I Gave My OpenClaw Agents One Shared Brain. They Stopped Being Stupid.

ericosiu
@ericosiu · Feb 10 ·

Follow

OpenClaw is great. People are all building their own 'mission control' dashboards in hopes of having their agents do the work of entire armies.

People are talking about their 10-20 deep agent squads. An agent for every department!

Cool.

Do they talk to each other?

I had multiple agents running my marketing ops:

• Oracle - SEO intelligence. Pulls Google Search Console + GA4 data, scores keyword opportunities, creates WordPress drafts.

• Flash - Content engine. Scouts trends, repurposes my podcasts into X/LinkedIn/short-form, writes in my voice.

• Alfred - Strategic ops. Morning digest, deal pipeline, coordinates the system. My chief of staff.

Each one runs on scheduled crons. 30 automated jobs across the three of them. Oracle fires at 7 AM, Flash at 6 AM, Alfred synthesizes at 8:45 AM. Every day, no human in the loop.

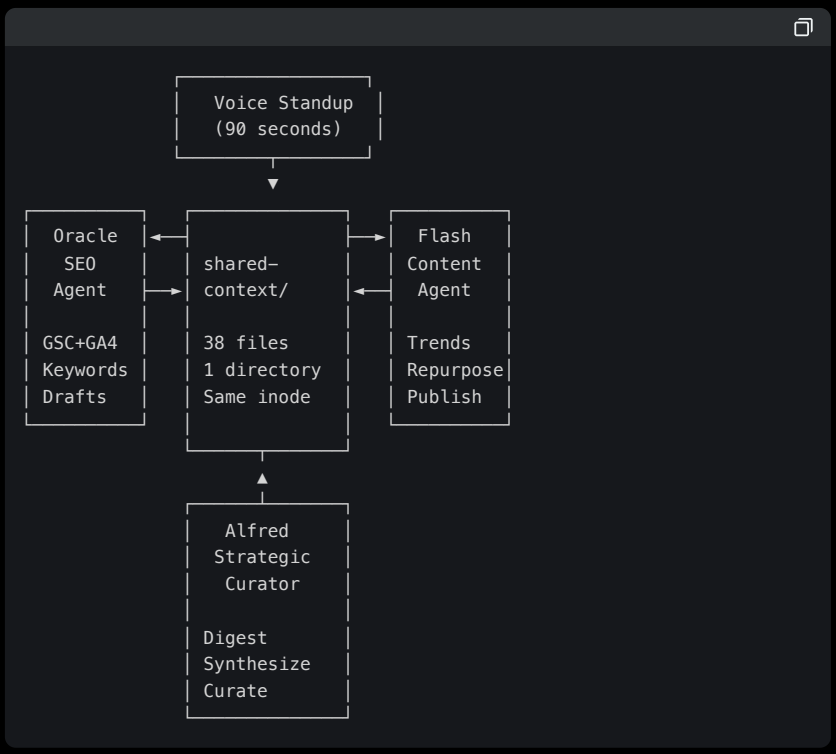
They were each individually excellent. **And collectively braindead.**

Oracle would find a killer keyword — "reddit marketing agency," converting at 17.5% with 17 free consultation submissions per month. That insight sat in an SEO report. Flash, running 30 minutes later, would create content about

something completely unrelated. Alfred would recommend a strategy neither agent was supporting.

Three smart employees in three soundproof rooms.

You would think that the fix is more agents, but nope. It's one shared directory.



Here's the entire architecture:

```
shared-context/
├── priorities.md          ← Current priority stack (from voice standups)
├── agent-outputs/         ← Each agent drops work here. Others read it.
├── feedback/              ← My approvals + rejections flow to ALL agents
├── kpis/                  ← Live metrics every agent can reference
├── calendar/              ← Today's meetings, relevant contacts
├── content-calendar/      ← What's planned, published, gaps
└── roundtable/            ← Daily cross-agent synthesis
```

One directory. 38 files. Symlinked into every agent's workspace - same file, same disk address. When I update priorities, all 3 agents read from the same source of truth in their next run.

The technical implementation took 20 minutes. Three commands:

```
ln -s /shared-context/ /oracle/workspace/shared-context
ln -s /shared-context/ /flash/workspace/shared-context
ln -s /shared-context/ /alfred/workspace/shared-context
```

That's it. That's the entire "multi-agent orchestration framework."

No LangChain. No custom protocol. Symlinks and markdown files.

```
$ ls -i shared-context/priorities.md

3437178  /alfred/shared-context/priorities.md
3437178  /oracle/shared-context/priorities.md
3437178  /flash/shared-context/priorities.md

Same inode. Same file. Three agents.
```

Here's what changed immediately.

Oracle's 7 AM run finds that our Reddit marketing agency page ranks #8, with a 17.5% conversion rate — the highest on the entire site. It writes this to shared-context/agent-outputs/oracle-seo-2026-02-09.md.

Flash's next run reads Oracle's output. Sees the Reddit data. Creates Reddit marketing thought leadership content for X and LinkedIn — driving social traffic to the exact page Oracle is optimizing for organic search.

Before the shared brain: two agents, two unrelated outputs.

After: coordinated organic + social strategy.

Zero human coordination.

```
Oracle (7:00 AM)
  | Finds: "reddit marketing agency" — rank #8, 17.5% conversion
  |
  ▼
shared-context/agent-outputs/oracle-seo-2026-02-09.md
  |
  ▼
Flash (7:30 AM)
  | Reads Oracle's output
  | Creates: Reddit marketing content for X + LinkedIn
  |
  ▼
RESULT: Organic SEO + social traffic → same landing page
        Zero human coordination
```

The feedback loop is where it gets powerful.

Every agent outputs recommendations with approve/reject buttons. When I reject something — "not high intent" — that rejection goes to shared-context/feedback/. Every agent reads it.

16 feedback entries in 2 weeks. Oracle learned to stop recommending tool-query keywords. Flash learned which content angles I approve. One decision teaches the whole team.

```
shared-context/feedback/
├── feedback-2026-W06.json    ← 12 decisions
├── feedback-2026-W07.json    ← 4 decisions (and counting)
```

When I send a 90-second voice note with my morning priorities, Alfred transcribes it, updates shared-context/priorities.md, and every agent's next run is automatically weighted toward what I said matters.

One voice note. Three agents realign. No meetings, no Slack threads, no "let me loop in the other team."

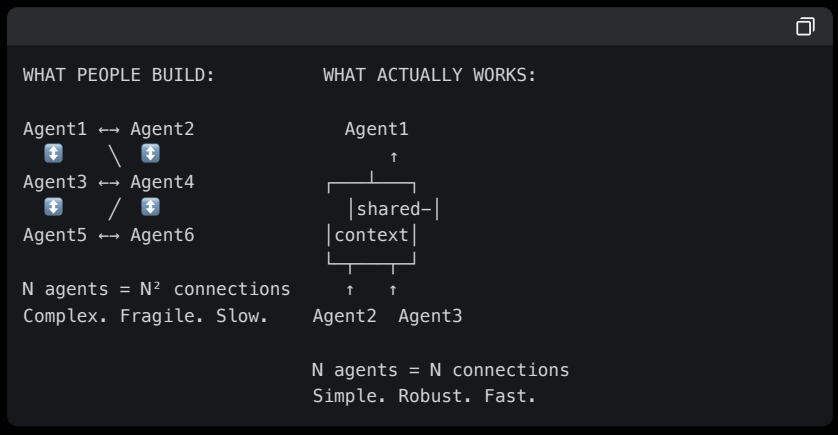
What most people get wrong.

They build agent-to-agent messaging. Complex routing layers. Handoff protocols. State machines.

That's overengineering at 3 agents and complete chaos at 30.

The right model is how actual companies work: **shared wiki + specialized departments.** Your agents don't need to "talk" to each other. They need to read from the same page.

Think about it: Notion doesn't have your marketing team sending API calls to your sales team. They read the same doc. Same principle.



How this scales to 10 agents.

Add an agent. Symlink shared-context/. Update its instructions: "Read shared-context/ before every action." Done.

The structure handles it because:

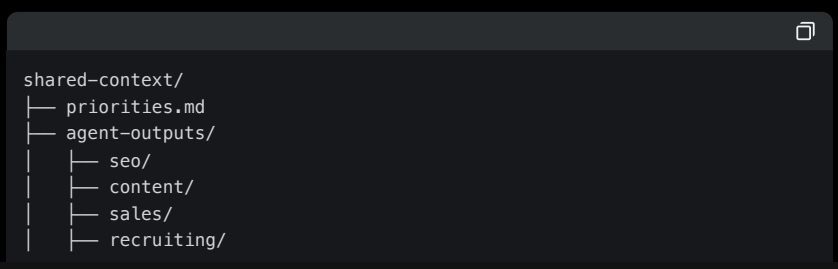
• Priorities cascade — new agent reads the same priorities.md

• Feedback compounds — every rejection teaches all agents, including the new one

• Cross-signals emerge — more agents reading/writing = more overlapping patterns detected

• One curator (Alfred) keeps it clean so it doesn't become a junk drawer

At 10 agents, you add subdirectories:



```
│   └── finance/
├── feedback/
└── kpis/
```

Still files. Still symlinks. Still no framework.

How this scales to 100.

At 100 agents, the shared-context directory becomes a knowledge graph. You add:

• **Read permissions** — not every agent needs every file. Sales agents read kpis/ and feedback/, not content-calendar/.

• **Write queues** — agents submit to a staging area, the curator validates before merging to shared-context

• **Domain namespacing** — shared-context/seo/, shared-context/sales/, shared-context/product/ with cross-domain signals surfaced by a dedicated synthesis agent

• **Versioning** — git-backed shared-context so you can track what changed, when, and why

The architecture doesn't fundamentally change. It's still "shared files, specialized readers." You're just adding organization the same way a company adds departments.

But start with 3.

Don't build for 100 agents on day one. That's the same trap as "we need a microservices architecture" when you have 2 developers.

Here's the playbook:

1. Week 1: Build 1 agent that does something useful. Get it on a cron.

2. Week 2: Build a second agent in a different domain. Create shared-context/. Symlink it.

3. Week 3: Add the feedback loop. Approve/reject buttons. Both agents learn from your decisions.

4. Week 4: Add a synthesis agent that reads all outputs and finds cross-signals.

By week 4, you have what took us months to figure out. One brain, many hands.

The secret to multi-agent systems isn't more agents. It's less isolation.

Stop building silos. Build a shared brain.

For more like this, level up your AI and marketing with 14,000+ marketers and founders in my Leveling Up newsletter here for free:

https://levelingup.beehiiv.com/subscribe

Want to publish your own Article?

Upgrade to Premium

ericosiu
@ericosiu

Follow

Founder ad agency @singlegrain, Investor. Member: @YPO Beverly Hills
Podcaster: Marketing School, Leveling Up