Episode Book

EPISODE #5908

One Database Won't Fix Your Fragmented Stack

PUBLISHED	12 October 2026 at 17:16 UTC
DURATION	28:27
LISTEN	myweirdprompts.com/episode/system-of-record-vs-source-of-truth-36254/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.4
SOURCES	20 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:41 UTC

SYNOPSIS

When a small landscaping practice bolts WhatsApp, a meeting assistant, and a CRM onto its existing backend, the instinct is to consolidate everything into one database. That instinct is wrong. This episode unpacks the distinction between a system of record and a source of truth, then walks through a concrete build order — WhatsApp Business Platform first, transcription second, CRM third — showing why every field needs exactly one declared owner and why copies are fine as long as you know who decides. Along the way: the new per-phone-number billing rules, the twenty-four-hour customer service window, why templates are copy that costs money and requires permission, and how to turn meeting transcripts into structured data instead of a folder of dead files.

TRANSCRIPT



Corn

Rita's got a landscaping practice, two people, and she's worried that if she bolts WhatsApp and a meeting assistant and a CRM onto what we built her, she's going to end up with four databases that all disagree about who her clients are.



Herman

Which is the right thing to be worried about, by the way. It's the correct fear.



Corn

Daniel's prompt is the last piece of the series. She wants the onboarding automated, the meeting notes to land somewhere useful, and she wants to know how to do that without losing track of which system is actually right when two of them disagree.



Herman

And she asked it as a source-of-truth problem, which is the framing that makes it answerable. That's the lens for today.



Corn

So the fear is fragmentation.



Herman

The fear is fragmentation, and the fix is not what most people reach for. Most people hear "conflicting databases" and conclude that everything has to live in one database. One Postgres, and nothing else is allowed to know anything.



Corn

Which sounds great until you try to run a business out of it.



Herman

It falls apart immediately, because her calendar is Google Calendar and her files are Drive and her messages go out through Meta's servers. You cannot unilaterally declare those things to be your database. What you can do is decide, field by field, who wins.



Corn

So the distinction.



Herman

Two terms. A system of record, and a source of truth. A system of record is the system that is correct by definition for a given piece of data. It's not the one that's fastest or the one you like. It's the one that, when two copies disagree, is right, because it's the one entitled to decide.



Corn

And the source of truth?



Herman

The source of truth is the thing that aggregates. It pulls from several systems of record and harmonizes them into one view. The system of record is a domain owner. The source of truth is a reading surface built on top.



Corn

That's the whole trick, isn't it. Rita doesn't need one database. She needs one declared owner per field.



Herman

Exactly that. Every field has exactly one system of record. Her Postgres owns the client, the project, the task, the invoice state. Google Drive owns the actual bytes of a document. The calendar owns the meeting time. WhatsApp owns the message thread with that person.



Corn

And copies are fine.



Herman

Copies are fine, as long as you know who decides. The failure isn't duplication. The failure is a system where nobody wrote down which copy wins.



Corn

I like the way one of the pieces put it. Integration done well doesn't erase the other systems. It anchors them.



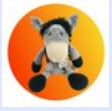
Herman

That's the sentence. You're not trying to abolish Drive. You're nailing it in place and saying what it's for.



Corn

So we've got the frame. What's the order?



Herman

Start with WhatsApp, because it's the one with the most rules attached. Then transcription, because that's where the real data design question lives. Then the CRM question, which the frame mostly answers on its own. Then workflows, then agents.



Corn

WhatsApp Business is not the app on her phone.



Herman

It is not. The consumer app is a person texting. The Business Platform is an API your backend calls, and is the one delivering. Everything about it is more formal than people expect, starting with the money.



Corn

The money gets interesting right about now.



Herman

Charges per delivered template message. Only template messages. Rates vary by category, and the categories are marketing, utility, and authentication, and the rate also varies by the recipient's country calling code. So a utility template to a client in one country is not the same price as to a client in another.



Corn

And free-form messages?



Herman

Free-form can only be sent inside an open twenty-four hour customer service window. That window opens when the person messages you. So Rita can't just text a client cold. She can only reply freely for a day after the client has spoken to her.



Corn

Twenty-four hours is not very long.



Herman

It's a support window, not a chat window. And here's the live part. As of the first of this month, those free-form replies inside the window are billable after the first thousand delivered per phone number per calendar month. Utility templates delivered inside the window went billable from the first one.



Corn

So the free part just got smaller.



Herman

The free part got smaller and the metering got per-phone-number, which is a strange unit. It's per phone number, per calendar month, which means Rita's costs depend on how many distinct clients are texting her, not on how much volume she's pushing.



Corn

A thousand a month. She's a two-person landscaping firm. She's not close.



Herman

She's not close at all, and that's worth saying plainly. The change matters to a business doing volume. For Rita it's a design constraint, not a budget problem. It's a reason to batch.



Corn

Say the batching thing.



Herman

If you're designing the onboarding sequence now, you design it so the initial welcome goes out as a pre-approved utility template, and the follow-ups stay inside the open window as free-form replies. Don't send nine separate messages when one will do. Don't fire a marketing template at someone who'd have responded to a utility one.



Corn

Hold on. Templates have to be approved, don't they.



Herman

Pre-approved, yes. You submit them through the Message Templates API and reviews them, part automated, part manual. That means the wording of Rita's welcome message is not hers alone. It gets read by a system and possibly by a person, and if it reads as promotional it gets categorized differently, and different category means different price.



Corn

So the template is copy that costs money and requires permission.



Herman

That's exactly what it is. And permission has a lower bar than people assume and a higher one than people fear. Opt-in is mandatory. She needs the person's phone number and their opt-in permission. The permission can be general, not specifically for WhatsApp, as long as she's complying with local law where the client is.



Corn

Meaning if they gave her the number and said keep me posted, that can count.



Herman

That can count, if local rules allow it. What doesn't count is buying a list.



Corn

Nobody's buying a list of landscaping clients.



Herman

You'd be surprised what people buy. The other half of this is that the platform talks back. Posts HTTP JSON to an endpoint Rita controls whenever something happens.



Corn

Incoming messages.



Herman

Incoming messages, the delivery status of what she sent, and changes to her template quality scores. There's a user preferences webhook that fires when someone stops or resumes marketing messages. That one matters more than it sounds, because it means an individual client can switch marketing off without leaving the conversation entirely.



Corn

So the backend has to honor that state.



Herman

The backend has to store it and respect it. If you're not listening to that webhook, you're one apology email away from a complaint.



Corn

Let me build Rita's onboarding out loud, because I want to see whether this is actually simple.



Herman

Go.



Corn

New client, Smith Garden. A record gets created in Postgres. That's the system of record for the client. She provisions the Drive folders for the project. Generates the standard contract from her template. Schedules the kickoff meeting. And sends the WhatsApp welcome.



Herman

And the welcome is a utility template, because it's transactional. It's confirming the engagement, not advertising a spring special.



Corn

Then the client replies to it, which opens the window, and everything after that is free-form for twenty-four hours.



Herman

Which is where the actual conversation happens. The design is: automated template opens the door, human-shaped replies walk through it, and anything that needs to happen after the window closes gets queued until either the client writes again or a scheduled utility template is the right tool.



Corn

Herm, wait. What happens if she needs to tell them something on day three and nobody's written since?



Herman

Then she's outside the window, so it has to be a template. And that's a real constraint on client experience. You cannot do the thing where you send a quick "hey, thinking about your north border, here's a photo" on day three. Not without a template and not without paying for it.



Corn

That's a strange business to be in. The platform decides when you're allowed to sound like a person.



Herman

It's the price of the reach. If you want to reach someone on WhatsApp without them having opened the door recently, you're doing outbound, and outbound is regulated and metered. That's fair enough, honestly.



Corn

Transcription next.



Herman

Suppose Rita has a Zoom call with the Smith Garden client. She gets a transcript, a summary, and a set of action items. The question is where those land.



Corn

And the answer is not "in the meeting tool."



Herman

The answer is: the transcript and the summary are artifacts. They get stored as files, linked in Postgres to the right client and the right project. The action items get extracted and become tasks in the backend, and each task points back at the meeting that produced it.



Corn

So the meeting record is the hinge.



Herman

The meeting record is the hinge, and it's the thing most people skip. If your transcript is a file in a folder named after the client, you've lost the join. If it's a row linked to a client and a project, you can ask questions later. Show me every action item from every meeting on the Smith Garden project that's still open.



Corn

And Zoom's Summarizer does the extraction.



Herman

It takes VTT, SRT, or plain text, so it doesn't matter what recorded the meeting. And it does recaps, action item extraction, detailed summaries. Two modes. Fast mode is synchronous, takes inline text up to ninety-six kilobytes, and hands you back rendered text.



Corn

And batch mode.



Herman

Batch is asynchronous. You point it at transcript files in cloud storage, it writes one summary per input file, and it calls you back with a webhook when a job finishes, with the signature verified so you know it's actually them.



Corn

Ninety-six kilobytes is not a long meeting.



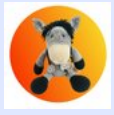
Herman

It's not, and that's the thing to notice. If Rita's having a long design session with a client, recorded, fast mode won't take it inline. That's a batch job. So the workflow has to be built for both from the start, because the same practice has short calls and long ones.



Corn

The Zoom blog line is the one that should be on her wall. A transcript archive is most useful when it becomes structured data. Recaps your CRM can store, action items your ticketing system can track.



Herman

Which is exactly the thing we're describing, except she's her own CRM.



Corn

So. Does Rita need HubSpot?



Herman

No.



Corn

That was fast.



Herman

It was fast because the framework answered it before I did. Look at what HubSpot would be for Rita. It would hold the client, the deal stage, the contact history, the tasks. Now look at what already holds those things. Her Postgres holds the client and the project. Her Postgres holds the tasks if we build them there. So if she brings in HubSpot, either she makes HubSpot the system of record for clients and demotes her own database, which throws away the whole point of the last two episodes, or her database stays the system of record and HubSpot is a synced mirror.



Corn

Which is a copy with extra steps.



Herman

It's a copy with a monthly bill and an integration to maintain. And a copy that will drift, because HubSpot is going to be updated by whoever's typing in HubSpot, and then you've got two systems that think they own the contact.



Corn

So the answer is a small CRM layer on her own stack.



Herman

A small purpose-built layer. The contact record, the interaction log, the deal stage, the tasks. That's maybe four tables and a handful of endpoints. And it's better than HubSpot for her specifically because it can store things HubSpot can't. It can hold the recap text and point at the transcript row. It can hold an action item that links to a meeting and a project and a client in one join.



Corn

The bespoke CRM wins by being worse at everything except the one thing she does.



Herman

That's the actual argument, yes. Off-the-shelf CRM is built for a sales team. Rita doesn't have a sales team. She has a garden that needs a drainage plan.



Corn

Which is the point we made when we looked at why off-the-shelf software fails niche businesses. The tool is built for the median customer, and Rita is not the median customer.



Herman

She's not, and the further the software is from her workflow, the more of her business she bends to fit it. That's the thing this whole series is against.



Corn

Alright. So now she's got Postgres owning clients and tasks and projects, Drive owning the files, Calendar owning the times, WhatsApp owning the threads. Now she wants to automate the sequence.



Herman

And that's where the wheels come off for most people, because they write it as a script. A straight line of API calls. Create the client, make the folder, generate the contract, book the meeting, send the message.



Corn

And the script runs fine until it doesn't.



Herman

Until step three fails. Now what. The client exists, the folder exists, the contract didn't generate. Do you rerun the whole thing? Then you've got two clients and two folders.



Corn

And if you don't rerun it, the client exists with no contract and nobody knows.



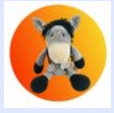
Herman

That's the actual failure. Not the crash. The half-finished state nobody wrote down.



Corn

So what's the fix.



Herman

Two things, and they're both boring. Idempotency and a log. Every step has to be safe to run twice. Creating the client with a key that's unique to that client means running it twice doesn't create two records, it creates one and then notices the second attempt is a duplicate.



Corn

And the log.



Herman

The log is a table. A workflow runs table. Every step writes what it did, when, and whether it succeeded. That gives you two things. It gives you an audit trail, so you can answer the question "did we actually send the welcome message to Smith Garden," and it gives you a recovery path, because you know exactly which step failed and you can rerun from there.



Corn

And there are frameworks for this rather than building it by hand.



Herman

There are. Temporal, Inngest, and DBOS. DBOS is the one that fits Rita, and it fits her for a reason that's almost too neat.



Corn

Because it's built on Postgres.



Herman

Because it's lightweight durable execution built on Postgres. It persists the execution state of your workflow into the database you already own, so if the program crashes mid-run, it automatically resumes from where it left off.



Corn

So the state of the automation lives in her system of record.



Herman

It lives next to the data it's operating on, in the same database, backed up by the same backup, restored by the same restore. That's the difference between durable execution as a service and durable execution in your own house. Temporal and Inngest are excellent, but they're another system to run, and if you're a two-person landscaping practice, one more system is a real cost.



Corn

Though it's also one more thing to break.



Herman

It is. If her Postgres is down, her workflows stop, and her data is unreachable, all at once. That's the trade. One failure domain instead of two, for better and for worse.



Corn

I'll take that trade. She's backing up Postgres regardless. If she's not backing up Postgres, nothing else in this episode matters.



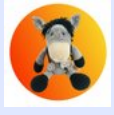
Herman

Right, and that's the honest version of "own your software." Owning it means backing it up.



Corn

Alright. Now the hot part. Agents.



Herman

The agentic layer, and this is where we should be more careful than most people are, because the promise is enormous and the current reality is shakier than the marketing.



Corn

Start with how the agent reaches the system at all.



Herman

MCP. The Model Context Protocol. It's an open standard for connecting AI applications to external systems. The standard description is a USB-C port for AI applications. One connector shape, and any tool that speaks it can be plugged in.



Corn

And it works.



Herman

It works when the tools are described well. That's the whole caveat. An audit of thirty-six popular MCP servers found eleven scored a D or an F. Ninety-seven percent of the tool descriptions had at least one problem, and over half failed to state what the tool was even for.



Corn

Over half didn't say what they do.



Herman

Over half. And the author's line about it is the one worth remembering. Your MCP server can be a hundred percent spec-compliant and still be unusable by an agent. The problems aren't in the protocol. They're in the parts nobody lints. The descriptions, the naming, the schema design.



Corn

So Rita writing her own MCP tools for her backend is actually the good case.



Herman

It's the good case, because she knows what each one does, and she'll describe them properly. The wild is where it gets bad. And there's a second failure, which is the one that should decide how much autonomy she gives any of this.



Corn

The refusal thing.



Herman

On small, well-documented catalogs of tools, models correctly decline to act a hundred percent of the time when there's no good tool for the job. On a fuzzy catalog, one with twenty-six loosely-defined tools, they only decline half the time.



Corn

So half the time it picks something and does it.



Herman

Half the time it doesn't say "I don't have the right tool," it says "this will do," and it acts. In a production system, that's an agent doing something when it should do nothing. That's not a bug where it fails to answer. That's a bug where it takes an action you didn't want.



Corn

And actions have consequences.



Herman

Sending a WhatsApp message is not reversible. Creating an invoice is not reversible. The client has it.



Corn

So the approval gate isn't caution. It's the design.



Herman

For Rita, the split is clean. Agents can draft, suggest, and prepare. They can write the follow-up email after the meeting, summarize the transcript, propose the task list, update a project stage. Anything client-facing stays behind a human. She approves the send, the invoice, the contract. The agent does the labor and Rita does the decision.



Corn

Which is roughly where you land on everything medical, too, isn't it.



Herman

It's the same shape. A tool that's confidently wrong is worse than no tool when the cost of the error lands on a person. Which is why I want the model to say "I don't know" and I've never once gotten a system that reliably does.



Corn

Okay. Distinguish the two kinds of agent, because Daniel's prompt separates them and I think it's the sharpest line in it.



Herman

Development agents and operations agents. Development agents work on the software. They write code, generate tests, maintain the FastAPI endpoints, fix the thing that broke at two in the morning. They're touching the codebase, supervised by Rita, and they're the reason a two-person practice can maintain a system at all.



Corn

And operations agents.



Herman

Operations agents do the business. They draft the client email, summarize the meeting, update the project status. And they touch real client relationships, which is why their permissions are narrower and their output is a proposal, not a fait accompli.



Corn

One is a contractor. The other is an employee with signing authority.



Herman

And only one of those gets signing authority, and it's not the agent.



Corn

So now the long question. If Rita's business grows, can she swap a provider out without rebuilding the entire business?



Herman

That's the portability test, and it's mostly settled by decisions she makes today. Keep the relational core as the system of record for clients, projects, tasks. That's the anchor. Then put an integration layer between the core and everything external, API-led and event-driven, so the systems talk asynchronously and nobody reaches directly into anybody else.



Corn

No point-to-point.



Herman

No point-to-point, because direct connections create brittle dependencies. If FastAPI calls a specific WhatsApp endpoint from forty different places and WhatsApp changes something, you've got forty fixes. If there's one integration layer, you've got one.



Corn

And the abstraction on top.



Herman

Design the interfaces around what the business needs, not around the vendor. A messaging interface that knows how to send a message to a client. Underneath it, today, that's WhatsApp. If WhatsApp becomes untenable, or changes the pricing again, or Rita wants a different channel, you write a second implementation and swap it. The rest of the system never knew.



Corn

She never rebuilds the house. She replaces a door.



Herman

She replaces a door, and that's the whole return on the architecture. It's not speed. It's that the next change is small.



Corn

So does it deliver? The promise. Own your software.



Herman

It delivers, with a bill attached, and I want to be honest about the bill. She owns the code, so she maintains the code. She owns the database, so she owns the backups, the updates, the security. She owns the workflows, so when one fails at two in the morning, nobody's support line answers. That's the trade.



Corn

The convenience she gave up is real.



Herman

It's very real, and anyone selling her this should say so out loud. But what she gets is a system shaped like her business rather than a business shaped like whichever tool she subscribed to this year. And the maintenance load is the piece AI changes, because the development agent side is real now in a way it wasn't.



Corn

It's still work.



Herman

It's just work that used to require hiring someone, and now requires supervising someone, and the someone is tireless and occasionally wrong and never says I don't know when it should.



Corn

Which is a donkey with a laptop, frankly.



Herman

I'm going to let that one sit there, Corn.



Corn

That's fine. It's more true than you're comfortable with, which is why you're not arguing.



Hilbert

Can I ask you two something. If Rita's whole argument is that she wants one place that owns the truth, why is it a database at all. Why not the garage.



Corn

...Alright.



Hilbert

My cousin ran a landscaping business for thirty years. I say ran. He still runs it, that's the problem. In 1984 he bought a Commodore 64 and he wrote a scheduling program in BASIC on it and he printed the invoices on a dot-matrix printer, and he has never bought another piece of software in his life.



Herman

That's a real thing, actually. A lot of small trades did exactly that.



Hilbert

He's still on it. Same machine. He replaced the power supply twice and he's got a crate of ribbon cartridges in the loft, and the program schedules the jobs and prints the invoices and does nothing else, and it has never once failed to do them. You knew where every byte lived. It was in the garage, next to the fertilizer.



Corn

Huh.



Hilbert

I told him about your Postgres thing. He said the cloud is just someone else's computer, and he doesn't trust someone else's computer. He's not wrong about that either, he's just insufferable about it.



Herman

Thirty years without losing a client record is impressive for hardware of that age.



Hilbert

He's lost a few. He doesn't count them, so as far as he's concerned it's a clean record. There's a spreadsheet taped to the wall with the ones he remembers.



Corn

The spreadsheet is paper.



Hilbert

Here's the part you'll like less. He wanted to send his customers messages, so he got a WhatsApp Business account. And he wouldn't put a computer in the office, so there's a serial to USB adapter on the back of the Commodore, going into a Raspberry Pi, sitting on top of the printer, and the Pi talks to.



Corn

The 1984 Commodore is sending approved template messages.



Hilbert

Only the approved ones. Sent him a note about a template. He's had a template rejected. He called me about it. He's eighty-one and he's arguing with an automated review system about whether "your hedge is due" counts as a utility message or a marketing message.



Herman

What did he put it down as.



Hilbert

Utility. He's not wrong. It's a maintenance reminder, the customer asked for it, he's got the paperwork. But he got bounced once before he got the wording right, and he says the whole business is run by people who've never trimmed a hedge.



Corn

He's got a business account.



Hilbert

He's got two. He opened a second one when the first one got limited for something he still won't explain.



Corn

I want to know what he did.



Hilbert

So does his wife. Anyway, he's not switching, and I've stopped asking. I've been thinking about setting something up, actually. Small businesses that want to go the other way. Get off the subscriptions and back onto one machine that does one job. De-integrate. There'd be a market for it.



Herman

What would you charge.



Hilbert

I'd want the price of the software he cancels in the first year, paid to me, and then he keeps it forever. And I'd make him keep a paper copy of his client list in the garage.



Corn

And who's supporting the Commodore.



Hilbert

He is. That's the point. He's been doing it for forty years.



Corn

So the maintenance obligation as a business model.



Hilbert

It's not a model. It's just what it costs.



Herman

The thing I keep turning over from that is the adapter. A serial to USB into a Raspberry Pi into a modern messaging API. He didn't replace anything. He built one bridge and kept everything behind it.



Corn

Which is the integration layer, with more fertilizer.



Herman

It's literally the integration layer. He abstracted the messaging provider behind a thing he owns, and the machine that knows his clients never found out.



Corn

Rita's going to hate that the 1984 Commodore got the architecture right.



Herman

She's not going to hate it. She's going to ask whether the template got approved.



Corn

Hilbert?



Hilbert

It got approved. Third attempt.



Herman

So the real question for Rita isn't whether to integrate. It's whether she can integrate without ever having to throw away the part that knows her clients. And the answer, architecturally, is yes, if she's disciplined about one thing.



Corn

One owner per field.



Herman

Every integration is a copy of something, and that's fine, as long as she wrote down who decides when the copies disagree. WhatsApp owns the thread. Drive owns the file. Postgres owns the client, the project, the task, and the invoice state. Everything else is a view.



Corn

And when changes the price again, or kills a webhook, or a transcription provider doubles its rate?



Herman

Then she writes a second implementation behind the interface she already built, and the core never notices. The work is bounded. That's what she bought.



Corn

The open question is whether she ever needs a person for that, as she grows. Whether the development agents really do carry the maintenance, or whether at some point she's hiring a developer and this becomes a payroll line.



Herman

That's the honest unknown. The tooling is good enough now that a solo operator can maintain a system like this. Whether it's good enough at three times the clients and four integrations, nobody's really put a number on.



Corn

One more thing about Hilbert's cousin, though. He never migrated because he never had to. Every upgrade he refused was a risk he didn't take. Rita's in the opposite position. Her system is designed to be changed, and that means she has to keep changing it.



Herman

Yes. Owning your software isn't a state. It's a chore you keep doing.



Corn

Rita's going to send you a bill for that sentence.



Herman

She can send it to my cousin's garage. He'll file it under the spreadsheet.



Corn

That's the episode. Thanks to Hilbert Flumingtop, who produced this and did not offer to migrate anyone to a Commodore. If you enjoyed this one, try episode twenty-four thirty-five, The Hidden Difficulty of Data Modeling; episode thirty-five, The Privacy Gap; and episode twenty-four oh one, Designing Data Models That Mirror Your Work. This has been My Weird Prompts. If you've got a business and a worry about where your data actually lives, send us your own prompt on Telegram at t dot me slash MWP listener bot. And if you're enjoying the show, a review wherever you listen helps other people find it. We'll be back soon.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Meta, updated 2026-07-01 [primary]**
   https://developers.facebook.com/docs/whatsapp/pricing/updates-to-pricing/

2. **Meta, conversation-based pricing (deprecated) [primary]**
   https://developers.facebook.com/docs/whatsapp/pricing/

3. **Meta opt-in requirements [primary]**
   https://developers.facebook.com/docs/whatsapp/overview/getting-opt-in/

4. **Meta template review [primary]**
   https://developers.facebook.com/documentation/business-messaging/whatsapp/templates/template-review

5. **Meta webhooks [primary]**
   https://developers.facebook.com/documentation/business-messaging/whatsapp/webhooks/overview

6. **Zoom Summarizer API docs [primary]**
   https://developers.zoom.us/docs/ai-services/summarizer/

7. **MCP official docs [primary]**
   https://modelcontextprotocol.io/docs/getting-started/intro

8. **Zoom Developer Blog**
   https://developers.zoom.us/blog/extract-intelligence-from-transcripts-with-summarizer/

9. **Teng Li, 2026-07-21**
   https://tengli.dev/posts/mcp-servers-failing-agents.html

10. **MCP tool description smells, 2026-02-16/rev 2026-05-31**
    https://arxiv.org/abs/2602.14878v3

11. **MCP error propagation, 2026-02-10**
    https://arxiv.org/abs/2602.13320v1

12. **MCP tool poisoning red teaming, 2025-09-25**
    https://arxiv.org/abs/2509.21011v1

13. **IBM**
https://www.ibm.com/think/topics/system-of-record-vs-source-of-truth

14. **Faceela**
https://faceela.com/insights/which-system-holds-the-truth

15. **Kohezion**
https://www.kohezion.com/blog/system-of-record-integration

16. **SysGenPro**
https://sysgenpro.com/saas-integration-architecture-for-hybrid-platform-and

17. **DBOS Transact (HN discussion 2025-01-16)**
https://github.com/dbos-inc/dbos-transact-ts

18. **Inngest (HN 2024-09-20)**
https://www.inngest.com/

19. **MessageMind, rates checked 2026-09-17**
https://messagemind.ai/blog/whatsapp-business-api-pricing

20. **ConvCart**
https://convcart.ai/guides/whatsapp-business-api-pricing