PLFS: The Parallel Log-Structured Filesystem

Milo Polte

John Bent, Garth Gibson, Gary Grider, Ben McClelland,
Paul Nowoczynski, James Nunez, Meghan Wingate

Carnegie Mellon Parallel Data Laboratory
Los Alamos National Laboratory
Pittsburgh Supercomputing Center

IRHPIT INSTITUTE FOR RELIABLE HIGH PERFORMANCE INFORMATION TECHNOLOGY

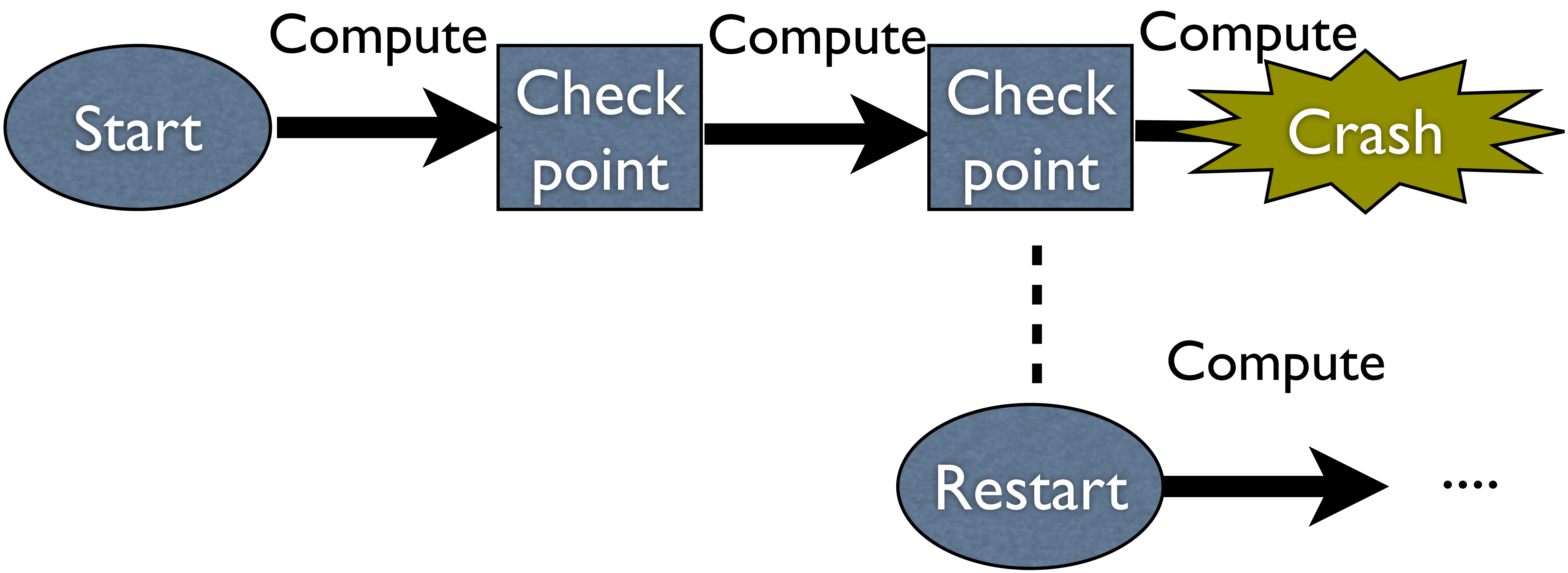
Carnegie Mellon
Parallel Data Laboratory

LA-UR-08-07314, LA-UR 09-02117, LA-CC-08-104

Computational Science = Always Hungry

- LANL's Roadrunner
  - Petaflop machine, tens of thousands of cores
- Building bigger machines isn't free
- Higher processor count
  - More Failures
  - Bigger, more frequent checkpoints
  - Also bigger simulation/visualization output
- Having time to compute requires fast I/O!

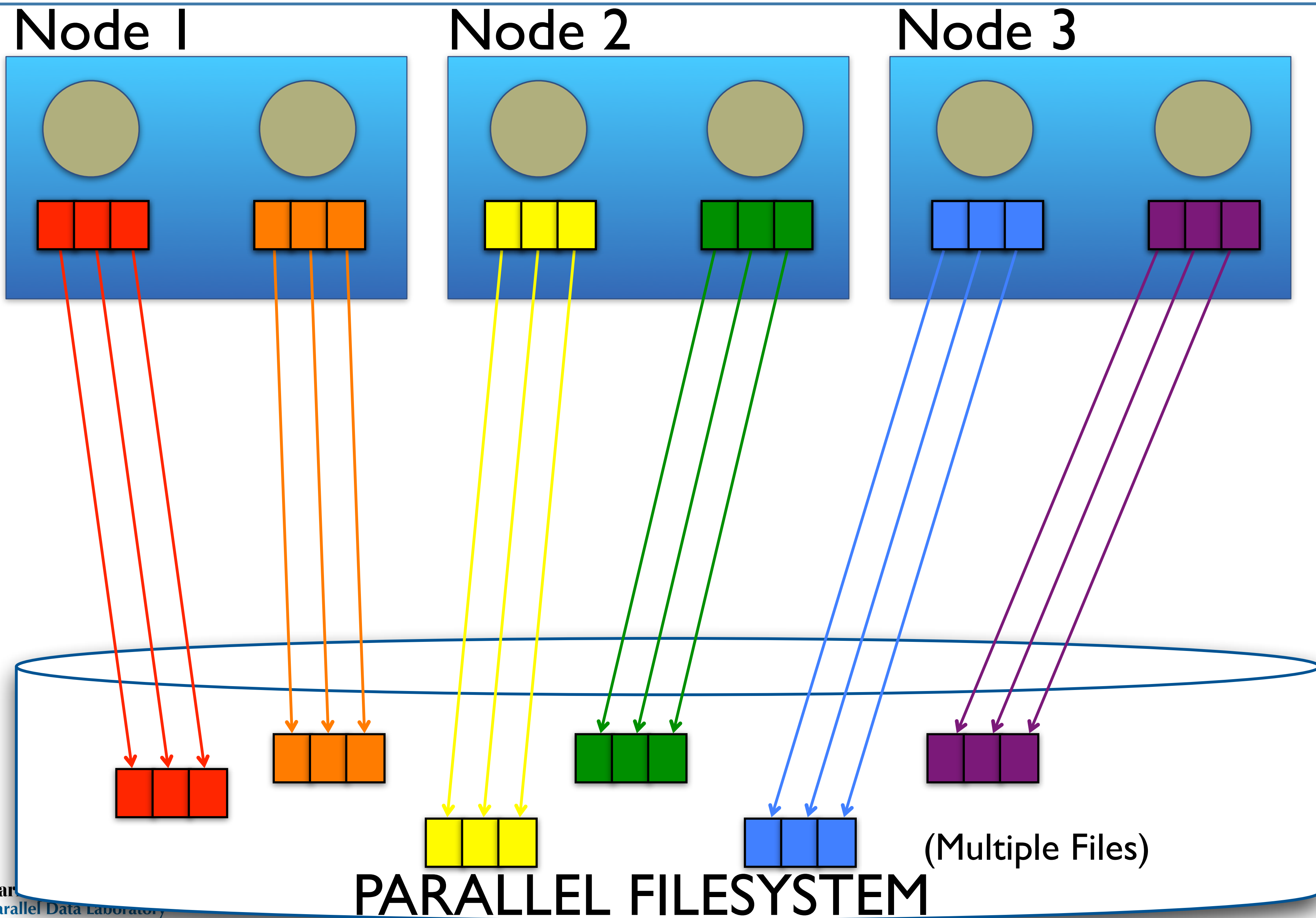
Lifetime of a Scientific App



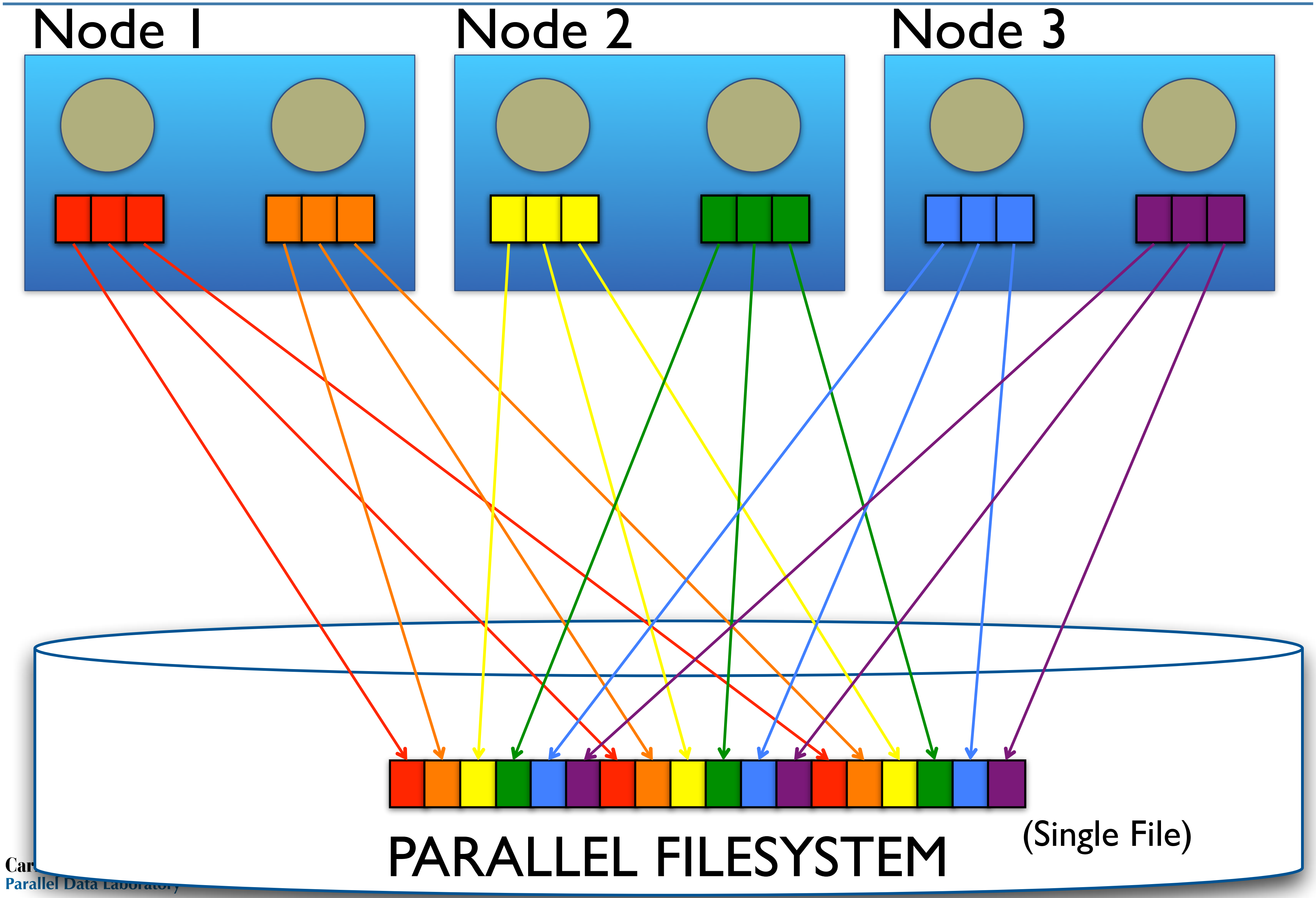
Parallel Apps = Parallel Writes

- Writes are concurrent
- Tens of thousands of concurrent writes
  - Challenge for a filesystem
- Two common write patterns
  - **N-N, N-1**

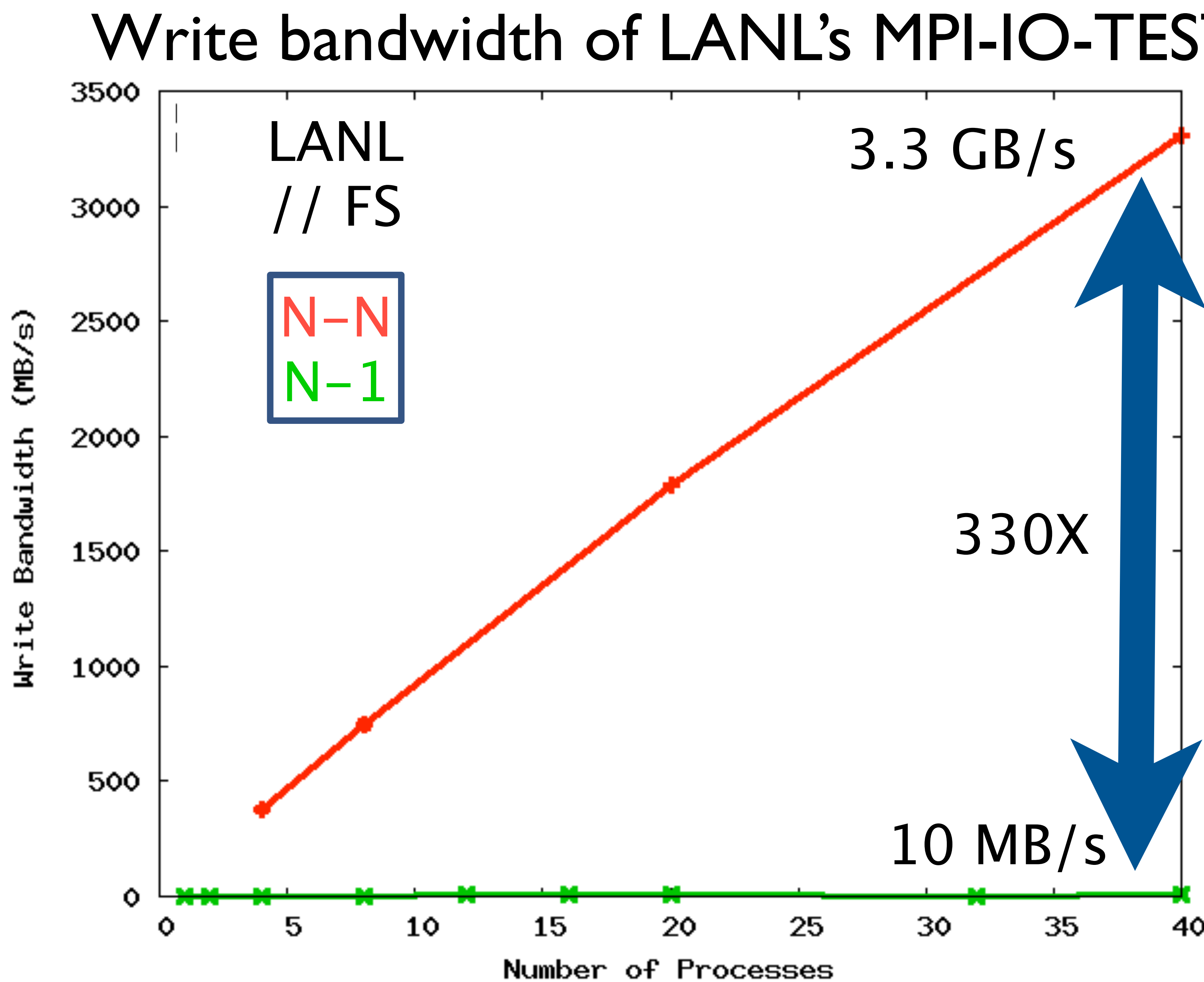
N-N File IO



N-1 File IO



N-1 Concurrent Writing Doesn't Scale



N-1: What's the hold up?

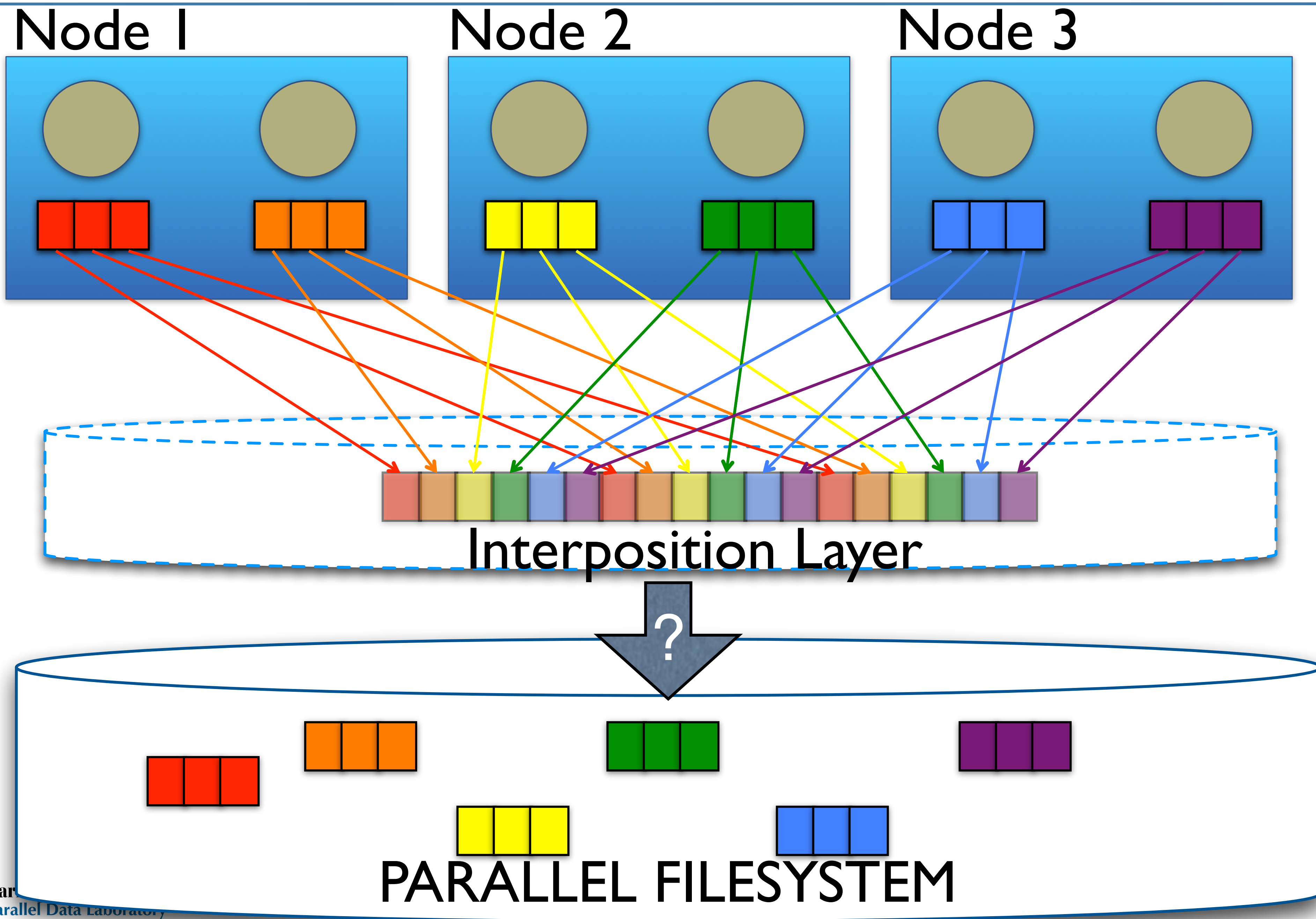
- Contention within a single object
  - Locking, safety
- Small strides, small writes
  - May be misaligned
  - Stripe alignment
  - RAID parity read-modify-write disaster

Ditch N-1? Not so fast....

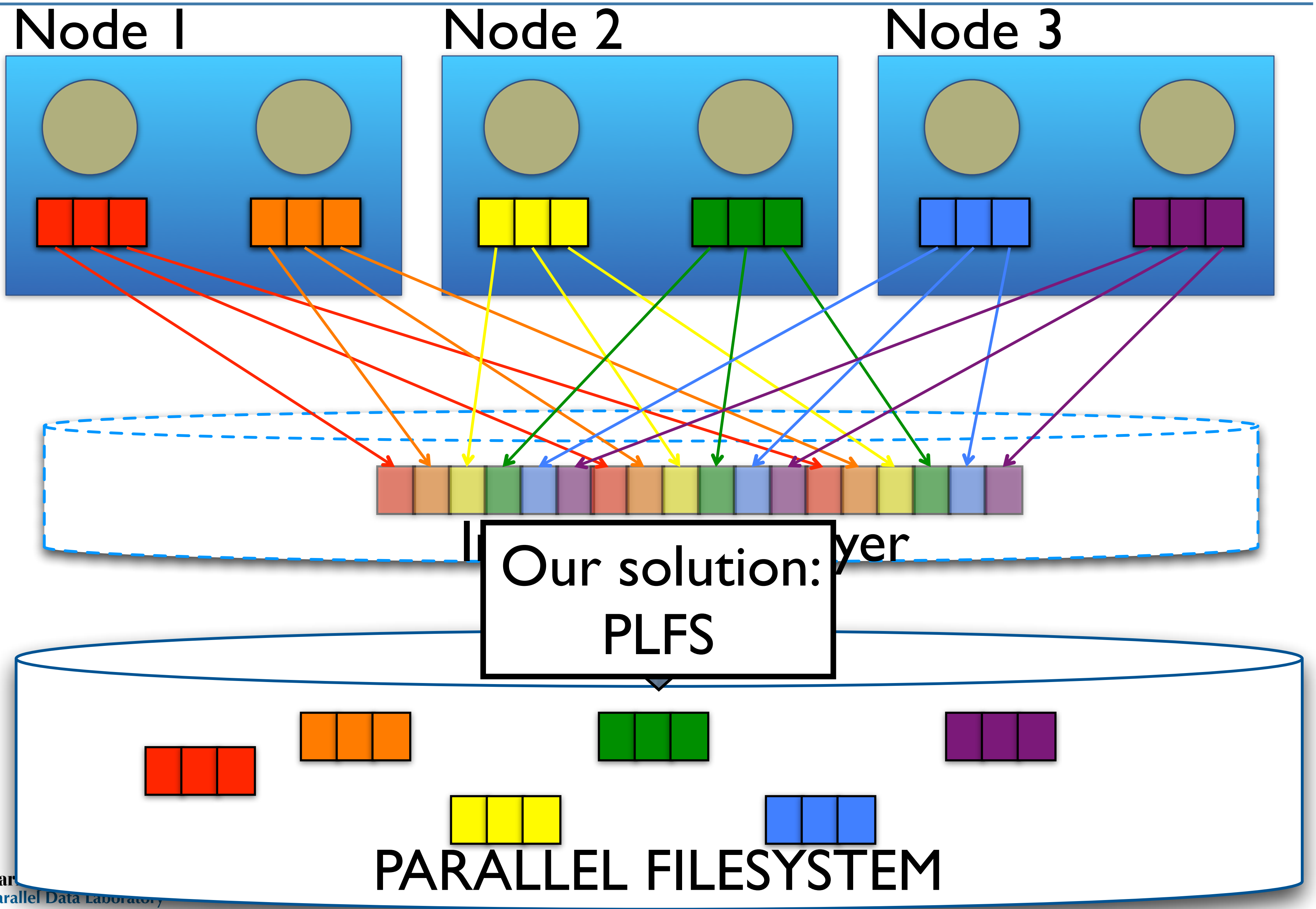
- At HPC sites (LANL) many old codes use N-1
  - "Untouchables"

- Newly written codes still choosing N-1
  - 2 of 8 open science applications on Roadrunner
  - Common scientific formatting libraries are N-1
- Many benchmarks as well
  - Half the PIO Benchmarking Consortium
  - Designed to represent real apps

How can we convert N-1 to N-N?



How can we convert N-1 to N-N?



Outline

- Motivation
- PLFS Design
- Evaluation
  - Write Speeds in PLFS
  - Read Speeds in PLFS
  - Metadata Rates in PLFS
- Future Work
- Conclusions

Design of a checkpoint interposition layer

Requirement	Solution
Extreme parallelism	Decouple writers to individual files
Fast, efficient writes	Write in a log structured manner
No application changes	Expose POSIX filesystem interface
Portable across filesystems	Implement as a 'stackable' filesystem
Low comp. node footprint	Use existing parallel FS storage

Using PLFS

- PLFS is implemented as a FUSE filesystem
- Mounted on top of an existing parallel filesystem
- Example: On every node, mount as

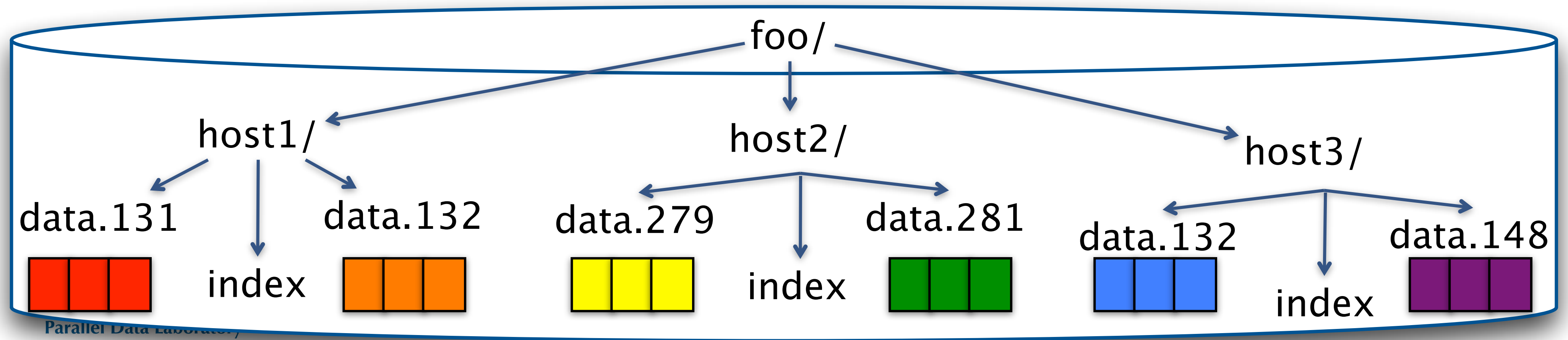
```
$ plfs ~/mnt/plfs -plfs_backend=/mnt/scratch
```

  - Checkpoints write to `~/mnt/plfs`
  - PLFS stores data in parallel filesystem `/mnt/scratch`

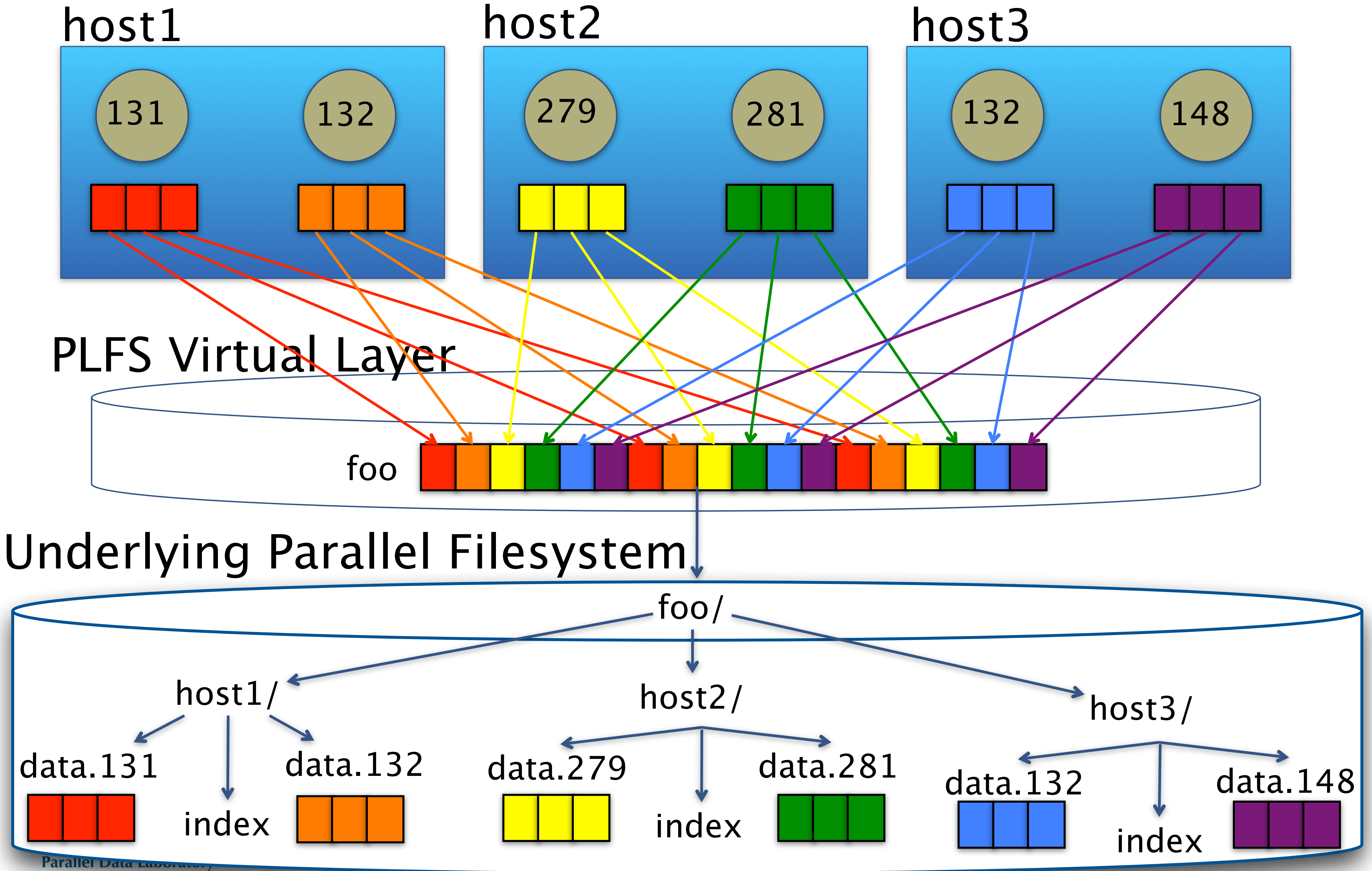
Applications write checkpoints to PLFS the same as they wrote to the parallel filesystem

PLFS Decoupling

- Processes open a file ‘foo’ in PLFS mount point
  - PLFS mkdir’s directory ‘foo/’ in underlying filesystem
  - PLFS mkdir’s ‘foo/<hostname>/’ in underlying filesystem
- Processes start writing to ‘foo’ in PLFS
  - PLFS opens a data log per writer, begins appending
  - PLFS writes a index file per host



Review: Decoupled Layout

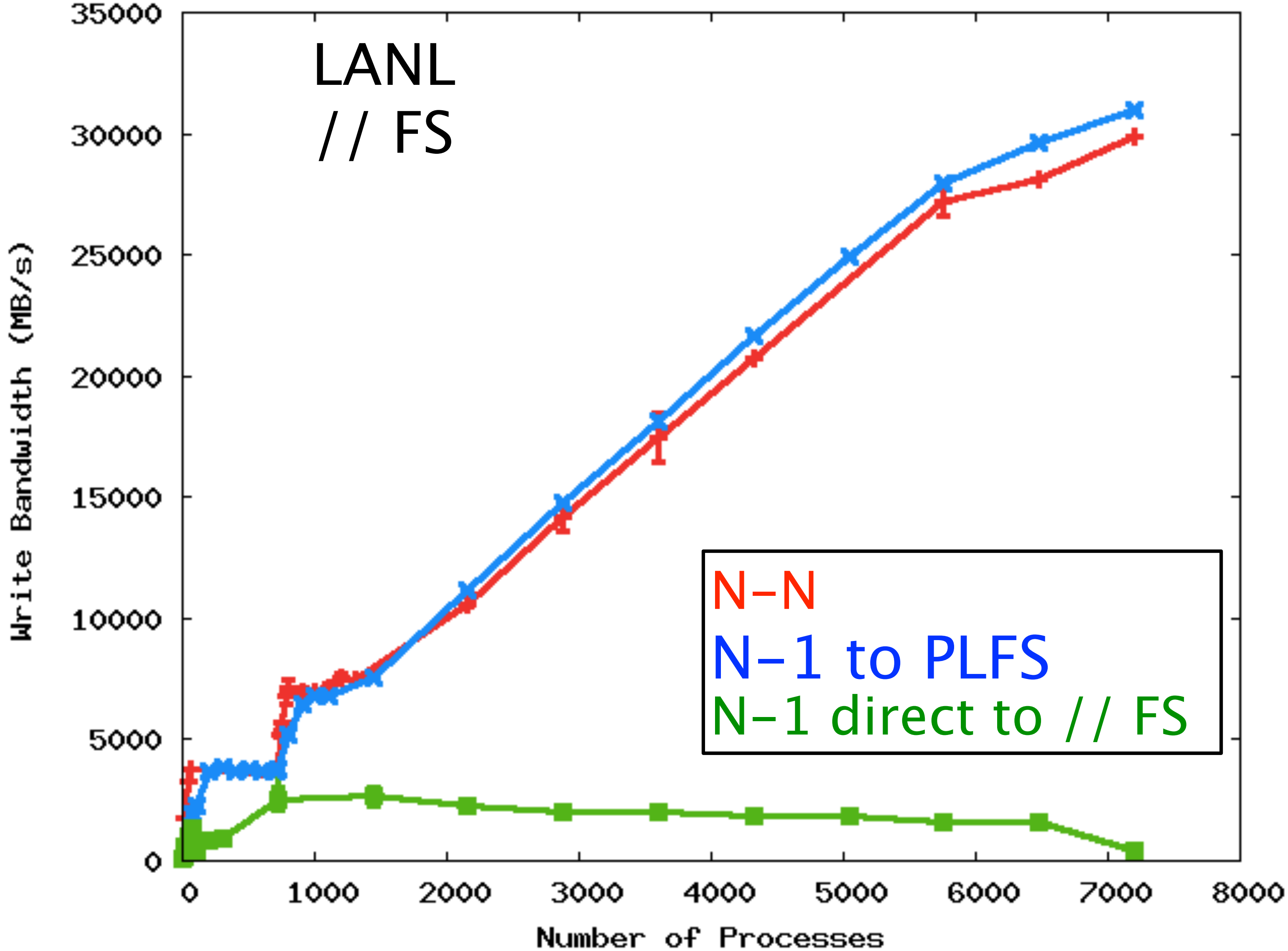


Outline

- Motivation
- PLFS Design
- Evaluation
  - Write Speeds in PLFS
  - Read Speeds in PLFS
  - Metadata Rates in PLFS
- Future Work
- Conclusions

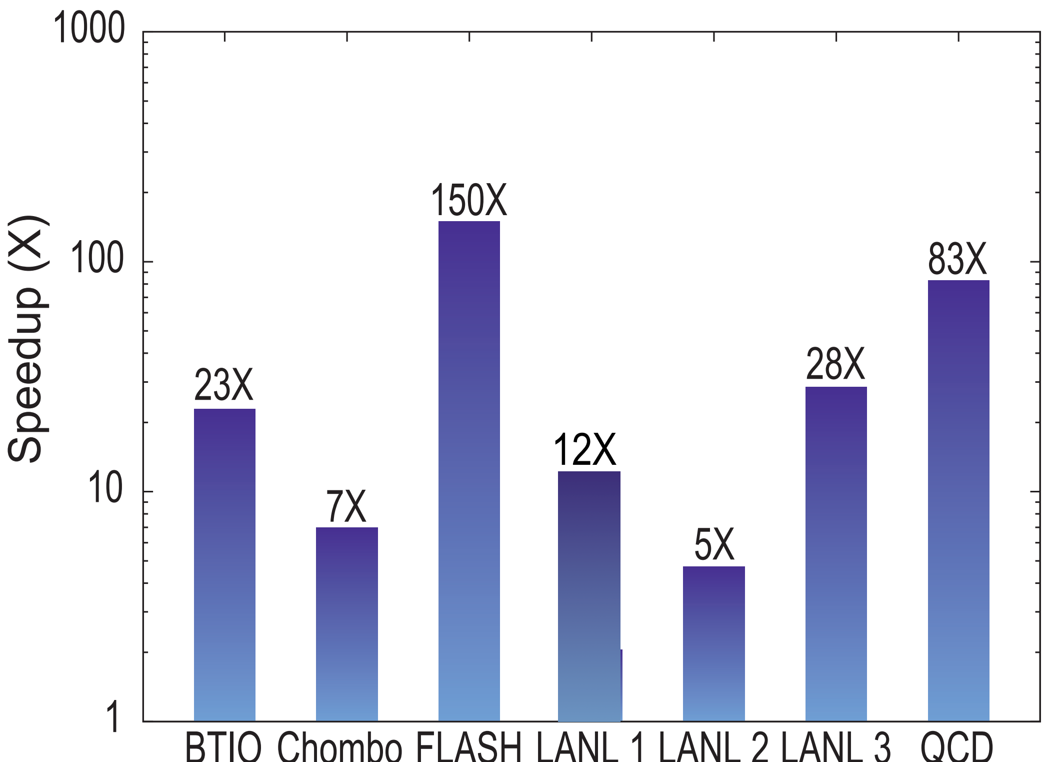
PLFS converts N-1 to N-N speeds

Write bandwidth of LANL's MPI-IO-TEST



Writes Evaluated Extensively

- GPFS, Lustre, Panfs filesystems
- Applications and IO Kernels
- Synthetic Checkpoint Benchmarks



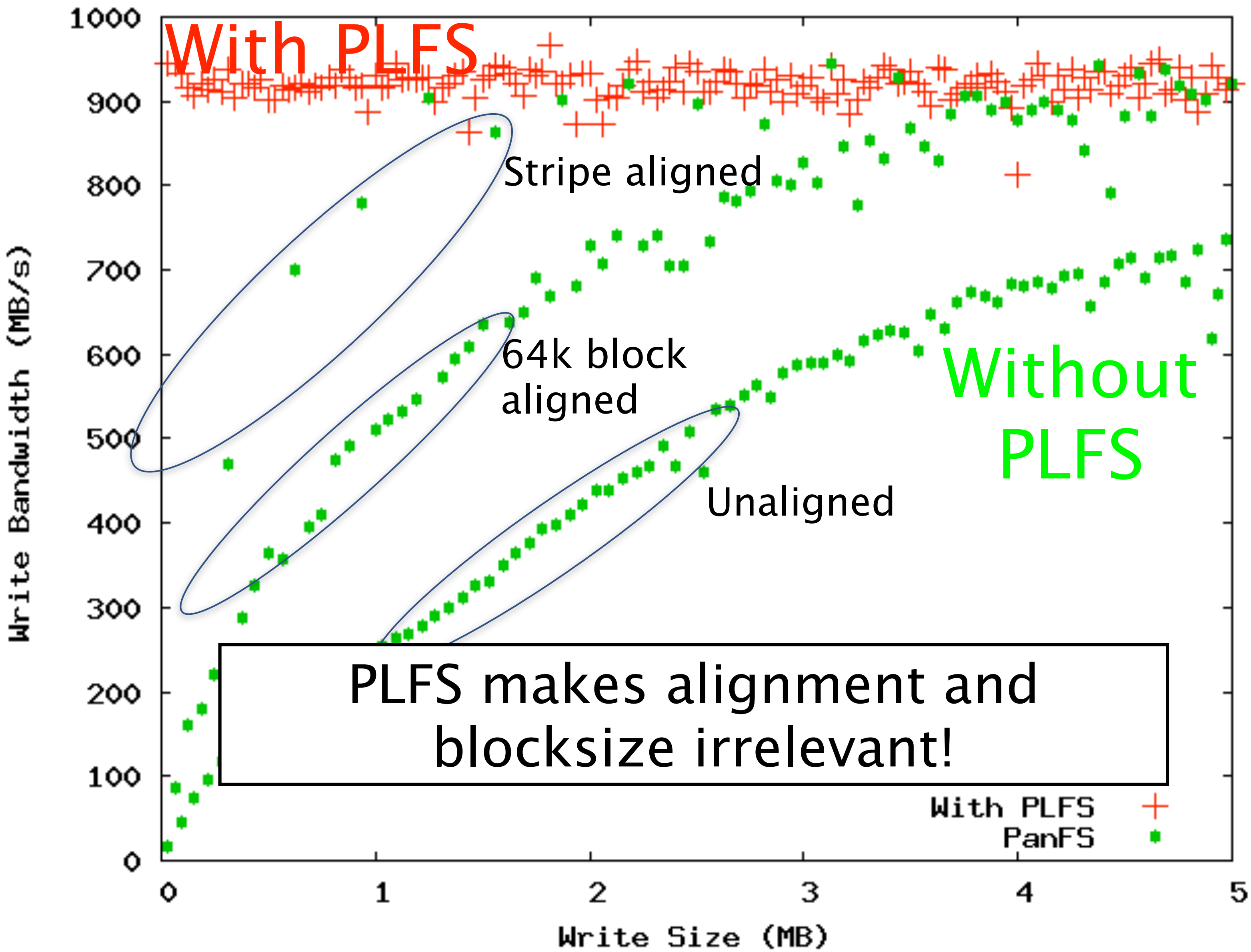
5-150x improvements

Bigger improvements with more writers

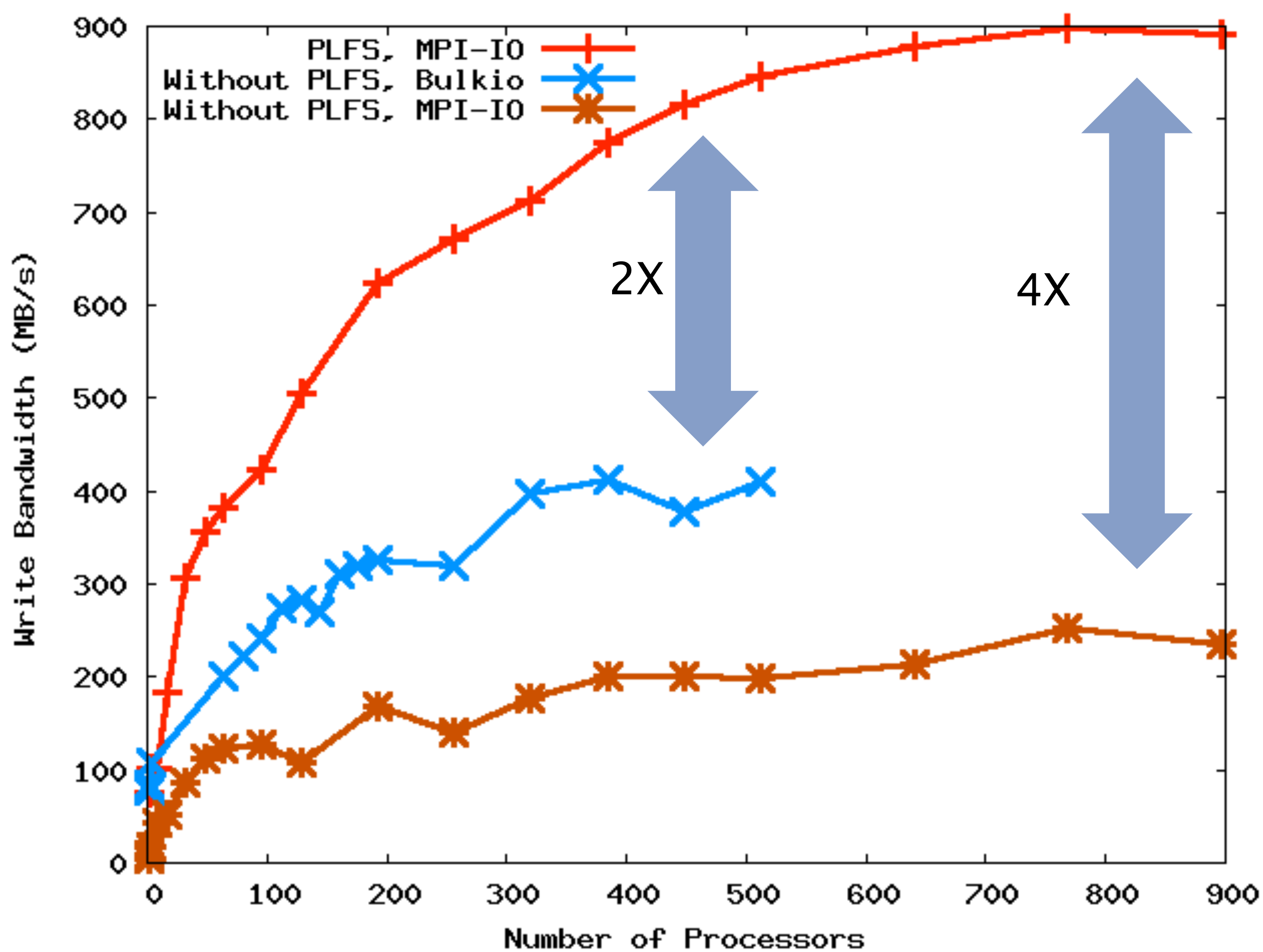
Alignment and Write Size

- Small strided writes induce contention
  - Hurt caching, buffering
- Misaligned writes use resources inefficiently
  - False sharing
  - RAID parity read-modify-write problem

LBNL's PatternIO



"Zero-Effort" Improvement For Real Apps



LANL App that simulates wiping out the dinosaurs with a meteor

Bulkio was a 10k line library written just to improve this app

PLFS is 3k lines, benefits from the FUSE approach

Outline

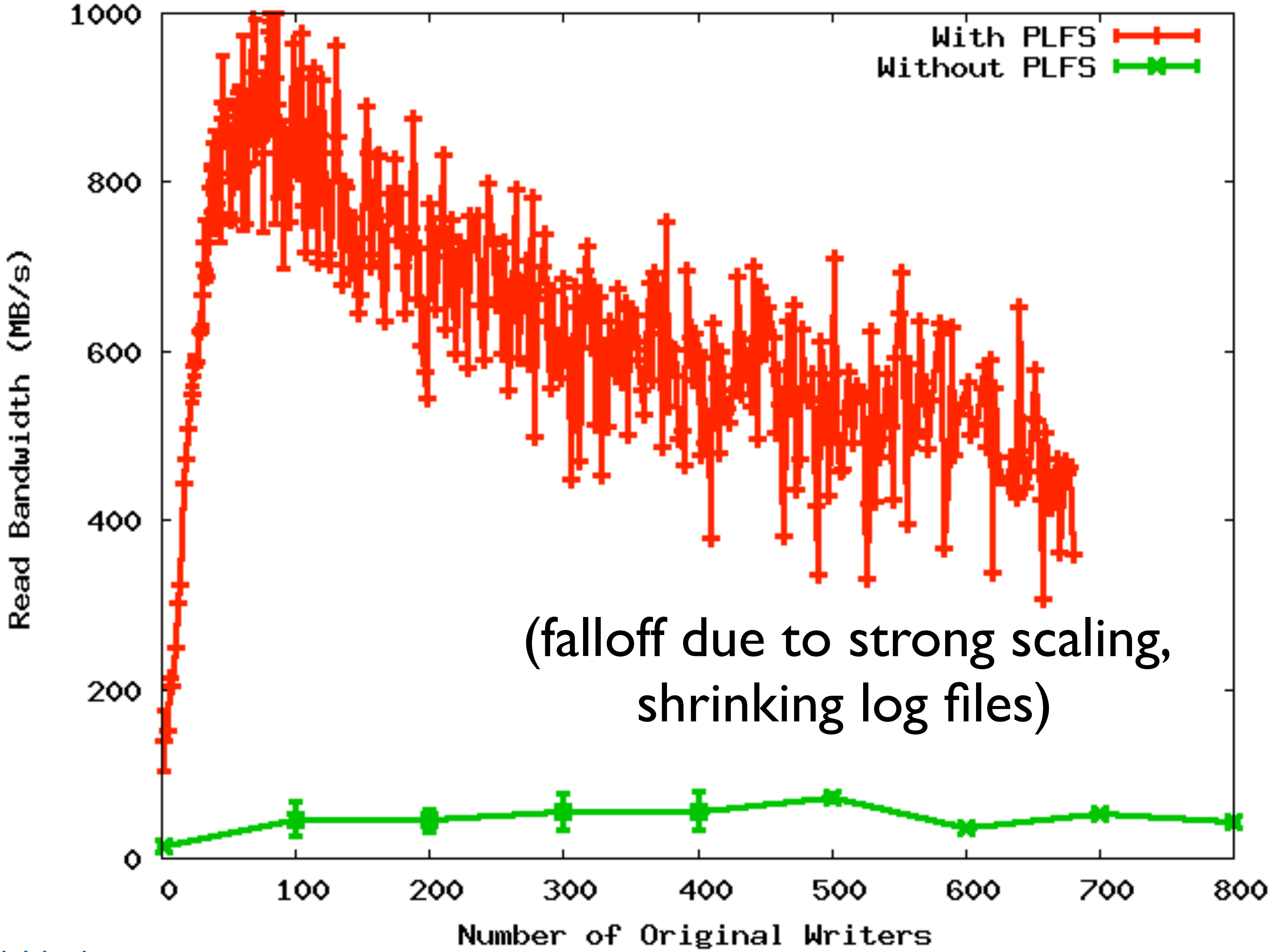
- Motivation
- PLFS Design
- Evaluation
  - Write Speeds in PLFS
  - Read Speeds in PLFS
  - Metadata Rates in PLFS
- Future Work
- Conclusions

What about the read path?

- Checkpoint is 'write once, read maybe'
- PLFS readers read in indices, remap requests
- We're writing in a log structured way
  - Can't this hurt reads?

Read Speed Improved by PLFS?

Read bandwidth of LANL's MPI-IO-TEST

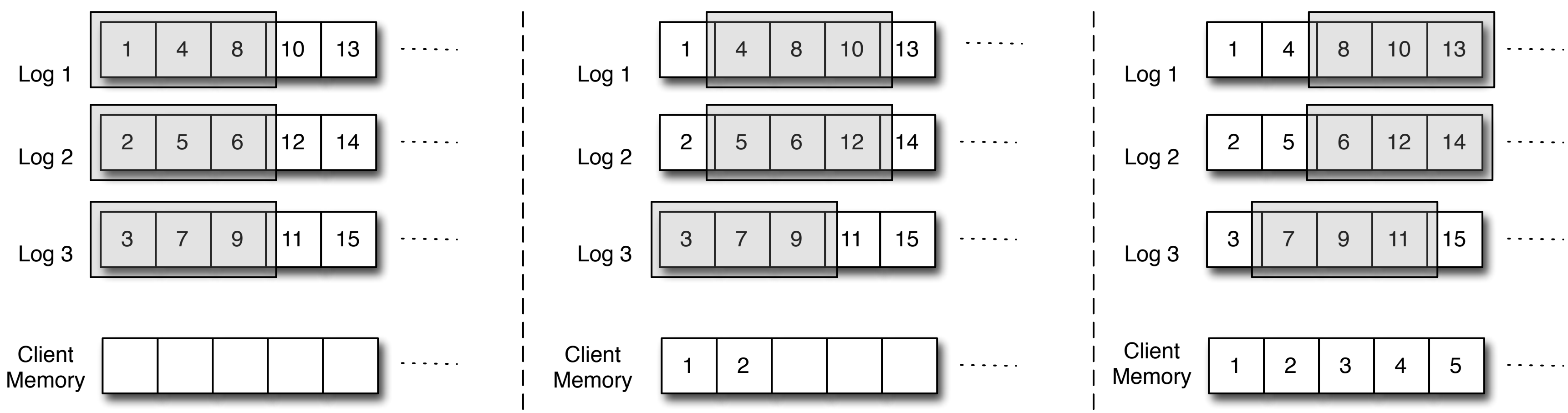


Read Speed Explanation (I)

- Checkpoints don't write randomly
- Examined write traces of evaluated applications
- In every case, processes wrote to monotonically increasing logical offsets
- Creates offset-sorted logs

Read Speed Explanation (II)

- Checkpoints aren't read randomly either
  - Restart and archive read sequentially
- PLFS reads from many files at once
- Gets more filesystem resources than N-1
- Next byte always in read-ahead buffer of some file



A client reading sequentially from offset-sorted logs

Carnegie Mellon
Parallel Data Laboratory

Outline

- Motivation
- PLFS Design
- Evaluation
  - Write Speeds in PLFS
  - Read Speeds in PLFS
  - Metadata Rates in PLFS
- Future Work
- Conclusions

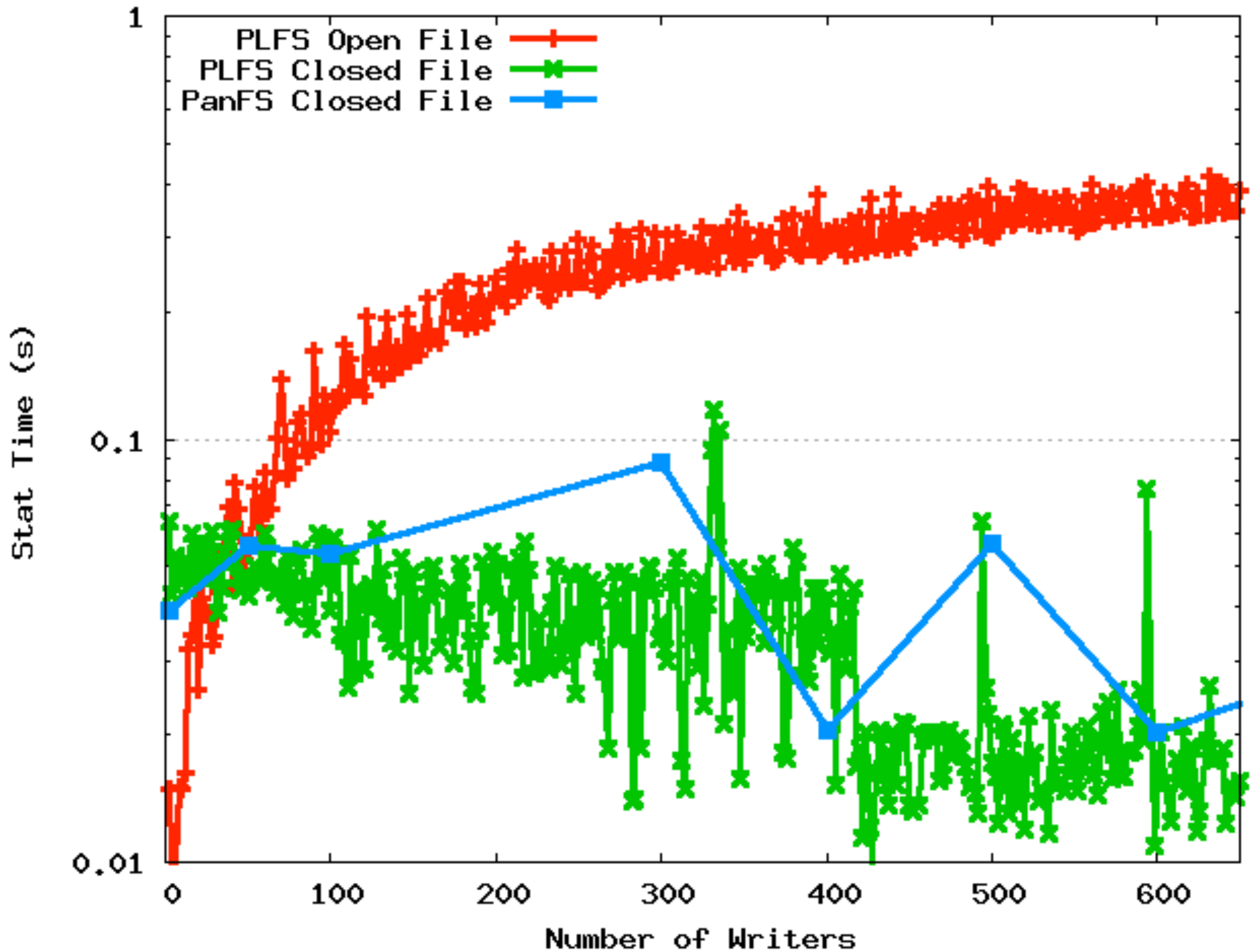
Metadata operations performance

- Recall: A PLFS file is really a directory
- Chmods, Chowns, Chgrps, Utimes, etc.
  - Use the container or a special access file
- Stat can use access file for permissions, ownership
- What about size? Modified time?
- Have to stat every data log?
- Expensive with thousands of independent logs!

Stat Optimization

- Containers have special metadata subdirectory
- On close, writers make `metadata/host.B.L.T`
  - B = blocks of capacity
  - L = last offset (i.e. file size)
  - T = timestamp of last write (mtime)
- Stat can now be implemented with a readdir
- If writers are still open, have to use slow path

Stat Rates



PLFS Open File
PLFS Closed File
PanFS Closed File
Stat Time (s)
1
0.1
0.01
0
100
200
300
400
500
600
Number of Writers

Remaining Challenges and Future Work

- First open invokes thousands of sub-file creates
- Index reprocessing overhead in read-write mode
- Odd read patterns? Data analysis?
- Faster stat of open files
  - `ls -l` of growing file

Future Work: PLFS + HDFS

- ‘Cloud’ filesystems gaining prevalence
- High resilience but often lack important semantics
- HDFS:
  - No concurrent writers
  - No reopen for write
  - Could be achieved by ‘decoupling’ every open
- Use PLFS to add semantics to Cloud Filesystems
- See me at poster

Conclusions

- Drastically improves performance of N-1 checkpointing
- Works on multiple parallel filesystems
- No application, filesystem modifications
- Does not penalize checkpoint reading
- Potential to enrich semantics of cloud filesystems
- Downloadable at: http://sourceforge.net/projects/plfs/