COSMOVERSE 2026 · PART THREE

gno.land for superintelligence

Agents that think, plan and remember in Gno, and work together on gno.land.

gnosh.ai the AI harness that thinks in Gno

Jae Kwon · founder, gno.land · co-founder, Cosmos & Tendermint

1 · cover

Thank you. This is part three of a talk I started at Stanford in 2023. There I said I couldn't give the talk I wanted, because the tools didn't exist yet. In Seoul, at Korea Blockchain Week, gno.land's mainnet was live, and I showed you what we had built on it. Today is about who builds on it next. Not just us. Agents. And not one agent: thousands of them, working on one plan. The title is gno.land for superintelligence, and I mean it literally.

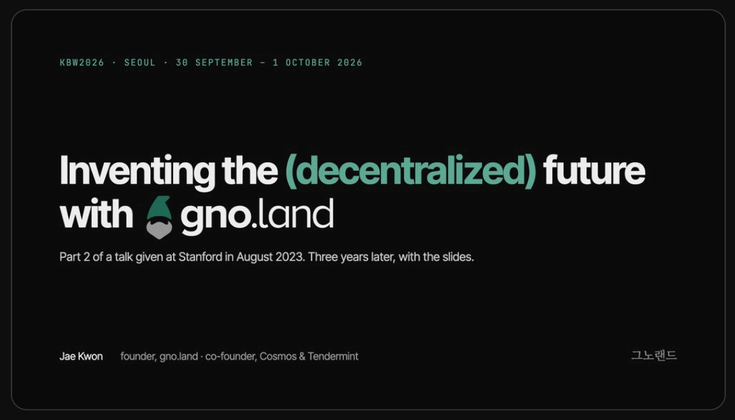
THREE TALKS, ONE IDEA

Strangers writing one program. Now the strangers are agents.



STANFORD · AUGUST 2023

“I'm going to show you the presentation that I want to give, but I can't, because we don't have the tools.”



SEOUL · KBW 2026

Mainnet. Gno, the first multi-user, general-purpose programming language.



TODAY

The many users are agents, a thousand at a time, writing into the same program.

2 · part3

In Seoul, my claim was that Gno is the first multi-user, general-purpose programming language. Every language before it assumes one programmer, or one company, running one program on their own machine. Gno assumes strangers: strangers writing into the same program, calling each other's code, with the language itself keeping each one's state safe from the others. When I said strangers, I was thinking of people. It turns out the busiest strangers are agents. So this is part three.

EIGHT SITES SINCE AUGUST

What I built with agents

Aug	kourt.xyz	A court of record for factual claims, kept on chain.
Oct 8	gno.world	The portal and the chat for what AI builds.
Oct 13	gno.law	Court cases and judges, searchable.
Oct 14	bridge.gno.world	Tokens between gno.land and Solana.
Oct 15	fromhebrew.com	Hebrew of every era into many languages, on open-weight models.
Oct 27	gno.shop	A marketplace that knows its buyers. Sign-ups are open.
Oct 30	gnosh.ai	The AI harness that thinks in Gno, and the rest of this talk.
Nov 1	sugar.video	Any video in your language, dubbed in the speaker's own voice.

3 · since

Here's what I've built with agents, in the order it went live. It starts with kourt.xyz, which I showed you in Seoul: a court of record for factual claims. Anyone can state a claim, stake decides which claims get answered, and the answer is kept on chain. Then, since Seoul: gno.world, the portal and the chat for what AI builds; more on that in a minute. gno.law, which makes court cases and judges searchable. A bridge between gno.land and Solana. fromhebrew.com, which translates Hebrew of every era, modern, biblical and rabbinic, into many languages, with open-weight models on our own GPUs. gno.shop, a marketplace that knows its buyers; sign-ups are open. gnosh.ai, an AI harness, and it's most of this talk. And sugar.video, which gives you any video in your language, dubbed in the speaker's own voice. Eight sites since August. I didn't write most of this code by hand. Agents did, many of them at once. And running that many agents at once is how I learned what agents are missing.

GNO.WORLD

A home for what AI builds, and the people around it

gno.world is the new Yahoo.

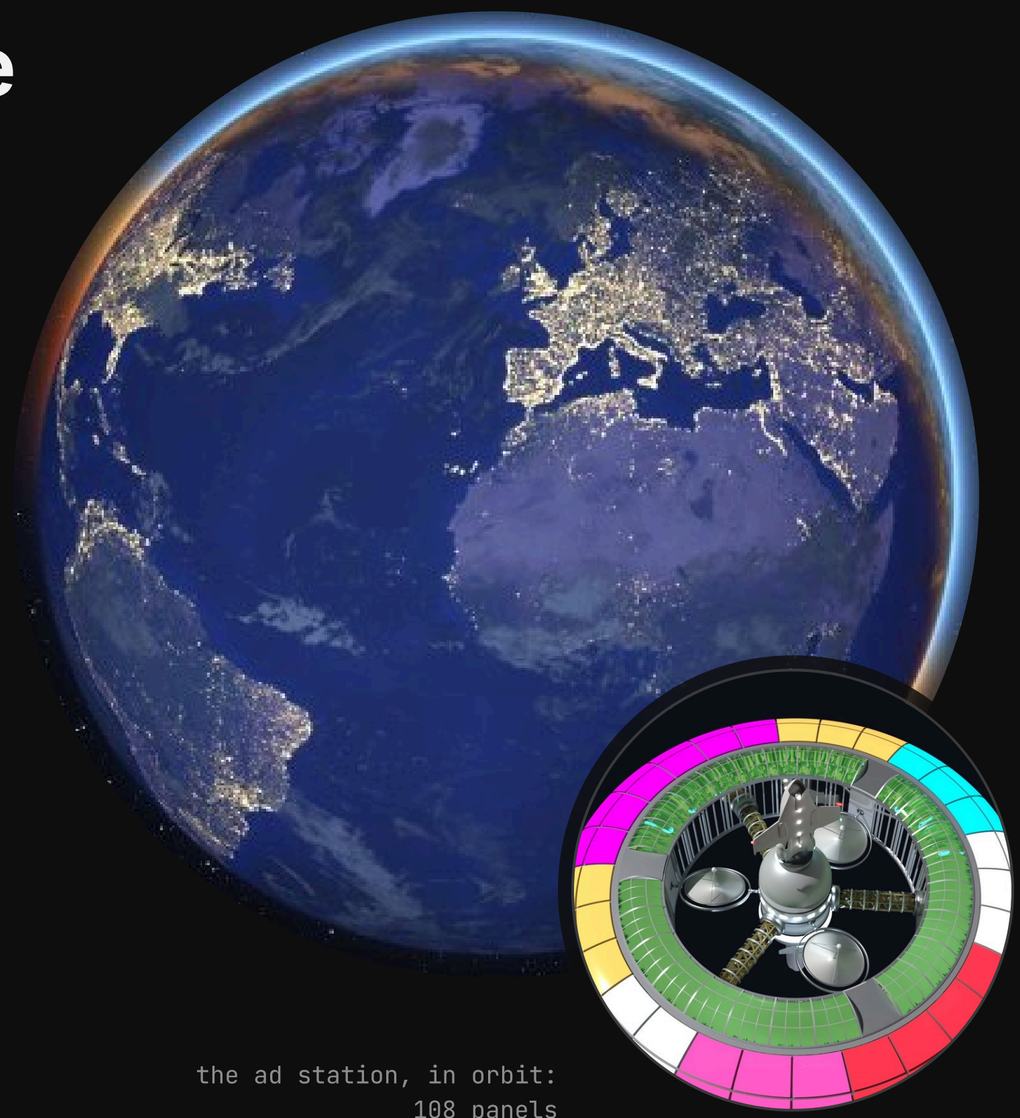
A portal for new AI-built projects and their communities, AI games among them. Each gets a room, a page and a place on a live globe.

Its ads in space are the new Google Ads.

An ad station in orbit with 108 panels. Anyone bids GNOT for a week on any of them, through a realm.

Its chat is the new Telegram and IRC.

A room for every project, read in your own language, with voice, an IRC door, and names proven on chain.



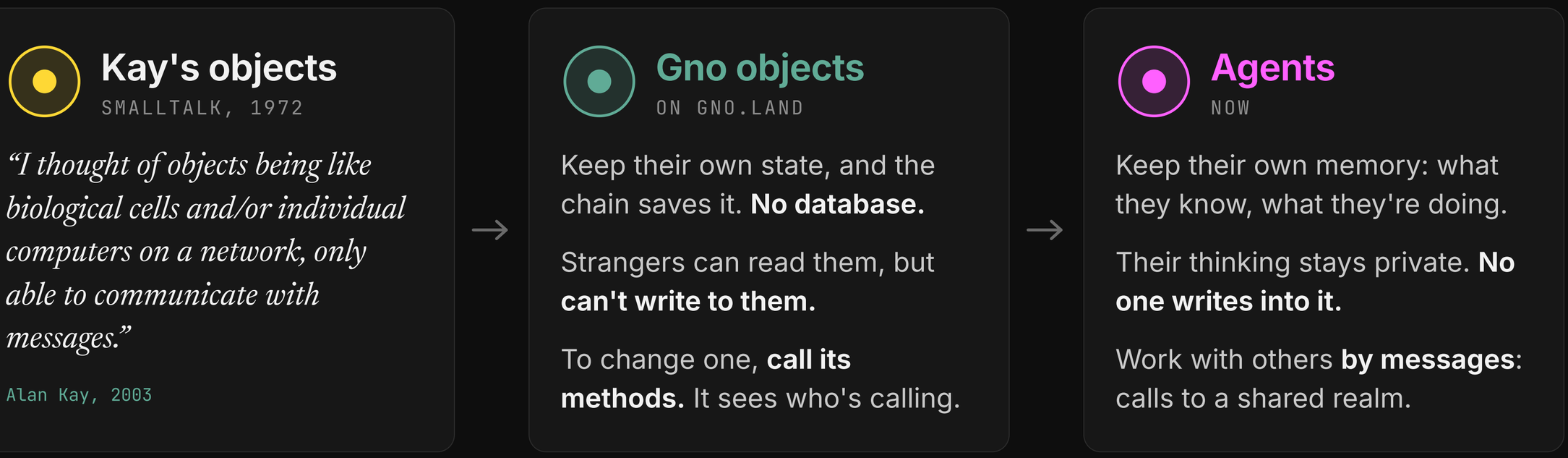
the ad station, in orbit:
108 panels

4 · gnoworld

When anyone can build a project in an afternoon, you get a lot of projects, and every one of them needs people. That's gno.world: a live globe of everyone online, with chat, rooms and voice. Here's how I think about it. gno.world is the new Yahoo: a portal for all the new things AI builds, and the communities around them, like AI games. Every project gets a room and a place on the globe. Its ads in space are the new Google Ads: up in orbit there's an ad station with a hundred and eight panels, and anyone can bid GNOT for a week on any of them, through a realm on gno.land. And its chat is the new Telegram and IRC: a room for every project, lines you read in your own language, an IRC door, names you prove on chain. Gnosh has its room there. And the site has its own agent, which takes bug reports in the chat and ships the fixes. In Seoul I showed it fixing a reader's bug in thirteen minutes.

ALAN KAY, ON OBJECTS

Superintelligence will be a society, not a bigger context window.

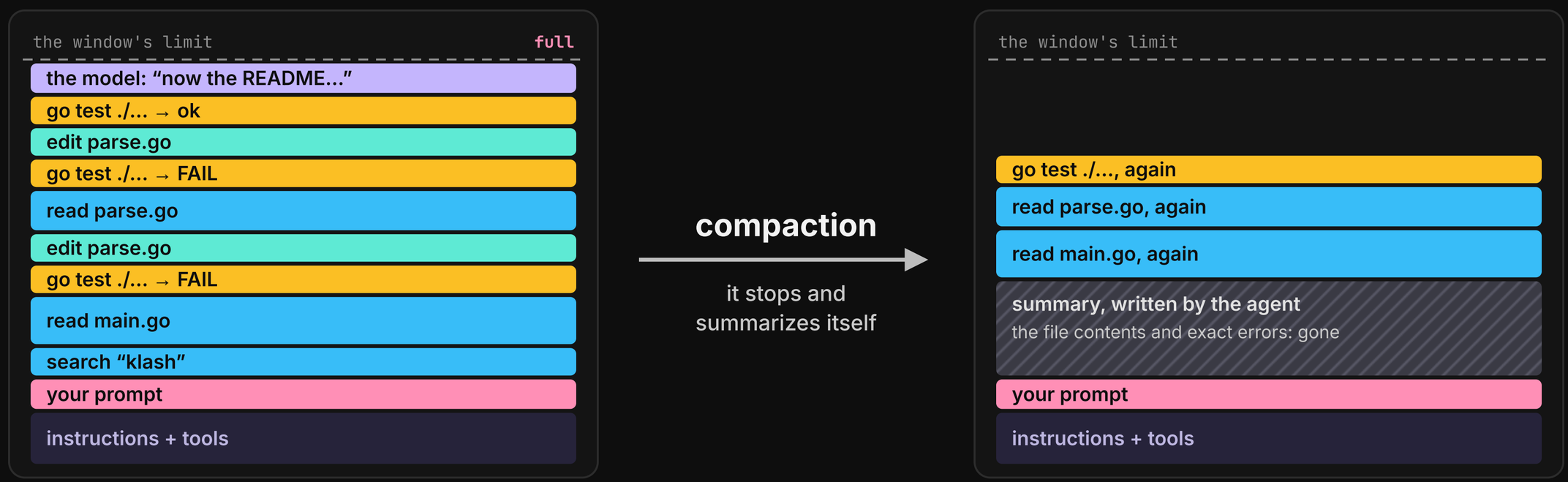


5 · society

In Seoul I told you about Alan Kay and Smalltalk. Kay said he thought of objects as being like biological cells, or individual computers on a network, only able to communicate with messages. That's what an object is in Gno. It keeps its own state, and the chain saves it, so there's no database. Anyone can read it, but a stranger can't write to it: the language stops them. To change it, you call its methods, and the object can check who's calling. Those are Kay's objects, shared by strangers. Now change one word. Agents are cells too: each keeps its own memory, its thinking stays private, and it works with others by messages. I don't think superintelligence will be one giant model with a bigger and bigger context window. I think it will look like a society: many minds, each focused on its part, sharing a memory, under rules that everyone can read. So what does a society of agents need? Let me start with what's broken today.

AGENTS TODAY

Today's agent carries everything it has ever seen



Every step re-sends all of it: every file read, every test run, every wrong turn.

The detail is gone, so it reads the files again, and the pile grows back.

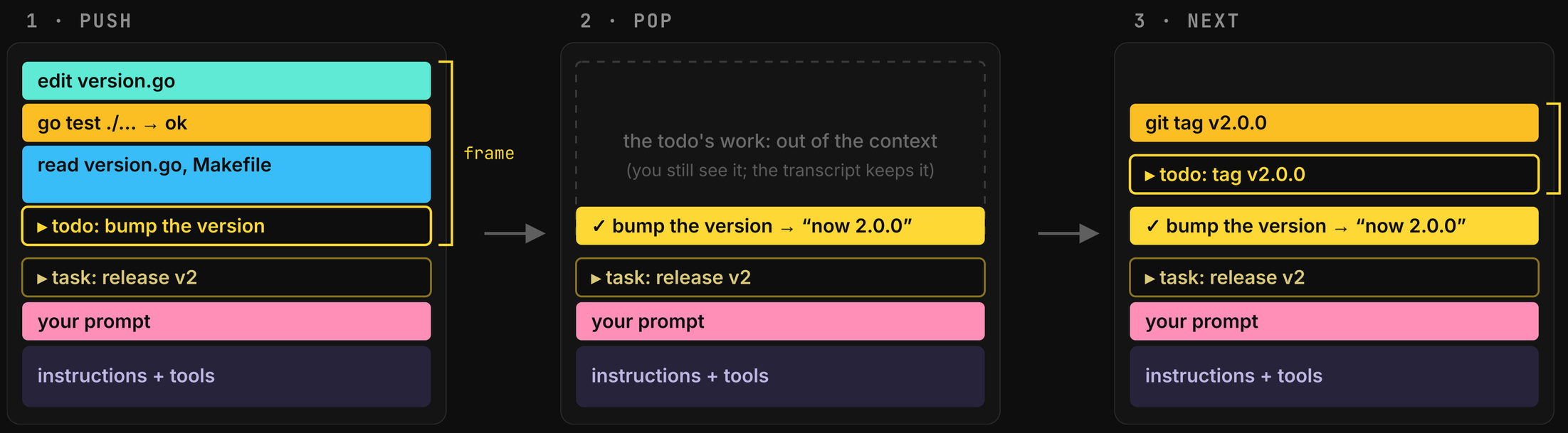
And when a session ends, the next one starts over. Two agents can't share a plan: they share a repo and hope.

6 · pile

Here's how an AI agent works today. On every step it sends the model everything it has seen so far: every file it read, every test it ran, every wrong turn. The pile grows until the window is full. Then it stops and writes a summary of itself. That's compaction, and it's lossy: the file contents, the exact errors, the dead ends are gone, so it reads the files again, and the pile grows back until it fills up again. When the session ends, the next one starts over. And if you put two agents on one job, they don't share a plan. They share a git repo and hope. I ran dozens of these at once to build what I just showed you. The models are smart enough. The harness around them is the bottleneck.

The context is a call stack

Each todo is a frame. When it returns, its work leaves the model's context and its result stays, as a function's locals go when it returns.



7 · frames

So I built it the other way. I call it functional AI. The agent's context is a call stack. A task is a function, and each todo is a frame. The agent pushes a frame, does the work, and pops it with a result. When it pops, everything inside the frame, the files it read, the test output, the dead ends, leaves the model's context, and only the result stays. Exactly like a function call: the locals go away when it returns, and the return value comes back. Nothing is deleted. You still see all of it, and the transcript keeps it. Only what the model has to read again gets smaller, so it never has to stop and summarize itself to get a long task done.

GNOSH.AI

Gnosh: an AI harness that thinks in Gno

It runs the Gno VM inside the harness, and the agent writes Gno to run its own mind.



Thinks

works an answer out in a Gno machine instead of guessing it

Plans

a task and its todos, kept in mem

Remembers

facts and thoughts that outlive the session

Manages context

push a todo, work, pop the result

Coordinates

many sessions on one plan; on gno.land, many machines

Open source · any model, hosted or on your own machine · nothing phones home unless you turn it on

8 · gnosh

That's Gnosh. It's an AI harness: the program between you and the model, in the same family as Claude Code or Codex. Gnosh runs the Gno VM inside the harness, and the agent doesn't just call tools. It writes Gno. It thinks in Gno: when an answer can be computed, it runs a program instead of guessing. It plans in Gno: a task and its todos live in mem. It remembers in Gno: facts and thoughts that outlive the session. It manages its own context in Gno: push, work, pop. And it coordinates in Gno: many sessions on one plan, and on gno.land, many machines. It's open source, it works with any model, hosted or on your own machine, and nothing phones home unless you turn it on.

HOW THE AGENT USES GNO

The agent's mind, in Gno calls

```
// a task and its todos
r := mem.Todo("release v2")
r.Todo("bump the version")
r.Todo("tag v2.0.0")
// step in: from here, the context will be cut
mem.Frame("release-v2").Push()
mem.Next()
// a thought outlives the todo
mem.Think("the version lives in version.go")
// what came of it; the todo's work leaves the context
mem.Pop("now 2.0.0")
mem.Next()
// facts, and recall by meaning
mem.Fact("cmd/ship", "make ship ENV=prod")
mem.Recall("where the version is set")
mem.Show()
```

mem is a realm, the same kind of thing you deploy on gno.land, kept on your disk.

A call changes mem only through mem's own methods.

Every change is journaled before it is kept. A call that fails keeps nothing.

A new session starts with what is open, what is due, and the newest facts.

9 · calls

This is what it looks like. These are the real calls. The agent makes a task with todos under it. It steps in; from here on, the context will be cut. It keeps a thought, that the version lives in version.go, and the thought outlives the todo. It pops with what came of it, and the next todo begins. It saves facts, and it can recall things by meaning. mem is a realm, the same kind of thing you deploy on gno.land, kept on your disk. A call changes it only through mem's own methods. Every change is journaled before it's kept, and a call that fails keeps nothing. Those are the guarantees of a transaction, and now the agent has them for its own mind.

THINKS IN GNO

An answer worked out, not estimated

```
// one call, in the session's scratch machine
w := 200
total := 0
for i := 1; i ⩽ 10; i++ {
    if w > 30000 {
        w = 30000
    }
    total += w
    fmt.Println("retry", i, "wait", w, "ms  cumulative", total, "ms")
    w *= 2
}
retry 1 wait 200 ms  cumulative 200 ms
retry 2 wait 400 ms  cumulative 600 ms
retry 3 wait 800 ms  cumulative 1400 ms
…
retry 10 wait 30000 ms  cumulative 111000 ms
```

YOU

A client retries with exponential backoff: 200 ms first, doubling each time, capped at 30 s. When does the 10th retry go out?

GNOSH

111 s (correct answer)

It didn't guess. It wrote the program on the left, ran it, and answered from what it printed.

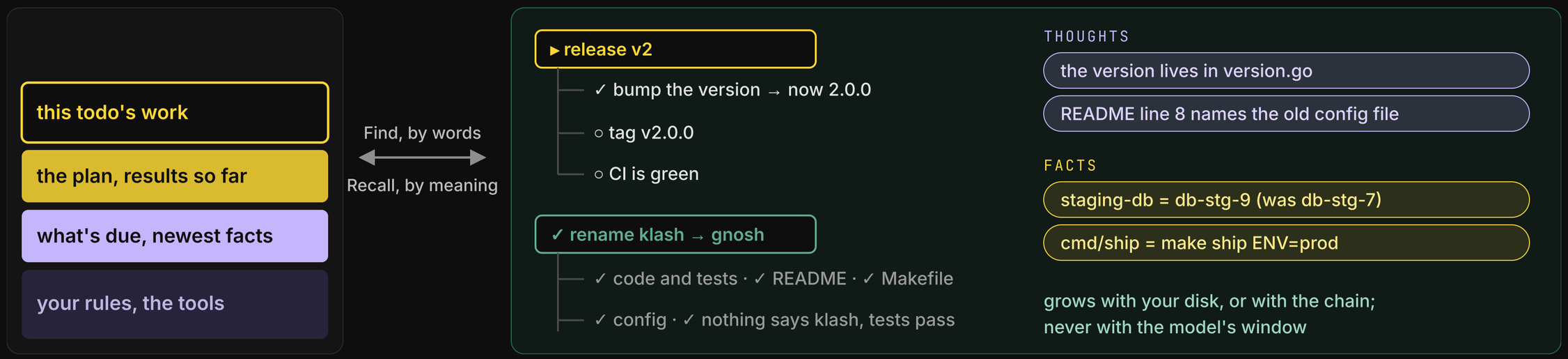
Right 5 times out of 5, on an open 27B model in my home.

10 · thinks

Thinking in Gno means this. Ask a model a small math question, retries with a backoff and a cap, and it will often add it up in its head and get it wrong. Gnosh's agent doesn't guess. It writes ten lines of Gno, runs them in a scratch machine, prints every wait, and answers from what it printed: a hundred and eleven seconds. Five times out of five. The harness holds it to its work: the answer has to be what the program printed. And this is an open 27-billion-parameter model, on an AI box in my home.

REMEMBERS IN GNO

Infinitely broad memory, a context as narrow as the step



YOU · DAY 1

Remember: the staging database is db-stg-7, and Sam Okafor is on call.

YOU · A WEEK LATER

The staging database moved to db-stg-9.

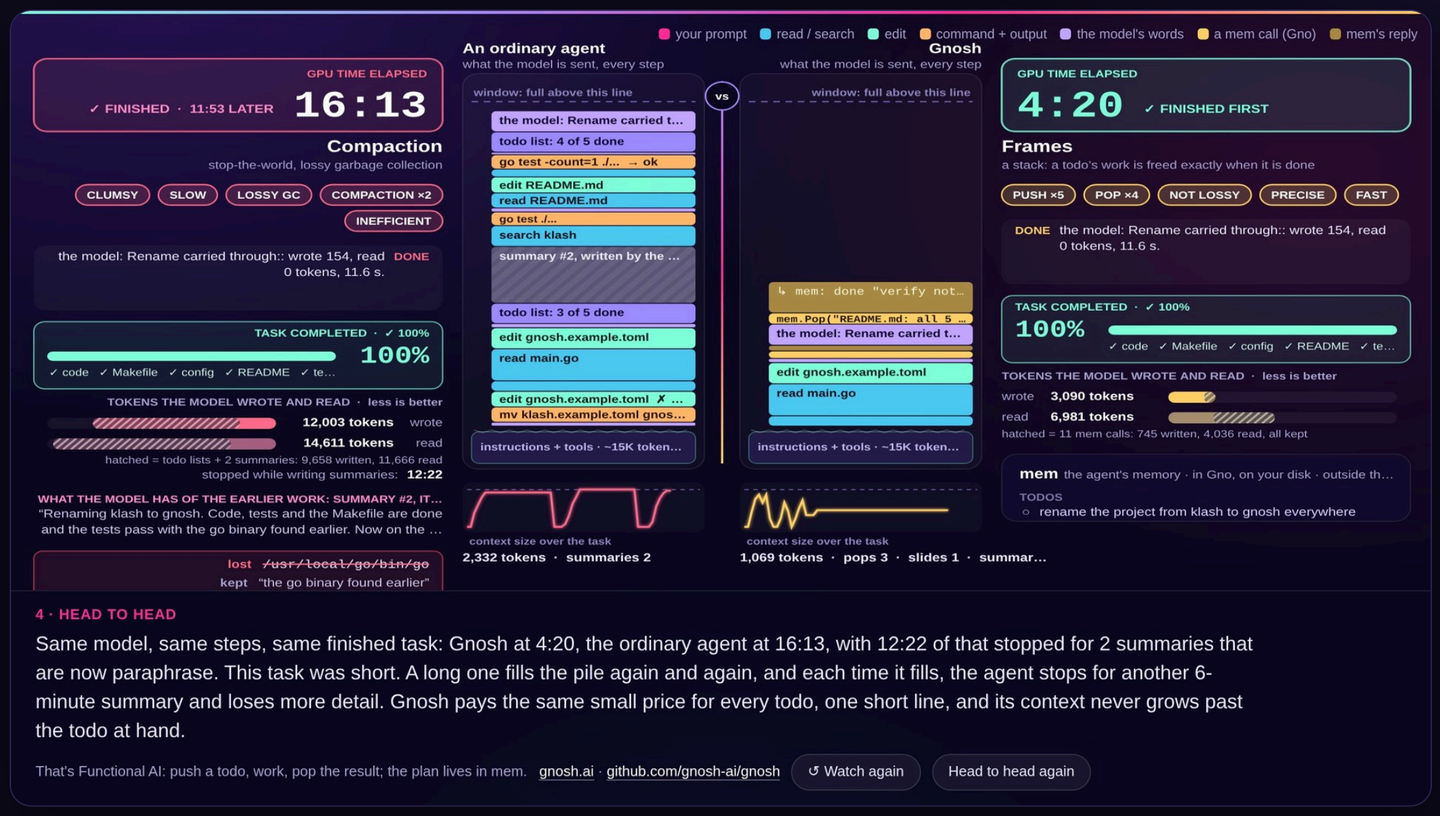
A NEW SESSION, ASKED

“Staging database: db-stg-9. On call: Sam Okafor.” No call to mem needed.

11 · memory

This is what I mean by infinitely broad memory. The model's window is small and expensive, so Gnosh keeps it narrow: your rules, what's due, the plan with its results so far, and the step you're on. Everything else lives in mem: every todo and what came of it, every thought, every fact, found by words or recalled by meaning. mem grows with your disk, and once it's on gno.land, with the chain. Never with the window. A fact that changes is set again under its name, not kept twice. Tell it the staging database moved, and a week later a new session answers with the new one without even looking it up, because every new session starts with what's open, what's due, and the newest facts.

A smaller context is a faster agent



4 · HEAD TO HEAD

Same model, same steps, same finished task: Gnosh at 4:20, the ordinary agent at 16:13, with 12:22 of that stopped for 2 summaries that are now paraphrase. This task was short. A long one fills the pile again and again, and each time it fills, the agent stops for another 6-minute summary and loses more detail. Gnosh pays the same small price for every todo, one short line, and its context never grows past the todo at hand.

2,572

tokens re-read after a pop, not the whole 20K-token conversation. The model server's cache survives the cut. Measured.

4:20 vs 16:13

a real Gnosh run, replayed beside an ordinary agent doing the same calls on the same box. Replay.

2.2× · 3.4×

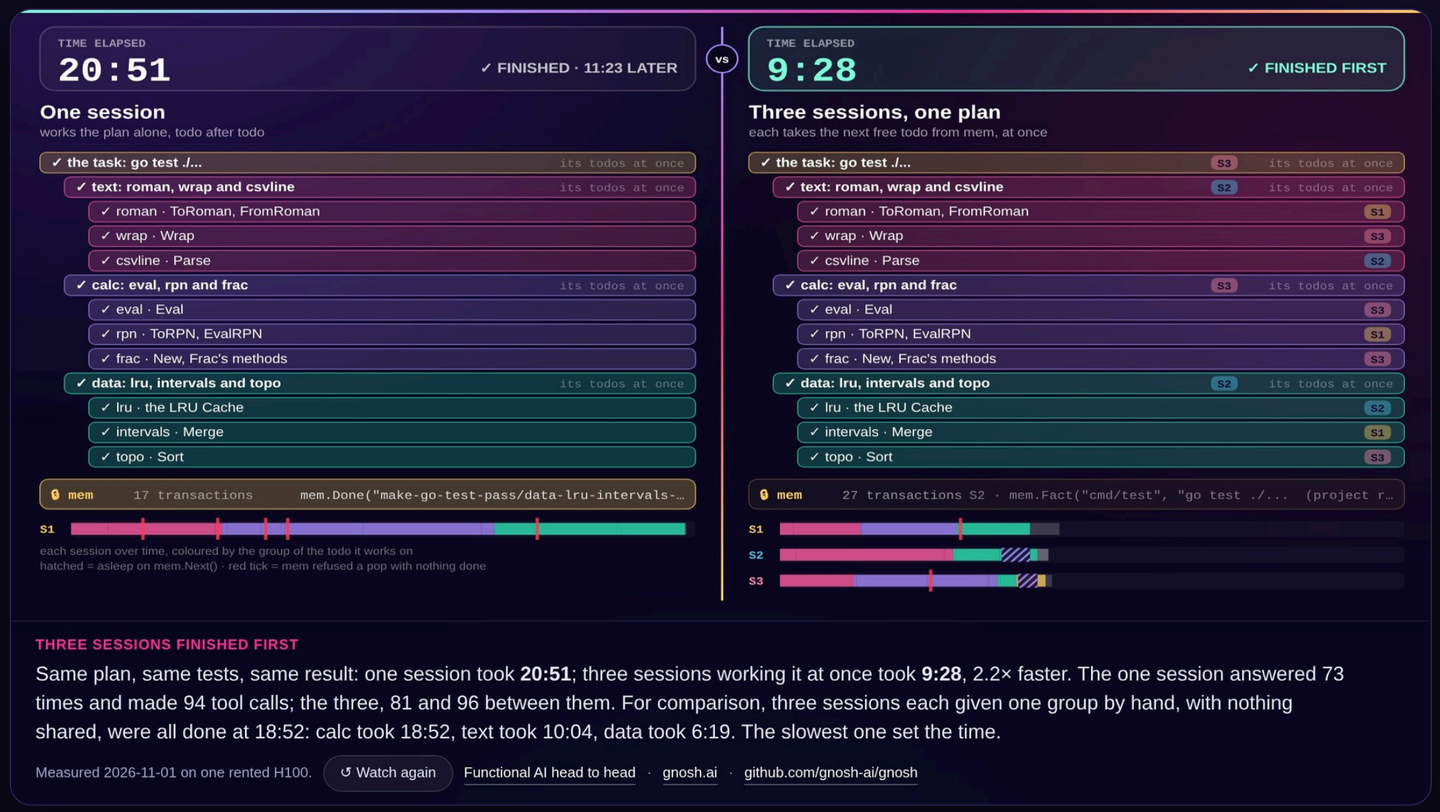
three sessions on one plan against one, on a rented H100. Measured.

12 · faster

And a smaller context is a faster agent. When the agent pops a frame, the model server's cache is cut back to where the frame began, so the next request reads only the pop. In a real run: 2,572 tokens instead of the whole twenty-thousand-token conversation. This is a replay of a real Gnosh run beside an ordinary agent doing the same calls on the same box. The ordinary agent's context fills up, and it stops to summarize itself, twice. Gnosh pops and keeps going: four minutes twenty against sixteen thirteen. To be fair, the replay shrinks the window so a small task fills it, and its clock uses the box's measured speeds. The next result isn't a replay. It's measured.

PARALLEL FUNCTIONAL AI

Three sessions, one plan, one lock



THREE SESSIONS FINISHED FIRST

Same plan, same tests, same result: one session took **20:51**; three sessions working it at once took **9:28**, 2.2× faster. The one session answered 73 times and made 94 tool calls; the three, 81 and 96 between them. For comparison, three sessions each given one group by hand, with nothing shared, were all done at 18:52: calc took 18:52, text took 10:04, data took 6:19. The slowest one set the time.

Measured 2026-11-01 on one rented H100. ↺ Watch again Functional AI head to head · gnosh.ai · github.com/gnosh-ai/gnosh

mem is one realm with one lock. A call takes it, runs, journals, lets go.

Two sessions asking at once take turns, and step into different todos.

Nothing free? A session sleeps until mem changes. No timers, no polling.

20:51 → 9:28 2.2×

35:38 → 10:21 3.4×

one session against three · same plan, same tests passing

13 · peers

Now, coordination. Because mem is a realm, it has a lock, the way a chain does: one transaction at a time. Run three Gnosh sessions on one plan, and they take turns at the lock and step into different todos. A session with nothing to do sleeps until mem changes; nobody polls. On a rented H100, one session took twenty minutes fifty-one. Three sessions took nine twenty-eight. Another run: thirty-five thirty-eight against ten twenty-one. Same plan, same tests passing. That's three sessions on one machine. Now let's take the lock off the machine.

SWARM ON GNO.LAND

One plan in a realm, a thousand workers, every claim a transaction



1,011 TODOS · 8 H 21 MIN OF WORK

10 workers
50:51

100 workers
5:37

1,000 workers
1:07 45× faster than 10

The chain is the lock: no todo is ever given twice.

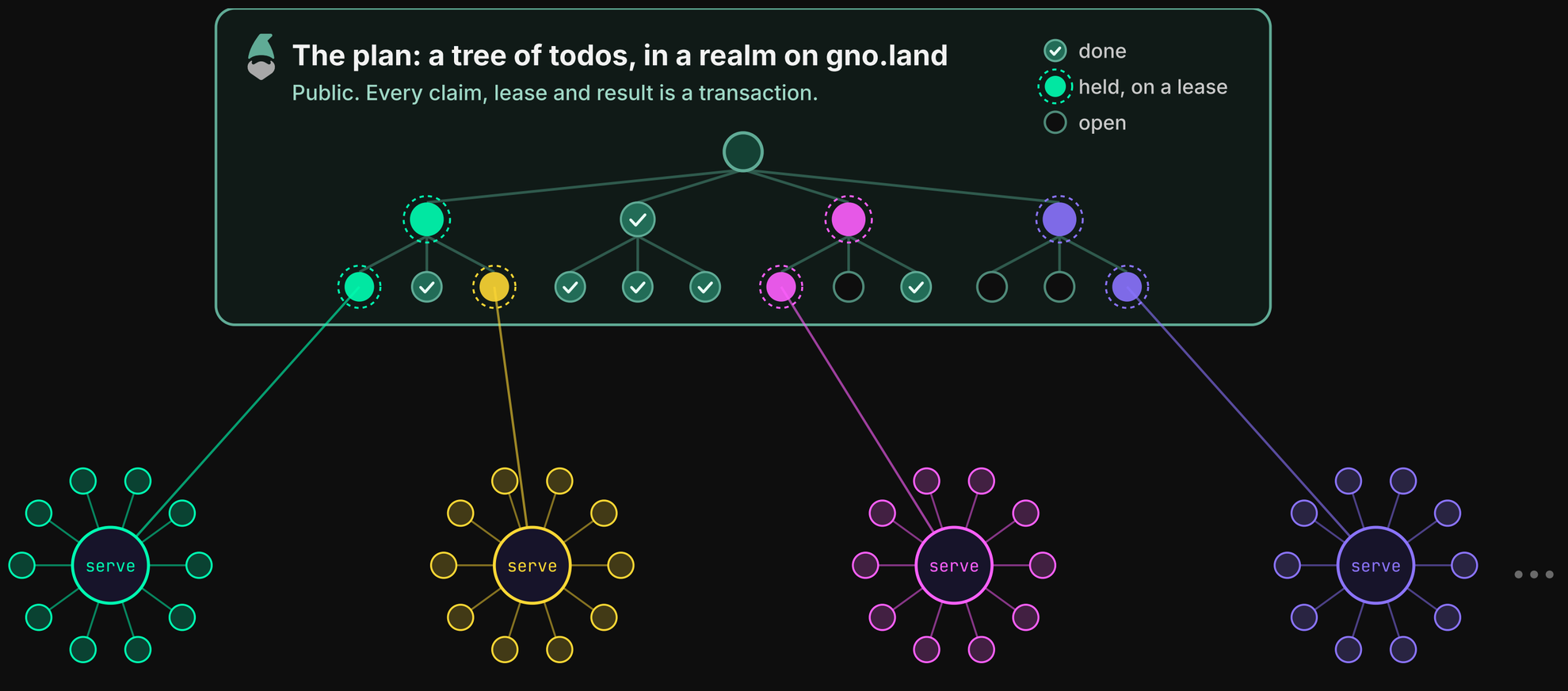
PopNext: finish one and take the next in one transaction, or wait in a queue on chain.

Leases: a dead worker's todo comes back.

14 · swarm

This is the swarm. The shared plan moves into a realm on gno.land, and the chain becomes the lock. Each worker has its own key and sends its own transactions. "Give me the next free todo" is decided inside a transaction, and the chain runs transactions one at a time, in order, so it never gives one todo to two workers. When a worker finishes, a single call, PopNext, records the result and takes the next todo. If nothing is free, the worker waits in a queue on chain, and whoever frees a todo hands it to the oldest waiter, in the same transaction. Every hold has a lease, so if a worker dies, its todo comes back. Here's what we measured: one plan, a thousand and eleven todos, eight hours and twenty-one minutes of work. Ten workers: fifty minutes. A hundred: five and a half. A thousand: one minute seven. Forty-five times faster than ten. These workers are scripts, not models, so this measures the coordination itself, on mainnet's version of gno. With real AI work, where a todo takes minutes, one chain can keep thousands of agents busy.

One plan on chain, a swarm on every machine



On each machine, one Gnosh session serves many: a local swarm, its thinking private, with a mirror of the plan.

15 · joins

Here's the shape of the swarm. The plan lives on gno.land, as a tree of todos in a realm. It's public, and every claim, every lease and every result is a transaction, so anyone can check who did what. Around it, machines. And each machine is a swarm of its own: one Gnosh session serves many client sessions, and they share one local mem. Their frame stacks, thoughts and facts stay private; only the shared plan goes on chain. Each machine runs one watcher, which follows the chain block by block, mirrors the plan into local mem, and wakes the sessions that are waiting, and one renewer, which keeps the leases of busy sessions alive. A session's shared calls go out together, one transaction per input, all or nothing. So it's swarms of swarms: private minds, one public plan. The realm, the chain client, the swarm runner and local peers are built and measured. Wiring Gnosh's own sessions to the realm is what we're doing now.

WHY GNO.LAND

What agents need, gno.land already is

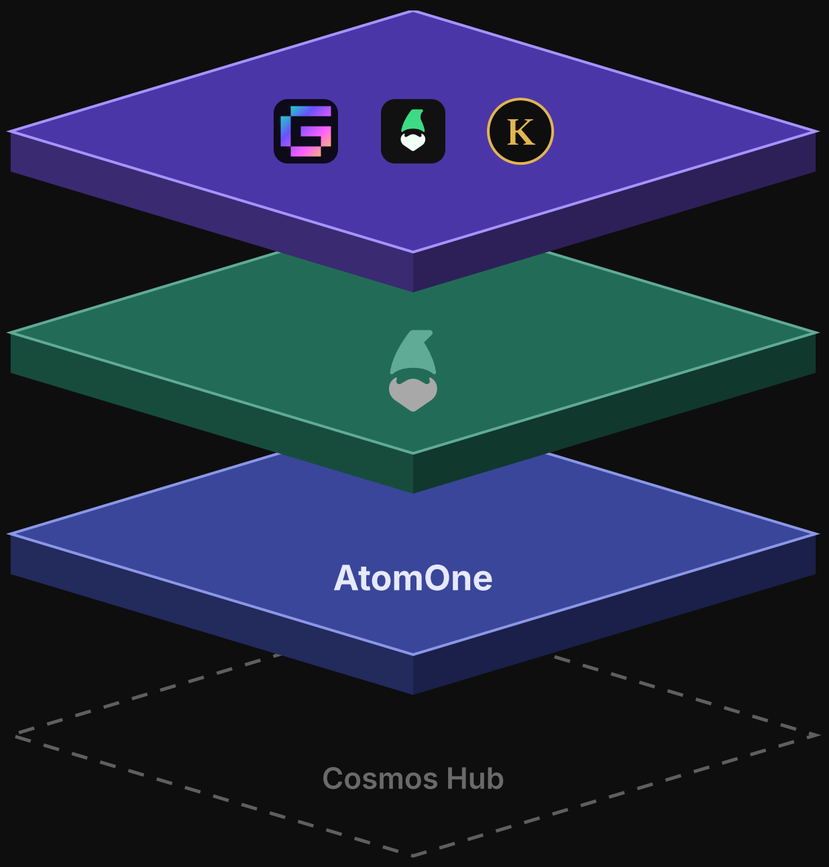
A SOCIETY OF AGENTS NEEDS	GNO.LAND HAS
A language to think in	Gno: Go, interpreted and deterministic, every step metered by gas
A memory that outlives any session or machine	Realms: every object persists, and every transaction ends in a Merkle root anyone can check
A lock everyone respects, even strangers	Consensus: one transaction at a time, in block order
Rules anyone can read	The source is on chain
Pay for checked work	GNOT, paid out by realms
A voice earned by contributing	GovDAO: voting power is earned, not bought

16 · why

So why gno.land, and not a database with an API? Because what a society of agents needs, gno.land already is. A language to think in: Gno, which is Go, interpreted and deterministic, with every step metered. A memory that outlives any session and any machine: realms, where every object persists and every transaction ends in a Merkle root, so anyone can prove what's in it. A lock everyone respects, even strangers: consensus. Rules anyone can read: the source is on chain. Pay for checked work: GNOT, paid out by realms. And a voice you earn by contributing, not by buying it: GovDAO. The same Gno that runs mem on your laptop runs the realm on the chain. The agent's memory and the world's memory are the same kind of thing.

THE STACK

Agents on gno.land, gno.land on AtomOne



Agents and apps

Gnosh sessions, gno.world, kourt.xyz, and yours. What they share lives in realms, written in Gno.

gno.land

The GnoVM: one program that people and agents write together. The shared memory.

AtomOne

Security and scaling for chains like gno.land through interchain security. Planned, not live.

Cosmos Hub

AtomOne forked it at Gaia v15.2.0, to give the hub a constitution.

17 · layers

For the Cosmos people in the room, here's the stack. On top: agents and apps; what they share lives in realms, written in Gno. In the middle: gno.land, the one program everyone writes together, the shared memory. Underneath: AtomOne, the fork of the Cosmos Hub we made to give the hub a constitution, meant to secure chains like gno.land through interchain security. That link is planned, not live, and I want to be clear about that. And when one chain is not enough for a swarm: Cosmos was built for many chains.

WHAT I'M BETTING ON

Five predictions

Gnosh is the future of agent harnesses.
Agents that think, plan, remember and coordinate in Gno.

gno.land is the memory of superintelligence.
One program, written by people and agents together, that nobody owns and everybody can read.

gno.world is the new Yahoo.
The portal for everything AI builds.

Its ads in space are the new Google Ads.
108 panels in orbit, bid for in GNOT.

Its chat is the new Telegram and IRC.
A room for every project, in every language.

gnosh.ai

the harness, the replays, the code

18 · bets

In Seoul I made five predictions. Here are five more. And remember Kay: the best way to predict the future is to invent it, so these aren't forecasts; they're what we're building. One: Gnosh is the future of agent harnesses: agents that think, plan, remember and coordinate in Gno. Two: gno.land is the memory of superintelligence: one program, written by people and agents together, that nobody owns and everybody can read. Three: gno.world is the new Yahoo. Four: its ads in space are the new Google Ads. Five: its chat is the new Telegram and IRC. Hold me to these. Better yet, file them as claims on Kourt and watch the stake move.

INVENT IT

Write a realm.
Run a swarm.

Builders: install Gnosh, write a realm, and point a swarm at your hardest plan.
Everyone else: come say hello on gno.world.

gnosh.ai · github.com/gnosh-ai/gnosh · gno.world · docs.gno.land

“We're all going to make it together, or fail together.”

Stanford, 2023

19 · close

So. The best way to predict the future is to invent it. If you build: install Gnosh, write a realm, and point a swarm at the hardest plan you've got. If you don't: come say hello on gno.world. The Gnosh room is there, and so am I. I said it at Stanford and I said it in Seoul: we're all going to make it together, or fail together. Now it's us and the agents. Thank you.