

A Computational Lower Bound for Bitcoin Chain Validation

Draft — model, assumptions, and methods. No results; all quantities shown as $\langle \text{measure} \rangle$ placeholders or as illustrative magnitudes marked as such.

Eric Voskuil
eric@voskuil.org

August 1, 2026

Abstract

Bitcoin implementations report synchronization performance as wall-clock time on arbitrary hardware, which compares nothing. For network transfer a normalized measure already exists: total block wire size divided by link rate is a lower bound no implementation can beat, and a measured run can stake a claim against it (in practice $\langle \sim 97\% \rangle$ is achievable). No equivalent exists for computation, because there is no natural unit of computation — a bit is a bit, but an operation is not an operation. We construct one: a per-chain vector of required operations, derived from consensus rules and chain data alone, priced by host-measured operation costs, yielding a provable computational floor per host: the best achievable placement of that work onto the host’s resources. The floor is a normalization denominator, not a benchmark: implementations claim a percentage of it, comparisons across heterogeneous hardware become meaningful, and divergence localizes to a resource. We define the model, its admissibility discipline, the measurement method, and a falsifiable validation criterion, and we document the candidate cost factors considered and discarded, with the structural arguments that discarded them.

1 Introduction

1.1 The problem

A claim that an implementation validated the chain in N hours is uninterpretable without a denominator. Hardware varies in core count, clock, instruction sets (SHA-NI, AVX-512, NEON), memory capacity and behavior, and network bandwidth — and these do not trade against each other linearly. The obvious normalizer, threads $\times$ clock, fails empirically: across two of our measurement hosts it predicts a $\langle 4.0\times \rangle$ validation ratio where $\langle 5.5\times \rangle$ is measured, an error of roughly a third, because identical thread-seconds buy different amounts of the specific operations chain validation requires.

1.2 The bandwidth precedent

Milestone (initial block download to a stated height) already has a working normalized measure. The chain’s serialized size is a chain-derived invariant; the link rate is a host-measured coefficient; their quotient is a floor no implementation can beat, because the bytes must cross the wire. The measure is tight — implementations reach $\langle \sim 97\% \rangle$ — for a structural reason worth naming:

the network is a distinct sequential resource, and all other work (parsing, hashing, storing) can execute concurrently on other resources, hiding behind the transfer. The floor bounds the necessarily sequential cost; everything else is *faster than the network*.

Observed milestone times scale proportionally with 2.5/5/10 Gbps links across our fleet (above a memory sufficiency threshold), which is exactly the property that makes the measure useful: proportionality against the binding resource is legible at a glance, and deviation from it is a finding.

1.3 Contribution

We construct the analogous measure for computation:

1. **An invariant operation vector** per chain, derived from consensus rules and serialized chain data, independent of any implementation, emitted at block granularity.
2. **A field cost model** per host: measured coefficients for each operation class.
3. **A floor** as the minimum, over placements of that work onto the host’s resources, of the maximum resource occupancy — with every coefficient taken at its minimum defensible value so the floor is provable rather than estimated.
4. **A validation criterion** — clustering of normalized claims across heterogeneous hosts — that makes the normalizer itself falsifiable.
5. **Negative results:** candidate placements and formulations excluded from the model by structural argument rather than measurement, documented because the exclusions carry as much content as the terms retained.

1.4 Non-goals

The floor is not an estimate of any actual run, not a benchmark, and not a certification of optimality. It is unachievable by construction, as the wire floor is; its value is that the distance to it is attributable.

2 Definitions and Discipline

2.1 The claim

No implementation can validate chain C to height H on host M in less than $T(C, M)$ of elapsed resource occupancy, where T is derived from C and consensus rules alone, parameterized by the measured resources and operation costs of M .

Falsification is a measured run below T . This forces the **one-sided discipline** that governs every modeling choice in this paper: each coefficient must take the minimum defensible value, and each count the minimum any correct implementation must perform. A floor set too low merely loosens the measure; a floor set too high can be beaten, and a beatable floor is not a floor. Consequently, any optimization that is *available to any implementation* must be priced into the floor whether or not any implementation employs it.

2.2 Invariant / field separation

- **Invariant layer (chain-side).** Operation counts derived from consensus rules and serialized chain data. Never observed from an implementation’s behavior — an instrumented implementation measures itself, making the limit circular and unstable under its own improvement. One derivation per chain, valid until the chain extends.
- **Field layer (host-side).** Measured unit costs per operation class per host. Re-measuring a host never requires re-deriving the chain vector, and vice versa.

The single point where execution enters the invariant layer: script **conditionals**. Which signature operations execute, and the sighash type each carries, are not derivable from serialization alone. The scan therefore records, per executed sigop, the *requirement* (input index, sighash type, group arity) — what consensus demanded of the data — from which costs are then computed in closed form. No cost is ever obtained by timing the recording implementation.

2.3 Boundary

Included: all computation any implementation must perform under the placement being evaluated — hashing, signature verification, and, where work is materialized for evaluation elsewhere, the necessary marshalling of that materialization (§6).

Excluded: disk and network I/O, block ordering and chain organization, store structure maintenance, durability (shutdown recovery, snapshot safety), memory-pressure effects (paging, write-back, cache displacement at scale), allocation and table sizing, and implementation complexity. These are the analogue of the protocol overhead excluded from the wire floor: real, universally paid, and not the quantity being measured.

Two boundary rulings that recur:

- **Memory sufficiency is not assumed; it is a term.** See §3.3.
- **A placement’s own unavoidable materialization is inside the boundary; its durability machinery is outside.** A batch verifier that redid lost work after a crash would be computationally identical to one that recovered; recoverability is therefore a node requirement, not a computational one.

2.4 Block granularity and era dependence

Every invariant quantity is emitted per block. Any range — full IBD, a reorg window, the marginal block today, a per-era comparison — is then a sum over blocks, never an average.

This is not a storage convenience but a first-class property of the measure, because the chain’s composition is strongly era-dependent in *opposite directions at once*: the legacy quadratic sighash cost concentrates early, while taproot — and with it the only work-reducing signature lever (§5.3) — concentrates late and grows monotonically. A chain-aggregate scalar therefore answers no question anyone asks. The two questions that are asked — *what does IBD to height H cost?* and *what does the marginal block cost today?* — have materially different answers with different binding terms, and the per-block emission makes both (and any other range) exact sums over the same vector. The floor is consequently a function over ranges, not a number, and every claim against it names its range.

2.5 Resources vary; the implementation does not

There is one denominator per host, not a family indexed by implementation choice. What differs between hosts is the *resource set* — core count and instruction paths, presence and speed of a coprocessor, memory, link — and the floor is computed against whatever that host provides.

This distinction is load-bearing rather than cosmetic. Conditioning the denominator on the approach an implementation happened to take would let a poor choice lower its own floor and inflate its claim: an implementation that marshals work to a coprocessor on a machine that has none would be scored against a denominator that assumed the marshalling was necessary. Because the floor instead takes the *best* placement of the required work onto the host’s actual resources (§3.1), a poor placement appears as a worse ratio — which is what the measure exists to reveal.

Work placement is therefore an optimization *inside* the floor, not a parameter of it. The candidate placements — which signatures are evaluated in line versus marshalled to a device, which multisig formulation is used (§5.2) — are enumerated when computing T , and the placement that minimizes elapsed occupancy defines it. A run claims a percentage of *its host’s* floor, exactly as a sync claims a percentage of *its own* link rate rather than the fastest link on the market.

3 The Resource Model

3.1 Floor as a minimum of a maximum

For a fixed placement A of the required work onto the host’s resources, elapsed time is bounded below by the busiest resource, since a perfect implementation overlaps all of them:

$$T_A = \max \left(\frac{\text{network bytes}}{\text{link rate}}, \frac{\text{CPU work}_A}{W_{\text{effective}}}, \frac{\text{device work}_A}{R_{\text{device}}}, \frac{\text{memory traffic}_A}{BW_{\text{memory}}}, \frac{\text{spill traffic}}{\text{storage rate}} \right)$$

The floor is then the best available placement:

$$T(C, M) = \min_{A \in \mathcal{A}(M)} T_A$$

where $\mathcal{A}(M)$ is the set of placements the host’s resources admit — on a machine without a coprocessor it is a singleton, and the device term vanishes. The inner maximum is a roofline model generalized to a heterogeneous multi-resource workload, subsuming the wire floor as its first term; the outer minimum is what makes the result a property of the host rather than of an implementation’s choices (§2.5).

This also explains why the compute floor is structurally harder than the wire floor. On the network, everything else hides behind the transfer, so one term dominates and the placement question does not arise. On the CPU, all on-core work — parsing, sighash construction, verification, marshalling — competes for the same cores, nothing hides, and the terms sum before the maximum is taken.

Work executed on a genuinely distinct resource enters as its own occupancy term, not as an addition to the CPU’s. This is what makes coprocessor verification modelable: given the hiding inequality (§6.3), device verification overlaps CPU marshalling, and that placement’s CPU term contains marshalling and hashing only.

3.2 Effective, not nominal, capability

Every field coefficient is a measured throughput, never a specification:

- $W_{\text{effective}}$ is measured aggregate scaling, not a core count. For this workload — long dependency chains of 256-bit modular arithmetic — SMT threads measure as approximately equivalent to cores ($\langle \text{measured} \rangle$), so W is the thread count; but this is an empirical result to be re-verified per host, not an assumption. A known caveat: throughput-bound fast paths (SHA-NI ports, wide vector units) should not exhibit the same SMT scaling as the latency-bound EC path. This distinction has not yet been observable because signatures dominate current validation profiles; it is retained in the model because each removal of a dominant term has promoted a previously ignorable one (§7.6).
- **Effective memory** is measured usable working-set capacity, not installed capacity. One fleet host with large installed memory exhibited a platform pathology limiting usable cache to a small fraction of it; a floor priced on installed capacity would have mispriced that machine and hidden the finding (§8.3).

3.3 Memory as a term

Capacity is not a rate and cannot enter the max directly; it gates the spill term. With effective capacity above the working set, spill traffic is zero and the term vanishes. Below it, required spill traffic is priced at the storage rate and can dominate every other term — which is the correct answer to the limiting thought experiment (a machine with 1 byte of RAM has an enormous floor, not an unchanged one). This makes the measure fair across hosts: an under-provisioned machine receives a higher floor rather than an efficiency penalty for hardware it does not have, symmetric with a slow link receiving a higher wire floor.

4 The Hash Term

4.1 Unit: per-class affine cost

Neither bytes nor SHA-256 compression invocations are an adequate unit. Bytes fail because padding dominates at Bitcoin’s message sizes. Compressions fail because they are not fungible: measured double-hash of a 32-byte message costs $\langle \sim 1.5 \times \rangle$ a single 32-byte hash, not $2 \times$, because the second pass reuses the buffer, injects state without endian transition, and draws on precomputed padding; and per-message costs (input, finalization, pad scheduling) are comparable to per-compression costs at small sizes.

The unit is therefore an affine model per class:

$$\text{cost}(\text{class}) = a_{\text{class}} \cdot \text{calls} + b_{\text{class}} \cdot \text{data blocks}$$

where the classes mirror the hashing algorithm’s interface — merkle pair hashing, double hash, single hash, streamed/accumulated hash, and RIPEMD-160 — because the interface is where the cost paths actually diverge (precomputed pad schedules for whole-block multiples, cached tag midstates, fused double-hash finalization). The a/b split is exactly what a size-sweep benchmark separates, so the field layer’s measurement is the harness that already exists.

The reference hashing library is asserted to contain all hash optimizations known to be feasible in practice (hardware SHA extensions; multi-buffer vectorization; single-message schedule vectorization; precomputed and cached padding; double-hash fusion; midstate caching). The floor’s hash coefficients are measured at this frontier. If the frontier moves, the coefficients — not the model — change.

4.2 Chain-side derivation

All hash counts are closed-form functions of serialized data plus the executed-sigop record:

- **Identity** (txid, wtxid): serialized sizes; wtxid omitted where identical to txid.
- **Merkle** (both trees, witness commitment): from per-block transaction counts, with the odd-duplication rule.
- **Sighash preimages**: lengths derivable per (input, scriptCode, sighash type) triple — legacy by the modified-serialization rules, BIP143 and BIP341 by fixed layout. The sighash type comes from the executed-sigop record; the length arithmetic is analytic.
- **Distinct-preimage rule**: preimages shared across signatures of one transaction (per-transaction midstates; same input, same hashtype) are counted **once**, because the data determines their identity — this is a fact about the chain, not a cache credit to an implementation (§7.2).
- **Script-level hashing**: HASH160, SHA-256, and RIPEMD-160 operations from script content.

4.3 The legacy quadratic term

Legacy (pre-BIP143) sighash cost is quadratic in transaction shape — each input hashes a modified serialization of the whole transaction — and its distribution is violently heavy-tailed; single historical transactions contribute gigabytes of preimage individually. This is the one term that cannot be modeled from aggregates and must be **summed exactly** per input during the scan. It is also strongly era-concentrated, which the block-granularity emission preserves.

4.4 Vectorization admissibility and the lane cascade

Multi-buffer hashing (independent messages across 4/8/16 SIMD lanes) is measured at $\langle \sim 6\times \rangle$ canonical throughput at full width. Its admissibility per class is governed by one criterion, developed in §6.2: **lanes must fill without leaving the scope the data is already in**. Merkle hashing qualifies uniquely — the digests are already a flat block-scope array — and is priced at the vector coefficient with an exact per-level cascade model: full lane groups greedily consumed at the widest available width ($16 \rightarrow 8 \rightarrow 4$), remainder at the sequential rate, driven by the per-block transaction-count distribution. Partially filled lanes are never used (measured as a regression), so small blocks correctly receive no benefit.

All other classes — identity, sighash preimages, script hashing — fill lanes only by crossing

transaction scope, which is marshalling, and is excluded by the argument of §6.2. They are priced at the sequential coefficient: $\min(\text{SHA-NI}, \text{canonical})$ per host.

4.5 RIPEMD-160

RMD-160 has no hardware acceleration on any host. Its cost relative to SHA-256 therefore *inverts* across hosts: $\langle \sim 4\times \rangle$ a SHA-NI compression, but *cheaper than* a canonical SHA-256 compression on hosts without SHA extensions. It is kept as a separate class precisely because its ratio to the SHA coefficient is not a constant. A corollary worth stating: hardware SHA extensions amplify RMD’s *share* of the hash term by the same factor they remove from everything else.

5 The Signature Term

5.1 Counts

Per block, by scheme: ECDSA single verifications; ECDSA multisig groups as a (k, n) **histogram**; Schnorr verifications, including tapscript threshold (CHECKSIGADD) rows — admissible in batch because BIP342 requires every non-empty signature to verify, so all-or-nothing evaluation is sound for them. Public-key parses are counted separately and are **placement-dependent** (§6.1).

5.2 Multisig: cost is not separable

The multisig term cannot be written as $\text{count} \times \text{rate}$, because count and achievable rate couple through the evaluation formulation, and which formulation is best depends on the resource it runs on. Three formulations bound the space:

formulation	verifications per k -of- n group	character
ordered walk	W , with $k \leq W \leq n$	sequential; W is data-determined (the consensus ordering rule makes the walk deterministic, so W is a chain property, not an implementation property)
band matrix	$k \cdot (n - k + 1)$	all cells independent; statically determinable from script alone; more work, less time on throughput substrates
key recovery	$\sim 2k$, independent of n	recover candidate keys from (r, s) and test set membership

These are candidates in the placement optimization of §3.1, not alternative denominators: the floor takes the per-histogram-cell minimum at the rates of whichever resource the work is placed on. The band’s extra verifications buy independence, which is worth nothing on a substrate that has no idle parallelism to absorb them and everything on one that does — so the walk wins on cores and the band can win on a device. The static bound $\Sigma k \leq \Sigma W \leq \Sigma n$ brackets the walk’s cost without execution; the envelope is narrow because the histogram concentrates on cells with $n - k \leq 1$.

Key recovery will never be implemented — the population where it wins (large n relative to k , mostly early bare multisig) is negligible — but it remains in the model because the floor must take the true minimum: with it priced, the floor is unbeatable by a future implementation of it, and the model itself quantifies why building it is not worthwhile. The same holds for the band matrix on cores (§7.1).

5.3 Schnorr batch verification

BIP340 signatures admit algebraic batching: a random linear combination collapses u verifications into one multi-scalar multiplication, a genuine work reduction (unlike throughput batching, which redistributes unchanged work). The reduction is capped well below the naive multi-scalar figure by a per-signature cost that does not batch: lifting each R from its x -coordinate is a field square root, unavoidable, $\langle \sim 10\text{--}15\% \rangle$ of a verification. Net $\langle \sim 2\text{--}2.5\times \rangle$, flattening beyond batch sizes of $\langle \sim 1k \rangle$; batch size is a free tuning parameter over a uniform captured stream, not a per-block constraint. Whole-batch failure is acceptable for proof-of-work-committed data (essentially never exercised in IBD; the cold path iterates to attribute the failure) and inadmissible for unconfirmed relay — the floor concerns the former.

Batching requires capture, so the Schnorr term is a min between inline and capture-plus-batch (§6.3). Because the reduction is available to any implementation, it is priced into the floor regardless of implementation status.

5.4 ECDSA batch impossibility

The same construction fails for ECDSA on existing chain data: DER signatures omit the recovery bit, each signature's R has two candidate parities, and forming a batch equation over u signatures requires guessing 2^u combinations — while determining parities individually *is* the verification. This asymmetry between the schemes is load-bearing for the whole model: it is why the dominant (ECDSA) term has no work-reducing lever and why Schnorr is treated separately. (Key recovery yields R 's parity but only by completing the verification, so it cannot rescue the batch.)

6 Marshalling and the Placement Optimization

6.1 Marshalling cost is necessary computation

Any placement that evaluates signatures away from the point of script execution — device dispatch, or in-core Pippenger batching — must first materialize the work: intercept the script pattern, extract and normalize the tuple, allocate and write rows, read them back at drain, and correlate results to inputs. Every implementation adopting such a placement pays this; it is therefore inside the boundary, as per-row computational constants (c_{cap} per scheme), while the orchestration that makes the capture durable and recoverable is outside.

The constants are **measured, not derived**, by differential runs: one run capturing and discarding (no verification), one run verifying inline (no capture); the difference isolates capture. Field experience motivates taking this seriously: measured capture cost on a highly optimized path is a large fraction of an inline ECDSA verification ($\langle \sim 50\% \rangle$ observed) — far above what enumerating the row operations suggests, which is precisely why it must be measured rather than modeled.

Capture is priced at 100% coverage. The small fraction of signatures whose scripts resist cheap static determination could be captured by executing to determine — the barrier is code complexity and table width, both outside the boundary — so the floor uses the uniform constant rather than a blended one.

Lossy canonicalization. A capture row is a canonical form, and canonicalization that discards information forces recomputation downstream: compressing an uncompressed public key into a fixed 33-byte row obliges the consumer to recover y by square root — an operation inline evaluation never performs. Two consequences: the public-key-parse count is **placement-dependent** (the same chain data yields different counts under different placements, so it is not a single chain-side scalar), and row-format decisions trade a per-row width cost multiplied across the entire population against recomputation that may land on a resource with idle capacity. Where the recovery lands on a device hidden behind marshalling, the narrow row wins outright; the general rule — size the row for the dominant case, let the minority pay recomputation on the hidden side — is a modeling result, not merely an engineering preference.

6.2 The viability criterion — and the negative result

Out-of-line evaluation pays only where per-operation cost greatly exceeds per-operation marshalling. The marshalling floor is set by scope escape: filling a batch requires leaving the opcode, the script, and the transaction, and that control-flow cost — not the copy — is the significant term.

Where the operation classes sit against any plausible capture constant:

- ECDSA / Schnorr verification: $\langle \sim 2 \text{ orders of magnitude} \rangle$ above — capture unconditionally viable.
- Every hash class (HASH160, fixed-layout sighash preimages, identity): the **same order** as the constant — marshalling is a total loss under every placement, on every host.

This closes the entire hash-batching question — multi-buffer capture of uniform preimages, fused vectorized HASH160 (SHA→RMD in transposed lanes, compare against targets without unpacking), device offload of preimage construction — as a structural negative result requiring no further measurement. These designs are retained in §7 because the criterion that kills them is one of the paper’s transferable results. It also explains, retroactively, why merkle is the only vectorized hash path in a mature implementation: it is the only class whose lanes fill in-scope.

6.3 The floor under available resources

The placement set $\mathcal{A}(M)$ has two members where a device is present and one where it is not. Evaluating T_A for each:

Signatures evaluated in core (the only placement available without a device):

$$\begin{aligned} \text{CPU work} = & \left[\text{Hash} + c_{\text{ecdsa}} \cdot (N_{\text{single}} + \Sigma W_{\text{min}}) \right. \\ & \left. + \min(c_{\text{schnorr}} \cdot V_s, (c_{\text{cap},s} + c_{\text{batch}}(u^*)) \cdot V_s) \right] \end{aligned}$$

Signatures marshalled to a device:

$$\begin{aligned}\text{CPU work} &= \text{Hash} + c_{\text{cap},e} \cdot N_{\text{ecdsa}} + c_{\text{cap},s} \cdot V_s \\ \text{device work} &= N_{\text{device}}\end{aligned}$$

with T_A in each case the maximum of that placement’s occupancies per §3.1, and $T = \min_A T_A$.

- $\Sigma W_{\min}$ is the per-cell formulation minimum of §5.2, evaluated at the rates of the resource the work is placed on.
- Where $R_{\text{device}} \geq$ the marshalling rate — the *hiding inequality*, verified once per host — the device term is not the maximum, and that placement’s cost **is** its marshalling plus the hash term. Otherwise the maximum selects $N_{\text{device}}/R_{\text{device}}$ and the device binds.
- Expansion of multisig group rows into band cells is replication of already-computed values — memcopy, outside the boundary — so marshalling scales with distinct signatures, not cells, and the expansion site does not affect the floor.
- Note the second placement is admissible only where a device exists. Marshalling to no device is not a placement; in-core throughput batching of ECDSA is dominated outright (§7.1), which is why the first expression carries no $c_{\text{cap},e}$ term.

7 Considered and Discarded

Each entry records the candidate, the argument that removed it, and what the removal contributed. Several removals are the model’s strongest content.

7.1 Discarded by structural argument (no measurement needed, on any host)

- **In-core throughput-batching of ECDSA** (band matrix or any parallel dispatch onto the same cores). In-core verification is already embarrassingly parallel at input/group granularity; batching adds no parallelism on a resource that has none spare, so its benefit is zero against positive marshalling cost. Dominated a priori — the placement is removed from $\mathcal{A}(M)$ by argument, not measured. *Contribution: the pruning argument itself, reused throughout.*
- **Hash-class marshalling in all forms** (§6.2): uniform-preimage multi-buffer via capture, fused HASH160 batches, hash tables for device dispatch. Killed by the viability criterion: operation cost $\approx$ marshalling cost. *Contribution: the criterion, and the explanation of why merkle is uniquely vectorizable.*
- **ECDSA algebraic batching** (§5.4). Impossible on DER-encoded chain data, not merely unprofitable. *Contribution: the scheme asymmetry that shapes the whole signature term.*

7.2 Discarded as category errors during model construction

- **Instrumenting the implementation’s hash/verify calls to obtain counts.** Measures the implementation, not the chain; circular, and unstable under the implementation’s own improvement. Replaced by closed-form derivation plus the minimal executed-sigop record.
- **Demanded/performed accounting** (crediting caches for work avoided). A cache is an implementation artifact; the *data’s* identity structure is what determines the minimal

count. Preimages identical by construction are counted once because consensus makes them identical, not because something cached them. Performed-minus-demanded belongs to the gap above the floor, never inside it.

- **k as the multisig floor.** The sig→key assignment is not recoverable from the data by inspection; recovering it *is* the verification work. A floor of k is unreachable by any implementation — a fiction that would make the floor unbeatable only in the sense of being meaningless. Replaced by the formulation minimum of §5.2.
- **A single static multisig minimum.** No formulation dominates all (k, n) cells on all resources; the term is irreducibly a min taken inside the placement optimization — the non-separability result.
- **A family of denominators indexed by the implementation’s approach.** Scoring a run against the floor of the path it chose lets a poor choice lower its own denominator and inflate its claim. Replaced by one floor per host, with placement optimized inside it (§2.5).

7.3 Discarded units

- **Bytes** (padding dominates at Bitcoin’s sizes) → compressions.
- **Compressions** (non-fungible: per-message costs comparable to per-compression costs; double-hash fusion) → messages + compressions.
- **Messages + compressions** (still misprices fused and pad-cached paths) → the per-class affine model of §4.1, with classes mirroring the algorithm interface. Each refinement was forced by a measured discrepancy, recorded in Appendix B.
- **threads × GHz as the normalizer.** The motivating failure (§1.1); measurably wrong by $\langle \sim 38\% \rangle$ across real hosts.

7.4 Discarded boundary candidates

- **Page dirtying / write-back pressure.** Vanishes with sufficient effective memory; a field condition, not a computational term. Memory enters as capacity gating the spill term (§3.3), not as a dirty-page cost.
- **Durability of capture** (shutdown recovery, snapshot safety). Node requirement; a computationally identical placement could redo lost work.
- **Allocation and table sizing.** Including the instructive extreme: widening every capture row to accommodate the uncompressed-key minority trades a huge memory cost for zero computational saving — the format question is real engineering but its resolution does not move the floor.
- **Code complexity** — including as the true cost of the last $\langle \sim 0.5\% \rangle$ of capture coverage, which is why the floor prices capture at 100% (§6.1).

7.5 Considered and retained in unusual roles

- **Key recovery:** retained in the min specifically so the model can demonstrate it is never worth implementing (§5.2).
- **Device preimage-construction offload:** bytes-per-compression analysis showed larger hash classes offload *better* than the naturally marshalling-shaped small ones — but the entire branch then fell to §6.2. Recorded because the intuition it reverses (small uniform items batch best) is common.

- **Cross-block merkle lane filling:** eliminates the structural lane-starvation tail of small trees; retained as the $\min(\text{per-block, cross-block})$ form of the merkle term, though its value is confined to the early chain.

7.6 A recurring pattern

Three times during construction, removing a dominant term promoted a previously negligible one: signature dominance made hash precision irrelevant $\rightarrow$ device placement removed signatures and made hashing binding $\rightarrow$ within hashing, the legacy quadratic sum became the precision-critical term; likewise SMT scaling differences and RMD’s share are invisible until the terms above them are removed. This is the argument for keeping the vector fully decomposed even where terms look ignorable today: **you cannot know which term is promoted next without having kept it separate.**

8 Method

8.1 The scan (chain-side, once per chain)

Per block: hash calls and data blocks per class (§4.1); the transaction-count value (merkle cascade input); the (k, n) multisig histogram; the executed-sigop record (input index, sighash type, group arity); the exact legacy sighash compression sum (§4.3); signature counts by scheme including threshold rows; public-key format counts (for the placement-dependent parse term). The record is reproducible: any party can recompute the vector from chain data, and disagreement localizes to a term.

8.2 The harness (host-side, per host)

- (a, b) per hash class from message-count $\times$ size sweeps, per available instruction path; the field coefficient is the min over paths per class.
- $c_{\text{ecdsa}}, c_{\text{schnorr}}$ inline; $c_{\text{batch}}(u)$ sweep for the Pippenger term.
- c_{cap} per scheme by the differential method of §6.1.
- $W_{\text{effective}}$ by aggregate throughput scaling (per workload class, given §3.2’s caveat); effective memory by working-set measurement; link rate; R_{device} once, to establish the hiding inequality.

8.3 Validation

The criterion is clustering, not any single ratio. A correct normalizer produces tight actual/floor percentages across heterogeneous hosts, as the wire measure clusters at $\langle \sim 97\% \rangle$. Dispersion is the normalizer’s error, making the construction falsifiable. The measurement fleet spans $\langle 3 \rangle$ operating systems, $\langle 4 \rangle$ instruction-set profiles (avx512+shani, avx2+shani, avx2, neon+sha2), $\langle 4\text{--}128 \rangle$ threads, and $\langle 16\text{--}768 \rangle$ GiB — heterogeneity is what gives the test discriminating power.

Residuals are findings. The measure’s purpose is not the percentage but what divergence localizes:

- A host binding on memory rather than network during milestone (observed: the small-memory SBC at $\langle 1.28\times \rangle$ its bandwidth-implied time) — predicted by the memory term.

- A host whose effective memory is a fraction of installed memory (observed: the macOS pathology) — independently discovered, subsequently fixed, with a large measured improvement. The feedback loop the measure exists to drive has already executed once, manually; the model automates its first step.
- The same-host device/in-core pair as a predictive test: measure c_{cap} by the differential method, *predict* the ratio between the two placements, compare to the measured pair. A fit to the pair itself validates nothing; the prediction does.

What scaling reveals. With per-host floors in place, cross-host comparison separates three outcomes wall-clock conflates: proportional scaling against the binding resource (healthy), binding on an unexpected resource (host or configuration finding), and non-scaling against the expected resource (implementation limit).

9 Results

Placeholder. Gated on the scan of mainnet to height $\langle H \rangle$ and harness runs across the fleet. To be reported:

- *The invariant vector for mainnet to height $\langle H \rangle$.*
- **Era decomposition (§2.4) as a primary result, not a breakdown:** *the floor as a function of height range; the opposing era profiles (legacy quadratic early, batchable Schnorr late) quantified; the IBD floor versus the marginal-block floor as distinct denominators with distinct binding terms; the taproot-share trajectory that determines when the Schnorr batch term becomes material.*
- *Per-host floors, with the selected placement reported; measured runs and the clustering figure (the lead validation result).*
- *Sensitivity analysis — which terms move the floor (the legacy quadratic sum; c_{cap}) and which provably cannot (recovery-favorable cells; Schnorr share at current adoption).*

10 Related Work

Placeholder: roofline modeling; libsecp256k1 batch-verification literature; SHA-NI and multi-buffer hashing literature; assumevalid/checkpointing as validation-avoidance (orthogonal: we bound performed validation, not whether to perform it).

11 Conclusion

A bit is a bit; an operation is not an operation. A chain-derived operation vector priced by measured host rates recovers, for computation, the normalizing power that bandwidth has always had for transfer — turning “is this implementation fast?” into a question with a denominator, and turning divergence from the denominator into an attribution rather than an anecdote.

A Closed-form preimage lengths

Legacy modified-serialization length per (input, scriptCode, hashtype), including varint deltas; BIP143 layout ($156 + |\text{scriptCode}|$, per-tx midstate inputs); BIP341 layout (fixed $175 +$

annex/script-path extensions, tag midstates); the compression function $C(L) = \lceil (L + 9)/64 \rceil$ and the distinct-preimage rule.

B The affine hash model

Fit procedure, per-class residuals, and the measured discrepancies that forced each unit refinement (§7.3), including the double-hash fusion measurement and the pad-cache decay curve.

C Formulation-minimum arguments

Walk determinism under the consensus ordering rule (W as chain property); band static determinability; recovery construction and its ≤ 2 practical candidates; per-cell dominance regions; ECDSA batch impossibility on DER.

D Scan record schema

Per-block record layout, versioning, and reproduction procedure.