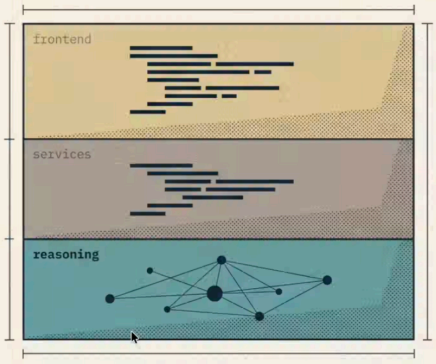
AF

The AI Backend

AgentField Day — The Autonomous Backend AI Hackathon



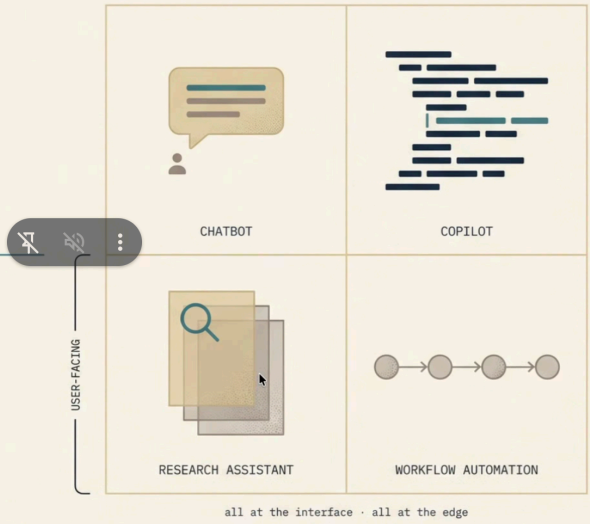
Santosh Kumar Radha
CO-FOUNDER & CTO, AGENTFIELD

Singapore · February 7, 2026

agentfield.ai

The world you know

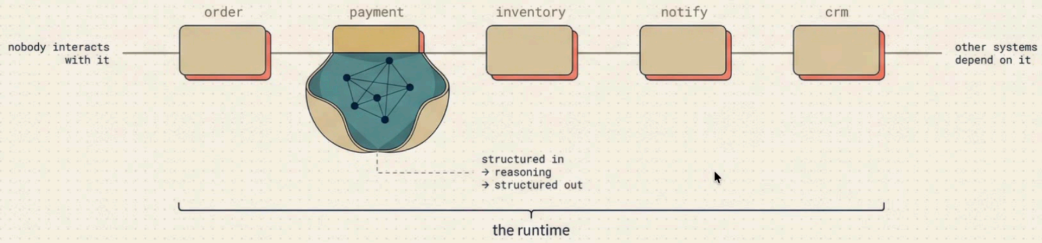
"Nearly every AI application you've built lives here."



agentfield.ai

The other kind

There's another kind of AI system that doesn't get talked about.

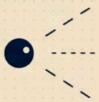


Once an AI system begins to act as part of a runtime, it stops being an assistant and starts behaving like software that other software depends on.

agentfield.ai

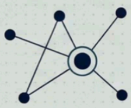
If something goes wrong, who notices first?

the litmus test



A person notices

- AI at the interface
- Outputs are suggestions
- Human carries ambiguity
- Chatbot, copilot, assistant



The system notices

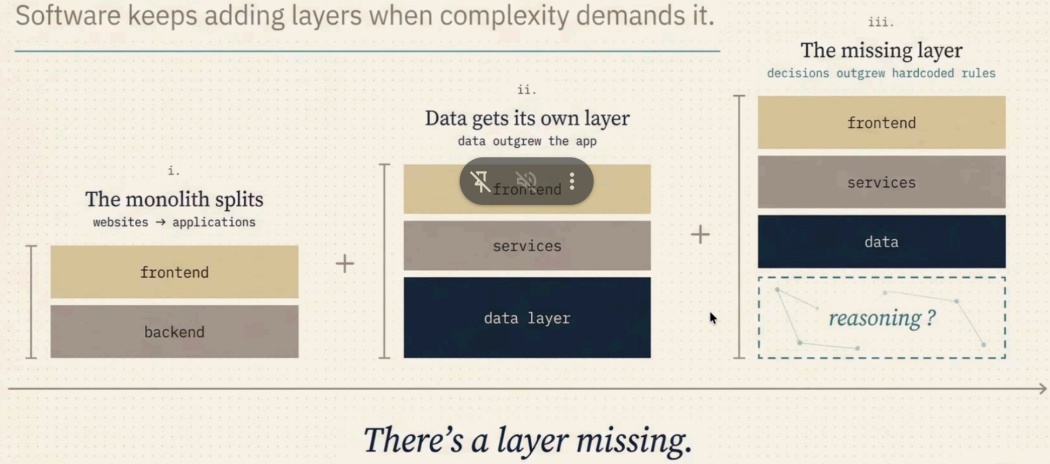
- AI in the runtime
- Outputs are dependencies
- Infrastructure carries ambiguity
- **The AI backend**

This single question cuts through all the confusion about agents, autonomy, and production readiness.

agentfield.ai

The pattern

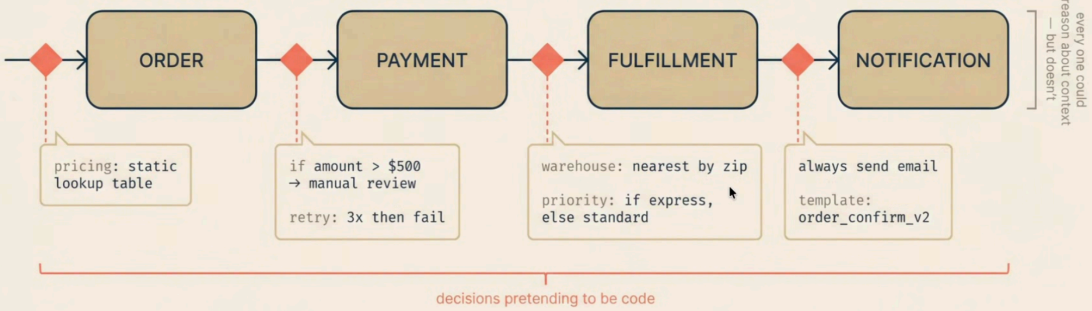
Software keeps adding layers when complexity demands it.



There's a layer missing.

agentfield.ai

The backend everyone recognizes



agentfield.ai

DEAD END #1

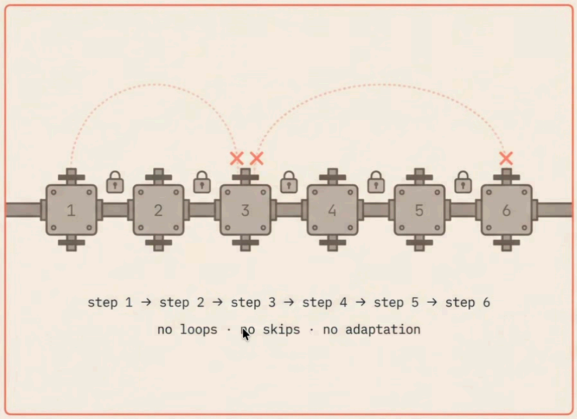
The DAG Trap

Some approaches treat AI like data pipelines. Fixed steps, explicit flows, one node feeding the next.

- Can't adapt mid-flow
- Can't loop back or reconsider
- Wrong mental model for reasoning

"Force reasoning into rigid pipelines and you've built a more expensive rules engine."

Predictable but brittle



DEAD END #2

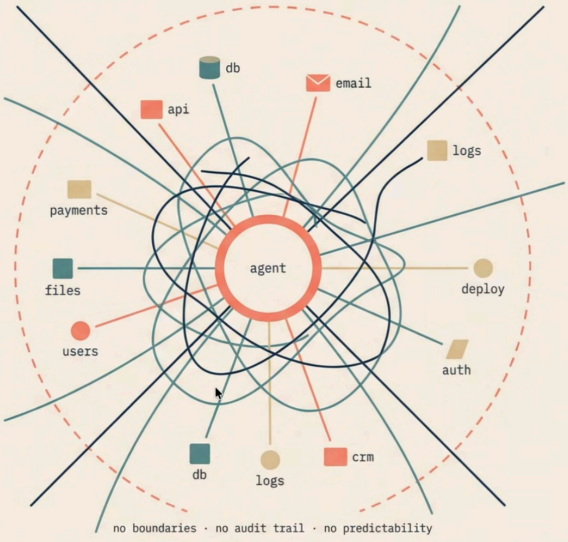
The autonomous agent fantasy

The opposite extreme: one orchestrator with access to every tool, deciding on the fly.

- Can't predict behavior
- Can't audit decisions
- Can't explain what happened

Demos beautifully. Fails catastrophically.

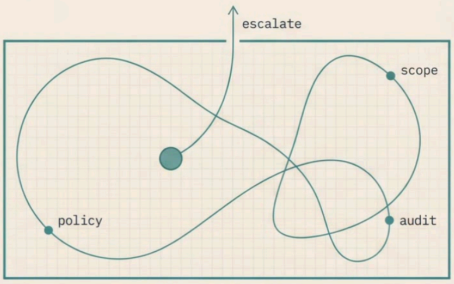
Flexible but chaotic



THE THIRD WAY

Guided autonomy

Reasoning freely within boundaries you define.



autonomous within bounds

	DAG TRAP	UNBOUNDED AGENT	GUIDED AUTONOMY
Adapts?	No	Yes	Yes, within bounds
Auditable?	Yes	No	Yes
Production-ready?	Brittle	Dangerous	**Yes**

Predictable enough to trust. Flexible enough to be useful.

agentfield.ai

Four primitives.

Here's what guided autonomy looks like in practice.

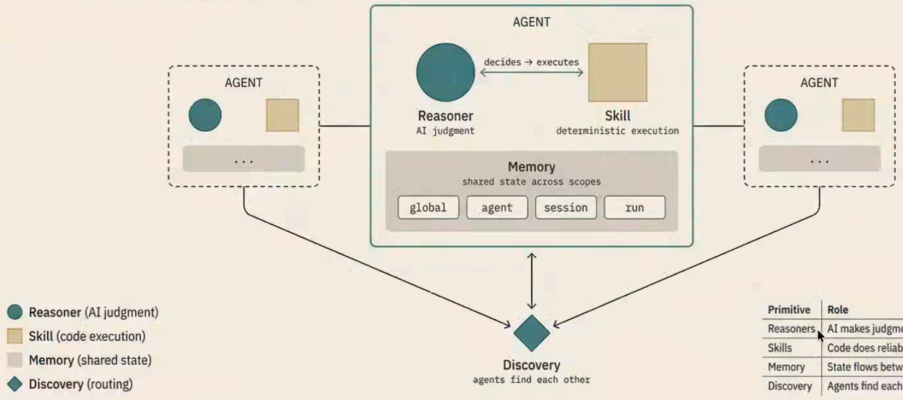
REASONERS SKILLS MEMORY DISCOVERY

That's the whole model.

agentfield.ai

The model

Four primitives. That's the whole system.



- **Reasoner** (AI judgment)
- **Skill** (code execution)
- **Memory** (shared state)
- **Discovery** (routing)

Primitive	Role	Analogy
Reasoners	AI makes judgment calls	The decision-maker
Skills	Code does reliable work	The executor
Memory	State flows between agents	Shared context
Discovery	Agents find each other	Service mesh

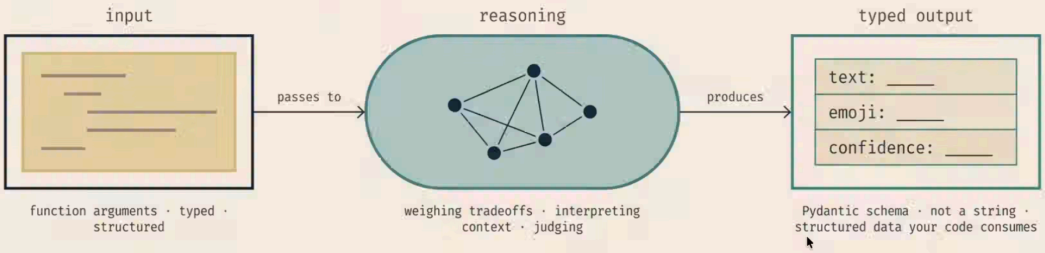
No DAGs. No YAML. No workflow DSLs. Decorate Python functions. Done.

agentfield.ai

Reasoners

Where AI makes judgment calls

Not free-form text generation — structured decisions with typed outputs.



function arguments · typed · structured

weighing tradeoffs · interpreting context · judging

Pydantic schema · not a string · structured data your code consumes

The AI fills in typed fields. Your code consumes structured data. No parsing, no hoping.

agentfield.ai

The code

This is literally all it takes.

- a decorated function
- Pydantic schema defines output shape
- Becomes a REST endpoint automatically

```python
class EmojiResult(BaseModel):
    text: str
    emoji: str

@app.reasoner()
async def add_emoji(text: str) -> EmojiResult:
    return await app.ai(
        user=f"Add an appropriate emoji to: {text}",
        schema=EmojiResult
    )
```

this makes it a reasoner

typed output — not a string

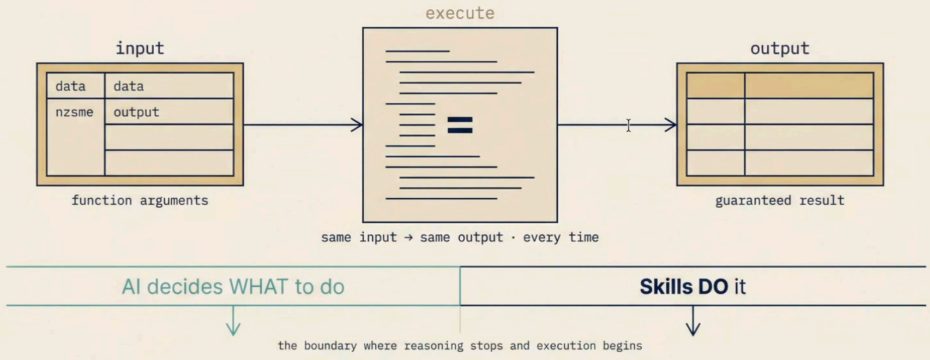
LLM call handled for you

agentfield.ai

Skills

Where code does the work

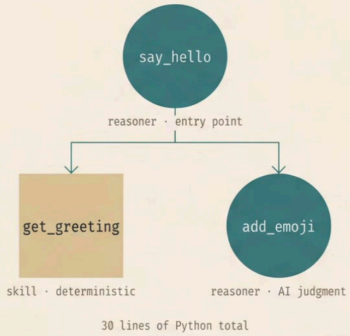
Deterministic functions. No AI. Reliable execution.



· Database writes, API calls, calculations, formatting
· Easy to test, easy to mock, easy to reason about

agentfield.ai

How they work together




```python
@app.reasoner()                                          # entry point
async def say_hello(name: str) -> dict:
    greeting = get_greeting(name)                        # skill (deterministic)
    result = await add_emoji(greeting["message"])        # reasoner (AI)
    return {"greeting": result.text, "emoji": result.emoji}
```

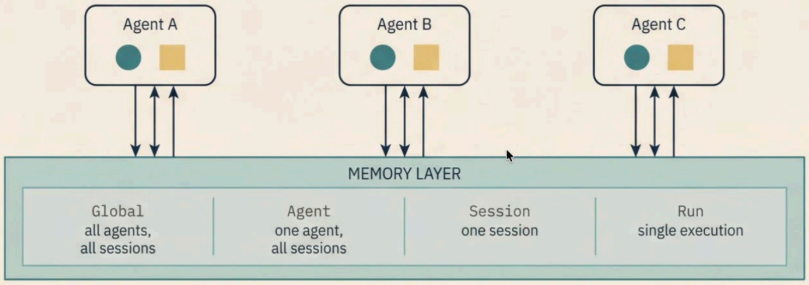
You already know how to build this. It's just Python.

agentfield.ai

Memory

State that flows between agents automatically

No Redis. No Postgres config. No message queues. No pub/sub.



Agent A stores research findings. Agent B retrieves them.
They never knew about each other.

agentfield.ai

Four lines.

Shared state + vector search across all your agents.

- No Redis setup. No Pinecone.
- Key-value AND vector search in one API
- Control plane handles persistence

Key-value

```
                                          → Agent A stores
await app.memory.set("customer_context", context)

context = await app.memory.get("customer_context")
                                          → Agent B retrieves
```

Vector memory

```
                                  → store embedding
await app.memory.set_vector(id, embedding, metadata)

results = await app.memory.similarity_search(query)
                                  → semantic search built in
```

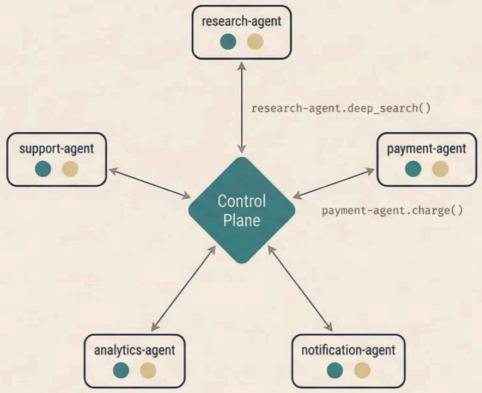
Memory just works.

agentfield.ai

Discovery

Agents find each other

No service registry.
No hardcoded URLs.
No message queue config.



agents don't know where other agents live · control plane handles routing

- Call agents by name
- Automatic routing & load balancing
- Context propagation
- Retries & circuit breaking
- Workflow DAG tracked automatically
- Scale agents independently

Your agents are microservices. You've built microservices before. These can reason.

agentfield.ai

One line.

Cross-agent communication.

- Agents call each other by name
- Workflow DAG tracked automatically
- Retries, circuit breaking — handled
- Compose agents from routers

Call another agent

```
result = await app.call(        # by name, not by URL
    "research-agent.deep_search",
    input={"query": "..."}
)
```

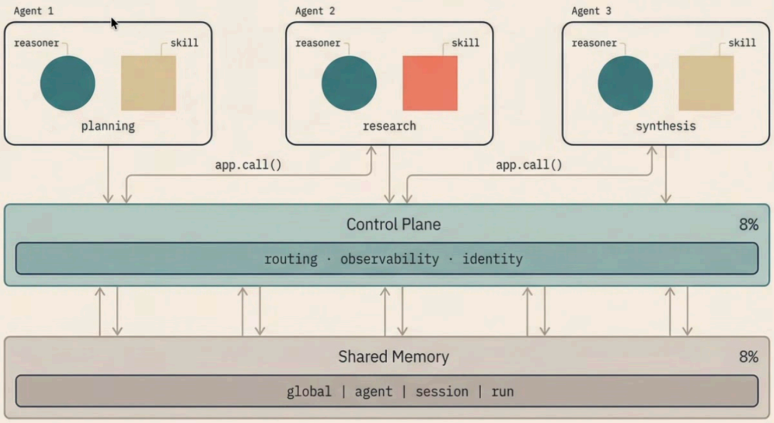
Compose with routers

```
app.include_router(planning_router)
app.include_router(research_router)
                                # like Flask blueprints
```

Your agents are microservices that discover each other at runtime.

agentfield.ai

Everything together

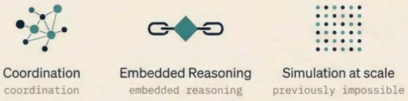


- ✓ Functions become REST APIs
- ✓ Agents run for hours, not seconds
- ✓ Laptop → Docker → Kubernetes
- ✓ Built-in observability
- ✓ Cryptographic audit trails

You write Python functions. We handle everything else.

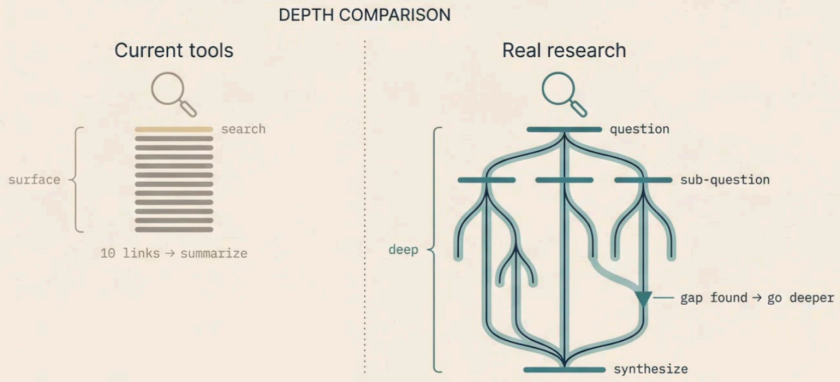
agentfield.ai

Now let's see what you can build with this.



EXAMPLE 1 · COORDINATION PATTERN

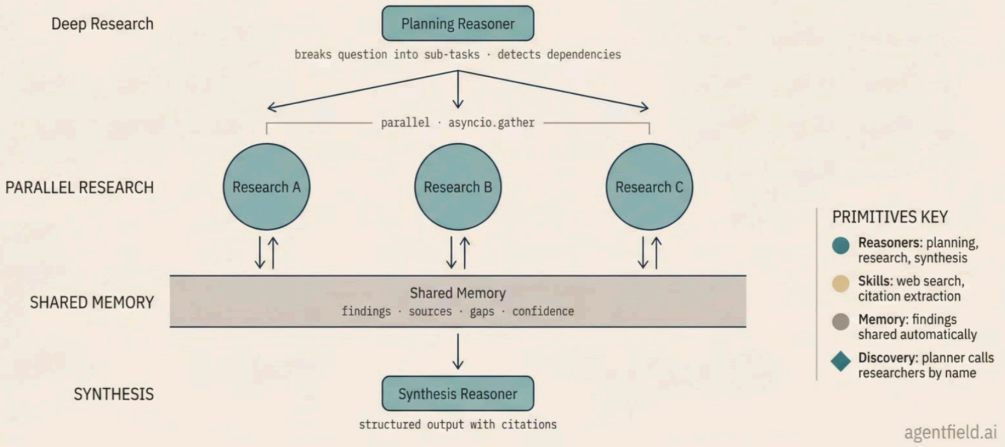
Research that goes deep, not just wide



No human coordinates this. The agents do.

The architecture

Recursive Planning + Parallel Execution



agentfield.ai

The code behind it

Four stages. Each one a decorated function.

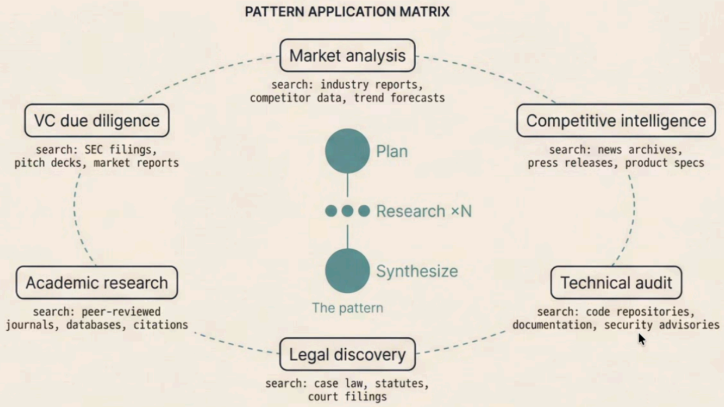
```python
# Plan
@app.reasoner()
async def plan_research(query: str) -> ResearchPlan:
    return await app.ai(
        user=f"Break into sub-tasks: {query}",
        schema=ResearchPlan
    )

# Execute in parallel
tasks = [research_subtask(t) for t in plan.tasks]
findings = await asyncio.gather(*tasks)
# not a DAG — just Python

# Share via memory
await app.memory.set("research_findings", findings)
all_findings = await app.memory.get("research_findings")

# Synthesize
report = await synthesize(query, all_findings)
```

Each step is a simple decorated function.

agentfield.ai

One architecture. Infinite domains.

Replace the search and the prompt — keep the coordination.

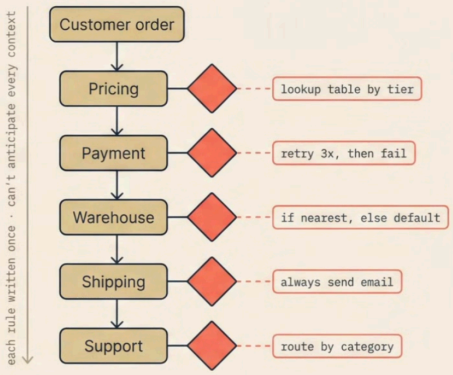


HACKATHON HINT

Same pattern. Replace the domain-specific parts. The coordination is the same.

EXAMPLE 2 · EMBEDDED REASONING PATTERN

Decisions pretending to be code



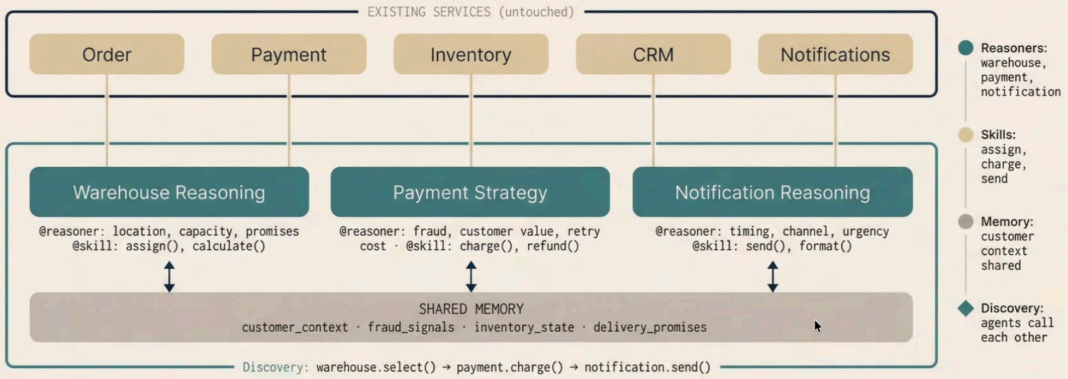
A typical order flow has dozens of decision points — all hardcoded.

- Which warehouse? → if nearest, else default
- Payment failed? → retry 3x, then fail
- Customer complaining? → route by category
- Shipping notification? → immediately, always email
- Pricing? → static lookup table

These aren't engineering problems. They're judgment calls encoded as if-else chains.

EXAMPLE 2 · EMBEDDED REASONING PATTERN

AI at every decision point



Your services stay. The reasoning layer sits alongside.

agentfield.ai

One decision point

A warehouse agent that reasons about context.

- memory
- reasoner
- skill

Reasoner — weighs tradeoffs

```
1  @app.reasoner()
2  async def select_warehouse(order: Order) -> WarehouseDecision:
3      customer = await app.memory.get("customer_context")
4      inventory = await app.memory.get("inventory_state")
5      return await app.ai(
6          user=f"""Select best warehouse.
7          Customer: {customer}
8          Inventory: {inventory}
9          Order: {order}""",
10         schema=WarehouseDecision
11     )
```

shared context from earlier in the flow

Skill — executes the decision

```
12 @app.skill()
13 def assign_warehouse(decision: WarehouseDecision) -> dict:
14     return {"warehouse_id": decision.warehouse_id,
15             "estimated_delivery": decision.delivery_date}
```

deterministic · testable · no AI

This is one decision point. An e-commerce backend has dozens.

agentfield.ai

Rules become policies

Static logic becomes adaptive. The impossible becomes routine.

What changes

if nearest warehouse, else default →	**minimize delivery time within constraints**
retry 3x, then fail →	**weigh customer value** vs **fraud risk** vs **cost**
always send email →	**adapt to preferences** and **urgency**

rules → policies · static → adaptive

What stays the same

- Your services
- Your database
- Your API
- Your deployment

the reasoning layer sits alongside — it doesn't replace anything

HACKATHON APPLICATIONS

Support triage	Dynamic pricing	Fraud detection	Content recommendation
sentiment + history	demand + competition	contextual reasoning	understand context

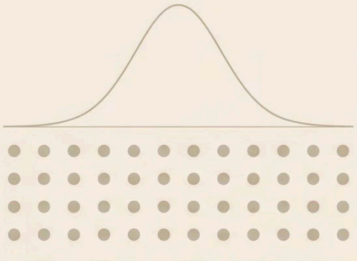
Any backend with hardcoded decisions is a candidate.

agentfield.ai

EXAMPLE 3 · PREVIOUSLY IMPOSSIBLE PATTERN

What if you could ask a thousand?

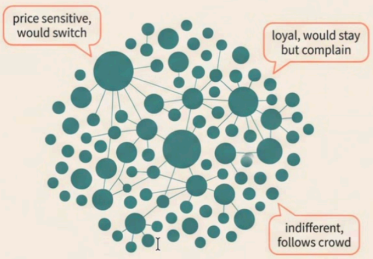
statistical models



probability distributions · Monte Carlo · mathematical rules

can't explain WHY

reasoning entities



each one reasons WHY

Each entity is a reasoner with a profile, context, and judgment.

- A simulated consumer doesn't follow a probability curve
- It reasons about whether to buy
- 1000 diverse consumers, each with different motivations

This isn't statistics. This is synthetic judgment at scale.

agentfield.ai

Same engine. Any scenario.

Domain-agnostic. Replace the entity and the question.

each entity reasons individually

Scenario	Entities	Decision
Price increase impact	1,000 consumers	Stay / switch / downgrade?
New policy rollout	500 employees	Comply / resist / workaround?
Discount optimization	2,000 shoppers	Buy now / wait / competitor?
Market entry	300 businesses	Adopt / ignore / block?
Election polling	5,000 voters	Candidate A / B / abstain?
User satisfaction	1,000 users	NPS score + reasoning why

Previously impossible. Now it is one Python file.

agentfield.ai

Three patterns to build from

Coordination

When to use

Multiple agents explore, share findings, synthesize

Example

Research, due diligence, knowledge extraction

Embedded Reasoning

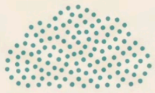
When to use

AI replaces if-else at decision points in existing systems

Example

E-commerce, support triage, fraud detection

Previously Impossible



When to use

AI entities doing things that could not exist before

Example

Simulation, synthetic judgment, prediction at scale

You can combine them. That is where it gets interesting.

agentfield.ai

Ideas to get you started

Or build something we have not thought of. That is how you win.

1 **Adaptive pricing engine**
Agents reason about inventory + demand + competition simultaneously
Embedded Reasoning

2 **Supply chain coordinator**
Shortage detected, alternatives evaluated, routes recalculated
Coordination

3 **PR security reviewer**
Agents find complex security patterns across code changes
Coordination Embedded Reasoning

4 **Synthetic user testing**
Simulate 500 users going through your product
Previously Impossible

5 **Intelligent contract analyzer**
Read contracts, cross-reference clauses, flag risks
Coordination

agentfield.ai