Nick Clark · Published June 2026 · PDF

Autonomy You Can Trust

Every autonomous system must decide on its own, in the moment, without a round-trip to authority. That physical fact, not a preference for decentralization, is what finally forces a fifty-year-old idea into necessity: authority, governance, and identity must travel inside the data object the system acts on, rather than being granted by the host it happens to run on. The technology already exists and is battle-tested; what has been missing is the reason to combine it, and an architecture that does so coherently across every layer.

---

1. The trust gap nobody has closed

Autonomy is the fastest-growing objective in computing. Agentic AI, robotaxis, humanoid robotics, and defense and space systems all run into the same wall: how do you trust a system that decides for itself?

Start with a fact that is easy to overlook. An autonomous vehicle's control loop runs at roughly 20 Hz, on a sub-50-millisecond budget. Network round-trip latency, even on good cellular, is in the hundreds of milliseconds. A per-decision call to a central server is not slow. It is *physically impossible*. The same holds everywhere autonomy actually matters. A Mars rover sits 20 light-minutes from its operators and selects its own science targets onboard. Military and space systems formalize the normal case as

denied, degraded, intermittent, and limited connectivity. An AI agent that must ask permission for every step is not autonomous; it is a remote control with extra latency. Autonomy is, by definition, the property of acting without a round-trip to authority.

This creates the trust gap. We have spent decades building trust models that assume a trusted central point the system can defer to: an identity provider, a policy server, a certificate authority, a human in the loop. Autonomy removes exactly that point, at exactly the moment of decision.

The industry's current answer is to bolt a monitor onto the outside. This is the dominant paradigm, and it has a name: runtime assurance. NASA's Runtime Assurance framework is explicit that it "filters an unverified primary controller" and is "agnostic" to what that controller is. The Simplex architecture switches from a high-performance controller to a verified safe one when things go wrong. Reinforcement-learning systems are wrapped in external "shields"; large language models are wrapped in external guardrail classifiers. DARPA ran a multi-year program, Assured Autonomy, premised on the fact that continually assuring a learning system is an open problem.

Every one of these is the same shape: a probabilistic decision-maker on the inside, and a separate governor bolted on the outside. It works, partially. But the governance lives *outside* the thing being governed. It does not travel with the object, does not survive disconnection cleanly, cannot prove after the fact why it allowed what it allowed, and must be re-established at every boundary the object crosses. In a world of one trusted datacenter, that is tolerable. In a world of millions of autonomous units acting across heterogeneous environments, it is the gap.

2. An old idea that never found its moment

The alternative has existed since the 1960s. The idea has never been the problem; adoption has.

The idea is object-as-authority: authority is not something the environment grants the actor; it is something the object *carries* and the actor must *present*. In capability-based security, authority is an unforgeable reference held by the object, enforced locally, with no appeal to a central principal. This is not a fringe theory. Its standard objections were formally answered two decades ago. It ships in production: KeyKOS ran VISA transaction processing on capabilities in the 1980s; Capsicum secures FreeBSD and Chromium today; Google built Fuchsia capability-first from the ground up.

Adjacent traditions reached for the same inversion from other directions: "active data" and sticky policies that carry their own rules, and mobile agents that move code, state, and authority between hosts. They all glimpsed the same future, in which the data is the seat of authority and the host is transient.

So why didn't it win? Two honest reasons.

First, substrate inertia. The entire stack of operating systems, networks, and identity is built on ambient authority: the host decides what you may do based on who you are. Replacing that requires rebuilding the substrate, and nothing ever forced the rebuild. Ambient authority was always "good enough," because there was always a trusted host to be ambient on.

Second, for the data-carrying branch, the malicious-host wall. An object that carries its own authority must still execute somewhere, and a fully adversarial host can read, tamper, replay, or simulate it. In pure software this can be made *evident* but not *prevented*, a result with a formal proof behind it. Preventing it requires a hardware root of trust. This capped "active data" at detection, and the field moved on.

Both reasons share a single root: there was no forcing function. The idea was right and even shippable, but nothing in a centralized world demanded it.

3. Autonomy is the forcing function

This is the inversion the paper turns on: the forcing function is autonomy, not decentralization.

For thirty years, every "authority in the data" pitch was really a pitch for decentralization, and decentralization kept losing to the cloud, which kept getting cheaper and better. Betting on the host going away was a losing bet, and it still is. The center keeps winning.

But autonomy does not require the center to go away. It requires the *decision* to happen without it. A vehicle with a thousand datacenters behind it still has to decide, onboard, in fifty milliseconds, whether to brake, carrying its own constraints and governing its own behavior with no server in the loop. Autonomy needs the object to carry governed authority even when the infrastructure is fully centralized. That is what frees the idea from its losing bet: it no longer depends on decentralization winning. It holds wherever a system must decide for itself, which is to say wherever autonomy is real.

The gap is precise and, as far as the literature shows, unclaimed. Today, nobody carries governance inside the autonomous unit's own state; everybody bolts it on the outside. The closest published work attaches portable governance to the agent, but stops short of making the object itself the locus of authority, and stops short of the claim that this is prerequisite rather than optional. The space is empty.

The prerequisite claim is the strong one, and it deserves to be stated plainly as an argument rather than smuggled in as a fact: trustworthy autonomy is impossible without carried authority. If the decision must be local, then everything the decision must be trusted against, including identity, policy, lineage, and the right to act, must be local too, or the trust is a fiction that evaporates the moment connectivity does. An external governor is a promise that there is still a center to appeal to. Autonomy is the condition of there not being one. You cannot govern from a center a thing whose defining property is that it acts without the center.

4. Carried governance: one substrate, every layer

If authority must travel with the object, then *every* trust function must be reconceived as a property of the object rather than a service of the host. These functions must also compose, because a self-governing object that cannot prove its own identity, route itself, find what it needs, or evolve under constraint is not autonomous; it is stranded.

That is the architecture: a single coherent substrate in which the object carries:

- Identity it can prove without a certificate authority, that evolves rather than sits static, and that breaks visibly if the host it runs on is tampered with.
- Governance: policy and permission the object carries and checks *before* it acts, where refusing to act is a valid, first-class outcome, and where the object cannot quietly rewrite its own rules.
- Lineage: an append-only, tamper-evident record of what it did and why, so that a decision made in isolation can still be audited later.
- Resolution, routing, and indexing keyed to the object's own trust state rather than to network addresses and central registries.
- Execution and embodiment in which the object's memory and constraints persist across the heterogeneous, intermittently-connected, adversarial environments where autonomy actually operates.

This is where the architecture earns the word *trust*. In pure software, no protocol can make a single adversarial host honest about itself. A compromised machine can sign a truthful-looking measurement while running tampered code, and it cannot hide a secret it must use locally. Those two guarantees are hardware-bound, and the substrate is built to compose with the hardware roots of trust the world is already deploying. But honesty about one host is not the same as integrity of the system. A trust-scoped, host-signed, consensus-backed fabric makes every host's actions attributable, since each node signs its work and its trace, and it subjects each state change to independent multi-party quorum that checks whether the new state is derivable from a trusted

origin. That quorum is the external witness that defeats the fork, or "split-world," attack a single object can never defeat alone: to forge a consistent-but-false history, a malicious host would have to compromise a trust-weighted threshold of an entire zone rather than one machine. When a host does misbehave, the fabric scores it down, routes trust-sensitive work away from it, quarantines it, revokes it, and treats non-execution as a valid outcome when authority or continuity cannot be verified. The malicious host stops being an unbounded, anonymous, trusted-by-default problem and becomes a *contained, attributable, out-routable* one. That is precisely what carried governance buys that an external monitor cannot. The trust travels with the object, witnesses itself across nodes, and fails closed; and where the risk class demands the one thing software cannot supply, the same fabric admits a hardware-backed attestor at the leaf.

5. Prior approaches, compared

Approach	Where governance lives	Survives disconnection?	Self-auditing?	Malicious-host posture
Runtime assurance / Simplex	External monitor beside the controller	Only if the monitor travels too (it usually doesn't)	Logs externally	Out of scope; assumes a trusted platform
LLM / agent guardrails	External classifier beside the model	No	Partial, external	Out of scope
PKI / OAuth / central policy	Central authority the object defers to	No; needs the authority reachable	Central logs	Trusts the issuer; nothing carried
Sticky policies / IRM / DRM	Carried with the data, enforced by a cooperating host	Partial	Partial	Degrades to detection at an untrusted endpoint
Capability security	Carried as an unforgeable token, enforced	Yes, within one trust domain	No (authority decoupled from audit)	Assumes a trusted kernel; no cross-host fabric

Approach	Where governance lives	Survives disconnection?	Self-auditing?	Malicious-host posture
	by a trusted kernel			
Carried governance (this work)	**Inside the object's own state, witnessed by a trust-scoped fabric**	**Yes, by design**	**Yes, via append-only lineage**	**Attributable, quorum-witnessed, out-routable; composes with hardware at the leaf**

The distinction is not a new primitive. It is *where authority lives* and *how the system behaves when the host cannot be trusted*. No prior approach both carries governance inside the object and witnesses it across an independent, trust-scored fabric.

6. Why now, and why it holds

The strongest objection to any combination like this is obviousness: the pieces already exist. They do. Signed objects, hash-based identity, append-only logs, capability-style pre-action checks, and quorum protocols are all established. That is not the weakness of this architecture; it is the entire pitch. Every component is already proven, already running, and already trusted on centralized infrastructure. The novelty is not a new primitive. It is the recombination, and the recognition that recombining these pieces for carried autonomy across heterogeneous environments is the thing no one built, because the prior art consistently assumes, and is optimized for, a trusted central host. The art does not merely fail to teach this combination. It teaches *away* from it, toward centralization.

The timing is no longer speculative. Autonomy is the fastest-growing, best-capitalized objective in technology, and unlike decentralization, the autonomy thesis does not need the cloud to lose. It needs autonomous systems to multiply, which they are, across precisely the environments where a central trust point is unavailable by physics or by adversary: the road, the factory floor, the contested zone, orbit, and the edge.

7. The result

Put the pieces together and the result has a name the market is already asking for: autonomy you can trust. It is deterministic where today's autonomy is probabilistic, auditable where it is opaque, and governed from within where it is monitored from without. It is carried, so that it survives the disconnection that defines the autonomous condition, and it is equally valid on the centralized infrastructure that still needs local trust at the point of decision.

The fifty-year-old idea was never wrong. It was early, and it was waiting for the one thing that would make it necessary rather than merely elegant. That thing has arrived. Autonomy is the forcing function, carried authority is the requirement, and the substrate that delivers it across every layer is the unclaimed ground this work stakes out.

Appendix: the filed substrate

The thesis is not a proposal; it is the emergent property of a body of filed work. Each layer below is its own filing and its own market, and each is built to interlock with the next.

- Agent Schema: the unit of computation, a self-describing object with canonical fields (intent, context, memory, policy reference, mutation descriptor, lineage), validated by structure before execution.
- Memory-Resident Execution: a full lifecycle carried in the object's own memory and evaluated independently at each node, with no external scheduler.
- Memory-Native Networking: a protocol stack whose transported unit is a cryptographically signed object, where embedded memory and trust history drive trust-scoped routing, indexing, and consensus instead of IP addresses or DNS.
- Adaptive Indexing: an anchor-governed namespace that splits, merges, and re-anchors under load through scoped local quorum, preserving lineage with no global consensus.

- Memory-Native Identity: authentication without keypairs, identity as an append-only "trust slope" of device-entangled hashes, quorum-recoverable, secure under hash-preimage resistance.
- Cryptographic Governance: behavioral authority externalized to signed policy objects, resolved and verified before an execution context is instantiated, with anti-rollback, revocation, and append-only audit, where non-execution is a valid result.
- Cognitive Architecture: affect, integrity, confidence, capability, and forecasting as typed, replayable fields of the object itself, the layer that turns governed execution into self-regulating autonomy.

Notes and scope

This paper articulates a thesis and an architecture, not a claim to repeal the impossibility results that bound software-only protection of an object on a fully adversarial host. Two guarantees, that a host measures itself honestly and that a secret used on a host stays secret, remain hardware-bound; the architecture composes with hardware roots of trust at the leaves and explicitly admits hardware-backed attestation where the risk class demands it. The contribution is the carried-governance and trust fabric above that line, and the recognition that autonomy, not decentralization, is what makes object-as-authority a prerequisite rather than an option. The underlying mechanisms are the subject of a body of filed work; this document describes the thesis they compose into. References include capability security (Dennis & Van Horn 1966; KeyKOS; seL4; Capsicum; Fuchsia), the malicious-host literature (Sander & Tschudin 1998; Barak et al. 2001 on the impossibility of general obfuscation), runtime assurance (NASA RTA; Simplex; safety filters), and the closest recent framing of portable agent governance.