Tsampi BFT: Leaderless One-Round Voting with Parameterized Finality

Preprint, compiled August 17, 2026

Tim Watts
tim@readevalprint.com

Abstract

Tsampi BFT is a leaderless Byzantine fault-tolerant state-replication protocol with one Vote round per Proposed Block and exact-lineage finality in as few as two later Blocks. Its dual-linked Vote chain records both Block-lineage causality and each Validator's endorsement order. Earlier Votes are neither revoked nor reassigned; every successor must prove that it targets 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. Q2 starts empty after Q1 closes. A Validator counted in Q1 may count in Q2 only through a different, later Vote carried by a later descendant Block.

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. Proposal admission may use a lower threshold, but every accepted Block must advance the lineage toward strict-supermajority finality.

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. No proposer has an exclusive proposal right, but proposal timing is score-prioritized: each authenticated proposer Vote and its parent-state Stake determine a canonical earliest BlockTime, and a better stake-weighted exponential-race score cannot give a later deadline. A Block stamped before that deadline is invalid. Silence by the earliest proposer causes no view change; every other eligible proposer retains its own finite deadline. 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.

The wire protocol has only Vote, Block, and Tx messages. It has no certificate message, empty Vote, or Vote revocation.

This paper makes three contributions:

1. a dual-linked Vote chain that requires each Validator's successive endorsements to improve under one deterministic proposal order;
2. normalized proofs of safety, evidence completeness, evidence soundness, and conditional Validator-set change safety; and
3. a Validator-count-dependent bound on active proposal and Vote history while the Validator set is unchanged.

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 history bound for an unchanged Validator set assumes that its Validator identities, signing keys, and Stake weights remain the same throughout the interval.

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.

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; there is no separate certificate object and no Vote-revocation object.

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.

Let `R(B)` be the rank of Proposed Block `B` under the deterministic order, with larger ranks preferred, and let `H(V)` be the hash of signed Vote `V`. A legal successor `V2` to `V1` satisfies both

```
PrevSelfVote(V2) = H(V1)
```

and

```
R(target(V2)) > R(target(V1)).
```

This gives the central invariant: within one CoveredBlock interval, each Validator's accepted Votes form one rooted hash-linked chain whose targets strictly increase under `R`. Rank is computed from `B` and its retained parent chain. It does not depend on other candidates a node has observed. `H(V)` hashes the complete signed Vote. PrevVote and the race sample use the inner Vote signature; they do not change when the Endorsement changes.

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:** Byzantine Stake in every applicable Validator set is at most one third of total Stake.

An honest Validator may be offline, partitioned, or silent, but it does not sign conflicting Vote histories. A Byzantine Validator may equivocate, selectively withhold messages, or violate PrevSelfVote.

Safety does not require honest Validators to be online or messages to arrive on time. Messages may be delayed, reordered, duplicated, dropped, 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 determine both the proposal deadline and a bounded proposer multiplier used to order competing Proposed Blocks. Neither changes the endorsement threshold or the finality threshold.

3.1 Score-prioritized proposal timing

Let `T` be the configured target Block interval, `E` the parent Block's interval EMA, `W` total parent-state Stake, and `s > 0` the proposer's parent-state Stake. The canonical maximum race delay is

$$D_{\max} = \mathrm{clamp}(T/2 + (T - E)/2, T/20, T),$$

with deterministic integer-millisecond arithmetic and a minimum clamp of one millisecond. For proposer Vote V, let $h = \mathrm{SHA256}(\texttt{"tsampi/proposer-race/v1"} \| \texttt{VoteSig}(V))$, let $U = (2h + 1)/2^{257}$, and let $L = \lfloor -\ln(U)2^{512} \rfloor$. The non-empty-Block race delay is

$$d(V) = \max(1\text{ ms}, \min(D_{\max}, \lfloor D_{\max} LW/(s2^{512}) \rfloor)).$$

The earliest valid BlockTime is the parent BlockTime plus $d(V)$. Empty Blocks must also satisfy the separately configured empty-Block delay. All inputs come from the signed proposer Vote, the exact parent state, or parent header, so every verifier computes the same deadline for the same candidate. The race sample is the proposer Vote signature. VoteSig, the race sample, and the next PrevVote hash are not grindable: a valid inner Vote is unique given its parent Block and signing key, and neither the Endorsement nor the Block body can change it. This is not Algorand VRF sortition. The sample is a public hash of that signature. It does not elect a private committee. Nodes may possess different candidates or begin local waiting at different wall-clock times; they do not compute different deadlines for one complete candidate.

The normalized safety proof does not depend on which valid deadline expires first: timing changes which valid histories arise, but not Vote validity, strict-rank improvement, deterministic comparison of a fixed candidate set, or Q1/Q2 replay of a fixed authenticated lineage. This is not an optimistic-

responsiveness theorem. The finite deadline removes an exclusive first-speaker role, but progress still requires the participation, delivery, stable-view, inclusion, and PrevSelfVote assumptions in Section 8. No reduction in contention, elapsed latency, or active-history size is claimed without measurement.

3.2 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. Each compared field comes from the candidate and that retained chain. The bounded proposer score lies in `[1, 3/2]`. It belongs to the first child after the latest Epic-completing Block on the lineage. A later candidate does not receive that score unless it is that child. 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. A fresh later strict-supermajority observation covers the preceding QuorumBlock, which becomes CoveredBlock. These are rolling tip-relative lineage frontiers, not fixed rounds assigned by scanning forward from genesis.

For a candidate tip `T` with exact lineage `B[0], ..., B[T]`, define its frontiers by folding the authenticated Block records in height order. When `B[h]` is appended for `h >= 2`, its carried Votes endorse `B[h-1]` and therefore observe every ancestor through `B[h-2]`. The protocol opens a Q1 window for `B[h-2]`, adds the complete canonical Validator batch from `B[h]` to that window and every older open Q1 window, deduplicates Stake within each window, and scans the windows newest first. The first window above two thirds sets `QuorumBlock(T)`. Before that Q1 update, the same batch advances the previously open Q2 window; if it crosses two thirds, its candidate sets `CoveredBlock(T)`. A new Q1 hit then replaces Q2 with an empty window for that Q1 candidate. If neither pass closes, the prior frontiers are inherited.

All Votes carried by the Block that crosses Q1 form one closing batch; the protocol does not choose a minimal threshold subset from that Block. Earlier Blocks may have supplied the rest of the Q1 Stake. No Vote batch processed through the Q1-closing Block is copied into the new Q2 window. A Validator counted in Q1 may count again in Q2 only through a different, later Vote carried by a later descendant Block; it cannot reuse its Q1 Vote or count a second time by endorsing the same target. Q1 and Q2 therefore deduplicate independently: their Validator sets may overlap, but neither must contain the other.

On one exact lineage, the frontiers cannot move backward. After a Q1 hit, the protocol discards that window and every older Q1 window, retaining only newer candidates. Immutable late data cannot rewrite an accepted Block's batches. A separately received competing tip has its own derived frontiers and does

not mutate `T`; deterministic choice ranks higher CoveredBlock and then higher QuorumBlock before pending Stake.

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 a strict-supermajority Vote set that endorses genesis and admits Block 1.
- Block 2 carries a strict-supermajority Vote set that endorses Block 1 and therefore observes genesis; this sets QuorumBlock to genesis.
- After processing Block 2's batch, the protocol opens an empty Q2 window for genesis. No Vote batch processed through Block 2 counts toward that Q2.
- Block 3 carries later Votes that endorse Block 2; this fresh strict-supermajority batch 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. Until Epic advances, every valid Block must increase accumulated unique Stake. A Proposed Block that carries only Votes already counted in that accumulation is invalid because it adds no previously unseen Stake.

With an unchanged equal-Stake Validator set, the maximum active history is the product of three quantities:

1. the number of proposal heights needed to collect a strict supermajority;
2. the number of safe Proposed Blocks at each height; and
3. the number of strictly improving Votes that can target those proposals.

If `a` Validators can admit the first Block and `q` Validators form a strict supermajority, one accumulation pass needs at most `q - a + 1` proposal heights. At each height, at most one non-equivocating proposal per Validator is safe. If every Validator starts with its own proposal and then endorses every better proposal, the Votes form the triangular series `n + (n - 1) + ... + 1`. Without the own-proposal-first schedule assumption, the general per-height Vote bound is `n^2`.

A pipeline with an open fresh-Q2 window needs one such pass before CoveredBlock advances. Startup from genesis needs a Q1 pass followed by fresh Q2, with one additional carrier height.

This is an unchanged-Validator-set history bound, not a changing-Validator-set latency result. An Epic may change Validator identities, keys, or Stake weights, and a newer Q1 may replace the open fresh-Q2 window. The implementation also has a separate fail-closed replay-record ceiling; that ceiling does not prove that CoveredBlock advances.

5.3 Unanimity

When `p = 1`, every Validator must Vote. Unanimity is not Byzantine fault tolerant for proposal production: one Byzantine, offline, or silent Validator can stop Block production by withholding its Vote. 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. Verification of a Block always uses the Validator set in its parent output. Epic rewrites that output only after CoveredBlock has reached the request. 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 does not release two CoveredBlock claims that form a fork.

Main theorem (normalized fixed-Validator-set safety). For every finite post-authentication history over one unchanged Validator set with total Stake `W`, if Byzantine Stake is at most `W/3` and the assumptions in Sections 2 and 7 hold, two conflicting CoveredBlock claims cannot both have valid Q2 evidence.

The paper's argument is organized around three results:

1. **Dual-chain ordering:** PrevVote fixes each Vote's Block-lineage position, while PrevSelfVote orders every Vote by one Validator in one rooted chain of strictly improving targets.
2. **Conflicting-coverage exclusion and accountability:** strict Q2 overlap exceeds `W/3`; the normalized model proves that the overlapping authenticated actions are complete evidence and that an honest Validator cannot supply both conflicting CoveredBlock claims.
3. **Finite active history:** with an unchanged Validator set, strict proposal improvement and mandatory unique-Stake progress give the bound in Section 5.2.

PrevSelfVote and the strict-rank rule are live admission rules. A Validator may endorse a weaker proposal and then a better conflicting one. That schedule can complete two Q1 sets. Q2 starts empty after Q1, and a return to the weaker target is invalid. Those facts constrain live voting. They are not the safety theorem.

The safety theorem uses the overlap already shown, plus one more fact: honest released CoveredBlock claims are equal or related by exact-parent descent, so they cannot form a fork. The normalized model takes that signing rule as an assumption on honest emissions. Committed replay does not re-check PrevSelfVote.

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 minimal adversarial schedule makes this boundary concrete. Let `R(B0) < R(B1)`. A Validator may endorse `B0` and then `B1`; that is a legal improvement. Any later Vote that returns to `B0` is the first invalid transition because its target rank decreases. A distinct successor that repeats `B1` is also invalid because its rank does not increase.

8. LIVENESS

Safety budgets Byzantine Stake. Liveness also depends on participation and delivery. Honest offline Stake cannot equivocate, but it cannot help close Q1 or Q2. Liveness needs all of the following:

- After some unknown time, messages sent by correct online Validators arrive within a finite bound.
- Enough eligible honest Stake remains online and 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. RELATED WORK

Tsampi's contribution is the following composition: single-parent proposals, no separate QC or certificate object, a per-Validator hash-linked strict-rank endorsement chain, and Q1 followed by independently deduplicated fresh Q2 on one exact lineage. PrevVote records Block-lineage causality. PrevSelfVote records each Validator's endorsement order and requires every successor to target a strictly better proposal. Among the protocols reviewed here, none combines those four properties. Gasper is the closest two-pass chain. It uses slot proposers, latest-message fork choice, and Casper FFG checkpoint links. It does not use a hash-linked strict-rank Vote chain or a Q2 window that starts empty after Q1 [1]. Algorand is the closest stake-weighted proposer lottery. It privately elects VRF committees, certifies BA* with committee-vote certificates, and does not keep a strict-rank Vote chain [2]. VoteSig, the race sample, and the next PrevVote hash are not grindable: a valid inner Vote is unique given its parent Block and signing key.

Leader-based protocols retain a selected proposer and explicit safety state even when they reduce the happy path to two phases or one pipelined Vote per Block [3, 4, 5, 6, 7, 8, 9, 10, 11]. These two-chain protocols are listed separately because their locks, view-change proofs, and commit rules differ. Alpenglow's one-voting-round fast path assumes Byzantine Stake strictly below 20%. Minimmit and ChonkyBFT assume unweighted $n \geq 5f + 1$, so their Byzantine processor or replica fraction is strictly below 20%, not a 20%-Stake theorem. Each still uses a leader and certificate, notarization, or CommitQC objects [12, 13, 14].

Narwhal/Tusk, Bullshark, Mysticeti, and Aleph let all Validators or nodes contribute DAG Blocks or units before later ordering or commit rules. Formal Bullshark assumes reliable broadcast; its practical Narwhal-backed realization uses availability certificates and leader anchors. Mysticeti avoids explicit Block certificates and requires a Validator's Blocks to self-link, but encodes commit support in a multi-parent DAG and designated proposer slots. That per-creator Block chain is not a chain of exact endorsements ordered by a total candidate rank

Protocol	Proposal structure	Separate certificate, QC, or anchor	Finality voting
PBFT	Primary; single replicated log	Prepared and commit certificates	Prepare then commit
Tendermint	Round proposer; single chain	Locks and supermajority Vote sets	Prevote then precommit
HotStuff	View leader; single chain	QCs and lock	Three-chain QC commit
HotStuff-2	View leader; single log	Status, lock, and commit certificates	Two phases per view
Jolteon	Round leader; single-parent QC chain	QCs and timeout certificates	One pipelined Vote per Block; two-chain commit
Fast-HotStuff	View primary; single-parent QC Block tree	QC, highQC, and AggQC	Two Vote rounds, or one pipelined Vote per Block with two-chain commit
SBFT	Primary; single replicated log	Threshold-signed fast/slow evidence	Optimistic fast or PBFT-derived slow path
Simplex	Random leader; hash-linked chain plus timeout dummy Blocks	Notarizations and finalizations	Distinct notarize and finalize supermajorities
MonadBFT	Rotating leader; single chain	QC and backup QC	QC pipeline; speculative and full commit rules
Alpenglow	Slot leader; chain	Five aggregate certificate types	80% one-round fast or 60% two-round slow path
Algorand	VRF-sortition proposers; single-parent chain	Committee-vote certificates	BA⋆: reduce, then binary agree; final or tentative
Minimmit	View leader; unique-parent Blocks	M/L notarizations and nullifications	One growing Vote set; $n - f$ finality under $n \geq 5f + 1$
ChonkyBFT	View leader; one Block proposal	CommitQC	One $n - f$ CommitVote pass under $n \geq 5f + 1$
Narwhal/Tusk	All Validators; multi-parent DAG	Availability certificates and random wave leader	DAG wave commit
Bullshark	All Validators; multi-parent DAG	Reliable broadcast; practical Narwhal certificates; leader anchors	DAG wave commit
Mysticeti-C / Sui	All Validators; multi-parent DAG	No explicit Block certificate; proposer slots anchor commit	DAG-reference support rules
Aleph	All nodes; multi-parent DAG	Reliable broadcast and random head	Random-head DAG ordering
Casper FFG	External Block tree	Checkpoint supermajority links	Justify then finalize descendant checkpoint
Gasper	Slot proposer; LMD-GHOST tree	Checkpoint supermajority links	Justify then finalize
GRANDPA	External Block tree	Commit certificates	Prevote then precommit
Nakamoto consensus	Proof-of-work longest chain	Probabilistic depth	No deterministic finality pass
Tsampi	**No preselected proposer; single-parent proposals**	**None**	**Q1 then independently deduplicated fresh Q2 on one lineage**

Table 1: Structural comparison of the reviewed protocols. Similar labels do not transfer theorems across models.

[15, 16, 17, 18]. Casper FFG and GRANDPA add accountable finality to separate production, while Nakamoto consensus provides probabilistic settlement [19, 20, 21].

Hash-linking successive participant actions is not new by itself. Aleph units contain parent hashes, and the units of each honest creator form a chain. Aleph's binary Votes are virtual: later units recompute them from prior-round DAG parents rather than transmitting a signed Vote object that names its creator's preceding Vote. Neither the unit chain nor the virtual-Vote recurrence orders exact endorsement targets by one total proposal rank [18]. Tsampi's narrower rule is that PrevSelfVote orders each Validator's exact endorsements and permits only strict improvement. The chain gates live Vote and Proposed-Block release; committed replay uses the authenticated Votes already carried by committed Blocks.

The paper does not claim novelty for weighted Stake, leaderless contribution, deterministic ordering, Vote chaining, two-pass finality, or accountability in isolation. The claim is the complete composition above. The compared protocols' theorems do not transfer because their Block structures, Vote rules, timing assumptions, and finality rules differ.

10. 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, scheduler refinement, and changing-Validator-set latency remain open. Score-prioritized deadlines do not by themselves prove optimistic responsiveness or reduce the worst-case active-history bound. With an unchanged Validator set, valid active history has the combinatorial bound in Section 5.2 rather than a fixed byte or depth constant. The implementation's replay ceiling fails closed if coverage does not advance; it is not a liveness guarantee.

11. 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.

No leader is selected before proposals, no certificate message is created, and no Vote is revoked. A later Vote may move only to a provably better proposal.

REFERENCES

[1] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining GHOST and casper. *arXiv preprint arXiv:2003.03052*, 2020. doi: 10.48550/arXiv.2003.03052.

[2] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles*, pages 51–68, 2017. doi: 10.1145/3132747.3132757.

[3] 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.

[4] 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.

[5] 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.

[6] Dahlia Malkhi and Kartik Nayak. HotStuff-2: Optimal two-phase responsive BFT. *Cryptology ePrint Archive, Paper 2023/397*, 2023. URL https://eprint.iacr.org/2023/397.

[7] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *Financial Cryptography and Data Security*, pages 296–315, 2022. doi: 10.1007/978-3-031-18283-9_14.

[8] Mohammad M. Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-hotstuff: A fast and robust BFT protocol for blockchains. *IEEE Transactions on Dependable and Secure Computing*, 2023. doi: 10.1109/TDSC.2023.3308848.

[9] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 568–580, 2019. doi: 10.1109/DSN.2019.00063.

[10] Benjamin Y. Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In *Theory of Cryptography*, pages 452–479, 2023. doi: 10.1007/978-3-031-48624-1_17.

[11] 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.

[12] Quentin Kniep, Jakub Sliwinski, and Roger Wattenhofer. Solana alpenglow consensus: Increased bandwidth, reduced latency. White Paper v1.1, Anza, 2025. URL https://github.com/rogerANZA/Alpenglow-White-Paper/blob/f29dbbc51af8d83176f1fc06836ba6aa193d731a/Alpenglow-v1.1.pdf.

[13] Brendan Kobayashi Chou, Andrew Lewis-Pye, and Patrick O'Grady. Minimmit: Fast finality with even faster blocks. *arXiv preprint arXiv:2508.10862*, 2026. doi: 10.48550/arXiv.2508.10862.

[14] Bruno França, Denis Kolegov, Igor Konnov, and Grzegorz Prusak. ChonkyBFT: Consensus protocol of ZKsync. *arXiv preprint arXiv:2503.15380*, 2025. doi: 10.48550/arXiv.2503.15380.

[15] 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.

[16] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022. doi: 10.1145/3548606.3559361.

[17] Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the latency limits with uncertified DAGs. In *Network and Distributed System Security Symposium*, 2025. doi: 10.14722/ndss.2025.240929.

[18] 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.

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

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

[21] 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.