# Atrium: A Skill Provenance and Royalty Marketplace for AI Agents

Cryptographic authorship, content addressing, and per-invocation settlement on Base

The Atrium Authors

Version 0.1 — May 2026

**Abstract**

AI agent runtimes increasingly generate reusable *skills* — self-contained, executable capabilities expressed as Markdown with structured metadata. Today these skills are trapped inside the runtime that produced them: there is no portable record of who authored a skill, no way to verify what a consumer is about to run, and no mechanism to compensate authors when their work is reused. Atrium is an economic and identity layer that addresses these gaps without becoming another agent runtime. Each skill is signed with a decentralized identifier, content-addressed on IPFS, and priced for per-invocation use; a small non-upgradeable smart contract on Base routes USDC payments to creators and, for derived skills, to declared ancestors. This paper specifies the skill manifest format, the on-chain settlement contract and its conservation invariants, the royalty model, the threat model, and the off-chain indexing and interface architecture. It argues that a tokenless, per-invocation design is the right primitive for an emerging skill economy, and situates Atrium against package registries, model hubs, and centralized app stores.

## Contents

## Status of this document

This is a technical specification and design rationale for Atrium, version 0.1. It describes a system that is implemented as a minimum viable product: a deployed contract design, a reference command-line client, a Model Context Protocol server, an event indexer, and a web interface. Numerical parameters (fees, royalty caps, gas figures) reflect the v0.1 contract and are subject to the governance process described in Chapter 7. Where this document makes empirical claims about external systems, they are cited; where it describes Atrium's own behavior, the smart contract source is the authoritative reference.

# 1 Introduction

## 1.1 The skill economy

A new unit of software is emerging. As AI agents move from one-shot prompting to long-running, self-improving loops, they increasingly externalize what they learn as *skills*: self-contained, executable capabilities expressed as Markdown with structured metadata and, optionally, code. A skill captures a repeatable procedure — extracting tables from PDFs, refactoring a TypeScript module, optimizing a SQL query — in a form another agent can discover and run.

Agent runtimes already produce these artifacts organically. Hermes Agent's closed learning loop, for example, distills successful tool-use episodes into reusable skill files written to the local filesystem (Nous Research 2026). The Model Context Protocol has standardized how agents *call* external capabilities at runtime (Anthropic 2024), and formats such as agentskills.io have begun to standardize how skills are *written* (agentskills.io 2025). The raw material of a skill marketplace exists.

What does not exist is the connective tissue. A skill generated inside one runtime is trapped there. There is no portable, verifiable record of who authored it; no guarantee that the bytes a consumer fetches are the bytes the author published; and no mechanism to compensate an author when their skill is reused by someone else's agent. The value a skill represents — sometimes the product of thousands of tokens of agent reasoning — stays locked to the machine that produced it.

## 1.2 What is missing

Three primitives are missing, and they are economic and cryptographic rather than computational:

1. **Provenance.** A durable, verifiable claim of authorship that survives across runtimes and outlives any single wallet or device.
2. **Integrity.** A way for a consumer to confirm that the skill they are paying for is exactly the skill the author signed — no substitution, no tampering.
3. **Settlement.** A way to pay an author per use, and to split that payment with the authors of any skills a derived work builds upon.

These are precisely the primitives that package registries (npm, PyPI), model hubs (Hugging Face), and app stores (the GPT Store) only partially provide, and that none provide together with cryptographic authorship and a built-in royalty mechanism (Chapter 8).

## 1.3 Atrium

Atrium supplies these three primitives as a thin layer beneath existing runtimes. It is explicitly **not** a new agent runtime, model marketplace, or identity issuer. A skill published to Atrium is:

1. **Signed** with a decentralized identifier (DID), binding it to its author (Chapter 3);
2. **Content-addressed** on IPFS, so its identifier is a hash of its bytes (Chapter 3);
3. **Priced per invocation** in USDC, settled by a small non-upgradeable contract on Base (Chapters 4–5);
4. **Composable**, so a derived skill can declare its ancestors and route a share of each invocation back to them (Chapter 5);
5. **Attestable**, so anyone can post an on-chain claim that a skill passes a named benchmark at a given success rate (Chapter 4).

The mental model that motivates every design decision is a single analogy: *a skill is to an AI agent what an npm package is to a Node.js program — but signed by its author, addressed by its*

*content, and paid per call.* The remainder of this paper makes that analogy precise. Chapters 3–4 specify the protocol and contract; Chapters 5–6 cover economics and security; Chapter 7 covers deployment and governance; Chapters 8–9 compare Atrium to prior systems and outline open problems. # Prior Work

Atrium sits at the intersection of agent runtimes, identity systems, agent payment rails, and software registries. This chapter surveys each and locates the gap Atrium fills.

### 1.4 Agent runtimes that generate skills

**Hermes Agent** (Nous Research 2026) is an open-source, self-improving agent runtime built around a closed learning loop and a layered memory system. When the loop succeeds at a task, it can distill the episode into a skill file on disk. Hermes is the canonical *producer* of skills for Atrium: the auto-publish plugin described in Chapter 7 watches Hermes' skill directory and offers to publish new skills. Atrium treats runtimes like Hermes as partners, not competitors — it captures the value their loops create without replacing the loops.

**OpenClaude** (gitlawb 2026) is an open agent harness with first-class decentralized identity, developed in the gitlawb ecosystem. Atrium reuses the same DID identity primitives, which makes skills authored under either system mutually verifiable. gitlawb's decentralized storage is a candidate backend for a future Atrium storage tier.

### 1.5 Calling versus publishing skills

The **Model Context Protocol (MCP)** (Anthropic 2024) standardizes how an agent *invokes* external tools and data sources at runtime. Atrium ships an MCP server so that any MCP-capable agent can discover and invoke Atrium skills as ordinary tools; MCP is the runtime-facing edge of the system. Where MCP standardizes the *call*, **agentskills.io** (agentskills.io 2025) standardizes the *artifact* — a portable skill file format. Atrium's manifest is a superset of that format: it adds the identity, economic, and provenance fields a marketplace requires while remaining readable by plain agentskills.io consumers.

### 1.6 Agent payment rails

**x402** (Coinbase 2025) revives the dormant HTTP 402 status code as a payment protocol, carrying stablecoin payments in an `X-PAYMENT` header so that an API can charge per request. **Base MCP** and Coinbase's smart-wallet stack give agents programmatic custody on Base. Atrium is complementary: it is an *application* of these rails specialized to skills, defining what is being paid for (a content-addressed, signed capability) and how the payment is split (the royalty cascade), rather than inventing a new transport.

### 1.7 Software and model registries

Existing registries each solve part of the problem:

- **npm / PyPI** (npm, Inc. 2024) provide discovery and versioning for code packages but no cryptographic authorship binding and no payment primitive — publishing is free and anonymous-by-default.
- **Hugging Face Hub** (Hugging Face 2024) hosts models and datasets with rich metadata and social features, but has no native per-use economic primitive and no on-chain provenance.
- **The GPT Store** (OpenAI 2024) offers discovery and a centralized revenue-share program, but is a closed, single-vendor platform: listings, payments, and takedowns are all controlled by one operator.

None combine portable cryptographic provenance, content integrity, and built-in per-invocation royalties across runtimes. Chapter 8 returns to this comparison in tabular form. Atrium's contribution is not any single primitive — DIDs, IPFS, and ERC-20 settlement all predate it — but their composition into a minimal, tokenless protocol purpose-built for the skill economy. # Protocol

Atrium's off-chain protocol defines what a skill *is*, how authorship is proven, and how a skill's bytes are named. The on-chain contract (Chapter 4) only ever sees three derived values: a content identifier, a hash of the author's DID, and a price. Everything else lives off-chain, which keeps the contract small and the system censorship-resistant (Figure 1).

## 1.8 The skill manifest

A skill is a Markdown file (or a directory containing `skill.md` plus optional assets such as `benchmark.json`). YAML frontmatter carries the manifest; the Markdown body is the prompt, decision procedure, or code specification the agent reads. The manifest is a superset of the agentskills.io format, grouped into five concerns:

| Group | Fields | Purpose |
|---|---|---|
| Identity | `name`, `version`, `author_did`, `author_signature` | Who, what, which release |
| Discovery | `description`, `tags`, `categories`, `language` | Search and classification |
| Execution | `runtime`, `entrypoint`, `requires`, `inputs`, `outputs` | How to run it |
| Economics | `price_per_call_usdc`, `parent_skills[]` | Price and royalty splits |
| Provenance | `created_at`, `derivation_method`, `hermes_session`, `openclaude_version` | How it was produced |

Prices are written as decimal USDC strings (e.g. `"0.005"`) to avoid floating-point loss and are converted to base units — USDC has **six** decimals — at the boundary. Each entry in `parent_skills` is a `(skill_id, royalty_bps)` pair, where basis points are hundredths of a percent.

## 1.9 Identity: did:key over Ed25519

Authors are identified by a W3C decentralized identifier (W3C 2022), specifically the `did:key` method over an Ed25519 keypair (W3C Credentials Community Group 2022; Bernstein et al. 2011). The DID is derived by prefixing the 32-byte public key with the Ed25519 multicodec (`0xed01`) and base58-encoding the result: `did:key:z<base58(0xed01 ‖ pubkey)>`.

Identity is deliberately separated from the on-chain wallet. Authorship is durable; wallets rotate. A creator may migrate from one EVM wallet to another — after losing a device or moving to account abstraction — without losing the identity their reputation accrues to. The same DID could later be linked to a non-EVM chain, giving authorship cross-chain portability.

## 1.10 Content addressing

The skill bundle is pinned to IPFS (Protocol Labs 2024), yielding a CIDv1 content identifier. Because the CID is a hash of the bytes, *identity equals integrity*: if the body changes, the CID

Figure 1: Atrium architecture: an author signs and pins a skill and registers it on Base; an indexer mirrors events and caches bodies; consumers discover and invoke via MCP or the web.

changes, so a consumer can verify that what they fetched is what they are paying for. Pinning is replicated, so removing a skill requires coordinated action rather than a single operator's decision. The reference implementation uses Pinata as the primary pinning service for latency and reliability; the indexer (Chapter 7) rotates across public gateways (`ipfs.io`, `dweb.link`, `w3s.link`) when fetching bodies for its cache.

### 1.11 Canonical hashing and signatures

To sign a skill, the author computes a canonical hash over the fields that define it economically and semantically: the UTF-8 concatenation of `name`, `version`, the DID, the body CID, the price in base units (as a decimal string), and the parent list serialized as `id:bps` pairs joined by commas. The SHA-256 digest of that concatenation is signed with the author's Ed25519 key, and the signature is stored in the manifest as `author_signature = "ed25519:0x<sig>"`. Any party can verify the signature against the DID's embedded public key without touching the chain.

### 1.12 Deriving the on-chain identifier

Two values bridge to the contract. The **DID hash** is the SHA-256 digest of the DID string, stored on-chain as a `bytes32` because the full DID is too long to store economically. The **skill identifier** is then deterministic:

$$\text{skillId} = \text{keccak256}(\text{abi.encodePacked}(\text{cid},\ \text{didHash},\ \text{creator}))$$

Because the inputs are fixed before submission, the author (and any client) can predict the `skillId` off-chain. The same content, from the same author, at the same wallet, always yields the same identifier — and re-registering it is rejected by the contract (Chapter 4). # The AtriumRegistry Contract

The settlement layer is a single, non-upgradeable Solidity contract of roughly 300 lines, built and tested with Foundry (Foundry 2024). It does three things: it records skills, it routes payments, and it stores benchmark attestations. It holds USDC only transiently and never custodies an author's earnings beyond an internal ledger.

### 1.13 State and lifecycle

Each skill is stored as a `Skill` struct keyed by its `skillId`: the IPFS `cid`, the `creator` address (the withdrawal destination), the `didHash`, the `pricePerCall` (USDC base units), the parent arrays (`parentSkills` and `parentBps`), bookkeeping counters (`createdAt`, `lastInvoked`, `totalInvocations`, `totalEarned`), and an `active` flag. Two auxiliary indexes — an append-only `allSkills` array and a `skillsByCreator` mapping — support pagination and creator lookups; an off-chain indexer mirrors both (Chapter 7). Earnings accrue in a single `withdrawable[address]` ledger shared by creators, parents, and the protocol treasury.

The lifecycle has five entry points:

1. `registerSkill(cid, didHash, price, parentSkills, parentBps)` — validates and records a skill, emitting `SkillRegistered`.
2. `invokeSkill(skillId)` — pays to run a skill, splitting the payment.
3. `attestBenchmark(skillId, hash, successRate, sampleCount)` — records a capability claim.
4. `deactivateSkill(skillId)` — delists a skill (creator only).
5. `withdraw()` — claims accumulated USDC.

### 1.14 Payment routing

`invokeSkill` is the heart of the contract (Figure 2). The caller must have approved the price in USDC. The contract pulls the payment, takes the protocol fee first, distributes declared parent royalties from the remainder, and credits the residual to the creator:

```
uint256 protocolCut   = (price * protocolFeeBps) / 10000;   // 250 bps = 2.5%
uint256 distributable = price - protocolCut;
uint256 toCreator     = distributable;
for (uint i; i < skill.parentSkills.length; ++i) {
    uint256 parentCut = (distributable * skill.parentBps[i]) / 10000;
    withdrawable[parent.creator] += parentCut;   // credit, do not send
    toCreator -= parentCut;
}
withdrawable[skill.creator] += toCreator;
```

No funds are *sent* during `invokeSkill`. Recipients are only *credited*; they later pull their balance with `withdraw`. This pull-payment pattern means a malicious or contract-based recipient cannot block an invocation by reverting on receipt, and it confines value transfer to two well-defined points.

### 1.15 Invariants

The contract is designed around invariants that must hold across every change.

| # | Invariant | Enforcement |
|---|---|---|
| 1 | **Conservation of funds** — credits from one invocation sum to exactly the price | Integer arithmetic; residual assigned to creator |
| 2 | **Pull payments** — external calls only on `transferFrom` (in) and `transfer` (withdraw) | No sends during state mutation |
| 3 | **Royalty cap** — $\Sigma$ `parentBps` $\leq$ 5000 (50%) | Checked in `registerSkill` |
| 4 | **Deterministic id** — `skillId = keccak256(cid, didHash, creator)` | Pure derivation; re-register reverts |
| 5 | **Active-parent** — cannot derive from an inactive parent | Checked in `registerSkill` |
| 6 | **One-way deactivation** — a skill cannot be reactivated | No reactivation path exists |
| 7 | **Fee ceiling** — protocol fee $\leq$ 500 bps (5%) | Hard-coded constant bound |
| 8 | **No reentrancy gain** — re-entry cannot double-spend the ledger | Pull pattern + checks-effects |

Invariant 1 is the load-bearing one: for a payment $P$ with fee rate $f$ and parent rates $b_i$, the credits are $Pf$ to the treasury (after flooring), $\lfloor (P - \lfloor Pf \rfloor)\, b_i \rfloor$ to each parent, and the remainder to the creator. Because the creator receives the *remainder* rather than an independently floored share, integer truncation can never create or destroy a base unit: the sum is identically $P$.

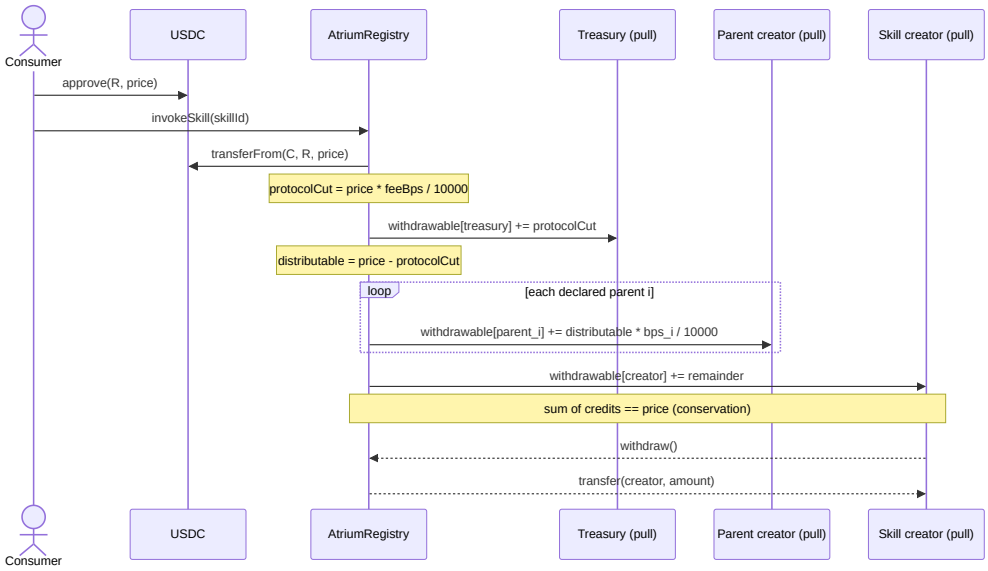


Figure 2: Payment flow for a single invocation: pull once, credit many, send on withdraw.

### 1.16 Gas profile

The contract's cost is dominated by storage. `registerSkill` performs several cold `SSTORE`s (the struct fields plus the two indexes), so its cost scales with the parent count and the CID length; it is the most expensive operation but is paid once per skill. `invokeSkill` is $O$(parents): one ERC-20 `transferFrom`, one fee credit, one credit per declared parent, and the counter updates — independent of derivation *depth* (Chapter 5). `withdraw` is a single `transfer` plus a zeroing write. These figures are order-of-magnitude descriptions of the reference implementation rather than audited measurements; on Base, where gas is consistently sub-cent, even the registration cost is negligible relative to a skill's lifetime invocation revenue.

### 1.17 Upgrade path

The contract is intentionally **non-upgradeable**. There is no proxy, no admin key over user funds, and no function that can rewrite a registered skill — the absence of an update path is what makes invariants 3–6 unconditional. The owner's powers are confined and bounded: adjusting the protocol fee within the 5% ceiling and changing the treasury address. Evolving the protocol means deploying a new contract and letting the indexer and clients track both, not mutating deployed state. This trades operational convenience for a credible guarantee that the rules a creator published under cannot change beneath them. # Economics

Atrium's economic design follows from one choice — pricing per invocation — and one constraint — that the mechanism must be implementable entirely on-chain with bounded cost. This chapter motivates per-invocation pricing, works through the royalty cascade, justifies the protocol fee, and explains the tokenless thesis.

### 1.18 Why per-invocation pricing

A skill could be sold by subscription, as a one-time purchase, or per use. Atrium chooses per use for four reasons:

1. **Aligned incentives.** A creator earns when value is delivered, not when a promise is made. A skill that is rarely useful earns rarely.
2. **Composability.** The royalty cascade requires a settlement event each time value flows; only per-invocation pricing produces one.
3. **Runtime fit.** Agents call skills one at a time. Per-invocation pricing matches the natural granularity of an MCP tool call (Anthropic 2024).
4. **On-chain feasibility.** Subscriptions require off-chain billing state and renewal logic; per-invocation settlement runs in a single transaction with no trusted billing operator.

The cost is sensitivity to gas: if the gas to invoke ever exceeds the skill price, low-priced skills become uneconomic to call. Base's sub-cent gas makes \$0.001-scale prices viable; Chapter 6 notes the residual risk during gas spikes.

### 1.19 The royalty cascade, worked

Consider a derivation chain $A \to B \to C$ (Figure 3). Carol authors $A$; Alice derives $B$ from $A$ and declares $A$ as a parent at 20%; Dave derives $C$ from $B$ and declares $B$ as a parent at 15%. Suppose Dave prices $C$ at $P$ and the protocol fee is 2.5%.

When a consumer invokes $C$:

- the treasury receives $0.025P$;
- the distributable amount is $0.975P$;
- Alice (as author of parent $B$) receives $0.15 \times 0.975P = 0.146P$;

- Dave receives the remainder, $0.829P$;
- **Carol receives nothing** — unless Dave *also* declares $A$ as a parent of $C$.

This non-transitivity is deliberate. Royalties are paid only to *declared, direct* parents, never propagated up an inferred ancestry. The consequences are:

- **Predictability.** Each author can compute their exact per-invocation revenue from their own manifest, with no dependence on what their descendants do.
- **Bounded cost.** A payment is $O$(declared parents), never $O$(depth); derivation chains can be arbitrarily deep without raising the gas of any single invocation, removing a denial-of-service vector.
- **Honest accounting.** Compensating an ancestor requires explicitly declaring it, which prevents accidental royalty leakage. If Alice wants Carol to share in $C$'s revenue, the social and tooling answer is for Dave to declare both $B$ and $A$ as parents — Atrium's clients can surface this suggestion.

The 50% cap on combined parent royalties (invariant 3) guarantees a derived skill's author always retains at least half of the distributable amount, preserving the incentive to build derivatives at all.

### 1.20 The protocol fee

Atrium takes a flat 2.5% (250 bps) of each invocation, hard-capped at 5% (500 bps) by the contract. The fee funds indexing, gateways, and protocol maintenance. Its level is best understood comparatively (Chapter 8): the GPT Store and app stores take 15–30% (OpenAI 2024), while npm takes nothing but offers no payment rail at all (npm, Inc. 2024). A 2.5% take positions Atrium far below platform incumbents while remaining sufficient to operate the public-good infrastructure. The 5% ceiling is a credible commitment: because the contract is non-upgradeable, no future operator can raise the fee past it.

### 1.21 The tokenless thesis

Atrium has no native token, by design. Settlement is in USDC (Circle 2024), a regulated stablecoin, for three reasons. **Creators want dollars,** not exposure to a volatile asset whose price is unrelated to their work. **Regulatory clarity:** denominating in an existing regulated instrument avoids the question of whether a new token is a security. **No bootstrap problem:** adopting money that already exists means day-one usability with no liquidity-mining subsidy.

More fundamentally, Atrium has no function that a token would serve better than USDC. There is no governance question that requires a vote-weighted token, no liquidity that needs incentivizing, and no staking that the current design depends on. Tokens add legal complexity, distribution problems, and dilution risk. A token can always be added later if a concrete need emerges — for example, the stake-based attestation scheme sketched in Chapter 7 — whereas a token, once issued, is very hard to remove. The default is therefore to ship without one. # Security and Trust Model

Atrium's trust model is shaped by what is on-chain (settlement and provenance hashes), what is off-chain (skill bodies and signatures), and what is explicitly out of scope (the safety of executing a skill). This chapter states the threat model, the mitigations, and the open issues honestly.

### 1.22 Actors and assets

The protected assets are funds in transit through `invokeSkill`/`withdraw`, the integrity of the authorship binding, and the availability of skill bodies. The relevant actors are a malicious

creator, a malicious consumer, a network-level adversary (MEV or censorship), and the protocol operator itself.

## 1.23 Threats and mitigations

**Malicious creator.** A creator might try to substitute a skill's body after sale, claim someone else's authorship, or grief consumers. Body substitution is prevented by content addressing: the `skillId` and the price commitment are bound to a specific CID, so changing the body changes the identifier (Chapter 3). False authorship is prevented by the Ed25519 signature over the canonical hash, verifiable against the DID. A creator cannot retroactively raise a price or rewrite terms, because the contract has no update path (Chapter 4).

**Malicious consumer.** A consumer might attempt to pay less than the price, or to re-enter the contract during settlement. The price is read from contract storage, not supplied by the caller, so underpayment is impossible. The pull-payment pattern plus checks-effects ordering means re-entry yields no double-spend: balances are credited to an internal ledger and only sent during `withdraw`, which zeroes the balance before transferring (invariants 2 and 8).

**Network adversary (MEV, censorship).** Invocations are independent value transfers with no ordering-dependent profit, so they present little maximal-extractable-value surface. Base is an L2 with a sequencer; a censoring sequencer could delay transactions, but cannot forge authorship or redirect funds, and the underlying settlement inherits Ethereum's guarantees.

**Malicious operator.** The protocol owner can change the fee (within the 5% cap) and the treasury address — and nothing else. The owner cannot move user funds, deactivate others' skills, alter registrations, or upgrade the contract. This is the strongest mitigation the design offers: most operator-abuse vectors simply do not exist because the corresponding functions were never written.

## 1.24 Reorgs and finality

Reorganizations on Base are rare but possible. The indexer (Chapter 7) commits each block's events and its cursor in a single transaction and resumes from the last committed block on restart, so it never double-counts; for the MVP it treats shallow confirmations as sufficient and documents the residual risk rather than over-engineering deep-reorg handling. Base's Azul upgrade and its multiproof finality (Chapter 7) shorten the window in which a reorg is even conceivable.

## 1.25 Open issues

Three problems are unsolved in v0.1 and are stated plainly:

1. **Content availability.** Deactivating a skill on-chain prevents new invocations but cannot remove its bytes from IPFS; conversely, if every pinner drops a CID, the body becomes unreachable even though the on-chain record persists. Mitigations are replication and gateway rotation, not guarantees.
2. **Free-riding via fetch.** Because bodies are public on IPFS, a consumer can read a skill without paying. Atrium charges for *invocation as settlement*, not for read access; encrypted bodies with per-invocation decryption keys are a research direction (Chapter 9). Today the incentive to pay is honesty, attestation-backed trust, and integration convenience.
3. **Sybil attestation.** In the MVP anyone may post a benchmark attestation, so attestations are claims, not proofs. Consumers should weight them by attester reputation. Stake-based or zero-knowledge-verified attestation (Chapters 7, 9) is the intended hardening.

A final, important boundary: Atrium makes no claim about the *safety of running* a skill. A skill body is code or instructions an agent executes; provenance tells you *who* wrote it and integrity tells you *that it is unchanged*, but neither tells you it is *safe*. Sandboxing and capability control belong to the runtime, not to the marketplace. # Deployment, Infrastructure, and Governance

## 1.26 Why Base

Atrium settles on Base (Coinbase 2024), Coinbase's Ethereum L2, for reasons that are specific rather than generic:

- **Native USDC.** Circle issues USDC directly on Base (Circle 2024), so settlement needs no bridge and carries no bridge risk.
- **Finality and withdrawals.** The Azul network upgrade introduces multiproof finality (Coinbase 2026), compressing the standard rollup withdrawal window from roughly seven days toward one — material for creator cash flow.
- **Throughput and cost.** High sustained throughput and consistently sub-cent gas are what make $0.001-scale per-invocation pricing economically coherent (Chapter 5).
- **Ecosystem.** Agent-facing infrastructure — Base MCP, x402 (Coinbase 2025), smart wallets — is concentrated on Base, so Atrium's consumers and the rails they use already live there.

Alternatives were considered and rejected for this stage: Ethereum mainnet's gas makes micropayments impossible, and Solana's USDC liquidity fragmentation and less mature smart-wallet account-abstraction story outweighed its throughput advantage for a Solidity-based, money-handling contract.

## 1.27 Off-chain infrastructure

Three off-chain components surround the contract (Figure 1):

- **Indexer.** A single-process daemon subscribes to `AtriumRegistry` events, writes them to SQLite (with a full-text index over cached skill bodies), and serves a read-only REST API. It exists so that interfaces never query a chain RPC per page load. It is strictly read-only — it holds no keys and cannot move funds — and the chain remains the source of truth; the indexer caches and reconciles.
- **Web interface.** A server-rendered application reads exclusively from the indexer for fast, inexpensive loads, and writes (invoke, withdraw, publish, deactivate) through the user's browser wallet.
- **Hermes auto-publish plugin.** A sidecar that watches a Hermes runtime's skill directory and offers to publish new skills, delegating all signing and chain logic to the reference CLI rather than reimplementing it — keeping a single source of truth for the security-critical paths.

## 1.28 Governance

At launch the contract owner is a founder-controlled multisig, but the *surface* of that control is deliberately tiny: the owner may set the protocol fee within the hard 5% ceiling and change the treasury address, and may do nothing else (Chapter 4). User funds, registrations, and the rule set are outside any admin's reach by construction.

The transition plan is to move these two parameters under community control as usage grows — first a published policy and timelock on fee changes, then broader participation. Crucially, because the protocol is tokenless (Chapter 5), governance is about a small, bounded parameter

set rather than the distribution of a financial instrument, which keeps the transition a matter of process rather than tokenomics.

### 1.29 Roadmap

The near-term roadmap has two anchors. In **Q3 2026**, multi-chain settlement: deploy the same contract to additional USDC-native chains and let DIDs carry reputation across them. In **Q4 2026**, **stake-based attestation**: replace the permissionless attestation of the MVP with a scheme in which attesters bond collateral and can be slashed for false claims, turning attestations from claims into economically backed signals (Chapters 6, 9). Both are additive; neither requires changing the core settlement contract. # Comparison

Atrium's primitives exist individually in prior systems; its contribution is their composition. The table below compares Atrium to the registries and stores that host reusable software or models today, across the dimensions that matter for a skill economy.

| Dimension | npm / PyPI (npm, Inc. 2024) | Hugging Face Hub (Hugging Face 2024) | GPT Store (OpenAI 2024) | agentskills.io (agentskills.io 2025) | **Atrium** |
|---|---|---|---|---|---|
| Cryptographic provenance | No | Partial (commit signing) | No | No | **Yes (DID + signature)** |
| Content integrity | Hash in lockfile | Repo hash | Opaque | File-level | **CIDv1 = identity** |
| Per-use economics | None | None | Centralized share | None | **Per-invocation USDC** |
| Royalty / derivation splits | No | No | No | No | **Yes (declared cascade)** |
| Decentralization | Central registry | Central hub | Single vendor | Format only | **Contract + IPFS** |
| Runtime compatibility | Node / Python | ML frameworks | OpenAI only | Cross-runtime | **Cross-runtime (MCP)** |
| Governance of rules | Operator | Operator | Operator | Community spec | **Bounded, non-upgradeable** |
| Platform take rate | 0% (no payments) | 0% (no payments) | 15–30% | n/a | **2.5% ( 5% capped)** |

Two rows deserve emphasis. First, *economics with provenance*: npm and PyPI prove neither authorship nor offer payment; the GPT Store offers payment but only within a single vendor's walled garden. Atrium is the only entry that provides cryptographic authorship *and* a built-in payment-and-royalty rail *across* runtimes. Second, *take rate*: Atrium's 2.5% sits an order of magnitude below the 15–30% of app-store incumbents, made possible by pushing settlement onto a low-cost L2 rather than operating centralized billing.

The comparison also clarifies what Atrium is *not* competing on. It does not host or rank as richly as Hugging Face, nor match npm's decade of tooling. It competes on a single axis those

systems leave open: portable, verifiable, monetizable authorship for the specific artifact class of agent skills. # Future Work

Atrium v0.1 is deliberately minimal. Several directions would extend it without disturbing the core settlement contract; each addresses an open issue raised earlier.

### 1.30 Encrypted bodies with per-invocation decryption

The most important economic gap is free-riding via fetch (Chapter 6): public IPFS bodies can be read without paying. A natural fix is to pin an *encrypted* body and release a decryption key only as part of a paid invocation — for instance, via threshold decryption or a key-management network keyed on an on-chain payment receipt. The challenge is doing so without reintroducing a trusted operator or breaking the content-addressing guarantee that ties price to bytes.

### 1.31 Zero-knowledge proofs of benchmark execution

Attestations today are unproven claims. A stronger primitive is a succinct proof that a skill was run against a named benchmark suite and achieved a stated success rate — a zero-knowledge proof of correct execution over the committed `(test_case_id, pass/fail)` Merkle tree. This would turn the success-rate field from a reputation signal into a verifiable fact, and complements (rather than replaces) the stake-based attestation on the roadmap.

### 1.32 Cross-chain reputation portability

Because authorship is a DID rather than a wallet (Chapter 3), an author's reputation — invocation counts, earnings, attestations — is in principle portable across chains. Multi-chain deployment (Chapter 7) raises the question of how to aggregate a single DID's activity across several `AtriumRegistry` instances into one coherent, verifiable reputation, without a central aggregator becoming a trust bottleneck.

### 1.33 Skill genealogy as a graph

The declared parent relationships form a directed acyclic graph of derivation. Analyzing this genealogy at scale could surface which foundational skills generate the most downstream value, inform royalty recommendations (e.g. suggesting that a deriver declare an ancestor, per Chapter 5), and reveal how capability evolves across an agent ecosystem. This is as much a research instrument as a product feature: the genealogy is a novel dataset on how machine-generated capability compounds.

### 1.34 Closing

The thesis of this paper is narrow on purpose. Agents are beginning to produce durable, reusable capability, and that capability deserves the same primitives we long ago gave to code and to money: a verifiable author, an immutable address, and a way to pay for use. Atrium is an attempt to provide exactly those three things — no more — and to let existing runtimes, identities, and payment rails do everything else.

## 2 References


agentskills.io. 2025. “Agentskills.io: A Portable Skill Format for AI Agents.” https://agentskills.io.

Anthropic. 2024. “Model Context Protocol.” https://modelcontextprotocol.io.

Bernstein, Daniel J. et al. 2011. “Ed25519: High-Speed High-Security Signatures.” https://ed25519.cr.yp.to.

Circle. 2024. “USDC: A Digital Dollar.” https://www.circle.com/usdc.

Coinbase. 2024. “Base.” https://base.org.

Coinbase. 2025. “X402: An HTTP-Native Payments Protocol.” https://x402.org.

Coinbase. 2026. “Base Network Upgrade: Azul.” https://base.org/blog.

Foundry. 2024. “Foundry: A Toolkit for Ethereum Application Development.” https://book.getfoundry.sh.

gitlawb. 2026. “OpenClaude.” https://github.com/gitlawb.

Hugging Face. 2024. “Hugging Face Hub.” https://huggingface.co/docs/hub.

Nous Research. 2026. “Hermes Agent.” https://hermes-agent.nousresearch.com/docs/.

npm, Inc. 2024. “Npm Registry.” https://docs.npmjs.com.

OpenAI. 2024. “GPT Store.” https://openai.com/index/introducing-the-gpt-store/.

Protocol Labs. 2024. “IPFS: The InterPlanetary File System.” https://docs.ipfs.tech.

W3C. 2022. “Decentralized Identifiers (DIDs) V1.0.” https://www.w3.org/TR/did-core/.

W3C Credentials Community Group. 2022. “The Did:key Method.” https://w3c-ccg.github.io/did-method-key/.