BUCK Notes – SNARK-Verified Batch Mint Design (Phase 7 Pivot)

Perry Kundert

2026-04-22

This memo records a substantial design pivot for the BUCK Notes mint path. The shipped Phase-7 implementation (`src/Notes.sol`, `circuits/mint.circom`) costs ~1.9M gas at N=2 and scales roughly linearly per leaf – a non-starter at batch sizes large enough to amortize identity-handshake and BUCK-approve overhead. The pivot moves the per-leaf Poseidon Merkle insertions *into* the mint SNARK; on-chain mint cost still grows with N (one G1 ECMUL per public input, plus per-leaf event + memory copy overhead), but each marginal leaf costs ~31K gas (measured) instead of the shipped design's ~800K per on-chain Poseidon insertion. Per-note mint amortizes from 39K at N=16 (measured, 624K total) to 35K at N=32 (measured, 1.12M total) toward a ~31K asymptote. Spend cost is independent of mint batch size (~360K, measured).

The corrected cost model `mint_tx_gas(N)` ~ 125K + 31K * N= puts the L1 practical ceiling at **N = 256-512** (block-share, not bytecode size, is the binding constraint at scale); L2 deployment lifts this by ~100x. Recommended initial pinned set: **{16, 32}** on L1 with stock Groth16 verifiers (both fit the EIP-170 24 KB ceiling); promote to **{16, 32, 128}** via calldata-IC hash-pinned delivery (a natural fit for the Holochain wallet); use proof aggregation above N=512.

This document covers: the circuit shape, the public-input layout, measured on-chain gas at N=16 and N=32 with projections through N=1024, trusted-setup infrastructure scaling, the EIP-170 24 KB verifier-bytecode wall and three escape hatches, and the recommended pinned-N policy.

Updating `alberta-buck-notes-flow.org` to reflect the new mint flow is the immediate next step.

Contents

1 Motivation

Approach	Per-leaf on-chain cost	N=2	N=16 (M)	N=32 (M)	N=128 (P)
Shipped (per-leaf on-chain insert)	~800K gas	~1.9M	~13M	~26M	~100M+
Pivot (in-circuit insert)	~31K gas (measured)	~190K	624K	1.12M	~4.1M

The shipped design is dominated by `N x TREE_DEPTH` Poseidon-T3 calls at ~30-40K gas each. Even N=16 exceeds the 30M block limit; N=128 is roughly two orders of magnitude past economic viability at any realistic gas price.

Moving the tree update into the SNARK moves the bulk of the per-leaf cost into the prover's off-chain work. The on-chain cost still grows with N – Groth16 verification performs one G1 scalar multiplication per public input, and `cm[0..N-1]` appear as N individual public inputs – but the empirical per-leaf cost is ~31K gas (measured at N=16 and N=32; pure ECMUL is ~6K but per-leaf `Appended` events, `cm[i]` memory copy and verifier-internal pub handling add the rest) versus ~800K gas for an on-chain per-leaf Poseidon ladder, a ~25x per-leaf saving that compounds as N grows. The exact per-N numbers and the cost-model fit are in *On-chain cost projection* below.

2 Design Sketch

2.1 Public inputs to mint_batch circuit (`4 + N` entries)

pub index	Signal	Description
`pub[0]`	`oldRoot`	The live note Merkle root the prover read from chain
`pub[1]`	`newRoot`	Root after inserting `cm[0..N-1]` starting at `nextLeafIndex`
`pub[2]`	`nextLeafIndex`	The live tree size the prover read from chain
`pub[3]`	`totalFace`	Sum of `v[i]`, range-bounded to 128 bits
`pub[4..4+N)`	`cm[0..N)`	The batch commitments, one per public input

We pass `cm[]` directly as individual public inputs rather than folding them into a `cmBatchHash`. Trade-off: +N ECMULs in on-chain verifier cost vs. -1 Poseidon-N constraint in-circuit and -1 keccak on-chain. At the prover sizes we care about (N <= 128) the Groth16 verifier cost grows linearly with N anyway, and the hash-elision keeps the public-input surface self-describing for the adapter (`pub[4+i] === cm[i]`).

2.2 Private witness

- Per leaf i in 0..N-1: `flavor[i], v[i], rho[i], idHash[i], predicate[i]`
- Per leaf i in 0..N-1: TREE_DEPTH path siblings needed to fold `cm[i]` into the rolling tree (derivable from the prover's local mirror of `filledSubtrees` plus the previously-inserted leaves of this same batch)

2.3 In-circuit constraints

- N x Poseidon-5: `cm[i] === Poseidon(flavor, v, rho, idHash, predicate)[i]`
- N x Num2Bits(128) on `v[i]`, plus Num2Bits(128) on `totalFace`
- 1 sum: `totalFace === sum(v[i])`

- N x TREE_DEPTH *dual* Poseidon-2 hashes implementing the rolling filled-subtrees insertion from `oldRoot` to `newRoot`. Each leaf walks two Merkle paths sharing the same siblings: `priorWalk[i]` uses `ZERO_VALUE` at the insertion slot (attests `oldRoot/tail`), `postWalk[i]` uses `cm[i]` (advances `rollingRoot`). The prover's filled-subtrees mirror supplies the left-subtree values; zeros at the right come from a constant `zeros[]` ladder.
- (Optional) `cm[i] !` 0= and `cm[i] !` ZERO_VALUE= belt-and-suspenders

2.4 Constraint counts and proving cost

Per leaf: ~1.7K opening + ~2 x 4.3K dual-walk = ~10K non-linear R1CS. The dual-walk (one Merkle path consumed for `oldRoot` attestation, one for rolling `newRoot` advance) is what lifts per-leaf cost above the N-single-walk budget of the shipped design.

Measured (M) at N=16 and N=32; projected (P) elsewhere:

N	R1CS	ptau	final zkey	Verifier .sol	Prover wall (snarkjs)
16	~166K M	2^19	160 MB M	14.5 KB M	~6 s on dev laptop M
32	~332K M	2^20	320 MB M	20.8 KB M	~12 s on dev laptop M
64	~660K P	2^21	~640 MB P	~33 KB P	~30 s P
128	~1.3M P	2^22	~1.3 GB P	~58 KB P	~1 min (rapidsnark) P
256	~2.6M P	2^23	~2.6 GB P	~108 KB P	~2-3 min (rapidsnark) P
512	~5.2M P	2^24	~5.2 GB P	~208 KB P	~5-8 min (rapidsnark) P
1024	~10M P	2^25	~10 GB P	~408 KB P	~10+ min (rapidsnark) P

The measured N=16 constraint count (~166K non-linear + ~185K linear) is ~1.7x higher than the initial `~96K` estimate in the original pivot memo. The delta comes from the *dual* Merkle walk: attesting `oldRoot` against the insertion tail and then advancing `rollingRoot` cost two Poseidon ladders per leaf rather than one. The `R1CS` column scales ~linearly in N from the measured N=16 / N=32 baseline.

The `Verifier .sol` column is the Solidity output of `snarkjs zkesv`. N=16 lands at 14.5 KB, N=32 at 20.8 KB. Linear extrapolation puts N=64 at ~33 KB – the first size to *breach* the 24 KB EIP-170 contract limit. See *Pinned-N economics* below for the implications and three escape hatches.

2.5 On-chain `Notes.mint(proof, oldRoot, newRoot, nextLeafIndex, totalFace, cms[])`

1. Require `oldRoot == roots[currentRootIndex]` (stale-state guard, pre-SNARK)
2. Require `nextLeafIndex == self.nextLeafIndex` (stale-state guard, pre-SNARK)
3. Require every `cms[i] < FIELD_R` and `newRoot < FIELD_R`
4. `mintVerifier.verifyMint(proof, oldRoot, newRoot, nextLeafIndex, totalFace, cms[])` – adapter dispatches to the per-N `MintBatchN${N}Groth16Verifier` by `cms.length`, builds `pub[0..4+N)` inline, calls the verifier.
5. `buck.transferFrom(msg.sender, address(this), totalFace)`
6. `roots[(currentRootIndex+1) % HISTORY] = newRoot`; advance index; `self.nextLeafIndex` + cms.length=

7. `noteFaceSum` + totalFace=

8. Emit one `Minted(issuer, totalFace, startIndex, count, newRoot)` plus per-leaf `Appended(cm, leafIndex)` for off-chain indexers. Calldata `cms[]` remains the canonical source (16 gas/byte); the event stream is a convenience for provers that want to subscribe rather than re-parse.

2.6 On-chain cost projection

Empirically, `Notes.mint` fits the affine model

```
mint_tx_gas(N) ~= 125K (fixed) + 31K * N
```

with both terms *measured* from the N=16 and N=32 forge runs. The fixed component covers state guards, two SSTOREs, `buck.transferFrom`, the Groth16 pairing check, the constant-size IC reductions, and event header overhead. The per-leaf component covers (a) the G1 ECMUL for the public input (~6K gas via precompile `0x07`), (b) memory copy of `cm[i]` into the adapter's `pub[]` buffer (~5K), (c) the per-leaf `Appended(cm, leafIndex)` LOG (~2K), (d) calldata for the 32 B `cm[i]` word (~512 gas), (e) verifier internal stack ops on the additional input (~17K).

The pure-ECMUL-only projection of ~6K/leaf in the original pivot memo under-counted by ~5x; the corrected number is the empirically-fitted ~31K.

2.6.1 Mint-path gas (measured M / projected P)

N	mint-tx total	per-note (mint / N)	block-share (30M cap)
16	624K M	39K M	~2.1% M
32	1.12M M	35K M	~3.7% M
64	~2.1M P	~33K P	~7% P
128	~4.1M P	~32K P	~14% P
256	~8.0M P	~31K P	~27% P
512	~16M P	~31K P	~53% (half block) P
1024	~32M P	~31K P	>100% (UNFEASIBLE) P

The per-note cost asymptotes at ~31K, not ~6K as the pure-ECMUL projection suggested. Doubling N from 16 to 32 saves ~4K/note; doubling again to 64 saves only ~2K/note; further doubling barely moves the needle.

The *block-share* column is the binding economic constraint: at N=512 a single mint occupies half a block (block builders heavily discount such txs); at N=1024 it cannot land at all. The shipped per-leaf design saturated the block at N=4; the pivot pushes that ceiling to N=512, but does not eliminate it. Reaching usable batch sizes beyond N=256 requires either L2 deployment (where block budgets are 100x looser) or proof aggregation (recursion).

2.6.2 Spend-path gas: independent of the mint N

The spend circuit (`circuits/spend.circom`) has a *fixed* public-input set regardless of batch mint size: `[noteRoot, nullifier, face, recipient, chainId]` (5 inputs). Measured `Notes.spend()` cost is ~360K gas (`SpendVerifier.t.sol` forge runs). Spending a note minted in an N=1024 batch costs the same as spending one minted alone. N is purely an issuer-side amortization knob; recipients are insulated from the choice.

2.6.3 Calldata delivery: IC on-chain vs. off-chain

The Groth16 verifier stores its `IC[]` (one G1 point per public input, each 64 B) inline in deployed bytecode. Measured: N=16 verifier .sol = 14.5 KB (~1.3 KB of IC constants), N=32 = 20.8 KB. Linear extrapolation puts N=64 at ~33 KB – the first size to exceed the EIP-170 24 KB contract limit.

Three escape hatches keep large-N verifiers deployable:

1. **Calldata-supplied IC, hash-pinned.** Deploy a minimal verifier shell that holds only `keccak256(IC_bytes)`; caller supplies `IC_bytes` as calldata, contract checks the hash, runs the pairing math. ~16 gas/byte calldata: N=128 adds ~135K gas/tx, N=512 adds ~528K gas/tx. Requires off-chain availability of the IC blob – trivial for a Holochain-backed wallet, or an IPFS pin / GitHub release.

2. **Sharded SSTORE2 data contracts.** Deploy K <= 24KB data contracts whose *bytecode* is the IC bytes; verifier reads via `EXTCODECOPY` (~3 gas/byte, ~35K for 8 KB of IC). Fully on-chain, no off-chain dependency, more deploy-time complexity.

3. **Plain SLOAD from a storage-IC contract.** Simplest; most expensive. Cold load 2100 gas/slot, ~555K gas for N=128. Only useful up to ~N=64.

For BUCK: the Holochain wallet makes option (1) attractive. The pinned set carries IC hashes on-chain; the wallet delivers the IC bundle with each mint/spend; calldata cost is predictable and the on-chain verifier bytecode stays small regardless of N.

2.7 Trusted-setup infrastructure scaling

Each pinned N needs its own Groth16 zkey (circuit-specific), which in turn needs a `ptau` of power `POW` >= log2(FFT domain) chosen by snarkjs's `g16s`. Empirically for this circuit:

N	POW	ptau size (final)	ptau pt2 wall	g16s peak RAM	machine class
16	19	600 MB M	~15 min M	~4 GB M	dev laptop M
32	20	1.1 GB M	~80 min M	~8 GB M	dev laptop M
64	21	~2.2 GB P	~3 h P	~16 GB P	dev workstation P
128	22	~4.4 GB P	~6 h P	~32 GB P	workstation / cloud P
256	23	~8.8 GB P	~12 h P	~64 GB P	64 GB VM P
512	24	~17 GB P	~24 h P	~128 GB P	128 GB server P
1024	25	~34 GB P	~48 h P	~256 GB P	high-RAM dedicated box P

Wall-clock times are *single-machine* figures using `snarkjs` pt2. The ptau itself is *universal* – in a real ceremony it is produced once by a multi-party contribution and reused across every pinned N that fits within its `POW`. Our dev-only script (`scripts/snark/setup.sh`) uses fixed entropy; production should import a community ptau (e.g. Perpetual Powers of Tau) up to the `POW` we need.

Practical thresholds:

- N <= 32 is tractable on an unmodified developer laptop overnight.
- N = 64..128 benefits from `rapidsnark` (native C++ prover) for setup throughput and is the practical ceiling for continuous-integration CI runs.
- N >= 256 wants a dedicated VM; at N = 1024 the ptau alone is 38 GB and the setup is measured in days. Ceremony coordination becomes the binding constraint, not raw compute.

2.8 Pinned-N economics: where to draw the line

The pinned set is a deployment-time policy choice. Three pressures push in opposite directions:

1. *Smaller N is easier to deploy.* N = 16 fits in a 14.5 KB verifier; no bytecode-size gymnastics required. Every doubling of N doubles the Solidity verifier size, crossing the 24 KB EIP-170 ceiling around **N ~ 32-64**. Beyond that threshold one of the three escape hatches from *On-chain cost projection* becomes mandatory.

2. *Larger N amortizes only modestly.* Per-note mint gas drops from 39K (measured, N=16) to 35K (measured, N=32) to a projected ~31K asymptote above N=128. The headline savings happen between N=2 (shipped path) and N=16 (pivot baseline); doublings beyond N=64 buy <2K/note.

3. *Block-share is the next binding wall.* At N=512 a single mint occupies ~half a 30M-gas block; at N=1024 it cannot land at all. L1 economics ceiling is therefore around **N = 256-512**; L2 deployments (Arbitrum, Base, Optimism) raise this by ~100x, where N = 1024+ becomes feasible.

4. *Spend cost is flat regardless of N.* Minting bigger batches does not make recipients' spends cheaper or more expensive. The "where to draw the line" question is purely about issuer economics and deploy-time complexity, not recipient UX.

Recommended initial pinned set on L1: **{16, 32}** using stock single-contract Groth16 verifiers (both measured to fit the EIP-170 24 KB ceiling: 14.5 KB and 20.8 KB respectively). Promote to **{16, 32, 128}** once the calldata-IC trick is implemented and validated; the wallet already needs to carry the circuit wasm anyway, so adding an IC blob is incremental complexity.

For L2 deployment the pinned set can extend to **{16, 32, 128, 512}** without hitting block-share concerns; N=1024 even on L2 is where proof aggregation (Nova / Halo2 / Plonky3) starts to pay off – below that, the affine `125K + 31K * N` cost model means recursion's fixed overhead does not amortize. Above N=512 recursion is the real answer, and the pinned-N dispatch machinery collapses to a single fixed-size aggregator verifier.

3 Hostile-Alice Security Analysis

The single behavioral change vs. shipped Phase 7 is that `commitmentExists[cm]` duplicate-detection moves from explicit on-chain check to economic self-enforcement. Walk through every attack vector:

3.1 A1. Underpay `totalFace`

`totalFace` is a public input bound to the SNARK by the sum constraint and Num2Bits(128). `transferFrom` pulls exactly `totalFace`. GUARANTEE: intact.

3.2 A2. Field-overflow on sum

Num2Bits(128) on each `v[i]` and on `totalFace` gives 62-bit margin against field wrap at N <= 2^64. GUARANTEE: intact.

3.3 A3. Duplicate `cm` across mints

Today: `commitmentExists[cm]` reverts. New: insertion succeeds at a new leaf. Alice paid `totalFace` twice for the same opening; the nullifier `Poseidon(rho, idHash, 4242)` is deterministic in the opening, so she can spend exactly one copy; the other is unspendable forever. She loses money; no one else is harmed. GUARANTEE: economically self-enforcing.

3.4 A4. `cm` collision with `ZERO_VALUE` or 0

`ZERO_VALUE` is a domain-separated keccak reduced mod `r`; collision with the Poseidon-5 commitment space is 2^-254. Forcing such a collision requires breaking Poseidon preimage resistance. Add `cm[i] !` 0= and `cm[i] !` ZERO_VALUE= in-circuit for belt-and-suspenders. GUARANTEE: intact.

3.5 A5. Lying about `oldRoot`

Contract asserts `oldRoot == roots[currentRootIndex]`. Lying causes revert. GUARANTEE: intact.

3.6 A6. Lying about `newRoot`

SNARK constraints derive `newRoot` from `oldRoot` plus N folded leaves; Alice cannot satisfy them with a fabricated `newRoot`. GUARANTEE: intact.

3.7 A7. Front-running by Bob to invalidate Alice's proof

Bob's mint lands first -> `currentRootIndex` advances -> Alice's `oldRoot` check fails -> Alice's tx reverts. Alice paid no BUCK (transferFrom never ran); she re-proves against the new root. GUARANTEE: no value loss; **new operational concern**: wasted prover time under contention. Mitigation: optimistic retry in the wallet. Eventual mitigation: sequencer / proof aggregation.

3.8 A8. Mint with `idHash` she does not own

Same as today: B-flavor mint does not bind `idHash` to a registered identity (that is A-flavor's deferred work). Alice's chosen `idHash` determines her nullifier at spend. GUARANTEE: intact.

3.9 A9. Mint junk leaves whose openings she does not know

The circuit requires Alice to know the opening; she pays `totalFace` per junk leaf. Equivalent to a paid donation to the pool. GUARANTEE: intact.

3.10 A10. Spend nullifier reuse

Spend path unchanged. `nullifiers[nf] = true` on first burn; second reverts. GUARANTEE: intact.

3.11 A11. Spend front-running

`recipient` is bound in the spend SNARK public inputs and tied to the witness via the existing ghost-bind constraint group. Bob cannot reuse Alice's proof with a different recipient. GUARANTEE: intact.

3.12 A12. Lying about `nextLeafIndex`

Contract asserts equality with on-chain state; SNARK consumes the same value when computing path indices for insertion. Lying either reverts on the contract check or produces a `newRoot` the contract was not asked to honour. GUARANTEE: intact.

3.13 Headline

Every hostile-Alice guarantee in the shipped Phase 7 design holds under the batch-mint pivot. Buggy *wallets* (e.g., `rho` RNG collision) are no longer caught by the chain, but the failure mode is "wallet owner forfeits one note", not "pool drains".

4 Concurrency Model

Scenario	Behaviour
Single user, sequential mints	Each mint reads the live root, no race
Single user, queued mints	Wallet keeps a local rolling state mirror; no race
Multiple users, shared traffic	Classic rollup race; losers revert cleanly
Adversarial griefing	Equivalent to gas-paid DoS of any rollup

Initial deployment recommendation: **optimistic retry** in the wallet (re-prove against the new live root on revert). Add a sequencer-style mempool only if real-world contention warrants it.

A future "aggregated mint" path can collapse multiple users' batches into one proof per slot, but it inherits the same trust model as any other sequencer.

5 Migration Plan from Shipped Phase 7

5.1 Circuits

- New `circuits/mint_batch.circom` parameterised on N.
- Reuses existing Poseidon-2 / Poseidon-5 templates; adds the rolling filled-subtrees insertion sub-circuit – the in-circuit dual of the shipped `Notes._insert`.
- `scripts/snark/setup.sh` extended to compile `mint_batch` at the pinned N set (see *Pinned-N economics* above). Initial pinned set: {16, 32}; promote to {16, 32, 128} once calldata-IC delivery ships.
- The shipped `mint.circom` stays as a reference until the batch path is battle-tested; the small-N adapter then retires.

5.2 Contracts

- `src/Notes.sol`:
  - Drop `_insert()`, `filledSubtrees`, `zeros[]`, `commitmentExists`.
  - Keep `roots[]` ring buffer (now updated by writing the SNARK-attested `newRoot`).
  - Add `nextLeafIndex` invariant check on mint.
  - Replace per-leaf `Appended` event with one batch event (or rely on calldata `cms[]`).

- `src/MintVerifierAdapter.sol` -> per-N adapter under `mapping(uint256 => address) public verifiers` keyed by `N`. `Notes.mint` dispatches by `cms.length`; adapter builds `pub[0..4+N) = [oldRoot, newRoot, nextLeafIndex, totalFace, cm[0..N)]` and invokes the per-N `MintBatchN${N}Groth`
- `src/IMintVerifier.sol` – `verifyMint(proof, oldRoot, newRoot, nextLeafIndex, totalFace, cms[])`.

5.3 Off-chain wallet

- `alberta_buck.wallet.notes` maintains a local mirror of the `filledSubtrees` state; for each batch derives the sibling paths per inserted leaf in order and emits the witness in the new layout.
- `scripts/snark/prove_mint_batch.js` – new prover script.

5.4 Tests

- `test/MintVerifier.t.sol`: end-to-end Groth16-verified mint at the pinned N (currently N=16, N=32 in flight); tamper tests for `oldRoot`, `newRoot`, `nextLeafIndex`, `totalFace`, `cm[]`; per-N dispatch test exercising two pinned sizes coexisting on one adapter.
- `test/Notes.t.sol`: drop the duplicate-cm test; add an economic test that demonstrates duplicate-cm self-punishment (paid twice, spendable once).

5.5 Documentation

- **Urgent first**: `alberta-buck-notes-flow.org` – walk the new mint flow end-to-end; replace the per-leaf insertion narrative with the batched-root-update narrative. Re-validate every "what does the chain see" claim against the new layout.
- `alberta-buck-notes.org`: update Notes-contract state and lifecycle sections; mark the shipped per-leaf path historical, batch-mint primary.
- `alberta-buck-ethereum.org`: rewrite the Notes section and the gas table entries.

6 Related Work: Serialized B1 Sub-Notes

The batch-mint pivot amortizes the cost of *inserting* N leaves into `noteRoot`. An orthogonal, B1-only amortization – *serialized sub-notes* – amortizes the cost of *producing* N bearer notes from *one* leaf, by anchoring a cryptographically committed sub-tree of N pseudorandom serials `s_i` under a single `cm_parent` and spending each against a shared per- parent nullifier bitmap. Mint-circuit reuse is exact: `cm_parent` is an ordinary Poseidon-5 output, and the mint-batch circuit hashes it identically to any other B1 leaf. The spend circuit diverges (adds a log_2(N) Poseidon-2 sub-tree walk) and the on-chain double-spend guard changes from a nullifier mapping to a per-parent bitmap.

Serialization is B1-only: each spend reveals `cm_parent` publicly, linking co-spent siblings. For bearer notes that linkage is free; for A1/A2 it defeats the privacy property.

See `alberta-buck-notes-serialized.org` for the full design, side-by-side data-flow diagrams, per-field security analysis, and cost comparison. Per-bearer-note mint cost drops to ~31K / N_fam vs. ~31K at the batch-mint pivot baseline – another 2-4 orders of magnitude for commonly-sized B1 families.

7 Open Questions

1. **Pinned N set / IC delivery roadmap**: ship N <= 32 with stock Groth16 verifiers first; sequence calldata-IC plumbing before promoting N = 64..128. Decide whether N >= 256 ever ships as a single Groth16 or we leap straight to recursion at that scale. See *Pinned-N economics*.

2. **Per-leaf events vs calldata-only**: keeping `Appended(cm, leafIndex)` eats ~2K gas per leaf in LOG cost; the saving is that off-chain provers can subscribe to the event stream rather than parse calldata. Worth measuring.

3. **Should `Notes` still expose `commitmentCount()`? `nextLeafIndex`** covers the same information; removes an SLOAD-able mirror.

4. **Cross-batch ordering**: if Alice posts batch B1 and Bob posts batch B2 in the same block, the ordering must be deterministic for both provers' `oldRoot` assumptions to hold. Use block ordering and let the loser re-prove; sequencer is the eventual fix.