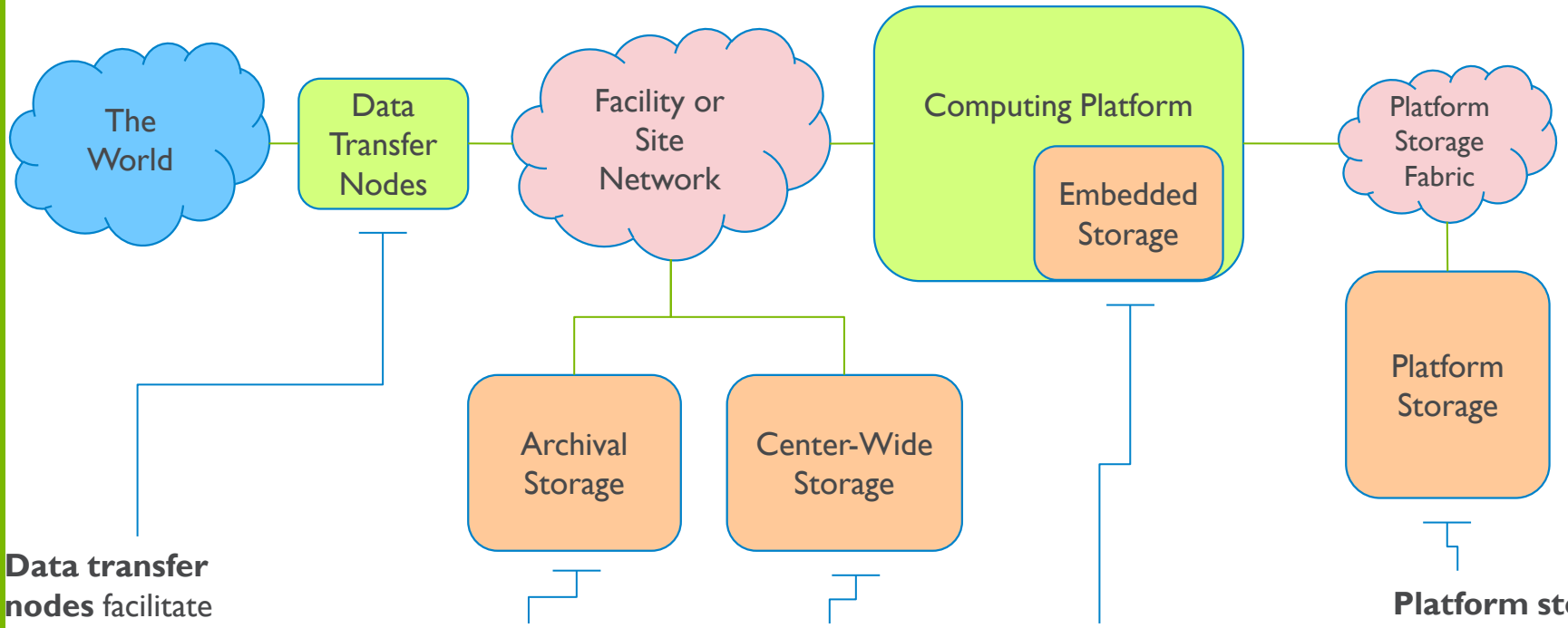
WHAT LQCD SHOULD KNOW ABOUT STORAGE (IN 30 MINUTES)

ROB LATHAM
Research Software Developer
Math and Computer Science

LQCD Kickoff,
Dec 2022

STORAGE IN AND AROUND THE PLATFORM



Data transfer nodes facilitate bulk data movement in/out of the facility.

Archival storage holds infrequently accessed data for extended time periods.

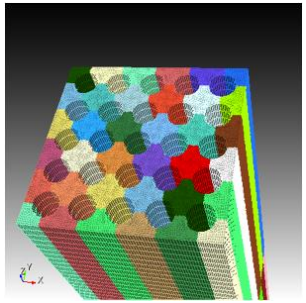
Center-wide storage allows sharing of data between systems.

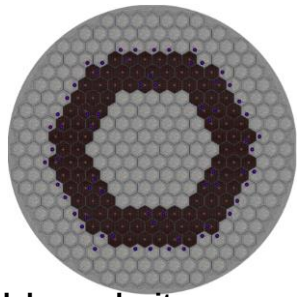
Embedded storage is high BW, low latency, and located on node or in the platform fabric.

Platform storage provides high capacity storage for data between runs.

APPLICATION DATASET COMPLEXITY VS. I/O

- I/O systems have very simple data models
  - Tree-based hierarchy of containers
  - Some containers have streams of bytes (files)
  - Others hold collections of other containers (directories or folders)
- Applications have data models appropriate to domain
  - Multidimensional typed arrays, images composed of scan lines, records of variable length
  - Headers, attributes on data
- Someone has to map from one to the other!





Model complexity: Spectral element mesh (top) for thermal hydraulics computation coupled with finite element mesh (bottom) for neutronics calculation

Scale complexity: Spatial range from the reactor core, in meters, to fuel pellets, in millimeters

Images from T. Tautges (Argonne) (upper left), M. Smith (Argonne) (lower left), and K. Smith (MIT) (right).

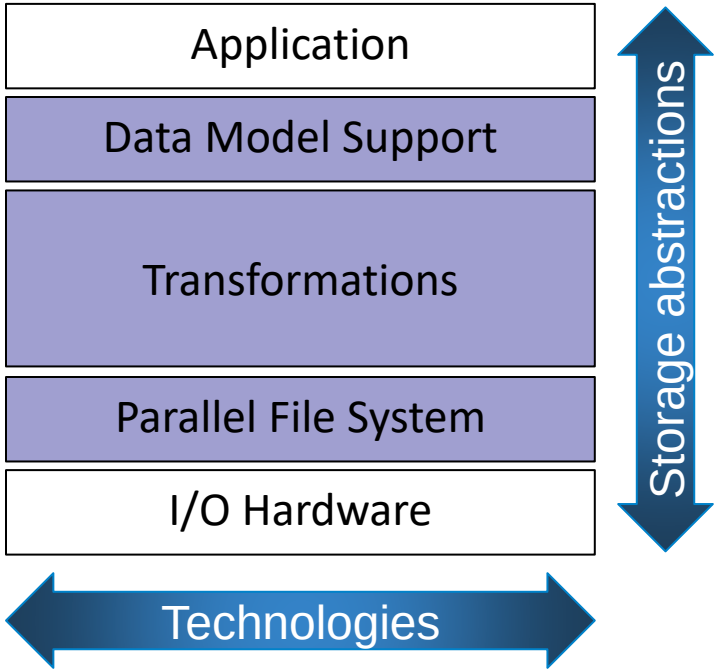
SURVEYING THE HPC I/O LANDSCAPE

A complex data management ecosystem

As evidenced by today's presentations, the HPC I/O landscape is deep and vast

- High-level data abstractions: HDF5, PnetCDF
- Parallel file systems: Lustre, GPFS
- Storage hardware: HDDs, SSDs, NVM

Understanding I/O behavior in this environment is difficult, much less turning observations into actionable I/O tuning decisions



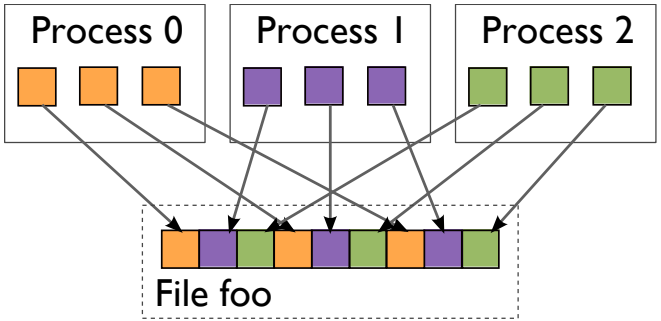
MPI-IO

- I/O interface specification for use in MPI apps
- Data model is same as POSIX: stream of bytes in a file
- Features many improvements over POSIX:
  - Collective I/O
  - Noncontiguous I/O with MPI datatypes and file views
  - Nonblocking I/O
  - Fortran bindings (and additional languages)
  - System for encoding files in a portable format (external32)
    - Not self-describing – just a well-defined encoding of types
- Probably not going to use this directly, but powers higher level libraries (e.g. HDF5, or application-specific abstraction)

I/O TRANSFORMATIONS

Software between the application and the PFS performs transformations, primarily to improve performance

- Goals of transformations:
  - Reduce number of I/O operations to PFS (avoid latency, improve bandwidth)
  - Avoid lock contention (eliminate serialization)
  - Hide huge number of clients from PFS servers
- "Transparent" transformations don't change the final file layout
  - File system is still aware of the actual data organization
  - File can be later manipulated using serial POSIX I/O



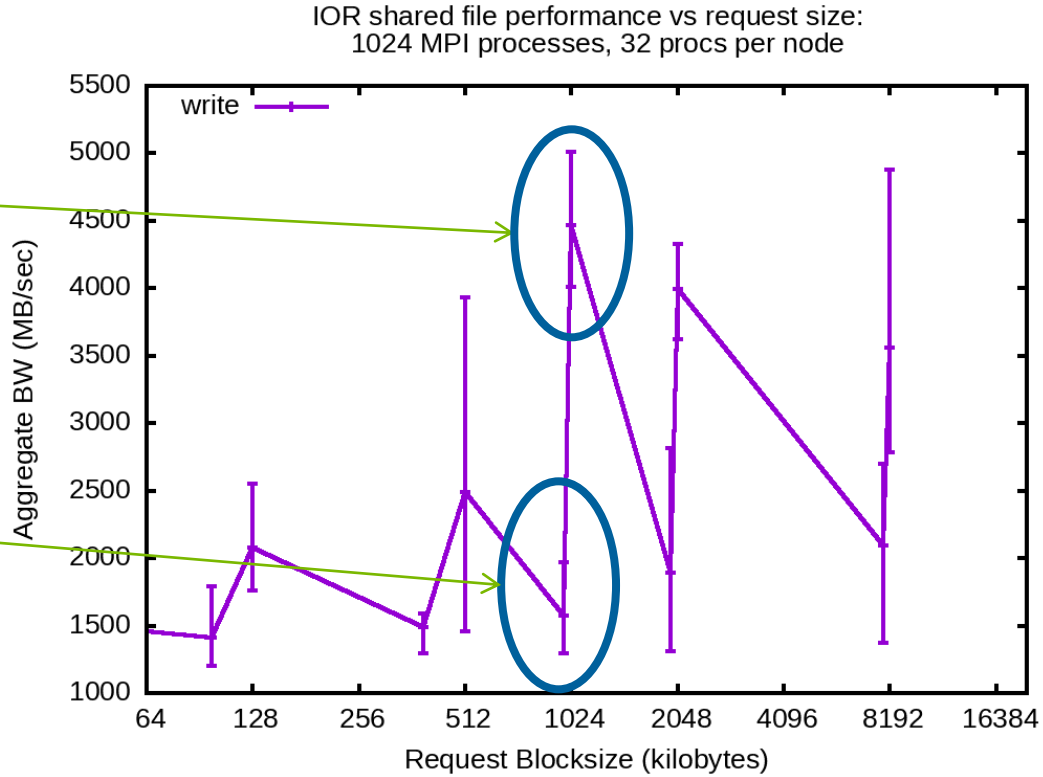
When we think about I/O transformations, we consider the mapping of data between application processes and locations in file

REQUEST SIZE AND I/O RATE

Request matches Lustre “stripe size”: good performance with low variability

Small deviations from “power of two” (e.g. 1024k vs 10^6) can tank performance

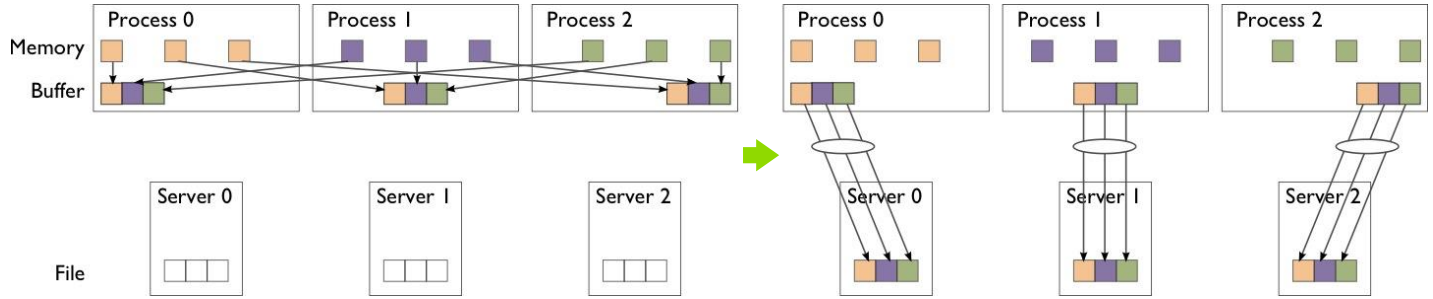
In general, larger requests better.



Tests run on 1K processes of HPE/Cray Theta at Argonne

AVOIDING LOCK CONTENTION

- **To avoid lock contention when writing to a shared file, we can reorganize data between processes**
- *Two-phase I/O* splits I/O into a data reorganization phase and an interaction with the storage system (two-phase write depicted):
  - Data exchanged between processes to match file layout
  - 0th phase determines exchange schedule (not shown)



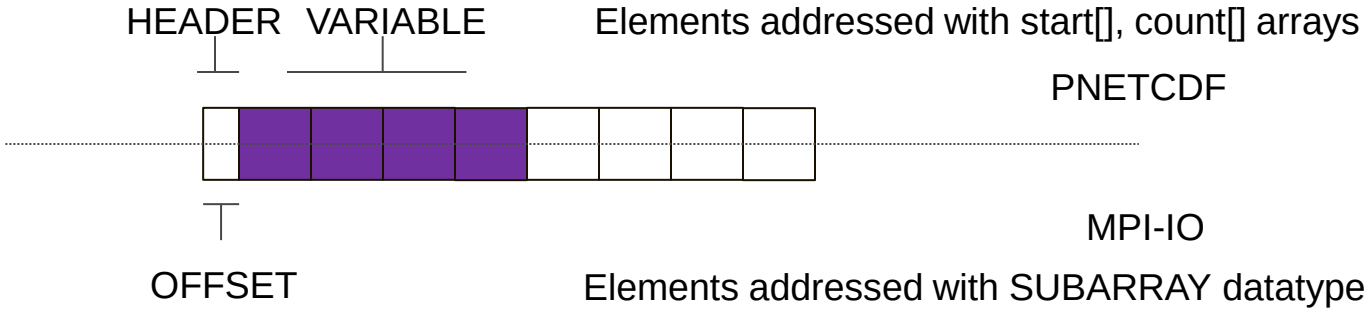
Phase 1: Data are exchanged between processes based on organization of data in file.

Phase 2: Data are written to file (storage servers) with large writes, no contention.

PARALLEL NETCDF (PNETCDF)

- Based on original “Network Common Data Format” (netCDF) work from Unidata
  - Derived from their source code
- Data Model:
  - Collection of variables in single file
  - Typed, multidimensional array variables
  - Attributes on file and variables
- Features:
  - C, Fortran, and F90 interfaces
  - Portable data format (identical to netCDF)
  - Noncontiguous I/O in memory using MPI datatypes
  - Noncontiguous I/O in file using sub-arrays
  - Collective I/O
  - Non-blocking I/O
- Unrelated to netCDF-4 work
- Parallel-NetCDF tutorial:
  - https://parallel-netcdf.github.io/wiki/QuickTutorial.html

PARALLEL-NETCDF AND MPI-IO



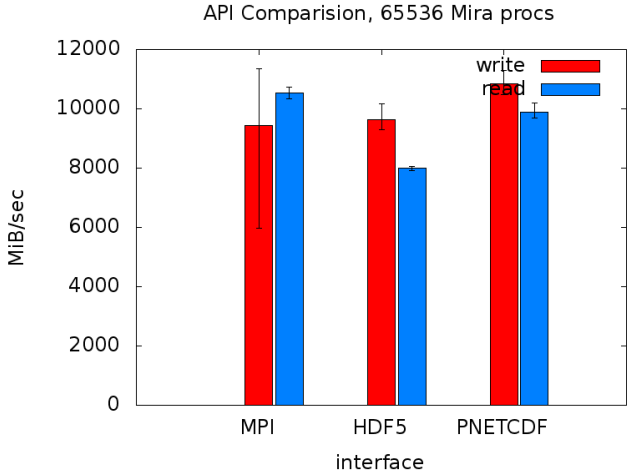
- `ncmpi_put_vara_all` describes access in terms of arrays, elements of arrays
  - For example, “Give me a 3x3 subcube of this larger 1024x1024 array”
- Library translates into MPI-IO calls
  - `MPI_Type_create_subarray`
  - `MPI_File_set_view`
  - `MPI_File_write_all`

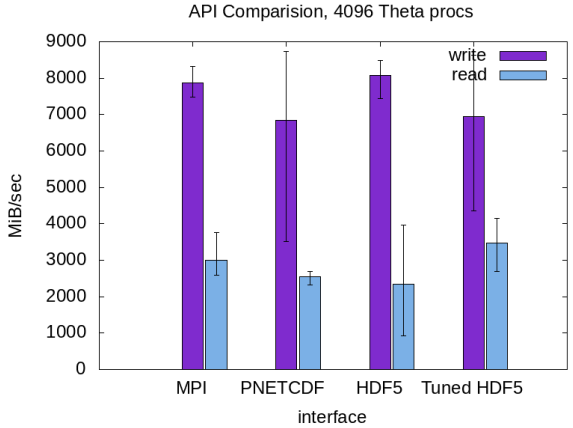
HDF5

- Hierarchical Data Format, from The HDF Group (formerly of NCSA)
  - https://www.hdfgroup.org/
- Data Model:
  - Hierarchical data organization in single file
  - Typed, multidimensional array storage
  - Attributes on any HDF5 "object" (dataset, data, groups)
- Features:
  - C, C++, Fortran, Java (JNI) interfaces
    - Community-supported Python, Lua, R
  - Portable data format
  - Optional compression (even in parallel I/O mode)
  - Chunking: efficient row or column oriented access
  - Noncontiguous I/O (memory and file) with hyperslabs
- Parallel HDF5 tutorial:
  - https://portal.hdfgroup.org/display/HDF5/Introduction+to+Parallel+HDF5

COMPARING I/O LIBRARIES

- IOR to evaluate HDF5, PnetCDF somewhat artificial
  - HLL typically hold structured data
- HDF5, PnetCDF demonstrate performance parity for these access sizes (6 MiB on Mira)
- I/O libraries deliver benefits with slight (if any) cost to performance
- Active work to improve efficiency (cf. HDF5 collective metadata)



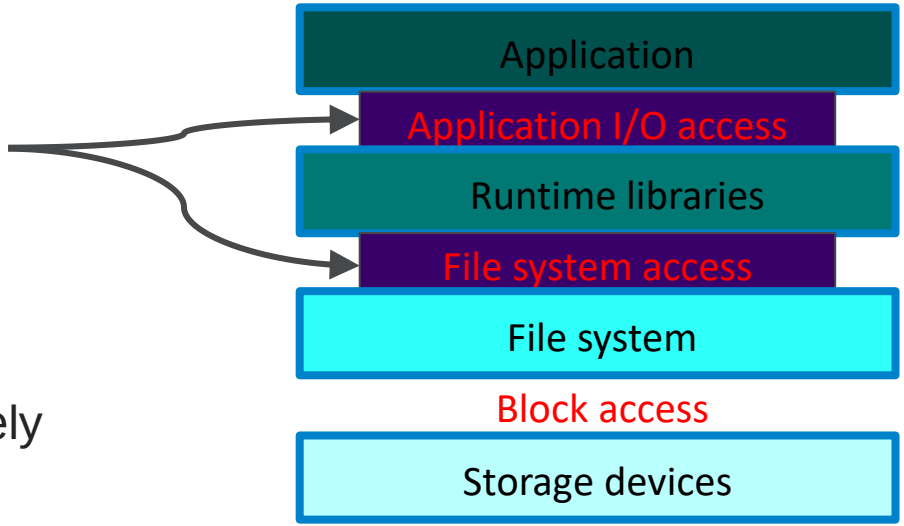


OTHER HIGH-LEVEL I/O LIBRARIES

- NetCDF-4: https://www.unidata.ucar.edu/software/netcdf
  - netCDF API with HDF5 back-end
- ADIOS: https://adios2.readthedocs.io/
  - Configurable (xml) I/O approaches
- SILO: https://wci.llnl.gov/simulation/computer-codes/silo
  - A mesh and field library on top of HDF5 (and others)
- H5part: https://gitlab.psi.ch/H5hut/src/-/wikis/home
  - simplified HDF5 API for particle simulations
- GIO: https://svn.pnl.gov/gcrm
  - Targeting geodesic grids as part of GCRM
- SCORPIO: https://e3sm.org/scorpio-parallel-io-library/
  - climate-oriented I/O library; supports raw binary, parallel-netCDF, or serial-netCDF (from master)
- … Many more. My point: likely, one already exists for your domain

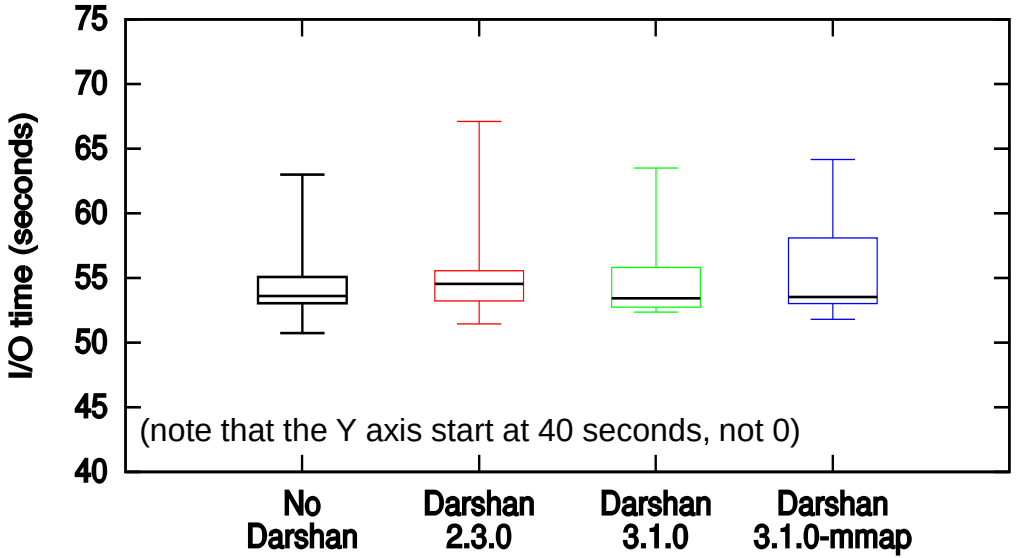
DARSHAN DESIGN PRINCIPLES

- Darshan runtime library inserted at link time or at run time
- Transparent wrappers for I/O functions collect per-file statistics
  - Statistics are stored in bounded memory at each rank
  - At MPI_Finalize(), counters are reduced, compressed, and collectively written to a single log
- No communication or storage operations until shutdown
- Command-line tools used to post-process Darshan logs



WHAT IS THE OVERHEAD OF DARSHAN I/O FUNCTION WRAPPING?

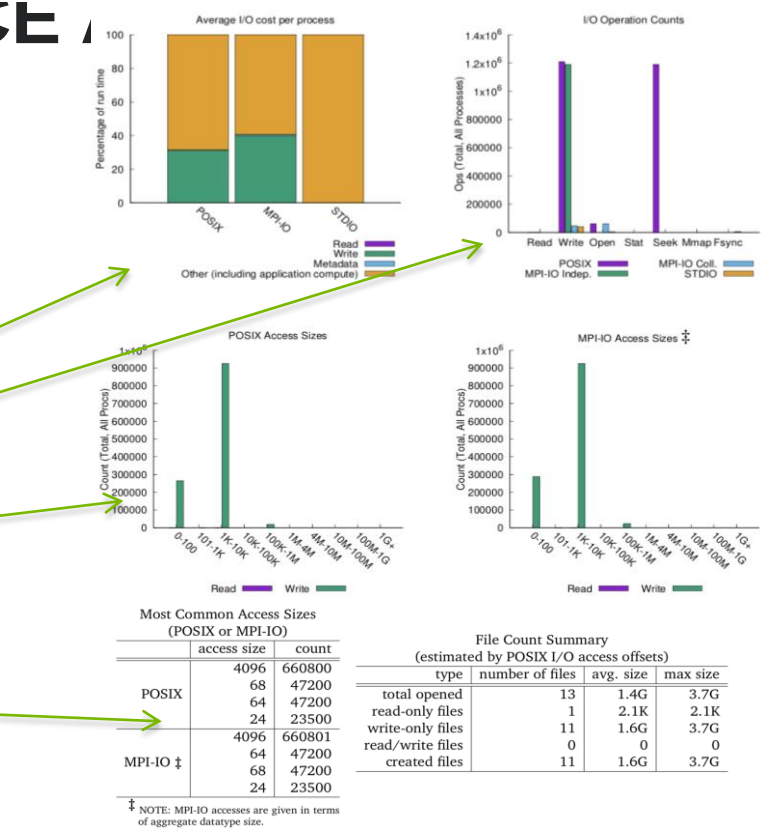
- Compare I/O time of IOR linked against different Darshan versions on *NERSC Edison*
  - File-per-process workload
  - 6,000 MPI processes
  - >12 million instrumented calls
- Note use of box plots
  - Ran each test 15 times
  - I/O variation is a reality
  - Consider I/O performance as a distribution, not a singular value



Snyder et al., "Modular HPC I/O Characterization with Darshan," in *Proceedings of 5th Workshop on Extreme-scale Programming Tools (ESPT 2016),* 2016.

JOB-LEVEL PERFORMANCE A

- Darshan provides insight into the I/O behavior and performance of a job
- **darshan-job-summary.pl** creates a PDF file summarizing various aspects of I/O performance
  - Percent of runtime spent in I/O
  - Operation counts
  - Access size histogram
  - Access type histogram
  - File usage



jobid: 9544193	uid: 59902	nprocs: 4720	runtime: 1512 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 411195 MiB at 24.77 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.4 MiB at 0.77 MiB/s

Most Common Access Sizes
(POSIX or MPI-IO)

	access size	count
POSIX	4096	660800
	68	47200
	64	47200
	24	23500
MPI-IO ‡	4096	660801
	64	47200
	68	47200
	24	23500

‡ NOTE: MPI-IO accesses are given in terms of aggregate datatype size.

File Count Summary
(estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	13	1.4G	3.7G
read-only files	1	2.1K	2.1K
write-only files	11	1.6G	3.7G
read/write files	0	0	0
created files	11	1.6G	3.7G

STRENGTHS (L) AND WEAKNESSES (R)

Two+ decades of research, with more to go

- We understand MPI applications really well
- Library (HDF5) and middleware (MPI-IO) good fit for bulk-synchronous
- File system specific optimizations (GPFS, Lustre, DAOS) implemented in MPI-IO libraries
- Application-oriented libraries (e.g. HDF5) insulate I/O tuning (bytes, offsets, servers) from application needs
- ML and AI ecosystems evolving rapidly, but haven't converged
- Strongest MPI-IO optimizations (collective) poor fit for temporally "skewed" code
- Need to adjust current approaches for task-oriented codes
- Don't have a great story for GPU codes yet

Mochi

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

Vision

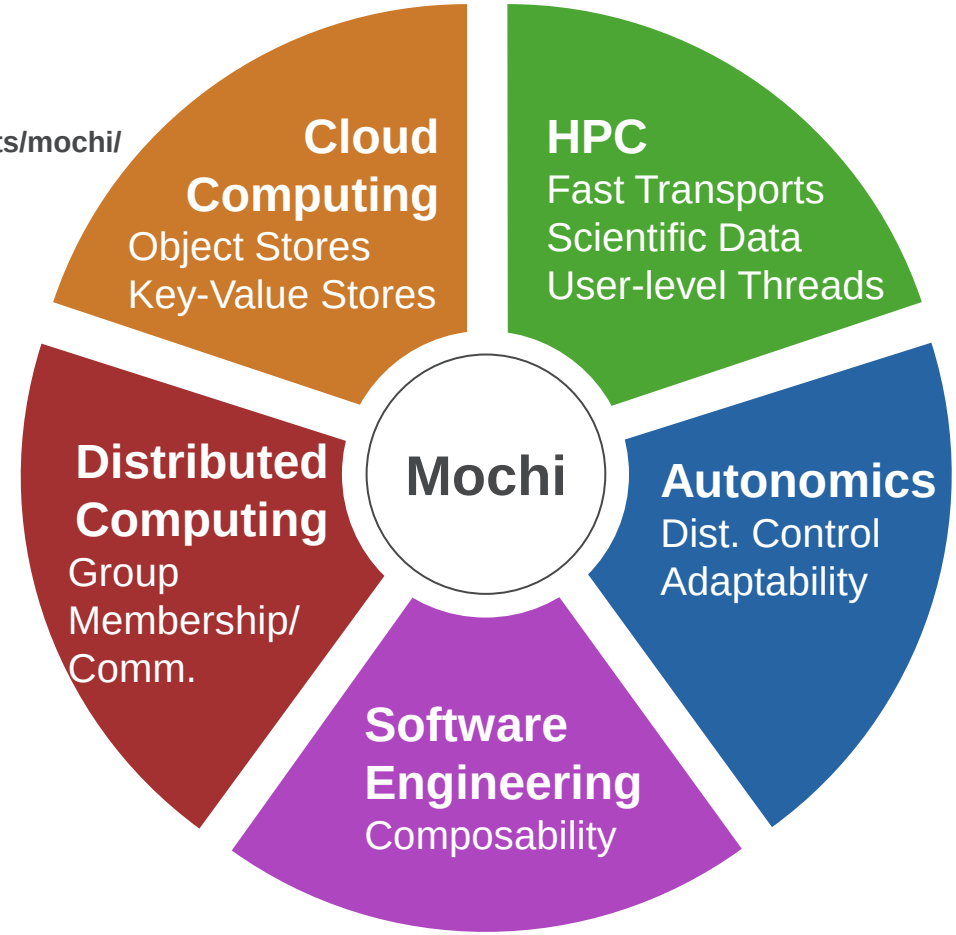
Lowering the barriers to distributed services in computational science through compositional approaches.

Approach

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

Impact

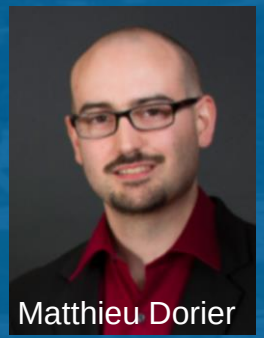
- Better, more capable services for specific use cases on high-end platforms
- Significant code reuse
- Ecosystem for rapid service development
  - Deep Hyper story



THANKS!


Rob Ross


Phil Carns


Matthieu Dorier


Kevin Harms


Kevin Brown


Rob Latham


Danqing Wu


Shane Snyder