ring0: A New Way to Build Blockchains
Lightweight

Luis Angel Gonzalez

Ring0 Research Team
angel@r-0.org

Abstract. Progress depends on finding and correcting errors, but a blockchain cannot improve by making its accepted past another thing to rewrite. Its machinery, from execution and storage to proof and consensus, will age. Replacing that machinery should not change what the network recognizes, who may change it, which obligations remain, or why its history can still be verified.
ring0 is a decentralized computer built around that distinction. In plain terms, ring0 is a giant table that freezes liars. Its identity is not a permanent stack, but the finalized, verifiable continuity of state, authority, obligations, and history across replaceable engines. Once accepted, no producer or operator gets to rewrite what the network agreed.
The initial profile separates execution from acceptance and distributes work among parallel atomicity zones, but it is only one way to advance the shared record.
ERAS lets the network outlive that profile. A proposed successor is a claim to be tested, not an upgrade merely because it was named or voted so. ERAS classifies a change by its effect on shared meaning and accepts the successor only when the exact candidate is authorized, ready, continuity-correct, finalized through one unambiguous handoff, and activated within its approved window. An Era may change how work is executed, stored, proved, or finalized, but it may not break the continuity that makes its results part of ring0. The engine changes. The network remains.

Keywords: Decentralized computer · Verifiable continuity · Layer-1 · Parallel execution · Network evolution

1 A Decentralized Computer for Useful Work

A decentralized computer becomes useful when independent participants can act on the same result without giving one machine the final word. To do so, it must make the chain inspectable: the state that was accepted, the transition that changed it, and the rules under which that transition was valid. Running the program is only part of the problem; its consequence must become common.

That problem predates blockchains. Distributed-systems research established how independent machines can order events and reach agreement despite faulty participants [5,6,20,21]. Cryptographic timestamping and Merkle trees showed

how a later alteration to a record can be exposed [18,19]. Together, these ideas made it possible to preserve a shared history whose integrity could be checked by someone other than its author.

Bitcoin made a public, independently verifiable history practical; Ethereum turned that history into a programmable state-transition system [7,8,9]. The ledger became a computer, but conventional state-machine replication ties verification to execution: validators establish the next state by re-executing the same transition against the same prior state. As programs and state grow, verification grows with them because agreement and computation are replicated together.

Two lines of work loosen that coupling. Partitioned designs divide a workload among cooperating parts [17,9]; succinct proof systems let a participant check a claim about a computation without repeating the computation itself [16,11]. Neither idea alone supplies the whole architecture. Parallel work must explain how independently processed effects rejoin one accepted history, while proof-carrying execution must explain who may produce, verify, and finalize the result.

ring0 combines these lines at the base layer. Its initial execution profile divides work among atomicity zones; a selected producer executes and proves a transition; validators check the evidence; and senators finalize after verification. When an effect crosses zones, the message carries evidence of a finalized transition rather than asking the destination to accept another zone's assertion. The design separates the cost of doing the work from the authority to declare it accepted [1].

This separation provides the first answer: independent verification does not require uniform work. A few machines can perform the expensive computation while a wider class of participants checks what those machines claim. Evidence travels. Computation does not.

The answer creates a deeper question. An execution engine, a database, a proof system, and consensus rules can all be replaced as better ideas arrive or failures are discovered. If each replacement can also redefine accepted state, widen authority, discard obligations, or sever history, the network has not evolved; it has changed identity. The next section therefore separates ring0 from the machinery of any Era and states what must remain continuous.

2 What ring0 Is

Each Era may change how ring0 executes, stores, proves, or finalizes work. Yet participants need one answer that survives those changes: what makes a result part of the same network? ring0 answers at the level of the shared record. It lets independent participants determine which state was accepted, under which rules, and what follows from it, without giving one operator the final word.

That answer defines the identity of the network:

> *ring0 is the finalized, verifiable continuity of state, authority, obligations, and history across replaceable engines.*

These four terms describe the minimum record that every Era must inherit. *State* is what the network currently recognizes. *Authority* determines who may change that state and within which limits. *Obligations* are accepted work or transfers that have begun but are not yet complete. *History* is the accepted path that explains how the present came to be. Continuity requires a successor to carry all four forward: it may not silently create permission, lose a pending duty, duplicate value, or make an earlier claim impossible to inspect.

A virtual machine, a storage design, a proof system, and consensus rules are therefore machinery, not identity. Together they form the engine of an Era. They may be replaced; the meaning of what the network has already accepted may not be replaced with them. Figure 1 shows this relationship: engines succeed one another while the same verifiable continuity joins their records.

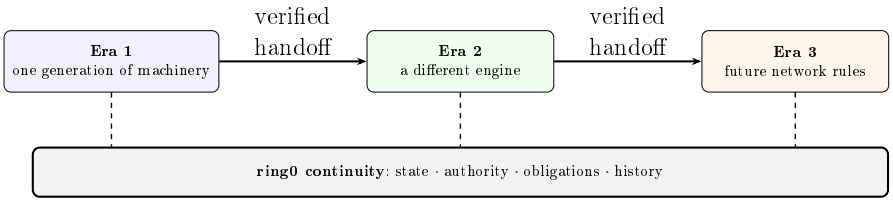

Fig. 1. The verified chain of preserved meaning, and not any machinery held in common, is what makes all three Eras part of the same network.

3 How ring0 Works

The previous section states what ring0 must preserve. Its first engine addresses a different problem: how to increase capacity without dividing the network into unrelated histories. ring0 assigns work to parallel *atomicity zones*. Each zone maintains its own state and transaction order, while messages between zones carry evidence of the state changes that produced them.

At a high level, the zones resemble multiple processor cores because they create independent places where work can proceed [13]. The comparison ends at coordination: unlike cores inside one machine, zones maintain separate local state, and communication between them still limits the achievable speedup [12]. They remain part of one network because they share a finality system and a common, inspectable history. That coordination cost is the price of not being separate chains.

Within a zone, the scheduler combines operations whose order does not change the outcome and preserves an explicit order when one operation depends on another. The rule is simple: parallel where the transition permits it, ordered where the transition requires it.

Figure 2 follows one transaction. RLNC disseminates it; APEX schedules it inside a zone; Jolt Pro GR executes the program and proves the transition; and MLE-DB binds that transition to the state the network had already accepted.

Validators check the evidence, and senators finalize the source block. If the action affects another zone, the resulting message enters a destination block carrying evidence of the finalized source transition. The destination verifies that evidence, applies the corresponding local transition, and finalizes it as part of its own block. Parallel local work is thereby reconciled with one shared history.

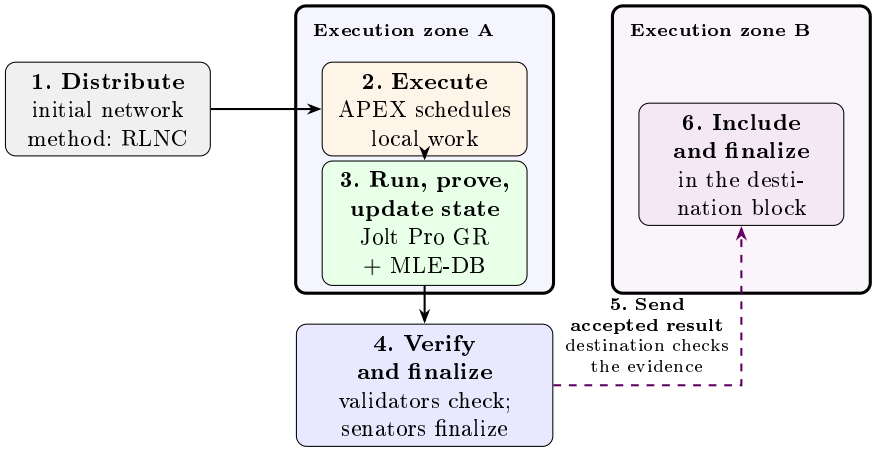

Fig. 2. A transaction flow in ring0. Independent transactions proceed in parallel, while a cross-zone effect becomes accepted only after the destination verifies the source evidence and finalizes its own block.

The initial engine assigns a concrete mechanism to each part of that path. RLNC [15] breaks transactions, blocks, and proofs into redundant pieces that relay nodes can combine while forwarding. APEX schedules work inside each zone [1]; Jolt Pro GR executes programs and proves their transitions [3]; and MLE-DB represents state so that the claimed transition can be checked against what the network had already accepted. These components implement the path; they do not define ring0.

Two branches remain optional. Eligible private work can place scheduling, execution, and proving inside an HSS-CoFHE confidentiality envelope, revealing only the authorized result [4]. After finality, an application may ask CHITA to keep a designated artifact available through separate storage providers, continued custody checks, and repair [2]. Ordinary transactions require neither branch. Figure 3 maps the architecture to this kickoff profile and separates its ordinary path from those optional branches.

The reference configurations report the following component-level results. APEX exceeds 1.1M transactions per second on a 96-core node. HSS-CoFHE runs each primitive circuit operation 3,506× faster than its baseline [1].

The machinery explains how work moves, but it does not decide who may produce, check, finalize, or preserve that work. The next section separates those responsibilities.

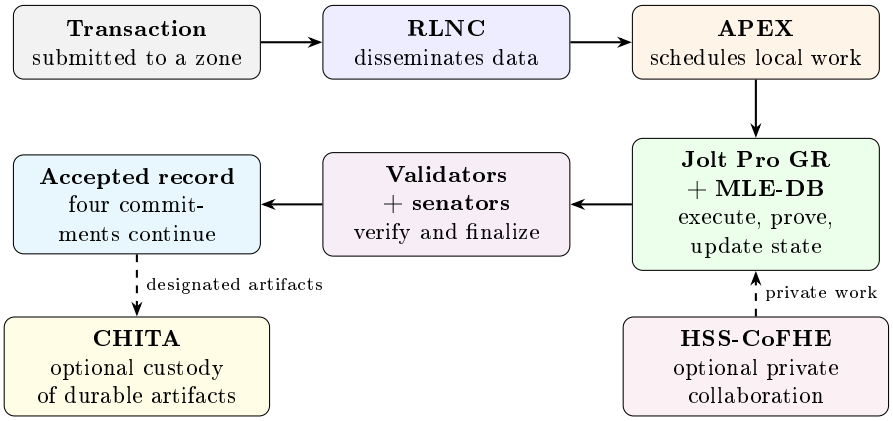

Fig. 3. The kickoff engine. The solid path carries ordinary work to an accepted record. The optional privacy component (HSS-CoFHE) supports selected private work, while the archival component (CHITA) receives only accepted artifacts that an application asks the network to keep available.

4 Who Does What

That engine is operated through distinct responsibilities, since producing a candidate block and making it final are different powers. The initial role taxonomy is nested: every senator is a validator, but not every validator is a senator; the slot producer is selected from the eligible senator set. Archival providers remain outside that hierarchy [1].

Open applications and users. People and applications submit transactions and receive verifiable outcomes, without needing to operate the infrastructure themselves.

Producers. A selected slot producer receives work for a zone, executes it, proposes the next state, and creates the proof. This is the computationally expensive role, not an independent class with authority to finalize its own output.

Validators and senators. Validators check the producer's evidence and required data instead of repeating all of the work. Senators are the staked subset assigned the finality role in the initial design. Their vote follows verification; it does not replace it.

Storage providers. Archival storage providers are separate from producers and senators. They keep encoded copies of durable work, answer custody checks, and repair a missing piece. They do not decide finality, and a finalized record does not depend on one provider's private database.

Figure 4 draws the nesting.

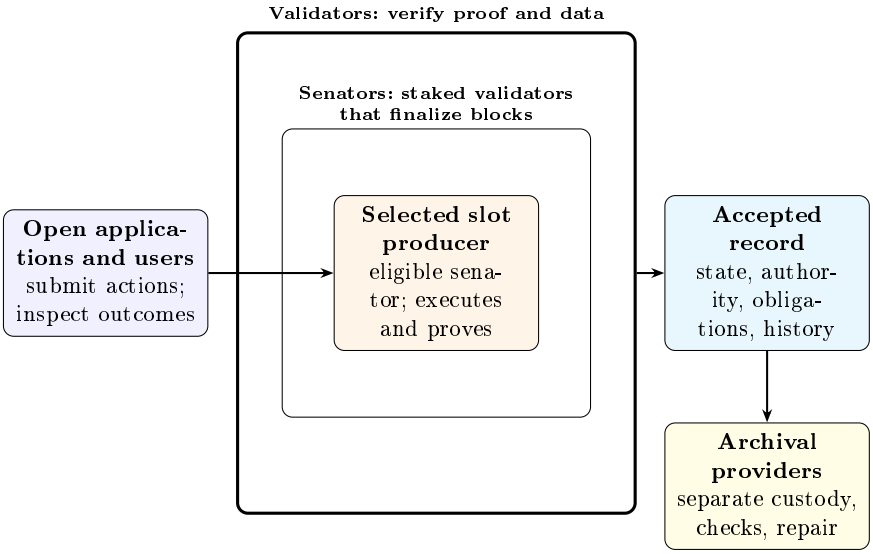

Fig. 4. Initial ring0 infrastructure. Senators are a subset of validators, and the slot producer is selected from the eligible senator set. The accepted record passes only designated durable work to archival providers, which remain a separate branch.

Each of these roles is therefore bounded by the others. Producers cannot make their own work final; senators cannot make an invalid proof correct; storage providers cannot alter the accepted record. An application can create useful work without becoming the sole authority on what happened.

5 ERAS: How ring0 Changes Without Losing Its Identity

The previous sections explain how one engine performs work and how distinct roles prevent a participant from authorizing its own result. A harder problem begins when the engine itself must change. Hardware improves, attacks evolve, and better designs appear. A network that cannot replace its machinery becomes obsolete; one that replaces its rules by decree asks users to trust whoever controls the upgrade. ERAS, the network's Era system [1], governs the space between those failures.

An Era is one chapter of the network: a period during which participants rely on one set of rules about who may authorize work, when a decision is final, what must remain available as evidence, and how a successor may take over. An Era may introduce new rules through an explicit network-wide change, but its handoff may not erase or silently redefine the four commitments that make the successor part of the same network.

State remains identifiable. A successor must preserve or explicitly transform the logical facts the network accepted, even when their database layout, proof

system, or root changes. An independent participant must still be able to identify the accepted starting point from which work continues.

Authority cannot silently widen. Every permission in the successor must come from authority already present in the predecessor or from an explicitly authorized transformation. Changing machinery cannot create ownership, delegation, or power by accident.

Obligations are neither lost nor duplicated. Pending messages, settlements, withdrawals, time locks, and similar work must complete, expire under an existing rule, or appear exactly once in the successor. An update cannot become a way to forget an inconvenient duty or perform it twice.

History remains interpretable. Every finalized block remains bound to the rules active when it finalized. A successor may introduce new rules, but it cannot reinterpret earlier transactions, receipts, outcomes, or finality. When verification depends on material outside the chain, the Era record must preserve its identity, the rules needed to interpret it, and a verifiable route by which it can be retrieved.

These commitments describe meaning, not machinery. An *engine* does the work. An *execution profile* fixes what counts as a valid result of that work. An *Era* fixes the global rules by which the network accepts, finalizes, and hands that meaning to a successor. ERAS classifies a change by which of these levels it can reach, not by the number of lines in its patch.

Class 0 — Implementation.
The machinery changes, but the accepted relation, proof interface, and visible result do not. This is a local replacement, not a protocol upgrade.

Class 1 — Bounded setting.
A versioned value moves within limits that the active profile or Era already authorized. The change cannot silently introduce a state migration.

Class 2 — Execution profile.
The way a zone represents, executes, proves, authenticates, protects, or prices work changes. The successor must show that protected facts and pending obligations carry forward through a finalized profile handoff.

Class 3 — Network Era.
Consensus, finality, global authority, cross-zone rules, light-client interpretation, or the rule for verifying the next successor changes. The old Era must finalize one network-wide handoff into the new Era.

Class 4 — Bounded containment.
No successor is installed. A temporary restriction narrows a function or limit whose target and bounds were already declared by the active Era.

Class 4 is not a general power to stop the network, nor is it a higher rung than Class 3. Its target, observable trigger, maximum reach, expiry, and attesting threshold are fixed before the emergency exists. Validators check its invocation under the current rules; no producer, developer, operator, or single validator can create or extend the power at the moment of danger. It may refuse new use of one declared optional function or reduce an existing limit. It cannot install code, create authority, move balances, cancel accepted receipts, change consensus or finality, or reinterpret finalized history. A permanent remedy follows Class 2 or Class 3, according to what it changes.

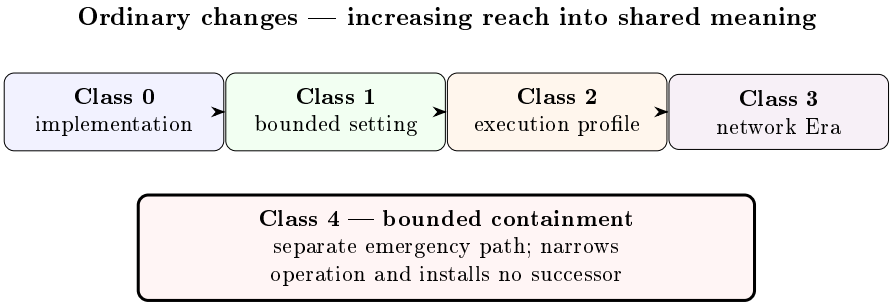

Fig. 5. Classes 0–3 are ordered by their effect on shared meaning. Class 4 is not the top of that ladder: it is a separately bounded restriction that temporarily narrows operation without adding authority or replacing the Era.

Classification determines the burden of proof; it does not activate a successor. When a Class 2 or Class 3 change introduces a new profile or Era, its rules, software, expected state transformation, operating requirements, and recovery plan are fixed as one reviewable candidate. That candidate becomes active only if it was authorized, is ready to operate, preserves the four commitments, joins predecessor and successor through one finalized handoff, and enters during its approved activation window.

The five conditions are not interchangeable. Governance may establish collective authorization, but it cannot make incompatible state compatible, an unready system reliable, or a broken transition correct. Testing may establish readiness but confers no authority. The finalized handoff is the single public boundary at which the five conclusions become one accepted change.

Before that boundary, failed evidence leaves the activation conditions unsatisfied and the predecessor remains authoritative. After that point, recovery moves forward only. Participants may restrict future service or prepare another successor, but they may not quietly restore the predecessor as the canonical present. Otherwise recovery would become an informal power to rewrite finality. ERAS does not promise that an Era will be perfect. It requires every Era to make its own replacement verifiable.

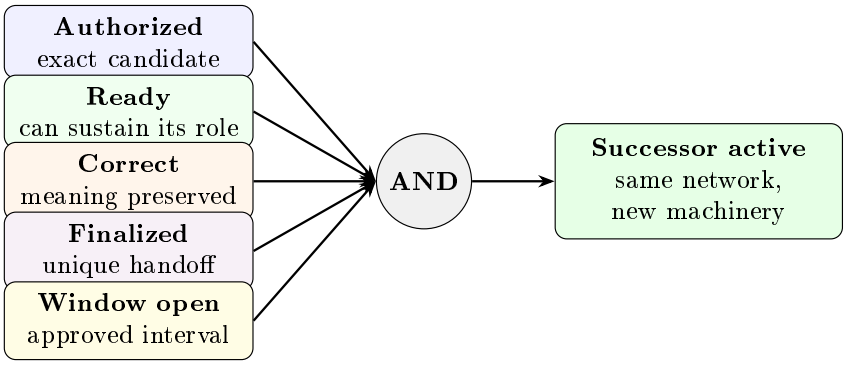

Fig. 6. A vote is one input, not the whole transition. Authorization, readiness, correctness, finality, and the activation window must all hold before the network accepts a successor.

6 Applications on ring0

The continuity above makes ring0 usable beyond any one engine or product. ring0 is open infrastructure: any person, team, or community may build above the network without joining its consensus engine or adopting a fixed application design.

An application remains responsible for its interface, workflow, private or large data, and user experience. It places on ring0 only the consequences that must become common: who authorized a change, which state or version became current, which obligation remains, or which receipt proves completion. ring0 does not host every screen, store every byte, or decide whether the surrounding work is useful.

Applications may support exchange, coordination, storage, metered services, or delegated and private computation. Their purposes differ, but their relation to ring0 is the same: each keeps its product logic and private data outside consensus while using the network for facts whose authority, acceptance, or continuity must be shared.

Figure 7 shows that boundary. Different kinds of application may depend on the same accepted record without becoming components of the protocol.

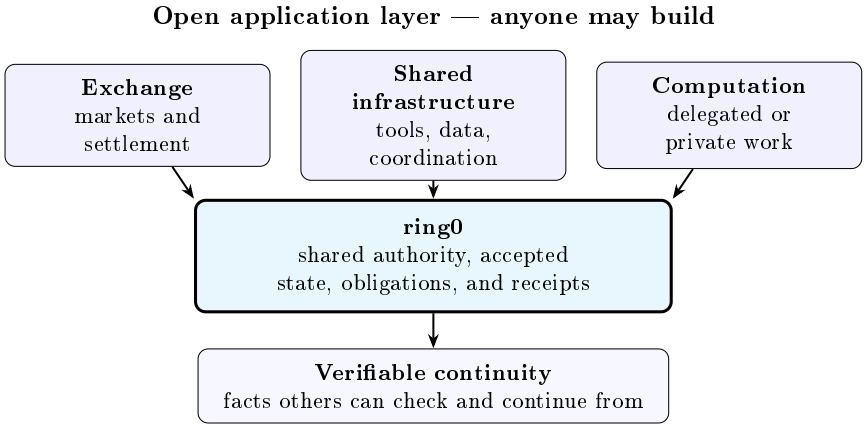

Fig. 7. An open application layer above one verifiable network. Applications own their products and data; ring0 carries only the shared facts whose authority, acceptance, or continuity must survive.

Open source and an accepted record solve different problems. Open source lets others inspect, fork, and reimplement an application; ring0 lets independent implementations share facts that none may silently rewrite. Published code does not prove what a running service did, and an accepted record does not certify every off-chain claim. An application can therefore change its interface, storage, and internal machinery while the facts that must survive a change of software, operator, or provider remain independently checkable.

7 Conclusion

Every design can be wrong, and every machine eventually becomes old. A decentralized computer cannot answer that fact by freezing its machinery. Nor can it call every replacement progress. Its identity must lie in what survives correction: the state it recognizes, the authority under which that state may change, the obligations still in force, and the history that lets independent participants verify how the present came to be.

ring0 answers with separation. Execution, verification, finality, and archival custody are distinct responsibilities; producing a result does not confer the authority to make it final, and preserving an artifact does not confer authority over accepted state. Applications remain above the protocol, bringing to the network only the facts whose acceptance or continuity must become common. Acceptance follows verified evidence and finality rules, not uniform re-execution.

Separation does not make an accepted transition infallible. A traceable transition can still be wrong. Its advantage is that the error remains open to inspection and challenge, and that correction must proceed through a rule-bound successor rather than an unrecorded rewrite. Progress requires the possibility of correction without making accepted history disposable.

No Era is the last. ERAS treats every successor as a claim to be tested: state, authority, obligations, and history must pass through one unambiguous, finalized handoff. Before that boundary, a defective candidate acquires no authority. After it, correction moves forward through another valid transition rather than quietly restoring the predecessor as the canonical present. ring0 does not promise a final machine. It requires each Era to make its own replacement verifiable. The stack changes; the network remains.

References

1. L. A. Gonzalez, *ring0: A New Way to Build Blockchains*. Companion manuscript, Ring0 Research Team (2026).
2. L. A. Gonzalez, *CHITA: Retrievable Archival Storage for a Sovereign Galois-Ring Layer-1*. Ring0 Research Team companion manuscript (2026).
3. L. A. Gonzalez, *Jolt Pro GR: A Galois-Ring-Native zkVM beyond Prime Fields*. Technical report, Ring0 Research Team (2026).
4. L. A. Gonzalez, *HSS-CoFHE: Cooperative Fully Homomorphic Encryption via Homomorphic Secret Sharing*. Technical report, Ring0 Research Team (2026).
5. L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978. doi: 10.1145/359545.359563.
6. L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982. doi: 10.1145/357172.357176.
7. S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. `https://bitcoin.org/bitcoin.pdf`
8. V. Buterin, *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*, 2014. `https://ethereum.org/whitepaper/`
9. G. Wood, *Polkadot: Vision for a Heterogeneous Multi-Chain Framework*, 2016. `https://assets.polkadot.network/Polkadot-whitepaper.pdf`
10. L. M. Goodman, *Tezos: A Self-Amending Crypto-Ledger (Position Paper)*, 2014. `https://tezos.com/position-paper.pdf`
11. L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain scaling using rollups: A comprehensive survey," *IEEE Access*, vol. 10, pp. 93039–93054, 2022. doi: 10.1109/ACCESS.2022.3200051.
12. G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the AFIPS Spring Joint Computer Conference*, 1967, pp. 483–485. doi: 10.1145/1465482.1465560.
13. J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988. doi: 10.1145/42411.42415.
14. R. Gelashvili, A. Spiegelman, Z. Xiang, and G. Danezis, "Block-STM: Scaling blockchain execution by turning ordering curse to a performance blessing," in *Proceedings of the 28th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2023, pp. 232–244. doi: 10.1145/3572848.3577524.

15. T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006. doi: 10.1109/TIT.2006.881746.
16. E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von Neumann architecture," in *23rd USENIX Security Symposium*, 2014, pp. 781–796. `https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson`
17. L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 17–30. doi: 10.1145/2976749.2978389.
18. S. Haber and W. S. Stornetta, "How to time-stamp a digital document," *Journal of Cryptology*, vol. 3, no. 2, pp. 99–111, 1991. doi: 10.1007/BF00196791.
19. R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology—CRYPTO '87*, 1988, pp. 369–378. doi: 10.1007/3-540-48184-2_32.
20. 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, 1988. doi: 10.1145/42282.42283.
21. M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 1999, pp. 173–186.