Mochi

http://www.mcs.anl.gov/research/projects/mochi/

Vision

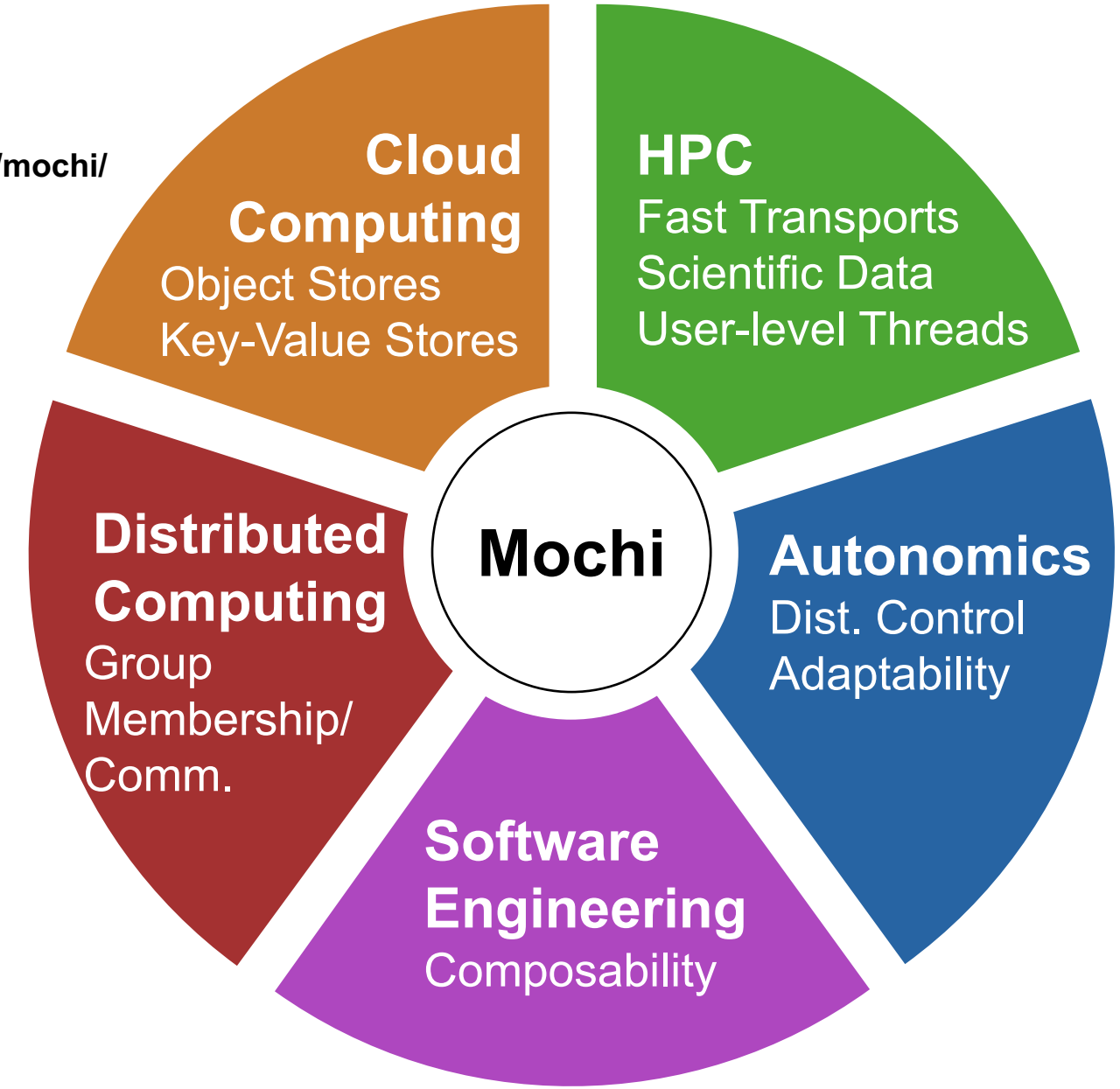
Lowering the barriers to distributed services in computational science.

Approach

- Familiar models (key/value, object, file)
- Easy to build, adapt, and deploy
- Lightweight, user-space components
- Modern hardware support

Impact

- Better, more capable services for specific use cases on high-end platforms
- Significant code reuse
- Ecosystem for service development



Carnegie Mellon University · HDF · Los Alamos National Laboratory EST. 1943 · Argonne National Laboratory

Mochi: What are we trying to accomplish?

We're trying to transform HPC data services from a monoculture to an ecosystem.

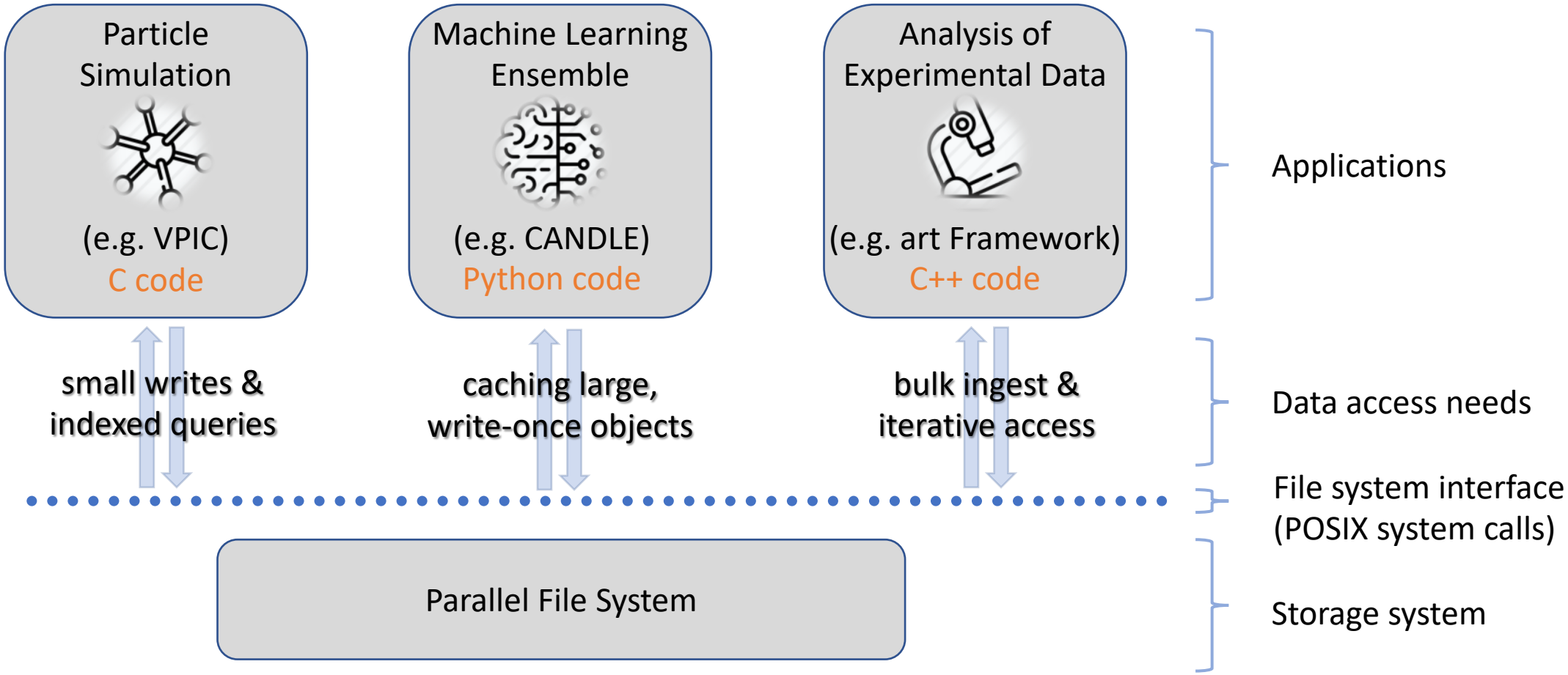
- Redefining how teams design and develop distributed services for use in HPC systems.
- Providing a portable "programming model" for these services.
- Providing a set of core building blocks.
- Demonstrating the methodology and tools with DOE science use cases.

We're trying to foster a community of service developers.

- Developing a set of training materials that will help others employ the tools.
- Making all these building blocks available to the larger community.

How is this traditionally done in HPC?

File system monoculture for data (dis)service

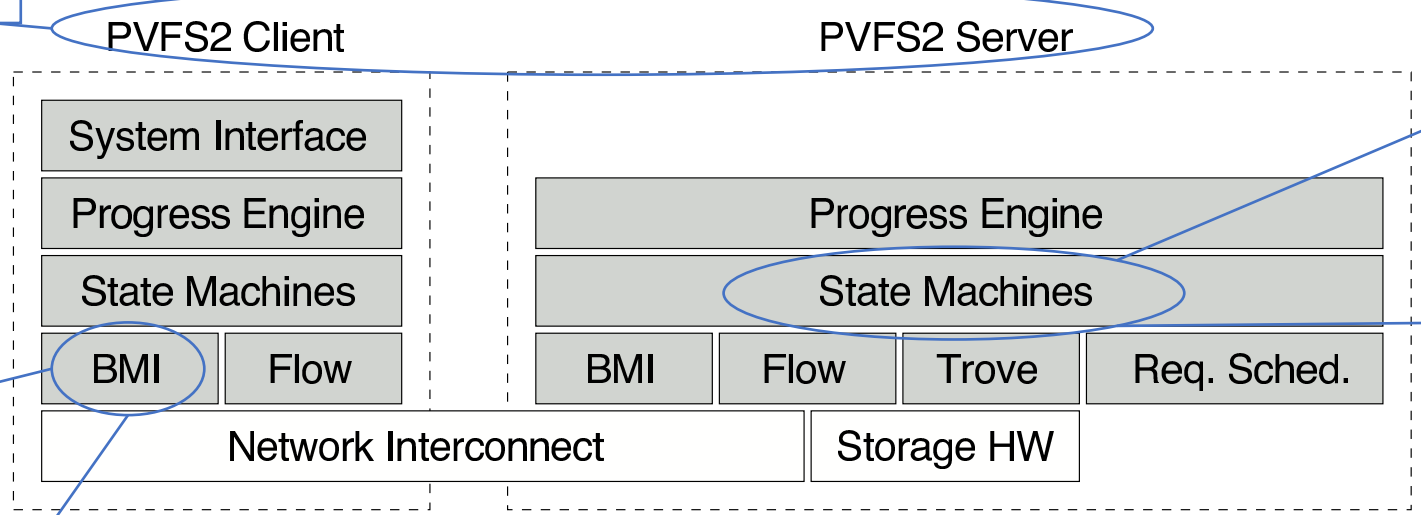


All applications use the same "one size fits all" file system interfaces, semantics, and policies for data access.

How is this traditionally done in HPC?

The internals of a parallel file system (PVFS2 example)

1. All applications bend to the same service: single data model (string of bytes), decomposition (blocks), and consistency.



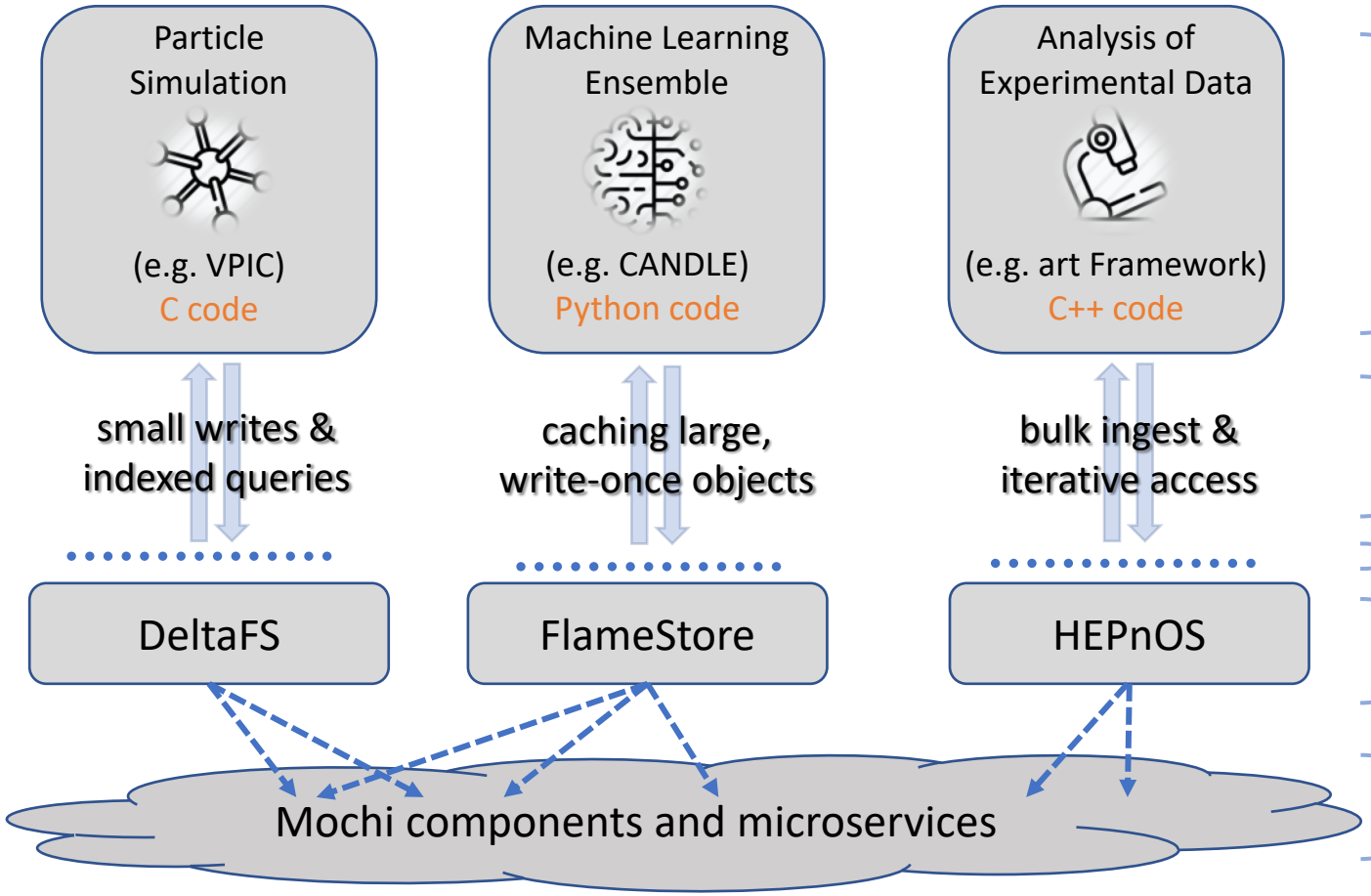
3. Programming is accomplished with proprietary state machine language. Effective but steep learning curve.

2. Portable networking layer (BMI) is the only component usable separate from PVFS. Service developers build RPC abstraction on top of it.

(Many) other researchers employ PVFS in their work, but it is difficult to build a *new* service from it.

What's new in the Mochi approach?

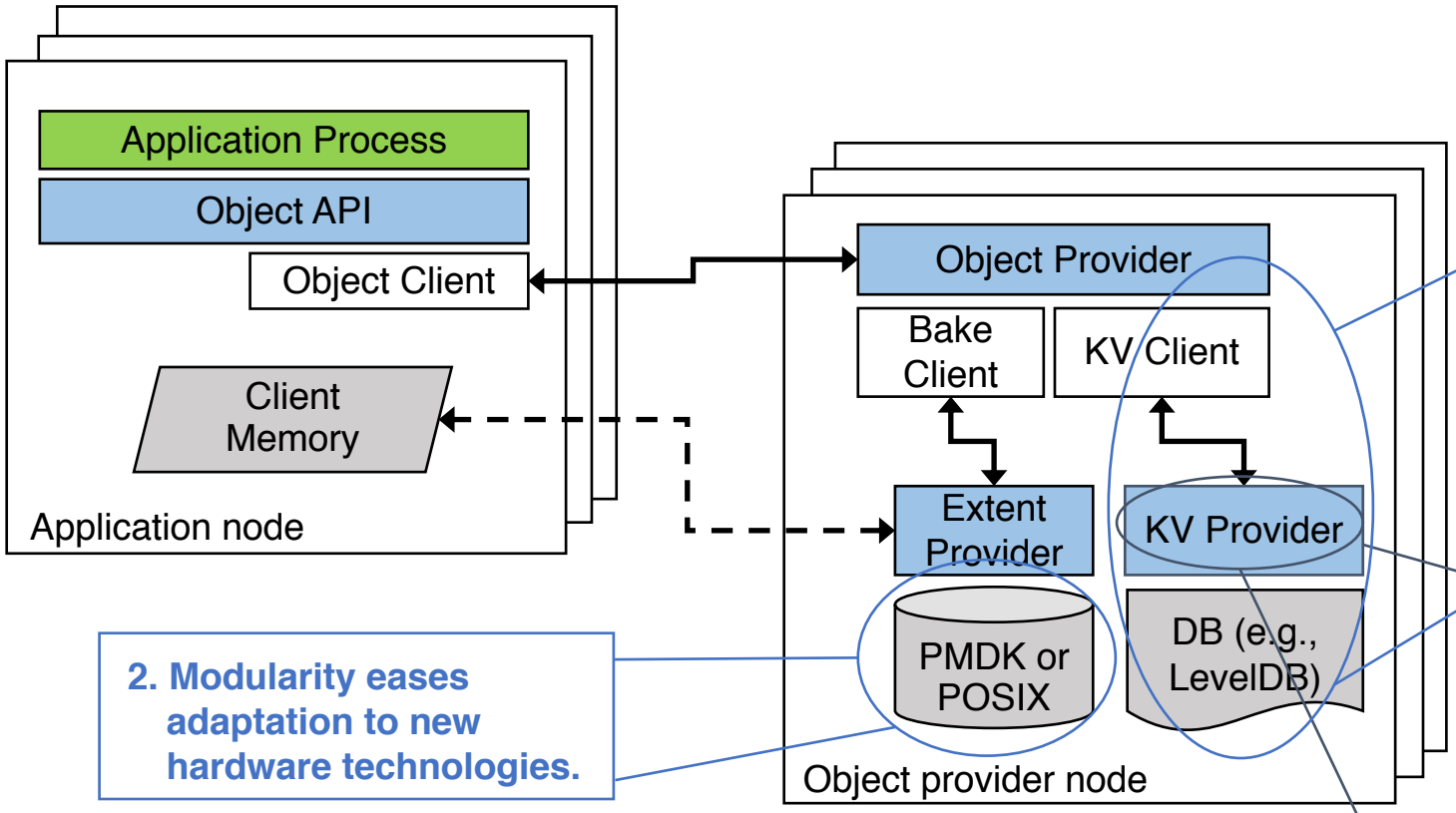
An ecosystem of services co-existing and reusing functionality



This approach has allowed us to simultaneously pursue multiple specialized service implementations.

Instead of "one size fits all", Mochi data services present tailored interfaces, semantics, and policies for data access while still leveraging robust building blocks.

What's new in the Mochi approach?

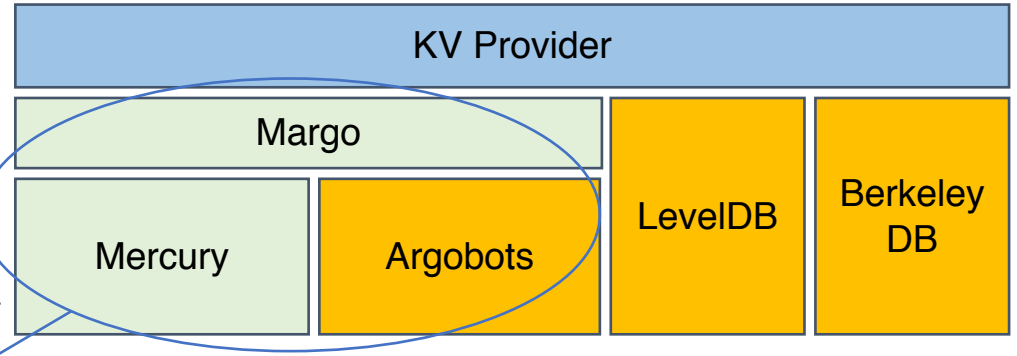


1. Core functionality developed as stand-alone components and "microservices", cleanly reusable in different configurations and products.

2. Modularity eases adaptation to new hardware technologies.

3. Multiple methods of programming (C, C++, Python), more accessible.

4. Portable RPC communication library designed for multi-service environments



Example Mochi Composed Services

Particle Trajectory Assembly (DeltaFS)

Goals

- Extreme scale file system metadata
- In-situ indexing with fast file retrieval

Features

- Specialized directories to efficiently support trillions of files in a single directory
- Software routing to scalably manage connections
- POSIX(-like) API

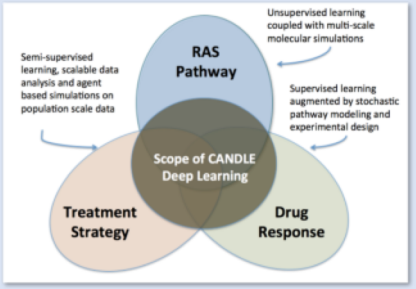
Transient Storage for Deep Neural Networks (FlameStore)

Goals

- Store deep neural network models during a deep learning workflow
- Retain most promising candidate models

Features

- Flat namespace
- Python API (Keras models)



Fast Event-Store for High-Energy Physics (HEPnOS)

Goals

- Manage physics event data through multiple analysis phases
- Retain data in the system to accelerate analysis

Features

- Write-once, read-many
- Hierarchical namespace (datasets, runs, subruns)
- C++ API

Carnegie Mellon University HDF Los Alamos NATIONAL LABORATORY EST.1943 Argonne NATIONAL LABORATORY

Mochi: Core Components

A programming environment for distributed services

Mercury

Portable RPC comm.

- Support for shared memory and multiple HPC transports: IB, GNU, PSM2, …
- RDMA for bulk data movement
- Busy wait or wake on network event

https://mercury-hpc.github.io/

Argobots

Lightweight runtime for concurrent execution.

- Utilize HW and OS constructs for performance
- hwloc-aware
- Custom scheduling and placement

http://www.argobots.org/

Margo

Simple service development.

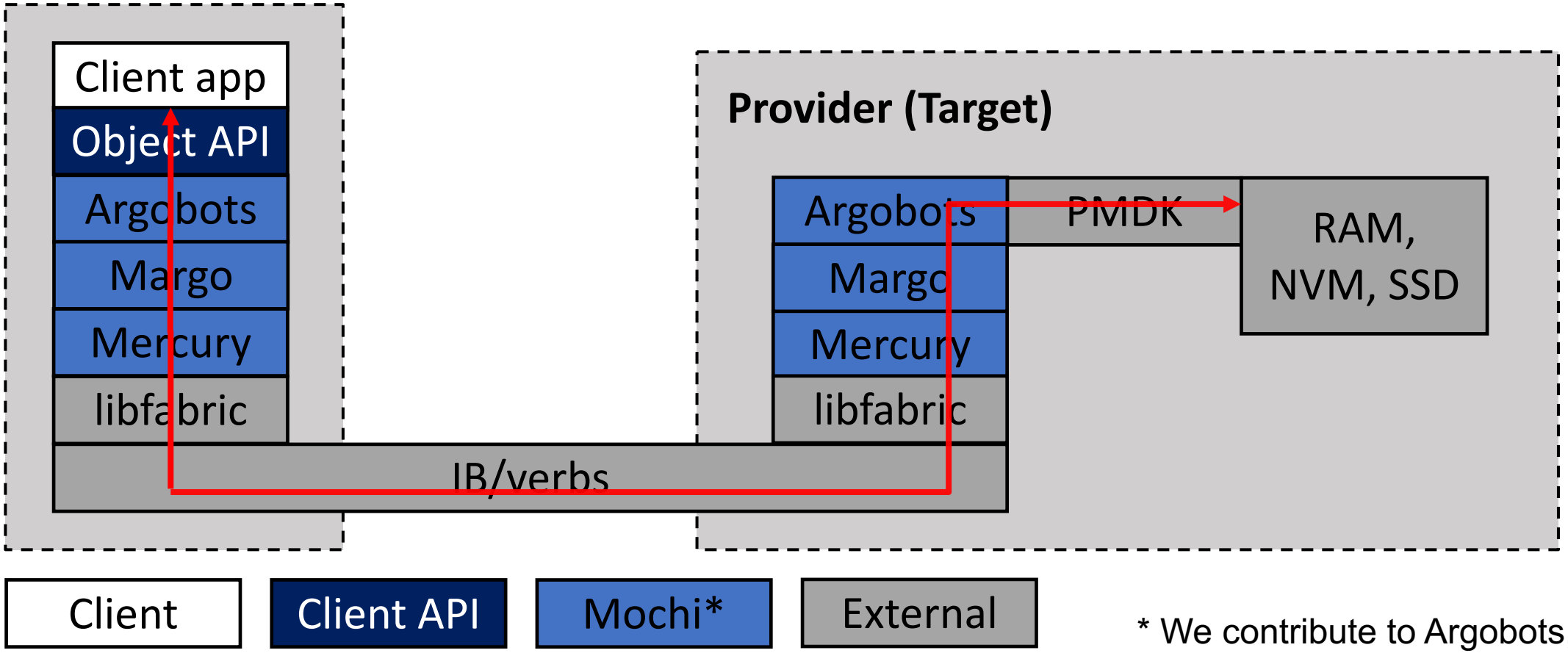
- Multi-threaded model
- Lightweight threads created to handle RPCs
- Express Mercury operations as blocking functions
- Uses Argobots to manage concurrency

https://xgitlab.cels.anl.gov/sds/margo

… and Thallium, if writing in C++!

Microservice Example: BAKE

A Composed Service for Remotely Accessing Objects



* We contribute to Argobots, but it's primarily supported by P. Balaji's team.

P. Carns et al. "Enabling NVM for Data-Intensive Scientific Services." INFLOW 2016, November 2016.

Agenda

8:30 – 9:00	Welcome and Introductions	Rob Ross
9:00 – 9:20	Summit System Details	Rob Latham
9:20 – 10:00	Mochi Core: Margo and Thallium	Matthieu Dorier
10:00 – 10:30	*Break*	
10:30 – 11:00	Core: SSG	Shane Snyder
11:00 – 11:20	Microservices: Bake	Phil Carns
11:20 – 12:00	Microservices: SDSKV and SDSDKV	Rob Latham
12:00 – 1:00	*Lunch*	
1:00 – 1:30	Composing Mochi Services	Matthieu Dorier
1:30 – 1:45	Mochi Best Practices	Phil Carns
1:45 – 3:00	Mochi Service Hacking Hands-On	
3:00 – 3:30	*Break*	
3:30 – 4:15	Hands-On Continued	
4:15 – 4:30	Recap, Feedback, and Next Steps	

Mochi Components: A Reference

	Component	Summary
Core		
	Argobots	Argobots provides user-level thread capabilities for managing concurrency.
	Mercury	Mercury is a library implementing remote procedure calls (RPCs).
	Margo	Margo is a C library using Argobots to simplify building RPC-based services.
	Thallium	Thallium allows development of Mochi services using modern C++.
	SSG	SSG provides tools for managing groups of providers in Mochi.
Utilities		
	ABT-IO	ABT-IO enables POSIX file access with the Mochi framework.
	ch_placement	ch-placement is a library implementing multiple hashing algorithms.
	MDCS	MDCS exposes remotely accessible counters for monitoring purposes.
	Shuffle	Shuffle provides a scalable all-to-all data shuffling service.
Microservices		
	BAKE	Bake enables remote storage and retrieval of named blobs of data.
	POESIE	Poesie embeds language interpreters in Mochi services.
	REMI	REMI is a microservice that handles migrating sets of files between nodes.
	SDSKV	SDSKV enables RPC-based access to multiple key-value backends.
	SDSDKV	SDSDKV provides a distributed key-value service using Mochi components.

Mochi Components: Core

Argobots

Argobots provides user-level thread capabilities for managing concurrency.

Argobots implements a lightweight runtime system that supports computation and data movement with massive concurrency. It leverages the lowest-level constructs in the hardware and OS: lightweight notification mechanisms, data movement engines, memory mapping, and data placement strategies.

Location	https://github.com/pmodels/argobots
Programming Language	C
Dependencies	
Status	Production
Python Wrappers	

Mercury

Mercury is a library implementing remote procedure calls (RPCs).

Mercury supports the execution of remote procedures in a variety of scenarios, including over high-performance network fabrics. Mercury also provides facilities to support handling of large RPC arguments (i.e., bulk data), including taking advantage of RDMA when possible.

Location	https://github.com/mercury-hpc/mercury
Programming Language	C
Dependencies	Libfabric (or other supported communication library)
Status	Production
Python Wrappers	

Carnegie Mellon University HDF Los Alamos NATIONAL LABORATORY EST.1943 Argonne NATIONAL LABORATORY

Margo

Margo is a C library using Argobots to simplify building RPC-based services.

Margo includes a set of Argobots-aware wrappers that translate Mercury's event-driven communication model into a more intuitive sequential user-level thread model. Margo implements a communication progress loop that consolidates best practices for polling and communication, and manages threads to execute handler functions for each RPC request.

Location	https://xgitlab.cels.anl.gov/sds/margo
Programming Language	C
Dependencies	Argobots, Mercury
Status	Production
Python Wrappers	https://xgitlab.cels.anl.gov/sds/py-margo

Thallium

Thallium is a C++14 library wrapping Margo allowing development of RPC-based services using all the power of modern C++.

Thallium wraps all Margo, Mercury, and Argobots concepts into C++ classes with automated memory management, replaces Mercury's serialization mechanism with a template-based mechanism, and uses modern C++'s variadic templates to turn RPCs into callable objects.

Location	https://xgitlab.cels.anl.gov/sds/thallium
Programming Language	C++14
Dependencies	Argobots, Margo, Mercury
Status	Production
Python Wrappers	

Scalable Service Groups (SSG)

SSG provides tools for managing groups of providers in Mochi.

SSG aids in the development of distributed services by allowing the tracking of groups of providers and notification of change in the group (e.g., losing a participant). The current SSG implementation leverages the SWIM weakly consistent group membership protocol to detect failures and evict noncommunicating members of the group.

Location	https://xgitlab.cels.anl.gov/sds/ssg
Programming Language	C
Dependencies	Margo, PMIX (optional), MPI (optional)
Status	Prototype
Python Wrappers	https://xgitlab.cels.anl.gov/sds/py-ssg

Mochi Components: Utilities

ABT-IO

ABT-IO enables POSIX file access with the Mochi framework.

ABT-IO works by managing blocking POSIX file operations with an Argobots execution stream so that file I/O can proceed concurrent with other Mochi operations.

Location	https://xgitlab.cels.anl.gov/sds/abt-io
Programming Language	C
Dependencies	Argobots
Status	Production
Python Wrappers	

ch-placement

ch-placement is a library implementing multiple hashing algorithms.

ch-placement can be used by a distributed storage system to access multiple hashing algorithms, distance metrics, and virtual node settings using a consistent API. This is typically used for mapping data and/or work across multiple providers.

Location	https://xgitlab.cels.anl.gov/codes/ch-placement
Programming Language	C
Dependencies	
Status	Production
Python Wrappers	

Mochi Diagnostic Counter Service (MDCS)

MDCS exposes remotely accessible counters for monitoring purposes.

Performance understanding can be complex in distributed systems. MDCS provides a base capability for remote counter access helpful in the development of monitoring capabilities for Mochi services.

Location	https://xgitlab.cels.anl.gov/sds/mdcs
Programming Language	C
Dependencies	Margo
Status	Prototype
Python Wrappers	

Three Hop All-to-All Shuffle

Shuffle provides a scalable all-to-all data shuffling service.

Shuffle reduces the memory/network resource load of an all-to-all communication pattern by adding additional hierarchy (3 hops: local in, remote, and local out). Shuffle provides an asynchronous "enqueue and return" API. It supports RPC batching and implements its own flow control mechanisms on top of Mercury.

Location	https://github.com/pdlfs/deltafs-shuffle
Programming Language	C/C++ with a C API
Dependencies	Mercury, MPI
Status	Production
Note	Related libraries: Nexus and Mercury-progressor

Mochi Components: Microservices

Bake

Bake enables remote storage and retrieval of named blobs of data.

Bake is a microservice for storing blobs on nonvolatile memory or file-based storage backends, using the PMDK object API for efficient storage to nonvolatile memory backends and POSIX calls for file-based backends. Bake does not itself provide sharding across multiple backends/instances.

Location	https://xgitlab.cels.anl.gov/sds/bake
Programming Language	C
Dependencies	Margo, PMDK or POSIX
Status	Production for PMDK backend, prototype for POSIX file backend
Python Wrappers	https://xgitlab.cels.anl.gov/sds/py-bake

Poesie

Poesie embeds language interpreters in Mochi services.

Poesie clients can send code (e.g., Lua, Python) to Poesie providers for remote execution on their behalf, providing a flexible method of extending an RPC-based service. Currently we are evaluating Poesie as a method for enabling rapid reconfiguration of Mochi composed services.

Location	https://xgitlab.cels.anl.gov/sds/poesie
Programming Language	C
Dependencies	Margo, Lua and/or Python
Status	Prototype
Python Wrappers	

REsource Migration Interface (REMI)

REMI is a microservice that handles migrating sets of files between nodes.

REMI works in terms of *filesets*, collections of files in the directory hierarchy. REMI uses RDMA and memory mapping to efficiently transfer potentially large filesets at once, managing creation of the hierarchy at the destination and notification on completion. This provides a basic capability for data management useful for constructing adaptive or hierarchical data services.

Location	https://xgitlab.cels.anl.gov/sds/remi
Programming Language	C++
Dependencies	Thallium
Status	Prototype
Python Wrappers	https://xgitlab.cels.anl.gov/sds/py-remi

sds-keyval (SDSKV)

SDSKV enables RPC-based access to multiple key-value backends.

SDSKV is a recognition of the importance of KV stores in modern data management. SDSKV enables RPC-based access to multiple KV backends, including LevelDB, Berkeley DB, and in-memory databases. SDSKV is typically helpful for metadata and small data management. SDSKV does not itself provide sharding across multiple backends/instances.

Location	https://xgitlab.cels.anl.gov/sds/sds-keyval
Programming Language	C/C++
Dependencies	LevelDB, Berkeley DB, or other backend KV store
Status	Production
Python Wrappers	https://xgitlab.cels.anl.gov/sds/py-sdskv

SDSDKV

SDSDKV provides a distributed key-value service using Mochi components.

SDSDKV leverages SDSKV, SSG, and ch-placement to provide key-value services sharded over multiple instances on a variety of different backends. SSG allows for detection of failure of members of the service. Current implementation is not resilient to faults.

Location	https://xgitlab.cels.anl.gov/sds/sdsdkv
Programming Language	
Dependencies	ch-placement, SDSKV, SSG
Status	Prototype
Python Wrappers	

This work is in part supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357; in part supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative; and in part supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program.

http://www.mcs.anl.gov/research/projects/mochi/