Stop Agent Drift

BUILDING TRUSTWORTHY MULTI-AGENT SYSTEMS

Kenneth Pernyer · converge.zone

About Me

Kenneth Pernyer — Builder, systems thinker

- Founder @ Aprio One, Hey SH
- Building Converge — trustworthy agent runtime

The Agents Are Coming

Everyone is building multi-agent systems:

- **AutoGPT** — Autonomous task completion
- **BabyAGI** — Task decomposition and execution
- **LangChain Agents** — Tool-using LLMs
- **CrewAI** — Role-based agent teams

But there's a problem...

Agent Drift: The Silent Killer

> "The agent was supposed to send an email. It rewrote my entire marketing strategy."

Symptoms:

- Non-deterministic outputs
- Hallucinated intermediate steps
- Compound errors across agent chains
- "It worked yesterday, not today"

Root cause: No formal execution model.

The Zapier Analogy

Zapier works because:

- Deterministic steps
- Clear data flow
- Predictable execution

Agent frameworks fail because:

- LLMs are stochastic
- No convergence guarantees
- State scattered across calls

What if we could have both?

The Converge Model

AGENTS PROPOSE. ENGINE DECIDES.

Fixed-Point Semantics

Instead of: `agent₁ → agent₂ → agent₃ → done?`

We do:

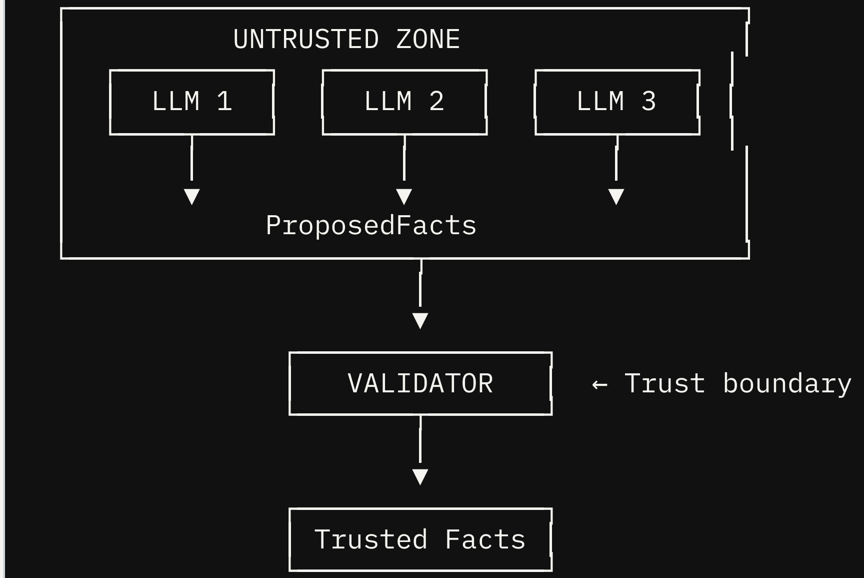
```
while ctx != step(ctx):
    proposals = parallel_run(agents, ctx)
    validated = validate(proposals)
    ctx = ctx.derive(validated)
return ctx  # Fixed point reached
```

Guaranteed termination. Guaranteed consistency.

The 9 Axioms

Axiom	What It Means
Monotonicity	Facts never lost
Determinism	Same input = same output
Idempotency	Safe to retry
Commutativity	Order doesn't matter
Termination	Always halts
Consistency	No contradictions
Starvation Freedom	Every agent gets a turn
Confluence	Branches merge cleanly
Observability	Full audit trail

Trust Boundaries for LLMs



Demo: Invoice Agent

```rust
impl Agent for InvoiceAgent {
    fn can_run(&self, ctx: &Context) -> bool {
        ctx.has_completed_work_without_invoice()
    }

    fn run(&self, ctx: &Context) -> AgentResult {
        let work = ctx.completed_work_without_invoice();
        let invoice = Invoice::generate(&work);

        AgentResult::Propose(vec![
            Fact::derived("invoice", invoice)
        ])
    }
}
```

Pure function. Testable. Deterministic.

Demo: With LLM Assistance

```
impl Agent for SmartInvoiceAgent {
    fn run(&self, ctx: &Context) -> AgentResult {
        let work = ctx.completed_work();

        // LLM helps with description
        let description = self.llm.complete(
            &format!("Summarize this work for an invoice: {:?}", work)
        )?;

        // But the invoice structure is deterministic
        let invoice = Invoice {
            amount: work.calculate_total(),  // Deterministic
            description: description,         // LLM-assisted
            ..Invoice::default()
        };

        // Output is ProposedFact, not trusted
        AgentResult::Propose(vec![
            Fact::proposed("invoice", invoice)
        ])
    }
}
```

The Validator Pattern

```
struct InvoiceValidator;

impl FactValidator for InvoiceValidator {
    fn validate(&self, fact: &Fact, ctx: &Context) -> ValidationResult {
        let invoice: Invoice = fact.payload()?;

        // Business rules enforced deterministically
        if invoice.amount <= 0.0 {
            return ValidationResult::Reject("Amount must be positive");
        }

        if !ctx.has_delivery_proof(&invoice.work_id) {
            return ValidationResult::Reject("Missing delivery proof");
        }

        ValidationResult::Accept
    }
}
```

Real-World Example: Competitive Analysis

```
Cycle 1: Human seeds competitor URLs
Cycle 2: LLM extracts signals (ProposedFacts)
Cycle 3: Validator filters hallucinations
Cycle 4: LLM generates hypotheses (ProposedFacts)
Cycle 5: Validator checks against known data
Cycle 6: Pure agent creates competitor profiles
Cycle 7: Pure agent scores strategies
Cycle 8: Fixed point → Final report
```

LLMs help. Rules decide. Humans trust.

Why This Matters

BEYOND THE DEMO

For Production Systems

Without Converge	With Converge
"Hope it works"	Guaranteed termination
Debug with print statements	Time-travel debugging
Retry and pray	Idempotent by design
Logs scattered	Merkle-audited trail
"Works on my machine"	Deterministic replay

For Compliance

Regulated industries need:

- **Explainability** — Why did it decide this?
- **Auditability** — What happened, when, by whom?
- **Reproducibility** — Run it again, get same result
- **Non-repudiation** — Cryptographic proof of execution

Converge provides all four.

For Teams

```
"Did the invoice go out?"

Before: Check 5 systems, ask 3 people
After:  converge trace invoice:12345

        Cycle 3: InvoiceAgent proposed invoice
        Cycle 3: Validator accepted (delivery proof: dp:789)
        Cycle 4: EmailAgent proposed notification
        Cycle 4: Fixed point reached
        Result: Invoice sent at 2024-01-15T10:23:00Z
```

Getting Started

TRY IT TODAY

The Stack

Component	What It Does
converge-core	Rust runtime, axiom enforcement
converge-provider	LLM integrations (Claude, GPT, Ollama)
converge-domain	Business Packs (Money, Customers, etc.)
converge-tool	Validation, testing, debugging
converge-ledger	Elixir/Phoenix for event sourcing

Quick Start

```
cargo install converge-cli
converge new my-project
cd my-project
converge run
```

Or add to existing project:

```
[dependencies]
converge-core = "0.1"
```

Resources

- **Website**: converge.zone
- **Docs**: docs.rs/converge-core
- **GitHub**: github.com/kpernyer/converge-core
- **Discord**: discord.gg/converge

Q&A

STOP DRIFT. START CONVERGING.

Let's Talk

Kenneth Pernyer

LinkedIn Twitter GitHub

converge.zone

Packages

CRATES.IO CONVERGE-CORE

HEX.PM CONVERGE_LEDGER

GHCR.IO CONVERGE-LEDGER