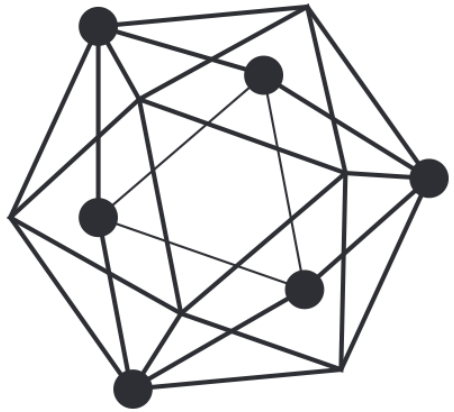
Tendermint

A leading BFT engine for building blockchains



Dr. Raju Halder
IIT Patna

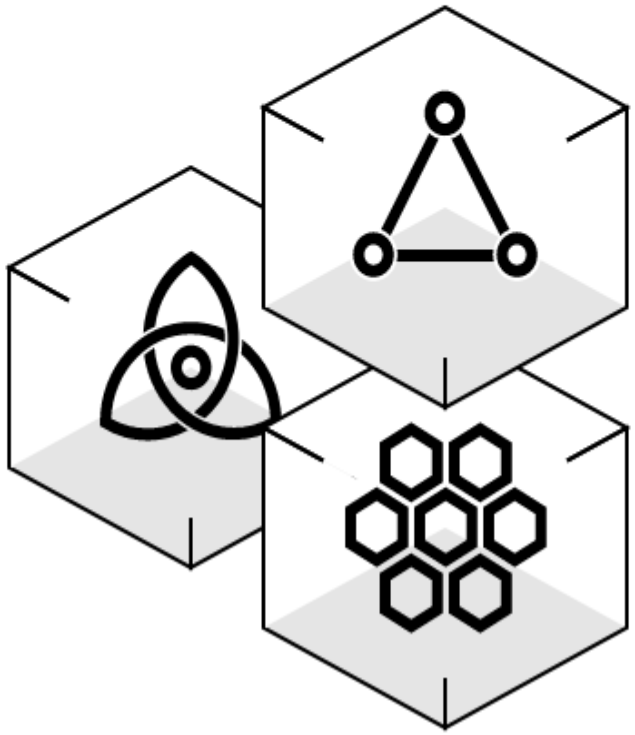
CS578

Tendermint

Tendermint Core | Cosmos SDK | IBC Protocol | Careers | Company

Building the most powerful tools for distributed networks.

We are Tendermint, a core contributor to the Cosmos Network.



https://tendermint.com/

ECOSYSTEM

Apps built with Cosmos & Tendermint

01 Binance DEX

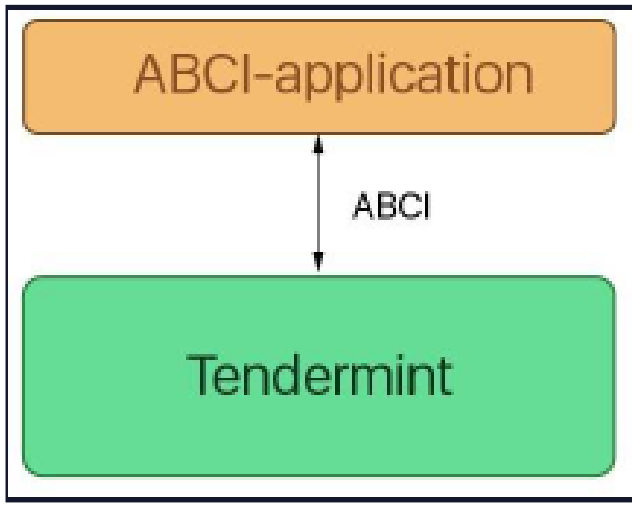
02 Oasis Labs

OASIS LABS

03 Terra

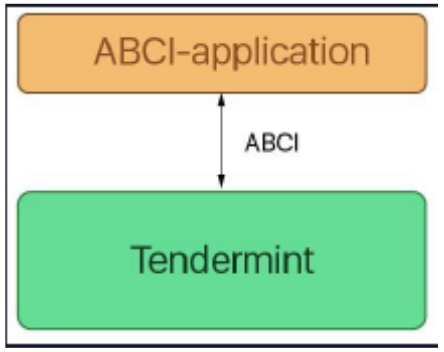
04 IRISnet

05 Regen Network



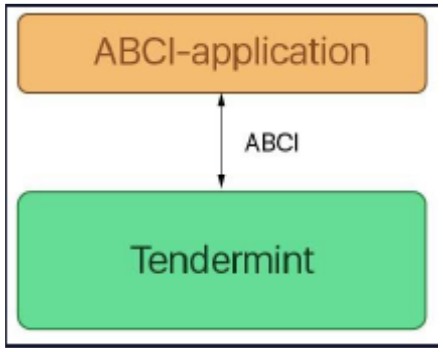
Introduction

https://docs.tendermint.com/master/introduction/what-is-tendermint.html



- Tendermint consists of two chief technical components:
  - a blockchain consensus engine - called **Tendermint Core**
  - a generic application interface - called the **Application BlockChain Interface (ABCI)**
- Tendermint Core ensures that the same transactions are recorded on every machine in the same order.
- ABCI enables the transactions to be processed in any programming language.
- Unlike other blockchain and consensus solutions, which come pre-packaged with built in state machines, developers can use Tendermint for BFT state machine replication of applications written in whatever programming language and development environment is right for them.

Introduction



- **ABCI** is an interface that defines the boundary between the replication engine (the blockchain), and the state machine (the application).
- Using a socket protocol, a consensus engine running in one process can manage an application state running in another.
- When Tendermint and ABCI applications run within the *same* process, Tendermint will call the ABCI application methods directly as Go method calls.
- When Tendermint and the ABCI application are run as separate processes, Tendermint opens four connections to the application for ABCI methods. The connections each handle a subset of the ABCI method calls.

Advantages of Tendermint:

- So far, all blockchains "stacks" have had a monolithic design.
- That is, each blockchain stack is a single program that handles all the concerns of a decentralized ledger, such as
  - P2P connectivity, "mempool" broadcasting of transactions, consensus on the most recent block, account balances, Turing-complete contracts, user-level permissions, etc.
- Reuse issue, Complex Maintenace
- This is especially true when the codebase is not modular in design and suffers from "spaghetti code".

Advantages of Tendermint:

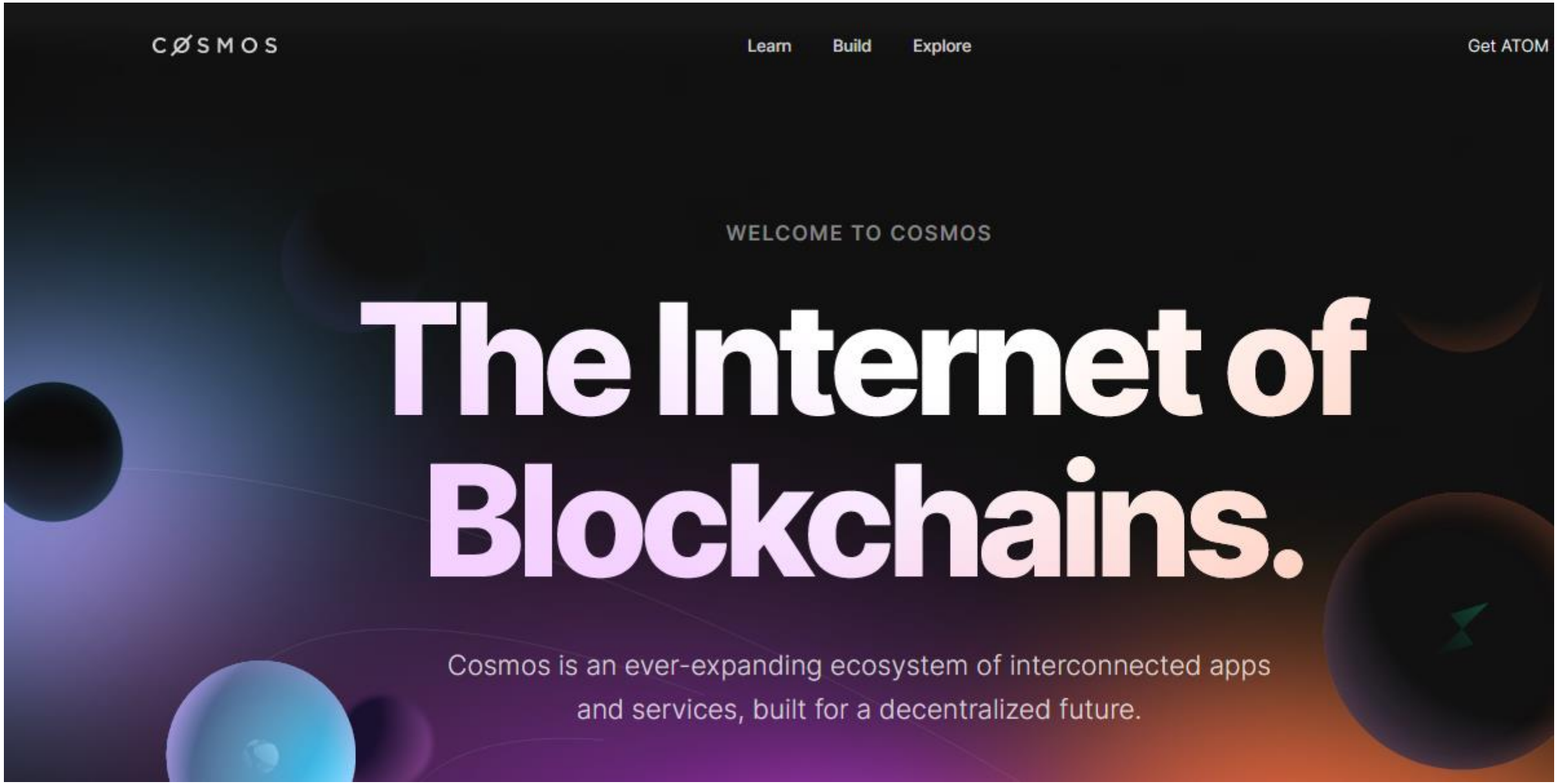
- Another problem with monolithic design is that it limits you to the language of the blockchain stack (or vice versa).
  - Ethereum supports bytecode virtual-machine, so it limits you to languages that compile down to that bytecode.

- Decoupling consensus engine and P2P layers from the details of application states.
  - We do this by abstracting away the details of the application to an interface, which is implemented as a socket protocol.

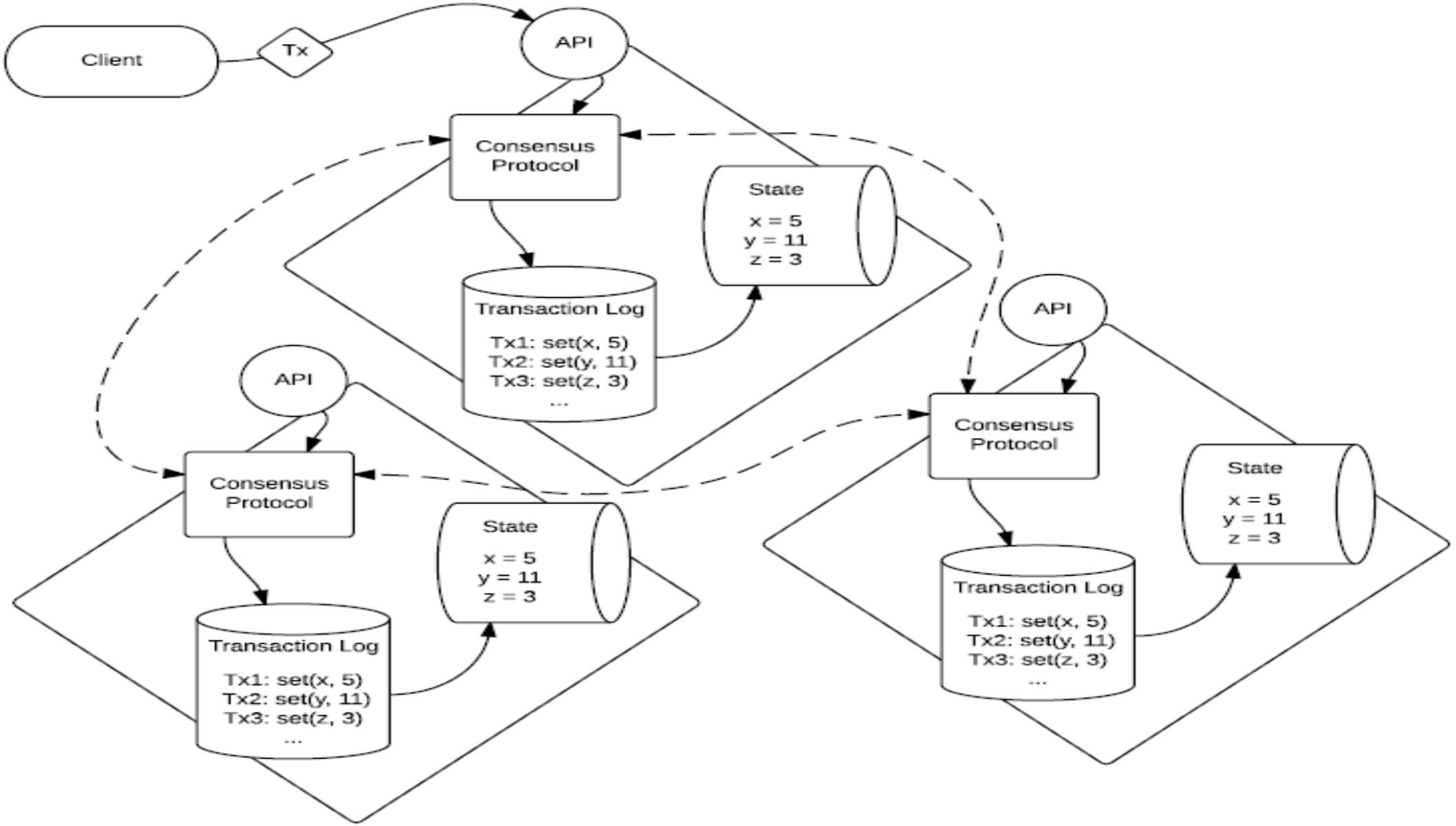
- Thus, we have ABCI, and its primary implementation, the Tendermint Socket Protocol (TSP, or Teaspoon).

An Analogy:

- Bitcoin is a cryptocurrency blockchain where each node maintains a fully audited Unspent Transaction Output (UTXO) database.
- If one wanted to create a Bitcoin-like system on top of ABCI, Tendermint Core would be responsible for
  - Sharing blocks and transactions between nodes
  - Establishing a canonical/immutable order of transactions (the blockchain)
- The application will be responsible for
  - Maintaining the UTXO database
  - Validating cryptographic signatures of transactions
  - Preventing transactions from spending non-existent transactions
  - Allowing clients to query the UTXO database.



https://cosmos.network/



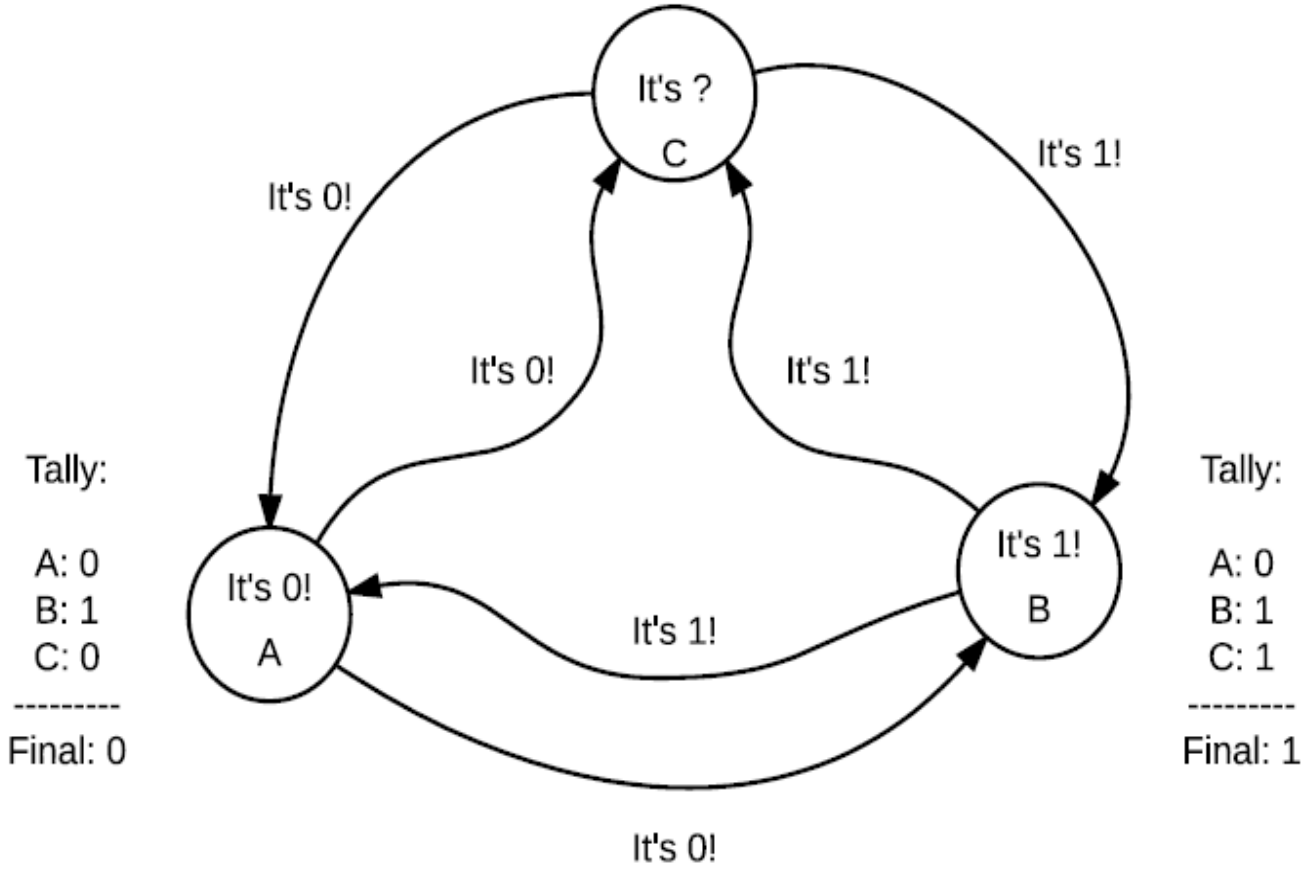
A replicated state machine replicates a transaction log and resulting state across multiple machines

Standard definition of the consensus primitive satisfies the following:

- **Termination:** every correct process eventually decides
- **Agreement:** if one correct process decides v1 and another decides v2, then v1 = v2
- **Validity:** if a correct process decides v, at least one process proposed v

Byzantine Fault Tolerance (BFT)

- Traditionally, consensus protocols tolerant of malicious behaviour were known as Byzantine Fault Tolerant (BFT) consensus protocols.
- Here C is a Byzantine Node.



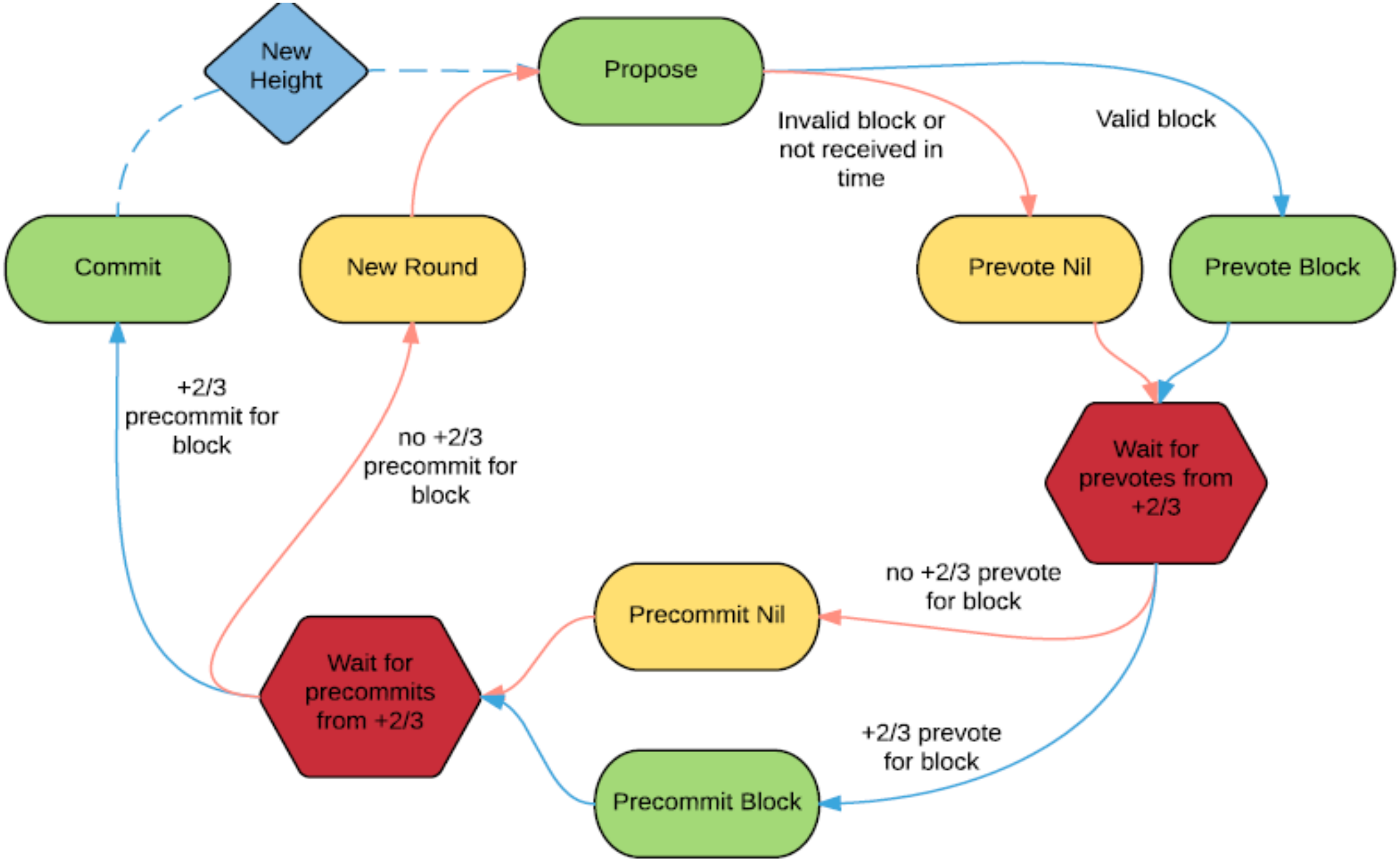
Tendermint BFT Consensus Algorithm

- Set of Validators
  - maintain full copy of the replicated state/blockchain
  - Propose new blocks
  - Vote new blocks
- Each validator is identified by public key
- A valid blockchain has one valid block at each height across the validators (safety property)
- Cycling Proposers: if one won’t process any transactions, others can pick up (liveness property)
- Tolerate: less than one-third are byzantine/offline/partitioned
- Multiple Rounds: Consensus start with Round-0

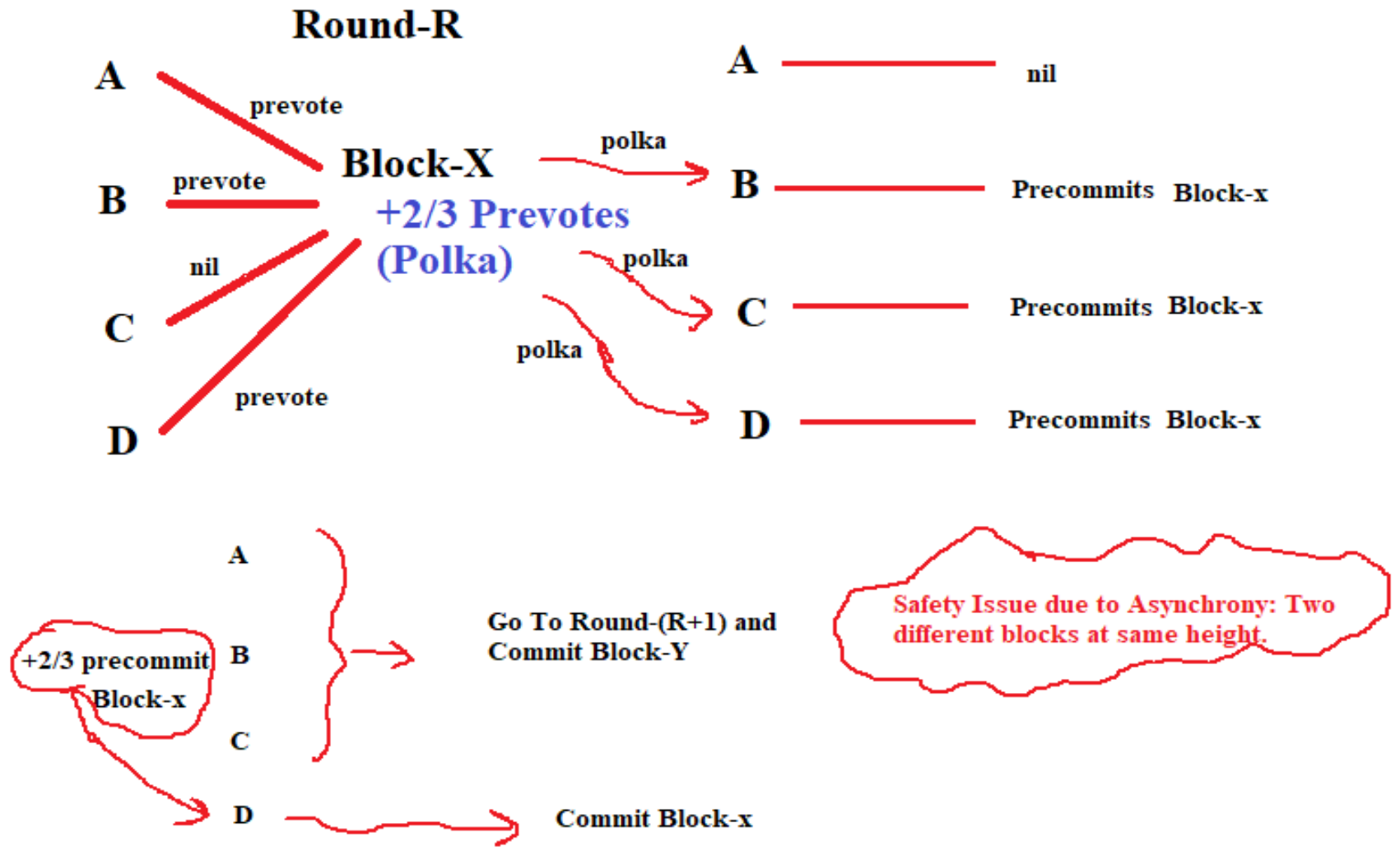
Tendermint BFT Consensus Algorithm

- **Proposals:** a new block must be proposed by the correct proposer at each round, and gossiped to the other validators.
  - If a proposal is not received in sufficient time, the proposer should be skipped.
- **Votes:** Follow two phases : pre-vote and pre-commit.
  - A set of pre-commits from more than two-thirds of the validators for the same block at the same round is a commit.
- **Locks:** Ensures no two validators commit a different block at the same height
  - determines how a validator may pre-vote or pre-commit depending on previous pre-votes and pre-commits at the same height.

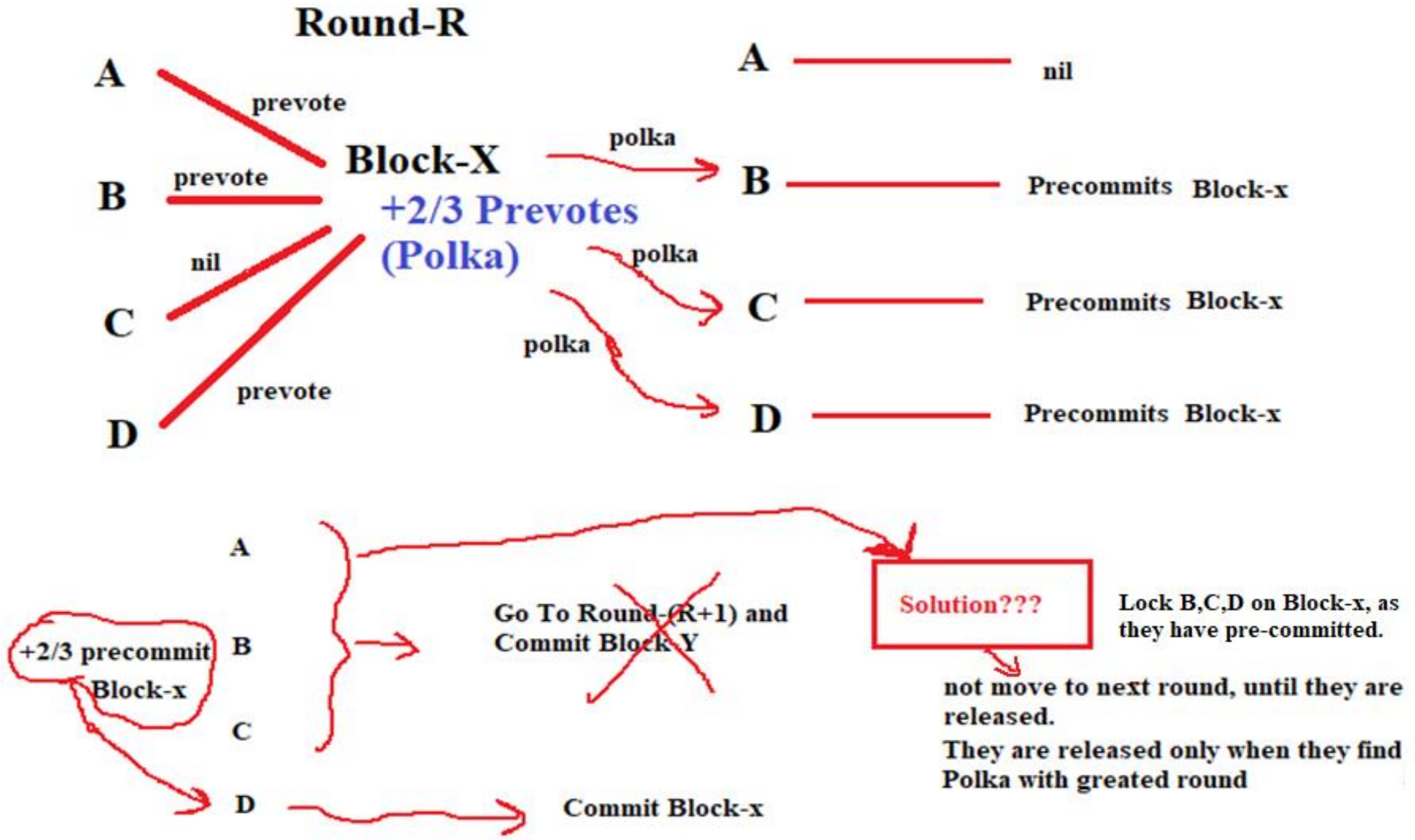
Tendermint BFT Consensus Algorithm



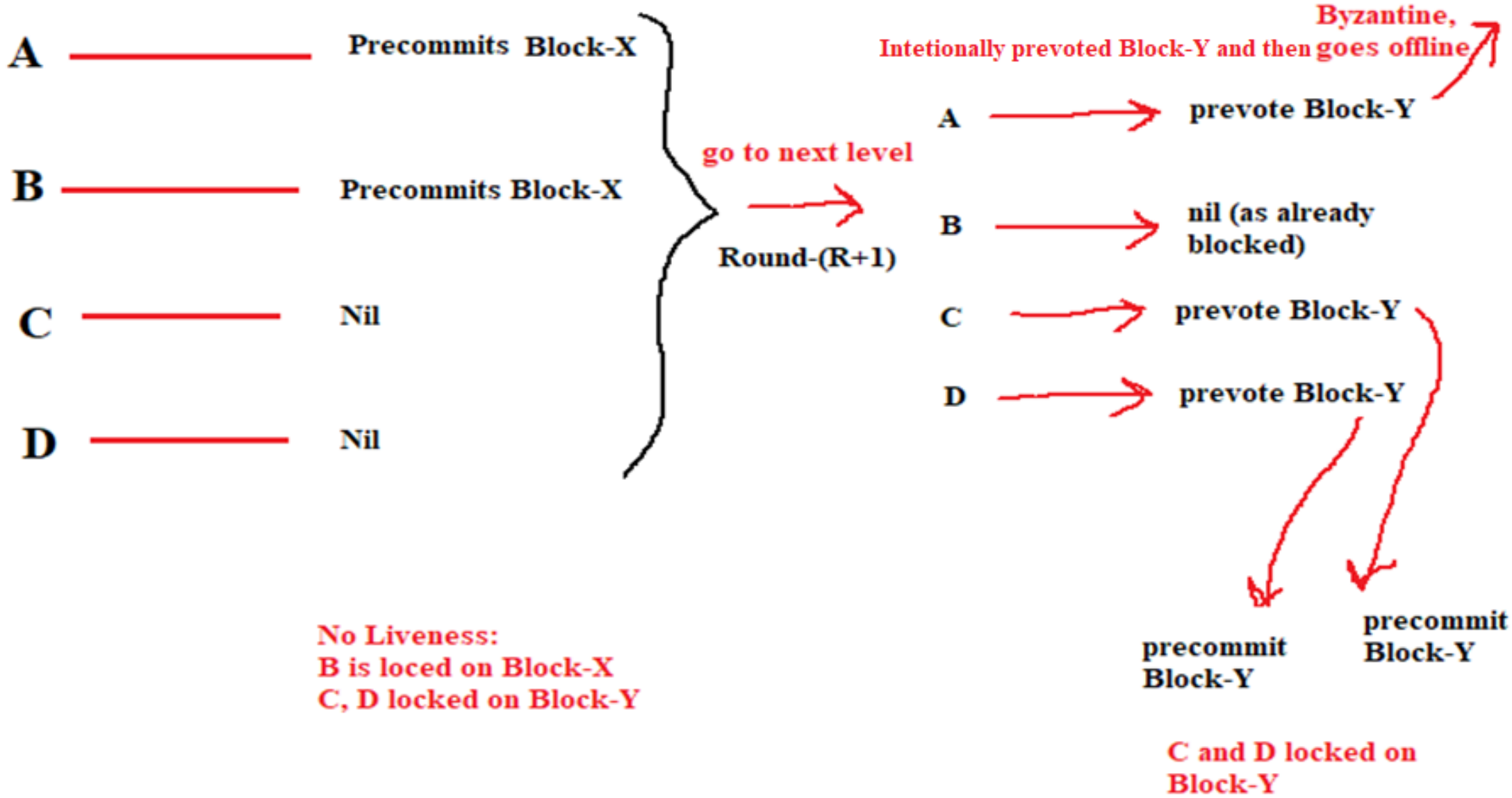
Tendermint BFT Consensus Algorithm: Safety?



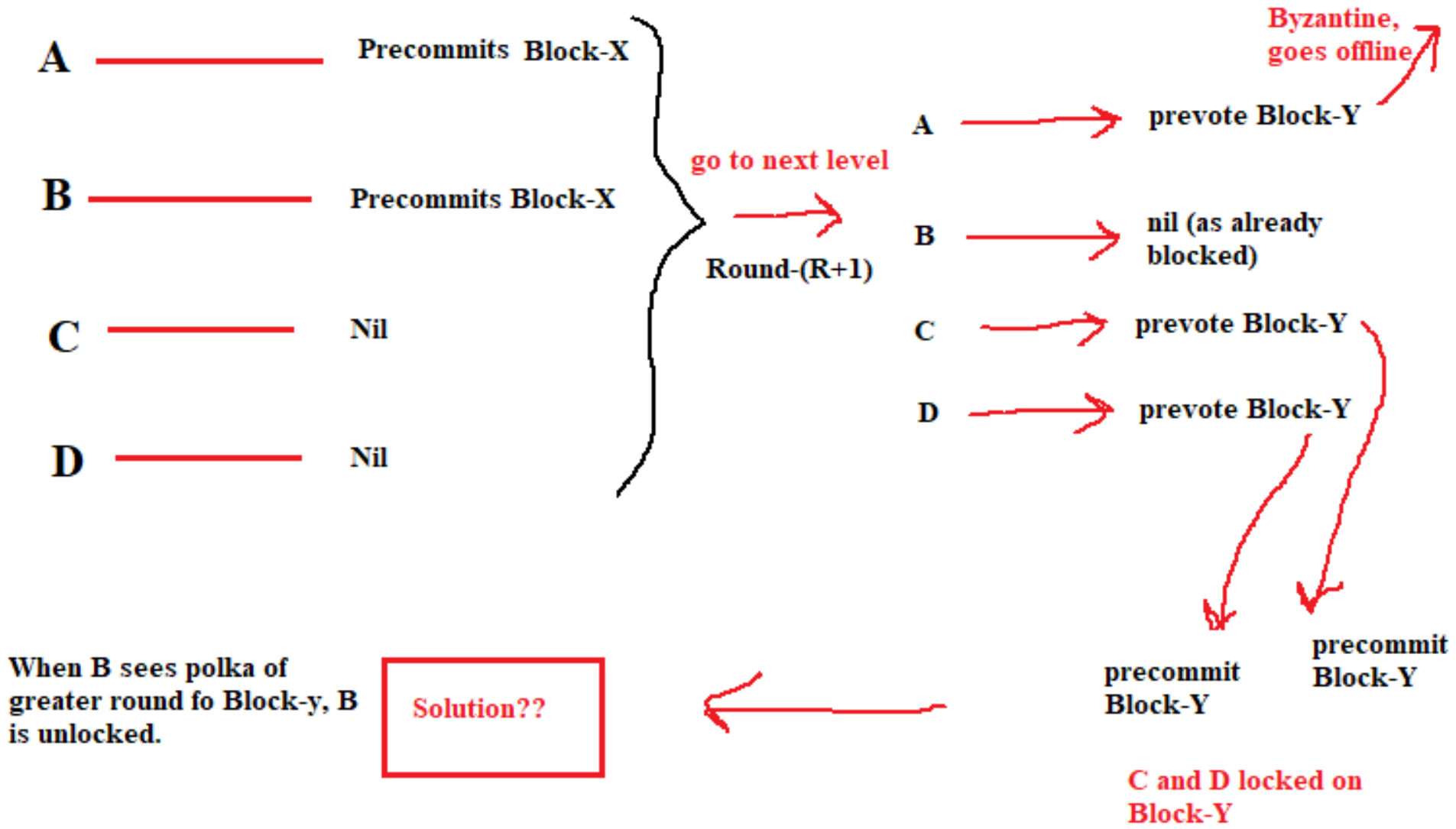
Tendermint BFT Consensus Algorithm: Safety?



Tendermint BFT Consensus Algorithm: Liveness?



Tendermint BFT Consensus Algorithm: Liveness?



There are two rules of locking:

- **Prevote-the-Lock:** a validator must pre-vote for the block they are locked on, and propose it if they are the proposer. It prevents validators from pre-committing one block in one round, and then contributing to a polka for a different block in the next round, thereby compromising safety.

- **Unlock-on-Polka:** a validator may only release a lock after seeing a polka at a round greater than that at which it locked. This allows validators to unlock if they pre-committed something the rest of the network doesn't want to commit, thereby protecting liveness.

Thank You