DAGSTUHL SEMINAR 17202

CHALLENGES AND OPPORTUNITIES OF USER-LEVEL FILE SYSTEMS FOR HPC

BUILDING BLOCKS FOR USER-LEVEL HPC STORAGE SYSTEMS

PHIL CARNS
Mathematics and
Computer Science Division
Argonne National Laboratory

May 14 – 19 , 2017

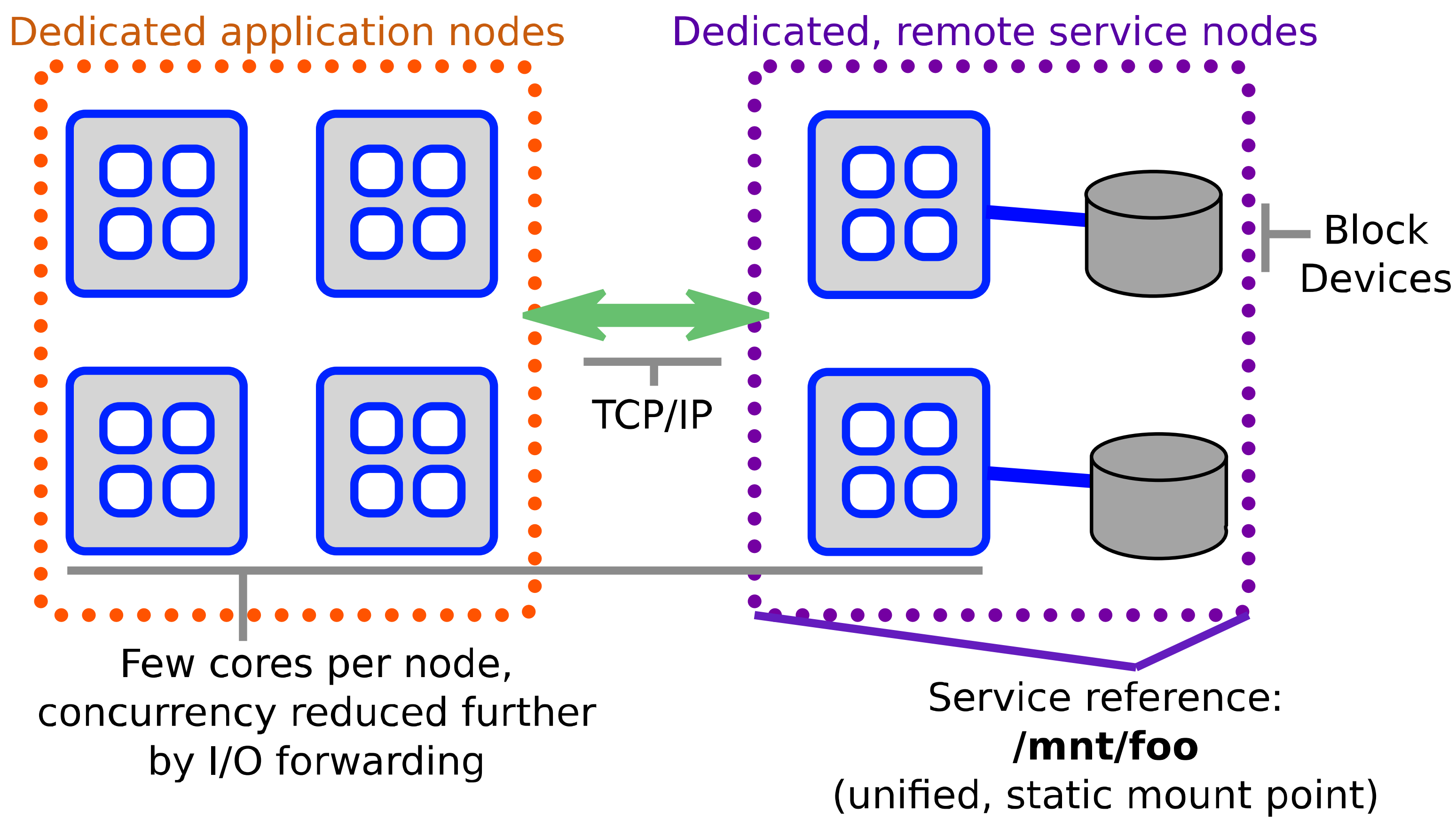
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH

WHY WOULD WE STILL TALK ABOUT DATA SERVICE BUILDING BLOCKS?

(OR ULFS, ULSS, OR MIDDLEWARE BUILDING BLOCKS…)

THE EVOLUTION OF STORAGE SERVICES

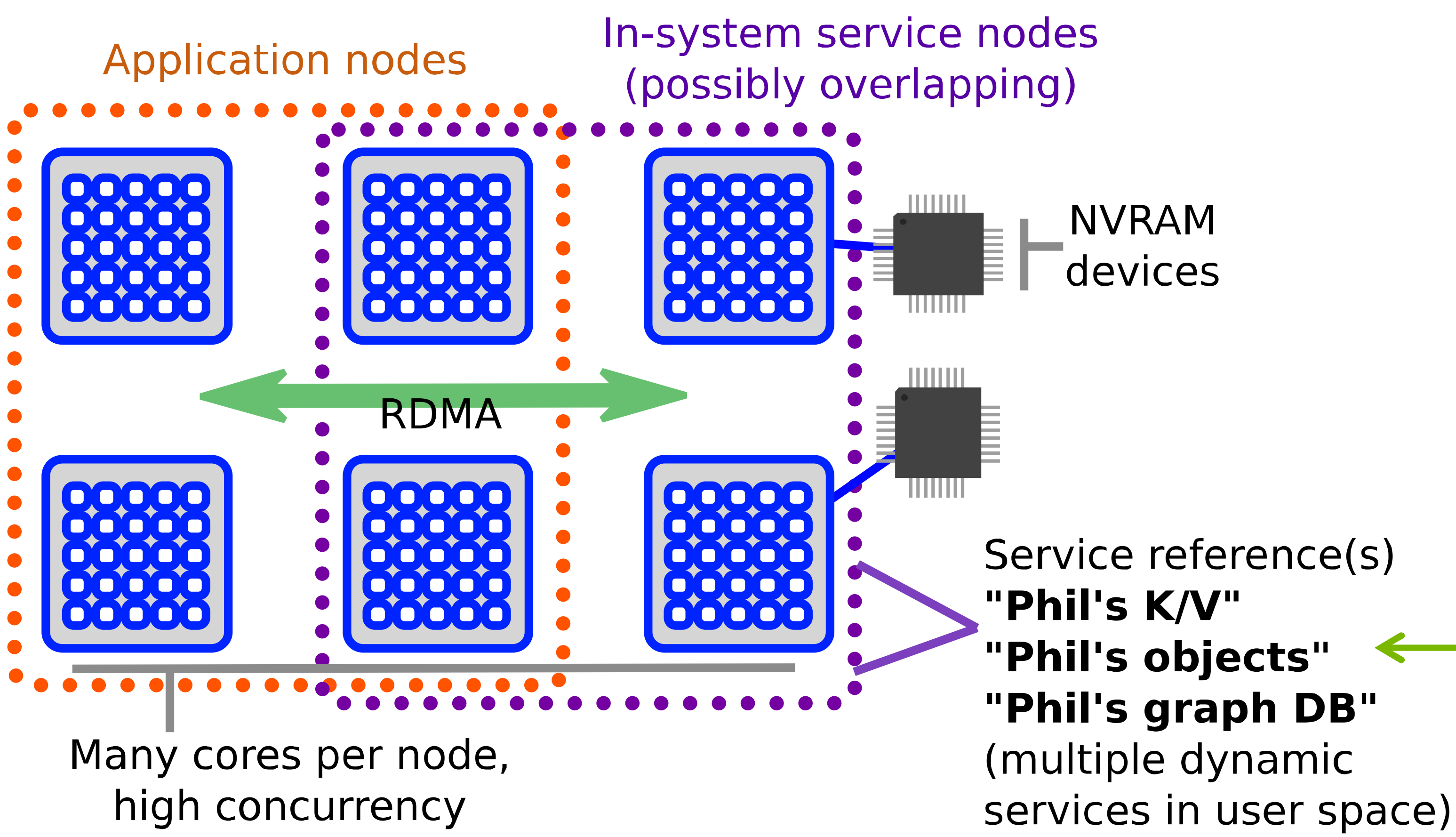
Let's reflect on our roots: a conventional storage service architecture



- We've drawn this figure (or versions of it) for a long time
- Has a lot in common with scalable Internet services
- Key technologies: *block devices, sockets, pthreads, kernel drivers*
- How did we tie them together? Did it really matter that much when operations took ***milliseconds*** to complete?

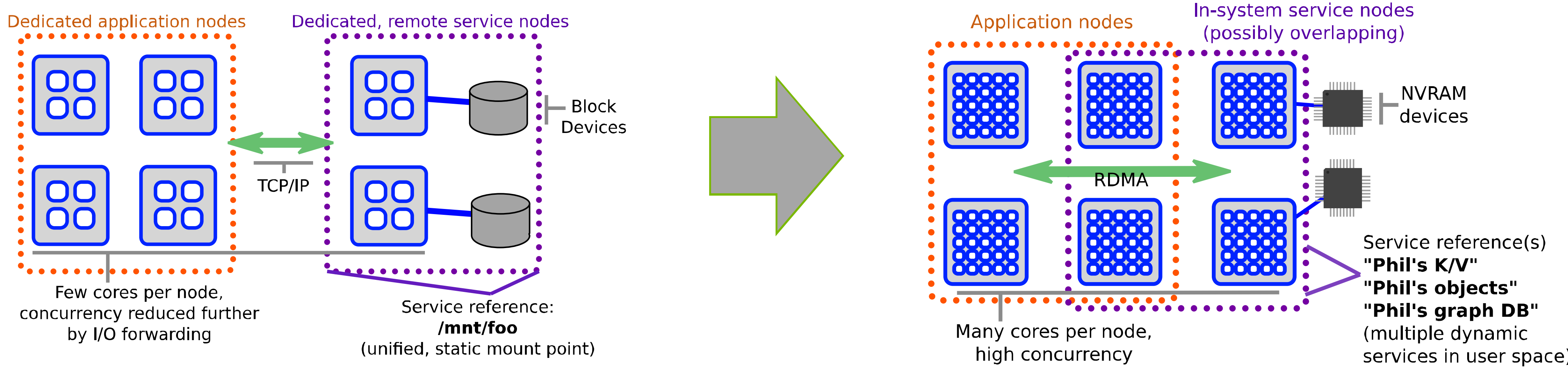
THE EVOLUTION OF STORAGE SERVICES

Next generation storage service architecture:



- Updated version of the same storage service diagram
- Key technologies: *NVRAM, RDMA, dynamic services, higher concurrency*
- Latency and jitter are more of a problem now than ever
- The high level dynamic service organization would look familiar to an Amazon Web Services user, though

HOW DO WE GET THERE?

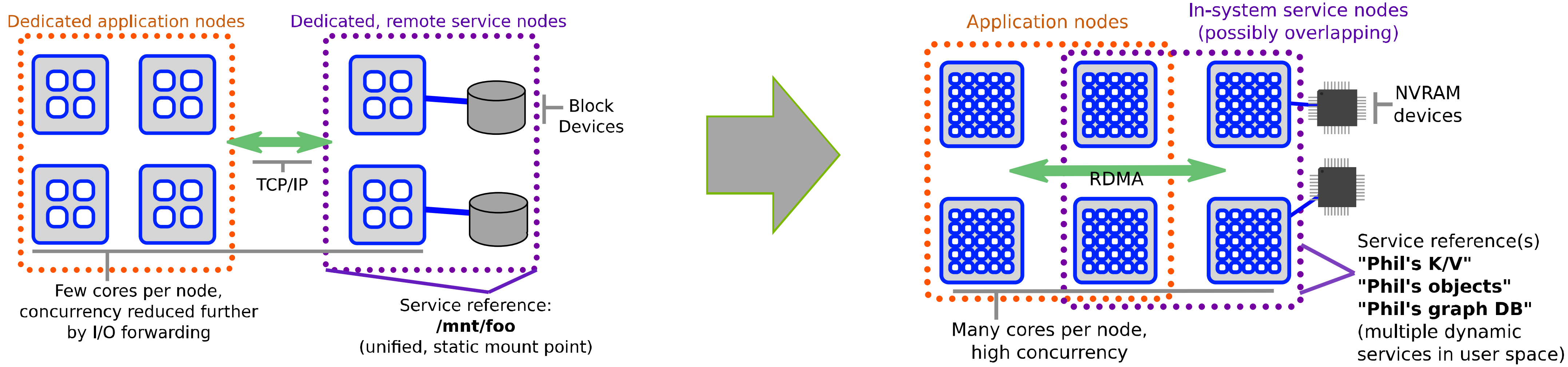


Technology transitions	
Block devices	Byte-addressable memory
Sockets	Small messages and RDMA
Pthreads or event loops	Lightweight threads
Monolithic services	Dynamic service groups
Remote daemons	Local and remote daemons

Every man for himself?

Maybe we can share engineering components and free up time for research!

HOW DO WE GET THERE?



Technology transitions		Example components
Block devices	Byte-addressable memory	NVML / libpmem
Sockets	Small messages and RDMA	Mercury RPCs
Pthreads or event loops	Lightweight threads	Argobots
Monolithic services	Dynamic service groups	SSG
Remote daemons	Local and remote daemons	Mercury component composition

Argonne NATIONAL LABORATORY

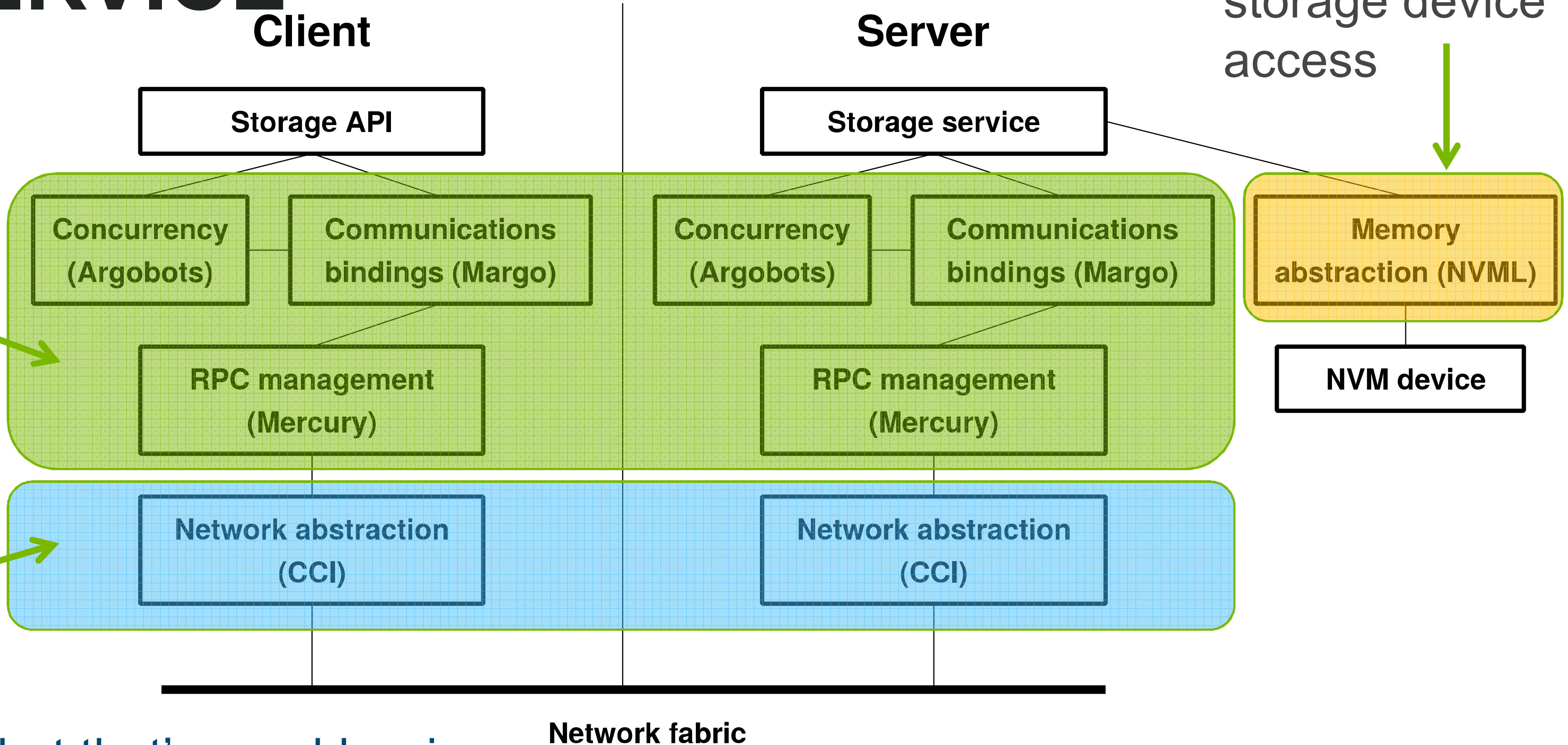
USER-LEVEL DATA SERVICE BUILDING BLOCK EXAMPLES

Argonne NATIONAL LABORATORY

EXAMPLE SERVICE DATA PATH

“Glue” holding resources together

User-space network access (could be OFI/libfabic)



User-space storage device access

Looks like many “layers”, but that’s a red herring when fishing for optimizations as long as the interactions between the layers are efficient:

The keys to optimization have changed:

- Avoiding privileged mode transitions
- Avoiding context switches in general
- Avoiding memory copies

P. Carns et al., “Enabling NVM for data-intensive scientific services,” in 4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 16), 2016.

BRIEFLY: LIBPMEM (NVML)

- A user space library for access to persistent memory regions, by Intel
- Just load/stores, right? What would you want an API for?
  - Well-defined control over persistence
  - Well-defined device naming/reference conventions
  - A family of derived libraries for data structures that understand persistent memory references, transactions, atomicity, etc:
    - Libpmemobj: object storage
    - Libpmemblk: fixed-size blocks
    - Libvmmalloc: malloc() replacement

      …
    - Pmemfile: file system in user space with no kernel VFS or block device
      - (i.e., a user-level file system)

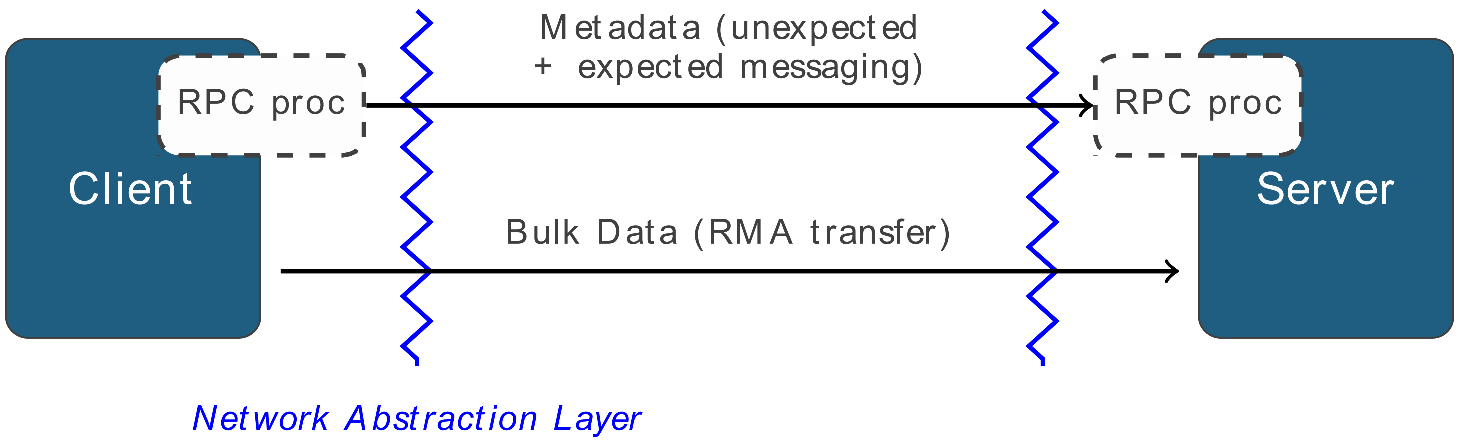
  - http://pmem.io/nvml/

COMMUNICATION

MERCURY:
A HIGH PERFORMANCE RPC FRAMEWORK

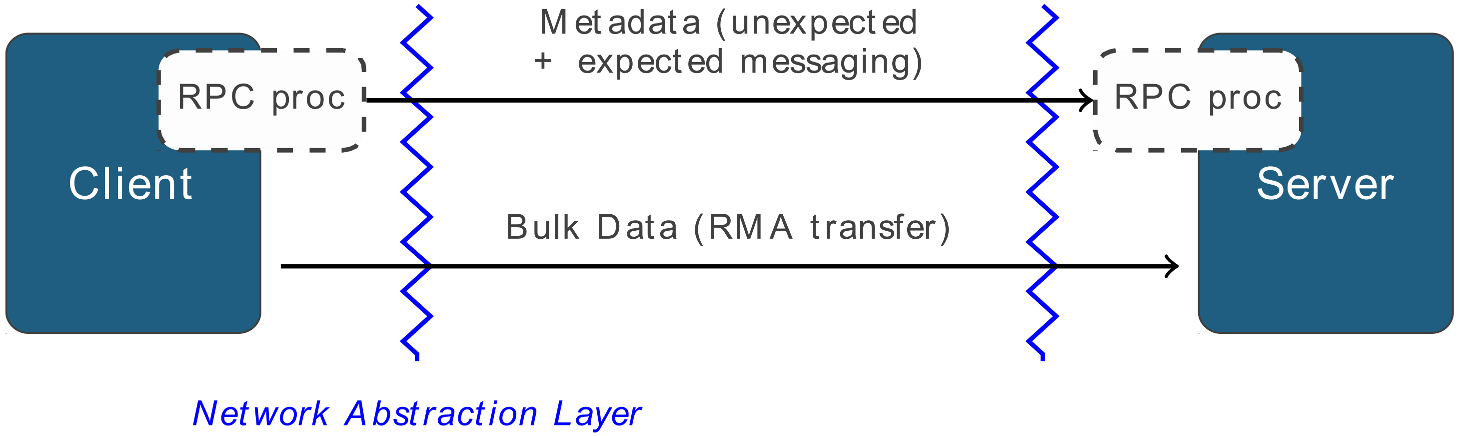
Mercury is an RPC system for use in the development of high performance system services. Developed by the HDF Group and ANL

- Portable across systems and network technologies
- Efficient bulk data movement to complement control messages
- Builds on lessons learned from IOFSL, Nessie, lnet, and others
- https://mercury-hpc.github.io/



MERCURY: A LITTLE MORE CONTEXT

- It's *not* a competitor to Portals, verbs, libfabric, GNI, etc.
  - …because it's not a communication transport library
  - It sits on top of transport libraries (using an plugin API for network abstractions)
- Provides simplifications for service implementers:
  - Remote procedure calls
  - RDMA abstraction (or emulation)
  - Protocol encoding
  - Clearly defined progress and event model
  - No restrictions on client/server roles
  - No global fault domain (MPI_COMM_WORLD)



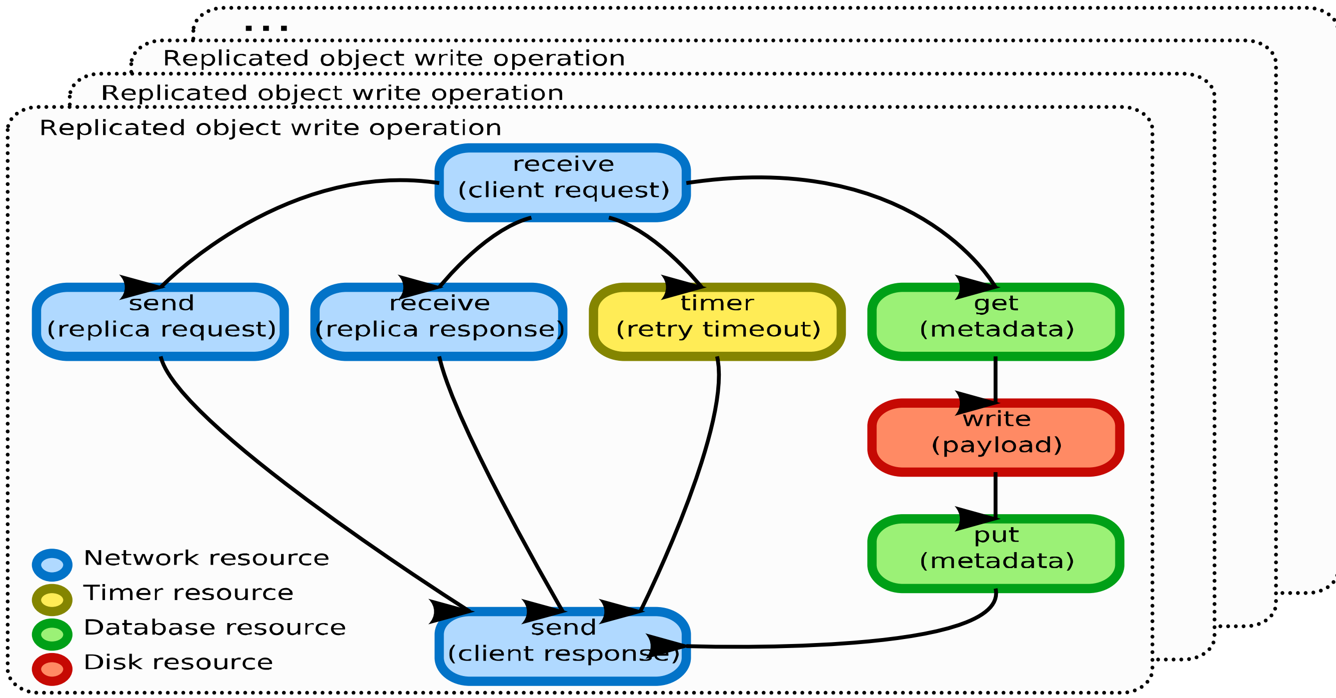
APPLYING MERCURY TO HPC DATA SERVICES

- Mercury has an abstraction layer for network transports. Notable examples:
  - **OFI**: via native plugin (Intel contribution)
  - **IB**: via CCI plugin (ORNL contribution)
  - **TCP/IP**: via BMI plugin
  - **SHMEM**: via internal plugin
  - **Local**: via internal code path, skips plugins
- There is a price, though: Mercury is minimal and fast, but a challenge to program (it is inherently event-driven)
  - We added a layer called “Margo” that adds an easy-to-use sequential interface
  - Relies on user-level thread scheduling for concurrency
  - Greatly simplifies service development while retaining native performance
  - https://xgitlab.cels.anl.gov/sds/margo/

CONCURRENCY

WHAT PROBLEM IS ARGOBOTS SOLVING BESIDES MAKING MERCURY A BIT EASIER TO PROGRAM?

- Observation from earlier: *many cores, and more I/O concurrency than there are cores on storage service nodes*
  - Example: N concurrent replicated write operations on one server daemon
  - Illustrated as a state machine in this figure (the arrows are weird; blame powerpoint)
  - Each request involves multiple steps: some will block, some will branch
  - The server needs to keep track of where each request is in state machine



- Confounding matters, each resource probably has its own progress/completion model

WE'VE TRIED A FEW PRODUCTIVITY SOLUTIONS…

- Threads (per request, or thread pools for various steps)
  - Context switch cost could be high
  - Might cause undue contention on network resource (i.e. who is driving the device?)
  - Wasteful if the threads are there only to wait, not compute
- Event-driven (events from async resources or from thread pools)
  - Has potential to avoid extraneous context switch cost
  - Maintenance and developer ramp-up cost is high due to stack ripping
- PVFS state machines
  - Formalize the event-driven transitions and makes them easier to conceptualize
  - Still stack ripping, though
- Aesop (from Triton/ASG project)
  - C language extensions and runtime library
  - Hides stack ripping and continuation logic, looks (mostly) like pleasant linear C code
  - The dark side of Aesop: trades a service maintenance burden for an even worse language extension maintenance burden!

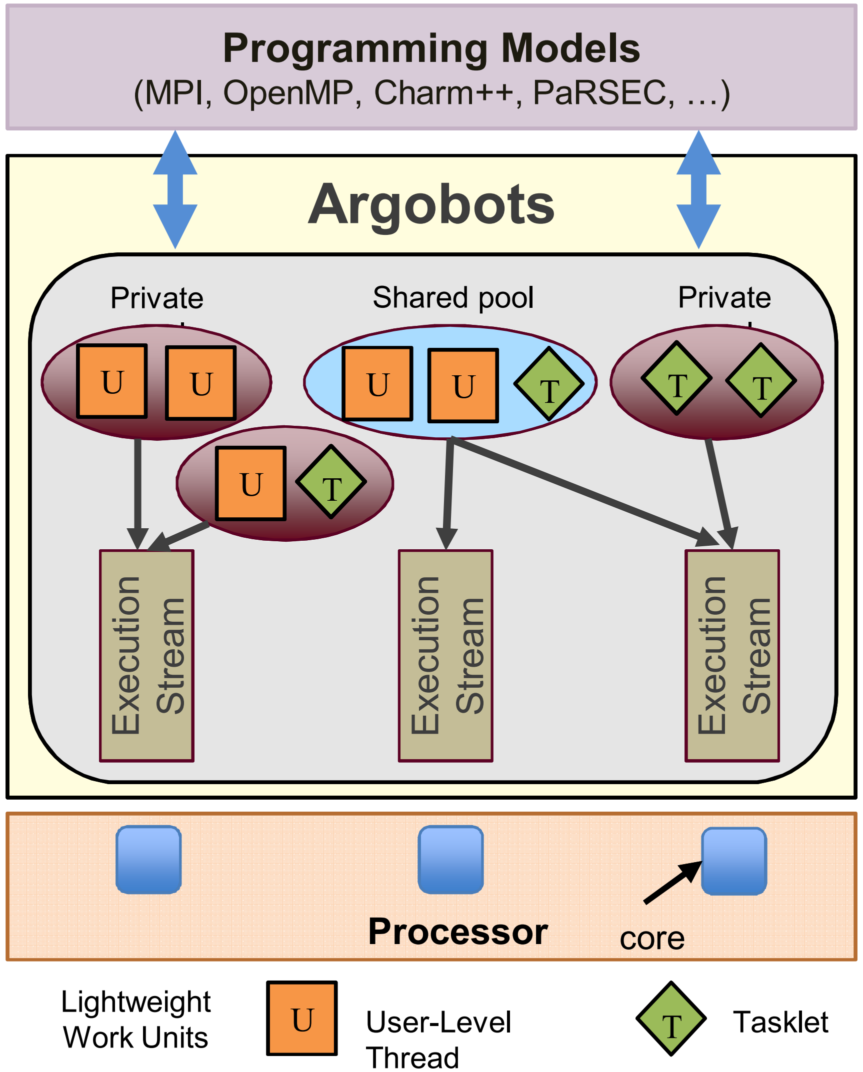
ARGOBOTS: A LIGHTWEIGHT THREADING/TASKING FRAMEWORK

Overview

- User-level threading: lightweight context switching among many concurrent threads
- Use multiple cores and control delegation of work to those cores
- http://argobots.org/

Key features for data services

- Lets us track state of **many** concurrent operations with simple service code paths and low OS resource consumption
- Custom schedulers (i.e., to implement priorities, or limit CPU usage)
- Primitives that facilitate linkage to external resources



MORE THAN JUST FAST DATA PATHS: HOW TO ORGANIZE DYNAMIC SERVICES

Argonne
NATIONAL LABORATORY

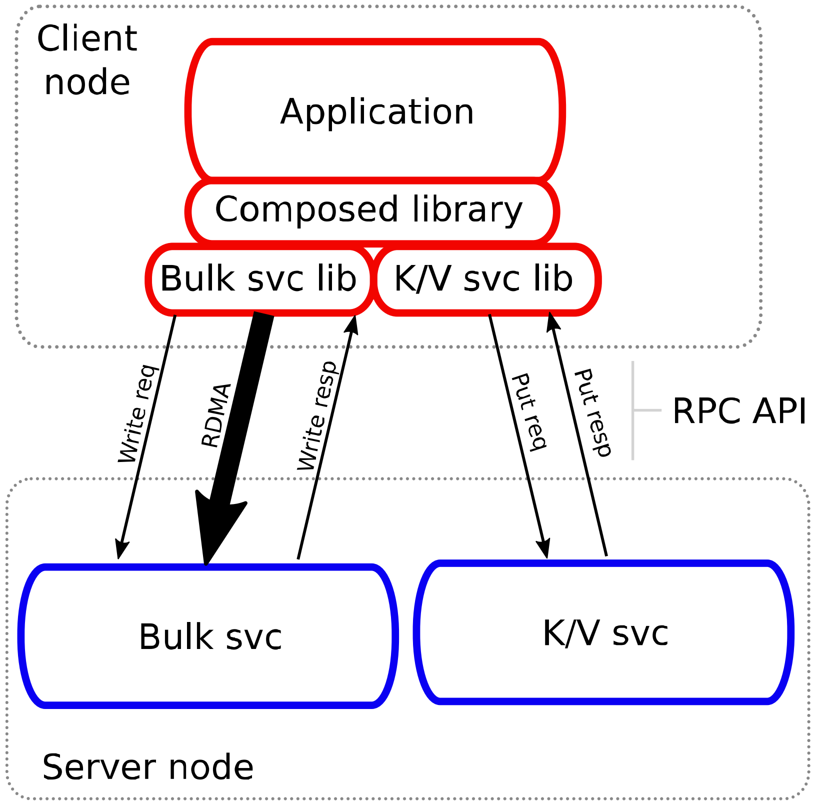
GLUING MULTIPLE COMPONENTS TOGETHER

Or: a story of software engineering with Mercury APIs

- Consider the example of 2 or more simple microservices that are combined to present a more complex data model
  - One service/component for bulk data storage
  - One service/component for key/value indexing
  - Combined, they allow you to create objects with custom names and indices
- E.g., **Create a 100 MiB object named "Fred"**
  1. Write 100 MiB into bulk data region
  2. Put key="Fred" value=data_region_reference into K/V store
- Complete file systems or scientific data models may require more complex combinations (multiple dimensions, other data structures, etc.)

COMPOSED AND CO-LOCATED SERVICES

Option 1 (“We can do this the slow way”)

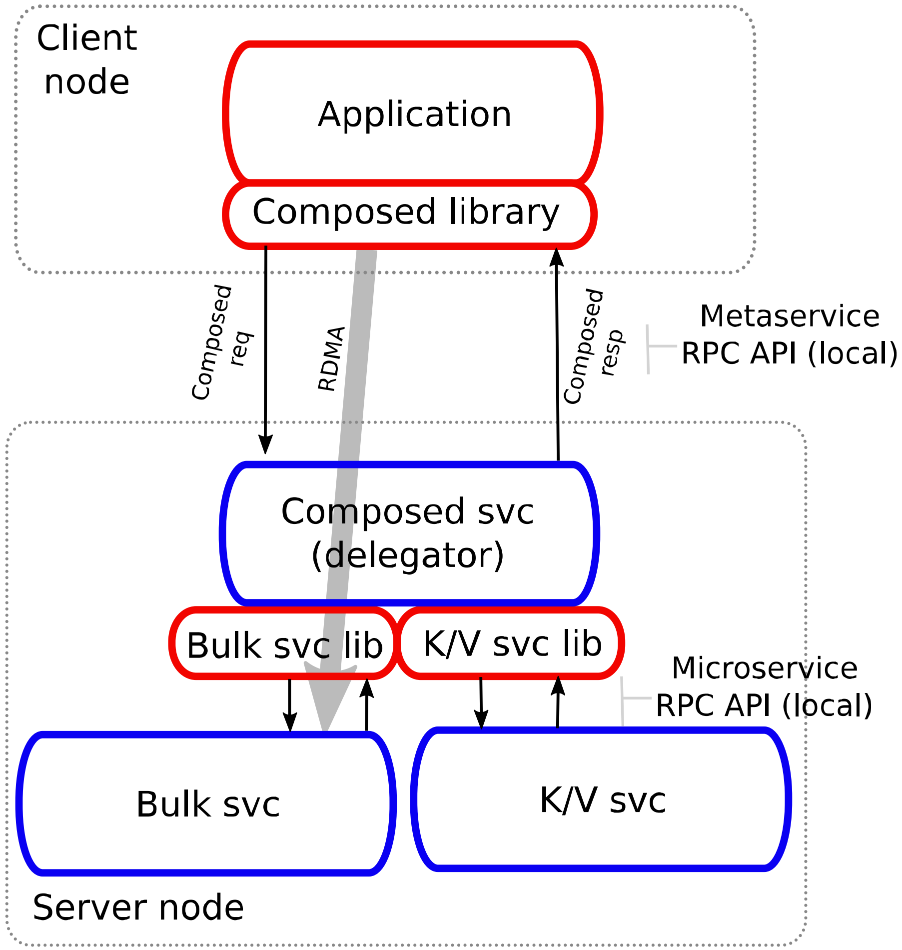


- Expose RPC API for each microservice to the client node
- Present a “composed library” to the application that so that it can create Fred objects
- Flexible: services could be anywhere
- ***Performance problem: 2 RPC round trips where you would only expect one in a conventional monolithic service***

- Compound/chain RPCs could solve this in theory, but difficult to implement and limit flexibility

COMPOSED AND CO-LOCATED SERVICES

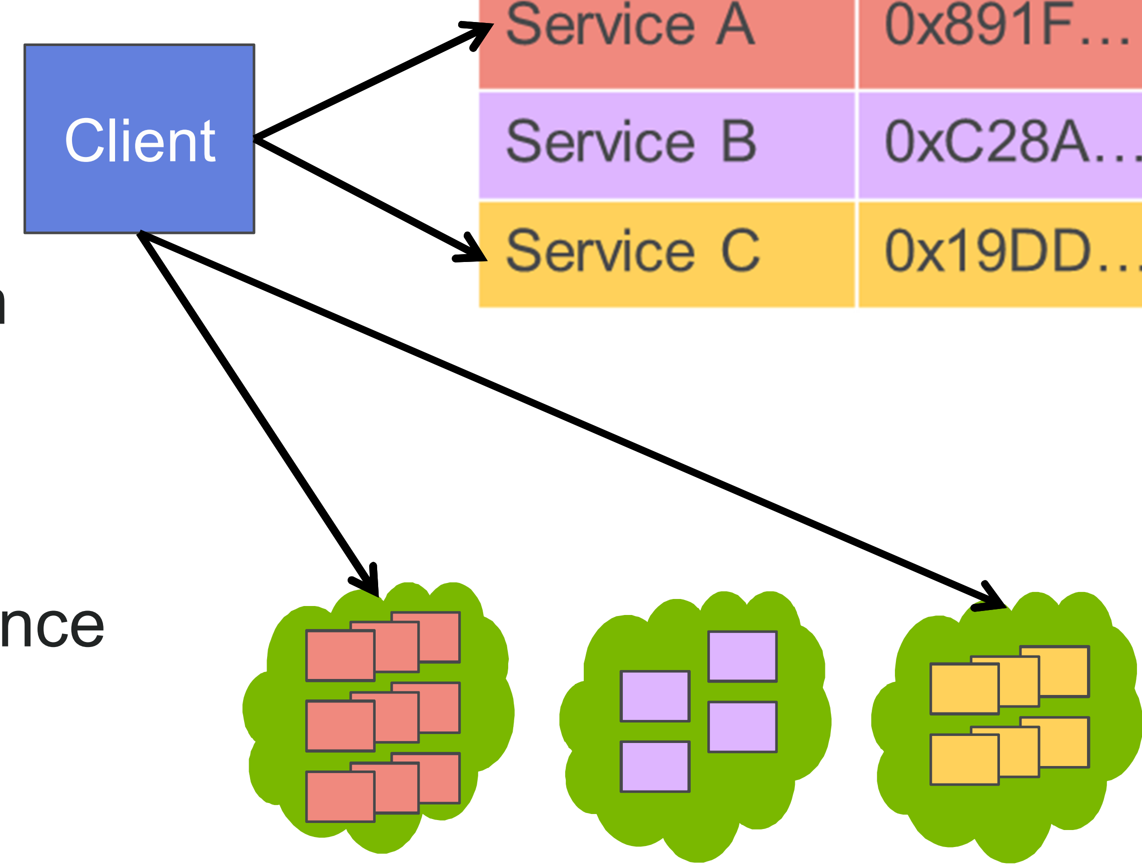
Option 2 (“I/O forwarding for high-level APIs”)



- Composed library exchanges RPCs with a composed service
- The composed service doesn’t do much:
  - Delegates RPCs to microservices
  - Propagates RDMA token (Mercury bulk handle) to the service(s) that will drive data transfer
- More generally, ***we can mix and match remote and non-remote components with the same API conventions; no change to service implementation***
- *Key question: how fast is local RPC delegation?* We will rely on fast paths in Mercury for intra-process and intra-node channels

HOW DO WE ORGANIZE MULTIPLE DYNAMIC SERVICES?

Group Name	Group ID
Service A	0x891F…
Service B	0xC28A…
Service C	0x19DD…



- SSG: scalable service groups
- https://xgitlab.cels.anl.gov/sds/ssg
- Adds group membership to Mercury
  - Group bootstrapping / wire-up / launch
    - Config file
    - MPI communicator
    - PMIx (one day)?
  - Provides identifiers to concisely reference groups of processes
  - Provides optional fault detection
- We think of this, combined with a pub/sub registry service, as scaffolding for assembling composable services

BACK TO THE BIG PICTURE

OBSERVATIONS

- New architectures and new hardware resources call for new building blocks for user-level storage services
- We can potentially speed up development and ease maintenance by sharing these building blocks
  - These will never be optimal for every use case
  - Consider when expertise/resources/use case warrant hand coding
- Ideally we can share not only low-level building blocks, but fundamental microservices: compose them and augment to serve use case
- Clear at this point in seminar that we need flexibility in provisioning and composition too
- **http://www.mcs.anl.gov/research/projects/mochi/**
  - Thanks to many at ANL, The HDF Group, LANL, and CMU

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.

THIS RESEARCH USED RESOURCES OF THE ARGONNE LEADERSHIP COMPUTING FACILITY, WHICH IS A DOE OFFICE OF SCIENCE USER FACILITY SUPPORTED UNDER CONTRACT DE-AC02-06CH11357.