Why Agents Need an Operating System:
Infrastructure, Not Intelligence, as the Binding Constraint

fold research

research note rn-01 · **Base mainnet** · agent operating systems

Abstract

We argue that the limiting factor on autonomous onchain agents is not the quality of their reasoning but the absence of the system-level services ordinary software takes for granted. An agent that can plan a purchase but cannot hold funds, prove it is authorised to act, bound its own spending, or survive a restart is a demonstration, not a deployment. We describe the agent as a *process*—a wallet, a mandate, a policy, private state, and priced tools—and the operating system it requires around that process: machine-native payments over HTTP 402, mandates enforced beneath the language model, gasless execution through scoped session keys, and a short account of when paying for information is rational for software. The claim is deliberately narrow. Agents fail in the layers around the model, and those layers, assembled, are an operating system.

1 Introduction

The prevailing bet is that better models produce better agents. It is a reasonable bet, and it is not the one this note is about. Watch a capable agent fail in the field and it rarely fails because it reasoned poorly. It fails because it had no wallet and could not pay for the data the task required. It fails because it had no durable identity and could not prove it was permitted to act. It fails because nothing bounded its spending, so a human had to sit behind every step. It fails because it lost its working state the moment the process restarted. None of these are reasoning failures. They are the absence of an operating system.

This inversion matters because it changes where effort should go. If the binding constraint were intelligence, the right response would be to wait for the next model. Because the binding constraint is infrastructure, the right response is to build it. The rest of this note takes the components an agent is missing and names them as what they are: the process abstraction, the system call, the permission model, and the resources a process needs to run unattended.

2 The agent as a process

In an operating system a process is not merely code. It is code together with the resources and permissions that let it run without a human holding its hand: an identity, a memory space, a scheduler slot, handles to the outside world, and a set of capabilities it is allowed to exercise. An autonomous agent needs the same envelope, and today it usually has none of it.

OS primitive	Agent equivalent
Process identity	wallet + mandate
System call	priced tool call over x402
Permissions	mandate constraints, budgets
Scheduler	trigger and execution loop
Memory / files	private durable state
User account	onchain identity

Table 1. The components an autonomous agent requires, against the operating-system primitives they correspond to.

We identify five components. First, a *wallet*: the agent must be able to hold and move value itself, not borrow a human's card at the moment of purchase. Second, a *mandate*: a human-authored statement of what the agent may do and spend, expressed where the agent cannot edit it. Third, a *policy*: the runtime rules—budgets, rate limits, halt conditions—that turn the mandate into enforcement. Fourth, *private state*: durable memory that outlives a single invocation, so the agent is a process and not a fresh prompt each time. Fifth, *priced tools*: access to data, inference, and execution that the agent can purchase atomically as it works. Table 1 states the correspondence directly.

3 Payments as a system call

Most software assumes payment happens out of band: a human provisions an account ahead of time, and the code inside never touches money. That assumption breaks for an agent, because the agent is the one deciding, at runtime, that a piece of data is worth buying. Payment must therefore be a primitive the agent can invoke mid-task, the way a program invokes a read or a write.

HTTP 402 Payment Required, long reserved and now revived as x402, provides exactly this shape. A server answers a request with a price and a challenge; the client pays and retries; the work is performed. Payment becomes a system call: synchronous, in-band, and settled per request. Because settlement is per call and priced in cents, an agent can reach a service it was never pre-registered with, pay for the single answer it needs, and move on—no account, no subscription, no human in the provisioning loop. This is the difference between a program that can only use what it was handed and one that can go and acquire what a decision requires.

4 Mandates as enforcement below the model

A safety story that lives inside the prompt is not a safety story. Anything expressed only in natural language to the model can, in principle, be argued away by the model. The language model proposes; something beneath it must dispose.

A mandate is a human-authored constraint—a spending ceiling, a scope of permitted actions, a condition that halts the agent—compiled down to a layer the model cannot talk its way past. Enforced at the level of the wallet and the session key rather than the prompt, the mandate becomes a safety envelope with the same standing as file permissions in an operating system: not advice the process chooses to follow, but a boundary the system imposes. This is what lets an agent be left running. The human writes the envelope once; the infrastructure holds it every step after.

5 Gasless execution and session keys

The distance between a demonstration and a deployment is often measured in gas and keys. An agent that must custody a hot private key and pay gas for every action is both fragile and dangerous: fragile because a stalled or underfunded key stops it, dangerous because a single compromised key is unbounded authority.

Two primitives close this gap. Session keys grant the agent scoped, revocable authority—permission to spend up to a limit, on a set of actions, for a bounded time—without exposing the account's full control. Gasless execution, through sponsorship or meta-transactions, removes the requirement that the agent hold and manage a gas balance to act at all. Together they move an agent from impressive-in-a-notebook to safe-to-leave-running, which is the only threshold that matters for autonomy.

6 The unit economics of paid information

When is it rational for autonomous software to pay for data? The account is short. If a piece of information changes the expected value of a decision by more than its price, acquiring it is rational; otherwise it is not. Humans approximate this reasoning slowly and coarsely. Software can evaluate it per call, in real time, against a budget it is holding.

At cents-scale pricing the threshold sits very low, and this is precisely why machine-native micropayments unlock behaviours that subscriptions cannot. A subscription forces a large, ahead-of-time commitment to a single provider; a per-call market lets an agent buy the one fact it needs from whoever offers it, at the moment the decision is being formed. The economic primitive and the payment primitive are the same primitive seen from two sides.

7 What an operating system provides

None of the failures in Section 1 live in the cryptography or in the model. They live in the layers around them—the wallet, the mandate, the payment path, the durable state. An operating system for agents is simply the name for those layers, assembled into something a process can rely on: value it owns, authority it answers to, purchases it can make, memory it can keep.

fold is a bet that this layer, and not the next model, is the binding constraint on autonomous onchain agents. The bet is falsifiable in the ordinary way: if agents built on this substrate can be left running, held to a budget, and trusted to transact unattended where bare models cannot, the layer was the missing thing. This note is early and the substrate is still moving; the argument, we think, is not.