Towards Autonomous Storage and I/O Systems for Scientific Data

Suren Byna

Computer Staff Scientist

Lawrence Berkeley National Laboratory

sbyna@lbl.gov

Keynote Speech at Computation World 2022 Congress

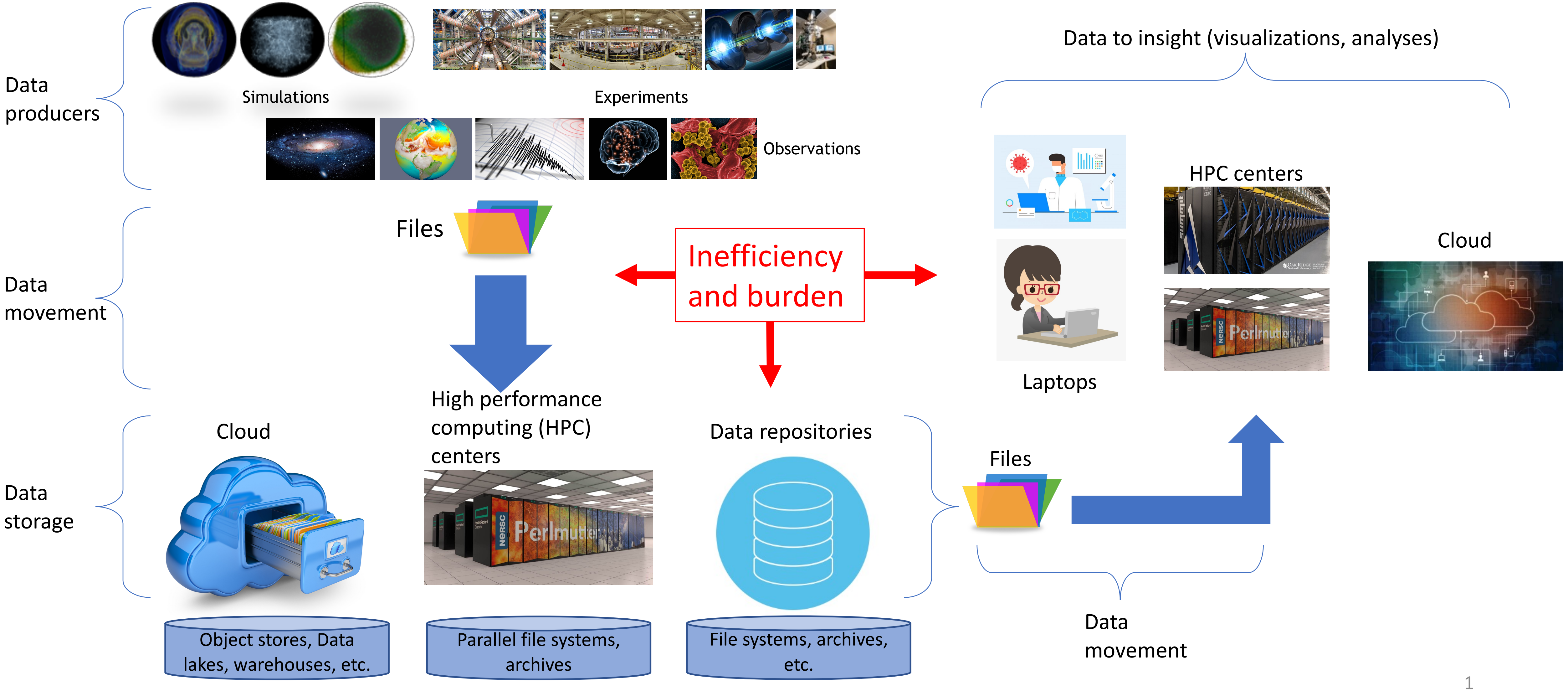
April 28th, 2022

BERKELEY LAB

Contributions from ExaIO, ExaHDF5, and Proactive Data Containers (PDC) teams

IARIA

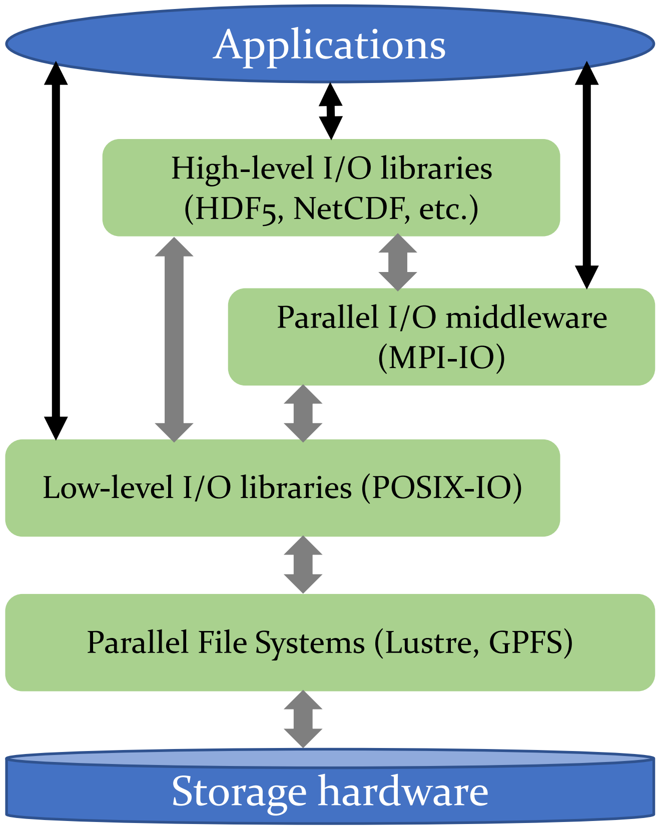
Scientific data storage and access - Current scenario



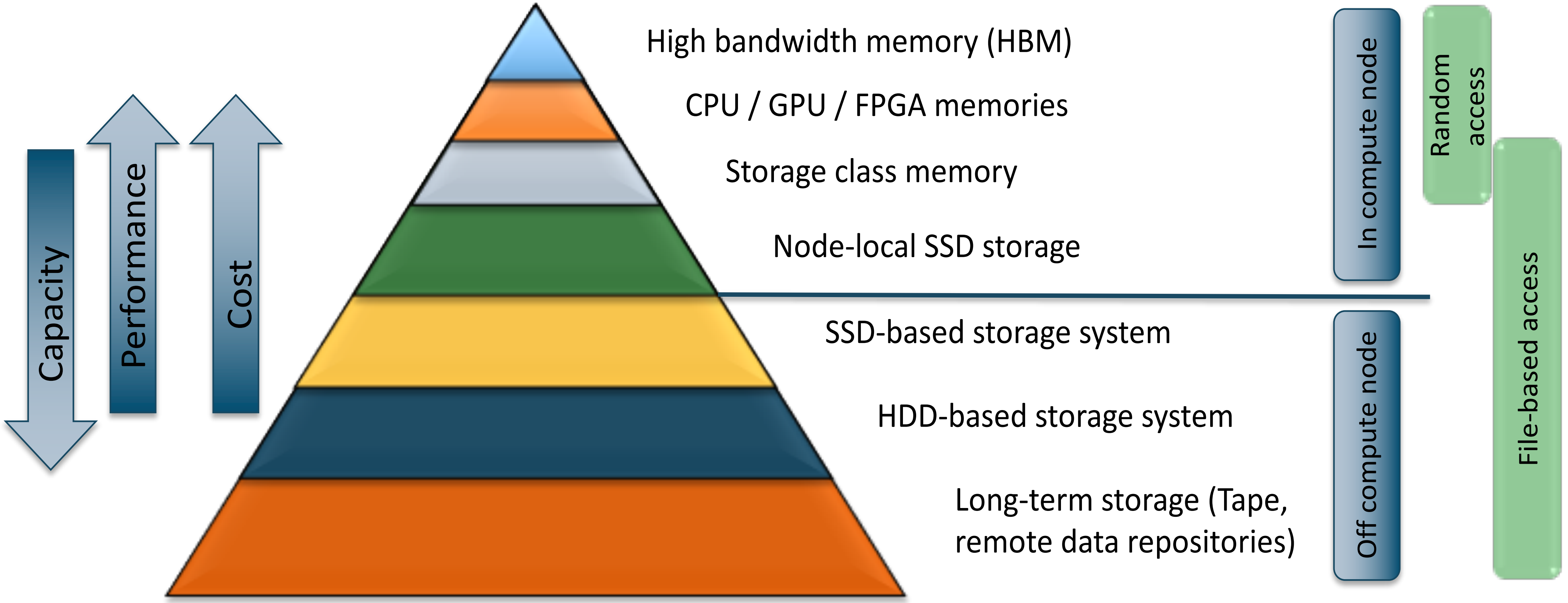
Outline of this talk (focusing on HPC systems)

- Basics, trends, and challenges
- Recent optimization strategies we developed
  - Asynchronous I/O
  - Data caching
- Proactive Data Containers (PDC) data management runtime system
- Future research directions

I/O software stack - Several layers with inter-dependencies



Architectural trends impacting I/O - deep memory and storage hierarchy



Trends in computing devices

- Heterogeneous processing devices
  - CPUs
  - GPUs
  - FPGAs
  - Special purpose accelerators
  - …
- Massive concurrency
- Locations of data generation and consumption
  - Traditional: In compute nodes
  - Trends: In network, in storage, and at the edge

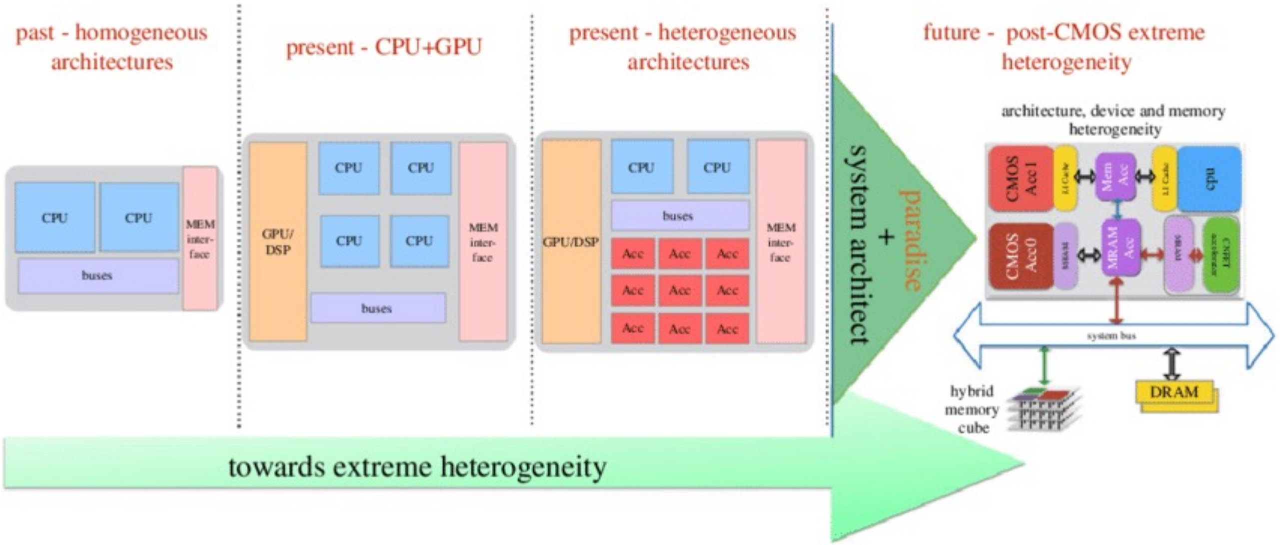
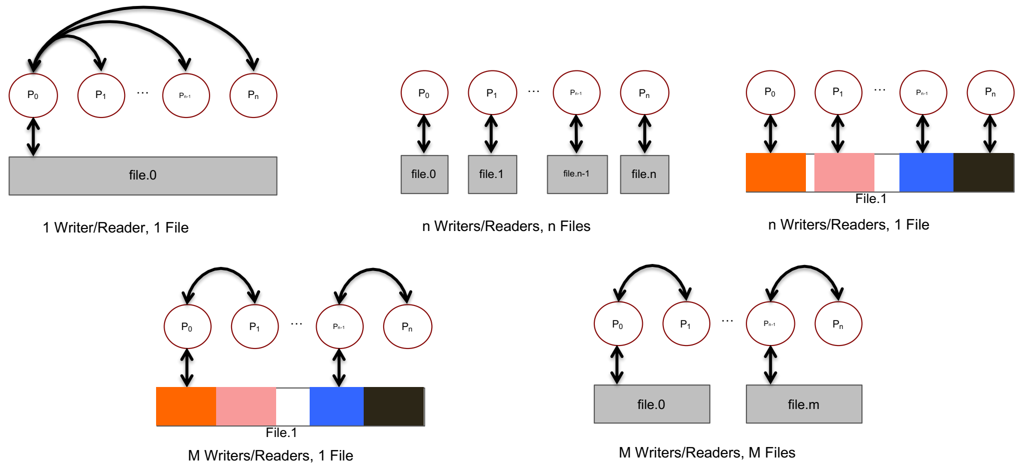


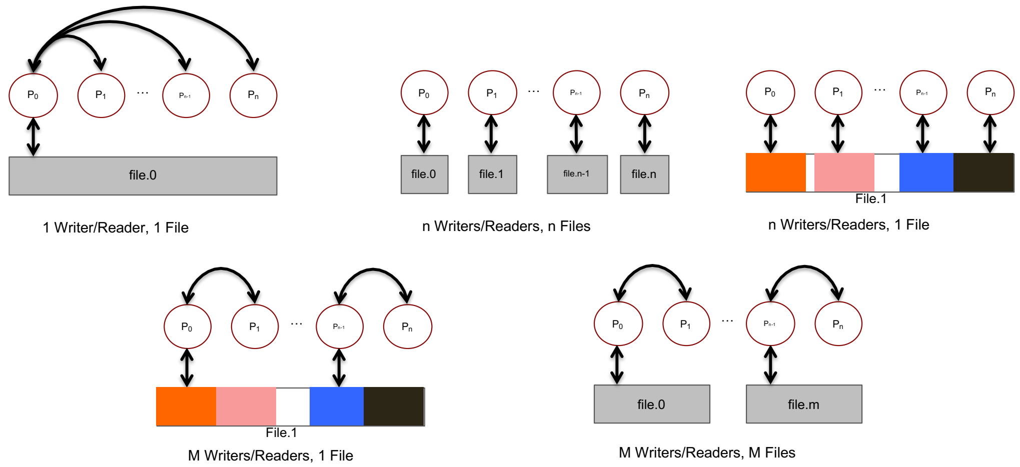
Image from D. Vasudeven, via J. Shalf, Extreme Heterogeneity workshop report

Application I/O Trends



- HPC simulation and analysis
  - Parallel applications storing or retrieving data
  - checkpoint, restart, and analysis - mostly defensive I/O
  - Partitioned regions of data in a large data object
  - Each process accesses (writes or reads) a single or multiple regions
  - Typically non-overlapping (there may be some overlapping at region boundaries)
  - Number of files: Single shared file (N-1), file-per-process (N-N), a few sub-files (M-N)
  - APIs: POSIX, MPI-IO, HDF5, etc.

Application I/O Trends



- HPC simulation and analysis
  - Parallel applications storing or retrieving data
  - checkpoint, restart, and analysis - mostly defensive I/O
  - Partitioned regions of data in a large data object
  - Each process accesses (writes or reads) a single or multiple regions
  - Typically non-overlapping (there may be some overlapping at region boundaries)
  - Number of files: Single shared file (N-1), file-per-process (N-N), a few sub-files (M-N)
  - APIs: POSIX, MPI-IO, HDF5, etc.
- ML / AI
  - Read-heavy
  - Random accesses to different parts of a file or multiple files (due to shuffling between epochs)
  - Small I/O requests to batches of data, metadata accesses
  - Large numbers of files - O (100,000)
  - APIs: Python, ML/AI frameworks (TensorFlow, PyTorch, Caffe, etc.)
- Experimental and observational facilities, Edge computing, etc.
  - Large numbers of small files, streaming data, sparse data, remote file accesses

What do users want?

Use case	Domain	Sim/EOD/analysis	Data size	I/O Requirements
FLASH	High-energy density physics	Simulation	~1PB	Data transformations, scalable I/O interfaces, correlation among

Easy interfaces to complex systems

CMB / Planck	Cosmology	Simulation,	10PB	Automatic data movement

Autonomous data movement and performance tuning

				transformations

Information capture, management, and search

TECA	Climate	Analysis	~10PB	Data organization and efficient data movement
HipMer	Genomics	EOD/Analysis	~100TB	Scalable I/O interfaces, efficient and automatic data movement

Users:

- Scientists
- App developers
- Supercomputing facilities
- System designers

Storage and I/O challenges

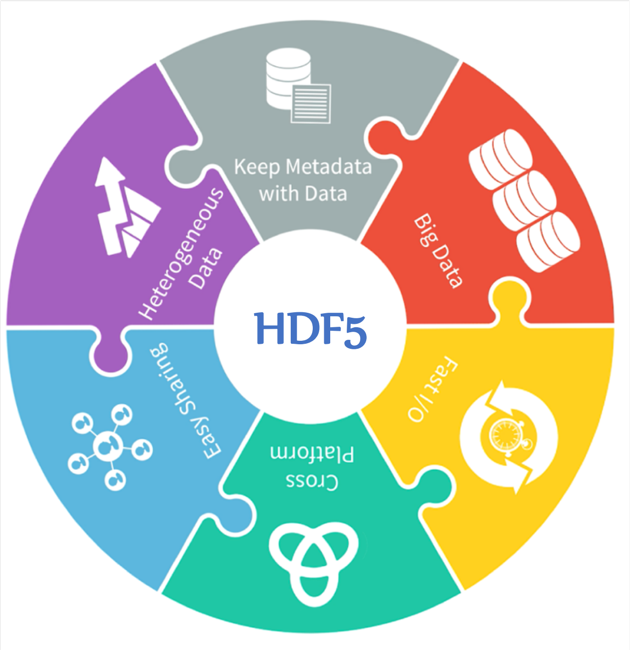
- Poor performance
  - POSIX semantics
  - Tuning options at different software layers
- Complexity with deep storage hierarchy
  - Using the performance and capacity storage layers efficiently and transparently
- Heterogeneity of storage devices
  - From memory-class to hard-disks to tape
- Variability of performance on HPC systems
  - Some part of the storage system is concurrently shared by multiple users
- New classes of applications
  - ML / AI applications are read-heavy with random access patterns
  - EOD and Edge computing applications
- New high-level programming models
  - Performance & Productivity

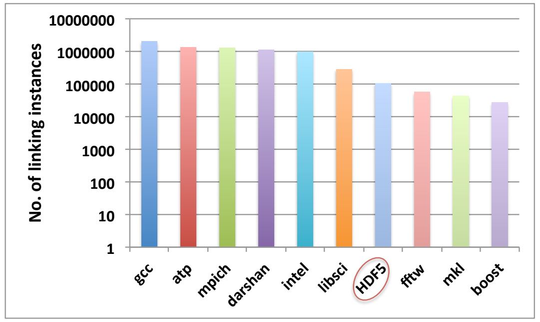
A few recent I/O optimization methods and systems

- ExaHDF5 / ExaIO
  - Asynchronous I/O
  - Using memory and storage hierarchy efficiently
- Proactive Data Containers (PDC)
  - Object-centric data management runtime system

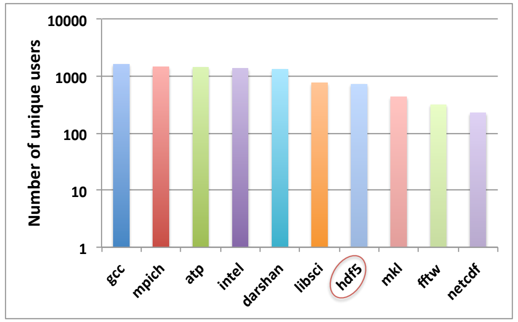
HDF5 self-describing file format and API for science apps

- HDF5 is a self-describing file format, API, and tools designed to store, access, analyze, share, and preserve diverse, complex data in continuously evolving heterogeneous computing and storage environments
  - Maintained by The HDF Group (THG)
- Heavily used on DOE supercomputing systems and diverse science applications across the world
- Many ECP applications have dependency on HDF5-based I/O
  - 17 critical
  - 11 important
  - 8 interested

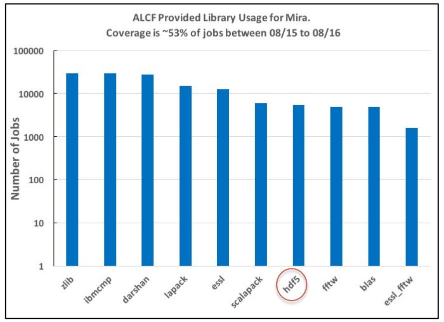




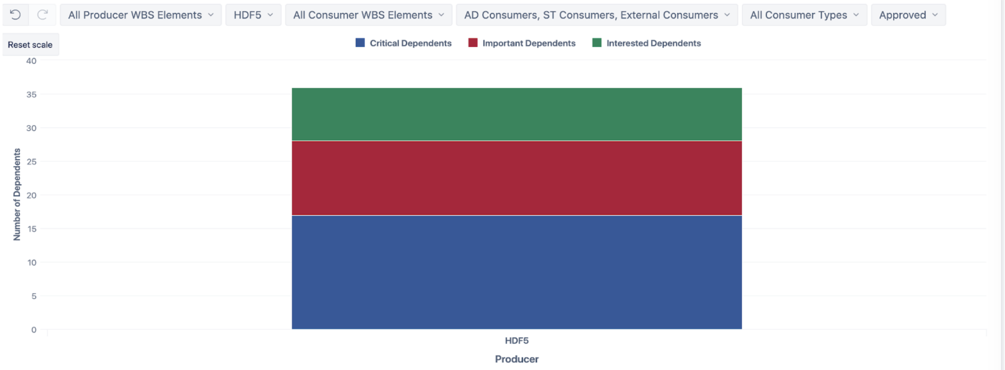
a. Number of linking instances on Edison (NERSC)



b. Number of unique users on Edison (NERSC)



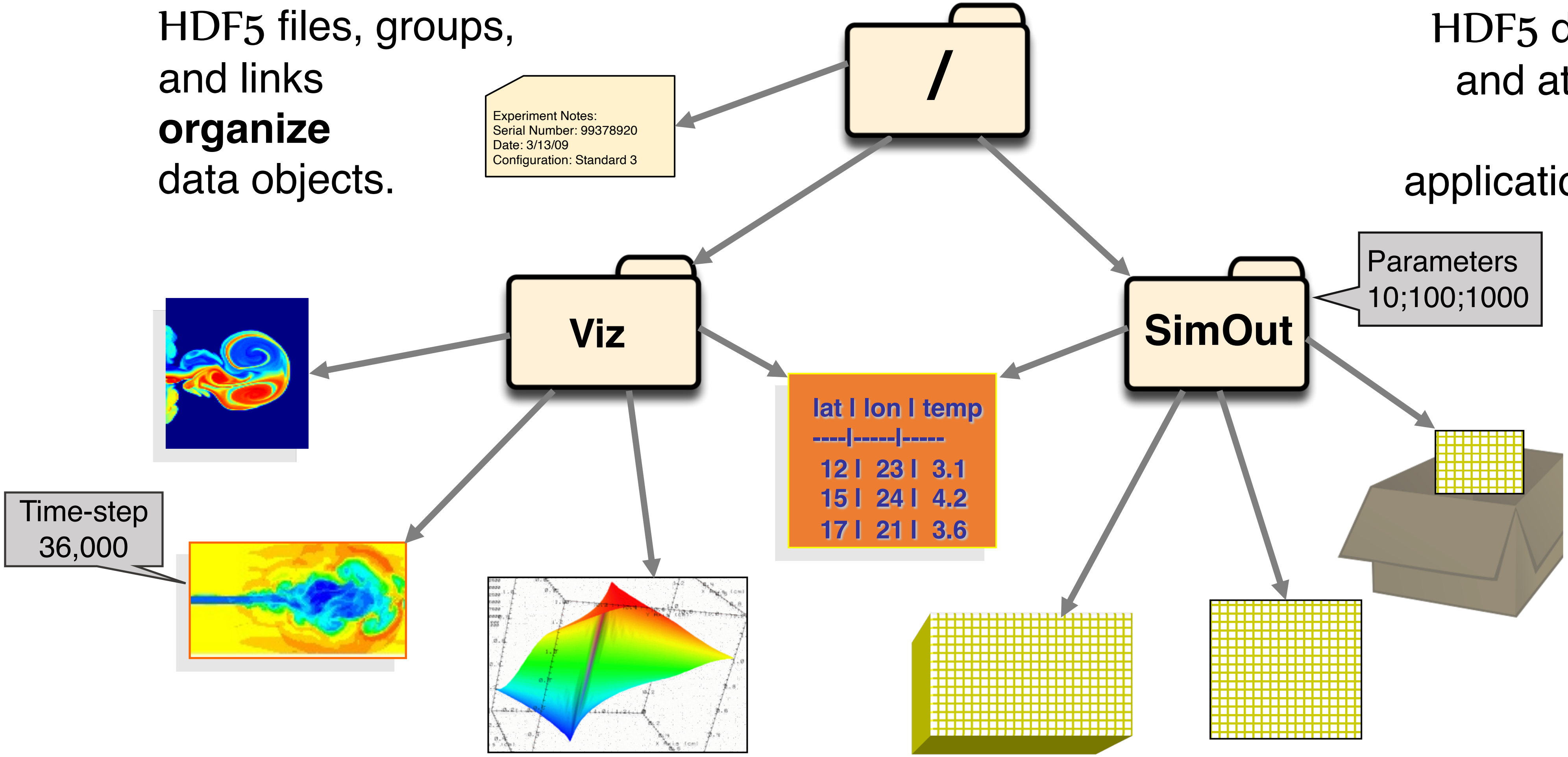
c. Number of linking instances on Mira (ALCF)



HDF5 Containers (Files)

HDF5 files, groups, and links **organize** data objects.

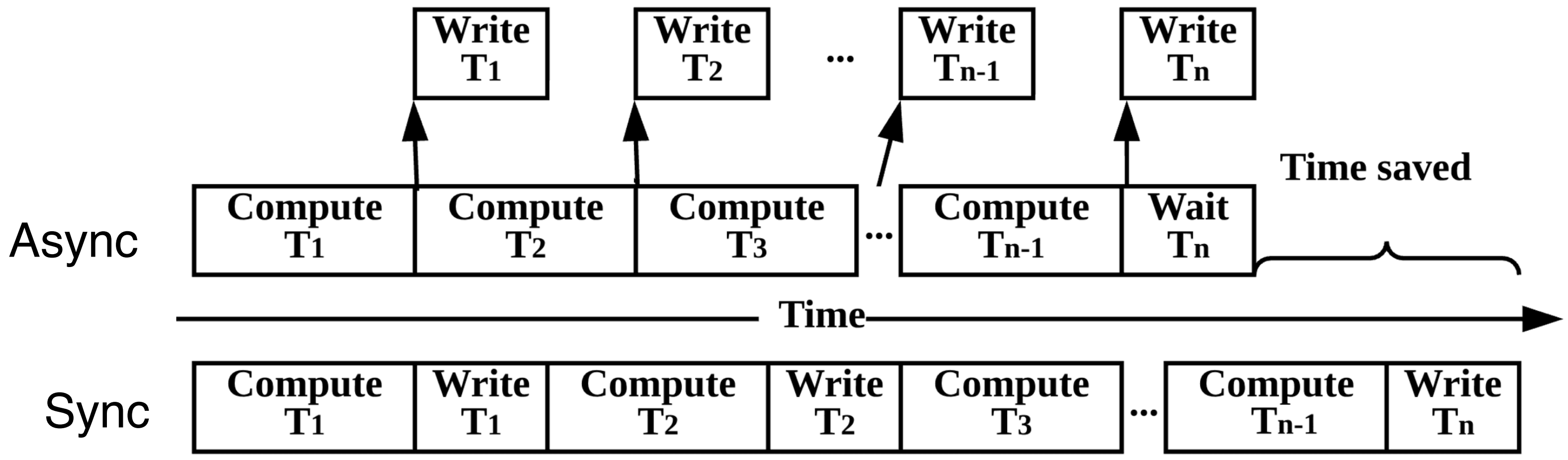
HDF5 datasets and attributes **store** application data.



Adapted from THG and Quincey Koziol

Async I/O VOL Connector

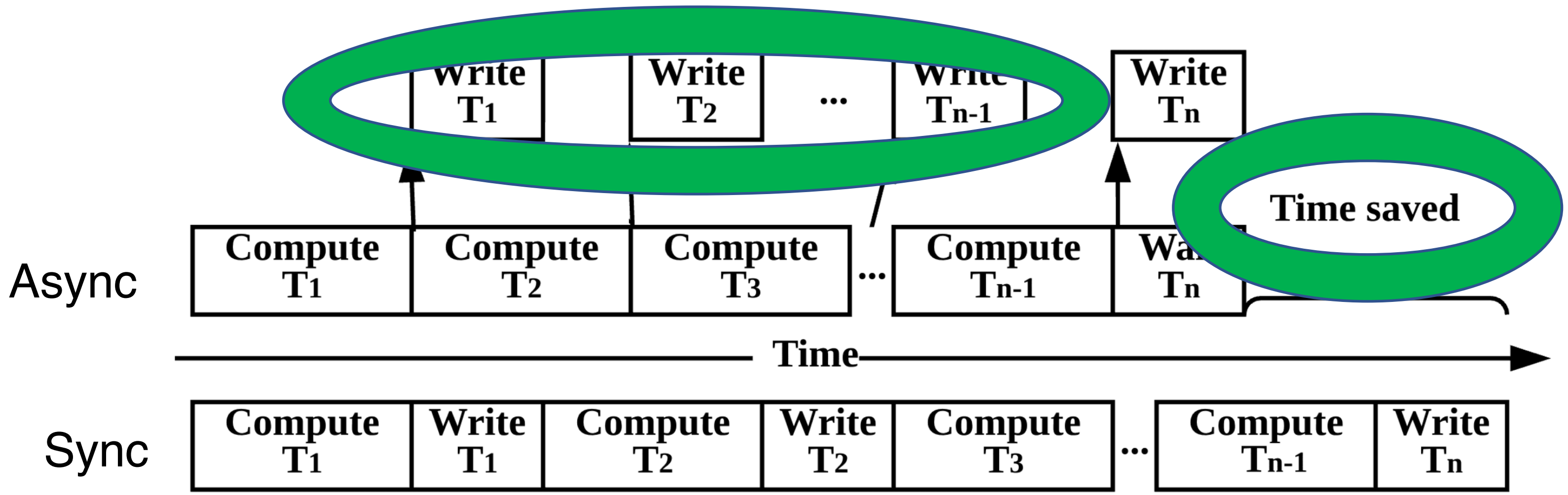
- **VOL Framework is an abstraction layer within HDF5 Library**
  - Redirects I/O operations into VOL "connector", immediately after an API routine is invoked
- Pass-through VOL connector
  - Can be stacked on any other connector, to provide asynchronous operations to it
- Uses an "event set" to manage async operations
  - Can extract more performance, e.g., enable async read and write:



Slide from Houjun Tang

Async I/O VOL Connector

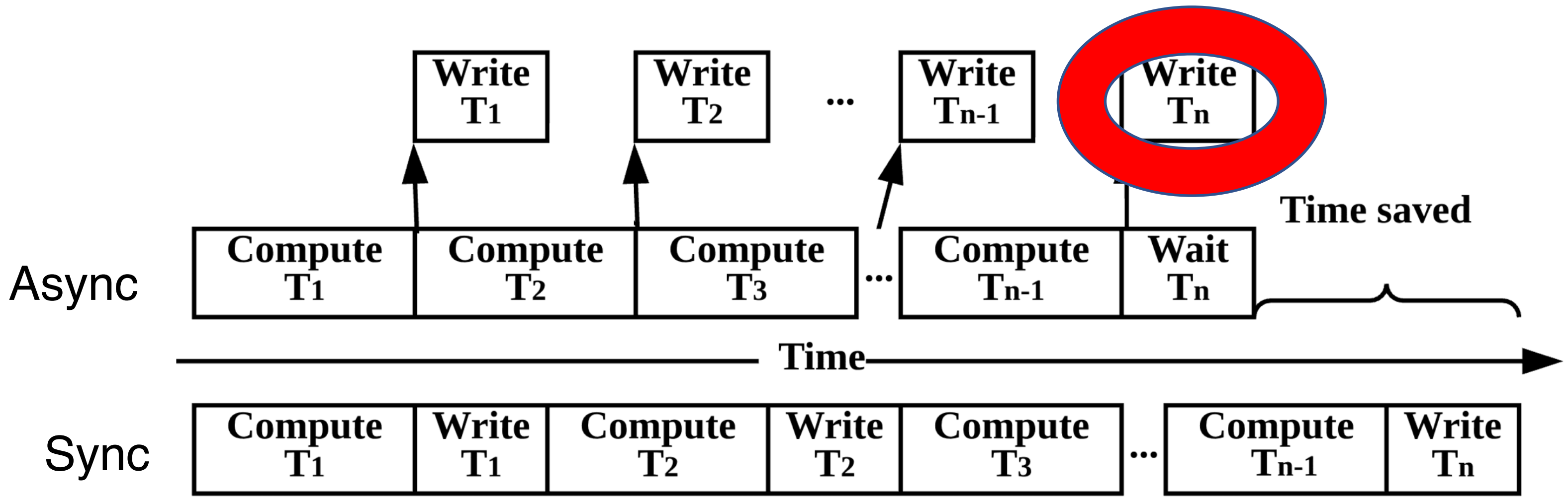
- Pass-through VOL connector
  - Can be stacked on any other connector, to provide asynchronous operations to it
- Uses an “event set” to manage async operations
  - Can extract more performance, e.g., enable async read and write:



Slide from Houjun Tang

Async I/O VOL Connector

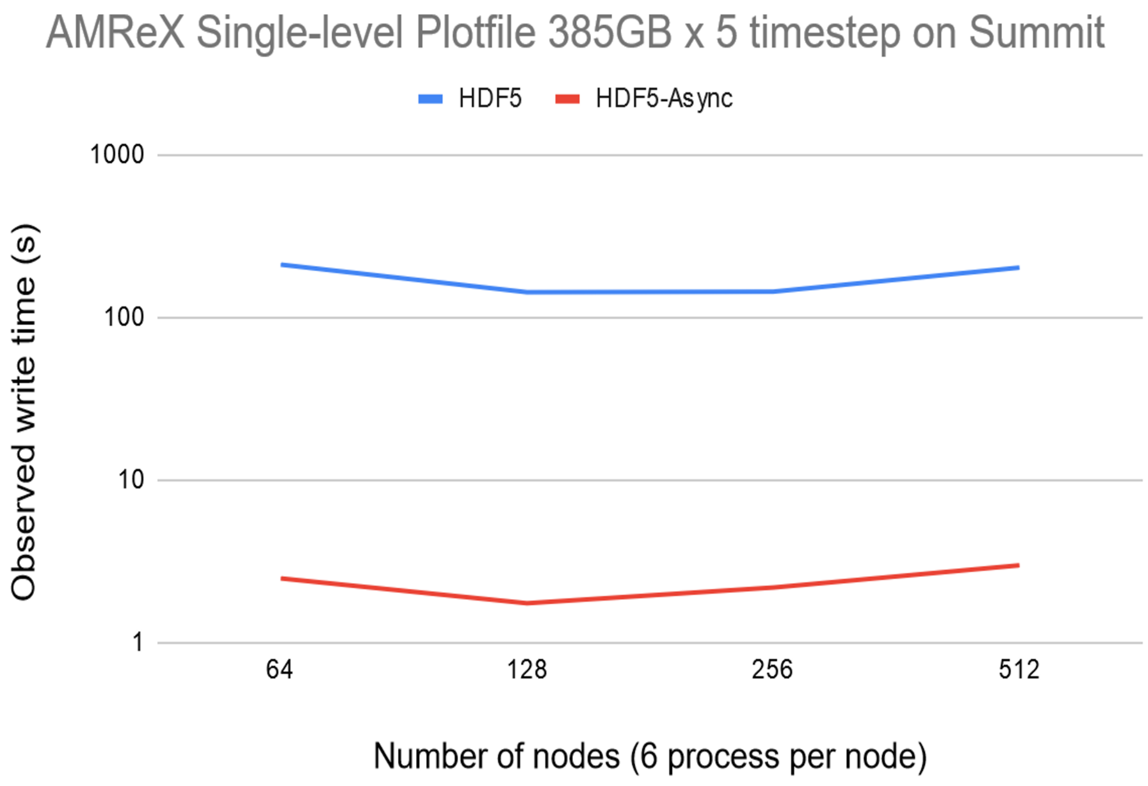
- Pass-through VOL connector
  - Can be stacked on any other connector, to provide asynchronous operations to it
- Uses an “event set” to manage async operations
  - Can extract more performance, e.g., enable async read and write:

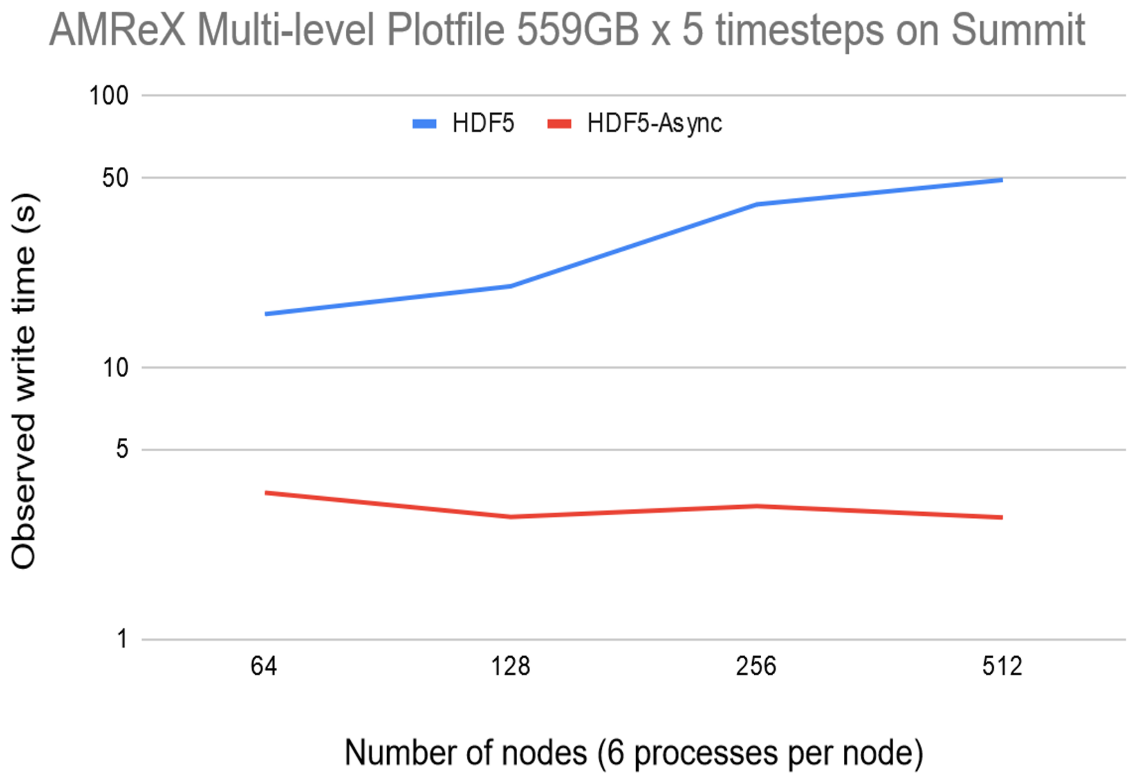


Slide from Houjun Tang

Async I/O – Low overhead and efficient hiding of I/O latency

AMReX: A software framework for massively parallel, block-structured adaptive mesh refinement (AMR) applications





Nyx: An adaptive mesh, cosmological hydrodynamics simulation code (left)

Castro: An adaptive mesh, astrophysical radiation/MHD/hydrodynamics simulation code

Adapted from Houjun Tang's slide

Async I/O VOL Connector

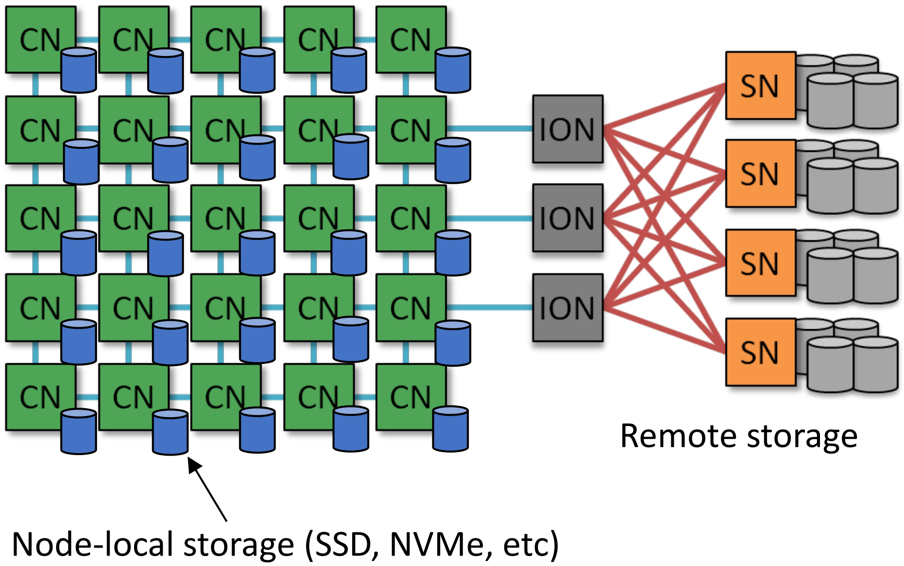
- Available now:
  - Source: https://github.com/hpc-io/vol-async
  - Docs: https://hdf5-vol-async.readthedocs.io/en/latest
- Future work:
  - Merge compatible VOL operations
    - If two async dataset write operations are putting data into same dataset, can merge into only one call to underlying VOL connector
    - Turn multiple ‘normal’ group create operations into a single ‘multi’ group create operation
  - Use multiple background threads
    - Needs HDF5 library thread-safety work, to drop global mutex
  - Switch to TaskWorks thread engine
    - A portable, high-level, task engine designed for HPC workloads
    - Task dependency management, background thread execution.

Houjun Tang, Quincey Koziol, John Ravi, and Suren Byna, "Transparent Asynchronous Parallel I/O using Background Threads", IEEE TPDS - Special Section on Innovative R&D toward the Exascale Era, 2021

Slide from Houjun Tang

Cache VOL Connector - Integrating node-local storage into parallel I/O

Typical HPC storage hierarchy



Theta @ ALCF: Lustre + SSD (128 GB / node),
ThetaGPU (DGX-3) @ ALCF: NVMe (15.4 TB / node)
Summit @ OLCF: GPFS + NVMe (1.6 TB / node)

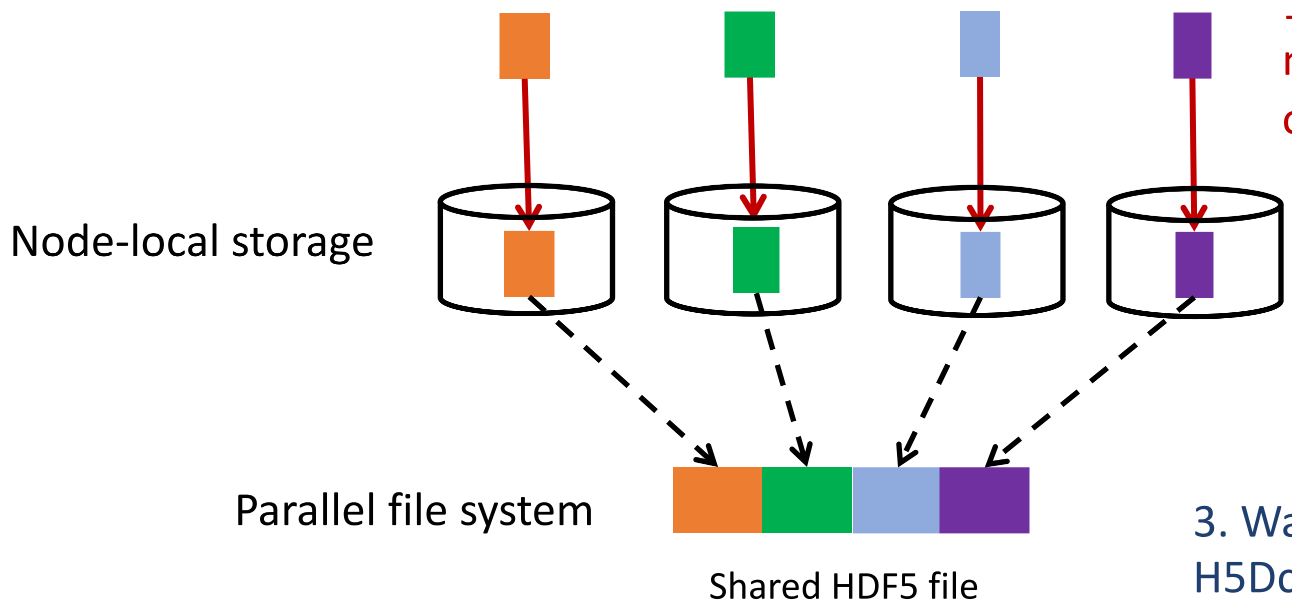
Cache VOL

- Using node-local storage for caching / staging data for fast and scalable I/O.
- Data migration to and from the remote storage is performed in the background.
- Managing data movement in multi-tiered memory / storage through stacking multiple VOL connectors (*async* -> *cache* -> *async*)
- All complexity is hidden from the users

Repo: https://github.com/hpc-io/vol-cache

Slide from Huihuo Zhang

Parallel Write (H5Dwrite) with cache VOL



1. Data is synchronously copied from the memory buffer to memory mapped files on the node-local storage using POSIX I/O.

2. Move data from memory mapped file to the parallel file system asynchronously by calling the dataset write function from the *Async VOL* stacked below the Cache VOL

3. Wait for all the tasks to finish in H5Dclose() / H5Fclose()

w/o caching	Compute	I/O (RAM→PFS)	Compute
w/ caching	Compute	RAM->NLS	Compute
			I/O: NLS->PFS

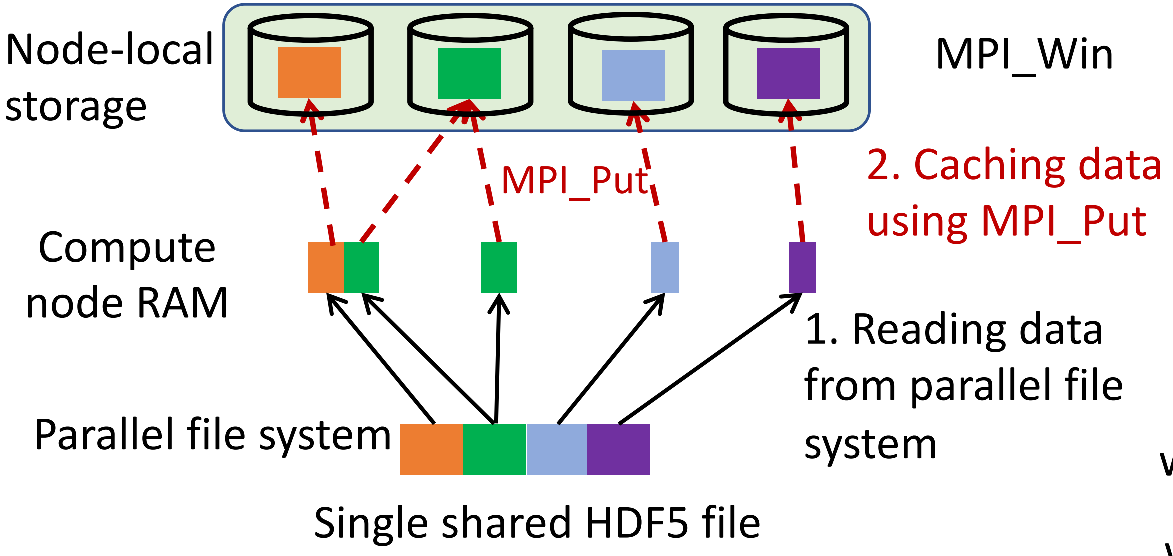
Partial overlap of compute with I/O

Details are hidden from the application developers.

Slide from Huihuo Zhang

Parallel Read (H5Dread) w/ cache VOL

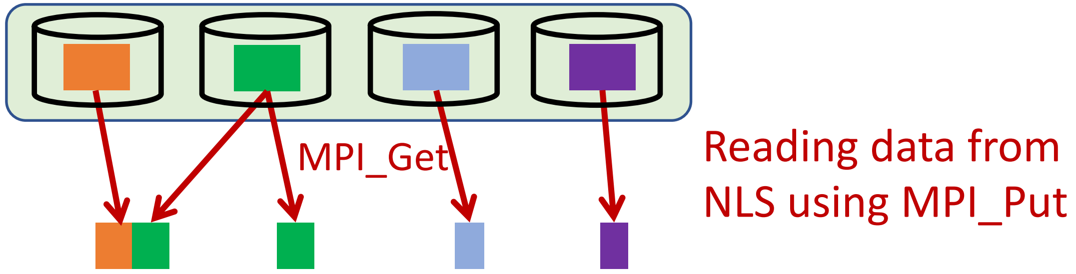
Create memory mapped files and attached them to a MPI_Win for one-sided remote access

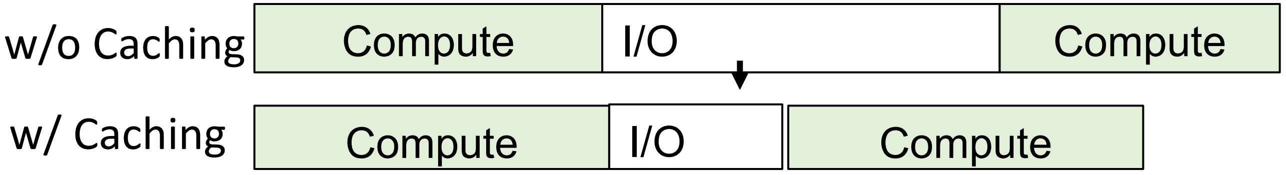


First time reading the data

One-sided communication for accessing remote node storage.

- Each process exposes a part of its memory to other processes (MPI Window)
- Other processes can directly read from or write to this memory, without requiring that the remote process synchronize (MPI_Put, MPI_Get)

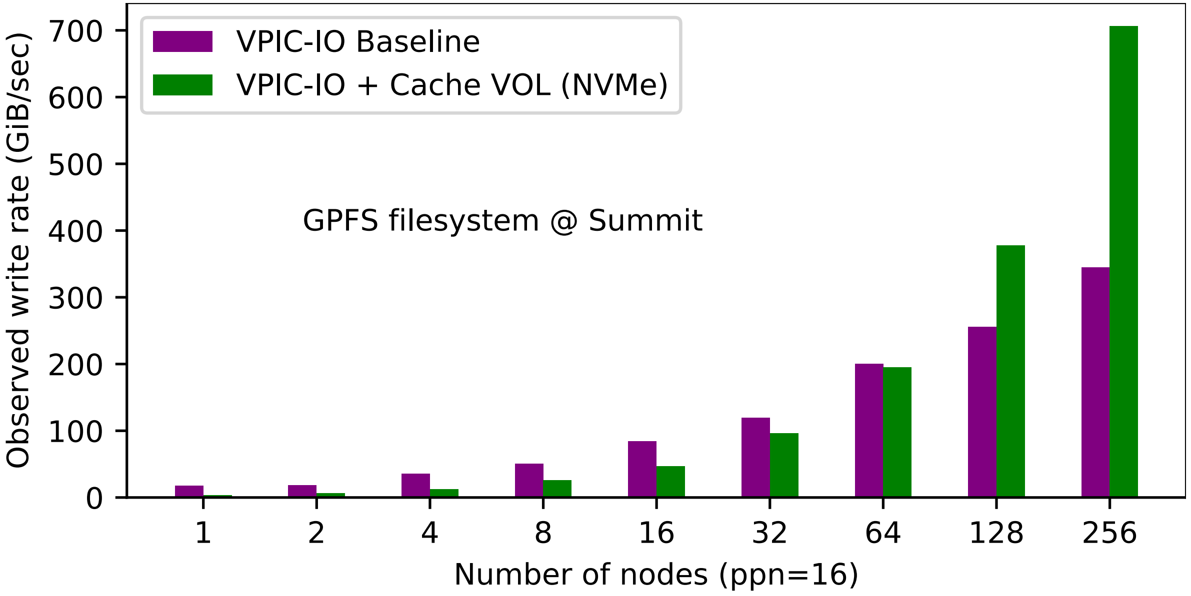


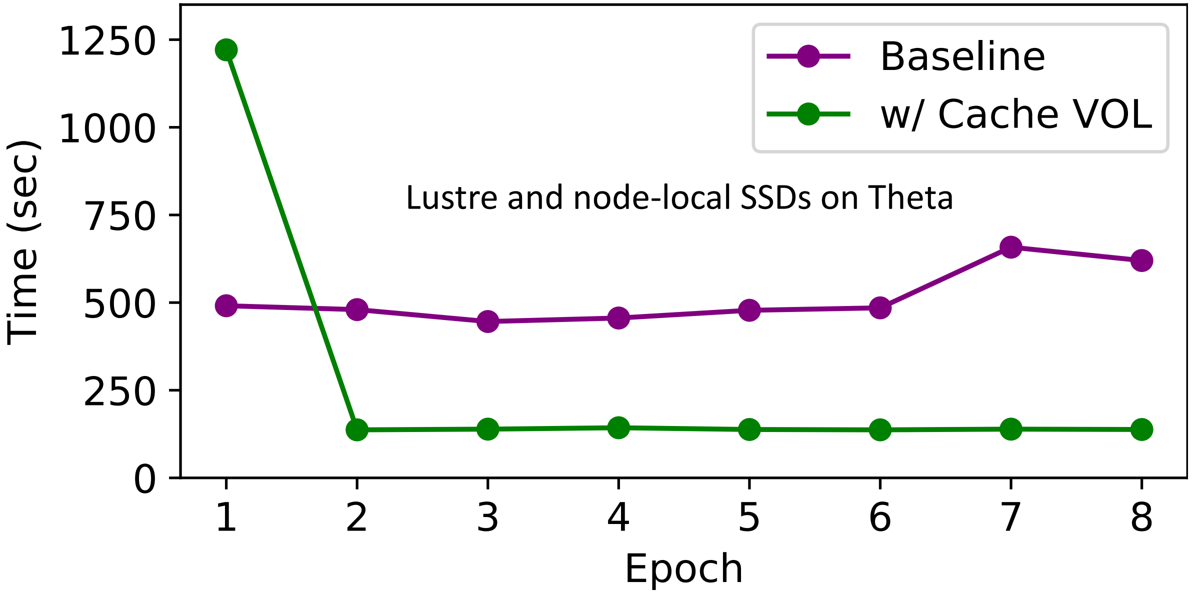


Reading the data directly from node-local storage

Slide from Huihuo Zhang

HDF5 Cache VOL uses node-local SSDs as cache → 2X to 3X faster





VPIC-IO is a kernel derived from a plasma physics simulation of solar weather interacting with the earth's magnetosphere. The simulation writes particle data to HDF5 file, where each variable is mapped to a HDF5 dataset

CosmoFlow is a 3D convolutional neural network model for learning the universe at scale. The model is implemented in TensorFlow with Horovod for data parallel training at scale.

Huihuo Zheng, Venkat Vishwanath, Quincey Koziol, Houjun Tang, John Ravi, John Mainzer, Suren Byna, "HDF5 Cache VOL: Efficient and Scalable Parallel I/O through Caching Data on Node-local Storage", Accepted to CCGrid 2022

Slide from Huihuo Zhang

What do users want?

Use case	Domain	Sim/EOD/analysis	Data size	I/O Requirements
FLASH	High-energy density physics	Simulation	~1PB	Data transformations, scalable I/O interfaces, correlation among
CMB / Planck	Cosmology	Simulation,	10PB	Automatic data movement … transformations
TECA	Climate	Analysis	~10PB	Data organization and efficient data movement
HipMer	Genomics	EOD/Analysis	~100TB	Scalable I/O interfaces, efficient and automatic data movement

Easy interfaces to complex systems

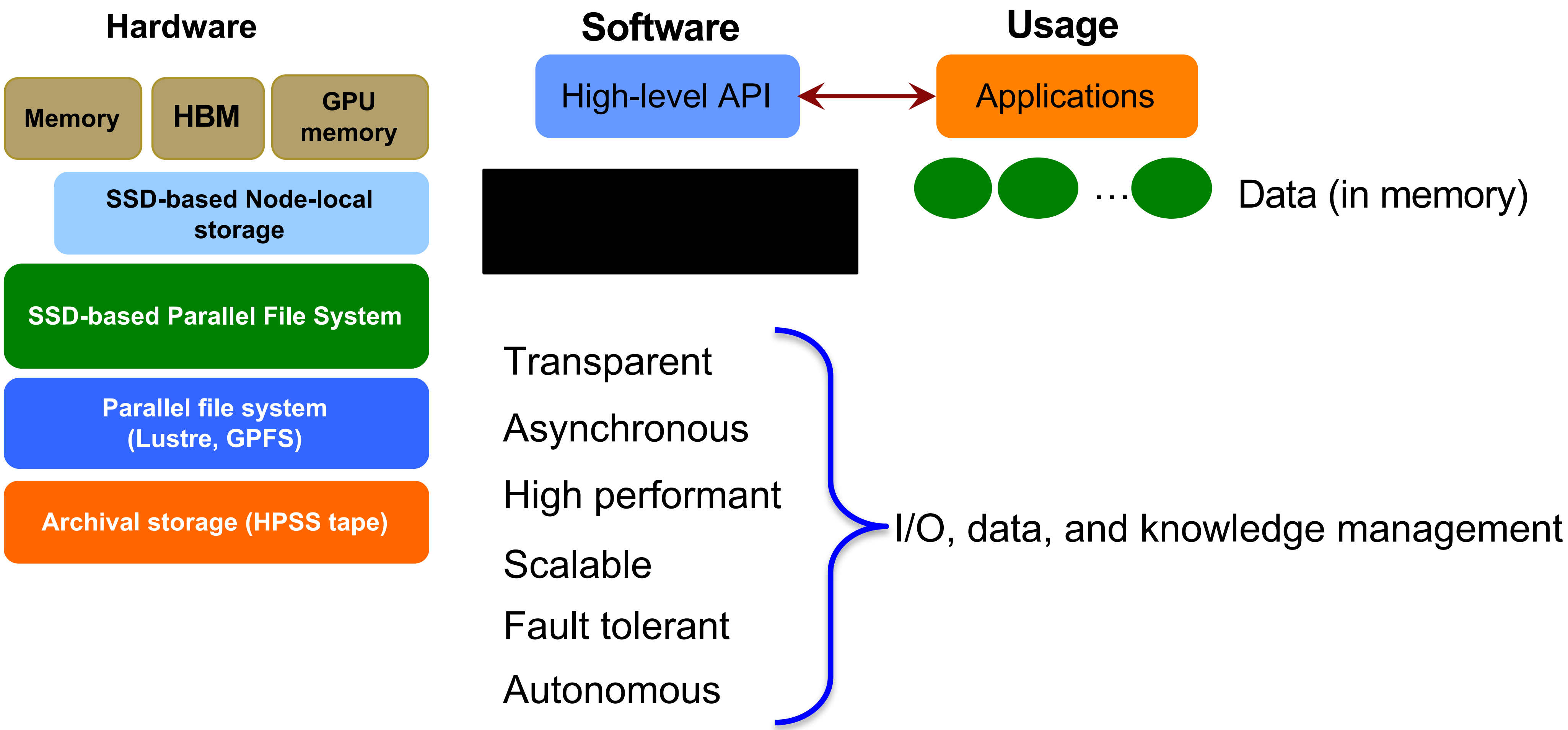
Autonomous data movement and performance tuning

Information capture, management, and search

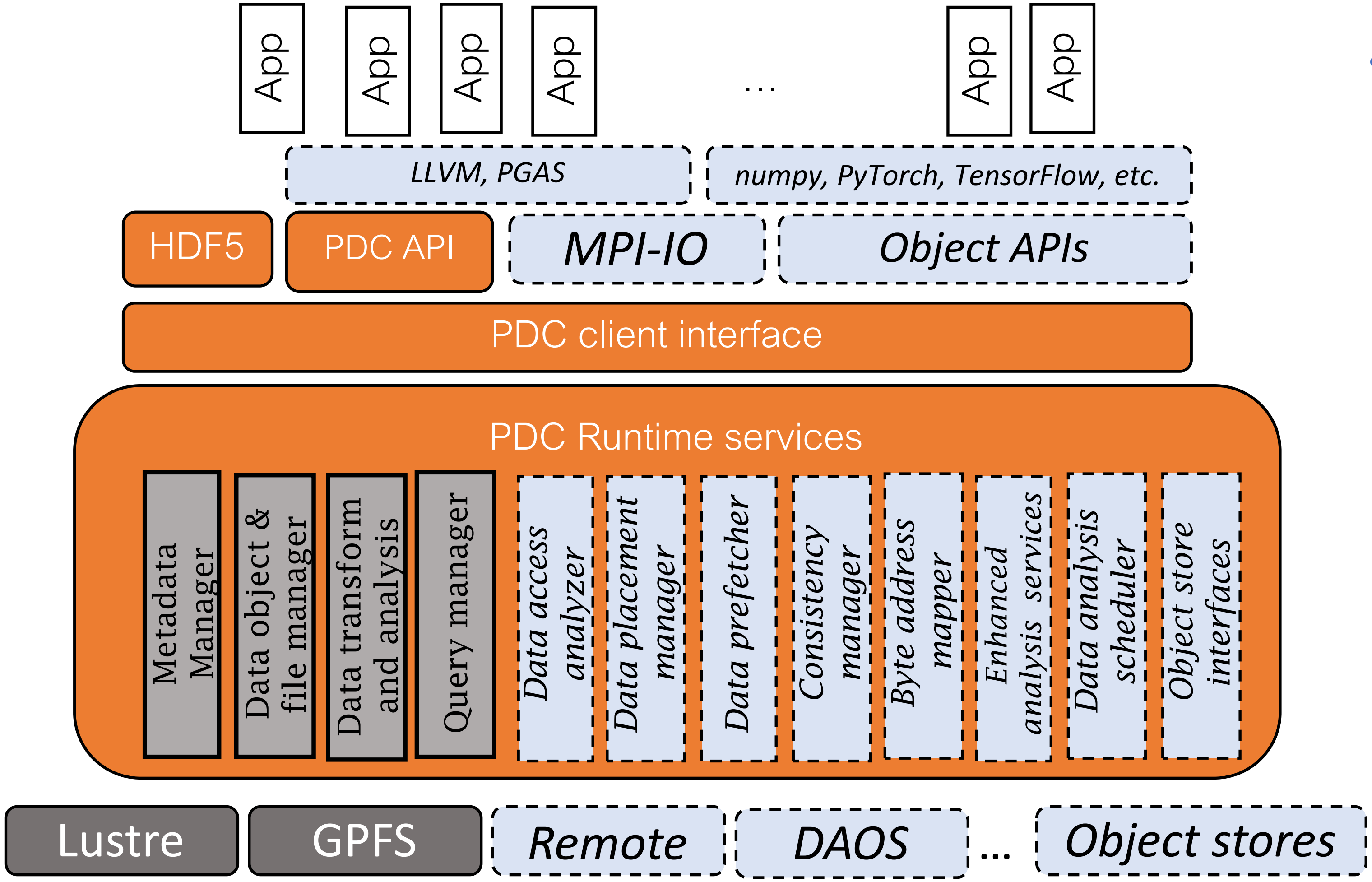
Users:

- Scientists
- App developers
- Supercomputing facilities
- System designers

Object-centric data management



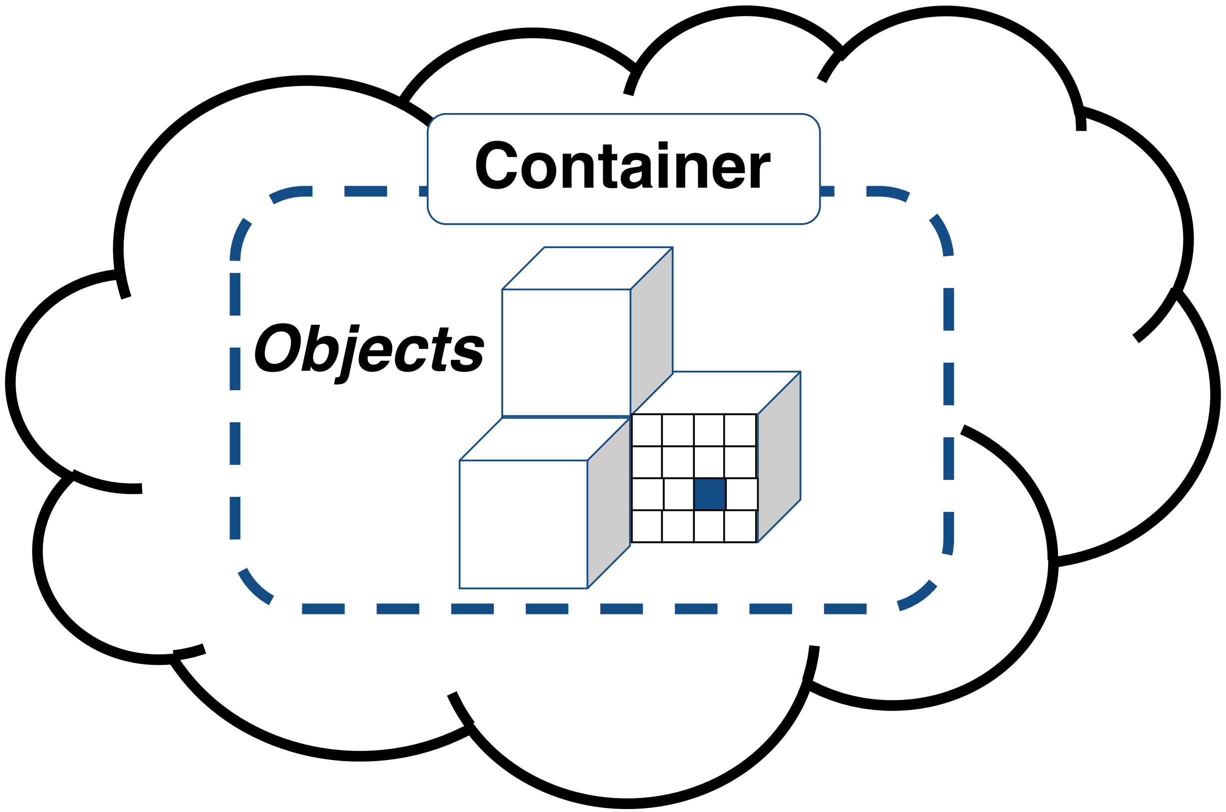
Proactive Data Containers (PDC) - An object-centric data movement runtime system



- Advantages with PDC
  - Application-level object abstractions - Freedom from file management
  - Transparent utilization of storage hierarchy and data movement
  - Superior and scalable performance
  - Live system for data management services
    - Metadata management, analysis, indexing and querying services, consistency, data placement, etc.

Object-centric PDC Abstractions

- Container – special case of object, metadata only
  - For grouping objects
- Object - metadata and data payload
  - Multi-dimensional arrays
- Regions
  - A logical region in a multi-dimensional array
  - expressed with offsets, sizes, and strides
  - multi-dimensional

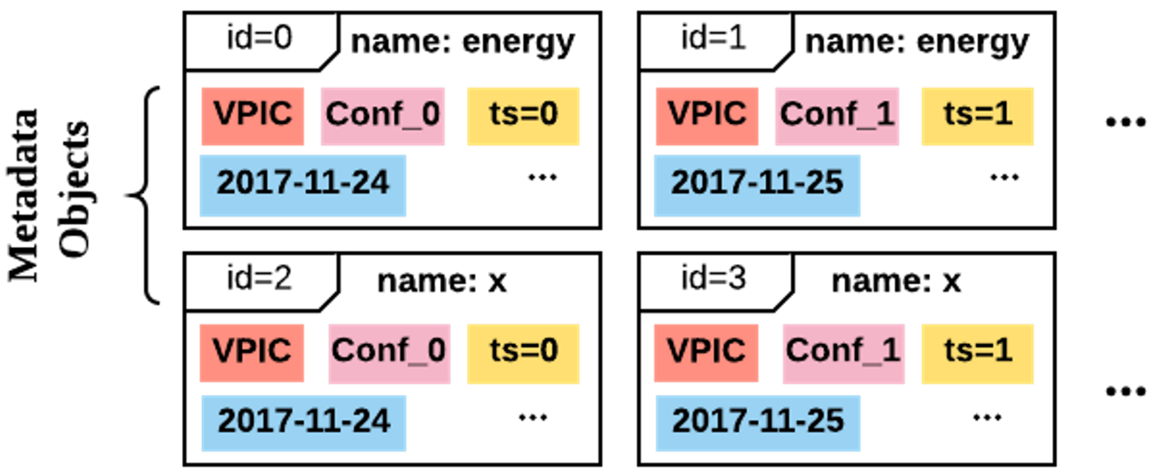


PDC metadata management

- Metadata is critical for finding objects previously written
- Predefined tags must be filled - either by users (using API) or by extracting from system
- User-defined tags improve findability of objects - KV pairs
- Unique IDs for metadata objects
- Querying for objects using tags
    - Exact match
    - Partial match

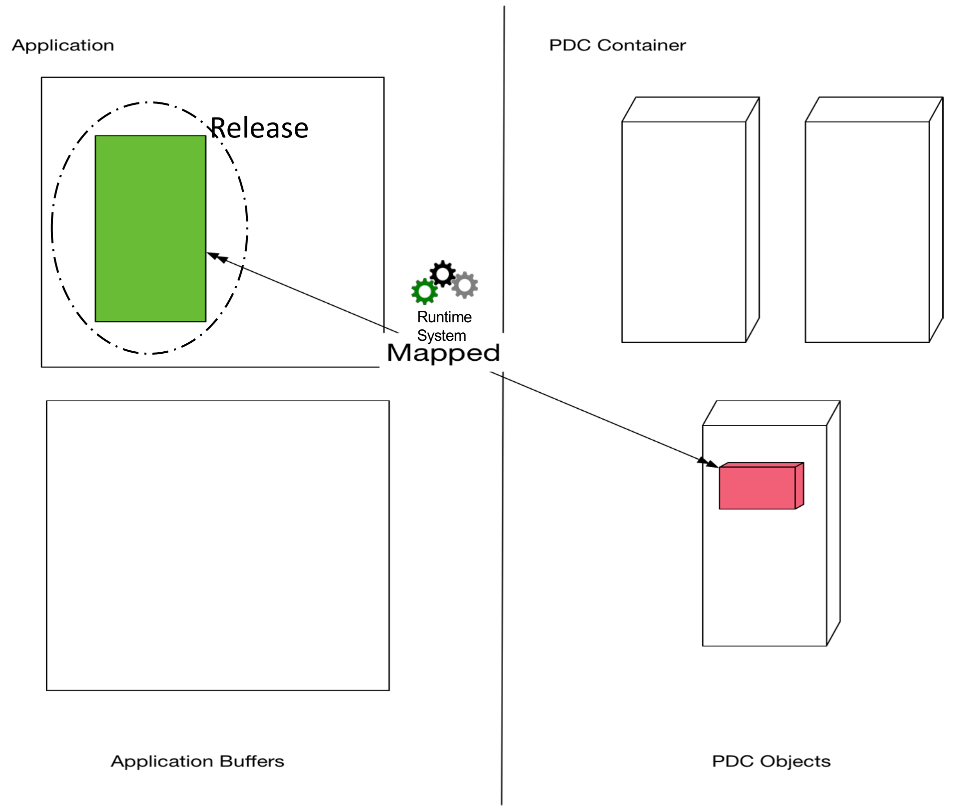
Pre-defined Tag	User-defined Tag
• **Object ID** • Data Location • System Info • **ID Attributes** - Object Name -Ownership - Application name -Timestep	• (Tag Name0, Value0) • (Tag Name1, Value1) • (Tag Name2, Value2) • ...

Flat Namespace



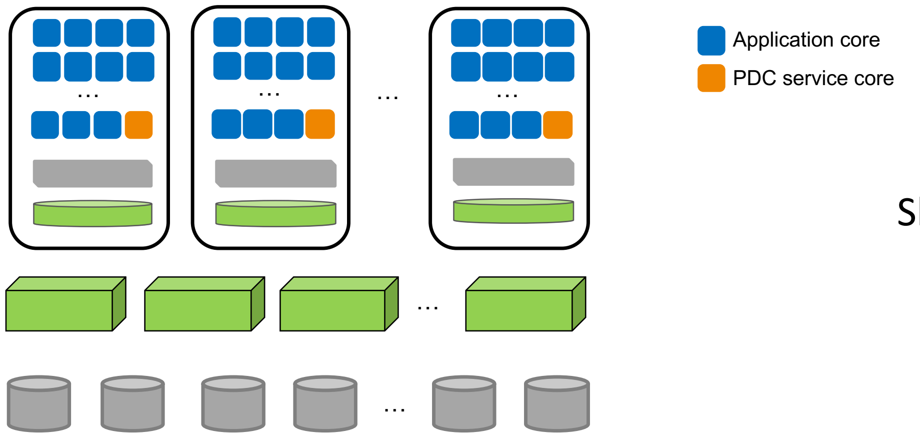
Object-centric PDC API

- No explicit data movement
- Container
  - create container
  - delete container
  - add / delete objects
- Objects
  - create object
  - add metadata
  - create regions
  - map objects / regions from source to destination
    - Source and destinations can be memory or PDC spaces
  - lock when updating an object
  - release informs PDC runtime for implicit data movement
  - find object (followed by “map” for reading)
  - Explicit put and get object functions are also available

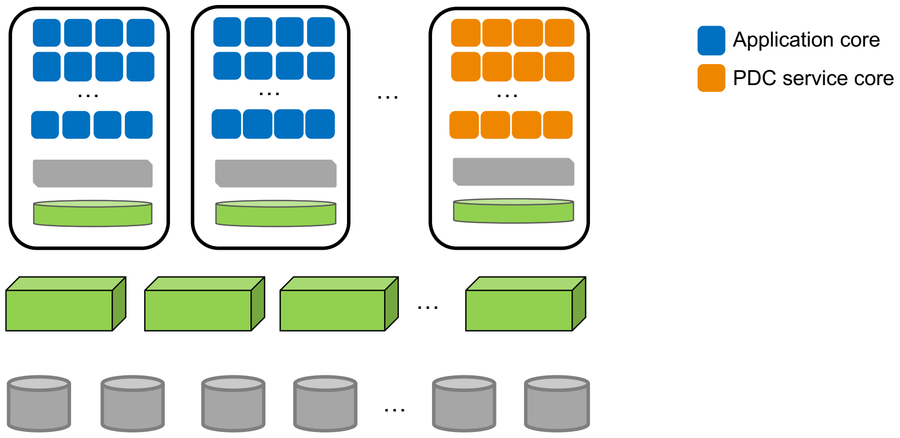


Transparent data movement in storage hierarchy

- Usage of compute resources for I/O
  - Shared mode – Compute nodes are shared between applications and I/O services
  - Dedicated mode – I/O services on separate nodes
- Users start PDC server and application links with the PDC client library
  - Set environment variables for informing PDC about the memory and storage locations
- Transparent data movement by PDC servers
  - Apps map data buffers to objects and PDC servers place and manage data
  - Apps query for data objects using attributes



Shared mode

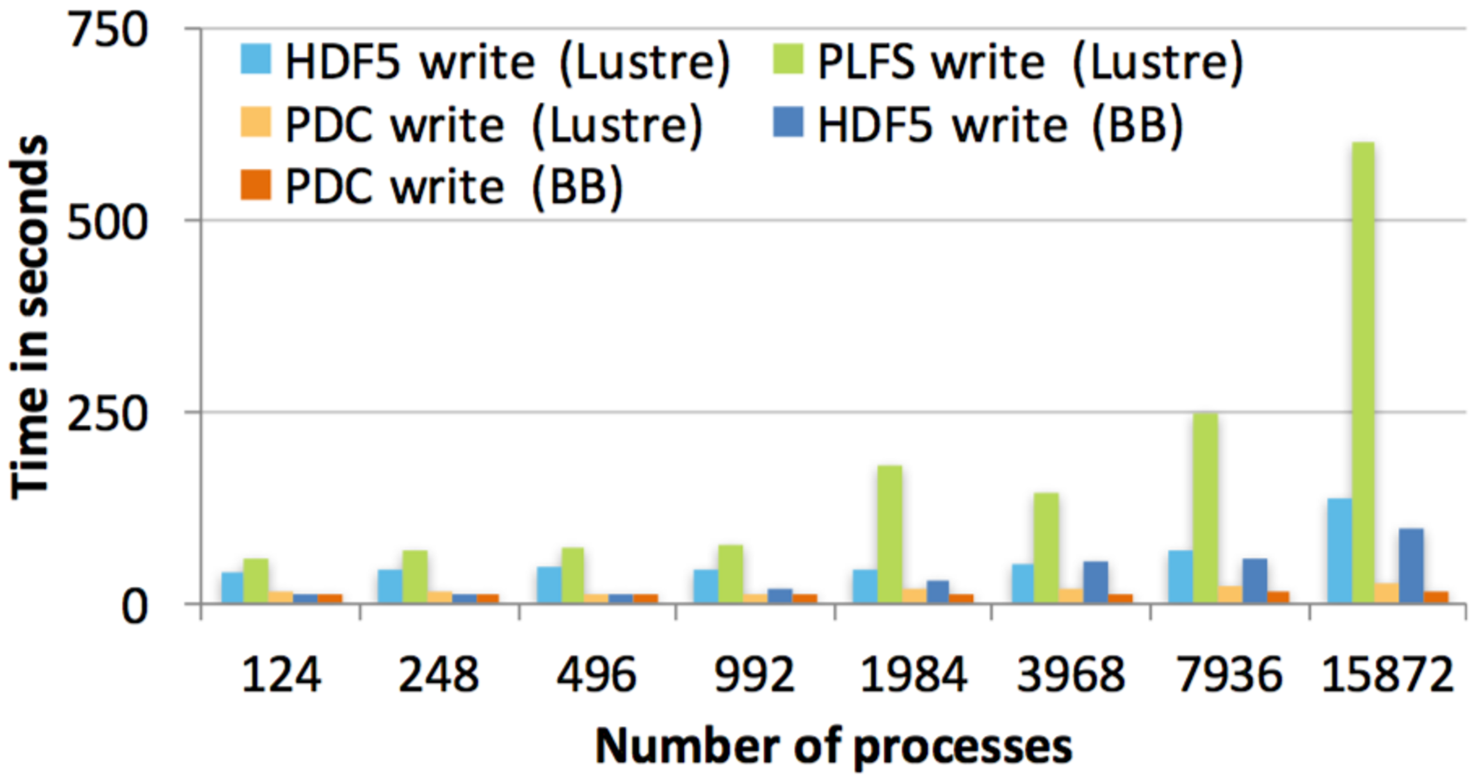


Dedicated mode

Experimental setup

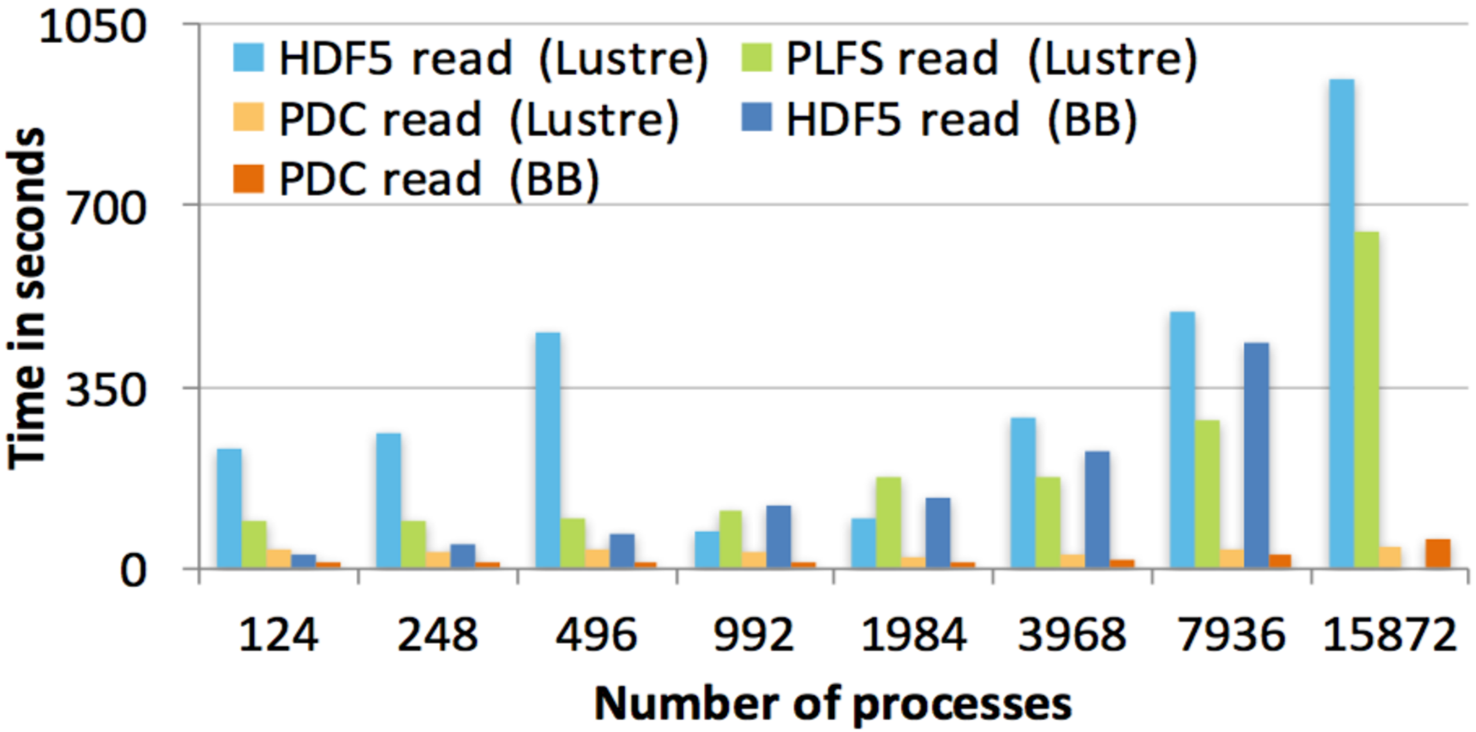
- Cori at NERSC
  - 1600 “Haswell” compute nodes
  - 128 GB DRAM/node
  - 32 cores/node
- SSD-based “Burst Buffer”
- HDD-based shared file system Lustre
  - *Shared* mode: One PDC server on each node, the remaining 31cores for user application execution.
  - *Dedicated* mode: PDC servers and user’s application are on separate nodes
- Mercury Remote Procedure Calls (RPCs), as the communication mechanism between client and server and between servers - TCP and Cray GNI
- Benchmarks
  - VPIC-IO: I/O of a large-scale plasma physics simulation code
  - BD-CATS-IO: I/O kernel of a big data clustering analysis code

VPIC-IO (Weak Scaling) Multi-timestep Write



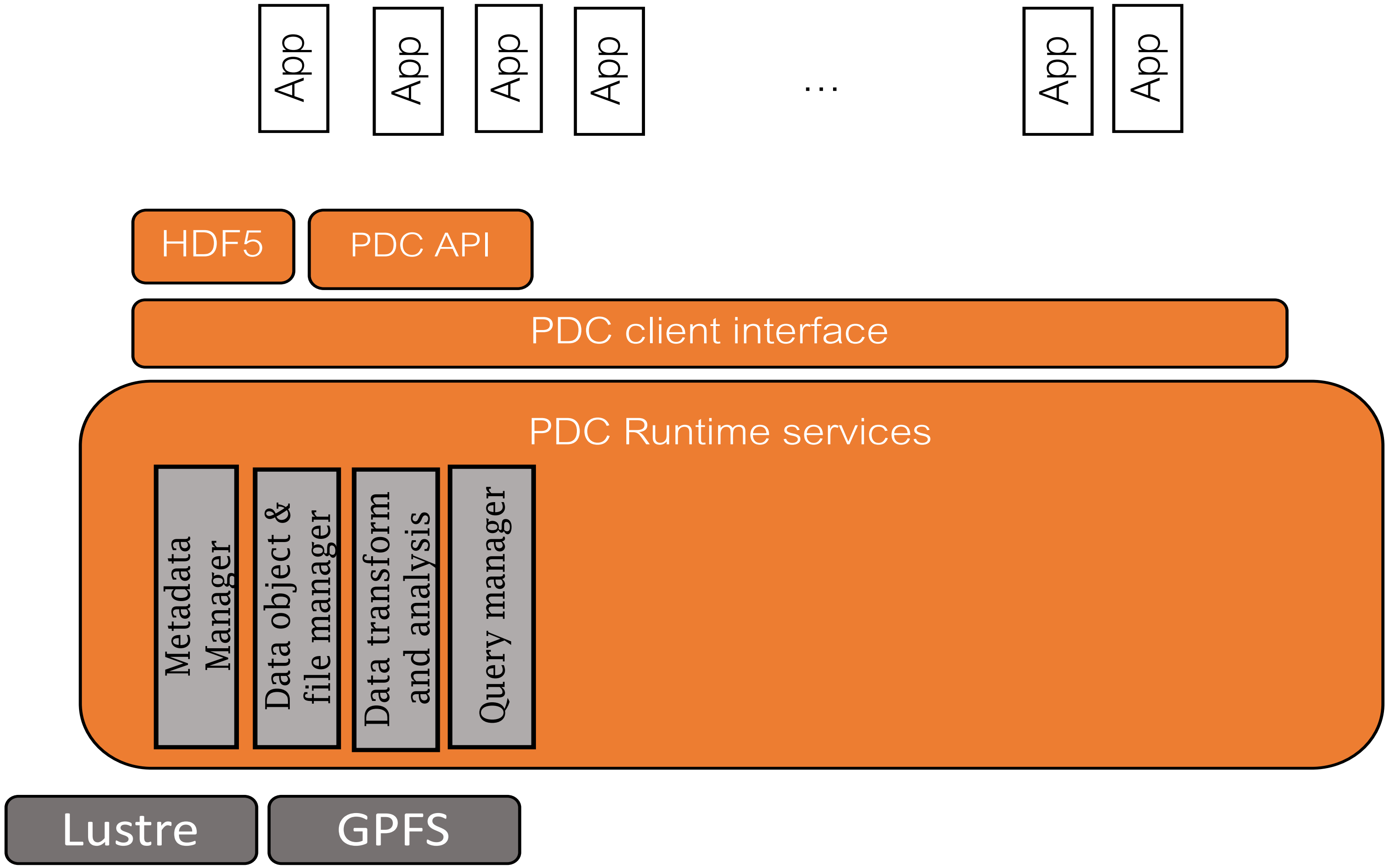
Total time to write 5 timesteps from the VPIC-IO kernel to Lustre and Burst Buffer on Cori. PDC is up to **5x** faster than HDF5 and **23x** over PLFS.

BD-CATS-IO (Weak scaling) Multi-timestep Read



Total time for reading data of 5 timesteps from the BD-CATS-IO kernel from the Lustre and from the burst buffer. PDC is up to **11X** faster than PLFS and HDF5.

Current status of PDC



- Current achievements in a next generation intelligent data management framework
  - Transparent
  - Asynchronous
  - High performance and scalable
  - Partial autonomy in utilizing storage hierarchy

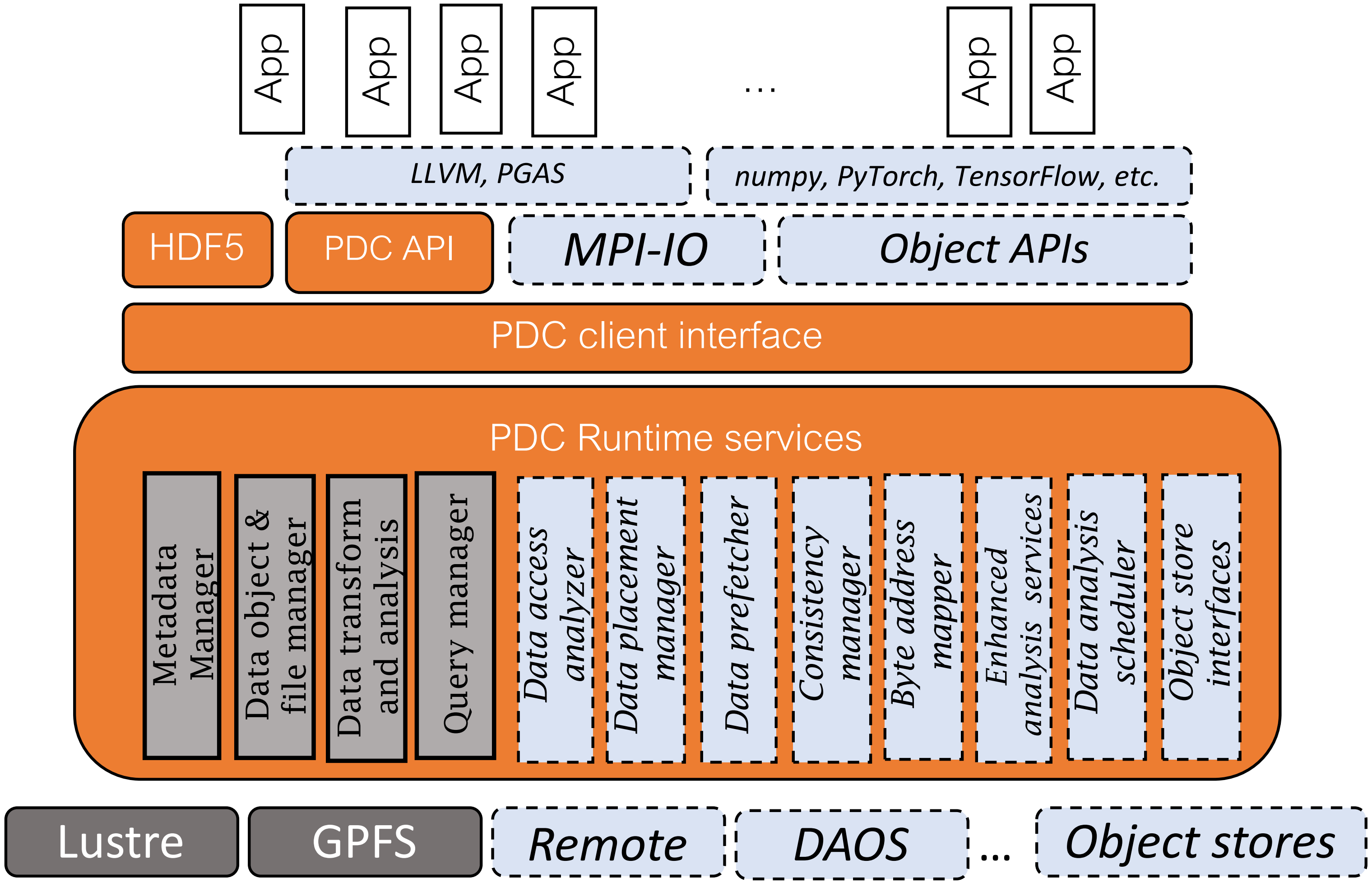
Source code:

https://github.com/hpc-io/pdc

Publications

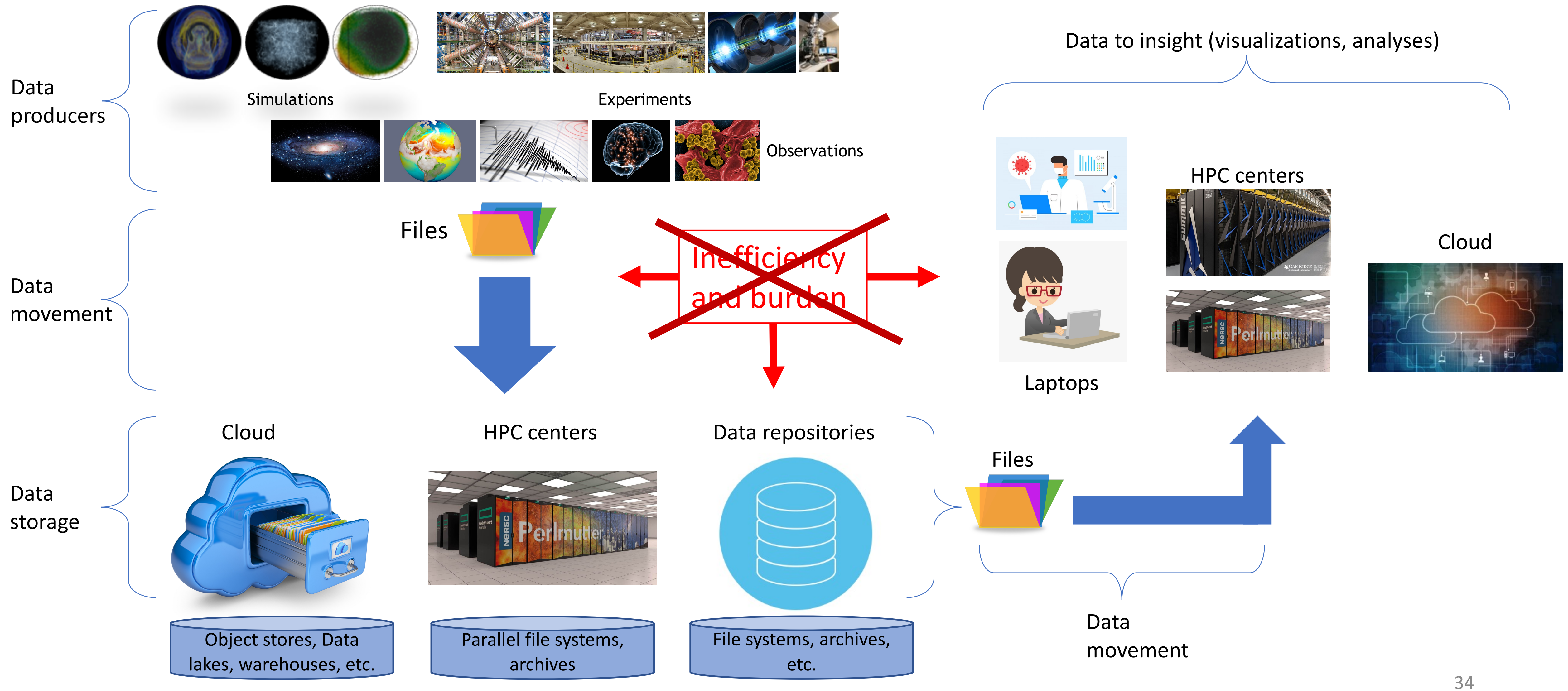
https://sdm.lbl.gov/pdc/

Proactive Data Containers (PDC) - Ongoing and future work



- Interface with Python and object (put and get) interfaces
- Extend to KV stores abstractions
- Data reorganization and placement based on access history
- Dynamic consistency
- Byte address mapping with PDC storage objects - working with John Shalf et al.
- Analysis on storage and network computing resources
- Federated data objects - remote access across multiple file systems at a site, multiple HPC sites, and cloud environments

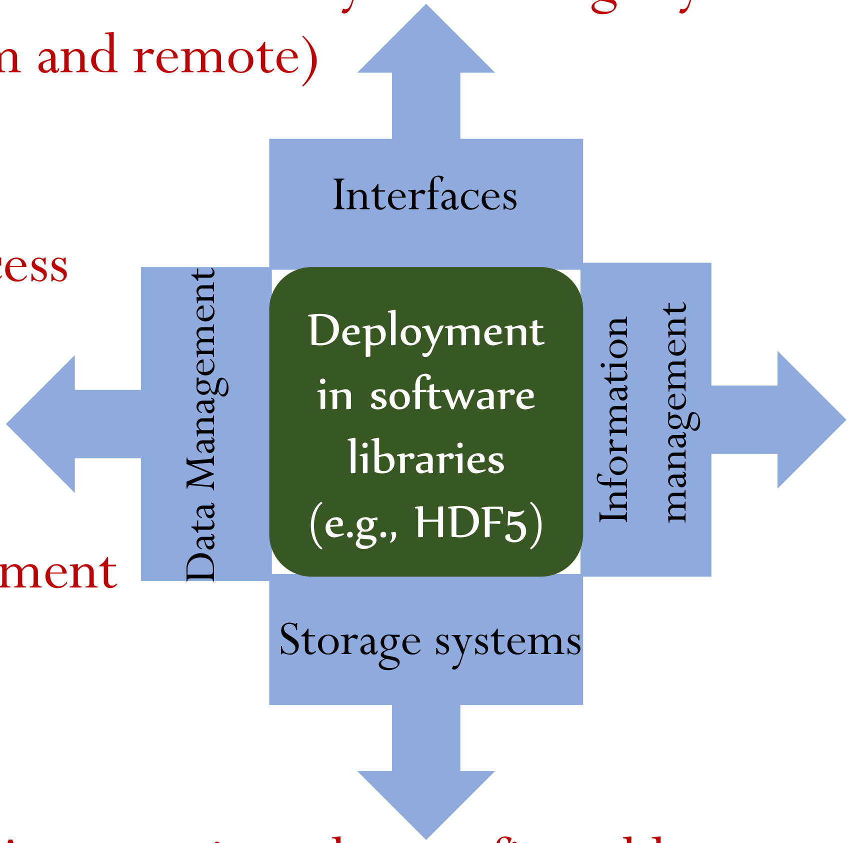
Scientific data storage and access - Looking into future



Future research to achieve autonomous storage and I/O management

* High-productivity interfaces for accessing data fast and easy
* Support for heterogeneous data models
* Unified access to memory and storage system access (in-system and remote)

* Seamless, high-performance access to data and information
* Metadata and provenance management to support FAIR+
* Using AI for efficient data movement



* In-flight data transformations feature extraction using in-storage, in-network computing
* Quantitative metrics for the FAIR principles and data quality
* Recommendation systems for relevant data using AI methods

* Autonomic and reconfigurable storage and I/O systems based on application needs
* Lightweight monitoring of storage systems

Conclusions

- Trends
  - Architecture - deepening hierarchy, high concurrency
  - Applications - increasing diversity (exascale, EOD, ML/AI, etc.)
  - User requirements - productivity and performance, knowledge management
- Our recent solutions at the I/O software level
  - Asynchronous I/O
  - Caching
  - Object-centric data management runtime system
- **Domain scientists should not be burdened with data (I/O and other data movement) and metadata management tasks**
  - **Achieving autonomous storage and I/O requires novel high-productivity interfaces, information extraction and management, automatic data movement, reconfigurable storage, etc.**

Suren Byna: sbyna@lbl.gov

Thanks and Questions

- Contributions from:
  - PDC project team
    - Houjun Tang, Kimmy Mu, Jerome Soumagne, Richard Warren, Teng Wang, Bin Dong, François Tessier, Quincey Koziol, Qiao Kang, Marc Snir, Chen Wang
  - ExaHDF5 / ExaIO team
    - Suren Byna, Scot Breitenfeld, Venkat Vishwanath, Houjun Tang, Qiao Kang, Jean Luca Bez, Huihuo Zheng, John Mainzer, Neil Fortner, Dana Robinson, Jordan Henderson, Jerome Soumagne, Richard Warren, Neelam Bagha, Elena Pourmal, Michela Becchi, John Ravi, Wei Zhang, Yong Chen, Kathryn Mohror, Sarp Oral, Adam Moody, Cameron Stanavige, Michael Brim, Seung-Hwan Lim, Ross Miller, Swen Boehm