SoK: Speed and Its Costs — A Systematization of High-Performance Layer-1 Blockchain Architectures

Mengzhong Ma
Nanyang Technological University
mengzhon001@e.ntu.edu.sg

***Abstract*—Between 2019 and 2026 the performance envelope of public Layer-1 blockchains moved by two orders of magnitude, and the designs that produced the shift are documented one system at a time. This survey takes the assembled chain as its unit of analysis and systematizes the monolithic high-throughput Layer-1 across six layers: dissemination, ordering, execution, state, metering, and the operational envelope that determines who can run a validator. Each layer receives a taxonomy on axes shared across exemplars and a costs subsection in which every speedup mechanism is paired with its documented drawback and an evidence grade recording whether that drawback is proved, measured in a laboratory, observed in production, or asserted in a whitepaper. From a screened corpus of 810 records we assemble 200 mechanism-cost pairs over 131 mechanisms and audit 299 published performance claims. Three results stand out. Thirty of the 200 costs are paid at a different layer from the one that earns the benefit, which is the structure a technique-first taxonomy cannot represent. Only 37 of the 200 cost claims rest on production measurement, and the two systems with the largest deployed throughput have no archival description of their core mechanisms. Of 92 numeric throughput claims, 14 state a validator count that can be corroborated in the source text. We close with five open problems, three of which are measurement gaps that a single well-designed study would close.**

***Index Terms*—Blockchain, consensus protocols, parallel execution, performance evaluation, distributed systems, systematization of knowledge.**

I. Introduction

The performance envelope of public blockchains moved by two orders of magnitude between 2019 and 2026, and it moved through engineering rather than through a single conceptual breakthrough. Separating transaction dissemination from ordering lifted a leader-based protocol from 1,800 transactions per second at one second of latency to 170,000 transactions per second at 2.5 seconds, and adding dissemination workers per validator pushed the same construction to 600,000 transactions per second with no latency penalty [1]. Removing certification from the DAG then bought back the latency, reaching half-second wide-area commit latency while sustaining over 200,000 transactions per second [2]. On the execution side, treating a committed block order as a speculation oracle rather than a constraint yielded up to 170,000 transactions per second on 32 threads [3]. Each of these results is real, reproducible, and reported by its authors with the conditions attached. None of them is a statement about a blockchain.

The gap between those two things is the subject of this survey. A deployed high-throughput Layer-1 is an assembled stack: an admission and dissemination layer, an ordering layer, an execution engine, a state store, a metering and fee mechanism, and an operational envelope that determines who can afford to run a validator. Its speed is a property of the composition, and so are its failure modes. Fault-injection benchmarking makes the point concretely: when a high-throughput chain is driven through a fault and recovery cycle, the degradation that persists afterwards can be traced not to the consensus protocol but to the interaction between the recovery path and the speculative executor [4]. A survey organized around consensus protocols cannot express that finding, and neither can a survey organized around concurrency control.

This is the state of the literature. We screened 34 surveys, reviews and systematizations that a reader might reasonably consult on this question and coded each one for layer coverage, for whether Solana, Aptos, Sui or Monad are objects of study rather than passing mentions, and for whether mechanisms are systematically paired with their costs. Twenty-nine address ordering and consensus; exactly one addresses metering and fee mechanisms. None covers all six layers, none takes the current generation of high-throughput systems as its subject, and two pair mechanisms with costs systematically. The comparative material that does exist at the system level is investor-facing grey literature in which advertised peak figures, laboratory benchmarks and mainnet observations are quoted interchangeably.

The numbers themselves are the second problem. We assembled every quantitative performance claim we could extract from the corpus and from the grey literature, 299 in total, and checked each extracted figure against its source text. Ninety-two are numeric throughput claims. For only 14 of those could we corroborate the validator count in the source text by requiring the number to appear adjacent to a validator, committee, replica or machine term rather than merely nearby, and just 6 of the 14 come from the system's own paper rather than from a competitor's related-work paragraph. The spread of the 92 figures runs from tens to hundreds of thousands of transactions per second, and without the measurement conditions the spread is uninterpretable: a figure obtained

on four co-located replicas with 512-byte no-op payloads and a figure obtained on a geo-distributed committee under contended application load are not the same kind of fact, and they are routinely tabulated in the same column.

A. Contributions

1. A six-layer decomposition of the high-performance Layer-1 stack, with an explicit taxonomy at each layer whose axes are shared across exemplars rather than invented per system.
2. A cost-first treatment in which every speedup mechanism is paired with its documented drawback and an evidence grade recording whether that drawback is proved, measured in a laboratory, observed on a production chain, or asserted in a whitepaper.
3. A cross-layer trade-off matrix that makes co-design effects visible, including the cases where a mechanism's cost is paid at a different layer from the one that provides its benefit.
4. A claim-provenance audit of published throughput and latency figures, tagged with validator count, geography, workload and contention, that separates comparable results from incomparable ones.
5. A systematic corpus and search protocol: 4,916 records retrieved from keyword and citation expansion over OpenAlex and arXiv, 3,663 unique, screened to a working corpus of 810 with explicit inclusion criteria and a separately labelled grey-literature track.

B. Scope

Our subjects are monolithic Layer-1 designs whose stated target is high single-shard throughput with sub-second to second-scale finality: Solana, Aptos, Sui and Monad in the first tier, with Sei, Fuel, Linera, MegaETH, Diem, Avalanche and Algorand as secondary cases. Ethereum, rollups and classical sharded designs appear as contrast cases and not as subjects, because layer-two and sharding scaling already have dedicated surveys and because the interesting question here is what happens when the scaling is pushed into a single execution domain instead of across many.

Two consequences of this scope need stating at the outset. First, a substantial part of the design space we survey has no peer-reviewed description. Solana's proof-of-history clock, its Sealevel executor and Turbine dissemination, and Monad's execution and storage engines exist as whitepapers, documentation and engineering write-ups. We cite these sources as design statements, never as performance facts, and we mark every such citation as grade E4. Second, our own reading of the mechanisms is necessarily an outside view: we did not operate validators, and where a claim could only be established by running one, we say so rather than inferring it.

C. Roadmap

Section 2 defines the terms and presents the pipeline decomposition. Section 3 positions the survey against prior systematizations. Sections 4 through 9 work up the stack, one layer per section, each closing with a costs subsection. Section 10 assembles the trade-off matrix; Section 11 presents the claim-provenance audit; Section 12 sets out the open problems we consider defensible, in the order we consider them tractable.

II. Background: what a transaction costs a chain

A useful decomposition of a high-throughput Layer-1 follows the life of a transaction rather than the module boundaries of any particular implementation. A client submits a transaction to some set of validators, which must decide whether to admit it; the transaction must be disseminated so that enough validators hold its bytes to reconstruct a block; some mechanism must fix a total order over the transactions that will be applied; the transactions must be executed against current state, which is where parallelism lives; the resulting state transition must be committed to a persistent, authenticated store; and the resources consumed at every one of those stages must be metered and priced, or the system has no defence against its own most profitable workload. The six layers we survey correspond to those stages, in that order, plus the operational envelope they jointly impose on node operators.

The decomposition matters because the stages have different scaling behaviour and different bottleneck resources. Dissemination is bandwidth-bound and embarrassingly parallel across validators, which is why it responds to horizontal scale-out: the linear throughput improvement obtained by adding dissemination workers per validator is a statement about bandwidth provisioning, not about agreement [1]. Ordering is latency-bound and fundamentally sequential in its commit rule, so its costs are counted in message delays and in what happens when a designated leader is slow or Byzantine. Execution is CPU-bound and contention-bound, so its throughput depends on a property of the workload rather than of the system: how much of the block can run concurrently without violating the committed order's semantics. State commitment is I/O-bound and grows monotonically with history. Metering is an economic mechanism, not a resource, and its failures show up as mispriced congestion rather than as saturation.

Three definitional points recur throughout, and the literature is not consistent about them. *Throughput* is reported either as transactions per second at the point of consensus commit or as transactions per second of completed execution; the two differ whenever ordering and execution are decoupled, which is precisely the case in every system we survey. *Latency* is reported as client-observed end-to-end time, as consensus commit latency, or as message delays to commit in a model; the first includes admission and execution, the second does not, and the third is unit-free. *Finality* is used for both the deterministic commit point of a BFT protocol and the probabilistic settlement assurance of a chain under partial synchrony, and in systems with a fast path for uncontended objects it has two distinct values depending on which path

a transaction took. Where we quote a figure, we name which of these it is; where a source does not, we record that as a property of the source.

The baseline against which this generation defines itself is a single-threaded, globally-ordered, globally-metered execution model in which every validator re-executes every transaction serially, dissemination and ordering are fused into one gossiped block, and one scalar resource unit prices all consumption. Each layer in the sections that follow is a departure from one clause of that sentence, and each departure buys throughput or latency at a cost that is paid somewhere else in the stack. The taxonomies we present are organized around exactly that question: what does this mechanism assume, what does it buy, and where does the bill arrive.

A. Threat and deployment model

We assume the standard partially-synchronous Byzantine model for the ordering layer, with the usual bound of fewer than one third faulty voting power, and we note where a protocol's performance claim depends on synchrony that the model does not guarantee. For the execution and state layers, the relevant adversary is often not Byzantine but economic: an adversary who submits workloads chosen to maximize conflict, to inflate state, or to exploit a metering dimension that is underpriced relative to the resource it consumes. Several of the costs we document are only visible under that adversary, which is one reason they are under-reported in evaluations designed to demonstrate peak throughput.

III. Prior systematizations and what this survey adds

The scalability literature is mature enough to have its own canon, and that canon is organized by *technique*, not by *system*. We screened 34 surveys, reviews and SoKs that a reader might reasonably expect to answer the question "how do today's high-throughput Layer-1 chains get their speed, and what does it cost them", and coded each for which of our six layers it covers, whether Solana, Aptos, Sui or Monad are objects of study rather than passing references, and whether it systematically pairs mechanisms with their drawbacks. None covers all six layers; none takes the tier-one systems as subjects; two pair mechanisms with costs systematically. Coverage is concentrated in one place: 29 of the 34 address ordering and consensus, while exactly one addresses metering and fee mechanisms, and seven address execution parallelism.

Consensus is where the field has been most thorough. The protocol-level surveys establish the design space of agreement itself [5–7], and one protocol family has now been reviewed in its own right [8]. DAG-structured designs — the structural innovation behind Sui's and Aptos's ordering layers — have two systematizations, one of DAG ledgers as system architectures [9] and one classifying DAG consensus protocols along an availability-versus-consistency split [10], while the techniques that make BFT scale to large validator sets are systematized separately [11]. What this body of work deliberately brackets is everything the consensus protocol hands off to: an ordering protocol that commits 200,000 transactions per second is a claim about dissemination and agreement, not about a chain, and the surveys are careful not to overclaim it.

The scaling surveys, in turn, are organized by where an intervention sits rather than by what a deployed chain does. The most complete recent example spans hardware acceleration through layer-two protocols [12], and the genre includes broad scalability reviews [13, 14], the boundary-drawing survey of distributed-ledger designs [15], and branch-specific treatments of sharding [16], coded storage and dissemination [17], storage growth [18] and networking [19]. Their exemplars are predominantly Bitcoin and Ethereum, which is the correct choice for a technique-first taxonomy and the wrong one for a reader who needs to know what Sealevel, Block-STM and Sui's owned-object path actually are.

Two recent efforts come closest to our framing and are worth stating precisely. Chahoki et al. [20] review blockchain concurrency as a research literature, covering concurrent execution, validation and sharding, and are the natural companion to our execution layer; they classify published techniques rather than deployed engines, and do not cross into ordering, metering or operational cost. Aliyu et al. [21] survey how the blockchain trilemma is operationalized and find the constructs and metrics too inconsistent to support comparison across studies — a result we take as a design constraint rather than a related work, since it is the reason our comparison chapter tags every performance figure with the conditions under which it was obtained.

Performance evaluation has its own small literature, and it is the part of the field most directly at odds with the numbers in public circulation. Systematic treatments of how blockchain performance is measured [22] and modelled [23] both predate or sidestep the current generation of chains, and the one framework that evaluates blockchains on common ground with realistic decentralised applications reports that none of the six systems it tested could support those application workloads, and that published results are typically obtained in isolation by the systems' own developers with experimental conditions omitted [24]. In our own audit of 239 published performance claims, of the 65 that give a numeric throughput figure only 22% state the validator count it was obtained with; that ratio, not any single headline number, is why a claim-provenance chapter belongs in this survey.

This survey therefore differs from its predecessors in three specific ways. It takes the assembled chain as the unit of analysis, so that co-design effects — an ownership-typed state model that only pays off because the consensus layer offers a fast path, a speculative executor that only pays off because the ordering layer fixed a block order first — are expressible rather than split across two taxonomies. It treats each mechanism's cost as a first-class object with an evidence grade attached, rather than as a "limitations"

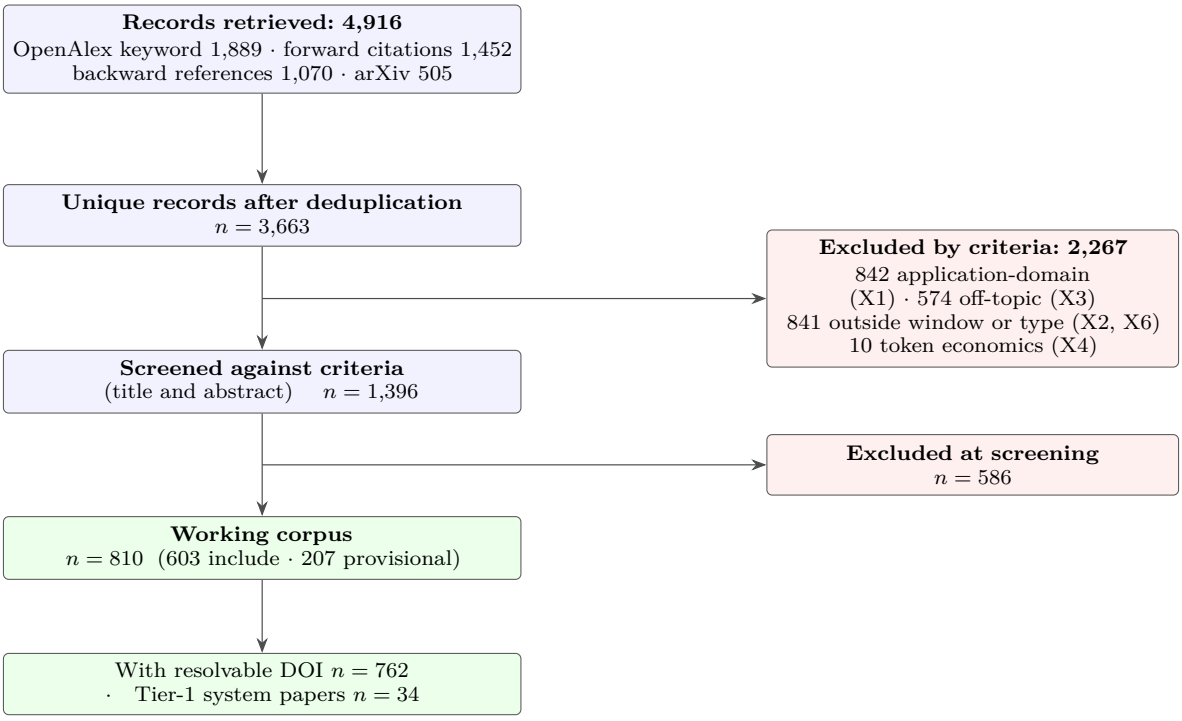


Fig. 1. Corpus construction. Records were retrieved by keyword search and one-step forward and backward citation expansion, deduplicated, screened against the registered inclusion and exclusion criteria, and reduced to the working corpus from which the layer sections were written. Exclusion codes refer to the criteria in Appendix A.

paragraph. And it reports performance claims with their measurement conditions, which converts an unusable set of headline numbers into a comparison a reviewer can check.

IV. Layer 1: Dissemination and networking

A. Design axes for the dissemination layer

The defining architectural commitment of the high-performance monolithic L1 is that data movement is no longer a subroutine of agreement. In the classical state-machine-replication formulation these systems inherit [25, 26], a leader proposes a block, the block is propagated, and replicas vote on it, so propagation sits inside the critical path and is charged to latency on every decision. Every deployed high-throughput L1 breaks this coupling, and how it does so is the most consequential choice in its stack. We classify dissemination mechanisms on five axes: whether disseminated data is *certified* — carries a quorum certificate proving availability — before the ordering layer may reference it; the *propagation structure* (mesh gossip, structured overlay, tree, or erasure-coded fan-out); whether the mechanism *scales out horizontally* by adding machines per validator; the *redundancy factor*, in bytes transmitted per byte of useful payload; and the *failure mode* when a participant is slow rather than crashed. The last axis carries most of the documented costs, and it is where the networking and broadcast surveys [19, 27] have least to say, cataloguing propagation structures thoroughly but treating the consensus protocol above them as an opaque consumer.

B. Decoupling dissemination from ordering

Narwhal made the separation explicit and remains the reference design for the Move-language family. It is a mempool that reliably disseminates and stores causal histories of transactions independently of any ordering protocol, so consensus above it agrees only on small certificates rather than payloads [1]. Two consequences follow, both measured on a five-region AWS deployment of m5.8xlarge instances with 10 Gbps links. The first is a level shift: Narwhal composed with HotStuff yields over 130,000 tx/s at under two seconds, against 1,800 tx/s at one second for HotStuff driving its own propagation. The second matters more architecturally — adding worker machines per validator raises throughput linearly to 600,000 tx/s *with no latency increase*, because the workers are not on the agreement path. Horizontal scale-out of dissemination is on this evidence (E2_lab) the largest single throughput lever in the stack, and it becomes available only once certification of availability is separated from ordering.

The same decoupling is achievable without a DAG, and the alternatives clarify what the DAG buys. Stratus uses a provably-available broadcast: a microblock is pushed to peers and an availability proof of at least f+1 signatures returned, with consensus proposing proofs rather than payloads [28]. This gives O(n) message complexity against $O(n^2)$ for Tendermint, Narwhal and Mir-BFT [29] and 5× to 20× the throughput of native HotStuff, Streamlet and PBFT at 128 replicas; being agnostic to the host protocol's view change, it is a drop-in for leader-based designs in a way a DAG mempool is not. DispersedLedger works from the

coding side, erasure-coding each block so a replica votes on *dispersal* and retrieves contents later [30]; its AVID-M construction costs O(|B|/N + λN) per node with a constant 32-byte overhead against 37N bytes for AVID-FP, sustaining 36 MB/s across sixteen cities at 800 ms. The three approaches place the availability guarantee in a certificate, a signature threshold, and a code respectively, and that placement determines what the protocol above can do.

A further line asks whether every validator must hold every byte. eAVID generalises verifiable information dispersal so fragments are disjoint across nodes and pruning needs no certificate and no coordination, halving steady-state per-node storage when the network is healthy enough for every node to complete dispersal [31]; its authors report integrating it into Autobahn, Bullshark, Walrus and Alpenglow, and signature-free graded broadcast pushes the same way by grading availability rather than treating it as binary [32]. eAVID's evaluation is not yet available, so its storage claim rests at E1_formal.

C. Overlay structure and streaming propagation

Where the DAG family removes propagation from the critical path, a parallel literature makes propagation itself faster. Latency-aware neighbour selection is the cheapest intervention: Perigee has each node retain peers that consistently deliver early, reducing broadcast latency 33% against random selection in simulation over 1,000 sampled Bitcoin nodes across seven regions [33]. Structured overlays achieve this by construction rather than adaptation [34], and coded broadcast attacks redundancy directly: Strokkur encodes transactions into LT codewords whose peeling decoder validates each decoded transaction independently, giving 1.8× overhead at a 95% delivery rate and 0.5 s latency and consuming 2× to 7.6× less bandwidth than Bitcoin's scheme [35]. Tree dissemination with aggregation cuts the leader's fan-out from O(n) to O(log n) [36], and Duet co-designs the tree with the consensus protocol above it, improving peak throughput up to 7.26× over gossip on the same topology at 100 to 300 EC2 nodes across ten regions [37].

Solana sits at a different point, and describing it accurately requires care about evidence. Turbine splits blocks into chunks disseminated through a hierarchical structure over UDP while transactions arrive over QUIC — a description we take from an independent comparative measurement rather than vendor material [38]. The intent is that the leader streams shreds continuously within its slot instead of assembling a block and then propagating it, collapsing the propose-then-propagate structure. That intent is documented only in the Solana whitepaper and engineering documentation, and no peer-reviewed or preprint source we retrieved states a message-delay bound, asynchrony guarantee, or faulty-leader behaviour for it. What *is* independently measured is its cost.

D. Costs

Bandwidth is the price of streaming redundancy, and it is now measured rather than asserted. In a controlled comparison of five deployed L1s on a twenty-node testbed of commodity-class VMs, Solana consumed the highest absolute bandwidth of any chain tested, from roughly 4.7 GiB at 100 TPS down to 2.3 GiB at 500 TPS, with Algorand second at 3.6 GiB to 1.3 GiB and Avalanche and Aptos most efficient [38] (E2_lab). The authors attribute this to flood-like, highly redundant propagation related to Turbine and its erasure coding, which buys something real: data stays recoverable even under loss. Two further findings sharpen the picture. Much of the bandwidth is time-dependent rather than transaction-dependent — heartbeats, empty block proposals and consensus maintenance inflate the footprint at low load — and idle cost varies by nearly four orders of magnitude, with Aptos nodes transmitting 218,856 KiB while idle against 298 KiB for Avalanche and 11 KiB for Redbelly. Validator bandwidth provisioning is therefore set by protocol maintenance traffic rather than by throughput, a cost invisible in any benchmark reporting only tx/s.

Certified dissemination creates an unbounded-memory liveness surface an adversary can drive deliberately. A certified DAG mempool keeps accepting and certifying blocks while the ordering layer above makes no progress, so uncommitted causal history accumulates. Lifefin shows this is exploitable rather than theoretical: Byzantine nodes that disseminate vertices but never vote for leader vertices, combined with a temporary denial of service against one correct node, prevent the remaining correct nodes assembling the 2f+1 committing quorum, and the resulting mempool explosion can compromise liveness in as little as eleven minutes [39] (E2_lab, against the Sailfish and Mysticeti codebases). The mitigation is itself costly: the asynchronous-common-subset fallback carries quadratic storage complexity, 712 kB at n=20 for Sailfish-Lifefin.

The dissemination layer's fairness cost is structural and was flagged by its designers. Narwhal's authors state that slow authorities are indistinguishable from faulty ones and the protocol proceeds without them, creating problems around fairness and incentives, and that the protocol relies on clients to re-submit a transaction not sequenced in time because the leader was faulty [1] (E2_lab). Both are transfers — the first to geographically disadvantaged validators, the second to client software — and neither appears in a throughput measurement.

Every propagation optimisation buys latency with a different currency. Perigee's latency-optimised topologies concentrate connections on fast peers, which its authors identify as an eclipse-attack surface [33] (E2_lab). Strokkur uses 27% more bandwidth than a competing scheme when all nodes are honest, and lowering its coding overhead requires a larger maximum codeword degree, making transactions wait longer in the coding window [35] (E2_lab). Trees are the starkest case: Duet reports

that at 10% random crashes 43% of proposer trees fall below quorum reach and at 33% none reaches quorum by tree alone, so the structure delivering 7.26× throughput also fails discontinuously, and its authors call the tree-phase timer a critical deployment-specific parameter [37] (E2_lab); shared-mempool designs face the analogue, since a larger availability quorum raises push-phase latency [28] (E2_lab). Coded dispersal has a proved floor: every honest server must download at least a $1/(N-2f)$ fraction of dispersed data to prevent a malicious-client attack, and the dispersal-phase Byzantine agreement costs $O(N^2)$ per node, which is why throughput drops as N grows from 16 to 128 at fixed block size [30] (E1_formal for the bound, E2_lab for the degradation).

The fastest chain in the ecosystem is the hardest to measure, a cost borne by the research community rather than by users. The first longitudinal multi-chain crawl of blockchain peer-to-peer networks covered 36 networks over nine months with fifteen crawlers, finding network sizes from under ten to more than 10,000 active nodes [40] (E2_lab). For Solana the authors could not run their own crawler, because Solana has no traditional peer-discovery mechanism, and relied on a third-party node-explorer API; their stated limitations compound this, since an active-only methodology misses nodes behind NAT or firewalls and running all probes from one AWS region introduces geographic bias. Any decentralization claim about the highest-throughput deployed L1 therefore rests on weaker evidence than the equivalent claim about Ethereum or Bitcoin.

V. Layer 2: Ordering and consensus

A. Design axes for the ordering layer

The ordering protocols deployed in high-performance L1s all live in the partially synchronous model of Dwork et al. [41], all route around the impossibility of Fischer et al. [42] by timeouts or randomisation, and all descend from a few named ancestors. What distinguishes them is a set of choices the existing consensus surveys [5, 7, 43, 44] do not systematise together, because they classify by consensus family rather than by position in a deployed stack. We apply seven axes to every mechanism in the accompanying taxonomy: *certification*; *commit rule and expected latency*, in message delays on each paper's own accounting; *behaviour under asynchrony*, separating protocols live through randomisation from those relying on timeouts; *behaviour under crash or Byzantine leaders*; *view change or fallback*, where "none" is itself a design claim; *bandwidth amplification and memory for uncommitted data*; and *evidence grade of the throughput claim* — the axis separating protocols whose latency is known from those whose latency is advertised.

The field has converged on a few commit-latency values and now trades other properties to reach them: three message delays is the claimed Byzantine lower bound, attained by an uncertified DAG at the standard fault threshold, while two is attainable only by weakening the threshold or restricting the transaction class. The existing DAG systematisation [45] organises this space architecturally and explicitly declines to evaluate performance, and a companion analysis frames DAG protocols by how knowledge propagates through the structure [46]; neither pairs a mechanism with its documented cost.

B. The pipelined leader lineage

HotStuff established the template every leader-based high-performance L1 still follows, and its central contribution was making leader replacement cheap. Reducing the safety argument to a three-chain rule and factoring liveness into a separate pacemaker yields a linear view change: after the global stabilisation time a correct leader drives a decision with $O(n)$ authenticators including leader replacement, against the cubic cost of its predecessors [47]. That separation is why the design proved portable into Diem, Aptos and Monad; it is also where implementation defects concentrate. The lineage then optimised chain length: Jolteon reduces three chains to two, cutting commit latency from seven rounds to five and measured latency by 200–300 ms, at the acknowledged cost of a quadratic view change of $O(n)$ messages of $O(n)$ size [48]; the same paper contributes Ditto, replacing view synchronisation with an asynchronous fallback to supply liveness where Jolteon has none, at an expected 13.5 asynchronous rounds against five synchronous. Later work continued shaving phases — one-phase speculation [49], linear-complexity two-chain variants [50], synchronous designs exploiting a known bound [51] — while deployed instances stay close to the Jolteon point.

MonadBFT is the current production representative, and its goal is leader-fault isolation. It speculatively finalises in 1.5 round trips and fully finalises in 2.5, with minimum commit latency 3δ, and proves a single Byzantine leader incurs only one timeout delay while views led by correct leaders operate at network speed [52] — closing the tail-latency hole the Jolteon-style pacemaker leaves open. Three caveats belong in the record: speculative commits can be revoked if the leader equivocates, wasting downstream work; full finalisation requires two consecutive honest leaders; and the paper contains no experiments. Its latency claims are stated as reductions against HotStuff's 3.5 round trips and faster variants' 2.5, and the throughput case for Monad as a deployed chain rests on vendor material, which we grade E4_vendor and do not treat as a performance fact.

C. Certified DAGs

Certified DAG protocols interpret a shared data structure instead of exchanging consensus messages, and Bullshark made this practical enough to deploy. It commits in two round trips during synchrony — a 75% improvement on DAG-Rider — while preserving asynchronous liveness at six rounds in expectation, fairness, and safety against a quantum adversary [53]. The architecturally decisive property is negative: Bullshark requires no view change and no view synchronisation, using information encoded in the DAG edges to maintain safety instead. It measures

125,000 tx/s at two seconds for fifty parties on the five-region AWS testbed used throughout this literature, at over 2× HotStuff's throughput; its authors report the partially synchronous version productionised by Mysten Labs and being integrated by Aptos, with the deployment experience separately documented [54].

The refinements that followed all target the same weakness — a wave-based commit rule serialises on the anchor. Shoal adds leader reputation computed from the causal history of already-ordered anchors, costing no extra messages because the input is the agreed DAG; it reports up to 40% lower latency failure-free and up to 65% under failures against baseline Bullshark, on 32-vCPU VMs across three Google Cloud regions with 118–251 ms inter-region round trips [55]. Shoal++ runs multiple anchors per round with a fast commit rule, reaching an average 4.5 message delays and sub-second latency at 100,000 tx/s, up to 60% below Bullshark in the common case and up to 10× below Mysticeti under failure, on 100 replicas across ten regions on four continents [56] — the sharpest demonstration that leader reputation, not commit-rule depth, dominates behaviour under faults. Parallel work has explored optimistic paths [57], reduced communication overhead [58], asynchronous latency reduction [59, 60], trusted-hardware acceleration [61], and independent delay analysis of the Narwhal–Bullshark composition [62].

BBCA-Chain moves against the family, reintroducing an explicit view change into a DAG. Its primitive, Byzantine consistent broadcast with complete-adopt, commits the leader's backbone block in one instance of three network trips all-to-all; on timeout a node probes it and obtains either an adopt outcome or a provable no-adopt, with 2f+1 no-adopt signatures letting the next view proceed [63]. This buys latency by surrendering the property that made wave-based DAGs attractive, and the primitive's linear form costs five network trips against three.

D. Uncertified DAGs and the latency floor

Dropping certification took DAG consensus to the claimed latency lower bound, and Mysticeti is both the protocol that did it and the only one here with a production measurement behind its headline number. Validators exchange signed blocks directly with no certification round, and a commit rule under which every block can be committed without delay gives Mysticeti-C a three-message-round commit — stated by its authors as the Byzantine lower bound — while sustaining over 200,000 TPS at 0.5 s WAN latency [2]. The deployment evidence makes this E3_mainnet rather than a testbed result: Sui's P50 latency fell from 1.9 s under Bullshark to 400 ms across 106 independently operated validators. The source of the saving is the general case for the certified-versus-uncertified choice: certified DAGs pay multi-round-trip certification, a per-wave commit delay, and a CPU cost for verifying supermajority signatures that grows with validator count and is particularly heavy for a crash-recovered validator, while uncertified designs need one signature generation and verification per block.

Mysticeti's cost is that liveness rests on timeouts rather than randomness, and the asynchronous branch of the family exists to remove that dependency. Mahi-Mahi keeps the uncertified structure but adds a global perfect coin from adaptively secure threshold signatures with asynchronous distributed key generation, restoring liveness under asynchrony and — because leaders are elected after the fact — preventing an adversary from delaying the leader's block; it reports 350,000 tx/s below two seconds and 100,000 tx/s below one second at fifty nodes across five AWS regions, at five message rounds or four in the shorter wave [64]. The trade is quantified: one to two additional message delays for asynchronous liveness, and a dual-mode design confirms that no single parameterisation obtains both robustness and minimal latency [65] — a source we cite only for framing, being an MSc dissertation rather than a peer-reviewed paper.

Cordial Miners shows how much DAG machinery is dispensable, dropping reliable broadcast entirely and relying on an equivocating miner being eventually detected and excommunicated. Its accounting gives three rounds good-case and 4.5 expected under eventual synchrony, five and 7.5 under asynchrony, with good-case O(n) and worst-case $O(n^2)$ message complexity, plus a two-tips-per-miner limit to stop a Byzantine miner flooding the structure [66]; the paper is theory only, so its numbers are E1_formal. Further variants reduce latency by weakening broadcast [67] or add a post-consensus ordering layer preserving the throughput path [68].

E. Buying latency with resilience

The most active current frontier is not the commit rule but the fault threshold, and its boundary is now formally established: tight bounds show fast Byzantine state-machine replication is achievable at an optimal replication factor of $n \geq 5f+1$, and a suboptimal factor buys recovery efficiency only by surrendering optimality [69]. This frames a whole 2025–2026 cohort. Minimmit achieves two-round finality at $n \geq 5f+1$ with nullification in place of a view change, measuring 146 ms view latency against 194 ms for Simplex and 190 ms for Kudzu on a simulated fifty-processor network across ten virtual regions [70]. Odontoceti applies the threshold to an uncertified DAG for the first two-round DAG commitment, at 300 ms median latency and 10,000 tx/s across thirteen AWS regions [71], and BlueBottle builds on Mysticeti at $n = 5f+1$ with sub-0.5 s commits above 200,000 tx/s [72]. Hermes uses prefix consensus so a slow or crashed leader does not void a view, giving 2δ with a timely honest leader and $2\Delta+\delta$ otherwise [73], and FinWhale reaches a two-message-delay fast path at $n = 3f+2p-1$, matching the fast-Byzantine lower bound [74].

Each of these states its resilience reduction openly, and the reductions are substantial. Odontoceti's authors are explicit that the protocol operates with 20% fault tolerance instead of the established 33%, and that requiring references to 4f+1 of 5f+1 parents rather than a two-thirds

quorum costs 40 ms at ten nodes and 65 ms at fifty against an unsafe variant [71]; BlueBottle notes each round waits for an 80% parent quorum against Mysticeti's 67% [72]. The pattern is consistent enough to state as a finding: in this literature, sub-three-message-delay commit is purchased with fault tolerance, at roughly thirteen percentage points of Byzantine threshold per message delay.

F. Leader designs that recover DAG throughput

A distinct family argues the DAG was never necessary — that throughput came from decoupling dissemination while the latency cost came from the DAG, so the two should be separated. Autobahn states this most clearly, combining a highly parallel asynchronous dissemination layer with low-latency partially synchronous consensus to reduce coordination to four message delays on the fast path and six otherwise. Its central claim concerns recovery rather than steady state: traditional view-based protocols suffer *hangovers* persisting beyond the return of a good interval, while DAG protocols recover gracefully but show poor good-interval latency [75]. Autobahn matches Bullshark's throughput at roughly 234K tx/s while halving latency from 592 ms to 280 ms, edging out batched and vanilla HotStuff. The evaluation caveat is material: the headline figures come from n=4 nodes intra-US on GCP, with scaling tests reaching n=20, so the comparison is not at the validator counts at which Bullshark and Mysticeti were measured.

Morpheus and Angelfish turn the same intuition into an explicit dial. Morpheus adapts to the throughput regime, finalising in 3δ at low throughput while matching Autobahn and Sailfish at high throughput, on the observation that DAG protocols induce undue overhead when needed throughput is low; it caps transaction blocks at two predecessors, avoiding the O(n) pointers per block that DAGs carry [76]. Angelfish parameterises the vertex-proposal rate so one protocol interpolates between leader-based and DAG operation, achieving at least 40% lower latency than Sailfish [77] at 30,000–100,000 tps on fifty geo-distributed nodes, with good-case bits per round of $O(\kappa x n^2 + n^3)$ against $O(\kappa n^3)$ for the Bullshark/Shoal/Sailfish group and $O(\kappa n^4)$ for Cordial Miners and Mysticeti [78]. Banyan adds a 2δ fast path tolerating up to p unresponsive replicas while preserving the absence of a view change [79], and Barnacle makes leader count itself adaptive, driving it from the committed DAG with no extra messages and no cryptography [80].

G. Consensusless fast paths

The most radical intervention removes consensus from the path of most transactions, resting on a narrow but real observation: transferring a single-owner asset requires agreement that the owner authorised it, not agreement on order. FastPay demonstrated this with Byzantine consistent broadcast alone, confirming transfers in a single round trip to a quorum and reporting over 80,000 transactions per second with twenty authorities at intra-continental confirmation latency under 100 ms [81]. Its fault behaviour is the strongest in this section and follows from having no leader: measured latency showed no statistically significant difference when up to f of ten authorities were unavailable. Zef extends the lineage to private payments, spending anonymous coins in two round trips and reaching 20,000 tx/s at a committee of ten [82].

Sui Lutris is the production realisation, and it matters here because it is a hybrid rather than a replacement: owner-only transactions take the consensusless two-round-trip path, shared-object transactions go through consensus, and both share a state machine with a reconfiguration protocol preserving safety across epoch boundaries [83]. It reports latency below 0.5 s for throughput up to 5,000 certificates per second against state-of-the-art real-world consensus latencies of three seconds, and withstands up to one third crashed validators without meaningful degradation. Mysticeti's fast-path variant folds the same idea into the DAG, weaving fast-path transactions into the structure to minimise signatures and messages [2].

H. Solana's slot-streaming design and Alpenglow

Solana is the highest-throughput deployed L1 in the subject set and the one whose ordering-layer evidence is weakest, and the asymmetry is itself a finding. Its design — proof of history as a verifiable clock, Tower BFT as the voting layer, a leader schedule fixed per epoch rather than advanced by view change, and continuous shred streaming within a slot — is documented in the Solana whitepaper and engineering documentation, which we grade E4_vendor. None of the sources we retrieved establishes a commit latency in message delays, an asynchrony guarantee, or a faulty-leader behaviour for the deployed protocol, so we record those axis values as not established rather than filling them from vendor descriptions. Two independently measured facts are available: Solana requires thirty additional blocks appended after a transaction's block, and both Solana and Avalanche were observed to stop globally when some network messages were delayed [38] (E2_lab) — a liveness observation no vendor description anticipates. Work on Snow-family liveness addresses the analogous problem for Avalanche [84], whose leaderless metastability design [85] trades deterministic finality for the absence of a leader, as Algorand trades it for committee sampling [86].

Alpenglow is the case in this layer where an E4 claim can be checked against analysis, and it does not survive the check intact. It separates dissemination (Rotor) from voting (Votor) at $n = 5f+1$, situating it in the fast-BFT cohort above, and several independent papers cite it as motivating their own work [69–71, 73]. Its advertised guarantee is one-round commitment while tolerating both fewer than 20% Byzantine and fewer than 20% crash faults. An independent analysis states that this is impossible on the cited bound and that careful analysis makes Alpenglow a 3δ protocol, since Rotor requires 2δ and Votor one more, and observes that this cohort's codebases — Minimmit, Kudzu and Alpenglow — provide no deployable implementations with

networking code and are intended for simulation only [72]. We report the advertised figure only as a design claim. The Alpenglow whitepaper carries no DOI and we could not verify it through any registry.

I. Cross-family comparison

J. Costs

Mean and tail latency diverge under DAG batching, and the cause is leader selection rather than the commit rule. In Shoal++'s failure experiment, with 33 of 100 replicas crashed, Bullshark and Mysticeti suffer drastically increased latency for want of a leader-reputation mechanism while Shoal and Shoal++ swiftly elect only live anchors — a gap of up to 10× against Mysticeti [56] (E2_lab). This is striking because Mysticeti has the *better* good-case commit rule, three message delays against 4.5. Load is the second source of divergence and it is geographic: the same work reports latency rising up to roughly 2× at high load because quorums must span more regions. Multi-leader configurations sharpen the tail further, and Barnacle's authors name the mechanism — production DAG deployments run a single leader because of head-of-line blocking, a slow leader stalling the pipeline for at least one leader timeout and for several waves when its slot falls back to the indirect decision rule, a risk growing with leader count [80] (E1_formal, E2_lab). Autobahn frames the layer's governing trade directly: view-based protocols optimise good-interval latency but suffer hangovers outlasting the disruption, while DAG protocols recover gracefully and pay in good-interval latency, and its own residual costs are correspondingly honest — some hangovers are unavoidable, the fast path pays higher tail latency from the larger n-sized commit certificate, and proposing only certified tips adds about 33 ms to form a proof of availability [75] (E2_lab). Angelfish's authors add that votes carry no transactions and so contribute nothing to throughput [78] (E1_formal, E2_lab): in this layer every message that is not a transaction is overhead.

The leader-based lineage's tail cost is larger and better documented, and this is the empirical case for the DAG. Under one to three crashes in a committee of ten, batched HotStuff throughput fell thirtyfold from 70,000 to 2,500 tx/s with a tenfold latency increase while Narwhal-HotStuff and Tusk held throughput [1] (E2_lab); Bullshark's independent replication found HotStuff dropping over tenfold in throughput with a fifteenfold latency increase at three faults [53] (E2_lab). The mechanism is the timeout: DiemBFT and Jolteon need two timeouts to advance past a crashed leader, ten seconds at a five-second setting, degrading to 30,000 tps at roughly ten seconds under one fault and 10,000 tps at fifteen seconds under two [48] (E2_lab). The surface is exploitable without violating safety: a performance attack on chained HotStuff degrades throughput while remaining invisible to safety monitoring [87] (E1_formal). Pipelining and overlapping are partial mitigations [88], as is attention to BFT stability over unreliable wide-area links [89].

Memory for uncommitted causal history is the DAG's characteristic resource cost, and its worst case is unbounded by design. Bullshark's authors state that all previous theoretical solutions require unbounded memory to hold the entire DAG, and that their bounded-memory implementation garbage-collects old rounds but then guarantees timely fairness only during synchrony [53] (E1_formal). They go further, and it reads as a limit result rather than an implementation note: meaningful fairness for all honest parties appears impossible with bounded memory in asynchronous networks, because any message can be delayed until after the relevant DAG prefix has been collected. The bandwidth side of the same cost is that DAG blocks carry O(n) parent pointers — the overhead Morpheus avoids by capping predecessors at two [76] (E1_formal) and that Angelfish quantifies as $O(\kappa n^4)$ bits per round in the good case for Cordial Miners and Mysticeti against $O(\kappa n^3)$ for the Bullshark group [78] (E1_formal). Dropping certification relocates rather than removes this cost: an edge to an unavailable proposal forces a fetch on the critical path, at least two additional message delays per round, which is why Shoal++ retains certification [56] (E2_lab).

Geographic decentralization imposes a latency floor no commit rule can cross, and deployed protocols are now close to it. The evidence is consistent across independent testbeds: Mahi-Mahi's five-region, Shoal++'s ten-region hundred-replica, and Odontoceti's and Sui Lutris's thirteen-region deployments all report inter-region round trips in the 100–250 ms range, and every commit rule must traverse a quorum including distant regions. Work addressing the floor directly confirms the achievable bound is set by inter-region propagation delay for a given placement [90, 91] (E2_lab), with geospatially aware stake assignment proposed as the remaining lever [92] (E1_formal, E2_lab). The consequence is direct: a sub-500 ms commit claim implies either a concentrated validator set or a quorum excluding the most distant participants. Cryptography on the commit path has a comparable floor — an optimistic ramp-threshold construction cut randomness latency overhead on Aptos mainnet by 71%, from 85.5 ms to 24.7 ms, but the accompanying impossibility result proves some extra delay is inherent for a wide family of consensus protocols [93] (E1_formal, E3_mainnet).

View-change complexity is a correctness risk rather than merely a performance cost, and the DAG family converts it into commit-rule complexity instead of removing it. That view change is where defects concentrate comes from Twins, which generates targeted Byzantine scenarios by duplicating a node's identity and reproduced both safety and liveness bugs in deployed BFT implementations including DiemBFT [94] (E2_lab). Wave-based DAG protocols advertise the absence of a view change as a correctness benefit, and the claim is partly justified — but complexity does not leave the system: LiDO-DAG exists because the safety and liveness of DAG commit rules are intricate enough to warrant machine-checked verification [95] (E1_formal). Leader-based designs

TABLE I
ORDERING-LAYER PROTOCOLS ON THE SEVEN SHARED DESIGN AXES. ENTRIES MARKED AS NOT ESTABLISHED COULD NOT BE SOURCED TO A PEER-REVIEWED OR PREPRINT DESCRIPTION.

Protocol	Certification	Good-case commit	Asynchrony	Faulty leader	View change	Throughput evidence
HotStuff	certified	3-chain, 3 extra batches	timeouts after GST	timeout rotation, $O(n^2)$ cascading	pacemaker	E2_lab
Jolteon / Ditto	certified	5 rounds / 5 sync, 13.5 async	none / async fallback	2 timeouts (10 s)	quadratic TC	E2_lab
MonadBFT	certified	3δ; 1.5 RTT speculative	timeouts	1 timeout, isolated	pacemaker + TC	E4_vendor
Narwhal–HotStuff	certified	4 RTT	mempool unaffected	O(n) worst case	inherited	E2_lab
Tusk	certified	4.5 RTT; 7 async	live, random coin	4.5 RTT	none	E2_lab
Bullshark	certified	2 RTT; 6 rounds async	live, fairness kept	throughput held	none	E2_lab
Shoal++	certified	4.5 msg delays	after GST only	live anchors elected	reputation	E2_lab
Mysticeti	uncertified	3 msg rounds	timeouts only	crash-free skip	none (HammerHead)	E3_mainnet
Mahi-Mahi	uncertified	5 (or 4) msg delays	live, perfect coin	direct skip rule	none	E2_lab
Cordial Miners	uncertified	3 / 4.5 exp.; 5 / 7.5 async	live w.p. 1	excommunication	none	E1_formal
BBCA-Chain	certified	3 trips (5 linear)	after GST only	probe ADOPT/NOADOPT	explicit	E1_formal
Autobahn	uncertified	4 md fast, 6 otherwise	lanes progress	reliable inclusion	rotating, 1 s timers	E2_lab
Angelfish	certified	$3\delta + 2\delta$	after GST only	TC + leader edges	rotating + TC	E2_lab
Hermes	n/a	2δ; $2\Delta+\delta$ fallback	fallback pre-GST	view not voided	none	E1_formal
Odontoceti	uncertified	2 rounds	not stated	p90 ≈ fault-free	early block prod.	E2_lab
Sui Lutris	certified	2 RTT fast path	inherited	$\leq 1/3$ crash tolerated	epoch reconfig	E3_mainnet
FastPay	certified	1 RTT	fully async	latency unchanged	none	E2_lab
Solana (Tower BFT)	none	not established	not established	not established	epoch schedule	E4_vendor
Alpenglow	certified	advertised 1 round; analysed 3δ	not established	not established	not established	E4_vendor

concentrate risk in a separable, well-attacked module; DAG designs distribute it into a commit rule that is harder to test and until recently lacked machine-checked proofs. Shoal's authors add that setting a proper timeout is non-trivial and highly network-dependent, and timeout handling adds significant implementation complexity [55] (E2_lab); Byzantine-behaviour detection must likewise be bolted on, because the base protocols do not identify equivocating or withholding validators [96] (E2_lab).

DAG structures create a larger ordering-fairness surface than the leader-based one they replaced, and this is now established systematically. Multi-proposer designs were motivated by the idea that concurrent proposal removes the single-proposer censorship point. A systematic attack-space study organised around the adversary, the protocol, the target and the deployment finds the opposite in practice: every one of six production DAG-based BFT protocols evaluated is vulnerable to at least a subset of the MEV attacks in the space, and which attacks succeed is dictated mostly by the protocol's own design rather than by attacker effort [97] (E1_formal, E2_lab). The most concrete instance measures Sui's deployed Mysticeti ordering and traces bias to an implementation detail — the gas-price re-sort uses a stable sort, so equal-fee transactions retain a prior ordering set by validator index. The lower-indexed side wins same-round ordering about 89% of the time with no attacker present, strategic silence raises that above 94%, and roughly $18,000 a day of MEV was observed being extracted from users on Sui mainnet; a per-epoch seeded shuffle drops same-round advantage to 44.8%, but strategic silence persists at 92.3% because its advantage comes from round position and is bounded only by causality [98] (E2_lab, E3_mainnet). Simulation agrees from the other direction: a coordinated Byzantine minority can manipulate DAG structure to reorder transactions, and reducing that vulnerability by raising client fan-out multiplies transaction duplication [99] (E2_lab); surveys of MEV on DAG-based ledgers have begun consolidating

these findings [100].

Order fairness is achievable on a DAG but its measured price ranges from modest to an order of magnitude, depending on the fairness notion. FairDAG quantifies this sharply: its absolute-batch variant reaches 83.5–84.9% of Tusk's throughput while improving fairness quality over the leader-based Pompe and Themis, whereas its stronger relative-ordering variant reaches only 11.9–12.6% [101] (E2_lab). Leader-based fairness protocols are worse still because broadcasting collected local orderings grows quadratically with participant count, and the optimal-resilience order-fair broadcast construction requires $O(n^2)$ messages per delivered payload and $O(n^2L + n^3\lambda)$ bits [102] (E1_formal). Later designs trade resilience for throughput instead: Herring sustains roughly 90% higher saturation throughput than FairDAG-RL with a fixed memory footprint but requires $n \geq 4f+1$ rather than $3f+1$, precisely because it operates on a DAG without weak edges [103] (E1_formal, E2_lab), and Tilikum reports up to 39× higher throughput than fair-ordering baselines while blocking known DAG reordering attacks, at the cost of committing $2f+1$ timestamp pairs plus hole-filler lists per transaction [104] (E1_formal, E2_lab). Commit-reveal blind ordering sidesteps the trade differently, integrating into a DAG at zero message overhead, but its threshold cryptography costs roughly 300 μs to disperse and 3,500 μs to retrieve for a six-of-sixteen scheme and the paper reports microbenchmarks only [105] (E2_lab).

Consensusless fast paths shift correctness obligations onto client software, a cost no throughput figure captures. Sui Lutris's authors state that consensusless protocols are sensitive to client bugs because an equivocation locks assets forever — a hazard their design bounds to a single epoch but does not remove — and that the object-centric model requires users to track multiple owned objects rather than one account, with the mixed-object path costing an additional round trip for certification [83] (E3_mainnet). FastPay's authors are similarly candid about measurement conditions, which matters for how numbers propagate: their experiments represent best-case laboratory performance, transactions from the same sender account cannot be parallelised, and the harness loads transactions via the local network interface without exercising the full wide-area stack [81] (E2_lab). Committee size rather than machine count binds this family — Zef's throughput falls from 20,000 tx/s at ten nodes to 6,000 tx/s at fifty because $2f+1$-signed transfer certificates must be verified without threshold signatures [82] (E2_lab).

The economic cost of cheap ordering capacity is that the capacity is consumed by traffic that does not pay for it. Measured on Ethereum rollups rather than the monolithic L1s that are this survey's subject, the mechanism is nonetheless an ordering-layer phenomenon. Spam MEV transactions consumed over 50% of block gas on Base and Optimism in the first quarter of 2025 while only 6–12% of these speculative probes resulted in an actual trade, and spam paid less than a quarter of total fees; spam gas grew 122× between the Dencun upgrade and its April 2025 peak while non-spam gas grew 11.2× [106] (E1_formal, E3_mainnet). The analytic result is what generalises: spam takes an increasing share of each additional unit of block capacity, so adding ordering throughput does not straightforwardly deliver more useful throughput. We flag the transfer of this finding to Solana, Aptos, Sui and Monad as unestablished, having located no equivalent mainnet measurement.

K. Open problems and flagged gaps

Four things could not be established from the retrieved literature. First, **Solana's ordering layer has no peer-reviewed or preprint characterisation**: we found no source establishing a commit latency in message delays, an asynchrony guarantee, or a faulty-leader behaviour for proof of history and Tower BFT as deployed, so those taxonomy values are recorded as not established, and the only independently measured facts we located concern bandwidth, finality depth, and a global stall under message delay [38]. Second, **Alpenglow's advertised latency is contradicted by the only independent analysis we found**, which recomputes it as 3δ and notes the absence of a deployable implementation with networking code [72]; its whitepaper carries no DOI. Third, **Byzantine-fault evaluation of DAG protocols remains an open research question by the field's own admission** [64], so the fault-behaviour axis rests on crash-fault experiments plus formal Byzantine safety proofs, not measured Byzantine performance. Fourth, **the testbeds behind the headline numbers are not commensurable**: validator counts range from four (Autobahn's headline run) to 106 (Mysticeti on Sui mainnet), machine classes from t3.large to m5d.8xlarge, and geographic spreads from intra-US to ten regions on four continents, so the cross-protocol throughput comparisons circulating in this literature are not like-for-like. Benchmark frameworks that would fix this exist [24, 107, 108] but are not used by the protocol papers. We could not retrieve Sailfish's full text — paywalled at IEEE S&P with no preprint located — so its taxonomy row is deliberately sparse, and we excluded one otherwise-relevant corpus record as non-English per the protocol's exclusion criteria.

VI. Layer 3: Execution and intra-block parallelism

Every deployed high-throughput L1 makes the same bet at the execution layer — that the transactions inside a block are mostly independent, so a multi-core validator can run them concurrently and still produce the state a sequential validator would have produced. What separates Solana, Aptos, Sui and Monad is not whether they take that bet but *where they get the information to make it*, and it is that choice, rather than the raw speed of the virtual machine, that determines how each system behaves when the bet goes wrong. This section classifies the deployed families on five shared axes, then shows that the dominant

cost of every one of them is the same phenomenon under different names. As the fraction of transactions touching hot state rises, all four designs converge on serial execution, and they differ mainly in who pays for the convergence and how visibly.

A. Design axes

We place each mechanism on five axes. **Where conflict information comes from** covers developer declaration, static analysis, runtime detection, type-system enforcement, and historical prediction. **When conflicts are resolved** distinguishes before, during, and after execution. **What happens on conflict** distinguishes abort-and-retry, serialization, rejection, and block-and-wait. **The determinism guarantee** distinguishes strict order-determinism — the parallel result must equal sequential execution in the block's preset order — from mere state-determinism, where all replicas agree but no single total order is reproduced. **Behaviour as contention rises** records the measured shape of degradation. The first axis is the generative one: the other four largely follow from it, which is why the taxonomy is not four parallel stories but one design space with four occupied corners.

The concurrency literature has been systematised before, most recently by Chahoki et al. [20], whose systematic review organises concurrency approaches for smart contracts into a taxonomy running from concurrent consensus through intra-block concurrent processing to DAG designs. Our axes cut across that classification rather than replacing it: we ask not what technique a paper proposes but what a deployed system gives up to run it. That review also issues a caution worth carrying: several works reason about smart-contract execution as shared-memory concurrency when unmodified Ethereum executes a block sequentially, so a result about "concurrent smart contracts" may describe a system nobody runs.

B. Ahead-of-time declared access sets

Solana resolves the conflict-information problem by moving it off the validator and onto the transaction author: every Solana transaction carries the list of accounts it will read and write, and the Sealevel runtime schedules non-overlapping transactions concurrently without ever needing to detect a conflict at runtime. This is the only one of the four families whose core mechanism has no archival technical description — the design is documented in vendor engineering posts and client documentation, which under our grading scheme makes the mechanism an **E4** design claim rather than a measured performance fact. What *is* measured, and measured well, is the consequence.

The most consequential finding in the recent execution literature is that declaration buys determinism at the price of conservatism, and the price is large enough to invert the naive ranking of the two models. Comparing matched samples of roughly a thousand recent mainnet blocks from each chain, Anjana et al. [109] report that Ethereum blocks contain about 63% independent transactions against roughly 8% on Solana, a longest conflict chain of 8.47% of block size against 30.97%, and an effective parallel width of 11.17 against 0.36. The reading is not that Solana's engine is worse but that a declared access set is a conservative superset of the set actually touched: the scheduler must respect every declared write, including those a run would never have performed. The same study finds that write-write conflicts account for roughly three-quarters of remaining Solana conflicts, a category that read parallelism cannot relieve and that only multi-versioning would.

Declaration also has a failure mode that speculation does not: it can simply be wrong. Anjana et al. [109] observe the success rate of non-vote Solana transactions falling from 85.32% in their oldest sampled period to 69.03% and then 54.50% in the most recent, attributing part of the decline to inaccurate access specifications and to the drift between the state assumed at declaration time and the state at execution time. That attribution needs qualifying, and a year-long measurement supplies the qualification: across 2.9 billion non-vote transactions sampled from August 2023 to July 2024, Zheng et al. [110] find 1.51 billion failures — a majority of all traffic — but the failure rate is 58.43% for accounts their classifier labels as bots against 6.22% for human-initiated accounts, and the top ten programs, led by an AMM and two DEX aggregators, account for 77.95% of all failures. The headline failure rate is therefore dominated by bot-submitted trading traffic rather than by mis-declared access sets; the study reports the failure concentration by account class and by program, and does not characterise the economic motive behind those submissions. The two findings are complementary rather than contradictory: declaration staleness is a real cost of the model, but it is a minority contributor to the aggregate number, and a survey that cites the aggregate as evidence of declaration burden would be overreaching. That drift is precisely what Heimbach et al. [111] quantify on the EVM side by re-executing transactions across historical pre-states: gas estimates change for 46.0% of Base and 13.9% of Ethereum transactions, and for MEV traffic on Base the write-set overlap between the landed state and a state five blocks earlier is only 30.8%. A declaration made when a transaction is signed is a statement about a state that will not be the state it executes against. The economics are unforgiving: Wadhwa et al. [112] prove that deciding whether a declared object is actually used is P-complete, as hard as executing the transaction, and that consequently no fee mechanism can eliminate both user risk (paying for declared-but-unused objects) and scheduler risk. Under their parameterisation with real Ethereum priority-fee data, the risk-division parameter needed for shill-proofness rises from 0.15 at the median priority to 0.625 at the mean and exceeds 1 at the 95th percentile, where no division in their framework suffices. Pricing parallel capacity rather than scalar gas is an active problem in its own right [113].

The declaration obligation is also a security surface: the rule that undeclared state is inaccessible is a genuine safety property, but it makes ownership and signer checks an

TABLE II
THE FOUR DEPLOYED FAMILIES OF INTRA-BLOCK PARALLELISM PLACED ON COMMON AXES.

	Declared access sets	Optimistic STM	Ownership typing	Deterministic pre-ordering
Deployed in	Solana	Aptos, Monad, parallel EVMs	Sui	research prototypes
Conflict information	developer declaration	runtime detection	type system + declaration	speculative pre-execution
Resolved	before execution	during execution	before execution	before execution
On conflict	serialize / reject if stale	abort and re-execute	reject, lock until epoch end	defer to later batch
Determinism	strict order-determinism	strict order-determinism	state-determinism only	strict order-determinism
Cost concentrates in	declaration accuracy, conservatism	re-execution waste	fast-path share, liveness	pre-execution phase

application responsibility, and binary-only fuzzing found 92 bugs across 52 of 6,049 mainnet programs using oracles for exactly those patterns, in an ecosystem where fewer than 2% of programs publish source [114].

C. Optimistic speculation with deterministic re-execution

Block-STM inverted the problem: rather than asking the author what a transaction will touch, run it and find out. Gelashvili et al. [115] combine optimistic concurrency control with a multi-version memory and a *preset* serialization order given by position in the block, so that a read returns the value written by the highest-indexed lower transaction, and a validation failure aborts the incarnation, marks its writes as estimates, and schedules re-validation of everything above it. The preset order, which the paper frames as a curse turned into a blessing, does double duty: it makes the result provably equal to sequential execution, and it turns an aborted incarnation's write-set into a free predictor of future conflicts. On an AWS `c5a.16xlarge` with 32 physical cores and hyper-threading disabled, the engine reaches up to 110k tps on Diem benchmarks and 170k tps on Aptos benchmarks, with contention controlled by shrinking the account set [3].

The degradation curve is the interesting part, and the authors publish it honestly: 20× over sequential at low contention, 9× at high contention, and at most 30% overhead when the workload is entirely sequential. That 30% is the pure cost of the machinery (multi-version lookups, validation tasks, atomic counters) paid on a block that had no parallelism to extract. The deeper structural cost is that under a preset order validations are not independent: a single failed validation forces re-validation of every higher-indexed transaction, so one hot key produces a cascade rather than a local stall. Systems built on top of Block-STM attack exactly this. Ezard et al. [116] delay resolving a dependency until the blocking transaction has passed validation rather than merely finished executing, and add a priority scheduler ordered by number of dependents, reporting up to 42% over Block-STM and 61% over a pessimistic baseline at 16 workers — but only with 100% prior knowledge of the conflict structure, and they find that priority scheduling "is only viable when a very high percentage of prior knowledge is available", with re-executions rising with worker count for every protocol tested. The same pattern recurs in the access-specification work of Anjana et al. [117], which augments Block-STM and PEVM with conflict information from static data-flow analysis or historical tracing and degrades gracefully to plain Block-STM when no specification is available.

Monad is the largest deployment of the optimistic family outside Aptos, and it is where the evidence base is thinnest. Its consensus layer has an archival write-up [52], but the optimistic parallel executor and the MonadDb state engine are described only in vendor documentation and ecosystem write-ups. We therefore record Monad's execution design as an **E4** design claim: optimistic execution of an ordered EVM block with conflict detection and re-execution, with signature recovery cached across re-executions, and with execution deferred behind consensus. We found no independent measurement of it. Academic parallel-EVM designs in the same family, such as operation-level concurrency in Lin et al. [118], and the deterministic-execution engine of Liu et al. [119], are measured, but they are not Monad.

D. Ownership typing and the consensusless fast path

Sui takes the third position: make the type system, not the author and not the runtime, carry the conflict information. State is a set of versioned objects tagged immutable, owned, or shared; a transaction naming only owned and read-only objects is authorised by a single signer, so validators can lock the object-version key, sign, and return, and a client assembling 2f+1 signatures has finality in two round trips with no validator-to-validator communication at all [120]. Transactions touching shared objects go to consensus. The architecture is a direct descendant of consensusless payment settlement [81], generalised from coins to arbitrary Move objects.

The reported numbers are strong (sub-0.5 s latency at up to 5,000 certificates per second with ten validators, about 4,000 with a hundred, across thirteen AWS regions) and the mainnet record is unusually well documented: 143 million objects, over 3.1 million certificates per day, no downtime over thirteen months as of mid-2024. But the same paper supplies the cost, and it is a distribution rather than a constant: the fast-path share of mainnet traffic is about

64% typically, peaks at 98%, and *drops to 1–2% on some days*, when the consensusless path is essentially inactive and the system is an ordinary DAG-BFT chain. Whether a workload gets the fast path is decided by how contract authors typed their state rather than by how much true contention exists, and shared-object use in deployed Sui contracts concentrates in exactly the application classes that dominate volume [121].

Ownership typing also relocates the liveness cost. Because safety comes from a lock on `(ObjectID, Version)` rather than from ordering, a client that equivocates — signs two transactions on the same object version — loses liveness on that object until the epoch ends, 24 hours in production, at which point reconfiguration unlocks it. This is a conscious trade: no abort cascade and no re-execution waste, but a user-triggered stall measured in hours rather than milliseconds. The object model further pushes bookkeeping onto users and wallets, since a balance is a set of coin objects rather than a number, and the paper explicitly advises developers to shape their types so that operations stay within a single owner.

E. Deterministic pre-ordering and conflict-free partitioning

A fourth family borrows from deterministic databases: compute the conflict information before the block is finalised, then execute a schedule that has no conflicts left in it. Prophet has untrusted, self-organising "reconnaissance coalitions" speculatively pre-execute pending transactions to obtain read/write sets, after which a stateless sequencing shard packs only mutually disjoint transactions into the global order; on a replay of roughly a million Ethereum mainnet transactions across 16 machines with links throttled to 20 Mbps and 100 ms, it holds the abort ratio near zero where an OCC baseline aborts about half of transactions with more than two inter-contract calls, and improves throughput 3.11× at 64 shards [122]. The mechanism's own cost is an entire extra phase: naive sequential pre-execution spends 87.5% of its time on communication, cross-shard messages grow to 322 bytes at 64 shards, and the incentive design for coalition participation is left open.

Prophet also reports the most useful number for understanding why conflict-free partitioning is hard at contract granularity: pre-executing batches of 3,000 transactions leaves nearly 90% of them in conflict, and storage-slot granularity plus dependency-ordering rules recover only 5.34% and 21.42% respectively. Spectrum preserves the consensus-established order exactly rather than merely producing *some* serial order, using speculative deterministic concurrency control with operation-level rollback, and reports 1.4×–4.1× over competing schemes on contended workloads [123]; Remora scales the idea out under a strict object-versioning ownership model to 250k TPS [124]. The wider family of deterministic concurrency-control protocols is now well mapped [125–127], and earlier execute-order-validate work established the pattern of aborting only provably unserializable transactions before ordering [128].

The theoretical boundary of this family is now known and it is discouraging. Hay and Friedman [129] show that deriving the execution schedule from the block's conflict graph rather than from the consensus order is the right move — respecting an arbitrary total order costs 371% on average and up to 7,400% in modelled latency across conflict rates from 0.001% to 25% — but they also prove that computing the optimal schedule is equivalent to minimal graph colouring for homogeneous transactions, hence NP-hard and inapproximable, and that for heterogeneous transaction lengths neither minimal colouring nor minimal weighted colouring is optimal. Every deployed and proposed scheduler is therefore a heuristic, and the gap to optimal is bounded only empirically.

F. Scheduling outside the executor

A distinct and under-recognised family moves the intervention upstream of the execution engine entirely. Anthemius sits between mempool and consensus and constructs blocks that are *easy to parallelise*, using non-exhaustive read/write hints and gas estimates to bound the critical-path gas of a block at `maxgas/c` for `c` cores, and reports up to 240% throughput on DEX workloads with no modification to the execution engine, consensus, or block structure [130]. Its costs are instructive because they are not throughput costs: tail latency grows with batch count and the slowest 25% of transactions pay a penalty; on an NFT workload dominated by repeat users the benefit nearly vanishes, since skipping one transaction from a user forces skipping all their later ones; and the leader gains discretionary power to delay transactions in a way formally indistinguishable from honest handling of a sequential workload, which the authors note makes the scheme incompatible out of the box with encrypted-mempool fair ordering.

FAFO pushes the same idea to an extreme, approximating read/write sets by simulation or EIP-2930 access lists, packing non-conflicting transactions into frames using pairs of 2,048-bit Bloom filters sized to fit half of L1 cache, and reporting over 1.1 million native-transfer TPS and roughly 130-way transaction-level parallelism on a single 96-core node while still Merkleising every block [131]. The declared costs are an 8% loss of parallelism to Bloom false positives and the need to drop and reschedule transactions whose actual access set diverges from the prediction; the undeclared one is that the headline contention robustness is demonstrated at a skew the authors did not actually test, having omitted the most realistic hotspot parameter as "indistinguishable" from a milder one. The scheduling problem has now been formalised as a pair of optimisation problems — scheduling an already-ordered block, and jointly selecting and scheduling from the mempool — with MILP formulations for ground truth and a greedy list-scheduling heuristic carrying a $(2 - 1/p)$ makespan guarantee; on real Ethereum blocks the exact solver times out at 6,000 s for eight cores while the heuristic reaches 74–100% of the bound in milliseconds [132].

G. Scale-out across machines

If a single validator's cores are the limit, the next move is more machines per validator. Pilotfish shards object state across ExecutionWorkers, with per-object pending queues encoding write-after-write and read-after-write dependencies and a versioned-queue algorithm that admits Move's dynamically discovered child objects; it reaches roughly 10× the single-node Sui baseline on compute-heavy workloads while holding latency near 50 ms [133]. The limits are candid. On a simple-transfer workload, scaling is linear only to four workers and is not maintained at eight because the system stops being compute-bound; contention becomes queueing delay, with throughput falling 35k → 30k → 22k tx/s and latency rising to 500 ms as the transactions per shared counter go 10 → 100 → 1000; crash tolerance requires 2fe+1 replicas *per execution shard*, multiplying the machine count per validator; and the entire evaluation is single-datacenter, so the geo-distributed case that motivates the design is untested. Thunderbolt combines a sharded DAG with pre-consensus concurrent pre-play, reporting a 50× speedup over serial DAG execution at 64 replicas and abort reductions of 50% against OCC and 90% against 2PL-no-wait, but shows the cross-machine tax plainly: throughput falls from 100k to 19k tx/s as the cross-shard share rises from 0% to 100%, and reconfiguring every 10 rounds costs 80k against 180k tx/s [134]. Disaggregating execution earns throughput by paying coordination, and the exchange rate worsens with the fraction of work that crosses a machine boundary.

H. Programming-model support

The most effective response to contention in the deployed systems is not a better scheduler but a change to what the contract asks for. Aptos's deferred objects let a developer mark a frequently mutated field — currently integer counters and strings up to 256 bytes — so that updates are captured as symbolic deltas rather than applied, with preconditions predicted and validated at commit; on single-fee-payer, single-receiver and limited-supply NFT-mint workloads, which are precisely the patterns that serialise an optimistic engine, this yields up to 12× [135]. The cost is a sharp boundary. Any `reveal` of a deferred value reintroduces the conflict, costing up to 4.4× throughput at a 10% reveal rate and running up to 3.1× *worse* than sequential at higher rates, and in the worst prediction case the technique yields only 1.8×. The mechanism does not remove contention; it lets a developer who understands it move contention out of the critical path, and punishes one who does not.

The same logic runs through the language-level work. Move's resource types and object ownership are what let Sui infer independence statically, and the guarantee extends to adversarial composition: Patrignani and Blackshear [136] formalise robust safety and build a static encapsulator certifying that a module's invariants hold when linked against arbitrary untrusted code. Specification is tractable at framework scale — the Move Prover verifies roughly 8,800 lines of Move against 6,500 lines of specification in CI, almost always in under 30 seconds [137], though the authors concede their specifications are more verbose than needed and that quantifiers make the underlying problem undecidable. On the EVM side, scope-annotated models partition contract state explicitly across engines with asynchronous relay for cross-scope calls [138, 139], and static ownership-and-commutativity analysis has been used to place state for sharding [140]. All of these shift the conflict question from run time to design time. A controlled comparison of functionally equivalent contract pairs finds the trade this implies: Move's type system cuts explicit runtime security-check density by 60% relative to Solidity at the cost of 47% more code, with identical cyclomatic complexity, on a preliminary sample of twelve pairs and eleven developers [141].

I. Costs

Contention collapse. The central cost of intra-block parallelism is that it evaporates exactly where throughput matters, and the trajectory is visible across a decade of measurement. Saraph and Herlihy [142] replayed sampled Ethereum blocks through a speculative engine and found estimated speedups falling from about 8× in 2016 to about 2× at the end of 2017, with a small set of contracts responsible for most conflicts. Reijsbergen and Dinh [143] measured conflict rates across seven chains, finding Ethereum's single-transaction conflict rate near 80% in 2016–17 declining to about 60%, and showed analytically that a two-phase speculative scheme yields only 1×–2× and can fall *below* 1×, while an idealised scheme with perfect a priori knowledge could reach 6× on eight cores. The gap between those two numbers is the entire value of conflict information, and it is why every subsequent system fights over where that information comes from. Measured against optimal schedules on real ordered blocks, Karmegam et al. [132] obtain only about 1.5× at two cores and 2.3× at eight, concluding that a fixed consensus order plus conflict-induced dependencies caps parallelism regardless of core count, a conclusion compatible with opcode-level parallelisation analysis [144], transaction-locality studies [145] and the formal theory of safe transaction parallelism [146]. The largest cross-chain study concludes that "no single parallel execution strategy is fit for all blocks", with substantial variance not only across historical periods but between adjacent blocks [109].

Re-execution waste and CPU amplification. Optimistic engines pay for contention in wasted work rather than in lost parallelism, and the waste is superlinear in an unhelpful direction: more workers means more speculation in flight and more aborts to cascade [116]. The floor cost is the 30% overhead a fully sequential workload pays to Block-STM's machinery [3]; earlier optimistic engines paid a comparable structural cost in block-graph metadata, 13.79%–34.55% of block size [147]. Rescheduling rather than retrying [148, 149] trades waste for scheduling latency, and hardware transactional memory makes retries cheaper

rather than rarer [150]. A correctness cost also appears once parallel execution is layered on an account model: intermediate state representation becomes ambiguous in ways that affect what a transaction observes [151].

Developer burden. Three of the four families relocate work to the application author, and none of them make the relocation optional. Declared access sets must be correct and current; ownership typing decides at design time whether a contract's traffic gets a consensusless fast path; deferred objects only help if the developer knows which field is hot and never reveals it. The failure modes differ from ordinary bugs in that they are silent and performance-shaped. A mis-typed shared object does not break; it just never takes the fast path. Quantifying this burden is harder than it looks, because the obvious proxy, transaction failure rate, is confounded by bot-submitted trading traffic, as the Solana figures above show. We are not aware of a study isolating the failure rate attributable to stale or incorrect access declarations, and we flag it as the cleanest open measurement in this lane. Optimistic speculation is the one family that asks nothing of the developer, which is a substantial and under-priced advantage for EVM-compatible chains.

Fairness and the block producer. Scheduling for parallelism hands the block producer a new lever. Anthemius makes this explicit — a leader that produces a sequential block is indistinguishable from a leader honestly handling a sequential workload — and the interaction with censorship resistance and encrypted mempools is unresolved [130]. Combined with the impossibility result on pricing contingent access [112], this suggests that parallel execution has not yet been reconciled with the fee and ordering mechanisms discussed at L5.

VII. Layer 4: State and storage

The state layer is where the execution layer's design choices are paid for, and it is where the high-throughput L1s have converged most tightly: all four Tier-1 systems abandoned the hash-addressed Merkle Patricia trie over a general-purpose key-value store, because that structure turns every logical state access into a logarithmic chain of random reads. The organising observation of this section is that the state model is not a downstream implementation detail of the execution model but its precondition. Solana can schedule on declared accounts and Sui can type objects as owned only because their state is addressed in units the transaction can name, and the authenticated structure over those units then determines whether the parallel engine is fed fast enough to matter.

A. Design axes

We classify the state layer on four axes. **State addressing** (account, object, or UTXO) determines what granularity of conflict information is expressible at all. **Commitment structure** (hash tries, vector-commitment trees, LSM-organised authenticated stores, unified KV-plus-Merkle designs) determines I/O amplification and proof size. **Growth control** distinguishes unbounded growth, rent or deposit pricing, and expiry or pruning. **Bootstrap cost** distinguishes full replay, checkpoint sync, and statelessness with per-block witnesses. The four are coupled: statelessness is affordable only with a commitment whose witnesses aggregate, and pruning is safe only with a commitment the pruned node can still verify against.

B. Account, object and what the execution layer can see

The distinction that matters for performance is not account-versus-UTXO in the classical sense but whether state is addressed in units a transaction can enumerate before it runs. Solana's account model separates program code from account data and requires each instruction to name the accounts it touches, which is what makes declaration-based scheduling possible at all; Sui's object model attaches an ownership tag and version number to each unit, so independence is a property of the type rather than of the access pattern [120]. Ethereum's model, inherited by Monad and the parallel-EVM chains, addresses state as 256-bit storage slots reached through contract code, which is precisely why conflict information there must be recovered by simulation, tracing or optimistic detection rather than read off the transaction. The object model also gives the state layer a growth lever the account model lacks: Sui marks deleted objects with version-zero tombstones and clears them at epoch change, explicitly because leaving them would create unbounded memory need. The UTXO contrast case, where state is consumed rather than mutated, shows how far the addressing choice propagates [152].

C. Authenticated structures and the I/O they impose

The quantitative case against the hexary Merkle Patricia trie is the most robustly established result in this layer. Replaying Ethereum to block 7.3 million, Ponnapalli et al. [153] measured that processing a single roughly 100-transaction block requires more than 10,000 random I/O operations — a hundredfold amplification — taking hundreds of milliseconds even on datacenter-grade NVMe, and that the Merkle witness for a single 100-byte account can exceed 4 KB, a 40–60× overhead, against a state that already exceeded 200 GB in 2019 and roughly 16 TB by late 2023 growing about 4 TB annually [154]. Every subsequent design in this space is a response to one of those two numbers.

The vector-commitment line attacks the update path. LVMT replaces hash-based inner nodes with an authenticated multipoint evaluation tree and stores version numbers rather than value hashes to avoid elliptic-curve multiplications, updating commitment proofs in constant time and delivering reads and writes 6× faster than the MPT on real Ethereum traces, which lifts end-to-end execution throughput up to 2.7× [155]; earlier work in the same direction attacked the trie's storage bottleneck directly [156]. The Verkle line is now deployed rather than proposed: SonicDB S6 implements an EIP-6800-compatible Verkle trie with occupancy-aware node specialisation and delta

nodes, cutting live-state storage by 97.8% (817.1 GiB → 17.9 GiB) and archive storage by 95% (16,141 GiB → 809 GiB) across 40 million Sonic mainnet blocks while running 3.2× the throughput of a persistent Geth Verkle baseline [157]. That system's own cost report is exactly the kind of detail this survey exists to surface: on its highest-volume day three blocks exceeded the 300 ms block budget, reaching 450 ms, and adding node specialisations past a threshold improves storage while degrading throughput — a storage/throughput Pareto frontier that must be chosen, not optimised away.

A third line reorganises the data layout rather than the cryptography. COLE stores each state's historical versions contiguously under a compound key in an LSM tree indexed by learned models, cutting storage up to 94% and raising throughput 1.4×–5.4× against the MPT [154]; its declared costs are O(n) worst-case write tail latency under synchronous merges, an asynchronous fix that multiplies memory by the level count, and incompatibility with forking chains. QMDB unifies key-value and Merkle storage in an append-only twig structure giving one SSD read per state access and O(1) I/Os per update at 2.3 bytes of DRAM per entry, reporting up to 2.28 million state updates per second past 15 billion entries [158]. AlDBaran removes disk from the hot path by statically sharding the key space so each thread owns disjoint sub-trees of a sparse Merkle tree, computing roots lazily and holding every byte in DRAM until block commit, reaching 48 million updates per second on 96 cores [159]. Its evaluation also contains the most important negative result in this area: attempting to reproduce QMDB's published figure on the same class of machine, the authors obtained 1.28 million and 0.80 million updates per second on two code drops and could not close the gap. One of the field's headline state-layer numbers has a failed independent replication attached to it, with no resolution we are aware of. Work continues on sliding-window Merkle variants with delayed writes [160], erasure-coded state storage [161], structures tuned to dynamic workloads [162], and the memory-checking locality bounds that constrain all of them [163].

D. Storage engines and the hot path

Beneath the authenticated structure sits an ordinary storage engine, and a recurring finding is that general-purpose engines fit blockchain access patterns poorly — a point made early in the database framing of blockchains [107, 164] and pursued since through semantics-aware key-value separation [165], cold-store segregation of ancient state [166], content-defined versioned storage [167], hybrid blockchain-database architectures [168], and surveys of deployed engine choices [169, 170]. The most architecturally aggressive response disaggregates I/O from execution: RainBlock has prefetching clients fetch and compact Merkle witnesses and supply them with the transaction, leaving miners to execute in memory against a retained top layer, reaching 20k tx/s across four regions on three continents [153]. The cost is a conversion rather than an elimination: disk I/O becomes network I/O and DRAM pressure, shards must fit in storage-node memory (the authors reject full in-memory state at every node precisely because requiring hundreds of gigabytes of DRAM "decreases decentralization"), and clients may conservatively abort transactions that would have succeeded.

For the deployed Tier-1 systems, the storage-engine layer is where the published record thins out fastest. Solana's operational guidance — separate NVMe devices for ledger and accounts so that sequential-write and random-read patterns do not interfere, with production nodes converging on 24 or more cores, 256–512 GB of ECC RAM, and multiple enterprise NVMe drives — comes from client documentation and infrastructure-provider write-ups, not from peer-reviewed measurement. We record it as **E4** and flag the absence: we found no archival study that measures the accounts-database access pattern of a production Solana validator, which is a conspicuous gap given that this is the system whose execution model most depends on fast account access.

E. State growth, pruning and expiry

State growth is the cost that no deployed high-throughput L1 has priced, and recent measurement makes its magnitude concrete and chain-dependent. Sampling 3,000 blocks per day per chain through 2025 on archival nodes, Heimbach et al. [111] find that Base added 435.22 GB of persistent state against Ethereum's 29.92 GB — a 14.55× ratio, with storage slots accounting for 61.4% of Base's growth and 50.9% of Ethereum's, bytecode a further 25.4% and 18.2%, and daily growth varying by up to an order of magnitude. Their conclusion that "collapsing a high-dimensional workload into scalar gas is inherently brittle" is the L4 counterpart to the L5 argument for multi-dimensional metering: scalar gas prices state growth only incidentally, so a high-throughput chain accumulates it faster than its fee mechanism registers. That this is a security property and not merely a capacity one is established by He et al. [171], who show an adversary can deliberately inflate victim nodes' state-storage consumption by manipulating trie structure.

Mitigations divide into pruning historical data and expiring live state, and only the former is deployed at scale. Accumulator-based pruning lets a full node discard blocks after a snapshot receives sufficient confirmations, bounding storage at the snapshot interval plus the confirmation window and cutting storage 85% in simulation, at a cost of roughly 0.35 s of proof verification per block during bootstrap [172]; the same work documents that a competing snapshot scheme depending on miner reaffirmation can be delayed 1,000–2,000 blocks by 38% adversarial hash power, a reminder that pruning protocols have their own liveness surface. Snapshot-and-prune designs targeting fast bootstrap [173] and erasure-coded storage of ledger and cold state, which cuts per-node storage by over 90% against a full Geth node at the price of needing group cooperation to answer historical queries [174], occupy the

same trade space; the mitigation landscape as a whole has been surveyed [175]. Live-state expiry, by contrast, exists in the deployed high-throughput systems only in weak forms: Solana's rent-exemption deposit prices account data size at creation but does not reclaim, and Sui's epoch-cleared tombstones bound only deleted-object metadata. None of the Tier-1 chains expires live state.

F. Sync cost, statelessness and the hardware floor

Bootstrap cost is where state growth becomes a decentralisation cost, and it is the least well measured quantity in this layer. Sui's certified checkpoints exist explicitly to "reduce the memory footprint and bootstrap cost of new participants", but the paper reports no sync-time figure, noting only that a joining validator must download and validate required state before signalling readiness [120]. Statelessness would remove the requirement to hold state at all by shipping a witness with every block, and the modelling work makes clear that this changes the node's role rather than removing the cost [176]. Benchmarking the two leading witness schemes, Oberst [177] estimates a Verkle multiproof at roughly 0.8 MB for 5,000 keys against roughly 2.6 MB for binary Merkle trees with PLONK proofs, but reports the Verkle implementation becoming memory-bound above $2^{24–2}25$ leaves while the SNARK alternative was abandoned above 2^7 leaves because proving took minutes, a sobering gap between the asymptotics and the artefacts. Witness compaction against a retained top layer offers a middle path, cutting witness size by about 95% relative to the stateless-client proposal [153]. The one place bootstrap cost has been attacked with measured results is Ethereum: on the finding that "more than 90% of accounts are inactive" (over 95% inactive across a week or a month in their trace to block 8M), sweeping the state trie at each checkpoint and requiring dormant accounts to present restore proofs cuts a compact sync from 2,592.54 MB to 1,206.97 MB against `geth` over blocks 7M–8M [178]. Bootstrap cost is a function of *live* rather than total state, which matters more, not less, for chains whose state grows an order of magnitude faster. No equivalent measurement exists for any Tier-1 system.

The synthesis across this layer is that the state engine, not consensus, now sets the hardware floor for participation. A chain executing in parallel at hundreds of thousands of transactions per second must commit state at a matching rate, and the designs that achieve it hold state in DRAM until block commit [159], assume NVMe arrays delivering millions of IOPS [131], or distribute state across machines [133]. Each raises the cost of running a validator, and none of these papers measures the effect on the validator set — the boundary at which this section hands off to L6.

G. Costs

I/O amplification. Authenticated state access, not execution, was the original bottleneck and remains the binding one for MPT-based chains: >10,000 random I/Os per ~100-transaction block, 40–60× witness overhead per account, and a further 5–10× physical-write amplification from layering a trie on an LSM store [153, 159]. The replacements work, and 6×, 94% and 97.8% are real measured improvements, but each buys its gain with a specific structural concession: constant-time commitment updates require version numbers in place of value hashes, learned column storage forfeits fork support, unified twig stores are append-only, and DRAM-resident Merkleisation halves throughput as soon as snapshots are enabled and halves it again at a 500 ms snapshot period.

Unpriced state growth. Growth is monotonic, an order of magnitude faster on high-throughput chains than on Ethereum, dominated by storage slots, and deliberately attackable [111, 171]. No Tier-1 system expires live state, so the entire burden falls on operators through disk, and on new entrants through sync time.

Sync time and the operator. This is the cost we are least able to document and the one most likely to matter. Ethereum has measured bootstrap-reduction work [178], but we found no peer-reviewed measurement of snapshot or state-sync time for any Tier-1 system, no measurement of Solana's accounts-database access pattern under production load, and no study relating state size to validator-set composition on a high-throughput chain. The available evidence is operator documentation and vendor guidance, which is **E4** and should not be read as a performance fact. We flag this as the most serious evidence gap in the L3/L4 lanes.

Reproducibility. QMDB's 2.28 million updates per second failed an independent reproduction attempt on comparable hardware [159], and the storage-layer literature contains almost no head-to-head evaluation on a shared workload: each system is benchmarked against MPT or Geth on its own hardware.

VIII. Layer 5: Metering and fee mechanisms

Metering is where an L1 states, in numbers, what it believes its own bottleneck to be. Every fee mechanism embeds four commitments: how many resources it distinguishes (**metering dimensionality**), at what granularity it prices contention (**pricing locality**: global, per-account, or per-object), how the price moves (**adjustment dynamics**), and who absorbs the cost of congestion that a transaction imposes on others (**contention externality bearer**). A fifth axis is usually left implicit and does most of the damage when it is wrong: what the mechanism **assumes about execution**, whether serial or parallel, with resource use known at submission or realised only afterwards. The deployed monolithic L1s changed the execution assumption first and the fee mechanism last, and the resulting mismatch is the organising fact of this layer.

A. Scalar gas and its assumptions

Scalar gas is less a pricing mechanism than a claim that a chain's costs are one-dimensional up to a fixed exchange rate, and that claim is now measurably false on the systems

that rely on it. Sampling 3,000 blocks per day on Ethereum and Base through all of 2025, covering 219.8 million and 256.5 million transactions respectively, Heimbach et al. [111] decompose execution gas by operation category and find the mix differs sharply between two chains running identical opcode prices: storage writes account for 34.9% of Ethereum's execution gas, while Base spends 29.2% on storage reads and 24.3% on compute (Table 1b of that paper gives 34.88%, 29.23% and 24.30% respectively). The mix also drifts within a single chain over a year, and the opcode schedule does not follow it; repricing happens only through infrequent hard forks. A second assumption fails alongside the first. Gas metering presumes that the state a transaction will touch is stable between simulation and execution, and the same study shows that re-evaluating transactions against states up to twenty blocks earlier yields a non-zero coefficient of variation in the gas estimate for 46.0% of Base transactions against 13.9% on Ethereum, with write-set overlap falling to 59.3% on Base at that lookback. Higher throughput shortens the interval over which state can be treated as fixed, so the most aggressive block schedules produce the least stable estimates.

The most expensive mispricing in scalar gas is temporal: it prices a permanent cost as a transient one. Persistent state added in 2025 amounted to 435 GB on Base against 30 GB on Ethereum [111], and that storage is charged once, to the submitting transaction, while every full node carries it indefinitely. He et al. [171] turn the same gap into an attack. Because a transfer costs a flat 21,000 gas regardless of how many Merkle-Patricia-Trie nodes it splits, an adversary can craft leaf insertions that deepen a target subtree and permanently inflate the resource cost of every later operation over it. Replaying mainnet blocks 14.99M–15.00M against a forked trie, the attack raises state-modification time by 111% over 10,000 blocks; on Sepolia and BSC testnets it touched 32% and 39.4% more trie nodes per block and lifted end-to-end execution time by 10.7% and 12.4%. The cheapest configuration reported costs 39.64 USD to degrade Optimism over 10,000 blocks. Six production chains confirmed the vulnerability and paid bounties, making this one of the few metering defects with an acknowledged deployment footprint rather than a modelled one.

B. Multidimensional metering

Pricing several resources separately is theoretically well settled and empirically almost untried. Diamandis et al. [179] give the canonical construction: write the network's objection to any resource profile as a convex loss, maximise transaction-producer utility minus that loss, and the dual variables of the resulting program *are* the per-resource prices. Gradient descent on the dual recovers the additive update rule, and a log-transform recovers the multiplicative one, so EIP-1559 falls out as the one-dimensional special case of a general design rather than as a separate invention. A no-lookahead update rule turns out not to be much worse than a designer who knows all future demand. Angeris et al. [180] recast the update as follow-the-regularised-leader and bound the average welfare gap to an oracular price-setter at $O(1/\sqrt{T})$ against a worst-case adversary, with a matching lower bound for any implementable algorithm. They note that the multiplicative variant is exactly the rule Ethereum runs for blob fees under EIP-4844.

Multidimensionality has limits that are also formal. Bahrani and Durvasula [181] show that once node costs are heterogeneous, which is exactly what parallel execution with state conflicts produces, no vector of per-resource prices suffices: there exists a d-resource market with a single node where the best inclusion-maximal allocation attains only 2/d of optimal surplus, and surplus maximisation in general requires a separate price per transaction. Their proposal replaces protocol-set prices with competing brokers submitting full routings, relocating the problem to broker competition rather than dissolving it.

C. Dynamic base fees

EIP-1559 is the only fee mechanism here with both a complete game-theoretic account and a production measurement record, which makes its failure modes a useful template for what the high-throughput chains have not yet established about their own. Roughgarden [182] formalises a fee mechanism as an allocation, payment and burning rule, and shows the deployed design is incentive-compatible for myopic miners and proof against off-chain agreements, and dominant-strategy incentive-compatible for users *except* when the base fee is excessively low, that is, during exactly the demand spikes the mechanism exists to handle. The companion economic analysis [183] is explicit that the benefit is fee predictability, not fee level, and that no fee mechanism lowers fees without added capacity.

Three lines of work then attack the adjustment rule itself. Leonardos et al. [184] prove global convergence of the base-fee dynamics for step sizes below a demand-distribution-dependent threshold and show that above it the system admits formal Li–Yorke chaos: the base fee stays bounded, but block occupancy oscillates between full and empty. Their agent-based runs, 720 simulations of 600 blocks each calibrated on Ethereum blocks 10.90M–10.94M, locate Ethereum's deployed step size of 1/8 inside the stable region for the demand distributions they test, which is a statement about Ethereum's twelve-second blocks and not about a chain running sub-second slots. Azouvi et al. [185] show the rule is manipulable rather than merely fragile: proposing empty blocks to depress the base fee and harvesting the gap on subsequent full blocks is profitable for a proposer holding well under 30% of block production at the deployed parameters, and non-attacking smaller proposers may rationally join without collusion. On the empirical side, Liu et al. [186], using mainnet data plus four self-run geodistributed nodes, find the London fork cut median waiting time by roughly nine seconds and compressed intrablock gas-price dispersion, but also raised the fork rate by about 3% as average block size grew from 64.05 to 78.01 kbytes. This is the first direct measurement

of a fee-mechanism change buying user experience with consensus margin. The alternative-mechanism literature converges on the same fragility from the design side: Ferreira et al. [187] prove that EIP-1559's utilisation-based update can oscillate even when demand sits below supply, and propose a welfare-based posted price that is ex post incentive-compatible for myopic bidders and guarantees at least a quarter of optimal welfare. The capacity target itself is a robustness parameter: Ganesh et al. [188] show that raising Ethereum's throughput target from 50% to 60% of block capacity drops guaranteed availability to about 99.66%, invoking the first-price emergency auction roughly once every 300 blocks.

Whether any of this survives at sub-second block times is the open question for the Tier-1 systems, and the only study to ask it is a simulation. Zhivkov and Chen [189] model an EIP-1559 adaptation at 600 ms blocks and report that trailing-average smoothing overshoots the base fee by up to 47% during spam while per-block adjustment establishes a spam barrier within 17–32 s against Ethereum's 4–6 minutes. That is a single agent-based study of one chain and should be read as a hypothesis about high-frequency fee control, not as a result about Solana, Aptos, Sui or Monad.

D. Pricing locality and hot state

Parallel execution most directly invalidates global pricing. Two transactions touching disjoint state compete for nothing but inclusion, so a common congestion price transfers the cost of one popular market onto every unrelated user. Solana and Sui have deployed local fee markets in response, and Acilan et al. [113] give the sharpest published characterisation of what was actually shipped: first-price auctions on congested storage keys, which are not incentive-compatible and require substantial user sophistication to bid into. The proposed successor on Solana, an exponential fee controller per write-locked account, exists as a vendor specification (SIMD-110, by Yakovenko and Zhu) and enters the peer-reviewed record only by citation, in Angeris et al. [180]. The mechanism that most distinguishes a high-throughput L1 from Ethereum is thus documented at E4 and analysed at E1, with nothing in between.

The welfare case for locality is nonetheless solid. Ndiaye [190] models a chain as N queues (contracts, objects, or pieces of state) under a single global inclusion constraint, and derives socially optimal relative prices as Pigouvian taxes that internalise local congestion and the global capacity shadow price separately. The same model delivers the warning: a revenue-maximising validator prefers uniform pricing and will allocate all capacity to the highest-paying queue, excluding traffic that is welfare-positive but locally cheap. Locality is therefore not something a profit-seeking block producer converges to on its own; the protocol must impose it, and must choose how much of the global capacity price each queue sees.

E. Fee design under parallel execution

The deepest result in this layer is an impossibility recent enough that almost nothing builds on it yet. Parallel schedulers price transactions by the contention they cause, but a transaction's *realised* state access depends on execution outcomes: a swap that reverts touches far less than one that succeeds. Wadhwa et al. [112] formalise this as contingent execution and prove that no fee mechanism can simultaneously eliminate user risk (paying for declared objects that go untouched) and scheduler risk (uncollected revenue when a transaction under-executes), except by pre-executing every transaction or charging a constant fee. The escape route is closed constructively: deciding realised object usage is P-complete, so a parallel scheduler cannot compute it on the critical path. Any deployed mechanism therefore picks a point on a risk-division parameter α, and the two natural anchors, charging for what was declared and charging for what was used, are the endpoints of that line rather than neutral defaults.

Worse, the intermediate points are not automatically safe. Calibrating their object-weighted mechanism against Ethereum's observed priority-to-base fee ratios, Wadhwa et al. [112] find shill-proofness requires $\alpha \geq 0.15$ at the median ratio, $\alpha \geq 0.625$ at the mean, and is unattainable at the 95th percentile. They also show that the Shapley rule and the time-proportional makespan rule of Acilan et al. [113] both admit profitable shill transactions in both directions: a user can insert a decoy that lowers their own gas, and a scheduler can insert one that inflates collected revenue. The underlying design space is genuinely hard rather than merely unexplored: evaluating seven gas computation mechanisms against nine formal properties, Acilan et al. [113] find none satisfies all of them, with Shapley and Banzhaf pricing failing tractable gas estimation outright, leaving weighted-area and time-proportional-makespan as the surviving candidates. Neither is deployed anywhere.

A parallel literature shows the trouble is not confined to parallelism. Bahrani et al. [191] prove that once a block producer has private value in block contents, which is what MEV is, no non-trivial mechanism is incentive-compatible for both users and producer, and the mechanisms that satisfy both charge nothing and carry no welfare guarantee. Landis and Schwartzbach [192] show that an agent able to commit to a smart-contract bidding strategy can strip essentially all revenue from first-price, second-price, and EIP-1559 mechanisms alike whenever valuations are sufficiently concentrated. Constructive responses exist: discretised bids restore positive revenue under full incentive compatibility [193], a known honest-user bound plus an encrypted mempool yields $\Theta(h)\cdot$median revenue [194], a Bayesian relaxation gives constant-ratio approximations [195], and an N-price burning auction claims all three standard properties under congestion [196]. Each buys its way out through an assumption the deployed chains do not satisfy. Redistribution has the same shape: Damle et al. [197] prove that guaranteeing positive rebates requires abandoning proposer individual rationality once unconfirmed

transactions are excluded from rebates and the proposer can manipulate.

Two designs step outside the auction frame entirely. Kiayias et al. [198] prove EIP-1559 cannot implement any non-trivial diversity policy, because under congestion its single price surges for every transaction type at once, pricing out low-urgency traffic rather than delaying it. They propose delay-differentiated tiers that let patient transactions buy latency instead of exclusion. Kiayias et al. [199] tokenise block space into transferable access rights whose priority scales with token value and age, and prove that predictable fees, near-optimal worst-case inclusion delay and off-chain-proofness can hold at once. Both add a second market; neither is deployed.

F. Priority auctions and block building

The priority fee is where metering meets ordering. Solana's mechanism is a flat per-signature base fee of 5,000 lamports plus a compute-unit price, with no automatic adjustment. Zheng et al. [110], analysing 2.9 billion non-vote transactions across 53 sampled days between August 2023 and July 2024, find bot accounts fail 58.43% of their transactions against 6.22% for human accounts, that price-or-profit-not-met and invalid-status errors account for 67.18% of all failures, and that failed transactions pay *higher* fees per compute unit than successful ones while still consuming capacity. A pure priority auction over a parallel execution engine therefore converts contention into wasted capacity rather than into a price signal; the authors' own recommendation of a dynamic per-account fee structure is offered as future work.

The complementary result comes from rollups rather than from the Tier-1 L1s. Wang et al. [106] model spam bots as free-entry competitors for ordering-derived value under gas-limit charging and a minimum gas price, and validate on Base and Arbitrum: across 912 daily observations from January 2024 to June 2026, a 1% increase in the gas target is associated with a 1.63% increase in spam gas ($R^2 = 0.62$), spam gas grew 122× from Dencun to its April 2025 peak while non-spam grew 11.2×, and introducing or raising a gas price floor cut spam's share of total gas from 28% to about 10% on Base and by 27% in volume on Arbitrum. The mechanism is general, since charging reserved gas rather than gas used, as Solana and Monad do, is what makes spam cheap, but the measurement is not on either chain and the extrapolation should be flagged as such. The defensive literature is thinner still: Chotkan et al. [200] report that relay-layer sender reputation admits only 4.6% of spam while dropping about 3% of honest traffic, where fee filters and a simulated SIMD-110-style per-account penalty each let over half through, but this is one simulation replaying one 2022 Ethereum mint event.

Fee mechanisms also interact with who builds the block. Stouka et al. [201] extend the standard model to bribery and multiple includers, showing that reimbursing inclusion-list participants raises censorship resistance when they are rational rather than altruistic, at the cost of asking users to set two fees. Saraf et al. [202] quantify the baseline: with a single proposer and Ethereum's early-2026 burn multiple of roughly 8.7× the average tip, economic censorship resistance is about 0.09, meaning a briber needs to match only a small fraction of a transaction's total fee to buy its exclusion. Fee burning, adopted for its incentive properties, directly degrades the one property that makes inclusion credible.

G. Costs

The costs at this layer divide into four groups, and their evidence grades differ systematically. **Mispricing under parallel execution** is the most severe and rests almost entirely on E1 work: the user-risk/scheduler-risk impossibility and the shill-proofness thresholds come from a single 2026 paper [112], with the gas-computation-mechanism property analysis of Acilan et al. [113] as the only independent corroboration. No deployed system has been measured against either result. **Dimensional and temporal mispricing** is the best-evidenced: the drifting resource mix, the state-growth subsidy, and the instability of gas estimates are E3 measurements [111], and the storage-exhaustion attack they imply has been demonstrated and acknowledged by six chains [171]. **Fee-market spillover from hot state** is asserted everywhere and measured nowhere on a Tier-1 L1: the welfare argument is E1 [190], the characterisation of what Solana and Sui deployed is secondhand from vendor documentation [113], and the quantitative spam-displacement evidence is from Base and Arbitrum [106]. **Incentive failures under adversarial proposers**, covering base-fee manipulation, Stackelberg commitment attacks and the active-producer impossibility, are uniformly E1 and uniformly about Ethereum's mechanism; whether a sub-second, multi-leader chain inherits them, amplifies them, or escapes them is not established anywhere we could find.

IX. Layer 6: Robustness and the cost of decentralization

Layer 6 asks what the stack costs when the environment stops cooperating, and who is priced out of participating. Five axes organise the evidence: the **fault model actually exercised** (crash, equivocation, performance degradation, asynchrony, or economic bribery), the **degradation metric** reported (throughput, latency, chain quality, inclusion), the **resource floor** a validator must clear (CPU, memory, bandwidth, storage), the **decentralization resource** being measured (block production, stake, geography, client implementation, order flow), and the **estimator** used to measure it. The last axis matters: estimator choice can move a decentralization verdict further than chain choice.

A. Performance under faults

Headline throughput figures are fault-free figures, and the gap is large and systematically under-reported. The cleanest quantification is analytical: Wang et al. [87] build a multi-metric framework for chained HotStuff and

show that under their attack strategies, with one third of nodes Byzantine, chain growth rate falls to 4/9 and chain quality to 12/17 of the attack-free values, and that some of DiemBFT's engineering optimisations make it *more* vulnerable than the protocol it derives from. Optimisations that buy latency in the common case frequently spend robustness in the uncommon one, and the trade is rarely reported as such.

Direct fault injection reaches the same conclusion empirically. Bano et al. [94] generate Byzantine behaviour by running twin node instances that share credentials, executing 44 million scenarios per day against DiemBFT; they find no errors in the production protocol but recover several known BFT attacks within a dozen rounds and surface deliberately injected safety bugs in minutes. Its scenario space must be pruned heuristically, so absence of findings is weaker than it looks. On DAG-based ordering, the family that Sui, Aptos and the Narwhal lineage deploy, Mirzaei et al. [97] instrument six production Rust implementations (Narwhal-Tusk, Bullshark, Mysticeti, AlephBFT, Mahi-Mahi, Autobahn) with attacker hooks and run over fifty configurations at committee sizes 13, 25 and 49; every protocol tested was vulnerable to at least a subset of the attacks, and which attacks succeed is determined mostly by protocol design rather than attacker effort. A front-running attack raised success from 52.9% to 100% on Narwhal-Tusk, and vote withholding raised censorship success from 0.4% to 84.4% on Bullshark.

Degradation is not inevitable, which is what makes its absence from most evaluations a reporting failure. Tonkikh et al. [203] deploy 100 replicas across ten Google Cloud regions with round-trip times from 25 to 317 ms and show their prefix-consensus design sustains 260,000 TPS at sub-second latency while a 1% message-drop rate at every replica raises latency only about 15%, where an optimistic DAG baseline degrades sinusoidally and a partial glitch inflates its latency by roughly 35%. Latency variance also has an incentive origin rather than only a fault origin: Sun et al. [204] show that granting a proposer all tips rewards delaying the proposal to harvest extra ordering value, and report that tying rewards to observed transaction arrival times cuts latency variability by up to 95.9%. Spiegelman et al. [55] report up to 80% lower latency than vanilla Bullshark under failures by adding leader reputation and pipelining. Gai et al. [205] show a robust shared mempool delivers 5× to 20× the throughput of native leader-based protocols at 128 replicas and degrades gracefully under asynchrony, Byzantine attackers and Zipfian workload skew, at roughly 10% extra bandwidth for availability proofs. Ling et al. [89] report retaining 40% of throughput at a link-failure probability of 0.4 where HotStuff retains less. Monad's consensus layer claims tail-forking resistance, so that a QC-backed block cannot be abandoned unless its leader equivocates and a single Byzantine leader costs one view, but Jalalzai et al. [52] state it as theorems with no accompanying deployment measurement, and it should be recorded as an E4 design claim.

The one category of evidence the academic record almost entirely lacks is the deployed failure itself. Solana's April 2022 halt, in which spam bots submitting over four million transactions per second saturated validator bandwidth and memory and stopped the chain for roughly seven hours, followed by congestion episodes in which over 70% of non-vote transactions failed, enters the peer-reviewed literature only as motivating background in Chotkan et al. [200], citing press reports and vendor post-mortems. There is no peer-reviewed post-mortem of any Tier-1 mainnet outage, and the closest production corroboration is the elevated steady-state failure rate measured by Zheng et al. [110]. Outage causation on these systems is therefore an E4 record in a survey that grades everything else more strictly.

B. Ordering-layer attack surfaces

DAG-based ordering, adopted for throughput, creates extraction surfaces that the linear-chain MEV literature does not describe. Valentini et al. [100] make the structural point: because blocks in a DAG have multiple predecessors, manipulation acts on the ordering of *blocks* rather than of transactions within one, and most extraction techniques designed for linear chains are inapplicable. The quantitative picture comes from the fair-ordering work. Segalini et al. [104] measure front-running success against unprotected Narwhal-Tusk at 26.3% to 95.2% depending on the number of faults, falling to 0% under their fair-ordering construction across committee sizes 10 to 25 and up to five faults. Putnik and Decouchant [103] report that under a reversing-order attack their batch-order-fair protocol reverses 11–16% of transaction pairs where a HotStuff-based baseline reverses 35–44%. Both pay for it. The fairness layer is CPU-bound on pairwise weight-matrix construction, at 193.9 ms per phase against 63.9 ms for the next most expensive stage, and batch-order-fairness costs fault tolerance, requiring $n > 4f/(2\gamma-1)$ rather than $n \geq 3f+1$. Cryptographic prevention has a comparable profile: Malkhi and Szalachowski [206] integrate commit-reveal ordering into DAG transport with no extra messages, at about 3.7 ms of CPU per transaction using threshold encryption, reduced to roughly 66 μs with dispersal-based secret sharing, in a 6-of-16 micro-benchmark on a single machine; a whole-protocol evaluation is explicitly left as future work.

Concurrent proposers, the structural answer to censorship, bring their own surface. Saraf et al. [202] prove economic censorship resistance scales roughly linearly in the number of proposers, about 0.09 with one proposer against about 2.5 with 49 at Ethereum's early-2026 burn multiple, and that the throughput cost of duplicated inclusion is a one-time penalty (about 0.65 at their baseline under a duplication-penalising payment rule) rather than a compounding one; their calibration uses 10,000 recent Ethereum blocks. Against that, Landers and Marsh [207] enumerate MEV channels that exist only under concurrent proposal: same-tick duplicate stealing, proposer-to-proposer transaction auctions, and latency races in proof-of-availability certification, all in a stylised model of an Autobahn-style protocol at roughly 300 ms ticks. It is

purely analytical and, as far as we established, the only treatment of the topic: a hypothesis-generating model, not a characterisation of any deployed system.

The two extraction regimes measured in production both belong to other architectures. Daian et al. [208] observed nine months of Ethereum mempool traffic across six geodistributed nodes and found ordering-derived fees that dwarf block rewards, reaching 101.6 ETH against a 3 ETH block reward in the most extreme block observed, establishing ordering revenue as a consensus-stability problem rather than an application-layer nuisance. Öz et al. [209] analyse 13.7 million Algorand blocks and 745.8 million transactions and show that first-come-first-served ordering does shift extraction away from fee-driven front-running toward backruns spread uniformly across block positions, but that an adversary can push the network into fee-based congestion mode for roughly 3 ALGO, reopening front-running. Neither Solana's nor Monad's ordering-derived extraction has been characterised at this level of rigour in the peer-reviewed literature.

C. Hardware, bandwidth and who can validate

The decentralization cost of speed is paid at the validator hardware floor, which is visible in the very experiments that establish the throughput. The 260,000 TPS figure of Tonkikh et al. [203] is measured on 100 replicas of 64 vCPUs and 256 GB RAM each; Spiegelman et al. [55] use 32-vCPU VMs; the Narwhal results that underpin the DAG lineage report 160,000 tx/s with one machine per validator and 600,000 tx/s with ten machines per validator [1]. These are not incidental deployment details. They are the operating point at which the advertised number holds, and a validator that cannot meet it becomes the straggler the protocol is designed to route around. Leader-reputation schemes make this explicit by deprioritising slow validators as proposers [55]; the decentralization consequence of that design is a direct implication, not something the paper measures.

Core count converts into throughput at a measurable rate. Using Ethereum mainnet traces (blocks 21,631,019–21,635,079) on a 96-core machine, Karmegam et al. [132] show conflict-aware scheduling of an already-ordered block gains about 1.57× on two cores and over 2× on eight, while reward-maximising block construction gains up to 7.9× on eight; exact optimisation is intractable in practice, with MILP solvers timing out at 6,000 s on larger instances, so achievable parallelism depends on the quality of a validator's heuristics. The execution-engine literature reports the same asymmetry from the other side: Gelashvili et al. [3] measure Block-STM at up to 110k tps in Diem benchmarks and 170k tps in Aptos benchmarks with 32 threads, but 50k and 80k tps respectively on contended workloads. The headline numbers are low-conflict numbers, and the gap is the execution-layer twin of the hot-state fee problem in Layer 5. Bandwidth imposes a floor no execution optimisation removes: Chu and Wang [210] derive a throughput ceiling from access bandwidth and transaction size binding on the slowest included nodes, and their 2018 figures (90% of Bitcoin mining nodes above 5.7 Mbps, 90% of Ethereum nodes above 3.4 Mbps) show how far below datacentre provisioning the tail sits. Di Perna et al. [211] add the methodological corollary, showing across five chains including Solana that network topology is a first-order driver of throughput, latency and energy, so cross-system throughput comparisons are not meaningful unless topology and workload are held fixed.

D. Measuring decentralization

Quantitative decentralization is measurable, contested, and, critically for a comparative survey, sensitive to choices that most papers do not report. Ovezik et al. [212] systematise the pipeline into resource choice, pre-processing and metric, measure six chains, and show that estimation windows, inclusion thresholds and address clustering move the resulting numbers materially: the Gini coefficient of $\{3, 2, 1\}$ is 0.22 while that of $\{3, 2, 1, 0\}$ is 0.42, so whether zero-block participants are counted changes the verdict. They recommend resource-estimation windows of at least seven days and caution against windows under 150 blocks. Decentralization figures from different papers are therefore incomparable unless the pipeline matches. Earlier work shows the same fragility: Lin et al. [213] compute Gini, Shannon entropy and Nakamoto coefficient over 2019 Bitcoin and Ethereum blocks at three granularities and find that the Nakamoto coefficient sits between 2 and 5 across both chains while fixed windows miss cross-interval changes that sliding windows reveal. Zhang et al. [214] argue that transaction-level decentralization is the least-studied of the five facets they identify and propose an entropy-based index for it; Lee et al. [215] propose censorship-resistance and geographic-diversity quotients as complementary measures.

For the Tier-1 systems specifically, two studies matter. Motepalli and Jacobsen [216] collect validator snapshots from ten proof-of-stake chains including Aptos and Sui and report liveness Nakamoto coefficients between 2.93% and 16% of the validator set and Gini coefficients from 0.35 to 0.8 under linear stake weighting; replacing linear weight with square-root or logarithmic transforms improves their metric suite by 51% and 132% on average, with the same conclusion reached in their earlier study of stake distribution [217]. Motepalli et al. [92] add the geographic dimension across Solana, Aptos, Sui, Avalanche and Ethereum, reporting a Gini of eigenvector centrality between 0.527 and 0.941 (Sui 0.527, Aptos 0.544, Solana 0.791), with the top three countries holding over a third of stake on every chain measured. These are the only quantitative decentralization measurements of the Tier-1 systems we located: both from the same group, both single-snapshot (September–October 2024), and neither covering Monad. Any comparison of the high-throughput L1s on decentralization rests on those two papers.

E. Client and order-flow concentration

Two concentration risks sit outside the stake distribution and are usually omitted from decentralization scores. The

first is client diversity. Luo [218] instruments five Geth nodes against Ethereum's devp2p network, collects over eight billion messages across 743,739 unique peers, and finds only 78.05% of peers run the latest mainnet sub-protocol and just 24.5% carry a fully correct mainnet configuration, with 12.6% behind a recent hard fork and the mainnet share of reachable peers falling from 54.5% in 2018 to 13.07%. Ron et al. [219] treat the skewed client distribution as a systemic risk and prototype verifiable-execution proofs on a modified Lighthouse client to reward minority-client operators, at the cost of proof production and verification overhead. Howell et al. [220] measure node distribution across Bitcoin, Cosmos, Stellar and Lightning and find the smaller networks concentrated in a few cloud providers and data centres. No equivalent measurement exists for Solana, Aptos, Sui or Monad. Solana's move toward a second independent validator client is exactly the kind of change this literature exists to evaluate, and it has not been.

The second is order flow. Yang et al. [221] analyse 5.55 million Ethereum blocks and 4.9 million winning relay bids from September 2022 to October 2024 and find two builders produced over 85% of blocks; decomposing the resulting proposer loss, they show it comes overwhelmingly from inequality in block-building *capacity* (5.6–11.5%) rather than uncompetitive auctions (0.5–1.7%), with daily median estimated loss reaching 21.2% in 2024 and a few exclusive order-flow relationships responsible for a third of it. Ma et al. [222] reach a compatible conclusion analytically: under order-flow auctions the builder with a competitive advantage pays less and earns more, concentrating the builder market while proposer shares evolve as a martingale. The high-throughput L1s do not run proposer-builder separation in Ethereum's form, but they do concentrate ordering rights in a rotating leader with a sub-second slot and, on Solana, a dominant off-protocol block-engine and relay stack. Whether this produces the same capacity-inequality dynamic is unmeasured.

F. Costs

Throughput collapse under faults is real, measurable, and measured mostly on research protocols rather than on deployed L1s. The chained-HotStuff degradation figures [87], the DAG-BFT attack-success sweep [97], and the packet-loss comparisons [203] are E1 and E2; none of the Tier-1 systems has published a fault-injected throughput curve for its mainnet configuration, and no vendor throughput claim is accompanied by one. **Ordering-bias and MEV surfaces specific to DAG ordering** are established at E2 on research implementations of protocols that Sui and Aptos derive from, with defences that cost CPU, latency and fault tolerance in quantified amounts. **The hardware and bandwidth floor** is documented indirectly but consistently: the operating points in the throughput experiments are themselves the evidence, and the one study that prices cores against throughput [132] shows the return to hardware is steep enough that a validator's provisioning determines whether it keeps up. **The measurable decentralization consequence** of that floor is the weakest link in the chain: the geographic and stake concentration figures for Solana, Aptos and Sui [92, 216] are real but do not causally attribute concentration to hardware requirements, and no study we found measures validator hardware cost against validator-set composition on any of the Tier-1 chains. Finally, **published throughput claims are not independently corroborated at the scale advertised**: the two independent Solana measurements in the literature report roughly 2,812 TPS averaged over two months of RPC collection [223] and 663 TPS for a single application-level Rust benchmark [224], both counting differently and both far below headline figures. The divergence, as the benchmarking literature has argued for years [22, 211], reflects the absence of an agreed measurement definition as much as any property of the chain.

X. The cross-layer trade-off matrix

Collecting the costs documented in the six layer sections into one table makes two things visible that no single layer section can show: what kind of cost this generation of designs actually pays, and how often the cost is paid at a different layer from the one that provides the benefit. The matrix holds 200 mechanism-cost pairs over 131 distinct mechanisms, each row carrying the mechanism, the systems that deploy it, the advantage claimed, the cost, a cost type, an evidence grade and a citation. It is the artifact we would most want a reader to disagree with row by row, which is why every row names its source.

The evidence profile differs sharply by layer, and the difference is itself a finding. Ordering costs are the best-established: 23 of 49 rest on proofs, because latency in message delays and fault-tolerance thresholds are the kind of claim a paper can prove, and another 23 on controlled experiments. Metering costs are similarly proof-heavy for the opposite reason: 21 of 35 come from mechanism-design impossibility results rather than from measurement, because almost nobody has measured a deployed fee market. Execution and state are where production measurement concentrates (11 and 7 rows respectively at E3), which is unsurprising given that conflict rates and state growth can be read off a public chain without operating a validator. Across the whole matrix, 37 of 200 cost claims rest on production measurement and 9 rest on vendor documentation with no independent corroboration.

Three patterns recur across layers, and each is a co-design effect rather than a property of any one mechanism. The first is that **contention converts throughput into waste**, and the waste is charged twice. Optimistic speculation degrades from a 20-fold to a 9-fold speedup as contention rises and pays up to 30% overhead on a fully sequential block; declared access sets, being conservative supersets of what a transaction touches, leave roughly 8% of transactions in a Solana block independent against about 63% on Ethereum, with effective parallel width 0.36 against 11.17. The same contention then reappears at the fee layer, where a priority auction over a parallel engine

TABLE III
TRADE-OFF MATRIX BY LAYER: MECHANISMS, DOCUMENTED COST ROWS, DOMINANT COST TYPES, EVIDENCE-GRADE MIX, AND ROWS WHOSE COST LANDS IN A DIFFERENT LAYER.

Layer	Mechanisms	Cost rows	Dominant cost types	E1	E2	E3	E4	Cost lands elsewhere
L1 dissemination	13	16	bandwidth (5), memory (2), complexity (2)	2	14	0	0	3
L2 ordering	49	49	latency (8), security (8), complexity (7)	23	23	1	2	8
L3 execution	13	43	CPU (16), developer burden (8), complexity (5)	8	21	11	3	9
L4 state	6	24	storage (6), developer burden (3), CPU (3)	2	13	7	2	4
L5 metering	20	35	economic (17), fairness/MEV (6), complexity (4)	21	5	8	1	3
L6 robustness	30	33	CPU (7), decentralization (6), fairness/MEV (5)	4	18	10	1	3

prices the attempt rather than the outcome: bot accounts on Solana fail 58.43% of their transactions against 6.22% for human-initiated accounts, and the failures still consume scheduling capacity. A survey that treated execution and fees separately would report the first number as an execution limitation and the second as a market-design observation, and would miss that they are one phenomenon measured twice.

The second pattern is that **latency below three message delays is currently bought with fault tolerance**, not with engineering. The fast-BFT cohort trades roughly thirteen percentage points of Byzantine threshold per message delay saved, and no such design is in production, so the trade is attested only by laboratory and simulation results. Geography sets the floor underneath all of it: inter-region round trips of 100 to 250 milliseconds across five independent testbeds mean that a sub-500-millisecond commit claim implies either a concentrated validator set or a claim about something other than global commit. This is the clearest case in the matrix where a performance number and a decentralization property are the same measurement read in two directions.

The third pattern is that **capacity is bought with operator cost, and the bill arrives at the layer nobody surveys**. Authenticated state access already demands more than ten thousand random I/O operations per hundred-transaction Ethereum block, with 40 to 60-fold witness overhead per account; scalar gas prices that permanent cost as a transient one, so 435 GB of persistent state was added on Base in 2025 against 30 GB on Ethereum, and one reported configuration degrades a production chain's storage path for 39.64 USD over ten thousand blocks. Each of those is a state-layer or metering-layer row whose consequence is an L6 hardware floor, and the link that would close the argument — a study relating state size or commitment hardware to validator-set composition on a high-throughput chain — does not exist in our corpus.

Thirty of the 200 rows have a cost whose type belongs to a different layer than the mechanism that causes it, under a conservative assignment that excludes generic cost types such as complexity and security. That count understates the phenomenon, because it only catches costs the source paper itself named; the three patterns above were assembled from rows that individually looked like single-layer costs. It is nonetheless the quantitative form of this survey's central claim: a sixth of the documented costs in the high-performance stack are paid somewhere other than where they are earned, which is precisely the structure a technique-first taxonomy cannot represent.

XI. Claim provenance: what the published numbers support

The throughput and latency figures attached to high-performance Layer-1 systems circulate without their conditions, and the conditions are what determine whether two figures can be compared at all. We extracted every quantitative performance claim we could find in the corpus and in the grey literature, 299 in total, recording for each one the system, the metric, the value verbatim, and whatever the source states about validator count, geography, hardware, workload, contention and injected faults. We then checked each extracted figure against the source text and cleared any condition we could not corroborate by an adjacency match, so that a validator count survives in the table only when the number appears next to a validator, committee, replica or machine term in the same passage as the figure.

The result is a table in which most numbers are not comparable to most other numbers. Of the 299 claims, 81 carry three or more stated conditions, 35 carry two, and 183 carry one or none. Ninety-two are numeric throughput figures; 14 of those have a corroborated validator count, and 6 of the 14 come from the system's own paper rather than from a competitor's related-work paragraph. Attribution is itself a problem: 78 of the 299 rows are a paper reporting some *other* system's number, which is how a figure obtained under one testbed configuration propagates into comparison tables built under another. Nine rows come from industry and media sources, where advertised peaks and observed averages appear in the same column.

The incommensurability is structural rather than incidental. Across the ordering-layer literature alone, validator counts behind headline figures range from four to 106,

machine classes from `t3.large` to `m5d.8xlarge`, and geographic spreads from intra-US to ten regions on four continents. Throughput itself is reported at two different points in the pipeline, consensus commit and completed execution, which diverge in exactly the systems we survey because ordering and execution are decoupled. Workloads range from 512-byte no-op payloads to contended application traffic, and the ratio between them is not a constant: the whole point of the execution-layer findings is that realized parallelism depends on the workload's conflict structure. Independent benchmarking that drives realistic decentralised applications against real chains found that none of the six systems tested could support those workloads, and identified missing experimental conditions as the reason published results are not reproducible [24]; fault-injection measurement then shows performance degrading further once a chain is perturbed [4].

By evidence grade, 183 claims come from laboratory experiments, 37 from production measurement, 11 from formal analysis, 10 from simulation, 53 are either asserted in a paper without an attached experiment or reported by a vendor or media source, and 5 could not be graded from the source text. The two grades a reader most wants to compare, laboratory and mainnet, are also the two whose gap is least documented per system: for the tier-one chains we found no case where the same workload definition was run on a testbed and on mainnet by the same authors.

Three consequences follow for how the field should report performance, and they are cheap to adopt. A throughput figure should name the point in the pipeline it measures, the validator count and geographic spread, the workload's conflict structure, and whether any node was faulty; without the first and third of those, the number is not interpretable, and without the second it is not reproducible. A comparison table should not mix attribution classes, because a competitor's related-work paragraph is not a measurement. And a claim that exists only in a whitepaper should be labelled as a design target, which is how we have labelled Solana's and Monad's core figures throughout this survey: the absence of an archival description is a fact about the state of the literature, not a gap to be filled by quoting documentation as though it were a result.

XII. Open problems

The gaps worth naming are the ones where a specific, feasible piece of work would settle a question the field currently argues about from design documents. Our six layer sections flagged twenty-one such gaps; they cluster into five, ordered here by how tractable we judge them to be.

The chains with the largest deployed throughput have the thinnest archival description. Solana's ordering layer has no peer-reviewed or preprint characterization: no source we retrieved establishes its commit latency in message delays, its guarantee under asynchrony, or its behaviour under a faulty leader. Its execution runtime is in the same position, and so are Monad's execution engine and state database. This is not a matter of citation etiquette. It means the most consequential mechanisms in production — a declared-access-set scheduler serving the highest-volume public chain, and an optimistic parallel EVM whose design targets are quoted throughout the trade-off literature — are surveyed here through their measured consequences rather than through any description their authors have committed to. The tractable version of this problem is a characterization paper written by anyone, vendor or academic, that states the guarantees in the terms the literature uses. A harder version, and the one the field should want, is an independent measurement study of a production Solana validator's account-database access pattern under load, which does not exist despite Solana being the system whose performance depends most directly on it.

Parallel execution and fee mechanisms have not been reconciled, in theory or in deployment. The impossibility result is sharp: when resource use is contingent on execution outcomes, no mechanism eliminates both user overpayment risk and scheduler revenue risk without pre-execution or constant fees. It has no independent corroboration and no deployed counterpart. Meanwhile the fee markets that *are* deployed on tier-one chains have never been measured: Solana's and Sui's per-object write-lock markets are first-price auctions on congested keys, and no study reports their spillover onto unrelated transactions, their mispricing, or their welfare effects. Base-fee stability analysis, likewise, has been carried out only at twelve-second block times, and whether its convergence thresholds survive 400-millisecond slots is unknown. A measurement study of one deployed local fee market would be the single highest-value empirical contribution available in this area today.

Fault behaviour is evaluated on research implementations, never on the deployed configurations. Every degradation curve in our matrix comes from a research prototype or a cloud testbed. No fault-injected throughput curve exists for any tier-one mainnet configuration, and Byzantine-fault evaluation of DAG protocols is an open question by the field's own admission, so the fault-behaviour axis of our ordering taxonomy rests on crash-fault experiments plus formal Byzantine bounds. The gap matters because the costs that only appear under perturbation are precisely the ones absent from peak-throughput evaluations: the persistent post-recovery degradation traced to a speculative executor is the kind of finding that only fault injection produces.

The link between layer choices and decentralization is asserted rather than measured. Concentration measurements exist and hardware measurements exist, but no study causally relates a validator hardware or bandwidth floor to validator-set composition on any high-throughput Layer-1, and no study relates state size or state-commitment hardware to who can afford to run a node. Two further pieces are missing: no peer-reviewed measurement of snapshot or state-sync time exists for any tier-one system, which is the cost that actually gates validator entry, and client diversity is unmeasured

outside Ethereum even though Solana's transition toward a second independent validator client is an unexploited natural experiment. Solana's peer-to-peer layer also cannot be crawled with standard techniques, so its published decentralization figures depend on a third-party explorer and rest on weaker evidence than the equivalent Ethereum numbers.

Scheduling and storage both lack a shared evaluation basis. Optimal conflict-graph scheduling is NP-hard and inapproximable for homogeneous transactions, and for heterogeneous transaction lengths the natural colouring heuristics are not optimal, so the practical question is which approximation is best on real workloads — a question no common benchmark currently answers. Static extraction of read and write hints from contract logic is assumed by several hint-assisted schedulers and has not been implemented or evaluated end to end. The storage layer has the same problem in a sharper form: every authenticated-structure design is evaluated against a Merkle Patricia Trie or a Geth baseline on its own hardware and workload, which makes the published update-rate figures incomparable. A workload suite with a fixed conflict structure and a fixed state profile, usable by both scheduler and storage papers, would let the next five years of results be compared to each other.

A note on what we are not claiming. Three of these five clusters are gaps in *measurement* rather than in mechanism design, and measurement gaps can close quickly once someone runs the experiment. We have tried throughout to distinguish a mechanism whose cost is unknown from one whose cost is known and unattractive, because the two call for different responses: the first is an invitation to instrument a chain, and the second is a design constraint the next generation of systems will have to engineer around.

Appendix A
Methods, coverage and limitations

A. Search and screening

The corpus was built to the protocol registered before screening (`search_protocol.md`): 65 queries across eight layer groups over OpenAlex and arXiv, a 2019-2026 primary window with named foundational exceptions, and one backward and one forward citation step from 46 seed papers. Retrieval returned 4,916 records (1,889 keyword, 1,452 forward citations, 1,070 backward references, 505 arXiv), 3,663 unique after deduplication. Rule-based criteria removed 2,267 (842 application-domain, 574 off-topic, 841 outside the window or of an excluded type, 10 token economics). The remaining 1,396 were screened against the inclusion criteria at title-and-abstract level, yielding a working corpus of 810 records, 762 with resolvable DOIs. Each layer received a scored shortlist; every layer team also retrieved beyond its shortlist, and in two cases most cited work came from the full corpus or from fresh targeted retrieval rather than from the shortlist.

B. Evidence grading

Every cost claim in the trade-off matrix and every performance figure in the audit carries one of four grades: E1 for a proved claim, E2 for a controlled experiment, E3 for production measurement, E4 for a whitepaper, documentation, blog or media source. E4 sources are cited as design statements only. No performance number is attributed to Solana's Sealevel runtime, to Monad's execution engine or state database, or to Alpenglow, because no archival description of any of them exists in the corpus.

C. Access limitations

A material fraction of the venue-of-record literature could not be read directly: IEEE Xplore, the ACM Digital Library, ScienceDirect and MDPI are outside this environment's network allowlist, and several VLDB and ACM papers returned no open-access copy. Where an arXiv counterpart existed we read it and cited the venue of record; where none existed we cited at abstract-and-metadata depth and attributed no experimental detail. Concretely: the ordering lane read six venue-of-record papers from arXiv counterparts and could not obtain Sailfish at all, leaving four of its seven taxonomy axes unasserted; the execution lane could not obtain full text for roughly twenty publisher-hosted papers, including Spectrum, Remora, LVMT, ChainKV, MoltDB, Nurgle and EVMTracer; the metering and robustness lane read 18 of its 66 cited works from abstract and indexed metadata only. Vendor documentation for the tier-one chains could not be retrieved directly either, so hardware floors are reported as the measured operating points of published experiments rather than as published requirements.

D. Coverage gaps we know about

Tier-2 systems (Sei, Fuel, Linera, MegaETH) yielded no on-topic execution or state records and are absent from those sections. Monad appears in the taxonomy and matrix almost entirely through design documents and through papers that name it as a target without measuring it; no independent measurement of Monad was located. Two on-topic DOIs would not verify and were therefore not cited, one of them a Sui write-conflict analysis that is directly relevant to the execution layer. Several individual findings rest on a single unreplicated source and are labelled provisional in the text: the Solana-versus-Ethereum realized-parallelism comparison and Sui's fast-path share statistics are the two most load-bearing.

E. Corrections applied during assembly

Two statements from the layer drafts were tightened at assembly. The characterisation of Solana's transaction-failure composition now reports the account-class and program concentration the source measures, without attributing an economic motive the source does not state. The description of the Diablo benchmark now reports what

its abstract establishes, that none of the six evaluated systems supported the realistic application workloads and that published results generally omit their experimental conditions, because the full text was not retrievable from this environment.

F. Verification

The manuscript and tables cite 224 distinct DOIs; all resolve and none carries a retraction notice, and the bibliography holds one entry per distinct identifier. 105 resolve through CrossRef with full metadata; the remaining 119 are DataCite-registered identifiers (arXiv, Dagstuhl LIPIcs, episciences) for which resolution was confirmed and metadata taken from OpenAlex or the arXiv API. Counts of DOI strings taken case-sensitively from the prose run slightly higher, because a handful of identifiers appear in more than one letter case. Quantitative claims extracted from full texts were checked against the source text, and any measurement condition that could not be corroborated by an adjacency match in the same passage was cleared from the audit table rather than reported.

References

[1] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient bft consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, ser. EuroSys '22. ACM, Mar. 2022, pp. 34–50. [Online]. Available: http://dx.doi.org/10.1145/3492321.3519594

[2] K. Babel, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, A. Koshy, A. Sonnino, and M. Tian, "Mysticeti: Reaching the latency limits with uncertified dags," in *Proceedings 2025 Network and Distributed System Security Symposium*, ser. NDSS 2025. Internet Society, 2025. [Online]. Available: http://dx.doi.org/10.14722/ndss.2025.240929

[3] R. Gelashvili, A. Spiegelman, Z. Xiang, G. Danezis, Z. Li, D. Malkhi, Y. Xia, and R. Zhou, "Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing," arXiv (Cornell University), 2022.

[4] V. Gramoli, R. Guerraoui, A. Lebedev, and G. Voron, "Stabl: Blockchain fault tolerance," arXiv (Cornell University), 2024.

[5] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020. [Online]. Available: http://dx.doi.org/10.1109/comst.2020.2969706

[6] W. Zhong, C. Yang, W. Liang, J. Cai, L. Chen, J. Liao, and N. Xiong, "Byzantine fault-tolerant consensus algorithms: A survey," *Electronics*, vol. 12, no. 18, p. 3801, Sep. 2023. [Online]. Available: http://dx.doi.org/10.3390/electronics12183801

[7] G. Zhang, F. Pan, Y. Mao, S. Tijanic, M. Dang'ana, S. Motepalli, S. Zhang, and H.-A. Jacobsen, "Reaching consensus in the byzantine empire: A comprehensive review of bft consensus algorithms," *ACM Computing Surveys*, vol. 56, no. 5, pp. 1–41, Jan. 2024. [Online]. Available: http://dx.doi.org/10.1145/3636553

[8] B. Liang, L. Zheng, F. Yuan, Q. Wu, J. Deng, and J. Gao, "Performance evolution and optimization of the hotstuff consensus protocol: a systematic review," *Computing*, vol. 108, no. 7, Jun. 2026. [Online]. Available: http://dx.doi.org/10.1007/s00607-026-01695-0

[9] Q. Wang, J. Yu, S. Chen, and Y. Xiang, "Sok: Dag-based blockchain systems," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, Mar. 2023. [Online]. Available: http://dx.doi.org/10.1145/3576899

[10] M. Raikwar, N. Polyanskii, and S. Müller, "Sok: Dag-based consensus protocols," arXiv preprint, 2024.

[11] C. Berger, S. Schwarz-Rüsch, A. Vogel, K. Bleeke, L. Jehl, H. P. Reiser, and R. Kapitza, "Sok: Scalability techniques for bft consensus," arXiv (Cornell University), 2023.

[12] G. A. F. Rebello, G. F. Camilo, L. A. C. de Souza, M. Potop-Butucaru, M. D. de Amorim, M. E. M. Campista, and L. H. M. K. Costa, "A survey on blockchain scalability: From hardware to layer-two protocols," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 4, pp. 2411–2458, 2024. [Online]. Available: http://dx.doi.org/10.1109/comst.2024.3376252

[13] A. Hafid, A. S. Hafid, and M. Samih, "Scaling blockchains: A comprehensive survey," *IEEE Access*, vol. 8, pp. 125 244–125 262, 2020. [Online]. Available: http://dx.doi.org/10.1109/access.2020.3007251

[14] T. A. Alghamdi, R. Khalid, and N. Javaid, "A survey of blockchain based systems: Scalability issues and solutions, applications and future challenges," *IEEE Access*, vol. 12, pp. 79 626–79 651, 2024. [Online]. Available: http://dx.doi.org/10.1109/access.2024.3408868

[15] B. Bellaj, A. Ouaddah, E. Bertin, N. Crespi, and A. Mezrioui, "Drawing the boundaries between blockchain and blockchain-like systems: A comprehensive survey on distributed ledger technologies," *Proceedings of the IEEE*, vol. 112, no. 3, pp. 247–299, Mar. 2024. [Online]. Available: http://dx.doi.org/10.1109/jproc.2024.3386257

[16] G. Yu, X. Wang, K. Yu, W. Ni, J. A. Zhang, and R. P. Liu, "Survey: Sharding in blockchains," *IEEE Access*, vol. 8, pp. 14 155–14 181, 2020. [Online]. Available: http://dx.doi.org/10.1109/access.2020.2965147

[17] C. Yang, K.-W. Chin, J. Wang, X. Wang, Y. Liu, and Z. Zheng, "Scaling blockchains with error correction codes: A survey on coded blockchains," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–33, Jan. 2024. [Online]. Available: http://dx.doi.org/10.1145/3637224

[18] M. Boughdiri, T. Abdellatif, and C. Ghedira Guegan, "A systematic literature review on blockchain storage scalability," *IEEE Access*, vol. 13, pp. 102 194–102 219, 2025. [Online]. Available: http://dx.doi.org/10.1109/access.2025.3578451

[19] M. Dotan, Y.-A. Pignolet, S. Schmid, S. Tochner, and A. Zohar, "Survey on blockchain networking: Context, state-of-the-art, challenges," *ACM Computing Surveys*, vol. 54, no. 5, pp. 1–34, May 2021. [Online]. Available: http://dx.doi.org/10.1145/3453161

[20] A. Z. Chahoki, M. Herlihy, and M. Roveri, "Sok: Concurrency in blockchain – a systematic literature review and the unveiling of a misconception," arXiv (Cornell University), 2025.

[21] M. Aliyu, N. Kannengießer, and A. Sunyaev, "From concept to measurement: A survey of how the blockchain trilemma is analyzed," arXiv (Cornell University), 2025.

[22] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, "Performance evaluation of blockchain systems: A systematic survey," *IEEE Access*, vol. 8, pp. 126 927–126 950, 2020. [Online]. Available: http://dx.doi.org/10.1109/access.2020.3006078

[23] M. Esmaili and K. Christensen, "Performance modeling of public permissionless blockchains: A survey," *ACM Computing Surveys*, vol. 57, no. 7, pp. 1–35, Feb. 2025. [Online]. Available: http://dx.doi.org/10.1145/3715094

[24] V. Gramoli, R. Guerraoui, A. Lebedev, C. Natoli, and G. Voron, "Diablo: A benchmark suite for blockchains," in *Proceedings of the Eighteenth European Conference on Computer Systems*, ser. EuroSys '23. ACM, May 2023, pp. 540–556. [Online]. Available: http://dx.doi.org/10.1145/3552326.3567482

[25] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, Dec. 1990. [Online]. Available: http://dx.doi.org/10.1145/98163.98167

[26] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, Nov. 2002. [Online]. Available: http://dx.doi.org/10.1145/571637.571640

[27] Y. Lai, Y. Liu, H. Luo, G. Sun, C. Chi, and H. Yu, "Accelerating block and transaction propagation: A survey on broadcast protocols in blockchain networks," *IEEE Transactions on Network Science and Engineering*, pp. 1–24, 2025. [Online]. Available: http://dx.doi.org/10.1109/tnse.2025.3570870

[28] F. Gai, J. Niu, I. Beschastnikh, C. Feng, and S. Wang, "Scaling blockchain consensus via a robust shared mempool," arXiv (Cornell University), 2022.

[29] C. Stathakopoulou, M. Pavlovic, and M. Vukolić, "State machine replication scalability made simple," in *Proceedings of the Seventeenth European Conference on Computer Systems*, ser. EuroSys '22. ACM, Mar. 2022, pp. 17–33. [Online]. Available: http://dx.doi.org/10.1145/3492321.3519579

[30] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse, "Dispersedledger: High-throughput byzantine consensus on variable bandwidth networks," arXiv (Cornell University), 2021.

[31] R. Kerur, D. Malkhi, and M. K. Reiter, "eavid: Asynchronous verifiable information dispersal with post-dissemination pruning," arXiv (Cornell University), 2026.

[32] N. Polyanskii, "Vantage: Availability-graded broadcast for signature-free bft," arXiv (Cornell University), 2026.

[33] Y. Mao, S. Deb, S. B. Venkatakrishnan, S. Kannan, and K. Srinivasan, "Perigee: Efficient peer-to-peer network design for blockchains," arXiv (Cornell University), 2020.

[34] E. Rohrer and F. Tschorsch, "Kadcast-ng: A structured broadcast protocol for blockchain networks," *IEEE/ACM Transactions on Networking*, vol. 31, no. 6, pp. 3269–3283, Dec. 2023. [Online]. Available: http://dx.doi.org/10.1109/tnet.2023.3279897

[35] L. Yang, Y. Gilad, and M. Alizadeh, "Coded transaction broadcasting for high-throughput blockchains," arXiv (Cornell University), 2022.

[36] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Bft consensus with pipelined tree-based dissemination and aggregation," *ACM Transactions on Computer Systems*, vol. 44, no. 2, pp. 1–38, Apr. 2026. [Online]. Available: http://dx.doi.org/10.1145/3769423

[37] Y. Ye, R. Huang, G. Wang, M. Wan, Y. Duan, J. Zhang, and S. Liu, "Duet: Co-optimizing p2p message propagation and rotating-leader consensus," arXiv (Cornell University), 2026.

[38] A. Lebedev and V. Gramoli, "On the bandwidth consumption of blockchains," arXiv (Cornell University), 2026.

[39] J. Zhang, S. Yang, A. Sonnino, S. Loza, and A. Kate, "Lifefin: Escaping mempool explosions in dag-based bft," arXiv (Cornell University), 2025.

[40] L. Kiffer, L. Heimbach, D. Trautwein, Y. Vonlanthen, and O. Gasser, "Multiple sides of 36 coins: Measuring peer-to-peer infrastructure across cryptocurrencies," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 9, no. 3, pp. 1–32, Dec. 2025. [Online]. Available: http://dx.doi.org/10.1145/3771572

[41] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988. [Online]. Available: http://dx.doi.org/10.1145/42282.42283

[42] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985. [Online]. Available: http://dx.doi.org/10.1145/3149.214121

[43] J. Xu, C. Wang, and X. Jia, "A survey of blockchain

consensus protocols," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–35, Jul. 2023. [Online]. Available: http://dx.doi.org/10.1145/3579845

[44] T. Distler, "Byzantine fault-tolerant state-machine replication from a systems perspective," *ACM Computing Surveys*, vol. 54, no. 1, pp. 1–38, Feb. 2021. [Online]. Available: http://dx.doi.org/10.1145/3436728

[45] M. Raikwar, N. Polyanskii, and S. Müller, "Sok: Dag-based consensus protocols," in *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, May 2024, pp. 1–18. [Online]. Available: http://dx.doi.org/10.1109/icbc59979.2024.10634358

[46] M. Raikwar, T. Garrett, and R. Vitenberg, "Deconstruction of knowledge-building in dag-based dlts," *Distributed Ledger Technologies: Research and Practice*, vol. 5, no. 2, pp. 1–32, Jan. 2026. [Online]. Available: http://dx.doi.org/10.1145/3746647

[47] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, ser. PODC '19. ACM, Jul. 2019, pp. 347–356. [Online]. Available: http://dx.doi.org/10.1145/3293611.3331591

[48] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang, "Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback," arXiv preprint, 2021.

[49] D. Kang, S. Gupta, D. Malkhi, and M. Sadoghi, "Hotstuff-1: Linear consensus with one-phase speculation," *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, pp. 1–29, Jun. 2025. [Online]. Available: http://dx.doi.org/10.1145/3725308

[50] M. M. Jalalzai, J. Niu, C. Feng, and F. Gai, "Fast-hotstuff: A fast and robust bft protocol for blockchains," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 4, pp. 2478–2493, Jul. 2024. [Online]. Available: http://dx.doi.org/10.1109/tdsc.2023.3308848

[51] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2020, pp. 106–118. [Online]. Available: http://dx.doi.org/10.1109/sp40000.2020.00044

[52] M. M. Jalalzai, K. Babel, J. Komatovic, T. Klenze, S. Das, F. Elsheimy, M. Setrin, J. Bergschneider, and B. Gilkalaye, "Monadbft: Fast, responsive, fork-resistant streamlined consensus," arXiv (Cornell University), 2025.

[53] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: Dag bft protocols made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. ACM, Nov. 2022, pp. 2705–2718. [Online]. Available: http://dx.doi.org/10.1145/3548606.3559361

[54] K. Babel, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, A. Koshy, A. Sonnino, and M. Tian, "Bft consensus: From academic paper to mainnet (invited talk)," arXiv (Cornell University), 2023.

[55] A. Spiegelman, B. Arun, R. Gelashvili, and Z. Li, "Shoal: Improving dag-bft latency and robustness," arXiv (Cornell University), 2023.

[56] B. Arun, Z. Li, F. Suri-Payer, S. Das, and A. Spiegelman, "Shoal++: High throughput dag bft can be fast!" arXiv (Cornell University), 2024.

[57] X. Dai, W. Li, G. Wang, J. Xiao, H. Chen, S. Li, A. Y. Zomaya, and H. Jin, "Remora: A low-latency dag-based bft through optimistic paths," *IEEE Transactions on Computers*, vol. 74, no. 1, pp. 57–70, Jan. 2025. [Online]. Available: http://dx.doi.org/10.1109/tc.2024.3461309

[58] X. Dai, Z. Zhang, Z. Guo, C. Ding, J. Xiao, X. Xie, R. Hao, and H. Jin, "Wahoo: A dag-based bft consensus with low latency and low communication overhead," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 7508–7522, 2024. [Online]. Available: http://dx.doi.org/10.1109/tifs.2024.3409082

[59] X. Dai, C. Ding, W. Li, J. Xiao, B. Zhang, C. Yu, A. Y. Zomaya, and H. Jin, "Falcon: Advancing asynchronous bft consensus for lower latency and enhanced throughput," *Proceedings of the VLDB Endowment*, vol. 18, no. 7, pp. 2136–2148, Mar. 2025. [Online]. Available: http://dx.doi.org/10.14778/3734839.3734850

[60] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency," arXiv (Cornell University), 2022.

[61] L. Freitas, S. Motepalli, M. Pavlovic, and B. Livshits, "Rorqual: Speeding up narwhal with tees," arXiv (Cornell University), 2024.

[62] K. Hwerbi, I. Amdouni, C. Adjih, L. A. Saidane, and A. Laouiti, "Delay analysis of bft consensus : Case study of narwhal and bullshark protocols," *Computer Communications*, vol. 242, p. 108278, Oct. 2025. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2025.108278

[63] D. Malkhi, C. Stathakopoulou, and M. Yin, "Bbca-chain: Low latency, high throughput bft consensus on a dag," arXiv (Cornell University), 2023.

[64] P. Jovanovic, L. K. Kogias, B. Kumara, A. Sonnino, P. Tennage, and I. Zablotchi, "Mahi-mahi: Low-latency asynchronous bft dag-based consensus," arXiv (Cornell University), 2024.

[65] Z. D. Angeli, "Sharpedo: Dual-mode uncertified dag-based consensus protocol," arXiv (Cornell University), 2026.

[66] I. Keidar, O. Naor, O. Poupko, and E. Shapiro, "Cordial miners: Fast and efficient consensus for every eventuality," arXiv (Cornell University), 2022.

[67] Y. Zhou, J. Xiao, X. Dai, and H. Jin, "Plaindag: A low-latency asynchronous dag bft protocol with best-effort broadcast," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 9792–9805, 2025. [Online]. Available: http://dx.doi.org/10.1109/tifs.2025.3607262

[68] P. Ren, D. Hai, N. Sohrabi, and Z. Tari, "Multi-round visibility: A post-consensus ordering layer for dag-based bft," arXiv (Cornell University), 2026.

[69] N. Giridharan, M. Nguyen, I. Abraham, A. Clement, N. Crooks, and P. Sutra, "Optimality and trade-offs in fast bft smr (extended version)," arXiv (Cornell University), 2026.

[70] B. K. Chou, A. Lewis-Pye, and P. O'Grady, "Minimmit: Fast finality with even faster blocks," arXiv (Cornell University), 2025.

[71] P. V. Vos, "Odontoceti: Ultra-fast dag consensus with two round commitment," arXiv (Cornell University), 2025.

[72] P. V. Vos, A. Sonnino, G. Tsimos, P. Jovanovic, and L. Kokoris-Kogias, "Bluebottle: Fast and robust blockchains through subsystem specialization," arXiv (Cornell University), 2025.

[73] A. Ranchal-Pedrosa, D. Kang, N. Giridharan, D. Malkhi, M. Sadoghi, and B. Marsh, "Hermes: Low tail-latency via prefix consensus," arXiv (Cornell University), 2026.

[74] R. Ladelsky and R. Friedman, "Finwhale: An optimally resilient two-round terminating dag protocol," arXiv (Cornell University), 2026.

[75] N. Giridharan, F. Suri-Payer, I. Abraham, L. Alvisi, and N. Crooks, "Autobahn: Seamless high speed bft," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, ser. SOSP '24. ACM, Nov. 2024, pp. 1–23. [Online]. Available: http://dx.doi.org/10.1145/3694715.3695942

[76] A. Lewis-Pye and E. Shapiro, "Morpheus consensus: Excelling on trails and autobahns," arXiv (Cornell University), 2025.

[77] N. Shrestha, R. Shrothrium, A. Kate, and K. Nayak, "Sailfish: Towards improving the latency of dag-based bft," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2025, pp. 1928–1946. [Online]. Available: http://dx.doi.org/10.1109/sp61157.2025.00021

[78] Q. Yu, G. Losa, N. Shrestha, and X. Wang, "Angelfish: Leader, dag, or anywhere in between," arXiv preprint, 2025.

[79] Y. Vonlanthen, J. Sliwinski, M. Albarello, and R. Wattenhofer, "Banyan: Fast rotating leader bft," in *Proceedings of the 25th International Middleware Conference*, ser. Middleware '24. ACM, Dec. 2024, pp. 494–507. [Online]. Available: http://dx.doi.org/10.1145/3652892.3700788

[80] G. Danezis, Z. D. Angeli, A. I. Vitanov, P. Jovanovic, L. Kokoris-Kogias, A. Sonnino, P. Tennage, and I. Zablotchi, "Barnacle: Adaptive multi-leader scheduling for dag-based consensus," arXiv (Cornell University), 2026.

[81] M. Baudet, G. Danezis, and A. Sonnino, "Fastpay: High-performance byzantine fault tolerant settlement," arXiv (Cornell University), 2020.

[82] M. Baudet, A. Sonnino, M. Kelkar, and G. Danezis, "Zef: Low-latency, scalable, private payments," arXiv (Cornell University), 2022.

[83] S. Blackshear, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, X. Li, M. Logan, A. S. Menon, T. Nowacki, A. Sonnino, B. Williams, and L. Zhang, "Sui lutris: A blockchain combining broadcast and consensus," arXiv (Cornell University), 2023.

[84] A. Buchwald, S. Buttolph, A. Lewis-Pye, P. O'Grady, and K. Sekniqi, "Frosty: Bringing strong liveness guarantees to the snow family of consensus protocols," arXiv (Cornell University), 2024.

[85] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and probabilistic leaderless bft consensus through metastability," arXiv (Cornell University), 2019.

[86] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. ACM, Oct. 2017, pp. 51–68. [Online]. Available: http://dx.doi.org/10.1145/3132747.3132757

[87] M. Wang, J. Niu, F. Gai, M. M. Jalalzai, Y. Zhang, and C. Feng, "Chained hotstuff under performance attack," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 4, pp. 3737–3750, Jul. 2025. [Online]. Available: http://dx.doi.org/10.1109/tdsc.2025.3540980

[88] H. Oh and C. Park, "Pipelining and overlapping: Techniques to improve both throughput and latency in bft consensus blockchain," *IEEE Access*, vol. 12, pp. 66 408–66 418, 2024. [Online]. Available: http://dx.doi.org/10.1109/access.2024.3398787

[89] S. Ling, Z. Liu, Q. Li, X. Du, J. Chen, and K. Xu, "Stable byzantine fault tolerance in wide area networks with unreliable links," *IEEE/ACM Transactions on Networking*, vol. 32, no. 6, pp. 5310–5325, Dec. 2024. [Online]. Available: http://dx.doi.org/10.1109/tnet.2024.3461872

[90] C. Berger, L. Rodrigues, H. P. Reiser, V. Cogo, and A. Bessani, "Chasing lightspeed consensus: Fast wide-area byzantine replication with mercury," in *Proceedings of the 25th International Middleware Conference*, ser. Middleware '24. ACM, Dec. 2024, pp. 158–171. [Online]. Available: http://dx.doi.org/10.1145/3652892.3700756

[91] C. Berger, L. Rodrigues, H. P. Reiser, V. V. Cogo, and A. Bessani, "Chasing the speed of light: Low-latency planetary-scale adaptive byzantine consensus," arXiv (Cornell University), 2023.

[92] S. Motepalli, N. Garg, G. Zhang, and H.-A. Jacobsen, "Gpos: Geospatially-aware proof of stake," *ACM Transactions on the Web*, vol. 20,

no. 1, pp. 1–20, Feb. 2026. [Online]. Available: http://dx.doi.org/10.1145/3774629

[93] Z. Xiang, S. Das, Z. Li, Z. Ma, and A. Spiegelman, "The latency price of threshold cryptosystem in blockchains," arXiv (Cornell University), 2024.

[94] S. Bano, A. Sonnino, A. Chursin, D. Perelman, Z. Li, A. Ching, and D. Malkhi, "Twins: Bft systems made robust," arXiv (Cornell University), 2020.

[95] L. Qiu, J. Xiao, J.-Y. Shin, and Z. Shao, "Lido-dag: A framework for verifying safety and liveness of dag-based consensus protocols," *Proceedings of the ACM on Programming Languages*, vol. 9, no. PLDI, pp. 1392–1416, Jun. 2025. [Online]. Available: http://dx.doi.org/10.1145/3729306

[96] A. Chursin, "Adelie: Detection and prevention of byzantine behaviour in dag-based consensus protocols," arXiv (Cornell University), 2024.

[97] I. Mirzaei, H. Patel, C. Wu, and M. J. Amiri, "Competition, collusion, and corruption: The spectrum of mev attacks on dag-based bft consensus protocols," arXiv (Cornell University), 2026.

[98] I. Mirzaei and M. J. Amiri, "Fair on the surface: Transaction-ordering bias and mev in mysticeti dag-based bft protocol," arXiv (Cornell University), 2026.

[99] E. Mahe and S. Tucci-Piergiovanni, "Order fairness evaluation of dag-based ledgers," arXiv (Cornell University), 2025.

[100] G. J. Valentini, C. Mascia, and A. Bracciali, "Approaches to mev in dag-based dlts," *Journal of Mathematical Cryptology*, vol. 20, no. 1, Jan. 2026. [Online]. Available: http://dx.doi.org/10.1515/jmc-2025-0048

[101] D. Kang, J. Chen, T. T. A. Dinh, and M. Sadoghi, "Fairdag: Consensus fairness over multi-proposer causal design," *Proceedings of the VLDB Endowment*, vol. 19, no. 2, pp. 265–278, Oct. 2025. [Online]. Available: http://dx.doi.org/10.14778/3773749.3773763

[102] C. Cachin, J. Mićić, N. Steinhauer, and L. Zanolini, "Quick order fairness," arXiv preprint, 2021.

[103] M. Putnik and J. Decouchant, "Herring: Parallel batch-order-fairness on dag-based blockchain consensus," arXiv (Cornell University), 2026.

[104] G. Segalini, Y. Çolakoğlu, M. Putnik, and J. Decouchant, "Tilikum: Transaction fair ordering on a dag without weak edges," arXiv (Cornell University), 2026.

[105] D. Malkhi and P. Szałachowski, "Maximal extractable value (mev) protection on a dag," arXiv (Cornell University), 2022.

[106] W. Wang, A. Saraf, L. Heimbach, K. Babel, and F. Zhang, "Blockspace under pressure: An analysis of spam mev on high-throughput blockchains," arXiv (Cornell University), 2026.

[107] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, Jul. 2018. [Online]. Available: http://dx.doi.org/10.1109/tkde.2017.2781227

[108] C. Berger, S. B. Toumia, and H. P. Reiser, "Exploring scalability of bft blockchain protocols through network simulations," *Formal Aspects of Computing*, vol. 36, no. 4, pp. 1–29, Dec. 2024. [Online]. Available: http://dx.doi.org/10.1145/3689343

[109] P. S. Anjana, S. Ravi, and M. Herlihy, "Blockchain transaction conflicts: A historical perspective," arXiv (Cornell University), 2025.

[110] X. Zheng, Z. Wan, D. Lo, D. Xie, and X. Yang, "Why does my transaction fail? a first look at failed transactions on the solana blockchain," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 1489–1512, Jun. 2025. [Online]. Available: http://dx.doi.org/10.1145/3728943

[111] L. Heimbach, K. Babel, and J. Milionis, "Evm workloads in the wild: Evidence for multi-dimensional gas metering, state growth, delayed execution, and parallelism," arXiv (Cornell University), 2026.

[112] S. Wadhwa, A. Yaish, F. Zhang, and K. Nayak, "Perils of parallelism: Transaction fee mechanisms under execution uncertainty," arXiv (Cornell University), 2026.

[113] B. Acilan, A. Constantinescu, L. Heimbach, and R. Wattenhofer, "Transaction fee market design for parallel execution," *DROPS (Schloss Dagstuhl – Leibniz Center for Informatics)*, 2025.

[114] S. Smolka, J.-R. Giesen, P. Winkler, O. Draissi, L. Davi, G. Karame, and K. Pohl, "Fuzz on the beach: Fuzzing solana smart contracts," arXiv (Cornell University), 2023.

[115] R. Gelashvili, A. Spiegelman, Z. Xiang, G. Danezis, Z. Li, D. Malkhi, Y. Xia, and R. Zhou, "Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing," in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '23. ACM, Feb. 2023, pp. 232–244. [Online]. Available: http://dx.doi.org/10.1145/3572848.3577524

[116] F. Ezard, C. U. Ileri, and J. Decouchant, "Nemo: Faster parallel execution for highly contended blockchain workloads (full version)," arXiv (Cornell University), 2025.

[117] P. S. Anjana, M. Amini, R. Kapoor, R. Parmar, and R. Ramesh, "Efficient parallel execution of blockchain transactions leveraging conflict specifications," arXiv preprint, 2025.

[118] H. Lin, H. Feng, Y. Zhou, and L. Wu, "Parallelevm: Operation-level concurrent transaction execution for evm-compatible blockchains," in *Proceedings of the Twentieth European Conference on Computer Systems*, ser. EuroSys '25. ACM, Mar. 2025, pp. 211–225. [Online]. Available: http://dx.doi.org/10.1145/3689031.3696063

[119] Z. Liu, M. Unal, M. J. Parkinson, and M. Kogias, "Doradd: Deterministic parallel execution in the era

of microsecond-scale computing," in *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '25. ACM, Feb. 2025, pp. 282–296. [Online]. Available: http://dx.doi.org/10.1145/3710848.3710872

[120] S. Blackshear, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, X. Li, M. Logan, A. Menon, T. Nowacki, A. Sonnino, B. Williams, and L. Zhang, "<scp>sui lutris:</scp> a blockchain combining broadcast and consensus," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. ACM, Dec. 2024, pp. 2606–2620. [Online]. Available: http://dx.doi.org/10.1145/3658644.3670286

[121] R. Overko, "A study on shared objects in sui smart contracts," in *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, May 2024, pp. 1–7. [Online]. Available: http://dx.doi.org/10.1109/icbc59979.2024.10634357

[122] Z. Hong, S. Guo, E. Zhou, J. Zhang, W. Chen, J. Liang, J. Zhang, and A. Y. Zomaya, "Prophet: Conflict-free sharding blockchain via byzantine-tolerant deterministic ordering," arXiv (Cornell University), 2023.

[123] Z. Chen, T. Yang, Y. Zheng, Z. Zhang, C. Jin, and A. Zhou, "Spectrum: Speedy and strictly-deterministic smart contract transactions for blockchain ledgers," *Proceedings of the VLDB Endowment*, vol. 17, no. 10, pp. 2541–2554, Jun. 2024. [Online]. Available: http://dx.doi.org/10.14778/3675034.3675045

[124] Z. Liu, A. Sonnino, I. Zablotchi, E. K. Kogias, and M. Kogias, "Remora: Scale-out deterministic execution for smart contracts," *Proceedings of the VLDB Endowment*, vol. 19, no. 11, pp. 3076–3090, Jul. 2026. [Online]. Available: http://dx.doi.org/10.14778/3836663.3836674

[125] X. Wang, X. Li, Y. Peng, and H. Huang, "Comprehensive evaluation and opportunity discovery for deterministic concurrency control," *ACM Transactions on Architecture and Code Optimization*, vol. 22, no. 2, pp. 1–24, Jun. 2025. [Online]. Available: http://dx.doi.org/10.1145/3715126

[126] X. Wang, Y. Peng, H. Huang, and X. Li, "Dodo: A scalable optimistic deterministic concurrency control protocol," *Future Generation Computer Systems*, vol. 159, pp. 15–26, Oct. 2024. [Online]. Available: http://dx.doi.org/10.1016/j.future.2024.05.004

[127] X. Wang, Y. Peng, and H. Huang, "Gria: an efficient deterministic concurrency control protocol," *Frontiers of Computer Science*, vol. 18, no. 4, Dec. 2023. [Online]. Available: http://dx.doi.org/10.1007/s11704-023-2605-z

[128] P. Ruan, D. Loghin, Q.-T. Ta, M. Zhang, G. Chen, and B. C. Ooi, "A transactional perspective on execute-order-validate blockchains," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD/PODS '20. ACM, May 2020, pp. 543–557. [Online]. Available: http://dx.doi.org/10.1145/3318464.3389693

[129] Y. Hay and R. Friedman, "Batch-schedule-execute: On optimizing concurrent deterministic scheduling for blockchains (extended version)," arXiv (Cornell University), 2024.

[130] R. Neiheiser and E. Kokoris-Kogias, "Anthemius: Efficient & modular block assembly for concurrent execution," arXiv (Cornell University), 2025.

[131] R. Zarick, I. Zhang, D. R. Wong, T. Kim, B. Pellegrino, M. Li, and K. Wong, "Fafo: Over 1 million tps on a single node running evm while still merkleizing every block," arXiv (Cornell University), 2025.

[132] A. Karmegam, L. Kiffer, and A. F. Anta, "Exploiting multi-core parallelism in blockchain validation and construction," *DROPS (Schloss Dagstuhl – Leibniz Center for Informatics)*, 2026.

[133] Q. Kniep, L. Kokoris-Kogias, A. Sonnino, I. Zablotchi, and N. Zhang, "Pilotfish: Distributed execution for scalable blockchains," arXiv (Cornell University), 2024.

[134] J. Chen, A. Sonnino, L. Kokoris-Kogias, and M. Sadoghi, "Thunderbolt: Concurrent smart contract execution with non-blocking reconfiguration for sharded dags," arXiv (Cornell University), 2024.

[135] G. Mitenkov, I. Kabiljo, Z. Li, A. Spiegelman, S. Vusirikala, Z. Xiang, A. Zlateski, N. P. Lopes, and R. Gelashvili, "Deferred objects to enhance smart contract programming with optimistic parallel execution," arXiv (Cornell University), 2024.

[136] M. Patrignani and S. Blackshear, "Robust safety for move," arXiv preprint, 2021.

[137] D. L. Dill, W. Grieskamp, J. Park, S. Qadeer, M. Xu, and E. Zhong, "Fast and reliable formal verification of smart contracts with the move prover," arXiv (Cornell University), 2021.

[138] H. Wang, M. Pan, and J. Wang, "Crystality: A programming model for smart contracts on parallel evms," in *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '25. ACM, Feb. 2025, pp. 412–425. [Online]. Available: http://dx.doi.org/10.1145/3710848.3710879

[139] Z. Xu, H. Wang, and M. Sun, "Operational semantics for crystality: A smart contract language for parallel evms," arXiv (Cornell University), 2025.

[140] G. Pîrlea, A. Kumar, and I. Sergey, "Practical smart contract sharding with ownership and commutativity analysis," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, ser. PLDI '21. ACM, Jun. 2021, pp. 1327–1341. [Online]. Available: http://dx.doi.org/10.1145/3453483.3454112

[141] M. Vaccargiu, A. Pinna, M. I. Lunesu, and G. Destefanis, "Comparing smart contract paradigms: A preliminary study of security and developer experience," arXiv (Cornell University), 2026.

[142] V. Saraph and M. Herlihy, "An empirical study of speculative concurrency in ethereum smart contracts," arXiv (Cornell University), 2019.

[143] D. Reijsbergen and T. T. A. Dinh, "On exploiting transaction concurrency to speed up blockchains," arXiv preprint, 2020.

[144] X. Hu, B. Burgstaller, and B. Scholz, "Evmtracer: Dynamic analysis of the parallelization and redundancy potential in the ethereum virtual machine," *IEEE Access*, vol. 11, pp. 47 159–47 178, 2023. [Online]. Available: http://dx.doi.org/10.1109/access.2023.3267277

[145] M. Pigaglio, M. Król, and E. Riviére, "Exploring locality in ethereum transactions," in *2023 5th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, Oct. 2023, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/brains59668.2023.10317054

[146] M. Bartoletti, L. Galletta, and M. Murgia, "A theory of transaction parallelism in blockchains," *Logical Methods in Computer Science*, vol. Volume 17, Issue 4, 2021.

[147] P. S. Anjana, S. Kumari, S. Peri, S. Rathor, and A. Somani, "Optsmart: A space efficient optimistic concurrent execution of smart contracts," arXiv (Cornell University), 2021.

[148] D. Xian and X. Wei, "Sc-chef: Turboboosting smart contract concurrent execution for high contention workloads via chopping transactions," *IEEE Transactions on Reliability*, vol. 73, no. 1, pp. 216–229, Mar. 2024. [Online]. Available: http://dx.doi.org/10.1109/tr.2023.3296278

[149] J. Liu, P. Li, R. Cheng, N. Asokan, and D. Song, "Parallel and asynchronous smart contract execution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 5, pp. 1097–1108, May 2022. [Online]. Available: http://dx.doi.org/10.1109/tpds.2021.3095234

[150] Y. Li, H. Liu, J. Gao, J. Zhang, Z. Guan, and Z. Chen, "Accelerating block lifecycle on blockchain via hardware transactional memory," *Journal of Parallel and Distributed Computing*, vol. 184, p. 104779, Feb. 2024. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2023.104779

[151] V. Capocasale, F. Pedone, and G. Perboli, "Parallel transaction execution in blockchain and the ambiguous state representation problem," in *2024 19th European Dependable Computing Conference (EDCC)*. IEEE, Apr. 2024, pp. 131–138. [Online]. Available: http://dx.doi.org/10.1109/edcc61798.2024.00035

[152] S. Müller, A. Penzkofer, N. Polyanskii, J. Theis, W. Sanders, and H. Moog, "Reality-based utxo ledger," *Distributed Ledger Technologies: Research and Practice*, vol. 2, no. 3, pp. 1–33, Sep. 2023. [Online]. Available: http://dx.doi.org/10.1145/3616022

[153] S. Ponnapalli, A. Shah, A. Tai, S. Banerjee, V. Chidambaram, D. Malkhi, and M. Wei, "Rainblock: Faster transaction processing in public blockchains," arXiv (Cornell University), p. 333, 2019.

[154] C. Zhang, C. Xu, H. Hu, and J. Xu, "Cole: A column-based learned storage for blockchain systems," arXiv (Cornell University), 2023.

[155] C. Li, S. M. Beillahi, G. Yang, M. Wu, W. Xu, and F. Long, "Lvmt: An efficient authenticated storage for blockchain," *ACM Transactions on Storage*, vol. 20, no. 3, pp. 1–34, Jun. 2024. [Online]. Available: http://dx.doi.org/10.1145/3664818

[156] J. A. Choi, S. M. Beillahi, P. Li, A. Veneris, and F. Long, "Lmpts: Eliminating storage bottlenecks for processing blockchain transactions," in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, May 2022, pp. 1–9. [Online]. Available: http://dx.doi.org/10.1109/icbc54727.2022.9805484

[157] L. Crisci, L. Schuler, H. Jordan, and B. Scholz, "Sonicdb s6: A storage-efficient verkle trie for high-throughput blockchains," arXiv (Cornell University), 2026.

[158] I. Zhang, R. Zarick, D. R. Wong, T. Kim, B. Pellegrino, M. Li, and K. Wong, "Qmdb: Quick merkle database," arXiv (Cornell University), 2025.

[159] B. Kauer, A. Petrosyan, and B. Livshits, "Aldbaran: Towards blazingly fast state commitments for blockchains," arXiv preprint, 2025.

[160] N. Sheng, T. Zhou, H. Zhao, X. Li, and J. Xu, "Swmt: A sliding window merkle tree with delayed writes for scalable blockchain state management," *Proceedings of the ACM on Management of Data*, vol. 4, no. 1, pp. 1–25, Apr. 2026. [Online]. Available: http://dx.doi.org/10.1145/3786701

[161] Q. Que, X. Qi, Z. Zhang, Y. Yang, C. Jin, and A. Zhou, "Hyra: Scalable byzantine-resilient state storage engine with hierarchical erasure-coding," *Proceedings of the ACM on Management of Data*, vol. 4, no. 1, pp. 1–27, Apr. 2026. [Online]. Available: http://dx.doi.org/10.1145/3786658

[162] Z. Shangguan, A. Yaish, and D. Malkhi, "Authenticated data structures for dynamic workloads," arXiv (Cornell University), 2026.

[163] W. Wang, Y. Lu, C. Papamanthou, and F. Zhang, "The locality of memory checking," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. ACM, Nov. 2023, pp. 1820–1834. [Online]. Available: http://dx.doi.org/10.1145/3576915.3623195

[164] S. Nathan, C. Govindarajan, A. Saraf, M. Sethi, and P. Jayachandran, "Blockchain meets database: design and implementation of a blockchain relational database," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1539–1552, Jul. 2019. [Online]. Available: http://dx.doi.org/10.14778/3342263.3342632

[165] Z. Chen, B. Li, X. Cai, Z. Jia, L. Ju, Z. Shao,

and Z. Shen, "Chainkv: A semantics-aware key-value store for ethereum system," *Proceedings of the ACM on Management of Data*, vol. 1, no. 4, pp. 1–23, Dec. 2023. [Online]. Available: http://dx.doi.org/10.1145/3626713

[166] J. Liang, W. Chen, Z. Hong, H. Zhu, W. Qiu, and Z. Zheng, "Moltdb: Accelerating blockchain via ancient state segregation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 12, pp. 2545–2558, Dec. 2024. [Online]. Available: http://dx.doi.org/10.1109/tpds.2024.3467927

[167] S. Wang, T. T. A. Dinh, Q. Lin, Z. Xie, M. Zhang, Q. Cai, G. Chen, W. Fu, B. C. Ooi, and P. Ruan, "Forkbase: An efficient storage engine for blockchain and forkable applications," arXiv (Cornell University), 2018.

[168] Z. Ge, D. Loghin, B. C. Ooi, P. Ruan, and T. Wang, "Hybrid blockchain database systems: design and performance," *Proceedings of the VLDB Endowment*, vol. 15, no. 5, pp. 1092–1104, Jan. 2022. [Online]. Available: http://dx.doi.org/10.14778/3510397.3510406

[169] M. Kim and S.-M. Moon, "Features of various storage engines and analysis of their adoption in blockchain database," *KIISE Transactions on Computing Practices*, vol. 31, no. 2, pp. 105–110, Feb. 2025. [Online]. Available: http://dx.doi.org/10.5626/ktcp.2025.31.2.105

[170] Q. Wei, B. Li, W. Chang, Z. Jia, Z. Shen, and Z. Shao, "A survey of blockchain data management systems," arXiv (Cornell University), 2021.

[171] Z. He, Z. Li, A. Qiao, X. Luo, X. Zhang, T. Chen, S. Song, D. Liu, and W. Niu, "Nurgle: Exacerbating resource consumption in blockchain state storage via mpt manipulation," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2024, pp. 2180–2197. [Online]. Available: http://dx.doi.org/10.1109/sp54263.2024.00125

[172] S. R. B, "secureprune:secure block pruning in utxo based blockchains using accumulators," arXiv (Cornell University), 2020.

[173] P. Huang, X. Ren, T. Huang, A. S. Voundi Koe, D. S. Wong, and H. Jiang, "Snapshotprune: A novel bitcoin-based protocol toward efficient pruning and fast node bootstrapping," *Tsinghua Science and Technology*, vol. 29, no. 4, pp. 1037–1052, Aug. 2024. [Online]. Available: http://dx.doi.org/10.26599/tst.2023.9010014

[174] M. Xu, H. Guo, C. Ye, C. Liu, D. Yu, and X. Cheng, "Ec-chain: Cost-effective storage solution for permissionless blockchains," arXiv (Cornell University), 2024.

[175] Y. I. Alzoubi and A. Mishra, "Techniques to alleviate blockchain bloat: Potentials, challenges, and recommendations," *Computers and Electrical Engineering*, vol. 116, p. 109216, May 2024. [Online]. Available: http://dx.doi.org/10.1016/j.compeleceng.2024.109216

[176] S. Johnson, D. Hyland-Wood, A. L. Madsen, and K. Mengersen, "Stateful to stateless: Modelling stateless ethereum," *Electronic Proceedings in Theoretical Computer Science*, vol. 355, pp. 27–39, Mar. 2022. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.355.3

[177] J. Oberst, "Towards stateless clients in ethereum: Benchmarking verkle trees and binary merkle trees with snarks," arXiv preprint, 2025.

[178] J.-Y. Kim, J.-M. Lee, Y.-J. Koo, S.-H. Park, and S.-M. Moon, "Ethanos: Lightweight bootstrapping for ethereum," arXiv (Cornell University), 2019.

[179] T. Diamandis, A. Evans, T. Chitra, and G. Angeris, "Dynamic pricing for non-fungible resources: Designing multidimensional blockchain fee markets," arXiv preprint, 2022.

[180] G. Angeris, T. Diamandis, and C. Moallemi, "Multidimensional blockchain fees are (essentially) optimal," *DROPS (Schloss Dagstuhl – Leibniz Center for Informatics)*, 2025.

[181] M. Bahrani and N. Durvasula, "Resonance: Transaction fees for heterogeneous computation," arXiv (Cornell University), 2024.

[182] T. Roughgarden, "Transaction fee mechanism design," arXiv preprint, 2021.

[183] ——, "Transaction fee mechanism design for the ethereum blockchain: An economic analysis of eip-1559," RePEc: Research Papers in Economics, 2020.

[184] S. Leonardos, B. Monnot, D. Reijsbergen, S. Skoulakis, and G. Piliouras, "Dynamical analysis of the eip-1559 ethereum fee market," arXiv preprint, 2021.

[185] S. Azouvi, G. Goren, L. Heimbach, and A. Hicks, "Base fee manipulation in ethereum's eip-1559 transaction fee mechanism," arXiv (Cornell University), 2023.

[186] Y. Liu, Y. Lu, K. Nayak, F. Zhang, and L. Zhang, "Empirical analysis of eip-1559: Transaction fees, waiting time, and consensus security," arXiv preprint, 2022.

[187] M. V. X. Ferreira, D. J. Moroz, D. C. Parkes, and M. Stern, "Dynamic posted-price mechanisms for the blockchain transaction fee market (invited talk)," arXiv (Cornell University), 2022.

[188] A. Ganesh, J. D. Hartline, A. R. Sinha, and M. von-Allmen, "Fundamental limits of throughput and availability: Applications to prophet inequalities & transaction fee mechanism design," arXiv (Cornell University), 2024.

[189] P. Zhivkov and E. Chen, "Dynamic fee markets at sub-second timescales: Adapting eip-1559 for high-throughput blockchains," *Mathematics*, vol. 14, no. 7, p. 1232, Apr. 2026. [Online]. Available: http://dx.doi.org/10.3390/math14071232

[190] A. Ndiaye, "Parallel execution fee mechanisms," arXiv (Cornell University), 2024.

[191] M. Bahrani, P. Garimidi, and T. Roughgarden, "Transaction fee mechanism design with active block

producers," RePEc: Research Papers in Economics, 2023.

[192] D. Landis and N. I. Schwartzbach, "Stackelberg attacks on auctions and blockchain transaction fee mechanisms," arXiv (Cornell University), 2023.

[193] Y. Gafni and A. Yaish, "Discrete & bayesian transaction fee mechanisms," arXiv (Cornell University), 2022.

[194] K. Wu, E. Shi, and H. Chung, "Maximizing miner revenue in transaction fee mechanism design," arXiv (Cornell University), 2023.

[195] X. Chen, D. Simchi-Levi, Z. Zhao, and Y. Zhou, "Bayesian mechanism design for blockchain transaction fee allocation," *Operations Research*, vol. 73, no. 4, pp. 1944–1964, Jul. 2025. [Online]. Available: http://dx.doi.org/10.1287/opre.2024.0865

[196] K. Li, G. Qi, G. Shang, W. Deng, M. Xu, and X. Cheng, "A treatment of eip-1559: Enhancing transaction fee mechanism through nth-price auction," arXiv (Cornell University), 2024.

[197] S. Damle, M. Padala, and S. Gujar, "Designing redistribution mechanisms for reducing transaction fees in blockchains," arXiv (Cornell University), 2024.

[198] A. Kiayias, H. Κουτσουπιάς, P. Lazos, and G. Panagiotakos, "Tiered mechanisms for blockchain transaction fees," arXiv (Cornell University), 2023.

[199] ——, "Blockchain space tokenization," *Edinburgh Research Explorer (University of Edinburgh)*, 2024.

[200] R. Chotkan, B. Nasrulin, J. Decouchant, and J. Pouwelse, "Starvespam: Mitigating spam with local reputation in permissionless blockchains," arXiv (Cornell University), 2025.

[201] A.-P. Stouka, J. Ma, and T. Thiery, "Multiple proposer transaction fee mechanism design: Robust incentives against censorship and bribery," arXiv (Cornell University), 2025.

[202] A. Saraf, G. Kaklamanis, S. Wadhwa, and F. Elsheimy, "Price of censorship: Censorship resistance and throughput under rational concurrent proposers," arXiv (Cornell University), 2026.

[203] A. Tonkikh, B. Arun, Z. Xiang, Z. Li, and A. Spiegelman, "Raptr: Prefix consensus for robust high-performance bft," arXiv (Cornell University), 2025.

[204] W. Sun, Z. Xu, W. Ni, and L. Chen, "Intime: Towards performance predictability in byzantine fault tolerant proof-of-stake consensus," *Proceedings of the ACM on Management of Data*, vol. 3, no. 1, pp. 1–27, Feb. 2025. [Online]. Available: http://dx.doi.org/10.1145/3709740

[205] F. Gai, J. Niu, I. Beschastnikh, C. Feng, and S. Wang, "Scaling blockchain consensus via a robust shared mempool," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, Apr. 2023, pp. 530–543. [Online]. Available: http://dx.doi.org/10.1109/icde55515.2023.00047

[206] D. Malkhi and P. Szalachowski, "Maximal extractable value (mev) protection on a dag," *DROPS (Schloss Dagstuhl – Leibniz Center for Informatics)*, 2023.

[207] S. Landers and B. Marsh, "Mev in multiple concurrent proposer blockchains," arXiv (Cornell University), 2025.

[208] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2020, pp. 910–927. [Online]. Available: http://dx.doi.org/10.1109/sp40000.2020.00040

[209] B. Öz, F. Rezabek, J. Gebele, F. Hoops, and F. Matthes, "A study of mev extraction techniques on a first-come-first-served blockchain," arXiv (Cornell University), 2023.

[210] S. Chu and S. Wang, "The curses of blockchain decentralization," arXiv (Cornell University), 2018.

[211] V. P. Di Perna, V. Schiavoni, M. Matos, F. Fabris, and M. Bernardo, "The impact of network topology on performance metrics and energy consumption for blockchains: Towards repeatable benchmarking," *Distributed Ledger Technologies: Research and Practice*, Jul. 2026. [Online]. Available: http://dx.doi.org/10.1145/3828757

[212] C. Ovezik, D. Karakostas, M. Milad, A. Kiayias, and D. Woods, "Sok: Measuring blockchain decentralization," arXiv (Cornell University), 2025.

[213] Q. Lin, C. Li, X. Zhao, and X. Chen, "Measuring decentralization in bitcoin and ethereum using multiple metrics and granularities," in *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, Apr. 2021, pp. 80–87. [Online]. Available: http://dx.doi.org/10.1109/icdew53142.2021.00022

[214] L. Zhang, X. Ma, and Y. Liu, "Sok: Blockchain decentralization," RePEc: Research Papers in Economics, 2022.

[215] J. Lee, B. Lee, J. Jung, H. Shim, and H. Kim, "Dq: Two approaches to measure the degree of decentralization of blockchain," *ICT Express*, vol. 7, no. 3, pp. 278–282, Sep. 2021. [Online]. Available: http://dx.doi.org/10.1016/j.icte.2021.08.008

[216] S. Motepalli and H.-A. Jacobsen, "Decentralization in pos blockchain consensus: Quantification and advancement," *IEEE Transactions on Network and Service Management*, vol. 22, no. 4, pp. 2930–2943, Aug. 2025. [Online]. Available: http://dx.doi.org/10.1109/tnsm.2025.3561098

[217] ——, "How does stake distribution influence consensus? analyzing blockchain decentralization," in *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, May 2024, pp. 343–352. [Online]. Available: http://dx.doi.org/10.1109/icbc59979.2024.10634400

[218] J. Luo, "Unveiling ethereum's p2p network: The role of chain and client diversity," arXiv (Cornell University), 2025.

[219] J. Ron, Z. He, and M. Monperrus, "Proving and rewarding client diversity to strengthen

resilience of blockchain networks," *Distributed Ledger Technologies: Research and Practice*, vol. 5, no. 4, pp. 1–22, Jun. 2026. [Online]. Available: http://dx.doi.org/10.1145/3773288

[220] A. Howell, T. Saber, and M. Bendechache, "Measuring node decentralisation in blockchain peer to peer networks," *Blockchain: Research and Applications*, vol. 4, no. 1, p. 100109, Mar. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.bcra.2022.100109

[221] S. Yang, K. Nayak, and F. Zhang, "Decentralization of ethereum's builder market," arXiv (Cornell University), 2024.

[222] R. Ma, W. Tang, and D. D. Yao, "Analysis of the order flow auction under proposer-builder separation on blockchain," RePEc: Research Papers in Economics, 2025.

[223] G. A. Pierro and R. Tonelli, "Can solana be the solution to the blockchain scalability problem?" in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Mar. 2022, pp. 1219–1226. [Online]. Available: http://dx.doi.org/10.1109/saner53432.2022.00144

[224] F. Duffy, M. Bendechache, and I. Tal, "Can solana's high throughput be an enabler for iot?" in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, Dec. 2021, pp. 615–621. [Online]. Available: http://dx.doi.org/10.1109/qrs-c55045.2021.00094