Structuring PLFS for Extensibility

Chuck Cranor, Milo Polte, Garth Gibson

PARALLEL DATA LABORATORY
Carnegie Mellon University

What is PLFS?

- Parallel Log Structured File System
  - Interposed filesystem b/w apps & backing storage
  - Los Alamos National Labs, CMU, EMC, …
  - Target: HPC checkpoint files
- PLFS transparently transforms a highly concurrent write access pattern to a pattern more efficient for distributed filesystems
  - First paper: Bent et al, Supercomputer 2009
  - http://github.com/plfs, http://institute.lanl.gov/plfs/

Checkpoint Write Patterns

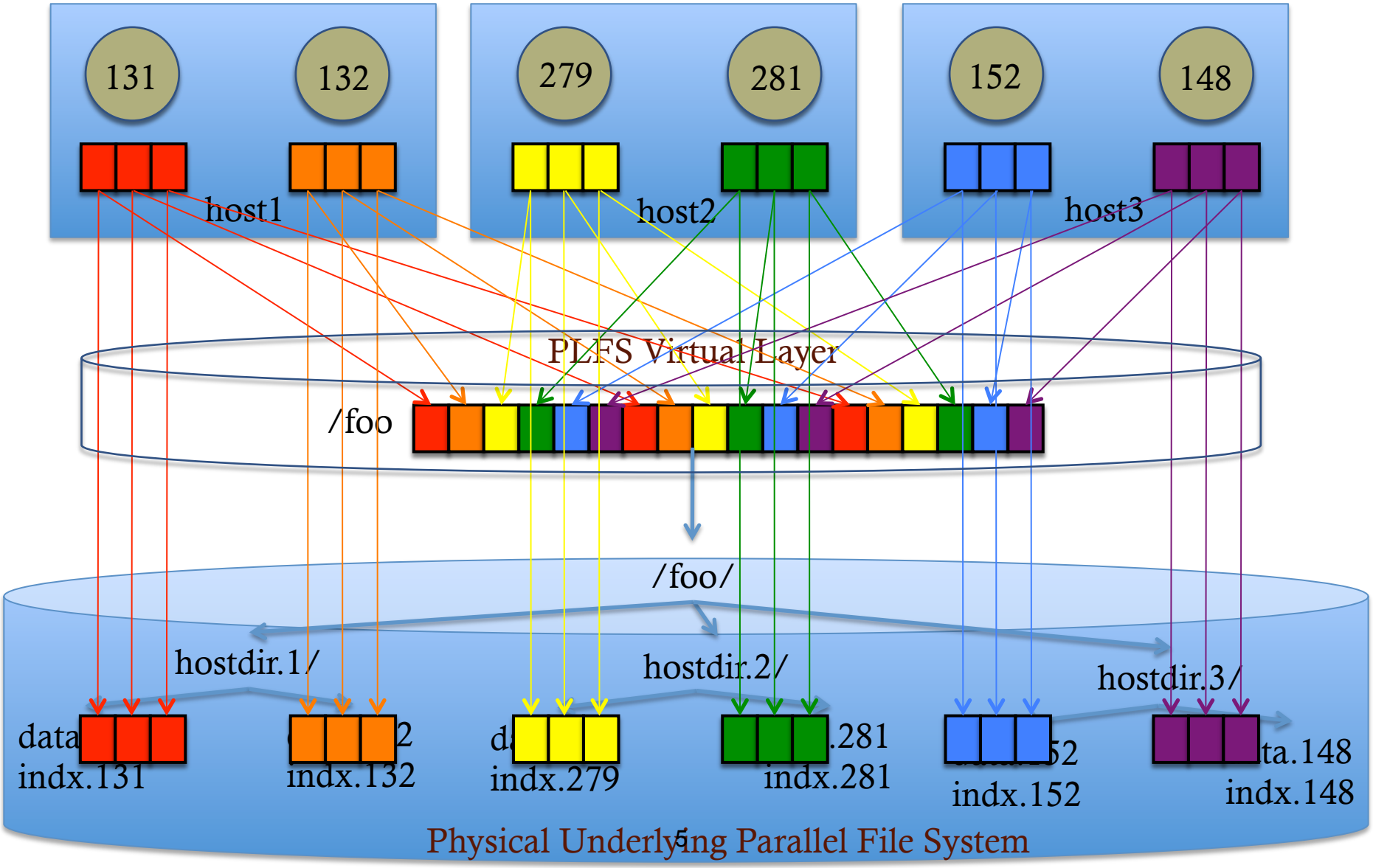
- The two main checkpoint write patterns:
  - N-1: all N processes write to one shared file
    - Concurrent I/O to a single file is often unscalable
    - Small, unaligned, clustered traffic is problematic
  - N-N: each process writes to its own file
    - Overhead of inserting many files in a single dir
    - Easier for DFS (after files created)
    - Archival and management more difficult
- Initial PLFS focus: improve N-1 case

PLFS Transforms Workloads

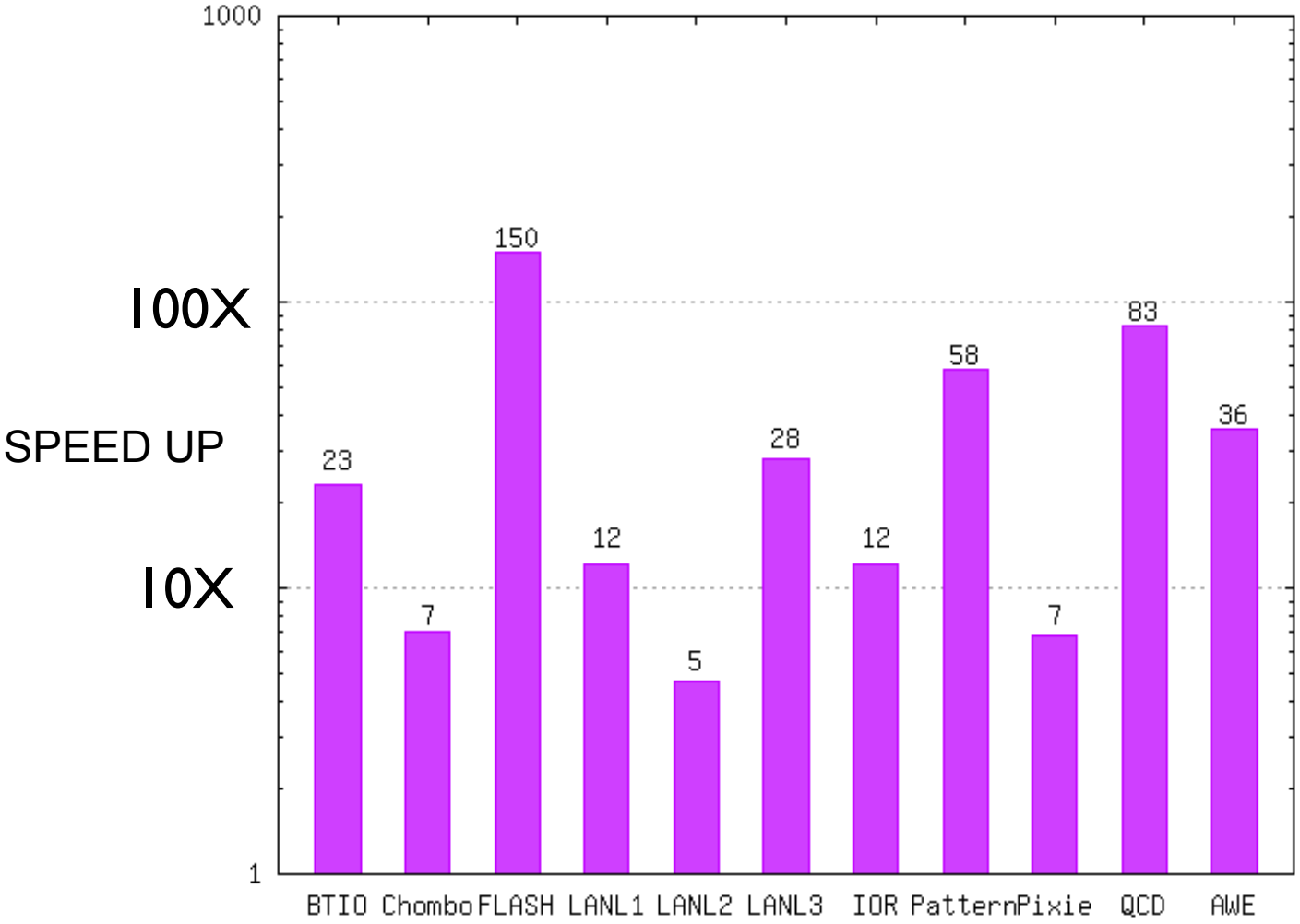
- PLFS improves N-1 performance by transforming it into an N-N workload

- FUSE/MPI: transparent solution, no application changes required

PLFS Converts N-1 to N-N



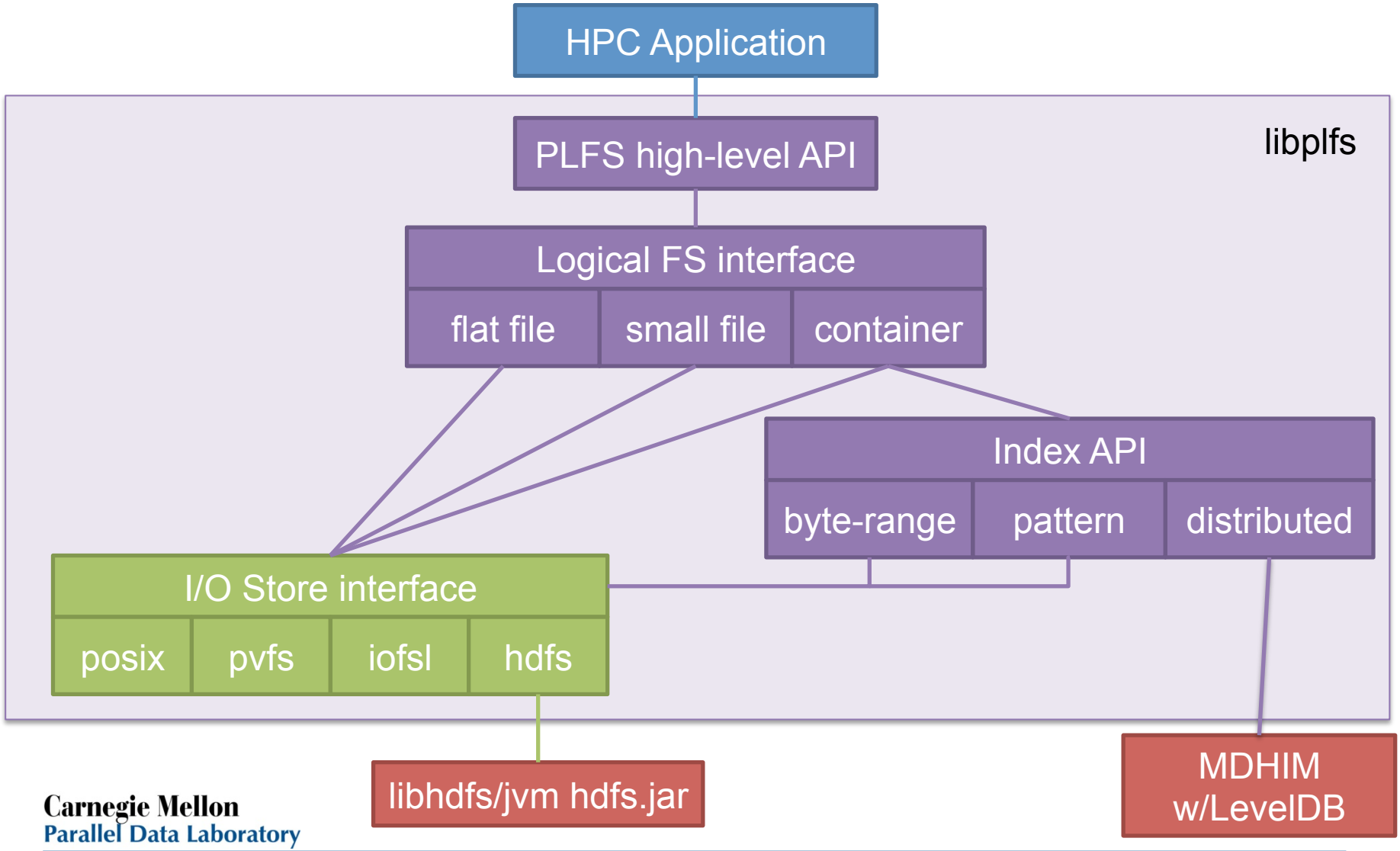
PLFS N-1 Bandwidth Speedups



The Price of Success

- Original PLFS was limited to 1 workload:
  - N-1 checkpoint on mounted posix filesystem
  - All data stored in PLFS container logs
- Ported first to MIO-IO/ROMIO
  - Feasibly deploy on leadership class machines
- Success with LANL apps: actual adoption?
  - Requires maintainability & roadmap evolution
  - Develop a team: LANL, EMC, CMU, …
- Revisit code with maintainability in mind

PLFS Extensibility Architecture



Case Study: HPC in the Cloud

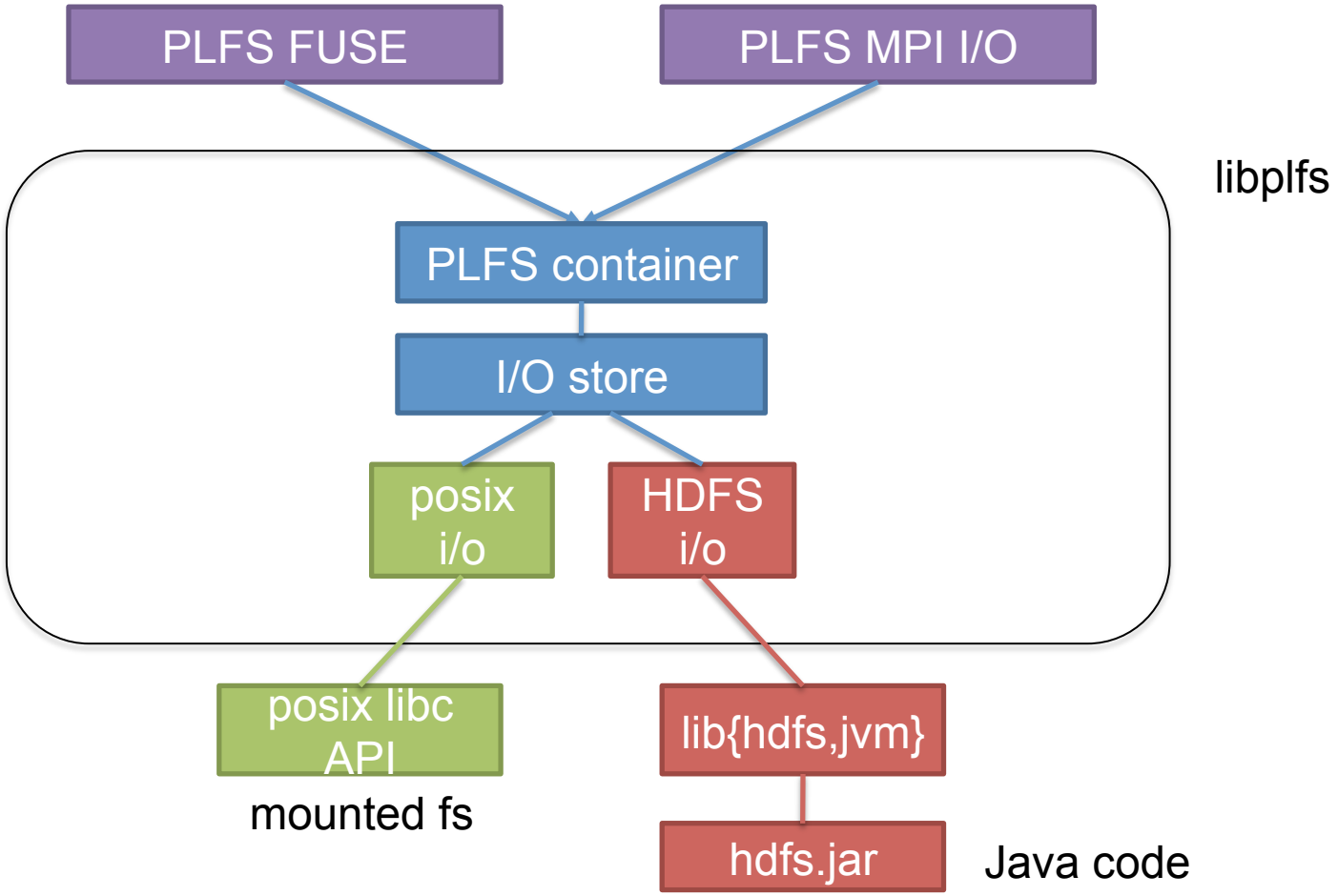
- Emergence of Hadoop: converged storage
- HDFS: Hadoop Distributed Filesystem
  - Key attributes:
    - Single sequential writer (not POSIX, no pwrite)
    - Not VFS mounted, access through Java API
    - Local storage on nodes (converged)
    - Data replicated ~3 times (local+remote1+remote2)
- HPC in the Cloud: N-1 checkpoint on HDFS?
  - Observation: PLFS log I/O fits HDFS semantics

PLFS Backend Limitations

- PLFS hardwired to POSIX API:
  - Needs a kernel mounted filesystem
  - Uses integer file descriptors
  - Memory maps index files to read them
- HDFS does not fit these assumptions

- Solution: I/O Store
  - Insert a layer of indirection above PLFS backend
  - Model after POSIX API to minimize code changes

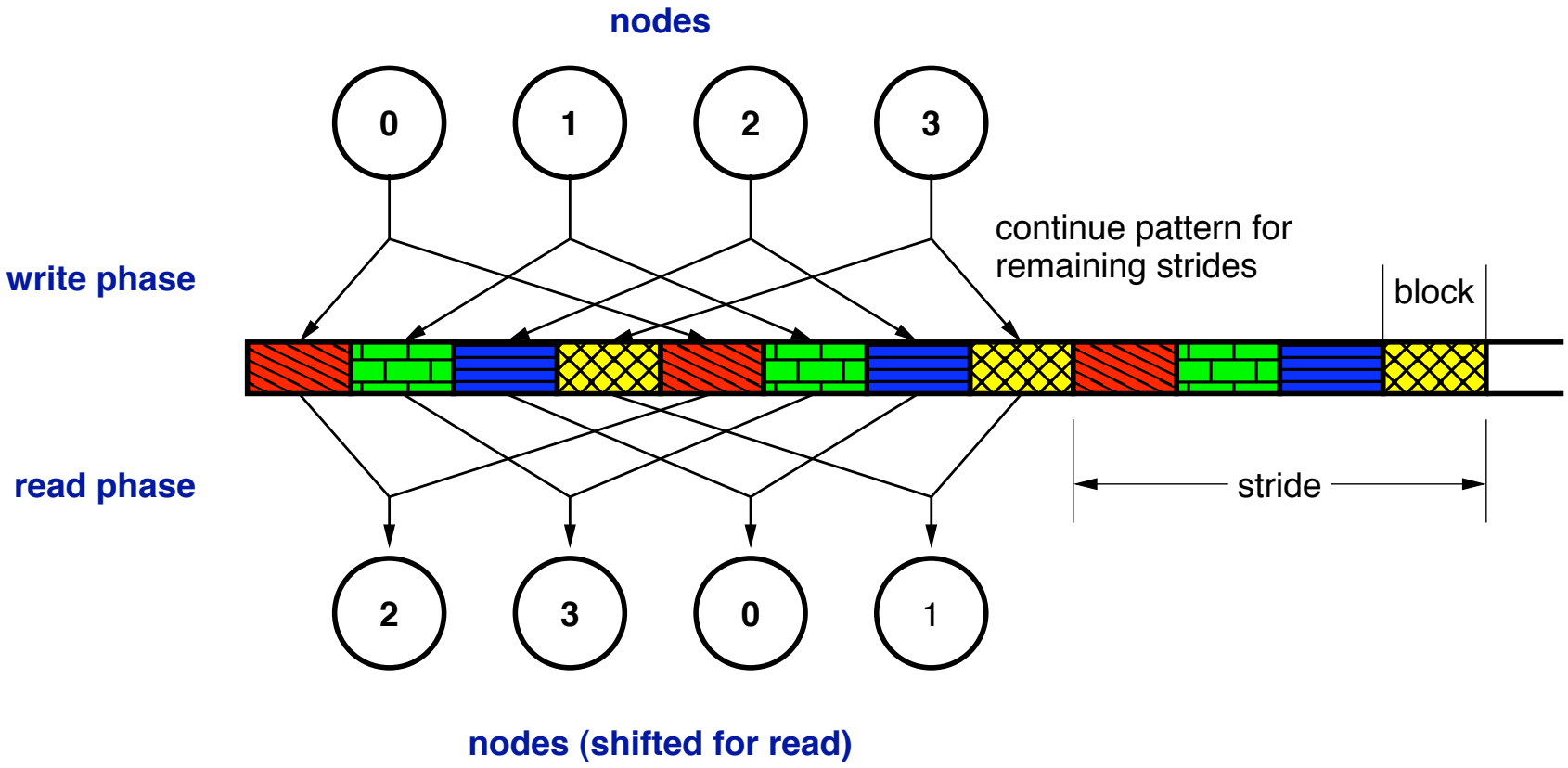
PLFS I/O Store Architecture



PLFS/HDFS Benchmark

- Testbed: PRObE (www.nmc-probe.org)
  - Each node has dual 1.6GHz AMD cores, 16GB RAM, 1TB drive, gigabit ethernet
  - Ubuntu Linux, HDFS 0.21.0, PLFS, OpenMPI
- Benchmark: LANL FS Test Suite (fs_test)
  - Simulates N-1 checkpoint, strided
  - Filesystems tested:
    - PVFS OrangeFS 2.8.4 w/64MB stripe size
    - PLFS/HDFS w/1 replica (local disk)
    - PLFS/HDFS w/3 replicas (local disk + remote1 + remote 2)
  - Blocksizes: 47001, 48K, 1M
  - Checkpoint size: 32GB written by 64 nodes

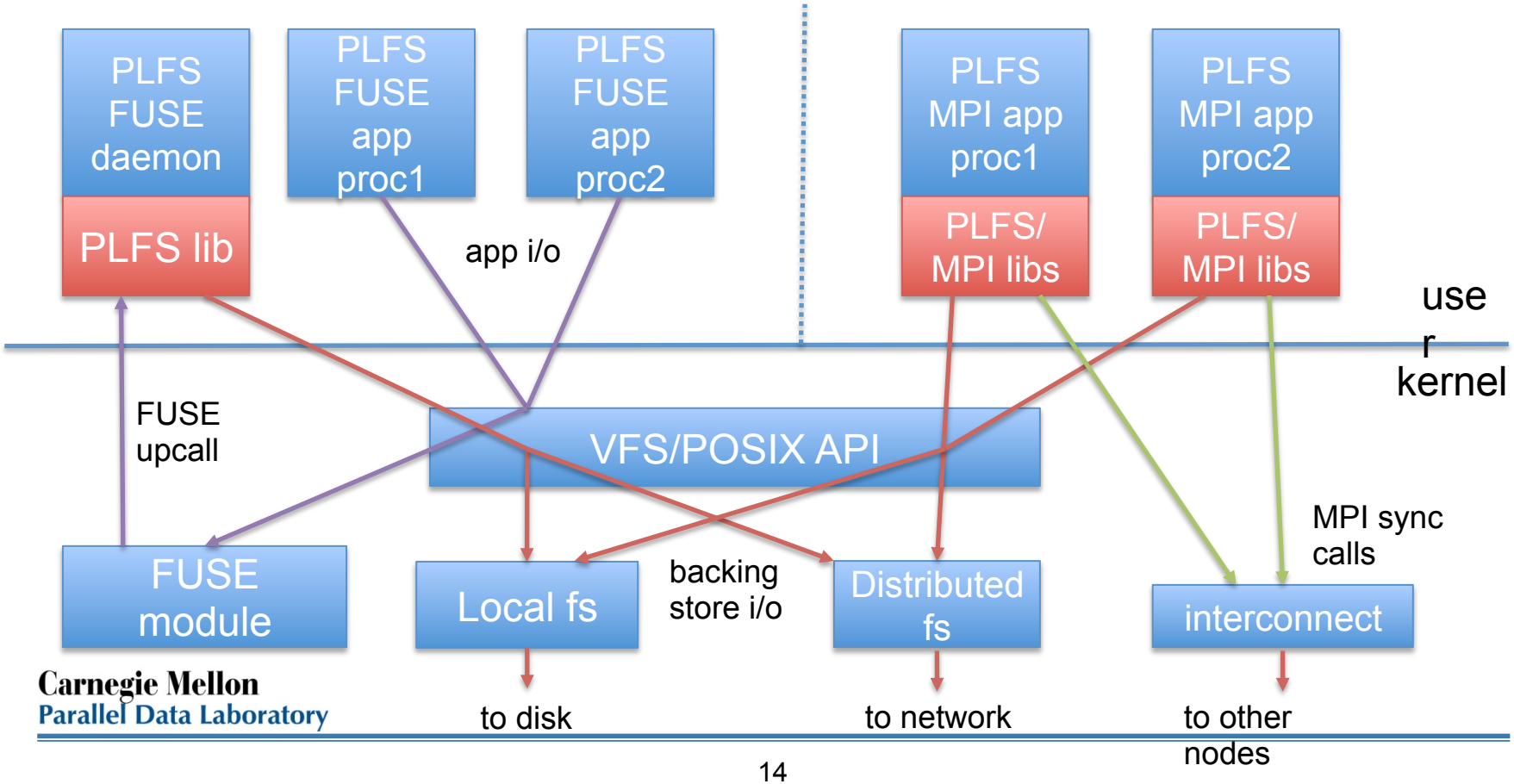
Benchmark Operation



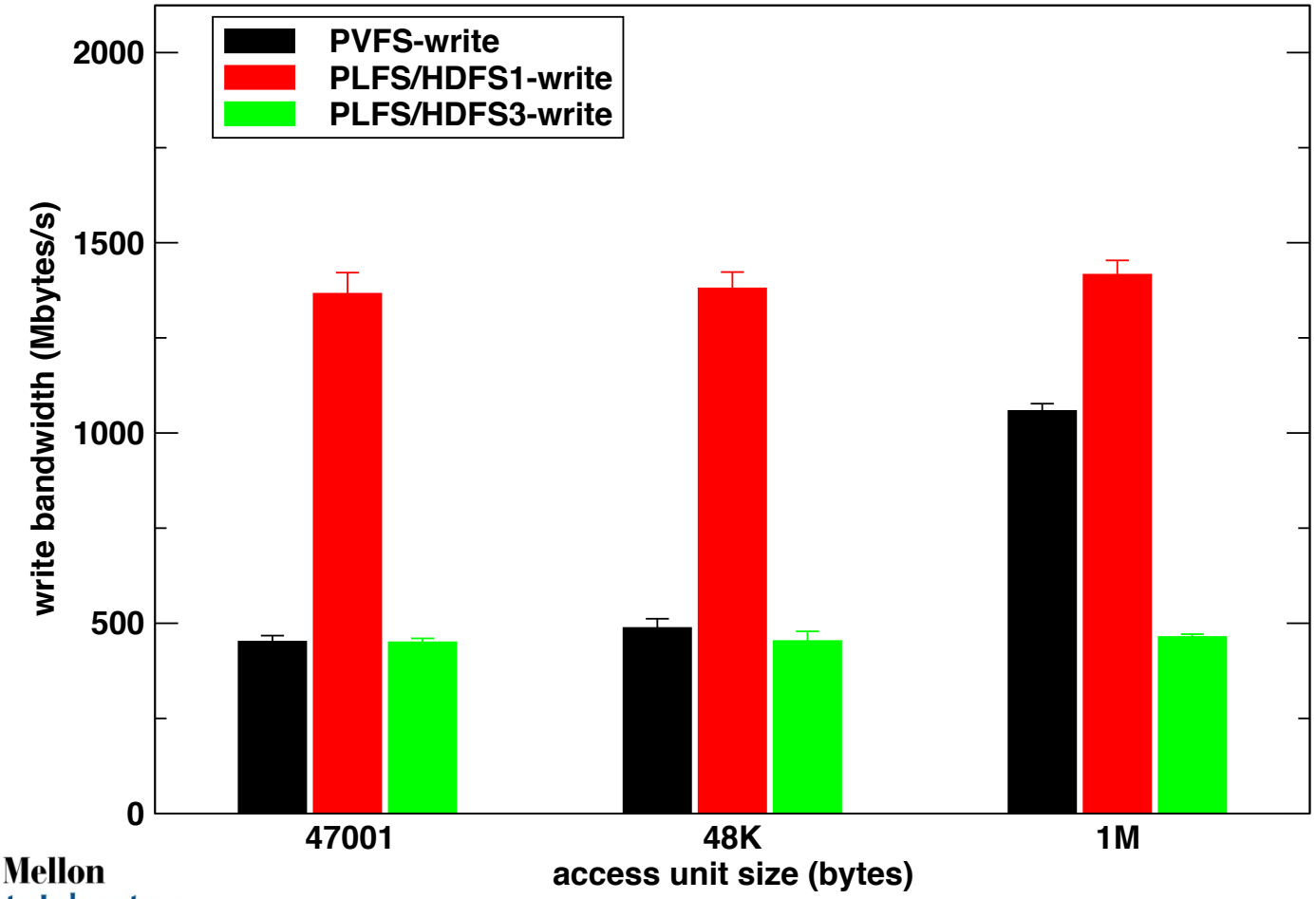
We unmount and cache flush data filesystem between read/write

PLFS Implementation Architecture

- FUSE filesystem and a Middleware lib (MPI)

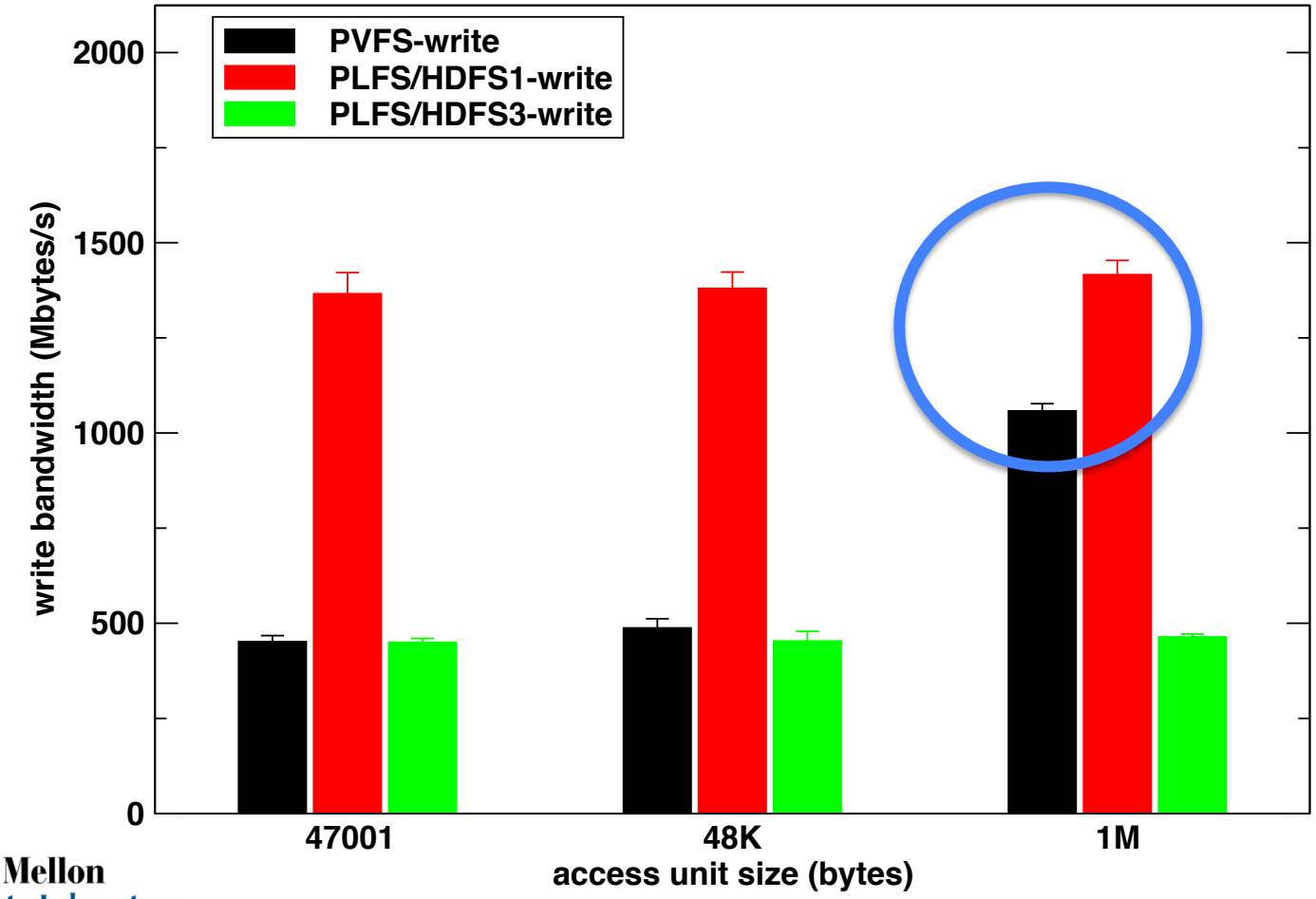


PLFS/HDFS Write Bandwidth



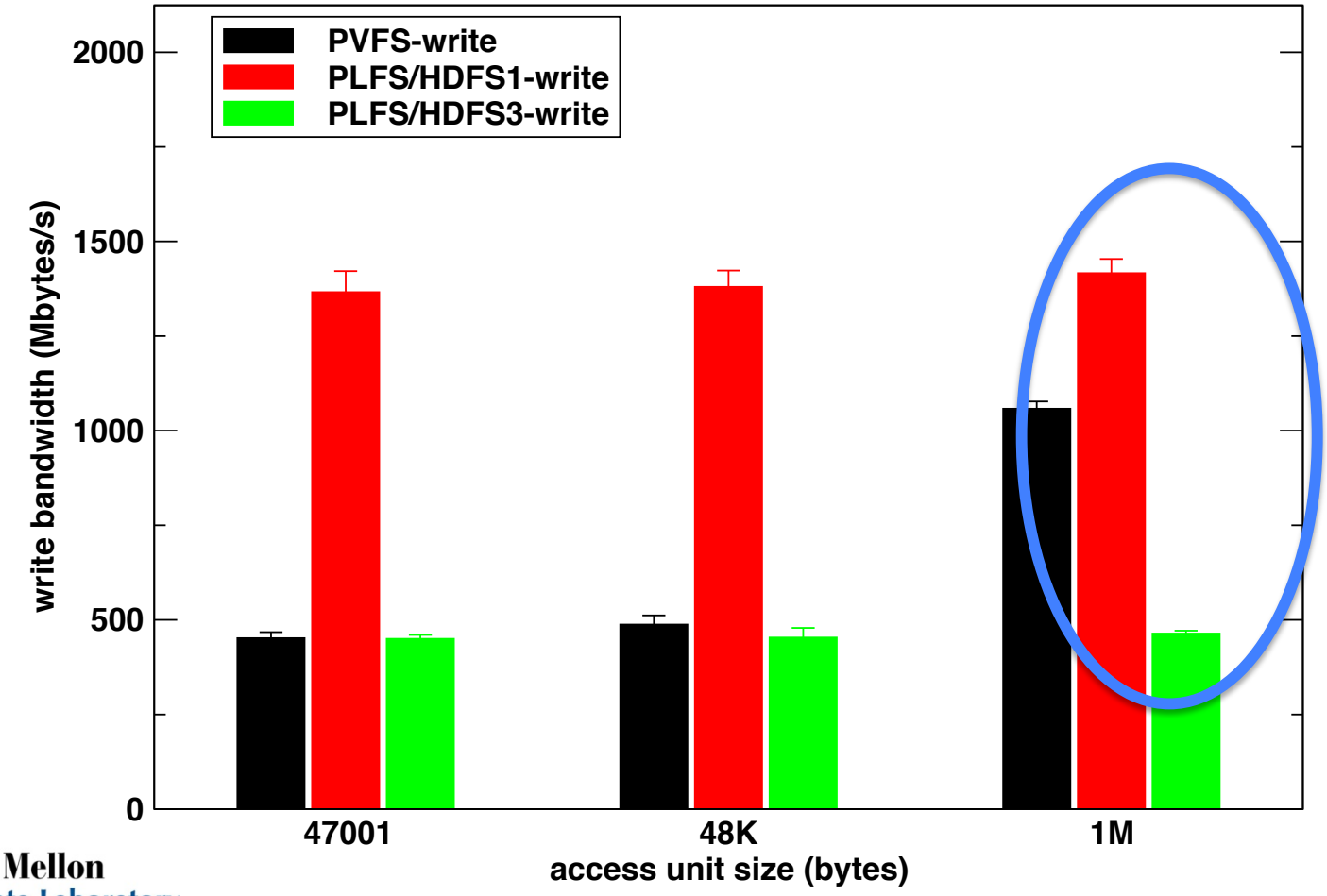
PLFS/HDFS Write Bandwidth

- PLFS/HDFS performs well (note HDFS1 is local disk)



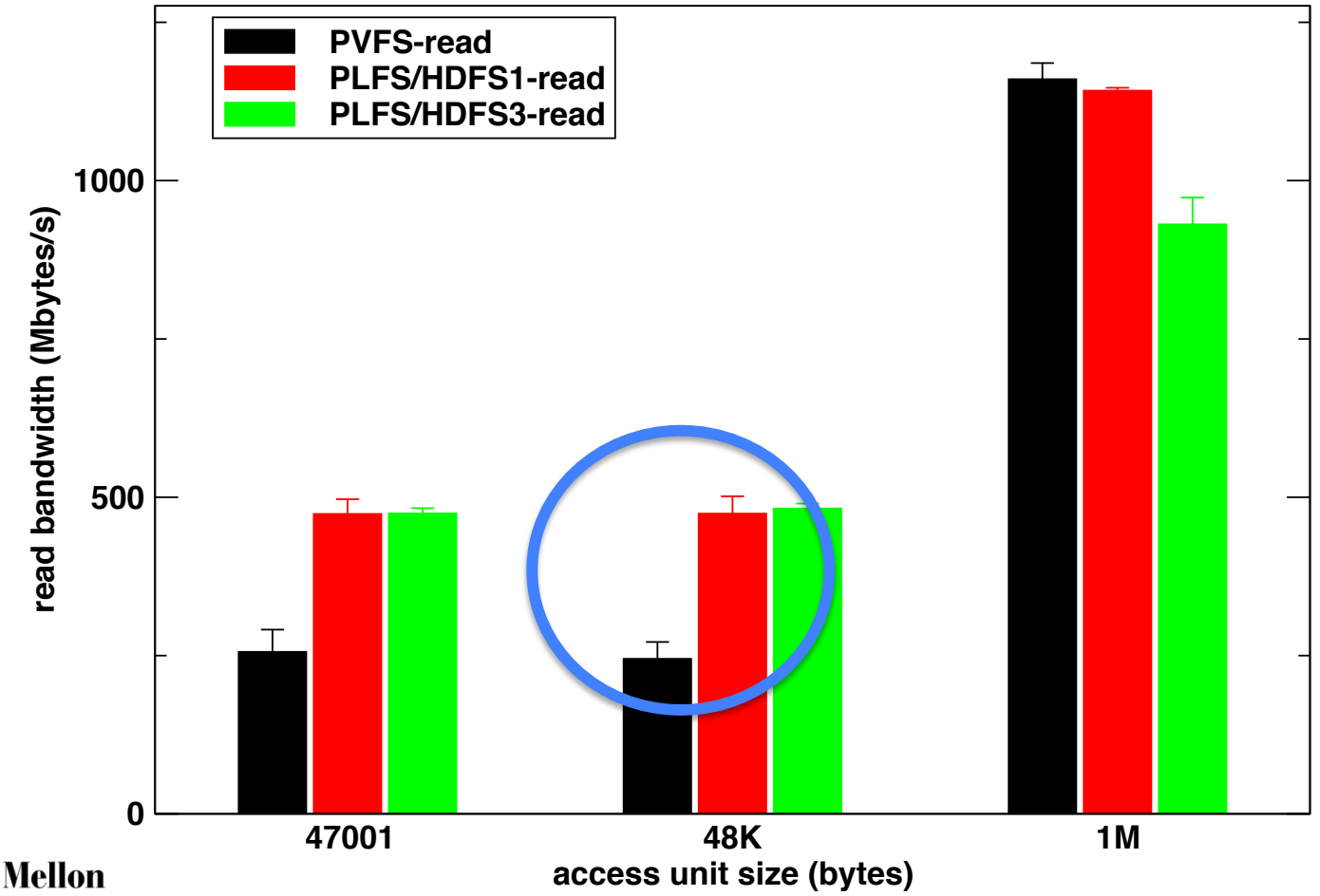
PLFS/HDFS Write Bandwidth

- PLFS/HDFS performs well (note HDFS3 is 3 copies)



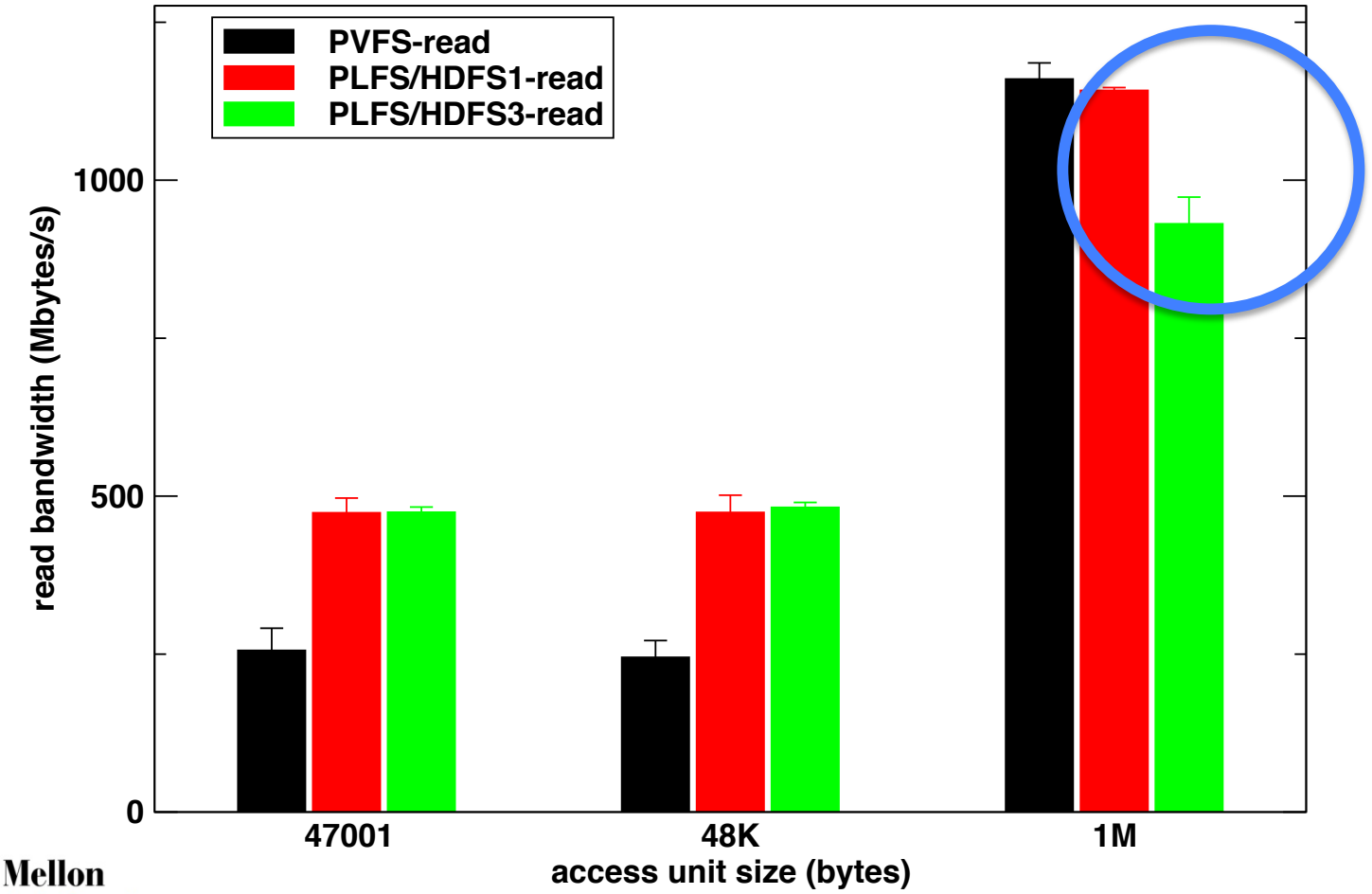
PLFS/HDFS Read Bandwidth

- HDFS with small access size benefits from PLFS log grouping



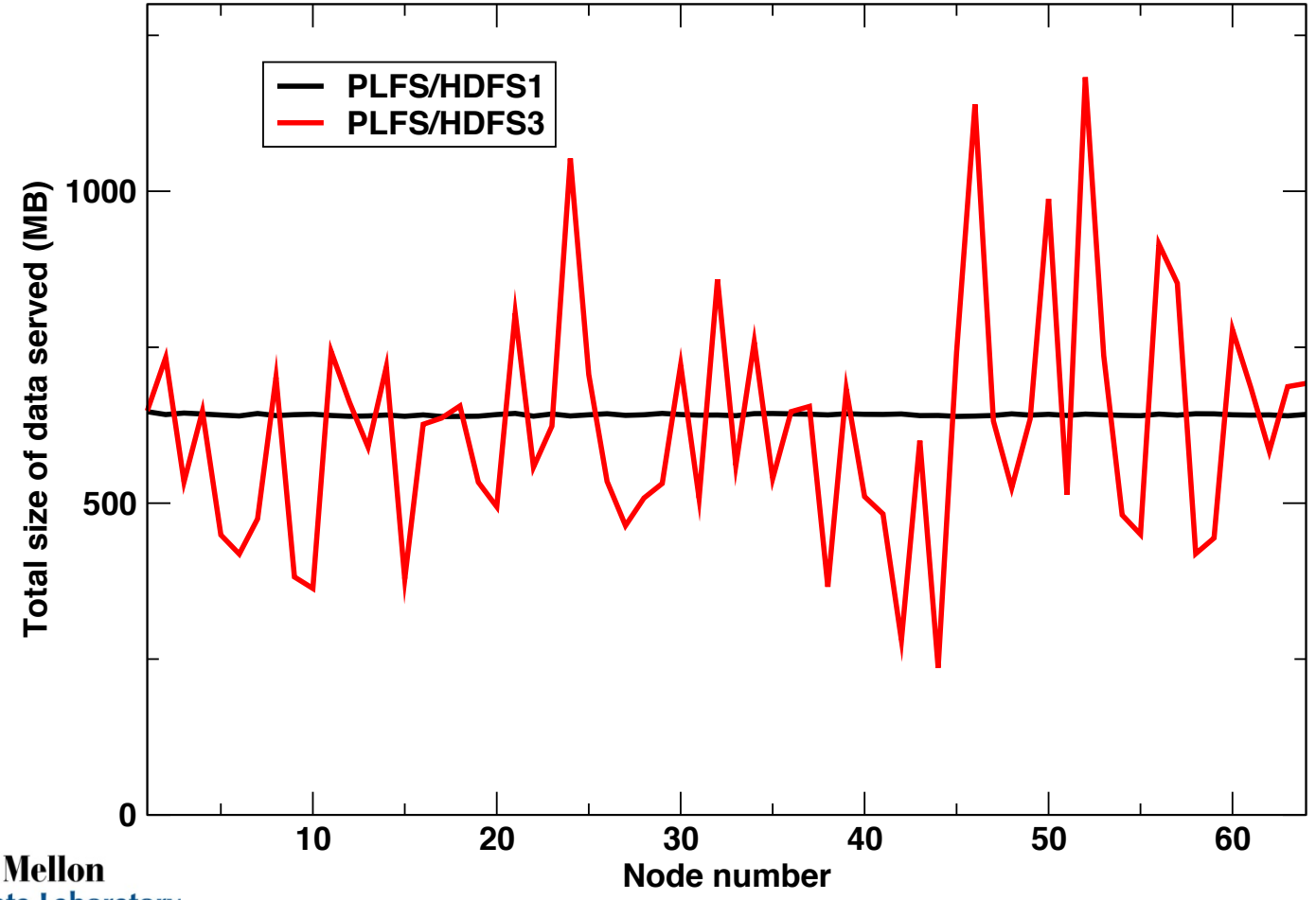
PLFS/HDFS Read Bandwidth

- HDFS3 with large access size suffers imbalance



HDFS 1 vs 3: I/O Scheduling

- Network counters show HDFS3 read imbalance



I/O Store Status

- Rewrote initial I/O Store prototype
  - Production-level code
  - Multiple concurrent instances of I/O Stores
    - Re-plumbed entire backend I/O path
    - Prototyped POSIX, HDFS, PVFS stores
      - IOFSL done by EMC
- Regression tested at LANL
- I/O Store now part of PLFS released code
  - https://github.com/PLFS

Conclusions

- PLFS extensions for workload transformation:
  - Logical FS interface
    - Not just container logs; packing small files, burst buffer
  - I/O Store layer
    - Non-POSIX backends (HDFS, IOFSL, PVFS)
    - Compression, write buffering, IO forwarding
  - Container index extensions
- PLFS is open source, available on github
  - http://github.com/plfs
  - Developer email: plfs-devel@lists.sourceforge.net