An Introduction to the Lustre Parallel File System

Tom Edwards
tedwards@cray.com

Agenda

- **Introduction to storage hardware**
  - RAID
- **Parallel Filesystems**
  - Lustre
- **Mapping Common IO Strategies to Lustre**
  - Spokesperson
  - Multiple writers – multiple files
  - Multiple writers – single file
  - Collective IO
- **Tuning Lustre Settings**
  - Case studies
- **Conclusions**

Building blocks of HPC file systems

- **Modern Supercomputer hardware is typically built on two fundamental pillars:**
  1. The use of widely available commodity (inexpensive) hardware.
  2. Using parallelism to achieve very high performance.

- **The file systems connected to computers are built in the same way**
  - Gather large numbers of widely available, inexpensive, storage devices;
  - then connect them together in parallel to create a high bandwidth, high capacity storage device.

Commodity storage

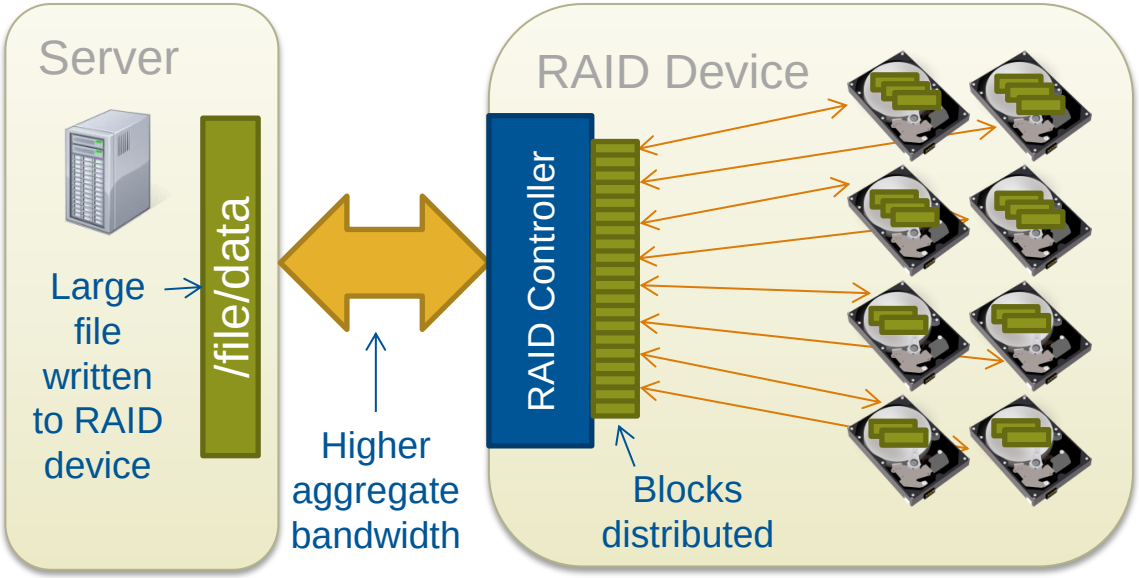
- **There are typically two commodity storage technologies that are found in HPC file-systems**

	Hard Disk Drives (HDD)	Solid State Devices (SSD)
Description	Data stored magnetically on spinning disk platters, read and written by a moving “head”	Data stored in integrated circuits, typically NAND flash memory
Advantages	• Large capacity (TBs) • Inexpensive	• Very low seek latency • High Bandwidth (~500MB/s) • Lower power draw
Disadvantages	• Higher seek latency • Lower bandwidth (<100MB/s) • Higher power draw	• Expensive • Smaller Capacity (GBs) • Limited life span

- **HDDs much more common but SSDs look promising.**
- **Both are commonly referred to as “Block Devices”**

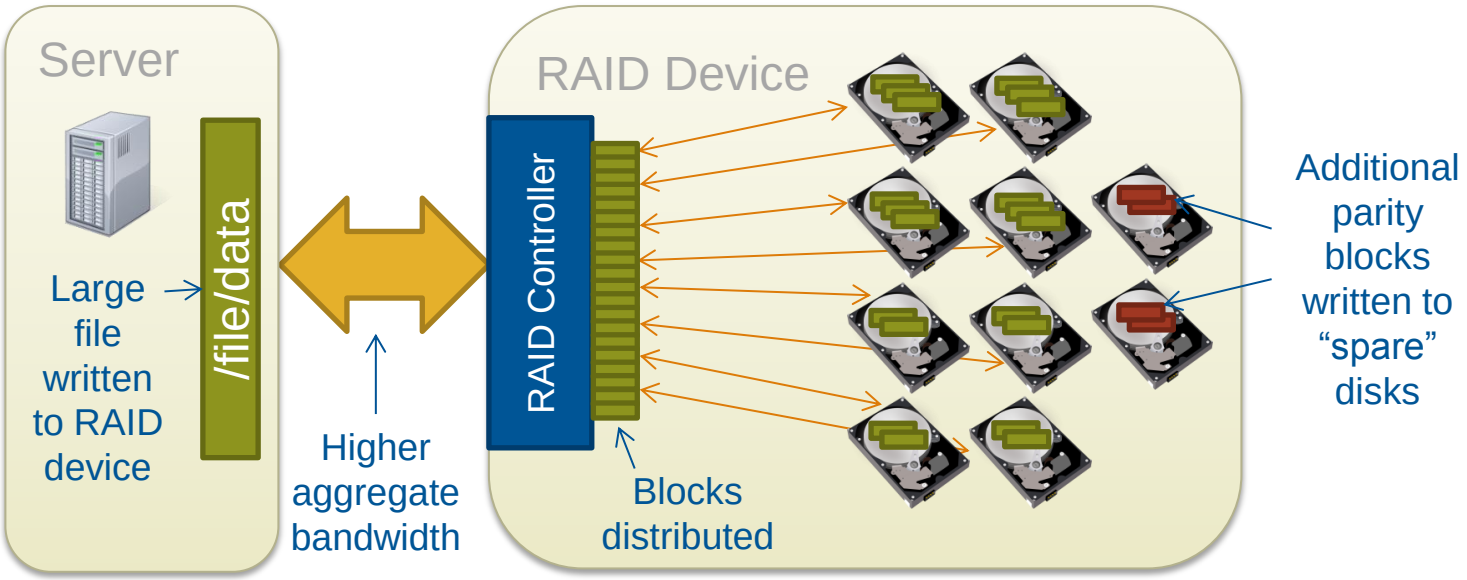
Redundant Arrays of Inexpensive Disks (RAID)

- RAID is a technology for combining multiple smaller block devices into a single larger/faster block device
- Specialist RAID controllers automatically distribute data in fixed size “blocks” or “stripes” over the individual disks
- Striping blocks over multiple disks allows data to read and written in parallel resulting in higher bandwidth – (RAID0)



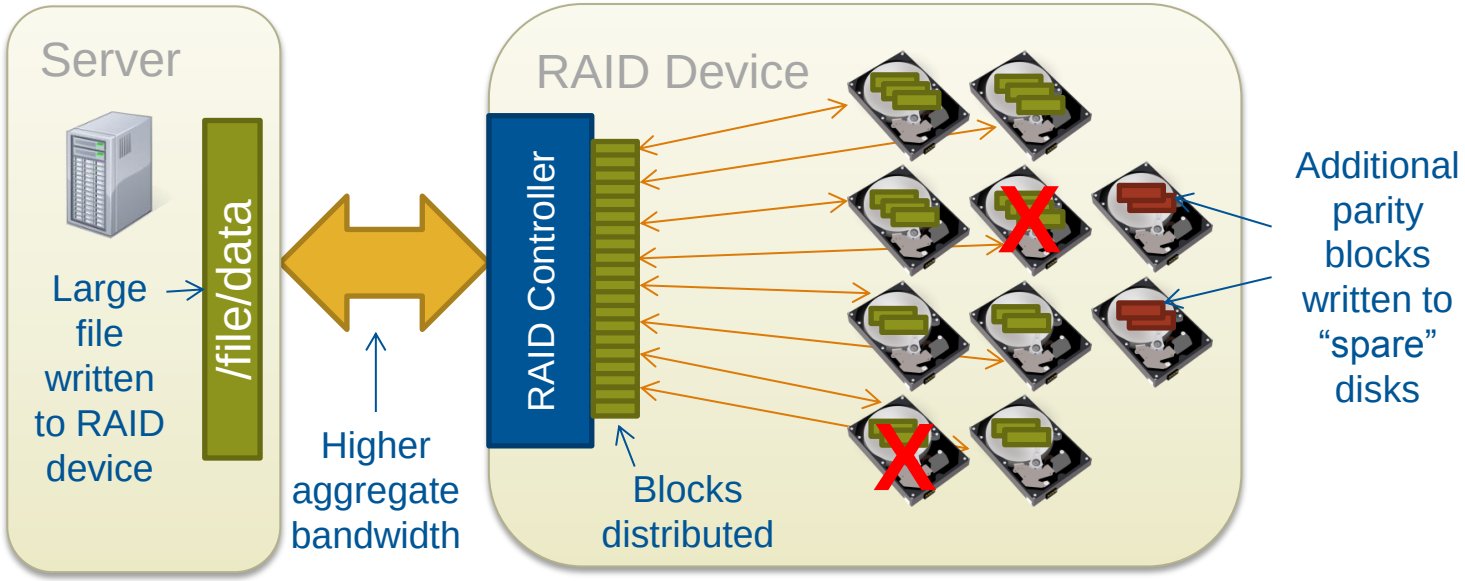
Redundant Arrays of Inexpensive Disks (RAID)

- Only using striping exposes data to increased risk as it is likely that all data will be lost if any one drive fails
- To protect against this, the controller can store additional "parity" blocks which allow the array to survive one or two disks failing – (RAID5 / RAID6)
- Additional drives are required but the data's integrity is ensured



Degraded arrays

- A RAID6 array can survive any two drives failing
- Once the faulty drives are replaced, the array has to be rebuilt from the data on the existing drives
- Rebuilds can happen while the array is running, but may take many hours to complete and will reduce the performance of the array

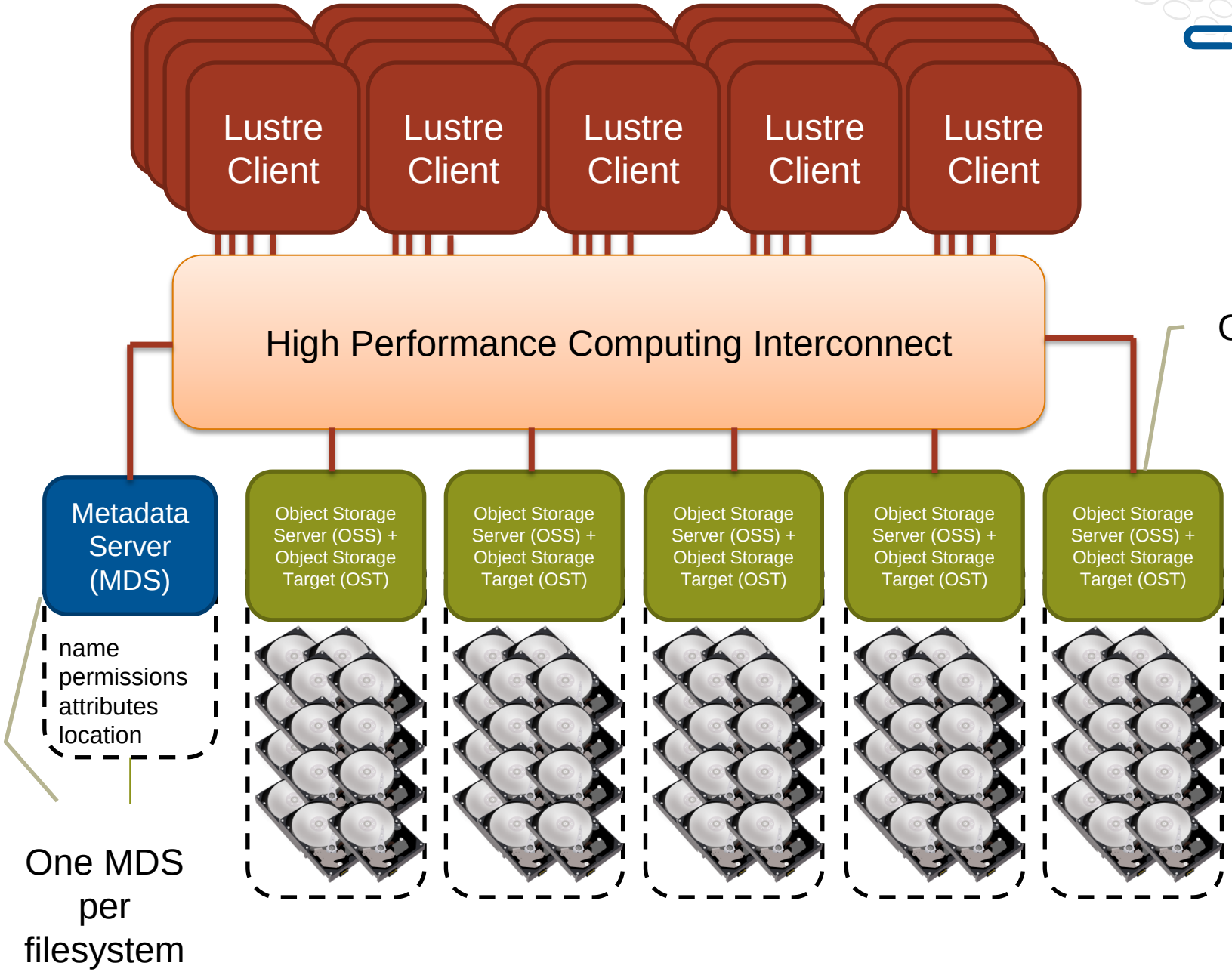


Combining RAID devices in to a parallel filesystem

- **There are economic and practical limits on the size of individual RAID6 arrays**
  - Most common arrays contain around 10 drives
  - This limits capacity to Terabytes and bandwidth to a few GB/s
  - It may also be difficult to share the file system with many client nodes.
- **To achieve required performance supercomputers combine multiple RAID devices to form a single parallel file system**
- **ARCHER and many other supercomputers use the Lustre parallel file system**
  - Lustre joins multiple block devices (RAID arrays) into a single file system that applications can read/write from/to in parallel.
  - Scales to hundreds of block devices and 100,000s of client nodes.

Lustre Building Blocks - OSTs

- **Object Storage Targets (OST) – These are block devices that data will be distributed over. These are commonly RAID6 arrays of HDDs.**
- **Object Storage Server (OSS) – A dedicated server that is directly connected to one or more OSS. These are usually connected to the supercomputer via a high performance network**
- **MetaData Server (MDS) – A single server per file system that is responsible for holding meta data on individual files**
  - Filename and location
  - Permissions and access control
  - Which OSTs data is held on.
- **Lustre Clients – Remote clients that can mount the Lustre filesystem, e.g. Cray XC30 Compute nodes.**


Lustre Client
Lustre Client
Lustre Client
Lustre Client
Lustre Client
High Performance Computing Interconnect
Multiple OSSs and OSTS
Metadata Server (MDS)
name
permissions
attributes
location
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
One MDS per filesystem

ARCHER's Lustre – Cray Sonexion Storage

MMU: *Metadata Management Unit*

1

Lustre MetaData Server

- Contains server hardware and storage

SSU: *Scalable Storage Unit*

2

2 x OSSs and 8 x OSTs

- Contains Storage controller, Lustre server, disk controller and RAID engine
- Each unit is 2 OSSs each with 4 OSTs of 10 (8+2) disks in a RAID6 array

3

Multiple SSUs are combined to form storage racks

ARCHER's File systems

Connected to the Cray XC30 via LNET router service nodes.

Infiniband Network

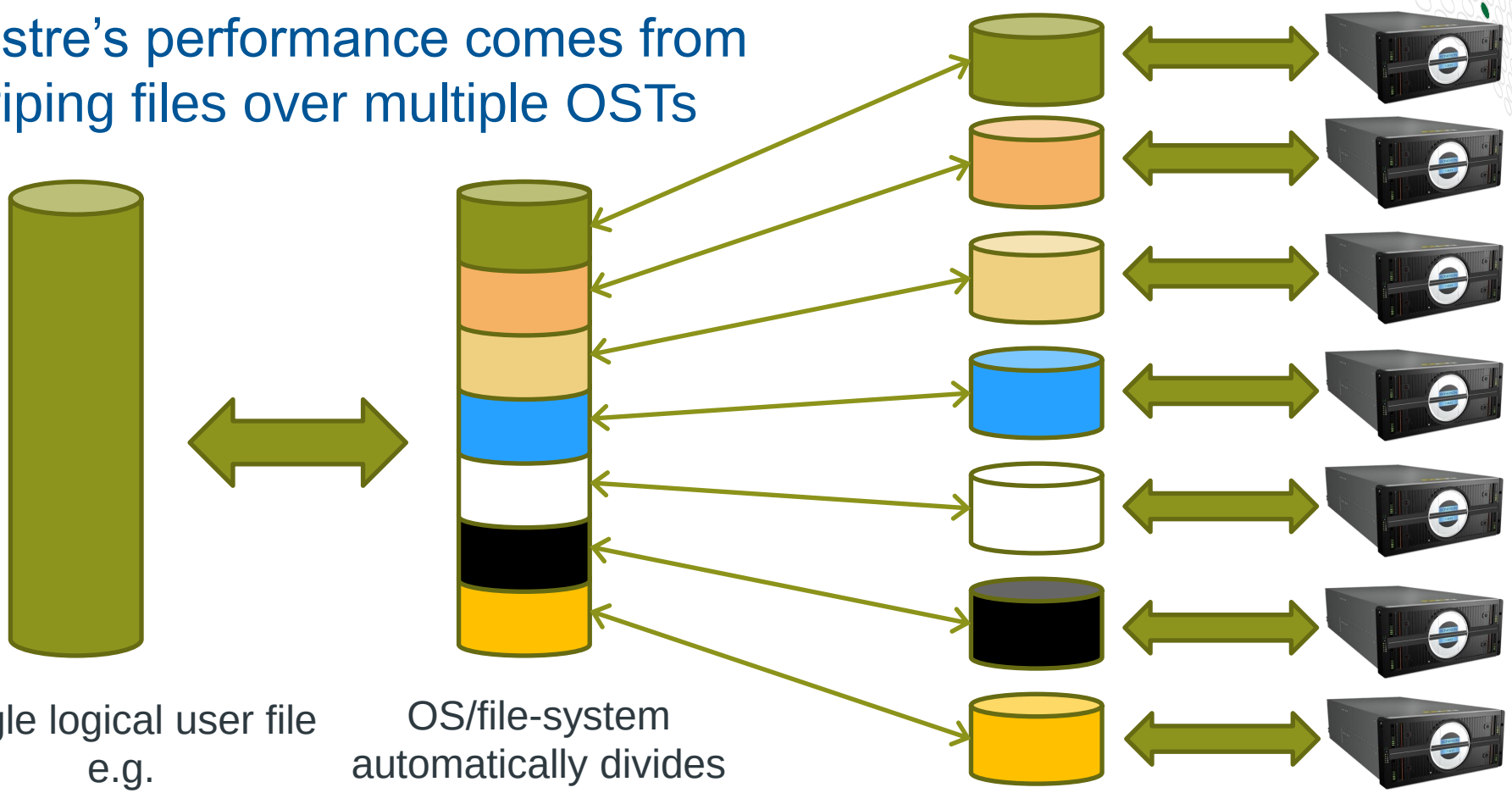
/fs2
6 SSUs
12 OSSs
48 OSTs
480 HDDs
4TB per HDD
1.4 PB Total

/fs3
6 SSUs
12 OSSs
48 OSTs
480 HDDs
4TB per HDD
1.4 PB Total

/fs4
7 SSUs
14 OSSs
56 OSTs
560 HDDs
4TB per HDD
1.6 PB Total

Lustre data striping

Lustre's performance comes from striping files over multiple OSTs



Single logical user file
e.g.
`/work/y02/y02/ted`

OS/file-system automatically divides the file into stripes

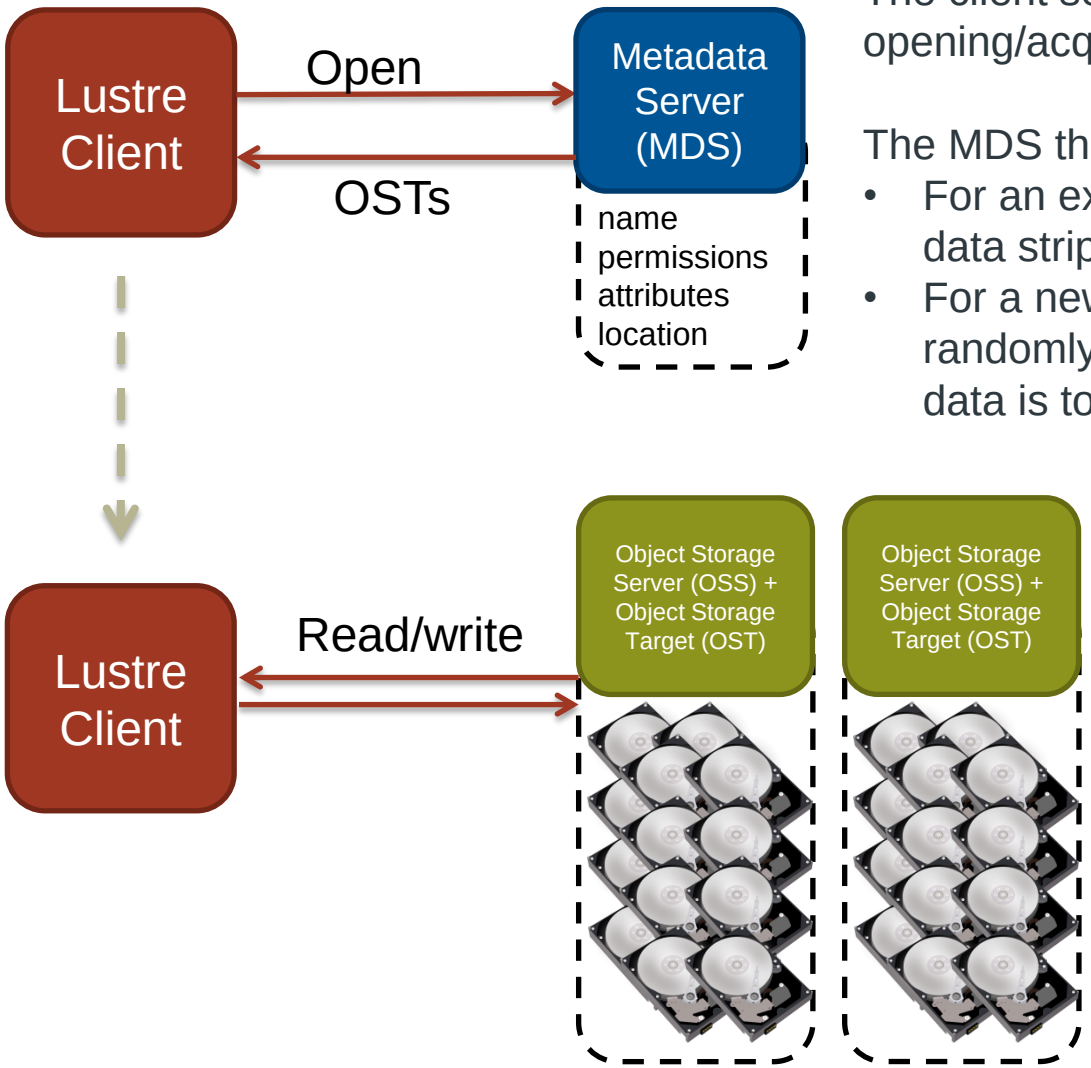
Stripes are then read/written to/from their assigned OST

RAID blocks vs Lustre Stripes

- **RAID blocks and Lustre stripes appear, at least on the surface, to perform the similar function, however there are some important differences.**

	RAID Stripes/Blocks	Lustre Stripes
Redundancy	RAID OSTs are typically configured with RAID6 to ensure data integrity if an individual drives failed	Lustre provides no redundancy, if an individual OST becomes available, all files using that array are inaccessible
Flexibility	The block/stripe size and distribution is chosen at when the array is created and cannot be changed by the user	The number and size of the Lustre stripes used can be controlled by the user on a file-by-file when the file is created (see later).
Size		Lustre stripe sizes are usually between 1 and 32 MB

Opening a file



The client sends a request to the MDS to opening/acquiring information about the file

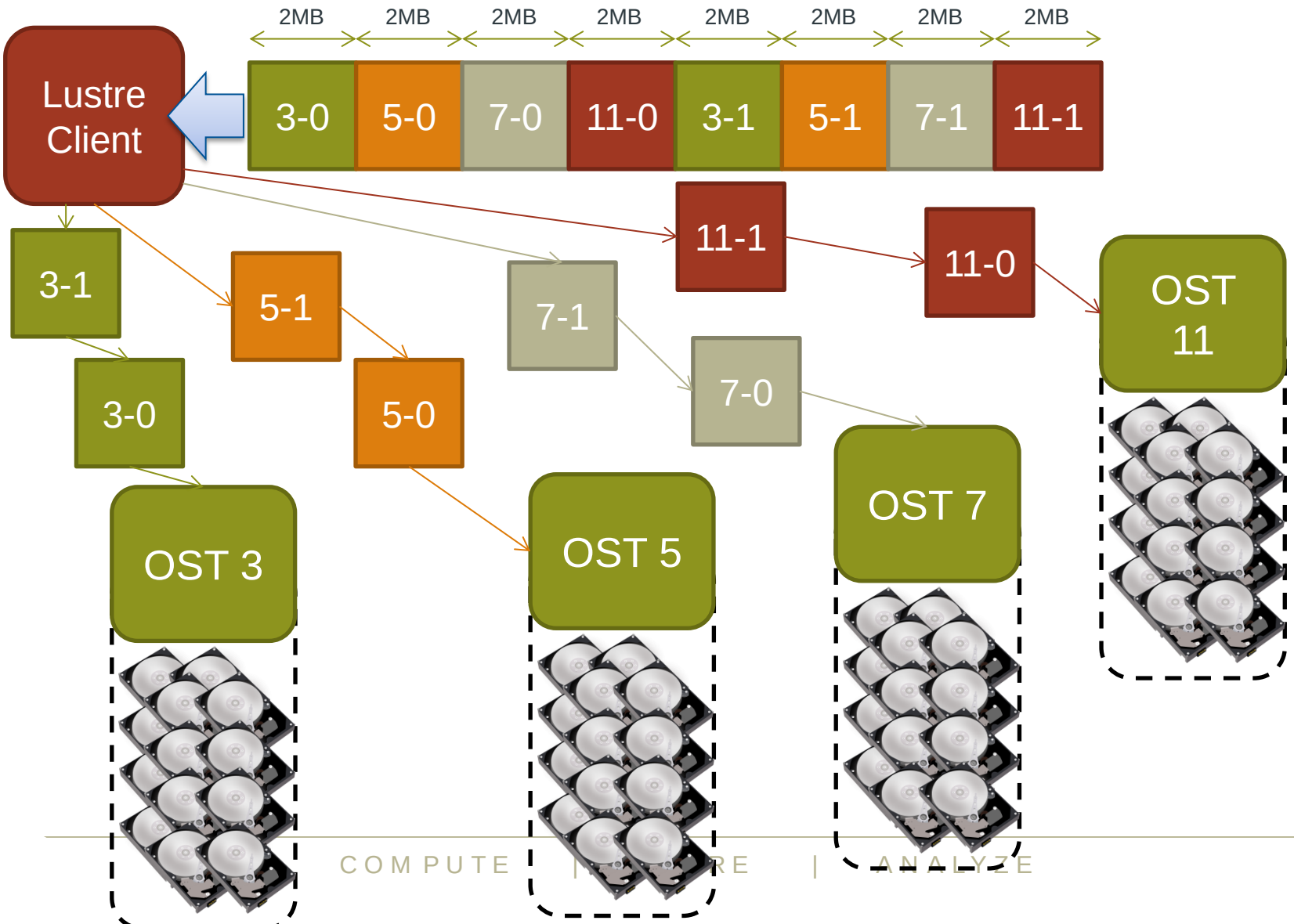
The MDS then passes back a list of OSTs

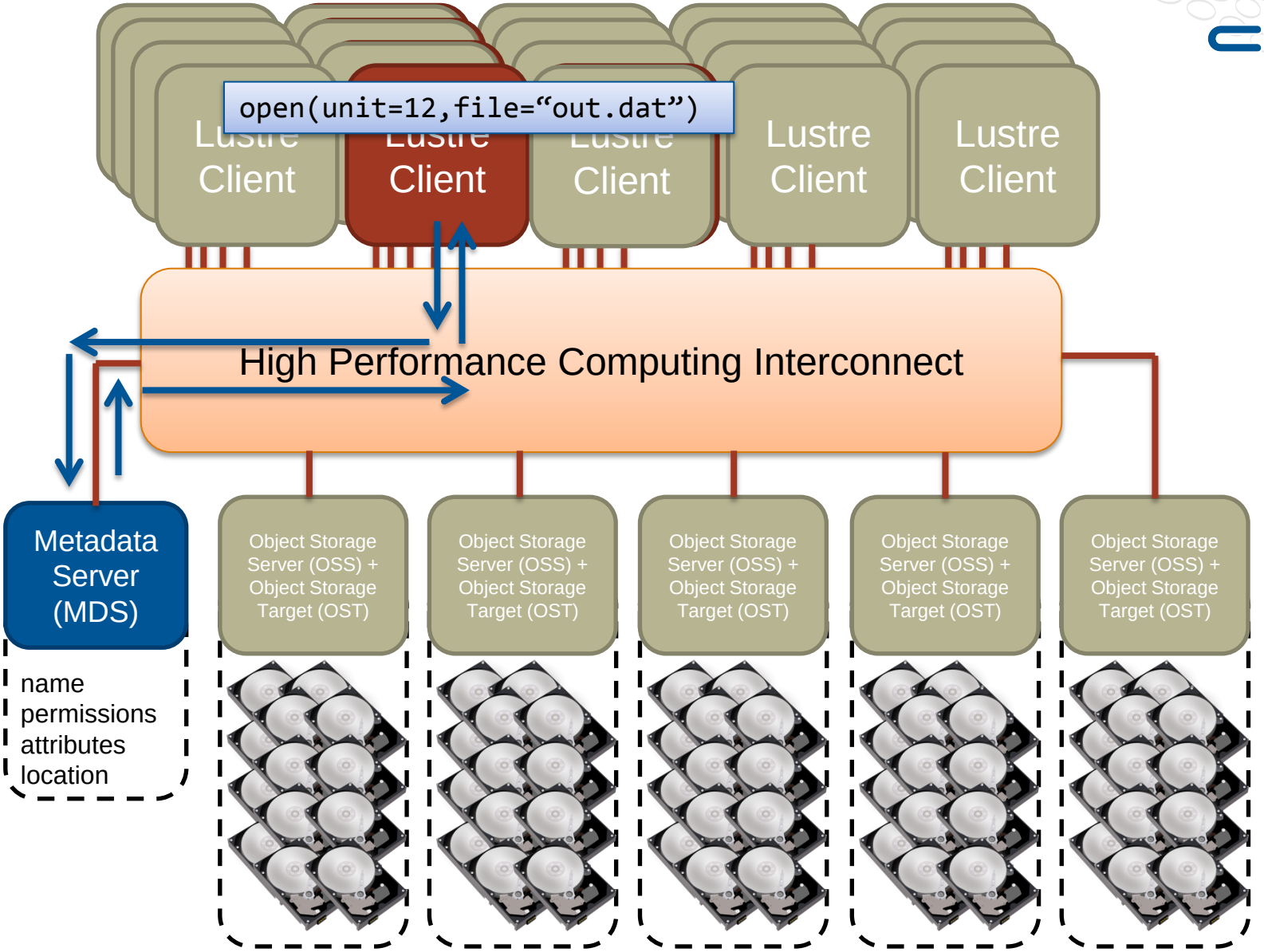
- For an existing file, these contain the data stripes
- For a new files, these typically contain a randomly assigned list of OSTs where data is to be stored

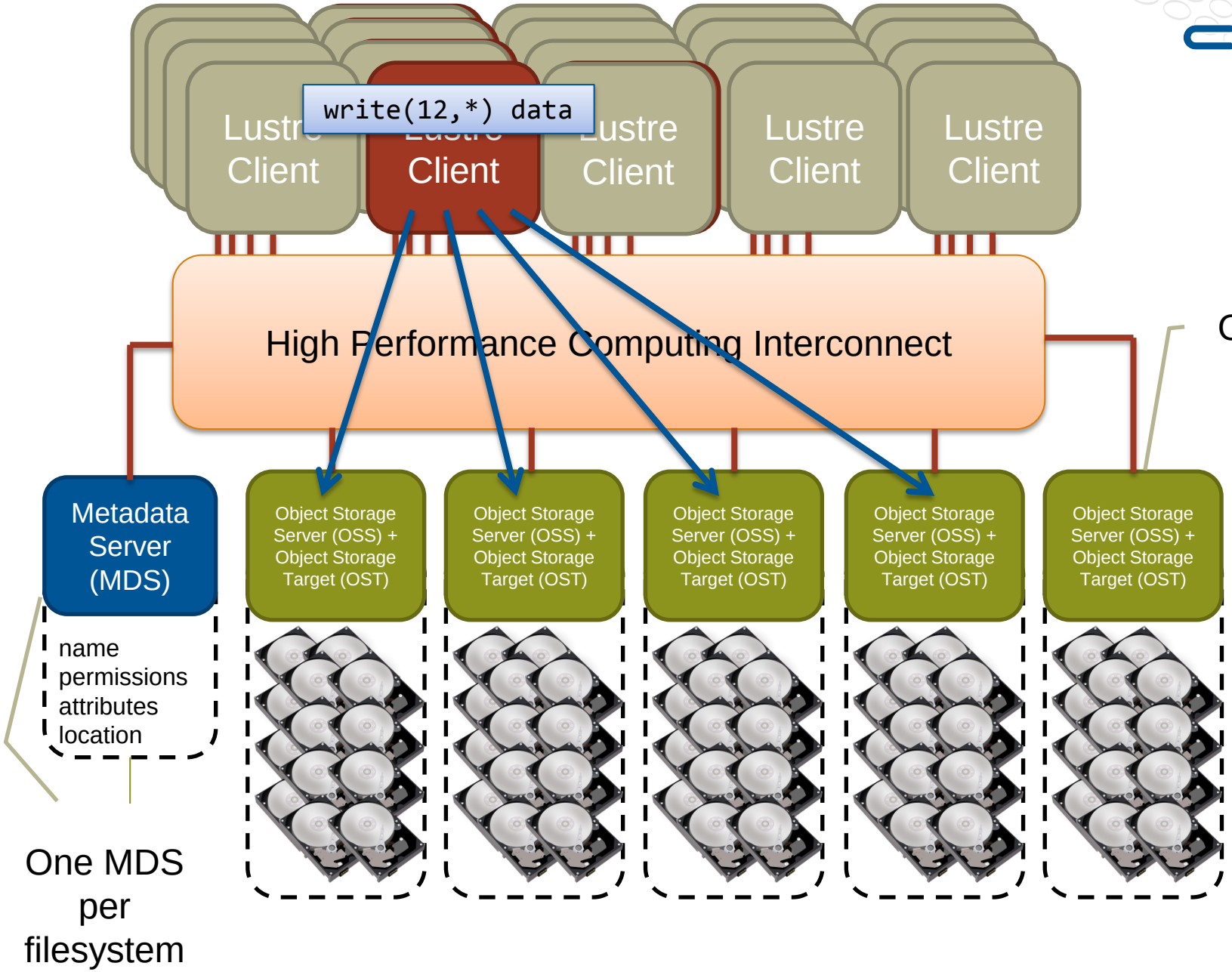
Once a file has been opened no further communication is required between the client and the MDS

All transfer is directly between the assigned OSTs and the client

File decomposition – 2 Megabyte stripes




open(unit=12,file="out.dat")
Lustre Client
Lustre Client
Lustre Client
Lustre Client
Lustre Client
High Performance Computing Interconnect
Metadata Server (MDS)
name
permissions
attributes
location
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)


write(12,*) data
Lustre Client
Lustre Client
Lustre Client
Lustre Client
Lustre Client
High Performance Computing Interconnect
Multiple OSSs and OSTS
Metadata Server (MDS)
name
permissions
attributes
location
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
Object Storage Server (OSS) + Object Storage Target (OST)
One MDS per filesystem

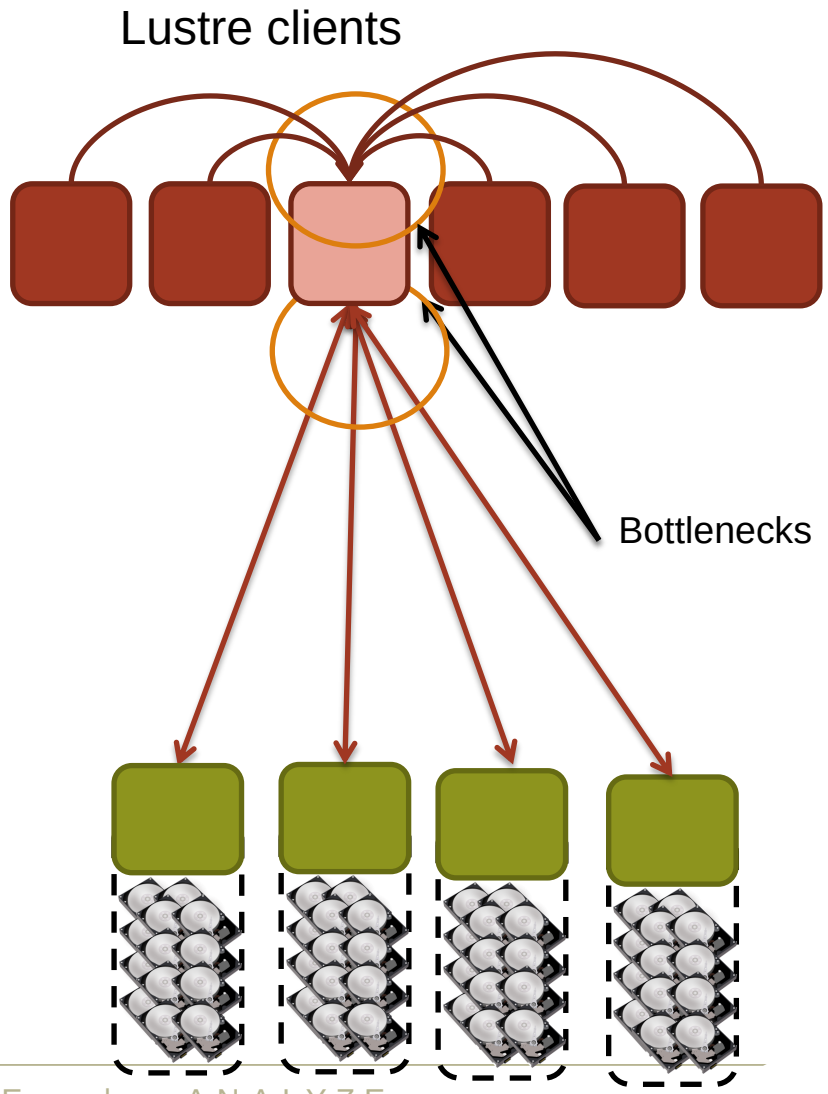
Key points

- **Lustre achieves high performance through parallelism**
  - Best performance from multiple clients writing to multiple OSTs
- **Lustre is designed to achieve high bandwidth to/from a small number of files**
  - Typical use case is a scratch file system for HPC
  - It is a good match for scientific datasets and/or checkpoint data
- **Lustre is not designed to handle large numbers of small files**
  - Potential bottlenecks at the MDS when files are opened
  - Data will not be spread over multiple OSTs
  - Not a good choice for compilation
- **Lustre is NOT a bullet-proof file system.**
  - If an OST fails, all files using that OST are inaccessible
  - Individual OSTs may use RAID6 but this is a last resort
  - **BACKUP important data elsewhere!**

Mapping Common I/O Patterns to Lustre

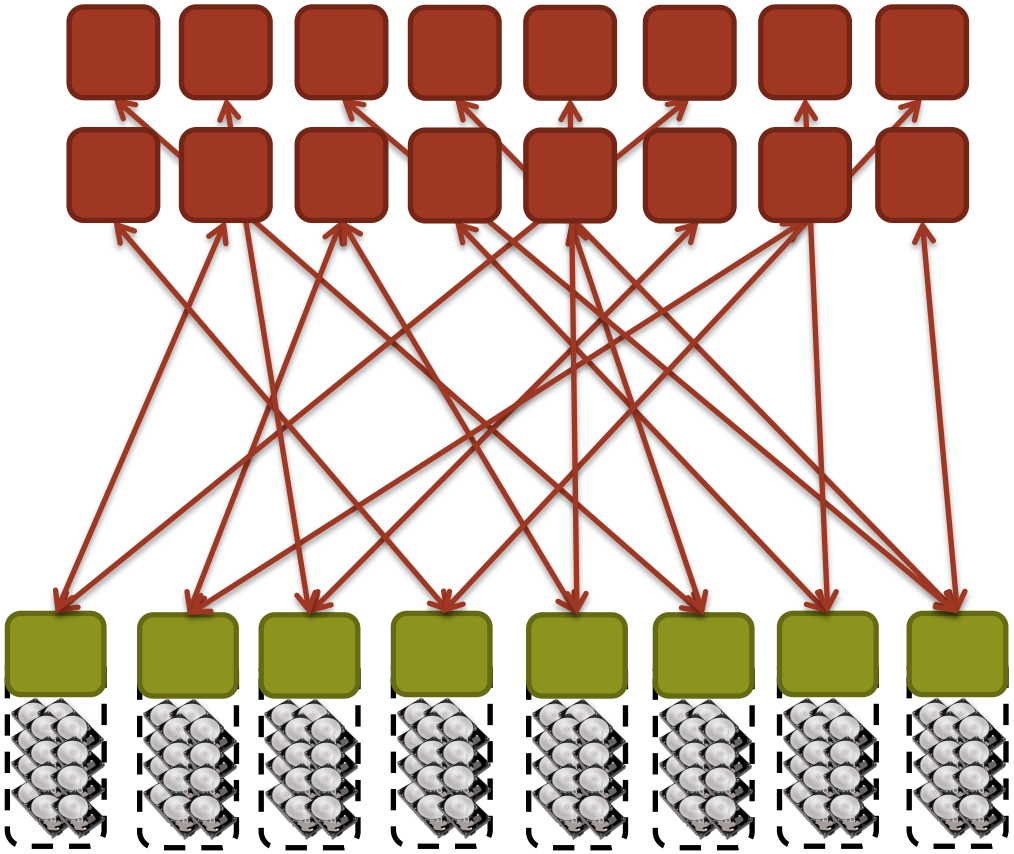
I/O strategies: Spokesperson

- **One process performs I/O**
  - Data Aggregation or Duplication
  - Limited by single I/O process
- **Easy to program**
- **Pattern does not scale**
  - Time increases linearly with amount of data
  - Time increases with number of processes
- **Care has to be taken when doing the all-to-one kind of communication at scale**
- **Can be used for a dedicated I/O Server**



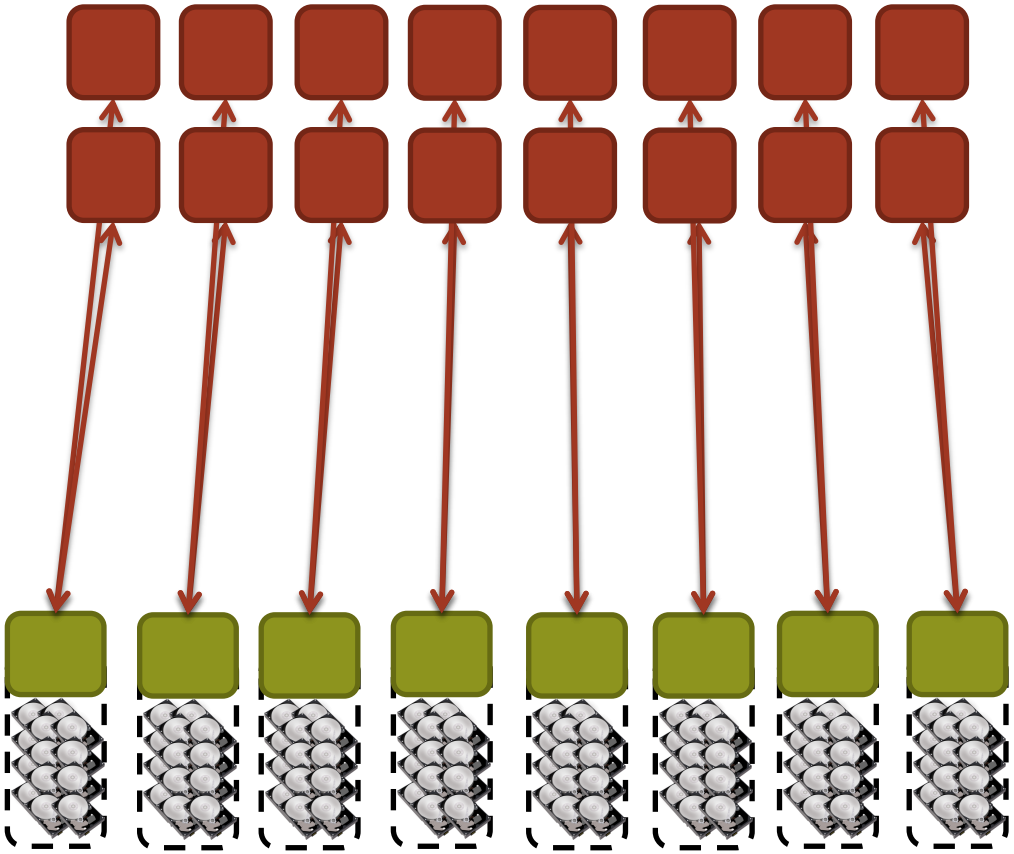
I/O strategies: Multiple Writers – Multiple Files

- **All processes perform I/O to individual files**
  - Limited by file system
- **Easy to program**
- **Pattern may not scale at large process counts**
  - Number of files creates bottleneck with metadata operations
  - Number of simultaneous disk accesses creates contention for file system resources



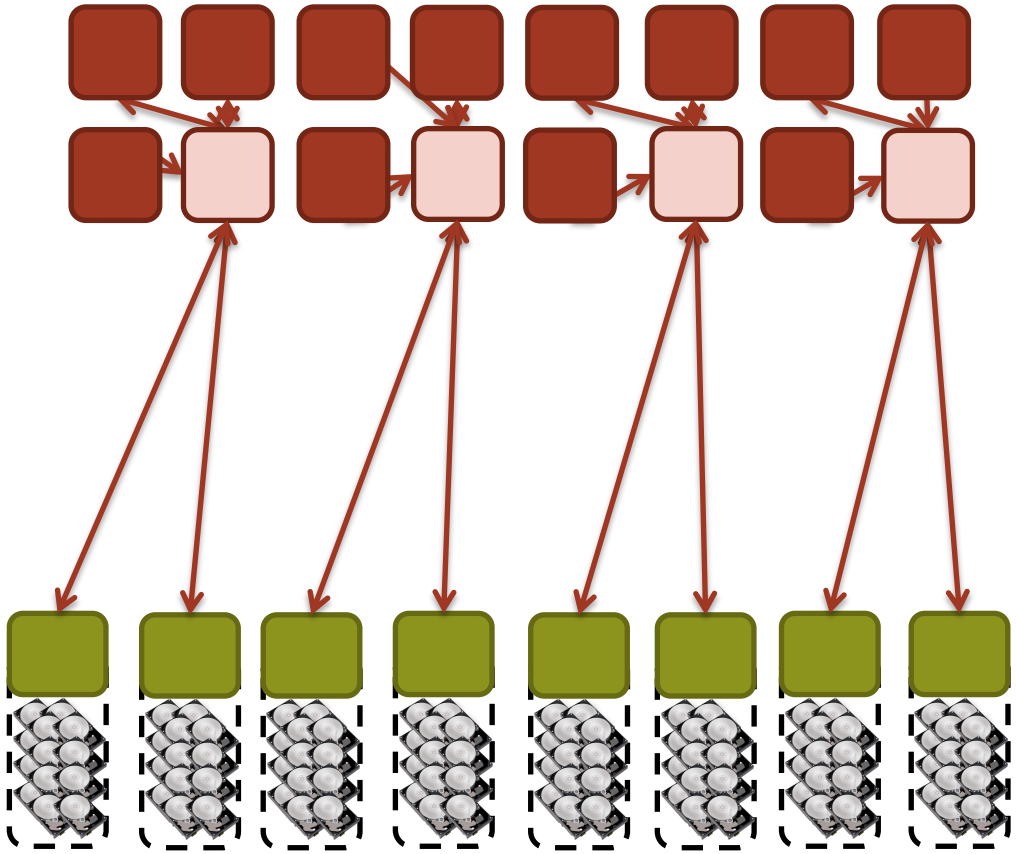
I/O strategies: Multiple Writers – Single File

- **Each process performs I/O to a single file which is shared.**
- **Performance**
  - Data layout within the shared file is very important.
  - At large process counts contention can build for file system resources.
- **Not all programming languages support it**
  - C/C++ can work with fseek
  - No real Fortran standard



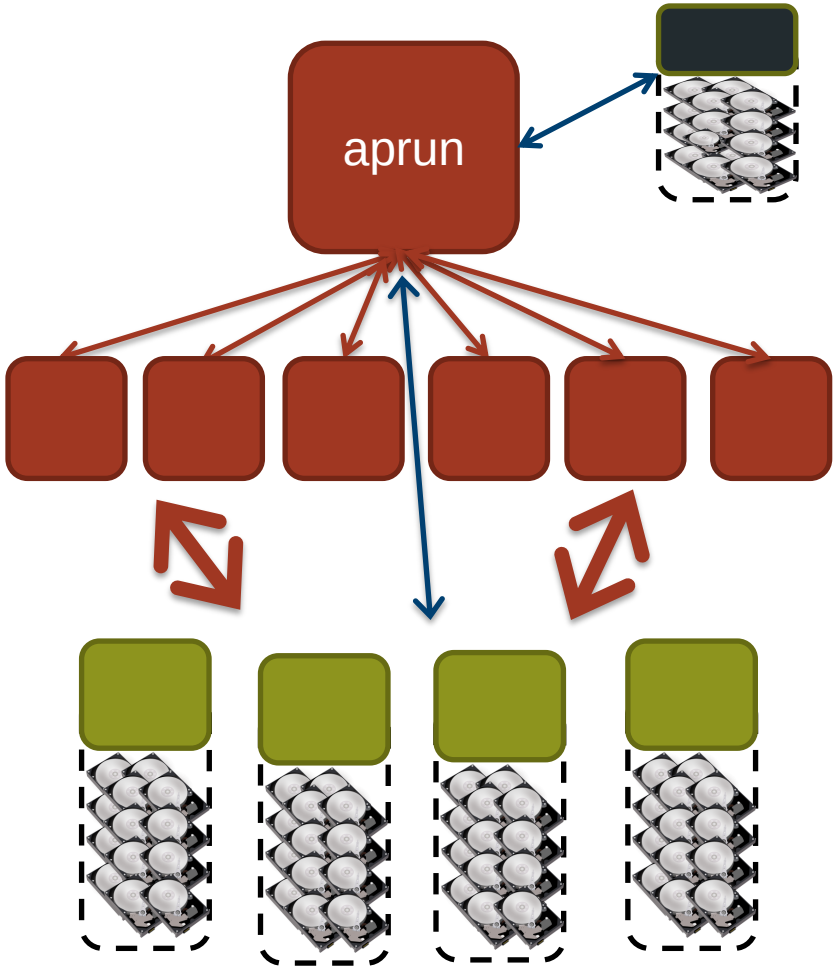
I/O strategies: Collective IO to single or multiple files

- **Aggregation to a processor in a group which processes the data.**
  - Serializes I/O in group.
- **I/O process may access independent files.**
  - Limits the number of files accessed.
- **Group of processes perform parallel I/O to a shared file.**
  - Increases the number of shares to increase file system usage.
  - Decreases number of processes which access a shared file to decrease file system contention.



Special case : Standard output and error

- **All STDIN, STDOUT, and STDERR I/O streams serialize through aprun**
- **Disable debugging messages when running in production mode.**
  - “Hello, I’m task 32,000!”
  - “Task 64,000, made it through loop.”



Tuning Lustre Settings

Matching Lustre striping to an application

Controlling Lustre striping

- **`lfs` is the Lustre utility for setting the stripe properties of new files, or displaying the striping patterns of existing ones**

- **The most used options are**
  - `setstripe` – Set striping properties of a directory or new file
  - `getstripe` – Return information on current striping settings
  - `osts` – List the number of OSTs associated with this file system
  - `df` – Show disk usage of this file system

- **For help execute lfs without any arguments**

```
$ lfs
lfs > help
Available commands are:
        setstripe
        find
        getstripe
        check
        ...
```

Sample Lustre commands: lfs df

```
crayadm@thor-1:~> lfs df -h
UUID                       bytes        Used   Available Use% Mounted on
snx11183-MDT0000_UUID        2.8T       25.0G        2.7T   1% /lustre[MDT:0]
snx11183-OST0000_UUID      169.4T      120.5T       47.0T  72% /lustre[OST:0]
snx11183-OST0001_UUID      169.4T      120.7T       46.9T  72% /lustre[OST:1]
snx11183-OST0002_UUID      169.4T       79.5T       88.0T  47% /lustre[OST:2]
snx11183-OST0003_UUID      169.4T       79.8T       87.7T  48% /lustre[OST:3]
snx11183-OST0004_UUID      169.4T      116.2T       51.4T  69% /lustre[OST:4]
snx11183-OST0005_UUID      169.4T      116.2T       51.4T  69% /lustre[OST:5]
snx11183-OST0006_UUID      169.4T      116.1T       51.4T  69% /lustre[OST:6]
snx11183-OST0007_UUID      169.4T       98.5T       69.1T  59% /lustre[OST:7]

filesystem summary:         1.3P      847.4T      492.9T  63% /lustre

crayadm@thor-1:~>
```

lfs setstripe

- **Sets the stripe for a file or a directory**
- **`lfs setstripe <file|dir> <-s size> <-i start> <-c count>`**
  - size: Number of bytes on each OST (0 filesystem default)
  - start: OST index of first stripe (-1 filesystem default)
  - count: Number of OSTs to stripe over (0 default, -1 all)
- **Comments**
  - Can use lfs to create an empty file with the stripes you want (like the touch command)
  - Can apply striping settings to a directory, any children will inherit parent's stripe settings on creation.
  - The stripes of a file is given when the file is created. It is not possible to change it afterwards.
  - The start index is the only one you can specify, starting with the second OST. In general you have no control over which one is used.

Select best Lustre striping values

- **Selecting the striping values will have a large impact on the I/O performance of your application**
- **Rules of thumb:**
  1. # files > # OSTs => Set stripe_count=1
     You will reduce the lustre contention and OST file locking this way and gain performance
  2. #files == 1 => Set stripe_count=#OSTs
     Assuming you have more than 1 I/O client
  3. #files < #OSTs => Select stripe_count so that you use all OSTs
     Example : You have 8 OSTs and write 4 files at the same time, then select stripe_count=2

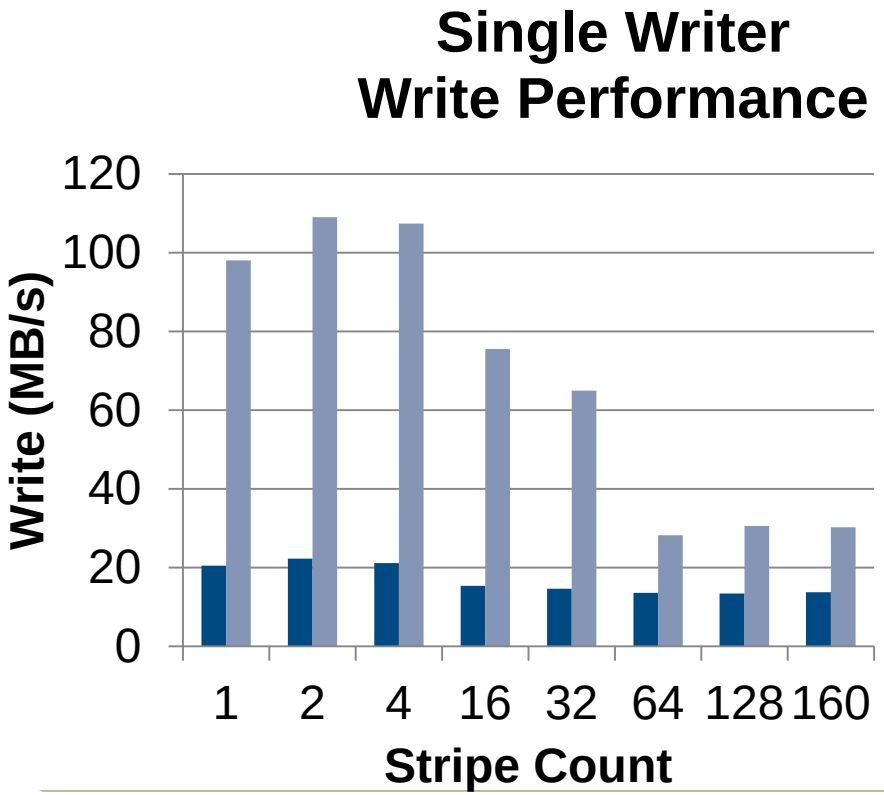
- **Always allow the system to choose OSTs at random!**

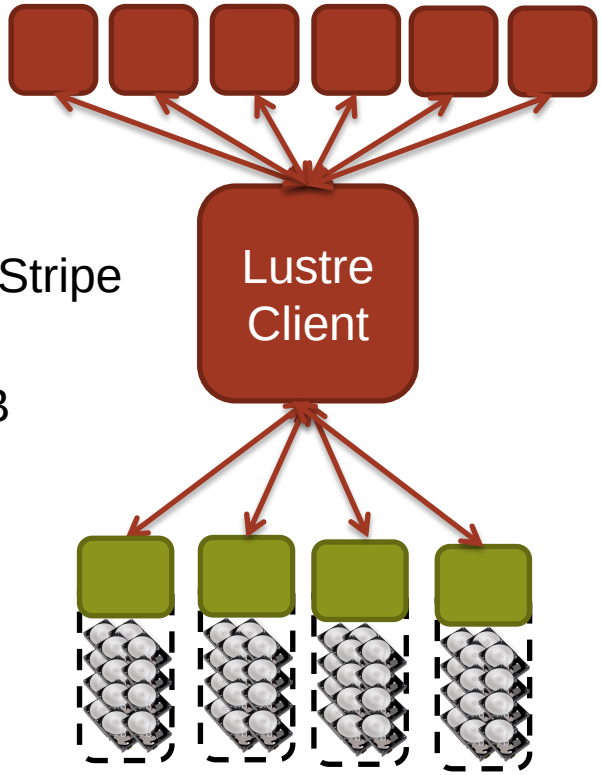
Sample Lustre commands: striping

```
crystal:ior% mkdir tigger
crystal:ior% lfs setstripe -s 2m -c 4 tigger
crystal:ior% lfs getstripe tigger
tigger
stripe_count:   4 stripe_size:    2097152 stripe_offset:  -1
crystal% cd tigger
crystal:tigger% ~/tools/mkfile_linux/mkfile BIGFILE 2g
crystal:tigger% ls -lh BIGFILE
-rw------T 1 harveyr criemp 2.0G Sep 11 07:50 BIGFILE
crystal:tigger% lfs getstripe BIGFILE
2g
lmm_stripe_count:   4
lmm_stripe_size:    2097152
lmm_layout_gen:     0
lmm_stripe_offset:  26
        obdidx           objid           objid           group
            26        33770409      0x2034ba9               0
            10        33709179      0x2025c7b               0
            18        33764129      0x2033321               0
            22        33762112      0x2032b40               0
```

Case Study 1: Spokesman

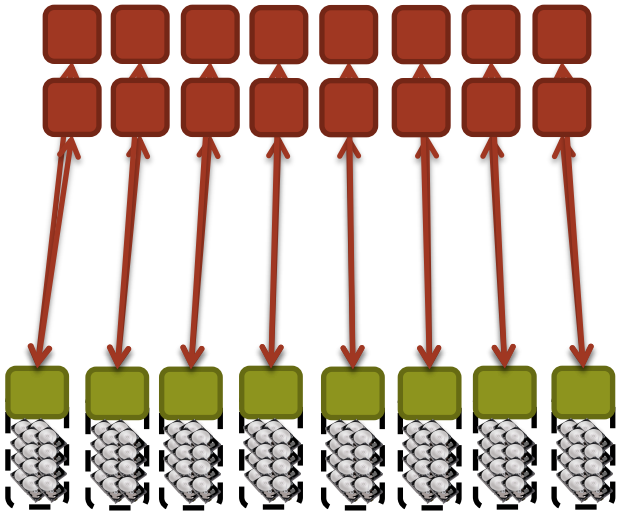
- **32 MB per OST (32 MB – 5 GB) and 32 MB Transfer Size**
  - Unable to take advantage of file system parallelism
  - Access to multiple disks adds overhead which hurts performance





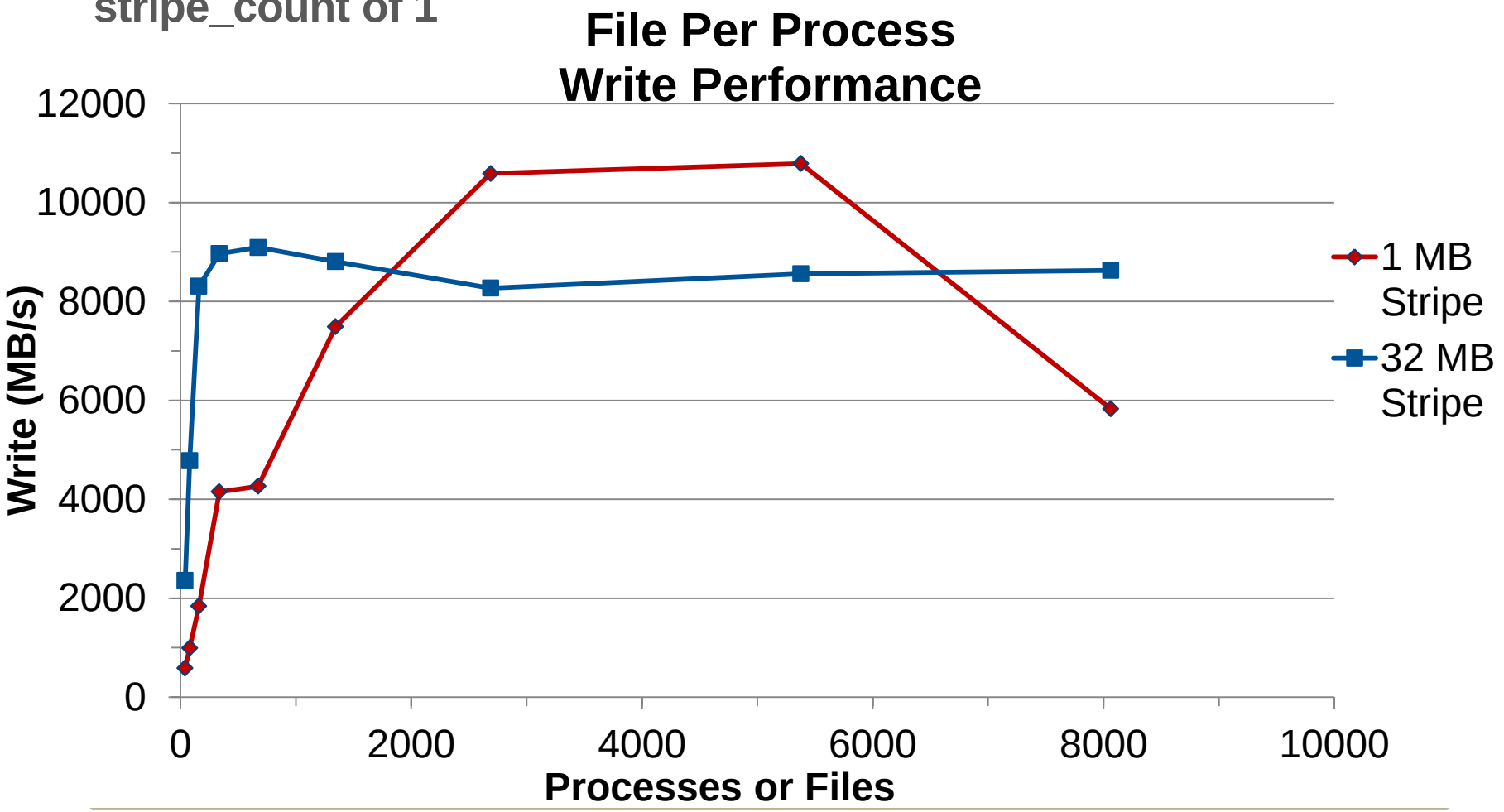
Case Study 2: Parallel I/O into a single file

- **A particular code both reads and writes a 377 GB file. Runs on 6000 cores.**
  - Total I/O volume (reads and writes) is 850 GB.
  - Utilizes parallel HDF5
- **Default Stripe settings: count =4, size=1M, index =-1.**
  - 1800 s run time (~ 30 minutes)
- **Stripe settings: count=-1, size=1M, index = –1.**
  - 625 s run time (~ 10 minutes)
- **Results**
  - 66% decrease in run time.



Case Study 3: Single File Per Process

- 128 MB per file and a 32 MB Transfer size, each file has a stripe_count of 1



Conclusions

- **Lustre is a high performance, high bandwidth parallel file system.**
  - It requires many multiple writers to multiple stripes to achieve best performance
- **There is large amount of I/O bandwidth available to applications that make use of it. However users need to match the size and number of Lustre stripes to the way files are accessed.**
  - Large stripes and counts for big files
  - Small stripes and count for smaller files
- **Lustre on ARCHER is for storing scratch data only**
  - **IT IS NOT BACKED UP!**