HPC I/O for Computational Scientists: I/O Transformations

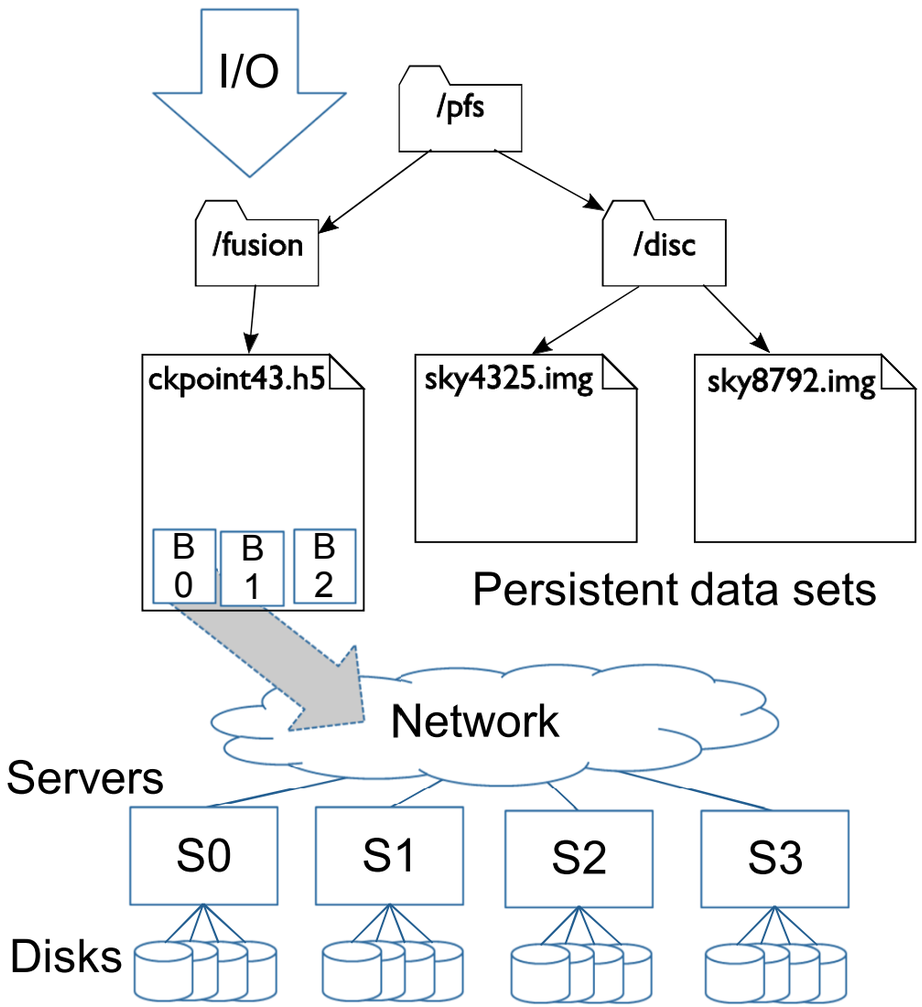
Presented to
ATPESC 2017 Participants

Rob Latham and Phil Carns
Mathematics and Computer Science Division
Argonne National Laboratory

Q Center, St. Charles, IL (USA)
8/4/2017

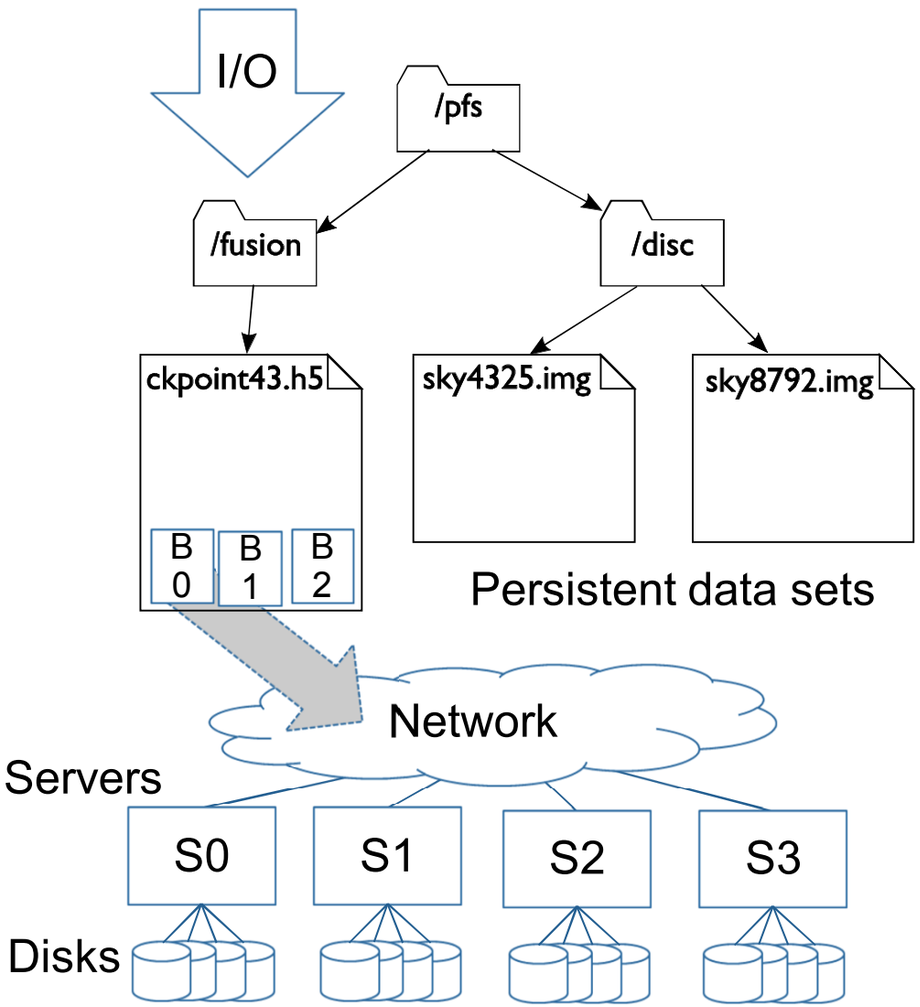
NNSA National Nuclear Security Administration
U.S. DEPARTMENT OF ENERGY | Office of Science
Argonne NATIONAL LABORATORY

First some background:
How flat files are stored on many servers and disks



- Single file example: ckpoint43.h5
- File is split (under the covers) into multiple blocks
- Those blocks are then striped across a subset of servers
- Each server then stores it's block on a collection of disks
- Most optimizations focus on making better use of servers in parallel

Policy details for striping files



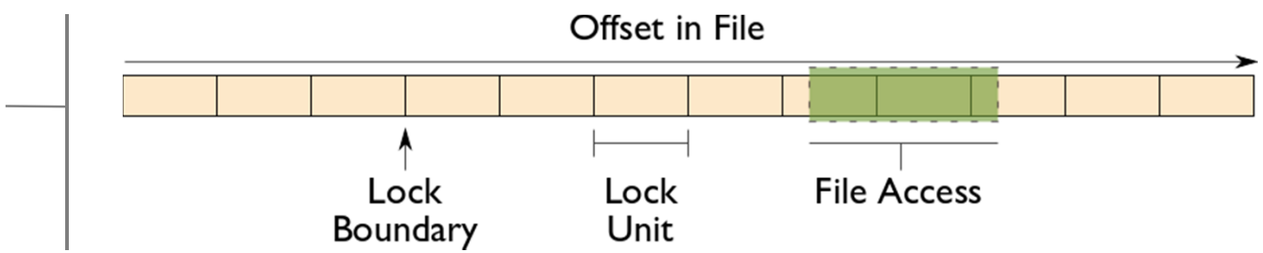
- Mira: happens automatically, and large files will use every server
- Theta/Cori: by default, each file will be stored on a single server
  - You can tune this setting for large files that will be accessed in parallel
  - See Darshan hands-on scripts (later today) for examples
  - Example: "lfs setstripe -c -1" on a directory to widely stripe all new files in directory

Managing Concurrent Access

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

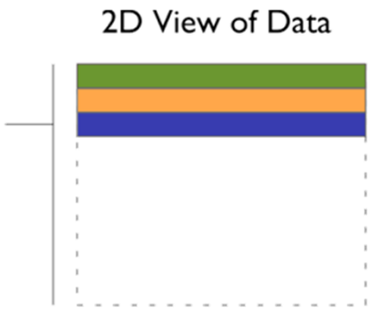
- Files are broken up into lock units
  - Unit boundaries are dictated by the storage system, regardless of access pattern
- Clients obtain locks on units that they will access before I/O occurs
- Enables caching on clients as well (as long as client has a lock, it knows its cached data is valid)
- Locks 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.



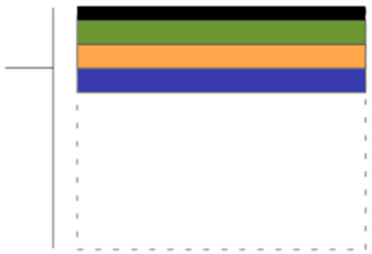
Implications of Locking in Concurrent Access

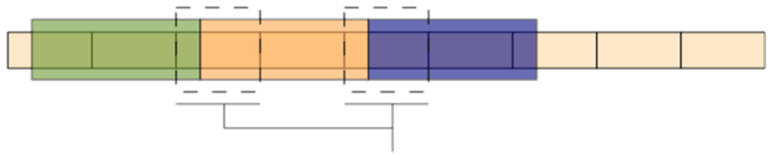
The left diagram shows a row-block distribution of data for three processes. On the right we see how these accesses map onto locking units in the file.



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

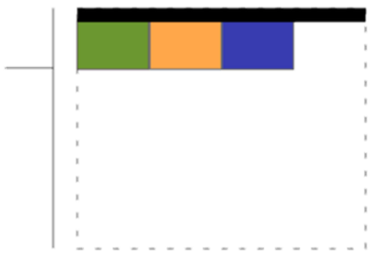
In this example a header (black) has been prepended to the data. If the header is not aligned with lock boundaries, false sharing will occur.

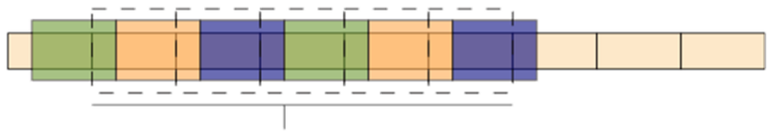




These two regions exhibit *false sharing*: no bytes are accessed by both processes, but because each block is accessed by more than one process, there is contention for locks.

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



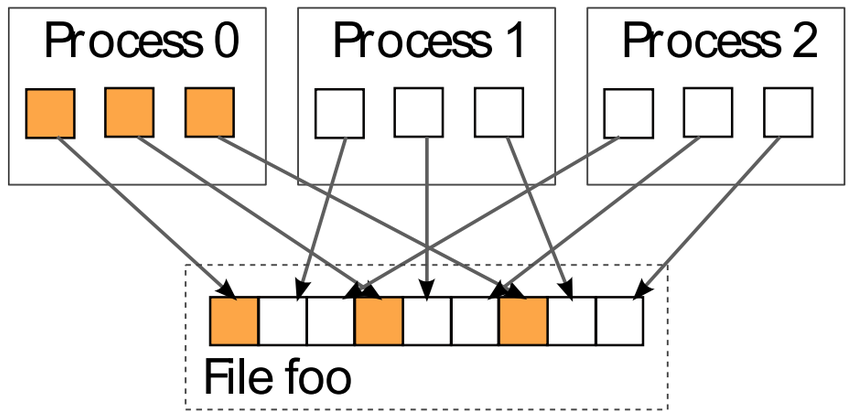


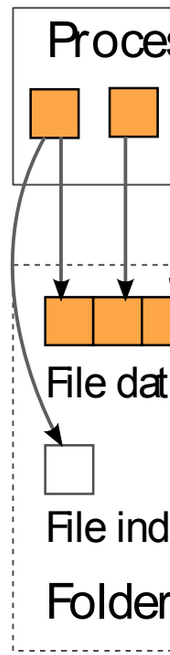
When a block distribution is used, sub-rows cause a higher degree of false sharing, especially if data is not aligned with lock boundaries.

I/O Transformations

Software between the application and the file system 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 as it would have been normally
  - i.e., the file system is still aware of the actual data organization



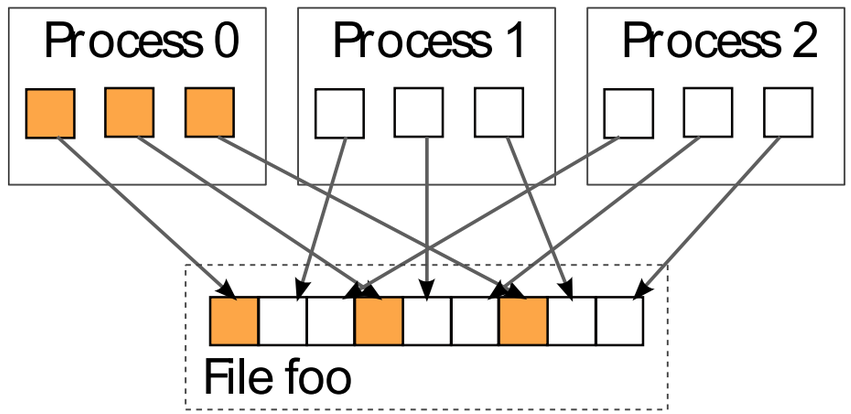


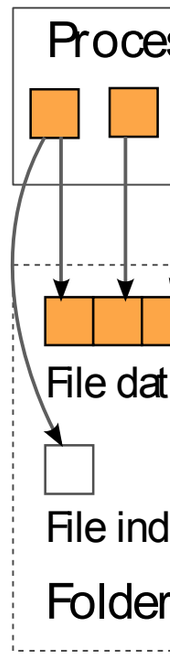
When we think about I/O transformations, we consider the mapping of data between application processes and locations in file.

I/O Transformations

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

- We will tour through a few examples of data transformations in the following slides
- The important thing to remember is that software already exists to do these things for you in HDF5, PnetCDF, ADIOS, and MPI-IO
- If you find yourself replicating these optimizations by hand, look around to see if you can find an off-the-shelf solution





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

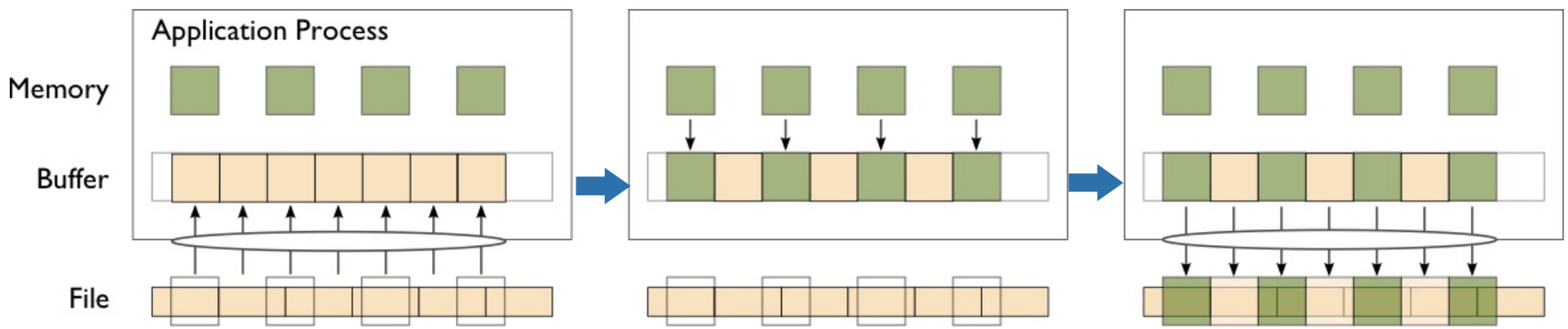
Argonne NATIONAL LABORATORY

ECP EXASCALE COMPUTING PROJECT

Reducing Number of Operations

Because most operations go over multiple networks, 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)
- Somewhat counter-intuitive: do extra I/O to avoid contention



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

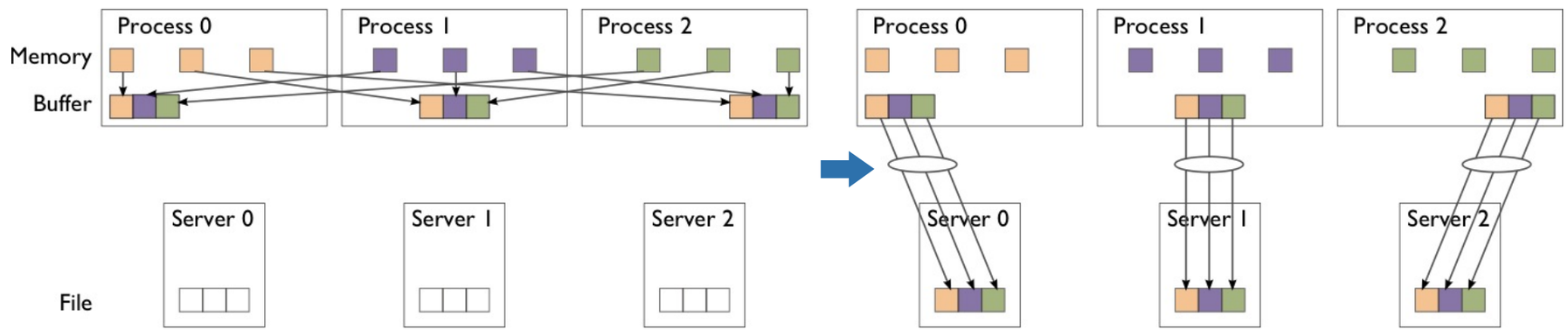
Step 2: Elements to be written to file are replaced in intermediate buffer.

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

Avoiding Lock Contention

We can reorder data among processes to avoid lock contention. *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.

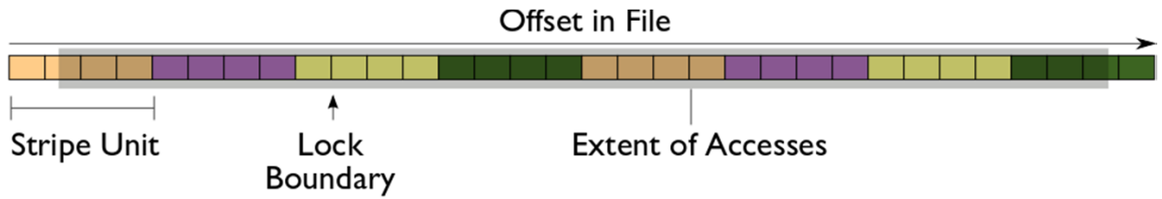
Phase 2: Data are written to file (storage servers) with large writes, no contention.

Two-Phase I/O Algorithms

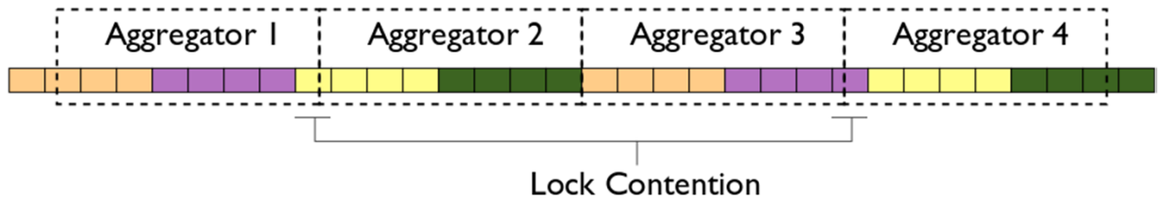
(or, You don't want to do this yourself…)

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.

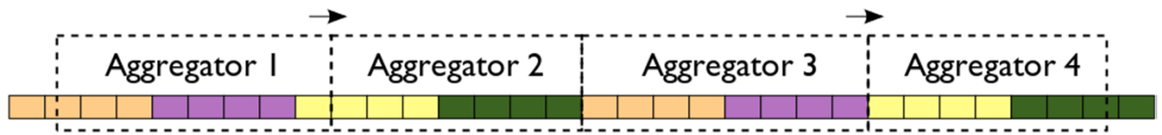
Imagine a collective I/O access using four aggregators to a file striped over four file servers (indicated by colors):



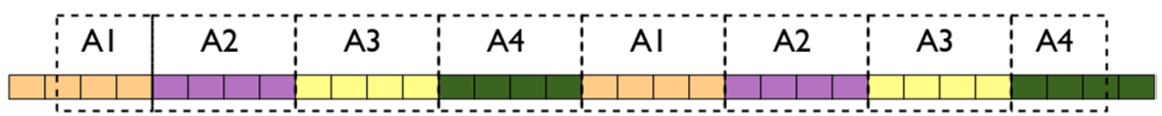
One approach is to evenly divide the region accessed across aggregators.



Aligning regions with lock boundaries eliminates lock contention.



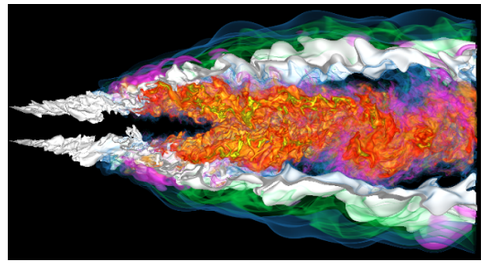
Mapping aggregators to servers reduces the number of concurrent operations on a single server and can be helpful when locks are handed out on a per-server basis (e.g., Lustre).

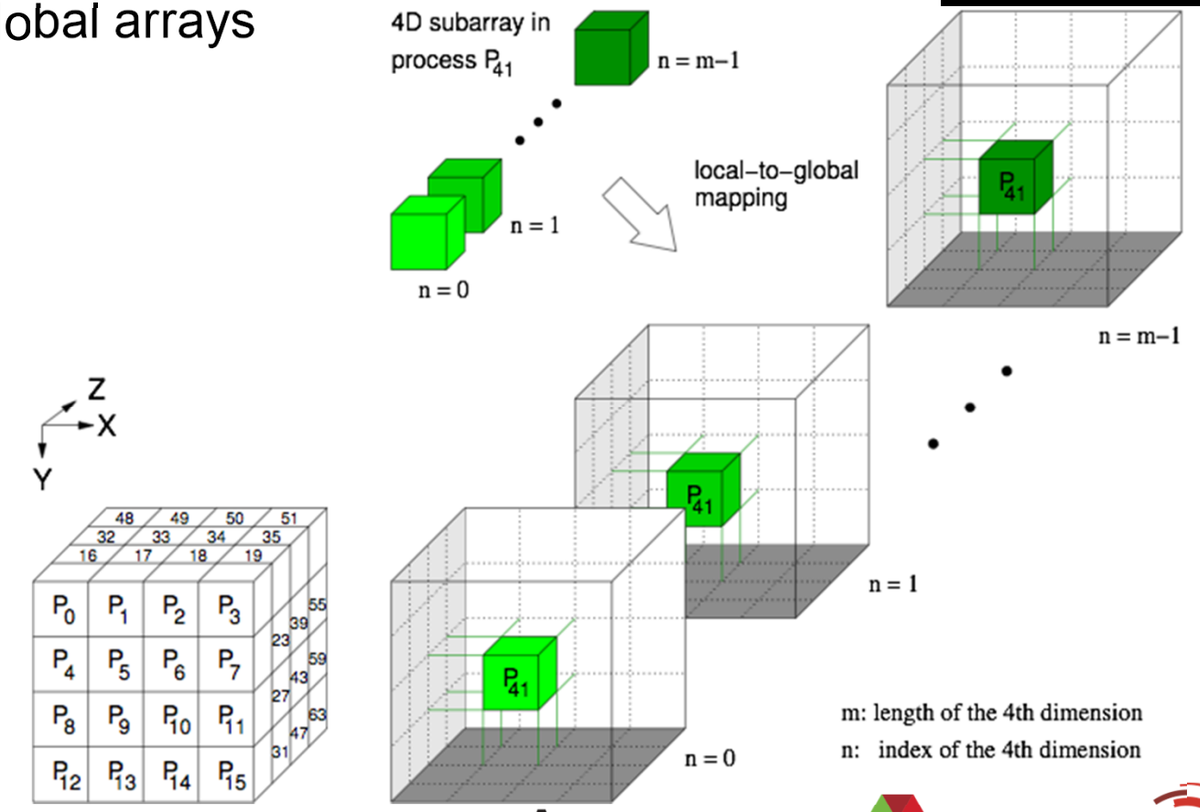


Today's systems also choose aggregators that are "closest" to storage

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.

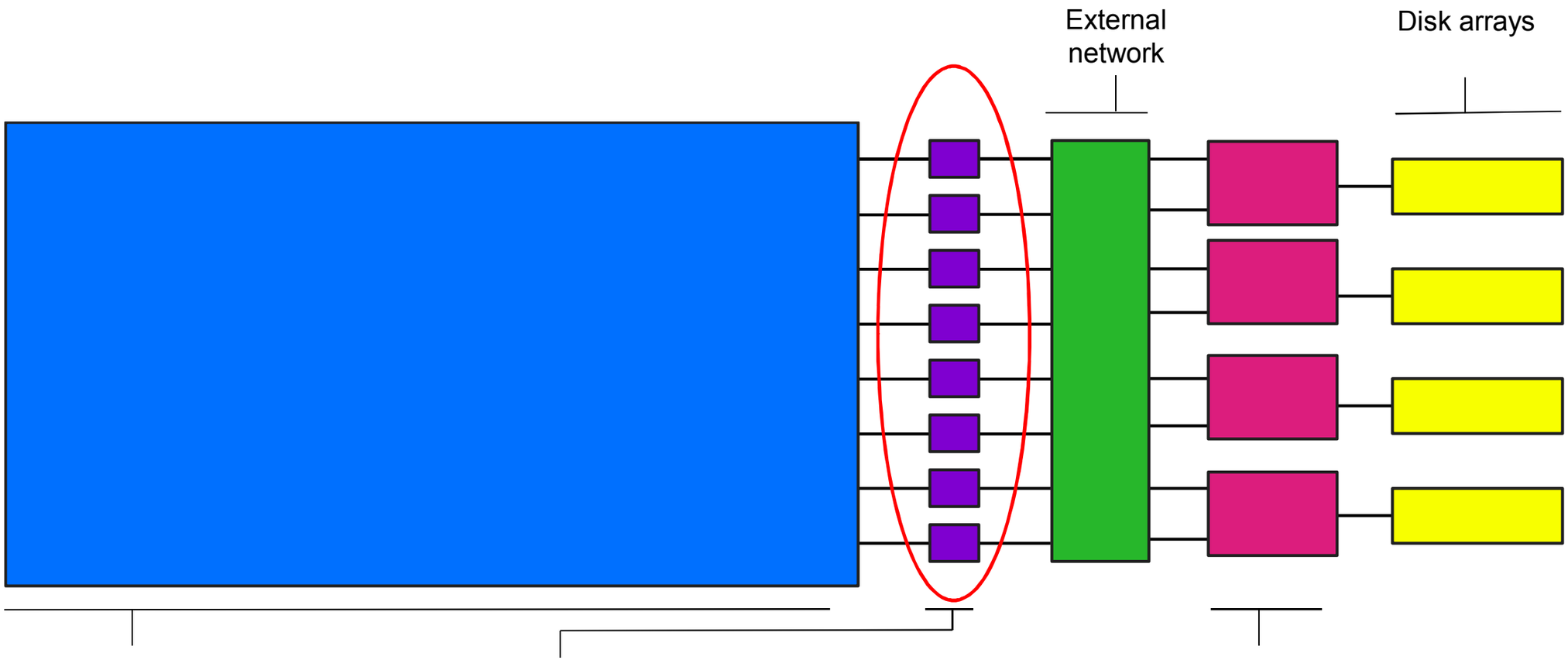
Impact of Transformations on S3D I/O

- 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

Application did the same thing in every case

	Coll. Buffering and Data Sieving Disabled	Data Sieving Enabled	Coll. Buffering Enabled (including 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



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 (e.g., on Mira) 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
  - Transparently transform data on its way to and from the file system
- On Theta, Lnet routers fill a similar role
  - Bridge networks
  - Shape and route I/O traffic for storage system

Transformations in the I/O Forwarding Step

The transformations can take many forms:

- Performing one file open on behalf of many processes
- Combining small accesses into larger ones
- Caching data
- Redirecting requests through shorter network routes

“Not So Transparent” Transformations

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

- Observation: 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
- Or alternatively, data could be stored in a different type of storage system entirely (not a file system)

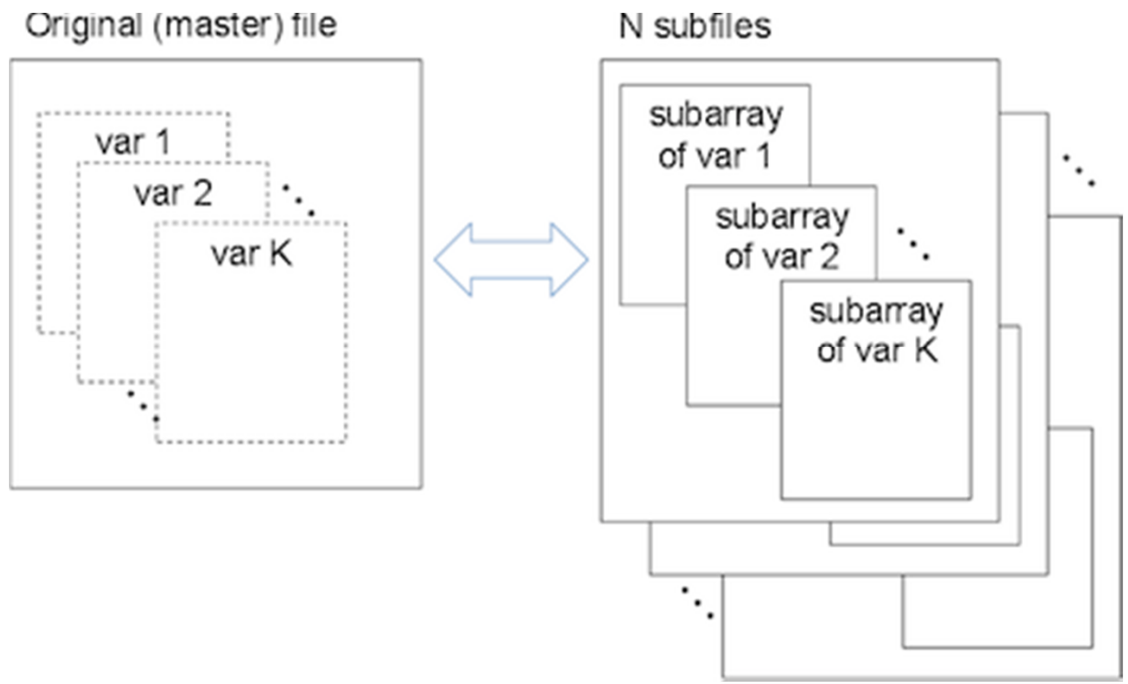
“Not So Transparent” Transformations

Example: PnetCDF subfiling

- Translates a single data set into multiple underlying files

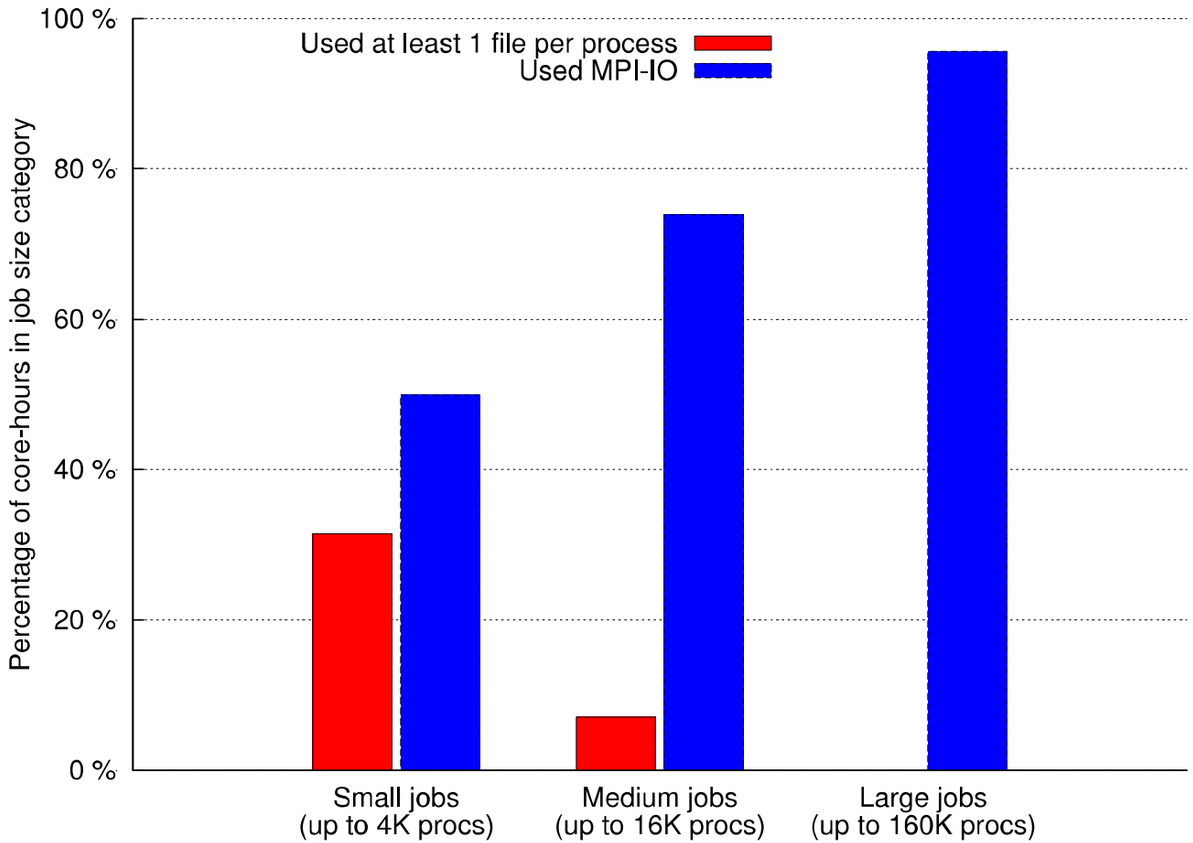
Example: HDF5 vol plugins

- Abstraction layer that can map an HDF5 data set to multiple files or even to completely different storage targets



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



Sometimes this is the fastest strategy, but becomes increasingly hard to sustain at scale.

I/O transformation summary

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
  - Future designs might provide a different underlying model

Takeaways

- Parallel file systems provide the underpinnings of HPC I/O solutions
- Data model libraries provide alternative data models for applications
  - PnetCDF and HDF5 will both be discussed in detail later in the day
- Characteristics of PFSes lead to the need for transformations in order to achieve high performance
  - Implemented in a number of different software layers
  - Some preserving file organization, others breaking it
- The down side: proliferation of layers complicates performance debugging
  - We'll address this topic later in the day

Next up!

- This presentation provided an overview of transformations in the HPC I/O stack
- The next presentation will walk through an example application case study for a first-hand look at how to program for HPC I/O