ATPESC 2016

HPC TRANSFORMATIONS: OPTIMIZING DATA SO YOU DON’T HAVE TO

ROB LATHAM
PHIL CARNS
Argonne National Laboratory
robl@mcs.anl.gov
carns@mcs.anl.gov

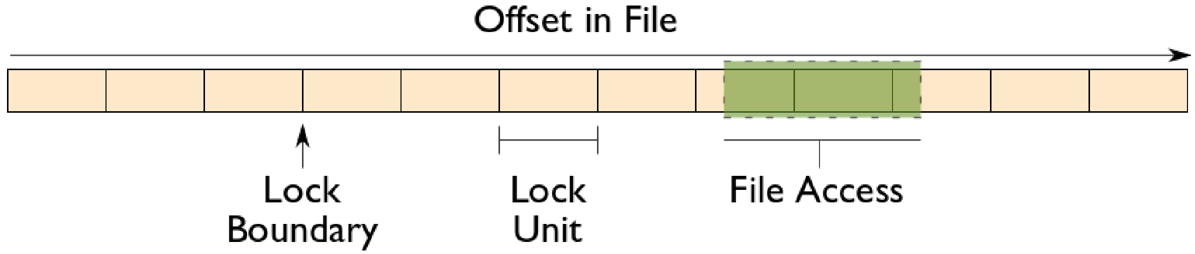
August 11, 2016
St. Charles IL

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.

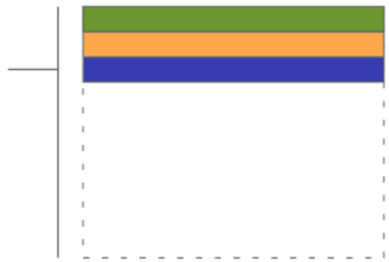


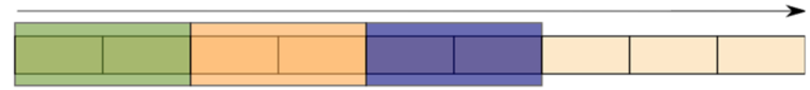
IMPLICATIONS OF LOCKING IN CONCURRENT ACCESS

2D View of Data

Offset in File

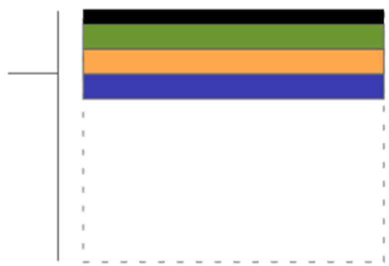
The left diagram shows a row-block distribution of data for three processes. On the right we see how these accesses map onto locking units in the file.

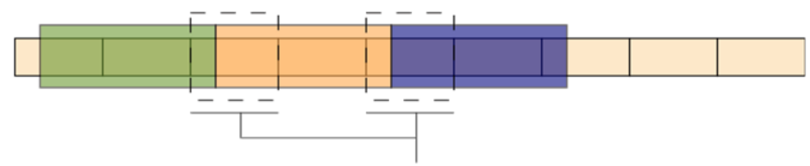




When accesses are to large contiguous regions, and aligned with lock boundaries, locking overhead is minimal.

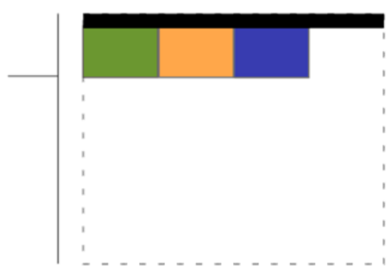
In this example a header (black) has been prepended to the data. If the header is not aligned with lock boundaries, false sharing will occur.

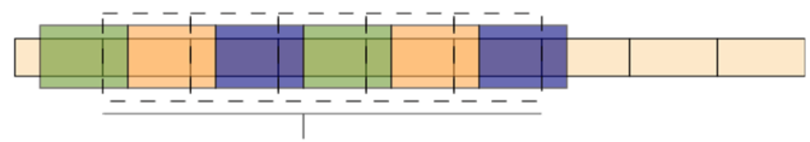




These two regions exhibit *false sharing*: no bytes are accessed by both processes, but because each block is accessed by more than one process, there is contention for locks.

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



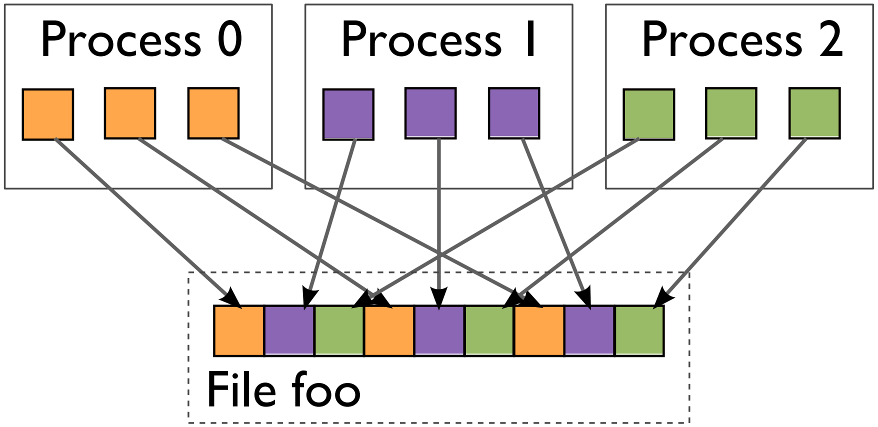


When a block distribution is used, sub-rows cause a higher degree of false sharing, especially if data is not aligned with lock boundaries.

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

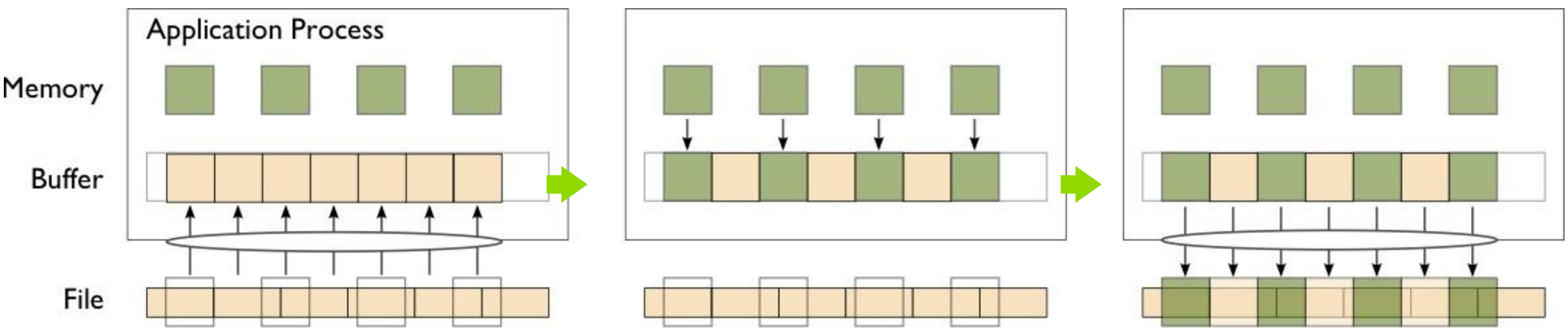


When we think about I/O transformations, we consider the mapping of data between application processes and locations in file.

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)
- 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).

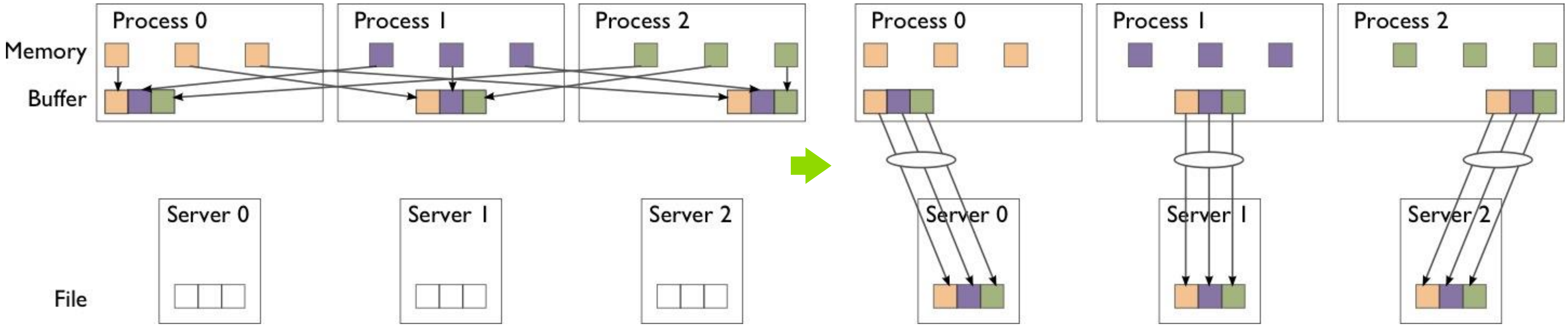
Step 2: Elements to be written to file are replaced in intermediate buffer.

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

AVOIDING LOCK CONTENTION

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.

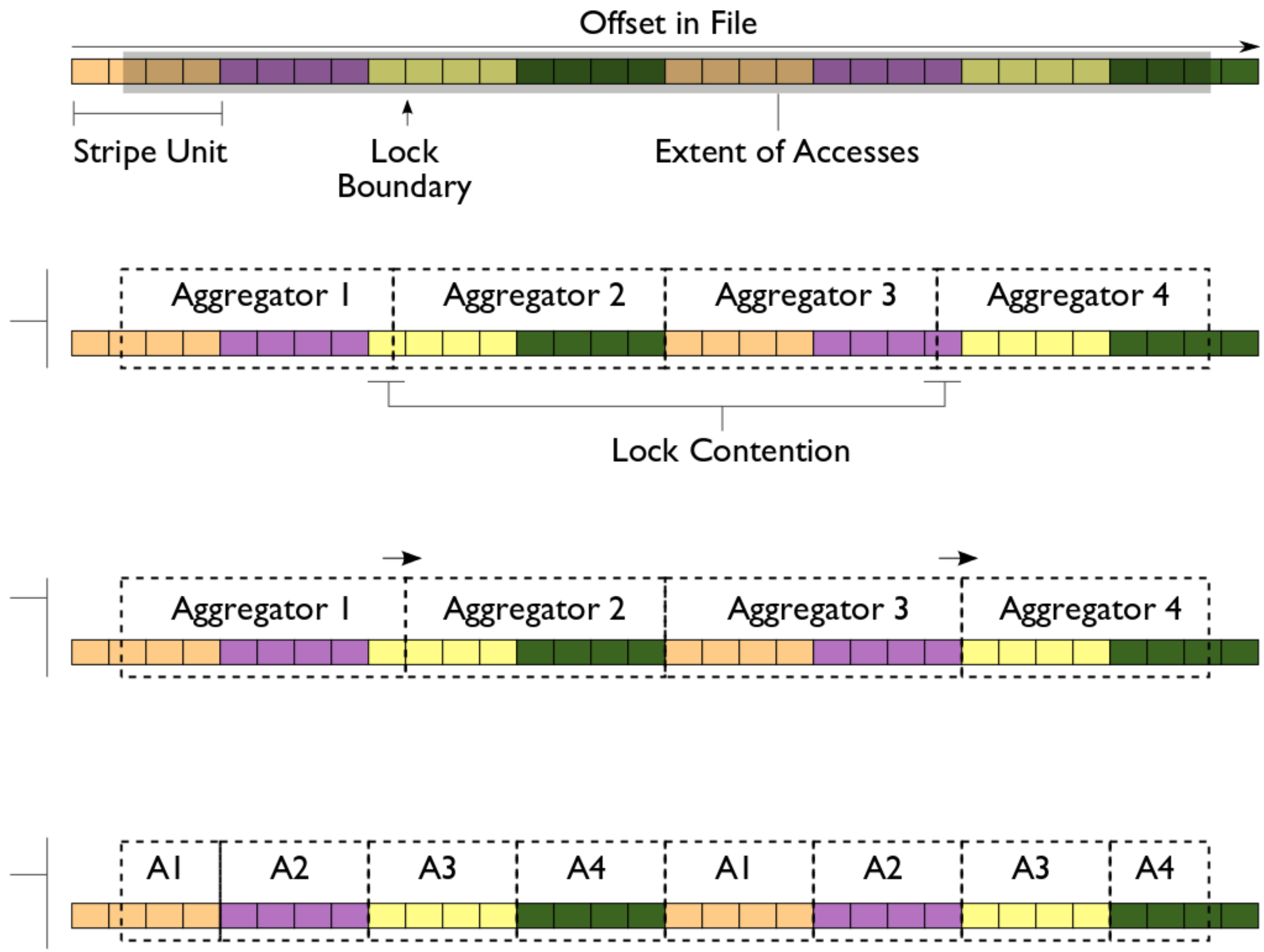
TWO-PHASE I/O ALGORITHMS
(OR, YOU DON'T WANT TO DO THIS YOURSELF…)

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.

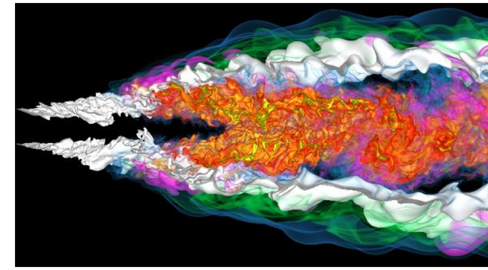
Aligning regions with lock boundaries eliminates lock contention.

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).

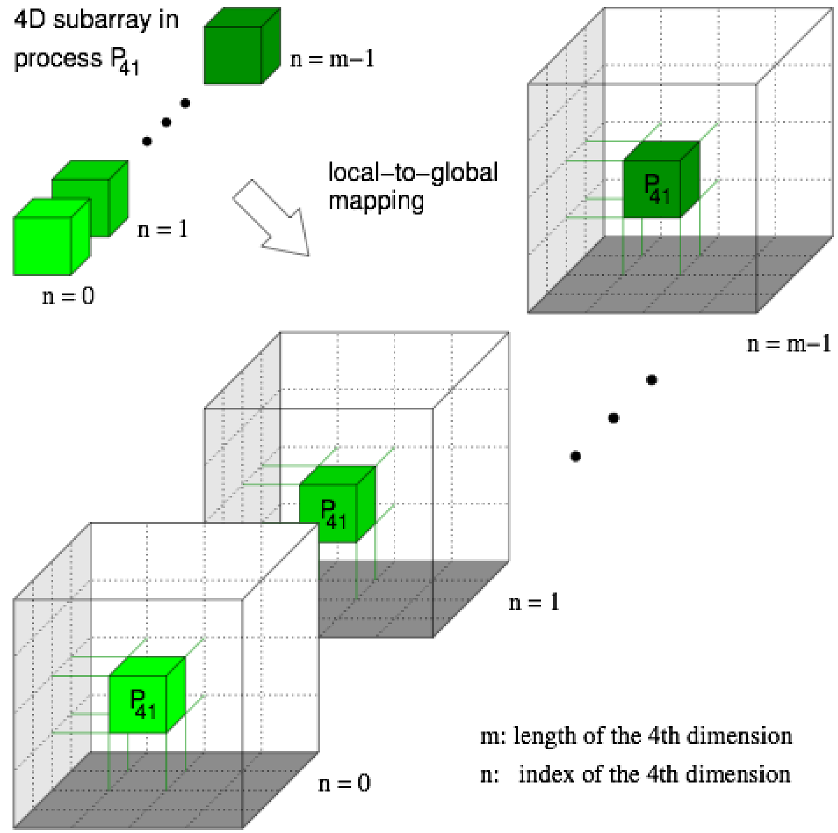


For more information, see W.K. Liao and A. Choudhary, "Dynamically Adapting File Domain Partitioning Methods for Collective I/O Based on Underlying Parallel File System Locking Protocols," SC2008, November, 2008.

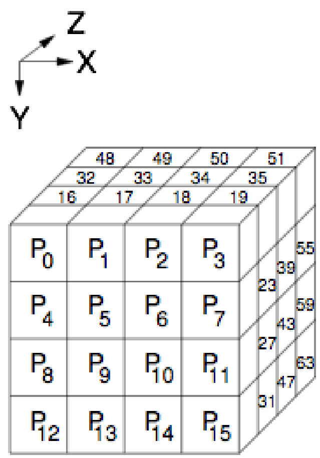
S3D TURBULENT COMBUSTION CODE



- 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.

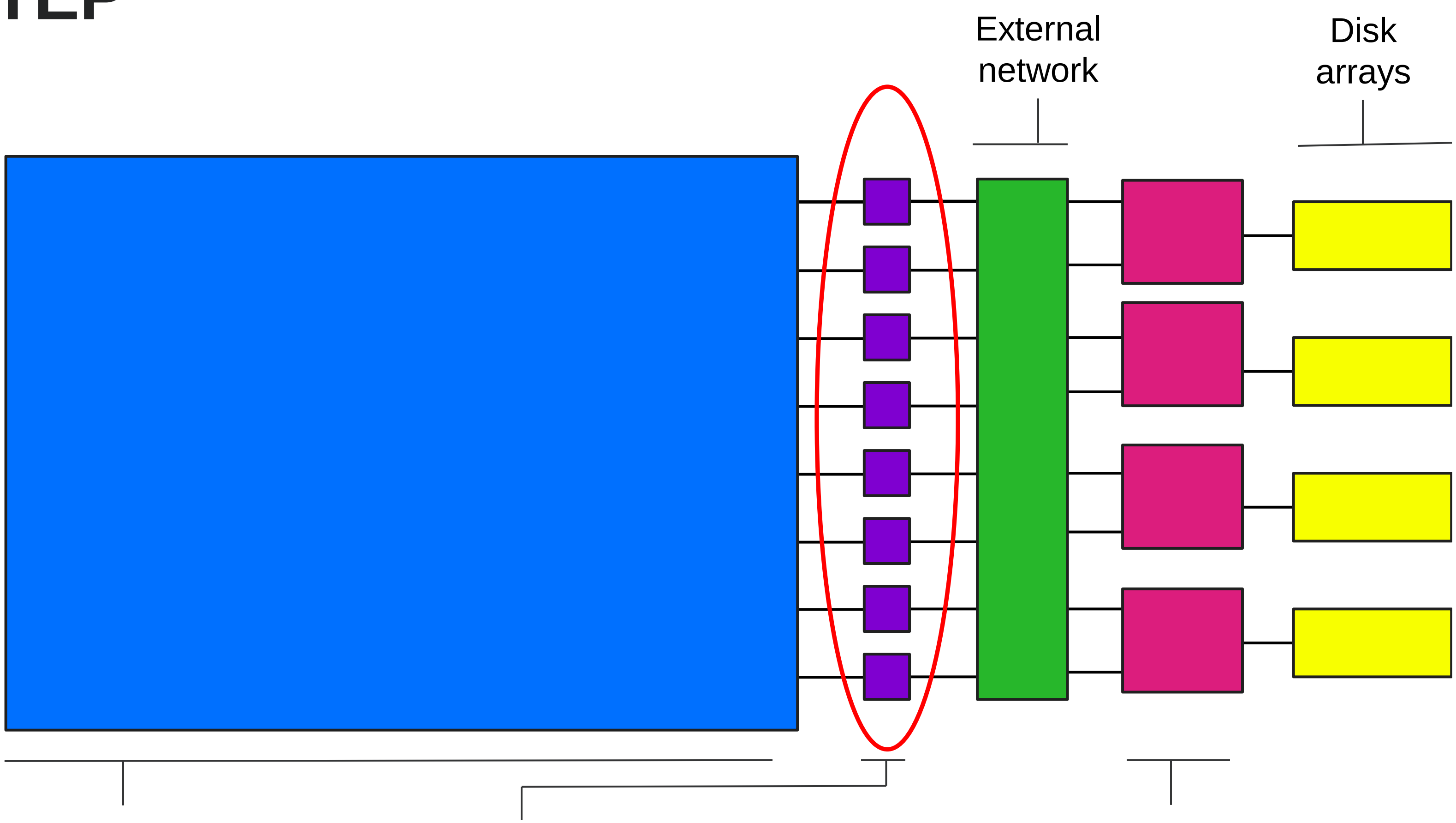


IMPACT OF TRANSFORMATIONS ON S3D I/O

- 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



Compute nodes

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

Storage nodes

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
  - Transparently transform data on its way to and from the file system

- 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

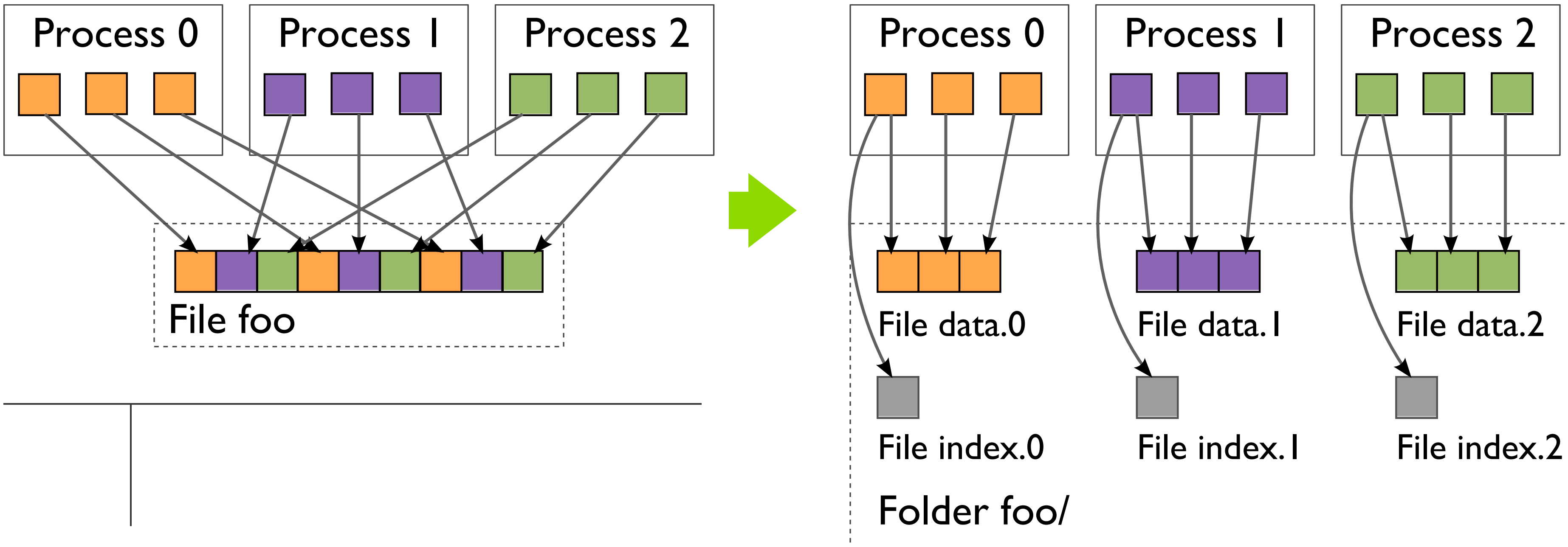
"NOT SO TRANSPARENT" TRANSFORMATIONS

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
  - It is transparent from the application/user perspective (it presents a virtual view of the data) but not from the storage system perspective

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.

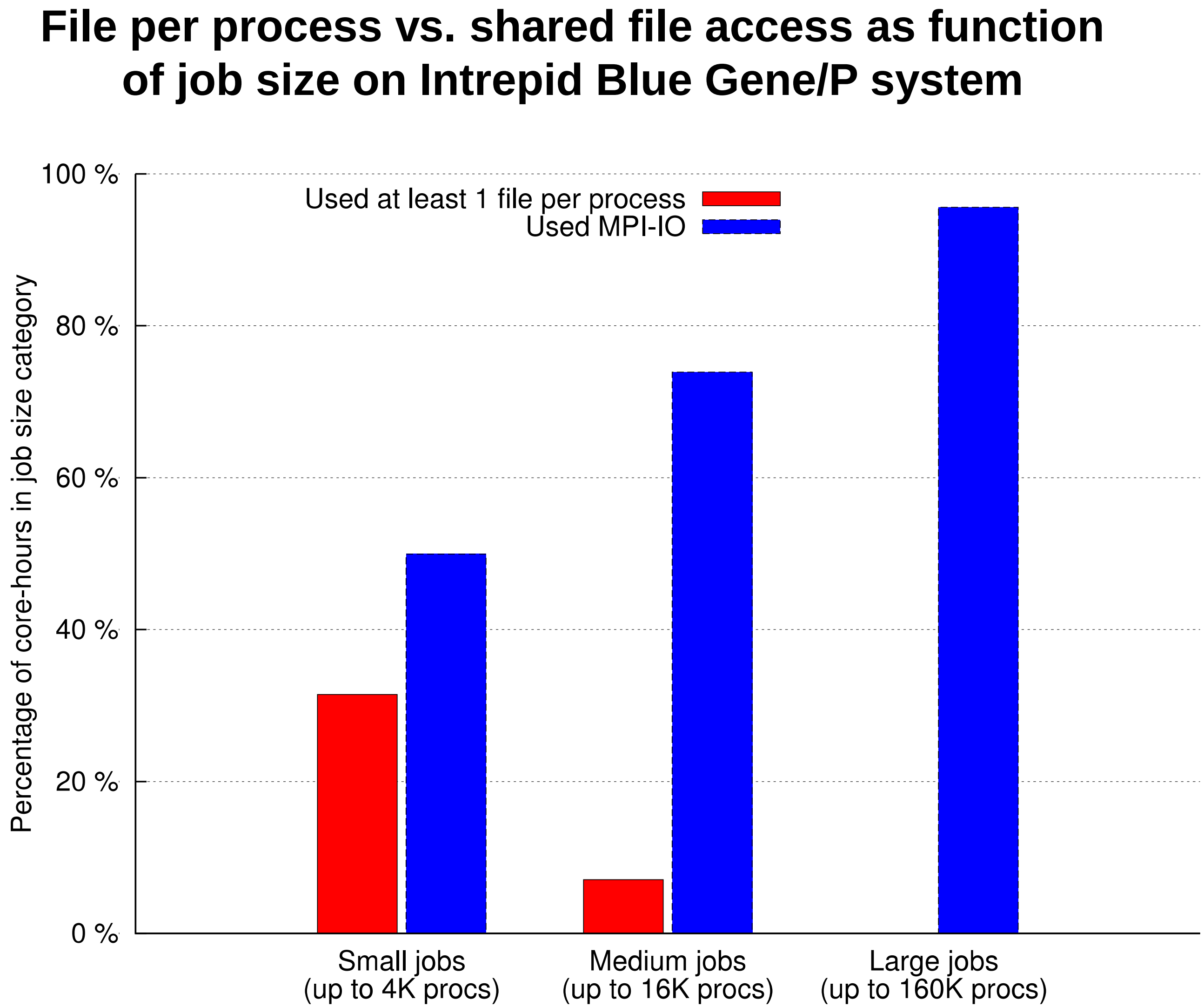
Transparent transformations such as data sieving and two-phase I/O preserve data order on the file system.

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?



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
  - Future designs might provide a different underlying model

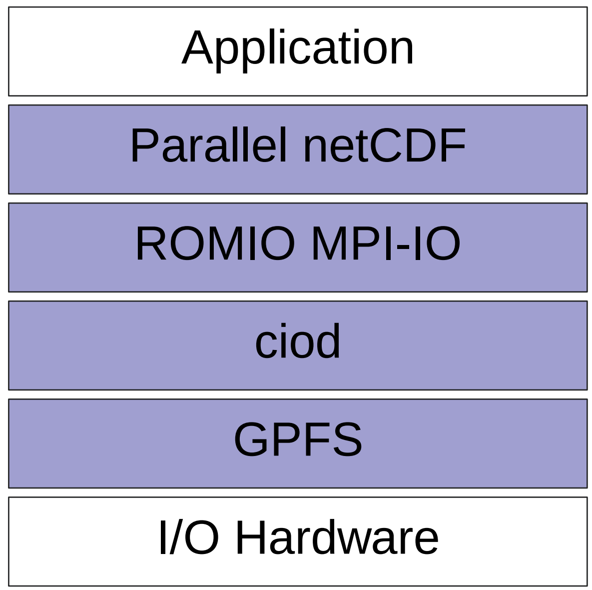
HOW IT WORKS: TODAY’S I/O SYSTEMS

AN EXAMPLE HPC I/O SOFTWARE STACK

This example I/O stack captures the software stack used in some applications on the IBM Blue Gene/Q system at Argonne.

Parallel netCDF is used in numerous climate and weather applications running on DOE systems.
Built in collaboration with NWU.

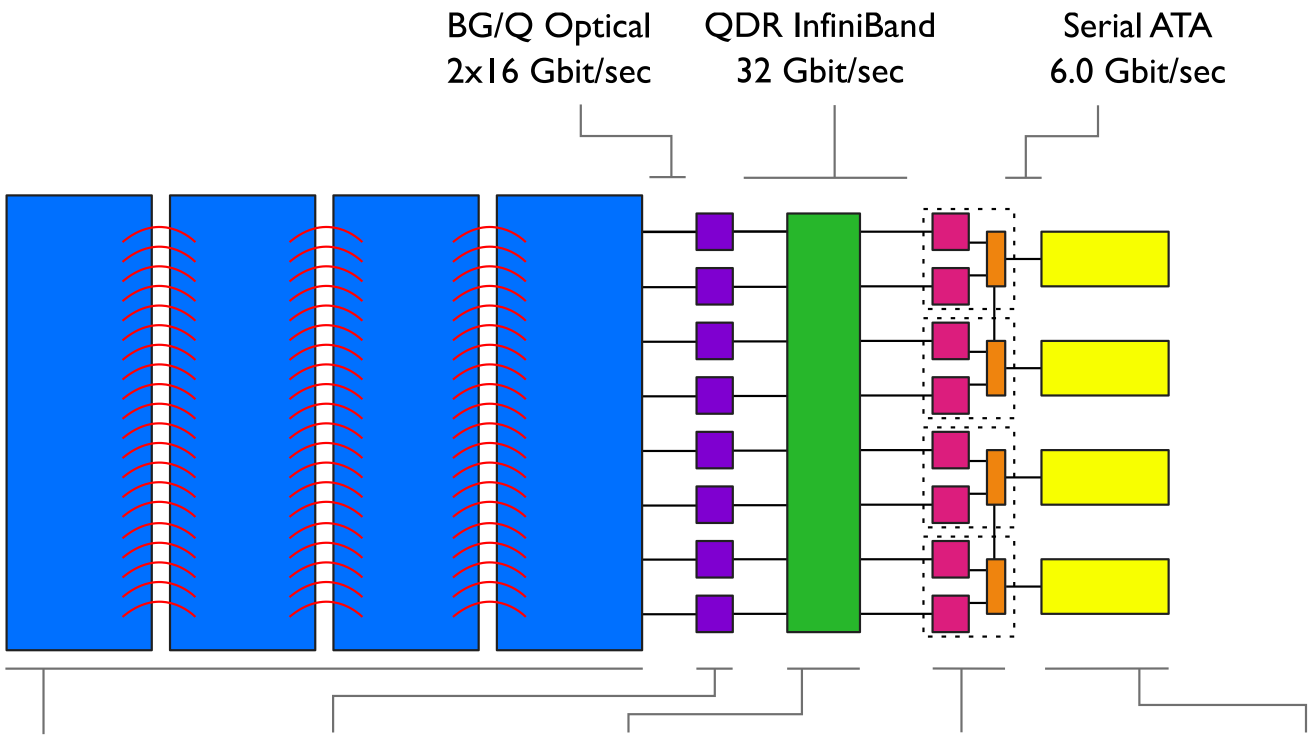
ciod is the I/O forwarding implementation on the IBM Blue Gene/P and Blue Gene/Q systems.



ROMIO is the basis for virtually all MPI-IO implementations on all platforms today and the starting point for nearly all MPI-IO research.
Incorporates research from NWU and patches from vendors.

GPFS is a production parallel file system provided by IBM.

MIRA BLUE GENE/Q AND ITS STORAGE SYSTEM



Compute nodes run applications and some I/O middleware.

768K cores with 1 Gbyte of RAM each

Gateway nodes run parallel file system client software and forward I/O operations from HPC clients.

384 16-core PowerPC A2 nodes with 16 Gbytes of RAM each

Commodity network primarily carries storage traffic.

QDR Infiniband Federated Switch

Storage nodes run parallel file system software and manage incoming FS traffic from gateway nodes.

SFA12KE hosts VM running GPFS servers

Enterprise storage controllers and large racks of disks are connected via InfiniBand.

32 DataDirect SFA12KE; 560 3 Tbyte drives + 32 200 GB SSD; 16 InfiniBand ports per pair

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

- Number of layers complicates performance debugging
  - Some ways of approaching this discussed later in the day