Title:	Lustre and PLFS Parallel I/O Performance on a Cray XE6
Author(s):	Kettering, Brett M. Torrez, Alfred Bonnie, David J. Shrader, David L.
Intended for:	CUG 2014, 2014-05-04/2014-05-08 (Lugano, Switzerland)
Issued:	2014-05-15 (rev.2)

Lustre and PLFS Parallel I/O Performance on a Cray XE6

Cray User Group 2014

Lugano, Switzerland

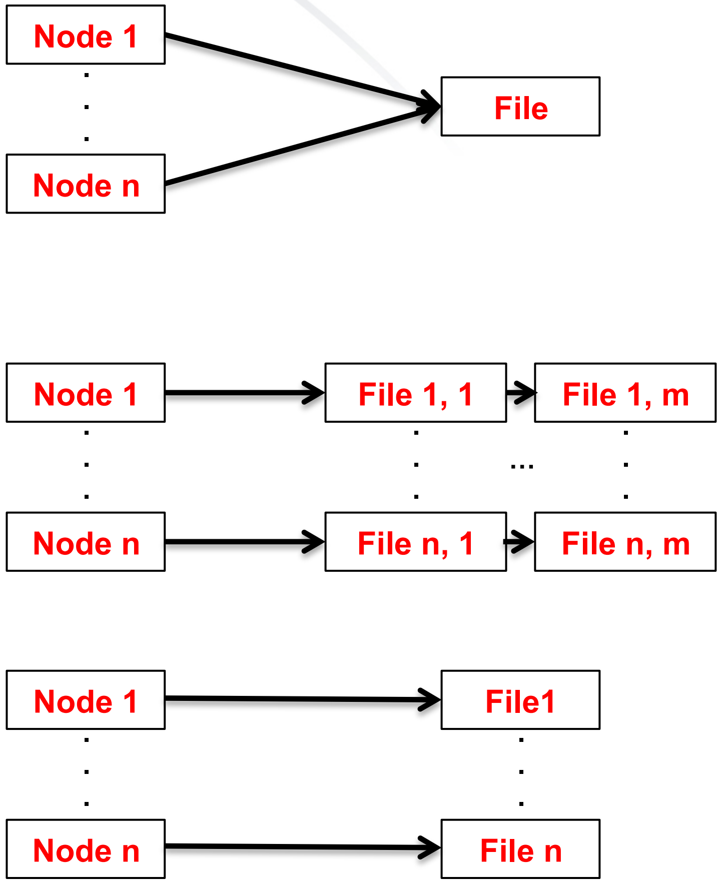
May 4 - 8, 2014

Many currently contributing to PLFS

- LANL: David Bonnie, Aaron Caldwell, Gary Grider, Brett Kettering, David Shrader, Alfred Torrez
  - Past Contributors: Ben McClelland, Meghan McClelland, James Nuñez, Aaron Torres
- EMC: John Bent & The EMC Engineering Team
- Carnegie Mellon University: Chuck Cranor
- Other Academics: Jun He (UW-Madison), Kshitij Mehta (U. Houston)
- Cray: William Tucker and the MPI Team
- Get it at https://github.com/plfs

Two of the three primary I/O models really benefit from PLFS

- Shared File, N-1
  - Strided: Data is organized by type
  - Segmented: Data is organized by pe
- Small File, 1-N
  - Each pe to its own set of many small files
- Flat File, N-N
  - Each pe to its own file
  - Distributes files across directories

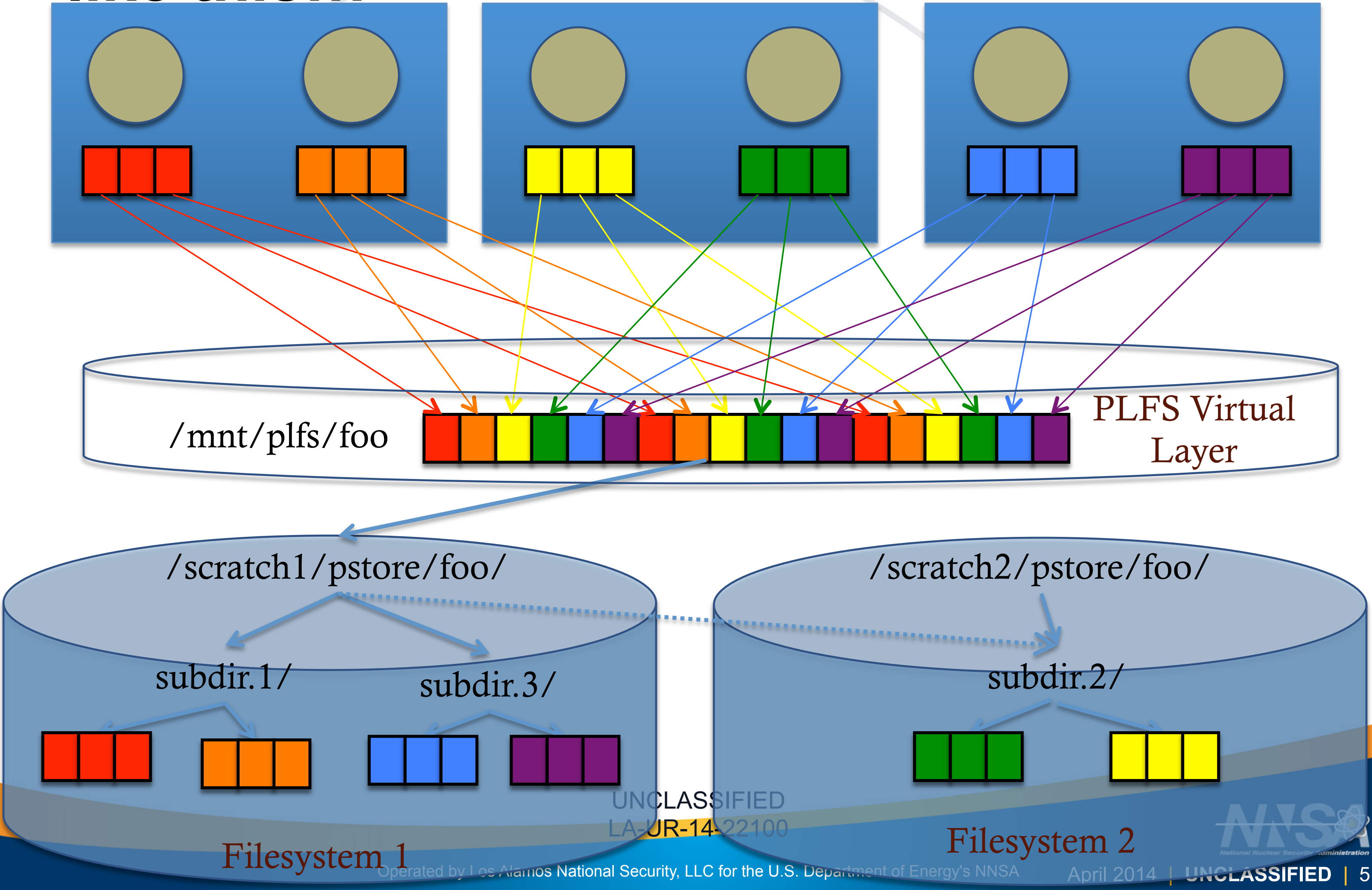


UNCLASSIFIED
LA-UR-14-22100

Shared File (N-1) addresses normal N-1 performance issues

- Issue: Excessive seeking
  - Use log-structured file approach
    - Write data sequentially as it arrives
      - One data file/writer
      - Index files to map physical location to logical location
        - Association with writers/nodes varies – see overhead information
  - Convert N-1 to N-N in PLFS containers
    - One or more backend storage locations (can be on different file systems)
- Issue: Region locking
  - Convert N-1 to N-N in PLFS containers
    - 2 or more writers cannot access the same file in the same data region (e.g. OST on Lustre)
- Consult the index to resolve reads

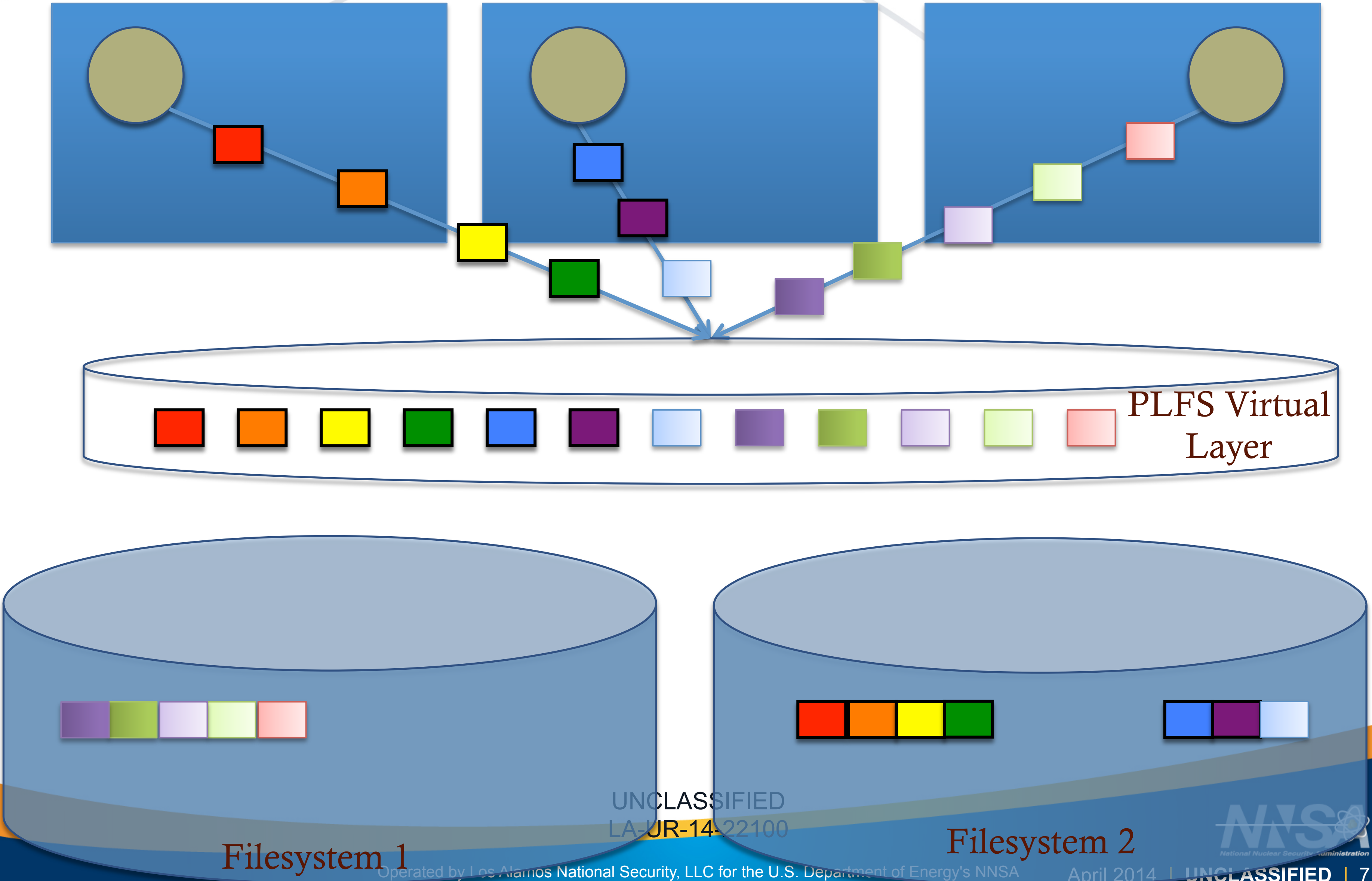
Shared File (N-1 strided) looks like this…



Small File (1-N) combines many, small files into a file per process

- Write multiple files to a single file per writer
  - Create a "PLFS container" on one or more backend file systems
  - Append each write to a log file per writer
  - Create an "index" for each writer to track what went where
- Cache locally for performance
  - Must explicitly flush to storage to provide current data to other nodes
    - Not POSIX compliant in this
- Consult the index to resolve reads

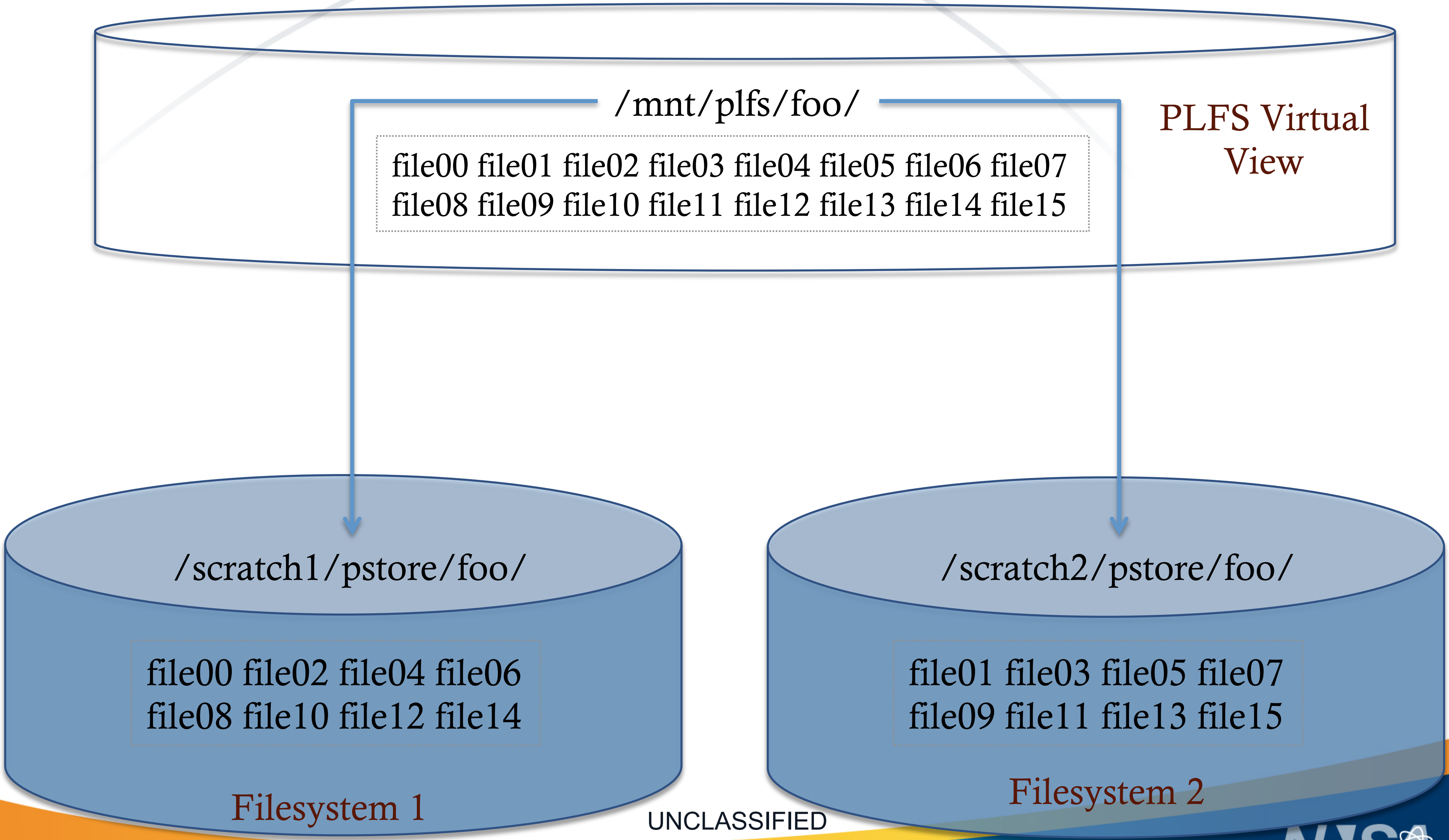
Small File (1-N) looks like this…



Flat File (N-N) distributes the files over multiple directories

- Creates/accesses files in an apparent shared directory
  - Distributes files over multiple directories that are potentially on multiple file systems
  - Constructs a user view that shows the files in the directory to which the application wrote
- Allows underlying file system to distribute the metadata load
  - Lustre can with clustered metadata servers
  - Panasas can with different volumes

Flat File (N-N) looks like this…


/mnt/plfs/foo/
file00 file01 file02 file03 file04 file05 file06 file07
file08 file09 file10 file11 file12 file13 file14 file15
PLFS Virtual View
/scratch1/pstore/foo/
file00 file02 file04 file06
file08 file10 file12 file14
Filesystem 1
/scratch2/pstore/foo/
file01 file03 file05 file07
file09 file11 file13 file15
Filesystem 2

Use one of 3 interfaces depending on your needs

- MPI/IO is the highest performance interface
  - Load a MPI module that has been patched for PLFS
  - Prepend "plfs:" to the file path used in MPI_File_open
  - Mount point must be defined in PLFSRC
- FUSE is primarily intended for interactive command access (e.g. ls, rm, mv, non-PLFS-aware archive tools, etc.)
  - Can use it for POSIX I/O
    - Performance may suffer if not configured for multi-threading
  - Mount point must be defined in PLFSRC and mounted using the PLFS daemon
  - Ensure FUSE buffers are larger than 128 KiB default
- PLFS API is high performance, but requires major programming changes
  - Must change source code to use plfs_* calls to handle the files

N-1 PLFS file overhead depends on interface – MPI/IO

- Recommend 1 backend/parallel file system for N-1
- 1 top-level directory (container)/backend to hold the rest of the files
- 1 hostdir/node/container to hold data and index files
  - PLFSRC num_hostdirs parameter sets limit on maximum hostdirs/top-level directory
- 1 meta directory/container to hold metadata caching files
- 1 ".plfsaccess" file to enable distinguishing PLFS container from logical directory
- 1 metadata file/hostdir to cache "stat" information
- 1 version file to help with backward compatibility checks
- 1 data file/writer process
- 1 index file/writer process

N-1 PLFS file overhead depends on interface – PLFS FUSE

- Recommend 1 backend/parallel file system for N-1
- 1 top-level directory (container)/backend to hold the rest of the files
- 1 hostdir/node/container to hold data and index files
  - PLFSRC num_hostdirs parameter sets limit on maximum hostdirs/top-level directory
- 1 meta directory/container to hold metadata caching files
- 1 ".plfsaccess" file to enable distinguishing PLFS container from logical directory
- 1 version file to help with backward compatibility checks
- 1 index file/node/container
- 1 metadata file/hostdir to cache "stat" information
- 1 data file/writer process

N-1 PLFS file overhead depends on interface – PLFS Lib

- Recommend 1 backend/parallel file system for N-1
- 1 top-level directory (container)/backend to hold the rest of the files
- 1 hostdir/node/container to hold data and index files
  - PLFSRC num_hostdirs parameter sets limit on maximum hostdirs/top-level directory
- 1 meta directory/container to hold metadata caching files
- 1 ".plfsaccess" file to enable distinguishing PLFS container from logical directory
- 1 version file to help with backward compatibility checks
- 1 data file/writer process
- 1 index file/writer process
- 1 metadata file/writer process

N-N and 1-N PLFS file overhead is simpler than N-1

- N-N:
  - Recommend ~10 backends/parallel file system
  - 1 data file/writer process distributed to one of the backends
  - Create 1 directory/backend when directory is created
    - Not necessarily at file creation time
- 1-N:
  - Recommend 1 backend/parallel file system
  - 1 data file/writer process in which the N files are stored
  - 1 index file/writer process with reference to which of N files in the 1 file it refers
  - 1 map file/writer process that maps the index number to a string name for the file to which it refers

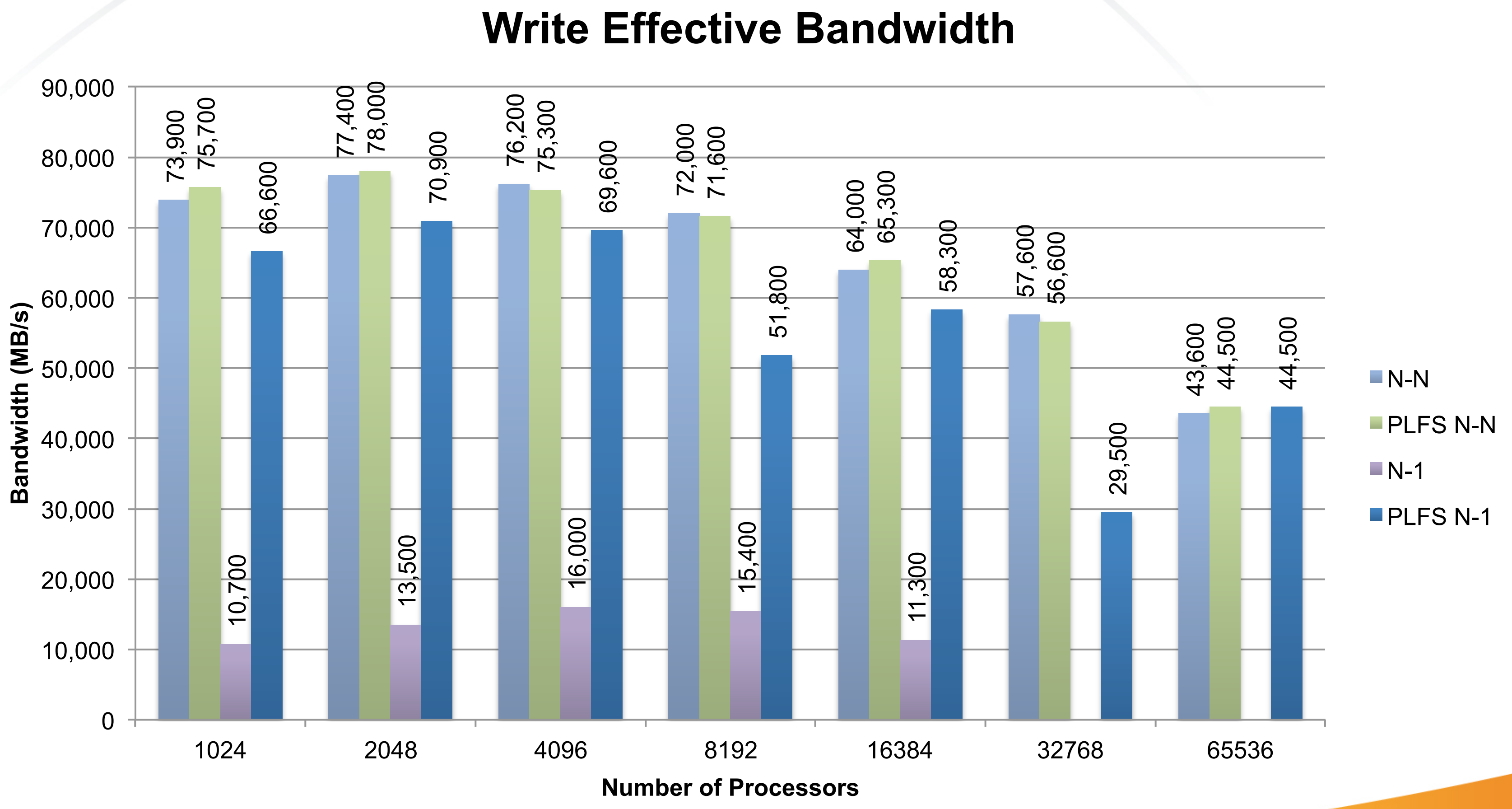
Use PLFS for very large parallel file I/O, primarily N-1 or 1-N

- Need to work with lots of data to amortize additional file open/sync/close overhead
  - Restart and graphics dumps
- Small files should go to NFS (/usr/projects, /users, /netscratch)
  - Executables, problem parameters, log files, etc.
- Small File (1-N), in particular, is not completely POSIX-compliant for performance reasons
  - Explicit sync calls required to ensure latest goes to or comes from storage
- Avoid O_RDWR to enhance performance
  - In RDWR mode any read must completely rebuild the index from storage – slow!
- Avoid stat'ing files to monitor progress

fs_test for maximum bandwidth scenario

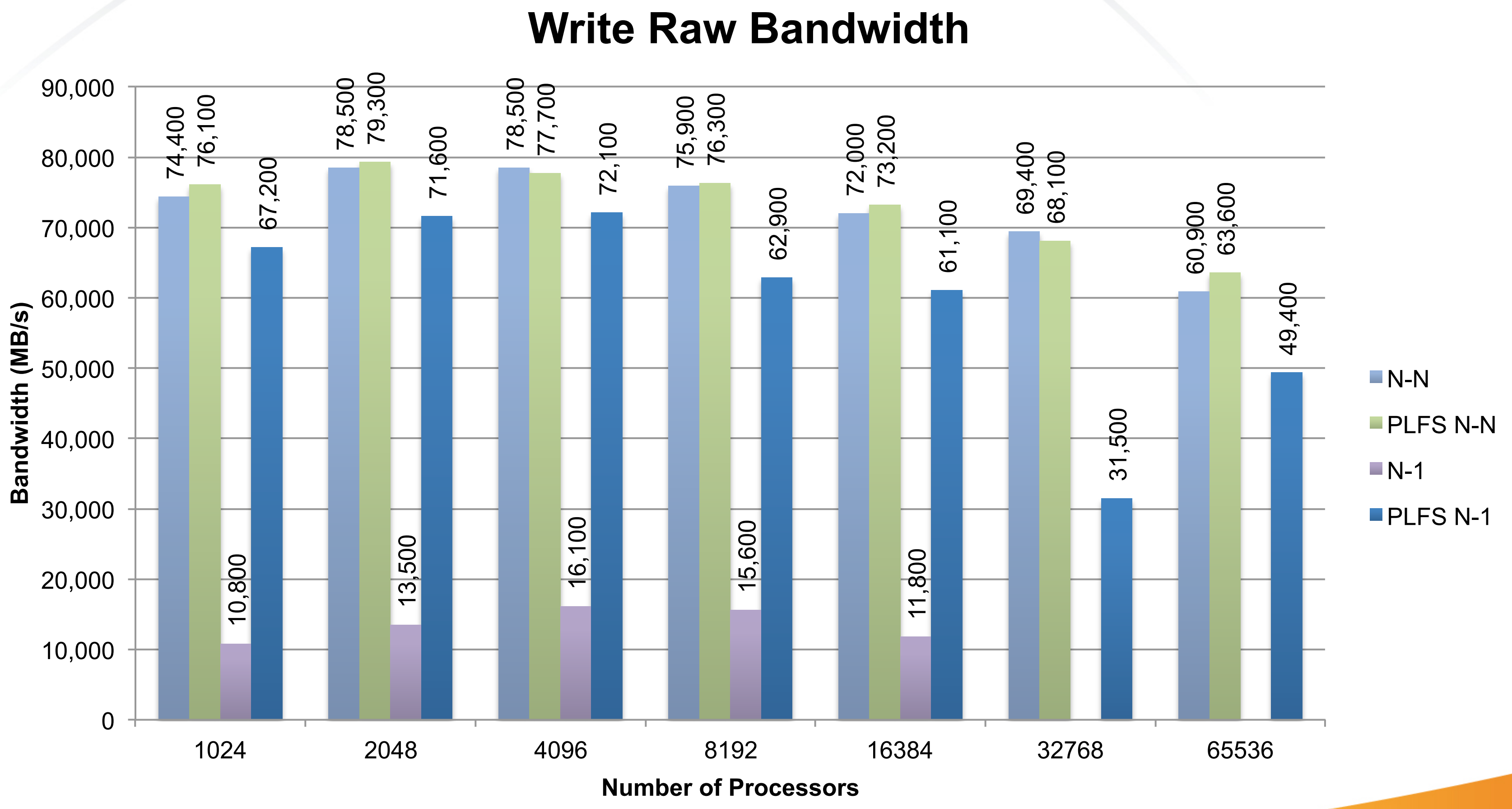
- Get it at https://github.com/fs-test
- Used PLFS's MPI/IO interface
- Unable to complete N-1 runs for 32K & 64K pes
- Used ACES Cray XE6 lscratch3 PFS
  – ½ the total Lustre hardware on the system
- Competed with other jobs running on system
  – See PLFS N-1 results at 32K pes

Significantly better N-1 performance & N-N comparable to non-PLFS



UNCLASSIFIED
LA-UR-14-22100

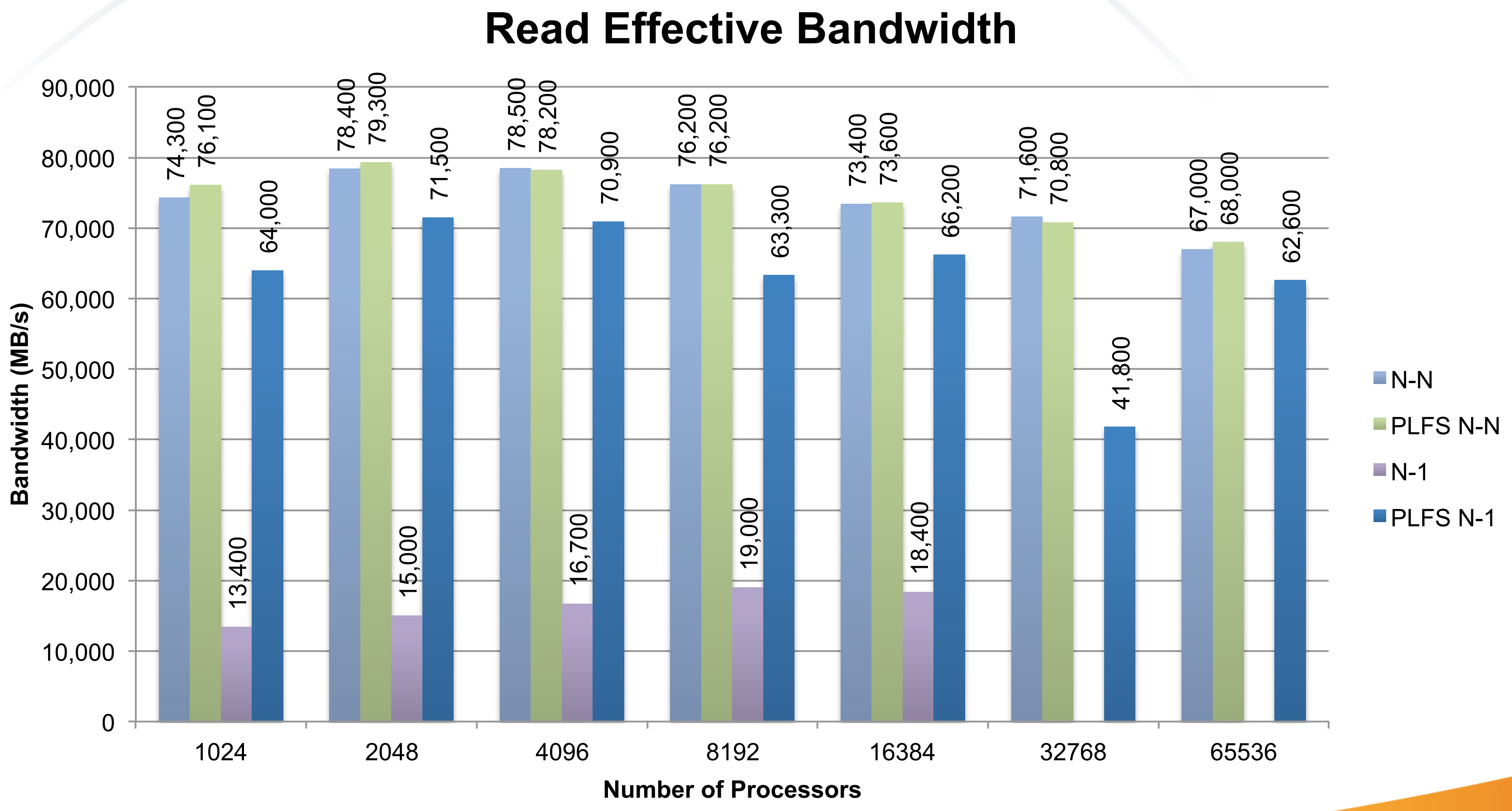
Raw bandwidth (removing create, sync, close) shows amortization



Number of Processors	N-N	PLFS N-N	N-1	PLFS N-1
1024	74,400	76,100	10,800	67,200
2048	78,500	79,300	13,500	71,600
4096	78,500	77,700	16,100	72,100
8192	75,900	76,300	15,600	62,900
16384	72,000	73,200	11,800	61,100
32768	69,400	68,100		31,500
65536	60,900	63,600		49,400

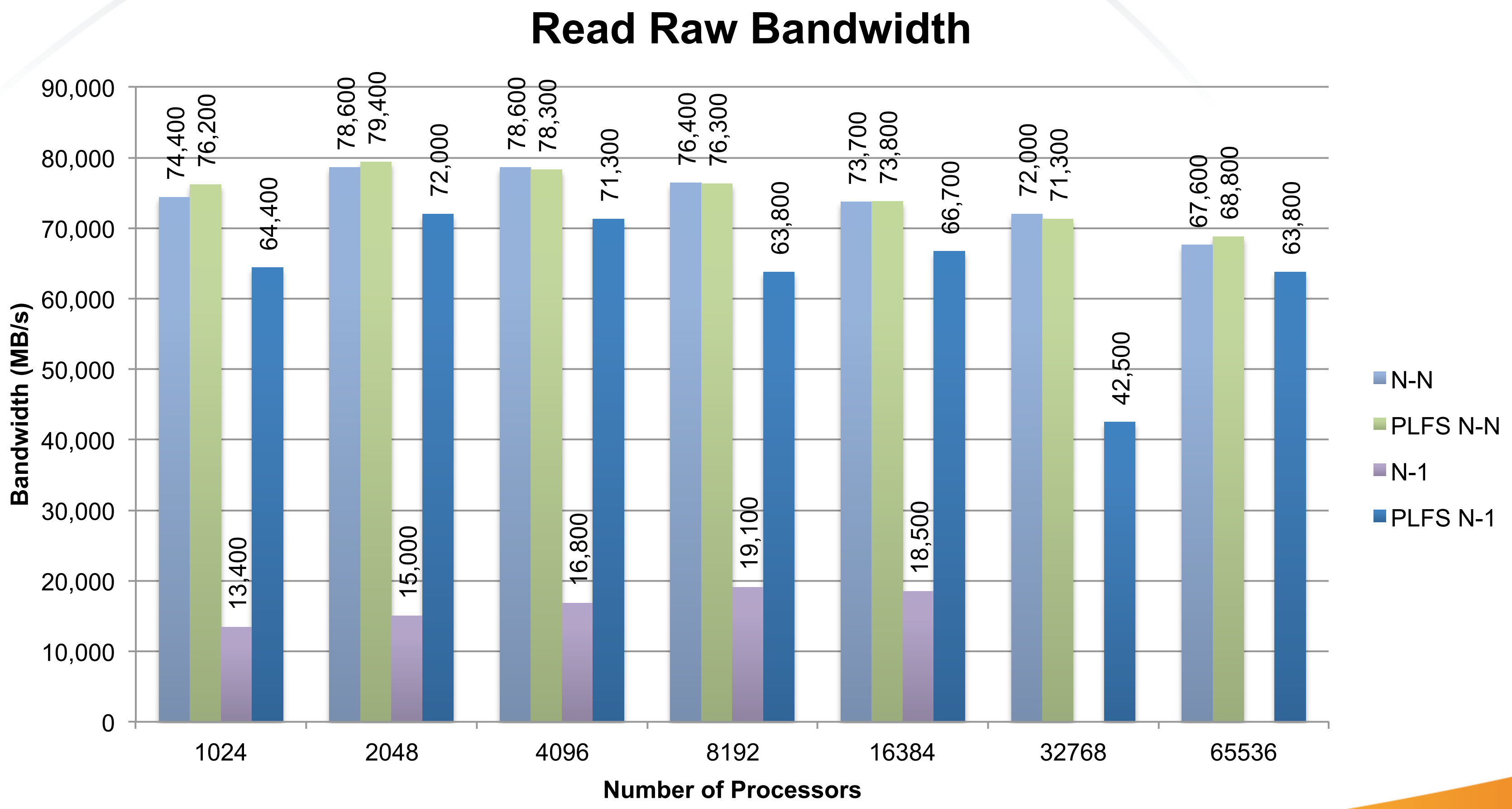
UNCLASSIFIED
LA-UR-14-22100

Read results similar to write results



UNCLASSIFIED
LA-UR-14-22100

Read raw shows same amortization in high bandwidth scenario

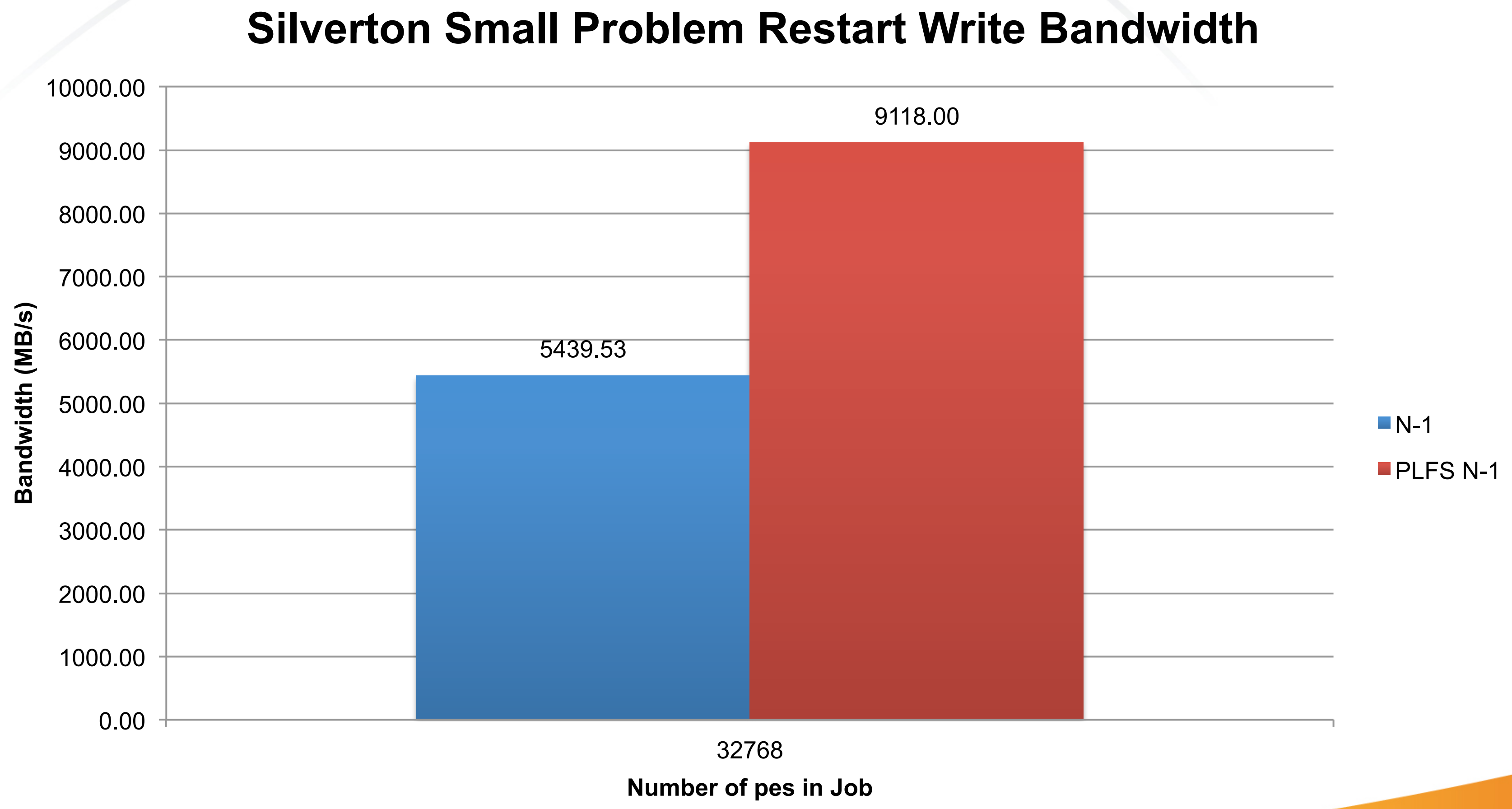


UNCLASSIFIED
LA-UR-14-22100

Measured small & large Silverton problem performance

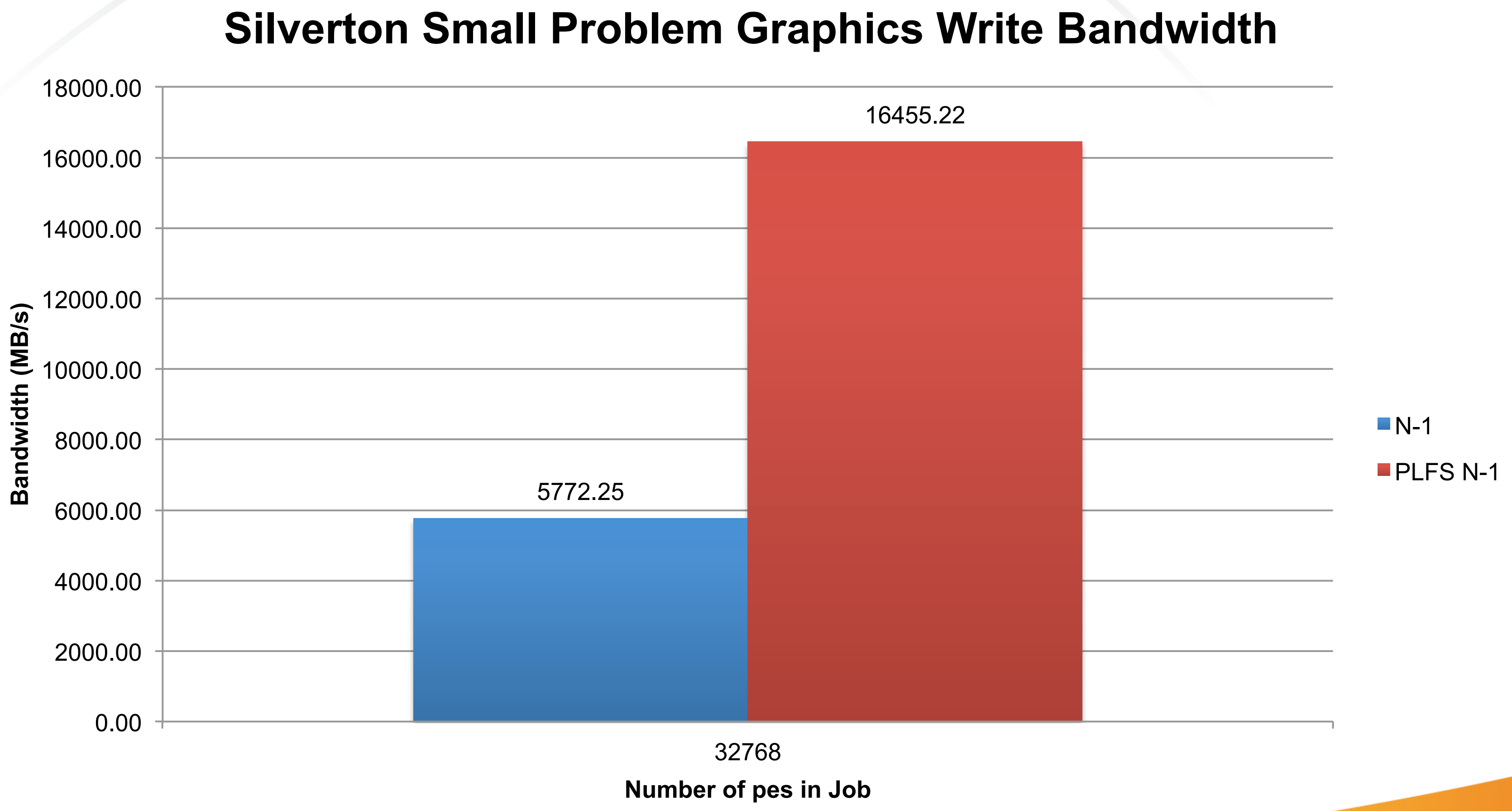
- 3-D Eulerian finite difference code to study high-speed compressible flow & high-rate material deformation
  - Multiple MPI/IO modes: N-1 strided, N-1 segmented, and N-N
- Used a small file size problem first in just N-1 strided
- Measured a large file size problem using all modes
- Used ACES Cray XE6 lscratch3 PFS
- Competed with other jobs running on system

Small problem N-1 strided write showed 1.68x improvement



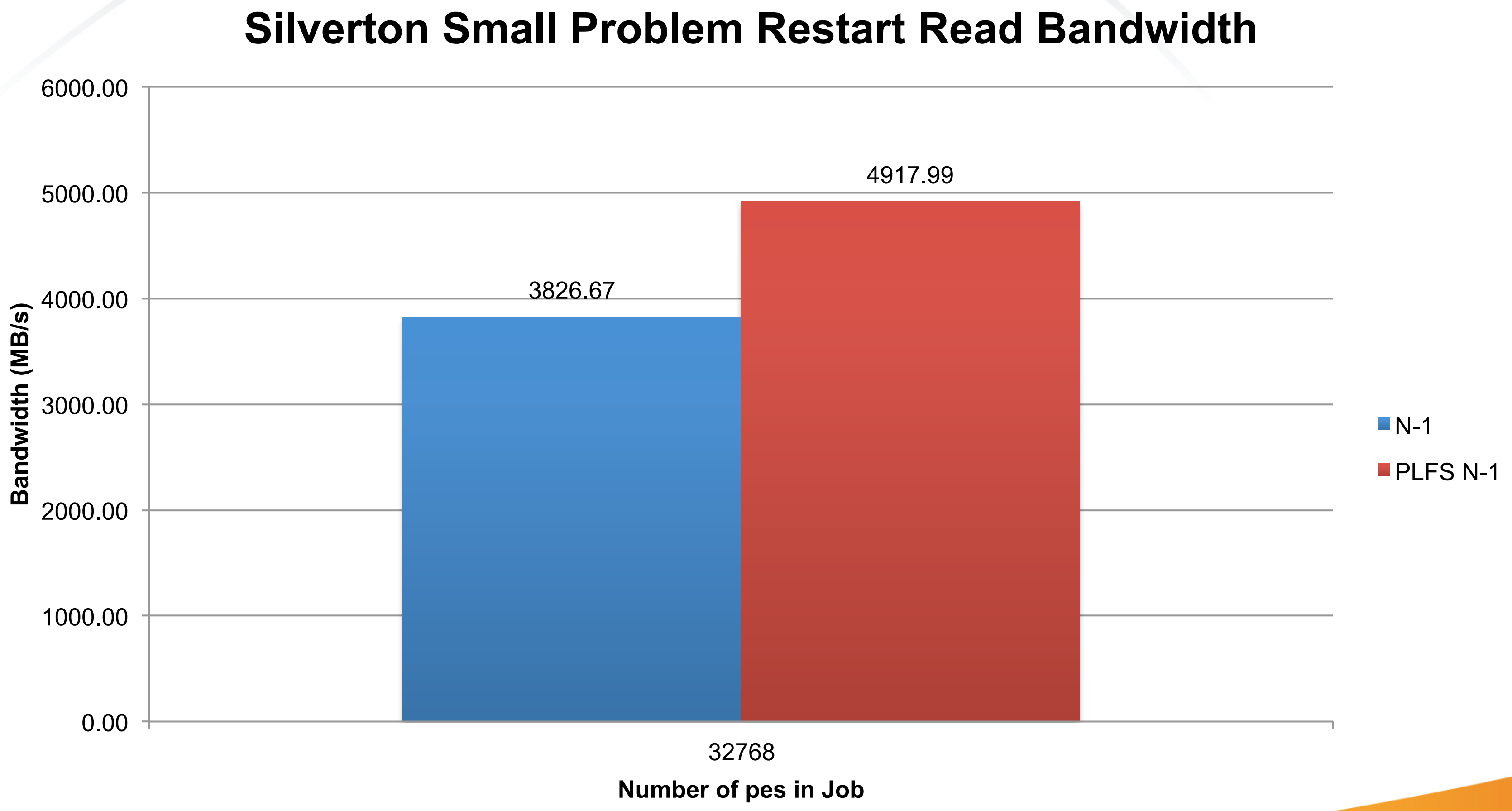
UNCLASSIFIED
LA-UR-14-22100

The smaller graphics file was even better, 2.85x



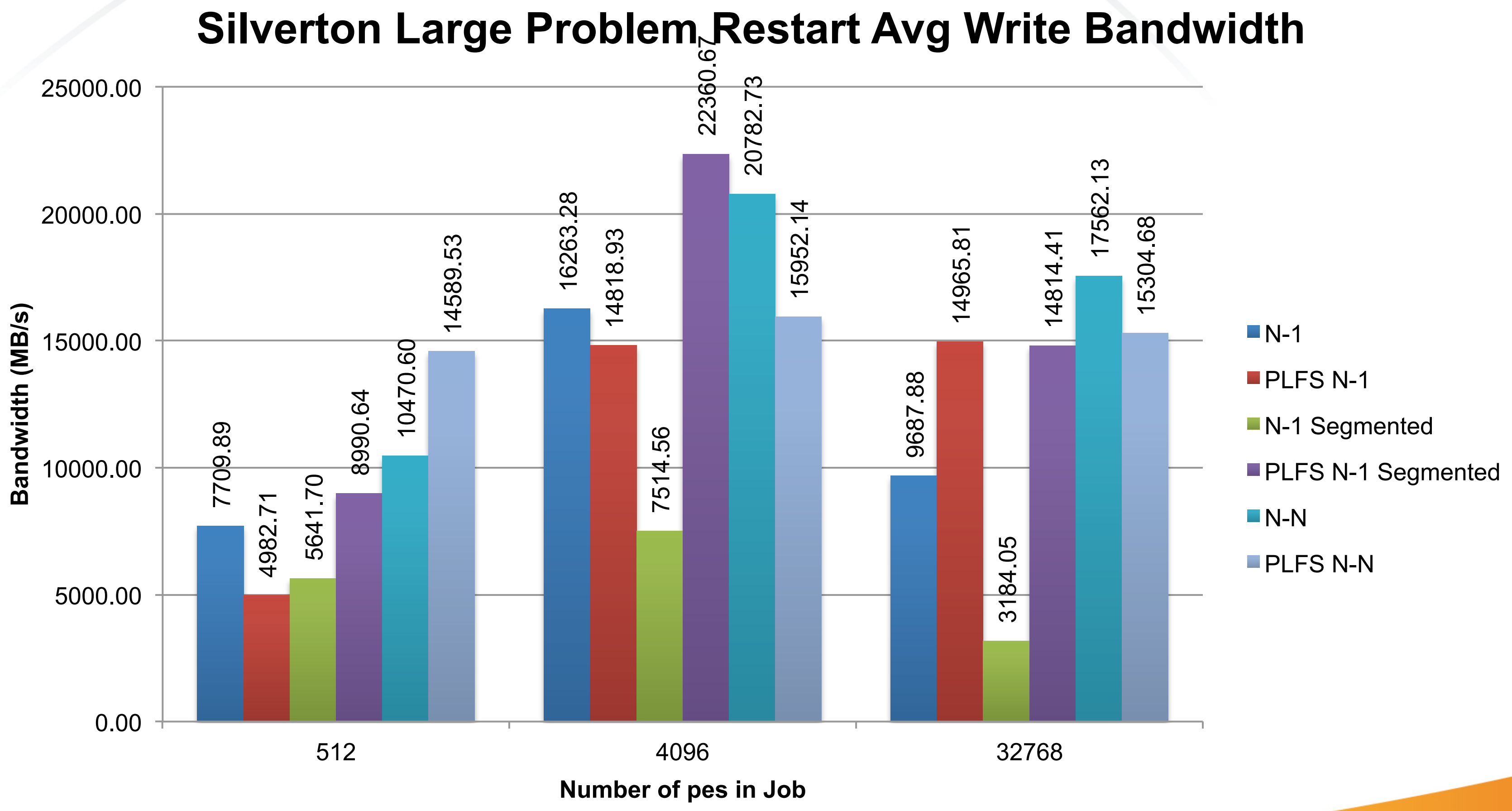
UNCLASSIFIED
LA-UR-14-22100

Small problem N-1 strided read showed 1.28x improvement



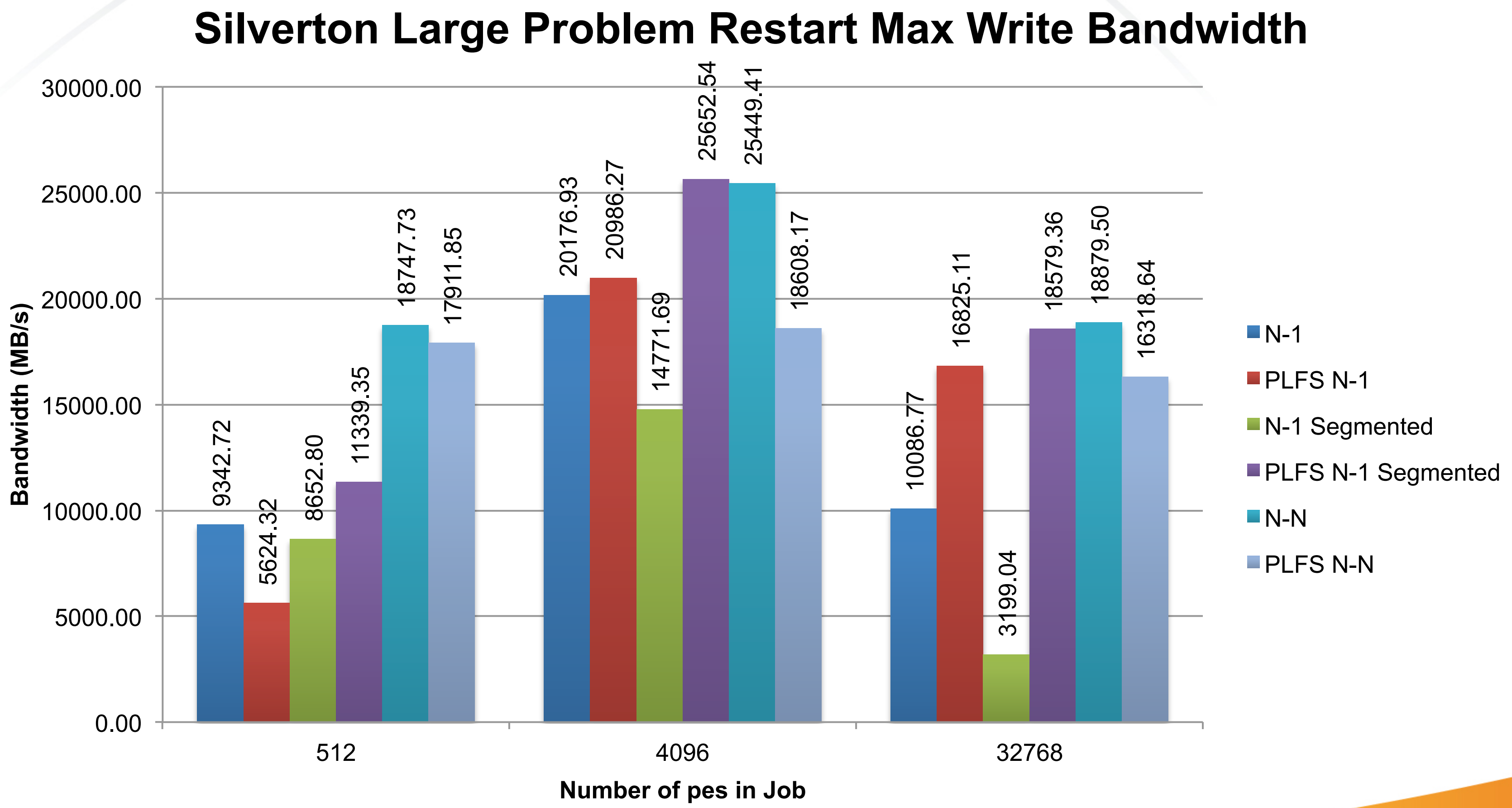
UNCLASSIFIED
LA-UR-14-22100

As writer count increased so did PLFS advantage



UNCLASSIFIED
LA-UR-14-22100

Occasionally conditions are just right – the best we can do on this problem

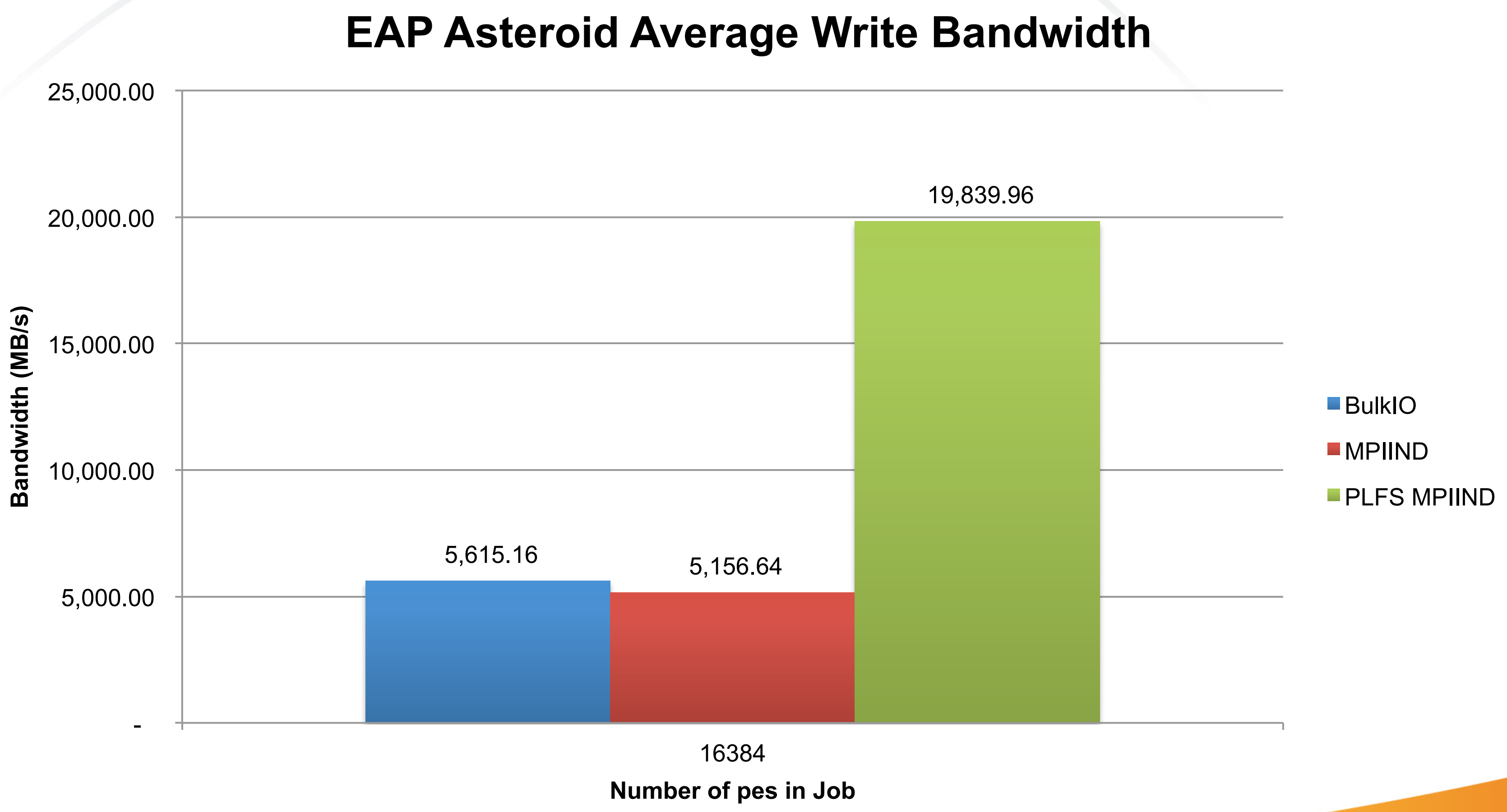


UNCLASSIFIED
LA-UR-14-22100

Measured small & large EAP problem performance

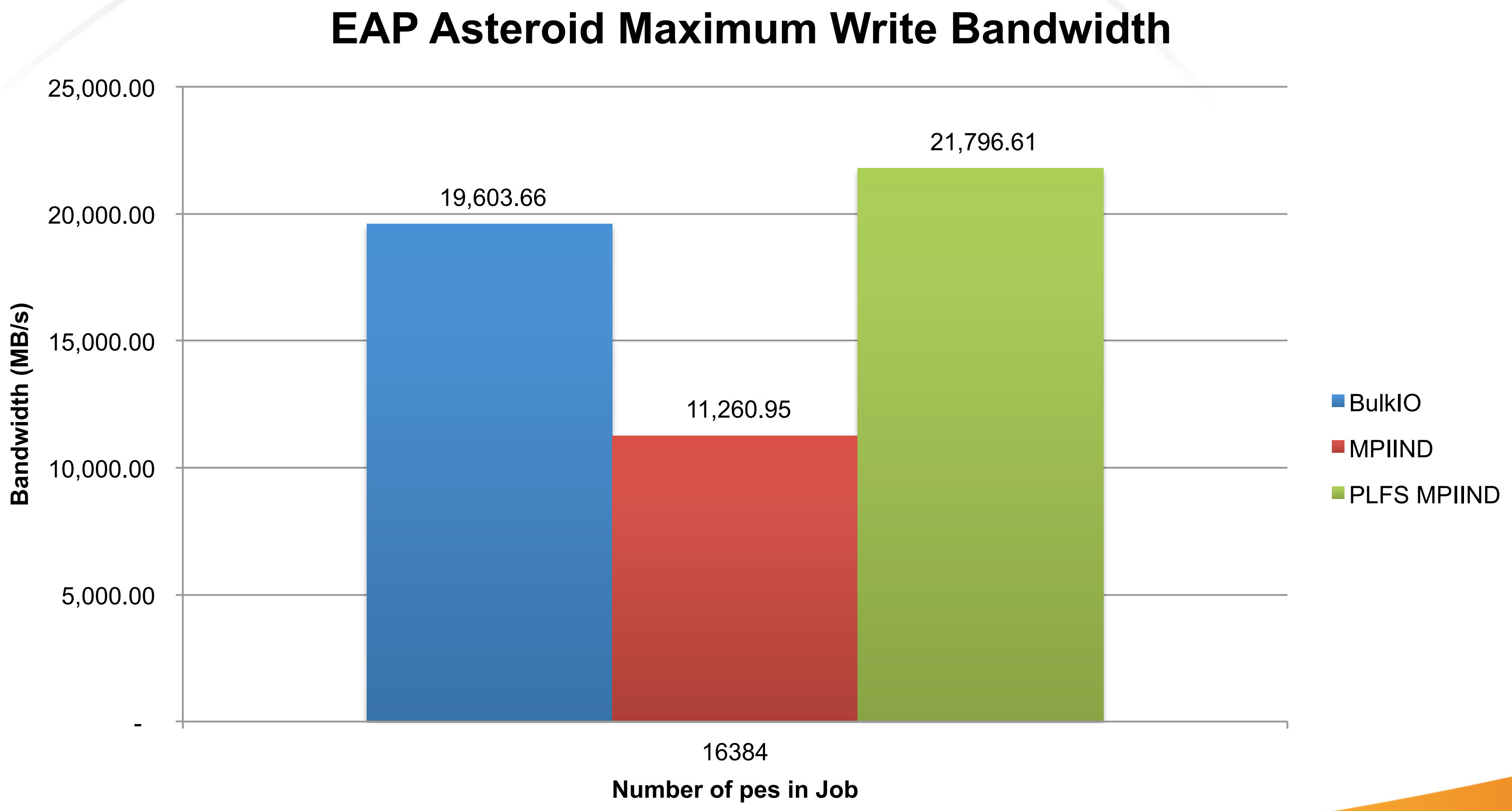
- 3-D Godunov solver using Eulerian mesh with AMR (Adaptive Mesh Refinement)
  - Multiple MPI/IO modes: BulkIO (aggregating N-1 strided) and MPIIND (N-1 strided)
- Used the Asteroid, a small file size, problem first on lscratch3
- Measured the "MD", a very large file size, problem on lscratch2, lscratch3, and, for PLFS, a virtual combination of lscratch 2, 3, & 4
- Competed with other jobs running on system
  - Averages caught by competition, so will show maximum values as better comparison

We think Asteroid BulkIO & MPIIND runs encountered I/O competition



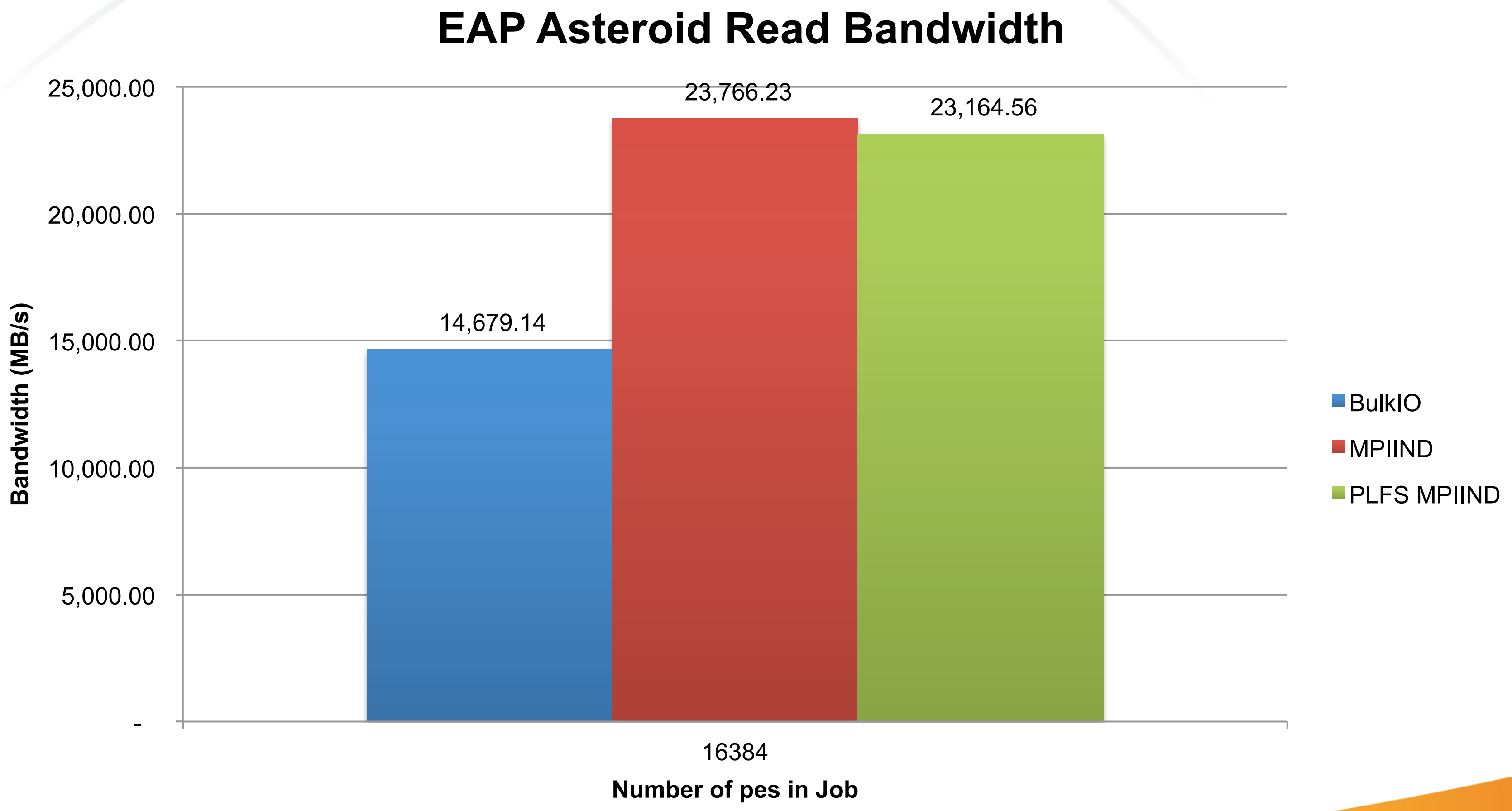
UNCLASSIFIED
LA-UR-14-22100

Max Asteroid write 1.1x over BulkIO & 1.94x over MPIIND



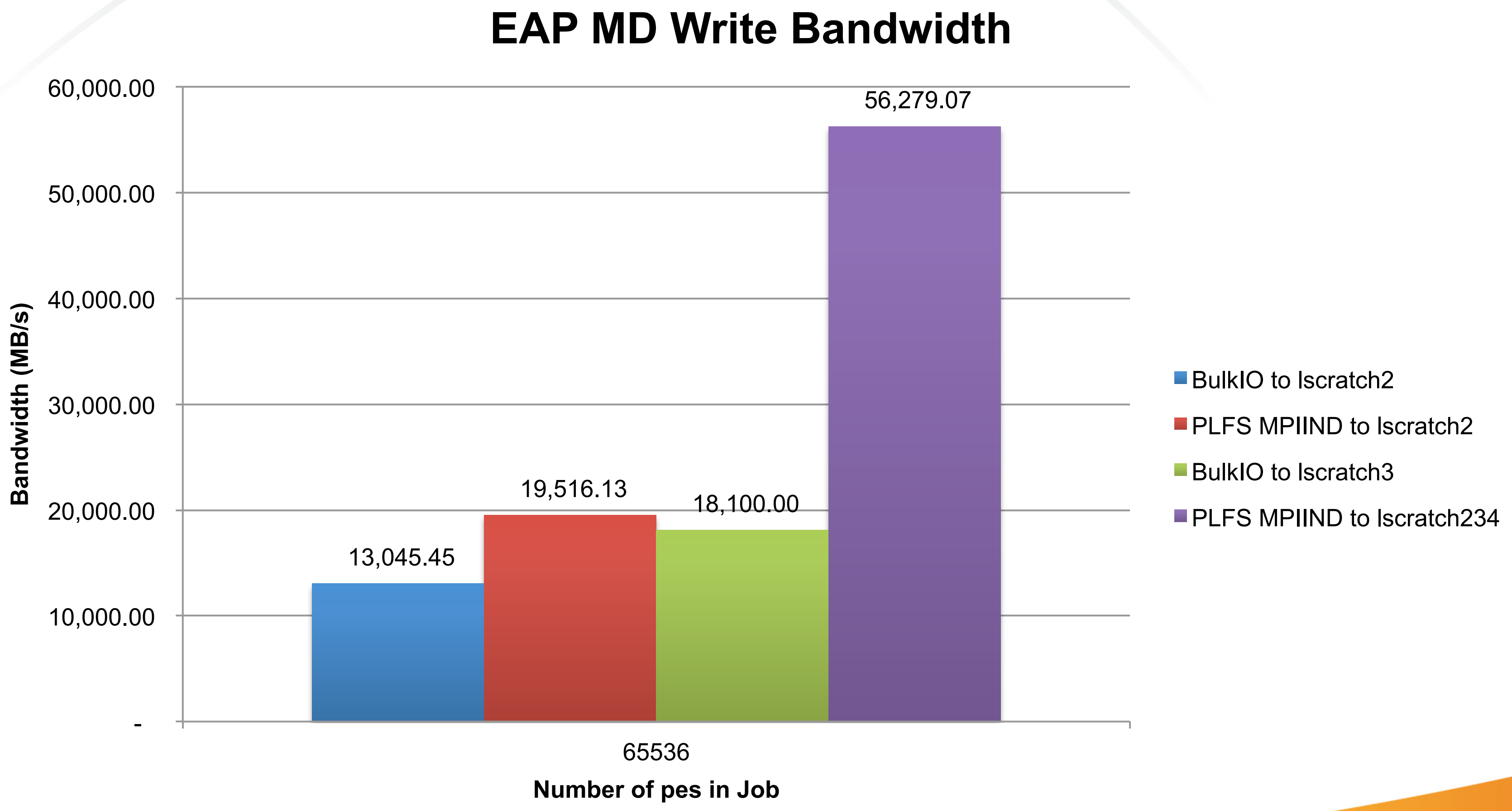
UNCLASSIFIED
LA-UR-14-22100

Asteroid read 1.58x over BulkIO, effectively equal to MPIIND



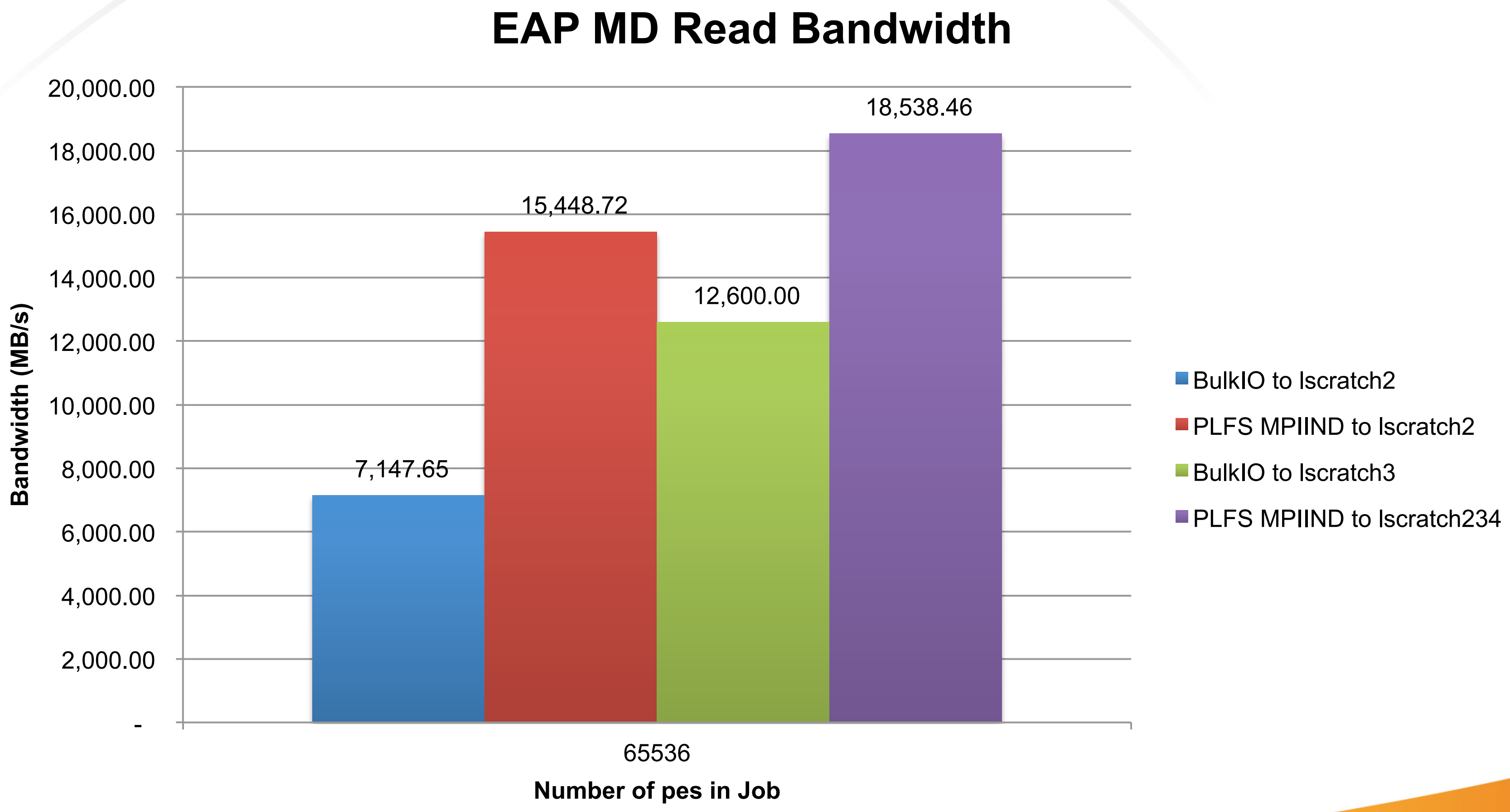
UNCLASSIFIED
LA-UR-14-22100

“MD” write 1.5x over BulkIO, big gain by combining multiple PFSes



UNCLASSIFIED
LA-UR-14-22100

"MD" read 2.1x over BulkIO, not too big gain by combining multiple PFSes



UNCLASSIFIED
LA-UR-14-22100

More work is needed for 7/24/365 production use

- Patch FUSE buffers from 128 KiB to at least 4 MiB
  - Allows tools reading files to amortize read overhead over more data
  - Early testing shows HSI archive tool sees 1.36x – 2.39x performance improvement, depending on file size
- Manage PLFS indexes in scalable manner
  - Every pe reads full index now, which can fill memory
  - Experimenting with MDHIM (Multi-Dimensional Hashing Indexing Middleware), a distributed key/value middleware
    - See https://github.com/mdhim
- Should Lustre (or other PFSes) improve their shared file performance, LANL will focus on PLFS as an enabling technology for Burst Buffer
  - MSST12 Paper
  - EMC/LANL Burst Buffer Demo at SC13
  - DOE Fast Forward Project