MCS CS SEMINAR SERIES

Argonne NATIONAL LABORATORY

MOCHI PROJECT OVERVIEW:

THE DEMOCRATIZATION OF DATA SERVICES IN HPC

PHIL CARNS
Mathematics and Computer Science Division
Argonne National Laboratory

June 13, 2023
Virtual presentation

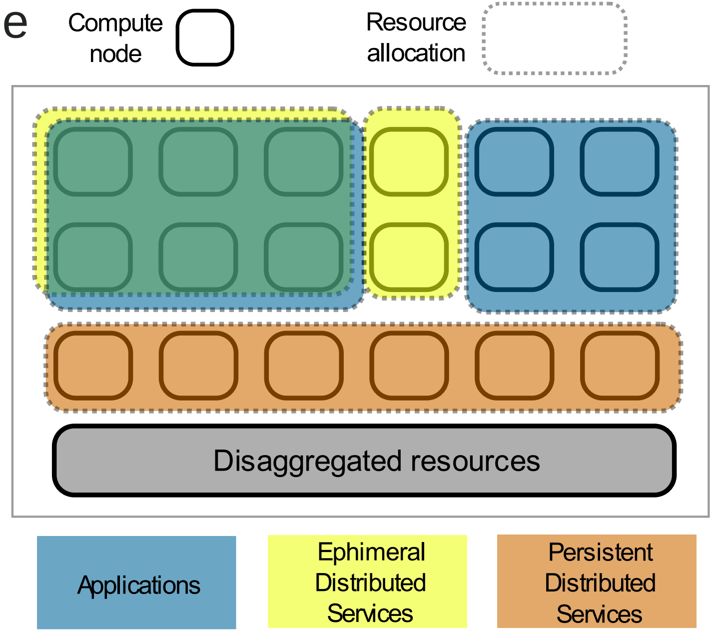
U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

DISTRIBUTED SERVICES IN HPC

DISTRIBUTED SERVICES IN HPC

and the modern runtime ecosystem

- A distributed service aggregates a collection of on-demand capabilities into a coherent whole, outside of the scope of an application.
- This is an increasingly important part of the runtime ecosystem for scientific computing.
- Why is this concept useful?
  - Manage state beyond the lifetime of a single application execution
  - Mediate shared access to that state
  - Decouple functionality from the application (i.e., “software disaggregation”)
  - Enable access to off-node resources



PARALLEL FILE SYSTEMS

The most successful examples of distributed services in HPC

- Anyone who has used a large-scale HPC system has also used a parallel file system.
- They are usually presented as large, high-performance storage volumes.
- Intel’s DAOS is the newest option; it’s not really a parallel file system, but rather a much more flexible distributed object store.
- Parallel file systems are *mission critical* due to their broad use and stewardship of persistent data.
- Fortunately, there are several mature, sophisticated designs available!

PVFS2 PARALLEL VIRTUAL FILE SYSTEM

daos

IBM Spectrum Scale

lustre®

IT'S A LOT OF RESPONSIBILITY

Design implications for broad PFS adoption

Crouching Figure of Atlas, Baldassare Tommaso Peruzzi (MET, 1992.304)

- A platform- or facility-wide file system must present a general-purpose API (usually POSIX files and directories).
- Conservative semantics are needed for the set of applications that *might* need it (e.g., directory locking for concurrent renames "just in case").
- The software must be sophisticated to manage concurrent storage, network, and server access, redundancy, security, high concurrency, and much more.
- The Unix/Linux OS model calls for file systems to be closely tied to the operating system for coherent access control.

Against all odds: parallel file systems are incredibly successful!
Why would we want anything different?

THE CASE FOR (DATA) SERVICE SPECIALIZATION

Is one file system enough?

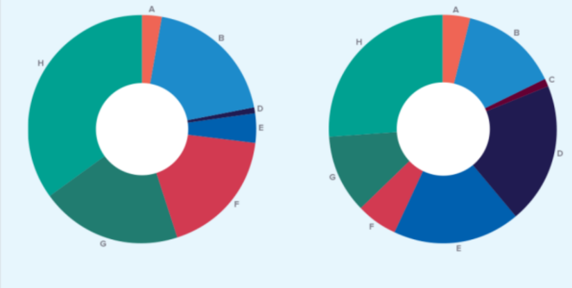
INCITE/ALCC

2022 INCITE NODE HOURS: **17.8M**

2022 ALCC NODE HOURS: **6.8M**

2022 by Domain

Domain	INCITE	ALCC
A. Biological Sciences	3 %	4 %
B. Chemistry	9	14
C. Computer Science	—	1
D. Earth Science	1	20
E. Energy Technologies	4	18
F. Engineering	18	6
G. Materials Science	20	11
H. Physics	35	26



- Today's scientific computing landscape is characterized by a wild diversity of applications, often several combined into one workflow.
- It's not difficult to imagine that many of these could be better served by a special-purpose service built to suit their use case, API, and semantics.
- Simultaneously, hardware is changing quickly: how can we quickly incorporate node-local storage, smart devices, and persistent memory?
- PFSs have historically required decades of development and support. It's not plausible for many science communities to take this on.

https://ar22.alcf.anl.gov/science/allocation-programs

… ENTER MOCHI

WHAT IS MOCHI, EXACTLY?

Mochi

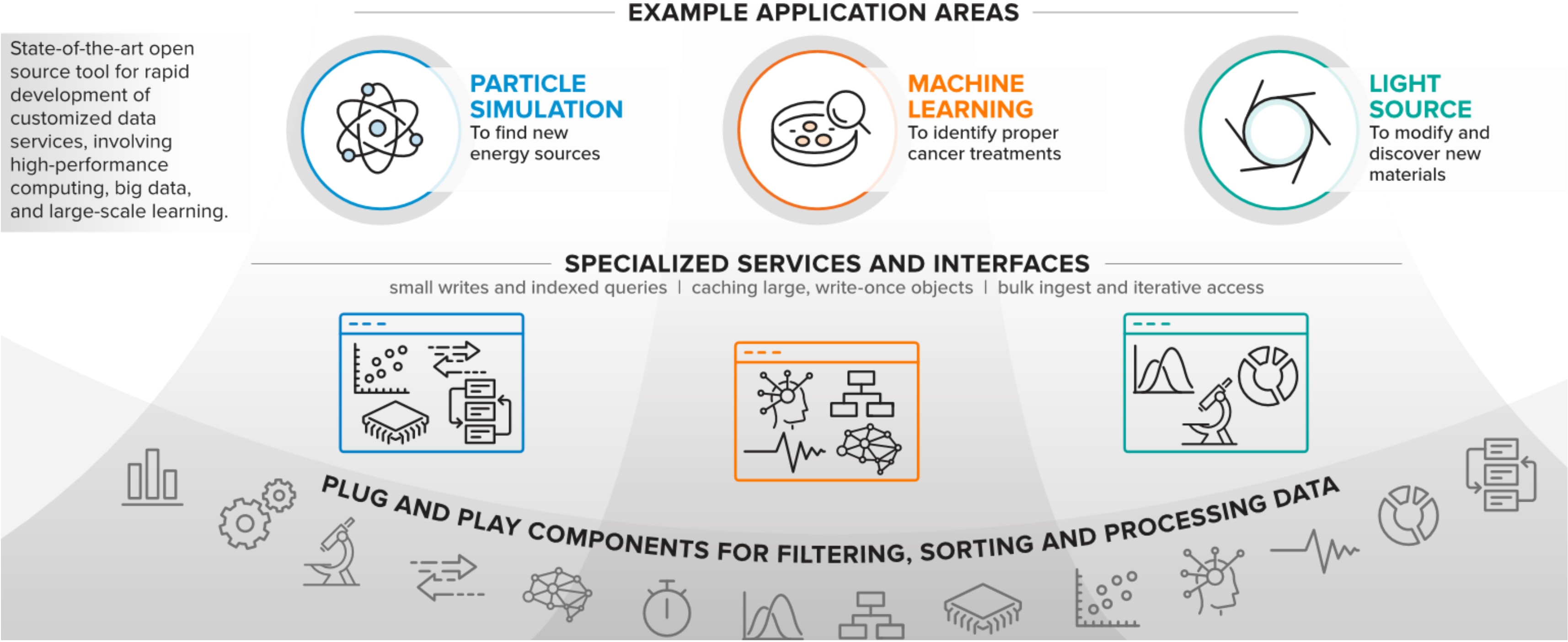
Mochi seeks to transform this data service monoculture into **an ecosystem of specialized services that are tailored to suit specific use cases and problem domains**. It accomplishes this by providing methodologies and tools for the rapid development of distributed HPC data services.

- A collection of reusable, robust, performant microservices and components
- A methodology for composing them into novel, domain-specific services
- API bindings in C, C++, or Python

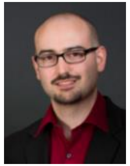
Mochi services are intended to augment, not replace, mission-critical parallel file systems in the HPC runtime ecosystem.

MOCHI

A framework and methodology for customizable services



THE TEAM


Phil Carns, Matthieu Dorier, Rob Latham,
Shane Snyder, and Rob Ross (PI)
Argonne National Laboratory




Tyler Reddy, Kyle Roarty, Galen Shipman,
and Qing Zheng
Los Alamos National Laboratory



George Amvrosiadis, Chuck Cranor, and Ankush Jain
Carnegie Mellon University

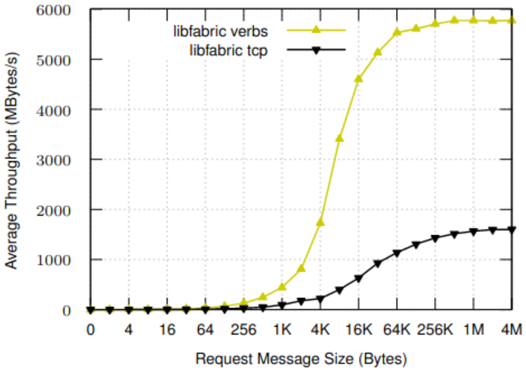
* also long-time contributions from Jerome Soumagne of Intel, formerly The HDF Group

MOCHI'S TECHNICAL ROOTS

Mochi launched in 2015, but two key underpinnings predate it

Mercury

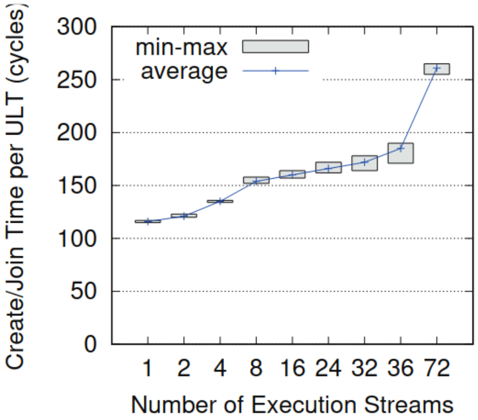
- HPC-oriented RPC framework
- Developed by ANL and THG
- Enables efficient access to native network transports for remote execution



Jerome Soumagne Et al., "Advancing RPC for Data Services at Exascale", 2020

Argobots

- User-level threading framework
- Developed by ANL & collaborators
- Enables efficient management of concurrent, asynchronous execution paths



Argobots

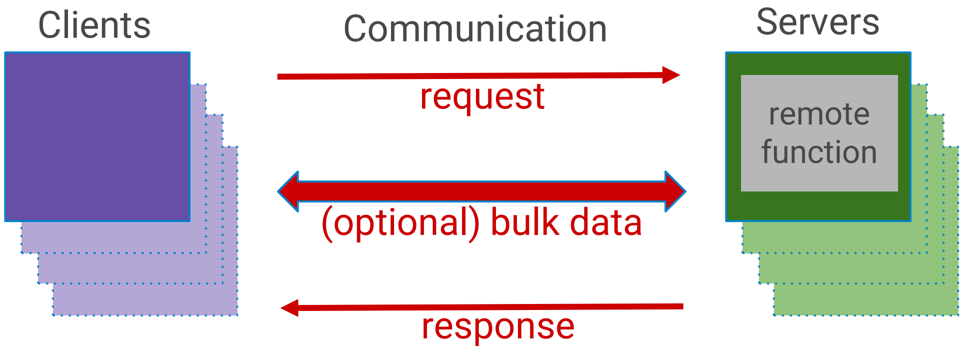
Sangmin Seo Et al., "Argobots: A lightweight low-level threading and tasking framework", 2018

EXAMPLES OF CURRENT MOCHI COMPONENTS

Conveniently, the Spack project (LLNL) emerged around the same time as Mochi and has proven crucial for dependency management.

	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.
	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.
	Sonata	Sonata is a Mochi service for JSON document storage based on UnQLite.
	Yokan	Yokan enables RPC-based access to multiple key-value backends.

AN RPC MODEL FOR MICROSERVICES



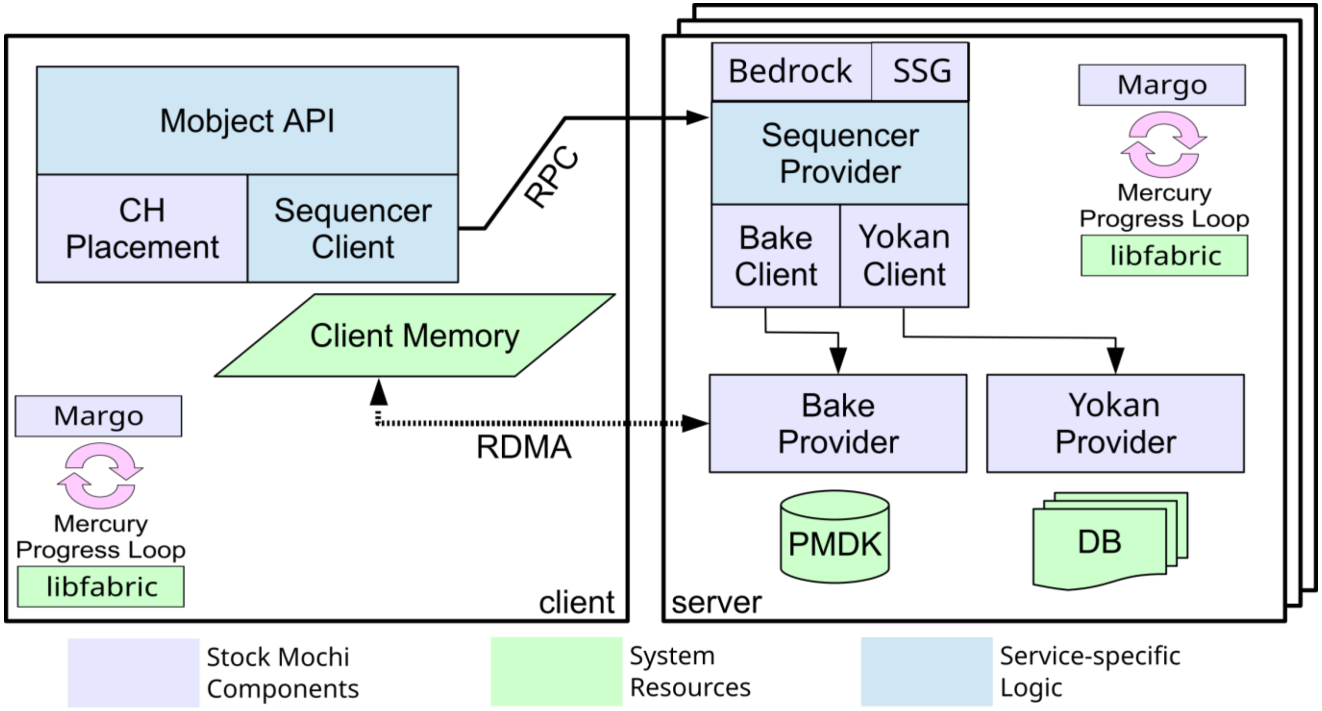
- RPC = “remote procedure call”
- Clients ask servers to execute remote functions on their behalf.
- Function inputs and outputs are encoded as needed.

RPC systems have been around in various forms for decades.
What’s unique about how Mochi does things?

- Designed for very high concurrency
- Explicit data transfers (e.g., a fast path for bulk I/O operations)
- Support for HPC hardware and protocols
- User-space execution without escalated privileges
- *Composability*: combining and layering RPC-based components into a coherent whole

AN EXAMPLE COMPOSITION

The Mobject distributed object store



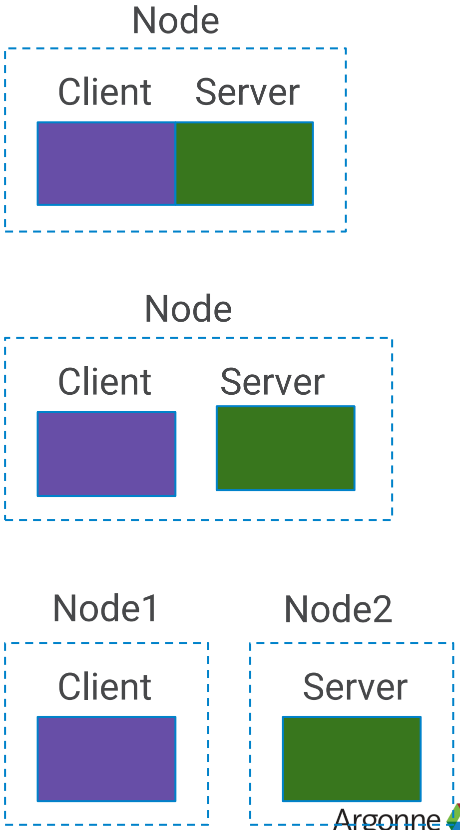
KEY TECHNICAL HURDLES AND SOLUTIONS

FLEXIBLE COMPONENT PLACEMENT

Varying use case and deployment scenarios call for flexibility in component and microservice placement.

Possibly options include:

- Services on set-aside (remote) nodes
- Services co-located (local) with the application
- Services embedded (same address space) within the application
- *Or more likely: some combination of the three*



FLEXIBLE COMPONENT PLACEMENT

How do you avoid API complexity and costly code changes when adapting services to different scenarios?

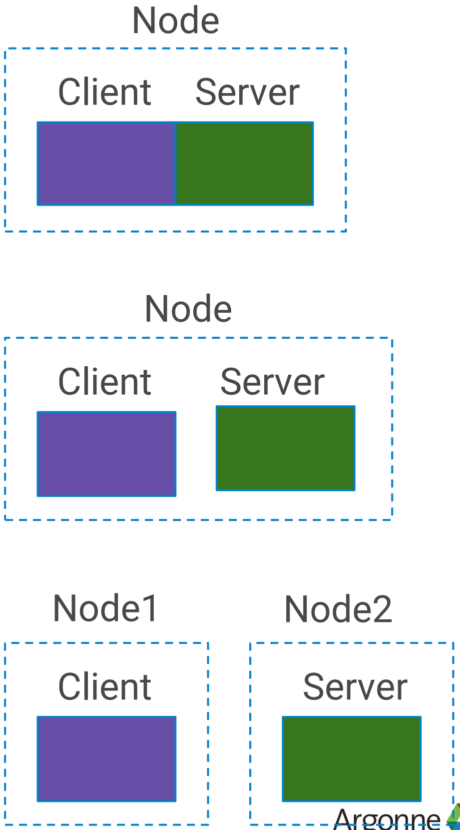
The Mochi approach: “***Every service API call is an RPC***”.

- API conventions and addressing do not change for different deployments or compositions.
- Mochi handles transparent transport selection.

RPC transport:
Function call &
memory copy

RPC transport:
Local pipe &
fast shared
memory

RPC transport:
HPC network
fabric & RDMA

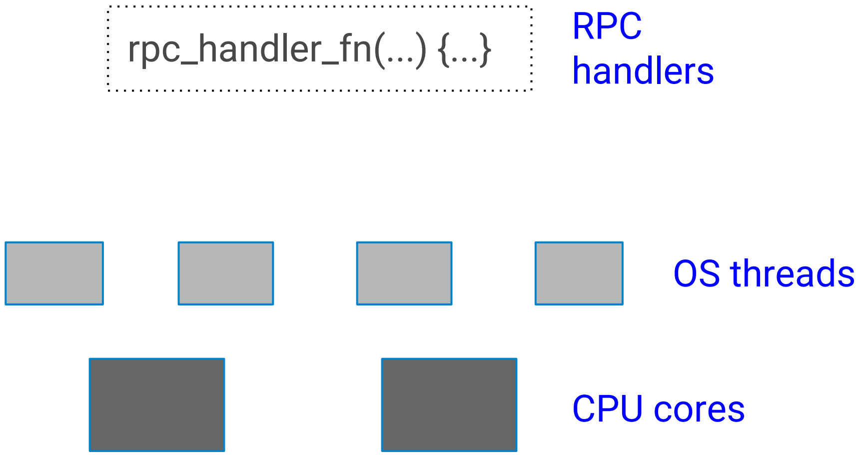


PROVISIONING AND MAPPING TO EXECUTION RESOURCES

Varying use case and deployment scenarios also call for flexibility in mapping remote function execution to hardware execution units.

Possible concerns:

- Where should a server execute a given RPC handler?
- Should you use more execution units?
- Should you use less execution units to reserve capacity for other on-node tasks?
- Should you throttle execution according to HW capabilities?

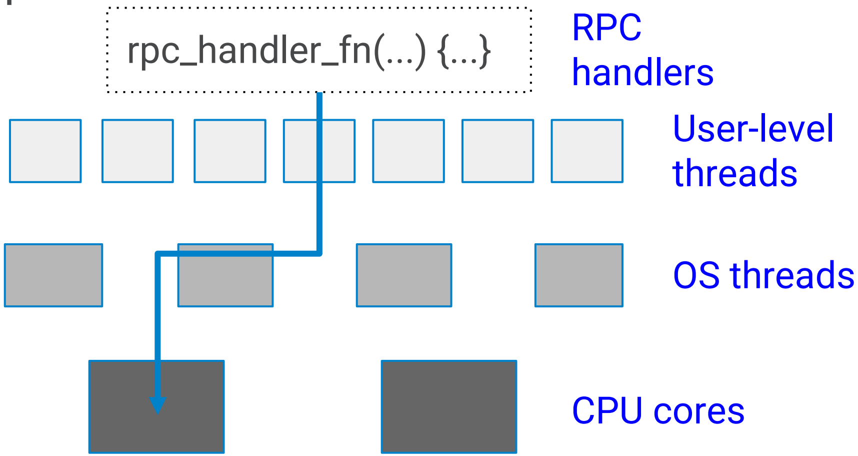


PROVISIONING AND MAPPING TO EXECUTION RESOURCES

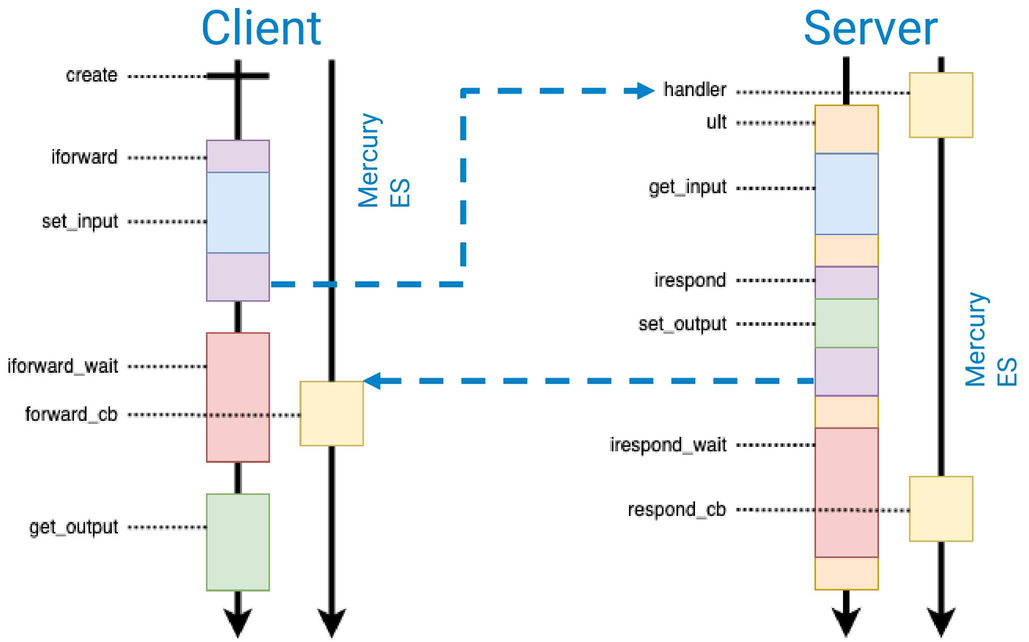
How do you avoid rewriting service logic for different RPC execution scenarios?

The Mochi approach: “***Every RPC handler is a user-level thread***”.

- RPC handler functions are defined when you register a new RPC type.
- Mochi will automatically execute these handlers on user-level threads, which map to operating system threads, which map to CPU cores.
- In Mochi, the resource mapping challenge is a configuration problem, not a software architecture problem.



PERFORMANCE INTROSPECTION

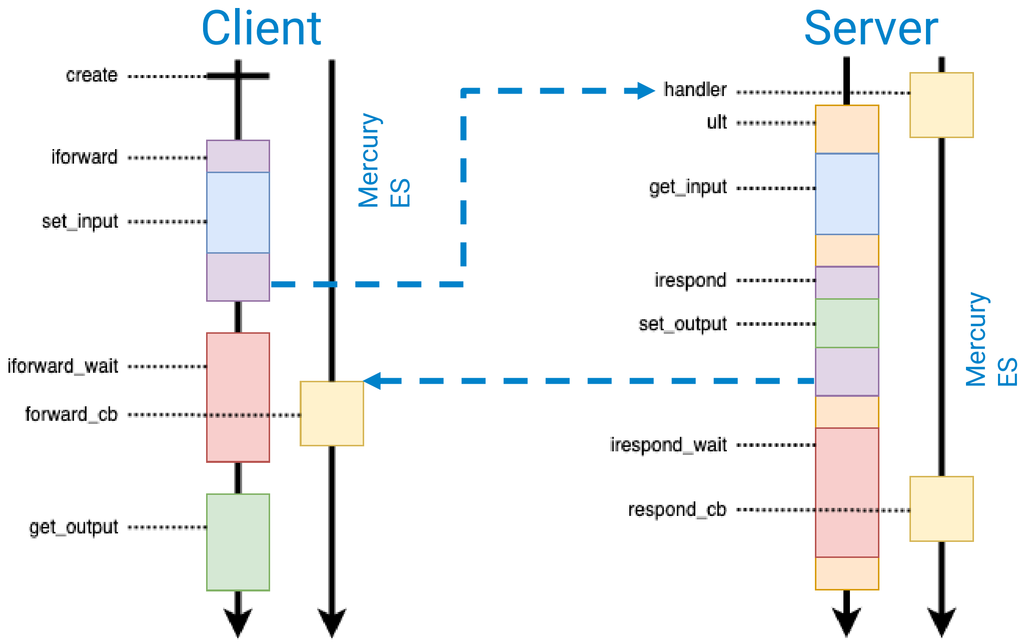


RPC invocation steps can be quite complex under the covers (i.e., encoding, decoding, asynchronous communication, context switches).

How should you approach performance profiling?

Fortunately, the *"Every service API call is an RPC"* and *"Every RPC handler is a user-level thread"* principles present unique opportunities.

PERFORMANCE INTROSPECTION



Message header injection and thread-local storage enable implicit, transparent, uniform propagation of:

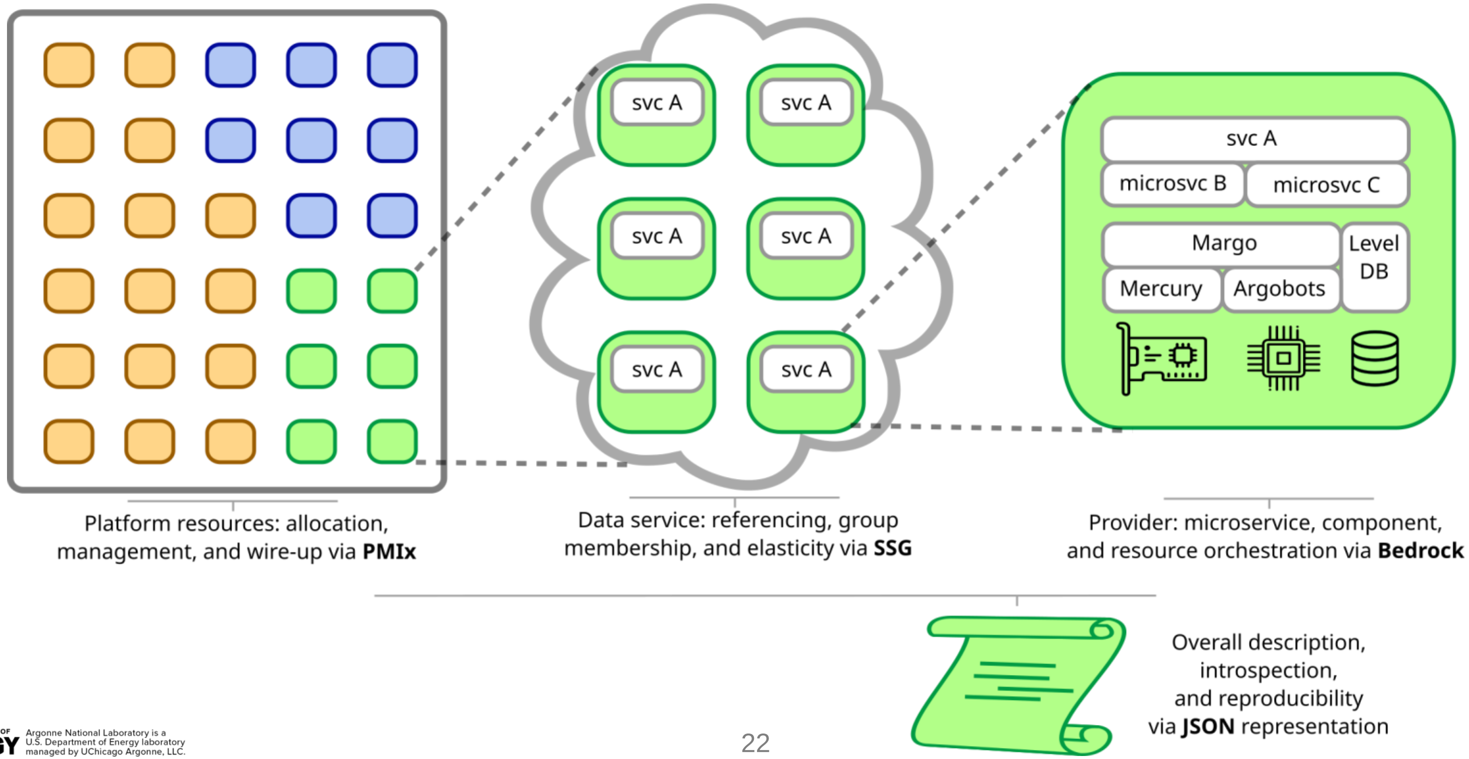
- Timing information
- Context identifiers
- Lineage of composed RPCs

Performance data can be emitted in JSON format at runtime with no code changes.

The data can then be visualized in Python and Jupyter Notebook.

RUNTIME CONFIGURATION

How to wire-up, group, and configure services



WHAT IS MOCHI USED FOR NOW, AND WHERE DO WE GO FROM HERE?

EXAMPLE SERVICES BUILT WITH MOCHI

Category	Service	Institution	Summary
Specialized file systems			
	DeltaFS	CMU	Transient file system service fwith highly paralleled indexing of file data and metadata
	Unify	LLNL & ORNL	Suite of specialized, flexible file systems that can be included in a user's job
	GekkoFS	JGU Mainz & BSC	Temporary distributed file system for HPC applications
	CHFS	U. Tsukuba	Ad hoc file system for persistent memory based on consistent hashing
	DelveFS	JGU Mainz	Semantic file system for object stores
Domain-specific data mgmt			
	HEPnOS	ANL & FNL	Transient, in-memory, distributed storage system for high energy physics (HEP) workflows
	FlameStore	ANL	Storage for deep learning models

EXAMPLE SERVICES BUILT WITH MOCHI

Category	Service	Institution	Summary
Alternative data models			
	DAOS	Intel	HPC-oriented platform object store with support for persistent memory
	HXHIM	LANL	Hexadimensional hashing indexing middleware
	Proactive Data Containers	LBNL	Object-centric data management system to take advantage of deep memory and storage hierarchy
	Mobject	ANL	In-system distributed object storage conforming to the RADOS API
Data access methods			
	DataSpaces	U. Utah	Programming system and data management framework for coupled workflows
	Hermes	IIT, THG, & UIUC	Hierarchical tiered storage and buffering management
	Benvolio	ANL	I/O forwarding and transformation service

EXAMPLE SERVICES BUILT WITH MOCHI

Category	Service	Institution	Summary
Performance diagnostics			
	Chimbuko	BNL	Workflow-level scalable performance trace analysis tool
	Symbiomon	U. Oregon	Integrated application/service performance monitoring
In situ analytics			
	Seer	LANL	Lightweight in situ wrapper library adding in situ capabilities to simulations
		Kitware	Platform for ubiquitous access to visualization results during runtime
	Serviz	U. Oregon	Shared in situ visualization service
	Colza	ANL	Elastic in situ visualization

CURRENT RESEARCH DIRECTIONS

- Elasticity:
  – Dynamically reconfiguring and resizing services in response to application or workflow needs
  – Two dimensions: Vertical (on-node resources) and horizontal (across nodes)
  – Requires several new core technologies: reconfigurability, instrumentation, and resource migration
- Support for smart devices:
  – Making use of programmable storage devices, network cards, and network switches in the data path
  – Exploring what algorithms are amenable to this technology
  – Developing methods to portably incorporate smart devices when present

YOUR HPC SERVICE HERE

We are always looking for new use cases and collaborators!

Feel free to reach out to any members of the team if you have any questions about Mochi or ideas for potential use cases.

Also, be sure to attend next week's follow-on seminar entitled **"Mochi in Practice: Data Services for High-Energy Physics and Elastic In Situ Visualization Workflows"** by Matthieu Dorier.

THANK YOU!

THIS WORK WAS SUPPORTED BY THE U.S. DEPARTMENT OF ENERGY, OFFICE OF SCIENCE, ADVANCED SCIENTIFIC COMPUTING RESEARCH, UNDER CONTRACT DE-AC02-06CH11357.