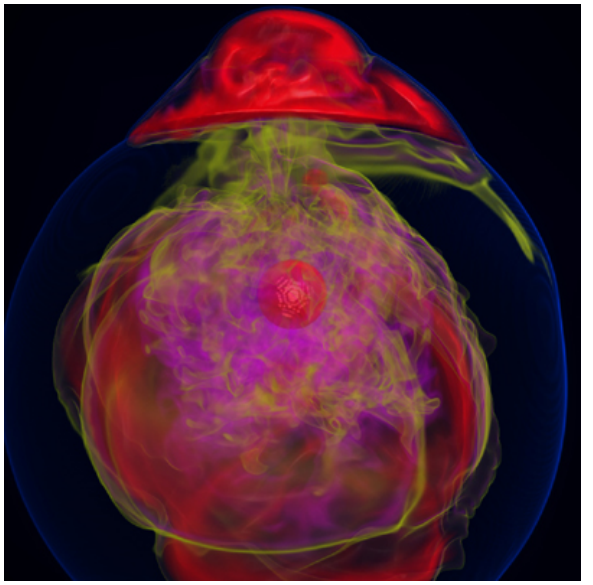
High Performance Computing I/O Systems:
Overview and Recent Developments

Rob Ross
Math and Computer Science Division
Argonne National Laboratory
rross@mcs.anl.gov

Computational Science

- Computational science is a major user of HPC
- Use of computer simulation as a tool for greater understanding of the real world
  - Complements experimentation and theory
- Problems are increasingly computationally expensive
  - Large parallel machines needed to perform calculations
  - Critical to leverage parallelism in all phases
- **Data access is a huge challenge**
  - Using parallelism to obtain performance
  - Finding usable, efficient, and portable interfaces
  - Understanding and tuning I/O

IBM Blue Gene/Q system at Argonne National Laboratory.



Visualization of entropy in Terascale Supernova Initiative application. Image from Kwan-Liu Ma's visualization team at UC Davis.

Outline

Today we will discuss HPC I/O systems, talk about some important concepts, look at some recent developments.

- Material derived from "HPC I/O for Computational Scientists" tutorial, presented earlier this year at ATPESC 2013
- Topics:
  - Overview of HPC I/O systems
    - Users
    - Software layers
    - Performance and optimization
  - Replacing the File Storage Model
  - Burst Buffers (maybe, depending on time)
- Interrupt whenever, goal is to have a good discussion, not necessarily get through every slide.

Argonne NATIONAL LABORATORY

Thinking about HPC I/O Systems

U.S. DEPARTMENT OF ENERGY

HPC I/O Systems

HPC I/O system is the hardware and software that assists in accessing data during simulations and analysis and retaining data between these activities.

- Hardware: disks, disk enclosures, servers, networks, etc.
- Software: parallel file system, libraries, parts of the OS

- Two "flavors" of I/O from applications:
  - **Defensive**: storing data to protect results from data loss due to system faults (i.e., checkpoint and restart)
  - **Productive**: storing/retrieving data as part of the scientific workflow
  - Note: Sometimes these are combined (i.e., data stored both protects from loss and is used in later analysis)
- "Flavor" influences priorities:
  - Defensive I/O: Spend as little time as possible
  - Productive I/O: Capture provenance, organize for analysis

Data Volumes in Computational Science

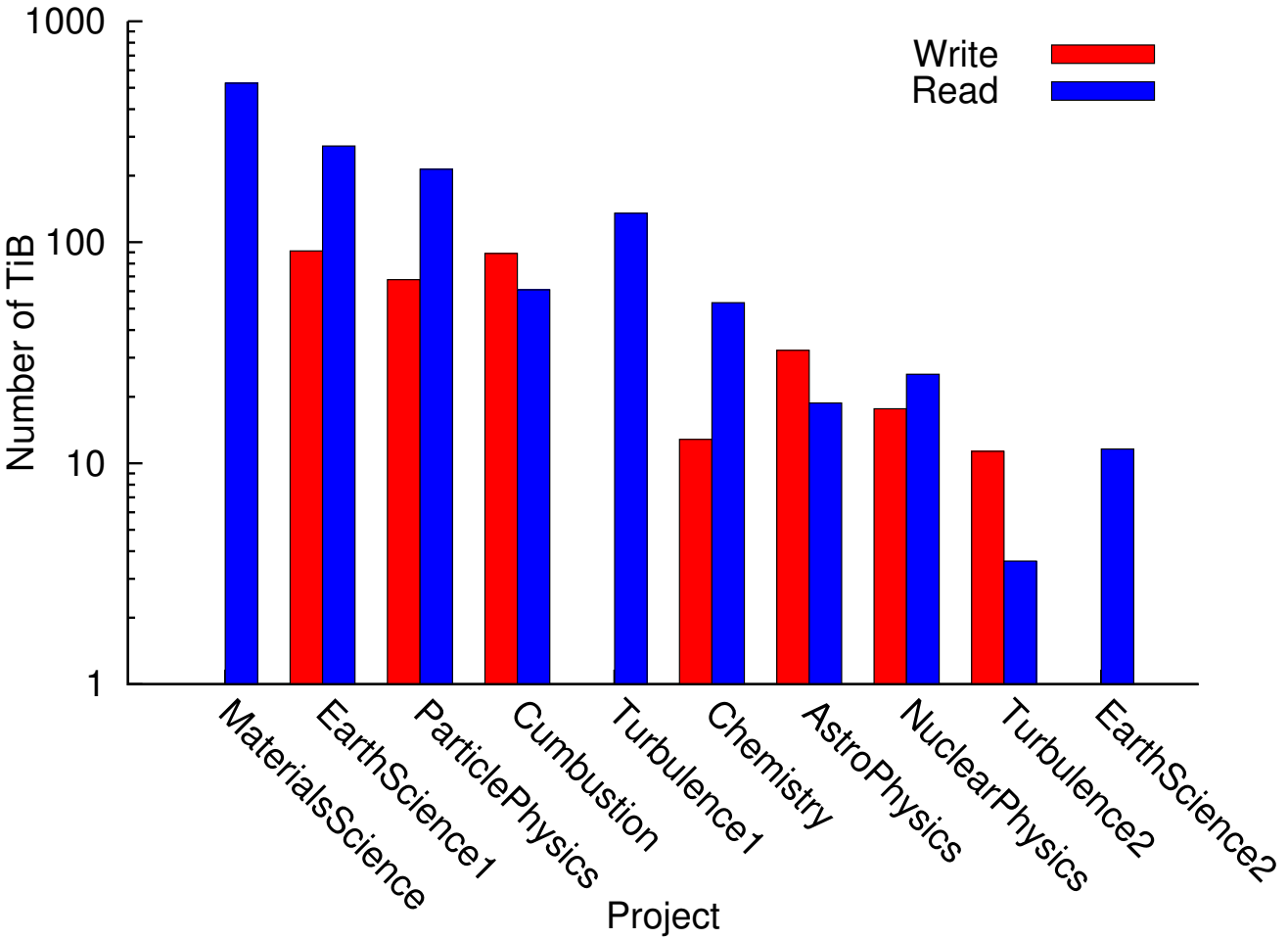
Science teams are routinely working with tens and hundreds of terabytes (TBs) of data.

Data requirements for select 2012 INCITE applications at ALCF (BG/P)

PI	Project	On-line Data (TBytes)	Off-line Data (TBytes)
Lamb	Supernovae Astrophysics	100	400
Khokhlov	Combustion in Reactive Gases	1	17
Lester	CO2 Absorption	5	15
Jordan	Seismic Hazard Analysis	600	100
Washington	Climate Science	200	750
Voth	Energy Storage Materials	10	10
Vashista	Stress Corrosion Cracking	12	72
Vary	Nuclear Structure and Reactions	6	30
Fischer	Reactor Thermal Hydraulic Modeling	100	100
Hinkel	Laser-Plasma Interactions	60	60
Elghobashi	Vaporizing Droplets in a Turbulent Flow	2	4

Data Volumes in Computational Science

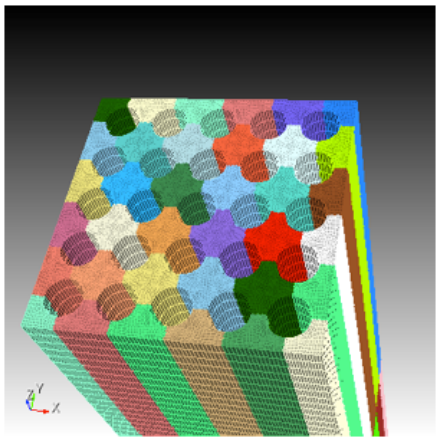
It's not just checkpoints – scientists are reading large volumes of data *into* HPC systems as part of their science.

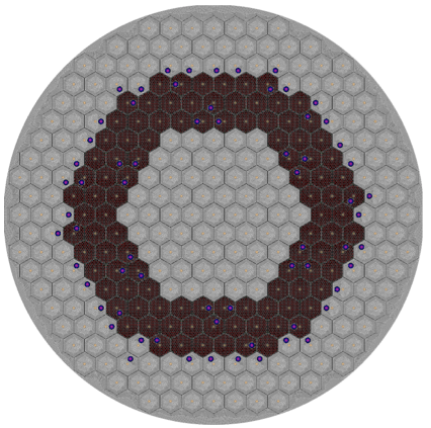


Top 10 data producer/consumers instrumented with Darshan over the month of July, 2011.

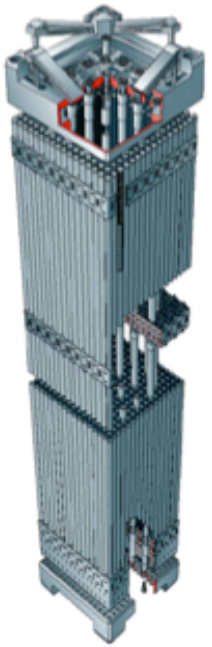
Data Complexity in Computational Science

- Applications have data models appropriate to domain
  - Multidimensional typed arrays, images composed of scan lines, …
  - Headers, attributes on data
- 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)
- Mapping from one to the other is increasingly complex.





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

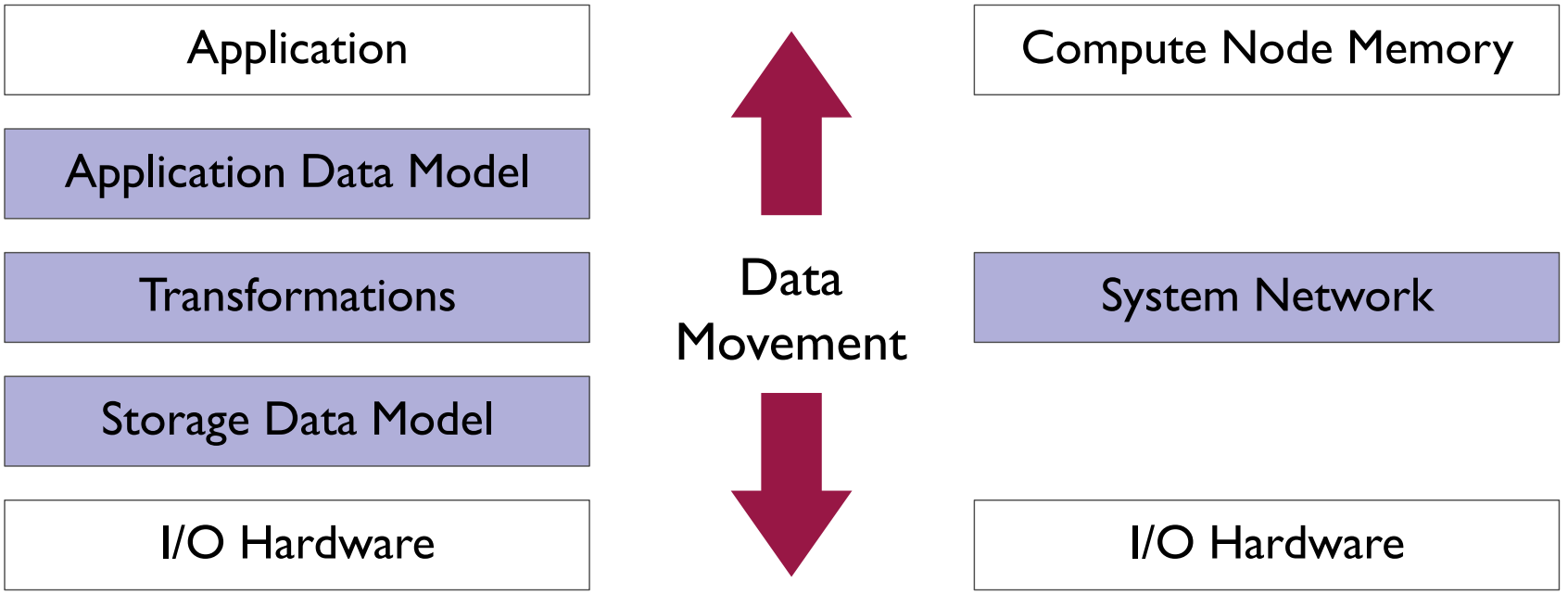


Scale complexity: Spatial range from the reactor core in meters to fuel pellets in millimeters.

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

Views of Data Access in HPC Systems

Two useful ways of thinking about data access are the "logical" view, considering data models in use, and the "physical" view, the components that data resides on and passes through.

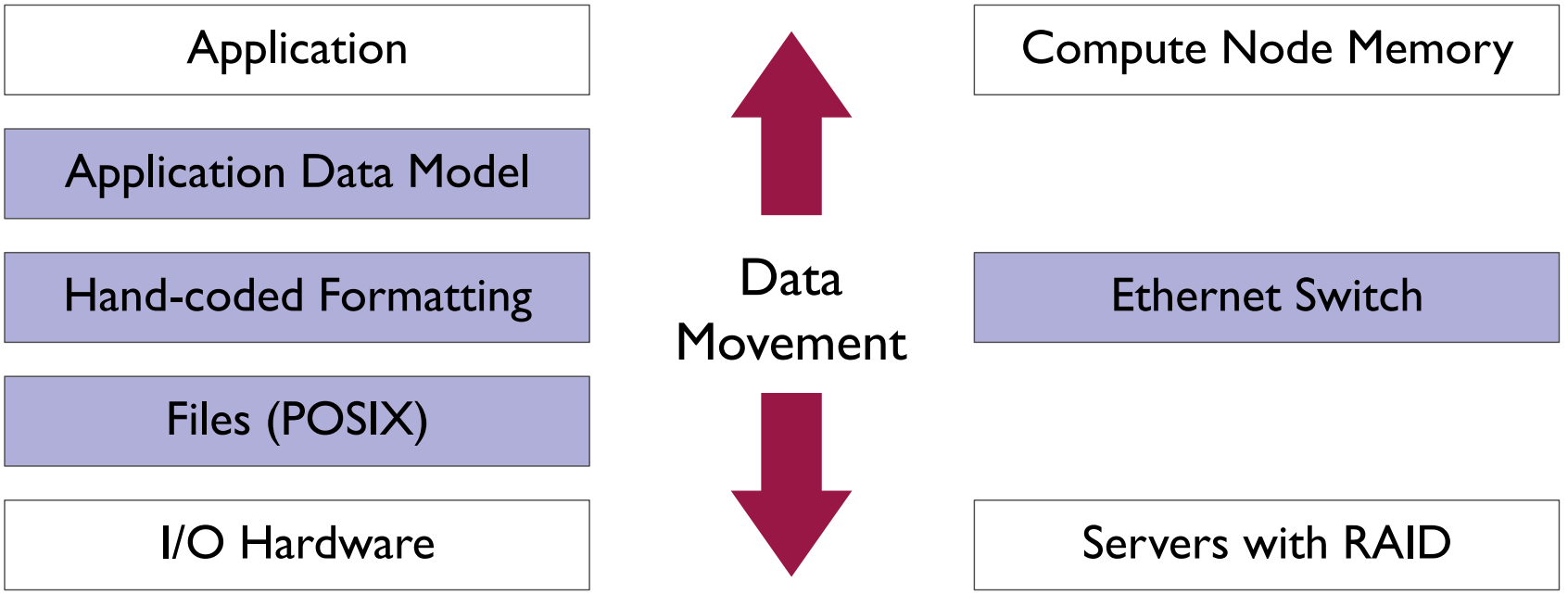


Logical (data model) view of data access.

Physical (hardware) view of data access.

Data Access in Past HPC Systems*

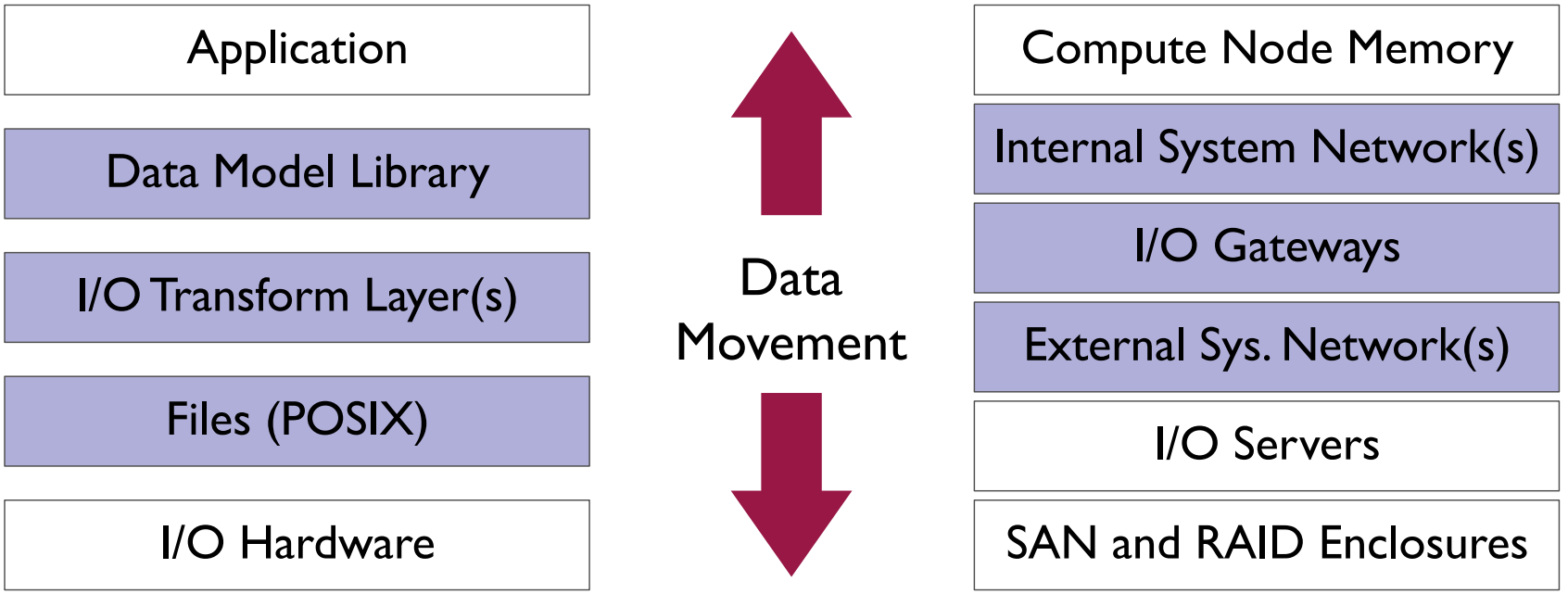
For many years, application teams wrote their own translations from their data models into files, and hardware model was relatively simple.



* We're simplifying the story here somewhat …

Data Access in Current Large-scale Systems

Current systems have greater support on the logical side, more complexity on the physical side.



Logical (data model) view of data access.

Physical (hardware) view of data access.

Thinking about HPC I/O Systems

- Two (intertwined) challenges when thinking about data access:
  - Mapping application data model onto storage
  - Driving all the components so you don't have to wait too long for I/O
- Often these two can be at odds
  - "Richer" data models might require more I/O
  - Transformations that make writing fast might make reading slow (or vice versa)
- Lots of computer science R&D has gone into tackling these two problems

- Next we will dive down into some of the details of HPC I/O

Argonne NATIONAL LABORATORY

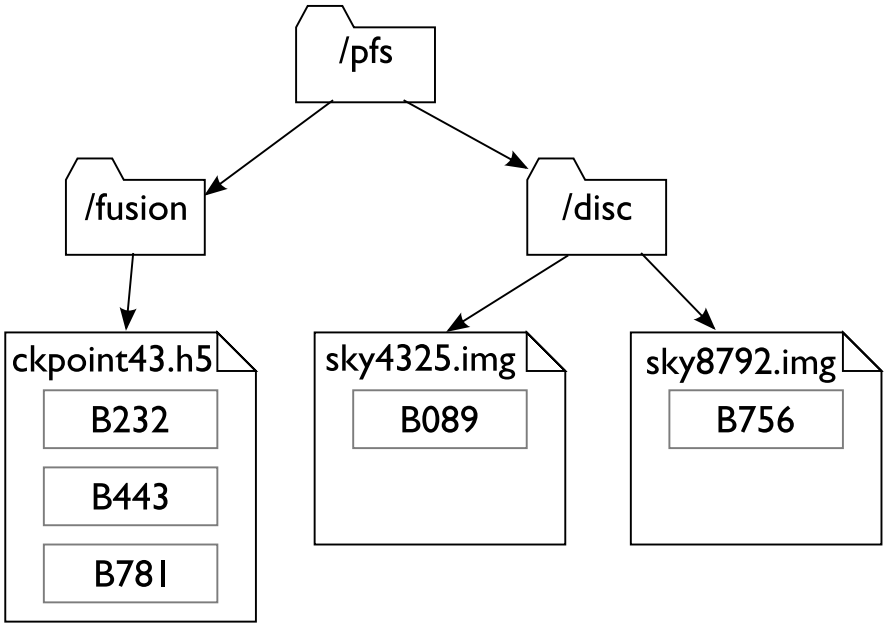
How It Works: HPC I/O Systems

U.S. DEPARTMENT OF ENERGY

How It Works

- HPC I/O systems provide a *file system view* of stored data
  - File (i.e., POSIX) model of access
  - Shared view of data across the system
  - Access to same data from the outside (e.g., login nodes, data movers)

- Topics:
  - How is data stored and organized?
  - What support is there for application data models?
  - How does data move from clients to servers?
  - How is concurrent access managed?
  - What transformations are typically applied?



File system view consists of directories (a.k.a. folders) and files. Files are broken up into regions called extents or blocks.

Storing and Organizing Data: Storage Model

HPC I/O systems are built around a *parallel file system* that organizes storage and manages access.

- Parallel file systems (PFSes) are distributed systems that provide a file data model (i.e., files and directories) to users
- Multiple PFS servers manage access to storage, while PFS client systems run applications that access storage
- PFS clients can access storage resources in parallel!

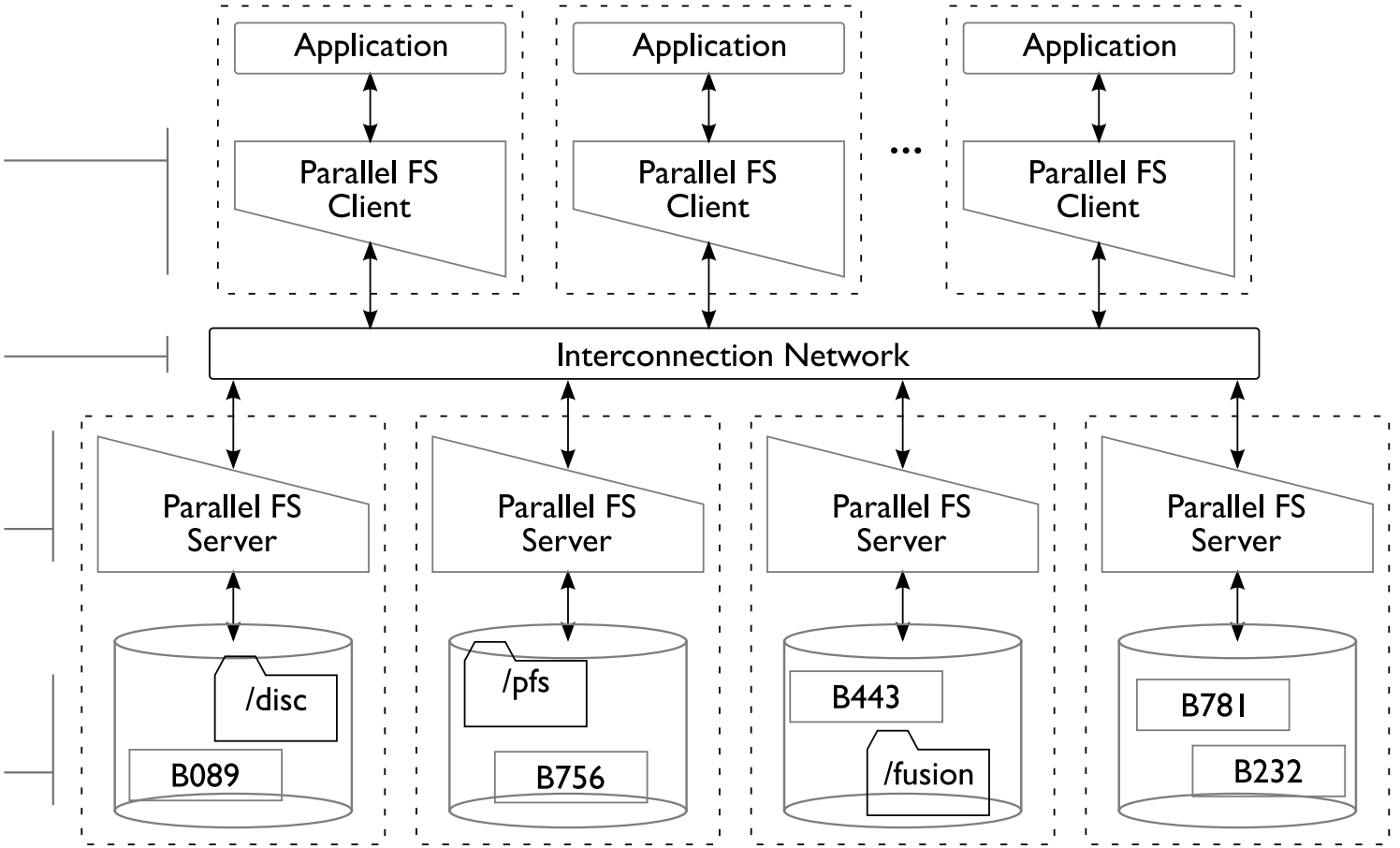
Reading and Writing Data (etc.)

PFS client software requests operations on behalf of applications. Requests are sent as messages (RPC-like), often to multiple servers.

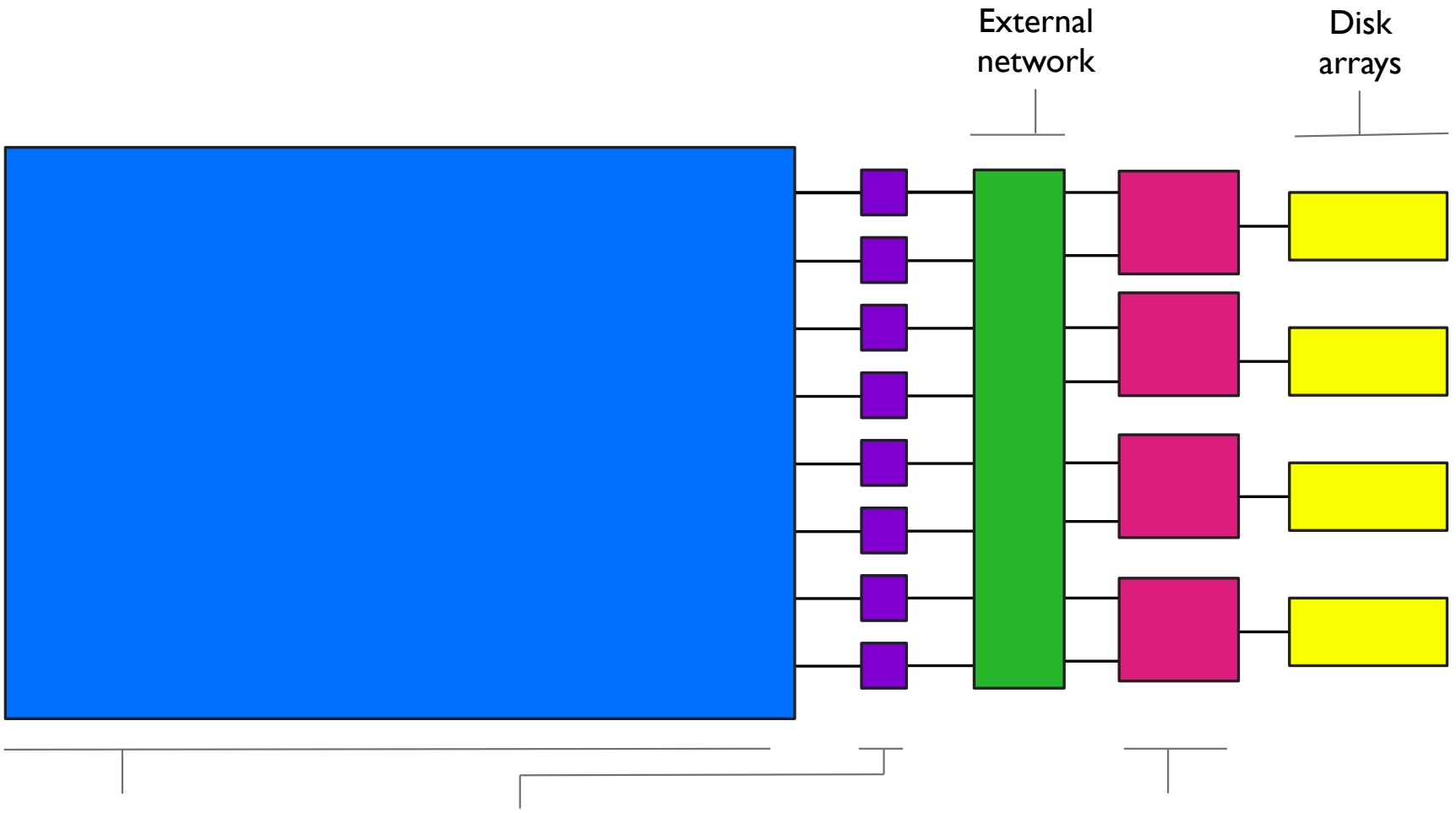
Requests pass over the interconnect, thus **each request incurs some latency**.

PFS servers manage local storage, services incoming requests from clients.

RAID enclosures protect against individual disk failures and map regions of data onto specific devices.



Leadership Systems have an additional HW layer



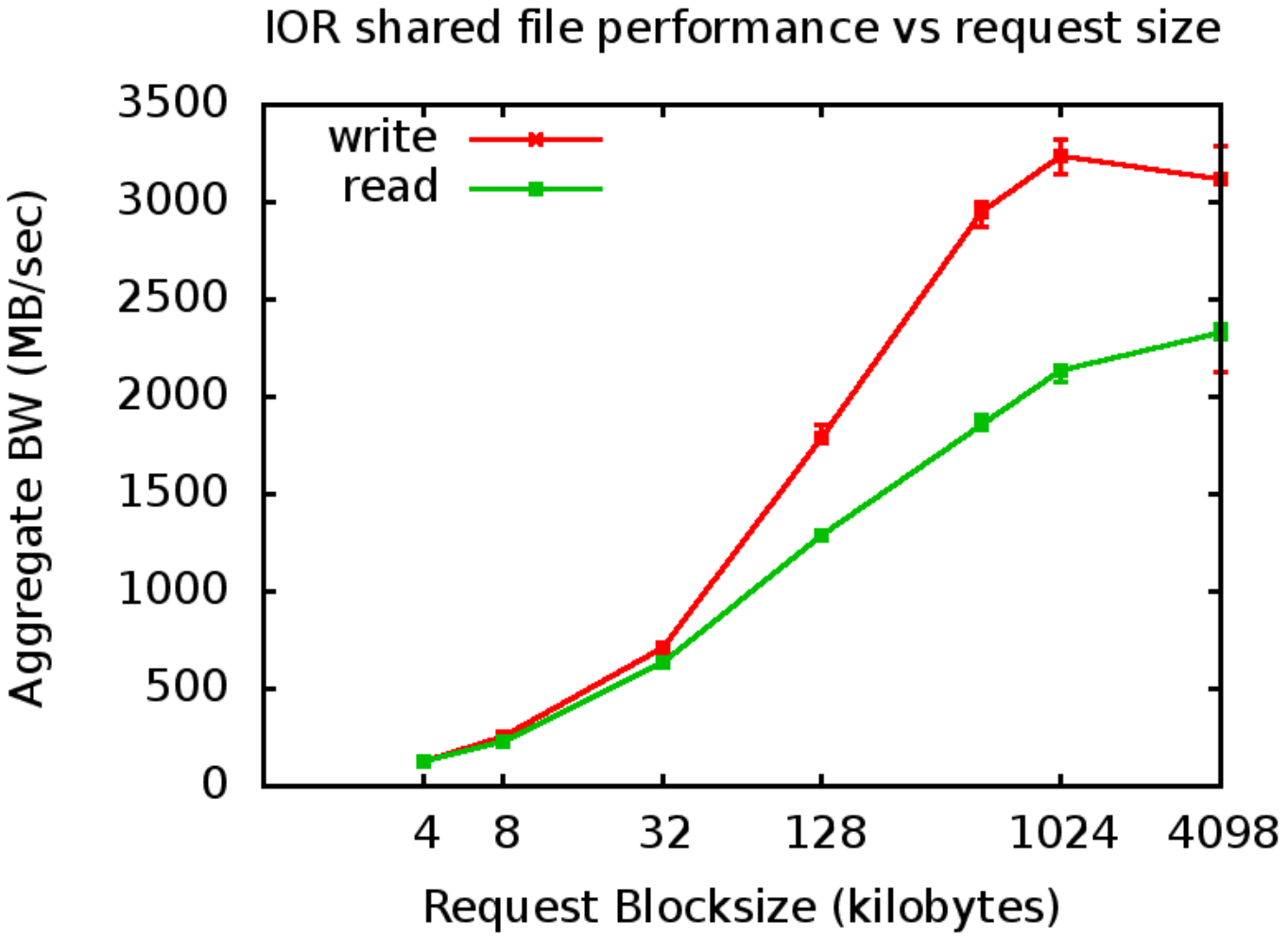
Compute nodes run application processes. Data model software also runs here, and some I/O transformations are performed here.

I/O forwarding nodes (or I/O gateways) shuffle data between compute nodes and external resources, including storage.

Storage nodes run the parallel file system.

Request Size and I/O Rate

Interconnect latency has a significant impact on effective rate of I/O. Typically I/Os should be in the O(Mbytes) range.



Tests run on 2K processes of IBM Blue Gene/P at ANL.

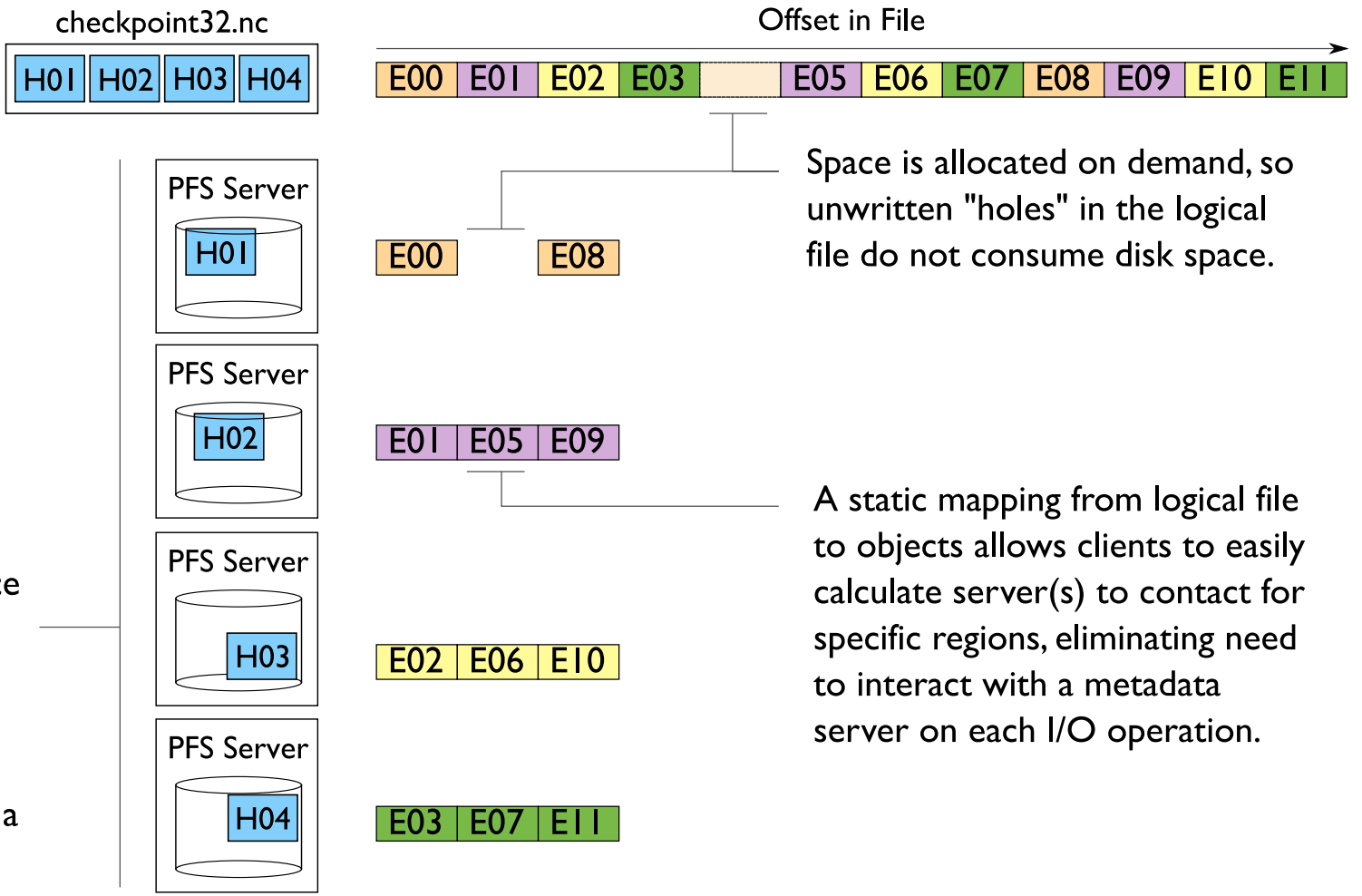
Data Distribution in Parallel File Systems

Distribution across multiple servers allows concurrent access.

Logically a file is an extendable sequence of bytes that can be referenced by offset into the sequence.

Metadata associated with the file specifies a mapping of this sequence of bytes into a set of objects on PFS servers.

Extents in the byte sequence are mapped into objects on PFS servers. This mapping is usually determined at file creation time and is often a round-robin distribution of a fixed extent size over the allocated objects.

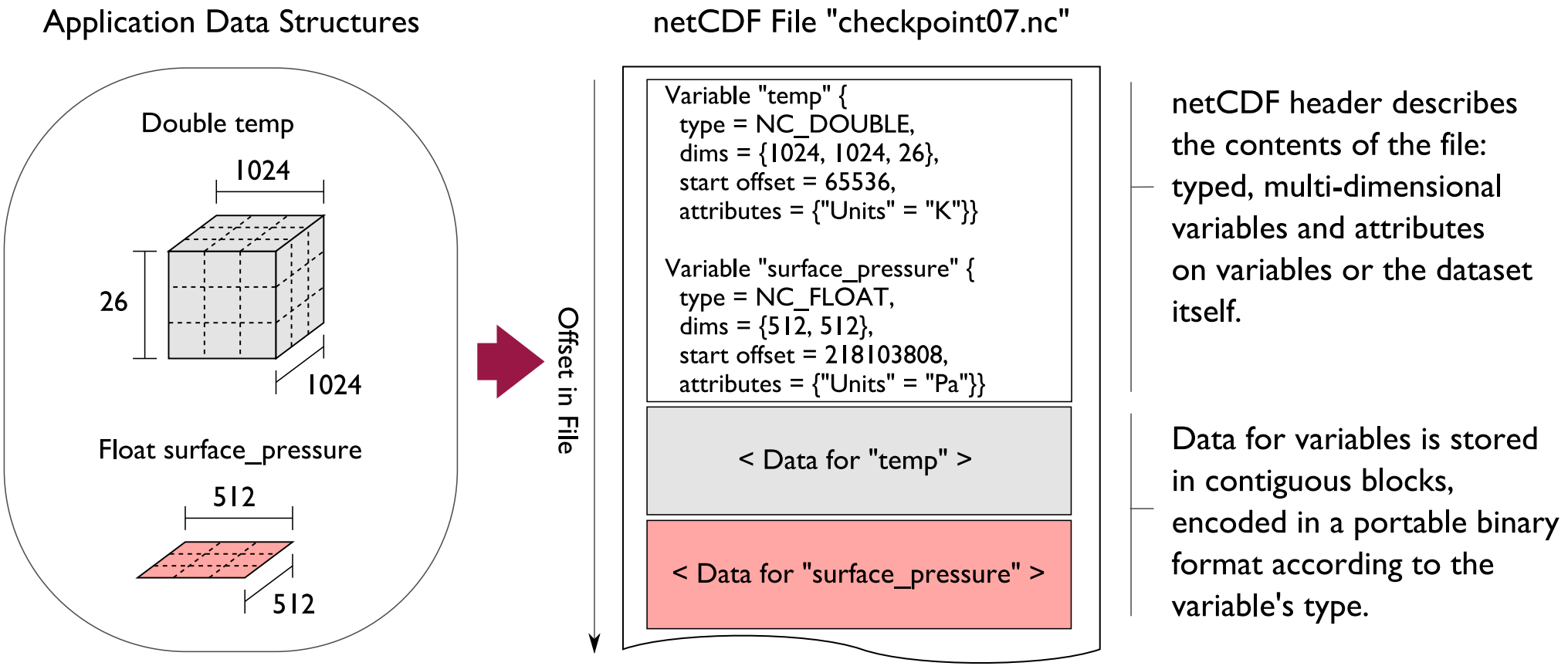


Space is allocated on demand, so unwritten "holes" in the logical file do not consume disk space.

A static mapping from logical file to objects allows clients to easily calculate server(s) to contact for specific regions, eliminating need to interact with a metadata server on each I/O operation.

Storing and Organizing Data: Application Model(s)

Application data models are supported via libraries that map down to files (and sometimes directories).



HPC I/O Software Stack

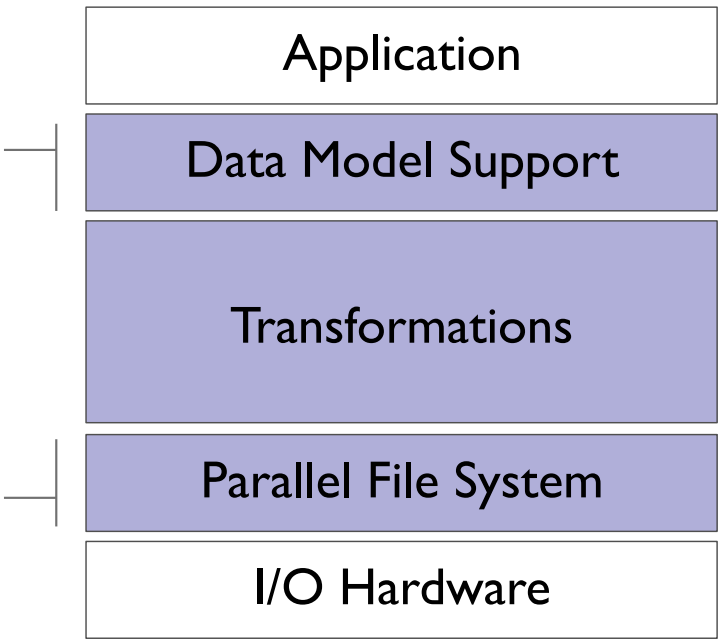
The software used to provide data model support and to transform I/O to better perform on today's I/O systems is often referred to as the *I/O stack.*

Data Model Libraries map application abstractions onto storage abstractions and provide data portability.

HDF5, Parallel netCDF, ADIOS

Parallel file system maintains logical file model and provides efficient access to data.

PVFS, PanFS, GPFS, Lustre



I/O Middleware organizes accesses from many processes, especially those using collective I/O.

MPI-IO, GLEAN, PLFS

I/O Forwarding transforms I/O from many clients into fewer, larger request; reduces lock contention; and bridges between the HPC system and external storage.

IBM ciod, IOFSL, Cray DVS

Argonne NATIONAL LABORATORY

How It Works: HPC I/O Performance

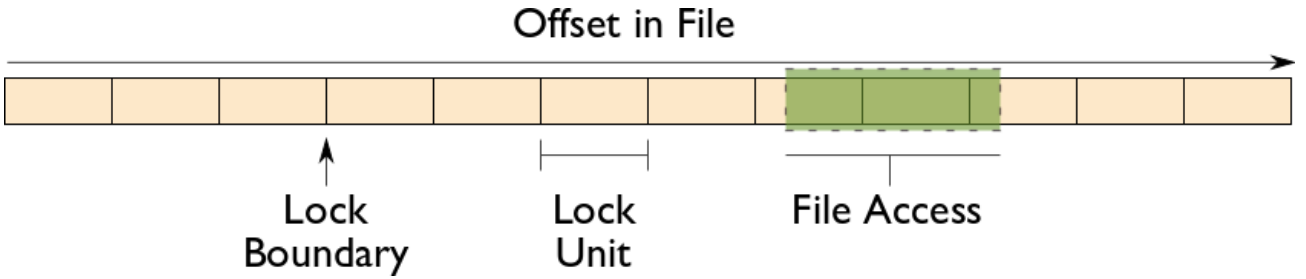
U.S. DEPARTMENT OF ENERGY

Managing Concurrent Access

Files are treated like global shared memory regions. Locks are used to manage concurrent access:

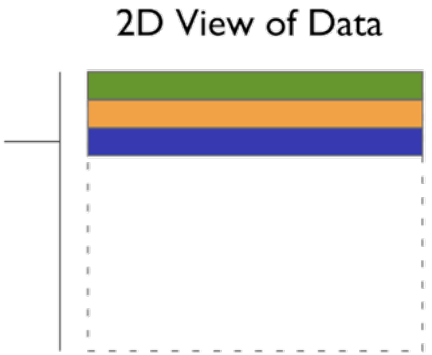
- Files are broken up into lock units
- Clients obtain locks on units that they will access before I/O occurs
- Enables caching on clients as well (as long as client has a lock, it knows its cached data is valid)
- Locks are reclaimed from clients when others desire access

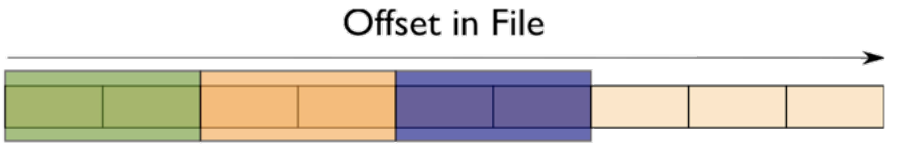
If an access touches any data in a lock unit, the lock for that region must be obtained before access occurs.



Implications of Locking in Concurrent Access

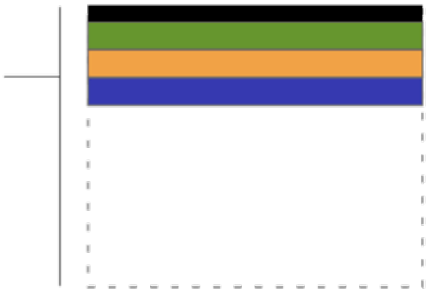
The left diagram shows a row-block distribution of data for three processes. On the right we see how these accesses map onto locking units in the file.

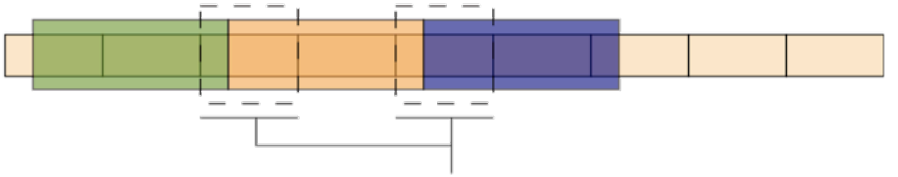




When accesses are to large contiguous regions, and aligned with lock boundaries, locking overhead is minimal.

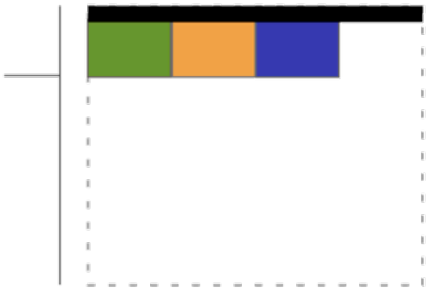
In this example a header (black) has been prepended to the data. If the header is not aligned with lock boundaries, false sharing will occur.

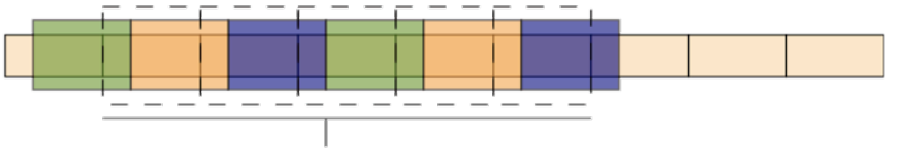




These two regions exhibit *false sharing*: no bytes are accessed by both processes, but because each block is accessed by more than one process, there is contention for locks.

In this example, processes exhibit a block-block access pattern (e.g. accessing a subarray). This results in many interleaved accesses in the file.



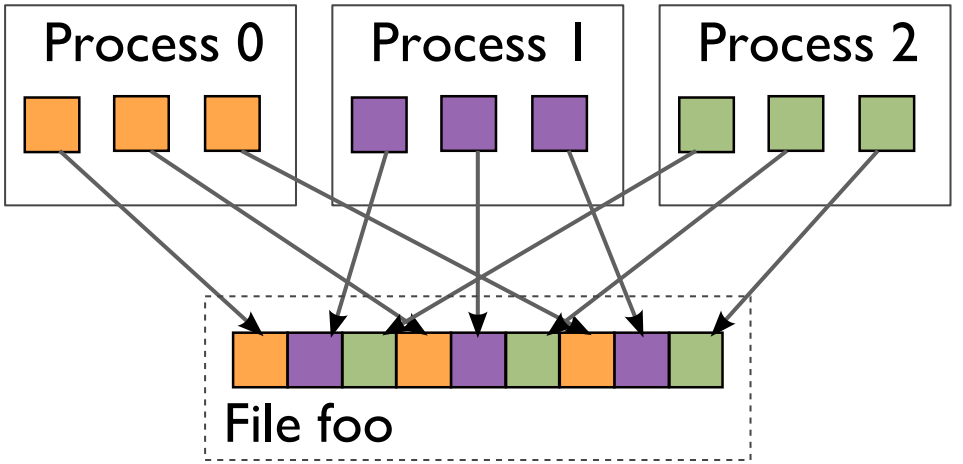


When a block distribution is used, sub-rows cause a higher degree of false sharing, especially if data is not aligned with lock boundaries.

I/O Transformations

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

- Goals of transformations:
  - Reduce number of operations to PFS (avoiding latency)
  - Avoid lock contention (increasing level of concurrency)
  - Hide number of clients (more on this later)
- With "transparent" transformations, data ends up in the same locations in the file
  - i.e., the file system is still aware of the actual data organization

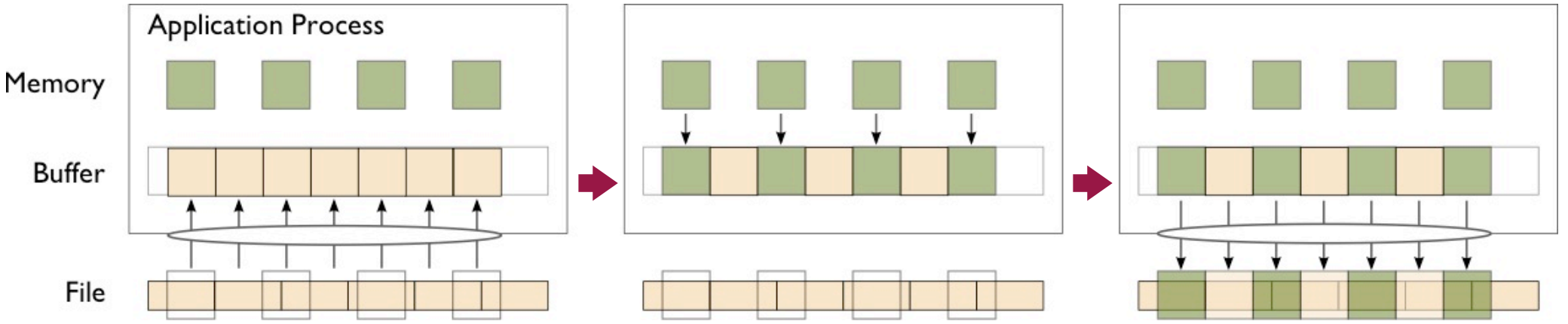


When we think about I/O transformations, we consider the mapping of data between application processes and locations in file.

Reducing Number of Operations

Since most operations go over the network, I/O to a PFS incurs more latency than with a local FS. *Data sieving* is a technique to address I/O latency by combining operations:

- When reading, application process reads a large region holding all needed data and pulls out what is needed
- When writing, three steps required (below)



Step 1: Data in region to be modified are read into intermediate buffer (1 read).

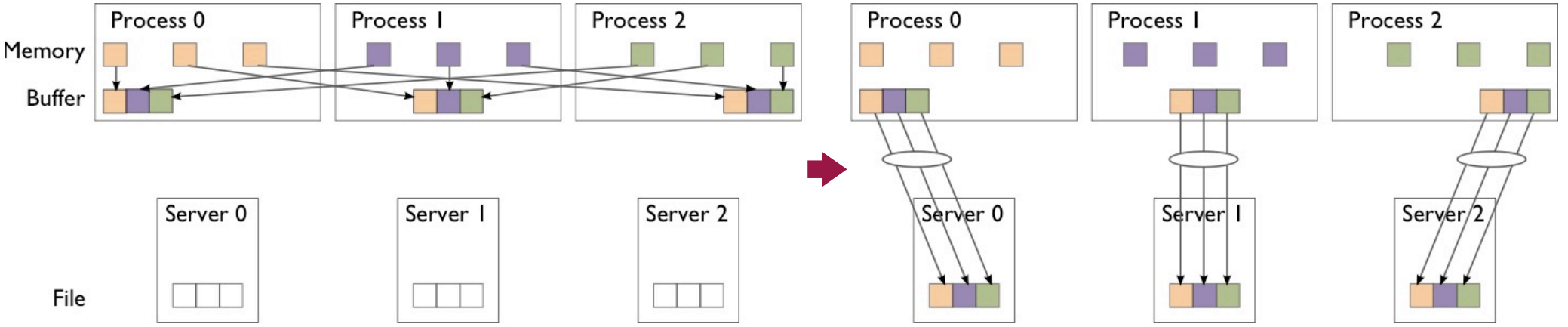
Step 2: Elements to be written to file are replaced in intermediate buffer.

Step 3: Entire region is written back to storage with a single write operation.

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
- 0^{th} phase determines exchange schedule (not shown)



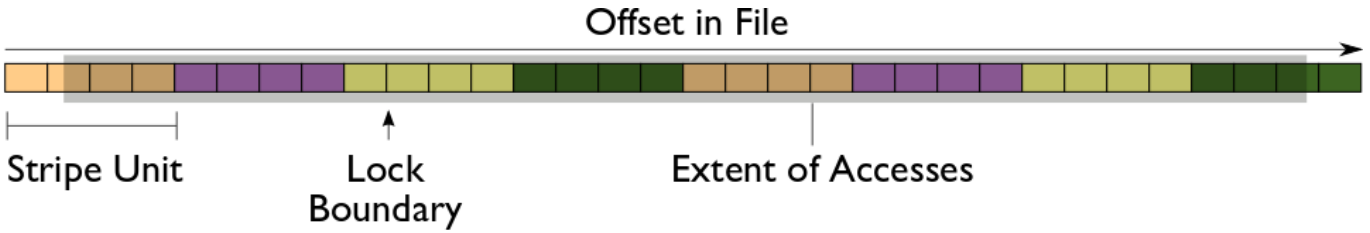
Phase 1: Data are exchanged between processes based on organization of data in file.

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

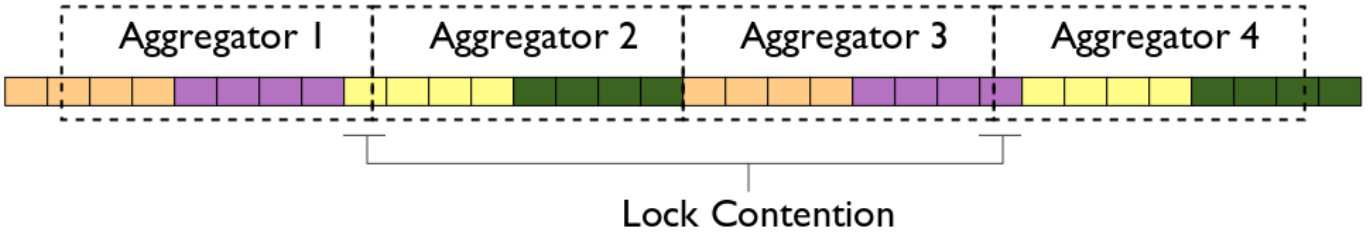
Two-Phase I/O Algorithms

(or, You don't want to do this yourself...)

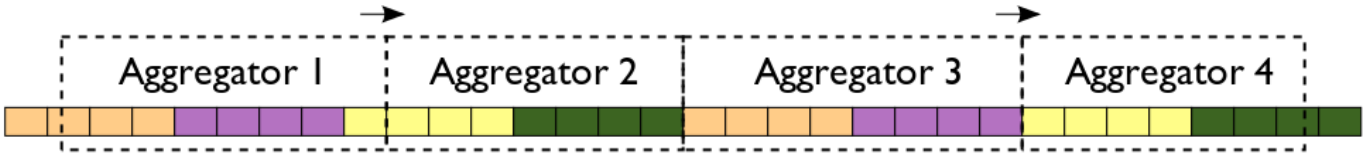
Imagine a collective I/O access using four aggregators to a file striped over four file servers (indicated by colors):



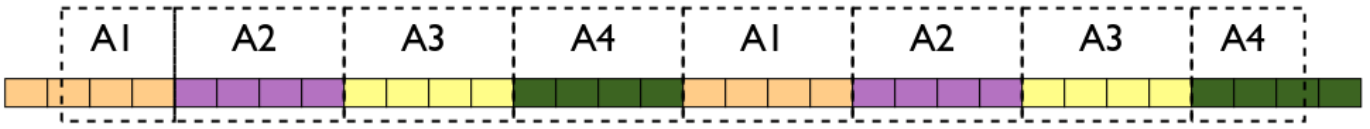
One approach is to evenly divide the region accessed across aggregators.



Aligning regions with lock boundaries eliminates lock contention.



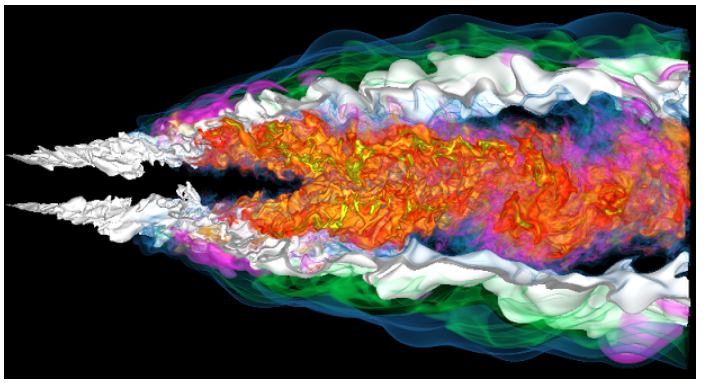
Mapping aggregators to servers reduces the number of concurrent operations on a single server and can be helpful when locks are handed out on a per-server basis (e.g., Lustre).

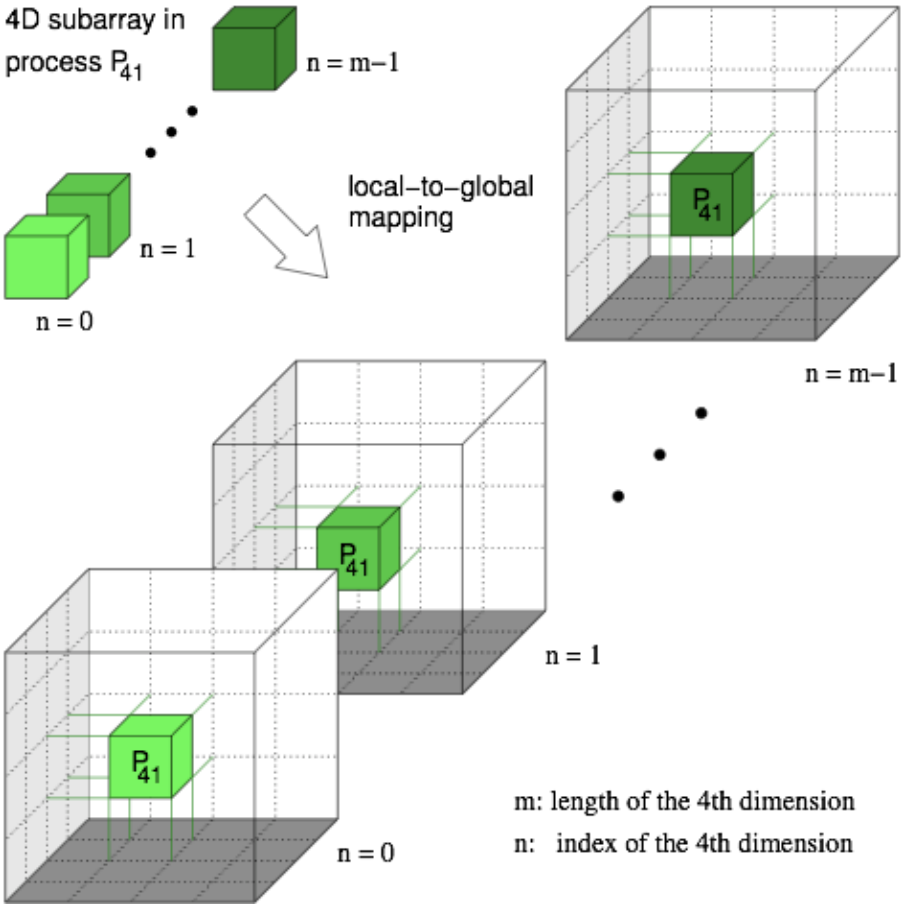


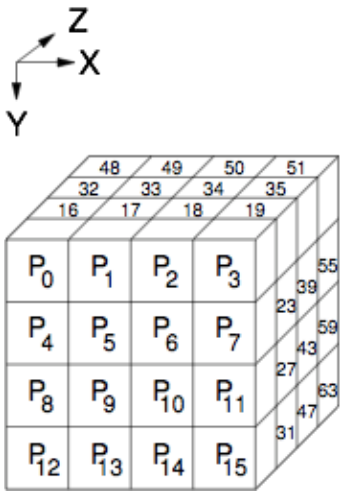
For more information, see W.K. Liao and A. Choudhary, "Dynamically Adapting File Domain Partitioning Methods for Collective I/O Based on Underlying Parallel File System Locking Protocols," SC2008, November, 2008.

S3D Turbulent Combustion Code

- S3D is a turbulent combustion application using a direct numerical simulation solver from Sandia National Laboratory
- Checkpoints consist of four global arrays
  - 2 3-dimensional
  - 2 4-dimensional
  - 50x50x50 fixed subarrays







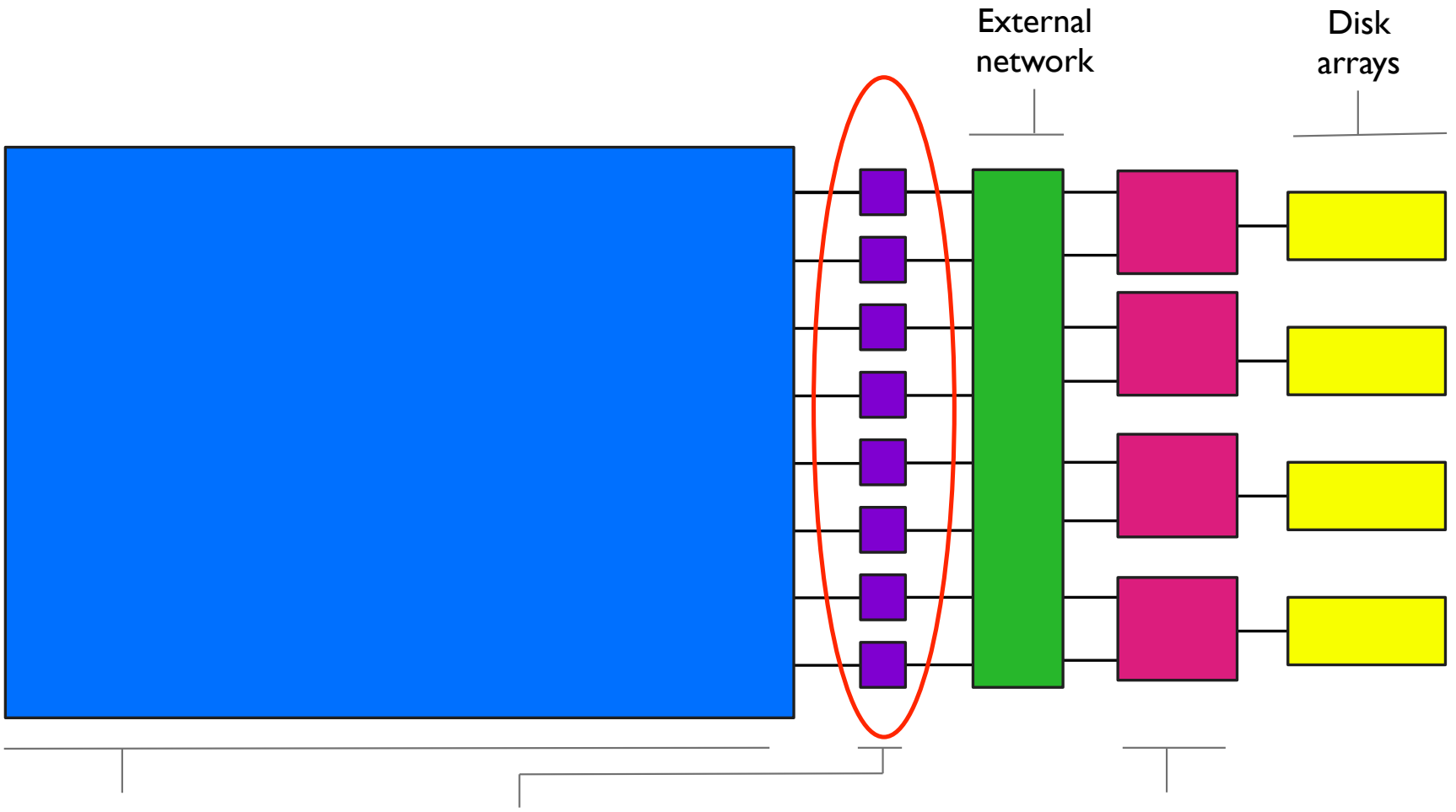
Thanks to Jackie Chen (SNL), Ray Grout (SNL), and Wei-Keng Liao (NWU) for providing the S3D I/O benchmark, Wei-Keng Liao for providing this diagram, C. Wang, H. Yu, and K.-L. Ma of UC Davis for image.

Impact of Transformations on S3D I/O

- Testing with PnetCDF output to single file, three configurations, 16 processes
  - All MPI-IO optimizations (collective buffering and data sieving) disabled
  - Independent I/O optimization (data sieving) enabled
  - Collective I/O optimization (collective buffering, a.k.a. two-phase I/O) enabled

	Coll. Buffering and Data Sieving Disabled	Data Sieving Enabled	Coll. Buffering Enabled (incl. Aggregation)
POSIX writes	102,401	81	**5**
POSIX reads	0	80	0
MPI-IO writes	64	64	64
Unaligned in file	102,399	80	4
Total written (MB)	6.25	**87.11**	6.25
Runtime (sec)	1443	11	6.0
Avg. MPI-IO time per proc (sec)	**1426.47**	4.82	0.60

Transformations in the I/O Forwarding Step



I/O forwarding nodes (or I/O gateways) shuffle data between compute nodes and external resources, including storage.

Transformations in the I/O Forwarding Step

Another way of transforming data access by clients is by introducing new hardware: *I/O forwarding nodes*.

- I/O forwarding nodes serve a number of functions:
  - Bridge between internal and external networks
  - Run PFS client software, allowing lighter-weight solutions internally
  - Perform I/O operations on behalf of multiple clients

- Transformations can take many forms:
  - Performing one file open on behalf of many processes
  - Combining small accesses into larger ones
  - Caching of data (sometimes between I/O forwarding nodes)

  Note: Current vendor implementations don't aggressively aggregate.

- Compute nodes can be allocated to provide a similar service

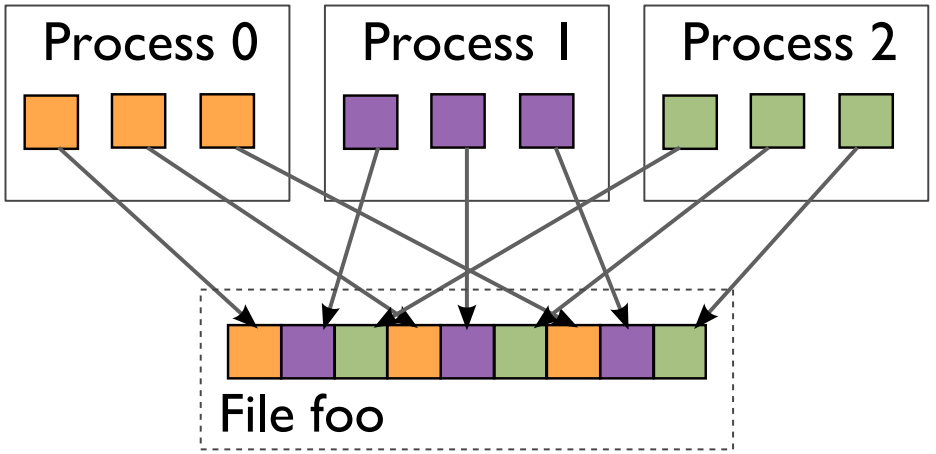
“Not So Transparent” Transformations

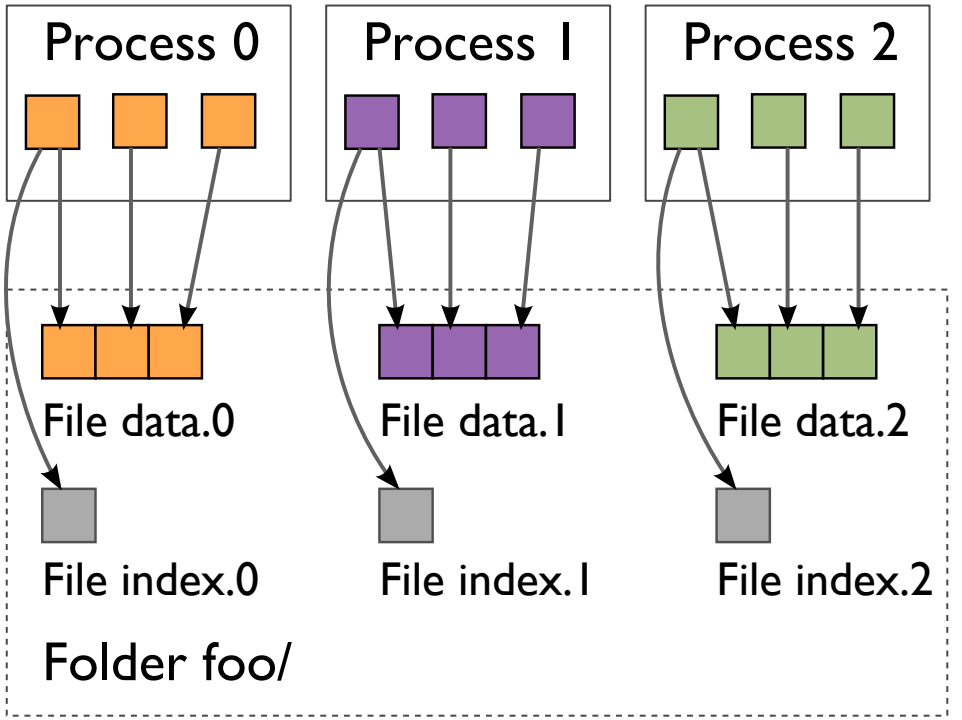
Some transformations result in file(s) with different data organizations than the user requested.

- If processes are writing to different files, then they will not have lock conflicts
- What if we convert writes to the same file into writes to different files?
  - Need a way to group these files together
  - Need a way to track what we put where
  - Need a way to reconstruct on reads
- Parallel Log-Structured File System software does this

See J. Bent et al. PLFS: a checkpoint filesystem for parallel applications. SC2009. Nov. 2009.

Parallel Log Structured File System





Application intends to interleave data regions into single file.

Transparent transformations such as data sieving and two-phase I/O preserve data order on the file system.

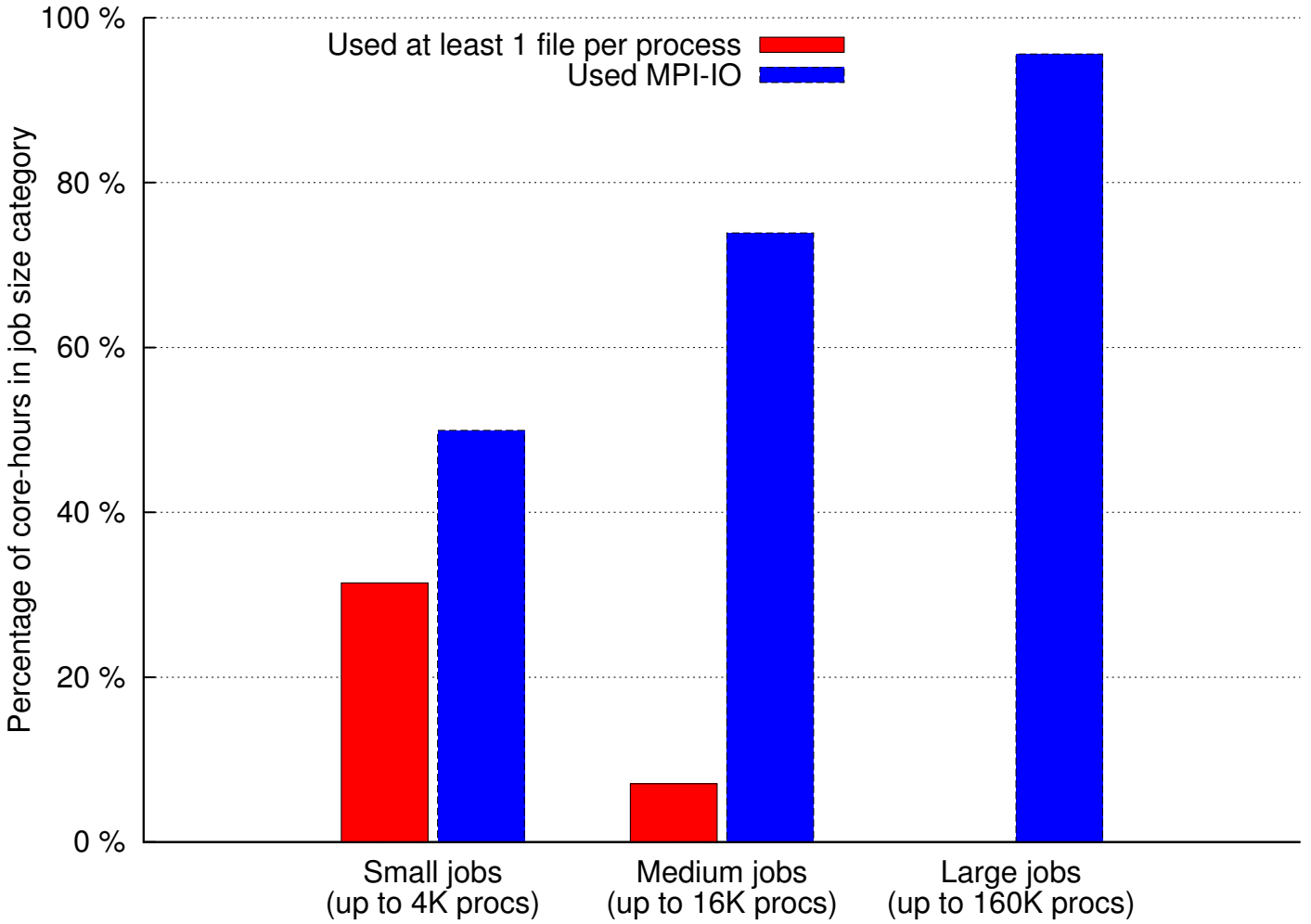
PLFS remaps I/O into separate log files per process, with indices capturing locations of data in these files.

PLFS software needed when reading to reconstruct the file view.

See J. Bent et al. PLFS: a checkpoint filesystem for parallel applications. SC2009. Nov. 2009.

Why not just write a file per process?

File per process vs. shared file access as function of job size on Intrepid Blue Gene/P system



I/O Transformations and the Storage Data Model

Historically, the storage data model has been the POSIX file model, and the PFS has been responsible for managing it.

- Transparent transformations work within these limitations
- When data model libraries are used:
  - Transforms can take advantage of more knowledge
    - e.g., dimensions of multidimensional datasets
  - Doesn't matter so much whether there is a single file underneath
  - Or in what order the data is stored
  - As long as portability is maintained
- Single stream of bytes in a file is inconvenient for parallel access
  - Later will discuss efforts to provide a different underlying model

Argonne NATIONAL LABORATORY

Replacing the File Storage Model

Many thanks to:

Dave Goodell
Cisco Systems

Shawn Kim
Penn State University

Mahmut Kandemir
Penn State University

The Problem with the File Storage Model

- The POSIX file model gives us a single stream to work with
- HPC applications create complex output that is naturally multi-stream
  - Structured datasets (e.g., PnetCDF, HDF5)
  - Log-based datasets (e.g., PLFS, ADIOS BP)
- Dilemma
  - Do I create lots of files to hold all these streams?
  - Do I map all the streams into a single file?

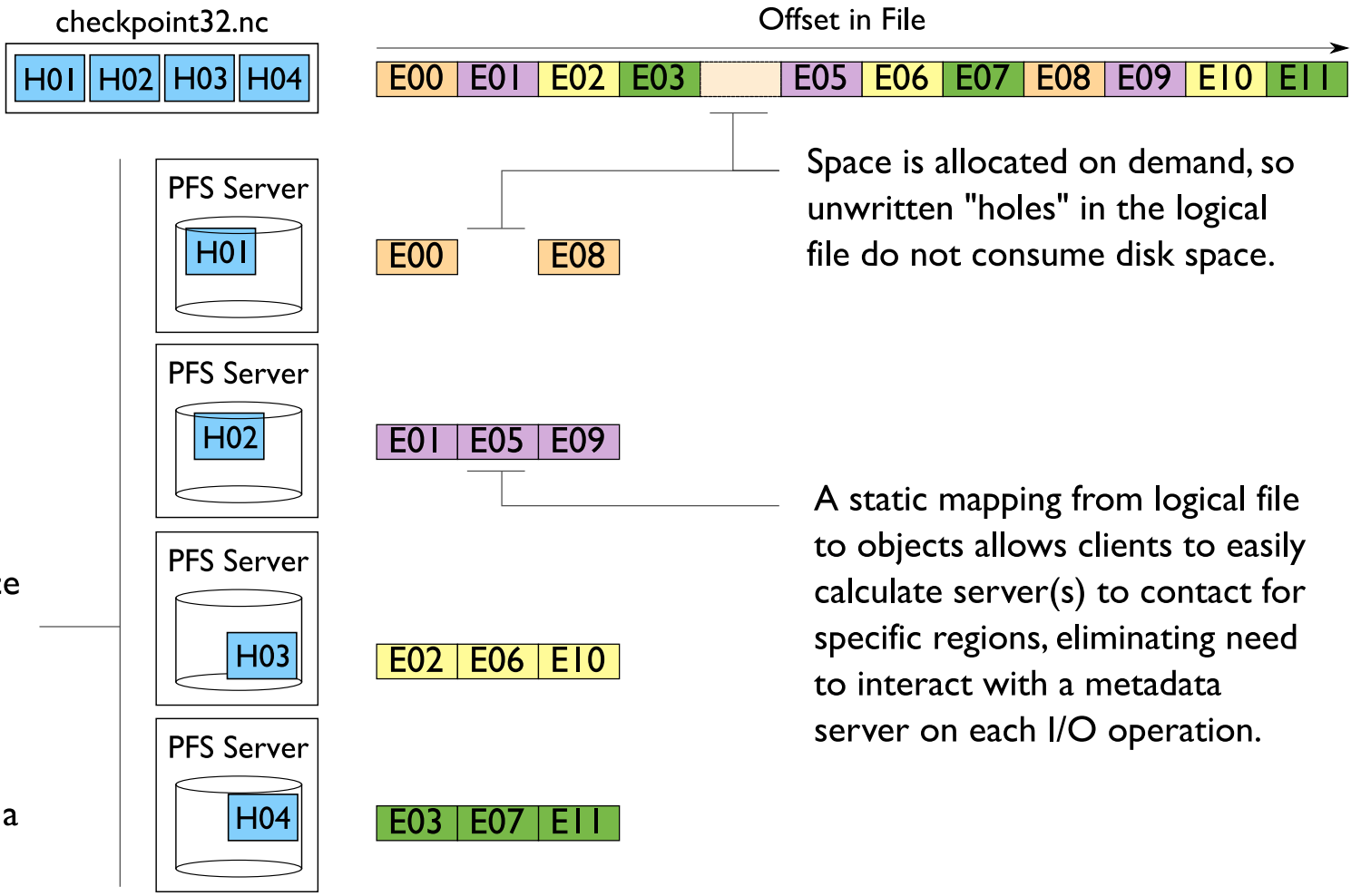
Recall: Data Distribution in Parallel File Systems

Modern parallel file systems internally manage multiple data streams; they just aren't exposed to the user.

Logically a file is an extendable sequence of bytes that can be referenced by offset into the sequence.

Metadata associated with the file specifies a mapping of this sequence of bytes into a set of objects on PFS servers.

Extents in the byte sequence are mapped into objects on PFS servers. This mapping is usually determined at file creation time and is often a round-robin distribution of a fixed extent size over the allocated objects.



Space is allocated on demand, so unwritten "holes" in the logical file do not consume disk space.

A static mapping from logical file to objects allows clients to easily calculate server(s) to contact for specific regions, eliminating need to interact with a metadata server on each I/O operation.

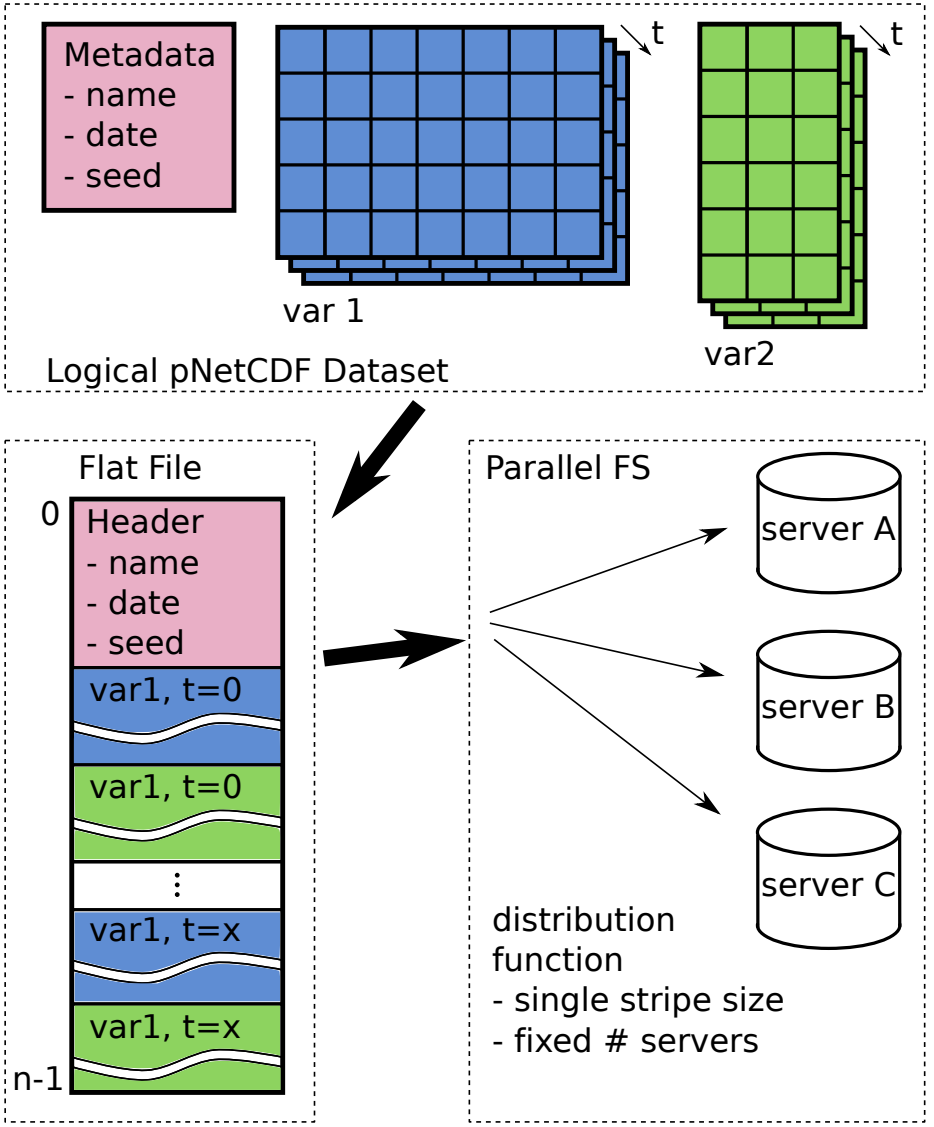
An Alternative Storage Model

- Expose individual object streams for use by users and I/O libraries
  - Users/libraries become responsible for mapping of data to objects
- Keep the name space as it is
  - Directories, file names, permissions

- General approach is being pursued by the Intel Fast Forward team (Intel, EMC, HDF5 Group) and also by ANL/Penn State

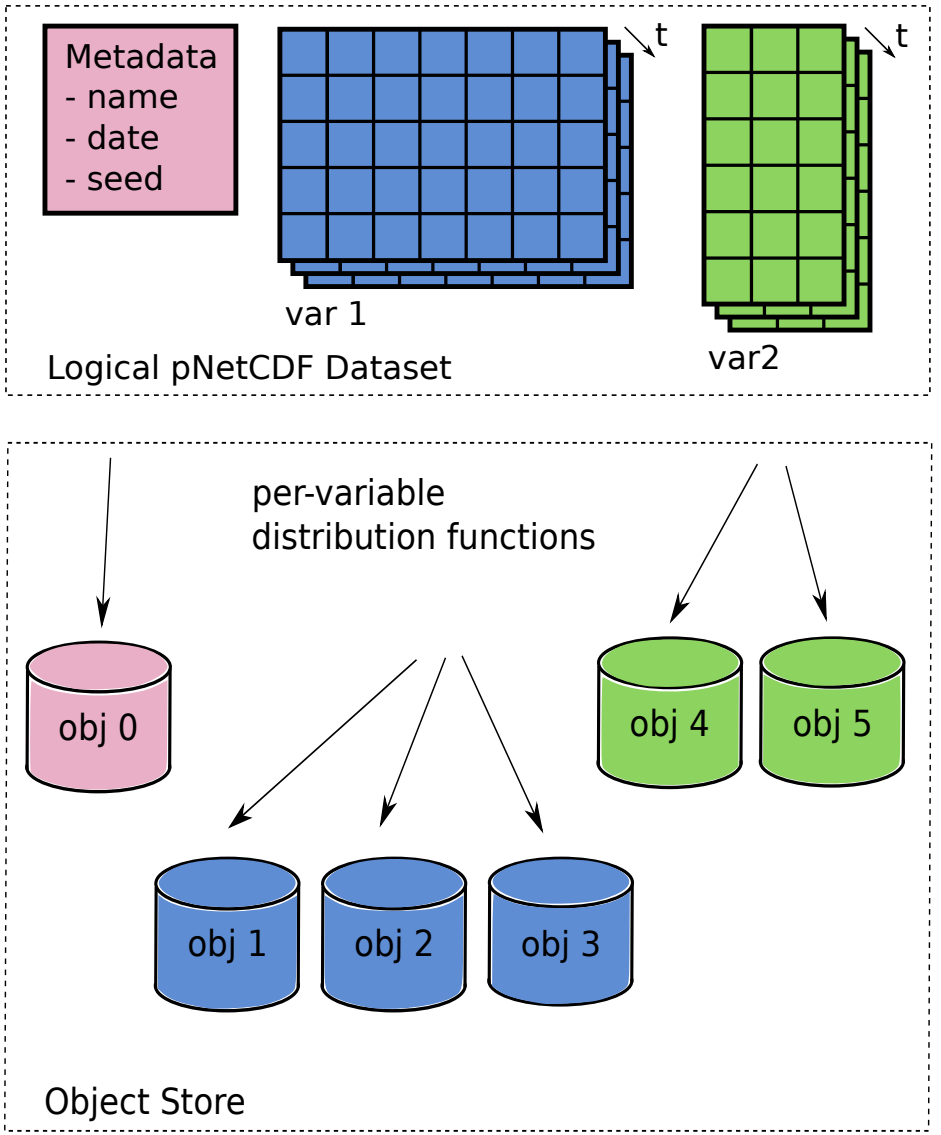
PnetCDF Mapping to POSIX Files

PnetCDF organizes data into byte stream. Record variables interleaved at end of file. Minimal awareness of FS properties.



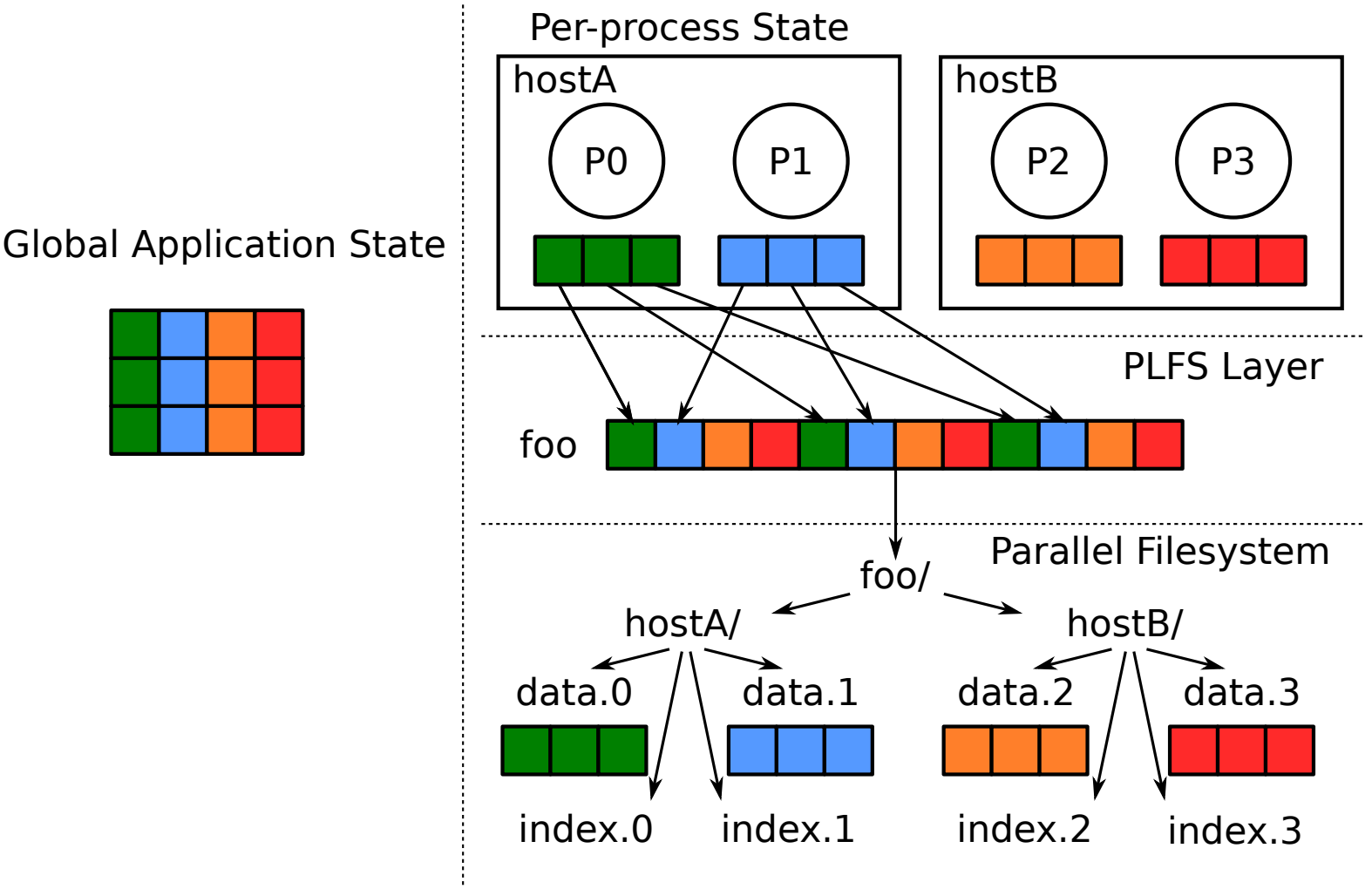
PnetCDF Mapping to Alternative Storage Model

Variables mapped into distinct objects. Resizing of one variable has no impact on others.



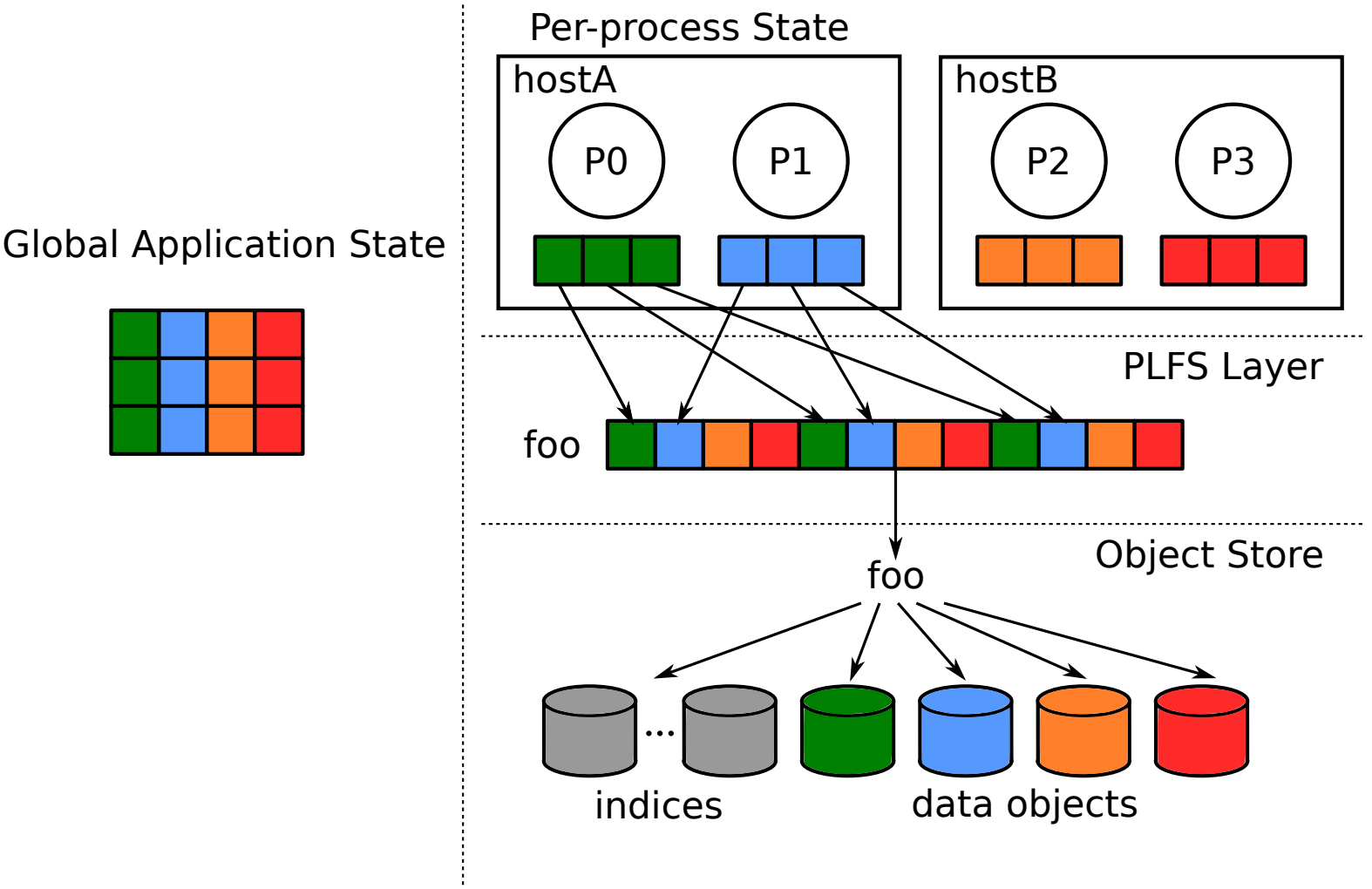
PLFS Mapping to POSIX Files

Data from a process lands in a unique file.



PLFS Mapping to Alternative Storage Model

Data from a process lands in a unique object; overhead to create objects much lower than overhead to create files.



Other Interesting Ideas

- Lots of alternatives being kicked around in various contexts:
  - Record-oriented storage
  - Forks
  - Search / **alternative name spaces**
  - Versioning storage
- Our hope is that we see a standard replacement emerge for shared block storage and the file system model

Wrapping Up

Wrapping Up

- HPC storage is a complex hardware/software system
- Principles used in optimization are applicable outside HPC
  - Aggregation
  - Transformation to match requirements of devices
  - Avoidance of contention
- New storage models are being developed
  - Exposing concurrency in storage system

- Generally HPC storage evolution has been slow, needs to catch up with data intensive storage systems!

Argonne NATIONAL LABORATORY

In-System Storage

Many thanks to:

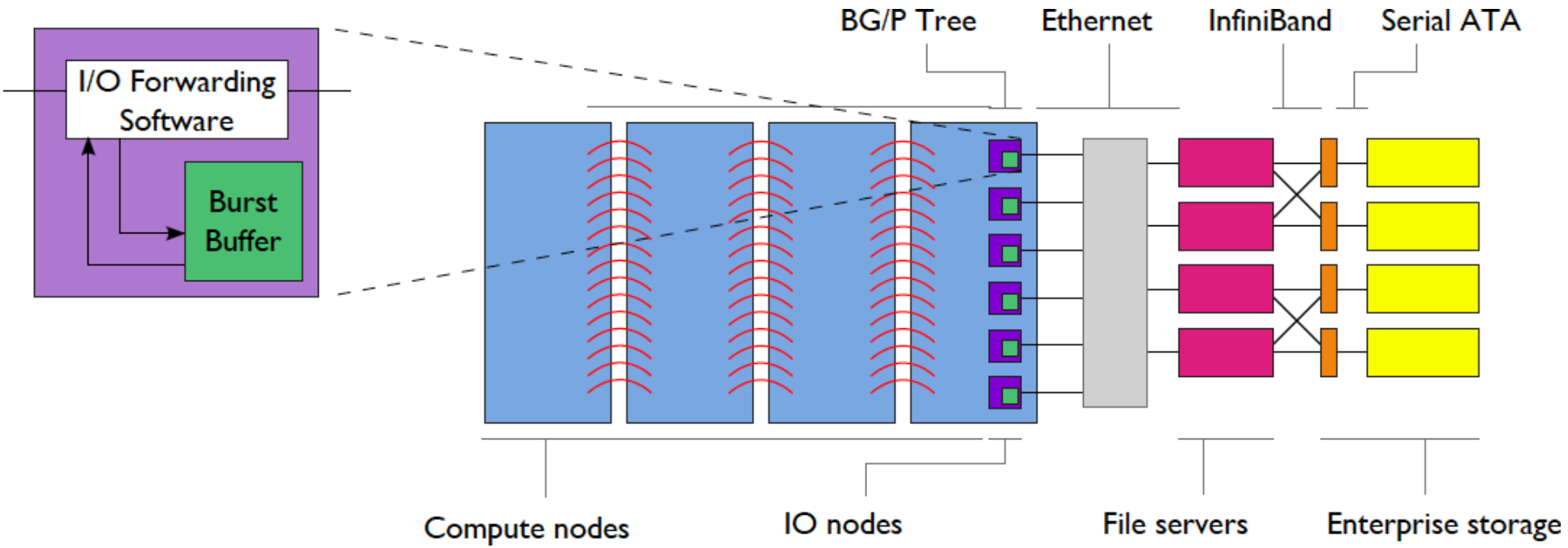
Ning Liu
Illinois Institute of Technology

Jason Cope
DataDirect Networks

Chris Carothers
Rensselear Polytechnic Institute

Adding In System Storage to the Storage Model

The inclusion of NVRAM storage in future systems is a compelling way to deal with the burstiness of I/O in HPC systems, reducing the peak I/O requirements for external storage. In this case the NVRAM is called a "burst buffer".

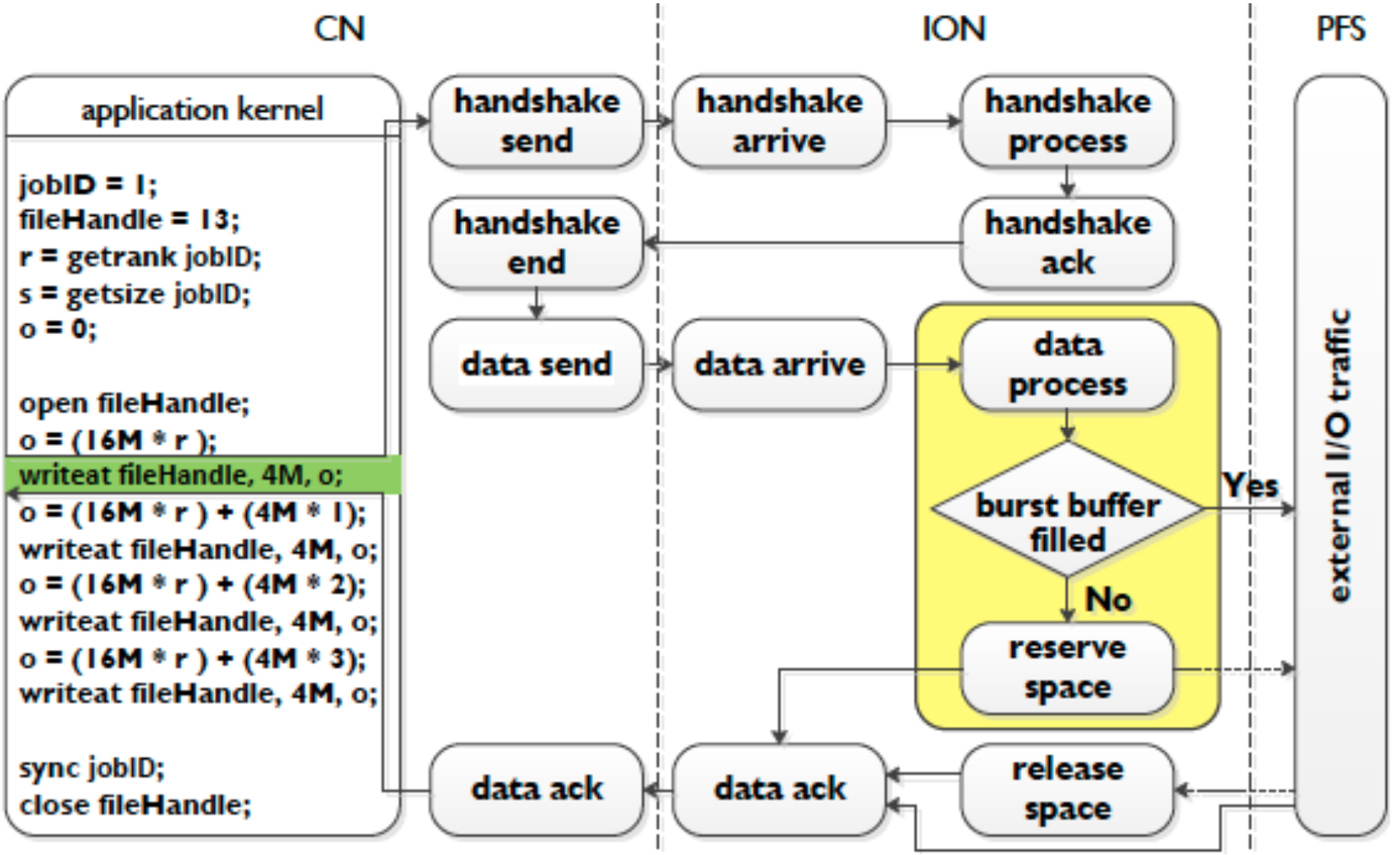


What's a Burst?

- We quantified the I/O behavior by analyzing one month of production I/O activity on Blue Gene/P from December 2011
  - Application-level access pattern information with per process and per file granularity
  - Adequate to provide estimate of I/O bursts

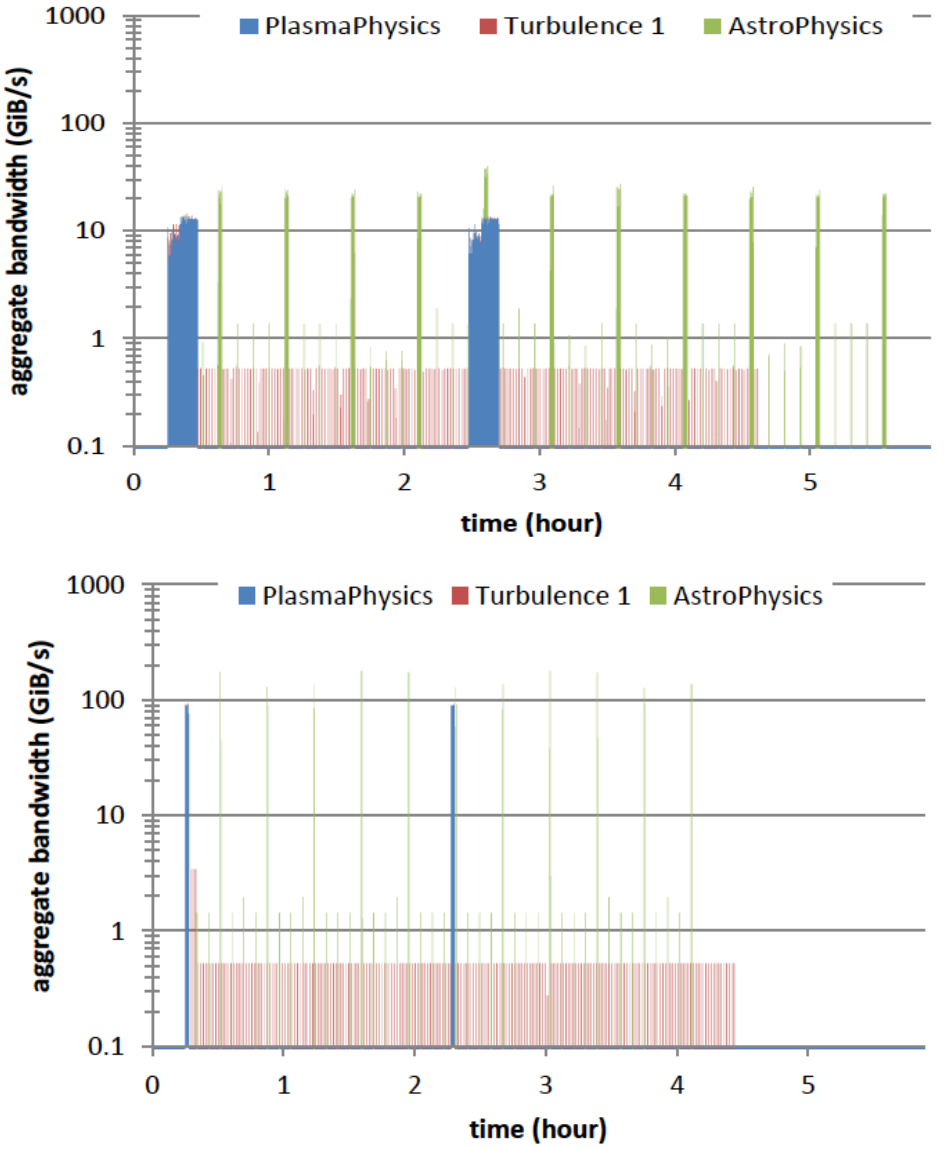
Project	Procs	Nodes	Total Written	Run Time (hours)	Avg. Size and Subsequent Idle Time for Write Bursts>1 GiB				
					Count	Size	Size/Node	Size/ION	Idle Time (sec)
PlasmaPhysics	131,072	32,768	67.0 TiB	10.4	1	33.5 TiB	1.0 GiB	67.0 GiB	7554
					1	33.5 TiB	1.0 GiB	67.0 GiB	end of job
Turbulence1	131,072	32,768	8.9 TiB	11.5	5	128.2 GiB	4.0 MiB	256.4 MiB	70
					1	128.2 GiB	4.0 MiB	256.4 MiB	end of job
					421	19.6 GiB	627.2 KiB	39.2 MiB	70
AstroPhysics	32,768	8,096	8.8 TiB	17.7	1	550.9 GiB	68.9 MiB	4.3 GiB	end of job
					8	423.4 GiB	52.9 MiB	3.3 GiB	240
					37	131.5 GiB	16.4 MiB	1.0 GiB	322
					140	1.6 GiB	204.8 KiB	12.8 MiB	318
Turbulence2	4,096	4,096	5.1 TiB	11.6	21	235.8 GiB	59.0 MiB	3.7 GiB	1.2
					1	235.8 GiB	59.0 MiB	3.7 GiB	end of job

Studying Burst Buffers with Parallel Discrete Event Simulation



Burst Buffers Work for Multi-application Workloads

- Burst buffers improve application perceived throughput under mixed I/O workloads.
- Applications' time to solution decrease with burst buffers enabled (from 5.5 to 4.4 hours)
- Peak bandwidth of the external I/O system may be reduced by 50% without a perceived change on the application side
- Tool for co-design



Application perceived I/O rates, with no burst buffer (top), burst buffer (bottom).

Beyond Burst Buffers

- Obviously lots of other potential uses
  - Checkpointing location
  - Out-of-core computation
  - Holding area for analysis data (e.g., temporal analysis, in situ)
  - Code coupling
  - Input data staging
  - …
- Improves memory capacity of systems
  - More data intensive applications?

- Placement of NVRAM will matter
  - On I/O forwarding nodes (as in our example)
  - On some/all compute nodes?