Mochi: an Approach to Composable Data Services

October 13, 2021

Neelam Bagha and Jerome Soumagne
The HDF Group

Bob Robey, Brad Settlemyer and Galen Shipman
Los Alamos National Laboratory

Philip Carns, Matthieu Dorier, Kevin Harms, Robert Latham, Pierre Matri, Rob Ross and Shane Snyder
Argonne National Laboratory

George Amvrosiadis, Chuck Cranor and Greg Ganger
Carnegie Mellon University

The HDF Group

Carnegie Mellon University

Los Alamos NATIONAL LABORATORY EST. 1943

Argonne NATIONAL LABORATORY

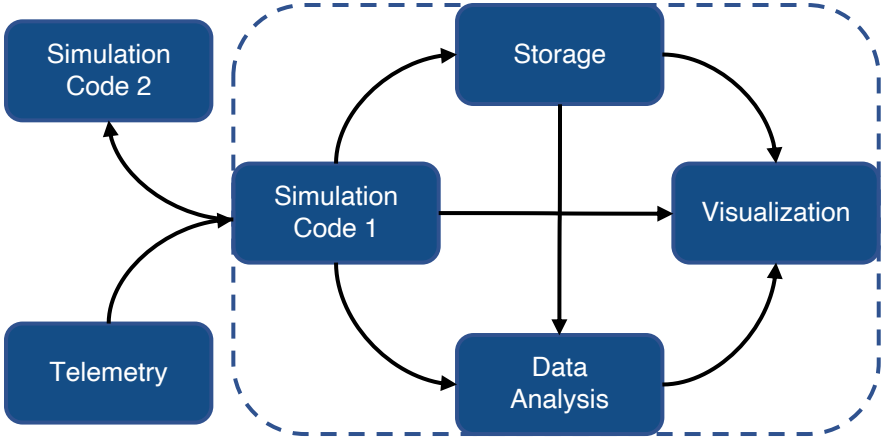
Outline

- **Why do we need data services?**
- **What is new in our approach?**
- **How do we know we're on the right track?**
- **What challenges do we face?**

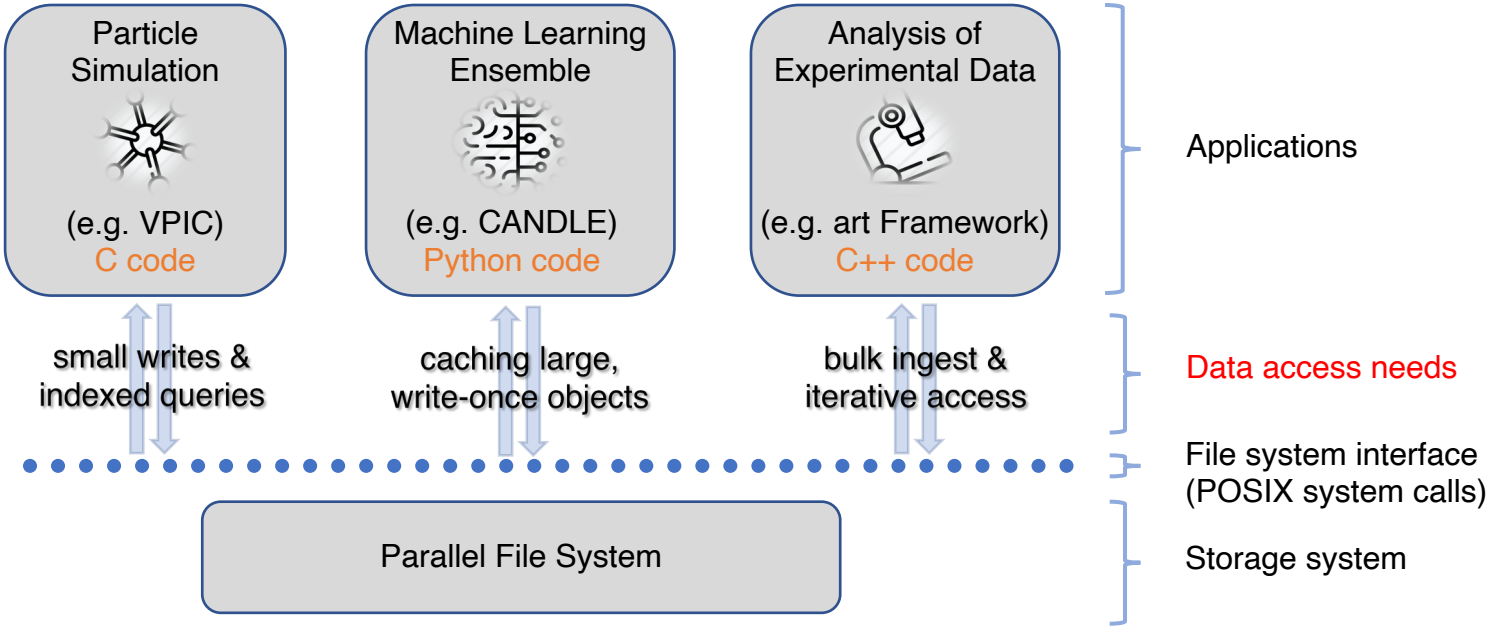
- **Terminology**
  - What is a data service
    - Component / set of components that provide a feature / set of features to the user in response to an application need
  - Monolithic data service ⇔ Micro-services

Evolution of HPC Workflows

- **Traditional workflow around monolithic parallel file system**
- **No longer the case – there is a need for more complex workflows**
- **Creating a data service is no easy task**
  - Any HPC data service must face similar challenges
    - Communication between applications
    - Resilience and fault tolerance
    - Deployment
    - Security



File system monoculture for data (dis)service

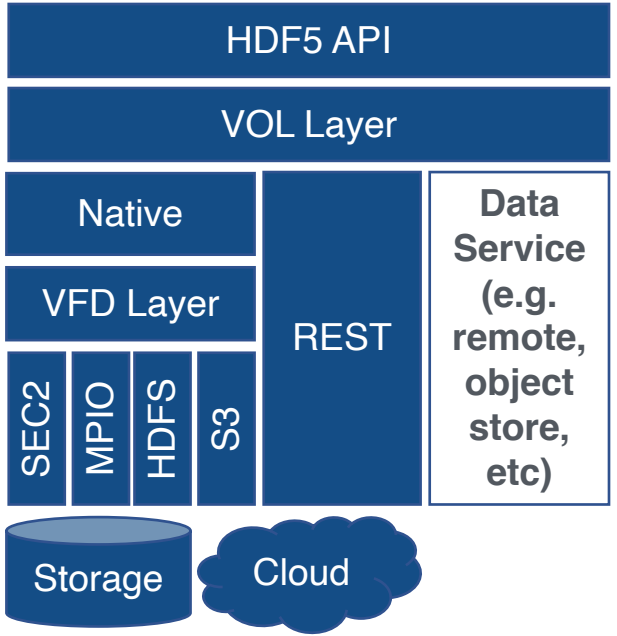


All applications use the same "one size fits all" file system interfaces, semantics, and policies for data access

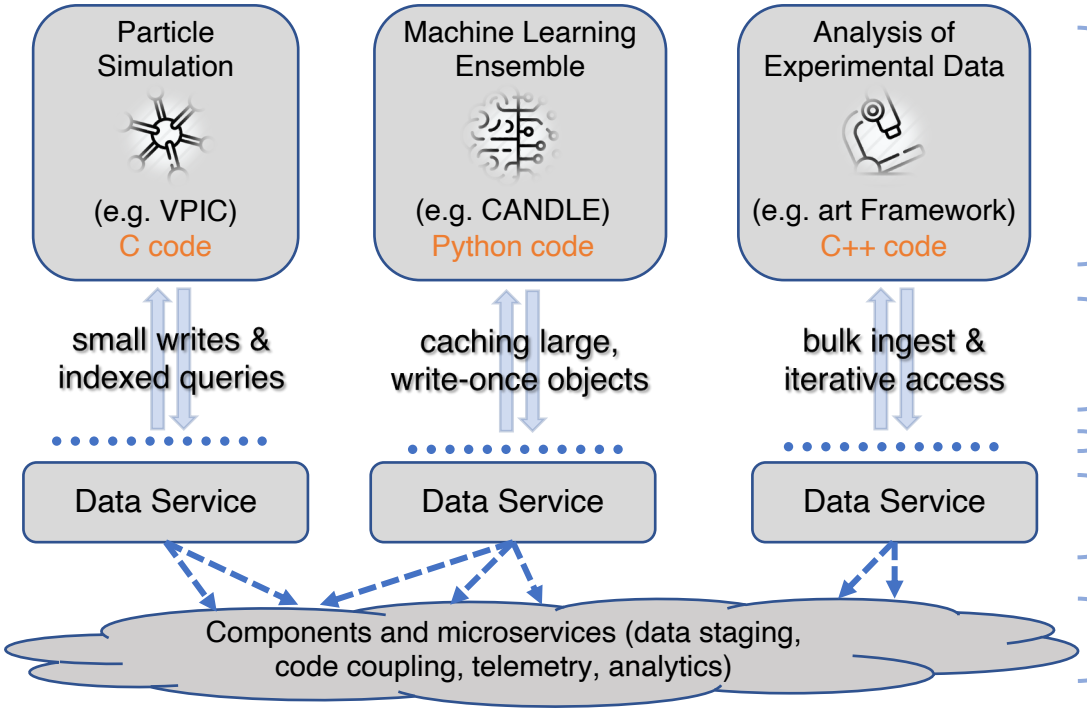
Credit: Phil Carns

HDF5 and VOL – Interface to Data Services

- **Virtual Object Layer**
  - Adapted to application data model
    - VFL too tied to native format
  - Plug-in / connector architecture to provide data service on-demand
  - Can easily switch from one connector to another and fulfill application needs



Ecosystem of services co-existing and reusing functionality

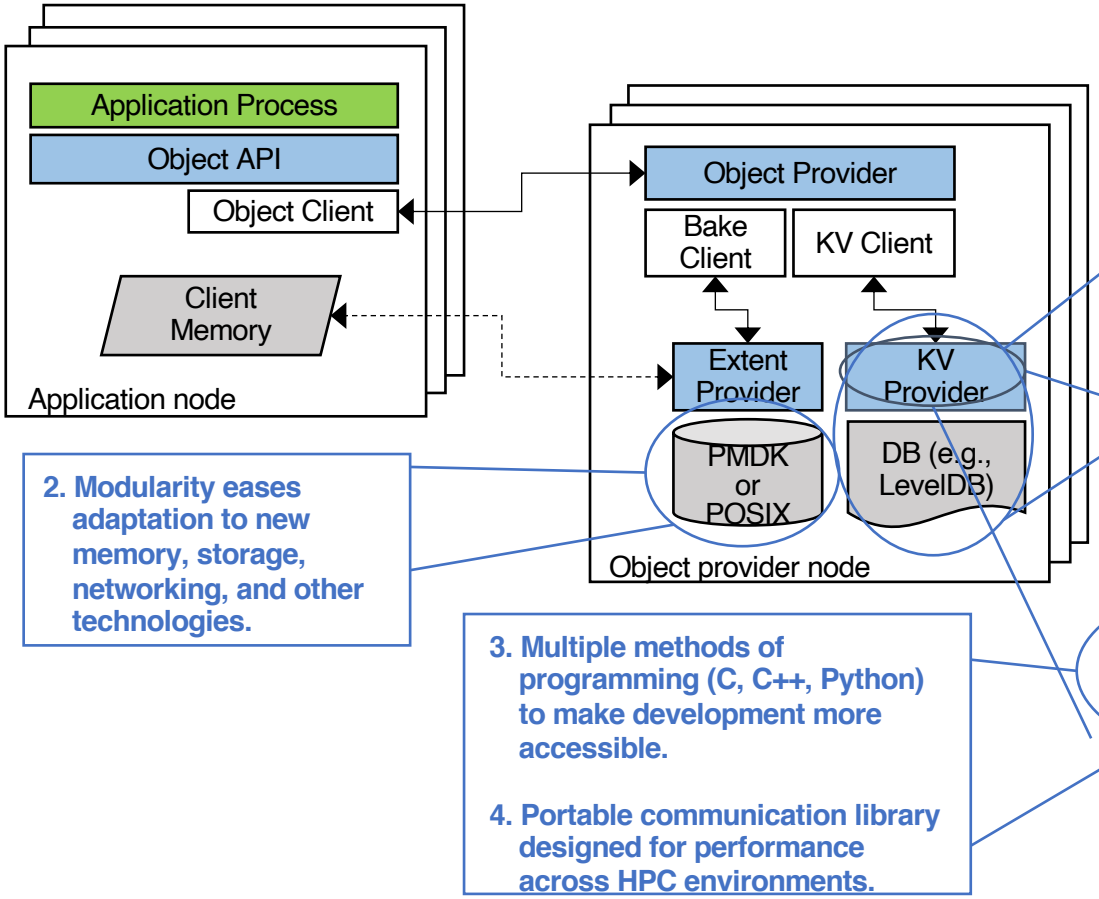


This approach allows to simultaneously pursue multiple specialized service implementations

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

Credit: Phil Carns

The principles behind Mochi



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

2. Modularity eases adaptation to new memory, storage, networking, and other technologies.

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

4. Portable communication library designed for performance across HPC environments.

Credit: Rob Ross

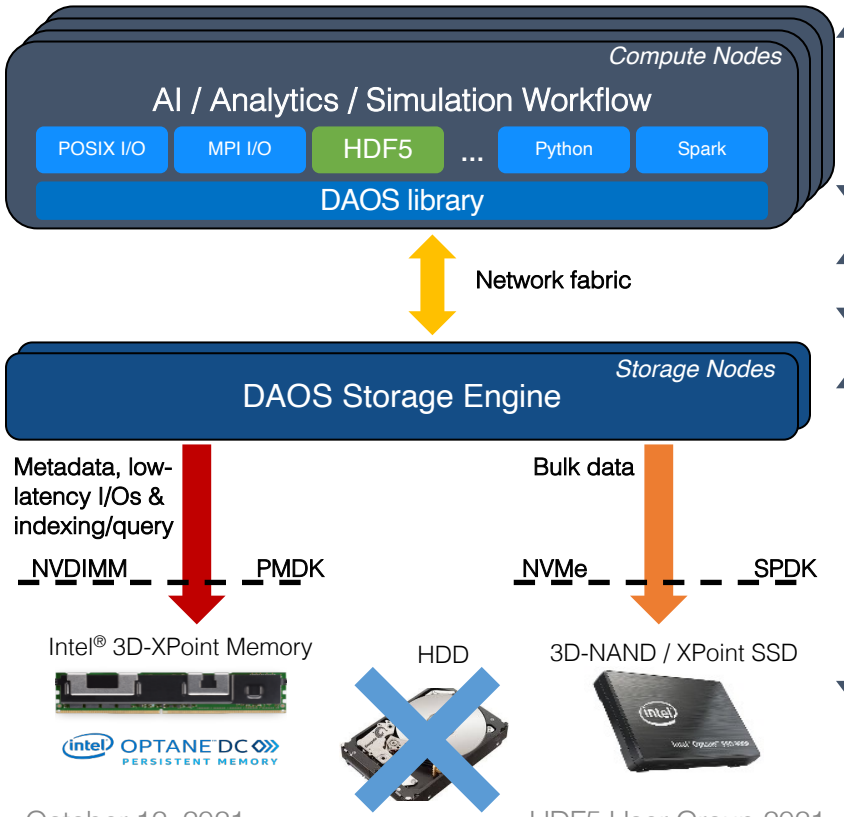
Components that are available today

	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.
	Bedrock	Bedrock is a bootstrapping and configuration system for Mochi components.
	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.
	Sonata	Sonata is a Mochi service for JSON document storage based on UnQLite.

How do we know we're on the right track?
Other projects using Mochi

Service	Institution(s)	Summary
Chimbuko	Brookhaven	Workflow-level scalable performance trace analysis tool
DAOS	Intel	Object store that provides high bandwidth, low latency, and high I/O operations per second (IOPS) storage containers to HPC applications
DataSpaces	Univ. of Utah	Programming system and data management framework for coupled workflows
GekkoFS	Univ. of Mainz	Temporary distributed file system for HPC applications
Hermes	IIT, THG, UIUC	User-space platform for distributing data structures
HXHIM	Los Alamos	Hexadimensional hashing indexing middleware
Proactive Data Containers	Berkeley	Object-centric data management system to take advantage of deep memory and storage hierarchy
Seer	Los Alamos	Lightweight in situ wrapper library adding in situ capabilities to simulations
Unify	LLNL and ORNL	Suite of specialized, flexible file systems that can be included in a user's job
	Kitware	Platform for ubiquitous access to visualization results during runtime
Dist. Systems Coursework	Tsukuba	Creating distributed systems using RPC and RDMA as part of an Information Systems course

Intel ® DAOS



- DAOS library directly linked with the applications
- No need for dedicated cores
- Low memory/CPU footprint
- End-to-end OS bypass
- KV API, non-blocking, lockless, snapshot support

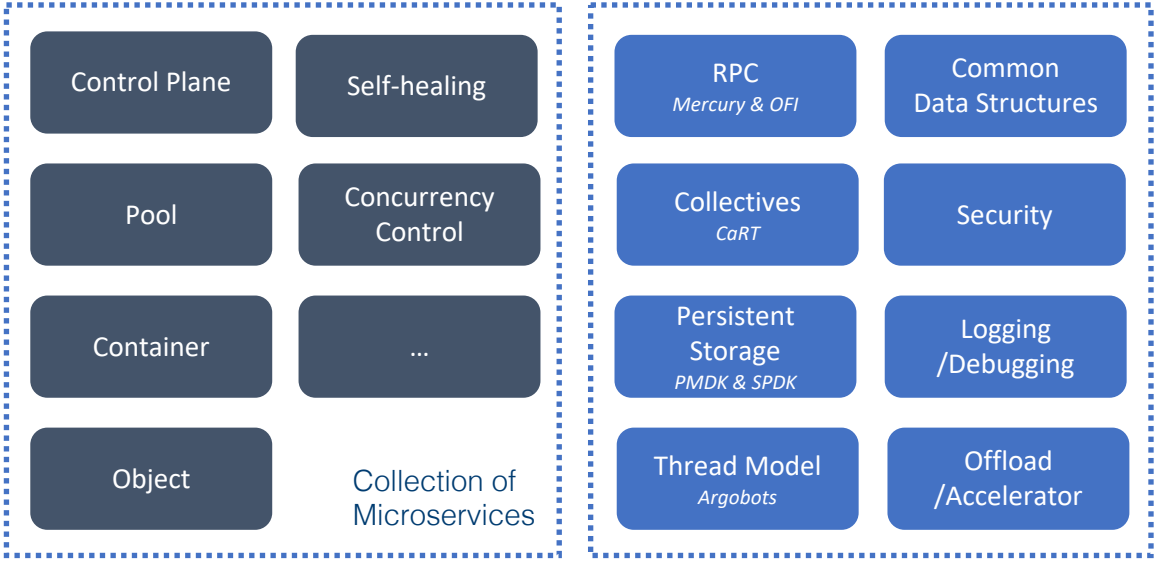
- Low-latency & high-message-rate communications
- Native support for RDMA & scalable collective operations
- Support for Infiniband, Slingshot, etc through OFI libfabric

- Fine-grained I/O with media selection strategy
- Only application data on SSD to maximize throughput
- Small I/Os aggregated in pmem & migrated to SSD in large chunks
- Full user space model with no system calls on I/O path
- Built-in storage management infrastructure (control plane)
- NFSv4-like ACL

Credit: Mohamad Chaarawi

Intel ® DAOS

- **Responds to application data access needs**
- **Extendable through microservice architecture**
- **Fine-grained I/O with media selection strategy**

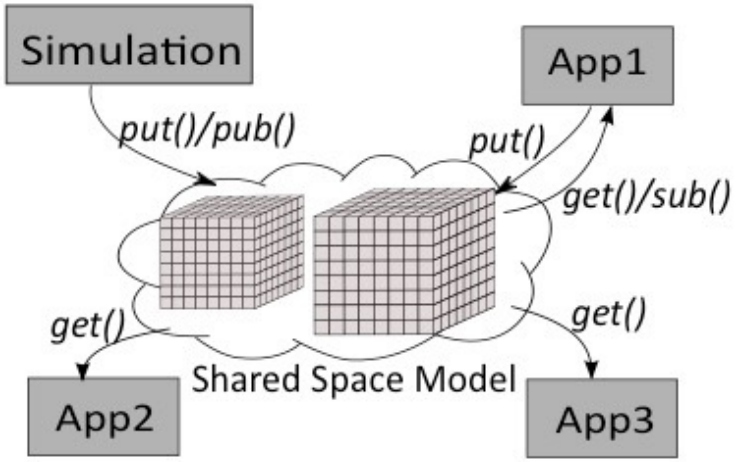


Credit: Mohamad Chaarawi

DataSpaces

- **Responds to code coupling and data staging needs**

- **In-memory data staging for coupling workflow components**
- **Optimized for HPC workflows**

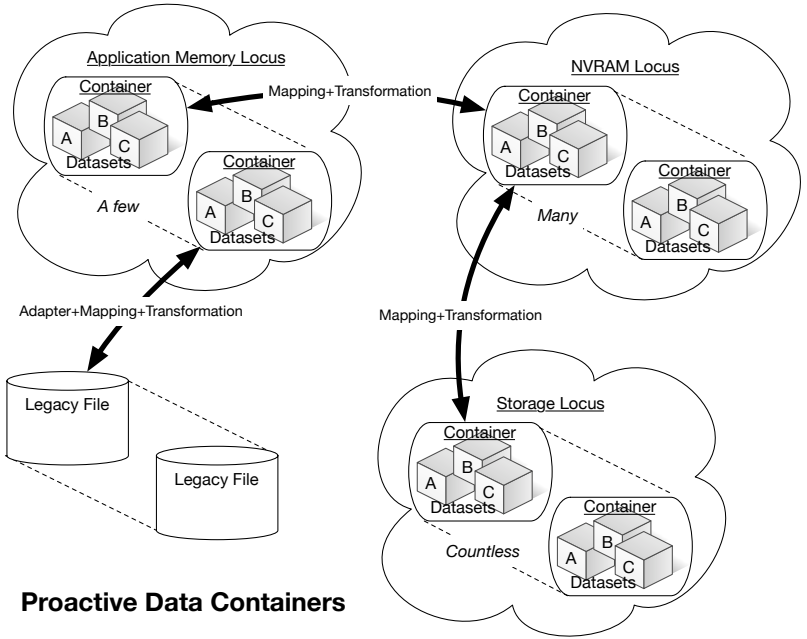


Abstract DataSpaces storage model

Credit: Philip Davis

Proactive Data Containers (PDC)

- **Responds to application data access needs**
- **Explore next generation storage systems and interfaces**
- **Object centric storage**
- **Support for extracting information from data**
  - Information management
  - Simulation time analytics
  - Interaction among multiple datasets



Proactive Data Containers

Credit: Suren Byna

Advantages and Challenges

- **Multiple services = Customization of environment**
  - Add value to vendor-provided capabilities
- **Services can allow for re-usability of functionality**
- **Complexity of deep layers complicates performance tuning**
  - Tailoring to applications has performance wins, but diagnosing and tuning requires additional tools.
- **Gaining the trust of users and facilities**
  - Teams can be reticent of trusting new services with their data, especially when long-term sustainability of software can be uncertain.

Conclusion

- **Scientific Achievement**
  - Mochi has enabled numerous DOE computer science teams, and industry, to more rapidly build new data services through a thoughtful design methodology and reusable components.

- **Significance and Impact**
  - Data services traditionally took many years to develop and productize. The Mochi project is shortening this development cycle, allowing teams to develop services specialized to their needs while still enabling significant component reuse.

- **Technical Approach**
  - Built using proven remote procedure call (RPC), remote direct memory access (RDMA), and user-level threading
  - Define methodology for design of services using common components wherever possible
  - Provide numerous typical capabilities via reusable components
  - Exploring learning approaches to configuration and optimization

Acknowledgments / Questions

- **Mochi**
  - https://www.mcs.anl.gov/research/projects/mochi/
- **Mercury RPC**
  - https://mercury-hpc.github.io

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.