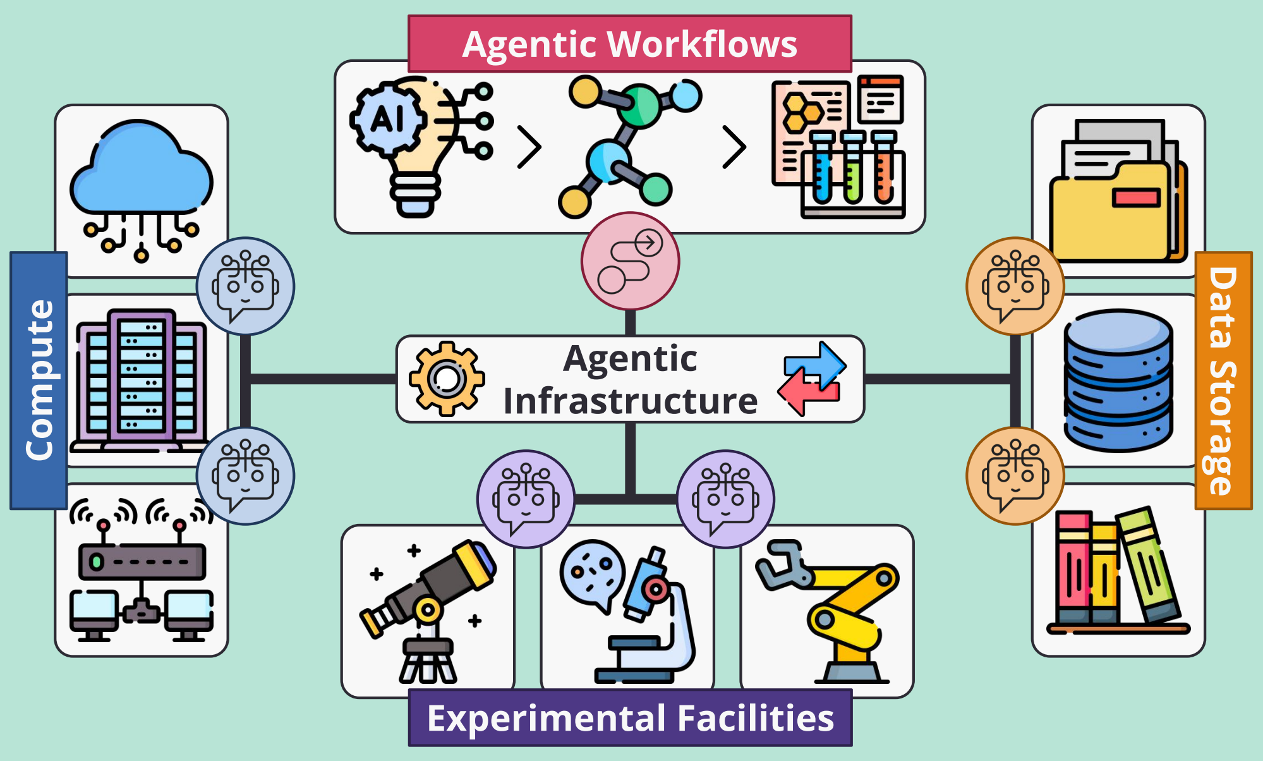
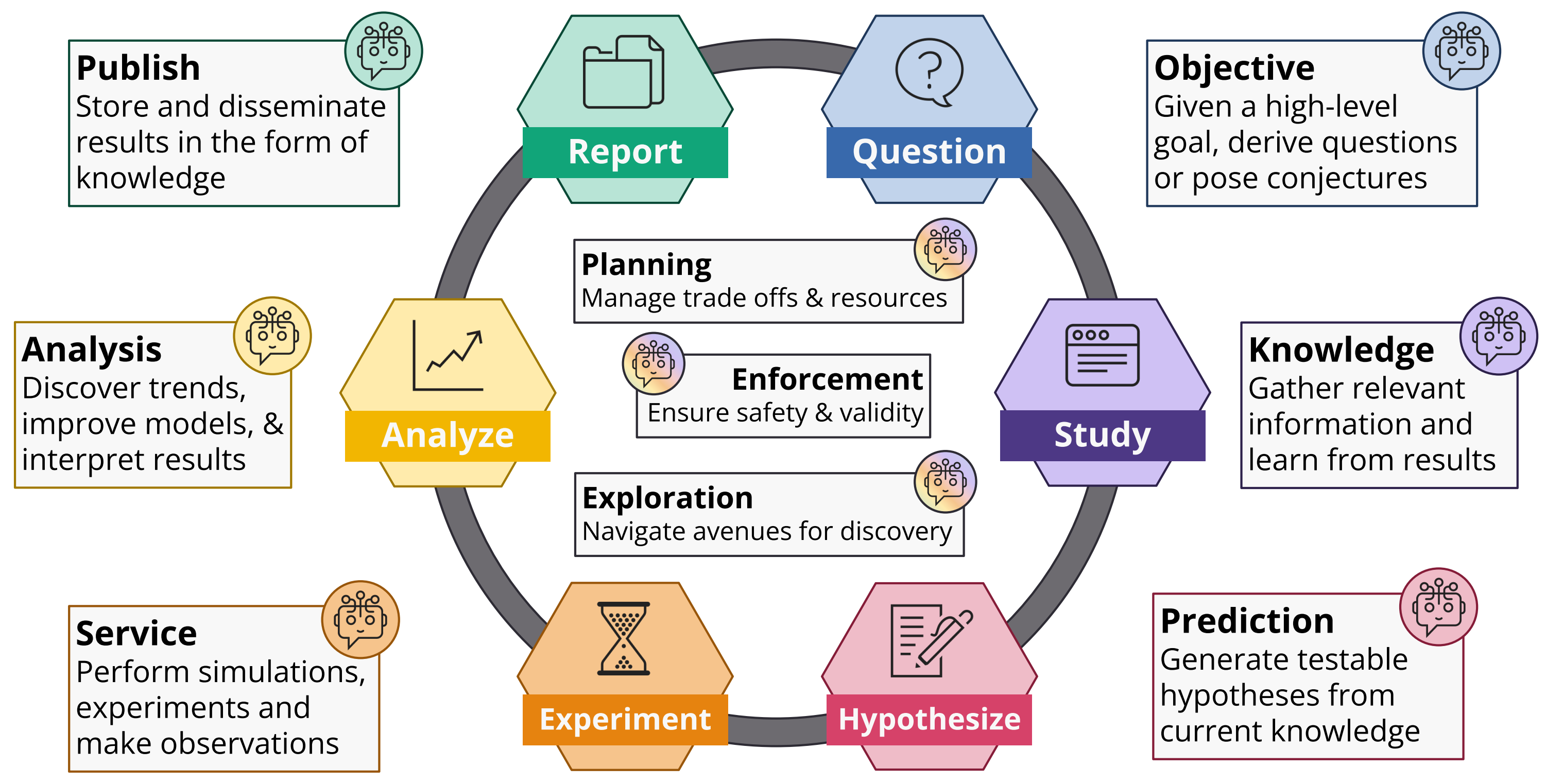
Empowering Scientific Workflows with Federated Agents

J. Gregory Pauloski, Yadu Babuji, Ryan Chard, Kyle Chard & Ian Foster

Abstract

Agentic systems are exploding in popularity in the AI community; however, existing frameworks do not integrate well with research infrastructure (RI). Academy is a middleware for deploying autonomous agents across federated RI (HPC clusters, experimental facilities, and data repositories). Academy provides abstractions for expressing stateful agents, managing asynchronous agent coordination, and distributed deployment across heterogeneous resources.




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

Agentic Discovery: Closing the Loop with Cooperative Agents

Federations of cooperative agents—deliberative, reactive, embodied, and more—will augment, and often replace, the human-in-the-loop in scientific endeavors. This prediction originates from two observations: human decision making tasks limit the rate of discovery and advancements in agentic systems are converging to a point where the complete scientific method (described above in the center loop) can be carried out autonomously by specialized agents.

Background

An **agent** is a program that can perform actions independently or semi-autonomously on behalf of a client or other agent.

Extension of **actor** model: local state and communication via asynchronous message passing.

Agent defined by **actions** it can perform, **control loops** that define autonomous behavior, and **local state**.

Agents come in many forms: deliberative/intelligent, reactive/observer, service, resource, embodied, learning, composite.

Multi-agent systems enable complex problem solving and emergent behavior through the cooperation of simple, specialized agents.

Academy: Build & Deploy Stateful Agents

What is the goal? A modular and extensible middleware providing primitives necessary to build and deploy arbitrary autonomous agents across federated research infrastructure.

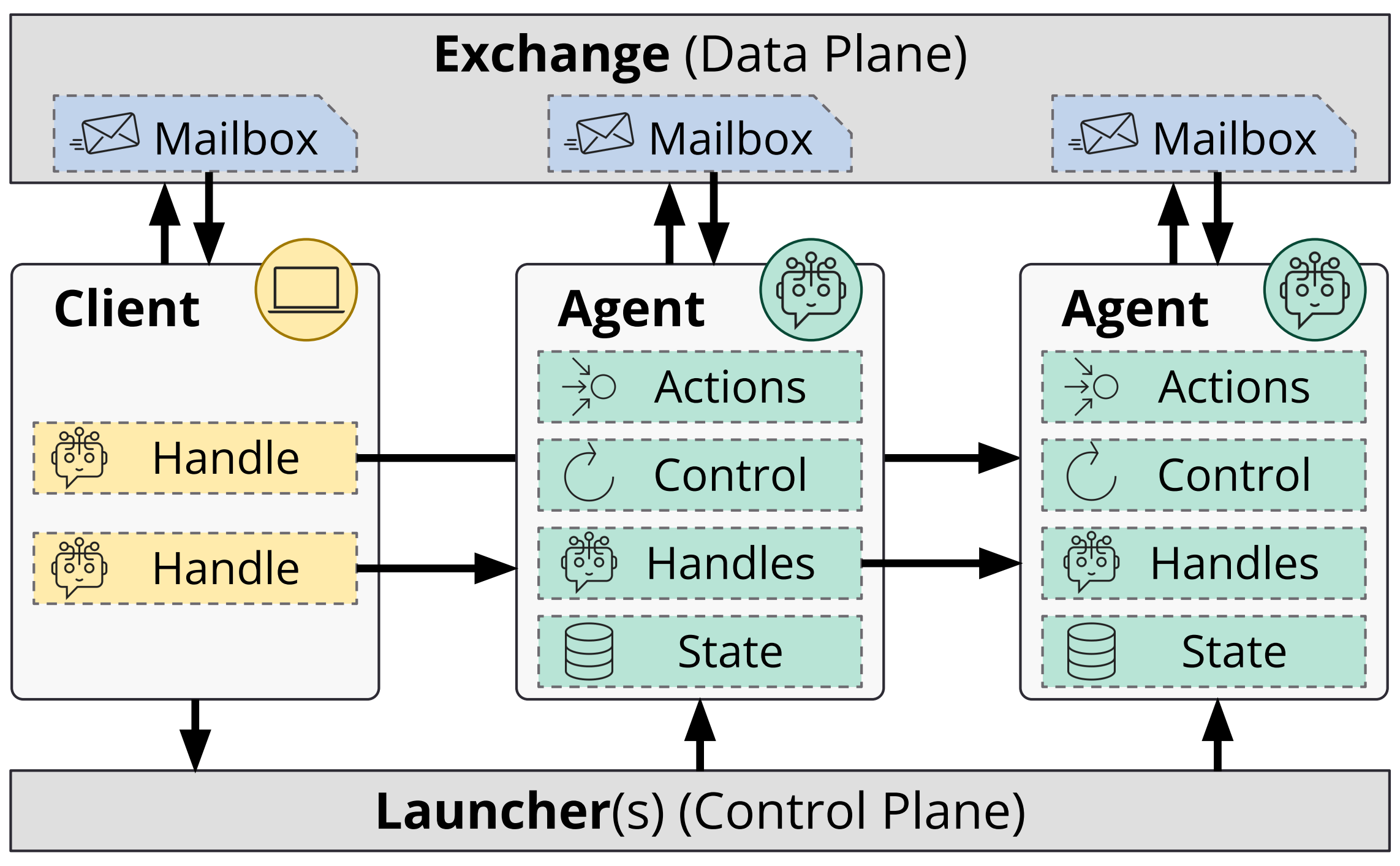
How do I write agents? Python classes (below) with methods decorated as *actions* that other agents can request and *control loops* that enable autonomous behavior and interaction.

How do I launch agents? In a local thread- or process-pool, distributed across a cluster with Parsl, or on federated resources with Globus Compute.

How do clients and agents communicate? Handles are references to remote agents that translate method calls into messages that are asynchronously communicated via mailboxes (unique per agent) managed by an exchange.

How is this different from AutoGen/LangChain/etc? Academy supports scientific applications (async & long-running execution, heterogeneous resources, high-throughput data flow, dynamic resource availability) whereas existing frameworks focus on improving reasoning capabilites of LLMs.

Academy Architecture



Agents and clients interact via ***handles*** *to invoke actions (async). Agent behaviors defined by* ***actions***, ***control loops***, *and* ***state***. *Academy decouples control and data planes through the* ***launcher*** *and* ***exchange,*** *which manage spawning agents and communication, respectively.*

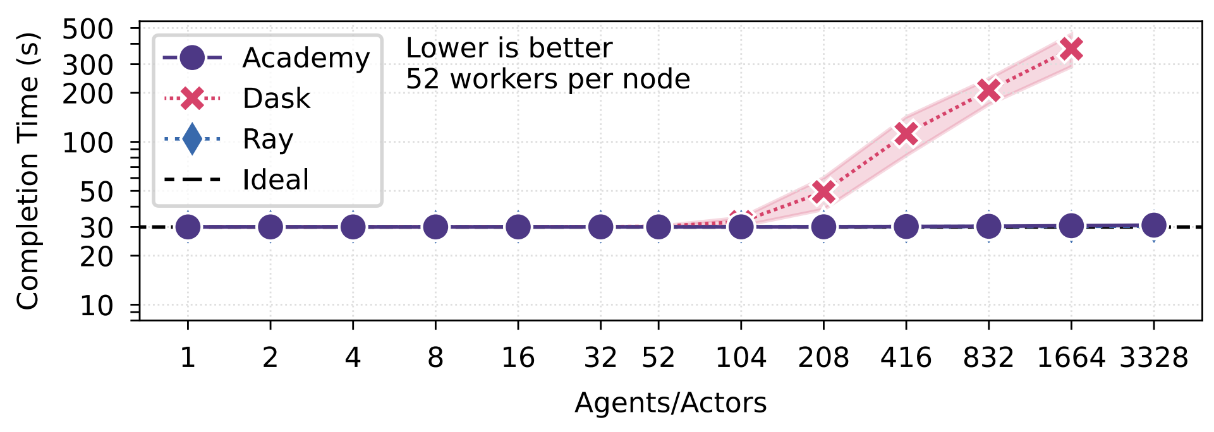
```python
import time, threading
from academy.behavior import Behavior, action, loop

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

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

    @loop
    def count(self, shutdown: threading.Event) -> None:
        while not shutdown.is_set():
            self.count += 1
            time.sleep(1)
```

← *Example agent behavior definition. Decorators mark special methods as agents or loops.*

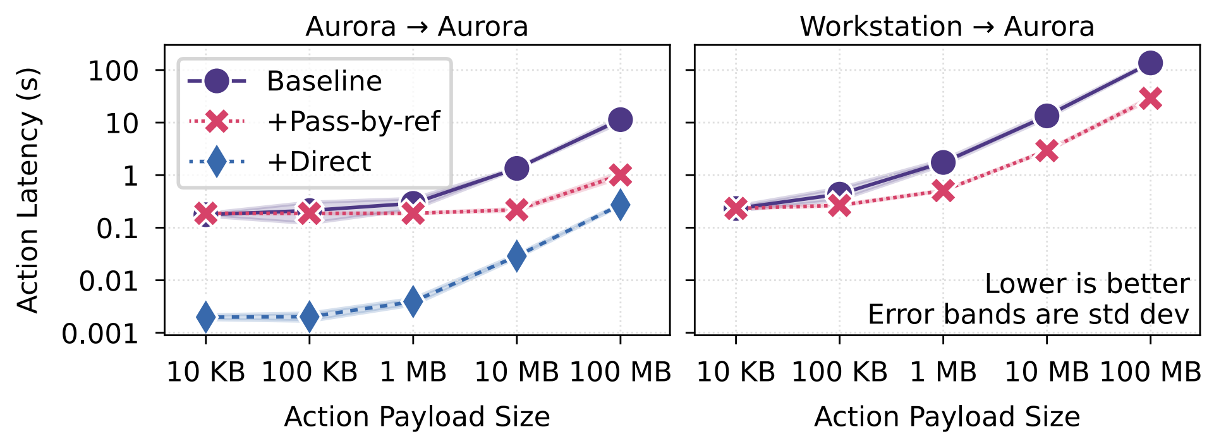
→ *Create and manage agents via the Exchange, Launcher, and Manager interfaces.*

```python
from academy.exchange.thread import ThreadExchange
from academy.launcher.thread import ThreadLauncher
from academy.manager import Manager

with Manager(
    exchange=ThreadExchange(),  # Can be swapped with
    launcher=ThreadLauncher(),  # other implementations
) as manager:
    behavior = Example()  # From Listing 1
    handle = manager.launch(behavior)

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

    handle.shutdown()  # Or via the manager
    manager.shutdown(handle.agent_id, blocking=True)
```

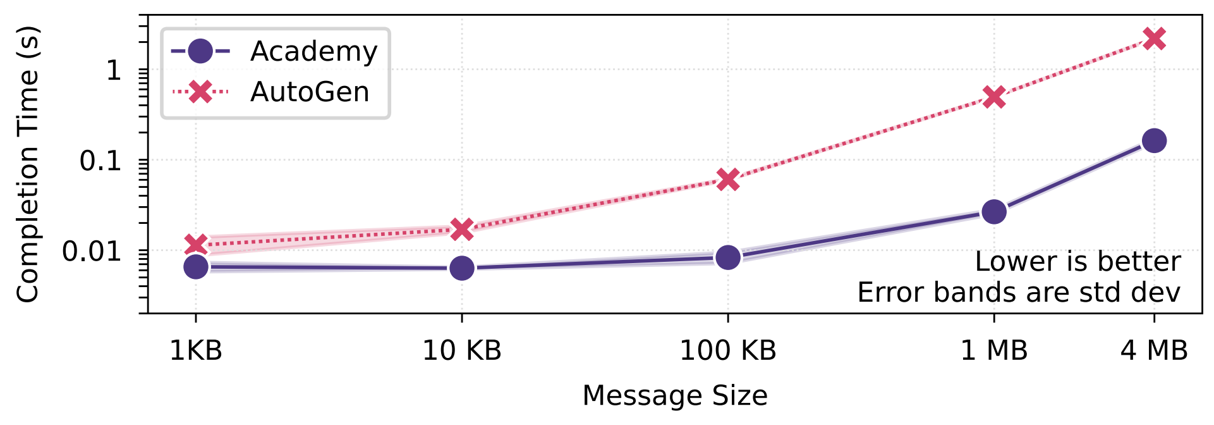
Academy Benchmarks



Academy scales to thousands of agents. Individual agents can handle thousands of requests per second—faster than Dask and competitive with Ray. (Weak scaling; 30 actions/agent; 1s sleeps)



Academy's distributed exchange is optimized for HPC and federated deployments. Pass-by-ref. with ProxyStore optimizes wide-area data transfer and direct messaging reduces latency between local agents.



Academy handles large data more efficiently than AutoGen's distributed gRPC runtime (limit 4MB messages).

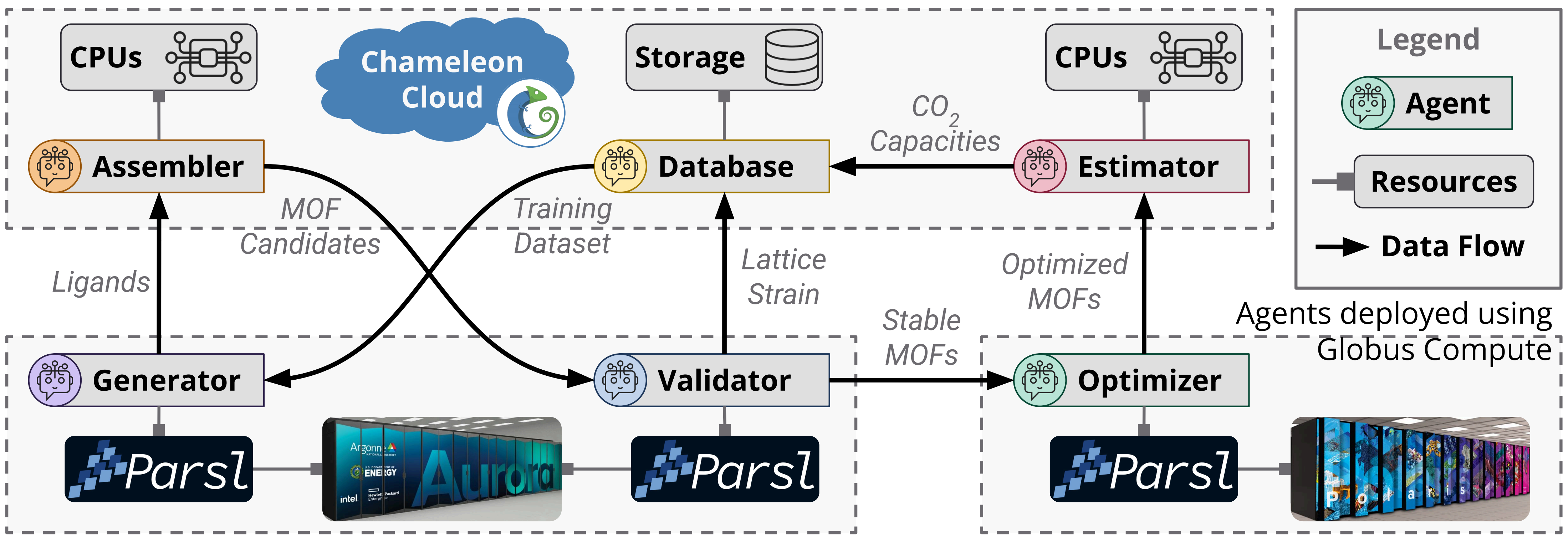
MOFA: GenAI Materials Discovery

Metal organic framework (MOF): porous polymers for gas storage
Goal: Discover high-performing MOFs for carbon capture apps
Methods: Combine generative AI and simulation to efficiently navigates the intractable combinatorial space of MOF structures:

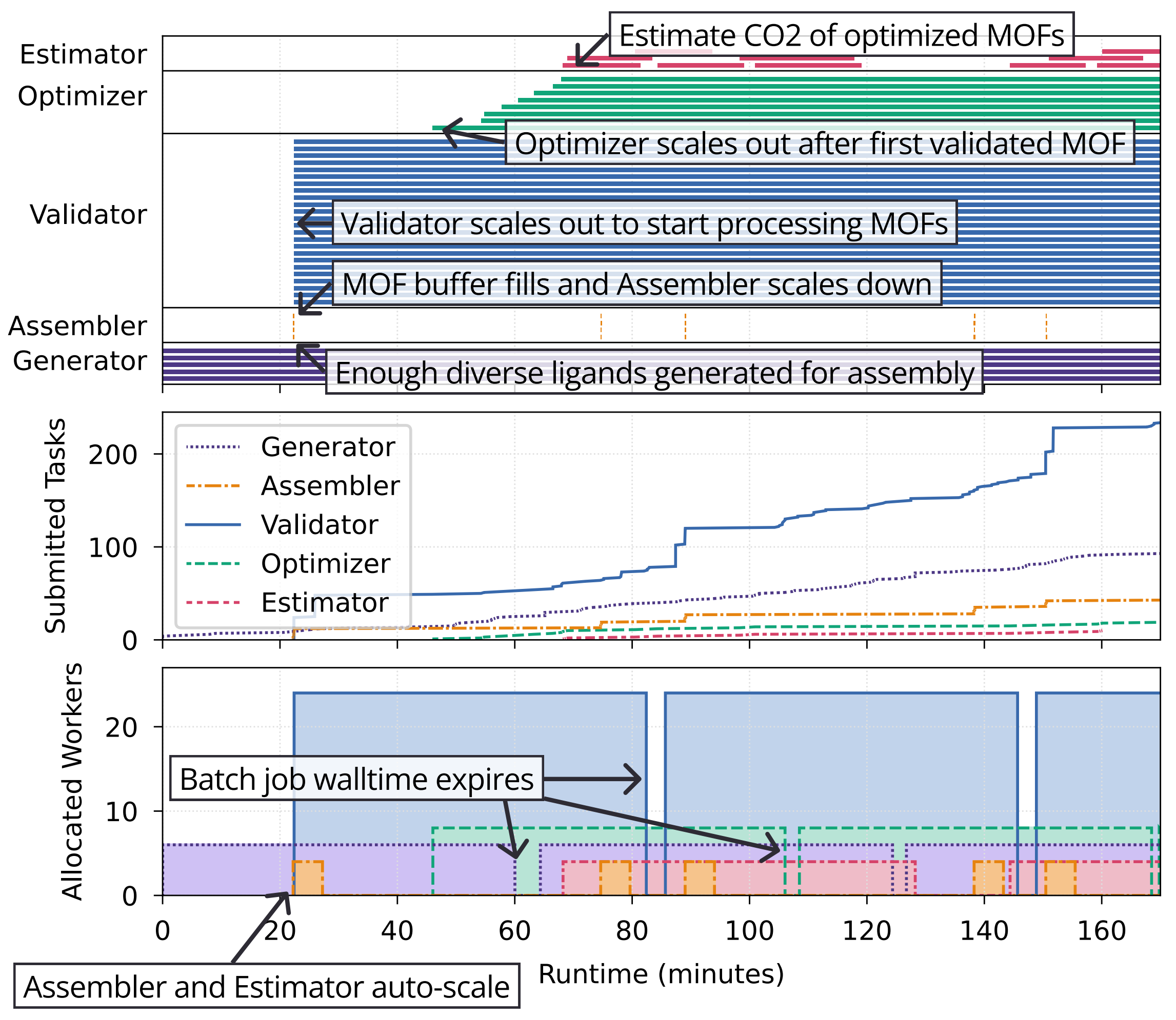
1. A generative AI model produces candidate ligands
2. Ligands are combined with metal clusters to assemble MOFs
3. MOFs are screened/validated using many MD simulations
4. CO2 adsorption is simulated and recorded to a database
5. GenAI model is periodically retrained on the accumulated results

Benefits of *Agentic* MOFA

- Agents deployed on the resource best-suited for the agent's responsibilities
- Agents scale resources in or out based on local state (i.e., workload)
- Loose coupling == easy to swap agent implementations w/ same behavior
- Easier integration of new agents. E.g., embodied agent in self-driving labs that synthesizes and evaluates best MOFs



Agents deployed using Globus Compute



Execution trace over three hours. (Top) Active tasks per agent. The y-axis height represents the maximum size of the resource pool allocated by each agent (i.e., CPUs or GPUs). Assembler tasks are short and infrequent. (Middle) Cumulative tasks submitted per agent. (Bottom) Active workers allocated in each agent's resource pool. Worker allocations vary with demand (as in Assembler and Estimator) or batch job wall times (as in Generator, Validator, and Optimizer).

THE UNIVERSITY OF CHICAGO
Argonne NATIONAL LABORATORY
globus labs
jgpauloski@uchicago.edu
github.com/proxystore/academy