Tsampi BFT: Leaderless One-Round Voting with Parameterized Finality

Preprint, compiled August 17, 2026

Tim Watts
tim@readevalprint.com

Abstract

Tsampi is a leaderless Byzantine fault-tolerant state-replication protocol. Any eligible Validator can collect one round of Votes for an exact parent and propose a single-parent Block; nodes then rank competing proposals deterministically. Its dual-linked Vote chain records both Block-lineage causality and each Validator's ordered Vote history. Within one CoveredBlock interval, each successive endorsement must target a strictly better proposal.

A strict supermajority has more than two thirds of the Stake in the Validator set used for a Block. Q1 is the strict-supermajority Vote pass that sets QuorumBlock. Q2 is the later, independently deduplicated strict-supermajority Vote pass that sets CoveredBlock. Q1-closing Votes do not count toward Q2. Q2 may use the same Validators, but only through later Votes carried by later Blocks.

Proposal admission is parameterized independently of finality. Under an unchanged Validator set and an endorsement threshold of at least two thirds, genesis reaches QuorumBlock in Block 2 and CoveredBlock in Block 3; the steady-state pipeline keeps a one-Block lag between them. At lower thresholds, later Blocks may accumulate unique Stake, but a fixed minority eventually stalls.

Machine-checked proofs establish fixed-Validator-set safety, evidence completeness, evidence soundness, and conditional safety across Validator-set changes in a normalized post-authentication model.

1. The idea

Leaderless means that Tsampi selects no Validator before proposal creation. Any eligible Validator can collect one round of Votes for an exact parent and create a Proposed Block. The protocol later chooses among valid Proposed Blocks with one deterministic order.

The protocol separates three decisions that many BFT protocols combine:

1. Who may propose? Any eligible Validator with enough Votes for the parent Block.
2. Which Proposed Block should a node prefer? The deterministic Block order answers this from the node's observed set.
3. When is a Block final? Later Blocks must first set QuorumBlock and then set CoveredBlock for that same lineage.

Leaderless proposal creation is not itself new. Tsampi's narrower distinction is that one round can admit several single-parent proposals, while deterministic choice and the dual-linked improvement rule keep later voting on an ordered path toward exact-lineage finality.

The endorsement threshold is also separate from finality. A low threshold allows a Proposed Block with less immediate coordination. It does not lower the Stake required to set QuorumBlock or CoveredBlock.

2. Model and assumptions

A **Validator** has positive **Stake**. A Block fixes the Validator set and Stake used to verify Votes that endorse that Block. A later Validator-set change does not reweight an earlier Vote.

A **Block** is an authenticated state transition with one parent Block and one proposer. A **Proposed Block** is a candidate child of one exact parent. A **lineage** is one exact parent chain. Two Blocks form a **fork** when neither is an ancestor of the other.

The protocol has only three message types: **Vote**, **Block**, and **Tx**. A Proposed Block is a Block in the proposal stage, not a separate message type. Votes carried by Blocks directly establish admission, QuorumBlock, and CoveredBlock; Tsampi has no separate certificate object or certificate message.

A **Vote** is a signed statement by one Validator. It has three logical parts:

- **PrevVote** links the Vote to the proposer Vote required by the parent relationship.
- **Endorsement** binds the Vote to one exact parent Block, one Validator, and that Validator's Stake.
- **PrevSelfVote** links a Validator's new Vote to its last accepted Vote while CoveredBlock is unchanged.

This is a dual-linked Vote structure. PrevVote links each Vote to the previous Vote required by the Block's parent relationship, establishing Block-lineage causality. PrevSelfVote links it to the preceding Vote by that same Validator, establishing a total order over that Validator's signed Votes within one CoveredBlock interval. Together the two links prove where a Vote belongs in both the Block lineage and its Validator's Vote history; they do not impose one global order across different Validators.

The safety results use four assumptions:

- **Signatures:** A signature identifies its Validator, binds the exact Vote and Endorsement, and cannot be forged.
- **Validation:** Correct Validators apply the same Block, Vote, Endorsement, lineage, and Stake rules.

- **Validator set:** Each Block fixes the Validator identities, keys, Stake, and total Stake used by later rules that name that Block.
- **Fault bound:** At least two thirds of the Stake in every applicable Validator set is honest and online. The remaining Stake may be Byzantine, offline, or silent.

Safety does not require timely delivery. Messages may be delayed, reordered, duplicated, or withheld.

3. PROPOSED BLOCKS AND THE ENDORSEMENT THRESHOLD

Let `W` be the total Stake in the parent Block's Validator set. Let `Q` be the Stake of the unique Validators whose valid Votes endorse that parent. Let `p` be the endorsement threshold, with `0 < p <= 1`.

A Proposed Block is valid only if:

- it names one exact parent Block;
- its proposer is an eligible Validator for that parent;
- it carries one valid proposer Vote;
- every carried Vote and Endorsement is valid for that exact parent;
- no Validator appears twice; and
- its state transition is valid.

The threshold rule is simple:

- For `0 < p < 1`, the Proposed Block requires strict `Q > pW`.
- For `p = 1`, the Proposed Block requires exact `Q = W`.

Equality at a fractional threshold fails. Equality at one means every Validator voted because every Validator has positive Stake. Duplicate Votes never add Stake. A carried Stake claim never overrides the Stake fixed by the parent Block.

The proposer Vote is part of the same Vote set. Its signature and Stake give the Proposed Block a bounded proposer score. The score helps order competing Proposed Blocks; it does not change the endorsement threshold or the finality threshold.

3.1 PrevSelfVote

Each Validator has one accepted PrevSelfVote chain while CoveredBlock is unchanged. The first Vote in that interval has no PrevSelfVote. Every later Vote must name the Validator's exact accepted predecessor.

A Validator may Vote for every valid Proposed Block it receives, provided that each new Vote targets a strictly better Proposed Block than its previous Vote under the complete deterministic order. PrevSelfVote identifies that exact predecessor and makes the improvement check verifiable. The rule permits Votes for successively better competing proposals; it does not limit a Validator to one proposal or one replacement Vote.

Every Vote endorses one exact Proposed Block. There are no empty Votes. A valid signed Vote is never revoked, even if another Proposed Block later wins; it remains an authenticated endorsement of its original target. Votes for losing proposals are not moved or reused on the winning lineage. A byte-identical Vote received again is only transport replay, not a new Vote.

Missing PrevSelfVote data makes the Vote unavailable. A new Vote that does not improve on its predecessor is invalid. A Validator cannot return to a weaker target or issue a distinct Vote for the same target.

A Validator may Vote across many competing Proposed Blocks, but one Proposed Block cannot carry two Votes from the same Validator. It cannot carry both one Validator's PrevSelfVote and that Validator's successor as separate Stake.

PrevSelfVote is a live admission rule. Before a Validator signs or releases a new Vote, the complete rooted predecessor chain must be available and valid back to the current CoveredBlock interval. Committed replay does not reconstruct that live admission process. It replays the authenticated Votes already carried by committed Blocks.

4. DETERMINISTIC PROPOSED-BLOCK CHOICE

A node chooses among the valid Proposed Blocks it has observed. The candidates must descend from one retained lineage root and have complete parent histories to that root.

The order compares, in sequence:

1. the height of CoveredBlock;
2. the height of QuorumBlock;
3. the fraction of unique Validator Stake recorded for the open progress event, adjusted by the bounded proposer score; and
4. the canonical Block identifier.

The first unequal field decides. The Stake comparison uses exact arithmetic. The final Block identifier breaks any remaining tie.

This is a strict total order. Two correct nodes with the same valid candidate set choose the same Proposed Block regardless of arrival order. Nodes with different observed sets may choose differently until message delivery gives them the same set. Deterministic choice is not, by itself, a liveness proof.

5. QUORUMBLOCK, COVEREDBLOCK, AND EPIC

The protocol counts unique Validator Stake along one exact lineage. Repeated Votes from one Validator do not add Stake to the same pass.

The Epic sequence starts at genesis and advances toward the tip. **Epic** counts completed strict-supermajority progress events since genesis. **EpicHeight** records the Block height that completed the latest Epic. Epic progress, QuorumBlock progress, and CoveredBlock progress are distinct events.

QuorumBlock and CoveredBlock are derived in the opposite direction. For each candidate tip, the protocol follows that tip's exact parent lineage backward. The newest strict-supermajority observation is QuorumBlock. Fresh later Votes in that observation cover the preceding strict-supermajority observation, which becomes CoveredBlock. These are rolling

tip-relative lineage frontiers, not fixed rounds assigned by scanning forward from genesis.

Q1 sets QuorumBlock. It records the newest Block that has a strict-supermajority first Vote pass. Q2 sets CoveredBlock. It requires a second strict-supermajority pass from later Blocks. Q1-closing Votes do not count toward Q2. The two passes deduplicate Validators independently, so Q2 need not be a subset of Q1 and need not be disjoint from Q1.

5.1 Threshold at or above two thirds

When `2/3 <= p < 1`, every valid Proposed Block carries more than two thirds of the parent Block's Stake. Under one unchanged Validator set and continued Block production:

- Block 1 carries the first strict-supermajority Vote set for genesis.
- Block 2 uses that set to set QuorumBlock to genesis.
- The same Votes open Q2 but do not count toward it.
- Block 3 carries later Votes and sets CoveredBlock to genesis.

After startup, the pipeline processes Q2 before it advances Q1 in each Block. QuorumBlock therefore leads CoveredBlock by one Block while these conditions hold.

5.2 Threshold below two thirds

When `0 < p < 2/3`, a smaller Validator group may produce a Proposed Block. Later Blocks accumulate unique Stake on that lineage until Q1 and Q2 each exceed two thirds.

A fixed minority cannot manufacture CoveredBlock. Each new Block must increase accumulated unique Stake until Epic advances. If the same minority repeats the same Votes, progress stops and the next Block fails.

There is no universal Block bound in this mode. One recorded four-Validator schedule first sets QuorumBlock for genesis at height 3 and CoveredBlock at height 5. That is one executable schedule, not a five-Block theorem. Another Vote order may advance a newer QuorumBlock before an older Q2 pass finishes.

5.3 Unanimity

When `p = 1`, every Validator must Vote. One missing Validator stops Proposed-Block production. If Blocks continue under one unchanged Validator set, the same one-Block QuorumBlock-to-CoveredBlock pipeline applies.

6. VALIDATOR-SET CHANGES

A Block output may change the Validator set or Stake. The change cannot authorize the Block that creates it. That Block is verified with its parent's Validator set and Stake. The new Validator set first applies to direct children.

A requested change waits until CoveredBlock reaches the request and a later Epic applies it. Raw Block height, wall time, message arrival order, or local process state cannot activate the change.

Each lineage reads the Validator set and Stake from its exact parent Block. Two forks may later contain different Validator sets. A child on one fork cannot use the Validator set from the other fork.

Historical Votes keep the Validator key and Stake fixed by the Block they endorsed. A later change does not reweight them or replace their verification keys.

A Validator-set change does not reset PrevSelfVote. A Validator that leaves and rejoins while CoveredBlock is unchanged keeps its prior PrevSelfVote chain. A newly added Validator with no prior Vote in that CoveredBlock interval may create one root Vote. Advancing CoveredBlock starts the next interval.

The changing-Validator-set safety theorem is conditional. At the first fork, the two sibling Blocks must derive the same Validator set and Stake from their shared CoveredBlock history. That set applies immediately to each sibling's children. Each branch's finality claim must descend from its sibling and satisfy the fixed-Validator-set safety premises under that common set. The proof establishes safety for finite normalized histories.

7. SAFETY AND ACCOUNTABILITY

Fix one Validator set with total Stake `W`. A Q2 set is strict when its unique Validator Stake is greater than `2W/3`.

Two strict Q2 sets overlap by more than `W/3`. If `Q_L` and `Q_R` are strict, their combined Stake is greater than `4W/3`, while their union has at most `W`. Their overlap must therefore exceed `W/3`.

Quorum overlap is necessary, but it is not the whole safety proof. The protocol also needs three facts:

1. Each overlapping Validator produced authenticated Votes for both claims.
2. Those Votes are valid evidence for the exact fork being claimed.
3. An honest Validator following PrevSelfVote cannot produce both conflicting Vote histories.

The normalized model proves all three under explicit assumptions. Evidence completeness says every Validator in the relevant overlap supplies the two authenticated Votes. Evidence soundness says accepted evidence proves a violation of reachable compliant signing behavior. Fixed-Validator-set safety says two conflicting CoveredBlock claims cannot coexist when at least two thirds of Stake is honest and online.

Evidence soundness also assumes strict height monotonicity, descent transitivity, and same-height uniqueness on one retained lineage. The model starts after decoding and signature checks. It does not prove network parsing, storage behavior, evidence submission, or penalty execution.

The PrevSelfVote rule matters here. Within one CoveredBlock interval, a Validator may move only to a strictly better Proposed Block. It cannot later return to an earlier or lower-ranked target. This blocks the concrete return path used by the smallest known attempted fork schedule.

8. LIVENESS

Safety and liveness have different assumptions. Liveness needs all of the following:

- After some unknown time, messages sent by correct online Validators arrive within a finite bound.
- Enough eligible Stake keeps producing valid Votes for the selected parent Block.
- Correct Validators eventually observe the same finite Proposed-Block set for each choice.
- Repeated valid Votes are eventually included in a Proposed Block.
- Validators can eventually satisfy PrevSelfVote for their next required Vote.

At unanimity, liveness requires every Validator. The Byzantine fault bound cannot supply that.

The normalized model proves conditional progress once these events are supplied. It proves that Proposed Blocks can continue and that later Q2 Votes can advance CoveredBlock in the model. It does not derive message delivery, scheduling, common-view convergence, or extension of the production-selected lineage.

9. EXECUTABLE VALIDATION

Finite tests check the algorithm's sharp edges. They cover threshold equality, duplicate Validators, malformed Votes, competing Proposed Blocks, delayed PrevSelfVote data, Validator-set changes, restart, and attempted forks.

The arithmetic corpus contains 1,118,208 ordered pair evaluations plus five transition vectors, for 1,118,213 cases. The generator uses 1,344 Validator sets with three to five Validators. Of the ordered pairs, 69,320 contain two strict Q2 sets. These latter numbers describe inputs and a subset; they are not extra cases.

Separate production-path tests reorder and duplicate Votes and Proposed Blocks. They also hold a Proposed Block until its missing PrevSelfVote arrives. The accepted result is independent of delivery order in the recorded cases.

10. RELATED WORK

Leader-based BFT protocols select a proposer or leader for a view. PBFT uses a primary and view changes. Tendermint rotates proposers and uses prevote and precommit locks. HotStuff and MonadBFT rotate leaders and use quorum certificates [1, 2, 3, 4]. Tsampi selects no Validator before Proposed Blocks exist.

That fact alone is not the innovation. Narwhal/Tusk lets every Validator contribute DAG data before a shared coin picks a later leader. Aleph lets every Validator create DAG units before common randomness picks a head [5, 6]. Both then order the chosen anchor's causal history.

Tsampi does something narrower and more direct. It has one parent per Block, no DAG ordering phase, and no random ordering anchor. One Vote round can produce several valid Proposed Blocks. The endorsement threshold controls which Proposed Blocks enter consensus. A deterministic order chooses among the observed set. PrevSelfVote constrains each Validator's later Votes. Later Blocks set QuorumBlock and then CoveredBlock on that exact lineage.

Casper FFG and GRANDPA add accountable finality to separate Block production. Bitcoin Backbone gives probabilistic settlement rather than a deterministic CoveredBlock rule [7, 8, 9]. Tsampi instead derives QuorumBlock and CoveredBlock from Votes carried by later Blocks on one lineage.

Vote chaining is not new by itself. Prior asynchronous vote-based work also links successive actions by the same participant [6]. Tsampi's novel vote-chain rule requires each new endorsement to target a strictly better Proposed Block than that Validator's preceding endorsement. PrevSelfVote makes this ordered improvement verifiable within one CoveredBlock interval. The chain gates live Vote and Proposed-Block release, while committed replay uses the authenticated Votes already present in committed Blocks.

The paper does not claim that weighted Stake, leaderless contribution, deterministic ordering, Vote chaining, two-pass finality, or accountability is new by itself. The claimed innovation is their composition above. The compared protocols' theorems do not transfer to Tsampi because their Block structures, Vote rules, timing assumptions, and finality rules differ.

11. LIMITS

The deterministic choice result assumes complete valid Proposed Blocks with complete lineages to one retained root. Missing data can differ across nodes, so equal behavior is claimed only for equal complete inputs.

Fixed-Validator-set safety, evidence completeness, and evidence soundness are proved only in the normalized post-authentication model and under the assumptions in Section 7. Changing-Validator-set safety also assumes exact parent-derived Validator sets at the first fork and the fixed-Validator-set premises on both sides.

Production liveness, throughput, elapsed-time finality, and changing-Validator-set latency remain open. Durable PrevSelfVote history can grow without a protocol depth bound while CoveredBlock does not advance, increasing storage and restart work.

The finite checks establish only their recorded outcomes. They are not a safety induction, a liveness proof, or an empirical evaluation. One narrow allocation check observed zero heap allocations for one over-capacity rejection; no end-to-end performance result is claimed.

12. CONCLUSION

Tsampi shows how a leaderless proposal stage can remain a single-parent chain. Validators may endorse successively better proposals in one dual-linked Vote history, deterministic choice resolves the observed candidates, and two strict-supermajority passes advance exact-lineage finality. Admission remains configurable without lowering the finality threshold.

The normalized proofs establish fixed-Validator-set safety, evidence completeness, evidence soundness, and conditional safety across Validator-set changes.

REFERENCES

[1] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pages 173–186. USENIX Association, 1999. doi: 10.5555/296806.296824.

[2] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *arXiv preprint arXiv:1807.04938*, 2019. doi: 10.48550/arXiv.1807.04938.

[3] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus in the lens of blockchain. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019. doi: 10.1145/3293611.3331591.

[4] Mohammad Mussadiq Jalalzai, Kushal Babel, Jovan Komatovic, Tobias Klenze, Sourav Das, Fatima Elsheimy, Mike Setrin, John Bergschneider, and Babak Gilkalaye. MonadBFT: Fast, responsive, fork-resistant streamlined consensus. *arXiv preprint arXiv:2502.20692*, 2026. doi: 10.48550/arXiv.2502.20692.

[5] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: A DAG-based mempool and efficient BFT consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 1–17, 2022. doi: 10.1145/3492321.3519594.

[6] Adam Gągol, Damian Leśniak, Damian Straszak, and Michał Świętek. Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. *arXiv preprint arXiv:1908.05156*, 2019. doi: 10.48550/arXiv.1908.05156.

[7] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2019. doi: 10.48550/arXiv.1710.09437.

[8] Alistair Stewart and Eleftherios Kokoris-Kogias. GRANDPA: A byzantine finality gadget. *arXiv preprint arXiv:2007.01560*, 2020. doi: 10.48550/arXiv.2007.01560.

[9] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology – EUROCRYPT 2015*, pages 281–310. Springer, 2015. doi: 10.1007/978-3-662-46803-6_10.