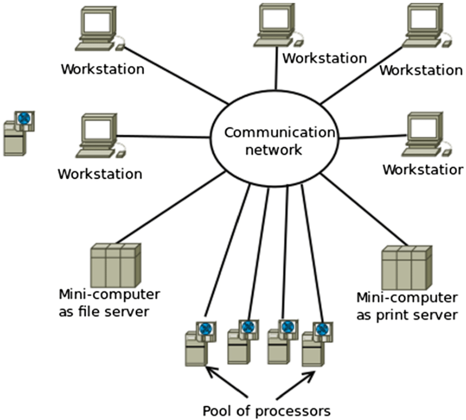
Distributed Systems

Distributed Systems

A collection of independent computers that appears to its users as a single coherent system.

Key Features:

- **Scalability** – Grow beyond one machine
- **Fault Tolerance** – Survive server failures
- **Geographic Reach** – Fast access worldwide
- **Cost Efficiency** – Use cheaper hardware
- **Resource Sharing** – Share devices and data like printers



a distributed system

History of Distributed Systems

- **1960s – Time-Sharing Systems:** Early form of resource sharing, enabling multiple users to interact with a single mainframe.
- **1970s – Networking and RPC:** Development of ARPANET and early protocols. Concept of Remote Procedure Call (RPC) introduced.
- **1980s – Workstations and LANs:** Emergence of Local Area Networks (LANs); client-server models gain popularity.
- **1990s – Internet and Web:** Rise of the World Wide Web;
- **2000s – Clusters and Grids:** High-performance computing with commodity hardware; birth of grid computing and early cloud computing.
- **2010s – Cloud and Big Data:** Rapid growth of cloud platforms (AWS, Azure); distributed storage (HDFS), computing frameworks (MapReduce).
- **2020s – Blockchain, and AI:** blockchain-based systems, and AI-driven distributed frameworks.

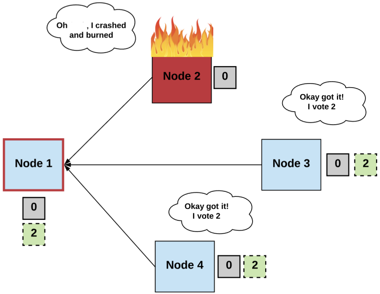
Problem in Distributed Systems

- Distributed systems consist of **multiple independent nodes**.
- **Failures**, network delays, or malicious behavior can lead to: **Conflicting data**, Inconsistent system state, System-wide failure

Challenge: How do we maintain consistency across autonomous, potentially faulty nodes?

Answer: Byzantine Fault Tolerance (BFT): handling arbitrary faults

$\Rightarrow$ **These concepts laid the foundation for solving agreement in unreliable environments.**

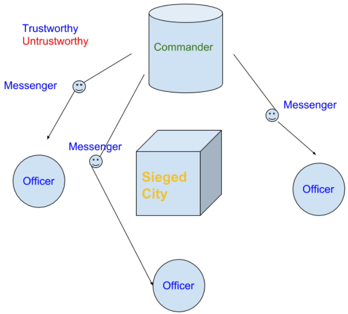


Distributed Systems Faulty Node

The Byzantine Generals Problem (BGP)

Lamport, Shostak, Pease [1978]

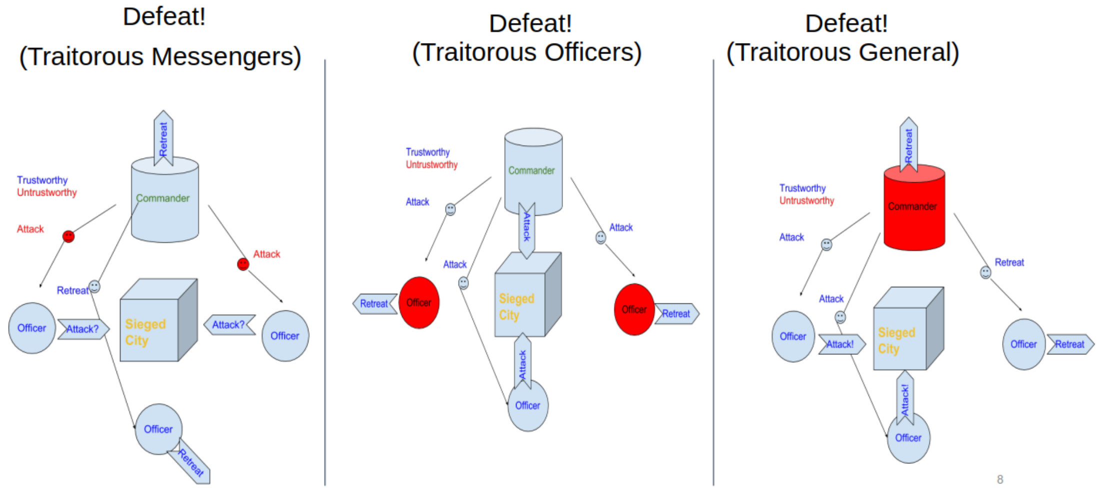
- Multiple generals (nodes) of Byzantine army surround a strong city (network)
- They must all agree to attack or retreat — partial agreement leads to defeat.
- Some generals may be traitors (Byzantine faults) who send conflicting or false messages.
- The challenge: how can loyal generals reach agreement despite these malicious actors?



GOAL – Achieve consensus:

- All loyal generals agree on the same decision.
- If the commanding general is loyal, then all loyal generals will obey the commanding general's order.

BGP Outcomes

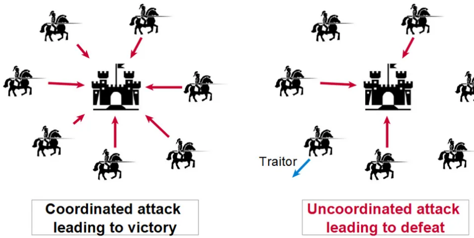


Attack or Not? Battle of Plassey (Palasi)

Generals of Siraj-ud-Daulah, Nawab of Bengal

Generals in the Battle:

- **Mir Jafar** – Commander-in-Chief *(betrayed)*
- **Rai Durlabh** – Senior General *(betrayed)*
- **Yar Lutuf Khan** – General *(betrayed)*
- **Omiruddin Khan** – Artillery Commander *(loyal but overwhelmed)*
- **Mohan Lal** – Loyal Officer and personal supporter of Siraj



British India company – Achieved Consensus?

Army of Nawab– Achieved Consensus?

Byzantine Fault Tolerance (BFT)

Definition:

- A system is considered **Byzantine Fault Tolerant (BFT)** if it can resist the challenges described by the Byzantine Generals Problem.
- A **Byzantine fault** refers to a **failure mode** where components either fail silently or behave arbitrarily/maliciously — making consensus extremely difficult.

Why It's Challenging:

- This is considered the most difficult form of fault tolerance.
- There are no constraints on how a faulty or malicious node might behave — it could lie, send inconsistent messages, or collude.

Example:

Consensus becomes significantly easier if nodes are either always truthful or always faulty in predictable ways

- BFT assumes no such behavior model.

Byzantine Broadcast Protocol

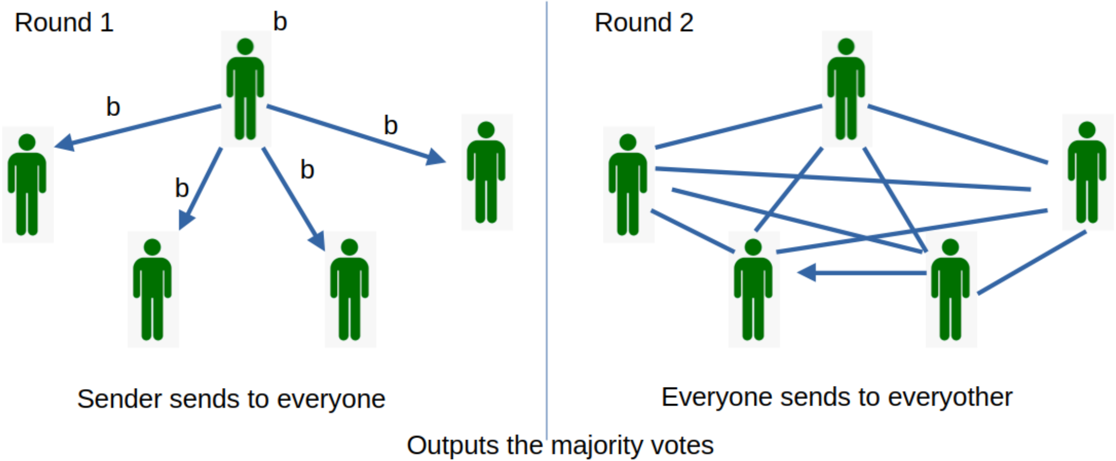
- Byzantine Broadcast is a fundamental problem in distributed systems where a designated sender (the leader) wants to send a value $b \in \{0, 1\}$ to all other nodes, and all honest (non-faulty) nodes must agree on the same value, even if some nodes are malicious (Byzantine faults). The protocol must satisfy:

Protocol Requirements:

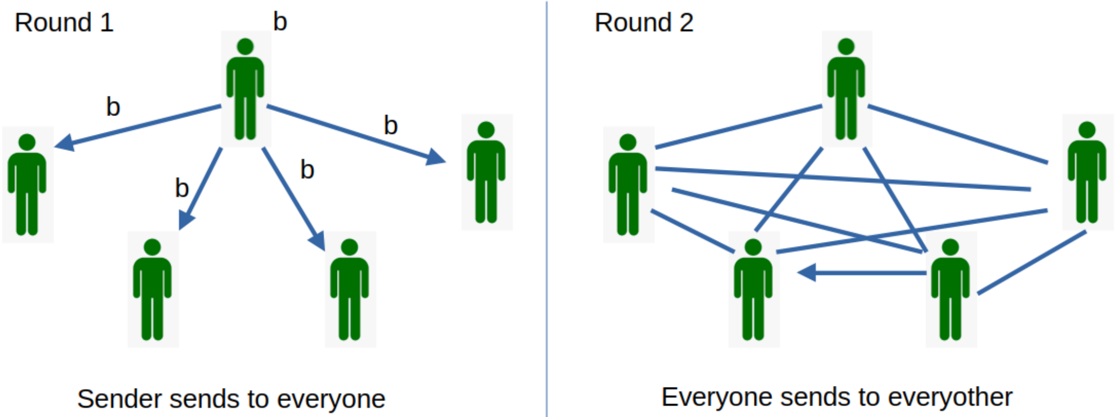
- **Validity** – If the sender is honest, all honest nodes agree on the sender's value.
- **Consistency** – All honest nodes agree on the same value, regardless of sender behavior.
- **Termination** – All honest nodes eventually decide on a value.

Corrupt nodes do not follow the protocol, form a coalition, hence can be treated as a single entity.

Naive Majority Voting Protocol



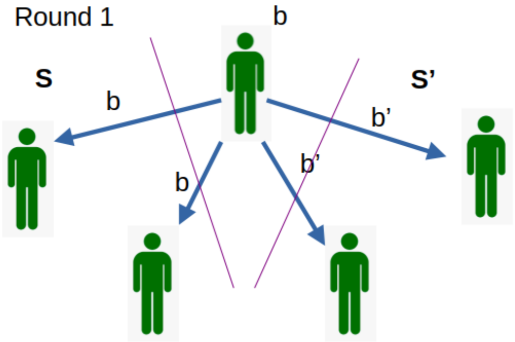
Naive Majority Voting Protocol

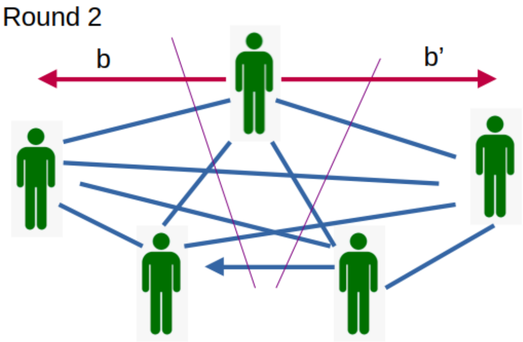


Outputs the majority votes

Is it a Byzantine Broadcast protocol? No

Inconsistency Under a Single Corrupt Sender

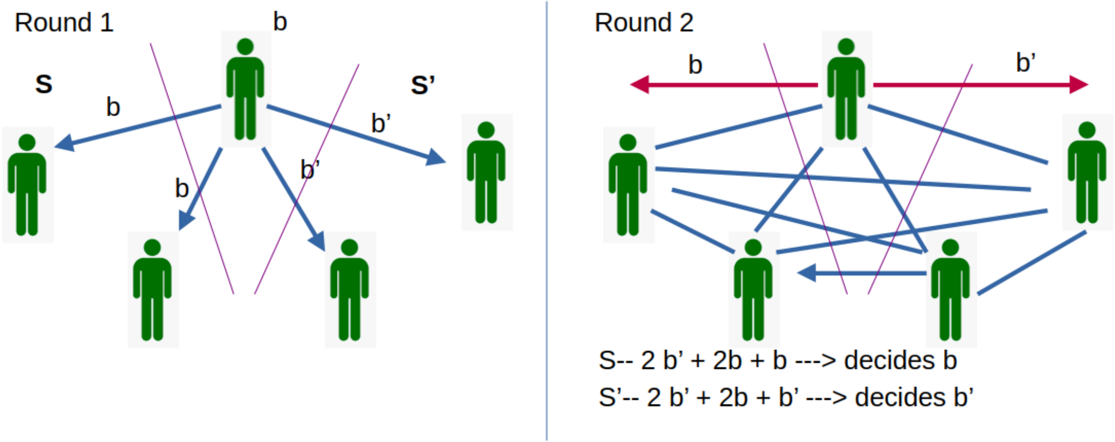




S-- 2 b' + 2b + b ---> decides b

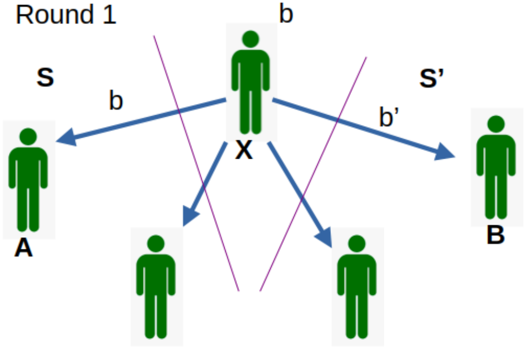
S'-- 2 b' + 2b + b' ---> decides b'

Inconsistency Under a Single Corrupt Sender

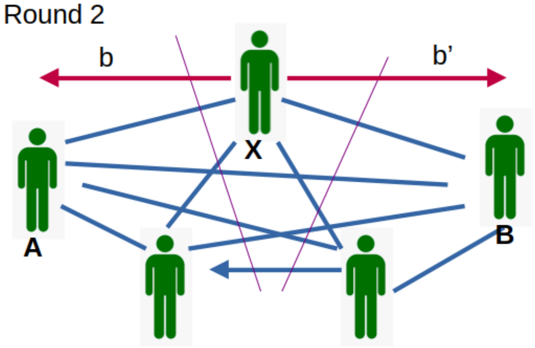


Why failed?

Reason and Fix



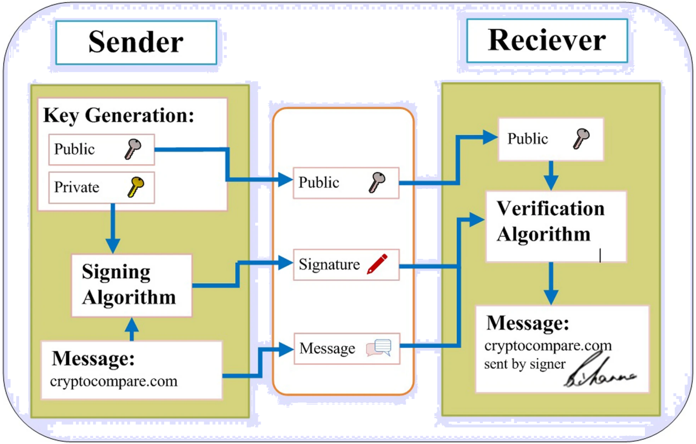
A gets b from X, B gets b' from X



A gets b' from B, A gets b from A

A gets directly b and via b'
X can deny that it sent different
Solution: Digital Signature

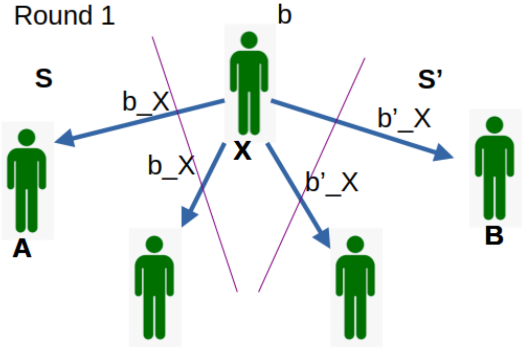
Digital Signatures Scheme



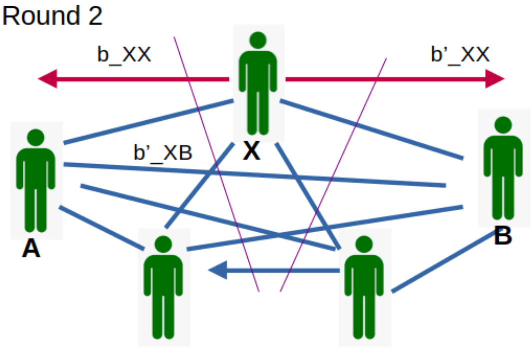
Guarantees:

- **Authenticity:** Confirms that the message was created by the claimed sender.
- **Integrity:** Ensures the message has not been altered since it was signed.
- **Non-repudiation:** Prevents the sender from denying that they signed the message.

Fix



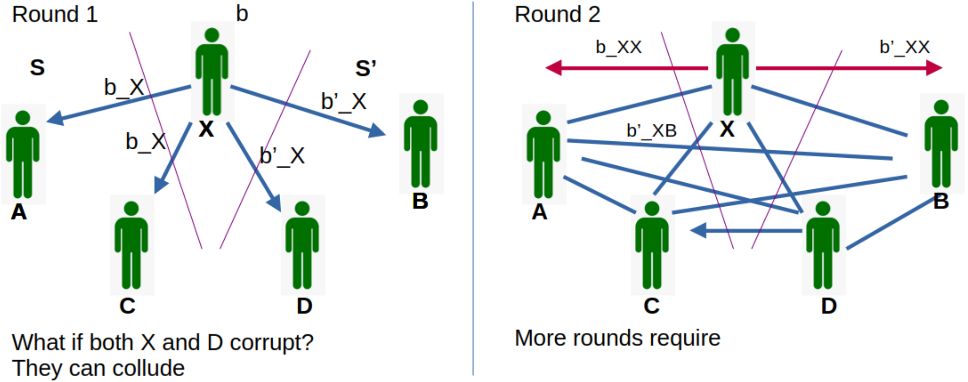
Every received signed message
is re-signed and forward



A gets b_XX and b_XB

A can rejects vote of X

Does it solve?



If a node receives m with r valid sign from distinct nodes, and has not forwarded it before:

- It appends its own signature.
- It forwards the message (now with $r + 1$ signatures) to all nodes.

The Dolev-Strong Protocol: Assumptions and Model

Model:

- **Synchronous Network** with **Authenticated Messages**
- Tolerates up to f Byzantine nodes among n total nodes

Assumptions:

- Synchronous network: messages are delivered in known bounded time.
- Digital signatures: each node can sign messages, and signatures cannot be forged.
- Protocol runs for $\mathrm{f}+1$ rounds.

Properties Ensured:

- **Validity:** If the sender is honest, all honest nodes deliver m.
- **Consistency:** All honest nodes deliver the same value.
- **Termination:** All honest nodes eventually make a decision.

The Dolev-Strong Protocol: Steps

Round 0: Sender Broadcasts

- The designated sender signs message m and sends it to all nodes.

Rounds 1 to f: Propagation

- For each round $r = 1$ to f:
  - If a node receives a message m with r valid signatures from distinct nodes, and has not forwarded it before:
    - It appends its own signature.
    - It forwards the message (now with $r + 1$ signatures) to all nodes.

Round $f + 1$: Final Decision

- Each node checks if it received message m with $f + 1$ distinct valid signatures.
- If yes, output m; otherwise, output a default value (e.g., $\perp$ or null).

Does it work? How?
Why $f + 1$ rounds?

Reality

In practical

1. Digital Signature is costly
2. It needs Public key infrastructure for keys

Can we achieve Byzantine Broadcast without digital signatures and without a PKI?

Yes! But has restrictions

Restrictions

Established:

1. If $f \geq \frac{n}{3}$, BB is **impossible**
2. If $f < \frac{n}{3}$, BB is possible

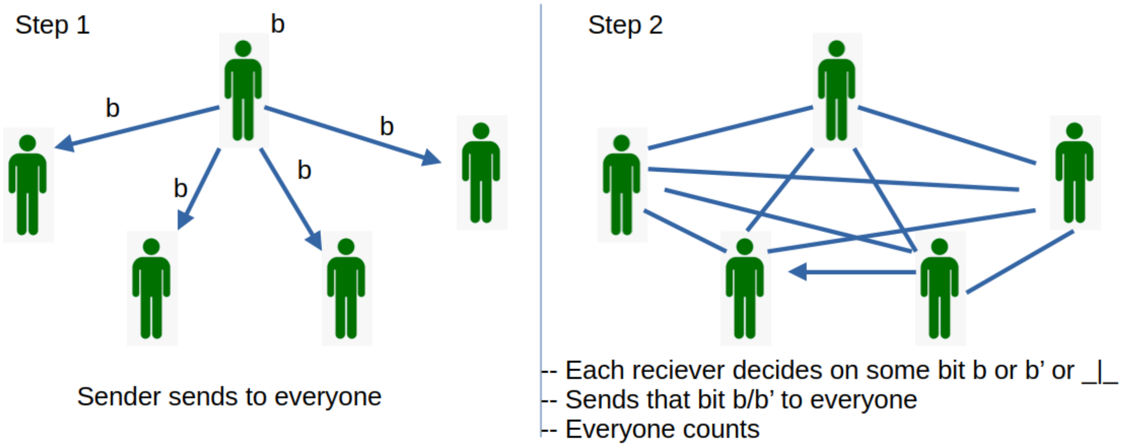
Intuition:

In the absence of cryptographic tools or trusted setup:

- Byzantine nodes can behave arbitrarily – sending different messages to different nodes, impersonating others, and splitting the view of the network.
- When $f \geq \frac{n}{3}$, there may not be enough honest nodes to reach agreement or detect inconsistencies introduced by faulty behavior.
- No deterministic protocol can guarantee agreement without some mechanism to authenticate or verify the origin of messages.

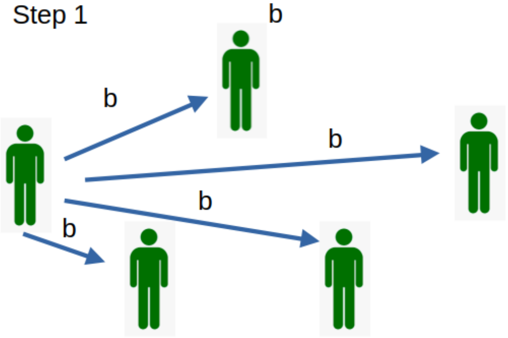
Byzantine Broadcast without Digital Signatures

Byzantine Broadcast without Digital Signatures



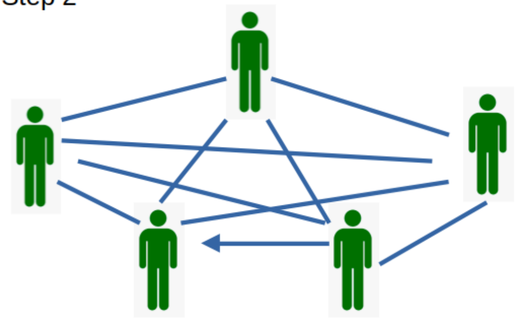
Byzantine Broadcast without Digital Signatures

Step 1



-- Change the Sender
-- Sender sends to everyone

Step 2



-- Each reciever decides on some bit b or b' or _|_
-- Sends that bit b/b' to everyone
-- Everyone counts

-- Repeat **k**- times
-- Finally each outputs its bit

BB without DS

- **Challenge:** What should be the value of k?
- **Selection Strategy:** How are the next senders selected?
- **Randomized Selection:**
  - A hash function $H(i) \to \{1, 2, \ldots, n\}$ is used.
  - H ensures that senders or leaders are selected uniformly at random.

The Protocol

- Initially:
  - The designated sender's sticky bit is its input bit.
  - All other nodes initialize their sticky bit to $\perp$.
- For each iteration $r = 1, 2, \ldots, k$:
  - **Round 0:** Leader L_r sends a proposed bit b to all nodes.
    - If L_r's sticky bit $\neq \perp$, choose b as its sticky bit.
    - Else, choose $b \in \{0, 1\}$ uniformly at random.
  - **Round 1:** Each node votes on a bit and sends it to all others.
    - If the node has a non-$\perp$ sticky bit, it votes using that.
    - Else, it votes using L_r's proposed bit.
    - If L_r proposed both bits or none, pick a bit arbitrarily.
  - **Round 2:** Nodes tally received votes.
    - If at least $2n/3$ votes are for the same bit b_0, update sticky bit to b_0.
    - Otherwise, set sticky bit to $\perp$.
- **Output:** Each node outputs its current sticky bit.

Byzantine Broadcast Without Digital Signatures

In this setting, we assume:

- No Public-Key Infrastructure (PKI)
- No digital signatures or cryptographic setup
- Synchronous communication model

Upper Bound Result:
When $f \geq n/3$, no protocol can guarantee the correctness properties of BB in the absence of signatures. This makes the bound of $n > 3f$ both necessary and tight.

Blockchain and State Machine Replication

Single-shot vs Repeated Consensus

- So far, we have considered **single-shot consensus**.
- In practice, many systems require consensus to be reached **repeatedly over time**.
- **Example:** Cryptocurrency systems like Bitcoin and Ethereum.
  - Maintain an ever-growing public ledger.
  - Record a sequence of all transactions.

Blockchain as Repeated Consensus

- We define a repeated consensus abstraction: the **blockchain**.
- Traditionally known as **state machine replication**.
- Deployed long before Bitcoin:
  - Used by companies like Google and Facebook.
- The term "blockchain" became popular with Bitcoin.

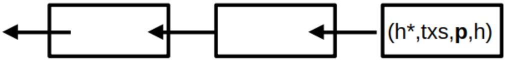
Blockchain as Repeated Consensus

- Goal: Nodes agree on a linearly-ordered, ever-growing log of transactions.
- Agreement is not per transaction, but per **batch** of transactions.
- A **block** is a batch of transactions (with metadata) *txs*.
- A **blockchain** is a chain of such blocks.

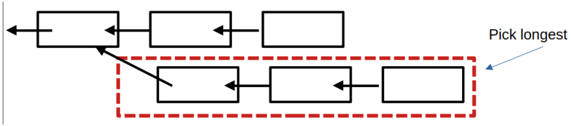
- $h*$ -> hash of previous block
- *txs* -> set of transactions
- h –> hash present block
- p –> puzzle solution

- $h*$ -> Already there
- *txs* -> collects from a pool of transactions.
- h –> computes
- p –> puzzle solution to compute

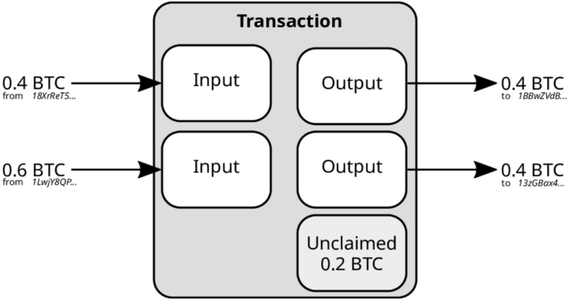
Cont.



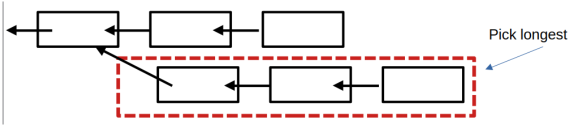
- Who compute the puzzle?
- Everyone-> who compute first will be considered as **leader**.
- Leader is going to propose it block.
- Follow Byzantine broadcast protocol for agreement.
- What if two computes at same time?
- What if it is delayed?
- Longest chain will be considered.



Transaction



Handling Double-Spending



Wait for a few blocks (6).

- If multiple transactions spend the same coin:
  - Application may accept only the **first** transaction in the log.
- For safety, merchants should:
  - Wait until $\texttt{tx}^*$ appears in the finalized log.
  - Ensure **no prior transaction** in the log spends the same coin.

Problems in Proof of Work (PoW)

Puzzle: compute nonce x such that $H(header||x) < 10^y$.

- **High Energy Consumption:** Miners perform costly computations (e.g., SHA-256) to find valid blocks.
- **Centralization Risk:** Mining power tends to concentrate in regions with cheap electricity or in large pools.
- **Latency and Throughput Limits:** Block times and propagation delays restrict scalability.
- **51% Attacks:** An attacker controlling majority hash power can censor or double-spend.
- **Hardware Waste:** ASICs are expensive and not reusable outside mining.

tcg crest
Inventing Harmonious Future

How Proof of Stake (PoS) Addresses PoW Issues

Consider some leader propose a block. How to decide who will propose.

- Bitcoin: PoW
- Ethereum 1.0 : PoW
- Ethereum 2.0 : PoS
- PoS: Probability of proposing block equivalent to how rich the node is.
  - Misbehavior leads to slashing of stake.
  - Cost of attack proportional to value at risk.

How is it efficient

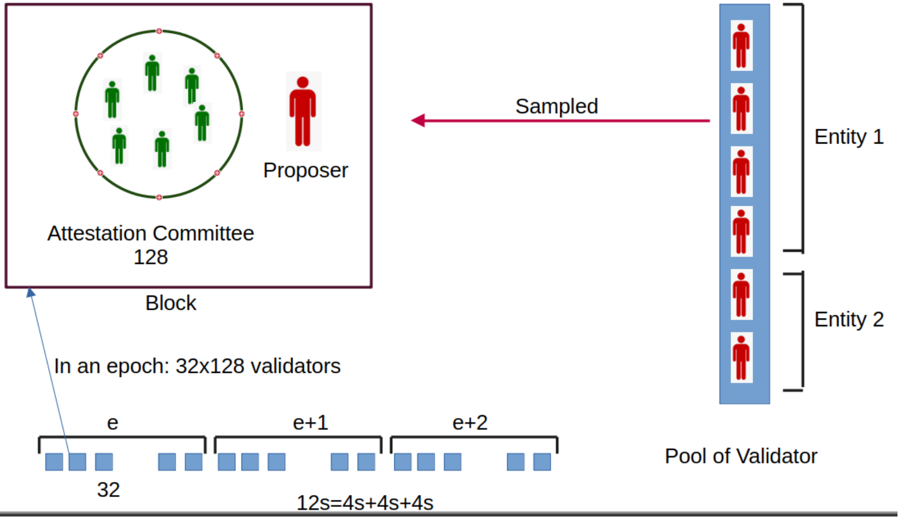
Leader selection

Deterministic though random–> eliminates hash computation

Committee

small size, Deterministic though random–> eliminates communication cost.

How does it works



Smart Contracts

Replace Transaction with a program (function)– Contract between parties

- **Definition:** Self-executing programs stored on the blockchain that run when predefined conditions are met.
- **Trustless Execution:**
  - No intermediaries needed.
  - Outcome is enforced by the code.
- **Deterministic:** Same inputs always produce the same outputs on every node.
- **Immutable and Transparent:** Code is visible and cannot be changed once deployed.
- **Applications:**
  - Decentralized Finance (DeFi)
  - Token issuance (NFTs –: Non-Fungible Tokens)
  - Decentralized Autonomous Organizations (DAOs) and governance

Types of Leader Selection Mechanisms in Blockchain I

- **Proof of Capacity (PoC)**:
  - Use hard drive space as mining resource.
  - *Used by:* Chia (XCH), Burstcoin (BURST).
- **Proof of Authority (PoA)**:
  - Pre-approved trusted validators.
  - *Used by:* VeChain (VET), Ethereum Kovan/Testnets.
- **Proof of Activity (PoA)**:
  - Hybrid of PoW (block creation) and PoS (validation).
  - *Used by:* Decred (DCR).
- **Proof of Elapsed Time (PoET)**:
  - Random wait time via trusted execution environments (TEE).
  - *Used by:* Hyperledger Sawtooth.

THANK YOU

FOR YOUR ATTENTION

tcg crest