Optimizing I/O at ALCF: Performance and Best Practices

Rick Zamora
rzamora@anl.gov

ALCF

Argonne NATIONAL LABORATORY

Acknowledgments

Content Contributed by many people @ ALCF

- Paul Coffman, pcoffman@anl.gov
- Kevin Harms, harms@anl.gov
- Venkat Vishwanath, venkat@anl.gov
- Francois Tessier, ftessier@anl.gov
- George Brown, gbrown@anl.gov
- Preeti Malakar, pmalakar@anl.gov

Outline

— Parallel I/O Basics

MIRA

— Mira I/O Architecture (BG/Q – GPFS)
— Optimizing I/O on Mira

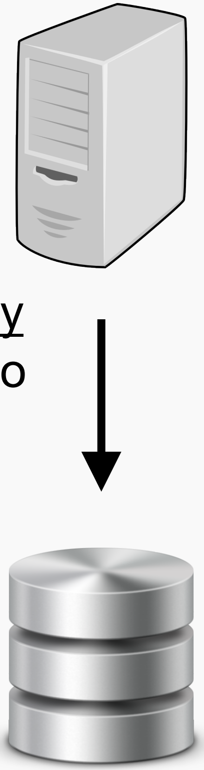
THETA

— Theta I/O Architecture (Cray XC40 – Lustre)
— Lustre File System Basics
— Using Cray MPI-IO
— I/O Profiling Tools on Theta
— Lustre Performance on Theta
— Node Local SSD Utilization on Theta

Parallel I/O Basics

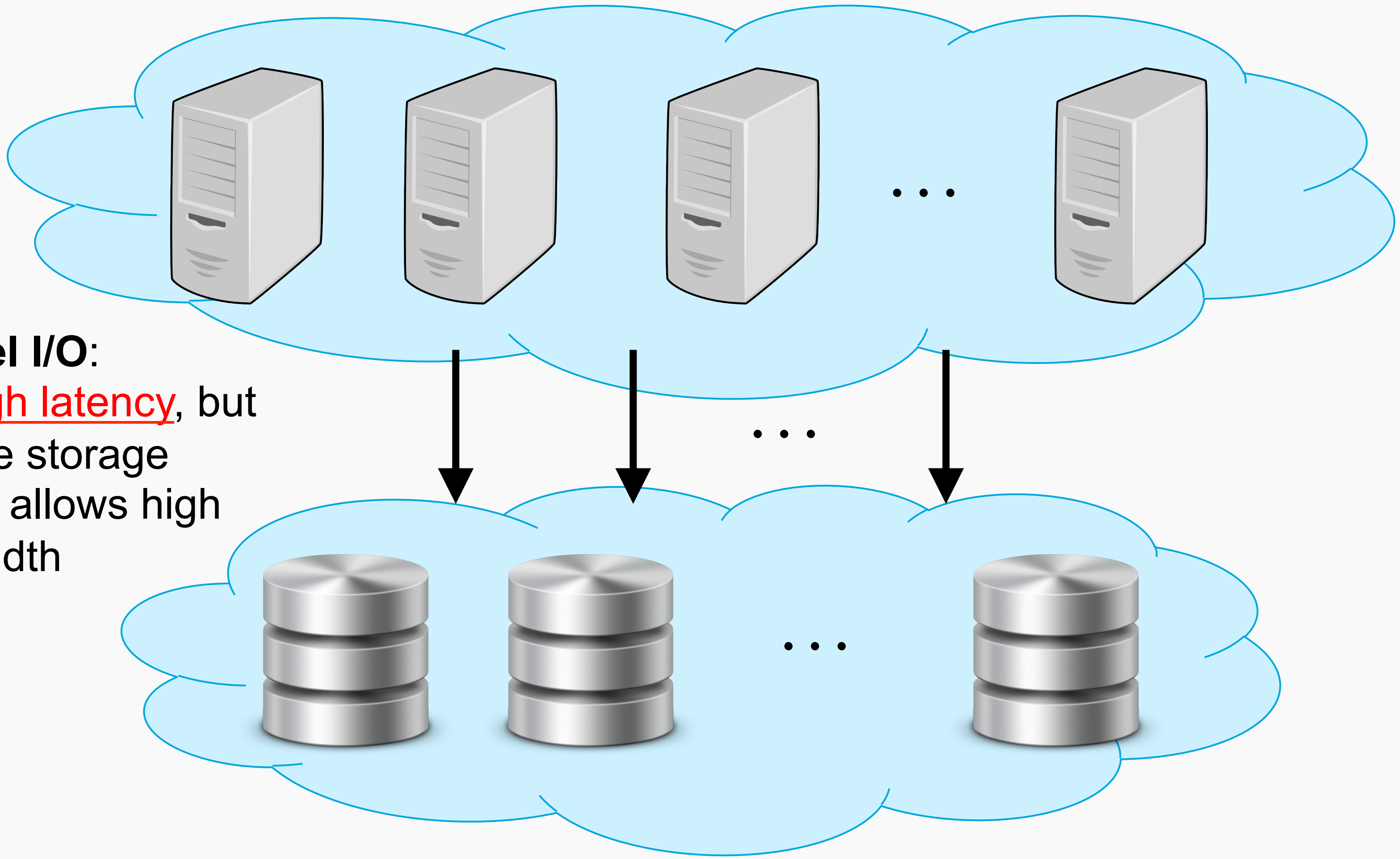
HPC I/O: Parallel File Systems

Traditional File System



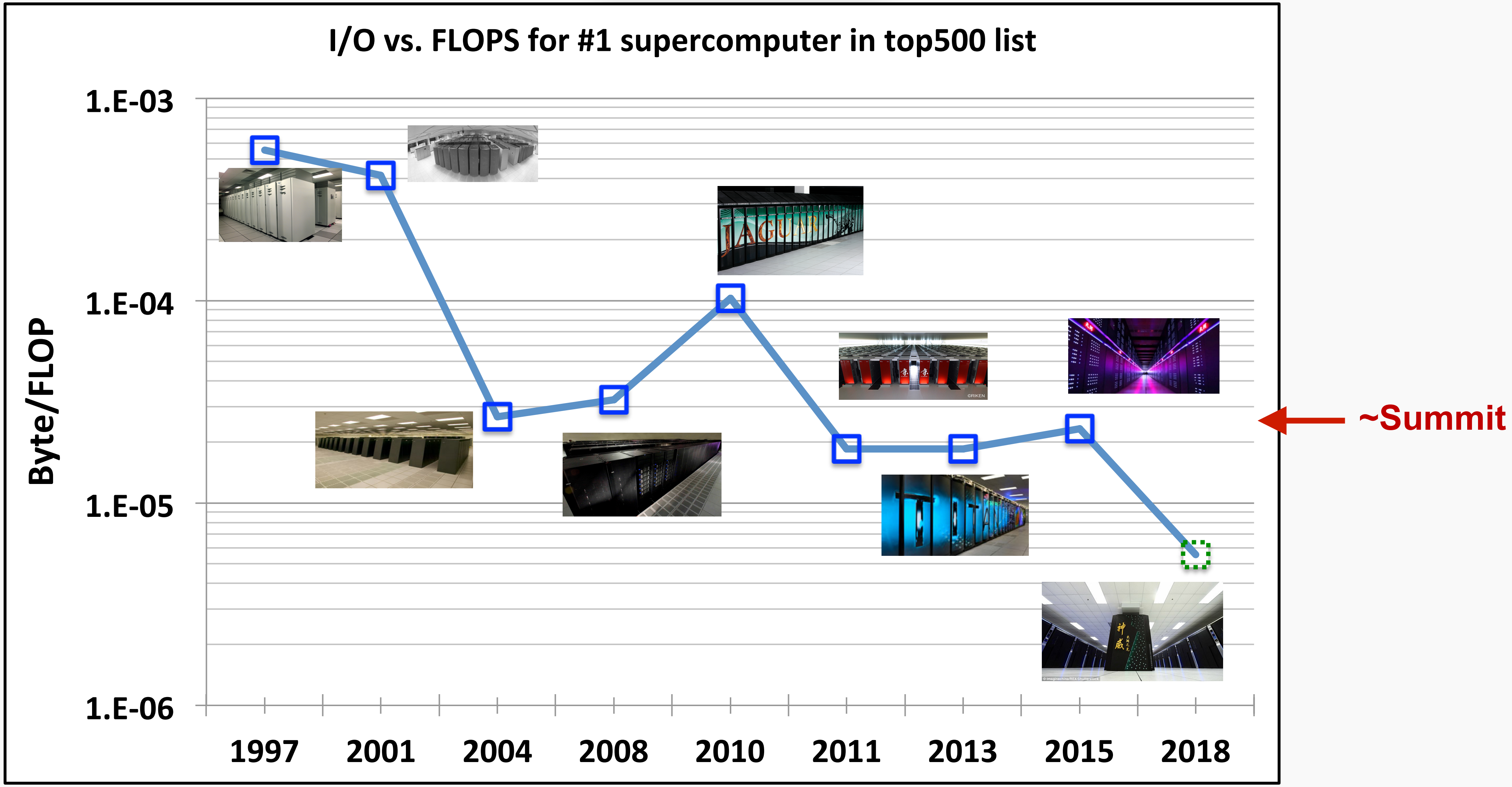
Serial I/O: <u>High latency</u> compared to RAM

Parallel File System



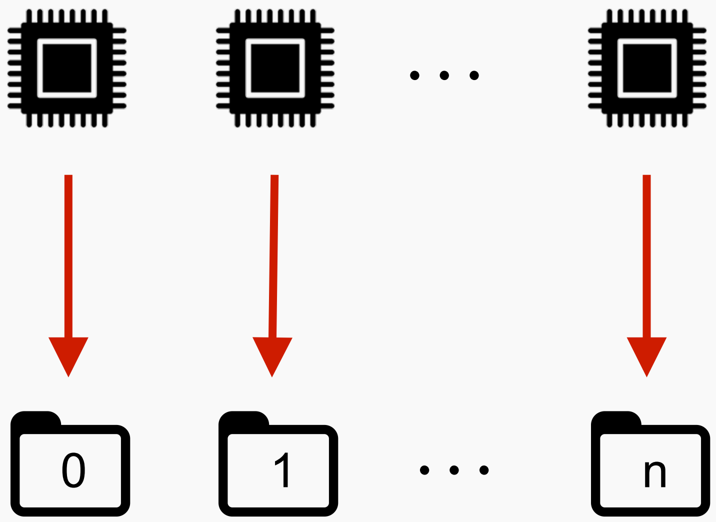
Parallel I/O: <u>Still high latency</u>, but multiple storage targets allows high bandwidth

Storage vs Computation Trend



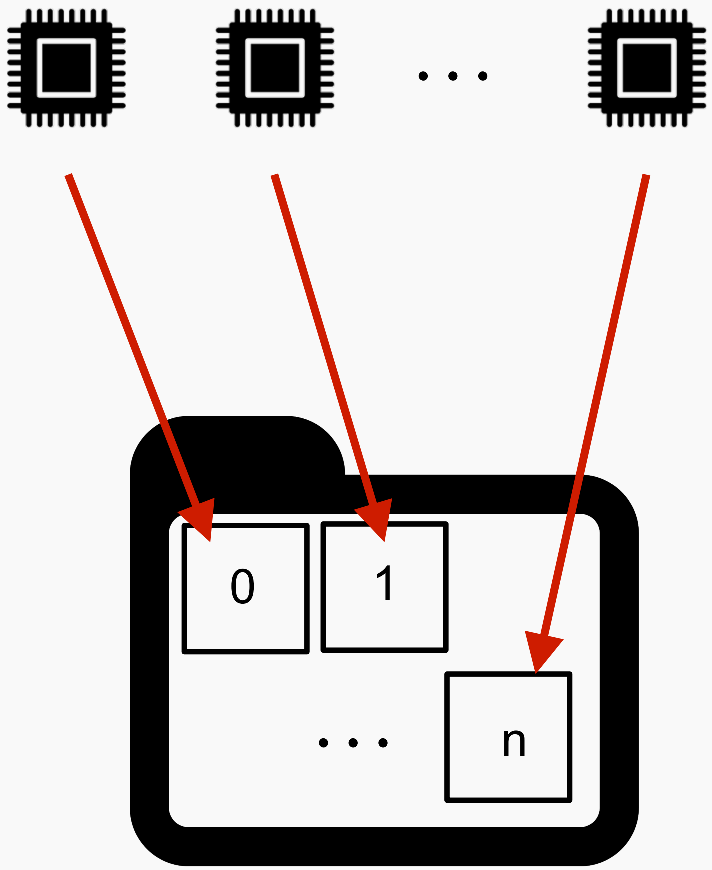
Types of Parallel I/O

File-per-process (FPP) Parallel

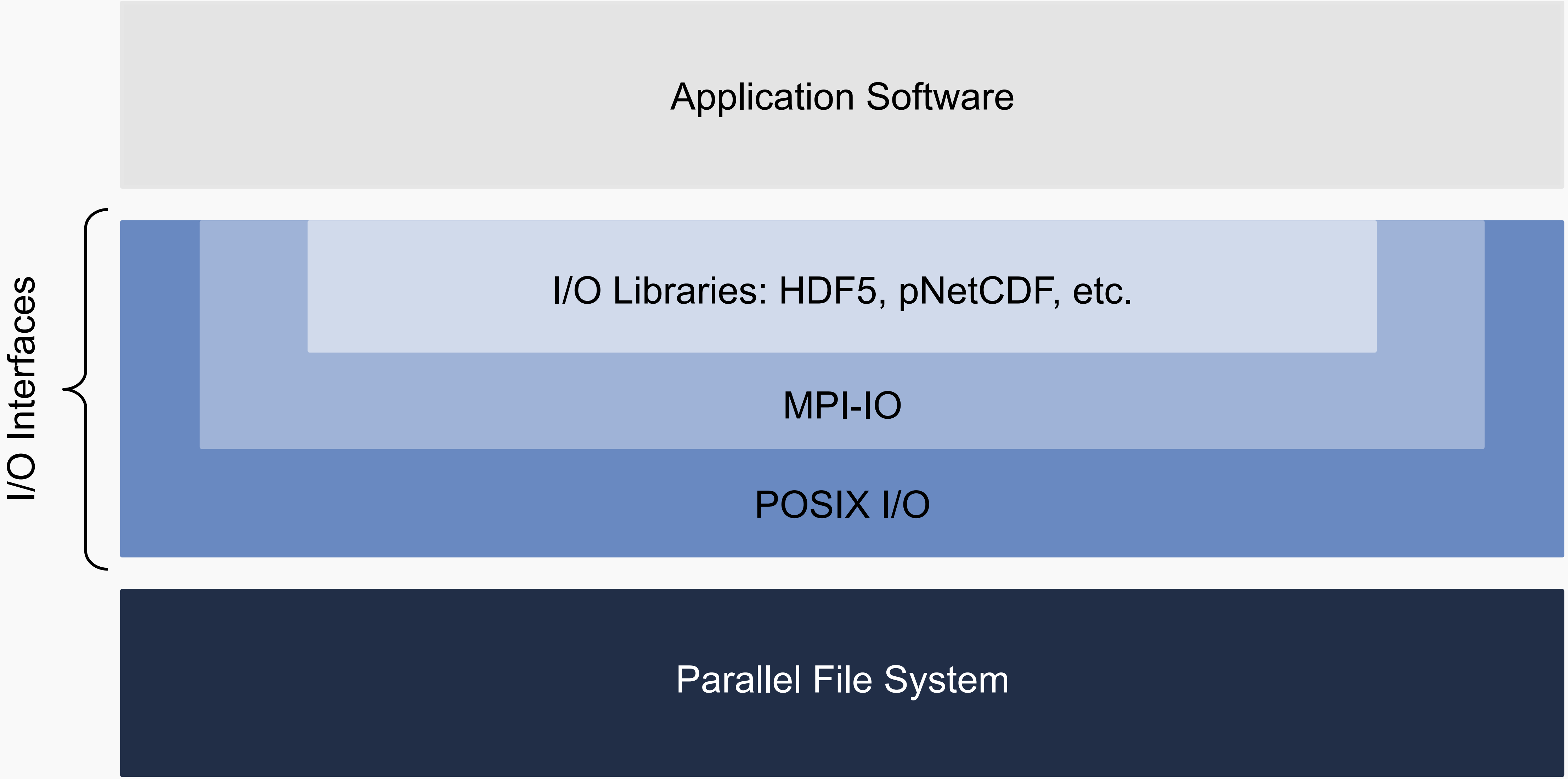


FPP can be fast for 10^1-10^3 ranks, but cannot scale to extreme scales (management and consumption issues)

Shared File Parallel



Typical I/O Software Stack



POSIX I/O Interface

Lowest-level I/O API

Pros

- Well-supported: Fortran, C and C++ I/O calls are converted to POSIX I/O
- Simple: File is a sequence of bytes
- Low overhead

Cons

- Shared-file parallel I/O is possible, but complicated (parallel access, buffing, flushing, etc. must be explicitly managed)
- File-per-process I/O is easy, but metadata and storage consumption is not scalable

MPI-IO I/O Interface

Typical interface for parallel I/O

- MPI-based replacement functions for POSIX

Independent MPI-IO

- Each MPI task is handles the I/O independently using *non-collective* calls
  - Ex. `MPI_File_write()` and `MPI_File_read()`
- Similar to POSIX I/O, but supports derived datatypes (useful for non-contiguous access)

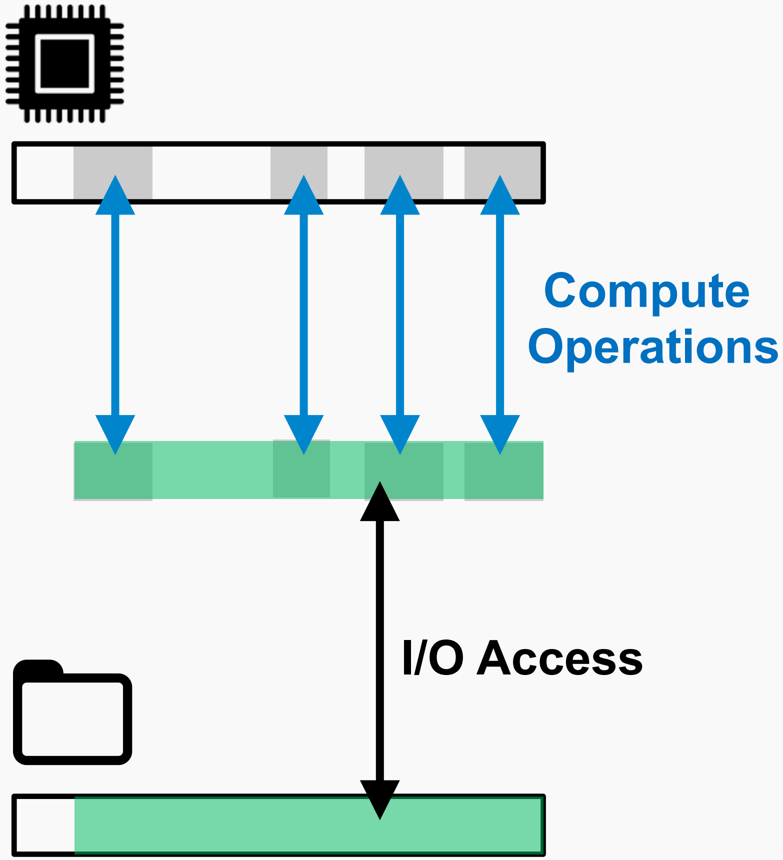
Collective MPI-IO

- All MPI tasks participate in I/O, and must call the same routines.
  - Ex. `MPI_File_write_all()` and `MPI_File_read_all()`
- Allows MPI library to perform collective I/O optimizations (often boosting performance)

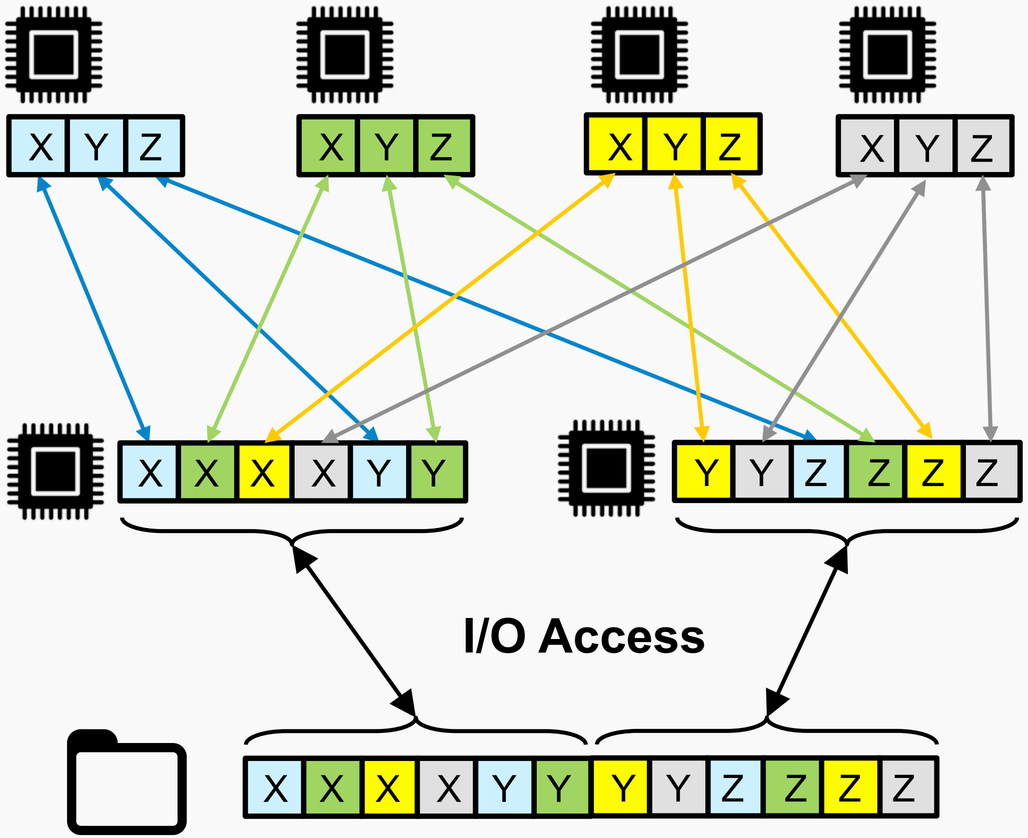
MPI-IO (or a higher-level library leveraging MPI-IO) is <u>recommended</u> on Mira & Theta

Common MPI-IO Optimizations

Data Sieving



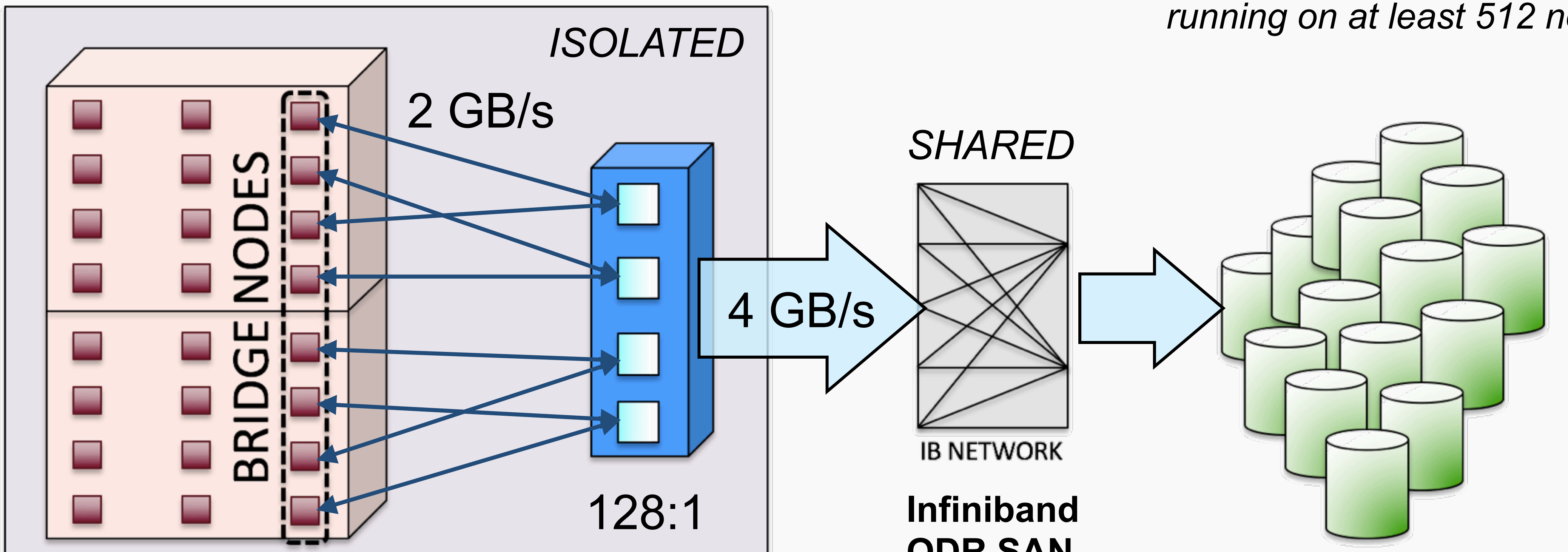
Two-Phase I/O (Collective Aggregation)



Mira I/O Architecture (Blue Gene/Q – GPFS)

Mira I/O Infrastructure: Overview

***Note**: I/O Nodes are dedicated resources when running on at least 512 nodes*



Compute Node Rack
1024 compute nodes
16 bridge nodes

I/O Nodes (Gateway)
2 bridge nodes connect to each I/O node

Infiniband QDR SAN
Connected to Cooley & Cetus

GPFS File system
`/mira-fs0`
`/mira-fs1`
`/mira-home`

IBM's General Parallel File System (GPFS)

IBM's GPFS is used for all parallel file systems on Mira

- Supports POSIX semantics
- Uses client-side and server-side caching
- Metadata is replicated on all file systems
- Quotas are enabled
  - `myquota` (home)
  - `myprojectquotas` (project)
  - Overrun quota error: `-EQUOTA`

Name	Type	Blocksize	Capacity	Speed
`mira-fs0`	project	8 MB	19 PB	240 GB/s
`mira-fs1`	project	8 MB	7 PB	90 GB/s
`mira-home`	home	256 K	1 PB	-

Optimizing I/O on Mira (BG/Q)

MPI-IO on Mira

Mira has great support for MPI-IO

- Leveraged by other major I/O libraries
  - Look in `/soft/libraries`
  - HDF5, NetCDF, pNetCDF, Adios, etc.
- Uses BG/Q-specific Optimizations
  - Handles alignment on block boundaries
  - Leverages Mira 5D Torus network

Important Note

MPI-IO scales well, but may run out of memory at full-machine scales

Usually related to MPI_Alltoall calls and discontinuous data types (Workarounds discussed soon)

Important MPI-IO Recommendations

- Use collective routines (eg. `MPI_File_write_at_all()`)
- Disable locking within the Blue Gene ADIO layer for non-overlapping writes using the following environment variable:
  - `--env BGLOCKLESSMPIO_F_TYPE=0x47504653`

Argonne NATIONAL LABORATORY

MPI-IO BG/Q Driver Tuning

e.g. `soft add +mpiwrapper-xl.legacy`

Advanced Options:

- Environment variable `BGMPIO_NAGG_PSET=16` (default 8)
- Hint: `cb_buffer_size=16m` (change the collective aggregation buffer size)
- Hint: `romio_no_indep_rw` can improve collective file open/close performance
  - Only does file open on aggregator ranks during `MPI_File_open`, for independent I/O (eg `MPI_File_write_at`) non-aggregator nodes file open at write time (deferred)

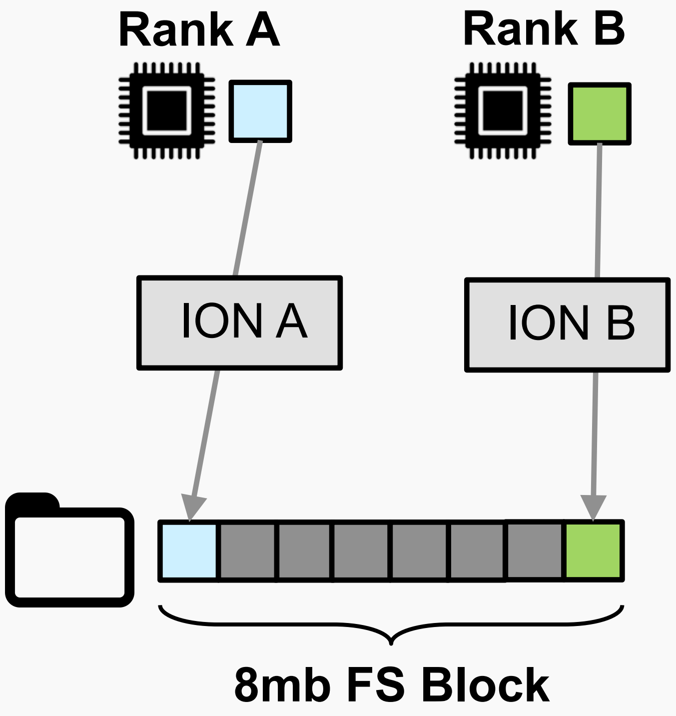
BGQ driver variables for memory-issue workarounds (often hurts performance):

- `--envs BGMPIO_COMM=1`
  - no MPI_Alltoall(v) calls – buffers can become large
- `--envs PAMID_SHORT=0`
- `--envs PAMID_DISABLE_INTERNAL_EAGER_TASK_LIMIT=1`
  - avoid heap fragmentation

Alignment

Use block-aligned I/O when using shared files

- The GPFS project file systems are all 8 MB
  - Unaligned access will be punished by GPFS locking
  - Larger, block-aligned accesses will perform best (eg. 8mb, 16mb, 32mb)
- Collective MPI-IO (`MPI_File_write_at_all`) takes care of this for you



Example:

- *MPI rank A and B happen to use two different I/O nodes*
- *Rank A writes the first MB of an 8 MB block*
  - *The GPFS client for rank A must acquire the lock for this fs block*
- *Rank B writes the last MB of an 8 MB block*
  - *The GPFS client for rank B tries to acquire the block for this block but must wait because it is in use*
- *Parallel I/O becomes serial for this workload*

Performance Tools on Mira

Darshan (https://www.alcf.anl.gov/user-guides/darshan)

- Stores I/O profiling summary in single compressed log file
  - Look in: `/gpfs/mira-fs0/logs/darshan/mira/<year>/<month>/<day>`

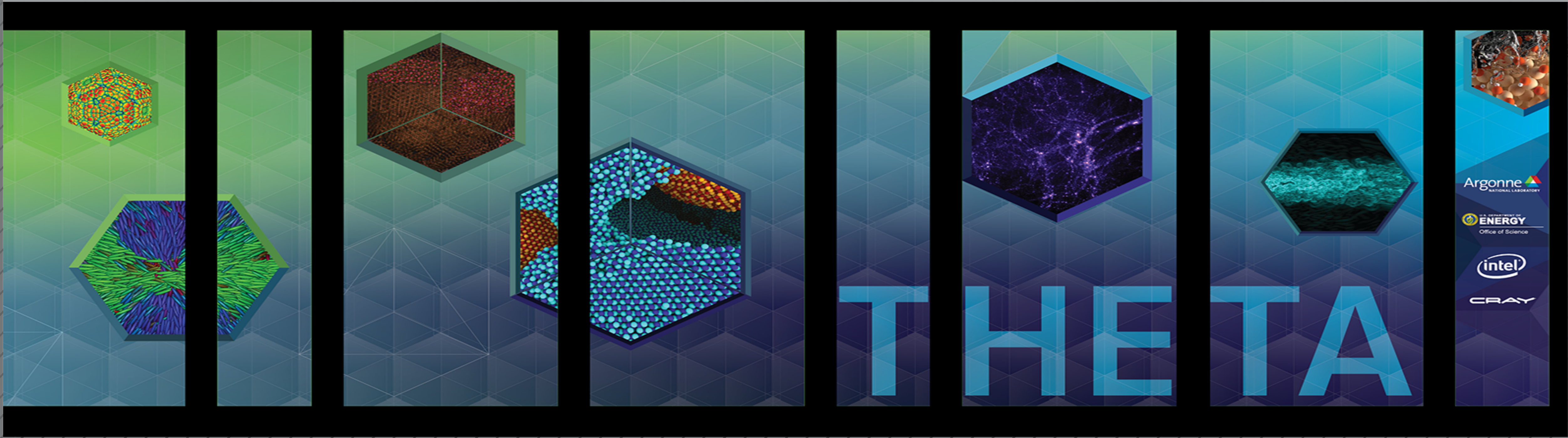
TAU (https://www.alcf.anl.gov/user-guides/tuning-and-analysis-utilities-tau)

- "`-optTrackIO`" in `TAU_OPTIONS`

mpitrace (http://www.alcf.anl.gov/user-guides/hpctw)

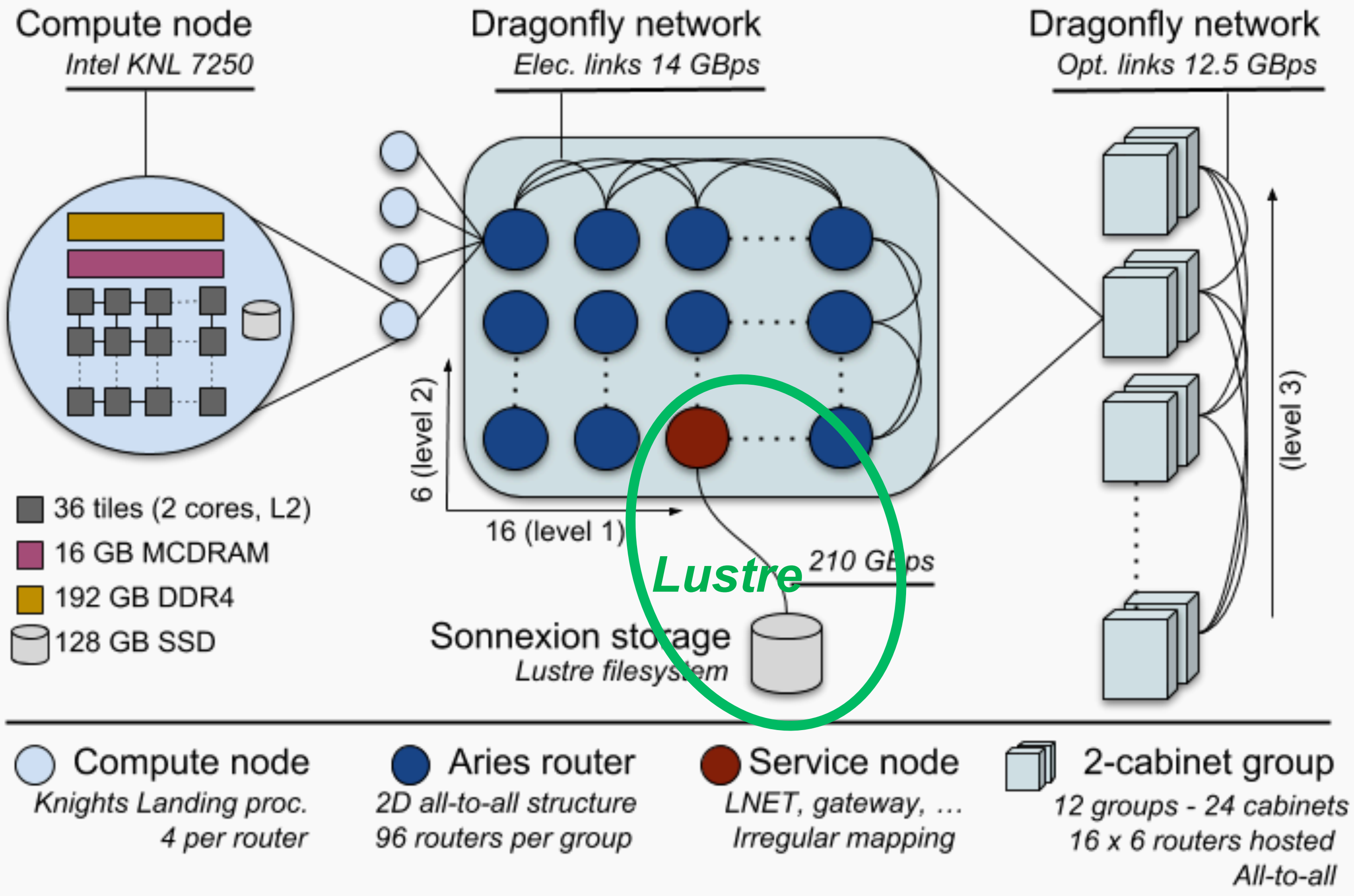
- List performance of `MPI_File*` calls
  - Show performance of underlying MPI-IO for IO libraries such as HDF5

Theta I/O (Cray XC40 – Lustre)



Argonne NATIONAL LABORATORY

Theta system Overview



Architecture: Cray XC40
Processor: 1.3 GHz Intel Xeon Phi 7230 SKU
Peak performance of 11.69 petaflops
Racks: 24
Nodes: 4,392
Total cores: 281,088
Cores/node: 64
Memory/node: 192 GB DDR4 SDRAM
(Total DDR4: 843 TB)
High bandwidth memory/node: 16 GB MCDRAM
(Total MCDRAM: 70 TB)

10 PB Lustre file system
SSD/node: 128 GB
(Total SSD: 562 TB)
Aries interconnect - Dragonfly configuration

Luster File System Basics

Lustre File System Basics

Clients = LNET Router Nodes
MDS = Metadata Server
MDT = Metadata Target
OSS = Object Storage Server
OST = Object Storage Target

Each file is distributed over 1+ OSTs, depending on the size and striping settings for the specific file.

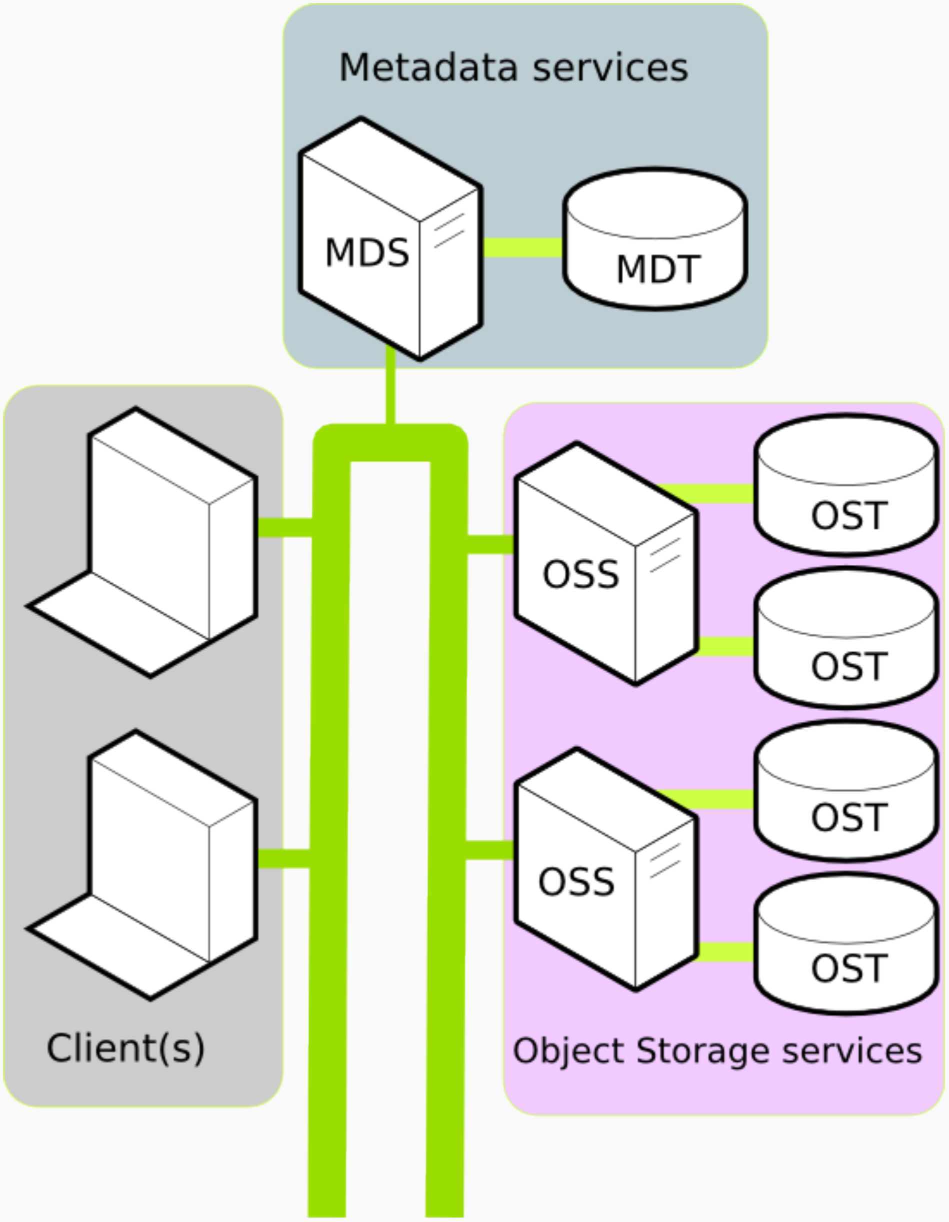


Image: https://wiki.hpdd.intel.com/display/PUB/Components+of+a+Lustre+filesystem

Theta – Lustre Specification

Current Version: lfs 2.7.2.26

Hardware: 4 Sonexion Storage Cabinets

- 10 PB usable RAID storage
- 56 OSS (1 OST per OSS)

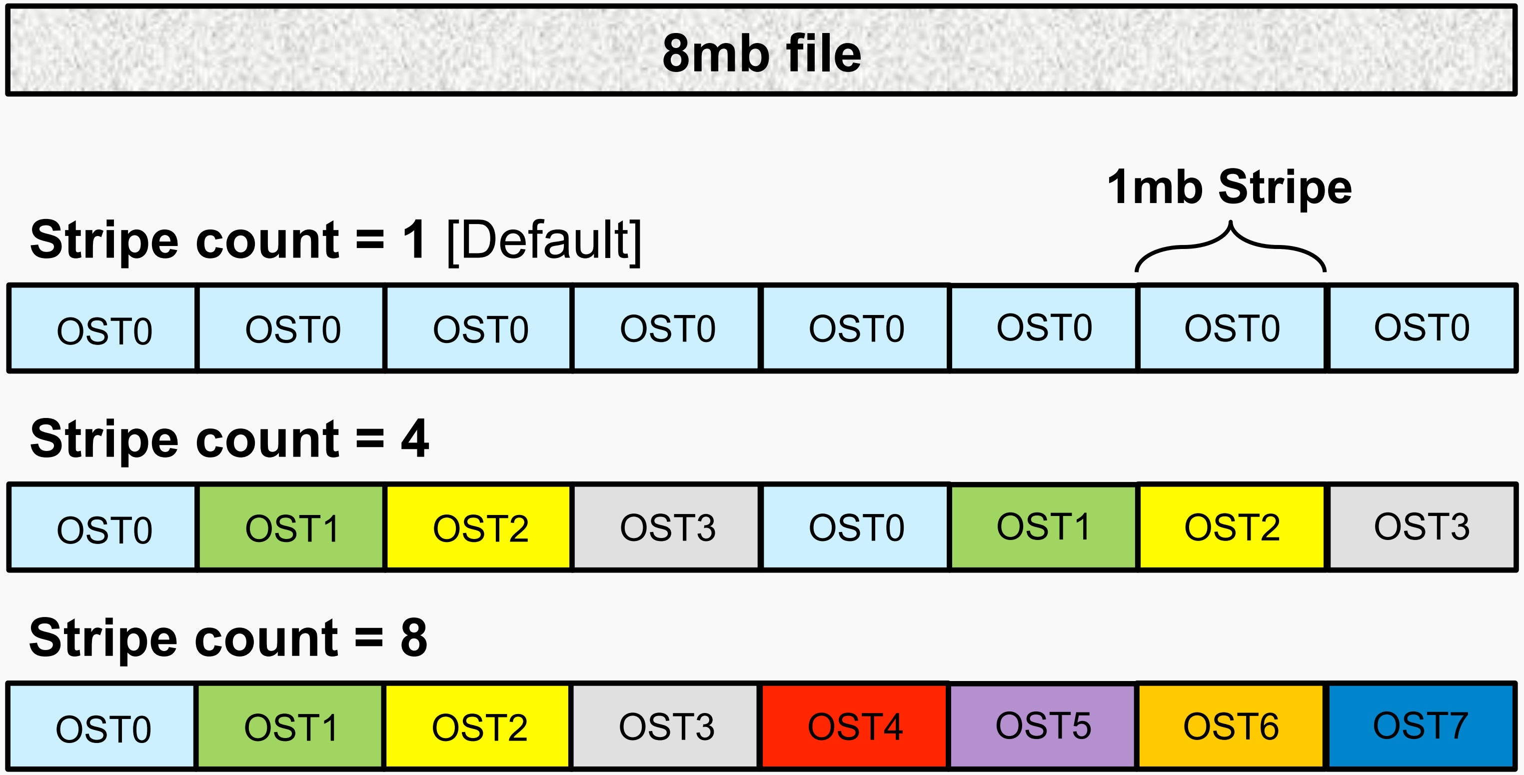
Note: OSS cache currently disabled by hardware (Sonexion)

Performance:

- Total Write BW **172 GB/s**, Total Read BW **240 GB/s**
- Peak Performance of 1 OST is 6 GB/s
  - **Lustre client-cache effects can allow much higher BW**

Lustre File Striping Basics
Key to Parallel Performance

Example: Consider a single **8mb file** with **1mb stripe size**…



Basic Idea
Files are *striped* across OSTs using a predefined striping pattern (pattern = count & size)

Stripe count
The number of OSTs (storage devices) used to store/access the file
`[Default = 1]`

Stripe size
The width of each contiguous OST access
`[Default = 1m]`

Note: `1m = 1048576`

Argonne NATIONAL LABORATORY

Lustre File System Utility: lfs

Manual: http://doc.lustre.org/lustre_manual.pdf

- List available lfs arguments: lfs help
- List name and status of the various OSTs: lfs osts <path>
- Set/Get striping information: lfs getstripe <path>
- Set/Get striping information: lfs setstripe <args> <path>
- Check disk space usage: lfs df

```
zamora@thetalogin6:~> lfs df
UUID                   1K-blocks       Used            Available       Use%    Mounted on
snx11214-MDT0000_UUID  3156416840      81210736        3032725640      3%      /lus/theta-fs0[MDT:0]
snx11214-MDT0001_UUID  3156416840      393420          3113542956      0%      /lus/theta-fs0[MDT:1]
snx11214-MDT0002_UUID  3683559388      312576          3640766348      0%      /lus/theta-fs0[MDT:2]
snx11214-MDT0003_UUID  3683559388      388484          3640690440      0%      /lus/theta-fs0[MDT:3]
snx11214-OST0000_UUID  180419603168    60505549136     118094544908    34%     /lus/theta-fs0[OST:0]
snx11214-OST0001_UUID  180419603168    61335067584     117265055940    34%     /lus/theta-fs0[OST:1]
...
snx11214-OST0036_UUID  180419603168    61094309592     117505721756    34%     /lus/theta-fs0[OST:54]
snx11214-OST0037_UUID  180419603168    60293444120     118306098300    34%     /lus/theta-fs0[OST:55]

filesystem summary:    10103497777408  3429401255528   6572198844780   34%     /lus/theta-fs0
```

Example: `lfs setstripe` (IMPORTANT)

The stripe settings are critical to performance

- Defaults are **not** optimal for large files

Command syntax:

```
lfs setstripe --stripe-size <size> --count <count> <file/dir name>
lfs setstripe -S <size> -c <count> <file/dir name>
```

```
zamora@thetalogin6:~> mkdir stripecount4size8m
zamora@thetalogin6:~> lfs setstripe -c 4 -S 8m stripecount4size8m/.
zamora@thetalogin6:~> lfs getstripe stripecount4size8m
stripecount4size8m
stripe_count:   4 stripe_size:    8388608 stripe_offset:  -1
```

Example:
`lfs getstripe`

```
zamora@thetalogin6:~> cd stripecount4size8m/
zamora@thetalogin6:~/stripecount4size8m> touch file.1
zamora@thetalogin6:~/stripecount4size8m> touch file.2
zamora@thetalogin6:~/stripecount4size8m> lfs getstripe .
.
stripe_count:   4 stripe_size:    8388608 stripe_offset:  -1
./file.1
lmm_stripe_count:   4
lmm_stripe_size:    8388608
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  14
	obdidx		 objid		 objid		 group
	    14	      47380938	    0x2d2f9ca	             0
	    36	      47391032	    0x2d32138	             0
	     0	      47405104	    0x2d35830	             0
	    28	      47397537	    0x2d33aa1	             0

./file.2
lmm_stripe_count:   4
lmm_stripe_size:    8388608
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  23
	obdidx		 objid		 objid		 group
	    23	      47399545	    0x2d34279	             0
	    39	      47406868	    0x2d35f14	             0
	     3	      47405323	    0x2d3590b	             0
	    29	      47395561	    0x2d332e9	             0
```

Important Notes about File Striping

- Make sure to use the /project file system (NOT /home)
  - /project is Lustre, /home is NOT
- Don't set the stripe_offset yourself (let Lustre choose *which* OSTs to use)
- Default Striping is stripe_count=1 and stripe_size=1048576
- Files and directories inherit striping patterns from the parent directory
- Stripe count cannot exceed number of OSTs (56)
- Striping cannot be changed once file created
  - Need to re-create file – copy to directory with new striping pattern to change it

Non-lfs Options:

- Can set stripe settings in **Cray MPI-IO** (striping_unit=*size*, striping_factor=*count*)
  - Ex: MPICH_MPIIO_HINTS=*:striping_unit=<SIZE>:striping_factor=<COUNT>
- Can do ioctl system call yourself passing LL_IOC_LOV_SETSTRIPE with structure for count and size
  - ROMIO example: https://github.com/pmodels/mpich/blob/master/src/mpi/romio/adio/ad_lustre/ad_lustre_open.c#L114

Using Cray MPI-IO

Cray MPI-IO Overview

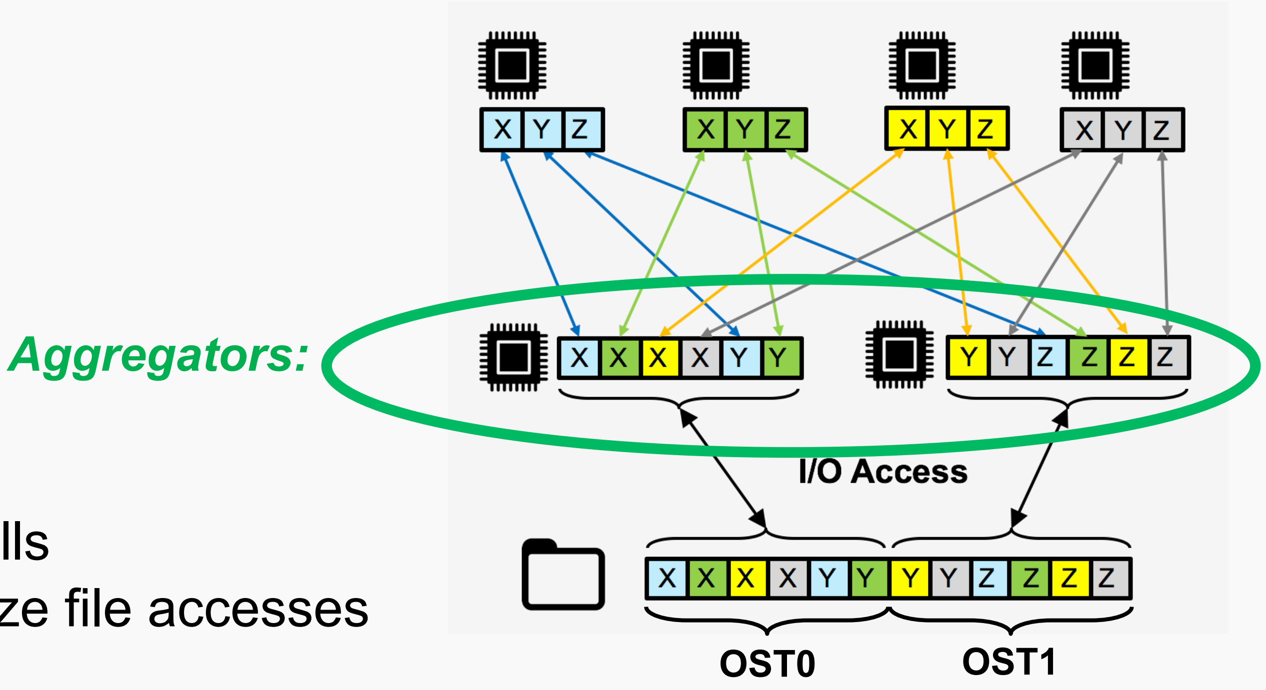
Cray MPI-IO is recommended on Theta

- Used by Cray-MPICH (default MPI environment on Theta - `cray-mpich` module)
- Based on MPICH-MPIIO (ROMIO)
- Optimized for Cray XC40 & Lustre
- Many tuning parameters: `man intro_mpi`

Underlying I/O layer for many I/O libraries

- HDF5 (`module load cray-hdf5-parallel`)
- PNetCDF (`module load cray-netcdf-hdf5parallel`)

Tuning Cray-MPI-IO: Collective Operations



Collectives (two-phase I/O): MPI_File_*_all calls

- Aggregate data into large/contiguous stripe-size file accesses

Tuning aggregation settings:

- Number of aggregator nodes (`cb_nodes` hint) defaults to the striping factor (*count*)
  - `cray_cb_nodes_multiplier` hint will multiply the number of aggregators
  - Total aggregators = `cb_nodes` x `cray_cb_nodes_multiplier`
- Collective buffer size defaults to the stripe size
  - `cb_buffer_size` hint (in ROMIO) is ignored by Cray
  - ROMIO's collective buffer is allocated (according to this setting), but not used

Note: To use open-source MPICH MPI-IO (ROMIO), use `cb_align=3`

Argonne NATIONAL LABORATORY

Tuning Cray-MPI-IO: Extent-lock Contention

Each rank (client) needs its own lock to access a given file on an OST

- When 2+ ranks access same file-OST combination: extent lock contention

Mitigation: `cray_cb_write_lock_mode=1` (shared lock locking mode)

- A single lock is shared by all MPI ranks that are writing the file.
- Lock-ahead locking mode (`cray_cb_write_lock_mode=2`) not yet supported by Sonexion
  - **All file accesses MUST be collective**
    - `romio_no_indep_rw` must be set to true
    - HDF5, PNetCDF, and Darshan wont work (rely on some independent access)

Example:

```
MPICH_MPIIO_HINTS=*:cray_cb_write_lock_mode=1:cray_cb_nodes_multiplier=
<N>:romio_no_indep_rw=true
```

I/O Profiling Tools on Theta

Cray-MPI: Environment Variables for Profiling

- `MPICH_MPIIO_STATS=1`
  - MPI-IO access patterns for reads and writes written to stderr by rank 0 for each file accessed by the application on file close
- `MPICH_MPIIO_STATS=2`
  - set of data files are written to the working directory, one file for each rank, with the filename prefix specified by the `MPICH_MPIIO_STATS_FILE` env variable
- `MPICH_MPIIO_TIMERS=1`
  - Internal timers for MPI-IO operations, particularly useful for collective MPI-IO
- `MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY=1`
- `MPICH_MPIIO_AGGREGATOR_PLACEMENT_STRIDE`
- `MPICH_MPIIO_HINTS=<file pattern>:key=value:…`
- `MPICH_MPIIO_HINTS_DISPLAY=1`

Darshan I/O Profiling

Open-source statistical I/O profiling tool (https://www.alcf.anl.gov/user-guides/darshan)

- No source modifications, lightweight and low overhead
  - Finite memory allocation (about 2MB) - Overhead of 1-2% total

USE:

- Make sure postscript-to-pdf converter is loaded: `module load texlive`
  - `darshan` module should be loaded by default
- I/O characterization file placed here at job completion:

  `/lus/theta-fs0/logs/darshan/theta/<YEAR>/<MONTH>/<DAY>`

  Format: `<USERNAME>_<BINARY_NAME>_id<COBALT_JOB_ID>_<DATE>-<UNIQUE_ID>_<TIMING>.darshan`

- Use `darshan-job-summary.pl` command for charts, table summaries

  `darshan-job-summary.pl <darshan_file_name> --output darshansummaryfilename.pdf`
- Use `darshan-parser` for detailed text file

  `darshan-parser <darshan_file_name>  > darshan-details-filename.txt`

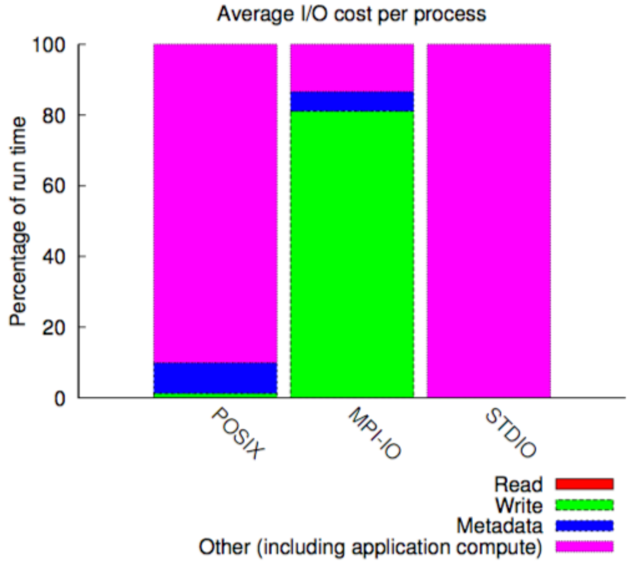
Darshan Output Example

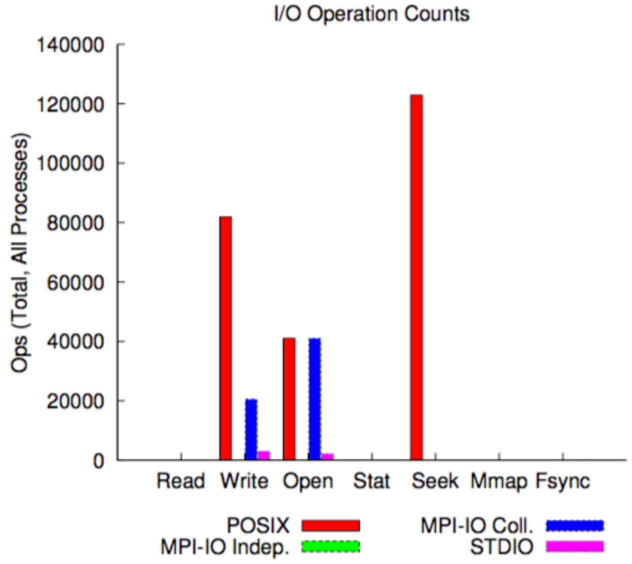
ior.7.7.0 (4/2/2018) 1 of 3

jobid: 212514	uid: 32915	nprocs: 1024	runtime: 32 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 136305 MiB at 2922.53 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.1 MiB at 3.16 MiB/s

. . .





. . .

../ior.7.7.0 -c -b 4M -t 4M -g -i 20 -w -a MPIIO

Lustre Metrics

Operations team records Lustre statistics throughout the day

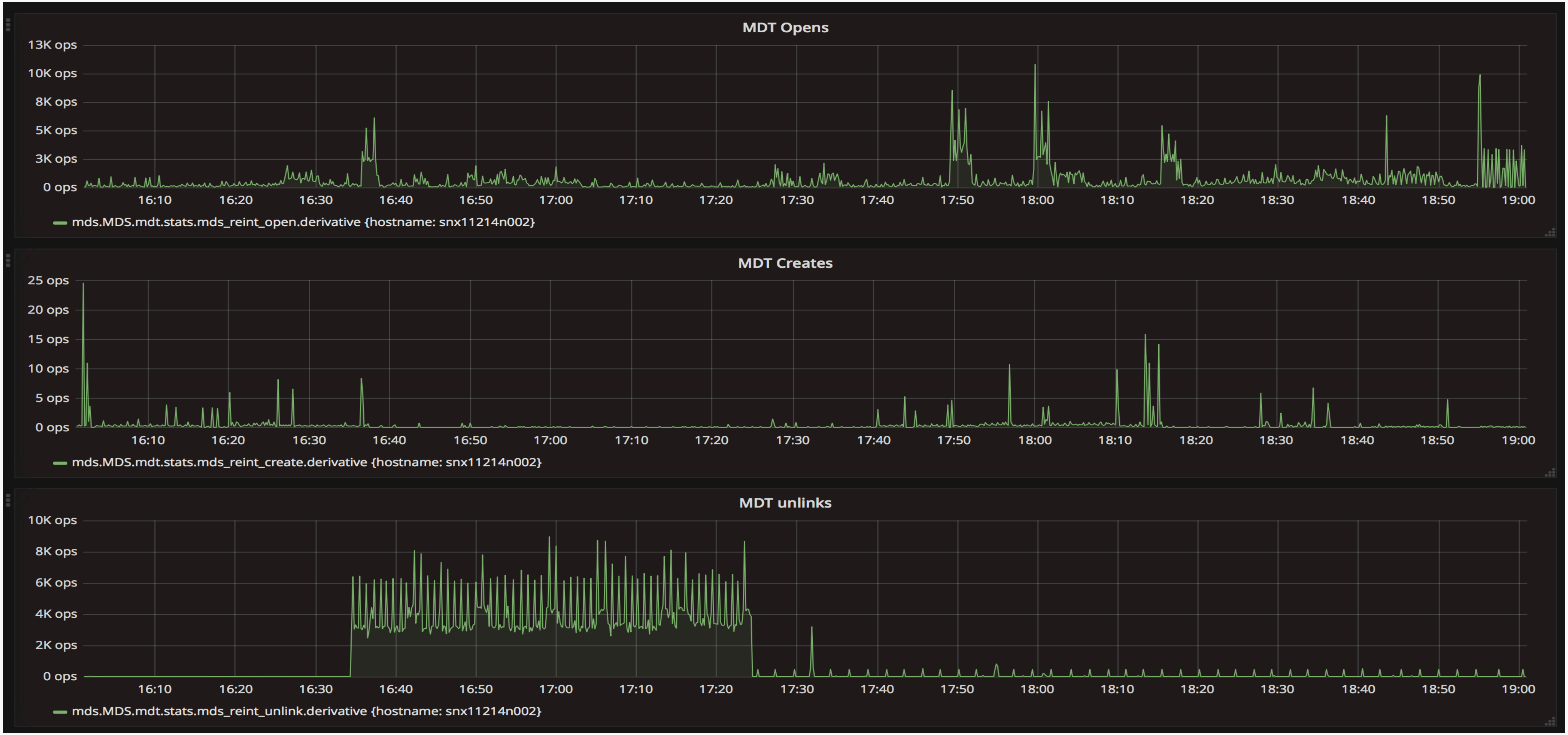
MDS

- Monitor all typical metadata operations, e.g. opens, creates, unlinks, renames, etc.

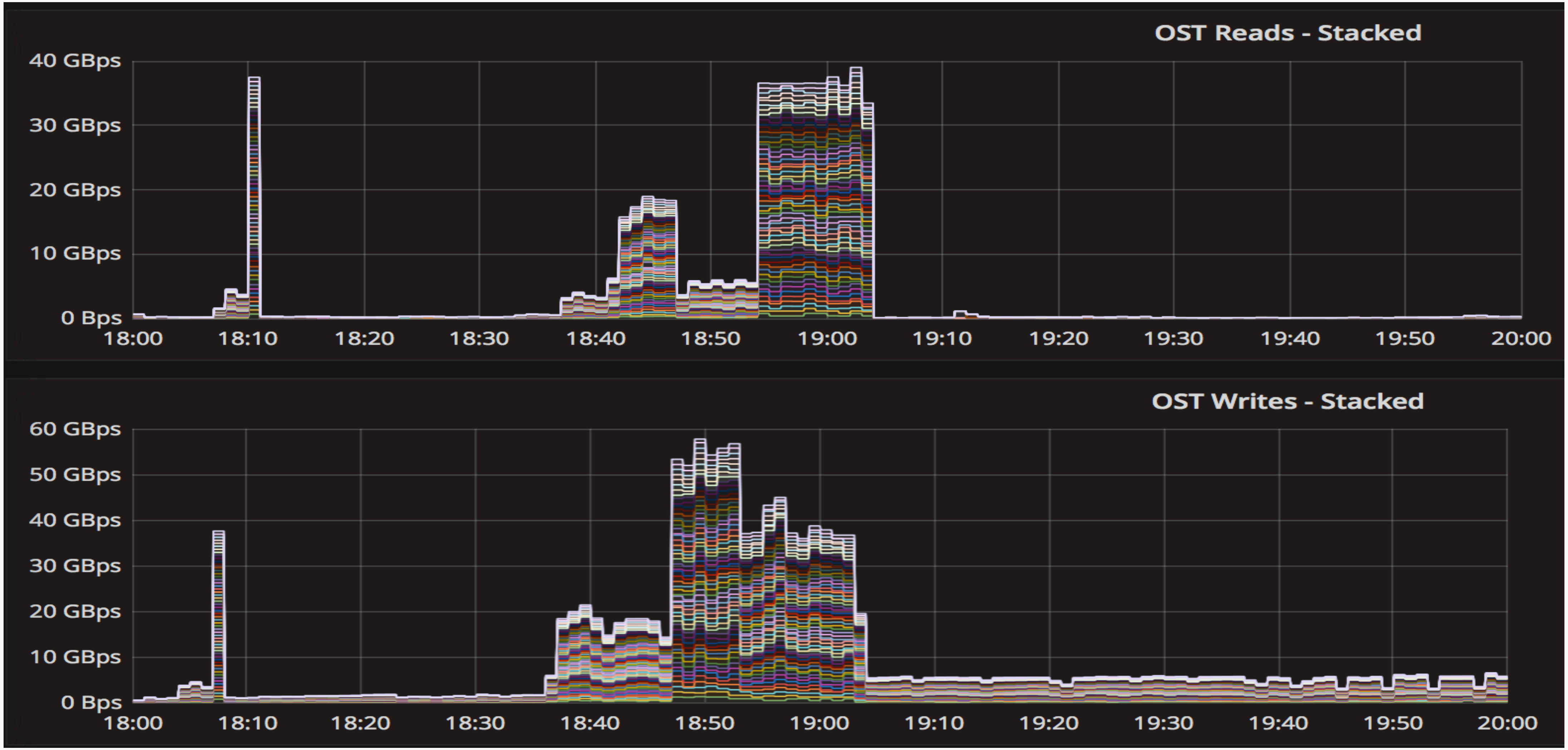
OSS

- Monitor reads/writes grouped by OST and OSS
- Monitor number of files and disk space

MDT Metrics Dashboard Example



OST Metrics Example (Shows Large IOR Run)



Lustre Performance on Theta

Dragonfly Network and Lustre Jitter

Communication is over shared networks (**No job isolation**)

- Currently 1 Metadata Sever (MDS) shared by all users
- MDS and/or OSS traffic surge can dramatically effect performance

When running IO performance tests, want either:

- run-time statistics (max, min, mean, median, etc.)
- Best of multiple trials (typically used here)
- Dedicated system

Argonne NATIONAL LABORATORY

General Luster Striping Guidelines

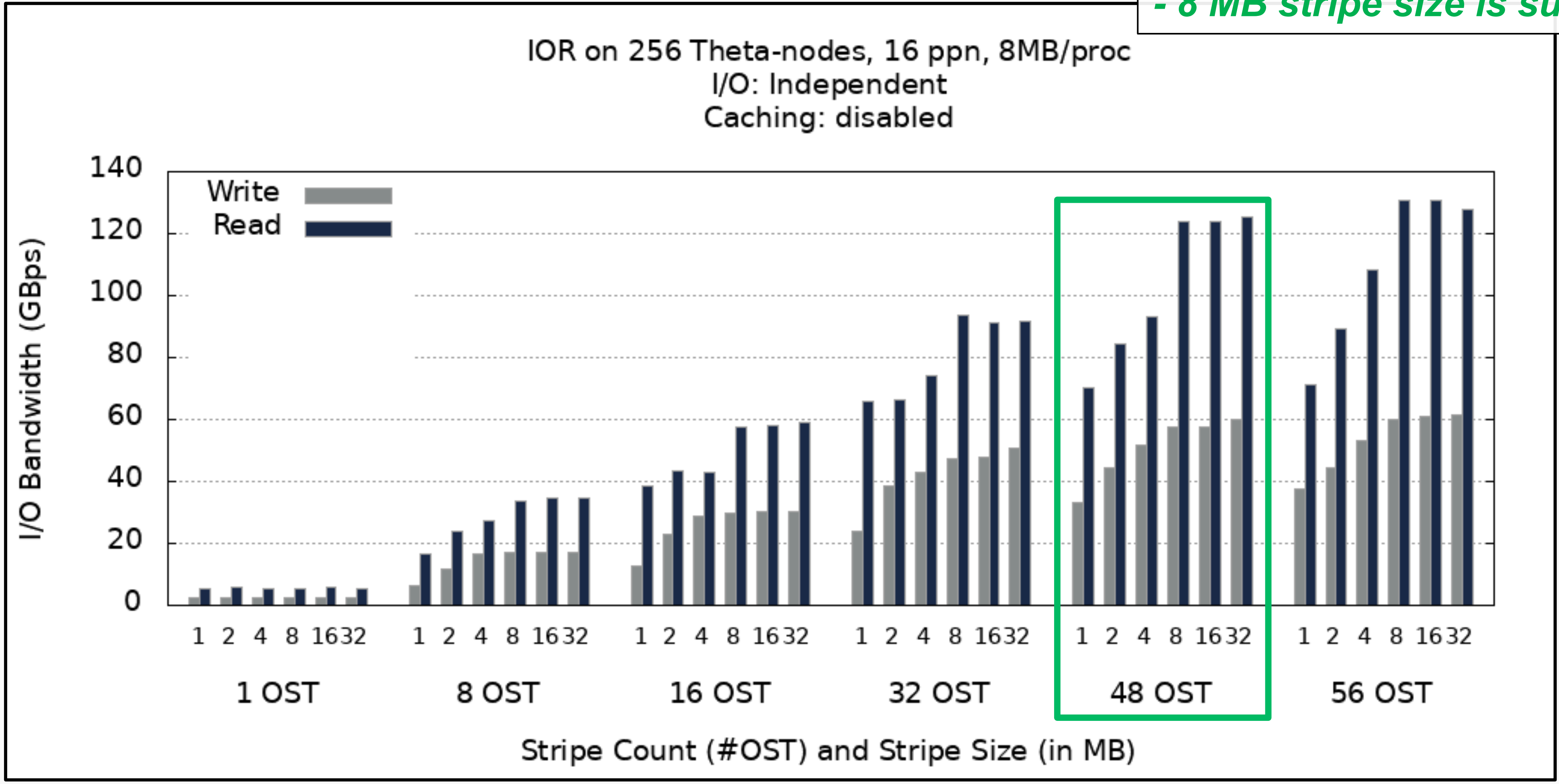
Large Shared Files:
- More than 1 stripe (default) usually best
  - Keep stripe count below the node count
  - ~8-48 usually good (not 56 - let Lustre avoid slow OSTs)
- Larger than a 1mb stripe (default) usually best
  - ~8-32 usually good
  - Note: large stripe sizes can require memory-hungry collective I/O

File-per-process: Use 1 stripe

Small files: Use 1 stripe

Shared File – 8MB/proc – Independent I/O
Client-side Caching DISABLED

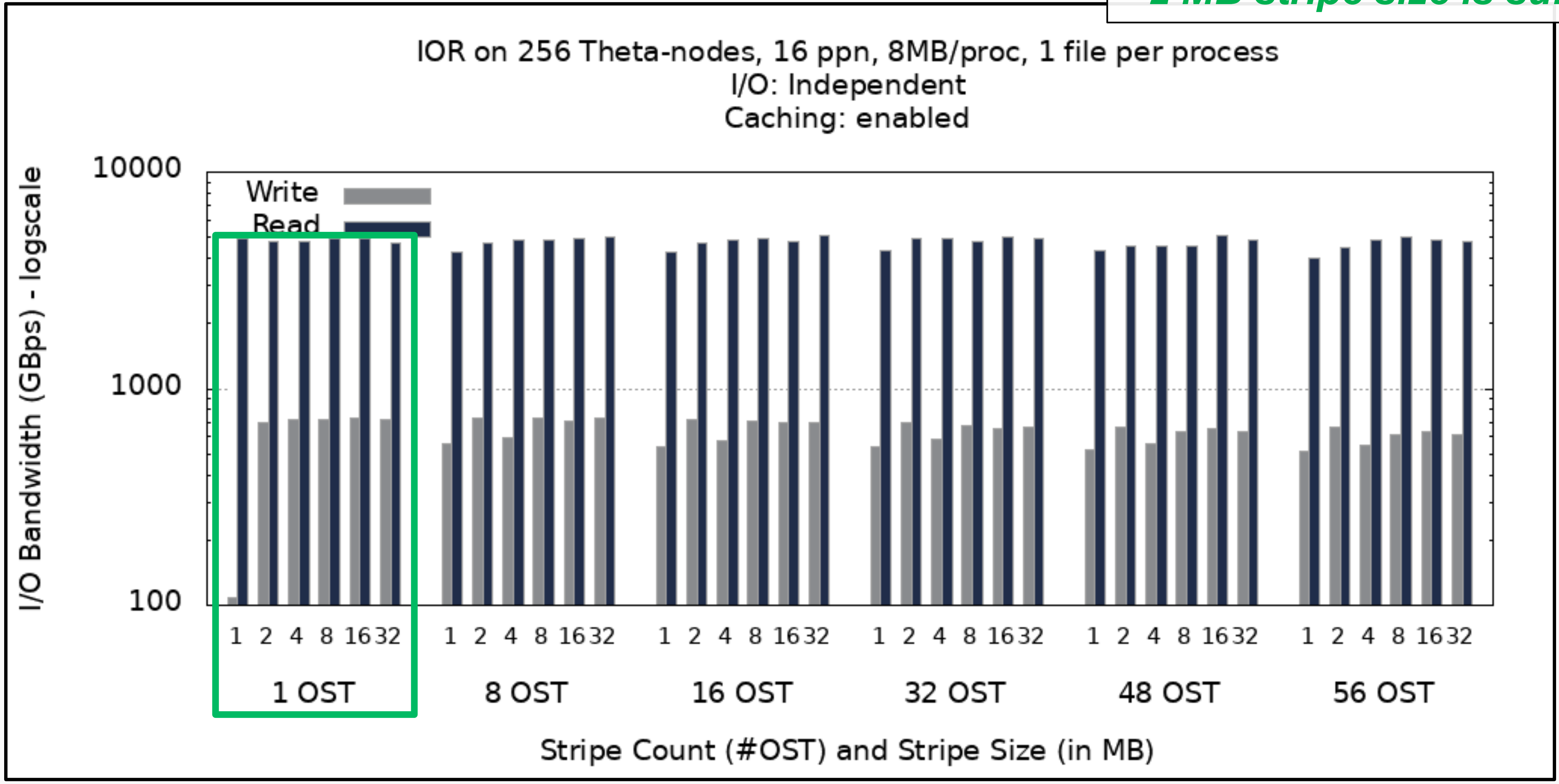
- More OSTs is better
- 8 MB stripe size is sufficient



Argonne NATIONAL LABORATORY

Shared File – 8MB/proc – Independent I/O
Client-side Caching ENABLED

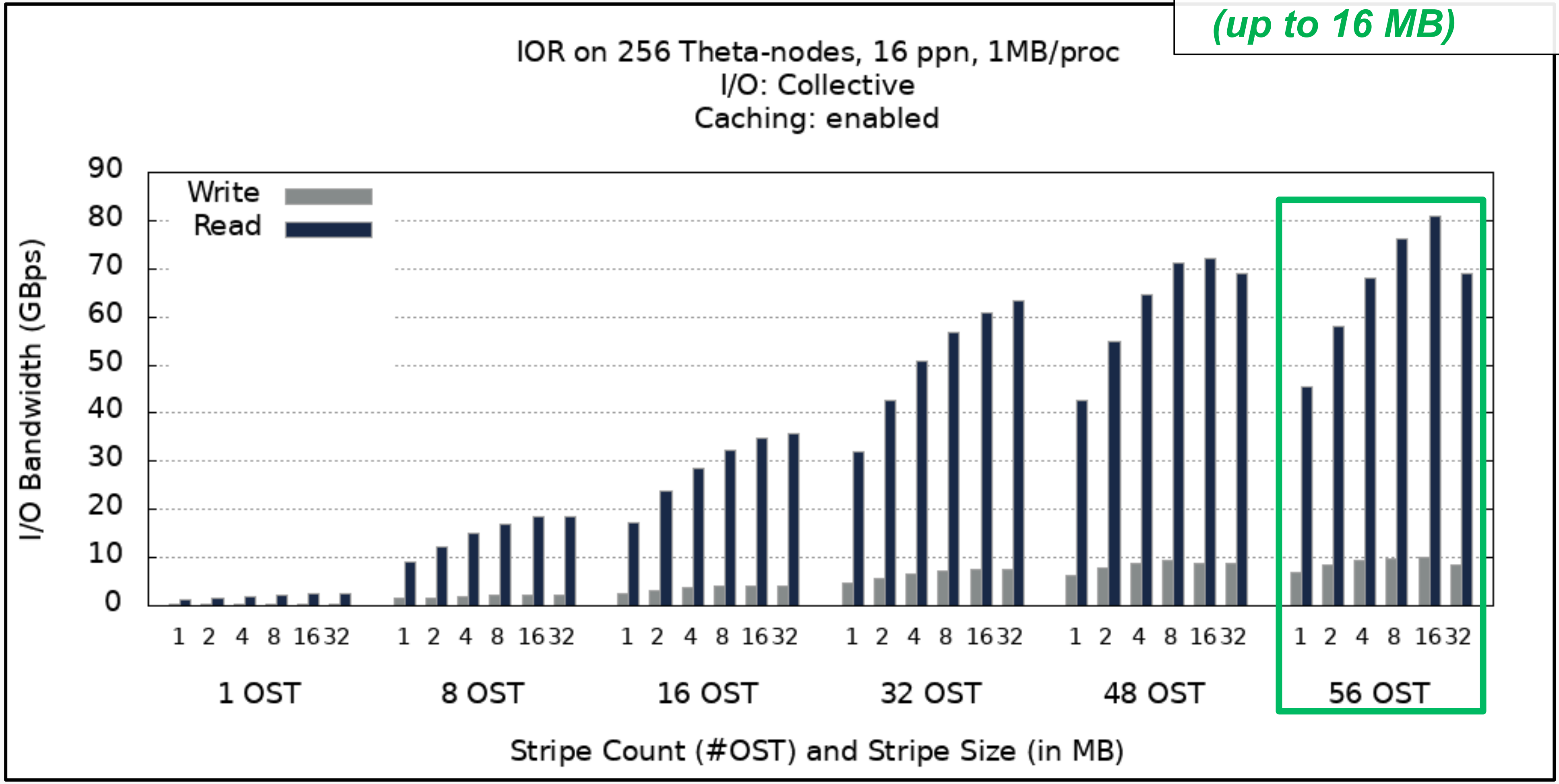
- Many OSTs are NOT necessary
- 2 MB stripe size is sufficient



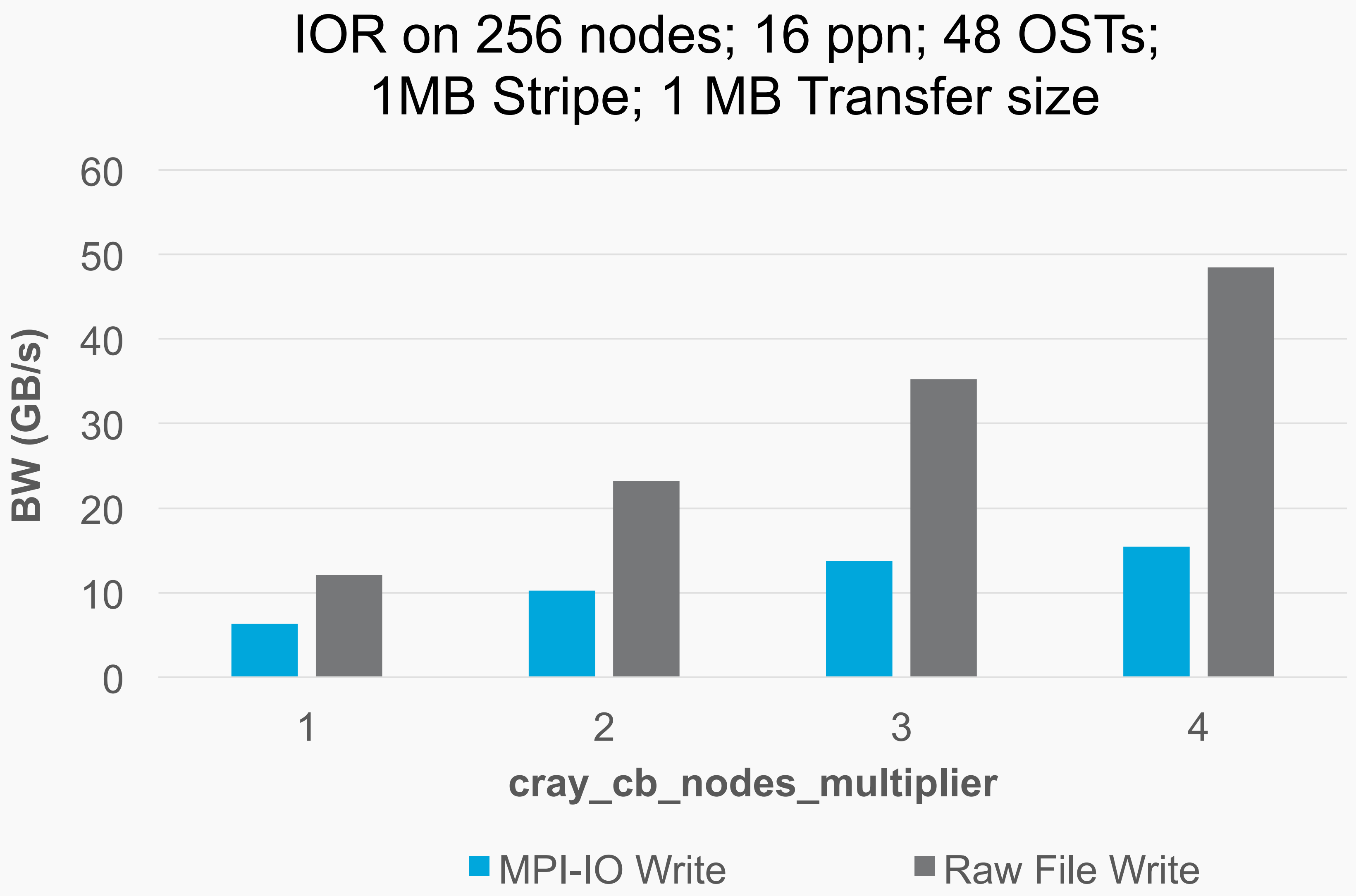
Shared File – 1MB/proc – Collective I/O

Client-side Caching ENABLED

- *More OSTs is better*
- *Larger stripe size is better (up to 16 MB)*



Collective I/O Shared-lock Performance



'Raw File Write' times taken from `MPICH_MPIIO_TIMERS=1` trace

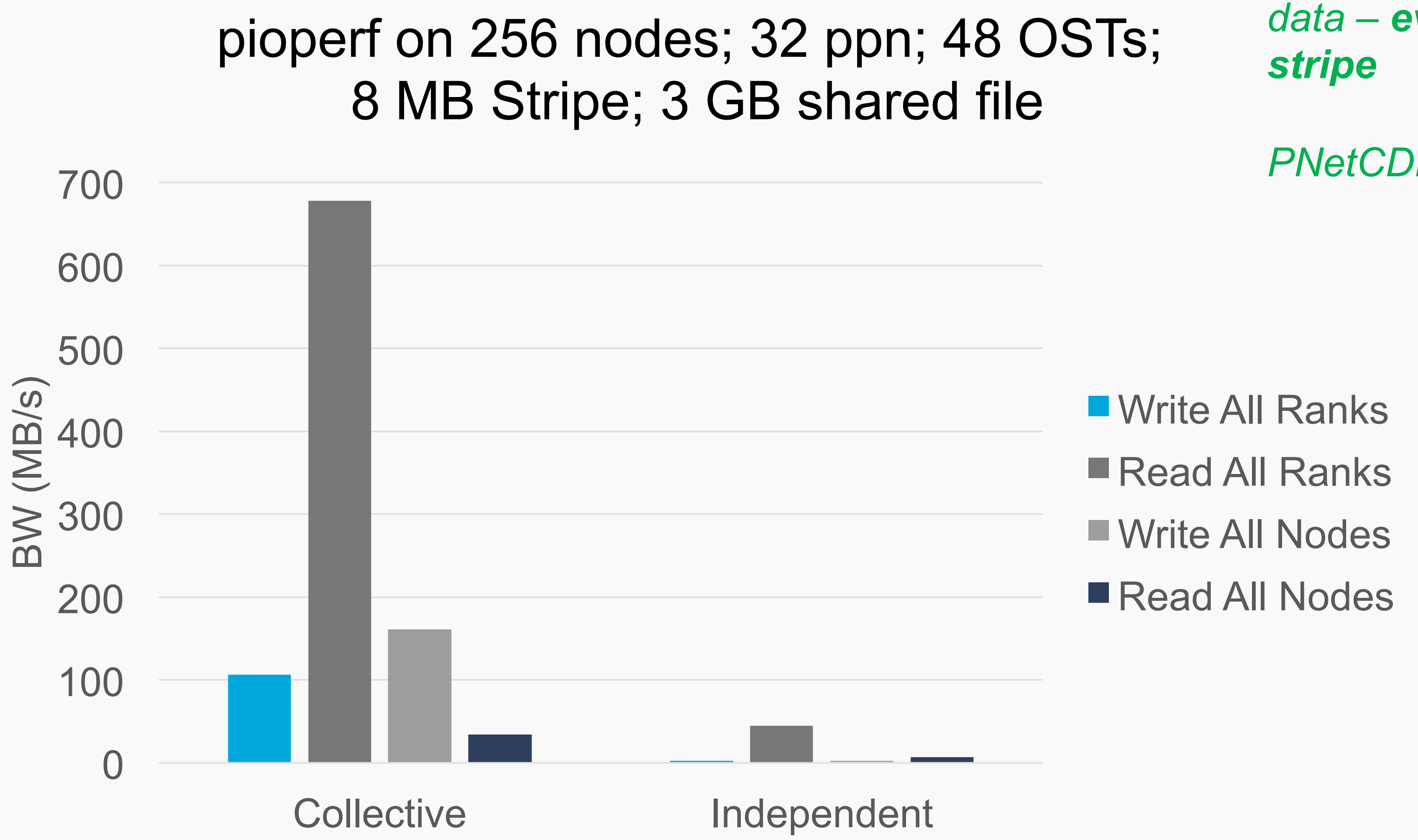
Raw File write linearly better (MPI-IO 1.5x faster at 4)

Collective I/O vs Independent I/O
Discontiguous Data

E3SM Climate Modeling Parellel I/O Library performance test tool (pioperf)

*8192 ranks with highly non-contiguous data – **every rank accesses every stripe***

PNetCDF interface (MPI-IO backend)



Node-Local SSD Utilization on Theta

Node Local SSDs on Theta – NOT a Burst Buffer

Local 128 GB SSD attached to each node

- Need to be granted access – PI contact support@alcf.anl.gov

  https://www.alcf.anl.gov/user-guides/running-jobs-xc40#requesting-local-ssd-requirements

SSD Use Cases:

- Store local intermediate files (scratch)
- Legacy code initialization with lots of small data files – every rank reads
  - Untar into local ssd

Tiered storage utility currently unavailable (Under investigation)

Using the SSDs on Theta

To request SSD, add the following in your `qsub` command line:

- **`--attrs ssds=required:ssd_size=128`**
  - This is in addition to any other attributes that you need
  - `ssd_size` is optional

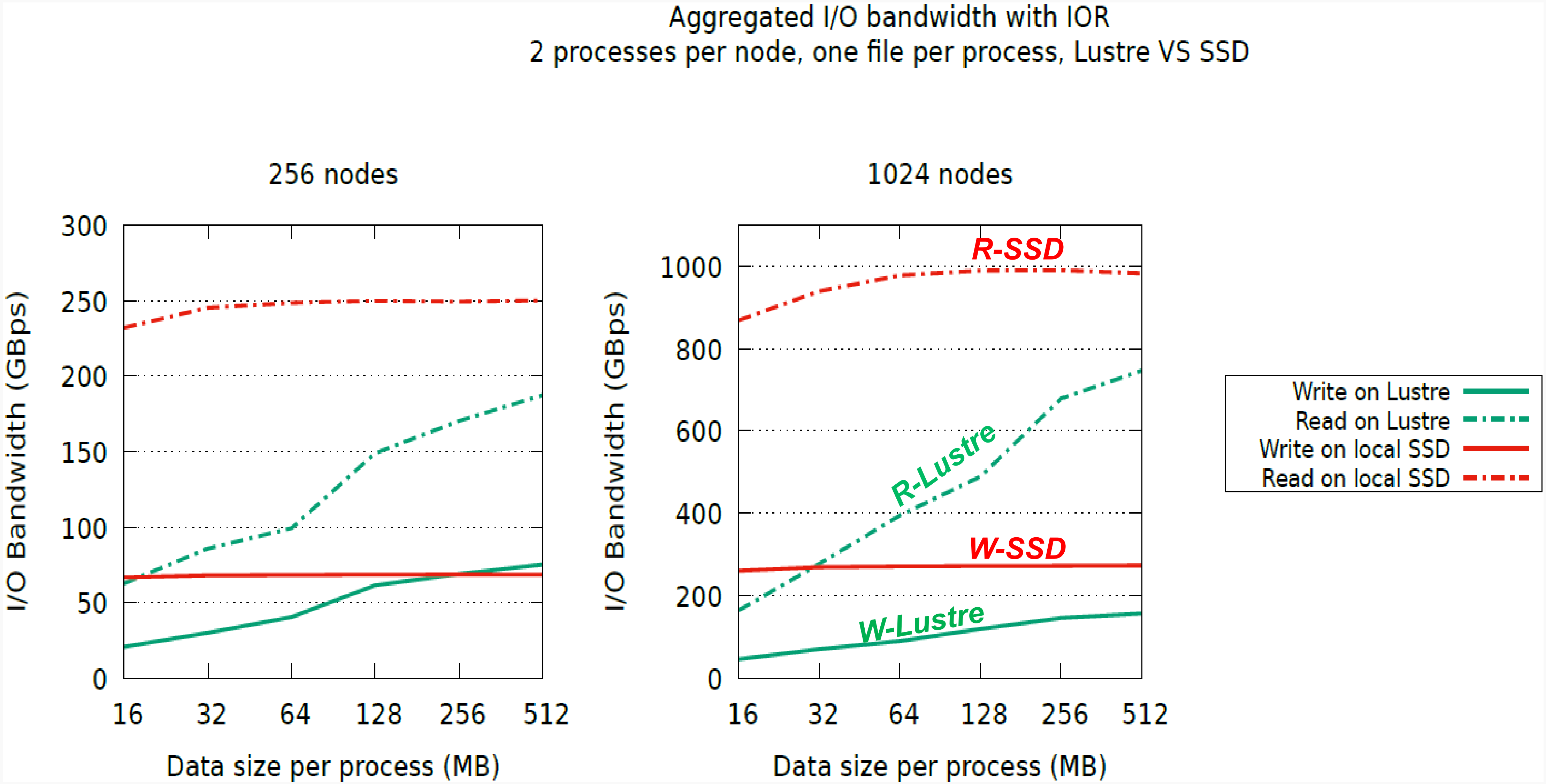
The SSD are mounted on `/local/scratch` on each node

- Data deleted when cobalt job terminates

SSD I/O Performance (per process): Read **1.1 GB/s** – Write **175 MB/s**

- Can scale to two processes
- Outperforms Lustre at scale (aggregated bandwidth)
- Node-limited scope
- Requires explicit manual programming

Node-Local SSD Performance



Argonne NATIONAL LABORATORY

Conclusion

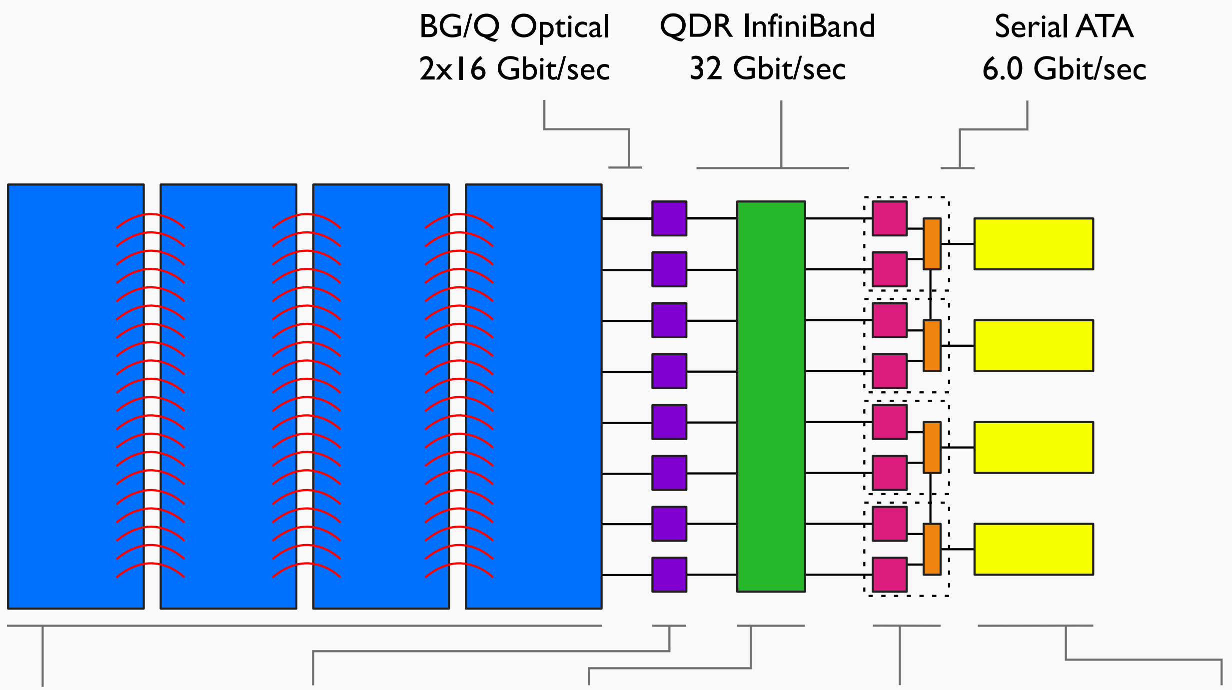
- High-performance I/O on both Mira and Theta often require MPI-IO (or an I/O library)

- Key to Theta is efficient Lustre access
  - Choose appropriate striping
  - Use optimized Cray MPI-IO
  - Use I/O libraries (HDF5, PNetCDF)
  - No tiered storage burst buffer implementation yet

ALCF Staff is available to help!

Appendix

Mira I/O Infrastructure: More Information



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

Other BG/Q Recommendations

Best to avoid file-per-process

- **10^3** files may be okay, but 10^4+ will become problematic

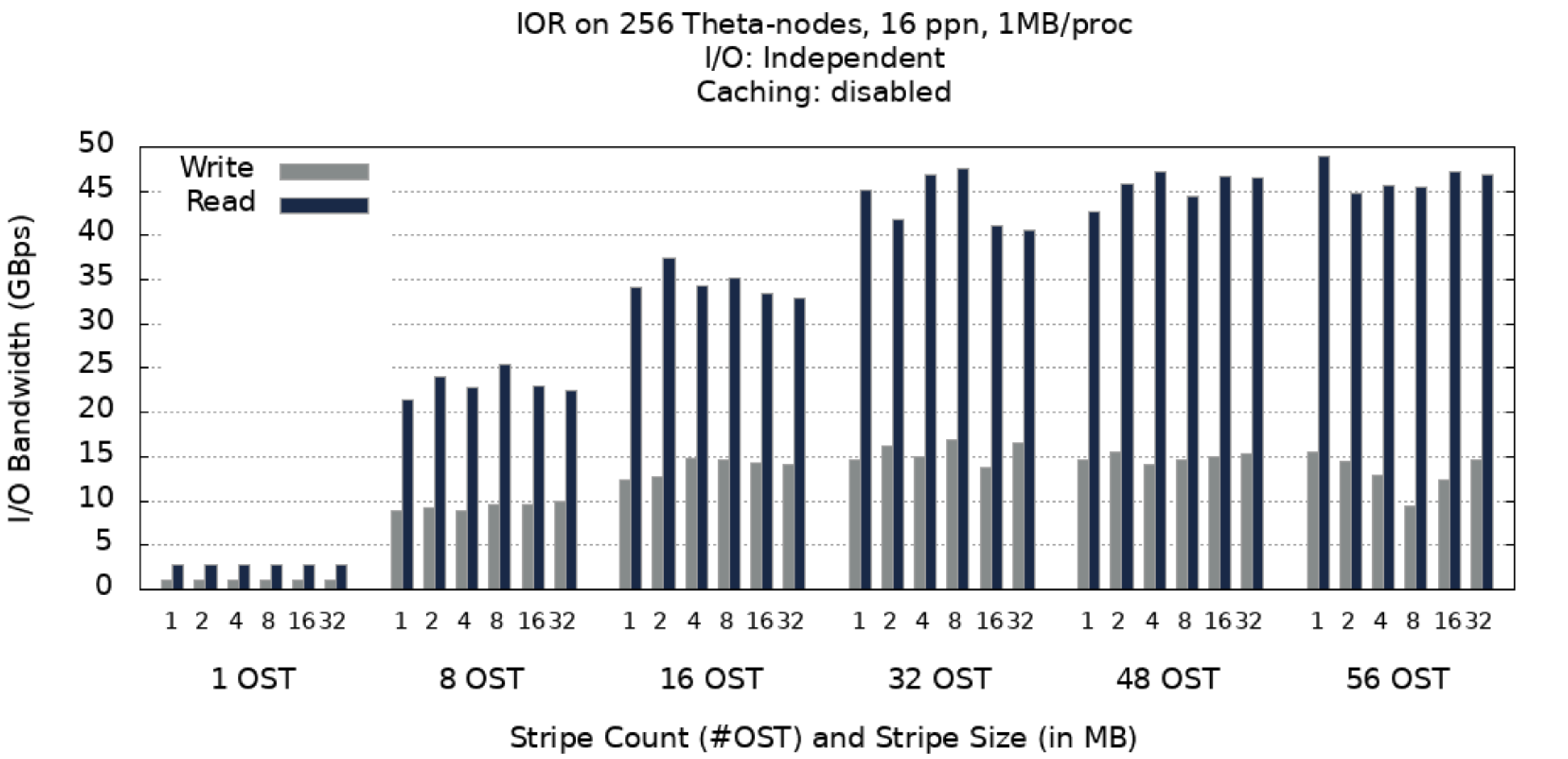
If file-per-process is a must…

- Pre-create the files before the job runs
- *or* Use a unique (pre-created) directory per file
- *or* Create all the files on 1 rank first, then reopen the files on the other ranks

POSIX note: Instead of `lseek` and `write`, use `pwrite`

Shared File – 1MB/proc – Independent I/O
Client-side Caching DISABLED

32 OSTs is sufficient
1 MB stripe size is sufficient



Argonne NATIONAL LABORATORY

Shared File – 1MB/proc – Independent I/O
Client-side Caching ENABLED

8 OSTs is sufficient
1 MB stripe size is sufficient