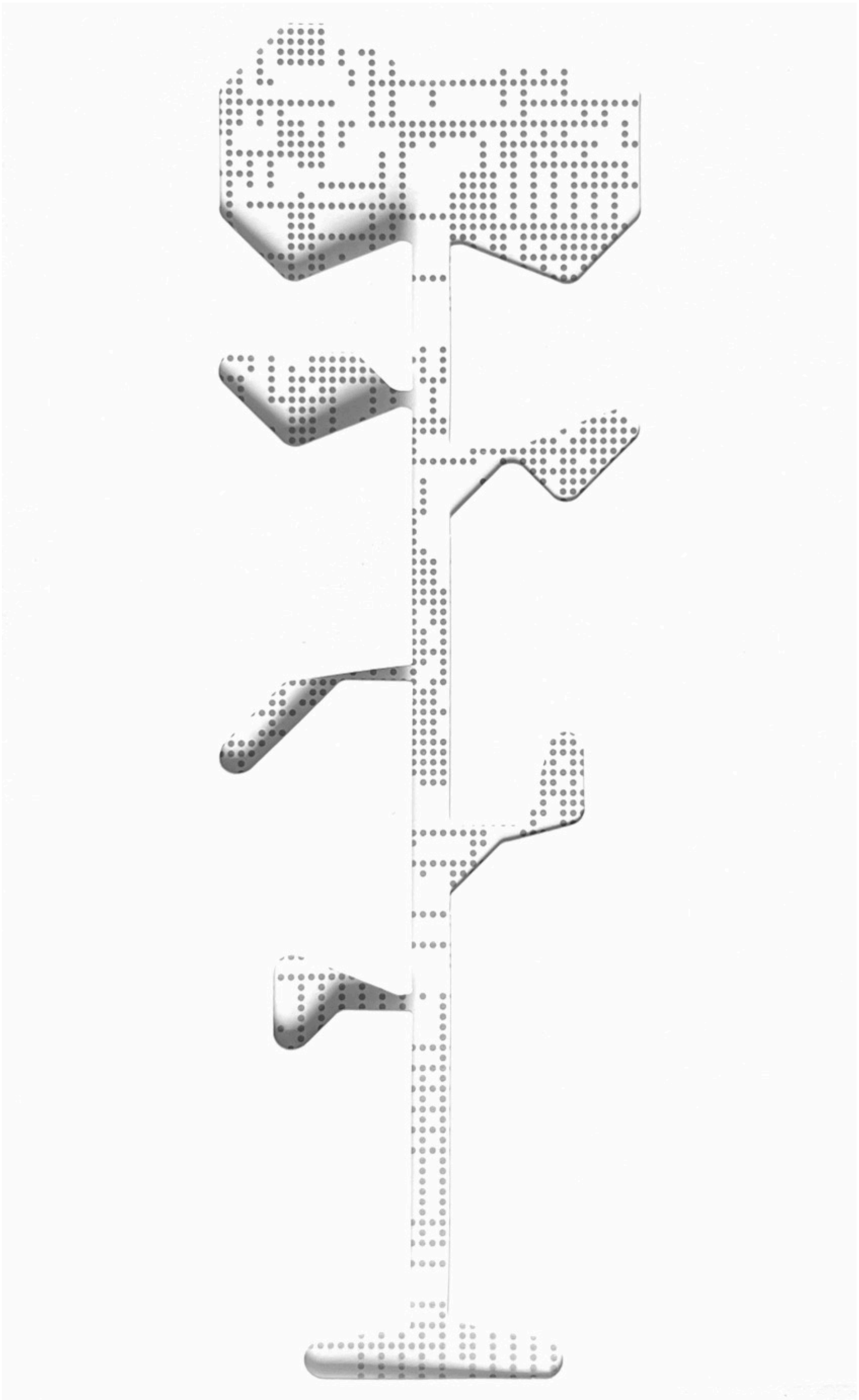
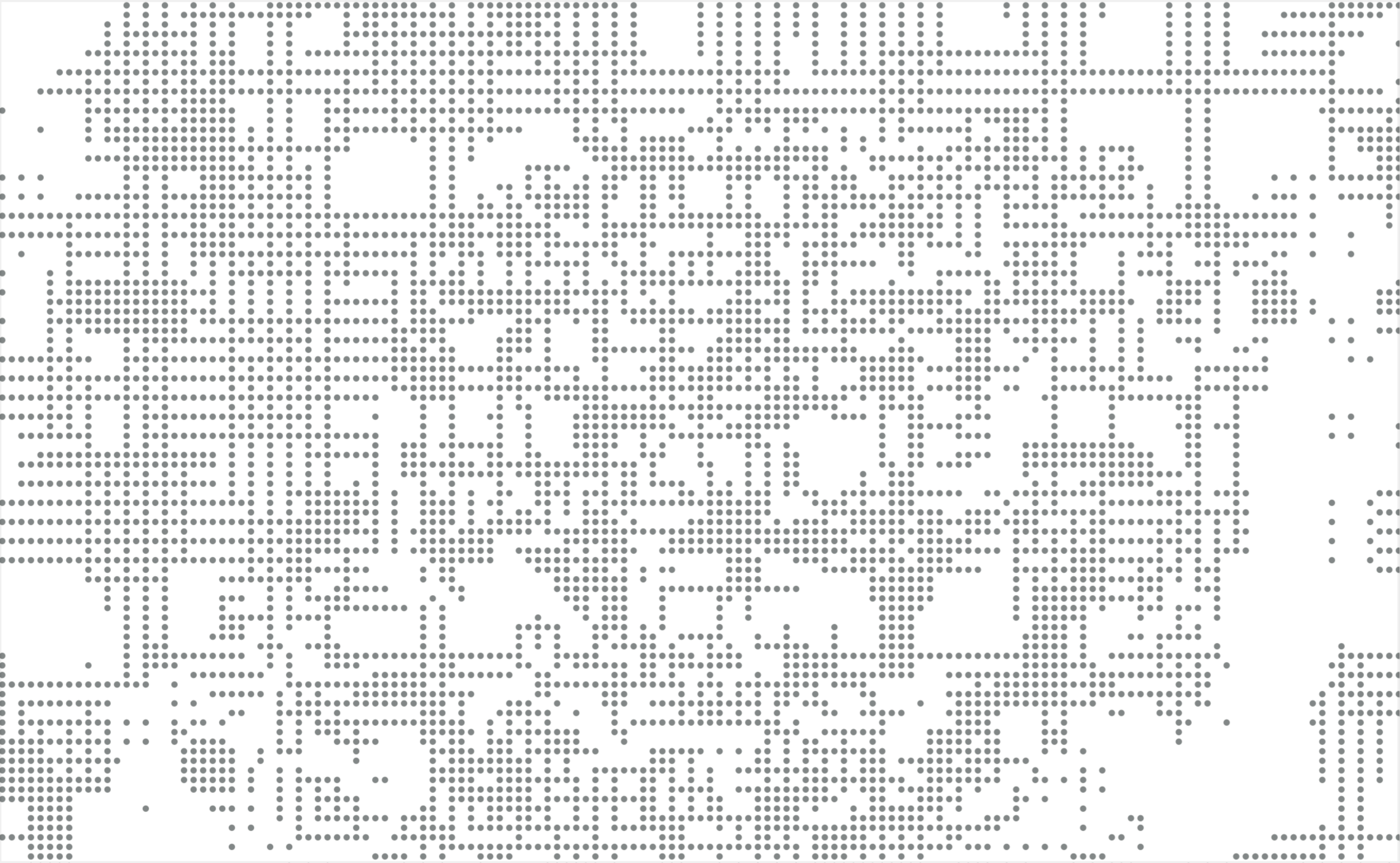
Nockchain		V0.438594	
		xf04950505	ZK_DECODE





5f8b7cda

93e4b5f2

6a1d7b3e

4c8a2d5f

72e9f3bd

1a6f4c9e

83b7a5d2

4d2e7f6a

7b9c2e5f

91a4d7b8

2e5f9b7c

3d6a8e4b

75c1d9a3

6e4b3f8a

82d7b5c1

9a5c7f2e

2e9a7f3c

8d1b4c6f

5c7e3a9b

6b2d4f8a

3f8c5b7e

7a1e9b4d

9c4f2a7e

1b6a8d3f

4d7e9b2c

2f3c7d6a

7b9a5f4e

6d2e4b1f

3a5c7d8f

8e7b2f9a

5f1c4b7d

9d6a3e8b

5f8b7cda

93e4b5f2

6a1d7b3e

4c8a2d5f

72e9f3bd

1a6f4c9e

83b7a5d2

4d2e7f6a

7b9c2e5f

91a4d7b8

2e5f9b7c

3d6a8e4b

75c1d9a3

6e4b3f8a

82d7b5c1

9a5c7f2e

FROM: BLACK LAKE, ONTARIO (NODE 0277) 2e9a7f3c

8d1b4c6f

TO:INTERZONE (NODE 4415) 5c7e3a9b

6b2d4f8a

DATE: 08-11-2098 3f8c5b7e

7a1e9b4d

CLASSIFICATION: ZK-REDACTED // UNCLASSIFIED 9c4f2a7e

The teenagers call it “freebasing,” at least in the government schools. Thankfully, none of your 12 great-grandchildren have any idea. Many blame the iPhone 72 Pro back in ‘89—once VR and LLMs were finally uni- 1b6a8d3f
fied. But we’re fine. Our devices never even see those packets. Disreputable senders don’t even try to breach AAA-rated NockNets anymore. I could prove it, too, 4d7e9b2c
but that went out of fashion in ‘92. It’s considered gauche to flaunt proofs, nowadays. Just last week 96% of the students in Brewster County Federal High School Supermax got one-shotted by God knows who... Institu- 2f3c7d6a
tional money says it was the Sino- Sinaloa, but I’m pretty sure it was a small outfit in Indonesia. That’s what little Johnny’s research shows, anyway; we’ll know when the data-market clears. He’s the oldest, Johnny; 7b9a5f4e
he’s a lot like you, that’s what Dad always told me. At first Johnny was mad because the NockPhone isn’t as bright as the others. But with an income of $75k/month at the age of 16, let’s just say his anger cooled. Run- 6d2e4b1f
ning four remote companies is never easy, but automa- tion and homeschooling ain’t what they used to be; all of the 23 agents on that thing are bumping IQs well above 330 with the strictest AAA-rated Thomist 3a5c7d8f
Virtue Ethics module on the market. Anyway, I can’t be long—N-time break-warps are still immature, expensive, and dangerous.

8e7b2f9a

If you’re reading this, congratulations, you’re the first recipient of a successful N-warp on mainnet. It’d be risky to tell you much else.

5f1c4b7d

I just wanted to let you know we’re safe.
And grateful for your sacrifices.

9d6a3e8b

In the beginning, the internet was a wilderness. The pioneers of cyberspace hoped for a Garden of Eden, but what they got was the Fall. Frontiers are uncivilized, so the internet was quickly beset by barbarians–in this case, spammers, hackers, trolls, and bots. In the resulting maelstrom, digital territories haphazardly formed, and cyberspace was divided into a smattering of bordered interiors protecting against the harsh wilderness.

There are two types of digital territory: the permissioned platform, and the permissionless protocol.

Permissioned platforms arise when a regime controls computers. In a permissioned platform, borders are declared by decree. Within the permissioned platform, subjects inhabit a walled computational garden. They must follow the ruler's decrees, or be denied service. Outside services are banned and affordances are restricted. Permissioned platforms may be subjected to standard regime analysis. Search engines, social networks, and payment platforms are all examples of permissioned platforms. Each one became highly contested ground in the 2020s, as different political tribes jostled for the power to algorithmically enforce worldviews upon their subjects.

Permissionless protocols, on the other hand, offer only a ruleset. Any agent is welcome to participate in the protocol (or ignore it). No agent is privileged over any other. Agents are driven by incentives to form a swarm that actualizes the protocol. Protocols constrain the actions of agents who wish to play the game. The rules of a good protocol are minimal, specific, and predictable. Anyone may combine a protocol's affordances to create new functionality.

Proof-of-Work (PoW) is a particularly pure protocol. Its digital territory consists of blockspace, commodified computation, and objective history. The PoW algorithm creates a consensus reality for the territory and discretely sequences time for all participants. In a territory founded with no pre-mine, a pristine wilderness is freely settled and developed by pioneers. PoW allocates value to those who speculatively sacrifice computational resources to the protocol precisely in proportion to the magnitude of their sacrifice. Miners, by sacrificing energy, capital, and computation, publicly signal to the broader market their belief in the protocol.

Anthropologically speaking, the public performance art of sacrifice generates a Priesthood, and with it civilization. Those who engage in costly activities for unclear prospects appear irrational and evoke a polarizing response. This polarization creates clear borders or boundaries, an interior and exterior, giving form and structure to a society. By proving his faith, he signals to others that he can be trusted to cooperate, although only with those who are willing to make the same sacrifice. Binary oppositions–sacred and profane, interior and exterior–are the basic structures of human societies. The priest grounds meaning through the authority of revelation, serving as mediator of these dualities. So long as meaning is grounded through these sacrificial acts, the *polis* remains stable and reproduces its *nomos* each generation.

PoW miners form a minimal priesthood, proving to the world that the protocol is meaningful through the public sacrifice of real energy and computation. The machine is only meaningful because it issues blocks, and the machine can only issue blocks because it is meaningful. A digital pyre of electricity and silicon creates value out of waste. Miners enforce the borders of the

digital territory by choosing which transactions to publish each block, warding off the demons of the harsh exterior. In doing so, miners fulfill the priestly function of mediation between realms and reinforcement of digital societal structure.

The Treaty of Westphalia gave European states full sovereignty over their own territories, introducing a norm of non-intervention with respect to each other's affairs. The blockchain today is a Westphalian moment for digital society. While different crypto communities squabble, they do not hack each other's consensus protocols. The protocol has sovereignty over the territory it defines, and its subjects have immense freedom otherwise. In the context of high-throughput, Turing-complete blockchains, we are nowhere near the limit of what digital states may become.

Technology, at its core, is asymmetry. Like a lever, which multiplies force, or fire, which transforms matter, technology disrupts pre-existing ways of life. Those who wield the purest forms of asymmetry with the most skill and vision define the future. For good or ill, mankind commands more power now than ever before, but neither skill nor vision are evenly distributed. Because our leverage has never been greater, returns on the courageous application of skill and vision have never been greater. The best entrepreneurs and creatives will increasingly defect to permissionless protocols in their calling to better align the world with truth and beauty.

Influence concentrates in the hands of a skilled few. Historically, the task of shaping civilization falls to an aristocracy. Aristocracy is only stable when its culture reinforces and elevates virtue. The unique personal obligation of the noble to be virtuous—noblesse oblige—is succinctly captured in the Roman legal principle, "salus populi est suprema lex." The health of the people is the supreme law. The best regimes are governed by a nobility personally connected to their subjects. It is hard to elevate the health of the people, in practice, without a personal love for them.

The current world hegemon, in its post-New-Deal form, forsakes personal obligations for bureaucratic institutional incentives. The bureaucrat is neither willing nor able to take personal responsibility for the health of his people. As a result, administrative states operate without accountability, incentivized only toward self-preservation and expansion of power. This system disconnects rulers from subjects, prioritizing global empire over domestic well-being. For a century, these bureaucrats leveraged broadcast media—newspapers, radio, and television—to pacify the populace.

The bureaucratic apparatus and mass media form a symbiotic system of control. Media, as extensions of human senses, reshape social interactions and cultural norms. The one-to-many nature of broadcast media created a mass society, eroding individual identity and local traditions. As passive receivers of standardized content, individuals became disconnected from their immediate surroundings, losing their sense of personhood. This dissolution was accelerated by the homogenizing effect of broadcast media, which created uniform global culture.

Bureaucracy and broadcast media intersected with permissioned digital platforms to terraform human consciousness and unmoor meaning. The age of permissionless protocols begins with an internet dominated by the permissioned platform, just as the internet began in a world dominated by broadcast media. In a world choked by bureaucracy, permissionless protocols provide a massive

Technology, at its core, is asymmetry. Like a lever, which multiplies force, or fire, which transforms matter, technology disrupts pre-existing ways of life. Those who wield the purest forms of asymmetry with the most skill and vision define the future.

opportunity to reclaim the vital fire of free civilization. These protocols structurally obsolesce the homogenizing forces that erode individual agency and traditional cultures.

Nockchain is the technical instantiation of these ideals—a civilization machine built as a permissionless protocol, governed by PoW, synthesizing the purest asymmetries we have discovered across cryptography, functional programming, and distributed systems. Nock, our small assembly language with only 12 rules, is the bedrock of a new digital territory. Nock is so simple that an individual may take full control over their computing environment, a precondition for true digital sovereignty. Nock's simplicity is a departure from the complex underpinnings of modern systems, which require large organizations to maintain everyday software.

Through continuous sacrifice, Nockchain miners will turn a virgin wilderness into a settled territory, becoming priests of a new digital state. By building useful structures on the settled territory, entrepreneurs will take responsibility for the health of their neighbors and finally build civilizations fit for the digital epoch.

In Nockchain, we see the potential to forge new modes of human interaction, untethered from the constraints of bureaucratic institutions and centralized platform control. Nockchain provides tools for individuals to directly shape their digital environments, much as the early pioneers of cyberspace once envisioned. A world of meaning is ours for the taking.

Design

What is Nockchain?

Nockchain is a permissionless protocol that launched with no pre-mine, issuing 100% of the native asset—$NOCK or “nocks”—to miners. It features a new Zero-Knowledge Proof-of-Work (zkPoW) consensus mechanism, differing from typical hashing-based proof-of-work chains. Nockchain uses zkPoW to subsidize competitive hardware optimization for the Nock Zero-Knowledge Virtual Machine (zkVM). $NOCK is the first entrant in a new asset class: zkPoW coins.

Nockchain starts simply, using a basic UTXO system to prevent double-spends. Smart Names, based on the Pay2ScriptHash covenant system, are designed to be forward-compatible with advanced scaling techniques and developer primitives. Nockchain is implemented as a Solid-State Blockchain using the Nock Stack and a new architectural paradigm, the Solid-State App. The Solid-State App architecture simplifies the creation of decentralized applications and distributed systems.

Nockchain is the first NockApp, the forerunner for an independent NockApp ecosystem. The NockApp ecosystem is optimized for developer experience and unit economics. Developer unit economics have been neglected across the crypto space, despite developers being the primary driver of value for permissionless protocols. We intend to empower developers with cutting-edge tools that reduce their capital costs and unleash their creativity.

Why Zero-Knowledge?

Zero-Knowledge Proofs (ZKPs) are the key to achieving the final evolutionary form of blockchains. Naively, to replicate a trustless consensus across machines, every computer has to resync all data and rerun all code from scratch. In low-throughput settings, this is acceptable, but there are information theoretic limits to scaling the naive approach that are rapidly being met in practice. ZKPs allow the compression of verification costs to a tiny, constant size, making the cost to replicate trustless consensus negligible. ZKPs reduce validation costs because a single proof can attest to the validity of an entire chain. Aside from data availability issues, ZKPs allow nearly unlimited throughput without impeding “full node” validation of the chain.

Quantitative scaling of the runtime for on-chain logic yields qualitative benefits as well, especially composability.
If a blockchain can be validated by a small proof, it is simple for one chain to validate another chain inside its execution logic: an otherwise impossible concept. This level of composability is sufficient for entirely decentralized and secure bridges between chains, making all blockchains trivially interoperable.

ZK research has long promised to deliver a series of improvements to blockchains—ranging from scalability, privacy, state minimization, trustless truth machines, verifiable identities, and interoperability. However, the promises of ZK have not yet been realized due to the market lacking sufficient commercially-viable proof capacity. To achieve this, provers must be made faster and more efficient; more physical compute is necessary. Proving will never be commercially-viable at scale without a market to incentivize it. However, hardware development has massive barriers to entry, being prohibitively expensive in time, talent, and capital.

Looking at Bitcoin as a case study, motivated miners increased the hashrate of Bitcoin exponentially for a decade. To put that in perspective, an arbitrary function improved in computational capacity by a factor of quintillion. The fastest

and surest way to cement a new hardware standard is to subsidize it with a block reward. The Nockchain mining incentive will make ZK-proving commercially viable for the first time.

Why a zk-Proof-of-Work Blockchain?

By the late ‘90s, all of the technical knowledge required to build Bitcoin existed in some tangible form on the public internet. That it took another decade to assemble correctly is testament to the specificity of the requirements. First came secure one-way hash functions, which today form the bedrock of much of modern cryptography. Then efficient public key signature schemes were developed, which enabled the deployment of the first digital money exchange protocols. The final essential ingredient required was proof-of-work consensus.

Initially engineered as a solution to email spam, proof-of-work was successfully bootstrapped into Bitcoin as its consensus anchor. Solving a computationally intensive puzzle with variable difficulty allowed, for the first time, the emergence of distributed computational markets at scale. By rewarding miners who provided valid verifiable proofs, Bitcoin incentivized the creation of large, efficient and profitable ASIC clusters performing specific computations iteratively for profit, while also securing the Bitcoin network as a byproduct of the process.

This was no small feat; many subsequent implementations removed the explicit link between remuneration and computation in the leader election process. Indeed, recent consensus research has focused almost exclusively on minimizing computation by clients (and thus maximizing end-user performance) as opposed to leveraging its incentivization within the protocol itself. This approach produces unsustainable proof markets because there is no incentivized computation on the layer that matters most. By incentivizing proofs at the consensus layer, Nockchain generates a sustainable proof market and delivers computational efficiency orders of magnitude higher than state-of-the-art Layer 2 chains by linking the required computation (in our case, zkSTARK computations) to the massive, distributed power of a PoW mining ecosystem.

Thus, we opt to remain within the Bitcoin threat model, to implement a proof-of-work consensus layer. Since the main computational requirement to use the system is ZKP creation, we have made the proof-of-work puzzle a randomized instance of the zkVM itself. By providing security through verified energy expenditure, this system is functionally equivalent to the battle-tested Bitcoin protocol—enjoying the same security benefits. The main innovation, however, lies in the observation that using STARKs as the underlying proof-of-work process will incentivize the rapid iteration of STARK computing solutions. By imposing the same incentive dynamics as Bitcoin, we expect miners to optimize the energy expenditure of zkVM transitions, the throughput of zkVM compute units, and energy sourcing at scale.

The aim is to allow physical computing resources to self-organize into systems that maximize zkVM execution throughput with respect to cost—solely through the iterative improvements to the zkVM transition puzzle that follow from the PoW incentive dynamics of the leader-election process. Critically, we designed the system components interdependently: the same computation required to prove/roll-up a normal transaction is also used to verify a proof-of-work instance, so the hardware designed to mine nocks will later scale blockchain payment systems. We’ve built a computationally efficient protocol designed to solicit the hardware improvements it will need over time.

Such a system was initially designed in [KB19], providing succinct state proofs for the chain as a byproduct of PoW while maintaining Nakamoto security guarantees. It was then shown in [KT20] that this useful zkPoW primitive can be used alongside Nakamoto consensus to achieve Nash equilibrium in the price of transaction fees over fluctuating transaction throughput. We believe that this is a fundamental paradigm shift in understanding the capabilities of PoW consensus in distributed systems. The design of PoW puzzles is hugely impactful and can be extensively leveraged for a network's computational needs. Our design resolves the problem faced by many rollup implementations: namely, prohibitive costs and performance requirements in the absence of a long-term economic model.

Moreover, the hardware acceleration we anticipate will happen in a decentralized—and viciously competitive—fashion, echoing the history of Bitcoin mining. Although solutions from established hardware professionals have recently started to surface in the ZK space, to our knowledge not one has yet demonstrated business viability with potential for scale. How could they? Given the nascent state of ZK-based blockchain protocols, we cannot yet point to even one ecosystem that is sufficiently mature to permit such actors to thrive. Such a system can only exist if it is built from the ground up with the correct incentives and consensus rules.

Fair Launch

A Fair Launch features no pre-mine, permissionless mining from the genesis block, and a secure PoW puzzle. Nockchain adopts the Fair Launch model to ensure fair competition, decentralization, and regulatory clarity. By distributing 100% of its coins to miners, Nockchain creates neutral rules for all participants from the start. This strategy minimizes advantages for the protocol developers and allows PoW to allocate the supply of coins based on skill, audacity, and skin in the game.

Token Issuance Schedule

Nockchain's issuance schedule has a standard halving of the issuance rate at the end of each epoch. The epochs are less traditional and defined using a backoff algorithm, such that each successive epoch is longer than the last until it hits a final max epoch length of 7 years. The hard cap on the number of nocks that will ever exist is set at 2^{32} (4,294,967,296).

Epoch	Years Post-Launch	Backoff Period in Months	Issuance Rate per Block
0	1/4	3	65,536
1	3/4	6	32,768
2	$1^{1/2}$	9	16,384
3	3	18	8,192
4	5	24	4,096
5	8	36	2,048
6	12	48	1,024
7	17	60	512
8	23	72	256
8	30	84	128
8	37	84	64
8	44	84	32
8	51	84	32
8	58	84	8
8	65	84	4
8	72	84	2
8	79	84	1

Planetary scale Memory Integrity Infrastructure

Planetary scale

Planetary scale Memory Integrity Infrastructur
Planetary scale Memory Integ
Planetary scale Memory Integrity Infrastructure
Planetary scale Memory Integrity Infrastructure
Planetary scale Memory Integrity Infrastructure
Planetary scale Memory Integrity Infrastructure

Planetary scale Memory Integrity Infrastructure
Planetary scale Memory Integrity Infrastructure

Memory

Planetary scale Memory Integrity Infrastructure

Integrity

Planetary scale Memory Integrity Infrastructure
Planetary scale Memory Integrity Infrastructure
Planetary scale Memory Integrity Infrastructure
re
rity Infrastructure

Infrastructure

Planetary scale Memory Integrity Infrastructure

Planetary scale Memory

Planetary scale Memory Integrity Infrastructure Planetary scale Memory Integrity Infrastructure

Planetary scale Memory Integrity Infrastructure Planetary scale Memory Integrity Infrastructure

Integrity Infrastructure

Technical Architecture

Blockchains are immutable, eventually-consistent distributed systems that allow mutually-distrustful parties to arrive at a global consensus. Most blockchain implementations use stateful, mutable, IO-bound implementations. Nockchain is built with a novel implementation approach—we call it a solid-state blockchain.

Nockchain wouldn't be Nockchain without Nock. Nock is a functional combinator language designed as an assembly-level executable specification layer for software. We execute Nock with Sword, a state-of-the-art solid-state interpreter designed specifically for performant and durable execution.

Nockchain's implementation has three main components. The kernel is implemented in Hoon, a systems-level functional programming language designedto compile to Nock. The kernel implements the core state machine which runs consensus and transaction logic. Sword, the Nock runtime, executes the kernel and manages persistence for all program data, including block updates and user actions. A manager process, referred to as the King, handles user and network IO and controls the Sword process.

The solid-state interpreter represents all state in a single-level store. The system automatically handles persistence of data and paging data between the hard disk and RAM. Computations in a solid-state interpreter are discrete, interruptible, atomic events that must either succeed or fail to update the state. A solid-state interpreter does not forget its execution history. Rather, it remembers its state between process invocations and computer power cycles, making persistence rather than ephemerality the default. Sword provides Nock execution as well as on-disk snapshot persistence with on-demand data paging. Persistence is handled automatically for Nock programmers.

The interface of the solid-state interpreter is small; we limit it to one core interaction: "compute this input against your current state." This will, in general, cause a new state to be persisted by the solid-state interpreter and return some result data to the module that began the interaction. We call this module the "King."

The King implements the networking necessary for global consensus of the blockchain state and the socket interface for user control of the node. It translates network packets and user commands into computations for Sword, and actuates Sword responses as network packets and user responses. This layering abstracts the details of peer-to-peer networking from the consensus logic of Nockchain. The King is the executable which is invoked to run Nockchain: it imports Sword internally as a library and instantiates Sword with the Nockchain kernel.

The Nock program interpreted by Sword is the Nockchain kernel. This kernel is a state machine that contains all of the "business logic" of Nockchain. It implements consensus, block and transaction validation, block creation (mining) and transaction creation (spending)—all as a function that takes the current state and a new input, and transforms them into a new state and descriptions of output effects.

Unlike most other blockchain implementations, we are able to directly represent large sets as data structures in our program even when these sets do not fit within working memory. Our algorithms are specified as pure functions operating over these sets, and persistence involves no manual IO at all. The solid-state interpreter removes all complications from abstractly representing the sets of blocks, notes (our term for unspent funds, called UTXOs), and transactions, allowing us to write direct and auditable code implementing all blockchain logic.

Our representation of the blockchain is several sets, together with the block identifier of the current heaviest block. We have a set of blocks, a set of transactions which may be referenced by blocks, and a set of balances. The block and transaction sets are indexed by block and transaction identifiers (respectively), which are simply merkle commitments to the block or transaction. A balance is the set of notes which are spendable at a particular block, so the set of balances is indexed by the block identifier. Each balance is itself a set of notes, which are indexed by note identifiers (merkle commitments to notes).

Most blockchains have a concept of a reorganization, commonly called a reorg. In general, local views of the current heaviest block may differ between miners and validating nodes. Thus, when a node hears a block, it may not have the node's own view of the current heaviest block for a parent. In most blockchain architectures, the node must rewind state mutations until it arrives at a common ancestor between its current heaviest block and the new heaviest block, then validate forward from that point.

In our architecture, every "new block" can be considered a reorg, usually with a depth of zero. If we hear a block whose parent we have not validated, we will request the parent recursively until we hear a block whose parent we have validated. This already-validated check is simply checking whether the parent block identifier is in our block set. Then, we can pop the stack of requested blocks and "play forward" the validation. Without rewind logic, our implementation is simpler, and a whole codepath with possible consensus-destroying bugs is eliminated.

In a standard blockchain architecture, we would have to maintain a working view of the set with regard to memory constraints on the host system and explicitly persist blocks, transactions, and notes. But with a solid-state interpreter, our entire consensus implementation is just a functional Nock program over immutable data structures. No IO is necessary for persistence and consensus logic is straightforward and easily auditable. Network IO is isolated from consensus logic: hearing a message on a network results in an evaluation of the function, and sending a message is the result of the output of the function.

Transaction Engine

Nockchain's transaction engine uses the Unspent Transaction Output Model (UTXO) for security, scalability, parallelism, and fungibility. In the UTXO model, assets are immutable—a UTXO can be created or destroyed but never *altered*. We refer to UTXOs as Nockchain Notes.

Smart contract functionality is achieved in Nockchain through the use of *recursive covenants*. A covenant is a predicate committed to by a Note that constrains the traits of its future outputs. A covenant scheme is *recursive* when its predicates may apply to the covenants of the future outputs of assets. Nockchain expands upon this scheme by giving each Note a commitment to a set of programs that must return true when passed the Note itself as an argument. We refer to these commitments as Smart Names. They additionally serve as unique identifiers which address notes inside a global hierarchical namespace of domains.

Subdomains at each level of this namespace place additively restrictive constraints on the notes hierarchically below them. We call this scheme a Predicate-Centric Namespace, which is a fully generalized superset of the already powerful Content-Centric paradigm. The immense benefits of predicate-centric naming have shaped many of our broader architectural decisions.

A primary benefit of Smart Names is an extra layer of safety for our recursive covenant scheme. One problem with recursive covenant schemes is that assets may be permanently encumbered with a set of constraints that are inescapable. This concern arises because existing address schemes only constrain how assets sent to a given address may be spent in the future, whereas Smart Name addresses enforce constraints on *the assets that are received*. This style of address scheme is beneficial for users who wish to authorize attempted transactions to their addresses.

Smart Names provide an extra dimension of expressivity and granularity for developers. Nockchain transactions are a handshake between the constraints placed on the transaction by the senders and receivers. This lends itself to expressive and batchable intent-type applications, since each output note of a transaction may place constraints upon itself. Smart Names also reduce application complexity, as arbitrary characteristics of the ledger state may be verified merely by checking for the existence of a given name in the namespace.

The use of a hierarchical predicate-centric namespace also has tremendous benefits for scalability and UX. Smart Names may be used to aggregate all of the notes owned by a given user underneath a single subdomain in the namespace. This convention naturally leads to an improved design for ZK-light-clients that blurs the distinction between light clients and full nodes. We call this Client-side Sharding. Client-side Sharding provides light clients with the same degree of sovereignty, or security guarantees, that full nodes enjoy.

Traditional ZK-light client systems provide an innovation upon previous light client designs, in that they enable trustless verification of the entire chain without a need to sync and process all chain data; users only need to store their particular notes of interest and to update their respective inclusion proofs for each block. However, traditional ZK-light clients depend upon Proof Serving Nodes to interact with the chain in any way. ZK-light clients rely on the goodwill of Proof Serving Nodes to query the location of their notes, generate their inclusion proofs, and serve them. In essence, ZK-light clients do not remove the tension between transaction throughput and

full-node accessibility, as they introduce a critical dependence upon Proof Serving Nodes to access their assets on-chain.

To our knowledge, Nockchain's architecture is the first to offer a solution to this problem that allows for truly lightweight full nodes. We treat each subdomain of the global namespace as a standalone chain shard. We term this Client-side Sharding because the underlying ledger is oblivious to the sharding behavior; sharding is handled entirely at the network layer through *hierarchical content-centric peering* (HCCP).

In an HCCP network, peers advertise the individual subdomains they wish to subscribe to, and only form network connections to peers with *intersecting advertised subdomains*. The resulting network topology provides peers with sufficient information to gossip the inclusion proofs necessary to connect subdomains up to the global state root. This architecture allows for proof serving to be the default method for gossiping state updates between *all* clients, and allows Nockchain to eliminate the two-tiered class system of light clients and proof serving nodes.

Blockchain Logic

The availability of a solid-state interpreter allows us to implement our blockchain logic as a Nock state machine, without any concern with IO for persistence. The state machine is invoked with a new block or transaction as input, and produces as output a new version of itself, along with a list of effects which it requests the runtime to actuate.

Our design permits us to use one code path for the three tasks or modes that a standard blockchain must accomplish:
- In a steady-state, a node is synchronizing and verifying the heaviest chain: hearing each new block from its peers as blocks are mined, and verifying that they properly extend the current heaviest chain.
- In a reorg, the node learns of a new branch of the chain with a heavier total weight, thus "orphaning" the branch it was verifying.
- When catching up, the node must as rapidly as possible receive the blocks making up its peers' view of the heaviest chain, and independently verify these blocks to achieve the consensus view of the chain.

Separate or partially separate codepaths for these modes or states opens an opportunity for consensus-breaking bugs. If verification results are different depending on whether a block is verified during steady state or reorg, for instance, then consensus could fracture between nodes which heard about different branches of the chain first.

Our solution is to maintain a set of verified blocks indexed by their block id, which is a hash commitment to the block. Upon hearing a new block header, we can rapidly verify this commitment, and then check whether we have already verified the block claimed as a parent. If we have already verified a block with this id, we can immediately return with no state update or effects.

If so, we can proceed to fully verify the block just heard, starting with its proof of work, and add it to the set of verified blocks. We test the weight of the chain terminating at this block against our current heaviest block, and if it is heavier, we will re-gossip the block to our peers and update our view of the current heaviest block.

In any case, we also emit an effect which requests from our peers the next block on their heaviest chain. If the just-verified block has height N, we request each peer's view of block N+1 on the heaviest chain. Peers respond either with that block or, if their heaviest chain does not contain N+1 blocks, with their current heaviest block.

If we do not have the parent for a block, we *will drop the block* and emit a request for its parent. Combined with the strategy of *always* requesting the next-heaviest block, this permits us to rapidly discover and verify the tip of the heaviest chain, without exposing nodes to resource-exhaustion attacks caused by the inability to verify that enqueued blocks have sufficient PoW.

On a timer (provided by our runtime), we again emit a request for block N+1 on the heaviest chain. In this case, N is the block height of our current heaviest block. This serves to ensure the liveness of consensus: if a block is not heard after being requested or gossiped, the timer serves as a re-request. If there is no update to consensus, every node will respond with the same heaviest block, which will cause no effects to be issued by the requesting node.

The steady-state and catchup cases are straightforward to trace, but the reorg case serves as a good example of how this logic

subsumes the multi-modal and imperative approach of most blockchain implementations.

In a reorg, the node hears about a new block through peer gossip, but it does not have the parent block. Therefore, it requests the parent from its peers by block ID, and *drops the gossiped block*. If it is missing several blocks in the ancestry of the original block, this process will repeat for the length of the reorganized branch, at which point the node will have the parent block and be able to verify the block returned for its request. Verification of this block will result in the request for the block at one greater height, which will result in the next block in the reorganized chain being returned, enabling rapid forward progress on verifying the new branch of the chain.

Note that this logic flow is applied to every block heard, and properly handles all possible states of the chain, so long as at least one peer is honestly responding to requests. For a node to respond to its request, it does not need to update the Nockchain kernel state machine: the runtime can simply examine the state and produce the proper response.

Proof-of-Work Puzzle Design

Proof-of-Work Puzzles are secure only if every miner must perform the same amount of computational work per attempt at finding the next valid block. If a clever miner figures out a way to reuse previous work to make one or more additional attempts at finding a block that can produce valid blocks, the security of the puzzle, and thus the consensus mechanism, is broken. Creating new types of secure puzzles is difficult.

In order to use a zkVM as a part of a secure PoW puzzle, the zkVM must satisfy a little-known special-soundness trait: Single-Witness Hardness. The formal definition of a ZKP is that given a function, an input, and an output, a witness may be generated. A witness is a transcript of each computational step of the function given the input leading to the output. In a ZKP, the witness allows the succinct verification of the specified computation. A ZKP is single-witness hard when for a particular input computation, there may only be one valid witness. If there is only one valid witness for a specified computation and the prover is simulation extractable, there may only be one valid proof of the computation. A prover is simulation extractable when each input has a unique proof representation.

The instantiation of the Nock zkVM used as the PoW puzzle for Nockchain is Single-Witness Hard. This means that for a particular input subject-formula pair, there is a single valid computational transcript, or witness, of said computation that can be feasibly computed. The Nock ISA is defined as a pure, deterministic function; this makes Single-Witness Hardness simpler to achieve without severe performance degradation. The Nock zkVM used for Nockchain is a standard DEEP-ALI STARK that uses AIR and FRI. It uses degree 4 constraints and has a security level over 100 bits. The prime field used within the witness is the well-known Goldilocks base field, chosen for its performance characteristics. The hash function used for Merkle tree commitments for building proofs is Tip5.