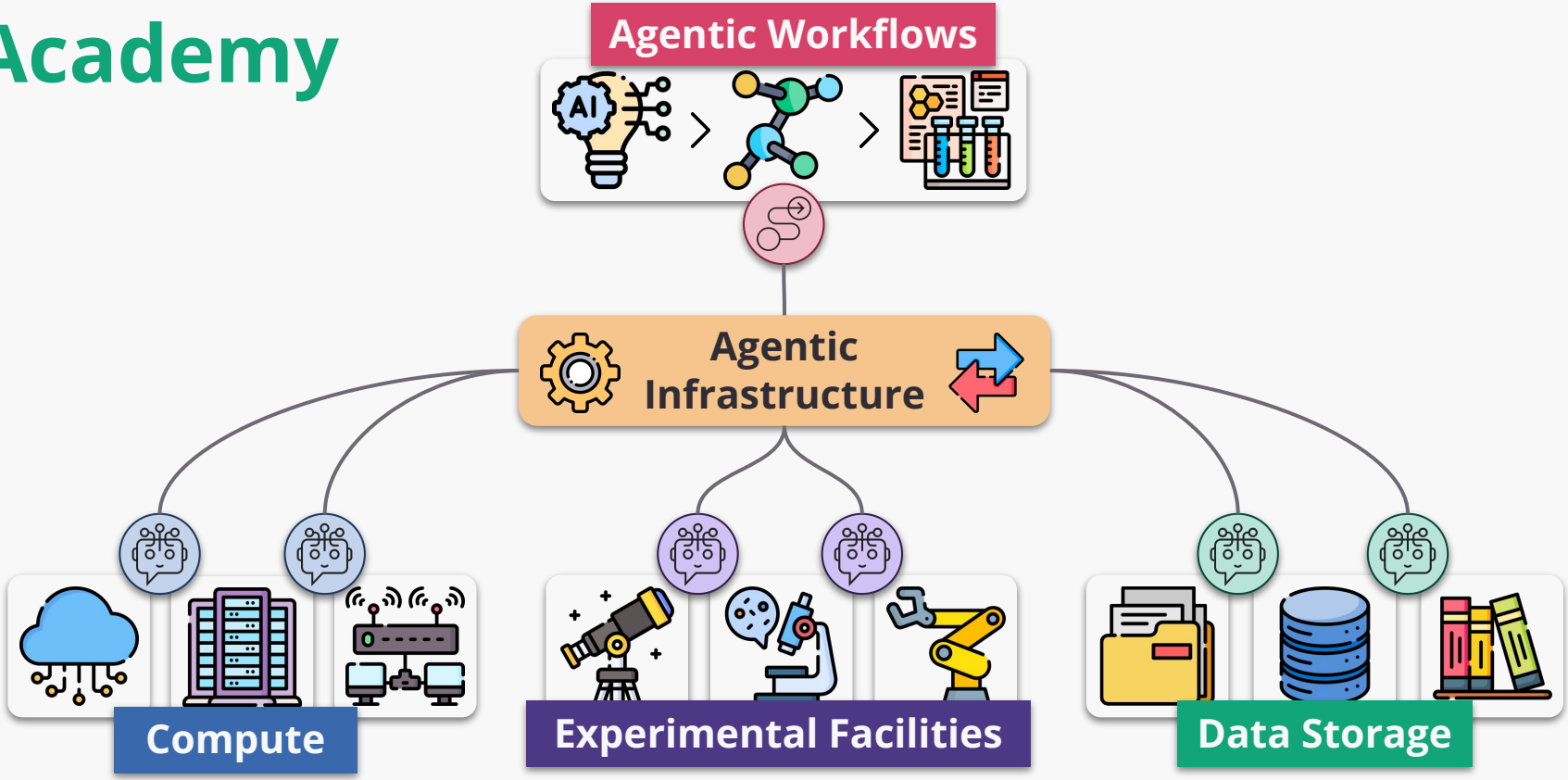
Academy: Empowering Scientific Workflows with Federated Agents

Presented by: Alok Kamatar

Workflows Community Initiative

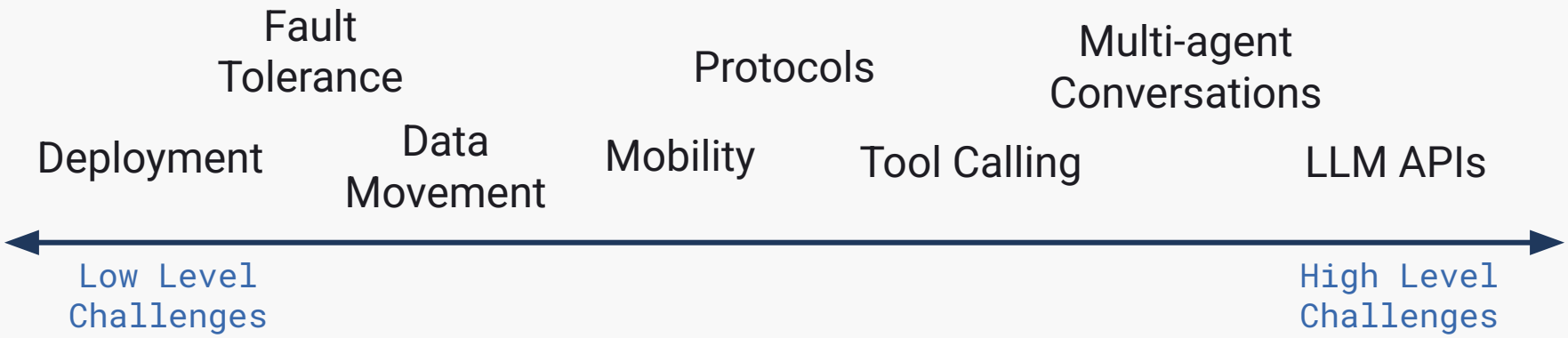
Academy


Agentic Workflows
AI
Agentic Infrastructure
Compute
Experimental Facilities
Data Storage

Agentic Middleware

Software layer that transparently manages the lifecycle, communication, and coordination of autonomous agents across distributed computing environments.

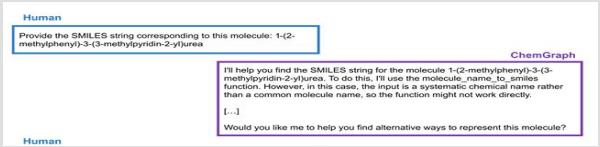
Agentic Middleware: Scope & Challenges



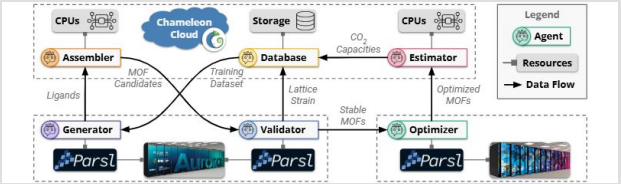
Academy → **Cool Science?** ← **LangChain, AutoGen, Pydantic AI, etc.**

Agentic Patterns Beyond LLMs

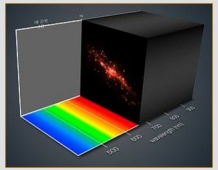
Conversational Assistants



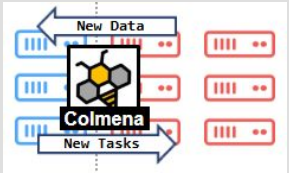
Multi-Site Applications

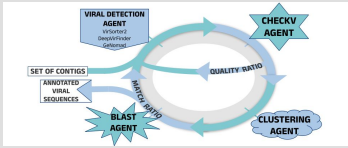


Integrating Instruments/Experiments

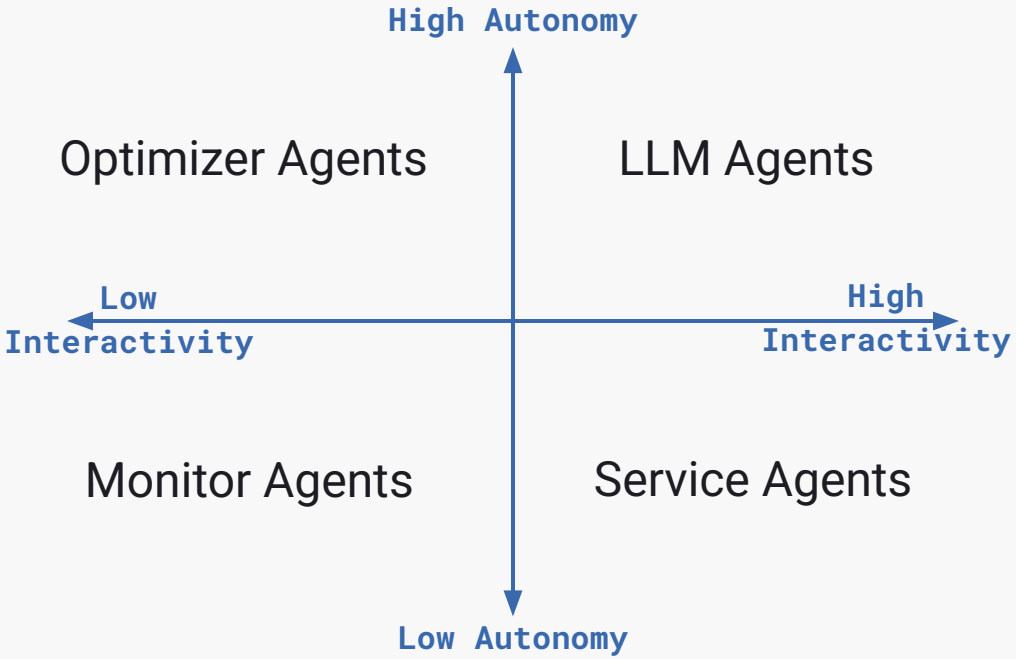


Steering Workflows





Agentic Middleware: Agent Behaviors



Other defining aspects:

- Persistent vs ephemeral
- General vs narrow purpose
- Embodiment

Long-running agentic science apps will incorporate many kinds of agent behaviors.

Academy primitives support the creation diverse agent types.

Agentic Middleware: Using Research Infrastructure

Centralized

- Agents co-located (workstation, cloud)
- Research infrastructure available via APIs (REST, SDKs, MCP Servers, …)
- Use infrastructure via tool calling

++ Rapidly growing library ecosystem
−− Limited APIs for infrastructure

LangChain, AutoGen, Pydantic AI, etc.

Decentralized

- Agents distributed across infrastructure
- Agents interact asynchronously
- Use infrastructure directly (actuate a robot, submit job, …)

++ Data locality, perf., loose coupling
−− Deployment complexity

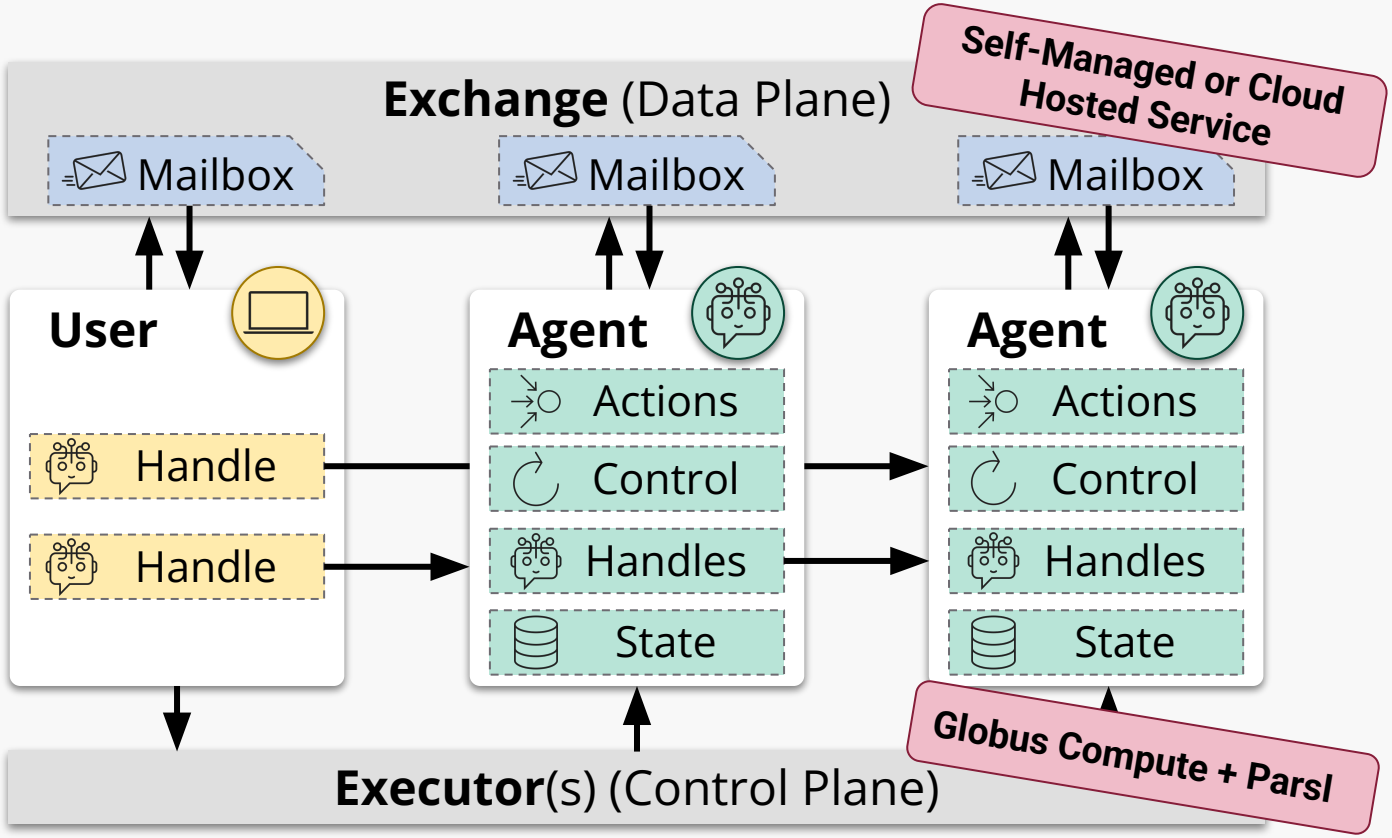
Academy

How does **Academy** support the expression of **diverse agent behaviors** and deployment across **distributed/federated resources**?

Focus 3: Coordinate async agent messaging

Focus 1: Program diverse agents and interactions

Focus 2: Deploy agents on federated resources



https://docs.academy-agents.org/latest/concepts/

Communication & Execution

Exchange

- ➔ Asynchronous communication through mailboxes
- ➔ Every agent/client in system has a unique mailbox
- ➔ Local & distributed implementations
- ➔ Optimized for low-latency
- ➔ Hybrid communication model
- ➔ Prefer direct communication between agents when possible; fall back to indirect communication via object store
- ➔ Pass-by-reference with ProxyStore for large data
- ➔ Authentication with Globus

Writing Apps in Academy

Agents defined by a behavior

Clients & other agents can request actions

```python
import asyncio
from academy.agent import Agent, action, loop

class Example(Agent):
    def __init__(self) -> None:
        self.count = 0  # State stored as attributes

    @action
    async def square(self, value: float) -> float:
        return value**2

    @loop
    async def count(self, shutdown: asyncio.Event):
        while not shutdown.is_set():
            self.count += 1
            asyncio.sleep(1)
```

Instance of a behavior is state

Control loops for autonomous behavior

https://academy.proxystore.dev/latest/get-started/

Single interface for managing your agents

Launch agent and get handle

Interact with agents via handles

```python
from academy.exchange.hybrid import HybridExchange
...
from academy.manager import Manager

gce = GlobusComputeExecutor('<UUID>')

async with await Manager.from_exchange_factory(
        factory=LocalExchangeFactory(),
        executors=gce,
) as manager:
    behavior = Example()  # From the prior slide
    handle = await manager.launch(behavior)

    result = await handle.square(2)
    assert result == 4

    await handle.shutdown()
```

Launch agents via Globus Compute

Pass handles to other agents

https://academy.proxystore.dev/latest/get-started/

Integrating Academy and LLMs

```python
from academy.handle import Handle
from langchain_core.tools import tool

def make_sim_tool(handle: Handle[Simulator]):
    @tool
    async def compute_property(smiles: str) -> float:
        """Compute molecule property."""
        return await handle.compute_property(smiles)
    return compute_property

tool = make_tool(agent_handle)
print(tool.args_schema.model_json_schema())
```

Turn agent handles into LLM framework “tools”

Comparison to Alternatives

Tool Servers (MCP)
Rely on externally reachable endpoints that are blocked by facility policies. Requires user to manage services, infrastructure, and VPNs

Func-as-a-Service (Globus Compute)
Easier remote execution (no VPN, infrastructure management) but tools must be stateless, short-running tasks

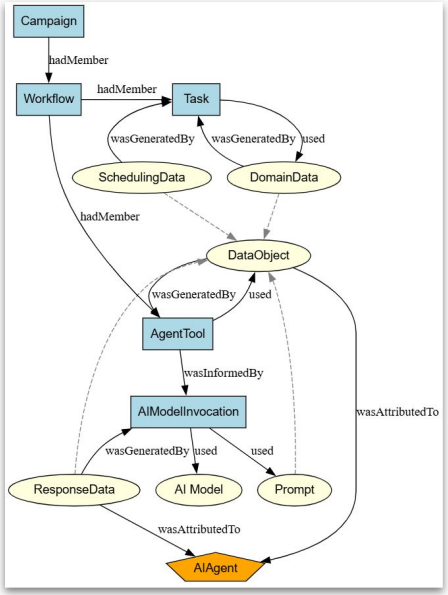
Provenance with Flowcept (On going)

- ➔ Provenance extensions for agentic workflows
- ➔ Captured via Code Annotation
- ➔ Leverage Tooling
  - ◆ Grafana
  - ◆ FAIR queries
  - ◆ Notebook exploration
  - ◆ LLM Interaction

```python
from academy.agent import Agent, action
from flowcept import flowcept_agent_tool

class Example(Agent):

    @action
    @flowcept_agent_tool
    async def square(self) -> float:
        return value**2
```



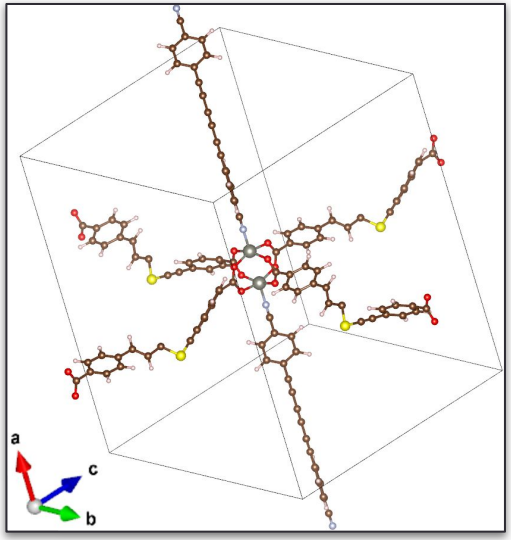
PROV-AGENT: Modeling Agentic AI Concepts with W3C PROV (2508.02866)

Use Case: **MOF Discovery**

Metal Organic Frameworks (MOF)

- ➔ Composed of organic molecules (ligands) and inorganic metals (nodes)
- ➔ The sponges of materials science!
- ➔ Porous structures that adsorb and store gases
- ➔ Topologies can be optimized for targeted gas storage → **Carbon Capture**

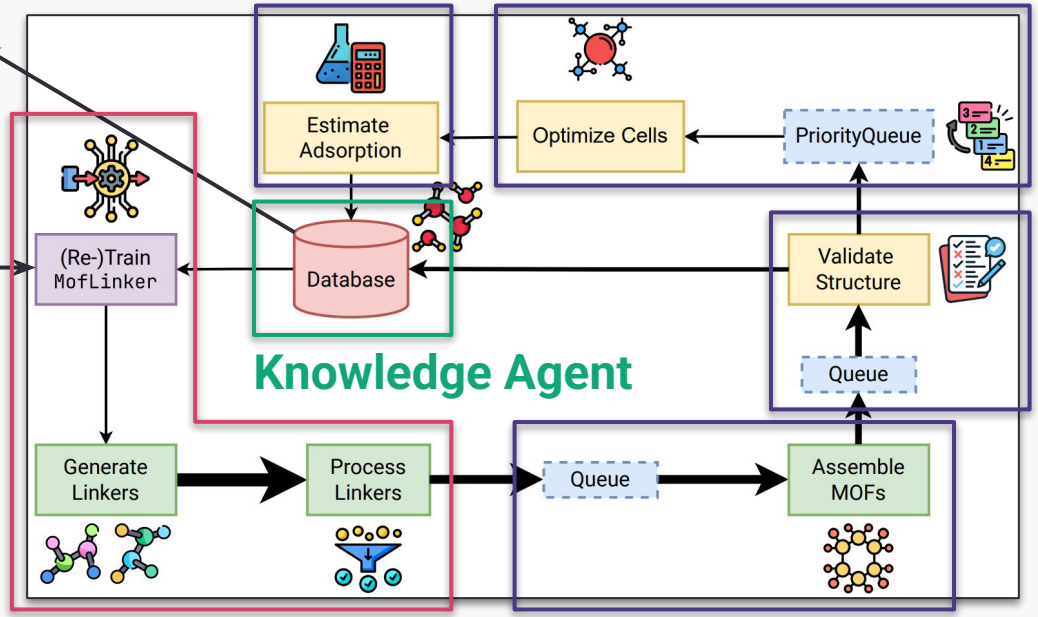
How to efficiently discover MOFs with desirable properties for target applications?



Intractable search space of ligand, node, & geometry combinations

MOFA: Online learning + GenAI + Simulation

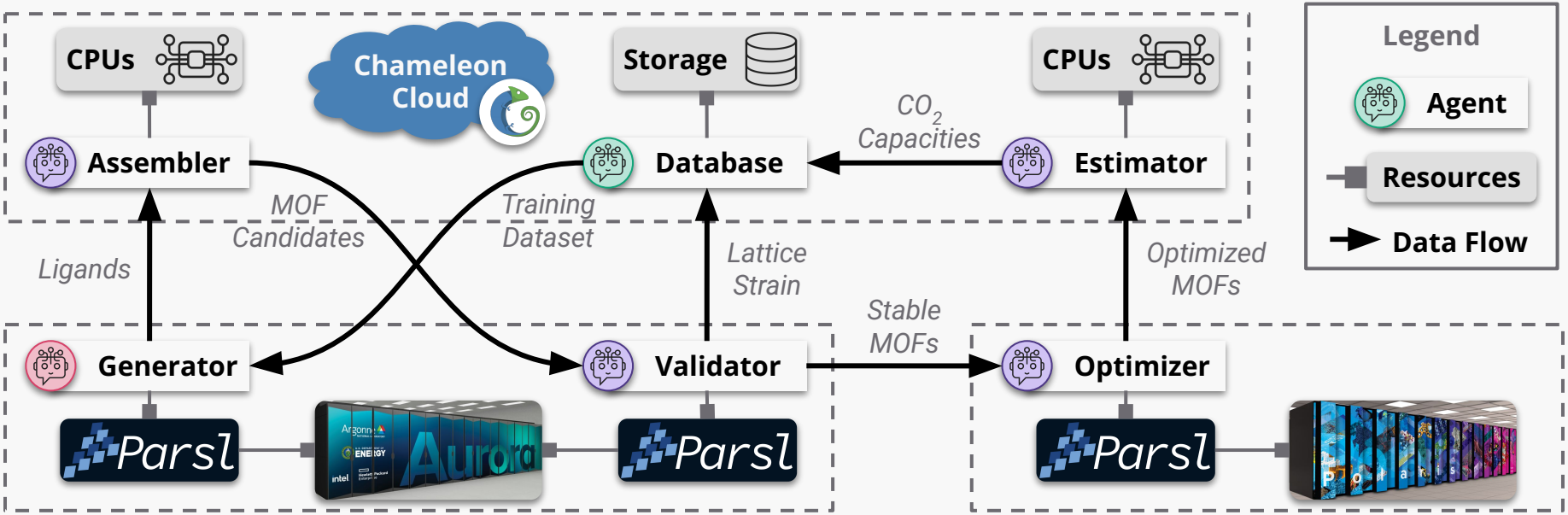
Embodied Agents*



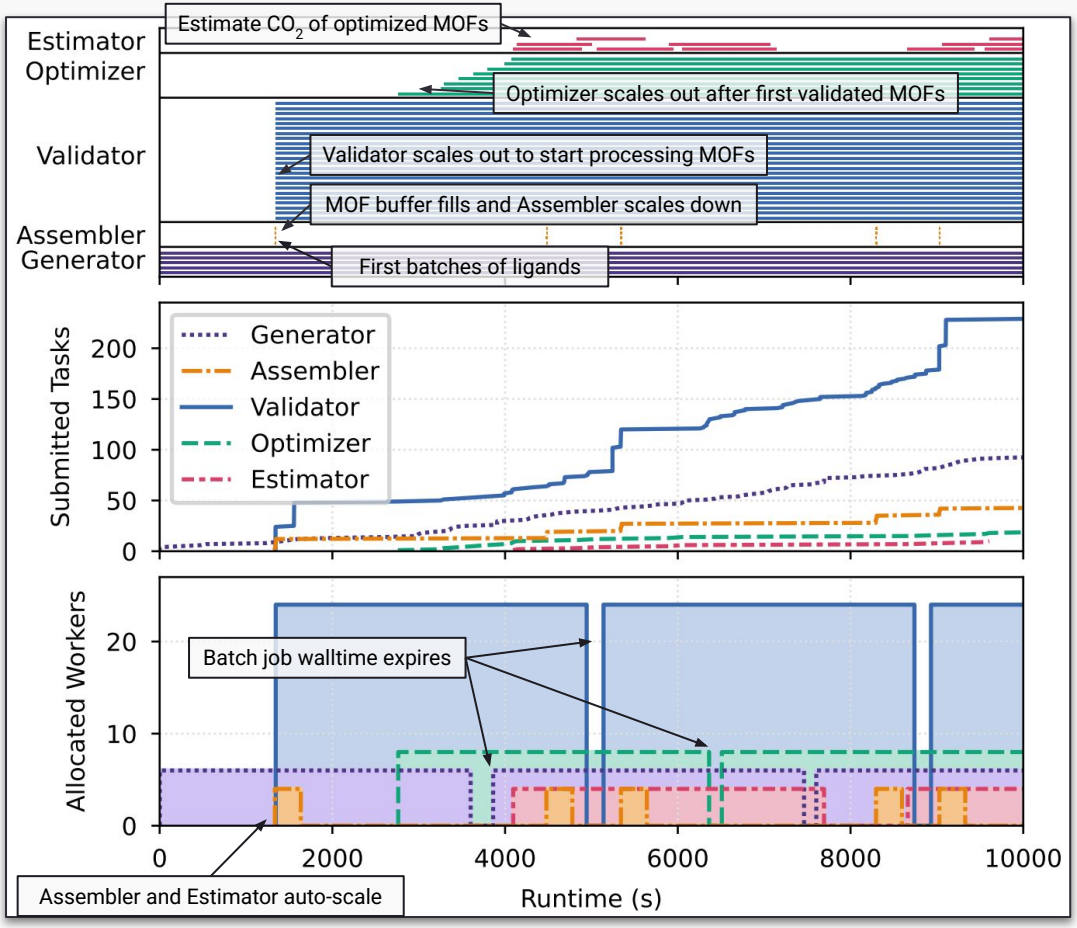
Computational Agents

Yan et al., "**MOFA: Discovering Materials for Carbon Capture with a GenAI- and Simulation-Based Workflow**" (Under Review)

MOFA through Autonomous Agents



Agents executed remotely via Globus Compute



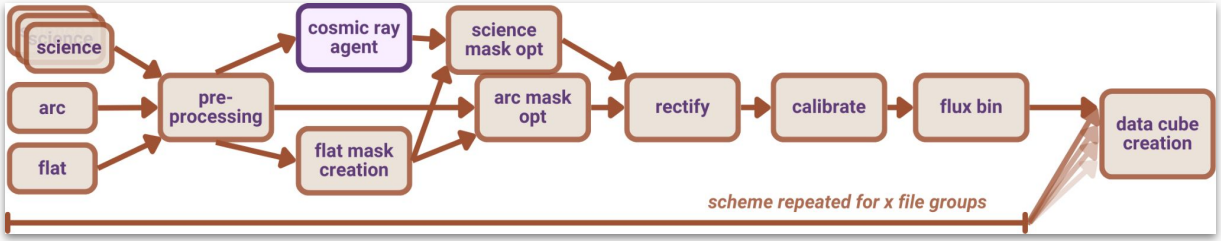
MOFA Agents Trace

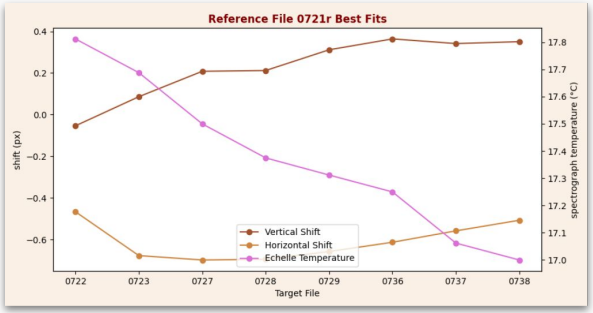
Why is this agentic model better?

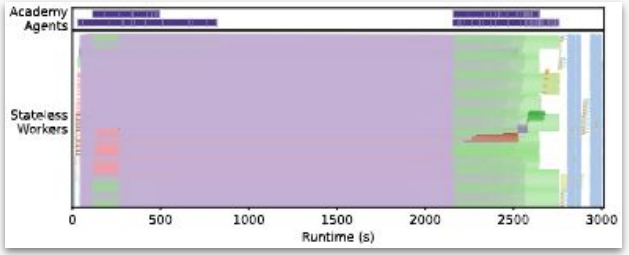
- ➔ **Placement:** Move agents to resources
- ➔ **Separation of concerns:** Resource acquisition and scaling based on local workload
- ➔ **Loose coupling:** Swap agents or integrate new agents (e.g., SDL)
- ➔ **Shared agents:** Multiple workflows can share agents (microservice-like)

Use Case: **Integral Field Unit Spectroscopy**

- ➔ Highly-sensitive to instrument calibration
- ➔ Optical parameters are **continuous in time**
- ➔ Resolution and speed can be improved using **stateful processing**







Questions?


J. Gregory **Pauloski**


Yadu **Babuji**


Ryan **Chard**


Alok **Kamatar**


Mansi **Sakarvadia**


Kyle **Chard**


Ian **Foster**

Daniel **Babnigg**

Renan **Souza**

Daniel **Rosendo**

Amal **Gueroudji**

Rafael **Ferreira da Silva**

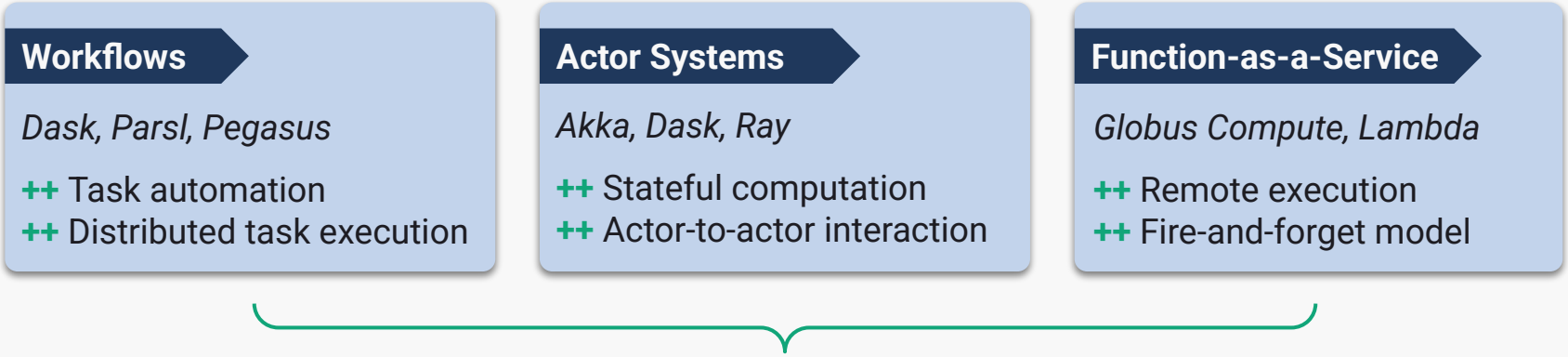
Reach out: alokvk2@uchicago.edu

Learn more: Academy-Agents.org

- ⭐Academy @ github.com/academy-agents
- Cite the paper: arxiv.org/abs/2505.05428v2
- Join Slack
- Follow the tutorial
- Get Started!

```
pip install academy-py
```

What does the middleware look like?



- *Fire-and-forget*: Agents spawned across remote/federated resources
- *Autonomy*: Agents have agency over their actions and local state/resources
- *Cooperative*: Agents interact to execute tasks & workflows

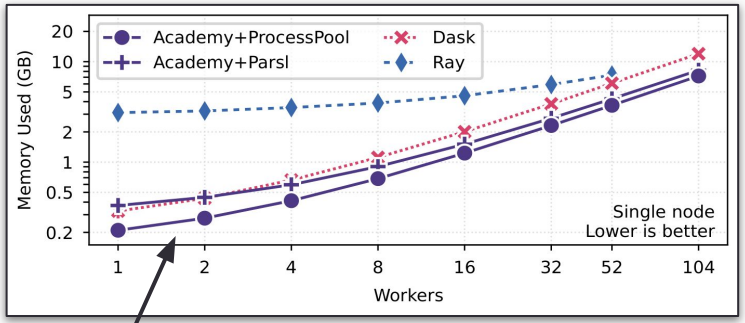
Features (rapid fire)

- ➔ Any number of actions & control loops
- ➔ Special purpose control loop decorators
- ➔ Multi-threaded/non-blocking action execution
- ➔ Startup and shutdown callbacks
- ➔ State persistence plugins
- ➔ Re-execution on failure
- ➔ Agents can launch other agents
- ➔ Discovery/lookup based on behavior
- ➔ Handle mailbox multiplexing

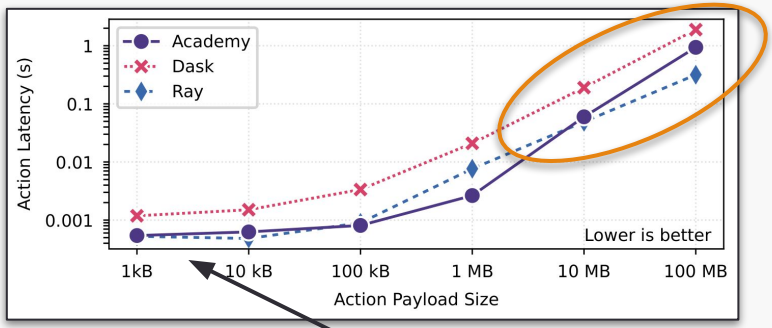
Any interesting? Ask about them at the end!

Comparisons to Actor Systems

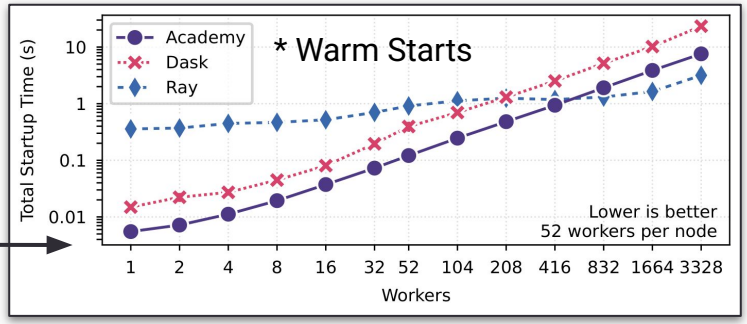
Why we need ProxyStore!



Low-memory overhead



Low-latency messaging



Fast start-up

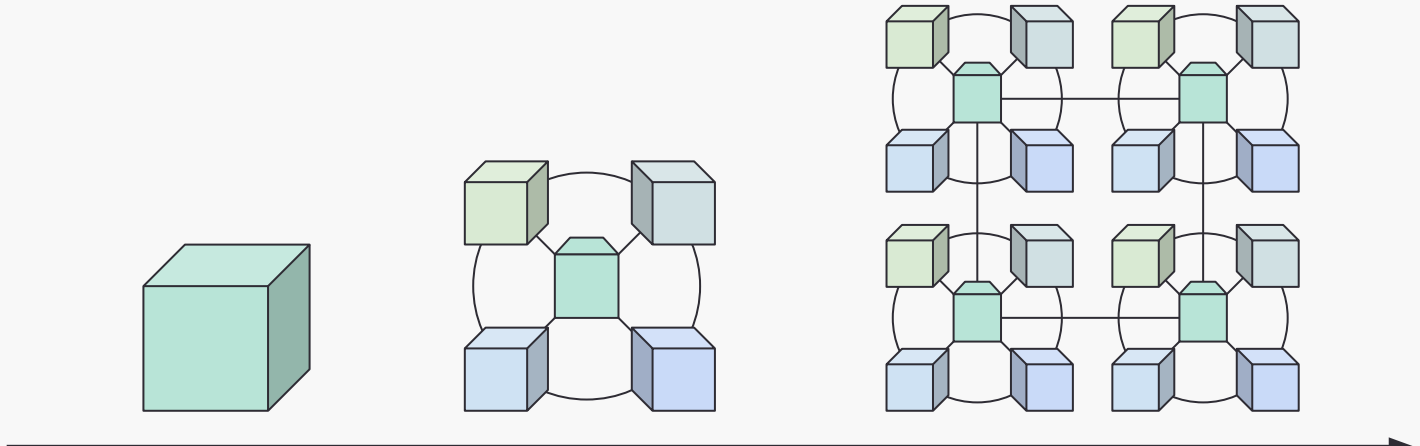
Experiments performed on Aurora @ ALCF

Supplemental

Autonomous discovery “harnesses the power of robotics, ML, and AI to **solve big problems** [...] **faster than ever before**.”

Credit: ANL, “**Science 101: Autonomous Discovery**”

Challenge 1: Complexity is a Barrier



Middleware & infrastructure is not ready for automating discovery

Challenge 2: Humans are a Bottleneck

Humans synthesize knowledge and propose hypotheses

Humans write, debug, and run programs

Humans interpret results to inform new hypotheses

} Inefficient use of research infrastructure

Agents can be the driving entities

- ➔ Persistent, stateful, cooperative
- ➔ Intermittent human oversight

} We need to be here

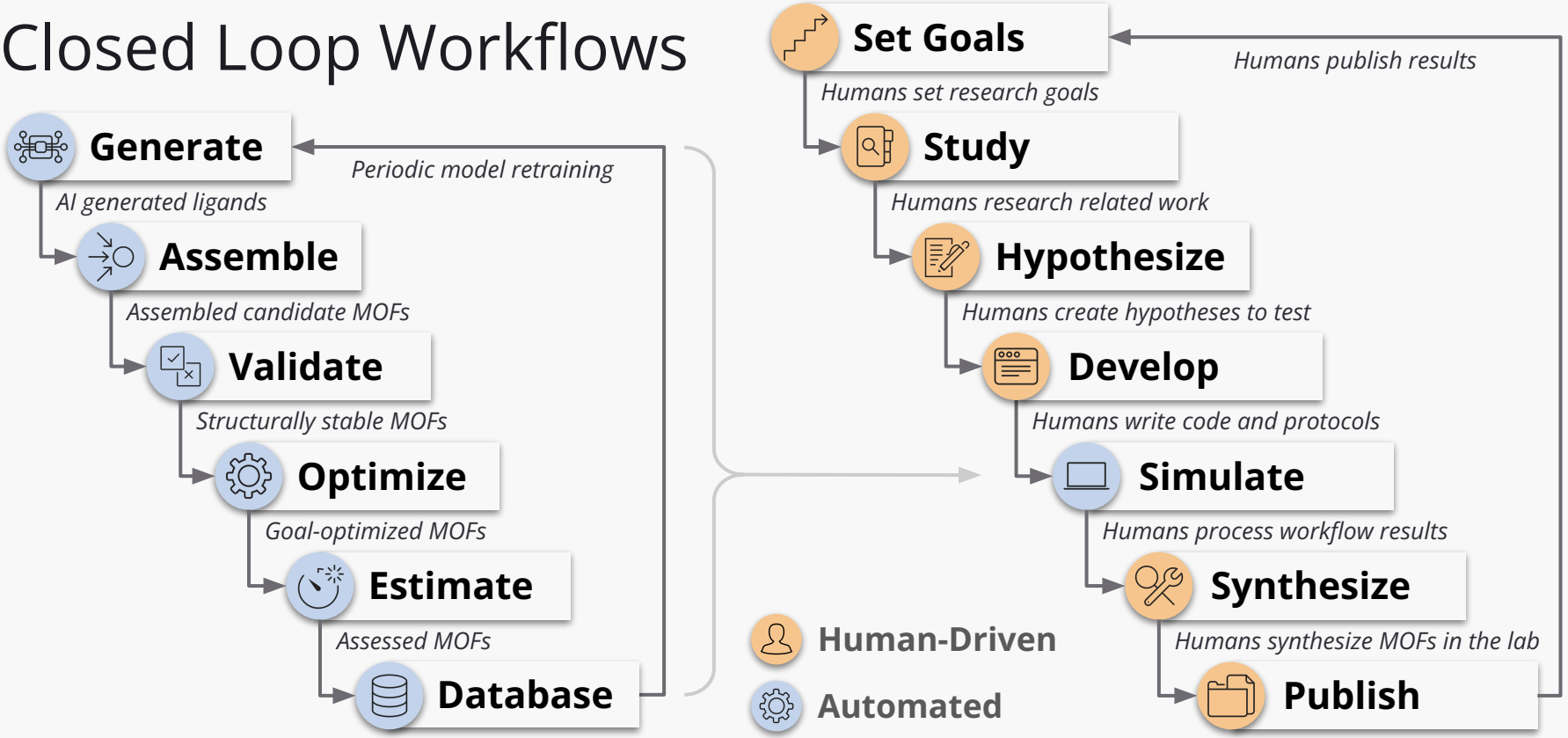
Credit: Ian Foster, "**Empowering Science with Intelligent Middleware and Embodied Agents**"

Solution: Multi-Agent Systems for Science

- ✓ Automate closed-loop processes
- ✓ Natural expression of scientific resources (compute, instruments, repositories)
- ✓ Operate autonomously but still cooperatively
- ✓ Execute multi-stage computational science processes
- ✓ Reduce mundane task responsibilities of scientists

The whole is greater than the sum of its parts.
- Aristotle

Closed Loop Workflows




Publish
Store and disseminate results in the form of knowledge
Report
Question
Objective
Given a high-level goal, derive questions or pose conjectures
Planning
Manage trade offs & resources
Analysis
Discover trends, improve models, & interpret results
Analyze
Enforcement
Ensure safety & validity
Study
Knowledge
Gather relevant information and learn from results
Exploration
Navigate avenues for discovery
Service
Perform simulations, experiments and make observations
Experiment
Hypothesize
Prediction
Generate testable hypotheses from current knowledge

Related Efforts

Actor Model

- ➔ Model for concurrent computing
- ➔ Message passing & local state
- ➔ Basic building block for agents

Supported in many systems:

- ✓ Azure Actors, Dask, Ray, etc.
- ✗ Autonomous agents
- ✗ Execute on federated resources

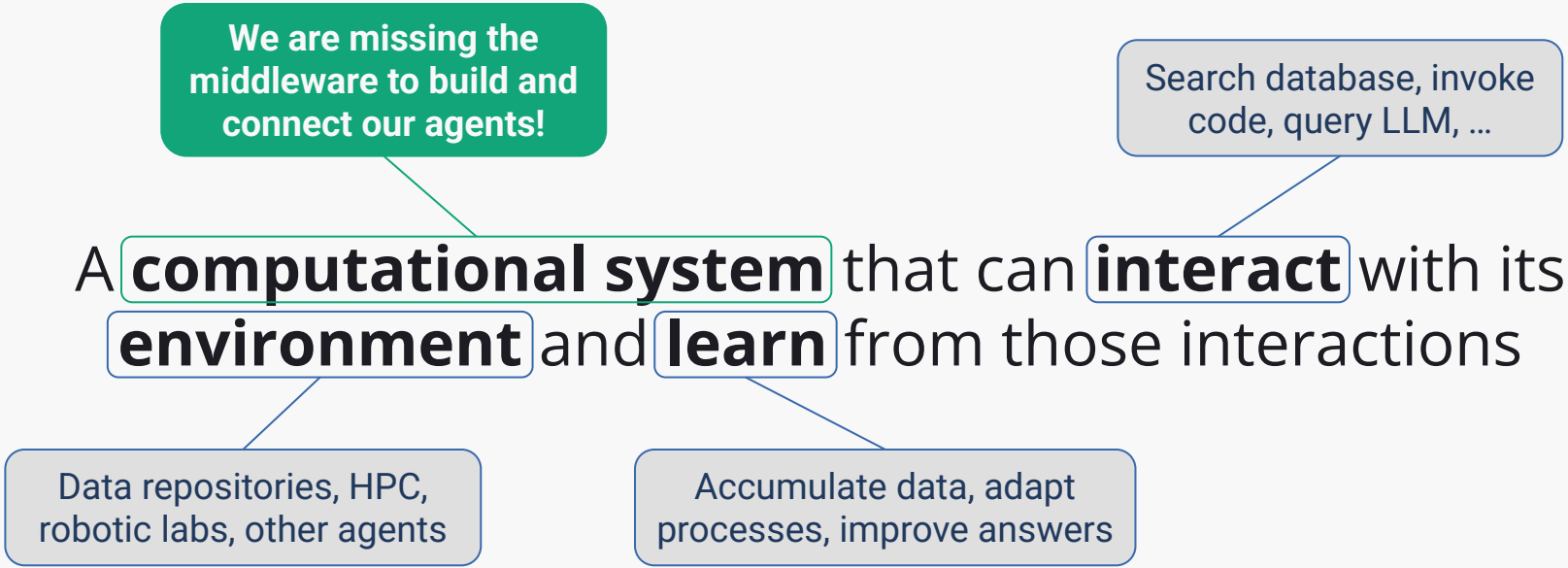
LLM Agents

- ➔ For multi-agent LLM conversations
- ➔ Agents assume roles & abilities
- ➔ Better reasoning and action chains

Supported in many systems:

- ✓ AutoGen, LangChain, OpenAI, etc.
- ✓ Call external tools
- ✗ Limited scope (LLM-based apps)
- ✗ Distributed execution

How do we build agents?

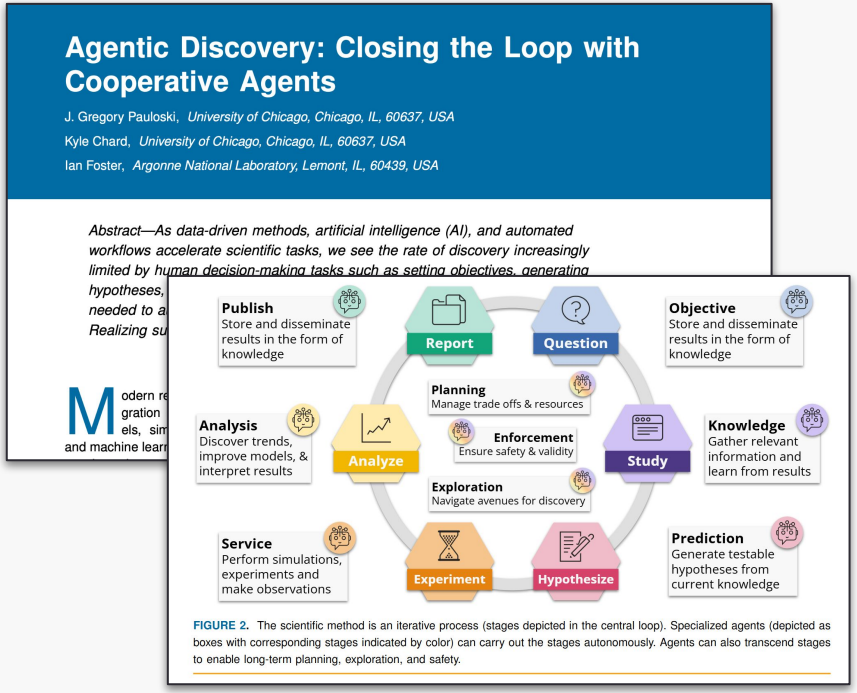


Credit: Ian Foster, "**Empowering Science with Intelligent Middleware and Embodied Agents**"

Middleware Open Challenges

- ➔ Access & privileges
- ➔ Agent discovery
- ➔ Asynchronous communication
- ➔ Fault tolerance
- ➔ Interfaces
- ➔ Mobility
- ➔ Persistent stateful execution
- ➔ Provenance
- ➔ Many more…

Areas we focused on…



Agentic Discovery: Closing the Loop with Cooperative Agents

J. Gregory Pauloski, *University of Chicago, Chicago, IL, 60637, USA*
Kyle Chard, *University of Chicago, Chicago, IL, 60637, USA*
Ian Foster, *Argonne National Laboratory, Lemont, IL, 60439, USA*

Abstract—As data-driven methods, artificial intelligence (AI), and automated workflows accelerate scientific tasks, we see the rate of discovery increasingly limited by human decision-making tasks such as setting objectives, generating hypotheses, needed to a Realizing su

odern re gration els, sim and machine learn

FIGURE 2. The scientific method is an iterative process (stages depicted in the central loop). Specialized agents (depicted as boxes with corresponding stages indicated by color) can carry out the stages autonomously. Agents can also transcend stages to enable long-term planning, exploration, and safety.

Under review in IEEE Computer

Autonomous Discovery Workflow Requirements

Express many kinds of agents
Autonomous, Intelligent, Embodied, Distributed, Mobile

✗ SOTA frameworks target single agent type

Execute agents in many places
Near compute or data

✓ P2P inter-agent messaging
✗ Remote launch & management
✗ Supported by SOTA systems