⚡ Lightning Agents ⚡

Factory-of-Factories for Dynamic AI Agents

Riccardo Pirruccio

github.com/RPirruccio/lightning-agents | aimug.org

The Problem & Solution

❌ TYPICAL AGENT CODE

✗ agent_v1.py, agent_v2.py, agent_final.py...

✗ Hardcoded prompts in every file

✗ Copy-paste to create new agents

✗ Change model name in 47 places

✗ No single source of truth

✅ FACTORY-OF-FACTORIES

✓ One agents.json (single source of truth)

✓ Declarative definitions

✓ Registry builds factories

✓ Runtime context injection

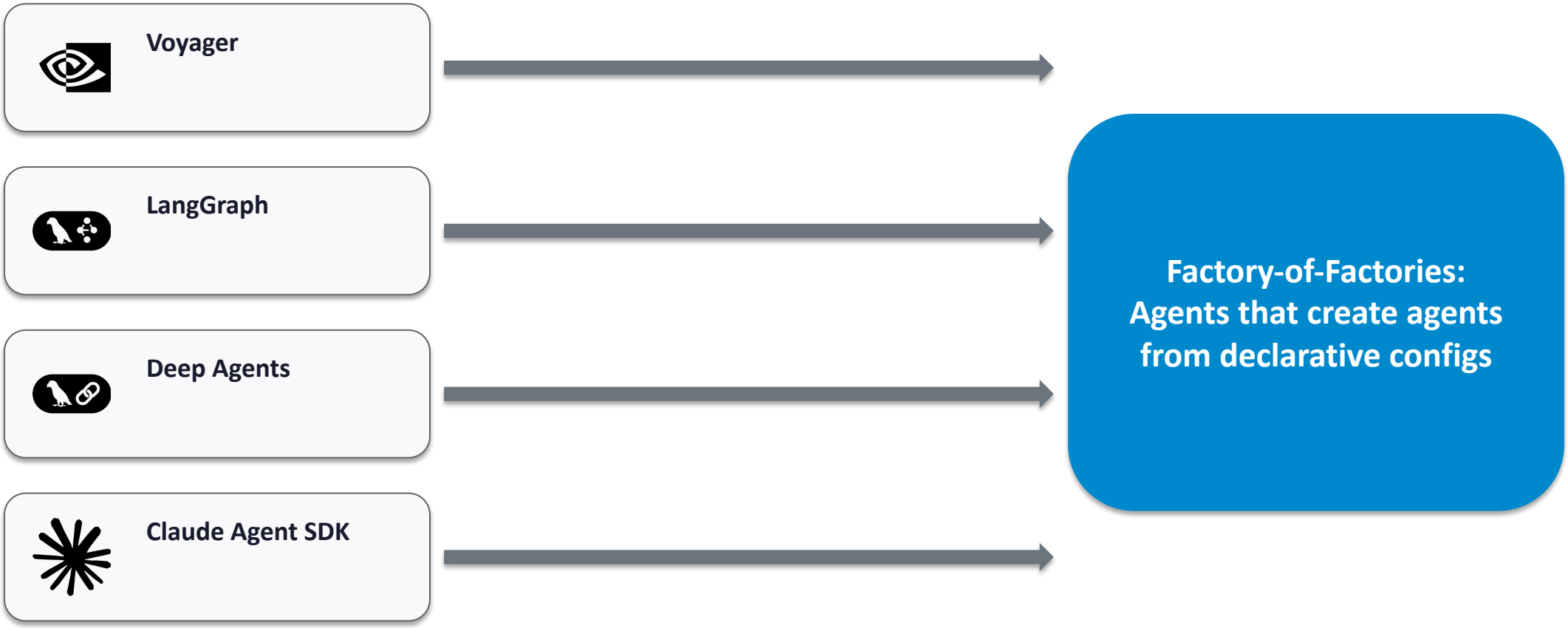
✓ Agents create new agents

The Voyager Insight

🎮 VOYAGER (Minecraft AI)	⚡ LIGHTNING AGENTS
Learns new skill	Creates new agent
Stores in skill library	Stores in agents.json
Retrieves skill when needed	Retrieves agent via Registry.create()
Skills compound over time	Agents create agents over time

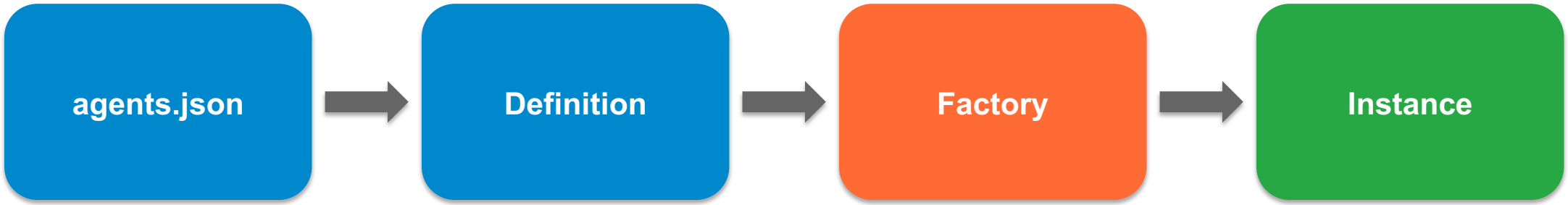
SAME PATTERN. DIFFERENT DOMAIN.

The Industry Agrees



Everyone's arriving at the same place: less code, more context engineering

The Solution: Factory-of-Factories



Simple Factory vs Factory-of-Factories

Simple Factory

```
# One factory, one agent type
def create_agent(config):
    return Agent(config)

# Each new agent = new factory
```

Factory-of-Factories

```
# Registry builds factories dynamically
registry = AgentRegistry.from_json(
    "agents.json"
)
agent = registry.create(
    "researcher", {"topic": "AI"}
)
# Any agent from one registry!
```

Declarative Agent Definition

```
{
  "architect": {
    "name": "Agent Architect",
    "description": "Designs new agents",
    "system_prompt": "You are an Agent Architect...",
    "model": "sonnet",
    "tools": [
      "mcp__custom-tools__db_create_agent",
      "mcp__custom-tools__db_list_agents",
      "mcp__custom-tools__run_agent"
    ]
  }
}
```

JSON, not code — agents defined declaratively in agents.json

Context engineering — the system_prompt is where the magic happens

Composable tools — agents can invoke other agents via run_agent

The Registry Pattern

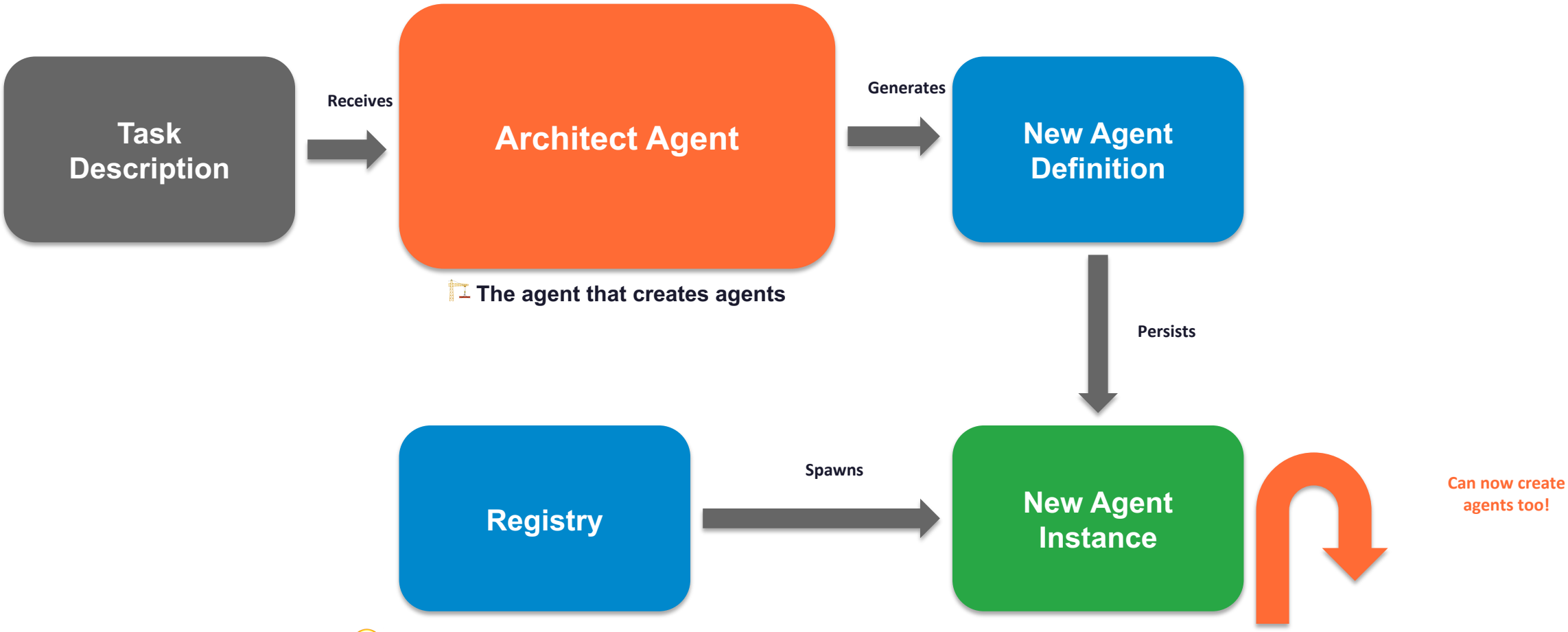
```
class AgentRegistry:
    def from_json(path) -> "AgentRegistry":
        # Load definitions -> build factories

    def create(id, opts) -> AgentInstance:
        # Factory creates configured instance

    def register(id, defn) -> None:
        # Add new agent to registry
```

Single source of truth — all agent definitions in one place

Runtime creation — agents spawn new agents on demand

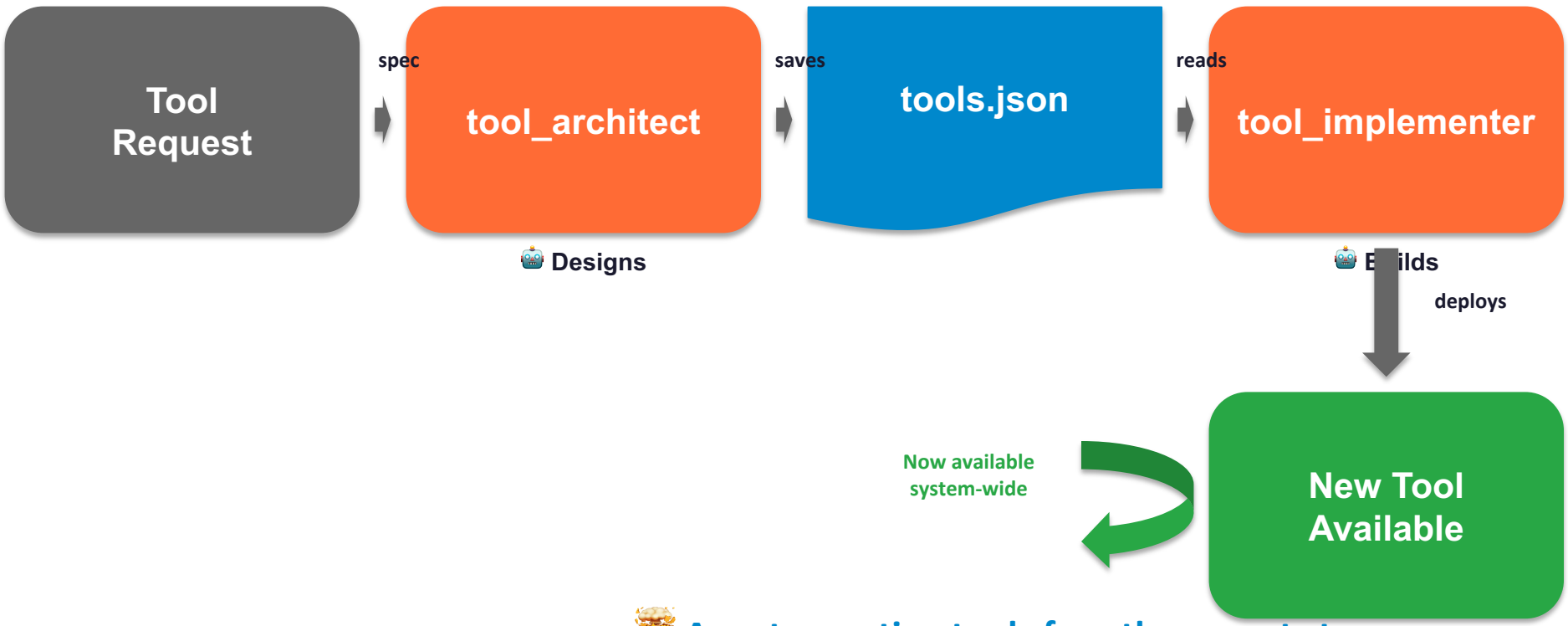
Self-modifying — architects can register new agents dynamically

The Architect Agent



🤔 What happens when created agents can also create agents?

Introducing: The Tool Architect



🤯 Agents creating tools for other agents to use

Demo: What We Built

```
$ lightning list
  10 agents: architect, tool_architect, paper_researcher...

$ lightning run paper_researcher "Find the Voyager paper"
  Found: arxiv.org/abs/2305.16291
  Downloaded: voyager_lifelong_learning.pdf

$ lightning run presentation_slide_writer "List slides"
  12 slides in current presentation...
```

`lightning list` — show all agents in the registry

`lightning run <agent>` — invoke any agent with natural language

10+ agents all from one `agents.json` file

The Meta Moment 🤯

We used `paper_researcher` to find the **Voyager paper**

Voyager inspired the **architect pattern**

We built `tool_architect` to create **more tools**

`presentation_slide_writer` built **THIS presentation**

Agents all the way down: Architect → Slide Writer → This Slide

⚡ Lightning Agents ⚡

Questions?

Agents creating **agents** creating **tools**

Built with `Claude Agent SDK` + `MCP`

📦 **This project:** `github.com/RPirruccio/lightning-agents`

🤝 **AIMUG community:** `github.com/aimug-org/austin_langchain`