BlockFin

A Fork-Tolerant, Leaderless Consensus Protocol

September
2020

The Essence of Blockchain

Decentralized public ledger. *Permissionless*. *Censorship* resistance

- Practical deployments are often centralized, permissioned, and offer no guarantees on censorship resistance
  - Examples: EOS with 21 validators. NEO with 7 validators. Both are permissioned
  - Cardano is currently fully centralized. Ripple is private blockchain
- Poor decentralization leads to possible censorship
  - Nodes can *selectively* reject transactions from certain users/regions or make it harder for others to join the network
- Most deployments are terribly inefficient and slow
  - Bitcoin: 7 txs/sec. Ethereum: 15-20 txs/sec. Dash: 48 txs/sec. Cardano: 10-15 txs/sec
- The ones that are *fast* are centralized, private, or not Byzantine tolerant
  - EOS: 50K txs/sec, centralized. NEO: 1K txs/sec, centralized, not Byzantine tolerant. Ripple: 1.5K txs/sec, private blockchain
- Most implementations *serialize* the block creation globally, thus limiting throughput
  - Serial block creation is necessary to address *double-spending*
  - Leader or delegation-based consensus forces one node or a delegation creating *one block at a time*

Typical Solutions and Their Drawbacks

- *Sharding* - Allows parallel block creation in multiple, *independent* shards
  - Works great if the shards can *remain* independent
  - But, too complex to manage, especially if cross-shard communication is required
  - If some shards grow disproportionate to others (and they *will*), shard *balancing* is required
- *Private Payment Channels* - Point-to-point *private* states are not registered in the blockchain
  - Enables instant transactions between participating nodes
  - Only the *final* state is registered in the blockchain
  - Works well in certain use cases, but transactions inside private channels are invisible and not traceable
- *One Chain per Account* - Each account is a separate blockchain
  - Nano uses this structure. Transactions are instant because the chain is private to the account
  - Works well generally, but it is hard to get *global* state of the blockchain
- But, there are no known solutions for true decentralization
  - Current mindset with leader or delegation-based consensus makes decentralization nearly impossible
  - Block rewards are *competition*-based, which result in eventual centralization because early adopters will get more powerful over time

BlockFin - Throughput and Decentralization

- Parallel block assembly and validation
  - All validators *concurrently* assemble and validate multiple blocks *together*
  - Block creation is not a *serial* process. This results in significant improvements in throughput
  - No wasted efforts to resolve chain forks because fork-tolerance is built into the protocol

- Separate *voting* and *counting* phases
  - Typical consensus protocols require all participants *agree* on the *same value* in the *same round* or within *finite* time. In asynchronous systems with large number of participants reaching consensus is very difficult
  - In BlockFin, all validators *vote* in one phase and *count* the votes in a *separate* phase. They are not obliged to *agree* in one round or within a finite time
  - Validators work on *multiple* blocks voting on some and counting on already voted blocks in a pipelined flow

- Account-based transaction model
  - The transactions recorded on the blockchain are also *referenced* in the affecting accounts
  - Both *committed* and *pending* transactions are available in the account, so account balance can be computed instantly
  - Transaction *ordering* is not necessary, which simplifies consensus protocol design
  - Improved security because transactions are recorded both on the blockchain and the respective accounts

- True decentralization
  - *All* validators are required to participate in the consensus. There is no leader or delegation elected
  - *Community* consensus instead of *competitive* consensus. *All* validators earn block rewards proportional to their stakes for *all* the blocks

BlockFin - Minimizing P2P Overhead

- Two-level Peer-to-Peer Network
  - With large number of validators consensus via peer-to-peer communication is nearly impossible due to large number of messages (N^2 complexity)
  - BlockFin organizes validators into a two-level peer-to-peer network - a) control network and b) consensus network
  - Control network involves *all* validators. This is used for health check and other housekeeping purposes
  - Consensus network uses a *subset* of validators that host a peer-to-peer, replicated message store

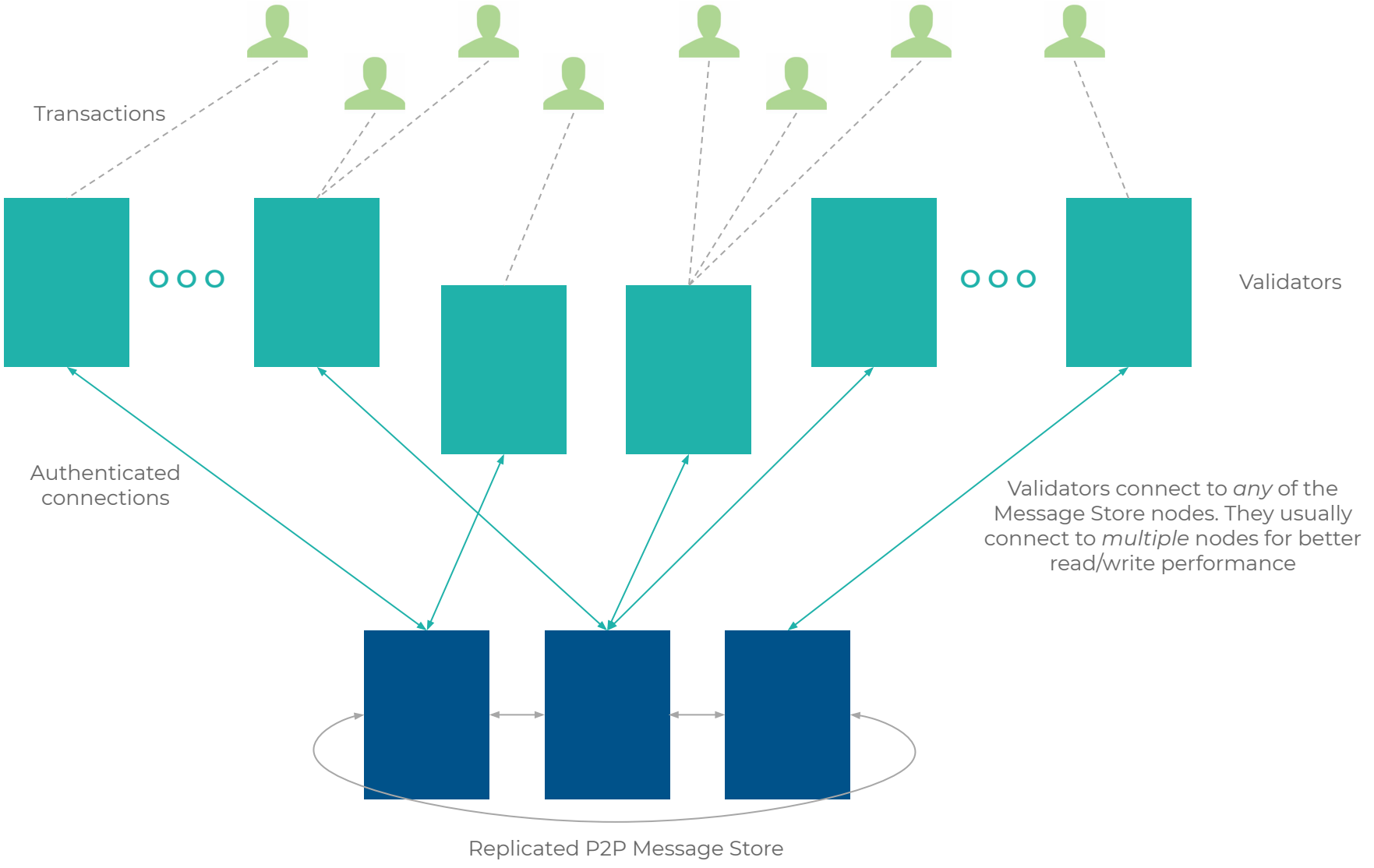
- Shared state instead of private state
  - The validators *don't* manage their state *privately* in their own instances. They use message store to manage their shared states
  - The validators *sign* their states with their *secret* keys, so adversaries cannot override or manipulate their states (decisions)
  - The validators read from and write to message store nodes instead of sharing their states directly with their peers. This improves consensus performance significantly because it reduces complexity from N^2 to Nm complexity where m is the number of reads and writes per validator and $m << N$

- Trust Issue
  - When a new validators join the network or existing validators come back online after sporadic downtimes, the validators don't have to *trust* arbitrary peers for blockchain data. The message store provides that support
  - The data *sync* is required if the messages store nodes go offline, but this separation between consensus protocol and message replication allows us to choose best of the breed solutions

BlockFin - Minimizing P2P Overhead

- Liveness and security
  - A two-level network makes it possible for a large number of validators participate in the consensus process, which would be impractical otherwise
  - The message store implements *authenticated* access, so only *signed* validators can connect to the store
  - However, since message store consists of a *subset* of validators the *liveness* and *security* of the network depends on it. A compromised message store compromises whole blockchain
  - The message store therefore needs to defend attacks on it. Possible attacks are data destruction, man-in-the-middle attack, collusion attack, DDoS, and attacks against data repair. The message store is designed to defend these attacks
  - Technology alone cannot guarantee liveness and security in distributed systems
  - All nodes, validators and message nodes alike, are *incentivized* to *follow* the protocol. The incentive structure doesn't favor *winning* of any kind so there is no benefit to be gained by *defeating* the system

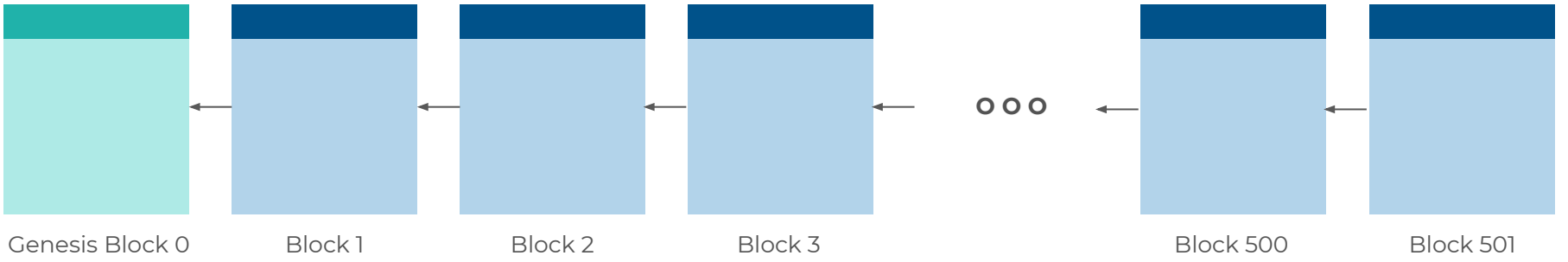
BlockFin - Two-Level P2P Network



BlockFin Consensus Process

BlockFin - Consensus Process

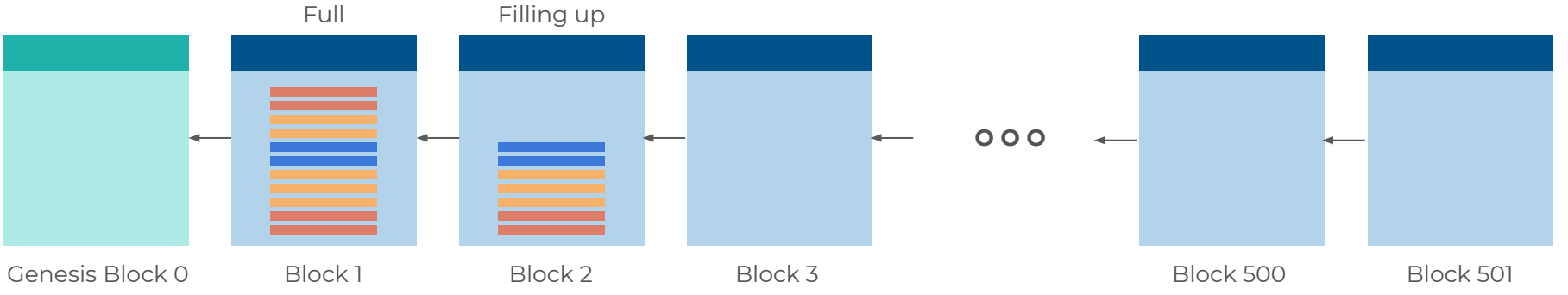
- Block creation vs block assembly
  - In traditional blockchains, the block producer *creates* a new block with transactions and *publishes* it to the blockchain
  - In BlockFin *empty* blocks are created at genesis time with some metadata, so *empty blockchain* exists at inception time. This ensures that the *structure* of the blockchain (block N linking to block N-1, etc.) is intact
  - In BlockFin blocks are *assembled* with transactions as validators receive them. As the empty blocks fill up, new empty blocks are added to the chain as needed in the future
  - Since empty blocks merely ensure the structure and *continuity* of the blockchain, any validator can add them to the chain when they see existing empty blocks are being filled up



Empty blocks waiting to be assembled

BlockFin - Block Assembly

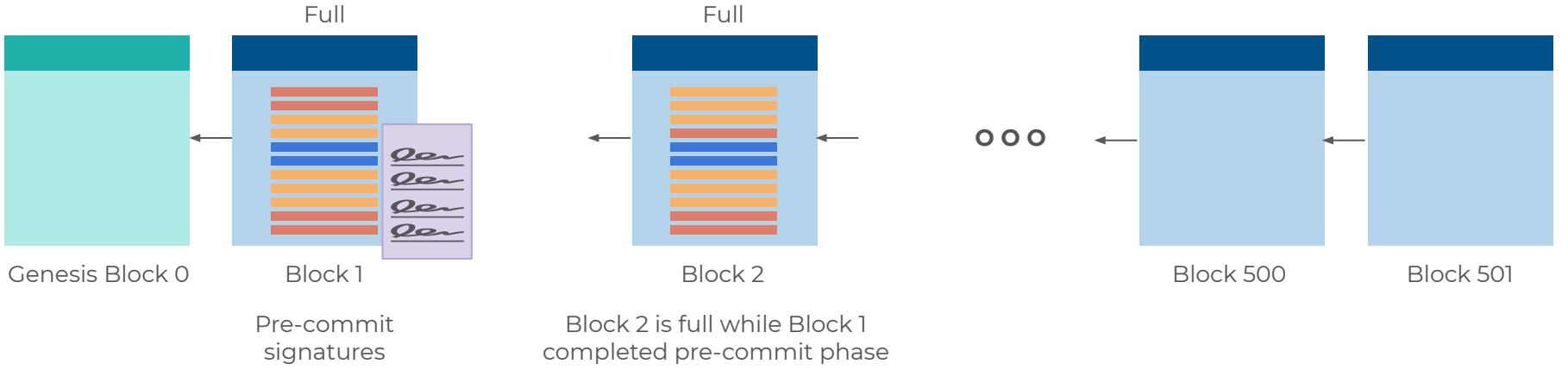
- Block assembly
  - *All* validators receive transactions from clients. All validators *assemble* incoming transactions into *next* available *empty* block, after performing basic validation of transactions. Unlike traditional blockchains, they don't broadcast transactions to other validators
  - When an empty block *fills up*, the validators move to the next empty block. The capacity C of the blocks varies from block to block depending on incoming transaction rate across all validators
  - Since validators assemble transactions concurrently and asynchronously *without* any *coordination*, they may overshoot C. The protocol allows it
  - The validators *read* the block capacity before *sending* the transaction to it, so the overshoot can happen for a duration of time required for C to *stabilize* among all message nodes
  - Depending on the incoming transaction rate, the empty blocks can fill up quickly. Validators don't wait for the filled up block to be *approved* before moving to the next block, so *multiple* blocks could be assembled in a short period of time



Multiple blocks could be assembled in a short period of time

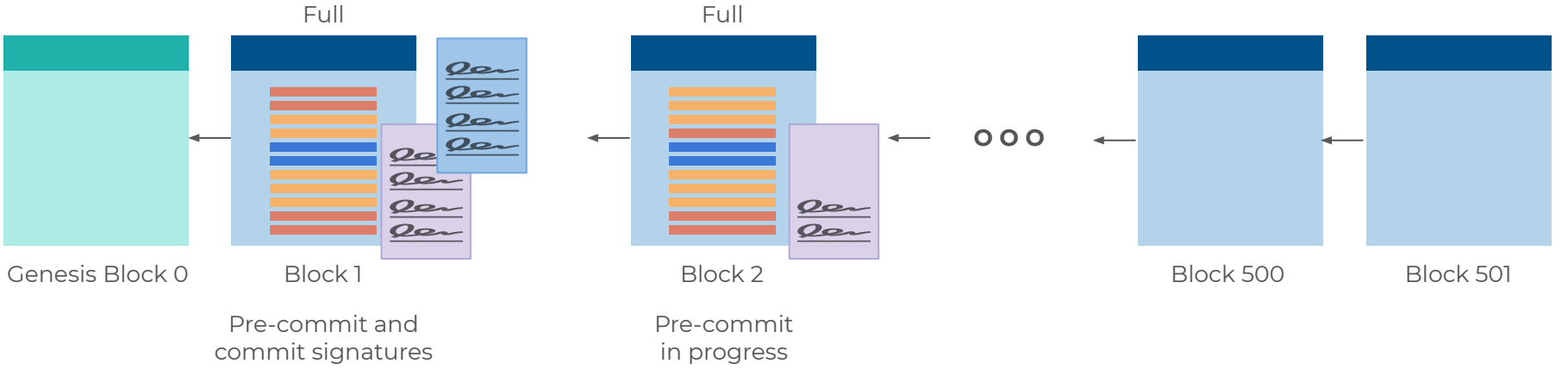
BlockFin - Validation Phase 1, Pre-commit

- All *Full* blocks go through a two-phase block validation process
  - *All* validators participate in *both* the phases
  - In pre-commit phase (phase 1) all the validators verify *all* the transactions included in the block. This is required because transactions are added to the block by *multiple* validators so all validators must ensure that no adversarial behaviors exist in the block
  - The validators sign (Ed25519 signature scheme) the block *conditionally* or *unconditionally*
  - *Conditional* pre-commit happens if some or all transactions fail verification. All validators create the *list of failed transactions* with their conditional pre-commit signature
  - *Unconditional* pre-commit is desired outcome where all transactions pass verification
  - The block advances to phase 2 as long as it gathers more than ⅔ (to qualify for Byzantine tolerance) conditional or unconditional pre-commit signatures
  - A block never fails or gets deleted from the blockchain even if all the transactions in it fail verification



BlockFin - Validation Phase 2, Commit

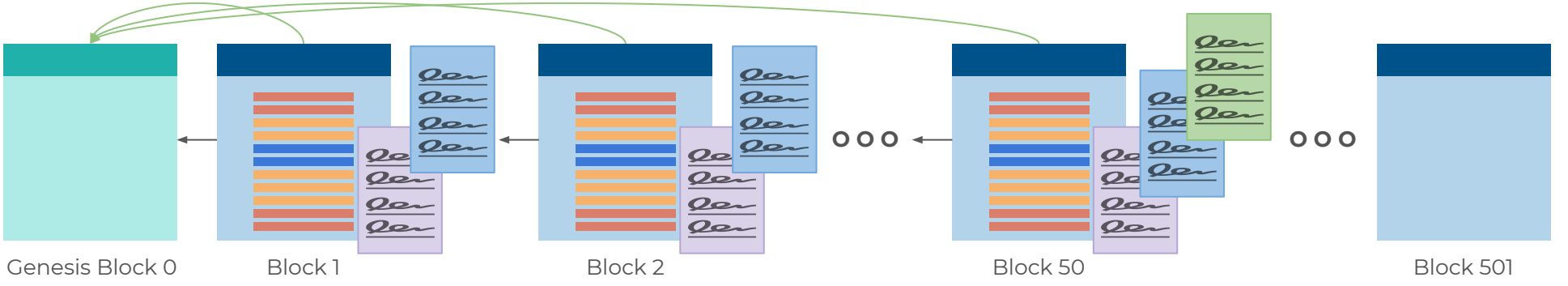
- Commit the block and execute transactions
  - *All* validators look for pre-committed blocks with more than ⅔ pre-commit signatures
  - The validators verify pre-commit signatures and countersign the block with the result of signature verification
  - This phase serves as *counting* phase because the validators count the pre-commit signatures
  - The commit is *conditional*, if the pre-commit is conditional
  - If the block gathers more than ⅔ commit signatures, the block is committed automatically. By extension the transactions included in the block are also automatically committed
  - If the commit is conditional, the *failed* transactions are automatically declined
  - The transactions are *finalized* (approved or declined) after the block is committed
  - Clients are expected to *wait* until the block is *committed* to decide on the *fate* of the transactions, including failed transactions



BlockFin - Block Sealing

- Block *Sealing* and detect adversarial behaviors
  - Protocol violations are possible. The violations can be malicious or unintentional
  - Periodic reconciliation is required to ensure that account balances are computed correctly
  - Block *sealing* accomplishes this. Every N* (50 here) *committed* blocks go through the sealing process
  - All validators participate in sealing process. They start from *last* sealed block and perform *exhaustive* verification of all *committed* blocks for protocol violation
  - The validators *sign* all the committed blocks *individually* after completing protocol violation checks
  - If the committed block gathers more than ⅔ signatures for sealing, it is considered *sealed*
  - A *link* to last sealed block (block #0 here) is created to indicate the *basis* on which sealing is done
  - The *last* of the committed blocks (block #50 here) in the group serves as *last sealed block* for the *next* group of committed blocks in the future

Committed blocks are *linked* to last *sealed* block, improving the security of the blockchain



N committed blocks effectively form a *fork* on the *last sealed* block and yet, part of the main blockchain. There is no wasted work

* Network decides N based on incoming transaction rate

BlockFin - *Implicit* Consensus and Finality

- Implicit consensus
  - BlockFin protocol doesn't have *explicit markers* for the completion of pre-commit, commit, and seal phases. This is because doing so requires leader election
  - The validators *never collectively agree* on the block state because of the impossibility of doing so in large networks
  - *Implicit decision* involves protocol-following validators *decide* on the state of the block depending on signatures collected. For example, when validators look at a block that has ⅔ + *valid* pre-commit signatures they *agree individually* that the block has completed pre-commit phase
  - By "***Agreement*** - If an honest node proposes a value *v*, then all honest nodes agree with *v*", we can conclude that if an honest validator agrees on the block state, then all honest validators agree with the same state
  - Implicit decisions lead to *implicit consensus*. This is powerful because even clients can choose to follow the BlockFin protocol and *implicitly agree* on the block state and hence the transactions included in it

- Implicit finality
  - Implicit consensus leads to implicit finality for both the blocks as well transactions included in them
  - A *committed* block is *finalized* in that it is *irreversible*. Any attempt to reverse or alter it by an adversary fails the *agreement* property described above
  - By extension, the transactions in a finalized block are also *automatically* finalized. The individual transactions may be *approved* or *declined* specifically, but they are finalized nevertheless

BlockFin - *Explicit* Finality with Sealing

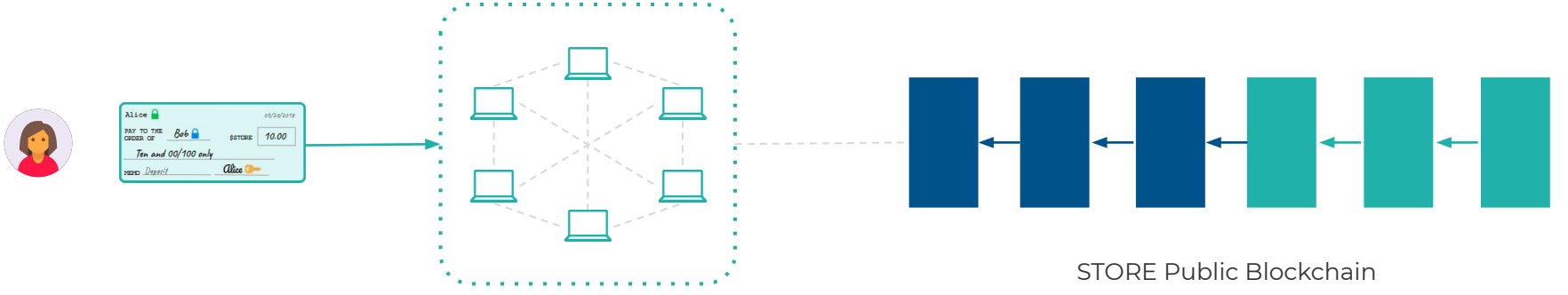
- Block sealing provides explicit finality
  - Block sealing is a slower, asynchronous, and batch-oriented phase, so its completion time is unpredictable
  - Until committed blocks are sealed, implicit consensus and finality are used by validators and clients alike. This calls for extra work, but only until the blocks are sealed
  - After the blocks are *sealed*, block and transaction finality are explicit
  - Whether explicit or implicit finality is used, the blocks are *irreversible* as soon as they are *committed*. The explicit and implicit finalities are just different ways to make the same decision

Life of a Transaction with BlockFin

Life of a Transaction with BlockFin

1. Alice installs STORE wallet and generates a secret key, public key, and wallet address for her.
2. Alice looks up Bob using Bob's address. Alice wants to send 10 $TORE* tokens to Bob.
3. Alice creates a transaction for 10 $TORE tokens and signs it with her secret key.

4. Alice submits the transaction to STORE's decentralized network of validators

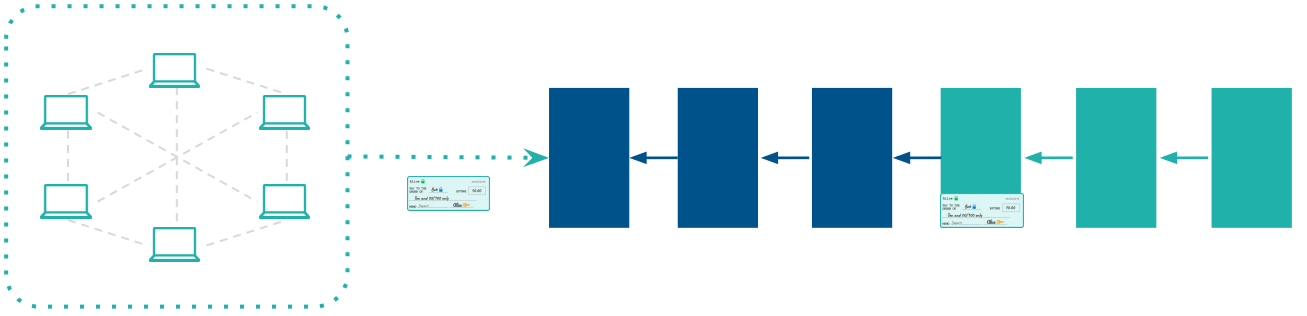


STORE Public Blockchain

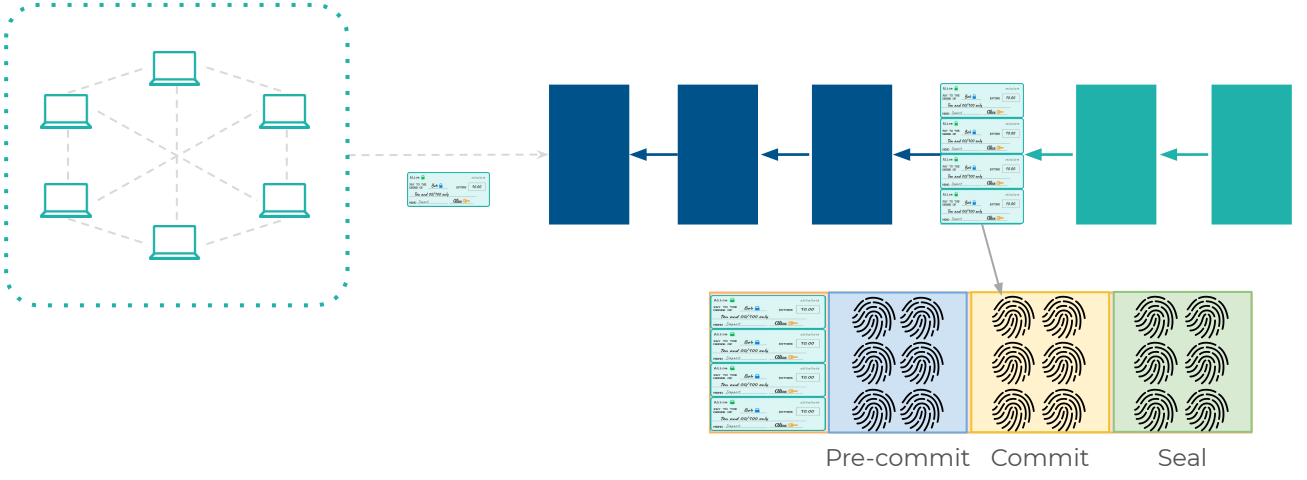
* For brevity, we assume that Alice has funded her account

Life of a Transaction with BlockFin

5. The receiving validator queues the transaction into the next available *free* block along with transactions received by other validator nodes

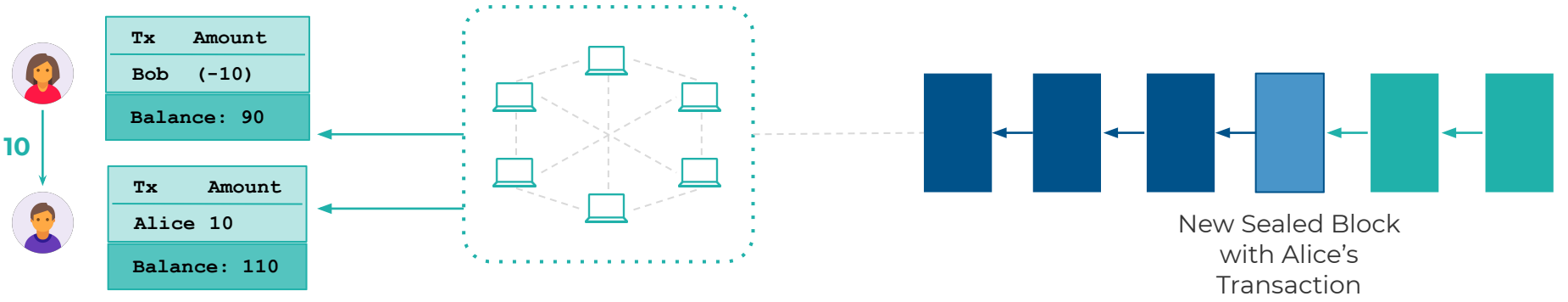


6. When the block gets *full* with transactions, the validators approve it with a 3-phase validation process. In each phase the validators *sign* the block with their *approval*



Life of a Transaction with BlockFin

7. Alice's account is debited with 10 $STORE tokens and credited to Bob's account



BlockFin Properties

BlockFin Properties

- Fork-tolerant
  - Multiple *committed* blocks technically create a fork on *last sealed* block, but are still part of the main blockchain
  - BlockFin uses the formation of chain forks to its advantage to improve throughput
  - There is no *throw-away* work because of *longest chain wins* or similar constructs

- Leaderless
  - *All* validators participate in *all phases* of block validation. The only rule is that they cannot perform multiple phases on the same block in the same round
  - Signature scheme is used to determine the completion of individual phases. There is no leader or delegation elected to vote or count
  - There will be lot of signatures collected per block, but it is a small price to pay for true decentralization

- Community Consensus
  - Leaderless property results in community consensus. Entire community takes responsibility of upkeep of the blockchain instead of a individual leaders or delegations
  - There is no advantage in *defeating* the system because there are no winners elected for block creation. The protocol is incentive compatible Nash Equilibrium such that deviating from the protocol doesn't result in net gain
  - Block rewards are shared by *all* validators proportional to their stake in the system

BlockFin Properties

- Censorship resistance
  - Community consensus discourages censorship (such as selective rejection of transactions) as other validators notice it immediately
  - Block sealing phase catches misbehaving validators for slashing and other punitive measures

- True decentralization
  - *All* validators participate in *all phases* of block validation
  - Collusion is impossible due to community participation

- Permissionlessness
  - Except for staking and KYC/AML (know-your-customer/Anti-money-laundering) requirements, there are no special privileges to join the network as validators
  - There are no discriminatory policies that grant special privileges to certain classes of validators

- High throughput
  - Parallel block assembly and validations results in higher throughput
  - Fork-tolerance avoids unnecessary *serialization* of block creation, so validators don't have to work hard to decide on the *longest chain*, etc.

FAQ

How do all the nodes know that they have *N* transactions replicated to them before starting block validation?

- In BlockFin, the transactions (messages) are *not* replicated to all peers, but they are recorded in the blocks in the P2P message store. The nodes don't maintain their *private* replicated state machine *locally* but have *shared* state maintained in the P2P message store
- The nodes do one *read* to find the block to add the transaction to and then do a *write* to add the transaction. Each node performs 2 operations per transaction in the best case. *Multiple* reads may be required in boundary conditions
- The nodes read from the P2P datastore in every *round* to determine if the block is ready for validation. In best case scenario, nodes do one *read* in a round, followed by one *write* with their *signed decision*
- The block validation happens on *rolling* basis because not all nodes would know about the block-readiness at the exact same time. So worst case scenario requires *multiple* reads followed by a *single* write
- The nodes are not obliged to decide *concurrently* in the *same* round. When a node makes a *decision*, it adds its *signature* with its *decision*. If *sufficient* signatures are collected (quorum) the consensus is reached *automatically*

Does BlockFin use PBFT, Zyzzyva, or similar protocols?

- As described above, BlockFin doesn't implement replicated state machine with *private* states but has a *shared* state where each node *signs* its entry in the shared state
- BlockFin uses account-based transaction model and not UTXO, where each account also maintains a list of transactions from and to it and *current* balance. This alleviates the need for *global ordering* of transactions. So, schemes like the above are not needed

FAQ

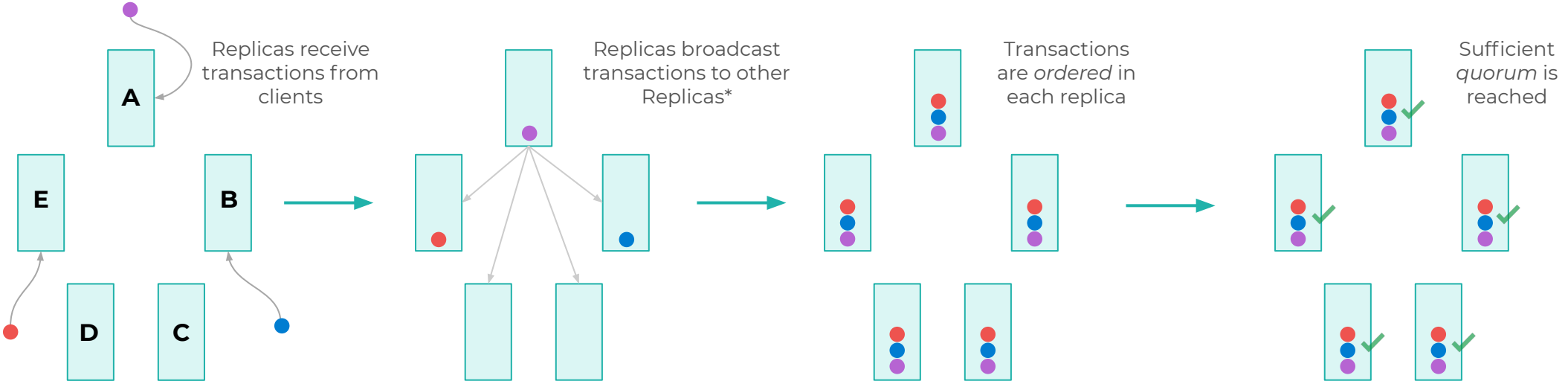
What happens when the validator set changes? How do you prevent a node from signing older blocks?

- In BlockFin, *all* validator nodes are required to participate in the consensus. There is no leader or delegation elected to validate a block
- When new validator nodes are added, N changes. This is a global count that every node is aware of, so f, the maximum faulty nodes allowed is calculated automatically using N = (3f + 1) for Byzantine tolerance
- The nodes cannot rewrite *history* without being noticed by all other nodes. All writes require a signature from the node performing the write operation, so if a node writes to a *sealed* block, it gets noticed by all others. BlockFin instruments *slashing* for misbehaving nodes
- If someone acquires private keys of old validators who are no longer on the network, their write operations will similarly be noticed by all other nodes

BlockFin vs Other Consensus Protocols

Typical Consensus Process

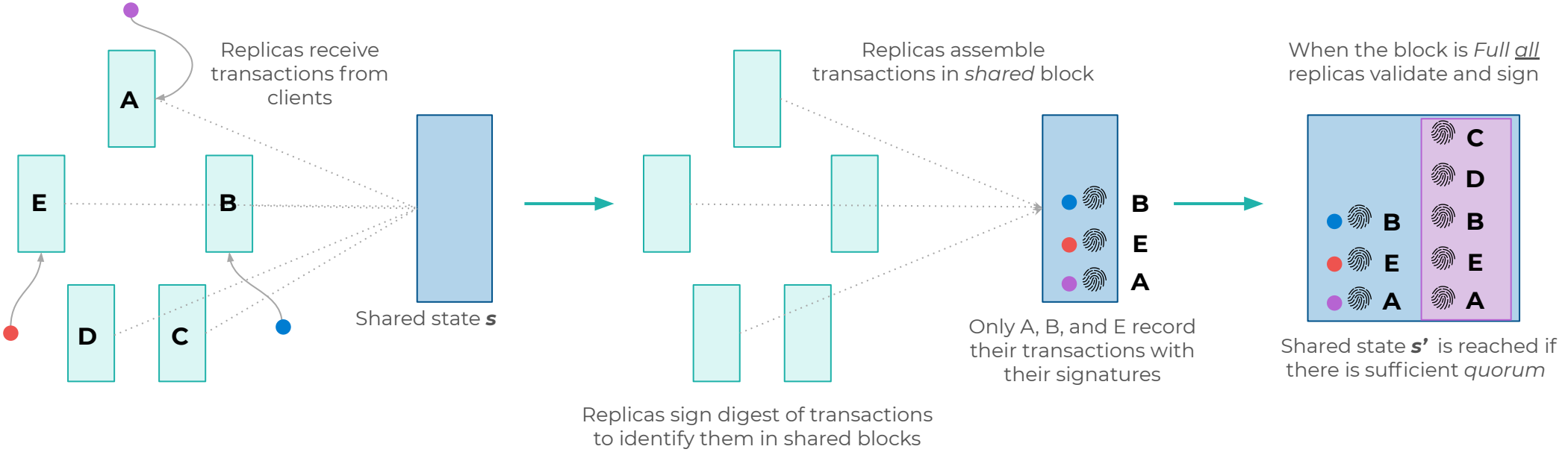
All replicas (validators) maintain their *private* states *locally*. The replicas *broadcast* their states to their peers to *reach* consensus



* For brevity, only the broadcast from replica A is shown here. Replicas B and E work similarly

BlockFin Consensus Process

All replicas (validators) maintain their *shared* states *publicly* protected by their signatures. The replicas *sign* their decisions in a 2-phase block validation process



BlockFin Consensus vs others in the wild

BlockFin Consensus	Other Consensus Protocols
Replicated State Machine: • Uses a hybrid P2P and Client-Server model to maintain *shared state* for participating replicas • Each replica *signs* its entry in shared blocks to identify itself and its decisions • The shared state is maintained in a *replicated* P2P datastore for resiliency and throughput • Replicas read from and write to shared state in P2P datastore • Replicas make shared decisions based on the shared state	**Replicated State Machine:** • Each replica maintains its own *local state* • Replicas use broadcast/gossip message primitives to share their local states with other replicas • Replicas make *local decisions* based on their local state
Communication Complexity: • Best case scenario is estimated to be (4N + 2m) *per block* where N is the number of replicas and m, the number of transactions in the block • There are 3 primary stages in BlockFin validation -- a) add transaction to the block, b) pre-commit, and c) commit. Adding each transaction involves a read and a write, resulting in 2m operations. Pre-commit and commit are block level operations and each phase involves a read and write operation per replica, resulting in 4N operations • Worst case scenario requires *several* reads, followed by a write for each stage. This is not modeled yet, but still will be orders of magnitude lower than N^2 • Only the replicas that receive transactions add transactions to the block, so there is no unnecessary data duplication and resulting communication overheads among all replicas. This is the primary reason for the low complexity in BlockFin	**Communication Complexity:** • Usually N^2 where N is the number of replicas • Some implementations optimize this number for non-adversary scenarios or optimize the message *size* for speed • As N grows, so would the complexity. In practical Byzantine setup, consensus may be harder to achieve when number of replicas is large • This complexity leads to delegation election for block finality, thus leading to highly centralized deployments in practice

BlockFin Consensus vs others in the wild

BlockFin Consensus	Other Consensus Protocols
New replicas joining the network: • The new replica connects to the P2P datastore and is up-to-date *instantly* because of the shared state • It can receive transactions and participate in block validations instantly for the same reason • The replicas can go offline sporadically without affecting their state. When they are back online, they can start their activities without any *sync* required • The nodes in P2P datastore can also go offline sporadically. They will sync to live state when they are back online	**New replicas joining the network:** • The new replica must *trust* one of the existing replicas to build its local state. It must download GBs of data (Bitcoin's blockchain is ~149GB and Ethereum's is ~57GB) before building its local state • The replicas must continually *sync* with each other to remain *current*
Attack Vectors: • The shared state makes P2P datastore target for DDoS attacks. But P2P datastore is replicated so there is no SPF • History rewinding (long range attack) is difficult because the entire blockchain is *public* all the time. Once the blocks are sealed, any attempts to modify them will trigger notifications to all replicas. Since each operation has replica's signature, it is easy to identify the offending replica	**Attack Vectors:** • Since practical implementations tend to be centralized (EOS for example has 21 *known* validators) they become targets for DDoS • If a replica has enough hash or cash power, it can mount long range attack because the blocks are built *privately* and *published* to the blockchain
Decentralization: • Practical deployments are truly decentralized because all replicas participate in block validation • Liveness is ensured via incentivization with block rewards. All nodes win portions of block reward for all blocks because BlockFin is a *community (leaderless) consensus* protocol. The incentivization ensures that at least ⅔ +1 replicas are online all the time	**Decentralization:** • Theoretically true decentralization is possible, but practical implementations have been highly centralized • Nondeterministic block rewards incentivize poorly • *Competitive (leader or delegation) consensus* models end up being highly centralized because stake-holders get bigger over time making it difficult for small players joining the network