Compositional *Computing*

The architectural commitments a system makes determine what it can become.

Jim Calhoun
Founder, The Grove Foundation
LinkedIn · the-grove.ai

THE GROVE FOUNDATION · MEMBER DRAFT · JULY 2026

ABSTRACT

ABSTRACT

The dominant AI architecture — orchestration — treats governance as a feature bolted on after the system ships. It works until it doesn't, and the "intelligence" never compounds. This paper describes an alternative set of commitments: strict compositionality at the substrate, the human as an adjudicative surface rather than a brake, and parallel-time information processing where multiple actors tap shared knowledge simultaneously because the composition rules are structural. Together they produce a paradigm — *compositional computing* — where governance, auditability, model independence, and cost collapse are not features but consequences of the architecture, and where every interaction, successful or failed, compiles into a durable asset the operator owns. The architecture determines which side of the firewall the value accrues. These commitments make the paradigm structurally inevitable.

KEYWORDS: compositional computing, compositionality, orchestration, cognitive capture, DEX, provenance, model independence, thin waist, invariant pipeline, grant token, variance collapse, progressive ratchet, Jevons paradox, negative findings

I

Composition Is Safe by *Construction*, Not by Policy

Compositional computing is a paradigm where humans and AI agents tap shared knowledge under strict, real-time governance — with identity, authority, and provenance held apart so that composition is safe by construction, not by policy.

The behavior of the whole is determined by its parts and the rules used to combine them — mathematically, not hopefully. Governance, auditability, model independence, and cost reduction are structural byproducts of this architecture, the same way type safety is a byproduct of a type checker. The guarantee is the architecture, not a policy layer someone promises to enforce.

> *We don't orchestrate. We compose.*

II

Guarantees Do Not *Survive* Orchestration

The AI industry has converged on two architectural models for multi-actor systems. Both treat governance as a feature rather than a structural property. Both fail.

Orchestration puts a conductor in front of many instruments. The conductor is the single point of failure, and the instruments have no guarantees about each other. Compose two orchestrated systems and the governance of each must be re-verified against the other — manually, expensively, and every time the system changes. The guarantees do not survive combination.

Marketplace models go further in the wrong direction. Many agents, no composition rules. Discovery is easy. Trust is impossible to verify at the moment of contact.

Both share the same structural deficiency: governance bolted on after the architecture shipped. Guardrails trigger after the write. Red-teaming patches holes the architecture created. Compliance teams audit what the system already did. This is dynamic typing with runtime checks applied to institutional reasoning. It works until it doesn't, and it never compounds. The effort invested in governing the system produces no durable asset. The next interaction starts from scratch.

This describes the fully cross-coupled case — the cost when every guarantee can interact with every other. It is also the common case when connecting AI systems whose governance interactions are unknown in advance.

Compositional computing is the third option. Many actors. Strict composition rules. Guarantees that survive combination.

III

The Floor Was Found by *Subtraction*

The architectural foundation of compositional computing did not come from a design committee. It was derived through constraint-driven subtraction — a methodology closer to materials science than to systems engineering.

One requirement: what would a system look like if cognitive capture — the condition in which the vendor processing your reasoning takes control of the reasoning itself — were structurally impossible? Not discouraged. Not audited. Impossible.

This is not a theoretical concern. In 2026, a frontier AI lab's chief product officer resigned from a major design company's board — and within days, the lab launched a competing design product built on capabilities developed while processing designers' workflows. The design company's stock dropped immediately. The same lab then entered legal services, small business operations, and financial modeling in rapid succession, each time packaging capabilities refined through exposure to those verticals' reasoning patterns. Billions in market cap evaporated across publicly traded SaaS companies. The pattern is structural: the vendor processes your reasoning, absorbs your vertical's judgment patterns, and enters your market using what it learned. Cognitive capture is not a risk to be managed. It is a business model already in production.

From that constraint, every architectural element was tested by removal. If the system still prevented capture without it, the element was not load-bearing and came out. The product framing came out. The interface layer came out. The model came out. The orchestration layer came out. Auditability as a feature came out. The label "governance platform" came out.

Three things would not come out.

Provenance would not come out. Capture runs on opacity. Remove provenance and the source goes dark, which is the only condition capture needs. Provenance is the trust substrate.

Judgment would not come out. Remove the human approval stage and the model's output becomes the truth at that desk. That is capture happening one interaction at a time. The gate is not a permission check — it decides who adjudicates.

No choke point would not come out. Remove model independence and one provider becomes the arbiter of what can be reasoned about. The starting constraint restated at the supply layer.

Beneath all three: a separation principle. The rules that govern reasoning must live in declarations the owner can read, own, and change — not inside an implementer's code. This principle is called DEX — Declarative Exploration Architecture — and it is where the subtraction stopped. The name borrows deliberately from Information Architecture, the discipline that gave the web its foundational organizing logic for how humans navigate information. DEX does the same for how humans and AI navigate reasoning: a structural organizing discipline that determines what can be explored, by whom, under what terms. Strip DEX and the three properties above still exist, but only as things a vendor promises to do for you.

> *Policy is a promise. Architecture is a guarantee.*

DEX is what the operator sees. But DEX alone is insufficient. Without strict compositionality at the engineering level — identity, authority, and state held apart as distinct operands under rigid, cryptographic rules — a declaration is a YAML file the runtime eventually ignores. Conversely, strict compositionality without DEX is a beautiful machine nobody can steer.

Together they form the floor. Same surface, two altitudes. Neither works without the other. Everything in this paper stands on it.

IV

The Invariant Is Minimal Because *That's the Point*

Every compositional computing system traverses the same five stages, regardless of domain, model, or provider. The pipeline is not a suggestion. It is an invariant. Nothing runs alongside it. No sub-pipelines. No bypasses. No exceptions. That rigidity is the source of every property that matters.

Telemetry. Every interaction produces a structured, operator-owned trace. This is the system's primary output — not the model's answer, but the record that produced it.

Recognition. Classify intent, assess confidence, route to the cheapest tier sufficient for the task.

Compilation. Assemble execution context from local knowledge — historical patterns, domain expertise, prior approvals. Invoke external models only when local knowledge is insufficient.

Approval. The zone-governed checkpoint. Every write is classified by what it changes, not by its category. Green: in-scope, autonomous. Yellow: supervised — the system proposes, the human approves. Red: scope-defining — the system is structurally incapable of acting here. Cannot, not won't.

Execution. The action fires. The output becomes telemetry for the next cycle. The audit trail is the byproduct, not the feature.

This pipeline is the thin waist of the cognitive hourglass. Below it: any data source, any interface. Above it: any model, any skill library, any output surface. A pharmaceutical research engine and a customer service system share nothing except the pipeline. They produce structurally compatible output. They compose without custom integration — for the same reason fundamentally different applications share the same network: they share the invariant.

The pipeline is minimal because minimality is what makes composition possible. Complexity lives in the skills and models above the waist, not in the pipeline itself. The pipeline does one thing — govern the interaction — and does it identically every time.

The floor is also light by construction. The Autonomaton Pattern reduces to three files and a loop: a routing configuration, a zones schema, a structured telemetry log, and the invariant pipeline that traverses them. Everything else is implication. The core declarations — zones, routing, telemetry, grants, provenance — remain structured, human-readable, and versionable. They do not grow into heavy client installations or specialized registry services. Weight that accretes until only specialists can change the system is a defect in the floor, not

an inevitable cost of governance. The operator must still be able to read and mutate the governing declarations directly. If they cannot, DEX has failed — and with it, every claim this paper makes.

---

v

Three Properties That Orchestration *Cannot Produce*

Guarantees survive *combination*

Identity, authority, and state are distinct operands that compose under rigid, cryptographic rules. They never blend. Compositional verification is established computer science — recent work applies assume-guarantee reasoning directly to neural-network verification (Duong et al., NeurIPS 2025). A declaration is not a suggestion the runtime might honor — it is a shape the compute layer is physically incapable of violating.

Three mechanical properties enforce this:

No caller discipline. If a rule matters, it is enforced at the resource boundary, never by trusting a module to remember. A system that relies on a producer behaving nicely has a defect. The boundary enforces the rule. The caller need not know the rule exists.

Air-gapped trust. Identity, authority, and state stay separate. The rules for combining them make collision structurally impossible. A feature that blends identity with authorization is a defect. The air gap is mechanical, not conventional.

Confused-deputy protection. The system declares exactly what it reads. A write is in-scope only if it cannot touch anything the system reads from. When in doubt, the system rules against itself — never in its own favor.

Compose two systems built on these rules and the guarantees of each carry through without re-verification. Orchestrate two systems and every governance assumption must be re-audited at the seam. That is the structural difference.

This property is designed, not automatic. It works because the rules travel with the data. Each keg carries its own access scope, provenance, and reserve declarations as structural properties — not as metadata in a separate system that might get out of sync. The Autonomaton Protocol (GRV-004) makes this explicit by requiring declaration before composition — ground must be established before any payload exchange begins. Inter-system composition works because the floor was built to carry it, not because compositionality is a free consequence of the other commitments.

Each component must satisfy the composition contract before it can participate — that verification has a real, per-component cost. What the architecture eliminates is the re-verification of every previously verified component when a new one joins. The existing network does not need to be reopened. The new component validates against the boundary. The guarantees of the existing components carry forward.

One further design principle: undeclared scope is closed, not open. If a keg does not carry an explicit scope declaration, it is inaccessible. Silence is not permission — it is denial. This keeps the structural type system coherent under composition and prevents missing declarations from being treated as permissive. GRV-004 codifies this as an invariant: what the operator is not claiming must appear in the declaration. Negative scope is load-bearing.

The human is the *adjudicative surface*, not the brake

In orchestrated systems, the human is the fallback that triggers when the automation fails. In compositional computing, the human is the adjudicative surface. The distinction is structural.

The system proposes. The human judges. Every approval and every rejection compiles into a durable asset. The grant token is source. The approved skill is source. Approval does not slow the system. It feeds it.

Rejections are equally load-bearing. They define the negative space — what the system will not do, what the operator will not tolerate. A well-used system encodes both what its operator wants and what its operator refuses, because that boundary was declared through use, not inferred from a training set.

The key structural guarantee: the autonomous loop cannot expand its own authority. This is not a policy, and it is not a cryptographic absolute. It is a designed separation of surfaces. The architecture maintains two surfaces: the autonomous loop, where the agent proposes and executes within its granted scope, and the operator-authenticated surface, where scope changes are authorized. The artifact that opens a scope change — the grant token — exists only in the vocabulary of the operator-authenticated surface. It is not blocked on the autonomous loop. It is absent from it. The autonomous loop cannot produce a grant token — for the same reason a function cannot call a method that does not exist in its type. The capability is not suppressed. It is structurally unavailable. The operator can expand scope. The system cannot. This asymmetry is what converts “cannot” from an aspiration into a designed property of the floor.

Over time, the system learns what the human would approve. It progressively smooths the hot paths — clearing obstacles, surfacing relevant context before it is requested, encoding the operator’s judgment into the system’s own architecture. The system extends the operator’s cognitive reach rather than replacing their judgment. Every approval authorizes a mutation that feeds the flywheel and raises the floor. The operator writes the source through the act of adjudicating.

Parallel access is safe because the operands are *held apart*

The first two properties describe why compositional computing is safe and why it improves through use. The third describes why it is fast — and why it unlocks capability classes orchestration structurally cannot reach.

Multiple actors — human and AI — tap the same knowledge simultaneously. The system composes terms of access, identity, authority, and the full lineage of every source at the moment of contact — not after the fact, not in a log for later review. At the instant the knowledge is tapped.

No coordinator bottleneck. No lock manager. No sequencing layer. The composition rules make parallel access safe by construction. An AI agent and a human researcher tap the same knowledge store at the same time. Each interaction carries its own identity, its own

authority, its own provenance chain. The interactions do not interfere because the operands are held apart.

Orchestration coordinates sequentially — through a conductor that becomes the throughput ceiling. Composition guarantees concurrently — because the guarantees are properties of the operands, not instructions the conductor must remember to enforce. The governance overhead does not scale with the number of actors. It is constant, because the governance lives in the architecture, not in the coordination layer.

This is the property that earns the word “computing.”

VI

The System Compiles Because the Answer *Converged*

Compositional computing produces a system that gets smarter, cheaper, and more sovereign through use. This is a structural property of the architecture, not a product roadmap.

The mechanism is variance collapse. Every approval or rejection labels the output. Over time, the variance in model responses for a given pattern narrows empirically. When standard deviation crosses the compilation threshold, inference is no longer adding value. That is the compilation signal — an empirical fact, not a heuristic.

T3	T2
Apex inference	Fine-tuned
High variance. The model adds real value.	Variance narrowing. Smaller models handle the pattern.
`Highest cost`	`Reduced`
T1	**T0**
Compiled skill	Deterministic
Negligible variance. A deterministic specification the operator owns.	Zero variance. No inference required.
`Marginal`	`Cost: zero`

The system does not demote because a cheaper model exists. It compiles because the answer has converged. A router sends the same prompt to a cheaper model and hopes. The ratchet has measured convergence and compiled the pattern into a durable asset — demotion backed by statistical fact, not a pricing table.

The ratchet turns one way. Every compiled skill is a permanent reduction in cost and latency, an asset the operator owns — inspectable, versioned, portable across providers. Delete the skill and the behavior stops. Edit the skill and the behavior changes. The same substrate is also the seed the operator mutates as goals evolve: new capabilities, new composition rules, and new goal-directed behaviors are authored through the same adjudicative surface that produced the original skills. The structure grows with the operator while the autonomous loop remains structurally incapable of expanding its own authority.

The same closed-world discipline that makes composition safe applies to the substrate itself. The Autonomaton runtime — the pipeline, the zone boundaries, the approval rules, and the skill library — is held locally as a first-class operand under the same composition rules that govern everything else. The system can inspect its own architecture the same way it inspects any other component. Every proposed mutation, whether a compiled skill or an operator-authorized change to composition rules, can be validated against a complete, local self-model before any write occurs. Illegal states that would break the air-gap, authority

separation, or the structural absence of the grant token from the autonomous loop cannot be constructed — they are refused by the same predicates that govern every other write. Vendoring is not packaging. It is the mechanism that keeps the floor load-bearing when the floor is itself the object being mutated under operator authority. The ratchet turns safely because the system that turns it is a first-class operand under its own rules.

This is profile-guided optimization applied to AI inference. Observe real telemetry — structured records of intent, classification, and the operator's adjudications. Find the hot paths where variance has collapsed. Recompile those patterns to collapse the inference cost toward zero. The system progressively clears obstacles from the operator's path — extending their reach, encoding their judgment, making the next interaction start from a higher floor than the last.

The economics follow from the architecture, not the other way around. The system was designed to compile institutional knowledge into durable, sovereign assets. Cost reduction is what happens when knowledge migrates from rented compute to owned infrastructure, from probabilistic inference to deterministic execution.

VII

The Actors Change. The Governance *Does Not.*

The architecture is domain-invariant. Three deployments that share nothing except the pipeline, the composition rules, and the governance model:

Pharmaceutical research. A team curates a cellar — not a library, a governed knowledge store where access terms are structural — containing twenty years of clinical trial data, molecular interaction models, and proprietary compound libraries. Each keg in the cellar carries provenance (who produced it, under what terms, with what confidence) and per-keg scope: some kegs are public to any researcher in the organization, some are member-scoped to a specific team, some are reserved and declared as such. A researcher and an AI agent tap three kegs simultaneously. The terms of access, the identity of the requestor, the authority granted by the cellar's operator, and the full lineage of every source compose at the moment of contact. The AI proposes a novel combination of compounds that prior literature dismissed. Green: the AI suggests. Yellow: a senior researcher approves before any

synthesis order fires. The approval enters the system as source. The cellar gets smarter. The next researcher's floor is higher.

Litigation. A global firm assembles a collection of precedent. Three associates and two AI agents work it in parallel. One agent surfaces a contradiction between opposing counsel's current motion and their filing from eighteen months ago. The provenance chain is instant. The associate traces it in two clicks. She approves. It compiles into a skill. The cost of that insight drops to zero, permanently. It belongs to the firm.

Music production. A musician curates a decade of stems, samples, and session notes. An AI agent proposes a harmonic structure drawing on sessions from 2019, 2022, and yesterday. The provenance is the creative record. The rejections matter as much as the approvals — they define the negative space of her taste. The system knows what she will not tolerate, because that boundary was declared, not inferred.

Every scenario is the same machine. The cellar holds the knowledge and the terms of composition. Kegs carry provenance and per-keg scope — what is public, what is member-scoped, what is reserved. The thin waist governs every interaction. Identity, authority, and state never blend. The AI proposes. The human adjudicates. Approvals compile into durable assets. The floor rises. The actors change. The domains change. The governance never changes.

VIII

Consequences, Not *Features*

The industry treats governance, auditability, cost reduction, and model independence as separate product categories. Compositional computing collapses them into structural consequences of a single set of commitments.

Governance is a consequence of the zone model. Every interaction traverses Stage 04. There is no un-governed path through the pipeline.

Auditability is a consequence of provenance. Every action traces to a telemetry entry, a classification, a skill, a zone boundary, and an approval. The audit trail is produced at

runtime, not reconstructed after the fact.

Cost reduction is a consequence of the ratchet. Patterns converge. Converged patterns compile. Compiled patterns execute at zero marginal cost.

Model independence is a consequence of the thin waist. Swap models; the governance, routing, skills, and audit trail remain intact.

> *A type checker does not sell safety as a feature. Safety is what happens when the types are enforced. Compositional computing does not sell governance as a feature. Governance is what happens when the composition rules hold.*

IX

The Floor Is Common. What Gets Built on It Is *Not Supposed to Match.*

Compositional computing reframes the relationship between AI capability and institutional sovereignty. The dominant assumption — that whoever controls the frontier model controls the value chain — holds only in an orchestrated world, where the conductor is the chokepoint and the model is the conductor's most expensive component.

In a composed world, the model is a swappable dependency. The durable value lives in the composition rules, the compiled skills, and the accumulated provenance — all of which belong to the operator. The model provides inference. The architecture provides custody. They are not the same asset.

For enterprises: the reasoning traces your team generates — corrections, approvals, rejections, iterative refinements — are the highest-value cognitive output your organization produces. Under orchestration, that output flows to the model provider. Under compositional computing, it compiles into sovereign assets you own. The architecture determines which side of the firewall the value accrues.

For the industry: the current generation of agent harnesses treats governance as a constraint on capability. Compositional computing inverts the relationship. Governance enables capability. The stricter the composition rules, the more safely actors operate in parallel, the faster the ratchet turns, and the more value compiles into assets the operator owns. Governance and capability compound together because they are properties of the same architecture.

For the buildout: hundreds of billions of dollars in capital commitments are flowing toward centralized inference capacity. The architectural layer that determines whether that investment compounds at sovereign nodes or evaporates as extractive dependency is receiving effectively none. That asymmetry is not stable.

> *The floor is common. What gets built on it is not supposed to match. A platform wants every node identical. A standard only cares that nodes can reach each other. Variance between nodes is the point.*

x

What the Architecture *Unlocks*

Compositional computing does not make orchestration faster. It eliminates the structural friction that makes governed composition expensive — and in doing so, unlocks four properties that orchestration cannot produce.

Speculative composition becomes *cheap*

Under orchestration, the cost of testing a hypothesis is dominated by the cost of assembling the arrangement: vendor negotiations, licensing agreements, compliance reviews, custom API integrations. A pharmaceutical researcher with a hunch about a dismissed molecular compound faces three vendor negotiations, two compliance reviews, and four months of

procurement choreography before anyone knows whether the hypothesis is worth $50 of compute time.

This creates a brutal selection bias. Only hypotheses that look promising enough to justify the procurement overhead ever get tested. Speculative combinations — the long shots that produce breakthrough discoveries — die in the queue because nobody can justify three vendor agreements for a hunch.

Under compositional computing, the cost of testing the hypothesis is the metered cost of the kegs the researcher taps. A $50 speculative composition that fails is a $50 lesson, not a four-month procurement cycle that produced nothing. The friction of governed composition approaches zero because the governance is structural — declared at the boundary of every keg, enforced by the composition rules, settled at the moment of contact. The number of hypotheses that get tested explodes, because the architecture eliminated the overhead that was killing them.

Term negotiation is itself a *composition operation*

Because identity is separated from authority at the substrate level, business terms are not front-loaded procurement. They are structural properties of the keg, composed at the boundary the same way data and capability compose.

The biotech startup's molecular simulation keg carries its terms: per-invocation metering at a declared price, licensed scope, volume thresholds, a trial tier for the first ten invocations. The university consortium's genomic dataset carries different terms: academic use free, commercial derivative triggers a royalty. These are not side agreements negotiated weeks before the composition. They are structural declarations enforced at the moment of contact.

When the researcher's AI agent proposes a three-keg composition, the researcher sees two things simultaneously: the scientific approach — which kegs, which workflow, which hypothesis — and the business terms of that specific arrangement. She adjudicates both in the same loop, at the same desk, in the same moment. Should she tap the premium simulation keg at $12 per invocation, or start with the lower-resolution alternative at $0.80 and

only escalate if the initial results warrant it? That is a judgment call that belongs to the scientist, not to a procurement team mediating between spreadsheets.

If she wants different terms — bulk pricing for her division, an academic rate, a trial window — the negotiation happens through the same protocol. The keg operator declares tiered terms. The researcher sees the tiers. She decides. No six-week review cycle. No procurement team translating between the scientist who understands the hypothesis and the vendor who understands the pricing. The separation of identity from authority means the researcher carries her own authority to compose within her granted scope, and the terms are resolved structurally at the keg boundary.

The compiled skill captures the *business arrangement*, not just the science

When the researcher approves and the workflow compiles into a durable skill, what gets compiled is not just "these three kegs, this workflow, these parameters." It is "these three kegs, at these terms, with this cost profile, producing this result."

The next researcher who encounters a similar receptor-binding question inherits both the scientific workflow and the commercial arrangement that made it viable. The firm knows exactly what it costs to test this class of hypothesis — not because someone estimated it, but because the cost was captured as part of the compiled asset. The arrangement that produced the discovery is itself a durable, inspectable, versionable object the firm owns.

This changes the economics of institutional knowledge. Under orchestration, a successful discovery is a result: a finding, a paper, a patent filing. The arrangement that produced it — the specific combination of data sources, tools, terms, and expert adjudications — evaporates. It lives in email threads, meeting notes, and the memories of people who may have moved on. The next team that wants to do something similar starts from scratch.

Under compositional computing, the arrangement compiles. The mechanism of discovery is itself the durable asset.

Failed compositions are *equally valuable*

A researcher taps three kegs, runs the simulation, and the hypothesis does not hold. She rejects the finding.

That rejection compiles. The system now encodes that this specific arrangement — these kegs, at these terms, under these parameters — does not produce value at this cost. The next researcher who considers the same combination sees the prior result. She does not repeat the $50 experiment. She either modifies the approach or moves on.

The firm's negative knowledge — what is not worth pursuing, under what conditions, at what price point — is itself a compounding asset. Under orchestration, failed experiments produce nothing: no compiled skill, no institutional record, no floor for the next researcher. The procurement cost was the same whether the hypothesis was transformative or worthless, and the failure left no trace in the system.

Under compositional computing, every adjudication — approval or rejection — raises the floor. The rejections define the negative space of what the institution has tested and dismissed, with full provenance, at known cost. That negative space is as structurally valuable as the positive discoveries, because it prevents the institution from paying to learn the same lesson twice.

Efficiency compounds for the *operator*, not the vendor

The argument for centralized inference has always been convenience: easy to adopt, best models on day one, no infrastructure to manage. The trade-off is cognitive capture, but most enterprises have not yet audited that trade-off. The four properties above change the calculus. For the first time, the capability advantages of sovereign architecture — self-evolving agents, compliance by construction, monetizable failure data — may overshadow the convenience factor. Convenience was only decisive because there was no capability advantage on the other side. There is now.

The structural economics run deeper than a feature comparison. Under centralized inference, the Jevons Paradox works against the enterprise. Every efficiency improvement in frontier models drives more usage. More usage generates more cognitive exhaust flowing to

the vendor. More exhaust deepens the vendor's moat. The enterprise is on a treadmill: the more efficiently it uses the vendor's models, the more the vendor learns about how the institution thinks. Efficiency deepens capture.

Under compositional computing, the same paradox reverses. The ratchet makes inference cheaper through compilation. Cheaper inference means more hypotheses get tested. More hypotheses means more compositions — successful and failed — compile into durable assets. More compiled assets means the system gets smarter, cheaper, and more sovereign. Usage explodes, but every unit of usage compounds on the operator's side of the firewall.

> *Same paradox. Opposite beneficiary. The architecture determines which side of the firewall the compounding happens.*

The deeper structural implication applies the same paradox to governance itself — not the cost of inference, but the cost of governed composition. Under orchestration, governance verification scales multiplicatively: $O(\prod n_i)$. Invest in better compliance tooling. Make each individual verification check cheaper. The efficiency gain enables more compositions — but each new system multiplies the verification surface. Make governance twice as efficient and add one system: the efficiency saved you half, the new system tripled the total. You are behind where you started. The scaling consumes the gain. This is why enterprise AI integrations still take months despite two decades of integration tooling. Jevons is working against you: governance efficiency enables compositions whose overhead exceeds the savings.

Under compositional computing, governance efficiency improvements compound. The ratchet compiles skills. Variance collapses. Governance gets cheaper. So the operator composes more systems. But each new composition still costs O(1) to verify. O(1) here describes the marginal validation at the composition boundary among already-compliant components — not the total cost of verifying each component against the contract in the first place. The efficiency gain is not consumed by scaling. It directly translates into more compositions, which produce more compiled skills, which make governance cheaper still. The flywheel: cheaper governance → more compositions → more compiled skills → cheaper

governance. Under orchestration, the same loop produces: cheaper governance → more compositions → more governance overhead → gains consumed. Same input. Opposite dynamics. One is a virtuous cycle. The other is a treadmill.

The divergence between the two paradigms is therefore not just large — it is self-reinforcing. The longer both architectures run, the wider the gap. Orchestration gets more expensive to govern with every system added. Compositionality gets cheaper with every adjudication compiled. The curves do not merely diverge. They accelerate apart.

The kegged negative finding is the most striking expression of this inversion — and it connects directly to the diagnosis that opens this paper. The reason vendors capture reasoning traces is not the text. It is the corrections. Process Reward Models, the next generation of AI training infrastructure, require complete execution trajectories: plan, error, diagnose, fix. The error-to-correction path is the highest-value training signal in the industry. Every rejection, every "that's wrong, try this instead," every iterative refinement is precisely the exhaust that frontier labs cannot synthesize without harvesting it from live operators. The negative findings that compositional computing kegs as operator-owned assets are the exact same signals that make reasoning traces valuable to vendors. The diagnosis and the unlock are the same object viewed from opposite sides of the firewall.

Under centralized inference, a failed experiment produces nothing for the operator — no compiled skill, no institutional record, no floor for the next researcher. The reasoning traces from the failure flow to the vendor, where they become training signal for the vendor's next model. Under compositional computing, that same failure is a kegged asset with full provenance, under declared terms, at known cost. It prevents the institution from paying to learn the same lesson twice. And because provenance makes it trustworthy and per-keg scope makes it tradeable, the institution can declare that keg as member-scoped to a research consortium, or licensed for external composition under metered terms. A pharmaceutical company that has kegged three years of failed molecular interactions has built something no centralized inference provider can replicate: a curated cellar of what does not work, available for governed composition by anyone willing to meet the declared terms.

The firm monetizes its failures. The knowledge economy has a substrate where negative results carry structural value. Entire generations of research produced findings that never reached the literature because negative results had no publication venue, no attribution

chain, and no economic value. Under compositional computing, every negative finding is a provenance-labeled, scope-declared, composable asset. The institution that produces it owns it. The institution that taps it compensates the producer under terms composed at the boundary. The floor rises for everyone, and the economics reward the institutions that do the most exploration — including the exploration that fails.

The cellar becomes a governed marketplace where terms are structural, composition is instantaneous, and every adjudication — approval or rejection — raises the floor.

This is what the architecture makes possible. Not faster orchestration. Not a better marketplace. A structural substrate where the friction of governed composition approaches zero — where the cost of exploring a hypothesis is the cost of the inference, not the cost of assembling the governance around it. Where every composition — successful or failed — compiles into a durable asset the operator owns. Where science and commerce are adjudicated in the same loop by the person closest to the work, because the architecture separated identity from authority and made the terms structural.

This paper describes the architectural commitments required to make that moment structurally inevitable.

TERMINOLOGY

Terms of *Art*

Compositional computing names the structural conditions it is designed to produce or prevent. These terms are canonical; they are intended to be copied verbatim into any composition without interpretive drift.

Cognitive capture. The condition in which the vendor processing an organization's reasoning takes control of the reasoning itself. Extends regulatory capture one domain over.

DEX. Declarative Exploration Architecture. The name borrows from Information Architecture — the discipline that organized how humans navigate information — and applies the same structural logic to how humans and AI navigate reasoning. DEX is the separation of

exploration logic from execution capability: the rules that govern reasoning live in declarations the owner can read, own, and change — not inside an implementer's code.

Compositionality. Identity, authority, and state held apart as distinct operands under rigid composition rules. Mathematical compositionality — the behavior of the whole determined by its parts and the rules for combining them.

Thin waist. The five-stage invariant pipeline with three zones. The cognitive hourglass. Never changes.

Cellar. The operator's curated collection of kegs, plus the business terms under which they compose. Per-keg scope (public, member, reserved) and circuit invitation travel with the cellar, not in a side agreement. Not a library. A governed knowledge store where the terms of access are structural — declared in the cellar manifest, enforced at the moment of contact.

Keg. A unit of distilled knowledge carrying provenance and per-keg scope. Traceable or asserted — pick one. Scope (public, member, reserved) is declared at the keg level, not at the operator level. What the operator is not claiming is declared explicitly in the reserve field — negative scope is load-bearing.

Two surfaces. Surface A: the autonomous loop, where the agent proposes and executes within granted scope. Surface B: the operator-authenticated surface, where scope changes are authorized. The grant token — the artifact that opens a scope change — exists only in Surface B's vocabulary. It is not blocked on Surface A. It is absent. Cannot, not won't.

Ratchet. Variance collapse. T3 → T2 → T1 → T0. Turns one way.

Grant token. The operator-issued authorization that opens a specific scope change. Exists only in the vocabulary of the operator-authenticated surface. Not blocked on the autonomous loop — absent from it. The designed asymmetry that keeps sovereignty at the operator.

Provenance. The trust substrate. Capture runs on opacity. Provenance is the countermeasure.

REFERENCES

References

1. The Grove Foundation. “The Autonomaton Pattern: Toward Self-Authoring Software Systems.” *GRV-001*, v2.0, June 2026. CC BY 4.0. https://the-grove.ai/standards/001

2. The Grove Foundation. “TCP/IP for the Cognitive Layer.” *GRV-002*, March 2026. CC BY 4.0. https://the-grove.ai/standards/002

3. The Grove Foundation. “The Autonomaton Protocol.” *GRV-004*, 2026. CC BY 4.0. https://the-grove.ai/standards/004

4. The Grove Foundation. “Structural Custody in Agentic AI.” *CIO Alert 007*, Q2 2026. https://the-grove.ai/alerts/structural-custody

5. Duong, Shriver, Nguyen & Dwyer. “Compositional Neural Network Verification via Assume-Guarantee Reasoning.” *NeurIPS 2025*. https://proceedings.neurips.cc/paper_files/paper/2025/file/5ca2e240108d09e213859ea7ef2324bb-Paper-Conference.pdf

Related reading