BLUE WATERS

SUSTAINED PETASCALE COMPUTING

Scalable I/O

Ed Karrels, edk@Illinois.edu

NCSA

GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

CRAY

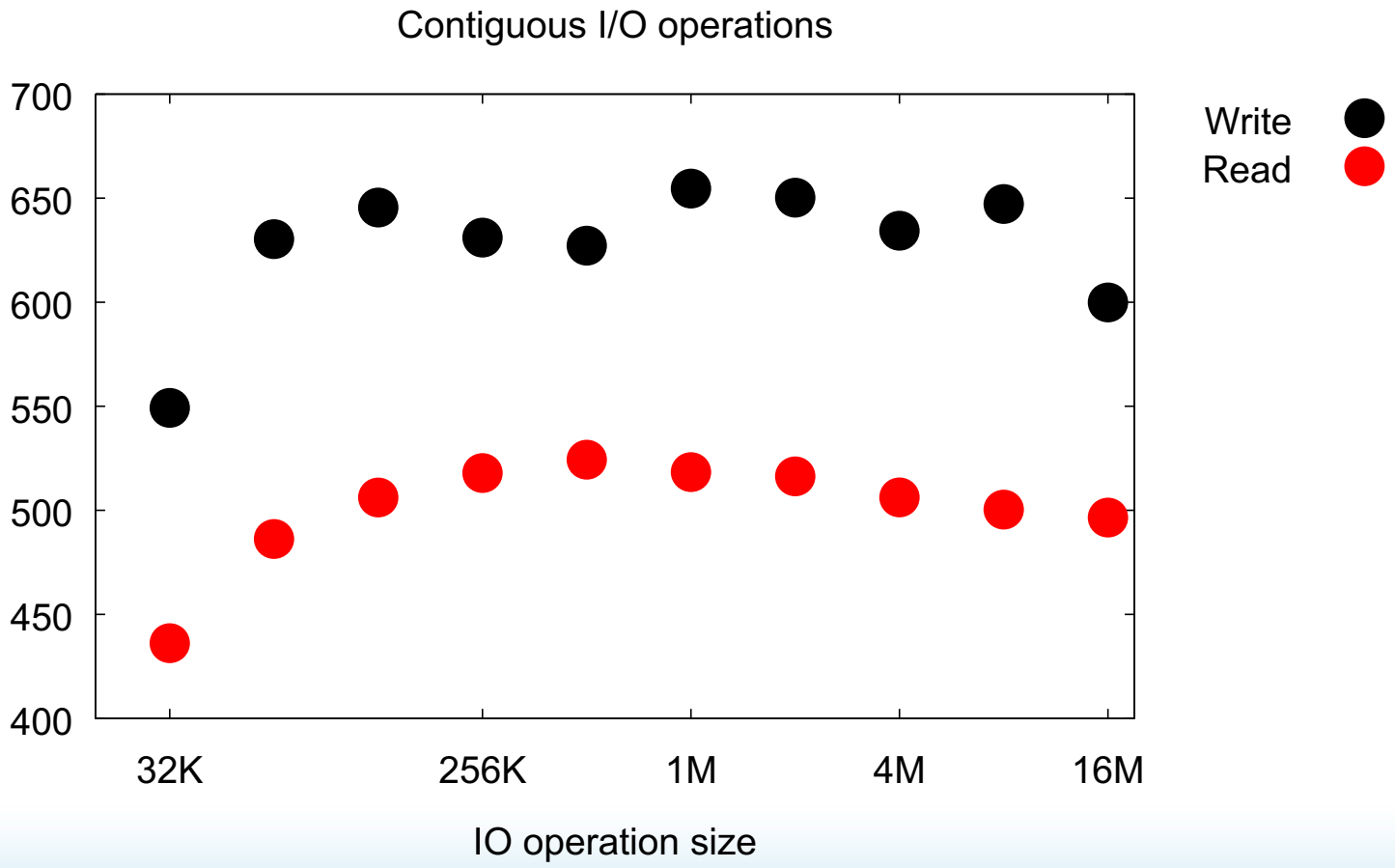
I/O performance overview

- Main factors in performance
- Know your I/O
- Striping
- Data layout
- Collective I/O

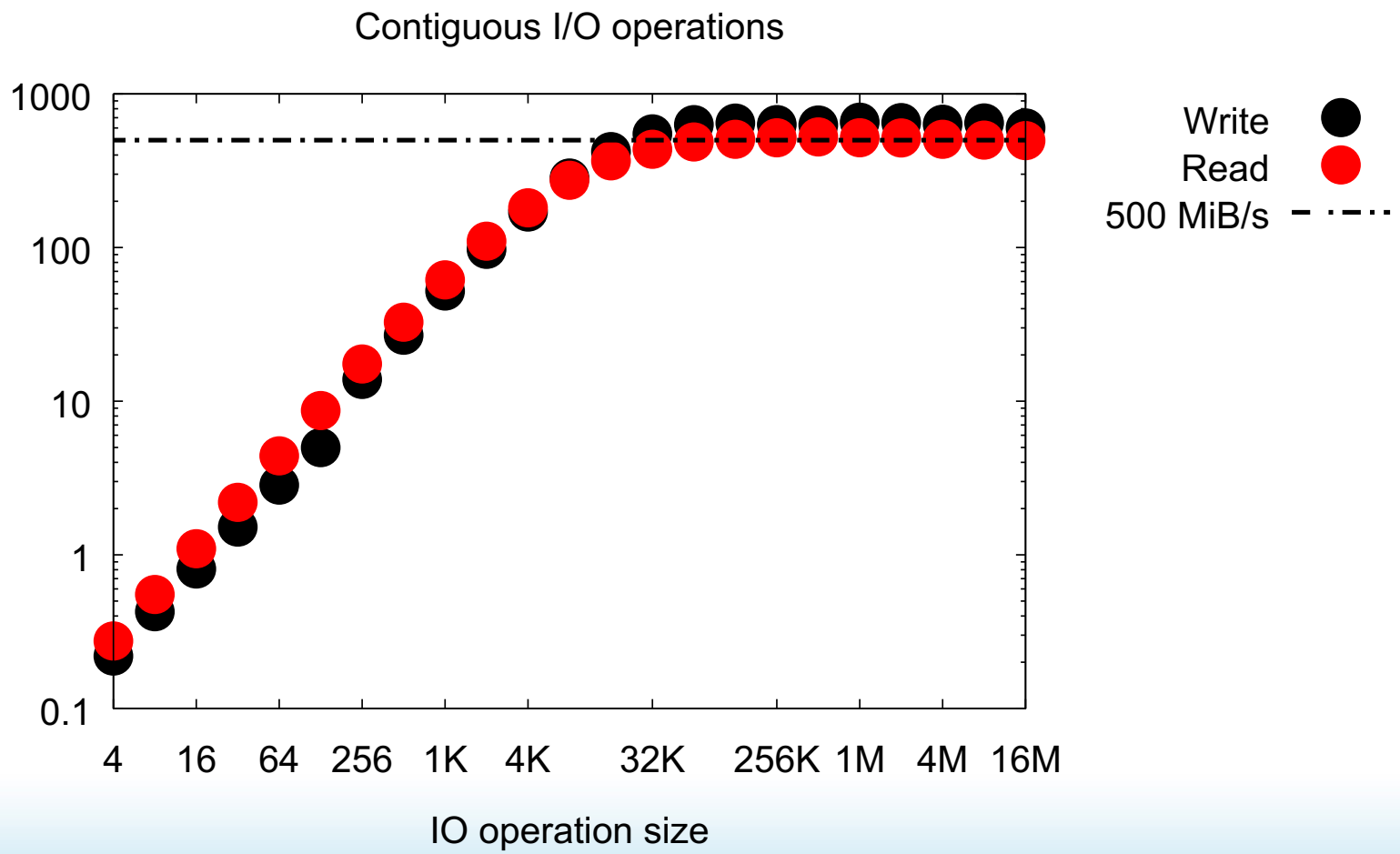
I/O performance

- Length of each basic operation
  - High throughput, high latency
  - Avoid using small reads / writes
- Locality matters
  - Avoid jumping around in the file

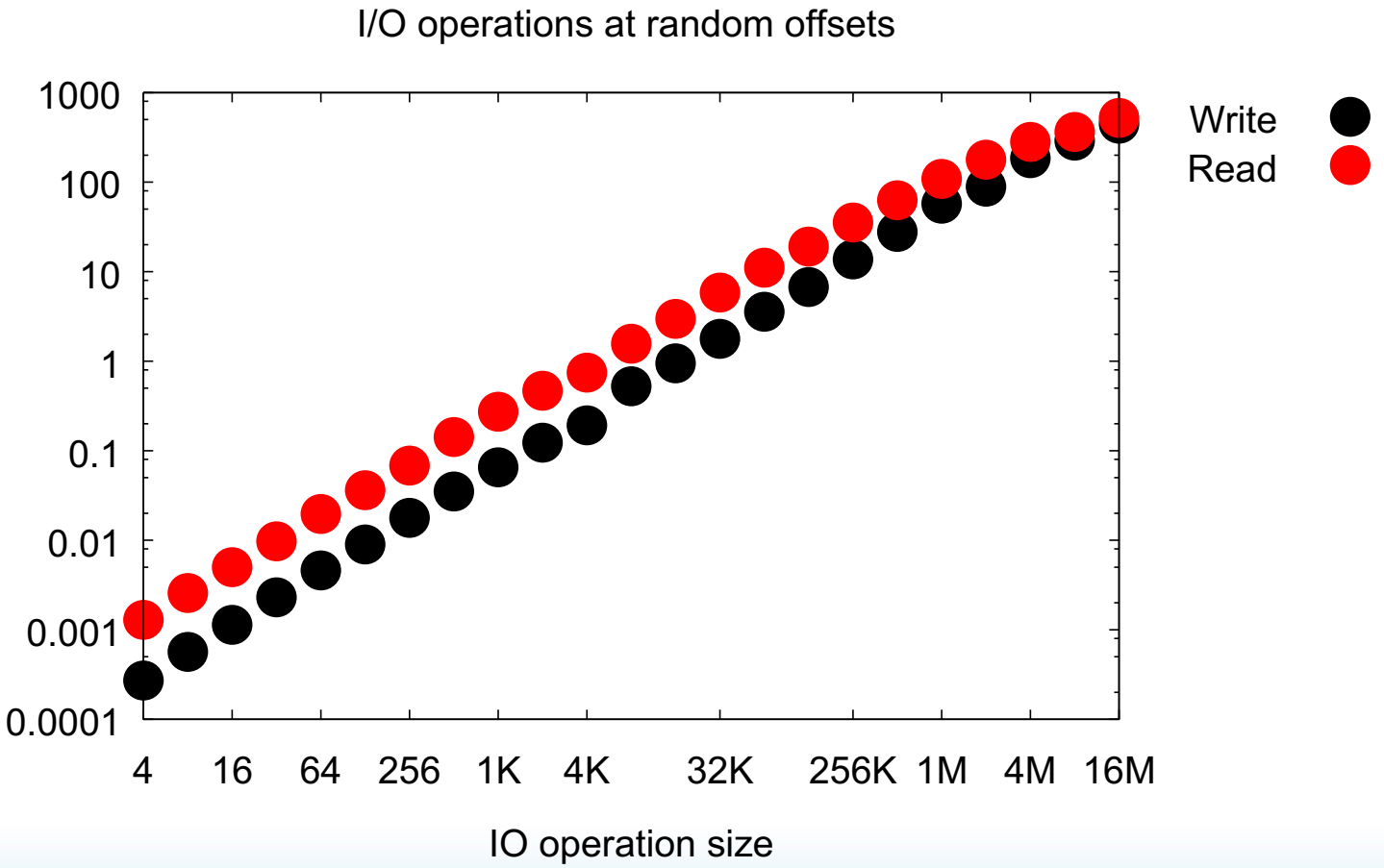
Single process I/O size



Single process I/O size



Single process random access



Know your I/O

- How fast (or slow) is it?
- How is the data organized on disk?
  - N-dimensional array?
  - Text or binary?
- Which subset of the data for each process?

Darshan I/O analysis

- Darshan collects logs of all I/O
- On by default
  - disable with "`module unload darshan`"
- Logs are in

  `/projects/monitoring_data/darshan/YYYY-MM/`
- Only writes logs on normal exit (needs call to MPI_Finalize)

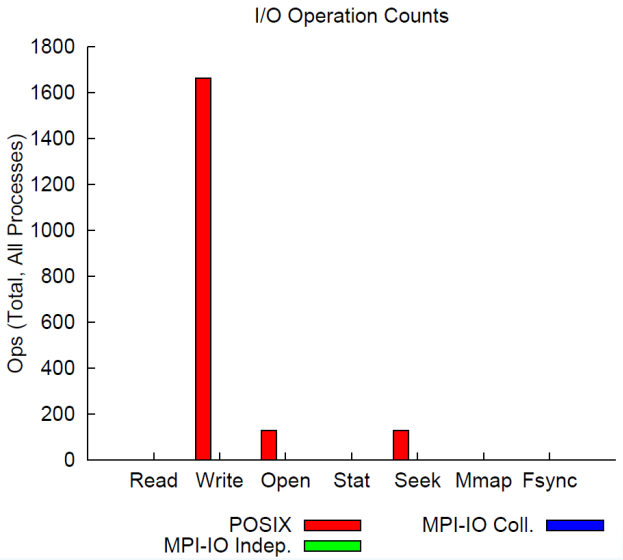
Darshan I/O analysis

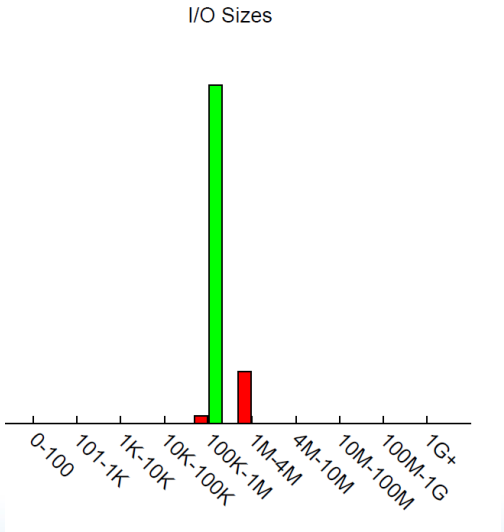
Most Common Access Sizes

access size	count
288	859963393
1179648	193537
700416	27648
144	198

File Count Summary

type	number of files	avg. size	max size
total opened	99	2.4G	2.5G
read-only files	96	2.5G	2.5G
write-only files	1	8.6M	8.6M
read/write files	0	0	0
created files	0	0	0





Darshan commands

- darshan-job-summary.pl
  - Generate PDF report of the whole job
- darshan-summary-per-file.sh
  - Generate per-file report
- darshan-parser
  - Extract lots of raw data

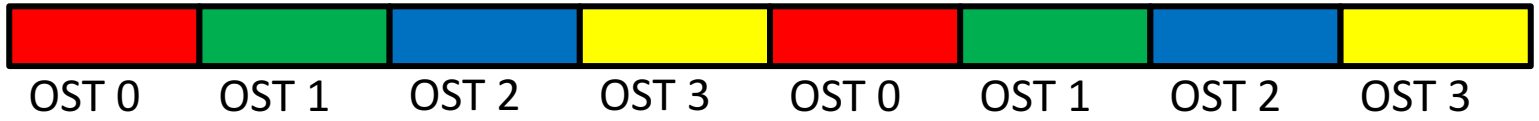
Darshan 3 not backward-compatible with 2

- Darshan 2: *.darshan.gz
- Darshan 3: *.darshan

```
module swap darshan/3.1.3 darshan/2.3.0.1
module unload gnuplot/5.0.5
```

Parallel I/O striping

- Blue Waters uses Lustre file system
- 360 OSTs
  - Essentially 360 independent file servers
- Striping parameters for each file
  - Size: length of each stripe
  - Count: number of file servers to use



Striping with Lustre

- `lfs getstripe <file or dir>`
  - Print striping parameters for a file

```
$ lfs getstripe foo4
foo4
lmm_stripe_count:   4
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  287
        obdidx           objid           objid           group
           287        59540013      0x38c822d               0
            11        65961401      0x3ee7db9               0
           151        65962343      0x3ee8167               0
            71        65963432      0x3ee85a8               0
```

Striping with Lustre

- `lfs setstripe –s <size> -c <count> <file or dir>`
- Directories
  - New files will inherit striping parameters
- Files
  - Set at creation time
  - `lfs setstripe` on a nonexistent file creates it

```
lfs setstripe –s 1M -c 16 foo16
cp foo1 foo16
```

Set striping in MPI IO

```
MPI_Info info;
MPI_Info_set(info, "striping_factor", "64");
MPI_Info_set(info, "striping_unit", "1048576");
MPI_File_open(..., info, ...);
```

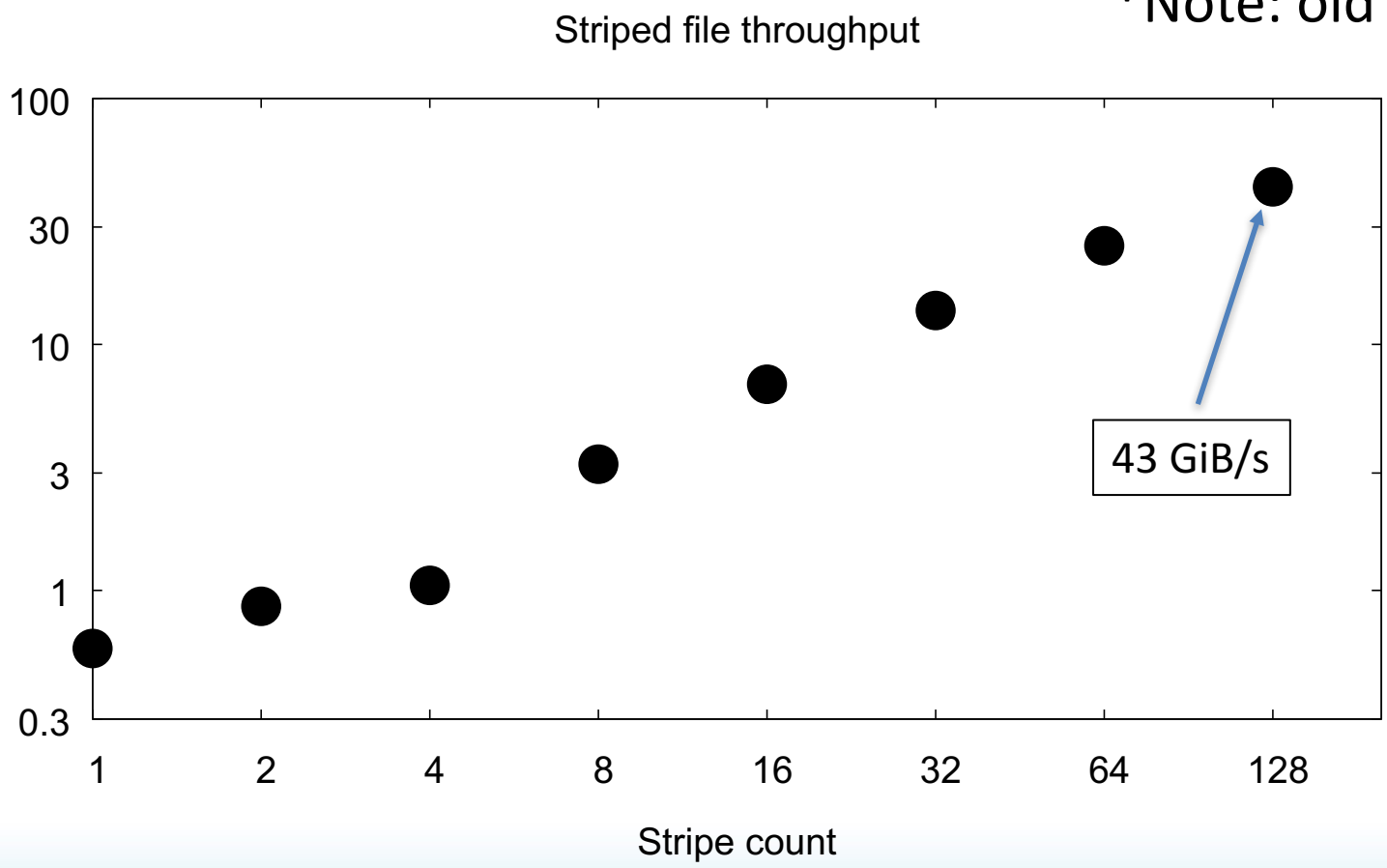
Striping on Blue Waters

- Maximum stripe count
  - home / projects: 36
  - scratch: 360
- Can't set > 36 on scratch directly (bug?)
  - Solution: set on directory, inherit

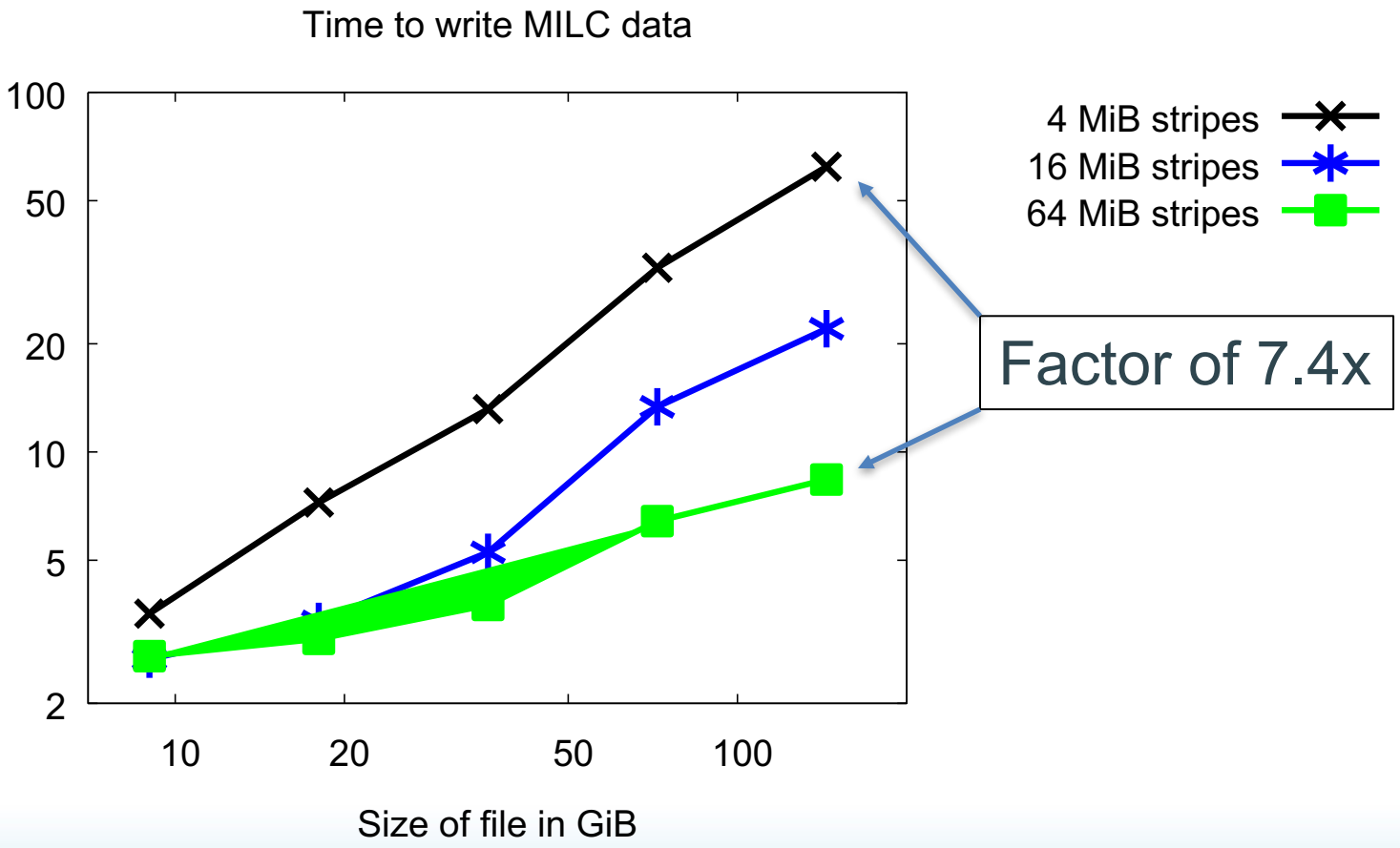
```
lfs setstripe -c 360 mydata/
touch mydata/foo
```

Single file, striped

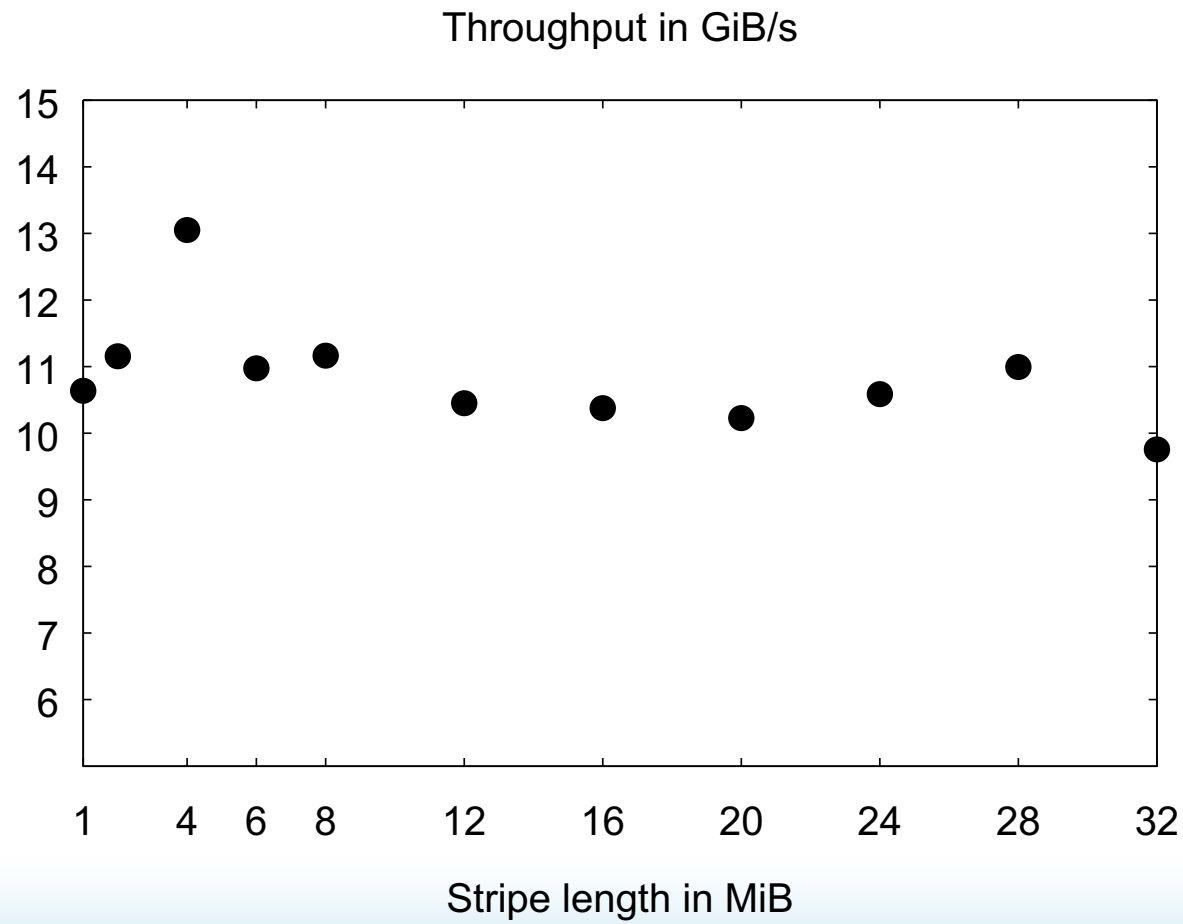
*Note: old data



Stripe length – bigger is better



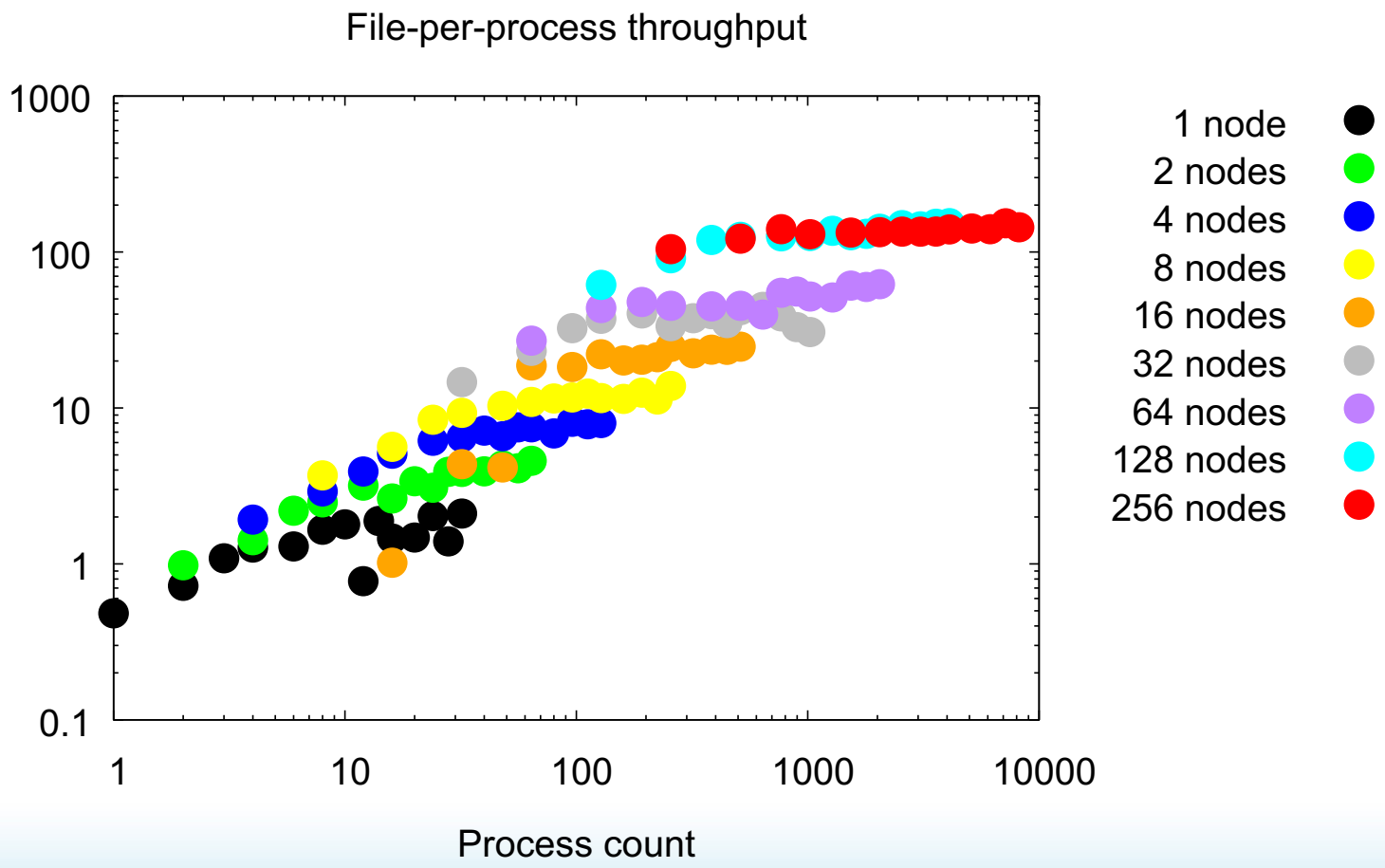
Stripe length – doesn't matter?



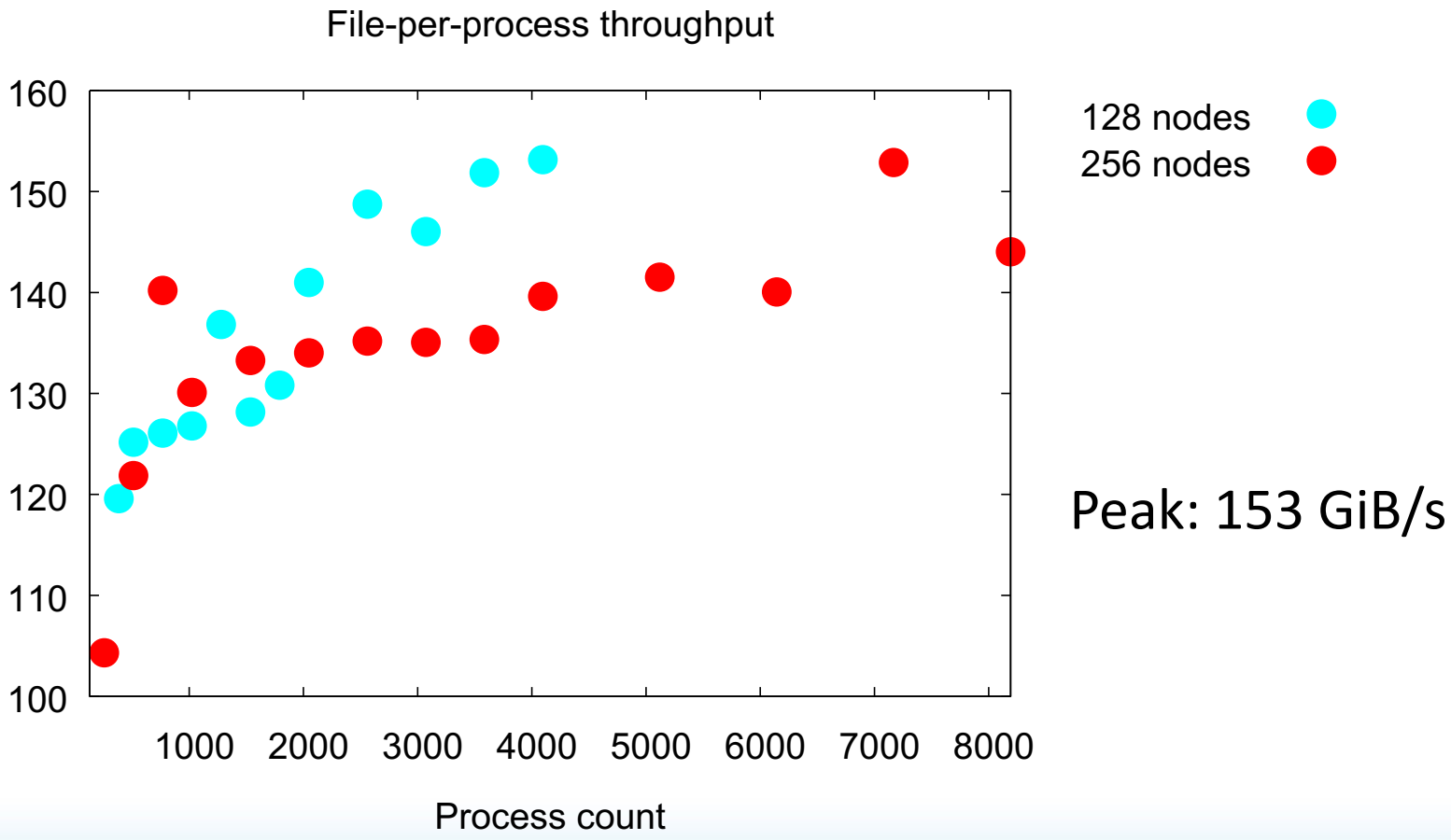
Striping with many files

- Leave stripe count = 1
  - Full file on one file server
- Each one assigned to a random file server

One file per process



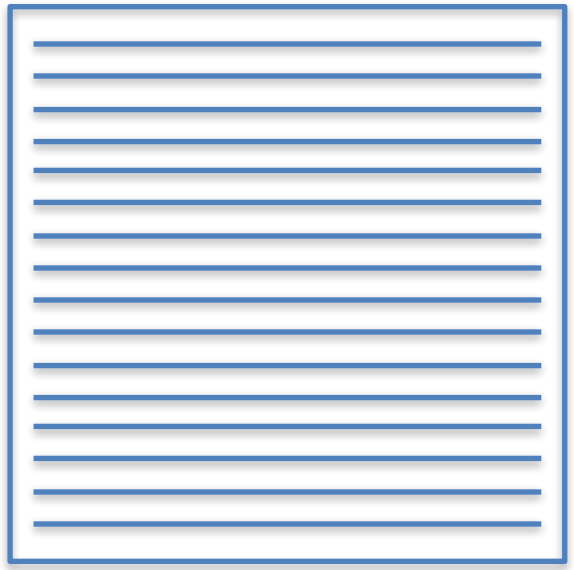
One file per process



Peak: 153 GiB/s

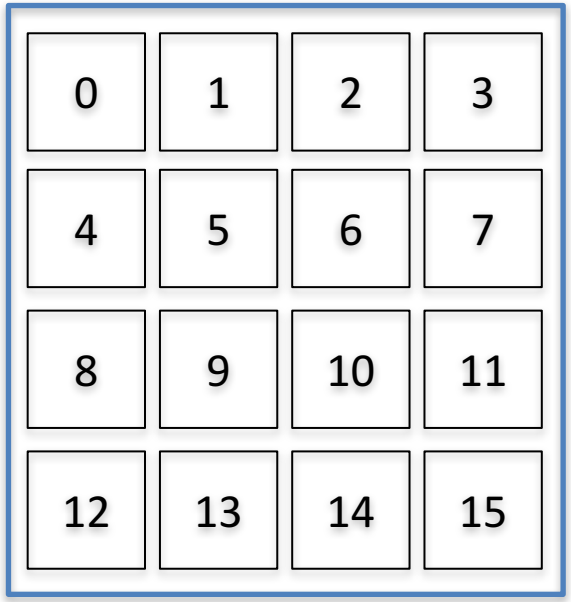
Data layout

- Common pattern: N-dimensional array



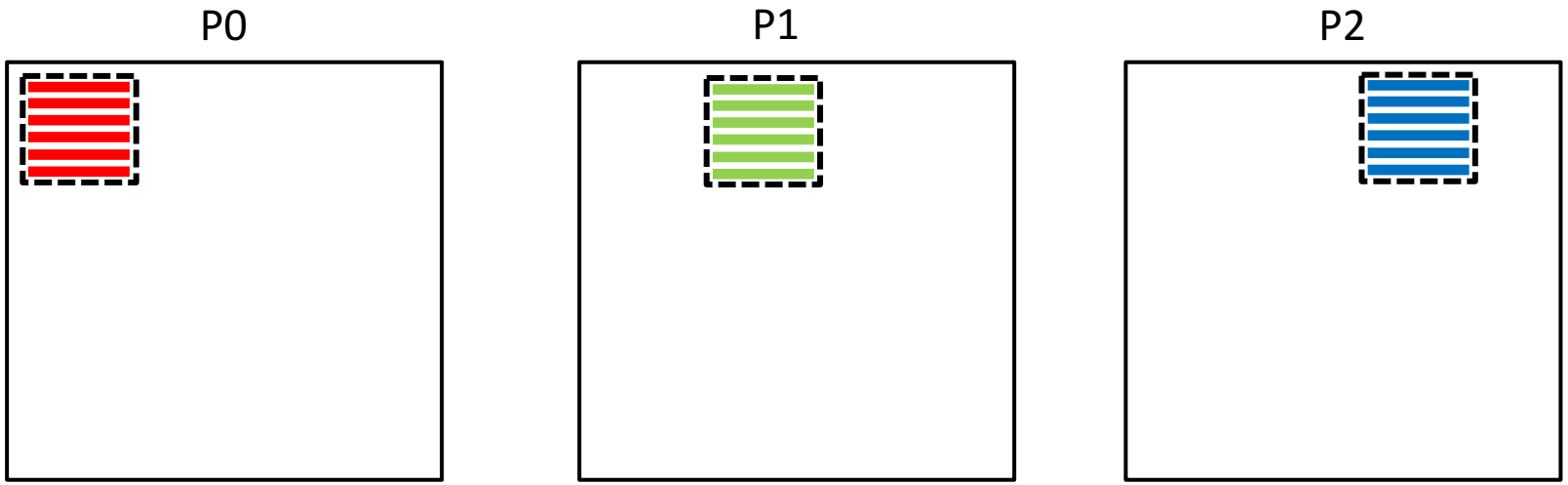
Data layout

- Tile on each process



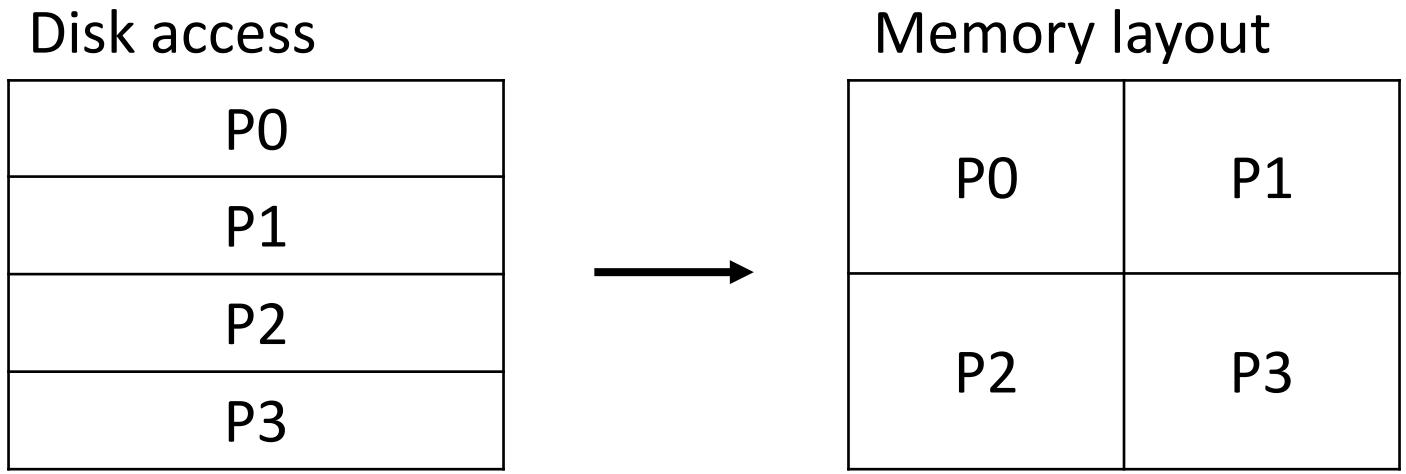
I/O for a tile

- Many small accesses (eek!)



Tile solution: collective I/O

- Small number of large I/O ops
- Redistribute data in memory



Collective I/O

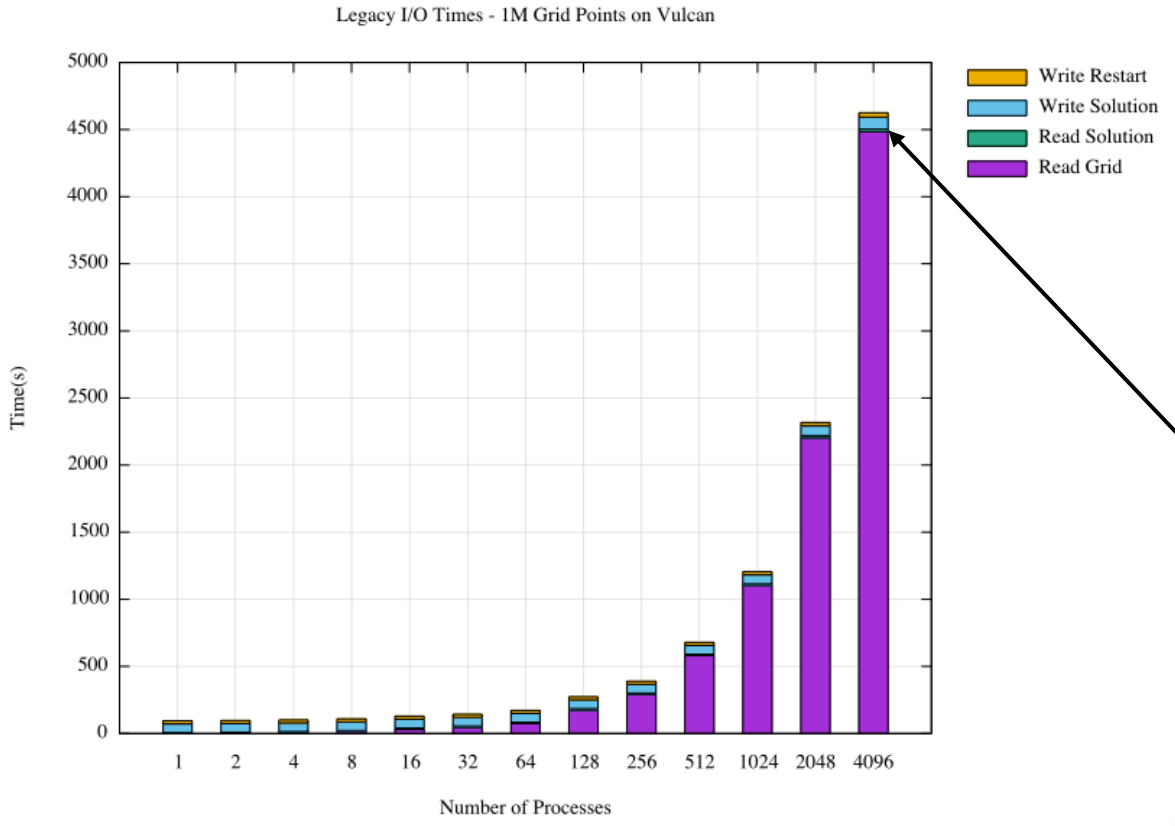
- Built into MPI-IO
  - Describe shape of data with MPI datatypes
  - Use Mesh IO library to make it easier
- Parallel HDF5
- Parallel NetCDF

Mesh I/O

```
int Mesh_IO_read
(MPI_File fh,
 MPI_Offset offset,
 MPI_Datatype etype,
 int file_endian,
 void *buf,
 int ndims,
 const int *mesh_sizes,
 const int *file_mesh_sizes,
 const int *file_mesh_starts,
 int file_array_order,
 const int *memory_mesh_sizes,
 const int *memory_mesh_starts,
 int memory_array_order);
```

- For N-dimensional meshes
- Describe size of full mesh and the submesh you want
- Collective operation
- Matching write function
- github.com/oshkosher/meshio

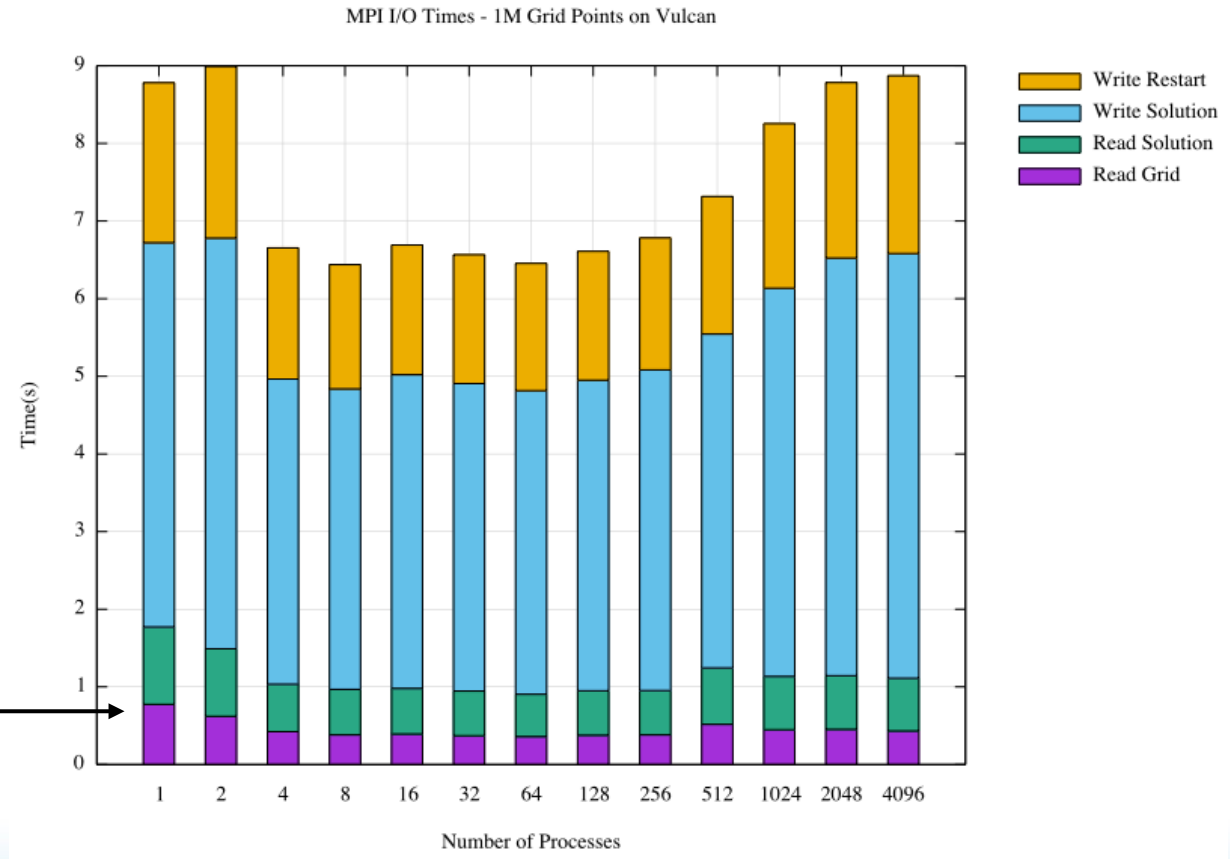
XPACC - tiles without collective I/O



- Constant data size (1 GB)
- Runtime grows with process count
- 4500 seconds on 4096 processes

XPACC - tiles with collective I/O

- Consistently under 1 second



Compression

- Highly compressible data – minimize IO
  - Bioinformatics (AGTCTGTCTTGC…)
- Example file: 40 GiB
  - Compress to 275 MiB (151x)
  - Serial scan in 9.6s (4.2 GiB/s)
- Tools - github.com/oshkosher/bioio
  - zchunk – read in chunks: offset+length
  - zlines – read lines of text: line number

Summary

- All reads/writes at least 64k
- Best performance: many files
- With single file, enable striping
- Use collective I/O to avoid small accesses

Questions / corrections / comments?

edk@Illinois.edu