Using Mochi to Build Data Services

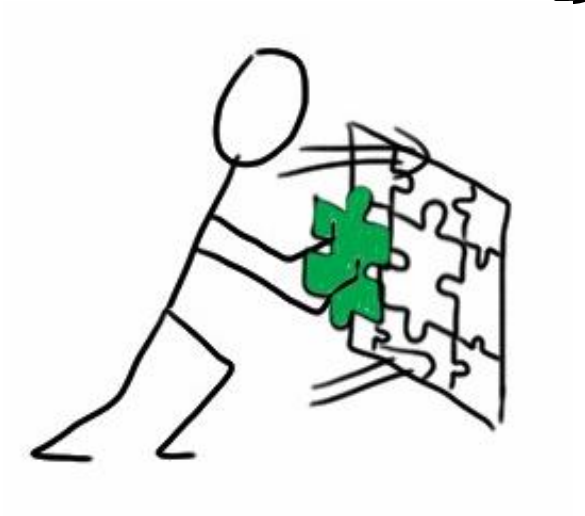
Philip Carns (Argonne National Laboratory)
Matthieu Dorier (Argonne National Laboratory)
Rob Ross (Argonne National Laboratory)
Jerome Soumagne (The HDF Group)

April 13, 2021

Argonne NATIONAL LABORATORY 75 1946–2021
The HDF Group
U.S. DEPARTMENT OF ENERGY | Office of Science

What's changing in HPC data services?

Mochi



Application pull:

- Use of HPC in experimental science (e.g., ATLAS/CMS)
- Artificial intelligence use cases
- Streaming data

Technology push:

- More capable storage technologies
- Compute in storage
- New networking APIs and capabilities

Image from M. Geurden, "Market Opportunity Identification: Push or Pull?,"July 2012, https://newentrepreneurship.nl/2012/07/02/market-opportunity-identification-push-or-pull/

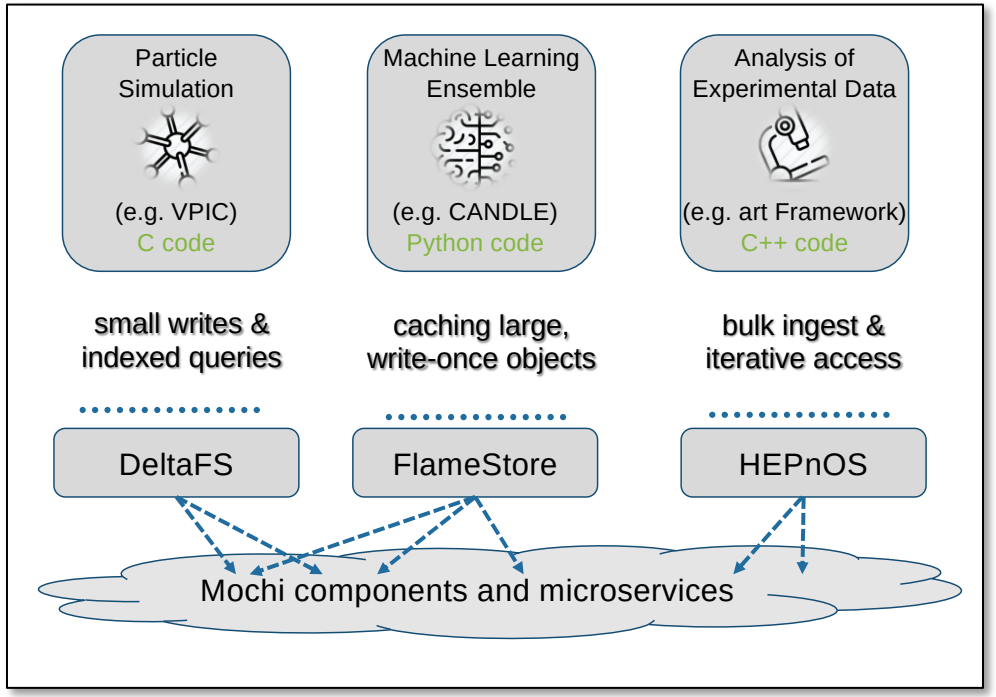
Mochi
Customized data services for DOE science

Mochi provides a toolkit for building high-performance data services for use on HPC platforms, and ECP computer scientists are using Mochi to build services for ECP application teams.

Mochi is a multi-institution project including Argonne National Laboratory, Carnegie Mellon University, the HDF Group, and Los Alamos National Laboratory.

Who uses Mochi?

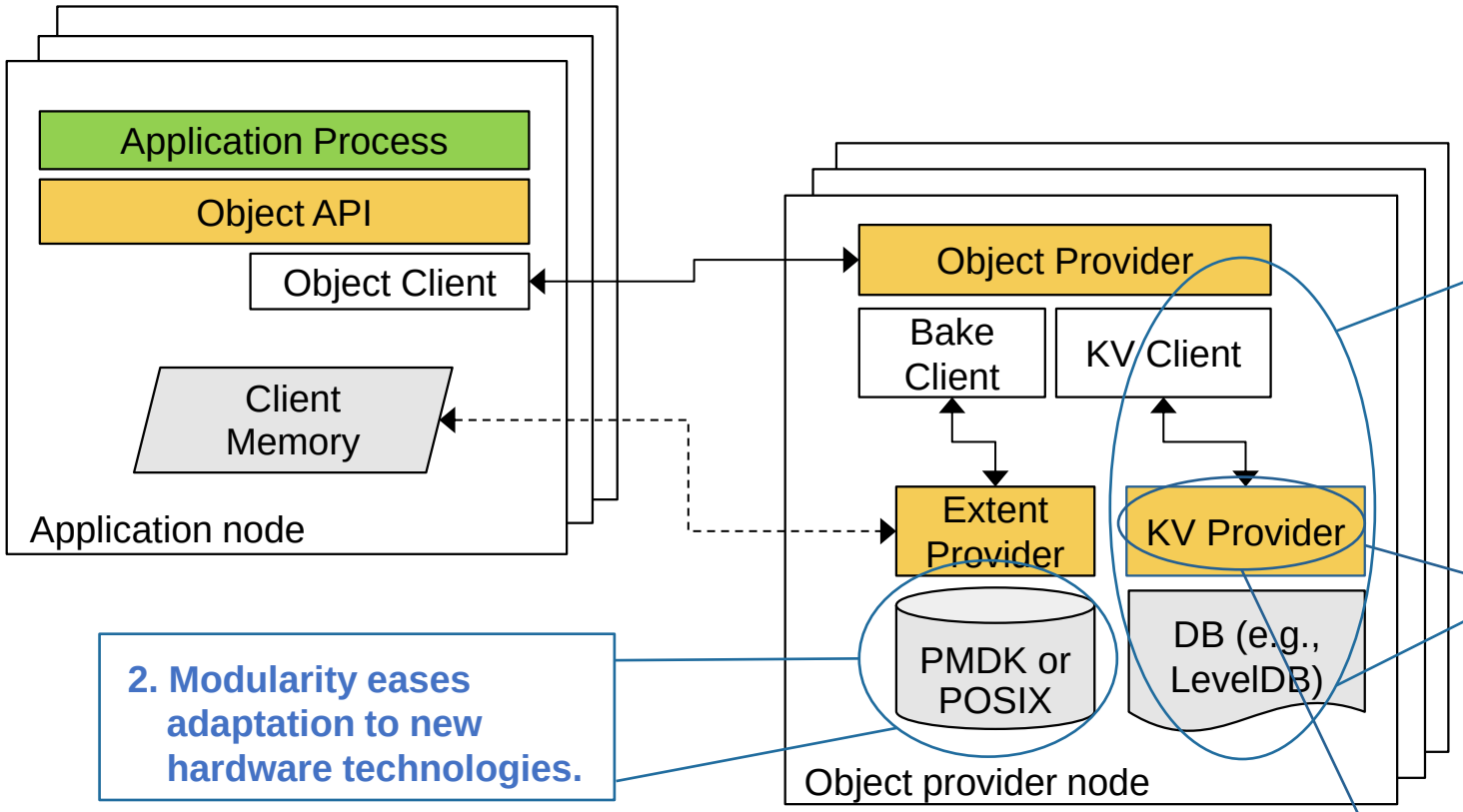
- End users benefit from the specialization of these services in terms of ease of use and performance.
- Computer scientists use Mochi to develop customized data services.



Mochi has been used to develop a number of services, including ones to store and index particle data, to manage learning data, and to provide fast access to high-energy physics detector data during analysis.

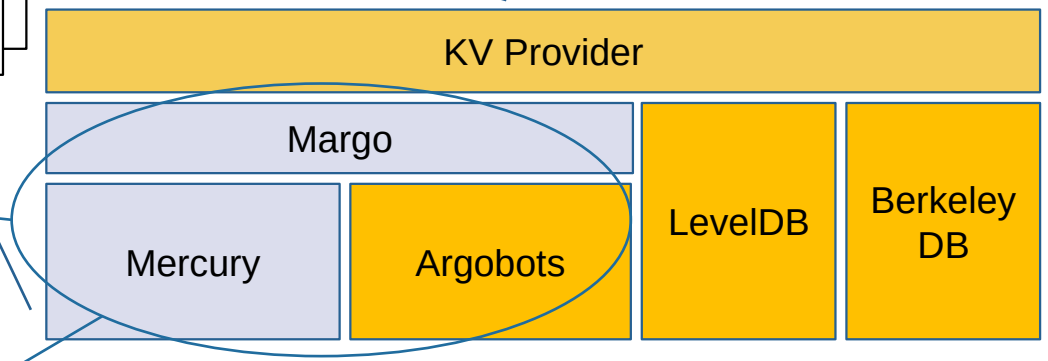
Within ECP, Mochi is also helping enable Unify, Chimbuko, DataSpaces, and Proactive Data Containers.

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.

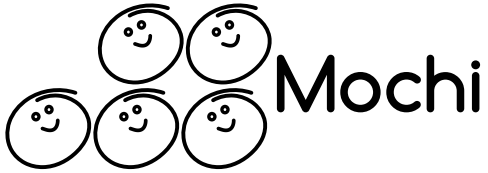


3. Multiple methods of programming (C, C++, Python), more accessible.

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

	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.

Agenda



2:30 – 2:40	Welcome and Introductions	Rob Ross
2:40 – 2:55	Getting Started	Phil Carns
2:55 – 3:10	Composition and Configuration	Matthieu Dorier
3:10 – 3:25	Networking with Mercury	Jerome Soumagne
3:25 – 3:30	Wrap-up	Rob Ross

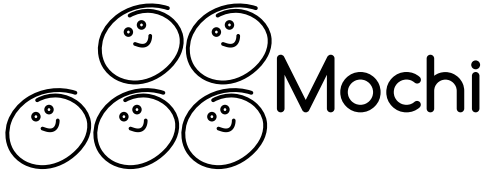
Getting Started with Mochi
&
Recent Updates

Getting Started

Mochi

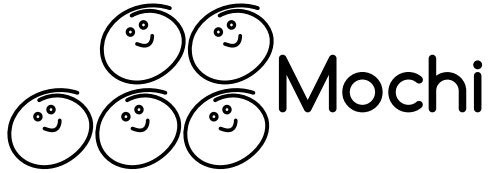
- Start here for documentation:
  - https://mochi.readthedocs.io/en/latest/
- Additional resources, including a mailing list and slack space, can be found on the project web page:
  - https://www.mcs.anl.gov/research/projects/mochi/
- Installation “recipes” are available for several popular ECP platforms
  - https://github.com/mochi-hpc-experiments/platform-configurations (spack environment examples)
  - https://github.com/mochi-hpc-experiments/mochi-tests (performance regression script examples)
- We will be continuing to improve the first-time user experience in upcoming deliverables

Installing Mochi with Spack



- We strongly recommend using Spack to install any Mochi components
  - Straightforward to do per-user installations without administrative privilege
  - Component dependencies are handled automatically
  - One unified yaml file expresses all preferred build settings (e.g., network transport, compiler, storage backend) for a given platform
  - Our team maintains an external package repository that enables rapid integration of new releases
- See https://mochi.readthedocs.io/en/latest/ for details

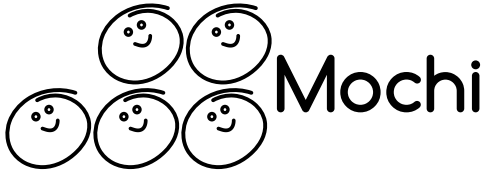
Mochi source code: now on GitHub!



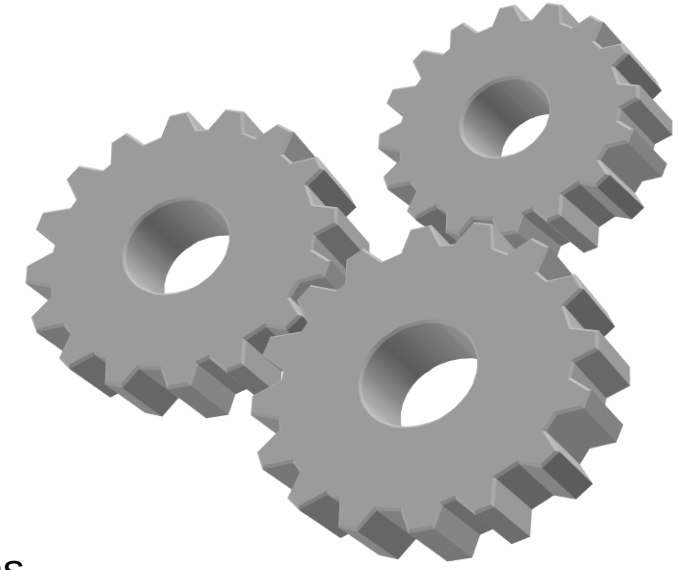
- All Mochi source code has been migrated to github.com as of March 2021
  - https://github.com/mochi-hpc/
  - The Mochi software is actually a collection of components maintained in separate repositories
  - Bug reports and contributions are welcome! Please note the CLA policy for contributions.
- Were you already using Mochi prior to the migration? Update your spack repository to refer to the new location.
  - https://www.mcs.anl.gov/research/projects/mochi/2021/03/24/the-mochi-github-migration-is-complete/

```
spack repo rm mochi
git clone https://github.com/mochi-hpc/mochi-spack-packages.git
spack repo add mochi-spack-packages
```

Performance diagnostics and profiling



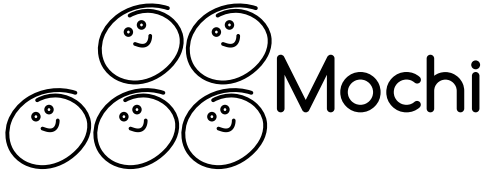
- How do you tune the performance of a Mochi service?
  - Step 1: Use the best (native) network transport for your platform
  - Step 2: Use Mochi diagnostic and profiling tools* to understand where service time is spent
- Basic performance diagnostic and profiling capability built into *any* Mochi service
  - No need to modify or recompile application or service
  - Automatically tracks Mochi RPCs
  - Automatically tracks RPC dependencies
  - Includes intra-node, inter-node, and inter-process calls



* Functionality developed by Srinivasan Ramesh of U. Oregon, see:

SYMBIOSYS: A Methodology for Performance Analysis of Composable HPC Data Services
Srinivasan Ramesh, Allen D. Malony, Philip Carns, Robert B. Ross, Matthieu Dorier, Jerome Soumagne, and Shane Snyder (to appear in IPDPS 2021)

Enable profiling of an existing service

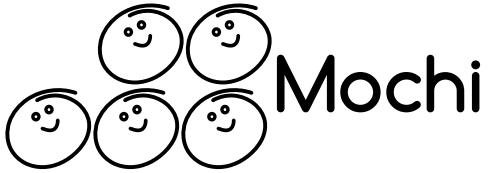


- Set environment variables to enable profiling
- Run your service / application
- Generate profile summary
- Render RPC dependency graph
- Look at the results!

```
> export MARGO_ENABLE_PROFILING=1
> export MARGO_ENABLE_DIAGNOSTICS=1

# run example

> margo-gen-profile
********************MARGO Profile Generator*******************
Reading CSV files from: /home/carns/
Done.
************************************************************

> dot -Tpdf graph.gv -o graph.pdf

> ls *.pdf
graph.pdf  profile.pdf
```

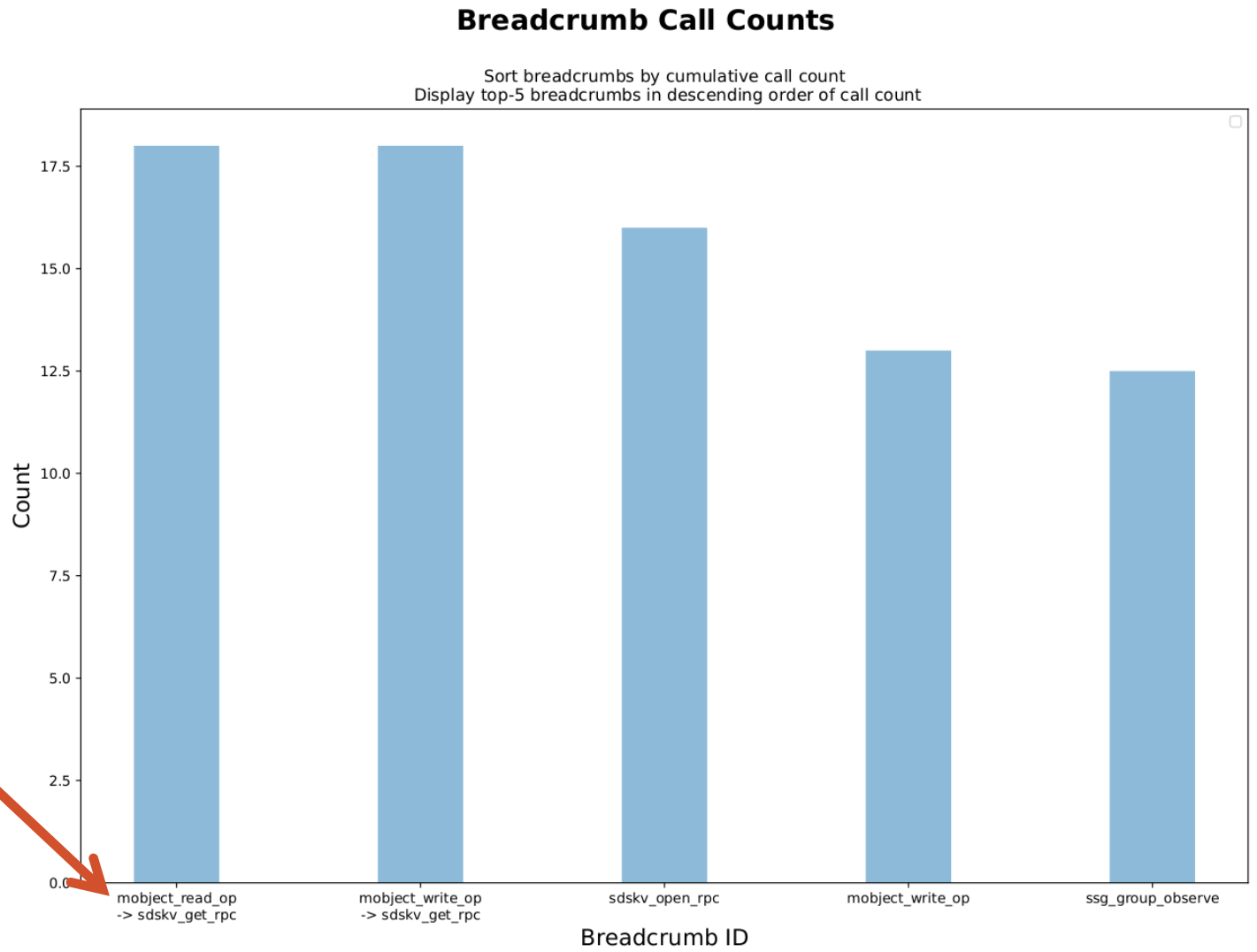
See README.md in mochi-margo for more information

How many times was each RPC call path executed?

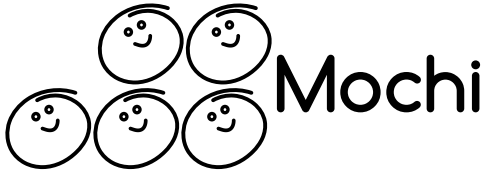


This graph shows the top 5 most frequently executed RPCS.

Note chains showing provenance of RPC invocations.

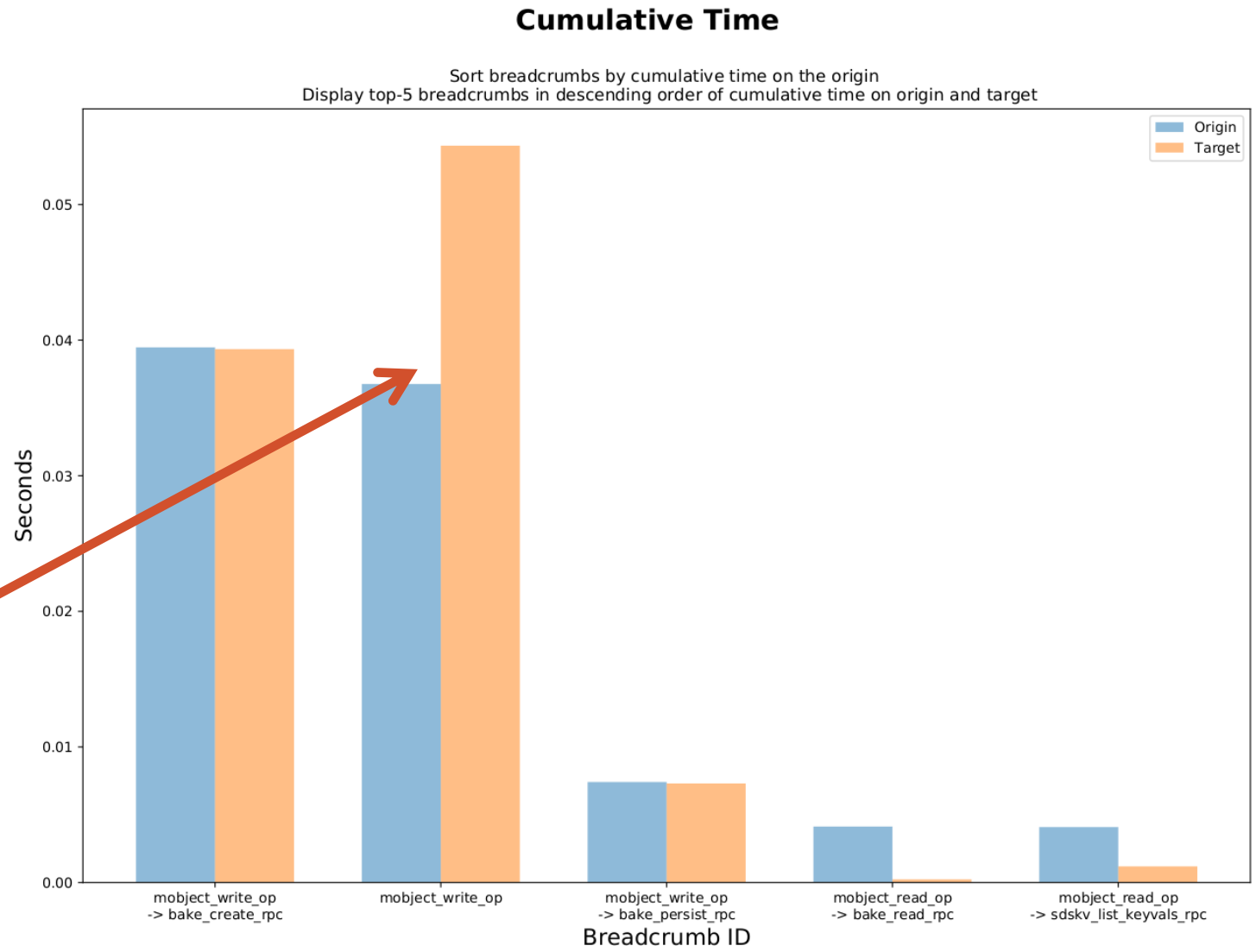


How much cumulative time was spent in each RPC?



This graph shows the top 5 RPCs in terms of cumulative time.

Unusual example: target (server) side of this RPC consumed more time than clients, indicating presence of completion delay after sending ack.

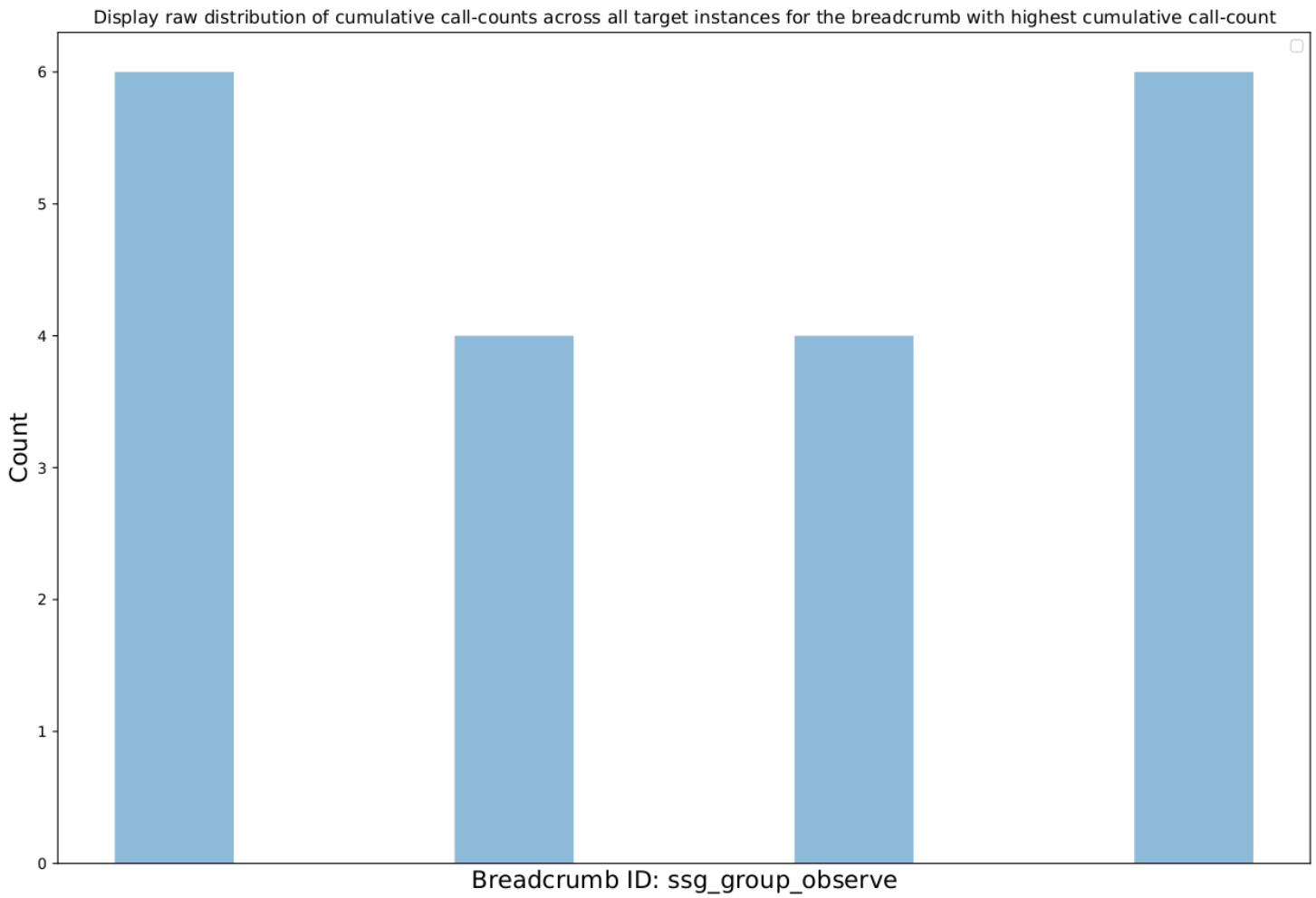


How were the RPCs distributed across servers?

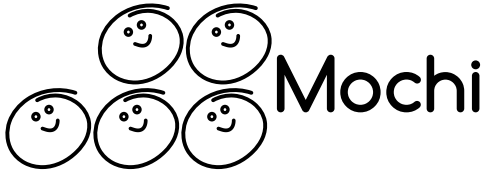
Mochi

For the most frequently executed RPC, how well was it distributed across available targets (servers)?

Skewed results here could indicate a hotspot or load imbalance.



Future work

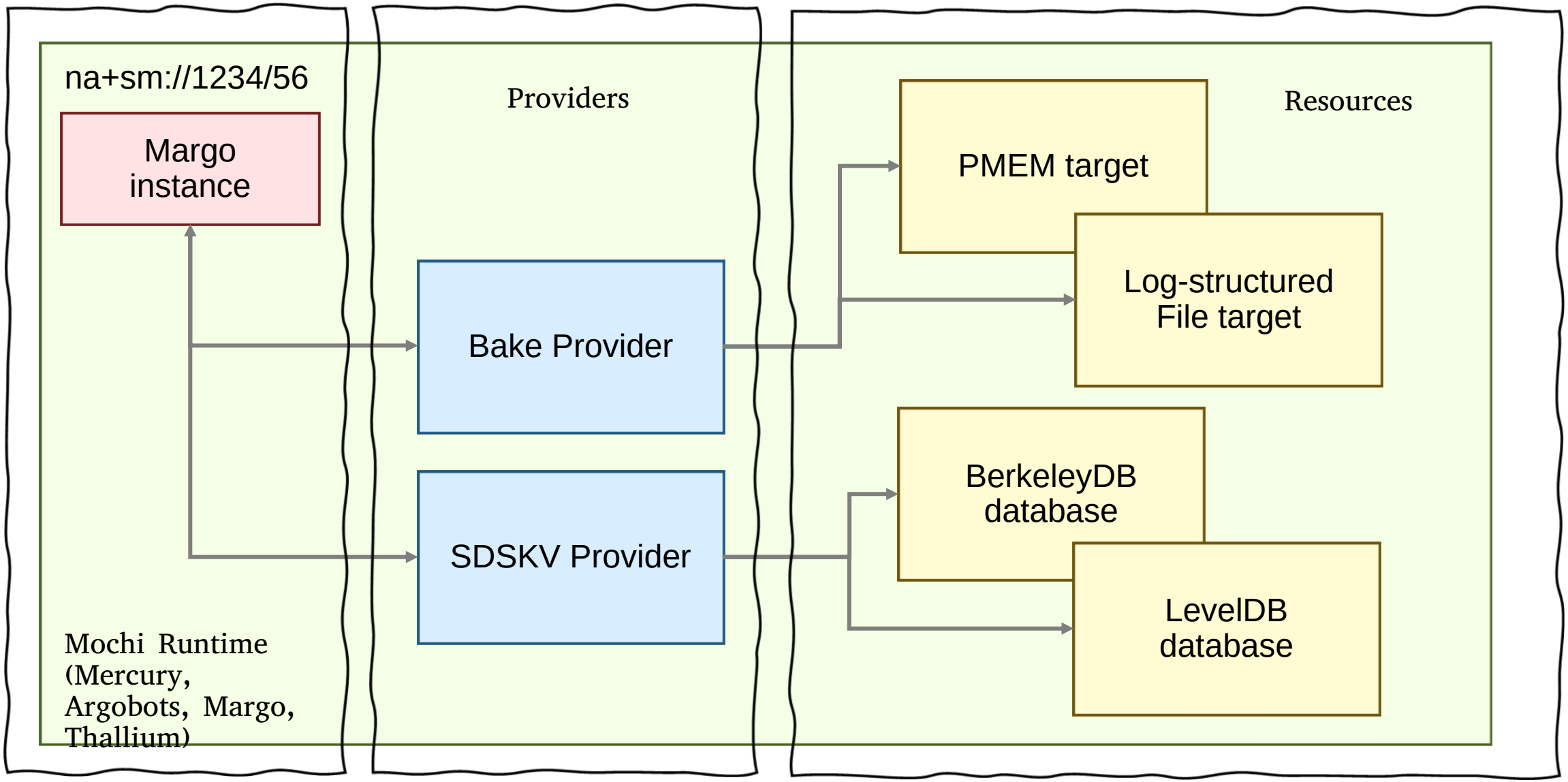


How do we plan to improve the “getting started” and “performance profiling” experiences with Mochi?

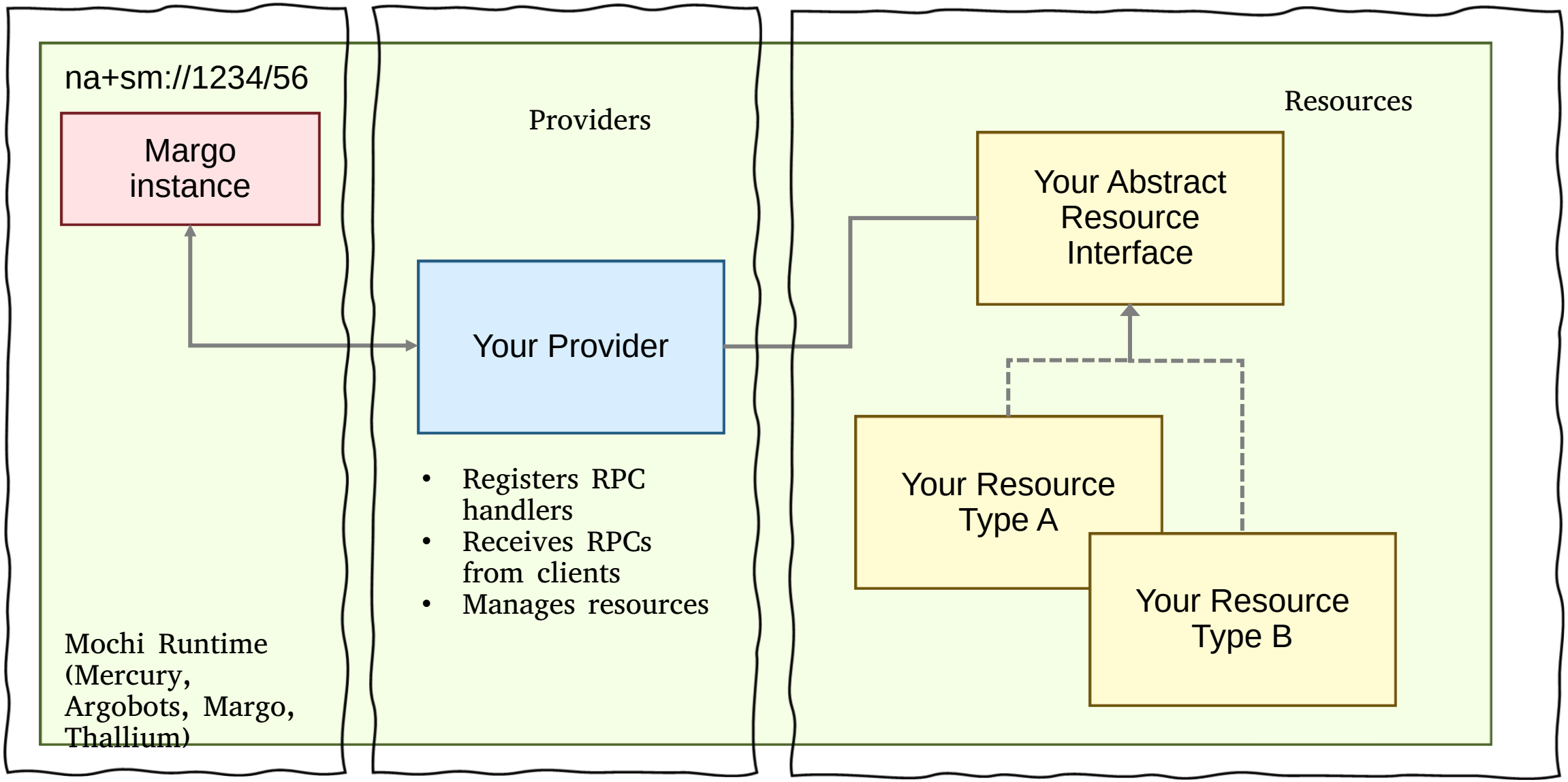
1. Create a structured “Hello Mochi” mechanism to get started with Mochi for the first time and confirm that it is working correctly on your system.
   - Automated as much as possible
   - Normalize support information for new users
2. Improve performance tuning capability
   - Auto-tuning and recipes where appropriate
   - More integrated capabilities to report status, statistics, configuration, and profiling

Composition and Configuration of Mochi Services

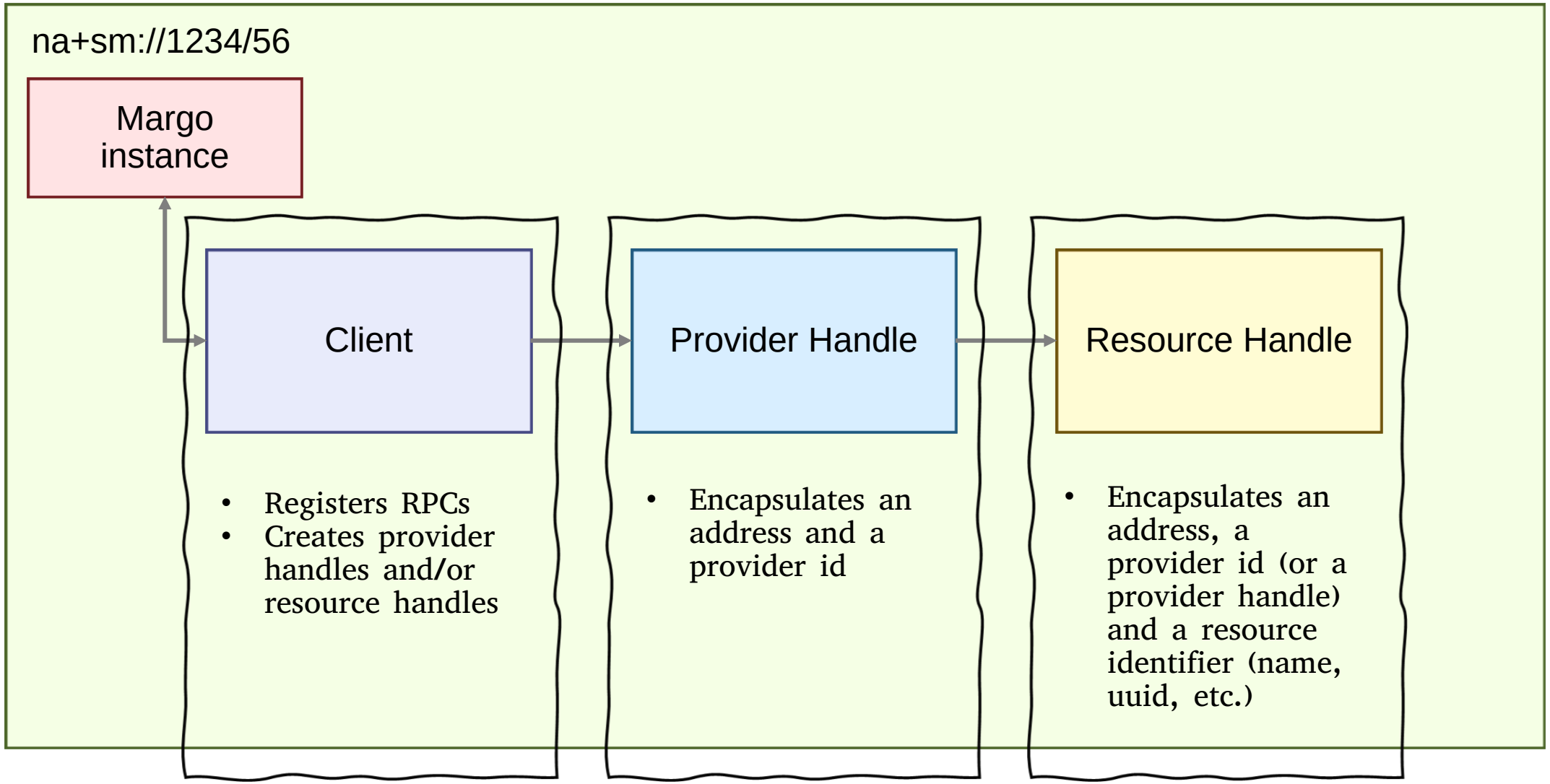
Architecture of a composed Mochi service



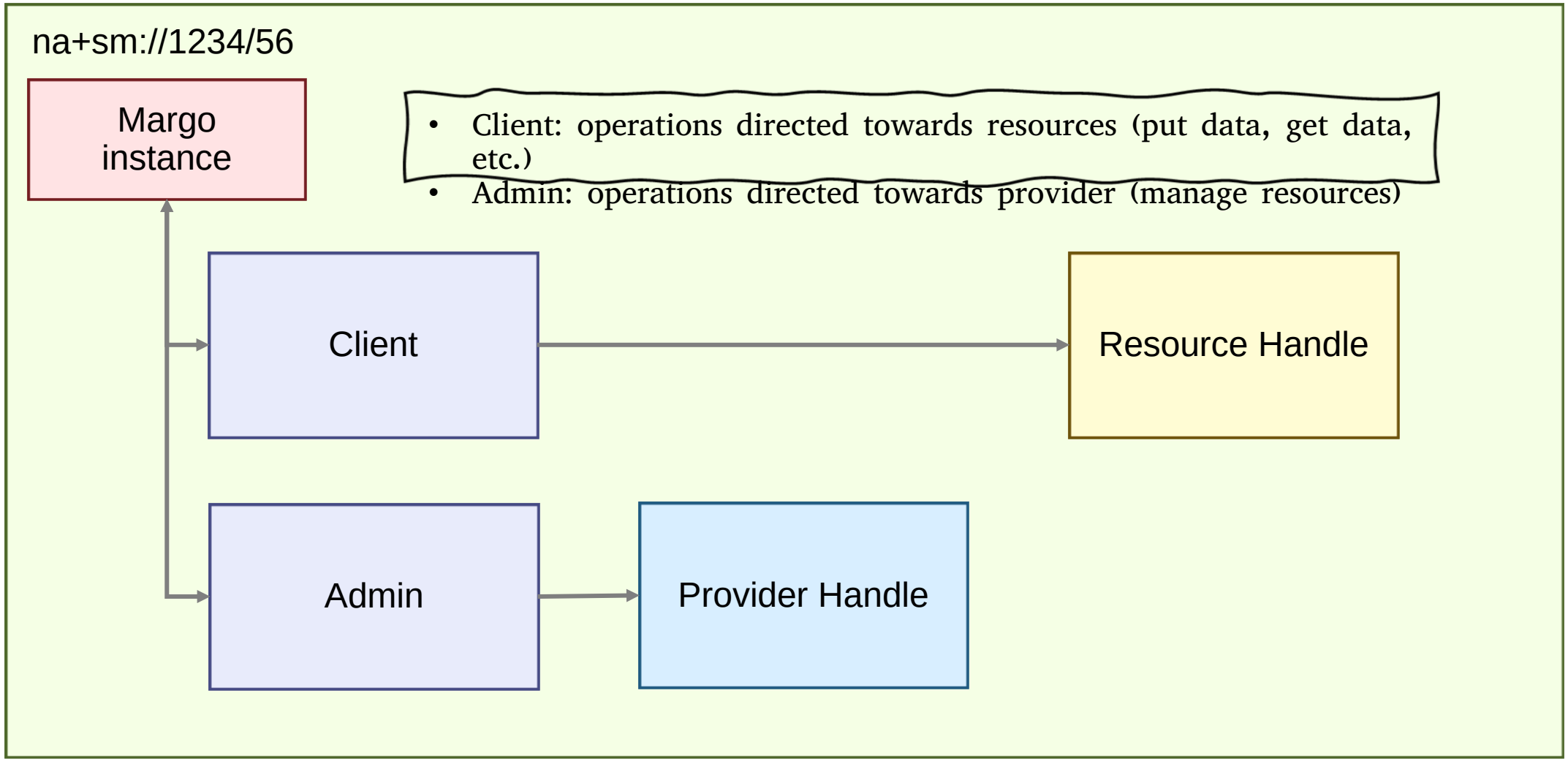
Architecture of a composed Mochi service



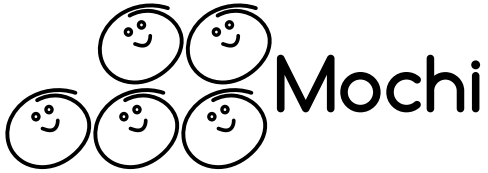
Architecture of a composed Mochi service



Architecture of a composed Mochi service



Mochi microservice templates



- Margo microservices (C)
  - https://github.com/mochi-hpc/margo-microservice-template
  - json-c for configuration
  - µnit for unit-testing

- Thallium microservices (C++)
  - https://github.com/mochi-hpc/thallium-microservice-template
  - nlohman_json for configuration
  - CppUnit for unit-testing

- Clone, run python setup.py to rename files and classes / functions / structures
- More information: https://mochi.readthedocs.io/en/latest/templates.html

Going away from hand-written daemons with Bedrock

- Key ideas
  - Describe components (providers, abt-io, ssg, etc.) to deploy on a node in a JSON file
  - Deploy a generic daemon that reads the JSON file
  - Query the deployment configuration at any time via RPC
  - Deploy new components dynamically at any time via RPC

- Advantages
  - No need for a custom composition in C/C++/Python
  - More reproducible configurations
  - Easier to share configurations for troubleshooting

- Full tutorial: https://mochi.readthedocs.io/en/latest/bedrock.html

JSON examples

Mercury config
(see documentation)

Argobots pools

Argobots xstreams

Default pools to use for progress and for running RPC handlers

```
{ "margo" : {
        "mercury" : { },
        "argobots" : {
                "abt_mem_max_num_stacks" : 8,
                "abt_thread_stacksize" : 2097152,
                "version" : "1.0.0",
                "pools" : [
                        { "name" : "my_progress_pool", "kind" : "fifo_wait", "access" : "mpmc" },
                        { "name" : "my_rpc_pool", "kind" : "fifo_wait", "access" : "mpmc" } ],
                "xstreams" : [
                        { "name" : "my_progress_xstream", "cpubind" : 0, "affinity" : [ 0, 1 ],
                          "scheduler" : { "type" : "basic_wait", "pools" : [ "my_progress_pool" ] } },
                        { "name" : "my_rpc_xstream", "cpubind" : 2, "affinity" : [ 2, 3, 4, 5 ],
                          "scheduler" : { "type" : "basic_wait", "pools" : [ "my_rpc_pool" ] } }
                ]
        },
        "progress_pool" : "my_progress_pool",
        "rpc_pool" : "my_rpc_pool”
}, …
```

JSON examples

```
…
  "bedrock": { "pool": "my_rpc_pool", "provider_id": 0 },
  "abt_io" : [
            { "name" : "my_abt_io", "pool" : "__primary__" }
  ],
  "ssg" : [
            { "name" : "mygroup", "bootstrap" : "init", "group_file" : "mygroup.ssg" }
  ],
…
```

SSG groups and ABT-IO instances

JSON examples

```
…
 "libraries" : {
        "module_a" : "examples/libexample-module-a.so",
        "module_b" : "examples/libexample-module-b.so"
 },
 "clients" : [
        { "name" : "ClientA", "type" : "module_a", "config" : {}, "dependencies" : {} }
 ],
 "providers": [
        { "name" : "ProviderA", "type" : "module_a", "provider_id" : 42,
          "pool" : "__primary__", "config" : {}, "dependencies" : {} },
        { "name" : "ProviderB", "type" : "module_b", "provider_id" : 33,
          "pool" : "__primary__", "config" : {},
          "dependencies" : {
                "ssg_group" : "mygroup",
                "a_provider" : "ProviderA",
                "a_local" : [ "ProviderA@local" ],
                "a_client" : "module_a:client" }
        } ] }
```

Libraries to dlopen, with definitions of microservices

Some microservice clients

Some microservice providers

Dependencies can by any named entity (client, provider, abt-io instance, SSG group) as well as addresses to providers on other processes

Bedrock module library (C)

```
static struct bedrock_module ModuleA = {
        .register_provider = ModuleA_register_provider,
        .deregister_provider = ModuleA_deregister_provider,
        .get_provider_config = ModuleA_get_provider_config,
        .init_client = ModuleA_init_client,
        .finalize_client = ModuleA_finalize_client,
        .get_client_config = ModuleA_get_client_config,
        .create_provider_handle = ModuleA_create_provider_handle,
        .destroy_provider_handle = ModuleA_destroy_provider_handle,
        .provider_dependencies = ModuleA_provider_dependencies,
        .client_dependencies = ModuleA_client_dependencies
};
BEDROCK_REGISTER_MODULE(module_a, ModuleA)
```

Fill out this data structure and compile into a dynamic library for Bedrock to load!

A C++ equivalent exists if you prefer (see documentation)

Networking with Mercury

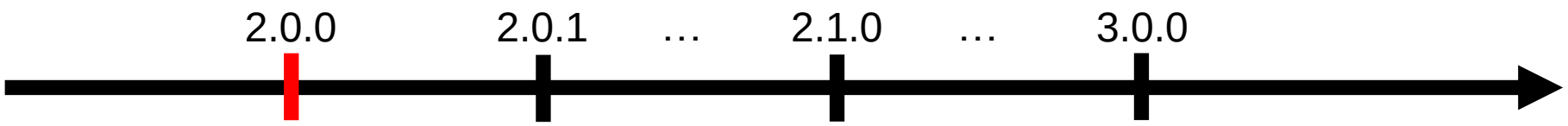
Mercury

Mochi

- Base low-level RPC component used for communication between Mochi services
  - Always consider higher-level components first before directly using the HG API
- No explicit concurrency / multi-threading done at that level
  - However, Mercury provides thread-safety
- Two main data transfer methods
  - Point-to-point RPC through eager messages
    - Connection-less semantics
  - Bulk data through RDMA
    - No memory copy
    - Potential buffer allocation / memory registration overheads (avoid doing these in hot code paths)
  - (Support for collectives is considered)

Mercury – Status and Roadmap

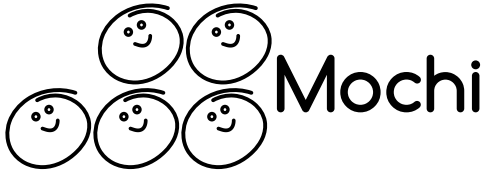
- 2.0.0 version was released in November
  - Support for immediate lookups through HG_Addr_lookup2()
  - Improved support of libfabric and support of new tcp provider
  - Improved shared-memory plugin with full connection-less endpoints support
  - Improved bulk interface with more efficient handling of I/O with small segment count
  - Improved efficiency of mercury proc routines
  - Improved polling mechanism
  - Improved cancellation of operations and error handling
  - Improved error / warning and debug logging



Mercury – Status and Roadmap

- 2.0.1 version released or about to be released
  - Mostly bug fixes
  - Improved error / warning and debug logging with log subsystems
    - HG_LOG_LEVEL=debug/warning/error
    - HG_LOG_SUBSYS=hg/na/mem/op/msg/rma
- 2.1.0 version (summer / fall timeframe)
  - Add support for UCX (tcp and verbs tested)
- 3.0.0 version
  - Extend addressing capabilities to address contexts (enhanced multithreading support and composability)

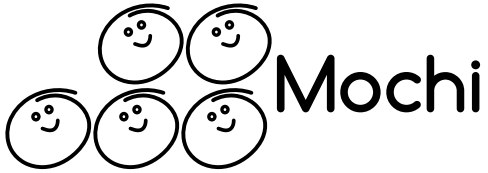
Mercury – Supported Transports



	tcp	verbs	shm	psm	psm2	gni
OFI	✓	✓	✗**	✗**	✓	✓
SM	✗	✗	✓	✗	✗	✗
UCX*	✓	✓	✗**	✗	✗	✗**
PSM*	✗	✗	✗	✓	✓	✗
BMI	✓	✗	✗	✗	✗	✗

* Not yet available in mainstream branch
** Not explicitly supported by mercury but may be supported by underlying library

Mercury – Known Issues and Tuning Knobs



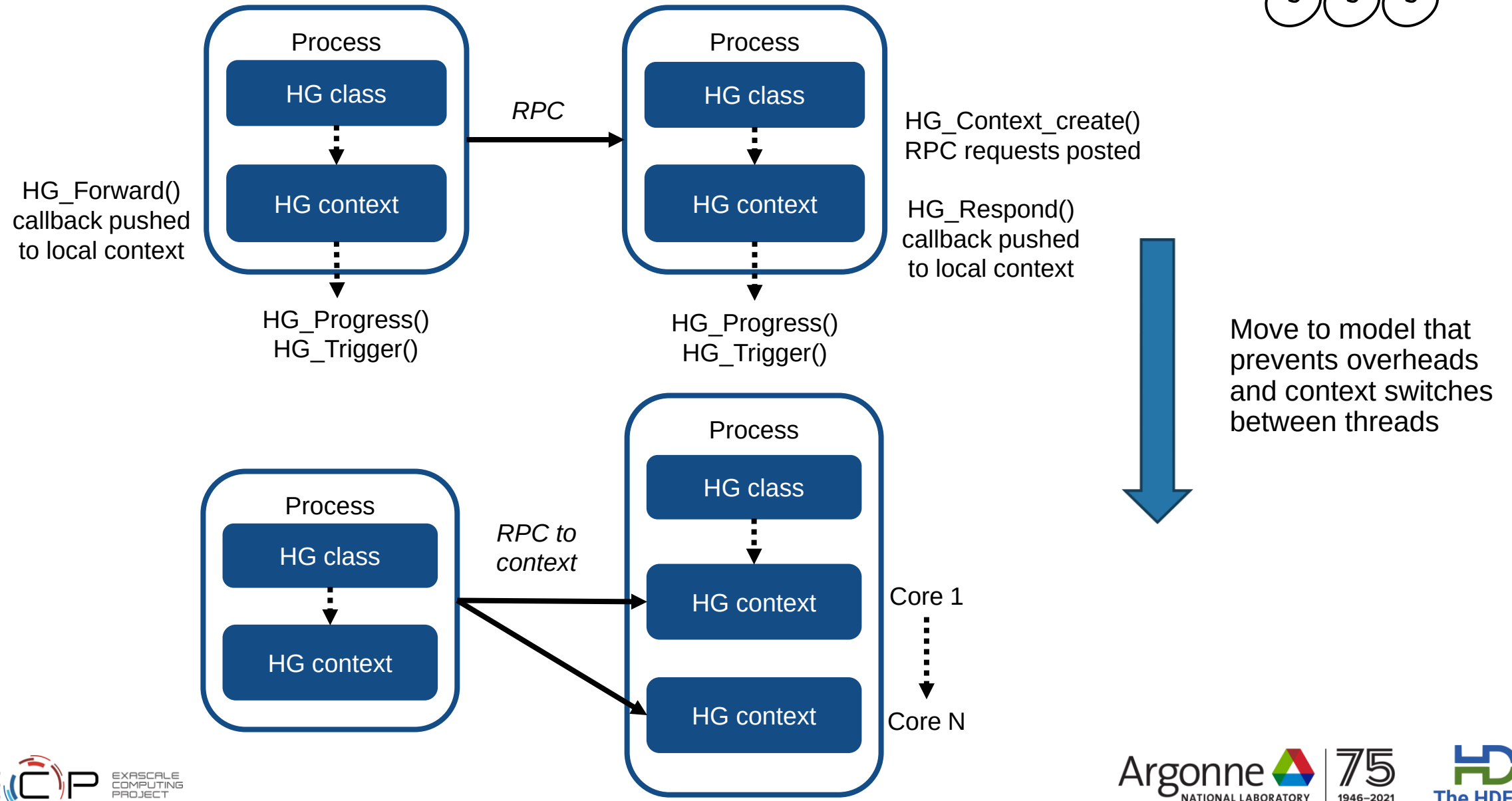
- Specific and recurring libfabric limitations
  - Progress thread (extra thread launched by OFI)
    - auto progress ←→ manual progress requires busy spinning
  - RxM (tcp and verbs): connection management and scalability issues
    - FI_UNIVERSE_SIZE must be set to max number of peers
- Initialization options can be passed to 'HG_Init_opt()'
  - `request_post_init`
  - `request_post_incr`

    Control number of requests posted by server to receive / process RPCs (addtl incoming RPCs are queued by transport layer)
  - `auto_sm`

    Turn on to use shared-memory transparently
  - `no_bulk_eager`
  - `no_loopback`

    Turn off if not needed to improve performance
  - (hint for eager size limit)

Mercury – Contexts and Multi-threading



Mercury – UCX Plugin

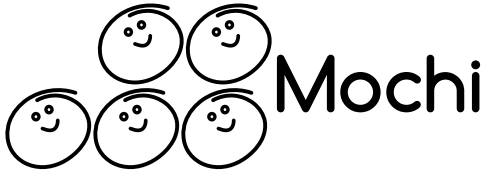
Mochi

- Uses UCP API
  - Transport selection/method is transparent (no need for explicit implementation support)
  - Class / Context creation → `ucp_worker_create()`
  - Send expected/unexpected → `ucp_tag_send_nbx()`
  - Recv expected/unexpected → `ucp_tag_recv_nbx()`
  - Put → `ucp_put_nbx()`
  - Get → `ucp_get_nbx()`
  - Progress → `ucp_worker_progress()`
- Current limitations
  - Initialization config options passed through UCX_XXX environment variables
  - Single UCP worker per class shared between contexts
  - Blocking progress not yet implemented

Credit: David Young (The HDF Group)

Wrapping Up

Thanks for being here!



- We're excited by all the interest that Mochi is garnering!
- We would like to meet one-on-one if you're interested:
  - Sign ups are at the URL below, or reach out to one of us
  - https://www.signupgenius.com/go/5080b48a4ac22a2fa7-mochi

- Any questions in our last couple of minutes?