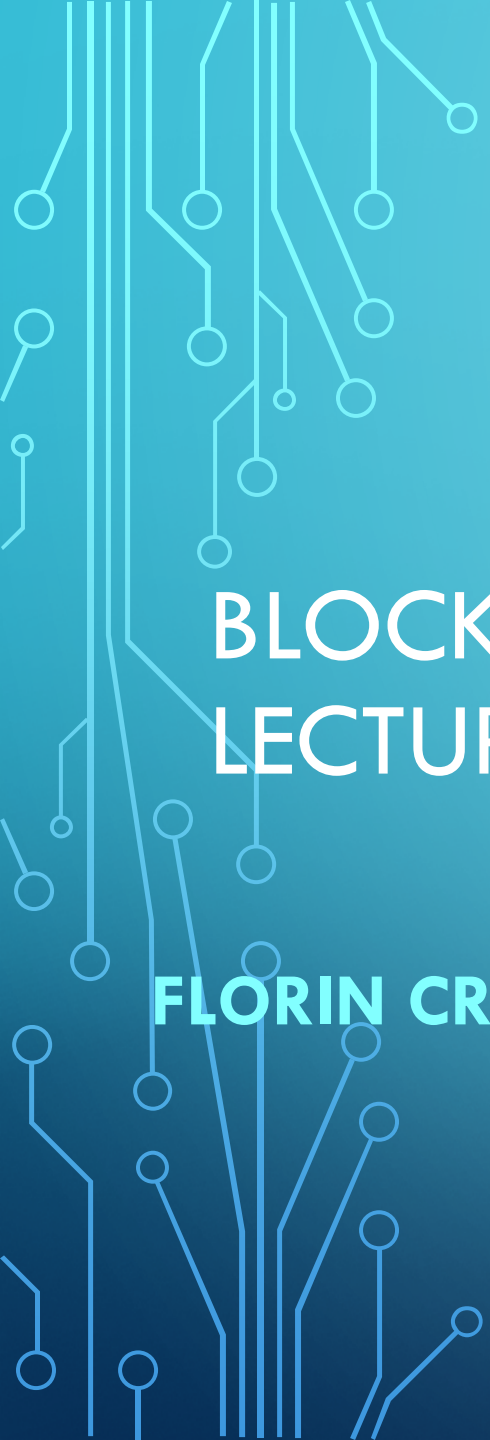


BLOCKCHAIN: SMART CONTRACTS LECTURE 8- CONSENSUS

FLORIN CRACIUN

IMPORTANT

Some of the following slides are the property of

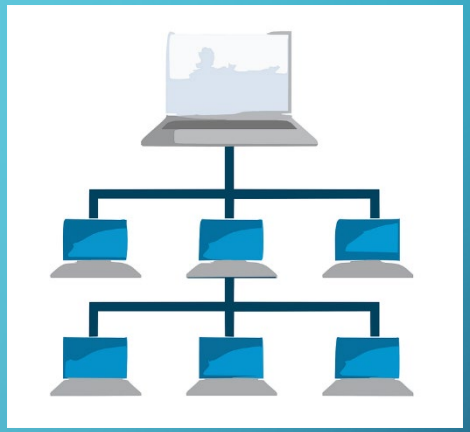
Dr. Emanuel Onica & Dr. Andrei Arusoaie

Faculty of Computer Science,

Alexandru Ioan Cuza University of Iași

and are used with their consent.

CONSENSUS IN DISTRIBUTED SYSTEMS
THE BASICS



A distributed system (informally):

- A set of distinct nodes (processes)
- Communication channels between nodes
- Working for a common goal
- Typically single-system view from the client perspective

Properties:

- Nodes operate concurrently
- Lack of global clock synchronization (typically asynchronous message passing)
- Failures of nodes can (and will) occur

A blockchain architecture is a type of distributed system

CONSENSUS IN DISTRIBUTED SYSTEMS
THE BASICS

Most common internal organization of a fault tolerant distributed system: a replicated state machine

- each node keeps a state log starting with an initial one
- state evolves following deterministic transitions
- the distributed system must achieve the common goal

Similar situation in a blockchain case:

- state log = the blockchain structure
- transitions = transactions changing the blockchain
- the nodes must have a unitary view over the blockchain

Caveat: Recent proof that implementing a cryptocurrency = decentralized payment system, does not require consensus in the sense of total order of transactions („The Consensus Number of a Cryptocurrency" – R. Guerraoui et al., PODC 2019)

CONSENSUS IN DISTRIBUTED SYSTEMS
THE BASICS

In general, the common goal of a distributed system requires a ***consensus*** *algorithm* in the presence of faulty nodes.

Two main characteristics for general proper operation of a distributed system:

- ***Safety:*** *Something bad will never happen.*
- ***Liveness:*** *Something good will eventually happen.*

A consensus algorithm is correct if it satisfies:

- ***Agreement*** *(safety property): all non-faulty nodes decide on the same output value* (sometimes defined including *validity* = the value is strictly proposed by the nodes)
- ***Termination*** *(liveness property): all non-faulty nodes eventually decide*

CONSENSUS IN DISTRIBUTED SYSTEMS
TYPES OF CONSENSUS

(Fischer, Lynch and Paterson, 1985): It is *impossible* to solve consensus in a totally asynchronous deterministic computation model (the typical context met).

- No fixed time limit for message delivery
- Nodes can fail at unpredictable times

However, in practice, the result can be circumvented:

- use some synchrony assumptions (classical approach)
- use some non-determinism

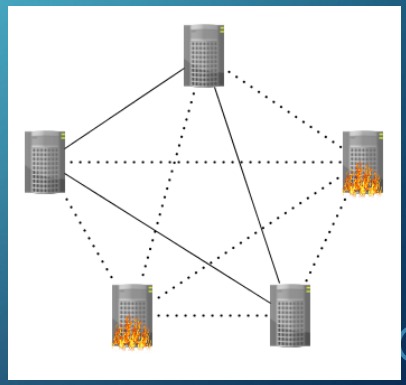
Two main flavors of fault tolerant consensus in distributed systems:

- ***Crash Fault Tolerant*** **(CFT)** – nodes can crash and stop communication
- ***Byzantine Fault Tolerant*** **(BFT)** – nodes can crash and stop communication and send corrupted messages

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

CFT consensus

- General guarantee: a system with $n=2f+1$ nodes can continue to operate if f nodes fail



- Significant solution examples:
  - Paxos (*The Part-Time Parliament*, Lamport 1998)
  - Raft (*In Search of an Understandable Consensus Algorithm*, Ongaro & Ousterhout 2014)

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Paxos:

- Notoriously difficult to understand
- Brief version: *Paxos Made Simple* (Lamport 2001)
- However... ambiguous parts have been detected => **„Do not try to implement the algorithm from this paper. "** (Lamport's website)

Users:

- Google's Chubby locking service:
  - allows clients to sync their activities and to agree on basic information about their environment
  - provides an interface similar to a distributed file system with advisory locks
- Google's Spanner newSQL database
- Microsoft's Autopilot data center management service
- ... and others

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Paxos:

- Description set in the context of a fictional parliament on the Paxos island in Greece, where members can take unexpected vacations

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Paxos – node roles:

- **Clients:** issue requests to the system and wait for responses (i.e., proposing a „law")
- **Proposers:**
  - advocates clients requests and ensure protocol advancement
  - one proposer can be considered having the role of leader
- **Acceptors:**
  - vote on requests
  - grouped in quorums, such as two quorums share at least one acceptor and a quorum includes a majority of acceptors
- **Learners:**
  - take note of a request that was accepted and inform the client of the results

Final scope: consensus on a request = proposed value *v*

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Basic Paxos – phases:

Phase 1:

- Proposer *P* receives a request from a client
- Proposer *P* chooses *version number n* greater than any previous of his chosen *n* (note: n is not the proposed value)
- Proposer *P* sends a <Prepare, *n*> message proposal to a quorum of acceptors
- Each acceptor A_i verifies if *n* is higher than for any previous proposal number received:
  - YES: A_i returns to *P* a <Promise, *n*, [*m*, *w*]> that guarantees that A_i will ignore any proposal lower than *n*; if such one already exists the message also includes:
    - *m* - the former accepted Prepare version
    - *w* - the corresponding accepted value
  - NO: acceptor A_i ignores the proposal

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Basic Paxos – phases:

Phase 2:

- Proposer *P* checks if a majority of promises has been received from the acceptors quorum
- Proposer *P* sets the proposed value *v* to previous accepted value *w* of highest version received in promises, if any; otherwise it proposes an original initial value *v* (*)
- Proposer *P* sends a <Accept, *n*, *v*> request to the quorum of acceptors
- Each acceptor A_i accepts the request if and only if A_i didn't promise meanwhile to consider proposals with a version higher than *n* (i.e., coming from a different proposer):
  - YES: A_i sends an <Accepted, *n*, *v*> message to all learners and to P
  - NO: acceptor A_i ignores the accept request

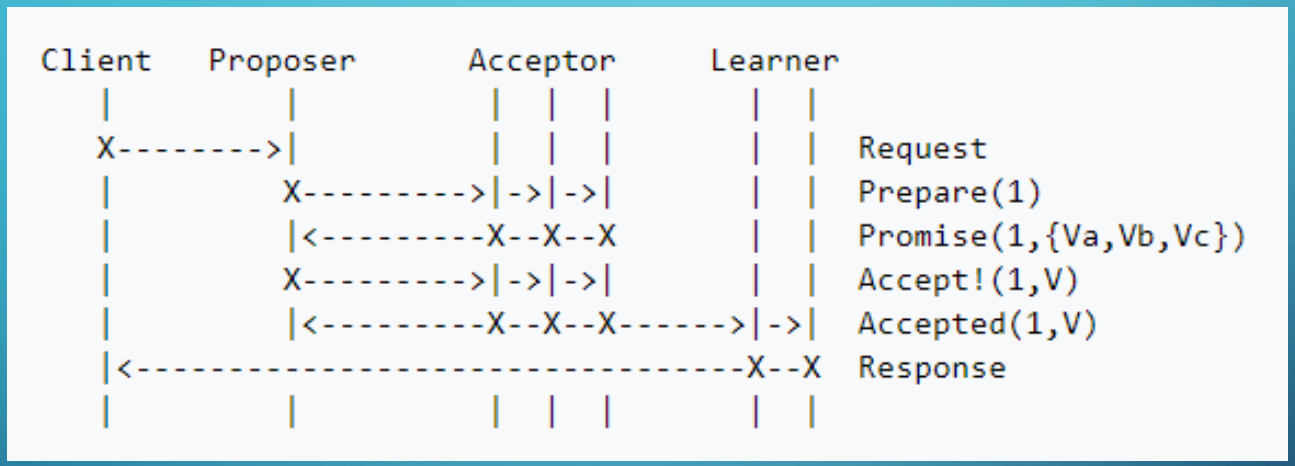
CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Basic Paxos:

- Learners receive the accepted confirmation from a majority of acceptors and finally inform the client about the chosen consensus.
- An acceptor can accept multiple proposals – i.e., from other proposers unaware of current decision process
- However (*) in phase 2 guarantees that the common value currently being decided on will be consistent on all nodes
- A proposer is de facto established as leader when acceptors accept its proposal

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Basic Paxos (no failures):

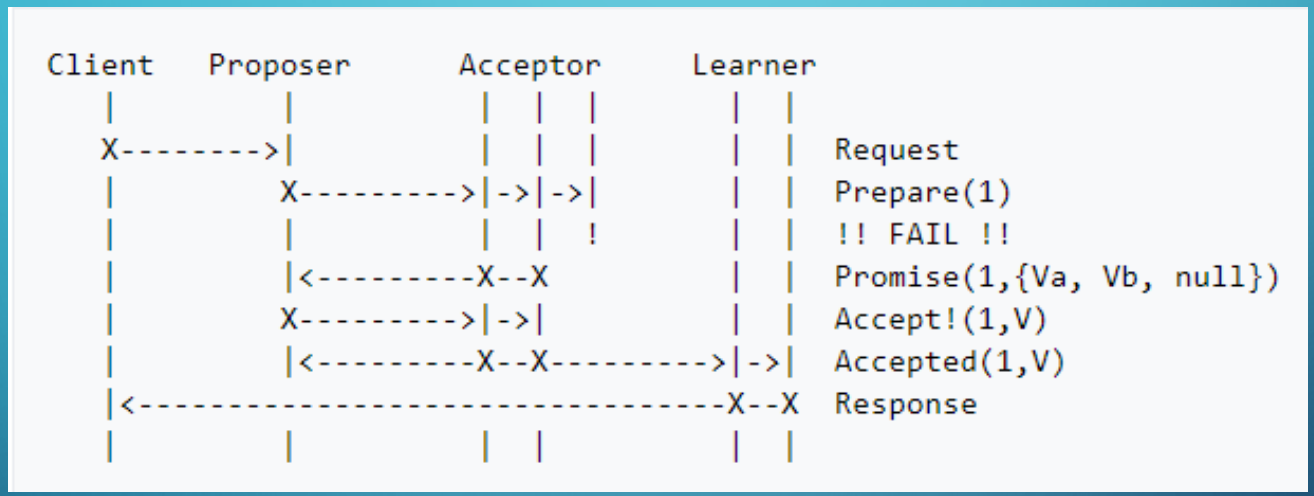


(Va, Vb, Vc are considered as initially similar to null values in the example)

Figure source: *wikipedia.org*

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Basic Paxos (failed acceptor):

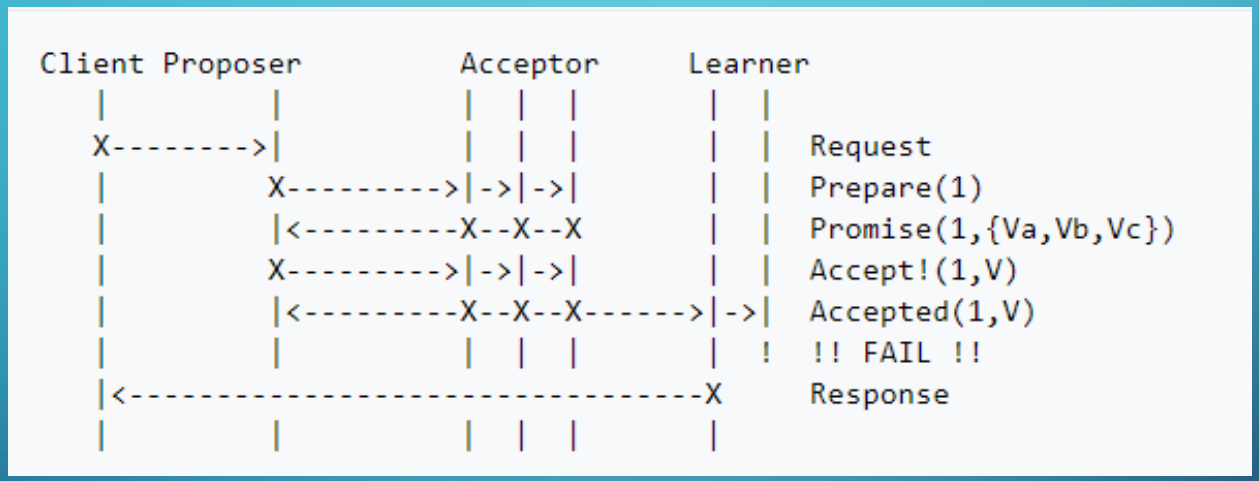


Va, Vb, Vc are considered as initially similar to null values in the example)

Figure source: *wikipedia.org*

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Basic Paxos (failed learner):

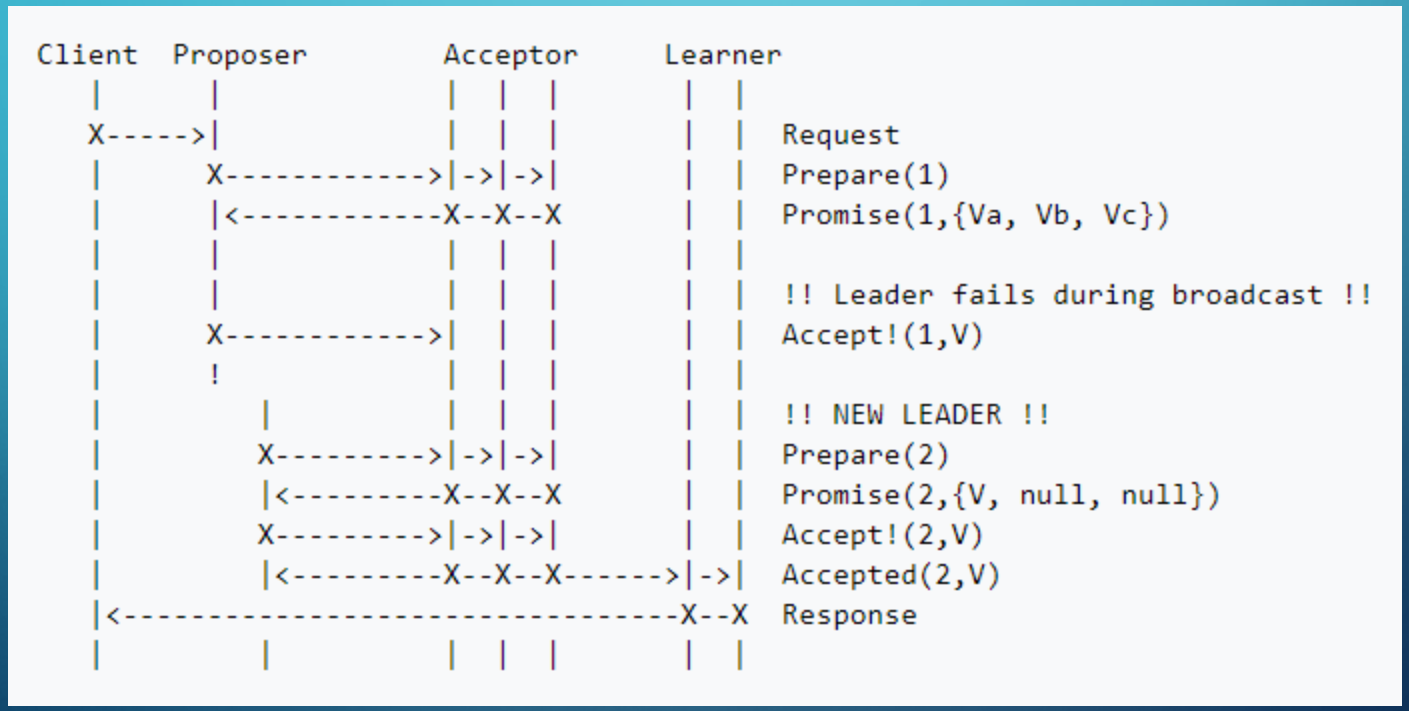


Va, Vb, Vc are considered as initially similar to null values in the example)

Figure source: *wikipedia.org*

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Basic Paxos (failed proposer, transition to new leader not detailed):



(Va, Vb, Vc are considered as initially similar to null values in the example)

Figure source: *wikipedia.org*

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Paxos provides safety – there's no situation when learners can end up with decision on two distinct values for one protocol iteration

Liveness is probabilistically achievable if sufficient nodes remain non-faulty. Also some synchrony (i.e., message timeouts) should be enforced.

Paxos main issues - many blank spots in description:

- „Ignore" situations in phase 1 and 2 can be replaced with NACKs sent by acceptors to stop proposers trying
- Mechanism to detect failures and some situations to act upon left open
- What if two proposers conflict?

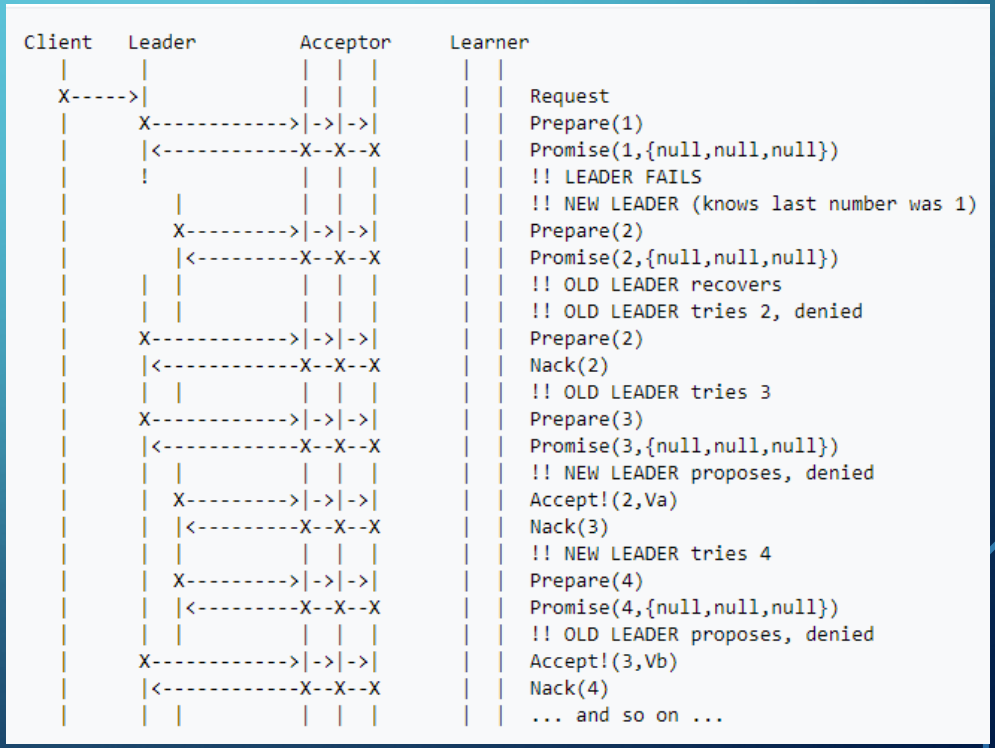
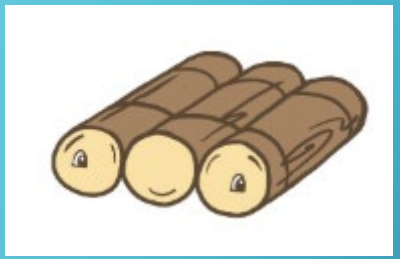


Figure source: wikipedia.org

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

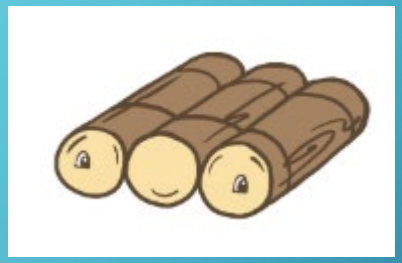
Raft consensus protocol



- Unofficial explanation of chosen name = built to evade Paxos
- Appeared after multiple variations or adaptations on Paxos (cheap Paxos, fast Paxos, etc.)
- Includes a clear leader election phase, after which the leader is fully responsible for managing the state log replication
- Defines precise timeouts for heartbeat synchronization with the leader, and triggering a new leader election when the current leader is down
- Implemented in Quorum blockchain platform

CONSENSUS IN DISTRIBUTED SYSTEMS
PAXOS AND RAFT (CFT CONSENSUS)

Raft consensus protocol



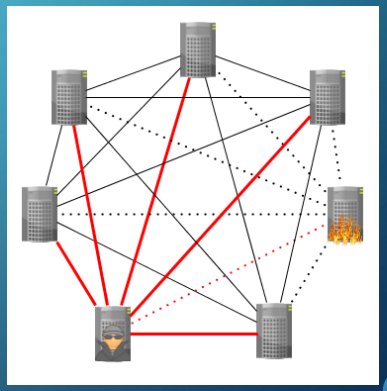
DEMO

http://thesecretlivesofdata.com/raft/

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

BFT consensus

- General guarantee: a system with $n=3f+1$ nodes can continue to operate if f nodes are faulty



- Significant solution examples:
  - PBFT (*Practical Byzantine Fault Tolerance*, Castro & Liskov 1999)
  - Zyzziva (*Zyzziva: Speculative Byzantine Fault Tolerance*, Kotla et al. 2007)
  - BFT-Smart (*State machine replication for the masses with BFT-SMART*, Bessani et al. 2014)

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

BFT consensus

- Why less than 3f+1 nodes is not enough?
- In a BFT setting faults follow arbitrary behavior (e.g., corrupted messages, crashes, etc.)
- Example for f=1

(can be generalized for higher values):

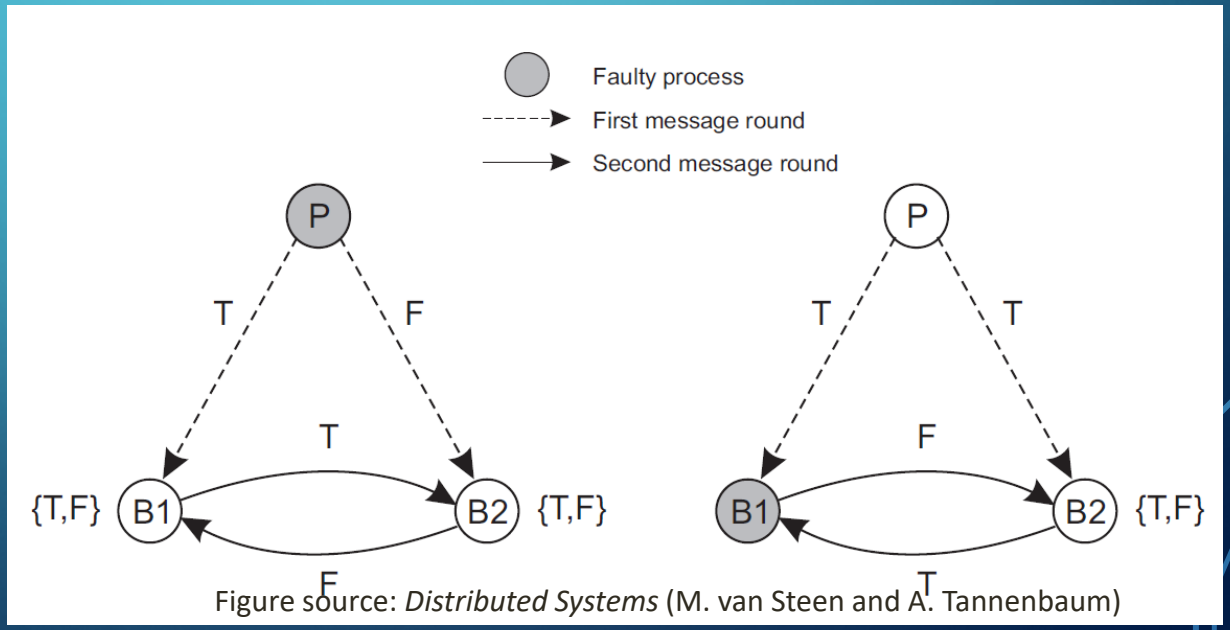


Figure source: *Distributed Systems* (M. van Steen and A. Tannenbaum)

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

PBFT – Practical Byzantine Fault Tolerance

- Proposed in 1999 by Miguel Castro and Barbara Liskov
- Probably the most cited BFT consensus solution
- Source for many other variations and optimizations

The setting:

- *R* replicas move through a series of *views* identified by consecutive numbering – current view = *v*
- During a view:
  - One replica is considered ***primary*** = $p = v \bmod R$
  - All other replicas are considered ***backups***
  - If the primary seems to be failed a view change is initiated

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

PBFT – protocol overview:

- A client sends a request to the primary replica *p*
- The primary multicasts the request to the backups
- Replicas execute the request, changing state, and reply to the client
- The client waits for f+1 consistent replies from different replicas
- Messages are digitally signed by senders and verified by receivers

Looks simpler than Paxos, right ? ☺

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Client operation

- Client c to primary:
  - Forms a message <REQUEST, o, t, c> where o is the requested op and t is a local timestamp (provides ordering and exactly-once semantics for messages originating from same source)
  - Sends the request to current assumed primary

- Replicas to client c:
  - Form and send a message <REPLY, v, t, c, i, r> where v is the view number, t is the local timestamp, i identifies the replica and r is the result to the request
  - Client c waits for f+1 replies with the same r, valid increasing timestamp and signature

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Client operation

- Use of synchrony - timeout on client request:
  - Client c will re-send request to all replicas
  - Replicas always keep the last reply sent to any client
  - If re-sent request has been seen and processed by replica the reply is re-sent to the client
  - If re-sent request is new:
    - if replica is not primary, replica forwards the request towards the primary
    - the primary is expected to resume the normal protocol by multicasting back request to all replica backups
    - if primary does not resume by multicasting back is detected as faulty and a change of view (and primary) is initiated

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Primary-Backups operation

- Three-phase protocol:
  - Phase 1: pre-prepare
  - Phase 2: prepare
  - Phase 3: commit

- Solving requests changes the replicated state log
- Phase 1 & 2 provide guarantee of total order for requests sent in the same view even with a faulty primary
- Phase 2 & 3 provide guarantee of committed (executed) requests are totally ordered accross views
- Maintaing the execution order of requests leads to consistent results at replicas

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Primary-Backups operation – Phase 1: pre-prepare

- After receiving a request primary *p* forms the message <PRE-PREPARE, v, n, d, m> where:
  - v indicates the view
  - n is an increasing sequence number
  - d is the request message digest
  - m is the original request message (excluded from the signature for performance purposes)
- The primary *p* sends the message to all backup replicas
- A replica accepts the pre-prepare message if:
  - digest and signatures are valid
  - current view = v
  - no other message in view v with sequence n was received
  - h < n < H, where (h,H) are boundary parameters set to prevent a faulty primary to exhaust the space of sequence numbers

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Primary-Backups operation – Phase 2: prepare

- After accepting a pre-prepare message backup replica *i:*
  - adds the pre-prepare message and request m to its log
  - forms a message <PREPARE, v, n, d, i> using the values in the pre-prepare message
  - adds the prepare message to its log
  - broadcasts the prepare message to all other replicas (including primary)
- A replica accepts the prepare message if:
  - digest and signatures are valid
  - current view = v
  - h < n < H, where (h,H) are boundary parameters set to prevent a faulty primary to exhaust the space of sequence numbers

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Primary-Backups operation – Phase 2: prepare

- Accepted prepare messages are added to the replica log
- Predicate *prepared(m,v,n,i)* is considered true at replica i when its log contains:
  - the request m and a pre-prepare for m in view v with sequence n
  - 2f prepare messages from different replicas matching the pre-prepare

- Phase 1 & 2 ensure the safety invariant:

  (prepared(m,v,n,i) = true) => (prepared (m',v,n,j) = false)

  for any non faulty j and any m' ≠ m

CONSENSUS IN DISTRIBUTED SYSTEMS

PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Primary-Backups operation – Phase 3: commit

- When *prepared(m,v,n,i)* becomes true at replica i:
  - replica i forms message <COMMIT, v, n, d, i>
  - multicasts the commit message to all other replicas

- A replica accepts the commit message if:
  - digest and signatures are valid
  - current view = v
  - h < n < H, where (h,H) are boundaries as defined before

CONSENSUS IN DISTRIBUTED SYSTEMS

PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Primary-Backups operation – Phase 3: commit

- Accepted commit messages are added to the replica log
- Following predicates are defined:
  - *committed-local(m,v,n,i)* true, when *prepared(m,v,n,i)* true and replica i accepted 2f+1 commits (including its own) matching the pre-prepare for m
  - *committed(m,v,n)* true, when *prepared(m,v,n,i)* true for all replicas i in a set of f+1 non-faulty replicas (global system state predicate)

- It can be shown that *committed-local(m,v,n,i)* => *committed(m,v,n)* following the commit phase
- This invariant guarantees that a request committed at a non-faulty replica will eventually be commited at at least f+1 replicas, providing consensus

CONSENSUS IN DISTRIBUTED SYSTEMS
PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Wrapping up

- Replica i will finally execute the *m* request and reply to the client when:
  - (a) *committed-local(m,v,n,i)* true
  - (b) replica i executed all requests with sequence numbers < n

- Safety is ensured: all replicas will execute requests in the same order maintaining replicas state consistent

- Liveness provided through timers set for executing requests:
  - a backup replica starts a timer for request execution when a request is received
  - if the timer expires the replica will initiate a view change protocol resulting in changing the primary and re-iterating the consensus phases

- Further details available in the PBFT paper

CONSENSUS IN DISTRIBUTED SYSTEMS

PRACTICAL BYZANTINE FAULT TOLERANCE (BFT CONSENSUS)

- PBFT – Operation summary

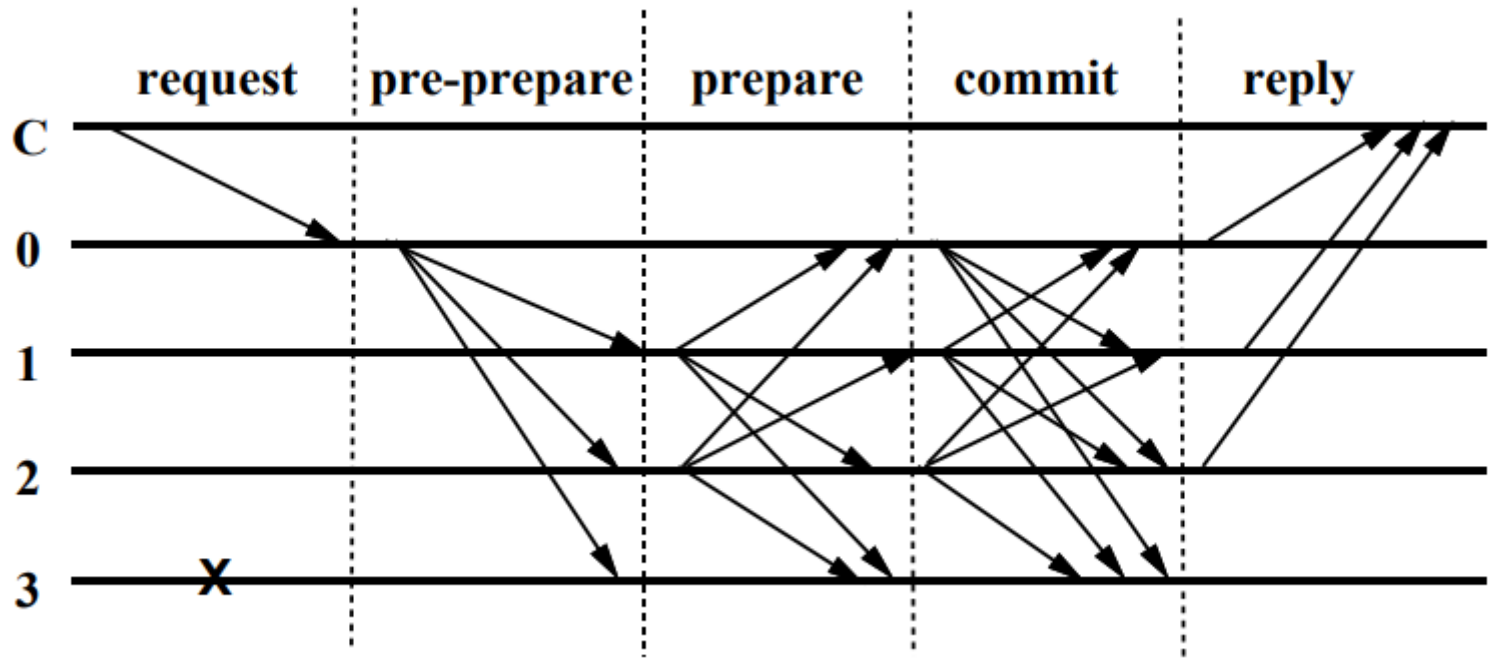


Figure source: *Practical Byzantine Fault Tolerance* (M. Castro and B. Liskov)

CONSENSUS IN BLOCKCHAIN

CONSENSUS IN BLOCKCHAIN

- A blochain architecture is a distributed system
- The effective blockchain is a replicated data structure
- So, why can't we use typical consensus algorithms?

- Well, we actually can in theory (and sometimes in practice) … However:
  - Issue 1: Unknown & untrusted users => CFT not enough
  - Issue 2: Size of network & high churn => scalability issues

CONSENSUS IN BLOCKCHAIN

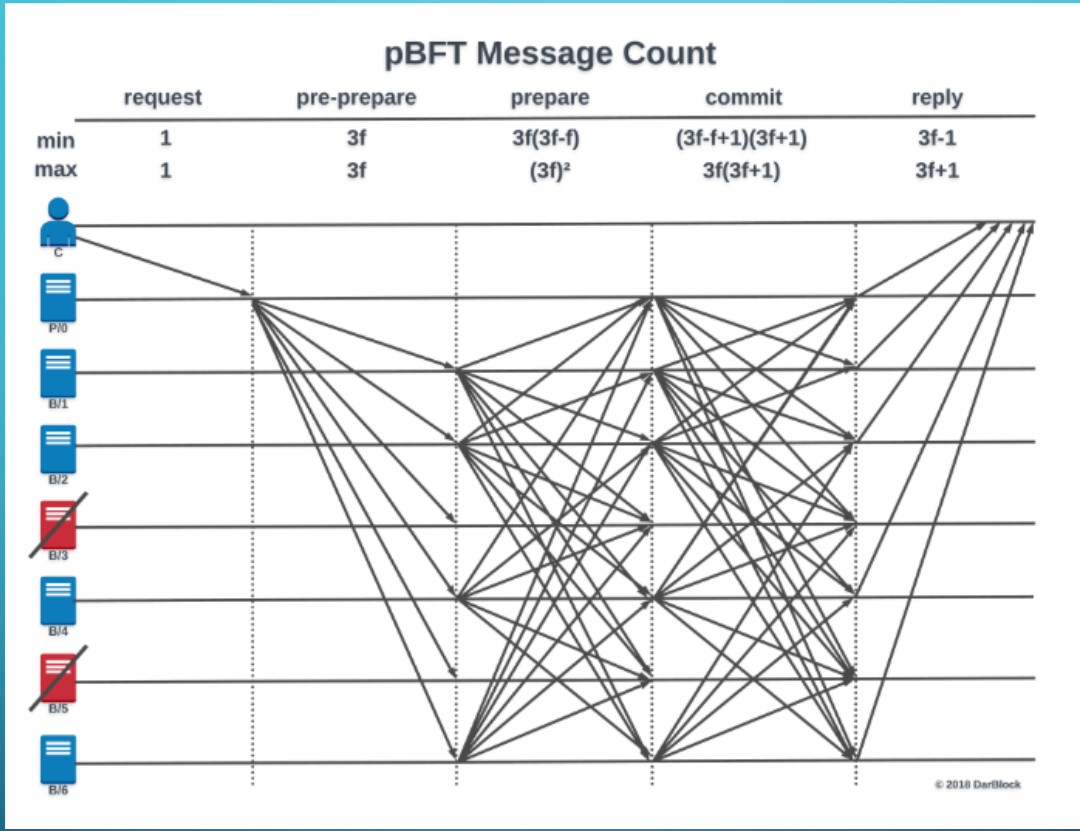


Figure source: *medium.com*

- PBFT 7 replicas (max 2 faulty): minimum 71 messages
- PBFT 10 replicas (max 3 faulty): minimum 142 messages
- PBFT 13 replicas (max 4 faulty): minimum 237 messages

CONSENSUS IN BLOCKCHAIN

- Remember the imposibility result: *can't have consensus* in a totally asynchronous deterministic computation model

- We have seen that „classical" consensus algorithms include some synchrony assumptions to bypass this

- Blockchain architectures: the first category of distributed systems to put more weight on using the 2nd way to circumvent the theory – use some non-determinism

CONSENSUS IN BLOCKCHAIN

- **Classical distributed consensus: typically a leader (proposer, primary) coordinates with peers (acceptors, backups) to reach the same result**

- **Blockchain consensus – typical framework:**
  - **A leader is elected through some lottery**
  - **The leader proposes a new block (equivalent to initial proposal in classical consensus)**
  - **The rest of peers check & agree on the block, including it in their chain (equivalent to having some implicit votes in classical consensus)**
  - **No tight coordination agreement in the final phase: nodes do not agree strictly *on the value* of the new block, but on *the high probability* that the block value is the correct one**

CONSENSUS IN BLOCKCHAIN
PROOF-OF-WORK (NAKAMOTO CONSENSUS)

- **We already mostly covered proof-of-work consensus in the intro & Bitcoin lectures**
- Quick recap:

  - Leader elected by criteria of mining power
  - First node able to finish a certain hard hash computation wins the lottery
  - Block is distributed to other nodes, which locally verify its correct linking with previous blocks in the chain and add it as last block
  - Case of forks (independent computation of new blocks) – solved by the rule of „longest chain wins”: nodes continue with first received new block until either current chain or the different one gets longer
  - Nodes are incentivized for finalizing first computation on a new block

CONSENSUS IN BLOCKCHAIN
PROOF-OF-WORK (NAKAMOTO CONSENSUS)

- **Safety:**
  - **„longest chain" rule ensure that all nodes will reach the same block**
  - **hardness of computation + incentivizing peers prevents malicious competing for replacing the correct chain**
- **Liveness:**
  - **incentives motivate the peers to keep computing towards generating the next block**
- **Both safety and liveness have a (safe) probabilistic degree:**
  - **in theory a parallel forked chain could go on forever**
  - **in theory an evil miner could amass enough power to secretely mine his own chain and overcome the length of the honest one**
  - **in theory all miners could do a full stop and not finish the next block**
- PoW is considered practically tolerant to byzantine faults due to above guarantees (Satoshi said so: https://www.mail-archive.com/cryptography@metzdowd.com/msg09997.html). However, no formal proof provided...

CONSENSUS IN BLOCKCHAIN
PROOF-OF-STAKE (POS)

- **Probably the most concerning PoW problem: energy consumption**

- **PoS idea: elect leader (block validator) based on economic power instead of computational power**
- **In brief: the peer who stakes more $ = locks these for a while, gets a chance proportional to the stake (and/or lock time) to „forge" the next block (equivalent of having the block „mined")**

- **Assumption: rich peers who stake (lock) more of their wealth have the interest in the proper functioning of the framework, which will increase the value of their coins**

CONSENSUS IN BLOCKCHAIN
PROOF-OF-STAKE (POS)

Flavors of Proof-of-Stake

Chain-based PoS:

- **A block validator is chosen from the group of stake holders using a function on the stake value**
- **Randomized block selection: validator choice function is a combination of a lowest hash computed by validator and highest stake**
- **Coin age based selection: validator choice function is a combination of time a stake was kept (longer = higher chances) and the stake value**
- **Other variations can exist, but in any case it is desired that the function to include some randomness factor (why?)**
- **Chosen block validator forges a new block and adds it to the blockchain getting typically as reward the fees of the transactions included in the block**

CONSENSUS IN BLOCKCHAIN
PROOF-OF-STAKE (POS)

Flavors of Proof-of-Stake

BFT-based PoS:

- **A block validator is randomly chosen from the group of stake holders based on the proportion of their stake value**
- **Validator forges and proposes a new block**
- **All other stake holders vote for confirming the block if they consider it valid**
- **If more than 2/3 voting power decides on validity the block is included in the blockchain; otherwise a new block validator is chosen and protocol restarts**
- **The final chosen block validator gets typically as reward the fees of the transactions included in the block**

CONSENSUS IN BLOCKCHAIN
PROOF-OF-STAKE (POS)

The nothing-at-stake issue:

- **Assume a fork occurs**

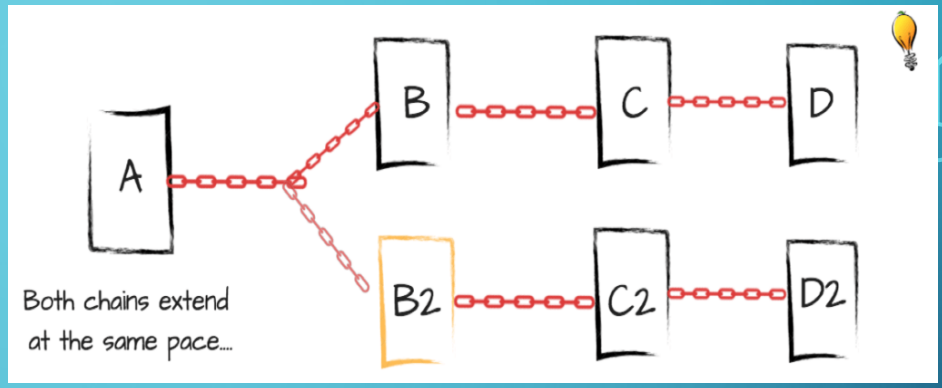


Figure source: *Nothing at Stake Problem – mangoresearch.co* (S. Dexter)

- **Essentially the cost of building forward on both forked chains is zero (locked stake is visible in each of the new forked blocks)**
- **Valid especially in early PoS versions where the stake = simply the amount a peer owns in the network**
- **Financial interest motivates further staking on both chains: chance on gaining more on transaction fees from whichever fork wins**
- **Disrupts the network and can lead to double spending attacks**

CONSENSUS IN BLOCKCHAIN
PROOF-OF-STAKE (POS)

Assumptions necessary for nothing-at-stake impact:

- **Peers will seek profit by following more forks even if it might hurt the network**
- **No peers will choose only one fork (to make it gain quickly over the other)**
- **The implementation permits or can be corrupted to allow staking on multiple forks**

Typical considered variant of mitigation (also for other possible attacks):

- **Stake is an actual deposit set to be much higher than transaction fees gained for one block**
- **Network penalizes any detected fraudulent activity by confiscating part of the stake and forbidding peer to be chosen as block validator**

CONSENSUS IN BLOCKCHAIN
PROOF-OF-STAKE (POS)

Multiple variations of PoS implementation

Probably the most significant: Delegated Proof-of-Stake (DPoS):

Idea: delegate the consensus job to a selected part of the network

- **Block proposers *(witnesses)* form a fixed limited subset**
- **The chosen block proposer is voted into power by users of the network**
- **The vote power is proportional to the stake size**
- **Peers can *delegate* their stake to an elector in a subset who will vote on their behalf amassing more voting power**

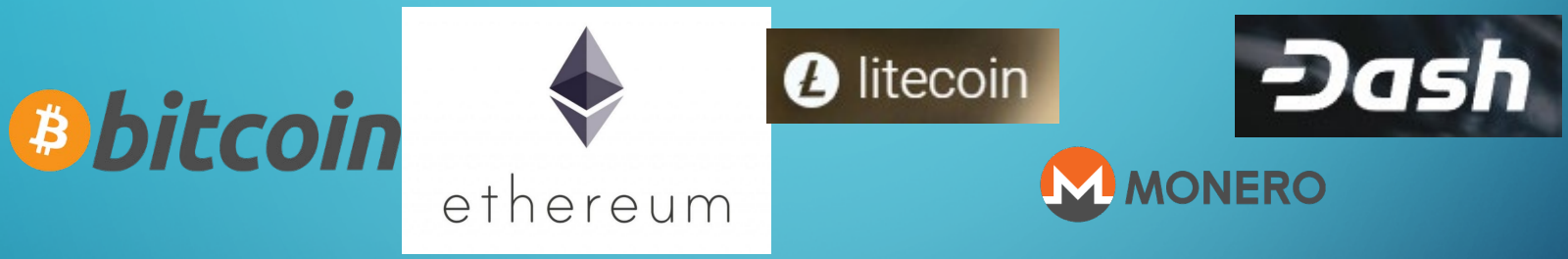
CONSENSUS IN BLOCKCHAIN
PROOF-OF-STAKE (POS)

DPoS implementantions vary:

- **Witness role sometimes overlaps with delegate role**
- **Witnesses (or sometimes delegates) are rewarded for proposing new blocks**
- **Rewards are sometimes shared with voters**
- **Misbehaving witnesses or delegates are voted down**
- **Peers interest is to act correctly since they compete for being voted (getting rewards for their work)**

CONSENSUS IN BLOCKCHAIN
POW VS POS IN PRACTICE

Majority of major public blockchain implementations still follow the PoW design:



PoS is seemingly the next choice due to high costs of PoW:

- **Ethereum – started migration to PoS in December 2020**
- **Tendermint – BFT based PoS framework used as building block in Cosmos (attempt to an Internet of Blockchains)**
- **Some cryptocurrencies: Nxt, EOS (DPos), Steem (DPos)**

CONSENSUS IN BLOCKCHAIN
ETHEREUM 2.0 – POS BASED

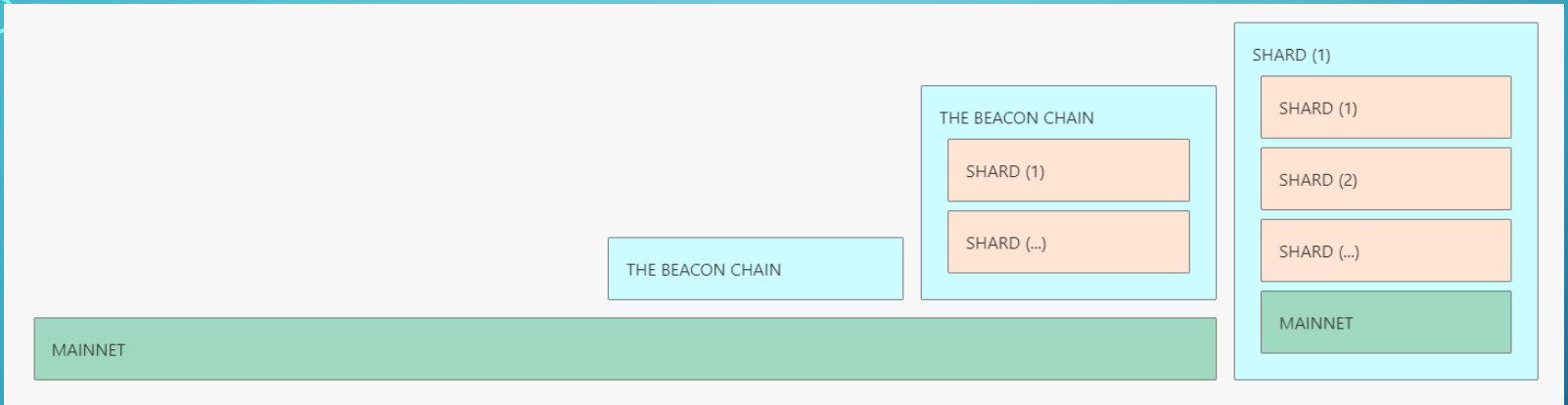


Figure source: *https://ethereum.org/en/eth2/*

- **Eth 2.0 advancement in phases via upgrades on mainnet**
- **Currently in Phase 0 – the Beacon chain**
- **Essentially an alternate chain for managing validators (stake, rewards, penalties) and their block attestations**
- **No current change in Ethereum mainnet operation**
- **Future goal: addition of shard chains – splitting mainnet storage and execution in order to scale horizontally**

CONSENSUS IN BLOCKCHAIN
ETHEREUM 2.0 – POS BASED

Some more info on Eth 2.0 economics:

- **Users can become validators after staking 32 ETH (or more)**
- **One-way transaction to a deposit contract on the Beacon chain**
- **A validator can be chosen to create a new block**
- **Other validators have to attest a proposed block (no mining required)**
- **All validators get rewards for their work – a sliding scale based on the total yearly amount of staked ETH is used**
- **If validators go offline they can receive penalties**
- **If validators validate a malicious block lose their stake**

CONSENSUS IN BLOCKCHAIN
ETHEREUM 2.0 – POS BASED

Some more info on Eth 2.0 PoS:

- **A shard block requires a *committee* attestation (128+ validators)**
- **The committee must propose and validate one block in a given time frame – *slot* (12s); 32 slots form an *epoch* (~6.4 min)**
- **A new committee is randomly formed after each epoch**
- **After a proposed shard block gets sufficient attestations a *crosslink* is created to confirm inclusion in the Beacon chain**
- **First block in an epoch is considered a *checkpoint* block**
- **When voting for a block in a slot, validators must also vote/reference the most recente checkpoint block**
- **A block is considered *justified* when 2/3 or more of the next epoch committee members reference their checkpoint block**
- **A block is considered *finalized* after the same amount of checkpoint refs in the following epoch after justification**

CONSENSUS IN BLOCKCHAIN
PROOF-OF-SPACE/TIME

Proof-of-Space idea: use storage capacity instead of computational power as in PoW

Variations:

- **Proof-of-Capacity – use a challenge that is memory hard in terms of solving instead of computationally hard as in PoW**
- **Directly provide rewards for information storage – i.e., „the challenge" is providing storage space for some real data**

Filecoin

STORJ

CONSENSUS IN BLOCKCHAIN
PROOF-OF-SPACE/TIME

Proof-of-Time idea: block validator chosen based on waiting a random duration of time instead of providing PoW

- **Must guarantee randomness of chosen time**
- **Must guarantee that peer is actually waiting for that time**
- **Proposed way of implementing: using Intel SGX**
  - **Secure enclave memory space not possible to tamper by node owner (but node owner can communicate to the enclave)**
  - **Can execute trusted code that would implement the waiting**
  - **First node to complete the random wait would win**

CONSENSUS IN BLOCKCHAIN
PROOF-OF-AUTHORITY

Proof-of-Authority idea: use approved validator accounts as authoritative chosen block proposers

- **Validators stake their identity (reputation) instead of currency**
- **The actual real identity must be confirmed by the validator and verifiable**
- **Procedure for establishing authority for an identity should be uniform for all candidates**
- **Getting approval to be a validator should be difficult and losing it due to misbehaving should severaly impact reputation (some systems – i.e., POA Network ask for obtaining an actual public notary license)**
- **Argument in favor vs PoS: an apparently high wealth stake might be just a tiny part of the real peers fortune, while compromising the real identity can have a much more serious impact**

Notable implementations: Kovan and Rinkeby test Ethereum platforms, Ethereum integration with Microsoft Azure

CONSENSUS IN BLOCKCHAIN
OTHER BFT CONSENSUS VARIATIONS

So, is there really no use for classic BFT consensus in blockchain?

Theoretically this can scale better in permissioned blockchains:

- **Smaller number of peers**
- **Access control based membership**

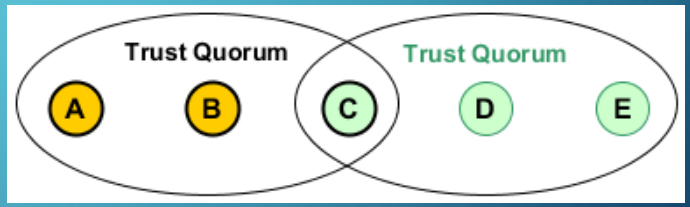
Significant examples:

- **IBM's Hyperledger Fabric – Smart BFT (PBFT variation)**
- **Consensys' Quorum – Istanbul BFT (PBFT variation)**
- **Both solutions also offer also modules for simple CFT consensus (assumption reasonable for small groups of trusted peers with membership controled access)**

CONSENSUS IN BLOCKCHAIN
OTHER BFT CONSENSUS VARIATIONS

Federated BFT consensus – main idea:

- **Peers group in smaller quorums of trust, achieving consensus locally**
- **Consensus propagates due to intersection of quorums (in some sense close to Paxos' condition of acceptors quorum intersection)**



- **No need for maintaining a global membership list (typically needed for various reasons in classical PBFT variations – i.e., for signatures verification)**
- **Significant implementations: Ripple (essentially a distributed not-decentralized consensus based ledger) and Stellar (Ripple fork)**