THE MAIN DOCS OF ETHERNOVA

Official long-form documentation — Draft revision, April 2026

Front matter

Title page

The Main Docs of Ethernova *Network · Protocol · Architecture · Applications* Maintained by the Ethernova Core Team and contributors. First public draft: **April 2026.**

Version and maintenance note

This documentation is a living manuscript. It describes the Ethernova network as it stands at the time of writing — late April 2026 — covering the active mainnet (Noven Fork at block 480,000, client v2.0.0 and above) and the architectural roadmap defined by Nova Improvement Proposal 4 (NIP-0004), of which Phases 1 through 3 are deployed on the public devnet.

if you look into the future, this document contains some things that are irrelevant

Some systems described here are not yet fully available on every network. In particular, large parts of the architecture defined in NIP-0004 — Protocol Channels, the full five-tier State Lifecycle, Multi-Dimensional Resource Metering, the Mailbox / Session / Identity / Subscription / GameRoom Protocol Objects, and the application primitives (email, chat, social, content manifests, games) — are partially deployed on the devnet, devnet-only, or still in active design. If you are reading this version of the document after those systems have shipped to mainnet, the relevant chapters should be read in light of the implemented architecture rather than the date on this page. **When in doubt, prefer the NIPs and the active repositories over any older public copy on the project website.** The website is a marketing surface and is not always synchronized with the latest protocol documents.

marketing surface and is not always synchronized with the latest protocol documents.

Reference URLs:

- Mainnet: https://rpc.ethnova.net · explorer https://explorer.ethnova.net · chain ID 121525
- Devnet: https://devrpc.ethnova.net · explorer https://devexplorer.ethnova.net · faucet https://faucet.ethnova.net · chain ID 121526
- Core client: github.com/EthernovaDev/ethernova-coregeth
- Devnet client: github.com/EthernovaDev/ethernova-devnet

Status legend

To keep promises and reality clearly separated, every protocol feature in this document is tagged with one of the following statuses. Where status varies across networks, both tags appear.

- **[Mainnet Live]** — Implemented, audited as far as our review process allows, and active on chain ID 121525 at or after block 480,000.
- **[Devnet Live]** — Implemented and active on chain ID 121526 (the public adaptive-EVM devnet). Not yet active on mainnet.
- **[Specified / Planned]** — Defined in a NIP or design document, but not yet implemented in either network.
- **[Experimental]** — Implementation exists but is not consensus-safe yet, or is gated behind a flag and used only for measurement.
- **[Outdated Public Copy]** — Marketing or older README text that has been superseded by a higher-priority source. Where this label appears, the document explains the discrepancy.

How to read these docs

The document is layered. Read it top-down as much as your interest demands.

If you are a curious user, the first three parts will give you a complete and honest picture without any technical commitment. If you are a product-minded builder, parts III and IV are the practical core. If you are a Solidity or EVM developer, part IV plus part V will tell you exactly what is familiar, what is new, and what is still missing. If you are a protocol engineer or you are evaluating Ethernova at architectural depth, part V into part VI is where the substance lives, and parts VII and VIII give the strategic and risk picture.

The appendices are intentionally short and act as quick references after a first reading.

Part I — Orientation

1. What is Ethernova?

Ethernova is a Proof-of-Work, EVM-compatible blockchain whose long-term goal is to be a *useful* decentralized computer for human applications — identity, communication, content, ownership, social interaction, and games — and not merely a settlement layer for financial assets.

In its present form (April 2026), Ethernova is a CoreGeth fork written in Go, secured by Ethash mining, with chain ID 121525, the native token NOVA, a target block time near eleven seconds, and a 10-NOVA block reward. It runs Solidity contracts unchanged, accepts MetaMask, integrates with Hardhat and Foundry, and exposes a standard eth_* JSON-RPC surface — but it also extends the EVM with nine native precompiles, a deterministic adaptive gas system, native account abstraction, an anti-MEV ordering rule, a state expiry mechanism, and a small set of features designed for human usability (smart-wallet recovery, gas refund on revert, batched and scheduled transactions).

These improvements landed together at mainnet block **480,000** as the **Noven Fork** (NIP-0002), released as client v2.0.0. This is the foundation Ethernova builds on. It is not the destination.

The destination is described in **NIP-0004**: a *Layered Deterministic Computer* with explicit notions of Protocol Objects, Deferred Execution, Protocol Channels, a tiered State Lifecycle, and multi-dimensional resource metering. Phases 1 through 3 of NIP-0004 are already deployed on the devnet. The rest is still in design and implementation. Ethernova in April 2026 is therefore best understood as a working EVM chain with one major fork shipped and one major architectural evolution under way — not as a finished product.

2. Ethernova in one minute

Ethernova looks like Ethereum if you only inspect it through MetaMask, Hardhat, or eth_chainId. Underneath that compatibility, it tries to fix five problems that Ethereum has not, in practice, fixed at Layer 1: state growth, MEV extraction, reentrancy as a class of bug, the punishing gas cost of failed transactions, and a still-debated account-abstraction story. In addition, Ethernova prices contracts adaptively — predominantly pure code receives a discount, storage-heavy code pays a small surcharge — using a deterministic post-execution adjustment that is identical on every node.

That is the present. In the planned next-generation architecture, Ethernova adds primitives that Ethereum simply does not have at the protocol level: mailboxes, content references, deferred message delivery, and bilateral state channels with a built-in arbitration scheme. The intent is that decentralized email, chat, social, and lightweight games become natural applications on Ethernova rather than awkward dApp simulations.

3. Why Ethernova exists

Public blockchains have, for over a decade, optimized almost exclusively for financial assets. The vocabulary is the giveaway: tokens, swaps, liquidity, vaults, yields, MEV. Even the phrase "world computer" is, in practice, used to describe a settlement layer for trades.

Ethernova starts from a different observation. The networks people actually use every day — email, messaging, social media, games, and the web itself — depend on properties that the standard EVM does not provide cheaply or naturally: asynchronous communication, sessions,

mailboxes, content addressing, real-time interaction, and bounded resource costs at the node. Building those experiences on top of Ethereum requires either bolting on heavy off-chain infrastructure or accepting block-time latency and gas costs that ordinary users will not tolerate.

Ethernova exists to ask, and answer with code, a narrower question: *can a deterministic, decentralized chain offer the primitives needed for human-scale software, without sacrificing the EVM ecosystem developers already understand?*

The answer the project commits to is "yes, in layers." Layer one (mainnet today) is a hardened, EVM-compatible PoW chain with the most-needed quality-of-life upgrades pulled forward. Layer two and beyond (NIP-0004) is the architectural evolution that turns those upgrades into application primitives.

4. The core narrative

The narrative of Ethernova is best read as three claims, in order:

First, **EVM compatibility is non-negotiable.** Every feature added to Ethernova must coexist with unmodified Solidity contracts and with the existing tool ecosystem. A contract that compiles and deploys on Ethereum must compile and deploy on Ethernova. A wallet that signs an Ethereum transaction must sign an Ethernova transaction. There is no value in winning an architecture argument by losing the developer base.

Second, **the EVM as it stands is not enough.** Real-world software needs delivery, not just settlement; sessions, not just calls; bounded growth, not just gas. Adding a hundred more opcodes to a synchronous, atomic, monolithic VM does not change those facts. The system has to grow new layers — message queues, channels, content references, and a state lifecycle — and those layers have to be defined by the protocol, not improvised by every dApp.

Third, **everything must remain deterministic.** Whatever Ethernova adds, every full node must still be able to reach the same state hash from the same input. Determinism is the line that separates a decentralized computer from a hosted service. The hardest engineering work on the project so far has been the determinism work: removing node-local profiling from gas math, replacing Go map iteration in consensus paths with sorted iteration, moving expiry metadata out of the state trie into an external index, and retiring features that proved consensus-unsafe.

These three commitments — compatibility, expansion, determinism — are the spine of the whole project. The Noven Fork honors all three on a small set of features. NIP-0004 attempts to honor them on a much larger one.

5. What Ethernova is trying to solve

The Noven Fork (mainnet today) addresses the following problems, each of which has a long history on Ethereum:

- **State growth.** Ethereum's state has crossed 200 GB and continues to grow without a built-in mechanism to shed unused contract storage. Ethernova introduces a two-part response: contracts inactive for 900,000 blocks (~115 days) on mainnet are archived into a slim record, and storage demotion lives entirely in an external index outside the consensus state trie.
- **MEV extraction.** Reordering transactions for profit has cost users an estimated nine-figure sum on Ethereum. Ethernova orders transactions by arrival time at a node, not by gas price, and rate-limits each sender to sixteen pending transactions to prevent FIFO spam.
- **Reentrancy as a class of bug.** From The DAO onward, reentrancy has been a major source of catastrophic loss. Ethernova adds a per-EVM reentrancy guard at the protocol level: a contract cannot reenter itself, but legitimate cross-contract patterns (A→B→C) still work.
- **Money lost on failed transactions.** On Ethereum, a transaction that reverts still consumes the full gas it used. Ethernova refunds 90% of execution gas on revert, with an anti-DoS cap so the rebate only applies to transactions that used less than 100,000 execution gas in total.
- **Account abstraction debt.** Ethereum has been discussing AA for the better part of a decade. Ethernova ships three converging pieces: a smart-wallet precompile with guardian recovery and key rotation; Tempo-style atomic batched, fee-delegated, and scheduled transactions; and Frame-style approvals with cross-frame introspection.

The next-generation architecture (NIP-0004) addresses a different and broader set of problems:

- The lack of asynchronous communication primitives.
- The lack of content addressing at the protocol level.
- The lack of any execution model beyond synchronous-atomic.
- The lack of an off-chain low-latency interaction protocol with deterministic on-chain arbitration.
- The single-dimensional gas model, which causes any one form of activity (a busy DEX, a flood of messages, a heavy oracle update) to spill congestion onto every other form of activity.

Each of these is hard. None is solved by a single opcode.

6. What Ethernova is not trying to be

It is worth saying clearly what Ethernova is not pursuing, because the public marketing surface can read more ambitiously than the actual roadmap.

Ethernova is **not** a Proof-of-Stake chain. PoW with Ethash is part of the project's identity; the design goals explicitly preserve Ethash and an ~11-second block time. Ethernova is **not** a Layer 2 of Ethereum, nor is it a chain that intends to migrate to one — its scaling answer is Layer-1 evolution, not bridges. Ethernova is **not** a high-frequency trading venue; the deliberate FIFO ordering and rate limiting actively work against the conditions HFT requires. Ethernova is **not** a privacy chain; privacy is **opt-in** through the shielded pool precompile, not the default mode.

Most importantly, Ethernova is not — even in the NIP-0004 vision — a general-purpose web browser. The "browser-like" content manifests described in NIP-0004 are a verifiable content-delivery system with an explicit permission model and a deterministic JavaScript-safe sandbox. They are closer to "static site plus on-chain interaction" than to a full web app. Promising more than that would be promising something the protocol cannot honestly deliver.

Part II — High-level understanding

7. Ethernova vs Ethereum: the fundamental difference

The fundamental difference is one of intent. Ethereum, as a settlement layer, is optimized to accept transactions, order them inside blocks, execute them atomically, and write the results into a single global state. Everything that does not fit that model — chat, email, social feeds, real-time games, long-lived sessions, low-latency messaging — is pushed off-chain into infrastructure the protocol has no opinions about.

Ethernova accepts that settlement is necessary, but rejects the conclusion that everything else must be invisible to the protocol. It treats application primitives — mailboxes, content references, sessions, identity attestations, social-graph edges — as first-class concepts the chain itself can name, validate, price, and arbitrate. Ethereum has tokens at the protocol level only as accidental ETH transfers; everything else is a contract pretending to be a token. NIP-0004 proposes that mailboxes, sessions, content references, identities, subscriptions, and game rooms exist as **Protocol Objects** in a dedicated trie alongside accounts, with their own lifecycles and their own pricing dimension. This is not a cosmetic change. It is a different theory of what the chain is for.

In the present implementation, that distinction is partial. The Noven Fork has already added native tokens, native oracle, and shielded transfers as protocol-managed concepts via precompiles; the NIP-0004 phases now landing on the devnet add the generic Protocol Object Registry and the Deferred Queue. The full picture — channels, capabilities, multi-dimensional metering — is still ahead.

8. Why EVM compatibility still matters

Compatibility is not nostalgia. The EVM has accumulated a developer base, a library ecosystem (OpenZeppelin, ethers, viem, Hardhat, Foundry, slither, mythril), wallet infrastructure (MetaMask, WalletConnect, hardware-wallet support), and a set of mental models that took years to standardize. Throwing all of that away for a clean-room VM imposes a permanent migration cost on every developer and every user.

Ethernova's commitment is therefore strict: any Solidity contract valid on Ethereum **must** deploy and execute on Ethernova without modification. This is an absolute principle in NIP-0004 and is enforced by the design of the Noven Fork (100% backwards compatibility, with block-hash equality verified from block 1 to block 445,000 between the v1.3.x and v2.0.0 client lines). Standard eth_* RPC methods continue to function unchanged. New surface area is added under a separate nova_* (or ethernova_* on the devnet) namespace, leaving the old surface intact.

The practical consequence is that every step up the Ethernova feature ladder is opt-in. A team that wants to ship an ERC-20, an NFT collection, a multi-sig wallet, or a DEX can do so today with their existing tooling and never touch a single Ethernova-specific feature. A team that wants the gas discount on pure code, the gas refund on revert, the anti-MEV ordering, or the precompiles for batched hashing and signature verification gets those benefits *automatically* without recompiling. Only the deeper integrations — Frame approvals, native tokens, the shielded pool, and (eventually) NIP-0004 Protocol Objects — require the developer to call new precompiles or, in the future, declare a non-default execution domain.

9. Where Ethernova is stronger than Ethereum

Ethernova's strengths today are concrete and measurable. They are not all dramatic, but they add up.

For end users, transactions that revert refund 90% of execution gas, which removes one of the most-resented experiences in DeFi: paying full price for a failed swap. Account-abstraction features are already shipped — guardian-based recovery, key rotation without changing the address, atomic multi-call transactions, fee delegation so a dApp can pay gas in NOVA on behalf of a new user, and scheduled transactions valid only after a future block or before an expiry block. Front-running is not impossible (no chain can fully eliminate it without sacrificing throughput) but the mempool's FIFO arrival-time ordering removes the gas-bidding war that produces sandwich attacks on AMMs.

For developers, the precompiles at 0x20 through 0x28 collapse common operations — batch keccak, batch signature verify, multi-token transfers, contract upgrades with timelock, oracle reads — into single-call primitives that cost less and remove a lot of boilerplate. The native multi-token precompile (0x25) in particular avoids the ERC-20 approve-then-transfer pattern that has been the source of countless phishing exploits. Adaptive Gas v2 silently rewards

contracts that do mostly pure work with a discount of up to 25%, while penalizing contracts that are storage-heavy (5 or more SSTOREs per call) by up to 10% — a small economic nudge that costs nothing to opt into.

For node operators, the State Expiry v2 mechanism uses an external LevelDB index rather than touching the state trie, which means abandoned contract state can be archived without changing the consensus state root and without producing the Windows/Linux RLP divergence that broke earlier attempts. EOAs are never expired. The expiration period on mainnet is 900,000 blocks (~115 days); on the devnet it is 1,000 blocks for testing speed.

For the protocol itself, every feature in the Noven Fork was iterated through public devnet phases, with five consensus bugs found and fixed before mainnet — a discipline that the rest of the Ethereum ecosystem rarely gets to apply at Layer 1.

10. Where Ethereum is still stronger today

Honesty here is more useful than enthusiasm. Ethereum is, in April 2026, still stronger than Ethernova on every axis that depends on scale, ecosystem maturity, and time.

Total liquidity, total developer count, total tooling polish, the variety of stablecoins and oracles, the depth of audited library code, the number of CEX listings, and the size of the validator/miner set are not even close. Ethernova has two listings (Gatevia and KlingEx as published on the website), a small set of bootstrap nodes, and a community organized largely around Telegram and Discord. The block explorer is functional but does not yet have the same indexing depth as Etherscan. There is no equivalent of Dune, no comparable subgraph ecosystem, no widely-deployed cross-chain bridge.

In the L2 era, Ethereum's effective throughput, when L2s are included, dwarfs anything a single PoW Layer-1 can deliver in absolute numbers. The devnet's measured 14.7 TPS in a 1,000-mixed-transaction stress test is real, but it is not a competitive number against an L2 rollup. Ethernova's strategy is not to win on throughput — it is to win on *fit* for application categories where settlement throughput is not the bottleneck.

Ethereum is also, by virtue of its size, harder to censor and more battle-tested under adversarial conditions. No PoW chain at Ethernova's current size can claim equivalent resilience. New users should size their exposure accordingly.

11. Honest trade-offs and complexity costs

Every architectural choice in this document has a cost. The most expensive choices are concentrated in NIP-0004, but they are worth naming up front because they shape how the rest of the document should be read.

Adding Protocol Objects, Deferred Execution, a Channel Layer, a five-tier State Lifecycle, and Multi-Dimensional Resource Metering increases the surface area of the consensus implementation by an estimated 3 to 4 times. Every additional layer is an additional source

of potential consensus splits if two implementations differ by a single edge case. The devnet has already produced two such splits — first when adaptive gas modified gas costs from per-node profiling (v1.0.1), then when state-expiry sweeping iterated a Go map in non-deterministic order (v1.0.7). Both were caught before mainnet, but the lesson generalizes: the more layers, the more places a tiny bug becomes a chain split.

Auditability scales similarly. Twelve precompiles already cover a lot of code paths; the full NIP-0004 set extends that to twenty-one, plus a Deferred Processing Engine, a Channel Arbiter, a Witness Verifier, and a Multi-Dimensional Metering subsystem. This is, frankly, a wide attack surface.

Node operator burden also rises. Today an Ethernova node behaves much like a Geth/CoreGeth node. With NIP-0004 active, operators must additionally understand state tiers, witness serving, channel relay, deferred-effect ordering, and per-dimension block limits. Hardware requirements remain modest (8–16 GB RAM, under 1 TB disk), but the operational mental model is heavier.

The compensating bet is that the layered architecture is also incrementally valuable: Mailbox and Deferred Execution alone enable decentralized email and notifications without any of the rest; Channel Layer alone enables real-time DMs and games; multi-dimensional metering alone improves fairness even before any new primitives are in use. Domain 0 — pure Ethereum compatibility — remains an always-safe fallback: if every new layer underperforms or fails to find users, Ethernova still functions as a hardened EVM chain with a useful Noven Fork.

12. Live vs in-progress (April 2026)

Below is the consolidated implementation map. It is the same map used throughout the rest of the document, summarized here for orientation.

Capability	Mainnet	Status Devnet
Ethash PoW, ~11 s blocks, 10 NOVA reward	Mainnet Live	Devnet Live (~5 s in practice)
Adaptive Gas v2 (deterministic, trace-based)	Mainnet Live	Devnet Live (v1.1.6+)
Per-EVM reentrancy guard	Mainnet Live	Devnet Live
90% gas refund on revert (≤100k execution gas)	Mainnet Live	Devnet Live
State Expiry v2 (external index, 900k blocks mainnet)	Mainnet Live	Devnet Live (1,000-block test threshold)
Tempo transactions (batch, fee-delegate, schedule)	Mainnet Live	Devnet Live
Frame AA (precompiles 0x23, 0x24)	Mainnet Live	Devnet Live

Anti-MEV FIFO ordering, 16 tx/sender	Mainnet Live	Devnet Live
Native multi-token (0x25)	Mainnet Live	Devnet Live
Shielded pool (0x26)	Mainnet Live	Devnet Live
Native contract upgrade with timelock (0x27)	Mainnet Live	Devnet Live
Native oracle with TWAP (0x28)	Mainnet Live	Devnet Live
Parallel transaction analysis (no execution change)	Mainnet Live	Devnet Live
novaProtocolObjectRegistry (0x29)	Specified / Planned	Devnet Live (NIP-0004 Phase 1)
novaDeferredQueue (0x2A)	Specified / Planned	Devnet Live (NIP-0004 Phase 2)
novaContentRegistry (0x2B)	Specified / Planned	Devnet Live (NIP-0004 Phase 3)
Mailbox / Session / Identity / Subscription / GameRoom Protocol Objects	Specified / Planned	Partial; full lifecycle pending
Nova opcodes (MSEND, SOPEN, etc.)	Specified / Planned	Specified / Planned
Protocol Channels	Specified / Planned	Specified / Planned
Capability-based call model	Specified / Planned	Specified / Planned
Multi-Dimensional Resource Metering	Specified / Planned	Specified / Planned
5-tier State Lifecycle (full)	Specified / Planned	Specified / Planned
Application primitives (email, chat, social, games, manifests)	Specified / Planned	Specified / Planned

The website's "24 phases shipped" framing covers the first block of rows here (mainnet-live capabilities). It does not cover NIP-0004, which is a separate, in-progress effort.

13. Who Ethernova is for

For users, Ethernova is most attractive if you value: PoW mining, EVM compatibility on a chain that takes user-experience problems (failed transactions, MEV, smart-wallet recovery, scheduled payments) seriously at Layer 1, and you are willing to accept that the ecosystem is still small and the price of NOVA is correspondingly volatile. It is less attractive if you need deep liquidity in a specific asset, fiat on-ramps that already exist, or a large network of dApps you can pick from on day one.

For developers, Ethernova is attractive if you write Solidity and want to keep doing so, and if the features in the Noven Fork (gas discount on pure code, refund on revert, batched

precompiles, native AA) materially improve your application or your users' experience. It is also attractive if you are interested in being early to a chain that is actually trying to add application primitives at the protocol level, and you are comfortable building on a smaller network. It is less attractive if you need every existing Ethereum tool to work without any rough edges, or if your application depends on liquidity Ethernova does not yet have.

For miners and node operators, Ethernova is attractive if you specifically want a profitable PoW chain with a Windows-friendly client and a clear development direction. It is less attractive if you prefer the larger, lower-margin ETC/ETH-PoW pools.

For protocol engineers and researchers, Ethernova is attractive if you find the engineering problem (preserving EVM compatibility while adding asynchronous primitives, channels, content addressing, and a state lifecycle) interesting on its own terms and want to read or contribute to a NIP that takes those problems seriously.

Part III — Practical user-facing overview

14. What people can do today

Concretely, on Ethernova mainnet today, a user can:

- **Hold and transfer NOVA** between addresses, with the same wallet UX as Ethereum (MetaMask, hardware wallets, mobile wallets that support custom EVM networks).
- **Use ERC-20 tokens and NFTs** deployed by the community on the Noven Fork. Standard contracts compile and run unchanged.
- **Use native tokens** issued through novaTokenManager (precompile 0x25), which require no approve step and transfer for roughly an order of magnitude less gas than ERC-20 transfers.
- **Recover a smart wallet through guardians** if they lose access. The novaAccountManager precompile (0x22) supports a guardian set with threshold voting, a 100-block timelock, and key rotation that does not change the address.
- **Send batched transactions** — for example, an approve-then-swap collapsed into a single atomic Tempo transaction (transaction type 0x04, up to 16 calls, all-or-nothing).
- **Have a dApp pay gas on their behalf** through Tempo fee delegation, while the gas is still denominated in NOVA.
- **Schedule transactions** to be valid only after, only before, or within a window of block heights — useful for limit orders, recurring payments, or AI-agent automation.

- **Use the optional shielded pool** through novaShieldedPool (0x26) for private NOVA transfers, with a per-withdrawal cap of 10,000 NOVA and a circuit breaker against pool-accounting bugs.
- **Trade NOVA on Gatevia and KlingEx**, and use exchange-listed pairs to enter or exit the network.
- **Mine NOVA** with any Ethash-capable GPU.
- **Run a node** on Windows or Linux, using the official client from EthernovaDev/ethernova-coregeth.

On the public devnet (chain ID 121526), a user can additionally:

- **Get free test NOVA** from https://faucet.ethnova.net (10 NOVA per request, 5-minute cooldown).
- **Interact with the deployed test contracts** (NovaToken at 0xd6Dc5b3E…, NovaNFT at 0xa407ABC4…, NovaMultiSig at 0x24fcDc40…) to see Adaptive Gas v2 in action and observe the discount applied to predominantly pure code.
- **Exercise the NIP-0004 Phase 1–3 precompiles** (novaProtocolObjectRegistry at 0x29, novaDeferredQueue at 0x2A, novaContentRegistry at 0x2B), all of which are devnet-only.

15. Network basics: wallet, RPC, explorer

Mainnet MetaMask configuration:

- **Network Name:** Ethernova
- **New RPC URL:** https://rpc.ethnova.net (alternative: https://nova-rpc.xbinodes.com, hosted by partner XBiNodes)
- **Chain ID:** 121525 (0x1dab5)
- **Currency Symbol:** NOVA
- **Block Explorer URL:** https://explorer.ethnova.net

Devnet MetaMask configuration:

- **Network Name:** Ethernova Devnet
- **New RPC URL:** https://devrpc.ethnova.net
- **Chain ID:** 121526
- **Currency Symbol:** NOVA
- **Block Explorer URL:** https://devexplorer.ethnova.net

The default local RPC is http://127.0.0.1:8545 for HTTP and ws://127.0.0.1:8546 for WebSocket. The devolopment-time JSON-RPC namespaces beyond the standard eth_*, net_*, web3_* are listed in part IV; on the devnet the namespace is ethernova_*, and on mainnet (per NIP-0004) the future Protocol Object queries will be exposed under nova_*.

16. A basic user journey

A new user's first hour on Ethernova typically looks like this:

1. Install MetaMask (or a comparable EVM wallet) and add the Ethernova network using the configuration above.
2. Acquire NOVA. On mainnet, this means buying it on Gatevia or KlingEx and withdrawing to the wallet address. On devnet, this means visiting the faucet and requesting 10 test NOVA.
3. Confirm the balance appears in MetaMask and the explorer shows the funding transaction.
4. (Optional, mainnet) Set up guardians on the smart-wallet precompile to make the account recoverable. This is a one-time call to novaAccountManager declaring 1–10 guardian addresses and a threshold.
5. Try a Tempo batched transaction by performing two related actions (e.g., a token approve and a transfer) in one all-or-nothing call.
6. (Optional) Try a scheduled transaction with validBefore set to a future block as a limit-order primitive.
7. Read the explorer at https://explorer.ethnova.net, which shows blocks, transactions, gas used, and adaptive-gas effects per contract.

For users coming from Ethereum, almost everything in this list — wallet setup, signing, exploring, transferring — feels identical to Ethereum. The unfamiliar parts are the recovery guardian step, the Tempo batch (a single signed transaction containing multiple calls), and, eventually, scheduled execution.

17. Application categories that fit

Ethernova is well-suited to applications where deterministic settlement, predictable fees on pure code, and protocol-managed primitives are advantages, and where ultra-high throughput is not the primary requirement.

Communication and notifications. Once novaDeferredQueue (0x2A) on the devnet matures into the full Mailbox / Session set on mainnet, decentralized email-style and notification-style applications become possible without trusted servers. Today, even before the full primitives ship, contracts can already use the deferred queue to schedule effects across blocks deterministically.

Identity and ownership. The shape of Ethernova's roadmap (Identity Object, content references, attestations) is built for use cases where the chain itself records *who claims what*: profile bindings, key rotations, attestations, content authorship.

Lightweight payment and subscription flows. Tempo fee delegation removes the "new user must own NOVA before doing anything" problem, and scheduled transactions cover recurring payments without requiring a centralized scheduler. This makes per-use-case payments and subscription primitives easier than on Ethereum.

DeFi that does not depend on flash-extracted MEV. Standard ERC-20 DEXes, lending markets, vaults, and yield aggregators all work on Ethernova unchanged. The FIFO mempool removes some forms of MEV but does not, by itself, eliminate adversarial trading; it does change the strategic landscape, generally in users' favor.

Games and turn-based interactions. With the future novaGameState precompile (0x31) and Game Room Protocol Objects, on-chain games can rely on commit-reveal randomness, compact state diffs, and a deterministic settlement path. Today, games can be built on the existing EVM with the help of native tokens for in-game assets.

Content-anchored social. novaContentRegistry (0x2B) on the devnet already supports content-hash + size + MIME-type registration with optional rent-backed expiry, which is the spine of any social application that wants on-chain integrity guarantees over off-chain payloads.

18. Application categories that do not fit

Ethernova is not the right home for high-frequency trading or any workflow that depends on winning on gas-price priority. The FIFO ordering rule deliberately removes that lever. Ethernova is also not ideal for applications that need very low latency on every interaction — block time near 11 seconds is not friendly to a use case that demands sub-second confirmation. The future Channel Layer in NIP-0004 is the answer to that limitation, but until it ships, Ethernova should be assumed to have block-time latency for any settled action.

Ethernova is not currently a strong fit for stablecoin-heavy DeFi, simply because the stablecoin presence on the chain is small. A team that needs a deep USDC/USDT pool at launch will be disappointed.

Ethernova is not a fit for applications that require massive raw computation — large-scale ML inference, video transcoding, anything that would saturate the compute dimension. Layer-1 chains, including this one, are not the right substrate. NIP-0004 anticipates this with a novaComputeBounty precompile (planned, 0x32) for verifiable off-chain computation, but the principle is unchanged: the chain is for *commitment* and *verification*, not for the computation itself.

Finally, Ethernova is not a privacy-by-default chain. The shielded pool is a tool the user opts into, not a default behaviour. Applications that require always-on privacy will find Ethernova less suitable than a purpose-built privacy chain.

19. Demo and starter app paths

A team looking for a first production-shaped app on Ethernova can pick from a small menu of realistic starting points. Each is sized to be buildable by a single developer or small team and each plays to one or two of the chain's distinctive strengths.

A **gas-aware AMM** that uses novaBatchHash and novaBatchVerify to compress per-block bookkeeping, written in Solidity against the existing EVM. This is the closest thing to a pure-DeFi starter and ships entirely with current mainnet features.

A **recoverable wallet onboarding flow** that uses novaAccountManager for guardian recovery and Tempo fee delegation so that a new user can interact without owning NOVA. This is the most user-visible advantage of the chain.

A **scheduled-payment / standing-order app** built on Tempo's validAfter / validBefore fields. Unlike Ethereum equivalents, no third-party relayer is required.

A **token-launch toolkit** that uses novaTokenManager (0x25) instead of ERC-20, removing the approve-then-transfer phishing surface and giving users a cheaper transfer path. This is a category where Ethernova has a real, today-shipped advantage.

A **devnet-only experiment** built against novaContentRegistry (0x2B) that registers content references for an off-chain payload (image, document, JSON manifest), then verifies them inside a Solidity contract. This is the first realistic dApp shape that could later become a social, publishing, or content-attestation app once the rest of NIP-0004 ships.

Part IV — Developer onboarding

20. The developer mental model

The simplest way to think about Ethernova as a developer is in three rings.

The **inner ring** is pure Ethereum. Solidity, Hardhat, Foundry, ethers, viem, MetaMask, all the patterns you already know. Deploy a contract, call a contract, listen to events. On Ethernova this works exactly as it does on Ethereum, with three silent improvements applied automatically: the per-EVM reentrancy guard is on by default, the 90% gas refund on revert is on for any transaction whose execution gas stays under 100,000, and Adaptive Gas v2 will quietly discount or surcharge your contract based on the ratio of pure to storage-write opcodes once it has accumulated enough samples (10 calls, 100 opcodes minimum). You do not need to write any new code to receive any of those benefits.

The **middle ring** is "EVM with Ethernova-aware features." Here you call the precompiles 0x20 through 0x28 directly from Solidity using the standard address.call() and address.staticcall() patterns. You also start using Tempo and Frame transaction types — submitted through the JSON-RPC layer, not through new Solidity syntax — to get atomic batching, fee delegation, scheduling, and contract-validated approvals. Most of this ring is pure-Solidity-from-Ethereum's-perspective, plus one new transaction-format option at the wallet/RPC layer. No compiler change is required.

The **outer ring** is NIP-0004. This is the future ring. It introduces explicit Execution Domains declared at deployment, new opcodes (MSEND, MRECV, MPEEK, MCOUNT, CREF, CVERIFY, SOPEN, SCOMMIT, SCLOSE), a capability bitmask propagated through call chains, Protocol Objects with their own address-space prefix (0xFF) and lifecycle, and a five-dimensional Resource Vector instead of a single gasLimit. This ring is partially live on the devnet (Phase 1–3 precompiles only) and otherwise still in design. When it lands, it will require either compiler-plugin support (for domain bytecode prefix and opcode emission) or, more commonly, library-level abstractions in a forthcoming @ethernova/* SDK.

A developer can stay in the inner ring forever and ship useful applications. The middle ring is where Ethernova's present advantage actually shows. The outer ring is where its long-term differentiation lives.

21. Tooling compatibility

The matrix below summarizes the practical state of the developer toolchain in April 2026.

Tool	Status today (Mainnet, Noven Fork)	Notes
Solidity (any 0.8.x)	Works unchanged	Compile with --evm-version istanbul if you target both networks; the devnet currently does not include the Shanghai PUSH0 opcode.
Hardhat	Works for standard contracts	A ready-to-use hardhat.config.js ships in ethernova-devnet/devnet/. The same shape works for mainnet by switching the RPC URL and chain ID.
Foundry	Works for standard contracts	No Ethernova-specific configuration required for inner-ring code.
MetaMask	Works	Add Ethernova as a custom network with the values in

		§15. Tempo / Frame transaction types are not natively rendered in MetaMask's UI yet; they are submitted through application code via eth_sendRawTransaction.
ethers.js / viem / web3.js	Works	All eth_* methods are unchanged. To call precompiles, use provider.call({ to: '0x20...', data: ... }) like any other contract.
Block explorer	Works	Standard Etherscan-shaped explorer at explorer.ethnova.net (mainnet) and devexplorer.ethnova.net (devnet). Protocol Object views and channel-state views will require an explorer extension when those primitives ship.
Slither / Mythril / static analyzers	Works on Domain 0	Custom precompiles will be reported as unknown addresses; this is cosmetic.
OpenZeppelin contracts	Works	The native reentrancy guard makes ReentrancyGuard redundant for internal protection (it remains useful for cross-contract guards).

Where the tooling is *not* ready yet: there is no public Hardhat plugin specific to Ethernova at the time of writing, no Foundry cheatcode for Tempo transactions, no MetaMask extension that natively renders Tempo or Frame transactions, and no @ethernova/* npm package for NIP-0004 primitives. Each of these gaps is a tractable contribution opportunity.

22. What works out of the box

Without writing any Ethernova-specific code, a developer's existing Solidity contracts get:

- The per-EVM reentrancy guard. A contract calling itself reentrantly will revert at the EVM level. Cross-contract reentrancy (A → B → C) is not blocked, because blocking it would break common patterns (DEX aggregators, oracle pulls, flash-loan composability).

- The 90% gas refund on revert, when execution gas stays under 100,000. The user only effectively pays the 21,000-gas base transaction cost plus a 10% anti-spam penalty on the executed work.
- Adaptive Gas v2, applied post-execution. The discount or surcharge is computed deterministically from the trace counters using computeStablePenaltyScore(). SLOAD and JUMPI are excluded from the penalty calculation so that the score does not depend on which code path executed. The result is integer-only, identical on every node, and verified to be cross-platform deterministic between Linux (CGO enabled) and Windows (CGO disabled) builds.
- The anti-MEV FIFO ordering at the mempool layer. This is invisible to contracts but changes the trading game theory.
- Free access to all 9 precompiles at 0x20 through 0x28 by simple address-call from Solidity.

23. What requires Ethernova-specific adaptation

Three categories of feature require deliberate use:

Tempo transactions require the application or wallet to construct a transaction-type-0x04 payload with a calls array, an optional feePayer co-signer, and optional validBefore / validAfter block bounds. This is a new transaction format at the RPC layer — the contracts being called are unaffected — and is currently submitted through eth_sendRawTransaction with the appropriate envelope.

Frame Account Abstraction requires an account contract (or an EOA wrapper contract) that calls the novaFrameApprove precompile (0x23) to declare approval for sending and/or paying gas on a transaction, and optionally calls novaFrameIntrospect (0x24) to inspect other frames in the same transaction (target, calldata hash, value, gas limit, function selector). This is how smart-contract wallets, conditional sponsorship, mixer approvals, and AI-agent automation are built on Ethernova without breaking existing tooling.

Native tokens through novaTokenManager (0x25) replace the ERC-20 pattern. Token issuance, transfer, and balance queries become precompile calls. The cost is roughly 5,000 gas per transfer compared to ERC-20's typical 50,000 gas, and the approve-then-transferFrom pattern is replaced by direct authorization, removing the phishing surface that ERC-20 has carried for a decade. Existing ERC-20 contracts continue to work; native tokens are an alternative, not a replacement.

24. What is still missing

There are real gaps a developer should plan around.

There is no widely-used Solidity library for the precompiles yet — each project tends to write its own thin wrapper around address(0x20).staticcall(...) and friends. A clean wrapper library is a high-leverage contribution.

There is no Hardhat or Foundry plugin for Tempo or Frame transactions, so test harnesses currently have to construct those payloads by hand or use the ethernova-devnet repository's test scripts as a reference.

There is no @ethernova/sdk for client-side code — applications wanting to interact with the deployed Phase 1–3 NIP-0004 precompiles on the devnet hand-roll the calls.

There is no Subgraph/Goldsky/Dune-equivalent indexer for Ethernova at the time of writing. Custom indexers built on the standard JSON-RPC eth_getLogs and eth_getBlockByNumber work, but the offerings are not turnkey.

The website's "Latest Release (v1.3.0)" download box is **[Outdated Public Copy]**: mainnet enforcement at block 480,000 requires v2.0.0 or higher per NIP-0002. The repository releases page is the authoritative source for the current client tag.

25. First workflow: connect, fund, deploy, call

A complete first session against the devnet looks like this:

25. First workflow: connect, fund, deploy, call

A complete first session against the devnet looks like this:

```
# 1. Clone the devnet repo (which ships a working Hardhat config)
git clone https://github.com/EthernovaDev/ethernova-devnet.git
cd ethernova-devnet/devnet

# 2. Install Hardhat
npm init -y
npm install --save-dev hardhat @nomicfoundation/hardhat-toolbox

# 3. Copy the prepared config and environment template
cp hardhat.config.js ../hardhat.config.js
cp .env.example ../.env
# Edit ../.env and set DEPLOYER_PRIVATE_KEY=0x...
```

The included hardhat.config.js declares an ethernova_devnet network pointing at https://devrpc.ethnova.net

with chain ID 121526. Fund the deployer address from the public faucet at https://faucet.ethnova.net.

```
# 4. Compile and deploy a contract npx hardhat compile npx hardhat
run scripts/deploy.js --network ethernova_devnet
```

To target mainnet, change the network entry to use https://rpc.ethnova.net and chain ID 121525. There is no other change.

26. Practical Solidity examples (precompiles 0x20 through 0x28)

The precompiles are addressable contracts. Calls follow the normal Solidity ABI pattern. The examples below are short and verified by the public test contracts on the devnet (NovaToken, NovaNFT, NovaMultiSig).

Batch keccak hash via 0x20. Costs 30 gas per item versus roughly 36 gas per item in pure Solidity, so the savings are small in absolute terms but compound over many items.

```
// Hash three 32-byte items in a single precompile call.
function batchHash3(bytes32 a, bytes32 b, bytes32 c)
    internal view returns (bytes32, bytes32, bytes32)
{
    (bool ok, bytes memory out) =
        address(0x20).staticcall(abi.encodePacked(a, b, c));
    require(ok, "novaBatchHash failed");
    return abi.decode(out, (bytes32, bytes32, bytes32));
}
```

Batch ECDSA verify via 0x21. Costs 2,000 gas per signature versus 3,000 gas for ecrecover. Useful for multisig, attestation, or any aggregated-signature workflow.

```
// Verify two (hash, r, s, v) triples; result is two left-padded
recovered addresses.
function batchVerify2(
    bytes32 h1, bytes32 r1, bytes32 s1, uint8 v1,
    bytes32 h2, bytes32 r2, bytes32 s2, uint8 v2
) internal view returns (address, address) {
    bytes memory input = abi.encodePacked(h1, r1, s1, v1, h2, r2, s2,
v2);
    (bool ok, bytes memory out) = address(0x21).staticcall(input);
    require(ok, "novaBatchVerify failed");
    return abi.decode(out, (address, address));
}
```

Smart-wallet guardian recovery via 0x22. A four-step lifecycle: register guardians once, initiate recovery, collect approvals, finalize after the 100-block timelock. Reads cost 2,000 gas; writes cost 10,000 gas.

```
interface INovaAccountManager {
    function setGuardians(uint8 threshold, address[] calldata
guardians) external;
    function initiateRecovery(address target, address newOwner)
external;
    function approveRecovery(address target) external;
    function finalizeRecovery(address target) external;
    function getKeyRotation(address addr) external view returns
(bytes32);
}
// Cast: INovaAccountManager(address(0x22)).setGuardians(2, [g1, g2,
g3]);
```

Native oracle read via 0x28. Returns a TWAP-smoothed price; the circuit breaker rejects any single-block price change exceeding 15%. Cost: 2,000–5,000 gas.

```
function getNovaUsdTwap() internal view returns (uint256 price1e18)
{
    (bool ok, bytes memory out) =
        address(0x28).staticcall(abi.encodePacked(uint8(0x01) /*
feed: NOVA/USD */));
    require(ok, "novaOracle failed");
    price1e18 = abi.decode(out, (uint256));
}
```

The remaining precompiles — novaFrameApprove (0x23), novaFrameIntrospect (0x24), novaTokenManager (0x25), novaShieldedPool (0x26), novaContractUpgrade (0x27) — follow the same pattern: an interface declared in the project's Solidity, a staticcall (for reads) or call (for writes), and ABI decoding of the return value.

27. Practical Tempo and Frame AA examples

Tempo atomic batch. Submit a transaction whose envelope contains an array of calls. If any call reverts, the whole batch is rolled back and the gas charged is bounded by the per-call overhead (5,000 gas per call to discourage CPU-bombing batches that revert at the end).

Pseudocode at the wallet layer (the exact RLP encoding lives in the client):

```
TempoTx {
    type: 0x04,
    calls: [
        { to: tokenAddr, value: 0, data: approve(spender, 1000) },
        { to: dexAddr,   value: 0, data: swap(tokenAddr, otherToken,
1000) }
    ],
    feePayer: 0x...,            // optional: another address co-
signs and pays gas
    validAfter: 0,
    validBefore: blockNow + 50, // expires after 50 blocks
    signature: senderSig,
    feePayerSignature: feePayerSig
}
```

Frame AA approval. Inside an account contract:

```
// Inside MyAccountContract, called as the first frame of a Frame
transaction:
function validateAndApprove(bytes calldata signature, bytes32
frameDigest) external {
    require(_verify(signature, frameDigest), "bad signature");
    // Approve sending AND gas payment for the next frame.
    (bool ok,) = address(0x23).call(abi.encodePacked(uint8(0x02)));
// 0x02 = both
    require(ok, "novaFrameApprove failed");
}
```

Conditional sponsorship via Frame introspection. A sponsor contract that only pays gas if the next frame transfers a particular token to itself:

```
function sponsorIfPaying(address expectedToken) external {
    // Read function selector of the next frame (selector = 0x05).
    (bool okSel, bytes memory selOut) =
        address(0x24).staticcall(abi.encodePacked(uint8(0x05),
uint8(1)));
    bytes4 nextSelector = abi.decode(selOut, (bytes4));
    require(nextSelector == bytes4(0xa9059cbb) /*
transfer(address,uint256) */,
            "next frame is not a transfer");

    // Read target of the next frame (selector = 0x01).
    (bool okTgt, bytes memory tgtOut) =
        address(0x24).staticcall(abi.encodePacked(uint8(0x01),
uint8(1)));
    address nextTarget = abi.decode(tgtOut, (address));
    require(nextTarget == expectedToken, "next frame targets the
wrong token");

    // Approve gas payment only.
    (bool okApprove,) =
address(0x23).call(abi.encodePacked(uint8(0x01)));
    require(okApprove, "approve failed");
}
```

These examples are deliberately small. Production code will wrap them in libraries; nothing here requires a custom compiler.

28. Practical NIP-0004 Phase 1–3 devnet examples

These examples target the devnet (chain ID 121526) only. They illustrate the shape of NIP-0004 primitives in their currently-deployed form. Slot assignments reflect the 2026-04-25 amendment to NIP-0004 §3.4.

Register a Protocol Object via 0x29 (novaProtocolObjectRegistry). The registry is type-tag-discriminated and uses a shared global ID nonce. The slot for specialized lifecycle managers (such as novaMailboxManager) is 0x2C and is planned for Phase 4.

```
// Register a generic Protocol Object of type-tag 0x01 (Mailbox, draft).

// On the devnet today the registry exists; the specialized Mailbox manager does not.

function registerMailbox(bytes32 ownerHint) external returns (bytes32 id) {

    uint8 typeTag = 0x01;

    (bool ok, bytes memory out) =

        address(0x29).call(abi.encodePacked(typeTag, ownerHint));

    require(ok, "registry failed");

    id = abi.decode(out, (bytes32));

}
```

Enqueue a deferred effect via 0x2A (novaDeferredQueue). The effect is committed to the state tree at block N as a pending entry, and is processed deterministically at the start of block N+1 in the Deferred Processing Phase. On the devnet today the queue is functional; the application-level opcodes (MSEND, etc.) that will eventually use it are not yet implemented.

```
function enqueueDeferred(bytes32 target, bytes calldata payload) external {

    (bool ok,) = address(0x2A).call(abi.encodePacked(target, payload));

    require(ok, "deferred enqueue failed");

}
```

Register a content reference via 0x2B (novaContentRegistry). A content reference is the on-chain commitment to an off-chain payload. Today it stores hash + size + MIME type + optional rent-backed expiry.

```
function registerContent(bytes32 contentHash, uint64 size, bytes16
mime)
    external returns (bytes32 contentRefId)
{
    (bool ok, bytes memory out) =
        address(0x2B).call(abi.encodePacked(contentHash, size,
mime));
    require(ok, "content registry failed");
    contentRefId = abi.decode(out, (bytes32));
}
```

These three precompiles are the *foundation* of the NIP-0004 architecture but are not, on their own, sufficient to build the full applications described in part VI. They are what is shippable today; the rest is the next phases of work.

Part V — Current protocol foundations (Noven Fork, mainnet)

29. Mainnet fundamentals

Ethernova mainnet is a CoreGeth-derived, Go-implemented EVM execution client. The fundamental parameters at the time of writing:

- **Chain ID / Network ID:** 121525 (0x1dab5)
- **Consensus:** Ethash, Proof of Work
- **Block reward:** 10 NOVA
- **Target block time:** ~10–11 seconds (Ethash difficulty adjusts to maintain the target)
- **Native token:** NOVA, 18 decimals
- **Latest mandatory client family:** v2.0.0 or higher (per NIP-0002 enforcement at block 480,000)

- **Genesis (block 0) hash:** 0xc3812eb81498965a3f9ff3e73d2f423934e6d440578d4f4fbb6623cc61c453d9

Earlier EVM-compatibility forks landed at block 105,000 (Constantinople, Petersburg, Istanbul) and block 110,500 (EIP-658 receipt status). The **Noven Fork** at block 480,000 is the activation of the v2.0.0 protocol: every Noven Fork feature in this part is gated on that block. Block hashes from block 1 to block 445,000 were verified identical between v1.3.x and v2.0.0 client lines, providing a strong consistency guarantee for pre-fork history.

30. Noven Fork overview

The Noven Fork — formally NIP-0002, Final, dated 2026-04-03 — is the largest protocol upgrade in Ethernova's history. Activation block is 480,000 on mainnet (devnet activation block was 20,500 for the same set of features). Nodes running pre-v2.0.0 clients are rejected at handshake past the fork block and cannot mine valid blocks beyond it.

The fork bundles eight independently-valuable protocol changes: Adaptive Gas v2, the per-EVM reentrancy guard, the 90% gas refund on revert, State Expiry v2 with an external index, Tempo transactions, Frame Account Abstraction precompiles, anti-MEV FIFO ordering, and the registration of all nine Nova precompiles (0x20 through 0x28). Backward compatibility is the headline property: every existing Solidity contract continues to function with no change required, and every existing tool continues to interoperate.

The fork is named after community developer **Noven**, who designed the deterministic adaptive-gas implementation and contributed several of the consensus-critical fixes uncovered during devnet testing.

31. Adaptive Gas v2

Adaptive Gas v2 is the deterministic, trace-based version of the gas-discount system. Previous experimental versions (v1.0.0–v1.0.1 on the devnet) modified gas costs from per-node profiling data and produced a 4–17 gas divergence between nodes, leading to BAD BLOCK errors. The v2 implementation uses only data from the current execution trace, only integer math, no global mutable state, no maps, and no floating point. Cross-platform identical results have been verified between Linux (CGO_ENABLED=1) and Windows (CGO_ENABLED=0) builds at the byte level.

Mechanism. The system counts opcodes during execution. The classifier excludes SLOAD and JUMPI from the penalty calculation so that two transactions taking different code paths through the same contract receive the same classification. Three buckets emerge from the SSTORE counter:

Category	SSTORE count	Adjustment
Pure	0	up to −25%
Light	0 (reads only)	up to −15%
Mixed	1–4	0%
Complex	5+	up to +10%

The adjustment applies to **execution gas only**. Intrinsic gas (the 21,000-base) is never modified. The discount or surcharge is computed post-execution and applied to the final gas charge.

Practical effect. The deployed test contracts demonstrate the model:

Contract	Calls	Pure ratio	Effect
NovaToken (ERC-20)	11+	99%	−25% discount
NovaNFT (ERC-721)	1+	100%	qualifying
NovaMultiSig	1+	99%	qualifying

A storage-heavy DEX that performs five or more SSTOREs per swap will pay the +10% surcharge. A pure-math library or a hash-computation contract will pay 25% less.

32. Native precompiles 0x20 through 0x28

Nine precompiles are live on mainnet at fixed addresses. Costs are summarized below; specific input/output formats are documented in the client repository.

Address	Name	Purpose	Gas (typical)
0x20	novaBatchHash	Batch keccak256 hashing	30 / item
0x21	novaBatchVerify	Batch ECDSA signature verification	2,000 / sig
0x22	novaAccountManager	Smart-wallet recovery, key rotation	2k–10k
0x23	novaFrameApprove	Frame-AA transaction approval	5,000
0x24	novaFrameIntrospect	Cross-frame inspection	2,000
0x25	novaTokenManager	Native multi-token operations	1k–500k
0x26	novaShieldedPool	Optional privacy	50k–100k

		(commitment-nullifier)	
0x27	novaContractUpgrade	Safe contract upgrade with timelock	50,000
0x28	novaOracle	Protocol-level price oracle with TWAP	2k–5k

novaTokenManager is worth singling out: it makes tokens *protocol objects* rather than smart contracts. Issuance, transfer, and balance queries become precompile calls. Token transfers cost approximately 5,000 gas (compared to ~50,000 gas for an ERC-20 transfer) and the approve-then-transferFrom pattern is gone — and so is the phishing surface that pattern carries on Ethereum. ERC-20 still works on Ethernova; native tokens are an alternative path.

novaOracle includes a per-block circuit breaker that rejects any price update changing the value by more than 15%, mitigating the flash-loan oracle-manipulation pattern. novaContractUpgrade has a built-in 100-block timelock and rejects empty code or size changes greater than 10×, mitigating the storage-corruption class of upgrade bug. novaShieldedPool caps single withdrawals at 10,000 NOVA and double-checks pool accounting on every withdrawal.

33. Per-EVM reentrancy model

The reentrancy guard is implemented per EVM instance. A contract calling itself while already on its own call stack triggers a revert at the EVM level. A contract A calling B which calls C is *not* blocked, because cross-contract composability is necessary for legitimate patterns (DEX aggregators, oracle pulls, flash-loan composability).

The guard is per-EVM rather than global so that concurrent callers — eth_call, the miner, the tracer, the simulator — do not interfere with each other. An earlier devnet implementation used a global guard and produced exactly that interference, which is one reason this design was chosen.

The guard does not replace contract-level access controls. Code that previously used OpenZeppelin's ReentrancyGuard does not need it for self-reentrancy protection but may still want it to guard cross-contract patterns it considers unsafe.

34. Gas refund on revert

When a transaction reverts on Ethernova, 90% of execution gas is refunded. The refund is constant — there is no runtime toggle — and is gated on an anti-DoS condition: the refund only applies if the executed work consumed less than 100,000 execution gas. This cap exists because, without it, an attacker could craft a 30-million-gas computation that intentionally reverts and pay only 10% of the cost. The cap eliminates that vector while still covering the overwhelming majority of normal user transactions.

A user submitting a swap that reverts because of slippage typically pays only the 21,000-gas base plus 10% of whatever execution work occurred. Compared to Ethereum, where the same revert costs the user the full submitted gas, this is a meaningful UX improvement and one of the most-cited reasons to choose Ethernova for retail-facing applications.

35. State Expiry v2

State Expiry v2 is the third generation of the design. The first two generations — LastTouched field embedded in StateAccount (v1.0.4), and lazy sweep of stateObjects (v1.0.7) — were both consensus-unsafe: the first changed the state-trie RLP shape (Windows/Linux divergence on encoding), and the second iterated a Go map (non-deterministic ordering across nodes). Both were caught on the devnet.

The v2 design moves *all* expiry metadata out of the consensus state trie and into an external LevelDB index with three key prefixes:

```
StateAccount{Nonce, Balance, Root, CodeHash}     → state trie
(state root UNCHANGED)

LastTouchedIndex{address → blockNumber}          → external DB  (no
consensus impact)

BlockIndex{blockNumber → []address}              → external DB
(deterministic sweep order)

ArchivedAccounts{address → SlimRLP}              → external DB
(resurrection data)
```

When a contract is touched (SetState, SetCode), its address is recorded in the external index. At the end of each block, touched addresses for that block are saved to a *sorted* list in the BlockIndex. At block N, the sweep looks up block N − 900,000 (mainnet) and archives any contracts in that bucket that have not been touched since. Because the input list is sorted by address bytes, the sweep order is deterministic across all nodes, regardless of the node's execution history. Because the data being deleted is the contract's storage trie (and the sweep produces no change to the state-trie RLP encoding for unaffected accounts), state-root divergence is structurally impossible.

EOAs are absolutely exempt: IsExpired() returns false for any non-contract account. Pre-fork accounts (LastTouched = 0) are not expired until they are first touched after the fork, removing the risk of mass expiry at the activation block.

The expiry threshold on mainnet is **900,000 blocks** (~115 days at 11-second blocks). The website's claim of "1,000 blocks" — and the corresponding "115-day" framing in older

marketing copy — should be read as **[Outdated Public Copy]** for mainnet; 1,000 blocks is the devnet's test threshold, set short to make the mechanism observable in practice.

Archived contracts are not lost. They are persisted as slim RLP records keyed by the original address, and can be restored via Merkle proof when (in the NIP-0004 future) the witness-restoration protocol is fully implemented.

36. Tempo transactions

Tempo transactions add three account-abstraction features that solve real user-experience problems on Ethereum. They are submitted as transaction type 0x04 and carry an array of calls plus optional fee-delegation and scheduling fields.

Atomic batching. Up to 16 calls are bundled into a single transaction. If any call reverts, the entire batch reverts. The classic use case is approve + swap collapsed into one transaction, removing the two-step UX and the dangling-allowance phishing surface.

Fee delegation. An optional feePayer field names another address that co-signs the transaction with its own ECDSA signature and pays gas on behalf of the sender. This means a dApp can sponsor gas for a new user who has no NOVA yet — the chief obstacle to onboarding. Critically, gas is *always* paid in NOVA, never in an arbitrary token. This is a deliberate design choice: ERC-20 gas payments would reduce the utility of the native token, so Ethernova rejects that direction. Fee delegation solves the UX problem within the NOVA-only constraint.

Scheduling. validAfter and validBefore block-height fields constrain when the transaction is eligible for inclusion. Scheduled transactions enable limit-order primitives, recurring payments, and AI-agent automation without third-party relayers or centralized schedulers.

The maximum scheduling window is 500 blocks ahead, a deliberate cap to prevent a mempool RAM-exhaustion attack with millions of long-future-dated transactions. Each call in a Tempo batch carries a 5,000-gas overhead so that a 16-call batch which intentionally reverts at the end cannot be used as a CPU bomb.

37. Frame Account Abstraction

Frame AA is implemented as two precompiles, 0x23 (novaFrameApprove) and 0x24 (novaFrameIntrospect), which together provide the building blocks for smart-contract wallets, conditional gas sponsorship, privacy approvals, delegated permissions, and AI-agent automation — without changing any existing tooling, and without introducing ERC-20 gas payments.

novaFrameApprove takes a single byte selecting the approval mode: 0x00 for "approve sending", 0x01 for "approve gas payment", 0x02 for both. Approvals are per-transaction and reset for each new transaction. The cost is 5,000 gas per call.

novaFrameIntrospect lets one frame inspect another in the same transaction. A second-byte selector chooses the field to read: 0x01 for the target address, 0x02 for the keccak256 of the calldata, 0x03 for the value, 0x04 for the gas limit, 0x05 for the function selector. The cost is 2,000 gas per call. Introspection is what makes conditional sponsorship possible: a sponsor can refuse to pay gas unless the next frame is, say, a token transfer to itself.

Together, these precompiles deliver the same outcome as Ethereum's EIP-8141 (Frame Transactions, endorsed by Vitalik Buterin), without breaking tools and without requiring ERC-20 gas. Smart-contract wallets validate signatures and call novaFrameApprove. Passkeys, multisig, and quantum-resistant signature schemes are all supported because the precompile does not constrain how the wallet validates.

38. Anti-MEV fair ordering

Transactions are ordered by arrival time at a node, not by gas price. A miner cannot reorder transactions in the mempool to extract sandwich profit. The mempool also rate-limits each sender to 16 pending transactions, preventing a bot from flooding the queue with min-gas dummy transactions to occupy FIFO priority slots. Both behaviors are enforced in the mempool and in the worker code path.

Fair ordering is not absolute MEV elimination — a sufficiently-resourced adversary running a global mempool observation network can still attempt some forms of extraction — but it removes the gas-bidding war that has, on Ethereum, produced an estimated $600M+ of cumulative MEV extraction. Combined with the gas-refund-on-revert behavior, the strategic landscape for retail users is materially improved.

39. Native tokens, oracle, shielded pool, contract upgrade

The remaining four precompiles in the Noven Fork each address a category of bug or UX problem with a long history on Ethereum.

Native tokens (novaTokenManager, 0x25). Tokens are protocol objects, not smart contracts. Storage is in accessors_ethernova_tokens.go. Creation costs 500,000 gas (a deliberately high price to prevent token-spam attacks against LevelDB) and a single creator is capped at 100 tokens. The transfer cost is around 5,000 gas, an order of magnitude cheaper than ERC-20. There is no approve step.

Oracle (novaOracle, 0x28). Prices are attested by miners and stored as a TWAP across recent blocks. The 15%-per-block circuit breaker limits the damage from oracle manipulation via flash-borrowed liquidity. Storage lives in accessors_ethernova_oracle.go.

Shielded pool (novaShieldedPool, 0x26). Implements a commitment-nullifier scheme for private NOVA transfers. Privacy is opt-in, not default. Single-withdrawal cap is 10,000 NOVA, double-spend is prevented by nullifier tracking, and pool accounting is double-checked on every withdrawal. Because the pool is native to the protocol, it cannot be sanctioned at the contract level the way Tornado Cash was on Ethereum.

Contract upgrade (novaContractUpgrade, 0x27). A 100-block timelock between proposal and activation. Empty code is rejected. Code size changes greater than 10× are rejected. Users can see pending upgrades in time to exit before activation. The whole proxy-pattern, storage-collision class of bug is sidestepped.

40. Backward compatibility promises

The Noven Fork is 100% backwards-compatible. Specifically:

- Any Solidity contract valid on Ethereum deploys and executes on Ethernova with no modification.
- All eth_* JSON-RPC methods continue to function unchanged.
- Block hashes from block 1 to block 445,000 are verified identical between v1.3.x and v2.0.0 client lines.
- The intrinsic 21,000-gas base of a transaction is never modified by Adaptive Gas; only execution gas is adjusted.
- Wallet UX (MetaMask, hardware wallets, ethers, viem) works without updates for Domain 0 transactions.
- Tempo and Frame transactions use new envelope types (0x04 for Tempo) but standard transaction types continue to function unchanged.

NIP-0004, when its phases land on mainnet, extends backward compatibility further: every Solidity contract continues to default to "Domain 0" (classic synchronous execution) and Nova features are strictly opt-in via Domain declaration and new transaction types. Domain 0 contracts cannot accidentally invoke Nova opcodes — those will revert.

Part VI — NIP-0004: the Layered Deterministic Computer

41. What NIP-0004 is, and what it is not

NIP-0004 is a proposal for the next-generation architecture of Ethernova. It is a Standards Track / Core proposal authored by Noven, currently with status **Draft (Phase 1–3 Deployed on Devnet)**, requiring the Noven Fork as a prerequisite. The proposal does not replace the Noven Fork; it builds on it.

What NIP-0004 *is*: a fundamental architectural evolution that adds five new layers on top of the existing EVM — Protocol Objects, Deferred Execution, Protocol Channels, a State Lifecycle system, and Multi-Dimensional Resource Metering — with the explicit purpose of letting Ethernova natively support applications traditionally requiring centralized servers, while preserving determinism, PoW consensus, EVM compatibility, and bounded node requirements.

What NIP-0004 *is not*: a clean-room VM replacement, a Layer-2 strategy, an attempt to run arbitrary application logic literally inside block execution, or a promise that decentralized applications will exactly match the latency and feature set of their centralized counterparts. The "browser-like" content manifests, in particular, are a verifiable content-delivery system with a bounded execution sandbox — closer to "static site plus on-chain interaction" than to a full browser. The proposal is honest about this trade-off in its own §12.

NIP-0004 is also *not finished*. Phases 1 through 3 are deployed on the devnet. Phase 4 onward — Mailbox lifecycle, Channel Layer, Witness verification, async callbacks, game state, content manifests, social graph, identity attestation — is specified but not yet implemented. The remainder of part VI describes the architecture as designed; status tags indicate where each piece currently stands.

42. Why it exists

Ethereum advertises itself as a "world computer" but in practice operates as a settlement layer. The discrepancy is structural, not accidental, and has five components.

The execution model is *fully synchronous and atomic*. There is no protocol-level concept of a session, a mailbox, or a multi-step interaction whose phases are themselves observable by the chain. Every interaction must be packed into a single transaction.

The state is *monolithic and ever-growing*. All state is stored forever, with no protocol mechanism for cleaning up unused contracts or expiring inactive storage. Ethereum mainnet state has crossed 200 GB and continues to expand.

The gas model is *one-dimensional*. Compute, storage, and bandwidth are all priced through a single metric, which means a busy DEX raises the gas price for chat applications, and a flood of messages raises the gas price for DeFi. Different categories of activity cannot be congestion-isolated from each other.

There are *no asynchronous communication primitives*. There is no inbox, no outbox, no message queue, no subscription, no channel at the protocol level. Every dApp that needs these reinvents them in contract storage.

There is *no native content-addressing*. There is no protocol-level way to reference off-chain content by hash with integrity guarantees attached.

NIP-0004 attempts to address all five at once, in layers, while honoring an absolute compatibility constraint with the existing EVM ecosystem.

43. Three truth layers

The proposal is built on a distinction between three layers of truth, each with its own determinism property.

Settlement Layer. On-chain, inside blocks, fully deterministic. This is where final state transitions live: ownership, proof verification, commitment anchoring. It is exactly what Ethereum already does.

Interaction Layer. Off-chain, but protocol-aware. Cryptographically bound and deterministically resolvable on-chain when needed. This is where real-time peer interactions live, using signed messages, channel states, and protocol-defined commitment schemes. The interactions themselves do not enter every block; the format, signing scheme, sequence numbering, and arbitration rules do.

Witness Layer. Archival, proof-restorable, verifiable via Merkle proofs against on-chain roots. This is where historical or inactive state lives — pruned from active nodes but reconstructable with cryptographic proof when needed.

The protocol defines the format, validation rules, and commitment schemes for all three. Even though interactions may occur outside blocks, their final correctness is always anchored to the chain and re-verifiable from first principles.

44. Nova Extension Layer

The chosen architecture is "EVM+ with Nova Extension Layer." Three other options were evaluated and rejected: precompiles-only (too limited), dual-mode VM (validator complexity doubles, tooling fragments), and full VM replacement (destroys practical compatibility). The selected approach extends the EVM with three categories of additions.

Nova Opcodes are new EVM instructions for operations frequently called by application code that need internal execution context (stack, memory, gas, caller chain). The category-1 mailbox opcodes (MSEND 0xf6, MRECV 0xf7, MPEEK 0xf8, MCOUNT 0xf9), category-2 content-reference opcodes (CREF 0xfa, CVERIFY 0xfb), and category-3 session opcodes (SOPEN 0xfc, SCOMMIT 0xfd, SCLOSE 0xfe) cover the high-frequency, low-overhead operations that justify being opcodes rather than precompiles. **Status:** [Specified / Planned] across the board.

Nova Precompiles are at reserved addresses 0x29 through 0x34. Phase 1–3 are already on the devnet:

Address	Name	Purpose	Status
0x29	novaProtocolObjectRegistry	Generic Protocol Object lifecycle	Devnet Live (Phase 1)
0x2A	novaDeferredQueue	Pending Effects Queue + block-prologue drain	Devnet Live (Phase 2)
0x2B	novaContentRegistry	Content reference	Devnet Live

		registration and lookup	(Phase 3)
0x2C	novaMailboxManager	Mailbox lifecycle management	Specified / Planned (Phase 4)
0x2D	novaSessionArbiter	Session channel dispute resolution	Specified / Planned (Phase 7)
0x2E	novaContentManifest	Manifest verification for browser-like systems	Specified / Planned (Phase 11)
0x2F	novaStateWitness	State witness proof verification	Specified / Planned (Phase 5)
0x30	novaAsyncCallback	Async callback registration and resolution	Specified / Planned (Phase 7/11)
0x31	novaGameState	Optimized game state management	Specified / Planned (Phase 11)
0x32	novaComputeBounty	Off-chain computation result verification	Specified / Planned (Phase 11)
0x33	novaSocialGraph	Protocol-level social graph	Specified / Planned
0x34	novaIdentityAttestation	Identity attestation verification	Specified / Planned

The slot assignments here reflect the 2026-04-25 amendment to NIP-0004 §3.4. The original draft put novaMailboxManager at 0x29, novaContentRegistry at 0x2A, and novaSocialGraph at 0x2B; during Phase 1–3 implementation those specialists were replaced by generalized counterparts, and the originals were relocated downstream.

Nova Protocol Objects are first-class entities in the Ethernova state tree, distinct from accounts and contracts, with their own address-space prefix (0xFF), protocol-managed lifecycles, and specialized semantics.

Type	Description	Status
Mailbox	Message queue bound to an address	Specified / Planned (Phase 4 lifecycle)
Session	Bilateral interaction channel	Specified / Planned (Phase 7)
Content Reference	Pointer to off-chain content	Devnet Live (basic form, Phase 3)
Identity	Attestations, key bindings, identity metadata	Specified / Planned
Subscription	Subscription relationship between accounts	Specified / Planned

Game Room	Multiplayer game room with contract-defined rules	Specified / Planned (Phase 11)

Every Protocol Object carries a small set of common fields (id, owner, type tag, type-specific state data, expiry block, last-touched block, rent balance) plus the type-specific state data appropriate to its purpose.

45. Three execution modes

NIP-0004 defines three execution modes, each with its own determinism property.

Mode 1 — Synchronous Execution (Ethereum-Compatible). Identical to Ethereum. Transactions execute inside blocks, atomic, sequential. All legacy Solidity contracts run in this mode. No changes. **Status: [Mainnet Live].**

Mode 2 — Deferred Execution. Transactions using MSEND, SCOMMIT, or novaAsyncCallback produce *deferred effects*. The effects are committed to the state tree at block N as pending entries; the actual side effects (message delivery, callback execution) happen at the start of block N+1, before regular transactions, in a Deferred Processing Phase. Deferred Processing is fully deterministic — order is determined by transaction index in block N — so no node sees a different result. **Status: [Devnet Live] foundation (Phase 2 queue), [Specified / Planned] for opcode-driven use.**

Mode 3 — Channel Execution (Off-Chain, Protocol-Bound). For interactions requiring low latency, NIP-0004 defines Protocol Channels: similar in spirit to state channels, but with a protocol-standardized format. Two parties open a channel via SOPEN (on-chain, ~11 seconds), then exchange signed state updates directly peer-to-peer with a monotonically increasing sequence number. Either party can SCOMMIT (checkpoint) or SCLOSE (settle) at any time. If a dispute arises — one party submits old state — novaSessionArbiter resolves it deterministically based on the highest valid sequence number. **Status: [Specified / Planned], Phase 7.**

The trust model in Mode 3 is "trust-minimized but not absolutely trustless": as long as one party is honest and connected to the network, they can prove correct state to the chain. This is materially better than a centralized server but does not match a fully synchronous on-chain settlement.

46. Execution Domains

To prevent contamination between execution modes, contracts declare an Execution Domain at deployment.

Domain Name	Description	description
0	Classic	Ethereum-compatible

		synchronous execution. Default.
1	Deferred	Execution producing deferred effects. Contracts must explicitly opt-in.
2	Channel	Contracts participating in Protocol Channels, with different state-management rules.

Domain 0 contracts cannot call Domain 1 or Domain 2 opcodes — those calls revert. Cross-domain interaction goes through a Domain Bridge Protocol: a Domain 0 contract can send a message to a Domain 1 contract via MSEND, but the response is deferred (not synchronous). The asymmetry is by design — it forces developers to think about asynchrony explicitly rather than discovering it at runtime.

Domain 0 is sacred. Whatever happens in Domain 1, Domain 2, or the Channel Layer, Domain 0 must continue to function exactly as a standard EVM. This is the always-safe fallback: if every new layer fails to find adoption, Ethernova still functions as a hardened EVM-compatible chain.

Status: Domain declaration is [Specified / Planned]. Today, on the devnet, all contracts implicitly run in Domain 0; the registry-and-queue precompiles are callable from Domain 0 contracts via standard address.call() semantics.

47. Capability model

NIP-0004 introduces capabilities as an extension of the EVM call model. When contract A calls contract B, in addition to msg.sender and msg.value, the protocol provides msg.capabilities — a bitmask declaring what the caller is permitted to do (read mailbox, send message, open session, access social graph, and so on).

Capabilities propagate through the call chain but can only be **narrowed**, never expanded. Contract A can call B with only "read mailbox" capability, and B cannot then send messages on A's behalf. Library contracts called via DELEGATECALL can have their capabilities restricted. The capability bitmask is enforced at the opcode level: each Nova opcode checks the relevant capability before executing, and any escalation attempt reverts.

This produces a fine-grained permission model without requiring every contract to implement its own access control. **Status: [Specified / Planned].**

48. State lifecycle tiers

Tier	Name	Recency window	Storage	Node requirement
1	Active	Last ~100,000 blocks	Hot, RAM-	All full nodes

		(~12.7 days)	accessible	hold complete Active state
2	Warm	100,001–1,000,000 blocks ago (~127 days)	Disk, separate trie	Full nodes maintain Warm state
3	Cold	1,000,001–10,000,000 blocks ago (~3.5 years)	Disk, Merkle commitment only	Full nodes hold root, not data; access requires witness
4	Archived	More than 10,000,000 blocks ago	Not stored by full nodes	Only historical state roots in block headers; restoration via archive provider
5	Expired	Explicitly expired by protocol	Deleted	Owner has 1,000,000-block grace period to revive; after that, only Merkle proof of prior existence remains

Tier transitions are *lazy*. They do not happen in a single block migrating millions of entries. Instead, on access: the node checks the tier based on last_touched_block; if state is in a colder tier, the transaction must pay a warming fee proportional to the tier gap, and for Cold or Archived state must include a State Witness (Merkle proof) as part of the transaction data. After access, state is promoted back to Active.

The tier of any state entry is fully determined by last_touched_block versus current_block. There is no ambiguity, no node-local information, and no race condition.

Status: The current State Expiry v2 mechanism (described in §35) implements a simplified version of this — Active state plus archival-on-expiry, with the threshold at 900,000 blocks on mainnet. The full five-tier system is [Specified / Planned].

49. Witness and restoration

When state in Cold or Archived tier needs to be accessed, the caller includes a StateWitness in the transaction: a Merkle proof demonstrating the state existed at a specific historical state root. The node verifies the witness against the historical state root stored in block headers; if valid, the state is restored to Active tier and the warming fee is deducted from the caller.

Witnesses can be obtained from archive nodes (which store all historical state and generate proofs on demand), from witness services (third parties indexing historical state, similar in role to Infura but for state proofs), or self-served (users who store their own historical data generate proofs themselves).

Forging a witness requires finding a hash collision on the keccak256 Merkle tree, which is cryptographically infeasible. Verification will be handled by the planned novaStateWitness precompile (0x2F). **Status: [Specified / Planned], Phase 5.**

50. Multi-Dimensional Resource Metering

NIP-0004 replaces Ethereum's one-dimensional gas with a five-component **Resource Vector**: Compute, State Read, State Write, Protocol Ops, and Proof Verify. Each transaction has a separate limit per dimension, and each block has a separate limit per dimension. Each dimension has its own base price, adjusted per block: if the previous block used more than 50% of a dimension's capacity, that dimension's price rises (up to ±12.5% per block, the same rate as EIP-1559 but per dimension).

The key implication is **congestion-domain separation**. If the network is busy with compute-heavy DeFi arbitrage, compute price rises, but Protocol Ops price stays low — chat and email do not become expensive because DeFi is busy. Conversely, a flood of mailbox messages raises Protocol Ops price without affecting compute.

For backward compatibility, transactions not using new features have their gasLimit mapped to a deterministic combination of compute + state read + state write, so existing tooling continues to function. Developers wanting fine-grained control can use an extended transaction format with per-dimension limits.

Status: [Specified / Planned].

51. Application primitives — email

Decentralized email on Ethernova combines a Mailbox Protocol Object as inbox, a Content Reference for body and attachments, and an Identity Object for addressing.

Workflow: Alice writes the body, encrypts it with Bob's public key (from Bob's Identity Object), uploads the ciphertext to decentralized storage (IPFS or equivalent), and submits a transaction calling MSEND(bob_mailbox_id, content_hash, metadata). In the next block, during Deferred Processing, the message is delivered to Bob's Mailbox. Bob's client polls his

mailbox, retrieves the content from decentralized storage using the content hash, and decrypts.

Latency: approximately 22 seconds (two block times) for delivery. Not instant, but acceptable for email.

Anti-spam mechanisms: every MSEND costs gas in the Protocol Ops dimension; the mailbox owner can set a whitelist or blacklist; the owner can require a minimum NOVA "postage" from unknown senders, claimable back if the sender is later whitelisted; the social graph can charge higher postage to senders not connected to the recipient. Spam is made economically expensive while legitimate communication remains cheap.

Status: [Specified / Planned] (Mailbox lifecycle in Phase 4; the Deferred Queue foundation is [Devnet Live] in Phase 2).

52. Application primitives — chat

Direct messages use Mode 3 (Channel Execution). Alice and Bob open a channel with SOPEN, then exchange signed messages directly peer-to-peer. Latency is network latency, not block time. Every N messages or every M seconds, one party commits a checkpoint state hash to chain via SCOMMIT. If connection drops or dispute arises, the chain becomes the arbiter, and novaSessionArbiter resolves based on the highest sequence number with valid signatures from both parties.

Group chat requires a different model because channels are bilateral. A ChatRoom contract is deployed in Domain 1 (Deferred). Group members send messages via MSEND to the ChatRoom's mailbox; the ChatRoom maintains a subscriber list and fans out messages to all subscriber mailboxes during Deferred Processing. Latency: approximately 22 seconds per message. For lower-latency group chat, a gossip overlay supplements this: members exchange messages peer-to-peer while sending periodic on-chain commitments. The client renders P2P messages with a "pending" indicator until on-chain confirmation.

Trust model: DM via channel is trust-minimized as long as one party can submit state to chain in a dispute window; group chat via Deferred is fully trustless (all messages on-chain); group chat via gossip overlay is trust-minimized with the property that any message that ever existed in P2P can be proven on-chain by any party who stored the signed message.

Status: [Specified / Planned] for both forms.

53. Application primitives — social

Social media on Ethernova is built from five primitives: Identity Object (profile, name, avatar, bio), Social Graph (via the planned novaSocialGraph precompile at 0x33), Mailbox (notifications and DMs), Content Reference (posts — payload off-chain, hash on-chain), and Subscription Object (so followers receive updates without polling).

Posting workflow: the user uploads payload to decentralized storage; submits ContentRef.create(hash, size, type) plus SocialContract.post(content_ref_id, metadata); the SocialContract (a Domain 1 contract) records the post in the user's timeline and triggers fanout to subscribers via Subscription Objects; subscribers receive notification in their mailboxes during Deferred Processing.

Feed construction is delegated to the client or an indexer — the chain provides authoritative data, not rendering. Per-post on-chain cost is approximately 200 bytes (content reference plus metadata).

Moderation operates at three levels: protocol (any user can block or mute another address in the social graph); application (the SocialContract can implement community moderation rules — flagging, reputation-based visibility); and content (the content reference can be allowed to expire by stopping rent payments, removing it from chain perspective, though off-chain copies may persist).

Status: [Specified / Planned]. The novaContentRegistry foundation (0x2B) is [Devnet Live] in Phase 3.

54. Application primitives — games

The approach is **Authoritative State Anchoring**. A Game Room Object is created on-chain with the game contract reference, player list, and any stake. Players interact via Protocol Channel — turn-by-turn or real-time — exchanging signed state updates. Periodically, or at game end, the final state is committed to chain. The contract validates the final state against game rules and distributes rewards or penalties.

Verifiable randomness uses commit-reveal: each player submits hash(seed) at block N, reveals seed at block N+1, and the final random value is the XOR or hash of all revealed seeds. Players who fail to reveal are penalized (stake slashed). For randomness not requiring player participation, novaGameState (0x31) provides a block-hash-based VRF.

State compression is automatic: only state diffs are committed, not full game state. After completion, the Game Room can be archived (demoted to Cold tier), so the storage burden for finished games is bounded.

A complete chess example: Game Room references the chess contract with two players each staking 10 NOVA; Alice and Bob open a channel and exchange signed moves peer-to-peer; every 10 moves a checkpoint is committed to chain; at game end, the final board state is submitted and the chess contract verifies the move sequence; the winner receives 20 NOVA minus gas; the Game Room is archived.

Status: [Specified / Planned], Phase 11.

55. Application primitives — content manifests

A Content Manifest is a structured document describing a decentralized "page": a manifest ID, a version number, a publisher address, a list of content references (HTML, CSS, deterministic JavaScript subset, images), an explicit permission block (can_read_identity, can_send_message, can_access_social_graph), a list of dependency manifests, and the publisher's signature.

Browsing a manifest works as follows. The client receives a manifest ID (from a link, a social post, a search). The client validates the manifest on-chain via novaContentManifest.verify(manifest_id) — confirming the publisher is valid and content references are not expired. The client downloads each content piece from decentralized storage using the content hashes, verifies the integrity of each piece (hash match), and renders according to type. If content is js_safe, it executes in a sandbox with capabilities restricted by manifest.permissions.

Important honest limitations apply. This is *not* a full web browser. js_safe is not arbitrary JavaScript — it is a deterministic subset running sandboxed, closer in spirit to a controlled WebAssembly execution than to a full browser runtime. There is no server-side rendering, no dynamic database queries — all state must come from the chain or be pre-computed. Manifests are immutable after publication; updates produce a new manifest with a version increment. The model is closer to "static site plus chain interaction" than "full web app."

Status: [Specified / Planned], Phase 11.

56. Phase 1–3 deployment status (devnet)

The currently-shipped subset of NIP-0004 on the devnet is:

- novaProtocolObjectRegistry (0x29). Generic Protocol Object lifecycle: create, get, list, delete across all object types. Type-tag discriminated, shared global ID nonce. The specialized novaMailboxManager originally drafted at this slot moved to 0x2C and is scheduled for Phase 4.
- novaDeferredQueue (0x2A). The Pending Effects Queue plus the block-prologue drain. Effects are enqueued in block N and executed deterministically at the start of block N+1, ordered by sequence number. This is the foundation that the Mailbox opcodes (Phase 4), AsyncCallback (Phase 7/11), and SessionUpdate (Phase 7) all rely on.
- novaContentRegistry (0x2B). Content reference registration and lookup with rich metadata: content hash, size, MIME type, availability proof, rent-backed expiry. A "DNS for content" on Ethernova.

Together, these three precompiles establish the Phase 1–3 foundation: identity (registry), asynchrony (queue), and content addressing (registry). They are sufficient for foundational experiments but not, on their own, for the full applications described above. The next phases — Mailbox lifecycle, State Witness, Session Arbiter, and the application-level opcodes — build on this foundation.

Part VII — Realistic product direction

57. Strongest near-term opportunities

Three categories of application can ship today on mainnet and use Ethernova's distinctive features without depending on anything still in NIP-0004 design.

Onboarding-friendly consumer dApps. Tempo fee delegation removes the "buy NOVA before you can do anything" obstacle. Combined with novaAccountManager guardian recovery, a new user can create an account, recover it if they lose access, and interact with sponsored gas — all without ever touching a centralized custodian. This is a meaningful UX improvement over Ethereum's current state.

Token launches with native-token economics. novaTokenManager (0x25) gives a project a tenfold gas discount on transfers and removes the entire approve/transferFrom phishing surface. For a project that does not need the full ERC-20 contract surface, this is a strict win.

Scheduled and automated finance. Limit orders, recurring payments, vesting schedules, and AI-agent-driven automation can be expressed natively through Tempo's validBefore/validAfter fields without third-party schedulers. This is a primitive Ethereum has discussed but never shipped at L1.

These three are the most credible "ready today" categories. None of them require reading further into NIP-0004 to ship.

58. Possible killer-app directions

Looking further out, four directions look genuinely differentiated for Ethernova once NIP-0004 phases land.

Decentralized email or notification systems built on Mailbox + Deferred Execution. The economic anti-spam mechanisms — postage requirements, social-graph-aware pricing, claim-back for whitelisted senders — are protocol-native rather than reinvented per-application. This is the simplest application primitive in NIP-0004 and the one most likely to attract early traction because the user need (a portable, censorship-resistant inbox) is well-defined.

Real-time DMs and small-group chat built on Protocol Channels. The user-visible latency is network latency, not block time, while the dispute-resolution path is deterministic on-chain. This is the use case the Channel Layer was designed for.

Content-anchored social built on Content References and the Social Graph precompile. The chain stores hash and metadata; payloads live in decentralized storage; feeds are constructed client-side from authoritative chain data. This is differentiated from existing

federated and decentralized social systems by the protocol-level social graph and the rent-backed content lifecycle.

On-chain games with off-chain interaction. Channel-based gameplay with on-chain settlement and protocol-managed game-room objects. The combination of fast interaction, deterministic settlement, and verifiable randomness is something Ethereum L1 cannot offer at acceptable cost.

These are *plausible* killer-app directions, not certainties. Each depends on phases of NIP-0004 that are not yet shipped.

59. What can attract developers first

Developers respond to leverage. The features most likely to convert an Ethereum developer into an Ethernova developer in the near term are, in approximate order:

The **gas refund on revert** — because it makes user-facing dApps less punishing without any code change.

The **batched precompiles** (0x20, 0x21) — because they offer measurable, easily-demonstrated savings.

Tempo transactions — because they enable user flows that are awkward to express on Ethereum and require no custom Solidity to use.

Native-token economics via 0x25 — for any project where the ERC-20 surface is overkill or where the approve-pattern phishing risk is a real concern.

Adaptive Gas v2 — because the "write pure code, get a discount" mechanic is a small but pleasant feedback loop that rewards good engineering.

The deeper NIP-0004 features will only attract developers once the application primitives — Mailbox, Channel, Manifest — are robust enough that real applications can ship on them. This is a real chicken-and-egg problem and is not yet resolved.

60. What can attract users first

Users care about outcomes, not architecture. The features most likely to attract users are:

Recoverable wallets. Losing keys is the single largest source of user loss in crypto. A guardian-recoverable smart wallet, with a 100-block timelock to give the user time to detect malicious recoveries, is a meaningful improvement.

Failed transactions costing less. The 90% gas refund is one of the easiest things to communicate to a non-technical user: "you do not pay for the transaction if it fails."

Sponsored gas onboarding. A new user starting a journey on a dApp without first having to acquire NOVA is a major reduction in friction. Tempo fee delegation makes this possible while keeping NOVA economically central.

No sandwich attacks on simple swaps. FIFO ordering does not eliminate all MEV, but it eliminates the pattern most visible to retail users.

These four are user-visible benefits that exist on mainnet today. They can be communicated without any architecture detail.

61. App categories that would fail on Ethernova

Some applications are bad fits, and saying so directly is more useful than promising universality.

High-frequency trading and gas-priority arbitrage are explicitly counter-incentivized. Any application whose competitive moat is "submit gas faster than the next bot" should not be on Ethernova.

Sub-second-confirmation applications are a poor fit until the Channel Layer ships. Ethereum L2 rollups are a better near-term home for sub-second UX.

Heavy on-chain computation — large-scale ML inference, video transcoding, anything that would saturate a compute dimension — does not belong on any Layer-1 chain, including this one. NIP-0004's novaComputeBounty primitive (0x32, planned) will help with verifiable off-chain delegation, but the chain is not the place for the computation itself.

Stablecoin-heavy DeFi at large scale needs liquidity Ethernova does not yet have. A protocol that requires deep stablecoin pools should not launch only on Ethernova.

Always-on privacy applications need a privacy-by-default chain. Ethernova's shielded pool is opt-in; building a privacy-first user experience on top of an opt-in primitive is structurally awkward.

62. A grounded ecosystem strategy

The realistic strategy for Ethernova in 2026–2027 is sequenced, not simultaneous.

Phase one (mainnet today): attract a small, high-quality cluster of dApps that take maximum advantage of the Noven Fork features — gas refund, batched precompiles, Tempo, native tokens, FIFO ordering. The goal is not to compete with Ethereum on liquidity but to demonstrate that the Noven Fork features pay for themselves in real applications.

Phase two (NIP-0004 Phase 4–5 on mainnet): ship Mailbox + Deferred Execution to mainnet. Build one canonical reference application (decentralized email or notifications) that demonstrates the model works end to end. Resist the temptation to ship every NIP-0004 phase at once.

Phase three (Channel Layer): once Mailbox is robust, ship the Channel Layer and one canonical real-time application (DMs or a small turn-based game). The relay-node ecosystem will need to bootstrap; expect a 12–18 month adoption ramp.

Phase four (full Resource Vector and State Lifecycle): only ship multi-dimensional metering once at least one application category is producing enough chain activity to make the dimensional separation observable. Otherwise the cost in implementation complexity outweighs the benefit.

The two principles binding the strategy: **ship in phases, not all at once**, and **measure adoption before adding features.** If the Mailbox infrastructure is unused after six months on mainnet, do not rush to add the Social Graph precompile; validate demand first. This is explicit advice from NIP-0004 §12.6 and is worth treating as authoritative.

Part VIII — Honesty: risks and limits

63. Architectural risks

The biggest architectural risk in NIP-0004 is **proliferation**. Every new Protocol Object type adds complexity to the state trie, requires more RAM for indexing and processing, and increases CPU time per block. The proposal's safe boundary is six Protocol Object types initially, with a ceiling of eight without rigorous evaluation. There must never be a mechanism allowing arbitrary Protocol Object creation — only protocol-defined types. The risk is that, under pressure to add use cases, this discipline erodes and the chain accumulates a long tail of half-used object types that none of the implementations can simplify away.

A second architectural risk is **Channel Layer centralization**. Channel Relay Nodes need bandwidth, and bandwidth needs incentives, which favor large operators. If the relay network becomes a small set of well-funded relays, the censorship resistance of the channel layer degrades, even though the chain itself remains decentralized. This is the classic infrastructure-versus-decentralization trade-off and there is no clean solution; only careful design and watchtower services to mitigate it.

A third risk is **Domain 0 contamination**. The current design keeps Domain 0 strictly compatible — Domain 0 contracts cannot invoke Nova opcodes. If, under pressure, that boundary is relaxed in any way, the always-safe fallback is gone and every Domain 0 contract becomes implicitly dependent on the correctness of the new layers.

64. Complexity risks

The Ethernova client is already a CoreGeth fork, which means it inherits the considerable existing complexity of go-ethereum. Adding the Deferred Processing Engine, Protocol Object Trie, Channel Arbiter, and Multi-Dimensional Metering on top of that is a very large engineering project. The codebase complexity rises by an estimated 3–4×. Every new layer is a potential source of consensus splits.

Testing complexity scales worse than codebase complexity. The number of possible state transitions in NIP-0004 is approximately the product of execution domains, channel settlements, deferred-effect orderings, state-tier transitions, and Protocol Object lifecycles — combinatorial, not additive. Comprehensive testing becomes extremely demanding, and undertesting becomes a structural risk.

Client development for application teams becomes harder than for Ethereum. Wallets and clients must handle peer-to-peer channel messages, deferred confirmations, Protocol Object queries, and witness retrieval. Ecosystem tooling will lag Ethereum's for a long time.

65. Consensus and implementation risks

Two consensus bugs have already been caught on the devnet — both before mainnet, validating the devnet's purpose, but each one a sobering data point. The first was a 4–17 gas divergence between nodes caused by per-node profiling driving gas adjustments (v1.0.0–v1.0.1). The second was a non-deterministic merkle root caused by Go map iteration in the state-expiry sweep (v1.0.7). Both have been definitively fixed (Adaptive Gas v2's pure-trace integer math, and State Expiry v2's external sorted block index), and verified with cross-platform Linux/Windows identical results.

The general lesson is that any feature modifying state-affecting outputs (gas, state root, balance, nonce) must use deterministic inputs only. Runtime profiling, map iteration, floating-point math, and any source of node-local variation must stay strictly out of the consensus path. Maintaining this discipline as NIP-0004's surface area grows is the single most important engineering challenge.

If a second node implementation is ever produced — for example, in Rust — ensuring identical determinism across implementations in Deferred Processing, Protocol Object mutations, and Channel settlement will be extremely difficult. Ethereum already struggles with multi-client determinism at the basic EVM level.

66. Tooling and onboarding risks

Wallet, indexer, and explorer tooling will lag the protocol. Today there is no public Hardhat plugin for Ethernova-specific features, no @ethernova/sdk for client-side integration, no MetaMask plugin that natively renders Tempo or Frame transactions, and no Etherscan-equivalent for Protocol Objects (because the objects do not yet exist on mainnet). Each is a tractable contribution opportunity, but each is also a real obstacle for developer velocity in the meantime.

The website is partially out of date relative to the NIPs. This is normal for a rapidly evolving project, but the practical effect is that a newcomer reading only the website and only the coregeth README will form an outdated impression. NIP-0002 (mainnet feature set) and the devnet repository's ROADMAP.md (implementation history and next-gen status) are more reliable.

67. Adoption risks

The realistic worst case is **nobody uses the new features**. All of NIP-0004's infrastructure ships, but developers continue to use Domain 0 (basic EVM) because the Ethereum ecosystem is mature enough and there is no compelling incentive to switch. Protocol Objects and Channels become dead code that adds attack surface without benefit.

Adjacent risks: state-lifecycle thresholds set too aggressively cause user-visible "witness required" experiences that feel broken; channel layer trust assumptions fail in practice because users go offline at the wrong moment and lose funds to old-state submissions; multi-dimensional pricing is mis-calibrated and either invites Protocol Ops spam (price too low) or makes communication non-viable (price too high). Each of these is recoverable but each requires real user data to recover from.

68. Where the design could fail

The most likely failure mode is *too much, too fast*. NIP-0004 is internally coherent, but it is a large project. Shipping the Channel Layer before Mailbox has any users, or shipping multi-dimensional metering before there is dimensional separation in real activity, would be expensive and unrewarded. NIP-0004 §12.6 explicitly prescribes phased shipping with adoption measurement between phases. The risk is that organizational pressure to ship a "complete" version of the architecture overrides that discipline.

The second most likely failure mode is *consensus regression at scale*. Every consensus-critical addition is a future debugging surface. If two unrelated features interact in a way that produces a state-root divergence under specific block conditions, it is the kind of bug that takes weeks to find and may require mainnet emergency upgrades to fix.

The third is *protocol ossification becomes impossible*. Ethereum can freeze the EVM and push changes to L2. Ethernova, with this many protocol-level features, will continuously need protocol updates. Long-term stability — the property that makes a base layer trustworthy for high-value applications — becomes harder to achieve.

69. What must go right

Three things must go right for the Ethernova thesis to succeed.

Engineering discipline must hold. Every consensus-critical feature must remain deterministic. Every shipped phase must be measured against real adoption before the next phase ships. Every breaking-change temptation must be resisted in favor of additive evolution.

At least one killer application must emerge. A protocol does not validate itself in the abstract; it validates itself by hosting an application that users find valuable and that genuinely could not exist as cheaply or as well on another chain. The most plausible

candidates are decentralized email, real-time decentralized DMs, and channel-based games. One of them needs to work, end-to-end, and find users.

The community must remain plural. A single-implementation chain with a small contributor base is fragile. Long-term resilience depends on growing the contributor count, attracting independent reviewers, and eventually supporting at least a partial multi-client world. None of this is automatic; it requires deliberate investment in documentation (this manuscript is one piece), public testnets, and onboarding processes that work for developers who do not already know the project.

Part IX — Final synthesis

70. What Ethernova already is

Ethernova in April 2026 is a working, EVM-compatible, Proof-of-Work blockchain with one major fork shipped (the Noven Fork at block 480,000), a hardened devnet that has caught two consensus bugs before they reached mainnet, a deterministic adaptive-gas system verified identical between Linux and Windows builds, a clear set of user-visible improvements (gas refund on revert, smart-wallet recovery, batched and scheduled transactions, native tokens, FIFO ordering), and a foundation of three NIP-0004 precompiles deployed on the devnet that establish the next-generation architecture's groundwork.

It is not a finished product. It is not a hype-stage promise. It is a working chain with a coherent direction and a small but real implementation discipline.

71. What Ethernova could become

If NIP-0004 ships in disciplined phases, Ethernova could become a chain on which decentralized email, real-time chat, content-anchored social, and on-chain games are *natural* applications rather than awkward simulations. The architectural commitments — Protocol Objects as first-class state, Deferred Execution for asynchrony, Protocol Channels for low-latency interaction, multi-dimensional metering for congestion isolation, state lifecycle for bounded growth — are consistent with that goal and have, on the devnet at least, started to demonstrate that they are implementable.

This is a long road. Each layer needs to ship, be adopted, and be validated by real applications before the next layer is justified. The most credible version of "what Ethernova could become" is incremental: a chain that consistently shows up with the next useful application primitive and the engineering discipline to make it consensus-safe.

72. The practical meaning of Ethernova in April 2026

For users, Ethernova means: a small but real EVM chain where smart-wallet recovery is built in, failed transactions cost 90% less, fee-delegated onboarding works, and the FIFO mempool removes the most-visible MEV pattern.

For developers, Ethernova means: every existing Solidity contract works unchanged, nine native precompiles offer concrete savings, Tempo and Frame AA solve user-experience problems Ethereum has been debating for years, and the next-generation architecture (NIP-0004) is shipping in observable phases with public devnet testing.

For miners and node operators, Ethernova means: a Windows-friendly Ethash chain with a clear upgrade path, modest hardware requirements, and a community-driven development process.

For protocol engineers, Ethernova means: a working laboratory for the question of whether a deterministic, EVM-compatible chain can grow asynchronous primitives, channels, content addressing, and a state lifecycle without sacrificing consensus or compatibility.

73. Closing summary

The two-line summary of this entire document: **Ethernova mainnet today is a hardened EVM chain with one major fork shipped, named the Noven Fork, that delivers user-visible improvements over Ethereum. Ethernova's longer-term architecture is defined by NIP-0004, which is partially deployed on the devnet and otherwise still being built.**

The manuscript will continue to evolve. The protocol will continue to evolve. Where this document and the NIPs disagree, the NIPs are authoritative. Where the website and the NIPs disagree, the NIPs are authoritative. Where the devnet repository and earlier marketing copy disagree, the repository is authoritative.

If you are a user, the action is to add the network to your wallet and try it. If you are a developer, the action is to deploy a contract on the devnet and call a precompile. If you are a protocol engineer, the action is to read NIP-0004 in full, run a node, and either find a problem the team has missed or contribute the next phase.

The most useful thing this document can do is be honest about where the project stands today. The most useful thing the project can do is keep shipping disciplined phases that the next revision of this document can describe as live.

Appendices

Appendix A — Glossary

Adaptive Gas v2. The deterministic, trace-based version of Ethernova's gas adjustment system. Discounts up to 25% for predominantly-pure contracts, surcharges up to 10% for storage-heavy contracts. Integer-only math, no maps, no global mutable state.

Capability. A bit in a bitmask propagated through call chains in NIP-0004. Capabilities can only be narrowed, never expanded. [Specified / Planned].

Channel. A bilateral interaction protocol in NIP-0004 (Mode 3 execution). Off-chain peer-to-peer messages with on-chain checkpointing and dispute resolution.

Deferred Execution. Mode 2 execution. Effects are committed to state at block N as pending entries; processing happens deterministically at the start of block N+1.

Domain 0 / 1 / 2. Execution Domains in NIP-0004: classic synchronous (Domain 0), deferred-effect-producing (Domain 1), channel-participating (Domain 2). [Specified / Planned].

Frame AA. Account Abstraction model implemented through novaFrameApprove (0x23) and novaFrameIntrospect (0x24), inspired by EIP-8141.

Mailbox. A message-queue Protocol Object bound to an address in NIP-0004. [Specified / Planned], Phase 4 lifecycle.

Noven Fork. The mainnet activation at block 480,000 of all features specified in NIP-0002. Requires client v2.0.0+.

Protocol Object. First-class entity in the Ethernova state tree distinct from accounts and contracts, with address-space prefix 0xFF and protocol-managed lifecycle.

Protocol Ops. One of the five resource dimensions in NIP-0004's Resource Vector. Measures operations on Protocol Objects.

Resource Vector. The five-dimensional generalization of gas in NIP-0004: Compute, State Read, State Write, Protocol Ops, Proof Verify.

State Expiry v2. Mainnet-live mechanism that archives contracts inactive for 900,000 blocks using an external LevelDB index, leaving the consensus state trie unchanged.

State Lifecycle. Five-tier model in NIP-0004 (Active / Warm / Cold / Archived / Expired) for bounded state growth. [Specified / Planned] in full form.

Tempo Transaction. Transaction type 0x04. Atomic batch of up to 16 calls, optional fee delegation, optional validBefore / validAfter scheduling.

Witness. A Merkle proof against a historical state root, used to restore Cold or Archived state to Active. [Specified / Planned] (precompile 0x2F).

Appendix B — Status matrix for major features

Feature	Mainnet	Devnet	Source
Adaptive Gas v2	Live	Live (v1.1.6+)	NIP-0002 §Adaptive Gas
Per-EVM reentrancy guard	Live	Live	NIP-0002 §Reentrancy
90% gas refund on revert	Live	Live	NIP-0002 §Gas Refund
State Expiry v2 (900k blocks	Live	Live (1k devnet)	NIP-0002 §State

mainnet)			Expiry; devnet ROADMAP Phase 15
Tempo Transactions	Live	Live	NIP-0002 §Tempo
Frame AA (0x23, 0x24)	Live	Live	NIP-0002; devnet Phase 12
Anti-MEV FIFO	Live	Live	NIP-0002 §Anti-MEV
Native tokens (0x25)	Live	Live	NIP-0002; devnet Phase 20
Shielded pool (0x26)	Live	Live	NIP-0002; devnet Phase 24
Native upgrade (0x27)	Live	Live	NIP-0002; devnet Phase 21
Native oracle (0x28)	Live	Live	NIP-0002; devnet Phase 22
novaProtocolObjectRegistry (0x29)	Planned	Live	NIP-0004 Phase 1
novaDeferredQueue (0x2A)	Planned	Live	NIP-0004 Phase 2
novaContentRegistry (0x2B)	Planned	Live	NIP-0004 Phase 3
Mailbox lifecycle (0x2C)	Planned	Planned	NIP-0004 Phase 4
Session arbiter (0x2D)	Planned	Planned	NIP-0004 Phase 7
Content manifest (0x2E)	Planned	Planned	NIP-0004 Phase 11
State witness (0x2F)	Planned	Planned	NIP-0004 Phase 5
Async callback (0x30)	Planned	Planned	NIP-0004 Phase 7/11
Game state (0x31)	Planned	Planned	NIP-0004 Phase 11
Compute bounty (0x32)	Planned	Planned	NIP-0004 Phase 11
Social graph (0x33)	Planned	Planned	NIP-0004
Identity attestation (0x34)	Planned	Planned	NIP-0004
Protocol Channels	Planned	Planned	NIP-0004 §4.1 Mode 3
Capability model	Planned	Planned	NIP-0004 §4.4
Five-tier State Lifecycle (full)	Planned	Planned	NIP-0004 §5.1
Multi-Dimensional Metering	Planned	Planned	NIP-0004 §6

Appendix C — Tooling matrix

Tool / Layer	Status	Notes
Solidity 0.8.x	Compatible	Use --evm-version istanbul for cross-network compatibility
Hardhat	Compatible	Devnet config in ethernova-

		devnet/devnet/hardhat.config.js
Foundry	Compatible	No Ethernova-specific config needed for Domain 0
MetaMask	Compatible (custom network)	Tempo / Frame UI not yet native; submit via eth_sendRawTransaction
ethers.js / viem / web3.js	Compatible	Standard eth_* surface unchanged
Block explorer	Functional	explorer.ethnova.net (mainnet), devexplorer.ethnova.net (devnet)
Slither / Mythril	Compatible on Domain 0	Custom precompiles reported as unknown — cosmetic
OpenZeppelin contracts	Compatible	ReentrancyGuard no longer required for self-reentrancy
Hardhat plugin (Ethernova-specific)	Not yet available	Open contribution opportunity
Foundry plugin (Ethernova-specific)	Not yet available	Open contribution opportunity
@ethernova/sdk (client-side)	Not yet available	Open contribution opportunity
Subgraph / indexer	Not yet available	Custom indexers via eth_getLogs work today

Appendix D — Source notes and source priority

This document was written using the following sources, in priority order. Where sources conflicted, the higher-priority source was used and the conflict was noted.

1. **NIP-0004.md** (Draft, Phase 1–3 Deployed on Devnet, amended 2026-04-25). Primary source for the next-generation architecture, Protocol Objects, Deferred Execution, State Lifecycle, Resource Vector, and application primitives. Highest authority.
2. **NIP-0002.md** (Final, 2026-04-03). Primary source for the Noven Fork mainnet specification. Authoritative for Adaptive Gas v2, the nine 0x20–0x28 precompiles, reentrancy semantics, gas refund on revert, state expiry parameters, Tempo, Frame AA, anti-MEV ordering, and backward compatibility.
3. **ethernova-devnet repository** README and ROADMAP.md. Primary source for actual implementation history, phase-by-phase devnet deployment, security audit results, and current devnet status.
4. **ethernova-coregeth repository** README and Releases. Used for upstream lineage, build prerequisites, license, and historical fork timeline. Treated as partially outdated where it conflicts with NIP-0002.

5. **ethnova.net** website. Used for public-facing endpoints, exchange listings, partners, and community channels. Treated as partially outdated marketing surface where it conflicts with NIPs or repositories.

Appendix E — Contradiction notes

Five contradictions across the source set are worth recording for future maintainers.

1. **State expiry threshold.** Website states "1,000 blocks"; NIP-0002 specifies 900,000 blocks for mainnet; devnet repo specifies 1,000 blocks for the test network. Resolved: mainnet 900,000, devnet 1,000.
2. **Latest mainnet client release.** Website displays "Latest Release (v1.3.0)"; coregeth README states "v1.2.7 (only supported version)"; coregeth Releases tag is v1.3.1; NIP-0002 mandates v2.0.0+ past block 480,000. Resolved: NIP-0002 is authoritative — v2.0.0+ is required on mainnet.
3. **Coregeth README fork claims.** README references fork enforcement at block 138,396; NIP-0002 specifies the Noven Fork at block 480,000. Resolved: the README documents the pre-Noven v1.x lineage; the Noven Fork is a later, separate activation.
4. **NIP-0004 precompile slot assignments.** Original draft put novaMailboxManager at 0x29, novaContentRegistry at 0x2A, novaSocialGraph at 0x2B; the 2026-04-25 amendment relocates them. Resolved: amended assignments used throughout.
5. **Devnet adaptive-gas runtime status.** Devnet v1.0.2 README says gas modifications are disabled; v1.1.6 chaos test confirms they are active and deterministic; NIP-0002 mainnet specifies adaptive gas v2 active. Resolved: not a true conflict — v1.1.6 reactivated adaptive gas with the deterministic implementation that became the mainnet specification.

Appendix F — Quick reference tables

Mainnet network. Chain ID 121525. RPC https://rpc.ethnova.net. Explorer https://explorer.ethnova.net. Currency NOVA. Consensus Ethash PoW. Block reward 10 NOVA.

Devnet network. Chain ID 121526. RPC https://devrpc.ethnova.net. Explorer https://devexplorer.ethnova.net. Faucet https://faucet.ethnova.net. Same currency, consensus, and reward.

Native precompiles (mainnet). 0x20 novaBatchHash 30/item. 0x21 novaBatchVerify 2,000/sig. 0x22 novaAccountManager 2k–10k. 0x23 novaFrameApprove 5,000. 0x24 novaFrameIntrospect 2,000. 0x25 novaTokenManager 1k–500k. 0x26 novaShieldedPool 50k–100k. 0x27 novaContractUpgrade 50,000. 0x28 novaOracle 2k–5k.

Devnet-only NIP-0004 precompiles. 0x29 novaProtocolObjectRegistry. 0x2A novaDeferredQueue. 0x2B novaContentRegistry.

RPC namespaces. Standard eth_*, net_*, web3_* on both networks. Devnet-specific: ethernova_forkStatus, ethernova_chainConfig, ethernova_nodeHealth, ethernova_evmProfile, ethernova_adaptiveGas, ethernova_optimizer, ethernova_callCache, ethernova_precompiles, ethernova_executionMode, ethernova_tempoConfig, ethernova_stateExpiry. Mainnet adds nova_* namespace as NIP-0004 phases ship.

Tempo transaction envelope. Type 0x04. Up to 16 calls. Optional feePayer (co-signed). Optional validBefore, validAfter (max 500-block window).

Appendix G — Getting-started checklist for users

1. Install MetaMask or another EVM wallet that supports custom networks.
2. Add Ethernova mainnet: name "Ethernova", RPC https://rpc.ethnova.net, chain ID 121525, currency NOVA, explorer https://explorer.ethnova.net.
3. Acquire NOVA via Gatevia or KlingEx, withdraw to your wallet.
4. (Optional) Configure guardian recovery on novaAccountManager (0x22) so that the account is recoverable if you lose access.
5. Confirm transactions appear in https://explorer.ethnova.net.
6. Try a Tempo batched transaction (e.g., approve + transfer in one all-or-nothing call) through any dApp that supports it.
7. (Optional) Switch to the devnet (chain ID 121526, RPC https://devrpc.ethnova.net) and request 10 test NOVA from https://faucet.ethnova.net to experiment safely.

Appendix H — Getting-started checklist for developers

1. Read NIP-0002 and NIP-0004 in full.
2. Clone ethernova-devnet for the included Hardhat configuration and reference contracts.
3. Compile and deploy a standard Solidity contract to the devnet using npx hardhat run scripts/deploy.js --network ethernova_devnet.
4. Call one precompile from your contract — novaBatchHash (0x20) is the simplest.
5. Submit a Tempo transaction (type 0x04) and observe atomic-batch behavior.
6. Read the devnet ROADMAP.md for the current Phase 1–3 NIP-0004 status.
7. Test against novaProtocolObjectRegistry (0x29) and novaContentRegistry (0x2B) on the devnet to develop intuition for the next-generation architecture.

8. Switch the network to mainnet (chain ID 121525, RPC https://rpc.ethnova.net) only after the application has passed devnet testing.
9. When deploying to mainnet, verify your client is v2.0.0 or higher, and verify the genesis hash matches 0xc3812eb81498965a3f9ff3e73d2f423934e6d440578d4f4fbb6623cc61c453d9.
10. Subscribe to Telegram and Discord for upgrade announcements; mainnet hard-fork upgrades are mandatory.