

The Participant Layer of the Internet

MOI Protocol

A base protocol where participants exist natively in computation

TCP made messaging a network primitive.

HTTP made resources a network primitive.

Contextual Compute makes participants a network primitive.

“Information can be processed anonymously. Value requires participants.”

Sarva Labs · moi.technology

March 2026

Confidential. For qualified investors and strategic partners only.

Three Problems. One Absence.

Three domains. Three problems that every practitioner recognizes. We will show they are the same problem, and that the problem is not coordination, not engineering, and not incentive design. It is an absence.

The Agent Problem: Trust Is Personal

You tell your AI agent: “Book me the cheapest Delhi–London flight, under \$800, pay with my Visa ending 6411.” The agent needs your passport details, your payment credentials, spending authority, and boundary conditions, all at machine speed.

Today, the agent holds an API key to your bank. If it goes rogue or gets compromised, your entire account is exposed. But the deeper problem is not delegation. It is that delegated authority is *personal*.

Illustration: Why authority must be personal

The same agent, identical code, identical model, must behave completely differently depending on who it acts for. Elon might authorize an autonomous agent to spend \$1M without approval. Sam might cap the same agent at \$50 for groceries with confirmation above \$30. The authority doesn’t come from the agent. It comes from the participant.

Where does this participant-specific authority and preferences live today? Nowhere. No agent framework today provides a computational locus for participant-specific authority. No blockchain stores participant-indexed state. The application can’t enforce it, because authority that is personal requires a person to exist inside the machine.

The Business Problem: Privacy Is Personal

You renew your passport. How many services hold your old number? Airlines, banks, visa agencies, insurance companies, corporate HR, travel agents, government portals. How many AI agents have cached copies? You don’t know. You can’t know.

The visible problem is fragmentation: thirty databases holding thirty copies of your passport number, each requiring a separate update. But fragmentation is the symptom, not the disease.

Illustration: Why privacy must be personal

Person A wants airlines to see their passport number but travel aggregators to receive only a verified credential. Person B wants biometric confirmation before any access, every time. Person C grants their corporate travel agent full access but masks credentials for personal bookings. Each person wants different privacy rules for the same data.

Today, this is impossible. Not because selective disclosure technology doesn't exist. It is impossible because the participant has no computational existence from which to express and enforce these preferences. Each service has *its* privacy policy applied to you. You don't have *your* privacy policy applied to them.

The Token Problem: Custody Is Personal

You deposit 100 tokens into a DeFi lending pool. Your tokens leave your control and enter a shared contract alongside \$2 billion from other depositors. One vulnerability in that contract, everyone loses everything. Bridge exploits alone have cost over \$2.8 billion: Ronin (\$625M), Wormhole (\$320M), Nomad (\$190M).

Illustration: Why custody must be personal

One participant is comfortable with high-risk DeFi strategies. Another wants conservative, fully collateralized positions only. But today's architecture forces everyone into the same custodial pattern: deposit into a shared contract, hope the code is correct. The machine has no way to keep assets with their owners during exchange.

One Root Cause

Three domains. Three problems. **One root cause: the participant does not exist in the machine.**

The missing element in each case is the same: **WHO**, the participant as a computational primitive. Not a user of the system. A dimension of computation itself.

The Tool Was Built for Information. The Use Moved to Value.

Modern computing and networking systems evolved around anonymous state and information flow. Turing (1936) formalized computation for a single operator and an anonymous tape. Shannon (1948) made information transmissible by separating it from meaning and context. Distributed systems replicated messages everywhere, achieving speed and fault tolerance through copying.

This worked. Perfectly. For information.

Information is copyable: you can duplicate a file without destroying the original. Information is observer-independent: the content of a message doesn't change based on who reads it. Information is reconcilable: if copies diverge, there exists a fact of the matter to converge on.

But the use of the tool changed. Progressively, computation became the substrate for value: money, identity, agency, delegated authority. And value is a fundamentally different category.

Value Is Different. Categorically.

Your bank balance is not a file. If it's copied to five servers, you don't have five times the money. You have one balance and four records about it. If two copies process conflicting transfers, the system doesn't have inconsistency; it has incoherence.

Property	Information	Value
Copying	Defined (duplicate freely)	Undefined (duplication = counterfeiting)
Observer	Independent (same for all)	Constitutive (meaning depends on WHO)
Reconciliation	Converge to ground truth	No ground truth without participants
State	Anonymous, global	Participant-indexed, contextual

These are different mathematical categories. No structure-preserving transformation connects them. A machine built for copying cannot handle value by adding consensus on top; such a machine is structurally inefficient. The operation that must be undefined (copying) is the machine's foundational operation.

The Recursion: What Happens When You Try

Every distributed system that simulates value on anonymous state inherits this contradiction and pays for it continuously:

$$\textit{Sally's balance} \neq \textit{Sally's agreed balance}.$$

You need agreement to know the balance. You need the balance to run agreement. This recursion is

the structural consequence of simulating value on an information machine. You can't shard it away. You can't optimize it away. A type error doesn't yield to coordination.

Every coordination cost in distributed systems and blockchains, reconciliation in banking, trust verification in enterprise, MEV and re-orgs in blockchains, is the system compensating for a missing dimension. The inefficiency is not in the implementation. It is in the model.

The Missing Dimension

The use of the tool changed. The tool didn't. Computation has three dimensions:

Dimension	What It Provides
WHAT	Logic, programs, state transitions
WHERE	Storage, memory, persistence
HOW	Execution, ordering, consensus
WHO	Participant, identity, ownership, agency — ABSENT

For seventy years, this was fine. Computation was a tool: you used it and left. The machine didn't need to know who you were because nothing inside it belonged to you.

That changed. Computation is now where we live. When computation becomes a substrate for value, participants need to exist inside it. Not as a row in a database. Not as a profile in an app. As a first-class computational primitive.

WHO has no infrastructure, until now.

Contextual Compute

Contextual Compute introduces WHO as the fourth dimension of computation. It is the correct mathematical structure for any system that manages value. This is **value computation**.

- **Classical compute (WHO = 0):** Anonymous global state. Information, files, messages. Copyable. Context-free. This is every system that exists today, including every blockchain.
- **Contextual Compute (WHO > 0):** Participant-indexed state. Value transfer, property rights, autonomous agents. Linear. Participant-bound.

Classical computation: $\delta(S, I) \rightarrow (S', O)$

Value computation: $\psi(P, C, I) \rightarrow (P, C', V)$

P persists. C transforms. V, value, is produced. The participant is the anchor.

Classical computation is a special case of Contextual Compute: the case where the participant dimension equals zero. You can run information on Contextual Compute. You cannot run agency on classical computation. Contextual Compute is the general theory. Today's computation is the special case.

The shift from anonymous to participant-indexed computation ripples across every domain:

Domain	From	To
Computation	$\delta(S, I) \rightarrow (S', O)$	$\psi(P, C, I) \rightarrow (P, C', V)$
Distributed Systems	Global ordering	Participant-centric ordering
Networks	App/resource-centric	Participant-centric
Trust and Safety	App feature	Infrastructure property
AI	General intelligence (AGI)	Emergent personal intelligence (EPI)
Economics	Transaction economics	Participation economics
Meta	Information	Value

Primitives and Protocols

Every computational paradigm is defined by a small set of irreducible primitives:

Paradigm	Core Primitives
Relational Model	relation, tuple, attribute
Object-Oriented	object, class, method
Actor Model	actor, message
Replicated State Machine	account, transaction, block
Contextual Compute	participant, context, interaction

Contextual Compute defines three primitives for value computation: Participant, Interaction, and Context. The protocols define how these primitives compose.

Participant

A Participant is the fundamental unit of existence in Contextual Compute. A participant represents an entity capable of owning state, exercising authority, and engaging in interactions. Participants may represent individuals, agents, organizations, or other autonomous entities. All value and authority are anchored to participants.

Interaction

An Interaction is the fundamental operation of Contextual Compute. An interaction occurs when one or more participants engage to transform value within their respective contexts. Unlike transactions, which operate on anonymous global state, interactions operate on participant-scoped context. Interactions define how context evolves.

Context

Context is the accumulated existence of a participant in computation. It emerges from interactions over time and encodes behavior, preferences, permissions, and authority.

Context is the participant-specific awareness graph: the compounding structure of every relationship formed, every decision made, every preference expressed, and every interaction declined. It is not data about the participant. It is what the participant *becomes* through computation.

The Context Superstate

The Context Superstate is the cryptographic data structure that represents a participant's context. It organizes and secures the participant's state, relationships, and permissions within the computational system. It ensures that a participant's existence, authority, and history are verifiable, portable, and composable across interactions.

Seven sub-contexts, one state, one participant:



Figure 1: Four markets converge on one primitive. Each domain reaches into the Context Superstate and finds what it needs, because what it needs is the participant.

Sub-Context	What It Holds
Assets	Native tokens, balances, metadata. Protocol-guaranteed conservation.
Trust	Witness sets, delegation chains, scoped authority mandates.
Storage	Participant-owned data, records, credentials.
Logic	Deployed code, interaction handlers, programmable rules.
Keys	Cryptographic keys, rotation history, recovery paths.
Preferences	Privacy policies, notification rules, interaction constraints.
Intelligence	Participant embeddings, interaction patterns, adaptation parameters. The substrate for emergent personal intelligence.

Always current. Always available. Always sovereign.

Illustration: Not a Wallet. Not a Honeypot. A Home.

A wallet, if stolen, gives the thief everything. A home has rooms, locks, and you choose who enters which room. The Context Superstate is a home.

Different rooms, different access. The airline sees your passport during the booking interaction. It never sees your asset balances. The DeFi protocol sees your token balance during the swap. It never sees your passport.

One breach, one participant. One Equifax breach exposes 147 million people. One bridge exploit drains \$625 million from everyone who deposited. With the Context Superstate, the worst case is one participant, never millions.

Tamper-evident by construction. The Context Superstate is a Merkle commitment, not a file on a server. Any modification invalidates the commitment and is rejected by the witness set.

These three primitives unify in a single computational function:

$$\psi(P, C, I) \rightarrow (P, C', V)$$

The participant persists. The context transforms. Value is produced.

Protocols

Context Protocol defines how contexts compose, transform, and maintain integrity. Every interaction consumes the pre-interaction context and produces the post-interaction context. State is never copied. It is transformed.

Interaction Finality Protocol defines how interactions achieve finality without global consensus. Agreement is scoped to the participants involved and their chosen witnesses. We don't replicate and reconcile. We witness and certify. Finality is immediate and irreversible.

Interaction State Machine defines how state transitions execute with linear semantics. The ISM provides domain-aware execution with semantic interaction types. Assets cannot be duplicated, re-entrancy is undefined, and conservation is structural.

The Participant Layer

Contextual Compute is the paradigm. The Participant Layer is the infrastructure that emerges from it.

The Participant Layer provides the computational substrate where participants exist natively: where state is owned, authority is scoped, interactions are witnessed, and context accumulates. It sits beneath applications, alongside data infrastructure, and above raw compute, completing the stack with the dimension it has been missing.

Dimension	Layer	What It Provides
WHO	Participant Layer	Principal-of-record, scoped authority, live revocation, interaction-level accountability
WHAT	Application Logic	Smart contracts, agent frameworks, enterprise apps, DeFi protocols
WHERE	Data & Compute	Cloud infrastructure, DA layers, storage, execution environments

Applications keep doing what they do. Infrastructure keeps providing where they run. LLMs keep providing general intelligence. The Participant Layer provides who is acting, with what authority, witnessed by whom. The applications remain. The trust model changes.

Contextual Compute → Participant Layer → MOI

Paradigm → Infrastructure → Network

MOI: The Open Contextual Compute Network

MOI is the open Contextual Compute network, the first implementation of the Participant Layer. Each component below implements a specific protocol from the Contextual Compute stack.

MDAG: Participant-Centric State

Concept. The Context Protocol requires that participant state evolves through interactions, not global state updates.

Implementation. MOI represents state through a Multi-link Composite Directed Acyclic Graph (MDAG). Each node records an interaction between participants and includes cryptographic pointers to the latest versions of their respective Context Superstates. Because interactions may involve multiple participants, a single node can link multiple context lineages, forming a composite structure.

Architectural difference. Unlike replicated state machines that maintain a single globally replicated state, MDAG organizes state evolution around participant interactions. Context Superstates evolve along participant lineage rather than through network-wide state updates.

Consequence. Interactions involving disjoint participants progress independently without requiring global ordering. The network verifies the integrity of context evolution rather than reconciling a shared global state.

KRAMA: Interaction-Level Agreement

Concept. The Interaction Finality Protocol requires that agreement is scoped to participants and their witnesses, not the entire network. We don't replicate and reconcile. We witness and certify.

Implementation. KRAMA is a Byzantine Fault Tolerant linear agreement protocol designed for interaction-level finality. For each interaction, the relevant witness nodes form an ephemeral validation cluster responsible for verifying and certifying the interaction outcome. Once finalized, the cluster dissolves.

Architectural difference. Replicated state machines perform consensus across a fixed global validator set that orders all transactions in a shared ledger. KRAMA scopes agreement to individual interactions, achieved among the witness nodes associated with the participants involved.

Consequence. Independent interactions finalize in parallel with immediate finality. Trust is modulated per interaction based on participant contexts. Mandate enforcement happens at the protocol level: an agent may attempt to exceed its authority, but the Context Superstate will not allow the interaction to commit unless it falls within the mandate's scope.

PISA + COCO: Interaction-Level Execution

Concept. The Interaction State Machine requires deterministic, linear execution over participant contexts.

Implementation. MOI separates execution into two components. PISA is the execution environment: a register-based virtual machine designed for deterministic execution of interaction-driven state transitions, supporting multiple interaction types including asset, logic, and file operations. COCO is the programming model: it treats Context Superstates as first-class constructs and enforces linearity at the type level. Re-entrancy is not prevented; it is undefined. The type system does not admit it.

Architectural difference. Replicated state machines invoke arbitrary functions on smart contracts for every operation, including simple transfers. MOI separates domain-native interactions (protocol primitives) from programmable logic execution. Common operations are built into the runtime, not user-defined code.

Consequence. Application logic remains largely stateless while committing deterministic context transformations. Network Objects, the first network-level primitives designed for linear types, ensure that assets are never data in a contract. They are native objects within participant contexts, with protocol-guaranteed conservation.

PoXt: Trust as Flow

Concept. Security in value computation must reflect ongoing behavior, not static commitment.

Implementation. PoXt derives trust from four dimensions: monetary commitment, behavioral reputation, social verifiability, and decentralization contribution. Trust is a flow, not a stock. Context Power is earned continuously, not locked once.

Architectural difference. Proof-of-Stake systems derive security from capital locked at one point in time. PoXt derives security from sustained, multi-dimensional contribution. Witnesses that collude lose Context Power permanently; the economic cost of tampering exceeds the benefit.

Consequence. Trust is granular, adaptive, and earned. The security model reflects the participant-centric nature of the network rather than relying on capital concentration.

The architecture is not designed. It is derived. Each layer is forced by the layer below, ultimately forced by the nature of value itself. If value is linear, WHO must be primitive. If WHO is primitive, contexts must be owned. If contexts are owned, agreement must be local. If agreement is local, finality is immediate.

Four Structural Effects of Contextual Compute

Effect	What It Means
Business Trust	Portable, verifiable trust anchored to participants.
Agent Safety	Delegated authority enforced at the infrastructure level.
Native Assets	Participant-owned value with safe tokenization and direct transfer.
Efficient AI	Participant-specific intelligence from the Context Superstate, with referral knowledge from general-purpose models.

Business Trust

Always current. Always trusted.

Enterprise digitization generates \$500 billion annually in trust-verification friction: reconciliation, compliance checking, cross-platform identity verification. The root cause is that trust is scattered across applications. Each service maintains its own view of the participant. None are authoritative. All are stale.

The Participant Layer anchors trust to the participant, not the application. Your passport, your compliance status, your verification history live in your Context Superstate. Services access them during interactions with permissions you define.

Illustration: Passport, one update, everywhere

Your passport number lives in your Storage sub-context. Services don't store it; they access it during interactions, with permissions defined by your Preferences sub-context. Update once. Every future interaction reads the current value. No broadcast. No stale copies. Nothing to leak because nothing was replicated.

Person A's privacy policy: airlines see the number, aggregators get a verified credential only. Person B's policy: biometric confirmation required every time. Both policies live in the participant's context, enforced by the participant, not by thirty different service providers.

The result is portable, verifiable trust. Cross-platform evidence without rip-and-replace. On-demand state access. Compliance that travels with the participant, not bolted onto each application.

Agent Safety

Apps and agents log into you. Not the other way around.

The agent economy is the forcing function. Agents demand the highest form of value: delegated authority at machine speed. An entire ecosystem has emerged around agent memory and context

persistence:

Mem0, Zep, Supermemory: Memory APIs that give agents persistent recall across sessions.

Plurality, AI Context Flow: Browser extensions for cross-LLM preference portability.

MCP (Model Context Protocol): Anthropic’s standard for agent-tool connectivity.

These are real solutions to real problems. But they share a structural limitation: every one is stateless memory, anchored to the application. Mem0’s memory lives in Mem0’s infrastructure. If Mem0 shuts down, your agent’s memory of you is gone.

Stateless memory works for preferences. It fails the moment agents need to act with authority:

What Agents Need	Why Memory Layers Can’t Provide It
Delegated authority	A vector database can store “user prefers budget flights.” It cannot express “user authorizes \$800, from Visa *6411, until Friday, revocable now.” Authority requires committed state, not cached recall.
Cross-boundary trust	My travel agent negotiating with the airline’s booking agent, with scoped authority from me, across trust boundaries, with witnessed finality. No memory API contemplates this.
Finality	When an agent spends your money, was the action witnessed, committed, irreversible? Or is it a log entry that could be rolled back or hallucinated?
Participant-specific authority	The same agent must behave completely differently depending on WHO it acts for. The authority derives from the participant’s context, not the agent’s config.

The memory market is building better filing cabinets. The agentic economy needs the person who owns the cabinet.

The Participant Layer provides both memory and existence. The critical insight: *if you solve existence, memory comes free*. Memory is one sub-context of the Context Superstate. If you only solve memory, you never get existence.

Illustration: Scoped agent mandate

```
Mandate: {agent: travel bot, scope: flights,
limit: $800, payment: Visa *6411, expires: Friday,
passport: read-once, revocable: immediate}
```

The mandate lives in the participant’s context. If the agent’s underlying LLM is manipulated through prompt injection, it does not matter: the Context Superstate enforces the mandate at the protocol level through KRAMA’s witness verification. The agent can attempt any action; only actions within the mandate’s scope will finalize.

	Agent Memory Players	Participant Layer (MOI)
State	Stateless (cached recall)	Stateful (computational existence)
Anchor	App-anchored	Participant-anchored
Authority	No	Scoped mandates, delegation, revocation
Privacy	Platform’s policy on you	Your policy on platforms
Finality	No	Witnessed, irreversible, provable
Value Actions	No	Native linear assets, commitments

Existence cannot be absorbed by LLM providers. OpenAI can add memory to ChatGPT. OpenAI cannot make ChatGPT into participant-level trust infrastructure: that requires linear state, witness attestation, cryptographic commitment, and a live validator network. That is a protocol, not a product feature.

Native Assets

Assets stay with owners. Not in contracts.

Smart contracts gave us programmable logic. You can write rules about value. But the value itself sits in the contract, not with the participant. Smart contracts can enforce rules about value. They cannot express *whose* value it is. That requires WHO.

1. Safe tokenization. Tokens are linear primitives in the participant’s context, with compliance that travels with the participant, not bolted onto the contract. A tokenized bond knows its compliance context: who can hold it, what jurisdictions apply, what reporting obligations attach.

2. Safe custodial exchange. Your assets stay in your Context Superstate throughout the exchange. The DEX is stateless logic. It never holds your assets. The worst case when application code has a bug is a failed interaction, not a drained pool.

3. Unified liquidity without pooling risk. Liquidity providers keep assets in their own contexts. The “pool” is an emergent aggregation of participant-held liquidity, not a single contract holding billions. \$2 billion in total liquidity means \$2 billion distributed across thousands of participant contexts, not one honeypot.

These capabilities are not features added to a blockchain. They are consequences of modeling value correctly.

Efficient AI

Emergent personal intelligence — dynamic and contextual. Not monolithic and impersonal.

The AI industry’s cost structure is unsustainable. Large Language Models maintain universal, participant-anonymous weights. Every inference searches the entire representational space. The

millionth query costs exactly what the first one cost, because nothing was learned, nothing was adapted, nothing was personalized. This is transaction economics applied to intelligence.

The Participant Layer introduces participation economics for AI. The Context Superstate accumulates a participant's variables and relationships — preferences, patterns, decision history, trust configurations — creating a small, dense, participant-specific intelligence that emerges from the Context Superstate with anchored referral knowledge from general-purpose models. The system relies less on expensive commodity LLM calls and more on accumulated context. Mature participants become cheaper to serve while their intelligence value increases.

Illustration: MaiSeer — the first test of participant-indexed intelligence

MaiSeer is a consumer prediction platform that constitutes the first empirical test of participant-indexed intelligence. Each participant's predictions, outcomes, and decision patterns accumulate in their Intelligence sub-context, creating a personal awareness graph that improves with every interaction. The system tests whether participant-indexed computation produces intelligence that anonymous models cannot, at a fraction of the cost.

The difference between general intelligence and personal intelligence is dimensional, not parametric. No amount of scaling produces a WHO. The Participant Layer provides the substrate for emergent personal intelligence: dynamic and contextual, not monolithic and impersonal.

The Participant Layer Is the New App Layer

Every SaaS developer is asking the same question: if LLMs can replicate my features, where is my moat?

The answer is the Participant Layer.

Three shifts are already underway. Trust moved from the application to the participant: scoped authority, delegation, and revocation live in the Context Superstate, not the app's database. Context moved from the platform to the participant: preferences, history, and credentials are participant-owned, not app-anchored. Intelligence moved from the model to the participant: accumulated interaction patterns create participant-indexed intelligence that no model provider can claim.

When trust, context, and intelligence all live with the participant, the application becomes thin: stateless logic plus a view layer. It doesn't store your data. It doesn't hold your authority. It doesn't own your intelligence. It computes on your behalf, within the boundaries you define, and returns results to your context.

The developer moat relocates. It is no longer feature depth: LLMs commoditize features. It is participant context depth: how deeply the application integrates with the participant's Context Superstate.

The application layer of the internet is about to relocate. The Participant Layer is the infrastructure it relocates to.

Three Markets, One Primitive

Trust for Agents (\$500B+). Memory tells an agent what you prefer. Existence tells an agent who you are, what you own, what you authorize, and who witnessed it. The Participant Layer provides both.

Trust for Value (\$2T+). Tokenization requires that digital representations maintain the linear properties of the underlying asset. The Participant Layer provides native linear assets, direct transfer, unified liquidity, and compliance that travels with the participant.

Trust for Enterprise (\$200B+). Enterprise digitization generates \$500 billion annually in trust-verification friction. The Participant Layer provides on-demand state access, cross-platform evidence, and portable compliance without rip-and-replace.

Intelligence for AI (\$5T+). The AI industry's linear cost scaling is unsustainable. The Participant Layer provides emergent personal intelligence through the Context Superstate: value computation that compounds rather than recomputes, making AI economically viable at population scale.

The Network Is Live

Metric	Status
Validators	2,100+ (live mainnet)
Annual Revenue	\$500K+ (pre-token, enterprise)
R&D Investment	7 years, \$8M bootstrapped
Capital Efficiency	40× vs comparable L1s
Technology	Patented. Provisional quantum-resistant patents.
Token	KMOI: staking, gas, identity. 10B supply, 50% gas burnt.

Closing

Computation was built to manage information. The use moved to value. Nobody changed the machine.

We changed the machine.

Apps and smart contracts define the WHAT. Cloud and data availability provide the WHERE. LLMs provide generic intelligence as a commodity. The Participant Layer and the Context Superstate provide the WHO, and with it: structural trust, portable context, and participant-centric intelligence.

Information can be processed anonymously. Value requires participants.

The Participant Layer. The network structure for value.

Contextual Compute. The general theory of computation.

MOI Protocol · moi.technology

Anantha Krishnan · Founder, CEO & Chief Architect

aikrish@moinetwork.io

Appendix: Core Thesis

1. Contextual Compute is a computational paradigm for managing value.
2. Value spans four domains: assets, trust, intelligence, and information.
3. Modern computing systems operate on anonymous global state, a model optimized for processing information.
4. Value interactions are different. They are inherently bound to participants and context.
5. Contextual Compute introduces participant-indexed computation, where state, authority, and intelligence are anchored to explicit participants rather than anonymous global state.

Classical computation: $\delta(S, I) \rightarrow (S', O)$

Value computation: $\psi(P, C, I) \rightarrow (P, C', V)$

Where: P = participant, C = context, I = input, V = value transformation

6. The infrastructure that emerges from this paradigm is the Participant Layer. It enables participant-scoped state, authority, and interaction.
7. Three structural properties follow:
 - Trust at the infrastructure level, across Web2, Web3, AI, and enterprise systems
 - Stateful participant existence: portable identity, memory, and context
 - Edge intelligence: emergent and personal, not monolithic and general
8. **MOI is the open network implementing Contextual Compute.**
9. Where information systems operate on anonymous state, Contextual Compute enables computation to operate directly on value. The evolution of networks reflects this shift.

TCP made messages a network primitive.

HTTP made resources a network primitive.

Contextual Compute makes participants a network primitive.

“Information can be processed anonymously. Value requires participants.”