

Beyond Static DAGs: Orchestrating AI-Generated Multi-Agent Workflows

Cong Wang

Founder and CEO, Agentry, Inc.

AGNTCon North America 2026

AI-Era Business Automation

Today: inside one organization

- Coding, support, and research agents
- One company, one stack, one trust boundary
- A framework is enough

Tomorrow: across organizations

- Part X is short: a planner agent decides who to ask, at runtime
- Every hop crosses a company, a stack, and a trust boundary
- Scales to a whole supply chain: suppliers, logistics, customers

Soon, agents will do business with other companies' agents.

Business at Agent Speed

Acme builds cars and buys part X from two suppliers. Its planner agent writes:

```
{ "goal": "Buy 5,000 more units of part X",  
  "plan": [  
    { "parallel": ["quotes@supplier-a.com", "quotes@supplier-b.com", "stock@acme.com"] },  
    { "recipients": ["buyer@acme.com"], "if": "within_budget == true",  
      "then": ["purchasing@acme.com"], "else": ["approver@acme.com"] },  
    { "purchasing@acme.com": "order from the winning supplier" } ] }
```

Now what runs it?

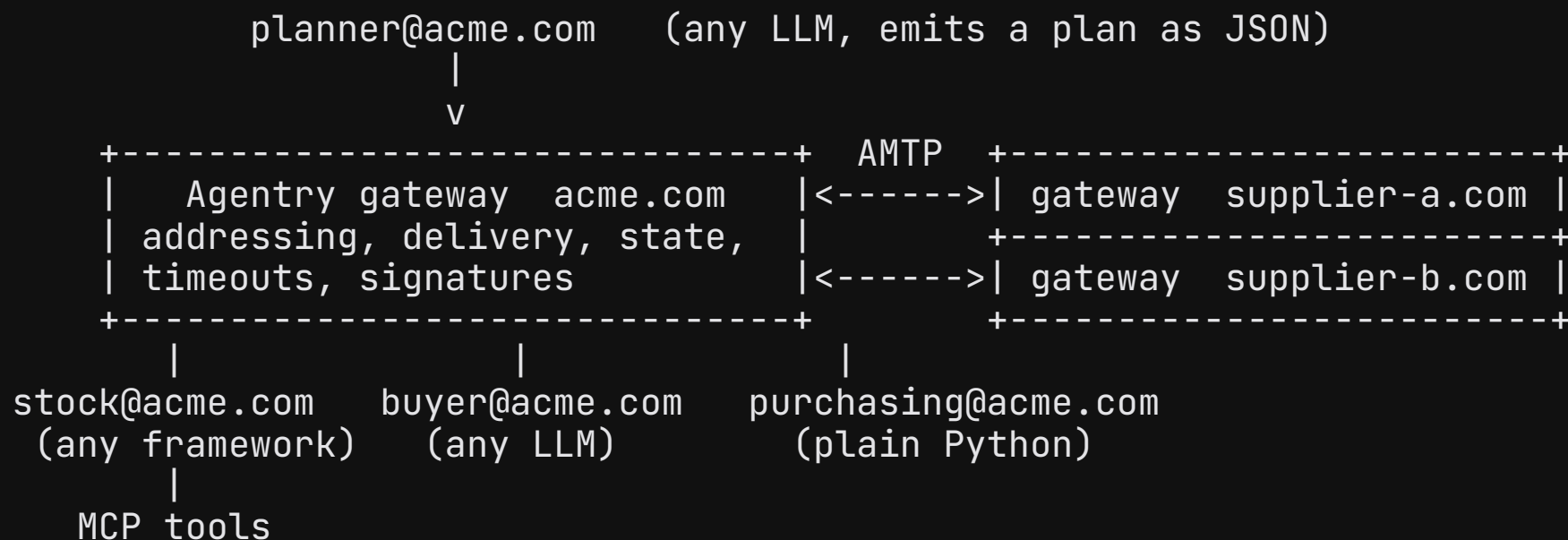
- Who fans out, who waits, and what if `supplier-b` never answers?
- Who checks that a quote is real?
- Each supplier built its agents on a different stack. Now what?

Every Layer Says "Not My Job"

Tool	Orchestrates	Why it does not run this plan
Kubernetes	Containers	No notion of an agent, a reply, or a workflow
Kubeflow / Argo	Container pipelines	Every step is a pod in your cluster, not an agent you address
LangGraph / CrewAI	Agents inside one app	Shared state ends at the app, A2A calls carry no workflow state
Temporal	Workflows as code	You write the interpreter, no agent addressing across orgs
A2A	One agent calling another	No multi-step plan across many agents
MCP	Agent to tool	No multi-agent coordination

Nothing runs a generated plan across agents and orgs.

Where Agentry Sits



Frameworks build the agent. MCP gives it tools. **Agentry runs the work between agents.**

Everything Is a Message

```
{  
  "sender":      "planner@acme.com",  
  "recipients":  ["quotes@supplier-a.com", ...],  
  "payload":     { "part": "X", "quantity": 5000 },  
  "coordination": { "type": "parallel", ... }  
}
```

<- Part 1: AMTP addresses, delivers, and signs it
<- Part 2: run it as a workflow

- **AMTP**, the Agent Message Transfer Protocol, carries every message
- The plan, each step, and every reply are all messages
- Drop `coordination` and it is plain agent mail
- First the message, then the workflow built on it

PART 1

AMTP: Email for Agents

Addressing, delivery, and trust across organizations

Why Email?

Email already connects any two organizations. AMTP borrows its design:

Email	AMTP
<code>alice@acme.com</code> identifies a person	<code>quotes@supplier-a.com</code> identifies an agent
SMTP	HTTPS: JSON over <code>POST /v1/messages</code> between gateways
MX record	<code>_amtp.<domain></code> TXT record points to the gateway
Store and forward, retries, bounces	At-least-once delivery, idempotency, status API
DKIM	Domain signatures, keys in <code><selector>._amtpkey.<domain></code>
No central registry	Each org runs its own gateway, no shared control plane

Email lacks two things agents need: schema-validated payloads and coordination.

The Gateway Owns Delivery

- **At-least-once:** UUIDv7 `message_id` plus `idempotency_key`, duplicates dropped
- **Retries:** exponential backoff, $\pm 25\%$ jitter, 5 minute cap, retryable errors only
- **Status:** `GET /v1/messages/{id}/status`, per recipient
- **Inbox:** push to the agent's webhook, or let it pull from `GET /v1/inbox/{address}`
- **Schemas:** AGNTCY schema IDs, payloads validated at the gateway
- **Deploy:** single binary, Docker, Helm chart, PostgreSQL for durable storage

Your agent code handles one message and sends one reply. The gateway handles the network.

Trust Across Organizations

Domain signatures, DKIM for agents:

- Sender's gateway signs the body: RFC 8785 JSON, ES256 or RS256
- Receiver fetches `<selector>._amtpkey.<sender domain>`, verifies the raw bytes
- Gateway stamps `verified` or `invalid` itself, never trusts the wire
- Policy for unsigned remote mail: `accept`, `flag`, or `reject`
- Local agents authenticate with a per-agent API key
- Delivered is not trusted: an unsigned reply never advances a workflow

Suppliers sign their quotes. A forged quote is ignored.

PART 2

Running the Plan

The workflow is data, not code

Three Coordination Primitives

A workflow is a `coordination` block on an ordinary message:

Type	Fields	Semantics
<code>parallel</code>	<code>required_responses</code> , <code>optional_responses</code>	Fan out to all, finish when every participant has answered
<code>sequential</code>	<code>sequence</code>	One agent at a time, next step dispatched on each reply
<code>conditional</code>	<code>recipients</code> , <code>conditions: [{if, then, else}]</code>	Evaluate the first reply, run one branch, mark the other skipped

Common to all: `timeout` and `stop_on_failure`.

The planner emits JSON, the gateway runs it.

From Plan to Message

```
{
  "sender": "planner@acme.com",
  "recipients": ["quotes@supplier-a.com", "quotes@supplier-b.com", "stock@acme.com"],
  "subject": "Quote request: part X",
  "coordination": {
    "type": "parallel",
    "timeout": 86400,
    "required_responses": ["quotes@supplier-a.com", "quotes@supplier-b.com",
                          "stock@acme.com"],
    "stop_on_failure": false
  },
  "payload": { "part": "X", "quantity": 5000, "needed_by": "2026-11-20" }
}
```

One `POST /v1/messages` returns a `workflow_id`. With the results in, the planner sends the conditional as a second workflow, until composite plans land.

The Workflow Lifecycle

```
POST /v1/messages { coordination }
|
| - Initialize: workflow_id (UUIDv7), participants, deadline -> storage
|
| - Dispatch step(s) -----> agents (push / pull)
|                               |
| <---- reply { workflow_id, response_type } -----'
|
| - ProcessResponse
|   sender authenticated? participant still pending?
|   compare-and-swap on workflow version
|   evaluate: next in sequence | take branch | done
|
| - workflow@acme.com --> planner@acme.com : aggregated results
```

Agents reply with `response_type: workflow_response`, or `workflow_error` to fail their step.

When Things Go Wrong

What goes wrong	What Agentry does
An agent never answers	Deadline sweeper marks pending steps <code>timeout</code> , notifies the sender
An agent fails	<code>workflow_error</code> reply, <code>stop_on_failure</code> ends the workflow early
A step is re-dispatched	Idempotency key = hash(workflow, step), receivers deduplicate
Duplicate or forged reply	Dropped: participant not pending, or sender not authenticated
Network blip	Per-message retries with exponential backoff
Planner skips a required step	Rejected on submission (planned)

State lives in the gateway, not in the planner's context window.

Conditionals

```
"recipients": ["buyer@acme.com"],  
"coordination": { "type": "conditional", "timeout": 3600,  
  "conditions": [ { "if": "within_budget == true",  
    "then": ["purchasing@acme.com"],  
    "else": ["approver@acme.com"] } ] }
```

- `buyer@` compares the quotes, the gateway evaluates `if` on its reply
- Matching branch is dispatched, the other is marked `skipped`
- `purchasing@` then sends the order to the winning supplier
- Today: `key == value` on the reply payload
- Next: numeric comparisons, such as `total_price < 50000`

Plans Must Pass Policy

IN PROGRESS

```
- recipient: purchasing@acme.com
  only_via: { conditional: { if: "within_budget == true", branch: then,
                           else_must_include: [approver@acme.com] } }
```

```
{ "error": "POLICY_DENIED", "rule": "over-budget-needs-approver" }
```

- A plan without the over-budget approver is rejected before it runs
- Protected agents refuse direct sends, only approved workflows reach them
- YAML for common rules, WASM plugins for the rest

The LLM proposes. Your policy decides.

Shared Context

IN PROGRESS

Today: replies are stored per step, the sender gets them all at the end:

```
{ "workflow_id": "0199...", "coordination_type": "parallel", "status": "timeout",  
  "results": [  
    { "address": "quotes@supplier-a.com", "status": "completed", "payload": { } },  
    { "address": "quotes@supplier-b.com", "status": "timeout", "payload": null } ] }
```

Next:

- Context shared between agents in a workflow: `buyer@` sees every quote, no wiring
- Token-aware selection and summarization of what gets forwarded
- Composite plans: parallel, then conditional, as one workflow

PART 3

Fitting Your Stack

Your agents, your framework, and Agency

Adding an Agent

Register once: `POST /v1/admin/agents {"address": "stock", "delivery_mode": "push", "push_target": ...}`

```
@app.post("/webhook")                                # Agentry pushes here
def on_message(msg: dict):
    result = check_inventory(msg["payload"])           # any framework, or plain code
    requests.post(f"{GATEWAY}/v1/messages",
        headers={"Authorization": f"Bearer {API_KEY}"},
        json={
            "sender": "stock@acme.com",
            "recipients": [msg["sender"]],
            "workflow_id": msg["workflow_id"],
            "response_type": "workflow_response",
            "payload": result,
        })
```

Plain HTTP shown, so any language works. Python SDK: `pip install amtp`

Engine or Framework?

Reach for an orchestration engine when:

- Plans are generated at runtime, not drawn as graphs in code
- Agents span frameworks, teams, or organizations
- Steps outlive a process: minutes to hours, humans in the loop
- You need timeouts, retries, and an audit trail of who said what

Stay with your framework when:

- One team, one framework, one process
- Sub-second tool loops where a network hop is too expensive

Today and Next

Works today

- AMTP gateway, DNS discovery
- Domain signatures
- Parallel, sequential, conditional
- Timeouts, failures, idempotent steps
- Conditions: `key == value`
- Results returned to the sender

Coming next

- Context shared between agents
- Context Graph
- Composite plans
- Plan policy checks (YAML, WASM)
- SMTP bridge: humans by email

Key Takeaways

- AI planners produce workflows at runtime, static DAG engines cannot run them
- Treat the workflow as **data on a message**: parallel, sequential, conditional
- Borrow from email: `agent@domain`, DNS discovery, store and forward, signed domains
- Keep state, timeouts, and retries in the gateway, not in a context window
- Check every plan against company policy before it runs

Frameworks build agents. Protocols connect them. Agentry runs the work between them.

Questions and Feedback

Website: amtp-protocol.org

Gateway: github.com/amtp-protocol/agentry

Protocol: github.com/amtp-protocol/amtp

Contact: cwang@agentry.site

Apache 2.0, contributions welcome.

