Programming the Continuum

Towards Better Techniques for Developing Distributed Science Applications

Dissertation Defense

by J. Gregory Pauloski

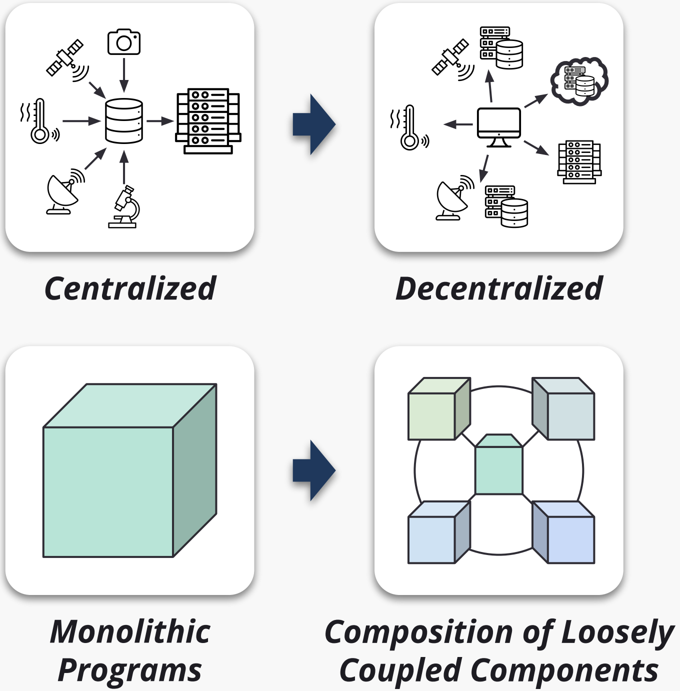
Committee: Kyle Chard (*advisor*), Ian Foster (*advisor*), Michael Franklin

Why Program the Continuum?

Better and More Ambitious Science

Computing continuum: cyberinfrastructure spanning edge devices, the cloud, and supercomputers

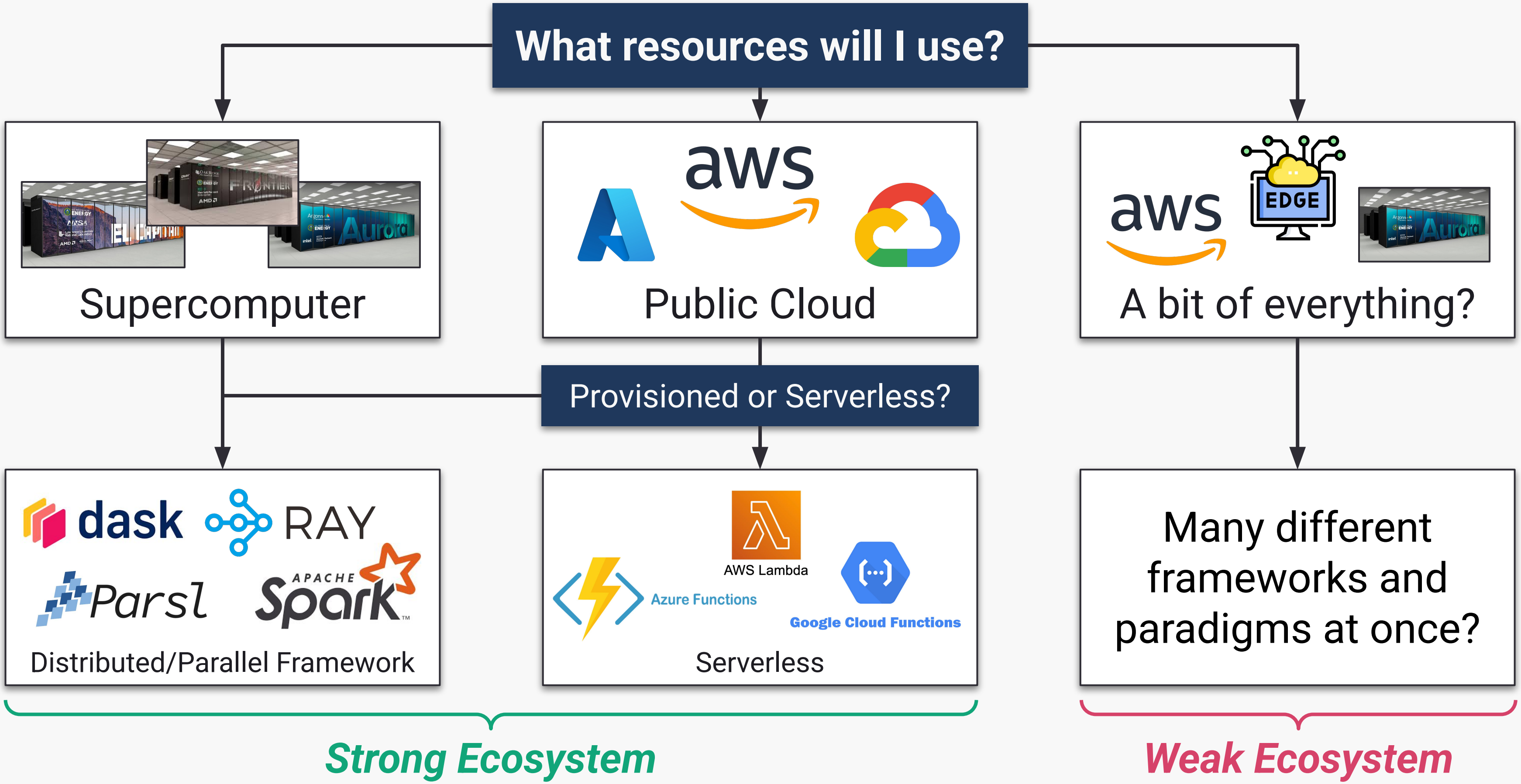
- ➔ Faster & more reliable networks
- ➔ Specialized accelerators
- ➔ Data locality
- ➔ Performance requirements
- ➔ Compute availability & costs
- ➔ Better cloud management



Imagine you are a computational scientist…

With a distributed science application to build…

What framework do you use?


What resources will I use?
Supercomputer
aws
Public Cloud
aws
EDGE
A bit of everything?
Provisioned or Serverless?
dask
RAY
Parsl
APACHE Spark
Distributed/Parallel Framework
AWS Lambda
Azure Functions
Google Cloud Functions
Serverless
Many different frameworks and paradigms at once?
Strong Ecosystem
Weak Ecosystem

Challenges in Programming the Continuum

Distribute computational tasks across federated devices? Possible.

- ➔ Globus Compute distributed FaaS Model

Manage intermediate data between tasks? Limited.

- ➔ Interoperability between distributed/parallel frameworks is challenging
- ➔ Cloud object storage is reliable/available but expensive for data-intensive apps
- ➔ P2P CDNs are good for edge devices but bad for clusters

Build persistent and loosely coupled components? Limited.

- ➔ Easy in cloud-native apps (microservice architectures)
- ➔ Hard in federated apps (requires ad-hoc solutions)

Programming the Continuum

New programming techniques ***enable and accelerate*** *task-centric* ***science applications*** *executed* ***across the computing continuum****.*

P1	What are the limitations in existing distributed computing frameworks?	eScience '24 (Best Paper)
P2	How to represent and efficiently move objects across federated systems?	SC '23 & HPPSS '24
P3	How to support common high-level data flow patterns?	TPDS '24
P4	How to build and deploy stateful agents across federated systems?	*IEEE Computer* & SC '25**

Better, easier, & faster science! — MLHPC '21, IJHPCA '23, HCW '23, IJHPCA '24, CCGRID '25 & Others In Review/Progress

**Under Review / In Progress*

Task Performance Suite

ProxyStore

Proxy Patterns

Federated Agents

Modern Science Applications are *Task-centric*

Applications are composed as a set of **discrete tasks** designed to **automate** computational processes to achieve a **scientific goal**

Benefits

- Heterogeneous Resources
- Software Modularity
- Monitoring
- Performance
- Reproducibility
- *and many more!*

Applications [1]

- Bioinformatics
- Cosmology
- High Energy Physics
- Materials Science
- Molecular Dynamics
- *and many more!*

Challenges [2]

- Coupling AI/ML/Quantum
- Cloud and HPC Integration
- Data Flow/Provenance
- Standards/Interoperability
- Performance
- *and many more!*

[1] "Scientific Workflows: Moving Across Paradigms" (https://dl.acm.org/doi/10.1145/3012429)
[2] Workflows Community Summit (https://arxiv.org/abs/2304.00019)

Task Execution Frameworks

Manage the execution of tasks in parallel across arbitrary hardware.

Workflow Management Systems

Define, manage, and execute workflows represented by a directed acyclic graph (DAG) of tasks

Explicit

DAG defined via configuration file or domain specific language

Implicit

Task dependencies derived through dynamic evaluation of a procedural script

Concurrent Executors

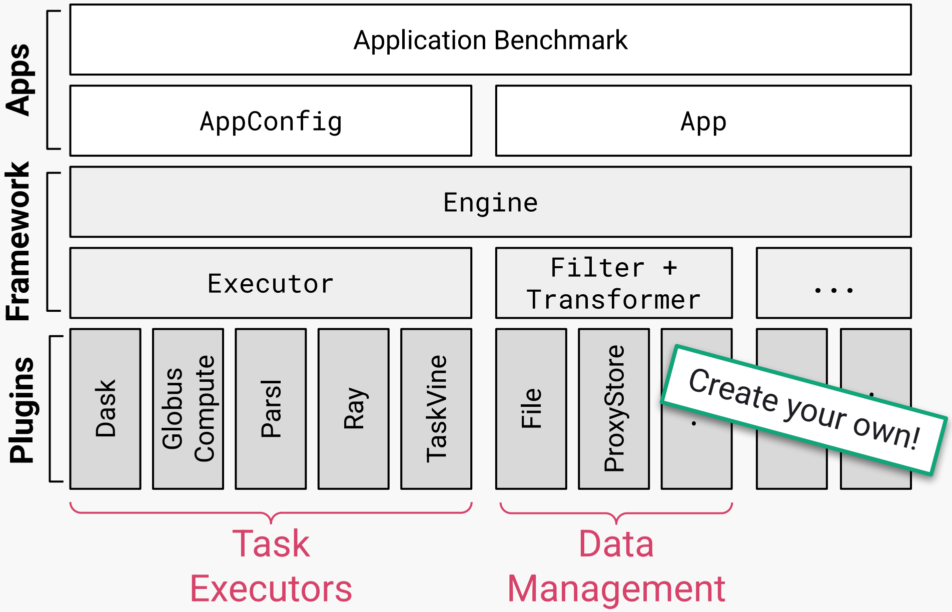
On-demand asynchronous execution of tasks

How do we **benchmark** and compare execution frameworks?

TaPS: Task Performance Suite

- ➔ Reference set of applications to standardize benchmarking workloads
- ➔ Robust and reproducible configuration system
- ➔ Benchmark task executors & data management systems

Guide future research!

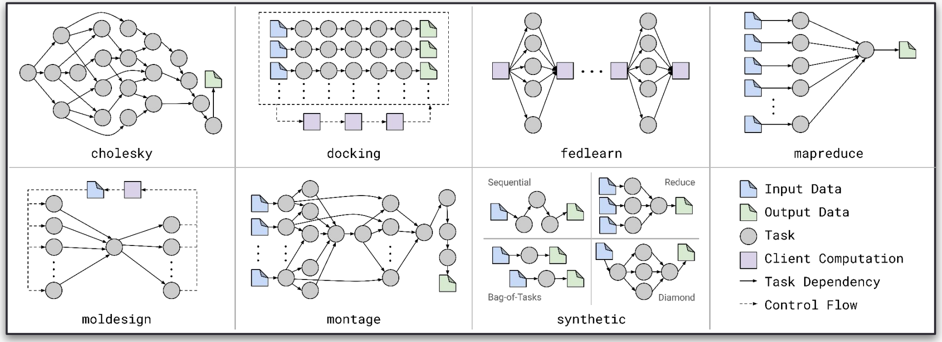


https://taps.proxystore.dev/latest/api/

Applications: *Benchmarking Workloads*

- ➔ Seven Real Apps
- ➔ Two Synthetic
- ➔ Diverse Patterns
- ➔ Diverse Domains
- ➔ Per-App Guides
- ➔ Add your own!

Type	Name	Domain	Task Type(s)	Data Type(s)
Real	`cholesky`	Linear Algebra	Python	In-memory
	`docking`	Drug Discovery	Executable, Python	File
	`fedlearn`	Machine Learning	Python	In-memory
	`mapreduce`	Text Analysis	Python	File, In-memory
	`moldesign`	Molecular Design	Python	In-memory
	`montage`	Astronomy	Executable	File
	`physics`	Mechanics	Python	In-memory
Synthetic	`synthetic`	—	Python	In-memory
	`failures`	—	*Depends on base app*	*Depends on base app*



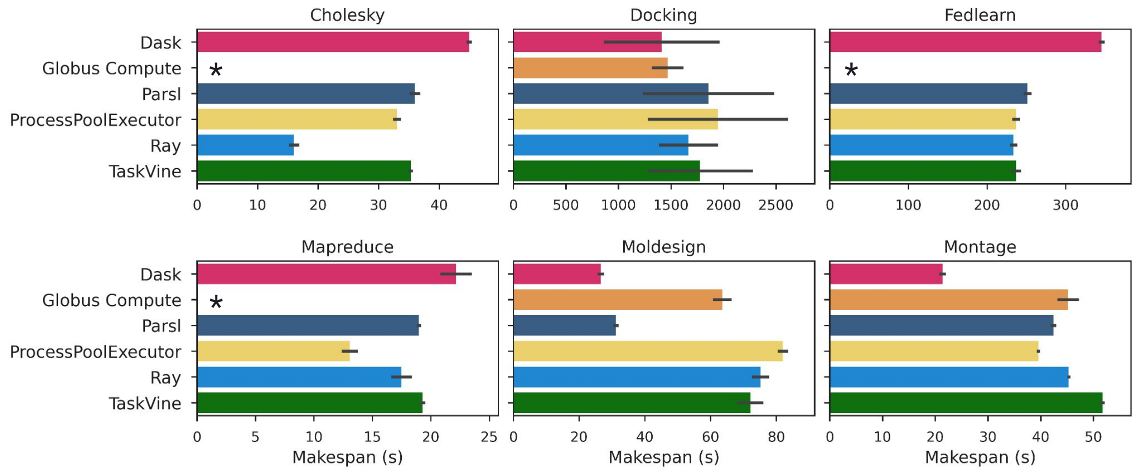
https://taps.proxystore.dev/latest/apps/

What did we learn?

Many things!

But most important to this story is…

- ➔ No single executor is the best at everything.
- ➔ Large-scale federated apps will need to use multiple concurrently for optimal results.



Application	Workers	Task Count	Max Serialized Object Size	Parameters
cholesky	64	385	24 MB	Matrix Size: 10 000×10 000, Block Size: 1000×1000
docking	32	192	$O(1)$ kB	Initial Simulations: 3, Batch Size: 8, Rounds: 3
fedlearn	32	48	20 MB	Dataset: MNIST, Clients: 16, Batch Size: 32, Rounds: 3, Epochs/Round: 1
mapreduce	32	33	114 MB	Dataset: Enron Email Corpus, Map Task Count: 32
moldesign	32	346	$O(1)$ MB	Initial Simulations: 16, Batch Size: 16, Search Count: 64
montage	32	419	$O(1)$ kB	—

*Task data exceeds Globus Compute 10 MB payload limit.

https://github.com/proxystore/escience24-taps-analysis

Task Performance Suite

ProxyStore

Proxy Patterns

Federated Agents

Representing Intermediate Objects

In a federated environment, how do we…

- ➔ **Represent** an object ***x*** such that the producer and consumers of ***x*** can globally reference ***x***?
  - Assume **x** is immutable in the context of intermediate objects
- ➔ **Communicate** ***x*** from producer to consumers when consumers
  - are not known ahead of time,
  - can be located in different places, and
  - have different optimal communication methods?

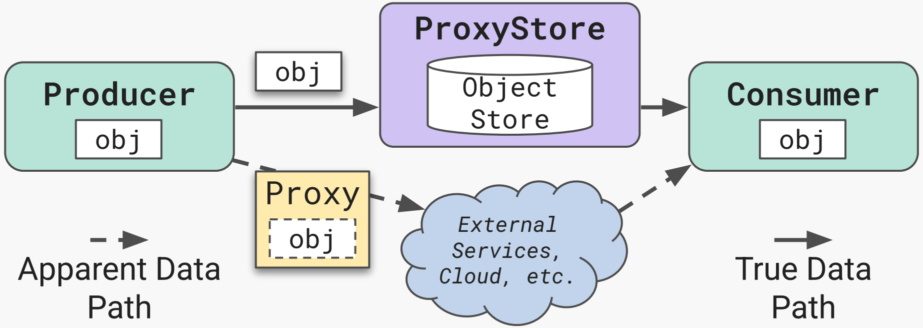
Representing Intermediate Objects

Case Study: **Ray**

➔ Ray **represents** ***x*** with an object ref
- ✓ Distributed reference counting
- ✓ Cheap to pass around
- ✗ Not valid outside of the Ray cluster it was created in

➔ Ray **communicates** ***x*** using RPCs
- ✓ Fast & direct within a cluster
- ✗ RPC not possible outside of cluster

ProxyStore



Data flow management library for distributed Python workflows

- Represent and efficiently move objects in federated applications
- Proxy ***transparently*** decouples control and data flow
- Best of both ***pass-by-reference*** and ***pass-by-value***
- Use any mediated communication method via plugins

Proxy Objects

What is a proxy (in this context)?

- Self-contained wide-area **reference** to a **target** object
- Transparently resolve target **just-in-time** when first used

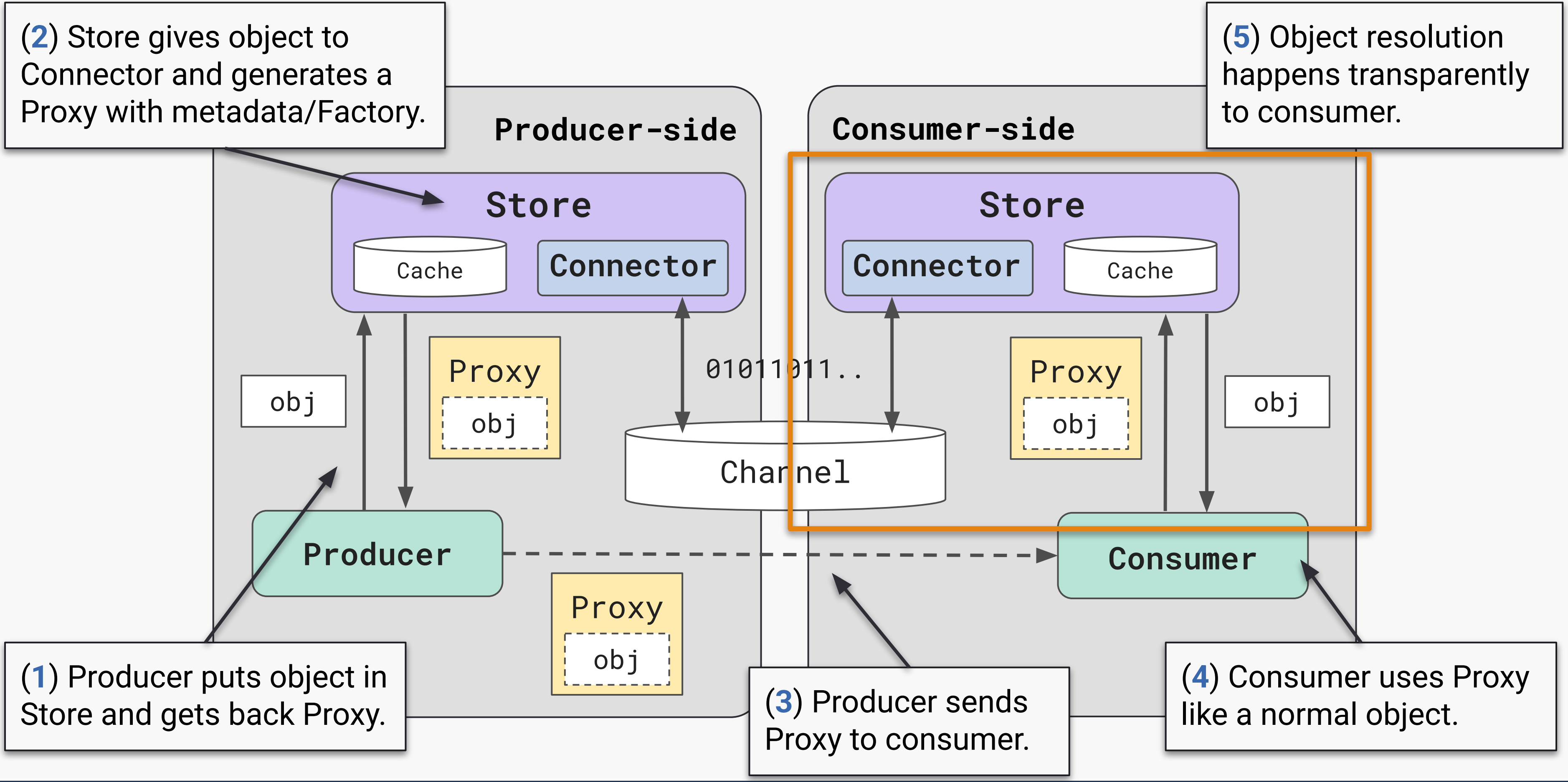
What are the benefits?

- Performance (pass-by-reference, async resolve, skip unused objects)
- Reduce code complexity
- Partial resolution of complex objects
- Access control

```python
from proxystore.connectors import RedisConnector
from proxystore.store import Store
from proxystore.proxy import Proxy

def foo(x: Bar) -> ...:
    # Resolve of x deferred until use
    assert isinstance(x, Bar)
    # More computation...

with Store('demo', RedisConnector(...)) as store:
    x = Bar(...)
    p = store.proxy(x)  # Anything can be proxied
    assert isinstance(p, Proxy)
    foo(p)  # Proxies can be passed-by-ref anywhere
```


(2) Store gives object to Connector and generates a Proxy with metadata/Factory.
Producer-side
Store
Cache
Connector
obj
Proxy
obj
01011011..
Channel
Producer
Proxy
obj
(1) Producer puts object in Store and gets back Proxy.
(5) Object resolution happens transparently to consumer.
Consumer-side
Store
Connector
Cache
Proxy
obj
obj
Consumer
(3) Producer sends Proxy to consumer.
(4) Consumer uses Proxy like a normal object.

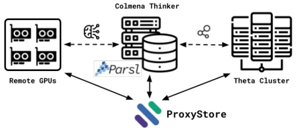
Connectors

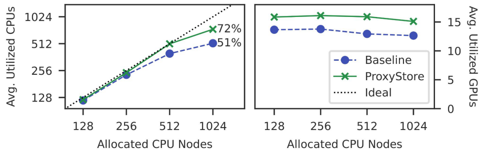
- Comprehensive **mediated** methods (producer/consumer may be temporally decoupled)
- `Connector` = Python **`Protocol`**
- **`MultiConnector`**: Policy-based routing between instances

Protocol	Storage	Intra-Site	Inter-Site	Persistence
File System	Disk	✓		✓
Redis/KeyDB	Hybrid	✓		✓
Margo	Memory	✓		
UCX	Memory	✓		
ZMQ	Memory	✓		
Globus	Disk		✓	✓
DAOS	Disk*	✓		✓
P2P Endpoint	Hybrid	✓	✓	✓

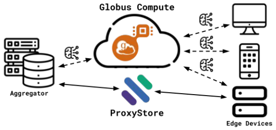
Where do we use ProxyStore?

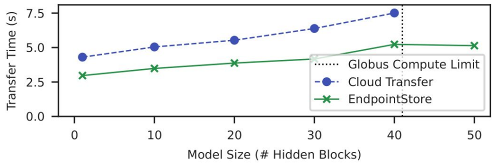
Multi-Site Workflows



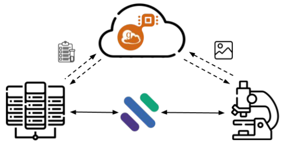


Federated Learning

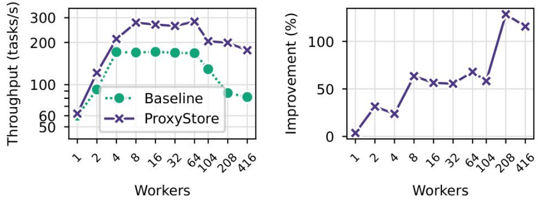




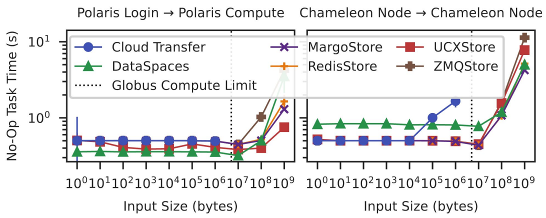
Real-time Processing



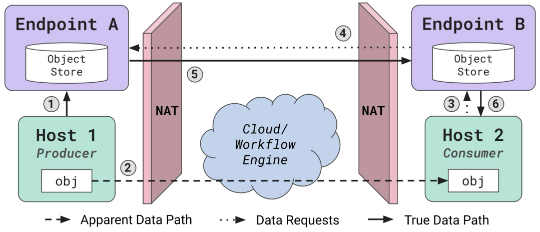
Configuration	Proxied	Time (ms)	Improvement
Globus Compute baseline	—	3411 ± 389	—
FileStore	Inputs	2318 ± 130	32.1%
	Inputs/Outputs	2160 ± 46	36.6%
EndpointStore	Inputs	2375 ± 98	30.4%
	Inputs/Outputs	2280 ± 107	33.2%



Reduce Scheduler Overhead



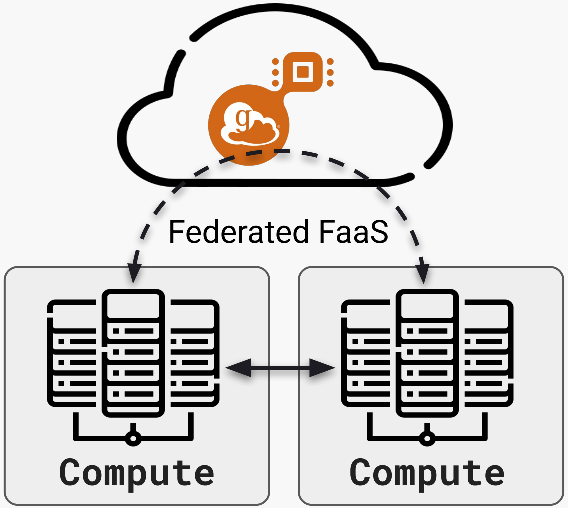
RDMA in FaaS Systems

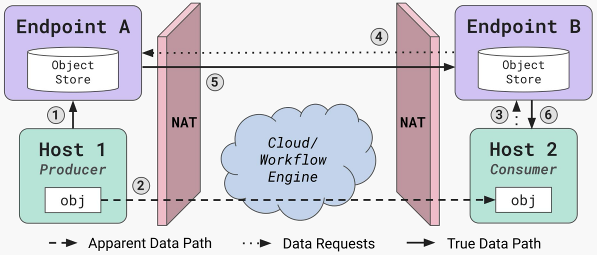


P2P Networking

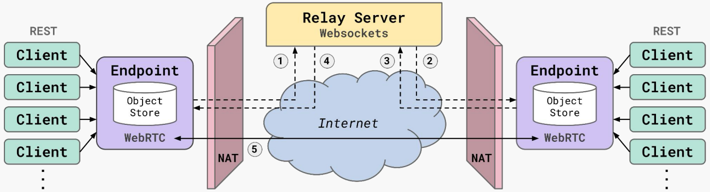
P2P Endpoints: Easy* Multi-Site Workflows

Moving data between sites through the cloud is impractical!





ProxyStore Endpoints: Move proxies through the cloud and data peer-to-peer with UDP hole punching



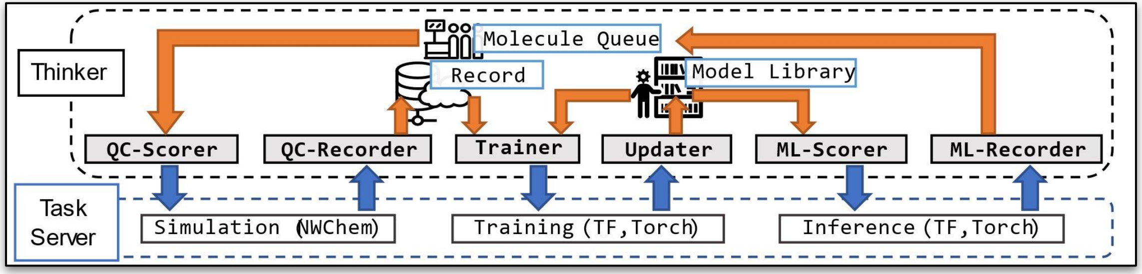

```
$ proxystore-endpoint configure demo --relay wss://relay.proxystore.dev
$ proxystore-endpoint start demo    # Runs as a daemon process
```

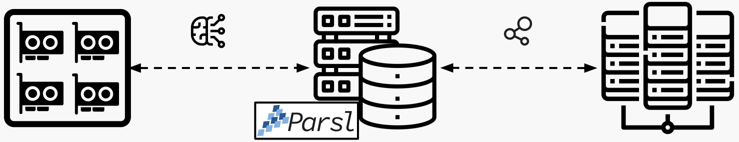
* *Easy = no SSH tunnels, one-time setup, no cloud fees*

docs.proxystore.dev/main/guides/endpoints/

Multi-site Active Learning

Science Goal: Use quantum chemistry simulations and surrogate ML models to efficiently identify electrolytes with high ionization potentials in a candidate set.





20 GPU Workstation
- Training Tasks
- Inference Tasks

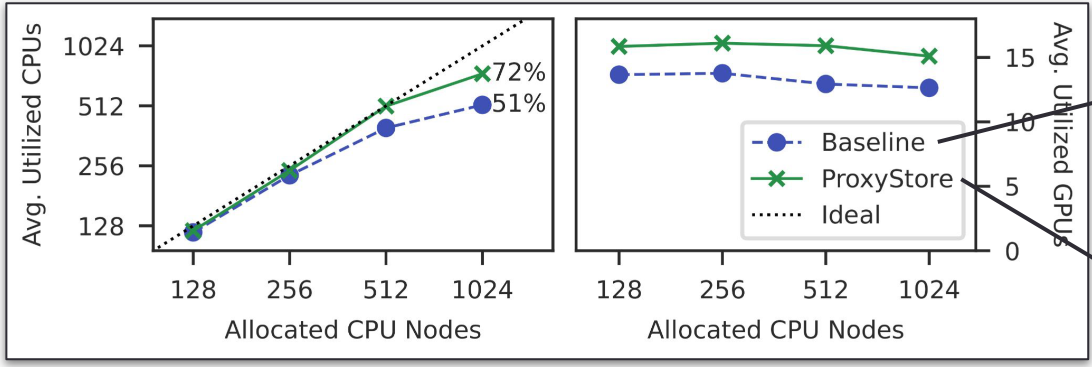
Workstation/Head Node
- Submit work
- Process results

1024 Theta KNL Nodes
- Simulation Tasks

Logan Ward, J. Gregory Pauloski, Valerie Hayot-Sasson, Ryan Chard, Yadu Babuji, Ganesh Sivaraman, Sutanay Choudhury, Kyle Chard, Rajeev Thakur, and Ian Foster. *Cloud services enable efficient AI-guided simulation workflows across heterogeneous resources*. In Heterogeneity in Computing Workshop at IPDPS. IEEE Computer Society, 2023.

Multi-site Active Learning

Systems Goal: Reduce task communication overheads in workflow system to increase system utilization and task throughput.



Baseline: Parsl manages all intermediate data (transfer via manually created SSH channels)

ProxyStore: MultiConnector
- *Simulation*: Redis
- *Training*: P2P Endpoints
- *Inference*: Globus Transfer/ P2P Endpoints

Takeaways
- Reduce overheads in task scheduler
- Reduce communication of re-used data
- Optimize communication method per data type
- No changes to task code needed

Task Performance Suite

ProxyStore

Proxy Patterns

Federated Agents

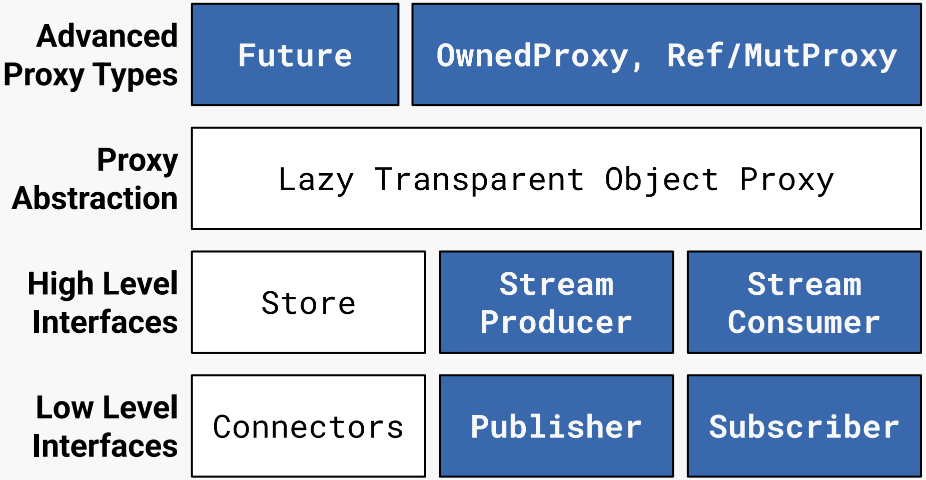
Yet...

Object proxy is a ***low-level*** paradigm:

- ➔ A great building block within larger frameworks
- ➔ Has known limitations

What are ***higher-level*** proxy patterns?

- ➔ Accelerate development of more sophisticated applications
- ➔ Address limitations



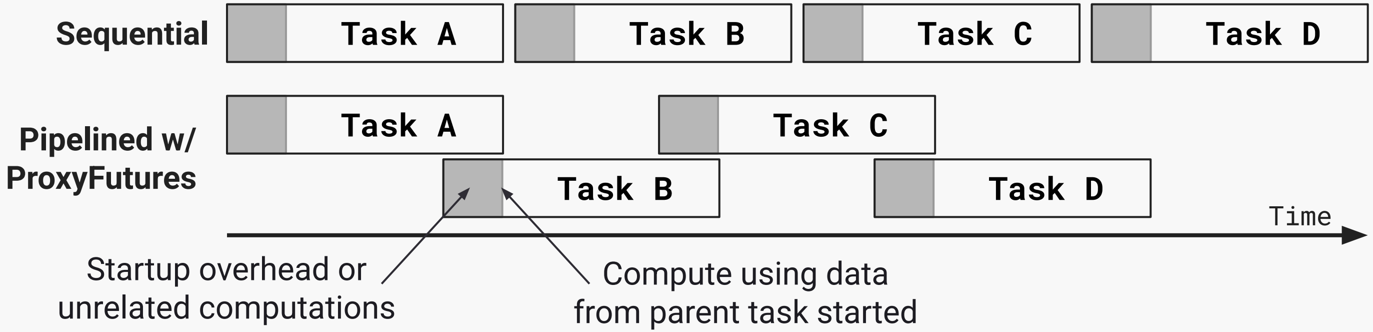
1. ProxyFutures

Futures in {Dask, Parsl, Ray, ...}

➔ Control & data synchronization tightly coupled (no optimization)
➔ Transfer mechanism fixed
➔ Not usable outside framework

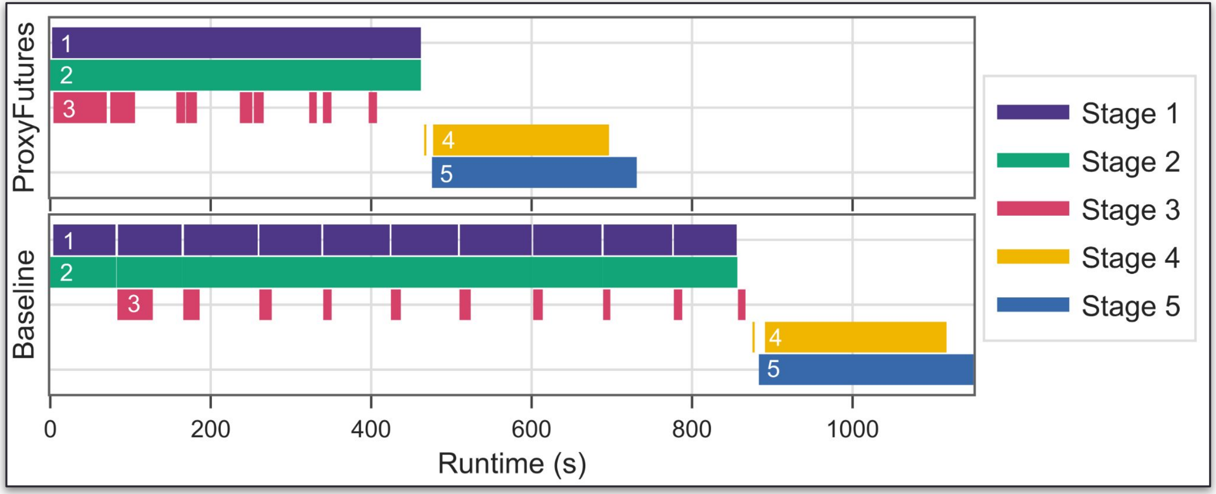
ProxyFutures

➔ Explicit & Implicit Usage
➔ Data synchronization only (good)
➔ Any transfer mechanism
➔ Framework-agnostic



1. ProxyFutures

36% reduction
in makespan!



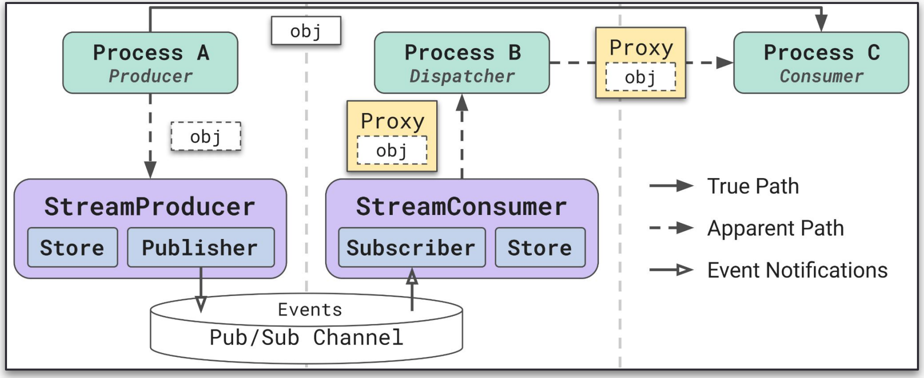
O(1000) tasks across 5 stages & complex data dependencies

1000 Genomes executed using **Globus Compute** (no task data dependency support) on Chameleon Cloud

2. ProxyStream

High-performance stream processing

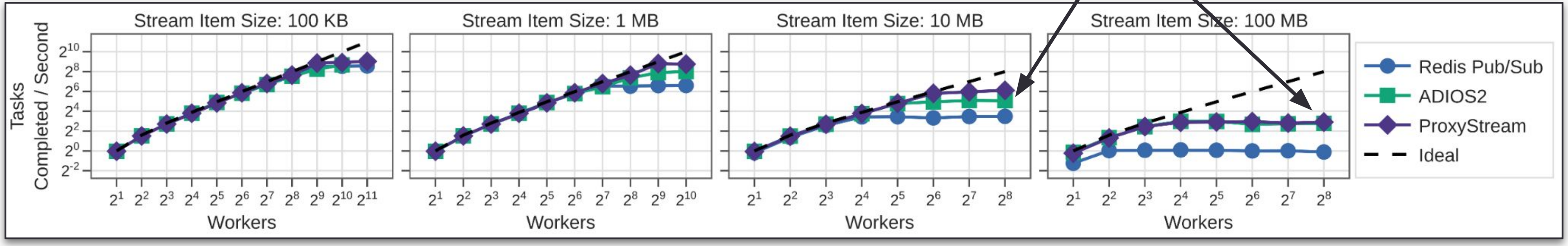
- ➔ Common in scientific computing
- ➔ Data are very large (suboptimal for Kafka-like systems)
- ➔ Quickly (1) decide if data should be used and (2) dispatch to node in cluster (e.g., for simulation)



- ➔ ProxyStream decouples metadata from bulk data transfer
- ➔ Send proxies + metadata through message broker
- ➔ Resolve proxies only when needed via more performant methods

2. ProxyStream

Performance equal to or better than state-of-the-art



Synthetic scaling test

- ➔ Stream process & dispatch
- ➔ Random data & simulated compute

DeepDriveMD

32% faster inference times
21% more inferences done

Use ProxyStream for stateful ML inference workers in pure-functional frameworks

3. Proxy Ownership

Memory Management with Proxies

➔ No reference counting—distributed reference counting in a federated environment is challenging

➔ Freeing a proxy can cause errors in other processes sharing the proxy (essentially a null-pointer exception)

➔ Forgetting to free can cause memory leaks

ProxyStore provides guidance on handling these but it's ultimately up to user

3. Proxy Ownership

Map scope of proxies to tasks.

➔ Child tasks can borrow a proxy from parent
➔ Borrowed proxy is valid for tasks' lifetime
➔ Out-of-scope proxies deleted
➔ `StoreExecutor` for easy integration of ownership with execution frameworks

Custom lifetimes for more complex scenarios.

➔ Code-segment, time-leased, and static lifetimes
➔ Extensible—create your own lifetime types

Rust Ownership/Borrowing Inspired 🦀

Enforced at Runtime

Bonus details!

Ownership Rules

1. Each object in the **`Store`** has an associated **`OwnedProxy`**
2. There can only be one **`OwnedProxy`** for any object
3. When the **`OwnedProxy`** goes out of scope the object is deleted

Reference Rules

1. At any given time, an **`OwnedProxy`** may be mutable borrowed once or immutably borrowed many times
2. An **`OwnedProxy`** and **`RefMutProxy`** are mutable references; a **`RefProxy`** is an immutable reference

Task Performance Suite

ProxyStore

Proxy Patterns

Federated Agents

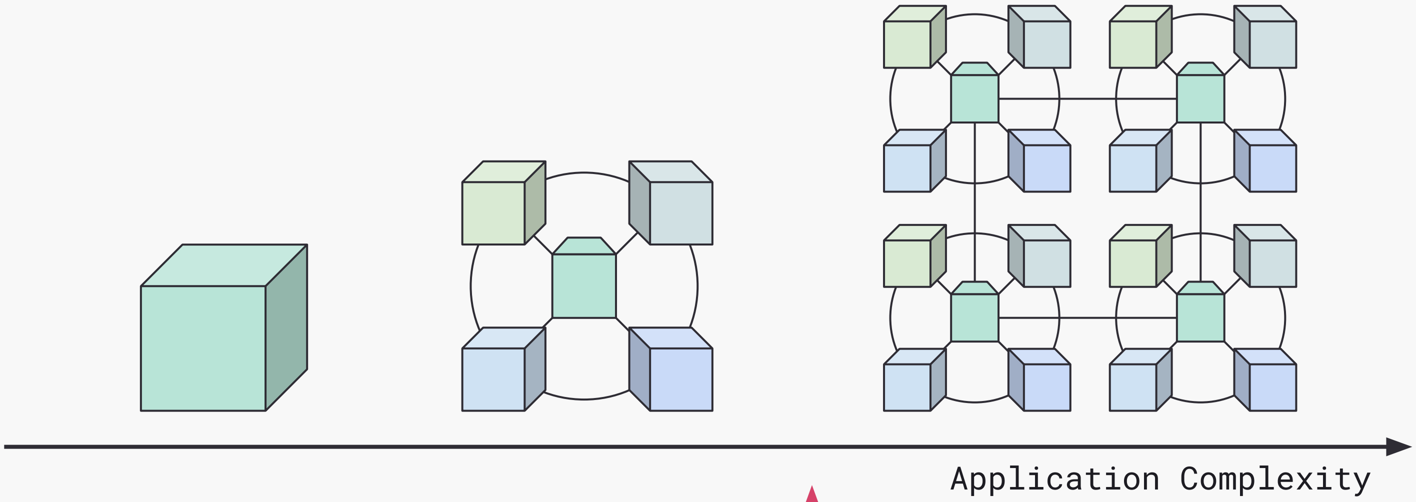
Last 2.5 years

Since candidacy (~4 months)

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

How do we build agents?

We are missing the middleware to build and connect our agents!

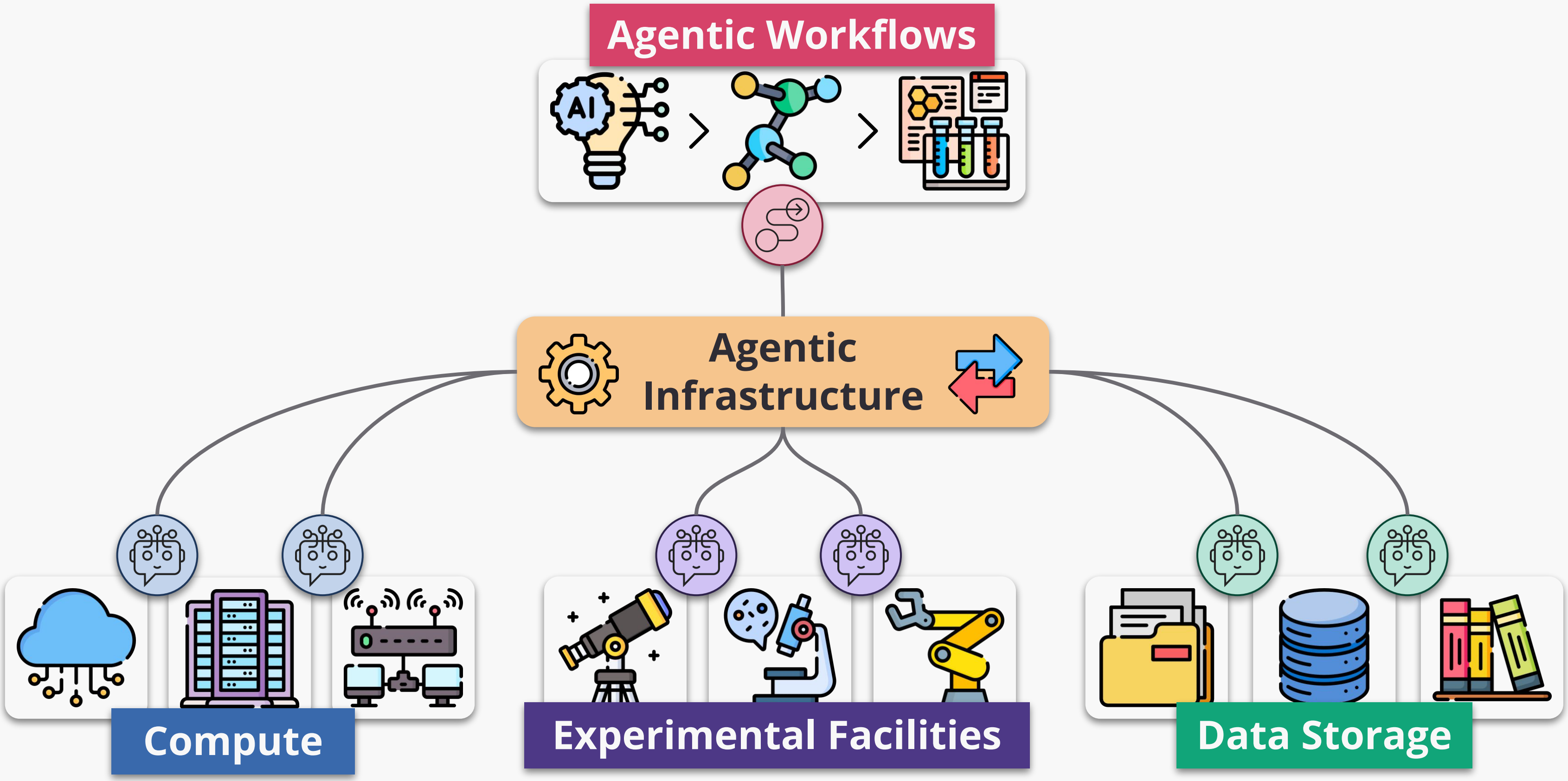
Search database, invoke code, query LLM, …

A **computational system** that can **interact** with its **environment** and **learn** from those interactions

Data repositories, HPC, robotic labs, other agents

Accumulate data, adapt processes, improve answers

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


Agentic Workflows
AI
Agentic Infrastructure
Compute
Experimental Facilities
Data Storage

Middleware Open Challenges

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

Areas we focused on…

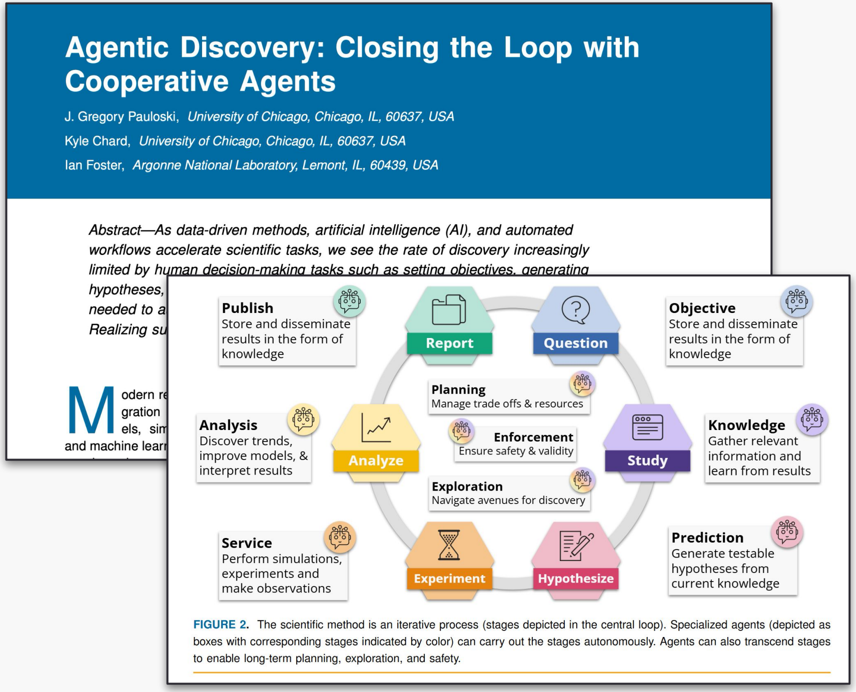


FIGURE 2. The scientific method is an iterative process (stages depicted in the central loop). Specialized agents (depicted as boxes with corresponding stages indicated by color) can carry out the stages autonomously. Agents can also transcend stages to enable long-term planning, exploration, and safety.

Under review in IEEE Computer

What does the middleware look like?

Workflows

Dask, Parsl, Pegasus

++ Task automation
++ Distributed task execution

Actor Systems

Akka, Dask, Ray

++ Stateful computation
++ Actor-to-actor interaction

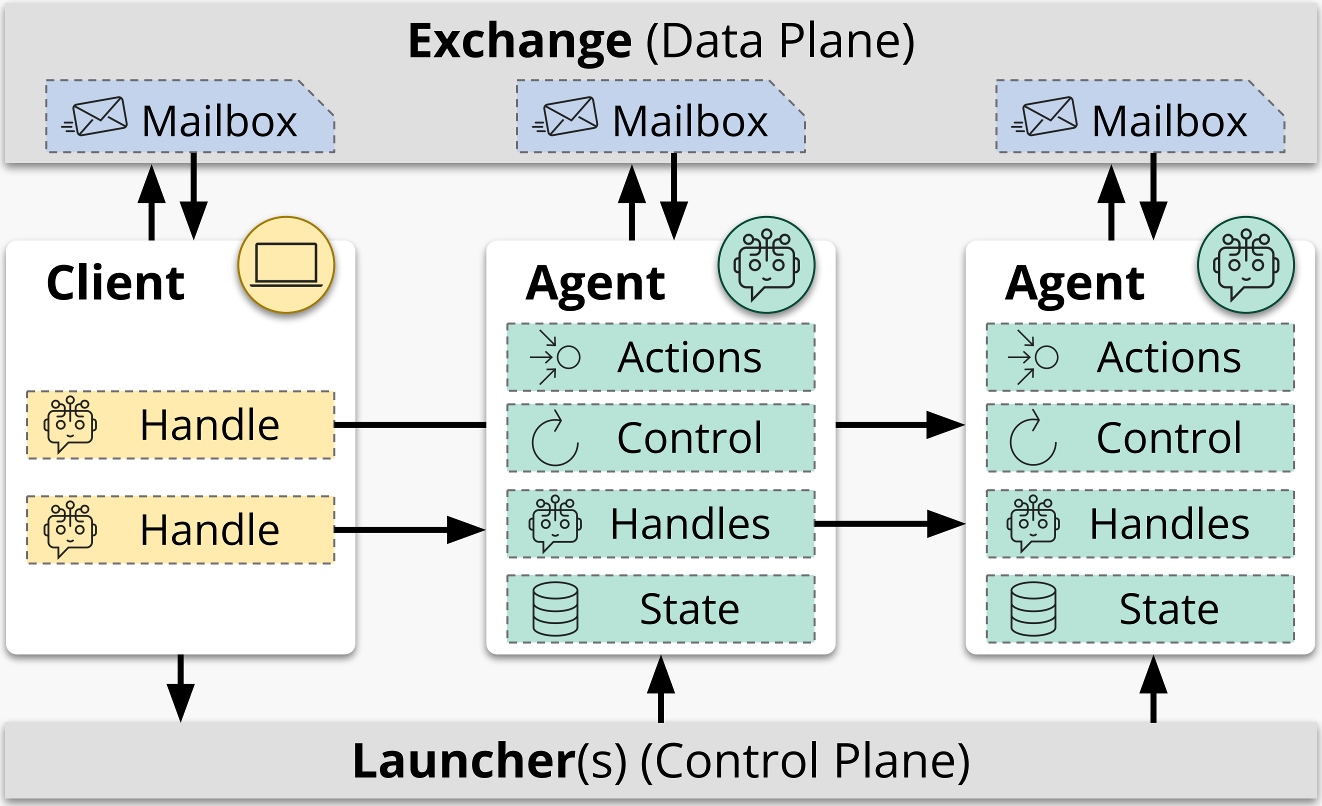
Function-as-a-Service

Globus Compute, Lambda

++ Remote execution
++ Fire-and-forget model

Academy

- *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


Exchange (Data Plane)
Mailbox
Mailbox
Mailbox
Client
Handle
Handle
Agent
Actions
Control
Handles
State
Agent
Actions
Control
Handles
State
Launcher(s) (Control Plane)

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

Launcher

- Not required but enables remote execution of agents
- Returns handle to launched agent
- Local threads or processes
- Distributed with Parsl
- Federated with Globus Compute

Agents defined by a behavior

Clients & other agents can request actions

```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)
```

Instance of a behavior is state

Control loops for autonomous behavior

Single interface for managing your agents

```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
    launcher=ThreadLauncher(),
) as manager:
    behavior = Example()  # From the prior slide
    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)
```

Choose exchange & launcher for environment

Interact with agents via handles

Pass handles to other agents

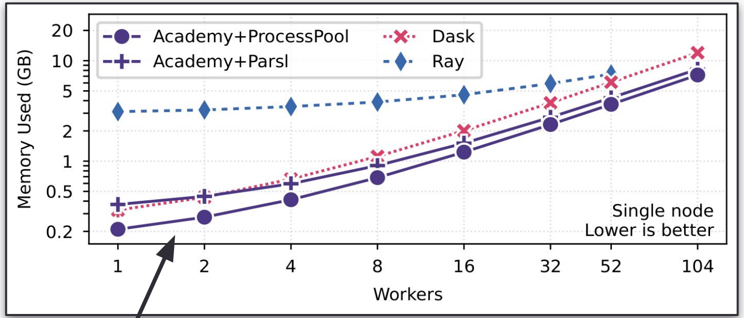
Features (rapid fire)

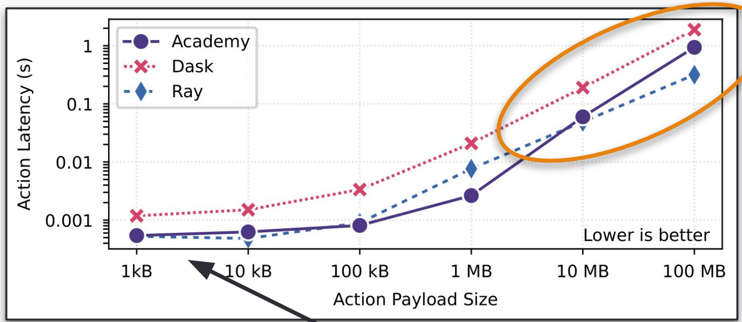
- Any number of actions & control loops
- Special purpose control loop decorators
- Multi-threaded/non-blocking action execution
- Startup and shutdown callbacks
- State persistence plugins
- Re-execution on failure
- Agents can launch other agents
- Discovery/lookup based on behavior
- Handle mailbox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

Any interesting? Ask about them at the end!

Comparisons to Actor Systems

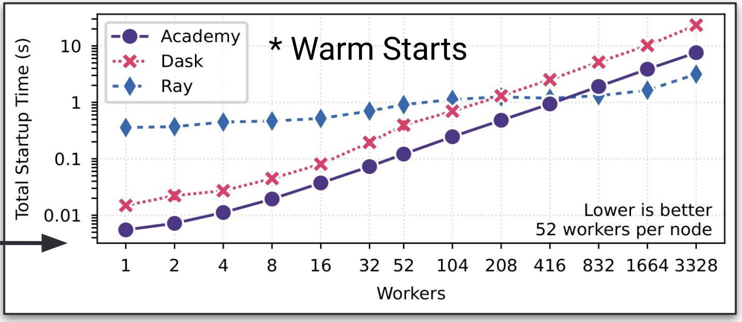
Why we need ProxyStore!





Low-memory overhead

Low-latency messaging



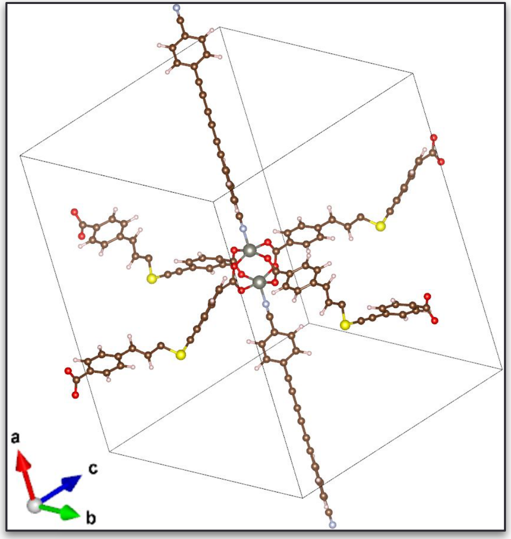
Fast start-up

Experiments performed on Aurora @ ALCF

Use Case: MOF Discovery

Metal Organic Frameworks (MOF)

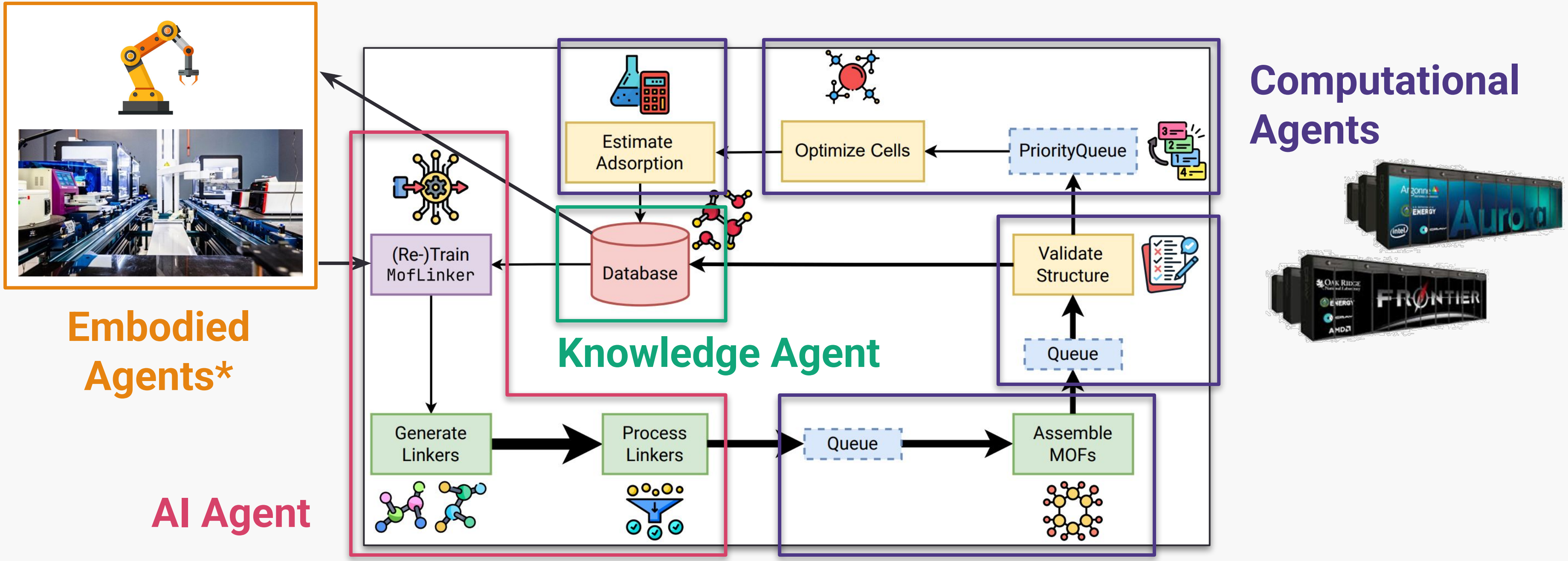
➔ Composed of organic molecules (ligands) and inorganic metals (nodes)
➔ The sponges of materials science!
➔ Porous structures that adsorb and store gases
➔ Topologies can be optimized for targeted gas storage → **Carbon Capture**



Intractable search space of ligand, node, & geometry combinations

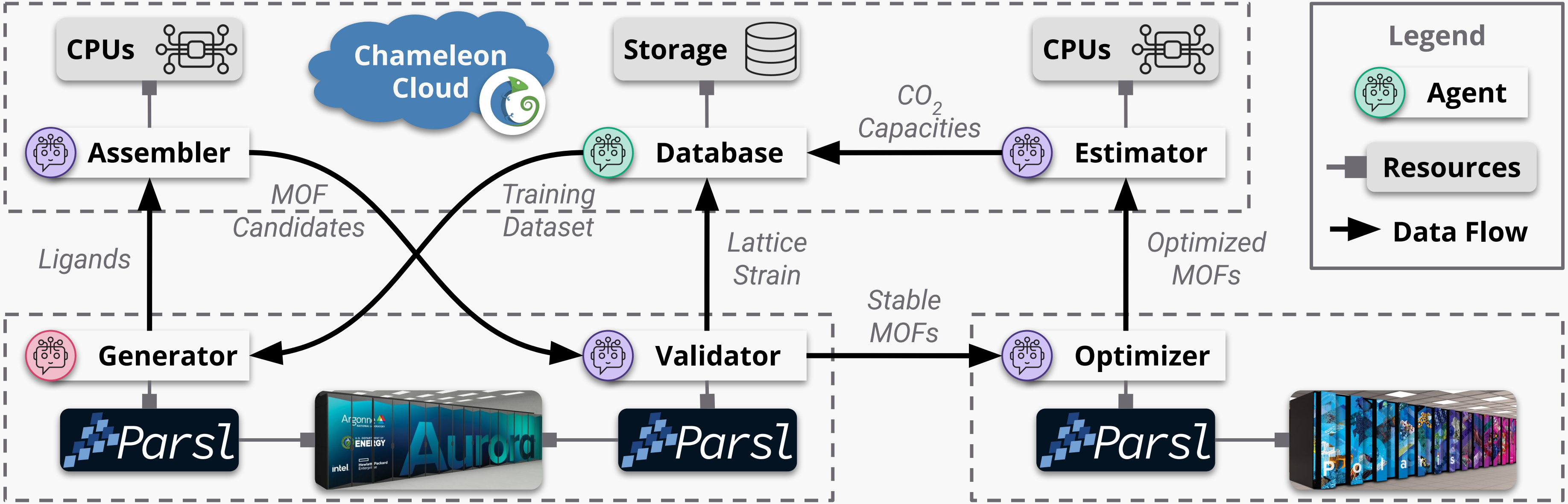
How to efficiently discover MOFs with desirable properties for target applications?

MOFA: Online learning + GenAI + Simulation

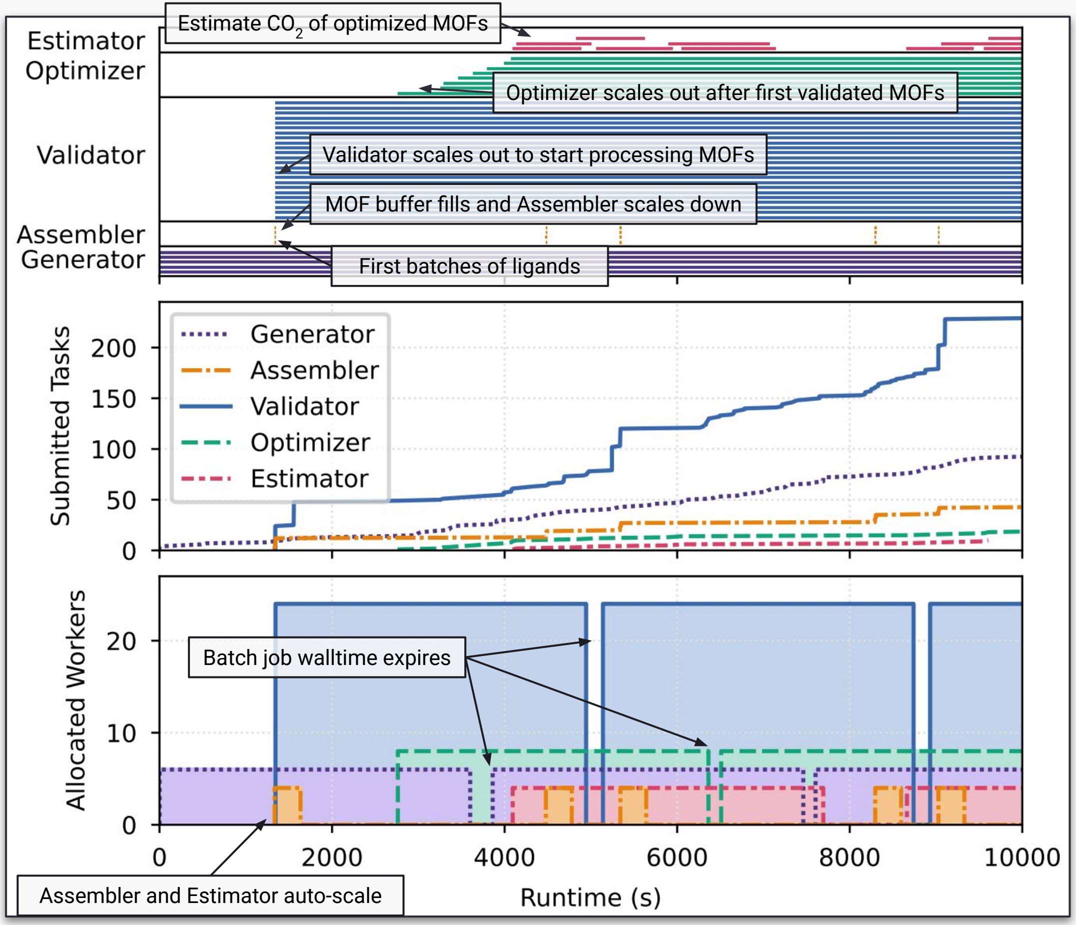


Yan et al., "**MOFA: Discovering Materials for Carbon Capture with a GenAI- and Simulation-Based Workflow**" (Under Review)

MOFA through Autonomous Agents



Agents executed remotely via Globus Compute



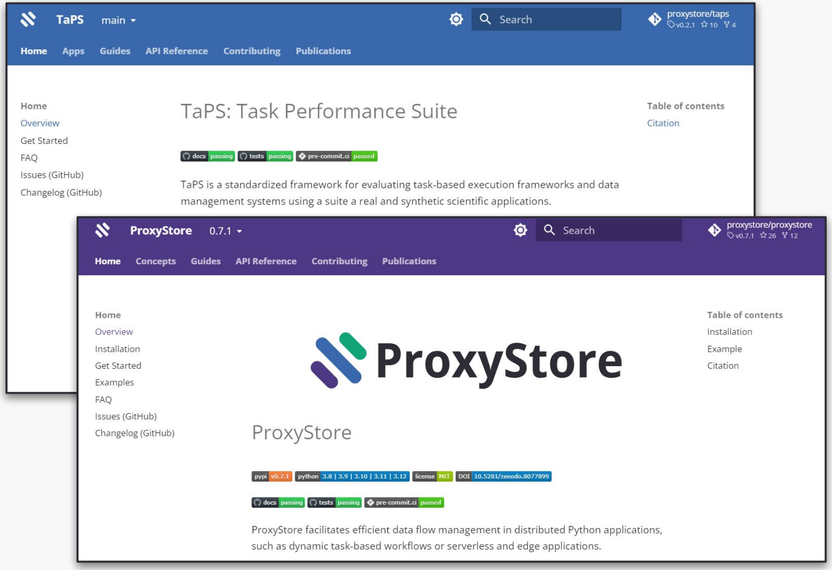
MOFA Agents Trace

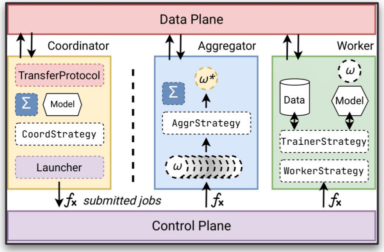
Why is this agentic model better?

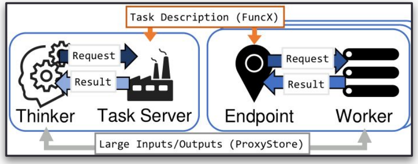
- ➔ **Placement:** Move agents to resources
- ➔ **Separation of concerns:** Resource acquisition and scaling based on local workload
- ➔ **Loose coupling:** Swap agents or integrate new agents (e.g., SDL)
- ➔ **Shared agents:** Multiple workflows can share agents (microservice-like)

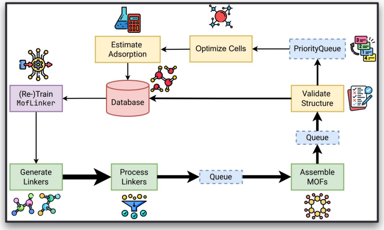
Summary

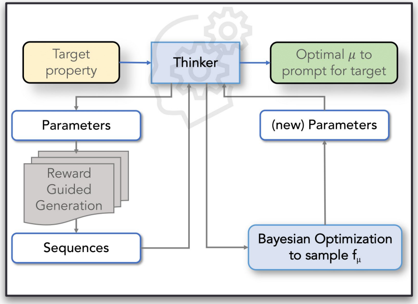
Impact

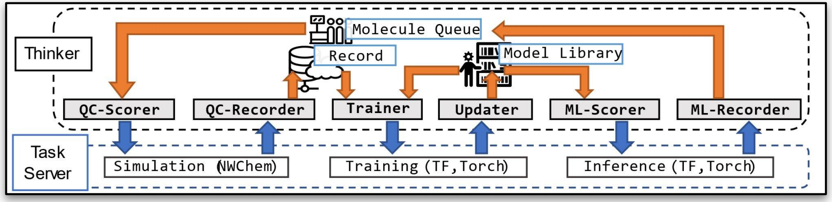












Empowering large-scale science through open-source software

Summary

New programming techniques ***enable and accelerate*** *task-centric* ***science applications*** *executed* ***across the computing continuum****.*

P1	**TaPS**: Support research in distributed/parallel execution	eScience '24 (Best Paper)
P2	**ProxyStore**: Better object references for federated environments	SC '23 & HPPSS '24
P3	**Proxy Patterns**: Better data flow patterns with object proxies	TPDS '24
P4	**Federated Agents**: Build science agents for autonomous discovery	*IEEE Computer* & SC '25**

Better, easier, & faster science! — MLHPC '21, IJHPCA '23, HCW '23, IJHPCA '24, CCGRID '25 & Others In Review/Progress

**Under Review / In Progress*

Programming the Continuum

Towards Better Techniques for Developing Distributed Science Applications

New programming techniques enable and accelerate task-centric science applications executed across the computing continuum.

- ➔ **TaPS** [eScience '24]
- ➔ **ProxyStore** [SC '23 & HPPSS '24]
- ➔ **Proxy Patterns** [TPDS '24]
- ➔ **Federated Agents** [IEEE Computer* & SC '25*]

Better, easier, & faster science!

MLHPC '21, IJHPCA '23, HCW '23, IJHPCA '24, CCGRID '25 & Others In Review/Progress

Questions?

github.com/proxystore

Contact:

J. Gregory Pauloski
jgpauloski@uchicago.edu

Reference:

github.com/proxystore
docs.proxystore.dev
taps.proxystore.dev

Acknowledgements:

- Argonne National Laboratory under U.S. Department of Energy Contract DE-AC02-06CH11357
- National Science Foundation under Grant 2004894 and Grant 2209919
- ExaLearn Co-design Center of the Exascale Computing Project (17-SC-20-SC)