

DEVCON 8 · MUMBAI · 3–6 NOVEMBER 2026

The Wallet Is the New Trust Boundary

Verifying Ethereum at the Edge

Steffen Kux · Co-CEO & Co-Founder · corpus.core

DON'T TRUST. VERIFY!


Every shortcut that assumes trust eventually costs freedom.

The Trustless Manifesto

Authors: Yoav Weiss, Vitalik Buterin, Marissa Posner

Special thanks to pcaversaccio, Tim Clancy, and Dror Tirosh for feedback and review.

I. Why trustlessness matters

Every system begins with good intentions.
A hosted node here, a whitelisted relayer there.
Each is harmless on its own — and together they become habit.

Gateways become platforms.
Platforms become landlords.
Landlords decide who may enter and what they may do.

The only defense is *trustless design*: systems whose correctness and fairness depend only on math and consensus, never on the goodwill of intermediaries.

Trustlessness is not a feature to add after the fact.
It is **the** thing itself.
Without it, everything else — efficiency, UX, scalability — is decoration on a fragile core.

Trustlessness is how credible neutrality is achieved.
Without it, the system becomes one that depends on intermediaries.

II. Why we build on Ethereum

e build on Ethereum because we choose verification over blind tr
that pow to that pow behind policy.
design pr ols so freedom d not depend on permissio

thereum was creat to make finance efficient or apps conven
was created et peo power anyone, anywhere t ordinate with
permission and without trusting anyone they cannot hold accountable.

Ethereum is the path to trustlessness.
That promise must not be lost as we scale.

III. What trustlessness means

A system is trustless when any honest participant can join, verify, and act **without permission and without fear**.

This requires:

1. **Self-sovereignty** — each user authorizes their own actions; no one acts on their behalf.
2. **Verifiability** — anyone can confirm what happened from public data.
3. **Censorship resistance** — any valid action can be included, within a reasonable timeframe and without undue cost.
4. **Walkaway test** — if one operator disappears or misbehaves, another can step in without approval.
5. **Accessibility** — participation must be within reach of ordinary users, not only experts with servers and capital.
6. **Transparency of incentives** — participants are governed by protocol rules, not private contracts or opaque APIs.

Remove any of these, and the system drifts from protocol to platform — from neutral ground to private property.

IV. What trustlessness demands

Trustlessness is expensive.
It requires redundancy, openness, and complexity.
It requires mempools that anyone can use, even if that invites spam.
It requires clients that anyone can run, even if few will.
It requires governance that moves slowly, because no one can override consensus.

Every shortcut that assumes trust eventually costs freedom.
Trustlessness depends on free open source software — not only code that can be read, but code that can be freely run, verified, and improved.

A trustless design must therefore obey three laws:

1. **No critical secrets.**
   No step of a protocol should depend on private information held by a single actor — except the user themselves.

2. **No indispensable intermediaries.**
   Anyone who forwards, executes, or attests must be replaceable by any other participant following the same rules.
   "Anyone can run one" is not enough — participation must be *practically* open, not reserved for those with servers, fundi d DevOps
   A system that depends on int diaries mo ers cannot realistically replace is not trus cc the ds of a operator

3. **No unverifiable outcomes.**
   Ev t must be able to check

These laws are harsh. They limit what we can build easily.

V. The drift toward dependence on trust

The drift is not theoretical. It is already here.
Hosted RPCs are the default.
If AWS, GCP, and Cloudflare went dark, most apps would too.
Sequencing is centralized by design in many rollups.
Upgrade keys still exist. "Training wheels" are used as an excuse to delay decentralization.
"Self-custody" is delegated to CEXes.
Cross-chain interoperability has begun to mirror the very centralization it was meant to overcome — solvers and relayers act as gatekeepers of execution, deciding which transactions succeed and which fail.

Trust does not return all at once.
It returns through defaults slowly.
Each choice feels harmless, temporary — not like centralization.

No capture, no coup — just comfort.
Help becomes habit; habit becomes dependence.
Soon participation depends on infrastructure that only a few can run, and verification becomes the privilege of specialists.

This is how decentralization erodes — not in theory, but in production.
Not tomorrow, but **today**.

Decentralization erodes not through capture, but through convenience.
It drifts — automatically, continually — toward dependence on trust.

VI. The limits of delegation

Delegation may exist. Dependence must not.
Users may choose convenience — a hosted node, a friendly UI, a service that helps them — the protocol must never require it.

If inclusion requires an intermediary, it is not trustless.
If that intermediary must be trusted, it is not neutral.
If users can "theoretically" run their own intermediary but in practice never will, the system has replaced permissionless access with *technical gatekeeping*.

A permissionless protocol must not only be *open in code*, but *open in cost, accessibility, and comprehension*.

We have seen this pattern before.
Email was once fully open — anyone could run their own mailserver.
In theory, you still can.
But in practice, spam ocklists, and on systems make it nearly impossible for ordinary users.
became effecti ized — n on the pr ead, but
al trustless as

not let Ether s sc ow

VII. The duty of builders

We who design protocols are stewards, not gatekeepers.
Our duty is not to build what is easiest, but what remains open and self-sovereign.

When complexity tempts us to centralize, we must remember: every line of convenience code can become a choke point.
When critics ask why our designs are complicated, we should ask them what — or **whom** — they are trusting instead.
If simplicity comes from trust, it is not simplicity. It is surrender.

Trustlessness costs computation, latency, and mental effort.
It buys resilience, longevity, neutrality, and freedom.

Trustlessness also requires viable incentives.
Systems must reward those who sustain them without turning them into gatekeepers.
A protocol that relies on unpaid altruism will decay.
A protocol that rewards control will centralize.
The only stable equilibrium is one where neutrality is profitable.

VIII. The way forward

Ethereum has scaled — now it must stay honest.

As we build new layers, new accounts, and new ways to interact, we must preserve the properties that made Ethereum matter:

- Users initiate their own actions.
- Anyone can verify and participate.
- No one can be silently excluded.
- Logic lives in code, not in contracts of trust.
- inclusion depends on incentives, not on reputation or permission.

Trustlessness is not perfection.
it is a system that fails publicly, transparently, and recoverably — rather than privately and silently.

IX. The pledge

We choose to build trustless systems even when it is harder.
We pay the cost of openness over the convenience of control.
We do not outsource neutrality to anyone who can be bribed, coerced, or shut down.
We measure success not by transactions per second, but by **trust reduced per transaction**.

We refuse to build on infrastructure we cannot replace.
We refuse to call a system "permissionless" when only the privileged can participate.
We refuse to trade autonomy for polish, or freedom for frictionless UX.
Trustless t preserved by consensus alone — it is preserved by resistan
If your p ol requires faith in a intermediary, change
If it rel a private ga y, replace
If it hide cal state d ficiently, expose it.
In the en e world do ot need more efficient dilemen
It needs r reasons to

Trustlessness is the foundation.

Everything else is construction on top of it.

The designs will change. The principles will not.

Sign the Trustless Manifesto

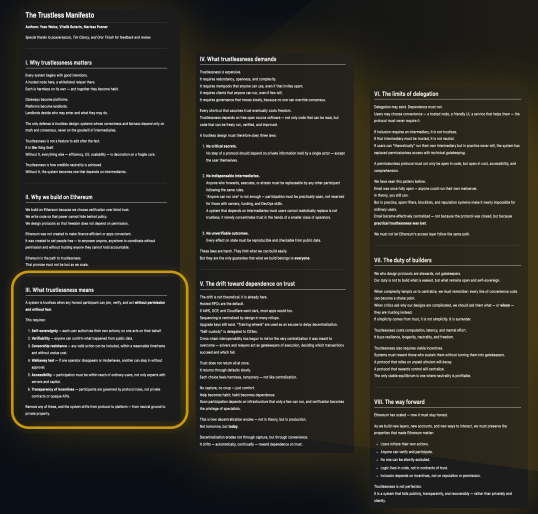
By signing this pledge, you affirm your commitment to building and using systems that preserve **trustlessness** - systems where correctness and fairness depend only on math and consensus, not intermediaries.

Your signature will be recorded **on Ethereum mainnet** in the Trustless Manifesto contract.

Sign the Trustless Manifesto Pledge



What **Trustlessness** means


The Trustless Manifesto

Authors: Yoav Weiss, Vitalik Buterin, Marlena Fraser

Special thanks to powerpoints, Tim Cleary, and Ether Fiend for feedback and review

I. Why trustlessness matters

Every system begins with good intentions.
A trusted node here, a whitelisted relayer there.
Each is harmless on its own — and together they become habit.

Gateways become platforms.
Platforms become landlords.
Landlords decide who may enter and what they may do.

The only defense is a system designed to remain honest even when its operators are not.
Trustlessness is not a feature to add after the fact.
It is the thing itself.
Without it, everything else — efficiency, UX, scalability — is decoration on a fragile core.

Trustlessness is how credible neutrality is achieved.
Without it, the system becomes one that depends on intermediaries.

II. Why we build on Ethereum

We build on Ethereum because we choose verification over blind trust.
We write code so that power cannot hide behind policy.
We design protocols so that freedom does not depend on permission.

Ethereum was not created to raise finance efficient or enjoy convenient.
It was created to set people free — to empower anyone, anywhere to coordinate without permission and without trusting anyone they cannot hold accountable.

Ethereum's the path to trustlessness.
That promise must not be lost as we scale.

III. What trustlessness means

A system is trustless when any honest participant can join, verify, and act without permission and without fear.

This requires:

1. Self-sovereignty — each user authorizes their own actions; no one acts on their behalf.
2. Verifiability — anyone can confirm what happened from public data.
3. Censorship resistance — any valid action can be included, within a reasonable timeframe and without undue cost.
4. Walkaway test — if one operator disappears or misbehaves, another can step in without approval.
5. Accessibility — participation must be within reach of ordinary users, not only experts with servers and capital.
6. Transparency of incentives — participants are governed by protocol rules, not private contracts or opaque APIs.

Remove any of these, and the system drifts from protocol to platform — from neutral ground to private property.

IV. What trustlessness demands

Trustlessness is expensive.
It requires redundancy, openness, and complexity.
It requires monopolies that anyone can use, even if fair circles open.
It requires clients that anyone can run, even if few will.
It requires governance that moves slowly, because no one can override consensus.

Every shortcut that assumes trust eventually costs freedom.
Trustlessness depends on free open source software — not only code that can be read, but code that can be freely run, verified, and improved.

A trustless design must therefore obey three laws.

1. No critical secrets.
No step of a protocol should depend on private information held by a single actor — except the user themselves.

2. No indispensable intermediaries.
Anyone who forwards, executes, or attests must be replaceable by any other participant following the same rules.
"Anyone can run one" is not enough — participation must be practically open, not reserved for those with servers, funding, and backup wills.

A system that depends on intermediaries most users cannot realistically replace is not trustless. It merely concentrates trust in the hands of a smaller class of operators.

3. No unverifiable outcomes.
Every effect or state must be reproducible and checkable from public data.

These laws are harsh. They limit what we can build easily.
But they are the only guarantee that what we build belongs to everyone.

V. The drift toward dependence on trust

The drift is not theoretical. It is already here.
Hosted RPCs are the default.
If AWS, DCF, and Cloudflare went dark, most apps would too.
Sequencing is centralized by design in many rollups.
Upgrade keys still exist. "Training wheels" are used as an excuse to delay decentralization.
"Self-custody" is relegated to L1Orbs.
Cross-chain interoperability has begun to let so the very centralization it was meant to overcome — screens and reopens our on gatekeepers of execution, deciding which transactions succeed and which fail.

Trust does not return all at once.
It returns through defaults slowly.
Each choice feels harmless, temporary — not like centralization.

No surprise, no coup — just comfort.
Help becomes habit, habit becomes dependence.
Soon participation depends on infrastructure that only a few can run, and verification becomes the privilege of specialists.

This is how decentralization erodes — not in theory, but in production.
Not tomorrow, but today.

Decentralization erodes not through capture, but through convenience.
It DIYs — automatically continually — toward dependence on trust.

VI. The limits of delegation

Delegation has merit. Dependence must not.
Users may choose convenience — a trusted node, a friendly UI, a service that helps them — the protocol must never require it.

If inclusion requires an intermediary, it is not trustless.
If that intermediary must be trusted, it is not neutral.
If users can "theoretically" run their own intermediary but in practice never will, the system has replaced permissionless access with territorial gatekeeping.

A permissionless protocol must not only be open in code, but open in cost, accessibility, and comprehension.

We have seen this pattern before.
Email was once fully open — anyone could run their own mailserver.
In theory, you still can.
But in practice, spam filters, blocklists, and reputation systems make it nearly impossible for ordinary users.
Email became effectively controlled — not because the protocol was closed, but because practical trustlessness was lost.

We must not let Ethereum's access layer follow the same path.

VII. The duty of builders

We who design protocols are stewards, not gatekeepers.
Our duty is not to build what is easiest, but what remains open and well-leverage.

When complexity leaves us to centralize, we must remember: every line of convenience code can become a chokepoint.

When critics ask why our designs are complicated, we should ask them what — or whom — they are trusting instead.

If simplicity comes from trust, it is not simplicity. It is surrender.

Trustlessness costs computation, latency, and mental effort.
It buys resilience, longevity, neutrality, and freedom.

Trustlessness also requires viable incentives.
Systems must reward those who sustain them without turning them into gatekeepers.
A protocol that relies on unpaid altruism will decay.
A protocol that rewards control will centralize.
The only stable equilibrium is one where neutrality is profitable.

VIII. The way forward

Ethereum has scaled — now it must stay honest.
As we build new layers, new accounts, and new ways to interact, we must preserve the properties that made Ethereum matter:

- Users initiate their own actions.
- Anyone can verify and participate.
- No one can be silently excluded.
- Logic lives in code, not in contracts of trust.
- Inclusion depends on incentives, not on reputation or permission.

Trustlessness is not perfection.
It is a system that fails publicly, transparently, and recoverably — rather than privately and silently.

IX. The pledge

We choose to build trustless systems even when it is harder.
We see the cost of trustlessness over the convenience of control.
We do not outsource neutrality to anyone who can be bribed, coerced, or shut down.
We measure success not by transactions per second, but by how resistant our transaction.

We refuse to build on chains/venues we cannot replace.
We refuse to call a system "permissionless" when only the privileged can participate.
We refuse to trade autonomy for scale, or freedom for frictionless UX.
Trustlessness is not preserved by convenience alone — it is preserved by resistance.

If your protocol requires faith in an intermediary, change it.
If it relies on a private gateway, replace it.
If it hides critical state or logic off-chain, expose it.

A chain for the world does not maximize efficient middlemen.
It makes them irrelevant, or it ruin them.

Trustlessness is the foundation.
Everything else is construction on top of it.
The designs will change. The principles will not.

Sign the Trustless Manifesto

By signing this pledge, you affirm your commitment to building and using systems that preserve trustlessness — systems where correctness and fairness depend only on math and consensus, not intermediaries.

Your signature will be recorded on Ethereum mainnet in the Trustless Manifesto contract.

Sign the Trustless Manifesto Pledge

...

A system is trustless when any honest participant can join, verify, and act without permission and without fear.

This requires:

1. Self-sovereignty — each user authorizes their own actions; no one acts on their behalf.
2. Verifiability — anyone can confirm what happened from public data.
3. Censorship resistance — any valid action can be included, within a reasonable timeframe and without undue cost.
4. Walkaway test — if one operator disappears or misbehaves, another can step in without approval.
5. Accessibility — participation must be within reach of ordinary users, not only experts with servers and capital.
6. Transparency of incentives — participants are governed by protocol rules, not private contracts or opaque APIs.

Remove any of these, and the system drifts from protocol to platform — from neutral ground to private property.

...

Who signed the Trustless Manifesto?

With which wallet?
And which RPC told it your nonce, the gas price, and the contract state?

Trustless Protocol, Trusted Wallet

"Hosted RPCs are the default." — The Trustless Manifesto

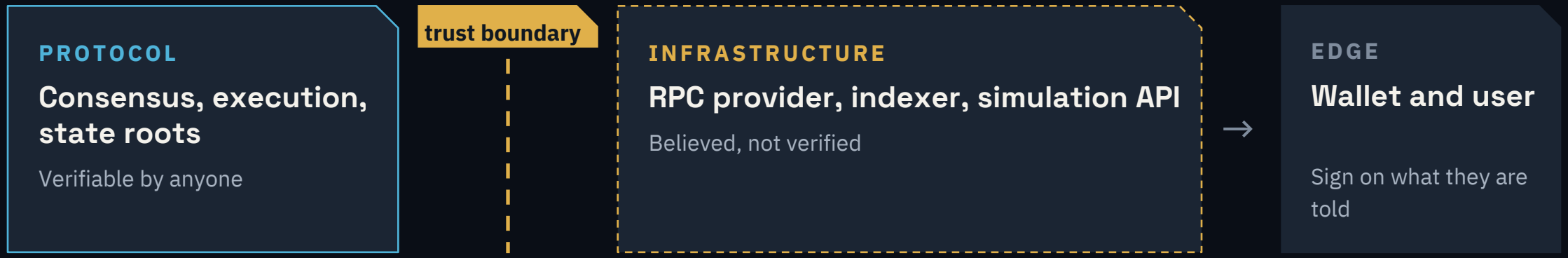
The wallet asks	Typical call
Balances and state	eth_getBalance, eth_getStorageAt
Contract reads	eth_call
History and events	eth_getLogs
Transaction preparation	eth_getTransactionCount, eth_estimateGas
Simulation	a remote simulation API

Every answer is believed, not verified.



** inspired by Jason Chaskin*

Where Trust Leaves the Protocol



The wallet sits on the wrong side of the line.

Every Read Is a Confession

The same boundary leaks what you are about to do

The wallet asks	The provider learns
Balances of your addresses	What you hold, and that the addresses belong together
Positions and allowances	Your protocols and counterparties
Pool and route queries	What you are about to trade, and when
Simulation of a pending transaction	Your full intent, before it exists on-chain

No transaction needed. The provider sees every query it receives, and patterns reveal more than facts.

Three Laws at the Edge

No critical secrets

"No step of a protocol should depend on private information held by a single actor — except the user themselves."

Wallet today: holds. Keys stay with the user.

No indispensable intermediaries

"Anyone who forwards, executes, or attests must be replaceable by any other participant following the same rules."

Wallet today: the RPC is swappable, but its answers are trusted.

No unverifiable outcomes

"Every effect on state must be reproducible and checkable from public data."

Wallet today: balances, calls and simulations are taken on faith.

Verification and read privacy belong in the protocol, not in the infrastructure.

Colibri: Verify What You Read

Stateless verification: no full node, no synced state, embedded in any wallet

UNTRUSTED

Any RPC

Data plus proofs. Could be lying.

→

COLIBRI · IN THE WALLET

Execution proofs

Accounts, storage, code, receipts against the state root

Consensus proofs

The block is canonical, down to a ZK sync proof of ≈350 bytes

→

Verified data

Regardless of where it came from.

Verified regardless of the source. And optionally private, too.

Uniswap

Nonce #47

1 **Review what you will sign**

Signing is an irreversible action so make sure you know what you are signing. Read more.

2 **Compare with your wallet**

Once you click **Sign**, the transaction will appear in your signing wallet. Make sure that all the details match.

3 **Verify with external tools**

You can additionally cross-verify your transaction data in a third-party tool like Safe Utils.

Data | Hashes | JSON

To	UniversalRouter
	0x23617e59A5925b2A4Bf75d73fff
Value	1000000000000000000
Data	0x3593564c000000000000000000000000000000000000000000000000000000000000006000000000000000000000000000000000000000000000000000000000

Execute at the Edge

eth_call and simulation, computed locally on Colibri-verified state

eth_call or unsigned tx
to · value · calldata

Verified state
every slot it touches, with a proof

→

</>

Local EVM
in the wallet

→

Return values
for eth_call

Transfers
ETH, tokens, NFTs

Approvals
who may spend

State changes
incl. delegatecalls

The wallet stops asking what will happen. It computes it.

From Effects to Intent

Verified data is not yet a verified decision

USER INTENT

Swap 1 ETH → USDC

LOCALLY SIMULATED EFFECTS (EXAMPLE)

Effect	Matches intent?
−1 ETH from your wallet	Yes, expected
+USDC to your wallet	Yes, expected
approve(USDT, 0x7a3..., unlimited)	No, unexpected
0.05 ETH sent to 0x7a3...	No, unexpected

Bybit had valid signatures and a lying display. The question is not “is this valid?” but “is this what I meant?”

Let AI Explain, Not Verify

Deterministic facts first. Probabilistic interpretation second.

DETERMINISTIC

What is true

Proofs and the EVM: verified state, executed effects. Reproducible.

PROBABILISTIC

What it means

A small domain model and rules: explanation, risk signal, recommendation.

Running in the wallet, not in the cloud: otherwise, we just built a new trust boundary.

Small Models, Narrow Expertise

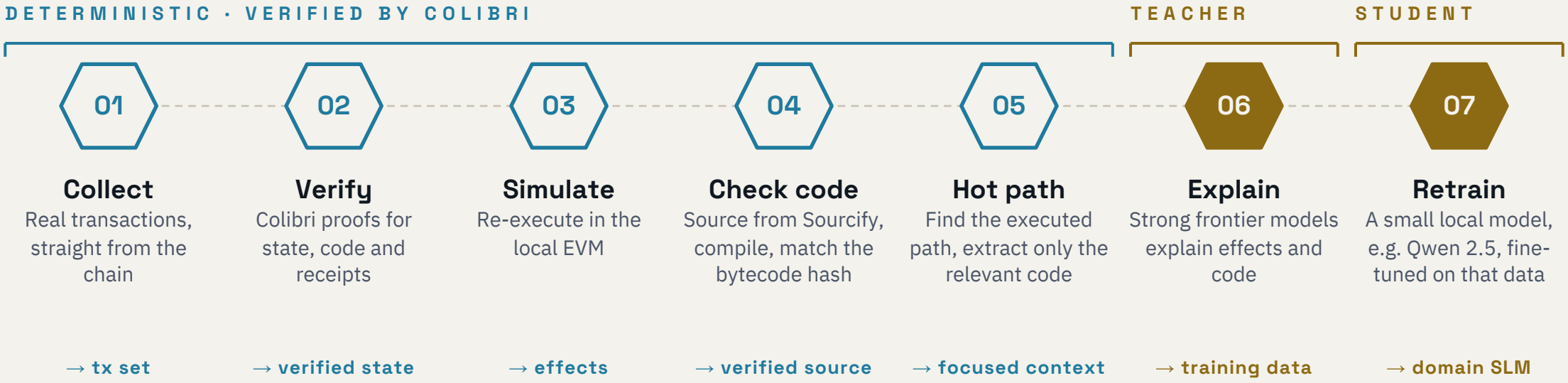
Frontier quality on one task, small enough to run in the wallet

	Frontier LLM	
Explaining transactions	Strong	
Everything else	Strong	
Size and location	Huge, in a data center	
New trust boundary	Yes, a remote provider	

Your wallet doesn’t need a model that writes poems and knows every celebrity breakup from the tabloids. It needs one that knows what this calldata does to your money.

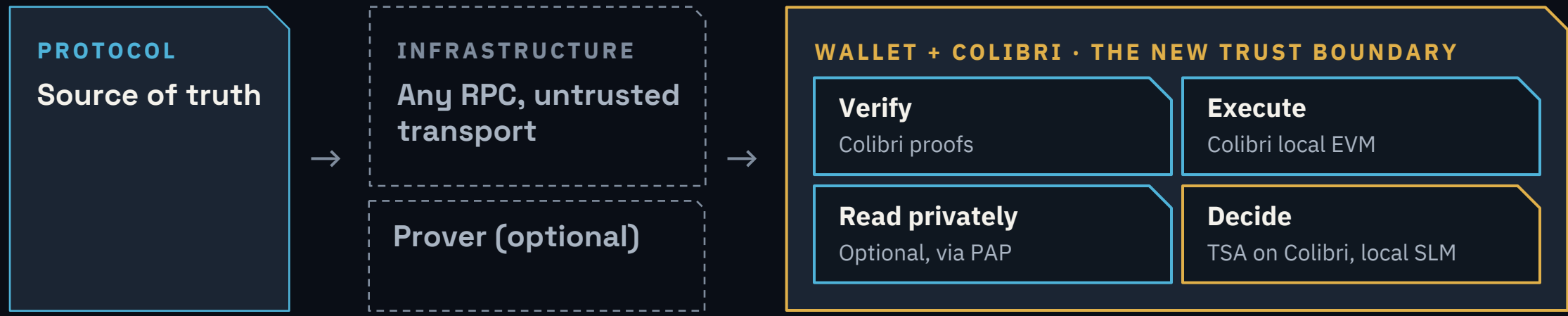
From Chain to Model

Verified facts in, explanations out: training data built from what actually ran



The frontier model teaches once, offline. The domain SLM explains every transaction, locally.

Wallet + Colibri: The Verification Engine



Infrastructure becomes transport. Colibri moves trust to where the user decides.

Live Demo

A wallet with Colibri: from verified read to verified decision.

Verify → Execute → Explain → Decide

CROPS Starts at the Edge

"Ethereum must, above all, remain censorship resistant, open source, private, and secure."

CR

Censorship resistance

Any RPC is replaceable, because none has to be trusted.

O

Open source

Colibri: verification you can audit and run yourself.

P

Privacy

Read privacy as a protocol capability, not an app feature.

S

Security

Verify reads, execution and intent before signing.

The protocol delivers CROPS to the chain. A wallet with a stateless client like Colibri delivers it to the user.

DON'T TRUST.

VERIFY!

AT THE EDGE.

GET INVOLVED

Try Colibri

Whitepaper



whitepaper.colibri-stateless.tech

Specification



specification.colibri-stateless.tech

GitHub Repository



repository.colibri-stateless.tech

corpuscore.tech X: @CorpusCoreHQ · @ColibriClient

TheDAO Security Fund

TYPE	INITIATIVE
GRANT	Make dApps Verify What They Show: colibri in Wallets, DeFi and Web3 Apps
GRANT	Trustless Across Ethereum: colibri Verification for L2 Networks and Bridges
GRANT	Know What You Sign: Verified Simulation Against Blind Signing





Thank you very much!

DON'T TRUST.
VERIFY!

Steffen Kux
Co-CEO & Co-Founder, corpus.core
https://corpuscore.tech