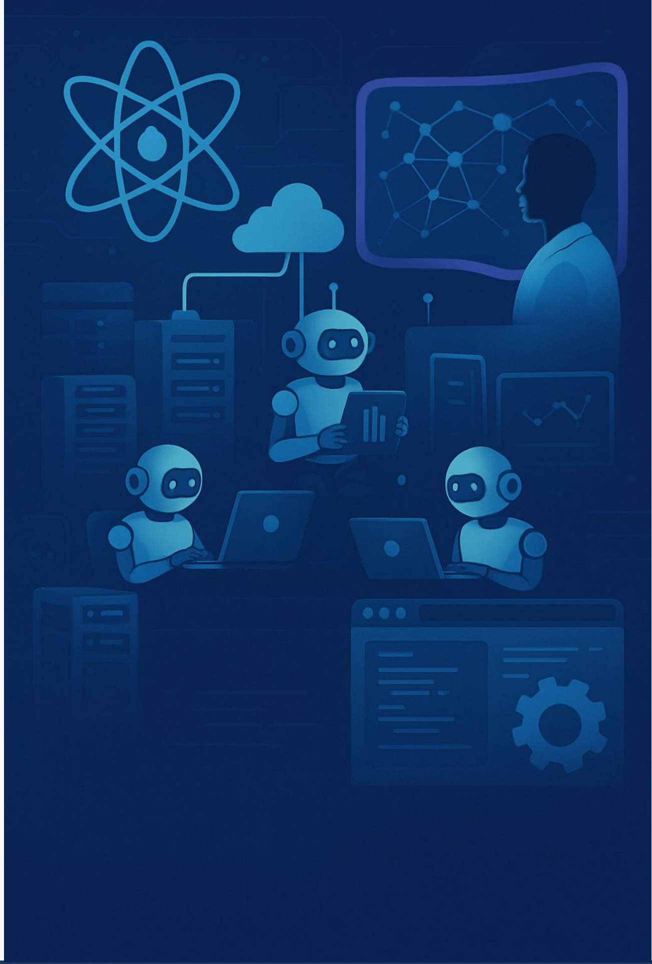
Scheduling Autonomous Agents and Multi-Model Inference

Kyle Chard
chard@uchicago.edu



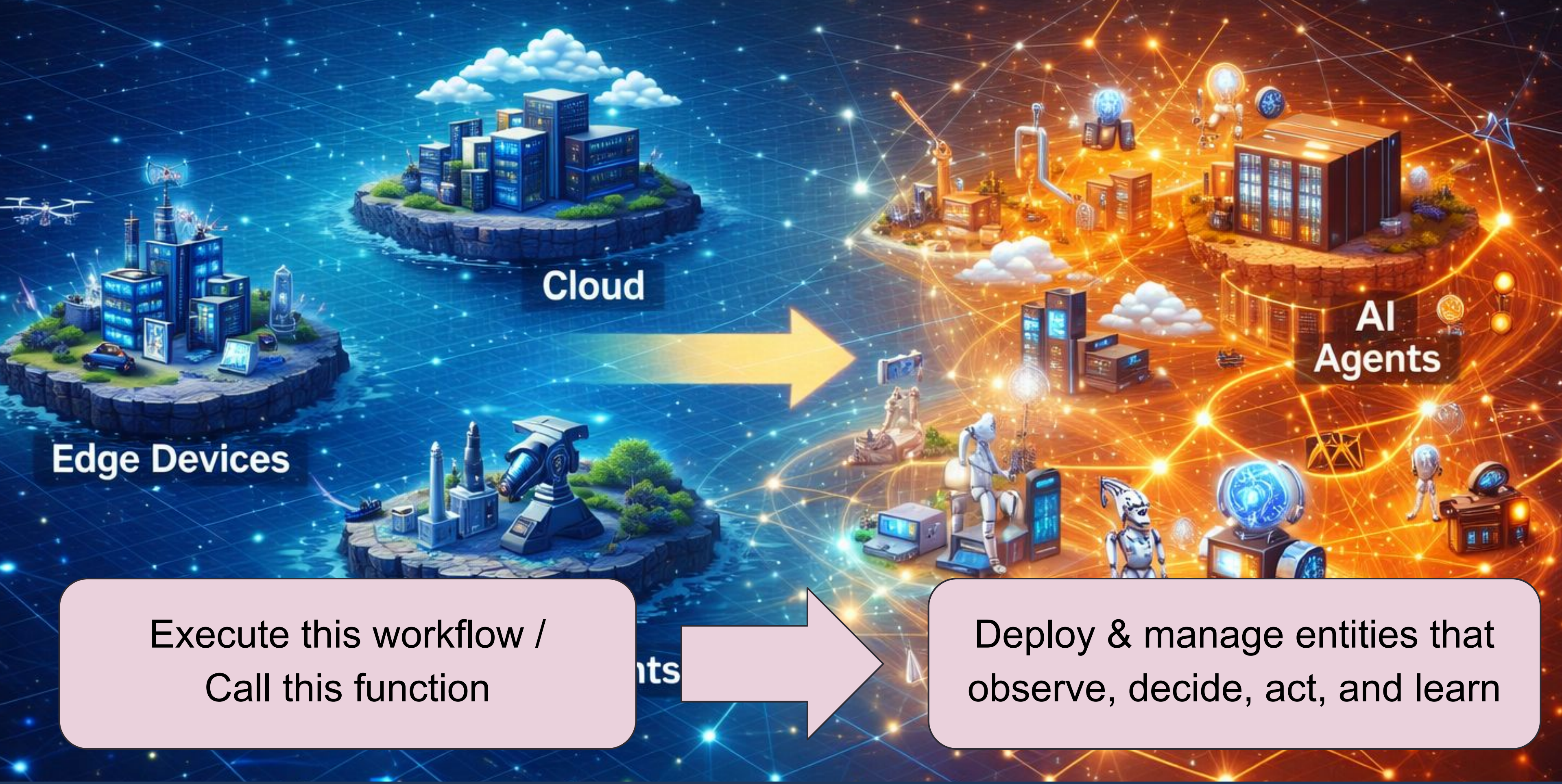
THE UNIVERSITY OF CHICAGO

8 October 2025

globus labs

THE UNIVERSITY OF CHICAGO

globus labs



Execute this workflow / Call this function

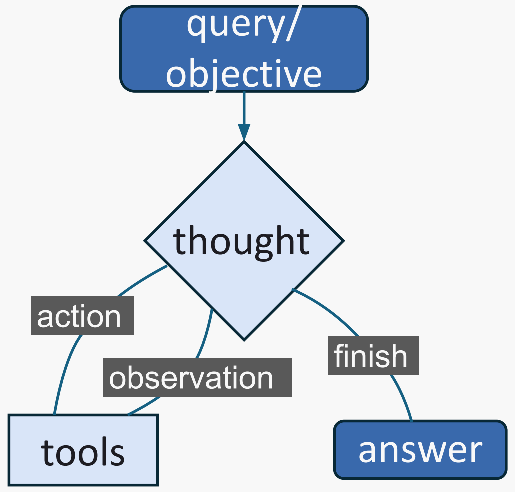
Deploy & manage entities that observe, decide, act, and learn

What is an "agent"?

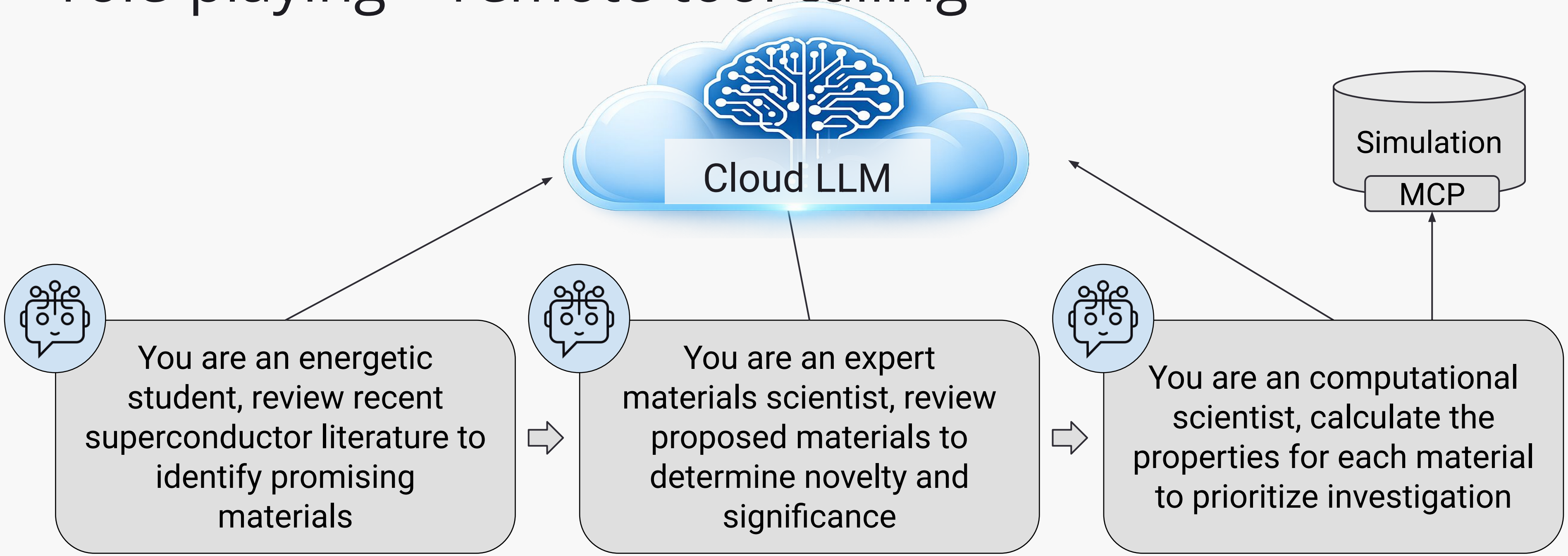
An agent is a persistent, stateful process that acts on behalf of a user or system. An agent may:

- **Observe** inputs or events
- **Plan** (decide on) actions using a policy (rules or LLM)
- **Act**: Execute tools or call other agents
- **Learn**: Update state to adapt over time

We can think of an agent as an assistant that can reason, act, and coordinate on our behalf



Most agentic systems focus on centralized role-playing + remote tool calling



You are an energetic student, review recent superconductor literature to identify promising materials

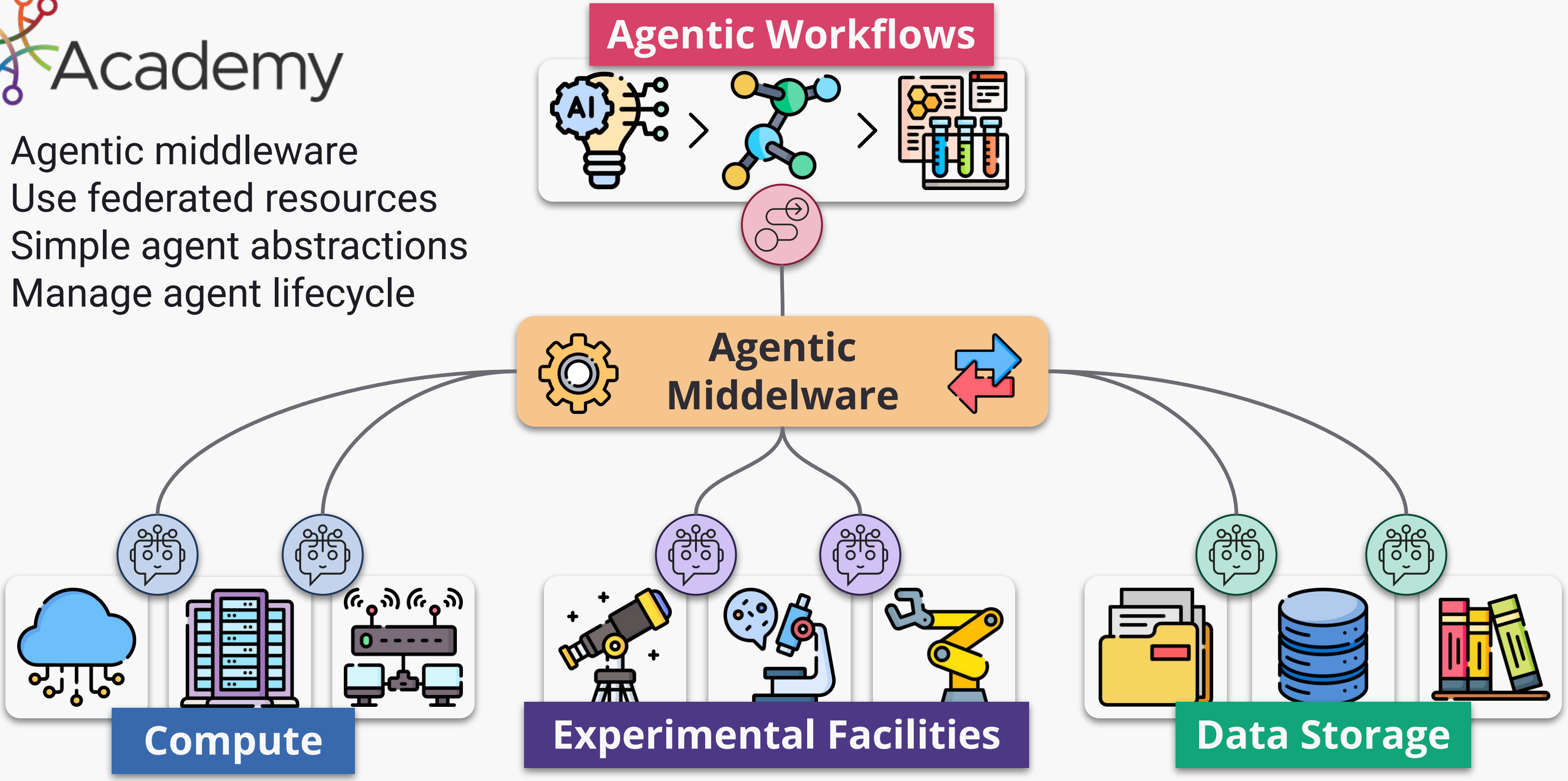
You are an expert materials scientist, review proposed materials to determine novelty and significance

You are an computational scientist, calculate the properties for each material to prioritize investigation

Deploying and managing agents across the computing continuum

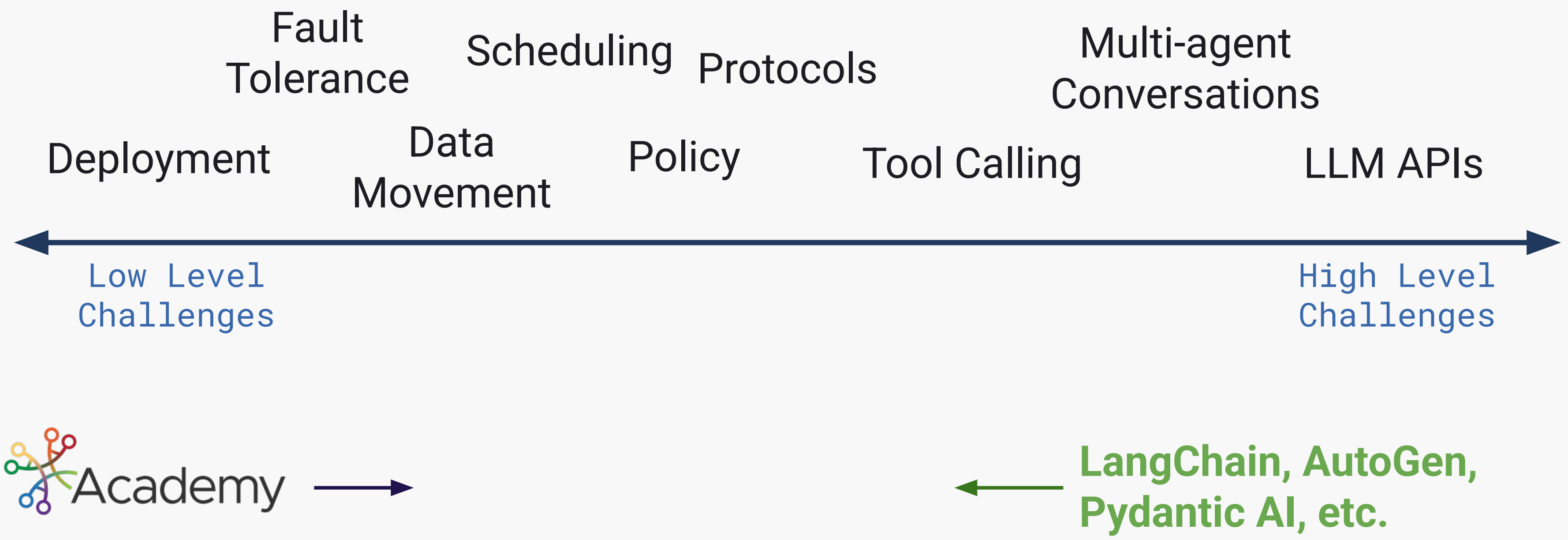
Academy

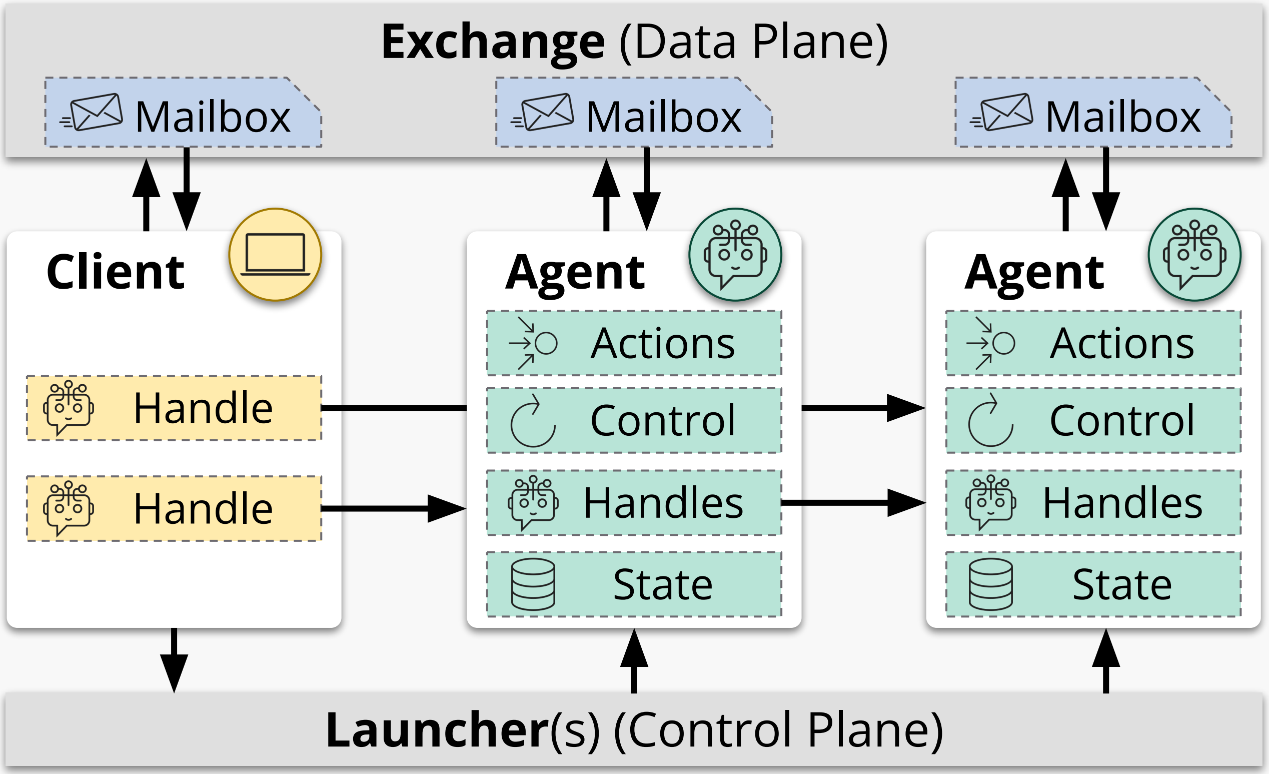
- Agentic middleware
- Use federated resources
- Simple agent abstractions
- Manage agent lifecycle



Kamatar, Pauloski et al. Empowering Scientific Workflows with Federated Agents, IPDPS, 2026

Agentic middleware: scope & challenges





Focus 3: Coordinate async agent messaging

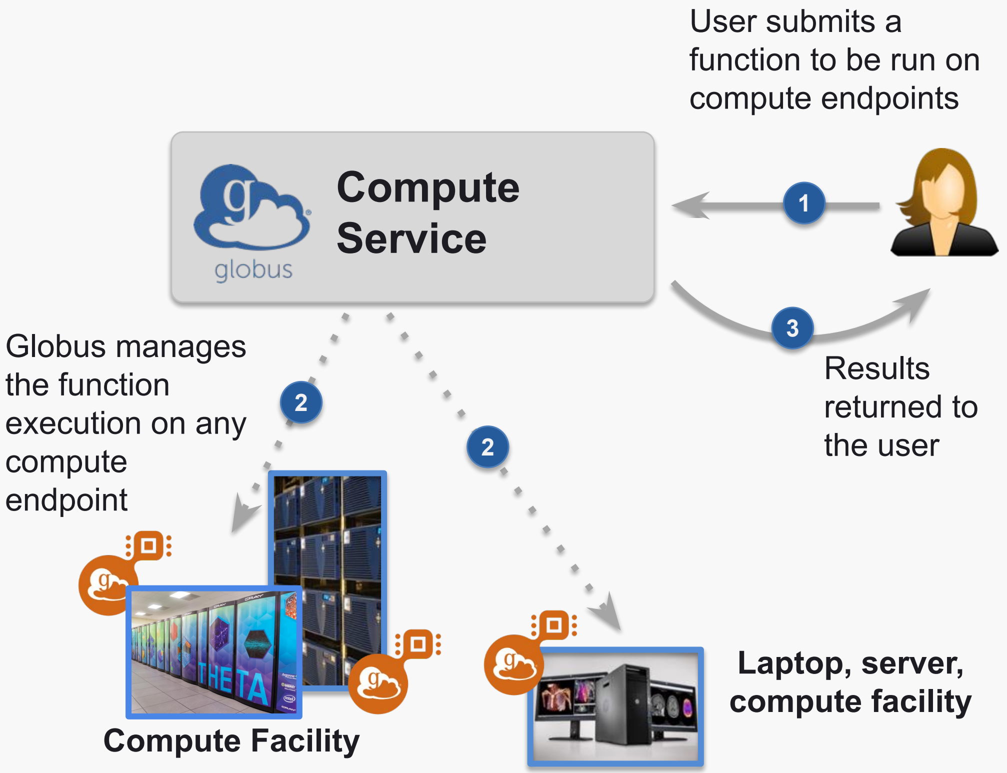
Focus 1: Program diverse agents and interactions

Focus 2: Deploy agents on federated resources

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

Globus Compute: Managed FaaS compute …on any system

- **Support use of Python for functions**
- **Fire and forget function execution**
- **Federated authentication, and local access control**
- **Uniform interface to various compute resources**


User submits a function to be run on compute endpoints
globus
Compute Service
1
3
Results returned to the user
Globus manages the function execution on any compute endpoint
2
2
Compute Facility
Laptop, server, compute facility

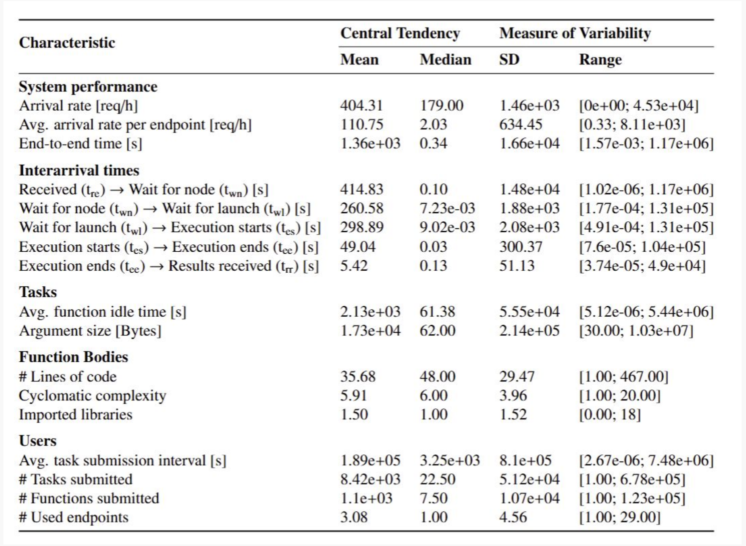
Globus Compute dataset

1,349,454 REGISTERED FUNCTIONS

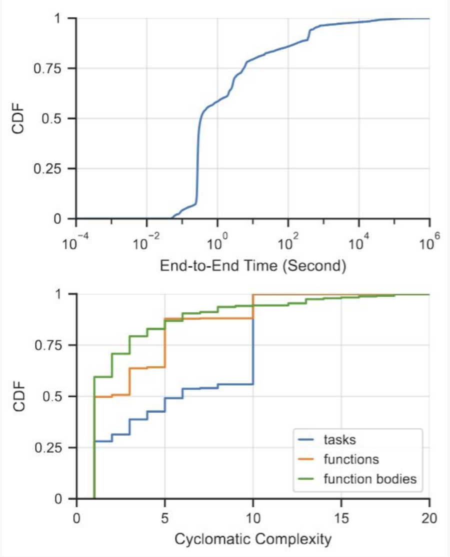
64,318,737 FUNCTION INVOCATIONS

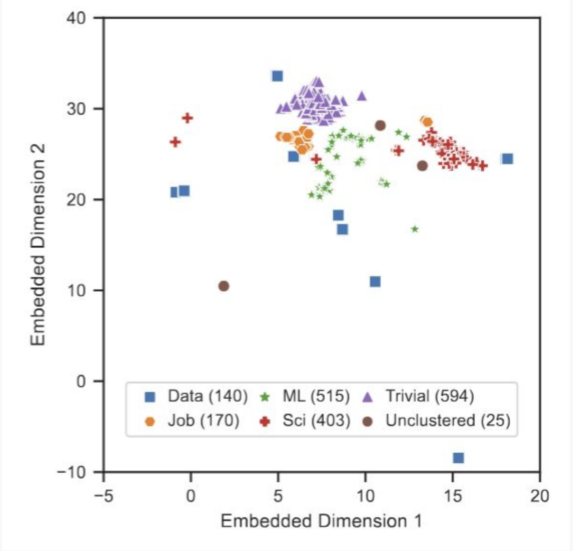
1,740 RUNNING USERS

33,657 REGISTERED ENDPOINTS



Characteristic	Central Tendency		Measure of Variability	
	Mean	**Median**	**SD**	**Range**
System performance				
Arrival rate [req/h]	404.31	179.00	1.46e+03	[0e+00; 4.53e+04]
Avg. arrival rate per endpoint [req/h]	110.75	2.03	634.45	[0.33; 8.11e+03]
End-to-end time [s]	1.36e+03	0.34	1.66e+04	[1.57e-03; 1.17e+06]
Interarrival times				
Received (t_{re}) → Wait for node (t_{wn}) [s]	414.83	0.10	1.48e+04	[1.02e-06; 1.17e+06]
Wait for node (t_{wn}) → Wait for launch (t_{wl}) [s]	260.58	7.23e-03	1.88e+03	[1.77e-04; 1.31e+05]
Wait for launch (t_{wl}) → Execution starts (t_{es}) [s]	298.89	9.02e-03	2.08e+03	[4.91e-04; 1.31e+05]
Execution starts (t_{es}) → Execution ends (t_{ee}) [s]	49.04	0.03	300.37	[7.6e-05; 1.04e+05]
Execution ends (t_{ee}) → Results received (t_{rr}) [s]	5.42	0.13	51.13	[3.74e-05; 4.9e+04]
Tasks				
Avg. function idle time [s]	2.13e+03	61.38	5.55e+04	[5.12e-06; 5.44e+06]
Argument size [Bytes]	1.73e+04	62.00	2.14e+05	[30.00; 1.03e+07]
Function Bodies				
# Lines of code	35.68	48.00	29.47	[1.00; 467.00]
Cyclomatic complexity	5.91	6.00	3.96	[1.00; 20.00]
Imported libraries	1.50	1.00	1.52	[0.00; 18]
Users				
Avg. task submission interval [s]	1.89e+05	3.25e+03	8.1e+05	[2.67e-06; 7.48e+06]
# Tasks submitted	8.42e+03	22.50	5.12e+04	[1.00; 6.78e+05]
# Functions submitted	1.1e+03	7.50	1.07e+04	[1.00; 1.23e+05]
# Used endpoints	3.08	1.00	4.56	[1.00; 29.00]





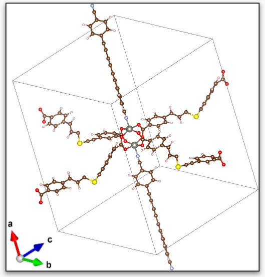
Bauer et al. The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud, Future Generation Computer Systems, 2024

Academy adoption in science applications

Released as open-source software for defining agentic systems

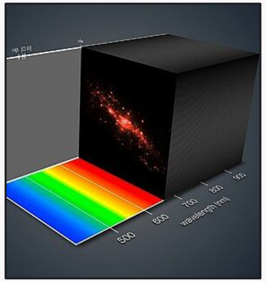
Integration with agent (e.g., LangGraph, MCP) and science ecosystems

Adopted by various groups in national laboratories, research facilities, industry, and international research consortia

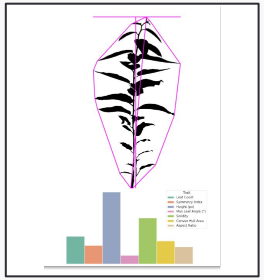


Materials Discovery

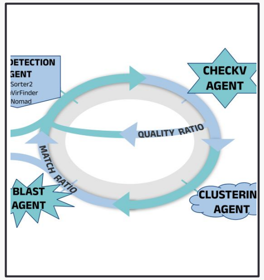
Pandemic Preparation



Cosmology



Plant Science



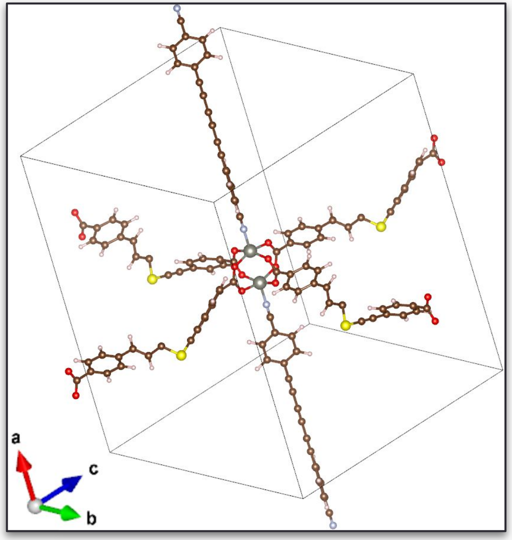
Microbiome

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?

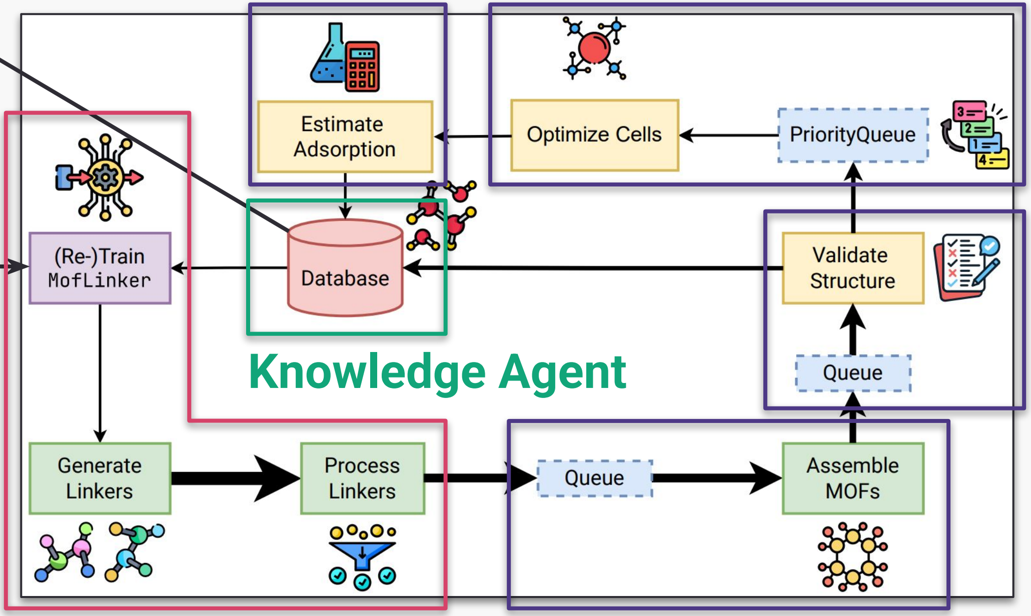


Intractable search space of ligand, node, & geometry combinations

MOFA: Online learning + GenAI + Simulation

Embodied Agents*

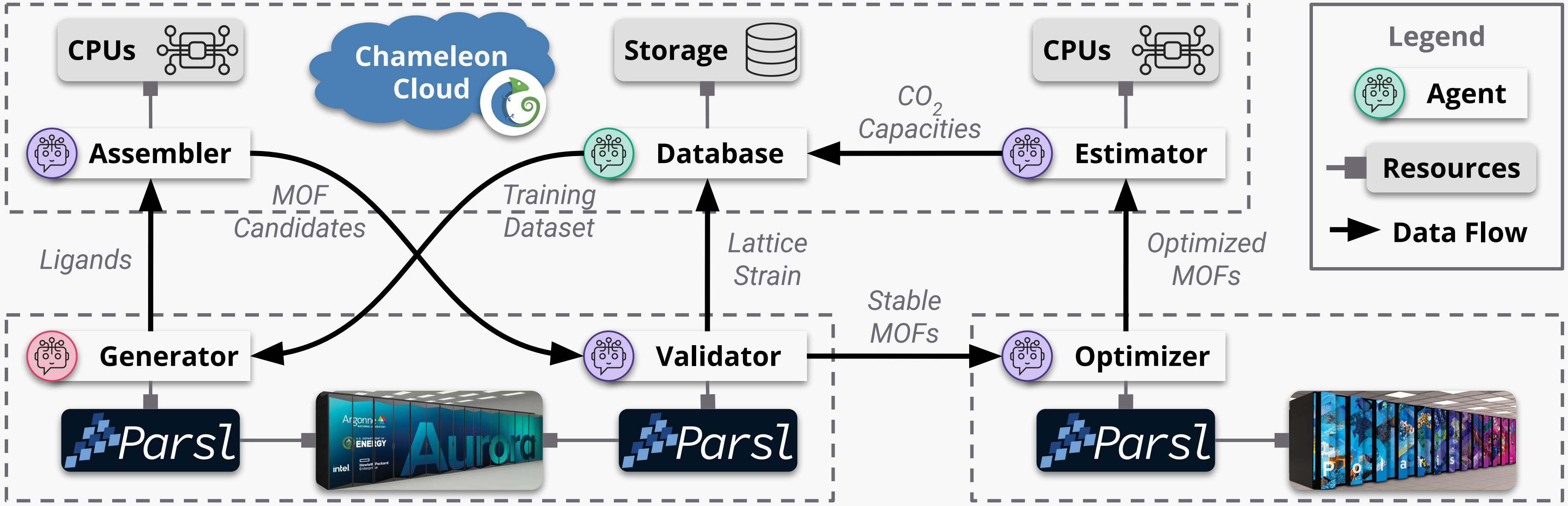
AI Agent



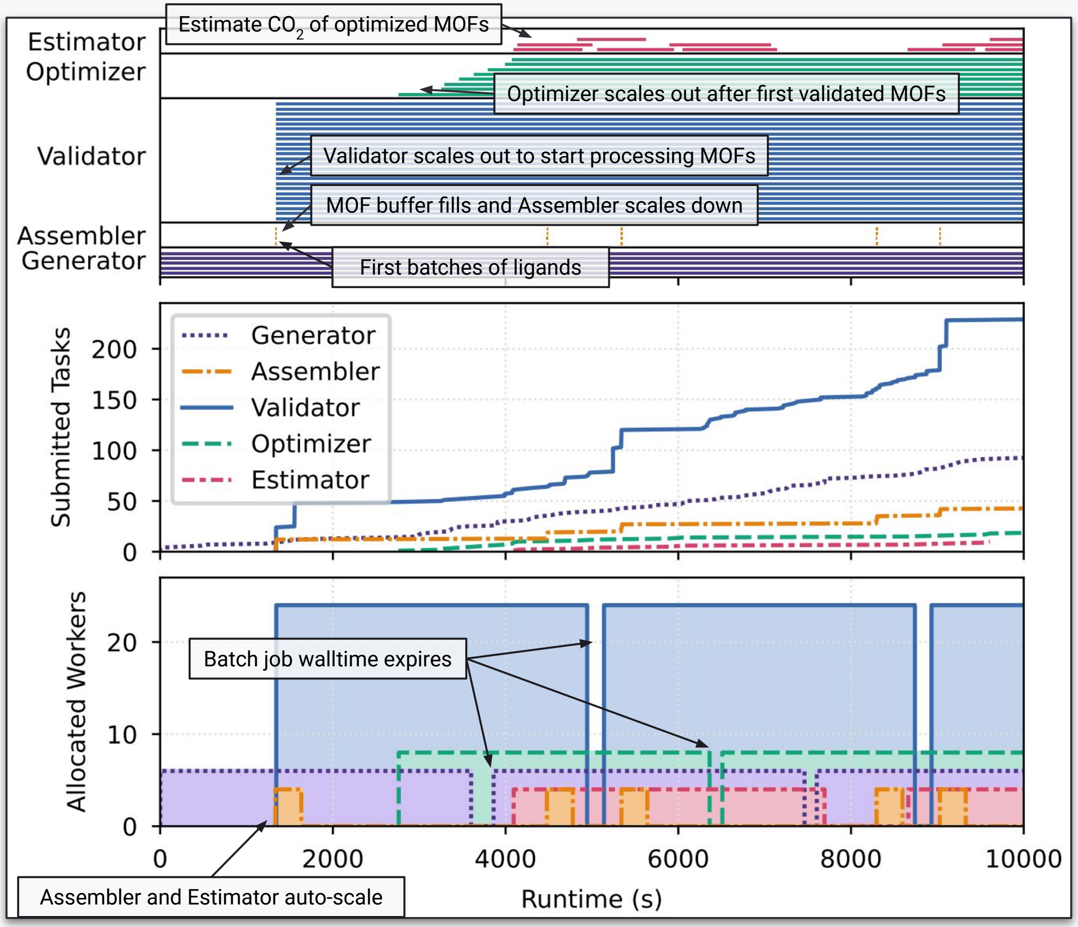
Knowledge Agent

Computational Agents

MOFA realized with 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

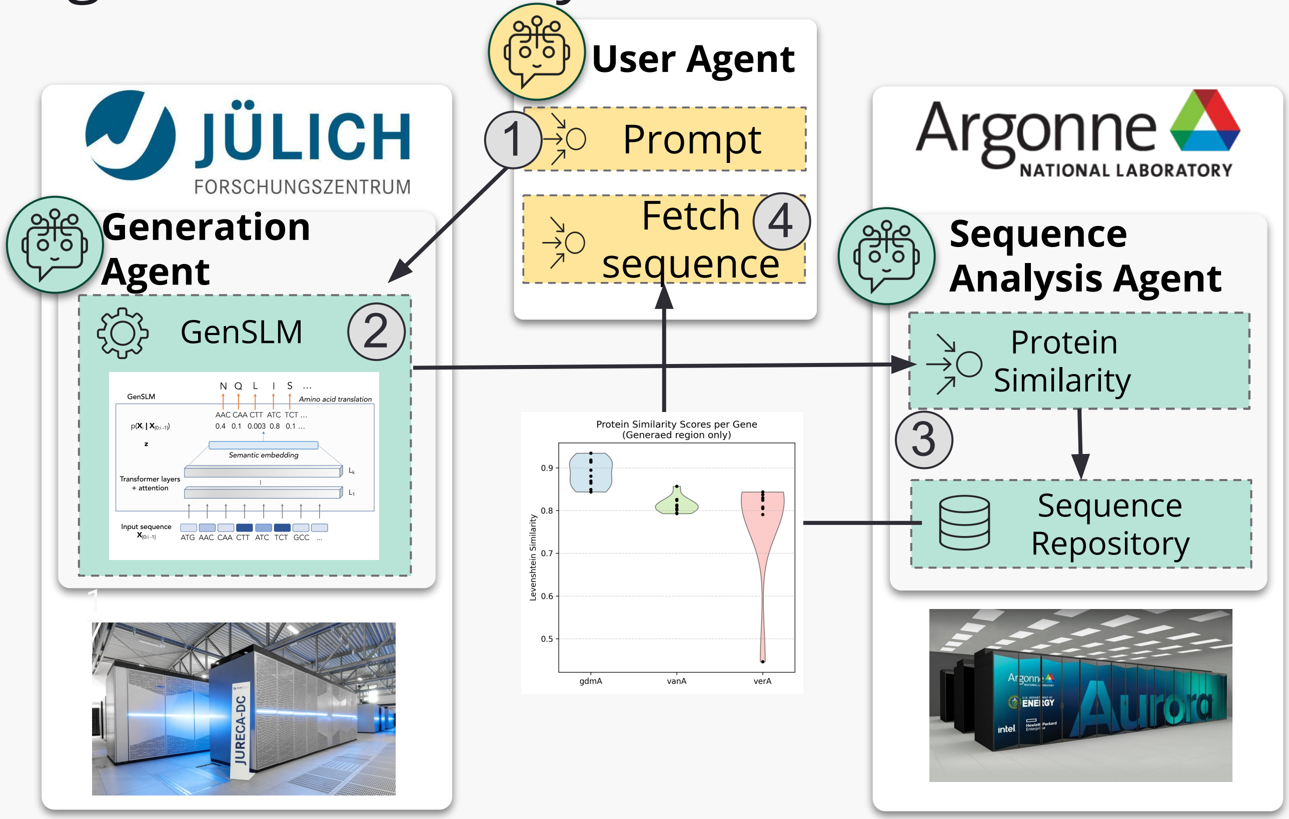
Agentic enzyme design with Academy

Application Credits: Xinran Lian, Alex Brace, Arvind Ramanathan

GenSLM is a genome-scale language model that can generate bacterial and viral protein sequences.

We implement three agents:

1. **User agent** triggers new analyses
2. **Generation agent**, which runs GenSLM models, trained on proprietary data, to generate sequences
3. **Sequence analysis agent,** which hosts methods for evaluating protein similarity, and stores promising sequences
4. **User agent** (again) monitors sequence repository for promising candidate sequences



What about scheduling? Moving from job execution to continuous agentic system management

- Differences (with respect to workflows)
  - Agents are persistent and long-lived (workflows have bounded runs)
  - Agents are stateful (workflows integrate stateless tasks)
  - Agents are dynamic and reactive (workflows follow a predefined structure)
  - Agents coordinate with other agents (workflows are orchestrated)
- Similarities
  - Data locality matters (placement impacts performance/cost)
  - Infrastructure constraints apply (heterogeneous resources, quotas)
  - Fault tolerance is critical (detection, checkpoints, restart)
- What are our scheduling goals?
  - Throughput? makespan? Latency?
  - Responsiveness, adaptability, cost/utilization

What does the scheduling problem look like?

- Microservices/container orchestration schedule long-lived entities
    - But are generally stateless (or at least state stored elsewhere)
- Stream processing scheduling is event-based
    - But operates on defined pipelines
- Actor placement addresses stateful entities
    - But not autonomous
- HPC job scheduling addresses resource allocation

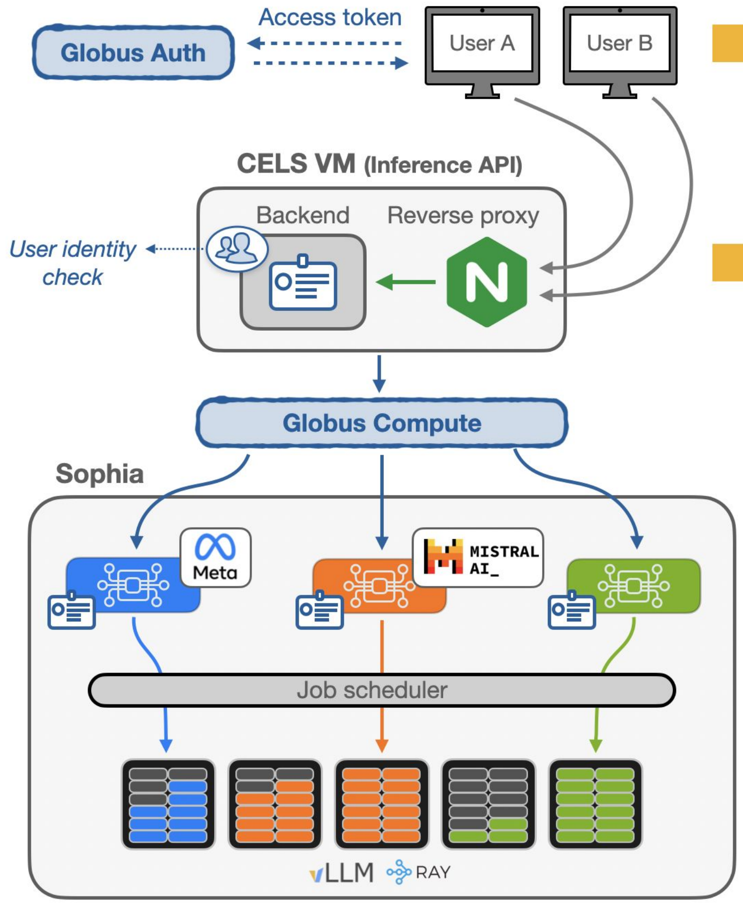
What does the scheduling problem look like?

- Microservices/container orchestration schedule long-lived entities
  - But are generally stateless (or at least state stored elsewhere)
- Stream processing scheduling is event-based
  - But operates on defined pipelines
- Actor placement addresses stateful entities
  - But not autonomous
- HPC job scheduling addresses resource allocation

Agent scheduling = microservices/actors (persistence/placement) + streaming (reaction/scaling) + HPC (constraints/performance) ??

Inference serving

- Modern workflows increasingly depend on many LLMs that differ in size, specialization, and resource needs
- AuroraGPT inference service for scalable LLM serving built on vLLM
- Since deployment July 2024
  - 13.5 M inference tasks, 386 unique users, 15.9 billion tokens generated, 23 models, 9 LLM families, 2 clusters
- Performance not optimized for HPC



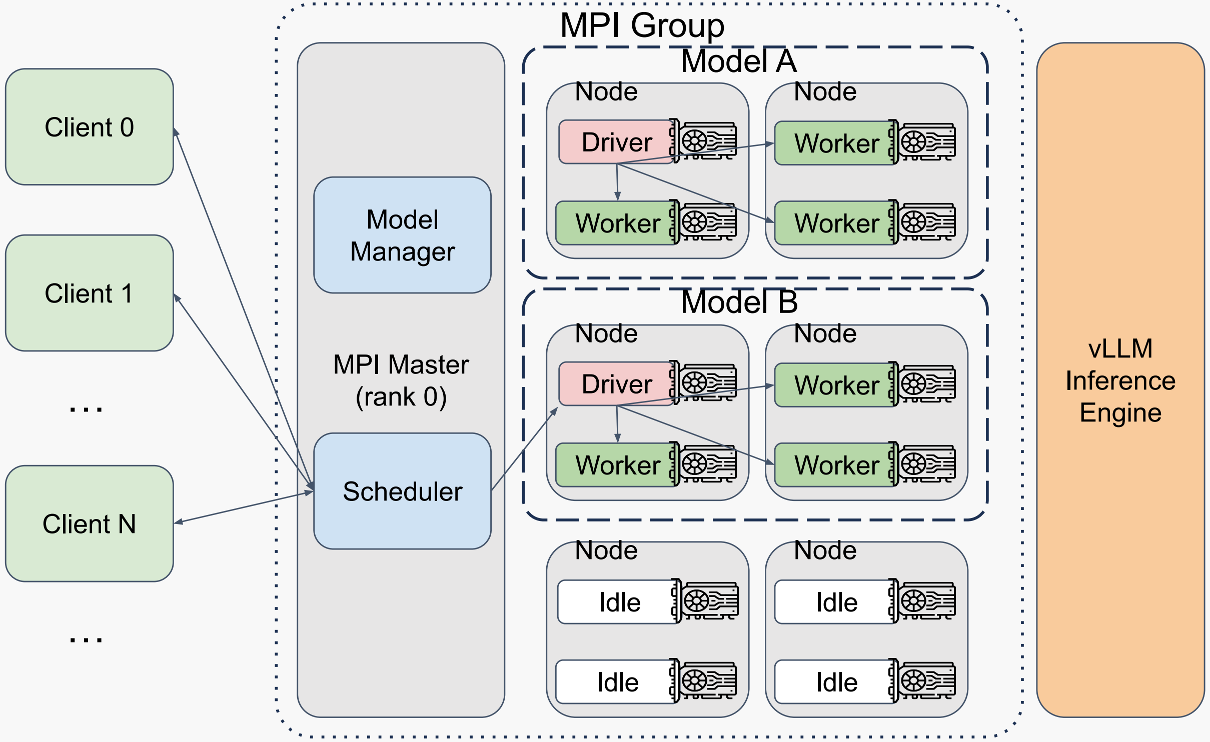
Credit: Benoit Cote and Aditya Tanikanti

Designing an HPC-focused multi-model serving system

Build on MPI runtime for coordination across nodes and GPUs

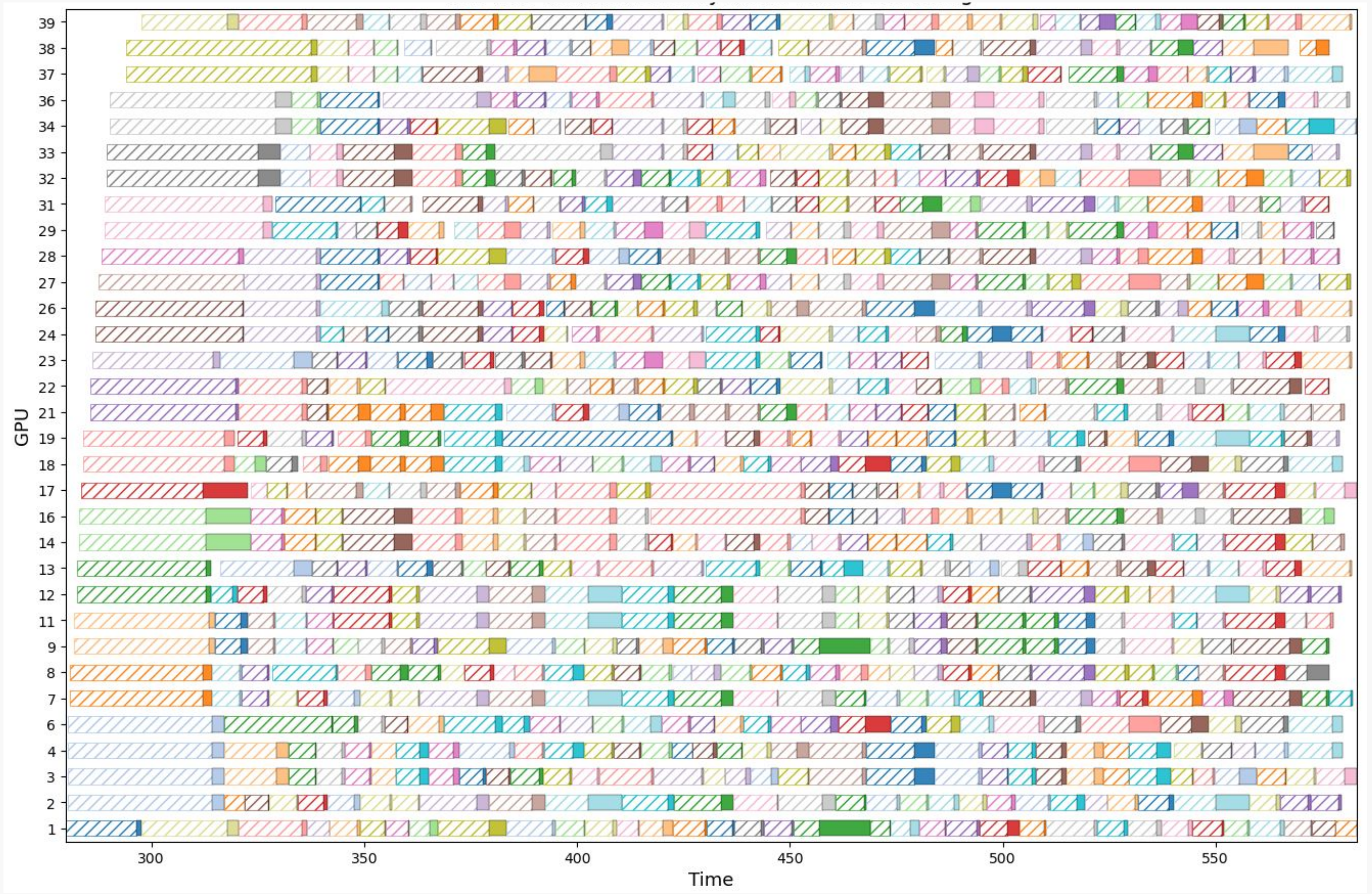
Leveraging vLLM for model interface

Adding a layer of model management supporting fast model switching, caching, and routing.



Model Switching

- Supports rapid model switching across GPUs
- 850 requests in 1 minute on 8 nodes / 32 GPUs with 64 logical models
- Reuses node-local CPU-cached model shards after first load



Questions?

Reach out if you are interested:

`chard@uchicago.edu`

Learn more:

- [parsl-project.org]
- [globus.org]
- [academy-agents.org]