noded

WHITE PAPER · FOR CS & AM LEADERS

The Last Silo

You cannot rally a company from inside a silo.

Why the customer success platform category was a strategic mis-step, what the job of a CSM or AM actually is, and what to build instead.

The Tuesday nobody logged

A CSM finishes a call with a strategic account. She does everything right. The recorder fires, the notes land in the CS platform, the health score ticks from green to yellow, a CTA is created, and a task appears in her queue. By every measure the platform was sold on, this worked.

Then she spends the rest of Tuesday doing the actual job.

She messages the support lead, because the customer said the latency ticket is now the reason procurement has gone quiet. She pings the PM, because the integration commitment made in the last QBR isn't on any roadmap she can find. She forwards the thread to the AE, because there's an expansion signal buried in the second half of the call. She drops a note in the exec channel, because the champion who has carried this account for two years just told her she's leaving in March. She updates the shared doc that the account team actually reads. She answers three Slack messages from people who wanted to know what's going on with this customer and had no way to find out except by asking her.

None of that is in the CS platform. Some of it *could* be typed in. Almost none of it *will* be, because typing it in would take longer than the work itself and the only people who would see it are the four people already on the call.

That Tuesday is the job. The health score is a byproduct.

And here is the uncomfortable thing about the platform that captured the byproduct and missed the job: it did not fail on execution. It executed the design exactly. The design was to build the customer success team an application. Which means that the most complete picture of the customer in the company now lives in a system that exactly one department opens.

That is a **silo**. You already know the word — it is the thing every system you have bought in the last decade and a half claimed to be solving. Your CS platform solved it for CS. It is **the last silo**: a complete picture of the customer, visible to one department, kept alive by that department's typing.

This paper is about why that shape was wrong from the beginning, why the agent era has made it actively expensive, and what the right shape looks like.

One job, two titles

Before going further, a note on vocabulary, because half the people this paper is written for will otherwise assume it is about somebody else.

Customer success manager and account manager are different titles with different compensation models, different reporting lines and, in a lot of companies, a quiet and unresolved argument about which one owns the renewal. Those differences are real and this paper is not going to pretend otherwise.

But they are differences of *incentive*, not of *shape*. Put a CSM's week and an AM's week side by side and the structure is the same one:

- **One person holds the whole story of a named account over time.** Not a deal, not a ticket, not a cohort — the account, continuously, including the parts that predate them.
- **Their output is produced by other departments.** Neither of them fixes the bug, ships the feature, approves the credit or writes the contract. Everything they are measured on is delivered by somebody who does not report to them.
- **So the job is persuasion and routing.** Both spend their week getting seven or eight other functions to act on something the customer said, and both are judged on whether the company as a whole behaved coherently.
- **And both are the only route into the account's reality.** When anyone else in the business needs to know what is actually going on with that customer, they do not run a query. They message a person.

Where AMs differ, they differ in a direction that makes this argument *stronger*, not weaker. AMs typically carry an explicit number, are more often measured on expansion than on retention alone, and — in most organisations — have less dedicated tooling than their CS counterparts, because the platform budget went to the CS org. The AM does the same connective work with a quota attached and a thinner stack.

So: **this paper says CSMs and AMs, and it means both.** Where a sentence works for only one of them, it says so. Everything about the last silo, the three costs and the shape of the fix applies identically to anyone whose job is to hold an account and move a company.

What the job actually is

Ask a CS or AM leader to describe their team's job and you will rarely hear "maintain the health score." You will hear some version of this: *we are the people who make the rest of the company act on what the customer needs.*

That is not a workflow. It is a network position.

CSMs and AMs are the **super connectors** — the highest-degree nodes in the company's internal graph. In a normal week one of them moves information between support, engineering, product, sales, finance, legal, the exec team, and the customer's own internal factions. Nobody else in the business talks to that many functions about that many things with that much specificity. The AE talks to a customer intensely and then stops. The support engineer sees one ticket deeply. The PM sees a hundred accounts shallowly. The CSM or AM is the only person holding the whole story of one account over time, and the only person positioned to make eight other functions do something about it.

The output of that role is not a record. **The output is alignment.** The deliverable is that the support queue reprioritises, the roadmap absorbs a commitment, the AE knows to call, the CFO understands the renewal risk, and the customer experiences a company that appears to be one thing rather than nine.

Which is why Customer Success has never really been just a department. At the companies that grow well, it is an ethos — the shared reflex that the customer's reality is everyone's problem, not one team's ticket queue. That reflex is where the compounding growth of a subscription business comes from, and it is the thing the best CS and AM leaders are actually trying to build.

Now hold those two sentences next to each other:

> **The job is to rally the entire company around the customer.**
>
> **The tool bought to support that job can only be opened by one department.**

You cannot rally a company from inside a silo.

Everything that follows is a consequence of that sentence.

How the category got here honestly

It is worth being fair to the platforms, because the mis-step was reasonable at the time and pretending otherwise makes the argument weaker.

In 2013 the customer success team had no system at all. It had spreadsheets, an inbox, and a quarterly panic. Building that team a dedicated application — with renewals in a list, a health framework, playbooks, and a lifecycle model — was a genuine advance. The category deserves credit for professionalising a function that had none. Gainsight in particular did something real: it gave customer success a vocabulary, a conference, a career path, and a seat at the table.

But the shape encoded three assumptions, and every one of them has aged badly.

Assumption one: customer success is a function with its own workflow. So the product is an application with its own login, its own tab, its own adoption problem. The moment you accept that shape, the reach of the tool is bounded by the org chart, and the connective work — the part that requires seven other functions to see the same thing — falls outside the product boundary by construction. It also explains why AMs were largely left out: they sit on the other side of an org line, so they were outside the boundary from day one.

Assumption two: the record can be maintained by hand. So the product is fed by CSMs typing. This is the assumption that generates the entire hidden cost structure of the category — implementations that commonly run twelve to twenty-four weeks with professional services, a dedicated or fractional admin to keep connectors, objects and playbooks alive, and hours per person per week spent logging activity and maintaining score inputs. It also generates the failure mode every practitioner recognises: the inputs depend on humans remembering, so the scores drift from reality, so people quietly stop trusting them, so the team drifts back to spreadsheets and Slack while the dashboards keep rendering.

Assumption three: the relationships that matter are the ones somebody typed into a field. They are not. The dependency between a support case and a renewal, the fact that a champion went quiet because of a specific unresolved bug, the roadmap commitment made verbally in a QBR — these get declared constantly, in calls and threads and forwarded emails and doc comments, and almost never in a CRM field, because there is no field shaped like *blocks* or *same root cause as* or *this is why she stopped replying*. We have written about that class of missing relationship in detail elsewhere.[1]

None of these were stupid. They were the only assumptions available before the action layer was readable and before an agent could do the reading. But they

compose into a single architectural fact: **the platform's picture is one department's typed derivative of a story that happened somewhere else.**

The three costs of the last silo

1. The typing tax — the one you can already put on a spreadsheet

This is the cost everyone knows about, so we will be brief. The license is one of four line items, and it is the one people quote. Underneath it sit the implementation, the admin coverage, and the hours per person per week that go into feeding the thing. Our own true-cost calculator sums all four honestly and lets you put your own numbers in; for most teams the license turns out to be less than half the bill.[2]

The important part is not the money. It is that **the typing tax and the trust problem are the same problem.** A record maintained by hand is a record that is only as current as the last person who remembered. Everyone in the building knows this, which is why the health score is discussed as a ritual rather than a fact.

2. The asking tax — the one nobody measures

This one is larger, and it never appears in a business case, because there is no invoice for it.

Every time someone outside the CS or AM org needs to know what is happening with an account, they cannot look. They ask the person who owns it.

The AE asks before a call. The support lead asks before an escalation. The PM asks before writing the quarter's roadmap. The CFO asks before the forecast meeting. The CEO asks in the hallway. A new AM inherits an account and asks for a brain dump. Every one of those is a person who could not see, so a person who could see had to stop and explain.

Now notice what that does to the super connector. Their scarce asset is *judgment about the account*. The asking tax spends that asset on *retrieval*. The most connected people in the company spend their week being a lookup service for the story, instead of doing the thing only they can do — deciding what the company should do about it and making it happen.

And notice the second-order effect, which is the one CS and AM leaders feel in their budget conversations. When the only route to customer context runs through a person, the function starts to look like overhead. It gets described as reactive. It gets

measured on tickets and touchpoints. The scarcity of the context is exactly what makes the department look like a cost centre, and the tool that produced that scarcity was sold as the thing that would fix it.

The last silo does not just hide the customer from the company. **It hides the value of the account team from the company.**

3. The AI tax — the one that just became urgent

For most of the category's life the last silo was a chronic condition. It became an acute one the moment your company started deploying agents.

Here is the shape of the problem. A CS platform's data is a *derivative*: scores, CTAs, lifecycle stages, timeline entries — structures computed from what a human typed into a form built for one department's workflow. Put an agent on top of that and you have built an agent that can see one department's typed derivative of the truth, bounded by the department's login, lagging by however long it has been since someone last remembered to log something.

That agent will answer *"what's the risk on the Contoso renewal?"* fluently and wrongly. It will summarise the fields. It will not know the renewal is blocked by a six-week-old case, because nothing in its context says those two things are related.

Meanwhile the same thing is happening in every other department. Support is standing up agents against the ticketing silo. Sales is standing up agents against the CRM silo. Product is standing up agents against the roadmap silo. Every one of those agents is confidently reasoning over a partial view of the same customer, and none of them can see what the others see. The org chart is being faithfully reproduced in the AI layer, one credentialed integration at a time.

This is why "our CS platform has added AI features" is not the reassurance the vendor thinks it is. **AI features on a silo produce a smarter silo.** The constraint was never the model's intelligence. It was the boundary of what the model can see and the lag on how it got there.

The good news is the same shape as the bad news. This is a *context layer* problem sitting underneath an agent problem — which means it gets fixed once, and every agent your company deploys afterwards inherits the fix. That is unusually good leverage, and right now it is sitting unclaimed in most companies.

It is also the strategic opening for whoever claims it. The person who fixes the customer context layer is not delivering a CS tool. They are delivering the foundation the company's entire customer-facing AI strategy has to stand on. There is no rule that says that person has to be in IT.

Measure your own silo rate

Do not take any of this on assertion. There is a number you can put on it, for one account, in about an hour.

Take one important account. Write down the ten things that are actually true about it right now — the things you would tell a new CSM or AM inheriting it on Monday. Not fields: facts. *The champion is leaving in March. The renewal is blocked by the latency ticket. They are evaluating a competitor for the reporting module. The integration commitment from the last QBR isn't on the roadmap. Procurement went quiet after the security review.*

Now, for each one, ask a single question:

> **Could someone outside the account team have found this out today, without asking the CSM or AM who owns it?**

Not "is it written down somewhere." Not "could they have found it if they knew where to look and had the login." Could a support lead, a PM, an AE, or a CFO have found this out on their own, this morning.

Count the noes.

> **Noes ÷ ten is your silo rate.**

Most teams find the majority of the list comes back as noes, and the items that score worst are consistently the most consequential ones — the reasons, the dependencies, the political situation inside the account. The facts that were typed into a field are the ones the rest of the company can see, and those are mostly the facts that matter least.

Two things to say about that number before you take it anywhere.

A high silo rate is normal. This is structural — not a hygiene failure and not an adoption failure — and it is the expected result of the architecture rather than a surprising one. Your team is not bad at documenting; they are documenting constantly, in the places where documenting costs nothing — threads, emails, docs, calls. The rate measures the shape of your architecture, not the diligence of your people.

Do not convert it into money. You will be tempted to multiply the asking tax by a loaded salary and put a number on a slide. Resist it; the number will be fictional and

the person you most need to convince will know. The silo rate is more persuasive as what it is — a count of the true things about your most important customer that your company cannot see.

One caveat on method, because you should apply the same scepticism to our instrument as to anything else in this paper: this is a deliberately qualitative exercise, not a sampled statistic. Ten hand-picked facts are not a random sample, and the categories that matter most — reasons, dependencies, internal politics — are the ones least likely to have ever been a field. It is designed to make a structural point vivid, not to produce a defensible percentage. The rigorous, sampled version of the same idea is the dark join rate, and the instrument for it is published, free and vendor-neutral.[3]

The version of this exercise that actually ends the argument is the list itself. Ten specific, named, true facts about a real account, each with a checkbox next to *"anyone outside the account team could see this."* That page is what you forward. It is much harder to argue with than a percentage.

What to build instead: a nervous system, not a platform

So what should exist instead? Not a better CS platform. Something with a different shape.

The distinction is not marketing. It is architectural, and it is the whole point.

A platform is a place you go. It has a boundary, a login, an adoption curve, and a department that owns it. Its value is bounded by how many people are willing to open it, which is why every CS platform business case eventually becomes an adoption conversation.

A nervous system is a thing that reaches everywhere. It has no destination. It does not ask you to go anywhere, because it is already in the places you are. Its value is bounded by how much of the body it is wired into.

Four properties follow from that. Treat them as a specification — the thing to build if you are building, and the thing to interrogate if you are buying.

It assembles itself from the work

If the story has to be typed in, you have rebuilt the typing tax and you will rebuild the trust problem behind it. The system has to read the **action layer** — the calls, the email threads, the Slack and Teams conversations, the shared docs and their comments, the ticket bodies — because that is where people actually declare what is true about an account. They do it there because it costs nothing: no required fields, no validation rules, no admin asking why a picklist is blank. Nobody experiences a Slack message as data entry.

Doing that well needs three signals working together. (These are the ones we built; the names are ours, the mechanics are not a secret.)

- **Interaction rank.** Every touch is an edge — a forward, a mention, a status change, a reply, even a read. Records the work keeps reaching for climb in rank and earn their way into the graph. It is PageRank applied to enterprise work, and it inherits PageRank's virtue: nobody has to flag anything, because importance is a property of the link structure.
- **Temporal decay.** Importance fades unless renewed. A record nobody has touched for a business cycle falls out of the active chain. This is what stops the graph turning into another archive, which is where most knowledge systems die — they accumulate, never forget, and within a year the signal is buried under everything that used to be true.
- **Intent filter.** A classifier over the language around a reference, not the identifier alone. *"Blocked by NOD-231"* chains the record in. *"Closing as duplicate of NOD-231"* keeps it out. Same identifier, opposite relationship.

Interrogate that last one hardest, whoever is building it. Naive extraction — regex for anything shaped like a ticket ID — produces a graph so noisy it is worse than nothing, because **a wrong edge in a customer graph costs more than a missing one.** The honest design runs heavily toward precision, and every edge should trace back to the sentence that created it. If a vendor cannot show you the sentence behind an edge, be sceptical of the edge. That includes us.

It is not another store of record

This is the test that disqualifies most things that call themselves a customer 360. If the answer to the silo is a new database that everyone has to be persuaded to populate and govern, you have bought silo number two and called it consolidation.

The graph should hold **pointers, not copies** — references, rank, and relationships. Salesforce stays the system of record. So does Zendesk, so does Jira, so does your warehouse. Records stay in the systems that own them, under those systems'

permissions and audit trails. Disconnect it tomorrow and everything is exactly where it always was.

This matters commercially as well as architecturally: it is what makes the decision reversible, and a reversible decision is a much easier decision.

It appears where people already are

The reach of the nervous system is the whole value, so the delivery surface is not a detail. If the answer to *"why did this account go quiet"* lives behind a login that only the CS team has, nothing has changed except the vendor.

So it has to be in Slack, where the support lead already is. In email. In Claude, ChatGPT and Gemini, where a growing share of your colleagues now start every question. And for agents, behind **one governed layer** — a single consolidated MCP endpoint over the whole customer stack, where you define what is exposed, who it is exposed to, and how it is described, rather than a separate credentialed integration per app per department.

That last part is what turns this from a departmental fix into a company-level asset. One context layer, every agent plugs into it, exposure is subtractive — what is not exposed does not exist as far as any agent is concerned.

It makes everyone else's systems better

The final property is the one that turns the super connector from a lookup service into a source of leverage.

Because the system is reading the work anyway, it can write the durable parts back into the systems the *other* departments live in — the champion change into the CRM, the blocking dependency onto the ticket, the next step onto the opportunity. Opt-in, attributed, reviewable; read-only until you say otherwise.

Think about what that does to your standing. Sales' CRM gets cleaner without an AE typing. Support sees the renewal context on the ticket. The data team's warehouse dimensions stop depending on whether someone remembered to log. Every one of those is a department that got something for free from a system you brought in.

That is what rallying the company around the customer looks like when it is implemented in architecture instead of asked for in a meeting.

What changes when you have one

Three things, and they are worth naming plainly because they are the actual reason to do any of this.

The company conversation. Customer context stops being something the company gets by asking your team and starts being something the company has. Your function stops being described as reactive, because the evidence of what it knows is now visible to everyone who was previously waiting on it. This is the single fastest way we have seen a CS or AM org get re-described from cost centre to connective tissue — not by arguing for it, but by making the context ambient.

The AI conversation. Every company is currently trying to work out where its customer-facing AI strategy stands up. The honest answer in most of them is "on top of whatever each department already had," which is how you get four agents with four partial views of one customer. The context layer is the missing foundation, and it is genuinely unclaimed. Walking into that conversation with the layer already running — and with a governance story about pointers, subtractive exposure and one identity per agent — is a different kind of meeting from asking for headcount.

The budget conversation. This is the one that makes the other two possible, and the argument is structural rather than a discount. The costs in the middle of this paper are all consequences of one design decision: that a human maintains the record. Remove that assumption and the implementation, the admin coverage and the data-entry hours do not get optimised — they stop existing as line items. That is why a system with this shape can cost a fraction of a platform with the other shape without being a worse product. It is doing less work, because there is less work.

Objections worth raising

"We spent eighteen months implementing Gainsight. This is not a conversation I can have." Then do not have it as a rip-and-replace conversation. Have it as an inputs conversation. The thing that has gone wrong with the platform is almost never the framework — it is that the framework is being fed by hand. Fixing the inputs makes the eighteen months look like a good decision rather than a sunk one, and it leaves the platform decision for renewal, where it belongs.

"Health scores are what my board sees. I cannot destabilise that." Keep them. A score is a model over inputs, and nothing here argues with your model. It argues with inputs that depend on whether someone remembered. The scores get more

defensible, not less, and you will be able to answer *“why is this one yellow”* with a sentence someone actually said rather than a field someone actually typed.

“Isn’t this just another tool for my team to adopt?” It is the opposite claim, and it is falsifiable, so hold whoever is offering it to the test: if your team has to go somewhere new to get value, the shape is wrong. The delivery surfaces should be the ones they are already in, and there should be no new store of record to govern, because the graph holds pointers rather than copies.

“Our AMs and CSMs don’t even use the same tools today. Won’t this make that worse?” It should collapse the problem rather than duplicating it. The whole point of reading the action layer is that it does not care which application a team was assigned — it reads the calls, threads and email where both of them already work. In practice this is the first system most companies have had where the AM and the CSM are looking at the same account story, which is worth something on its own before anyone mentions agents.

“If everyone can see the account, what is my team for?” This is the real question under a lot of the resistance, and it deserves a straight answer rather than a reassuring one. The scarcity model — where context flows through a person — feels like job security and functions as the opposite. It is precisely what makes the function look like overhead, keeps CSMs and AMs doing retrieval instead of judgment, and caps the book each of them can carry. Nobody has ever been promoted for being the only person who knew something. The connector’s value was never the lookup; it was deciding what the company should do and making eight functions do it. Visibility is how that ethos scales past the number of hours in your team’s week.

“We’re not doing much with AI yet. Is this premature?” It is the right order. Agent projects fail on the mess underneath, and fixing the layer once means every agent afterwards inherits the fix. In the meantime the graph pays for itself the boring way: a CRM that becomes current as a side effect of work happening, an account story anyone can self-serve, and prep time that collapses. The agents can come whenever the company is ready.

“Where does this fit less well?” Any system of this shape is strong when the account story flows through connectable tools — recorders, email, Slack and Teams, docs, tickets, CRM. If your customer interactions happen mostly on the phone and in person and never touch those systems, you will get less from day one. That limitation is inherent to reading the action layer; it is not a gap in one vendor’s connector list.

Build it, or buy it

So that is what you need: a system that assembles the account story from the work, keeps pointers rather than copies, shows up where your colleagues already are, and improves every other department's systems as a side effect. Not a place your team goes. A layer the company reasons on.

You can build it. Everything above is a design, not a secret, and for some organisations building is the right call — particularly those with unusual estates or hard sovereignty constraints. If that is you, take the design and build it honestly:

- Connectors across the full action layer, including email and document comments. Those two get skipped most often and hurt most to skip.
- An interaction-rank implementation over a continuously updating edge stream, which is not a batch job.
- A temporal decay model tuned to your business cycle, which you will get wrong twice before you get it right.
- An intent classifier good enough to tell *blocked by* from *duplicate of* across your domain vocabulary, with a labelled evaluation set so you can prove precision rather than assert it.
- Permission-aware resolution, so the graph never becomes a lateral disclosure path between teams.
- A cached serving layer, or your token spend will collapse the business case on its own.

That is a platform team and several quarters. Genuinely doable, and genuinely expensive.

Or you can buy it. **We built Noded because these are the problems we kept hitting, and this is the shape we concluded was the only one that solves them.** It reads the action layer rather than the storage layer, assembles a live Context Graph per account from pointers rather than copies, and exposes it in Slack, in email, in Claude, ChatGPT and Gemini, and to your own agents through one governed MCP layer. Data stays in the systems that own it. Write-backs are opt-in, attributed and reviewable. Nothing is ever trained on. SOC 2 audited.

Because there are no fields to maintain, there is no admin role to staff and no implementation project to fund — which is why it is priced like a productivity tool rather than a platform: per seat from $20/month, Growth at $1,000/month with unlimited users, and reading and asking always free.

If you already own a CS platform

Most readers of this paper do, and you do not have to choose on day one. There are two paths and they both start the same way.

Extend. Keep the platform and fix what feeds it. Your health frameworks, playbooks and muscle memory are real investments, and some of them are good. Noded becomes the layer underneath and keeps the platform current without anyone typing. This is the path that fixes the trust problem without touching the contract.

Replace. Run the post-sale motion in Noded and retire the admin rather than maintaining it.

Most teams do not decide up front. They run Noded alongside whatever they own today, and then make the platform decision at renewal with several months of evidence about where the work actually happened. That is a better way to make the decision than a bake-off, and it costs nothing to set up. The mechanics of both paths — exactly what syncs where, and what running the two side by side looks like week to week — are written up per platform rather than here: Gainsight, Planhat, ChurnZero.

Where to start

There is nothing to implement to get started. Connect your tools over a coffee, alongside whatever you run today, and look at the story it builds on your own accounts. Then decide what the platform line item is really buying. If your estate is complex, we will run the project with you rather than pretending it is a switch-on.

Run the silo-rate exercise first, though. One account, ten facts, an hour. Find out how much of your most important customer your company can actually see.

Because the job was never to keep the record. It was to rally the company around the customer — and **you cannot rally a company from inside a silo.**

Next steps: See your accounts in Noded · What your CS platform actually costs · How ingestion chaining works · Complex estate? We'll run the project with you →

Ingestion Chaining is a trademark of Noded AI.

---

1. *Dark Joins: the relationship isn't in the data, it's in the work* — the companion paper for IT and platform leaders, on relationships that are real and load-bearing in your business and exist in no schema

anywhere in your estate. getnoded.ai/post/dark-joins

2. *The True Cost of Your CS Platform* — an interactive calculator that sums license, implementation, admin coverage and the data-entry tax from your own numbers. getnoded.ai/cs-platform-tco-calculator

3. *The Dark Join Diagnostic* — a free, vendor-neutral, CC BY 4.0 instrument that samples thirty artifacts from one account and measures what share of the real relationships in your business exist in no system. Nothing leaves your browser. getnoded.ai/dark-join-diagnostic