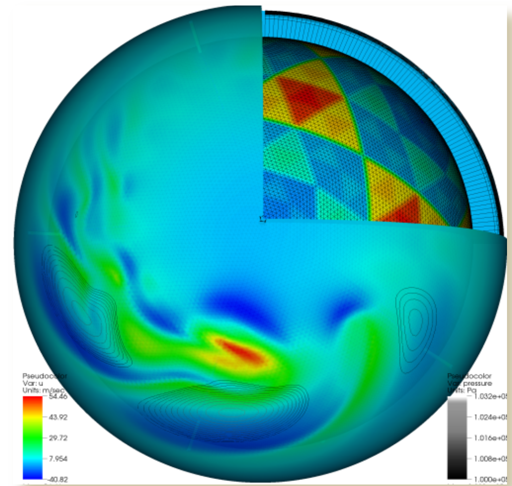
Best Practices for Reading and Writing Data on HPC Systems



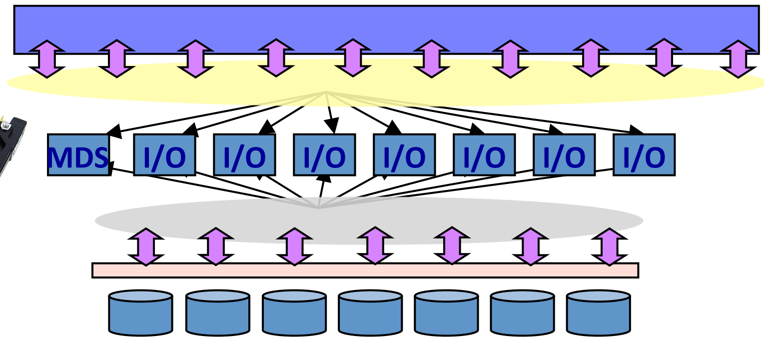
Katie Antypas

NERSC User Services
Lawrence Berkeley National Lab

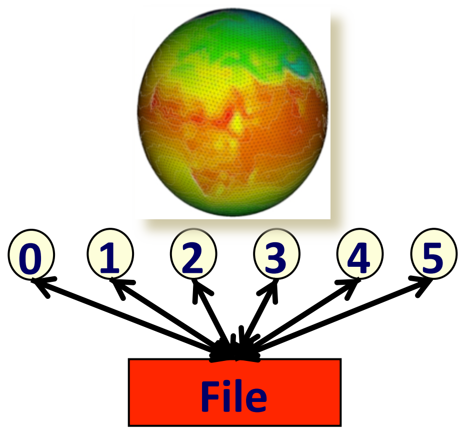
NUG Meeting

15 February 2013

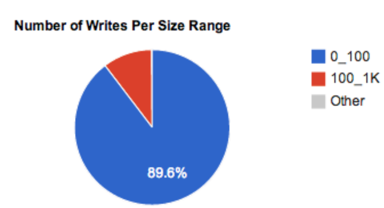
In this tutorial you will learn about I/O on HPC systems from the storage layer to the application



Storage devices and parallel file systems

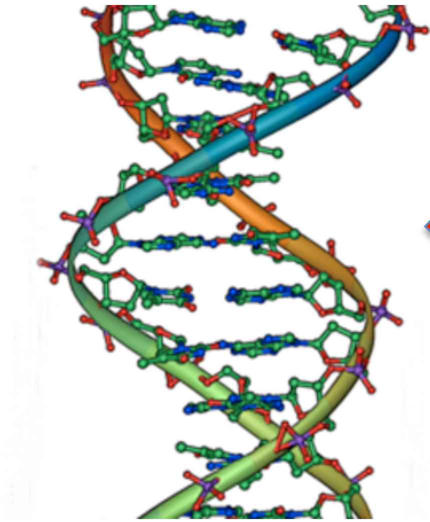


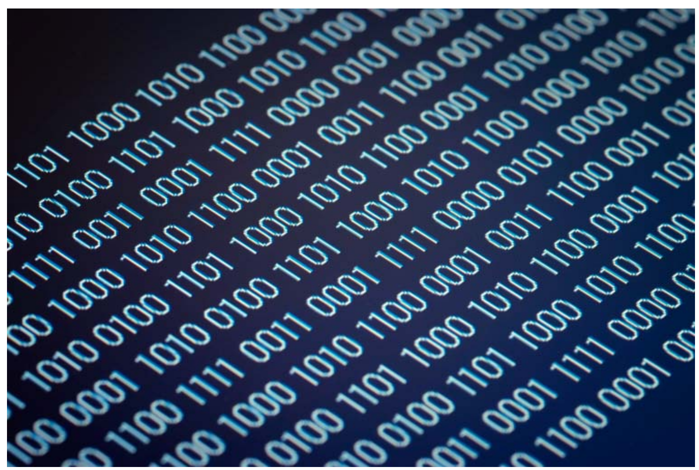
Use cases and best practices



Application I/O profiling

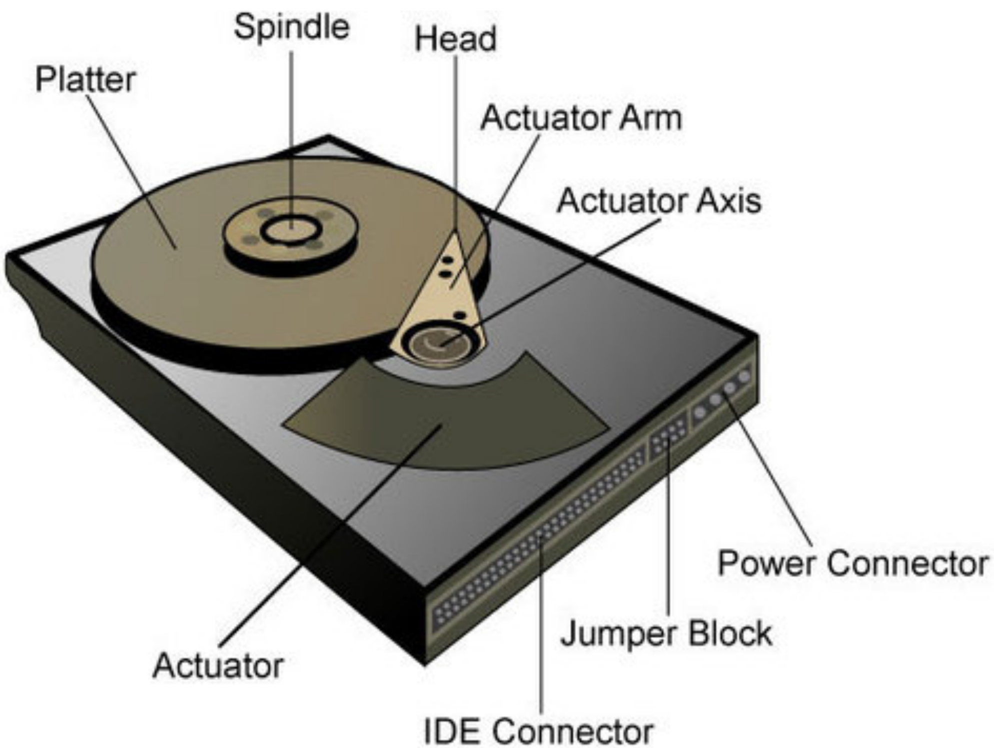
Layers between application and physical disk must translate molecules, atoms, particles and grid cells into bits of data

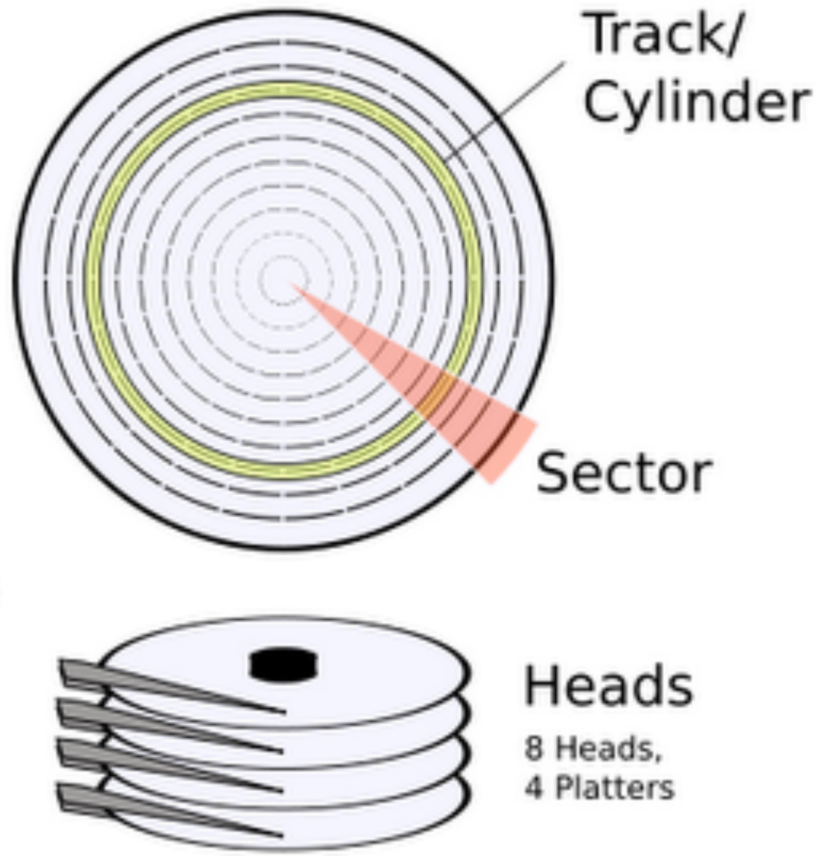




Source: Thinkstock

Despite limitations, magnetic hard drives remain the storage media of choice for scientific applications





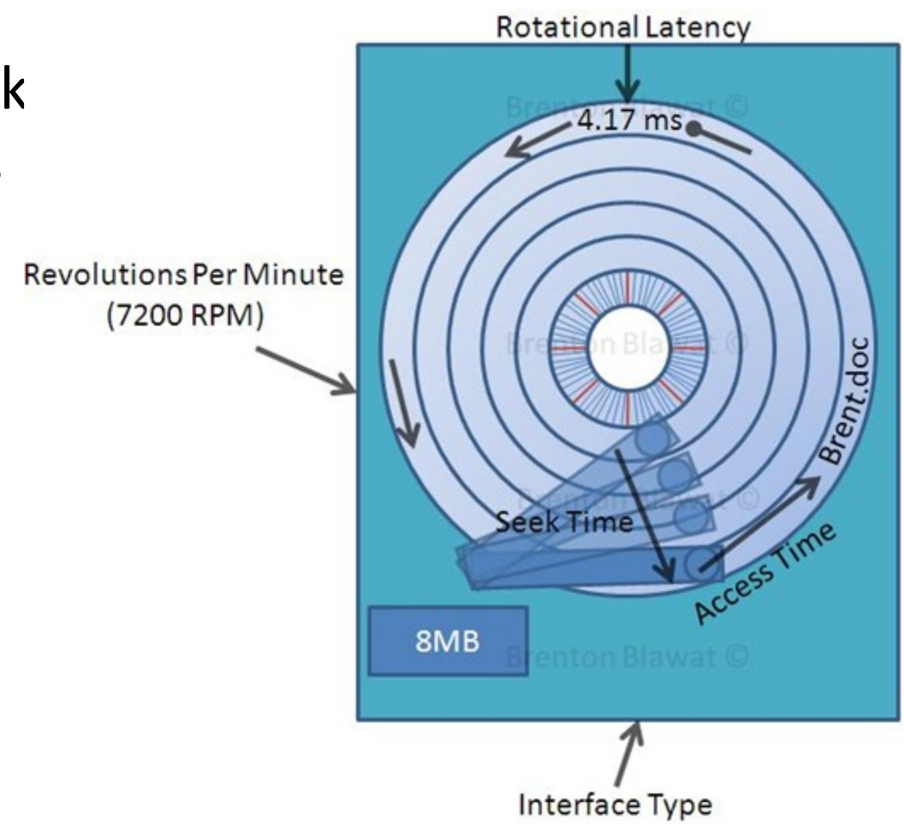
Source: Popular Science, Wikimedia Commons

The delay before the first byte is read from a disk is called "access time" and is a significant barrier to performance

- T(access) = T(seek) + T(latency)
- T(seek) = move head to correct track
- T(latency) = rotate to correct sector

- T(seek) = 10 milli-sec
- T(latency) = 4.2 milli-sec

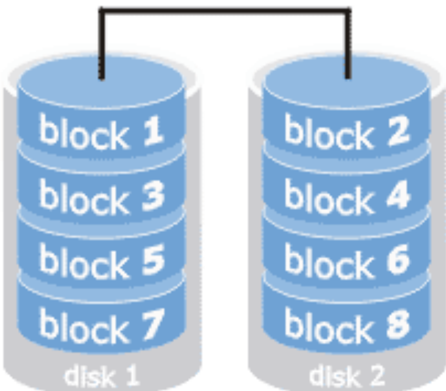
- T(access) = 14 milli-sec!!



~100 Million flops in the time it takes to access disk

Image from Brenton Blawat

Clearly a single magnetic hard drive can not support a supercomputer, so we put many of them together



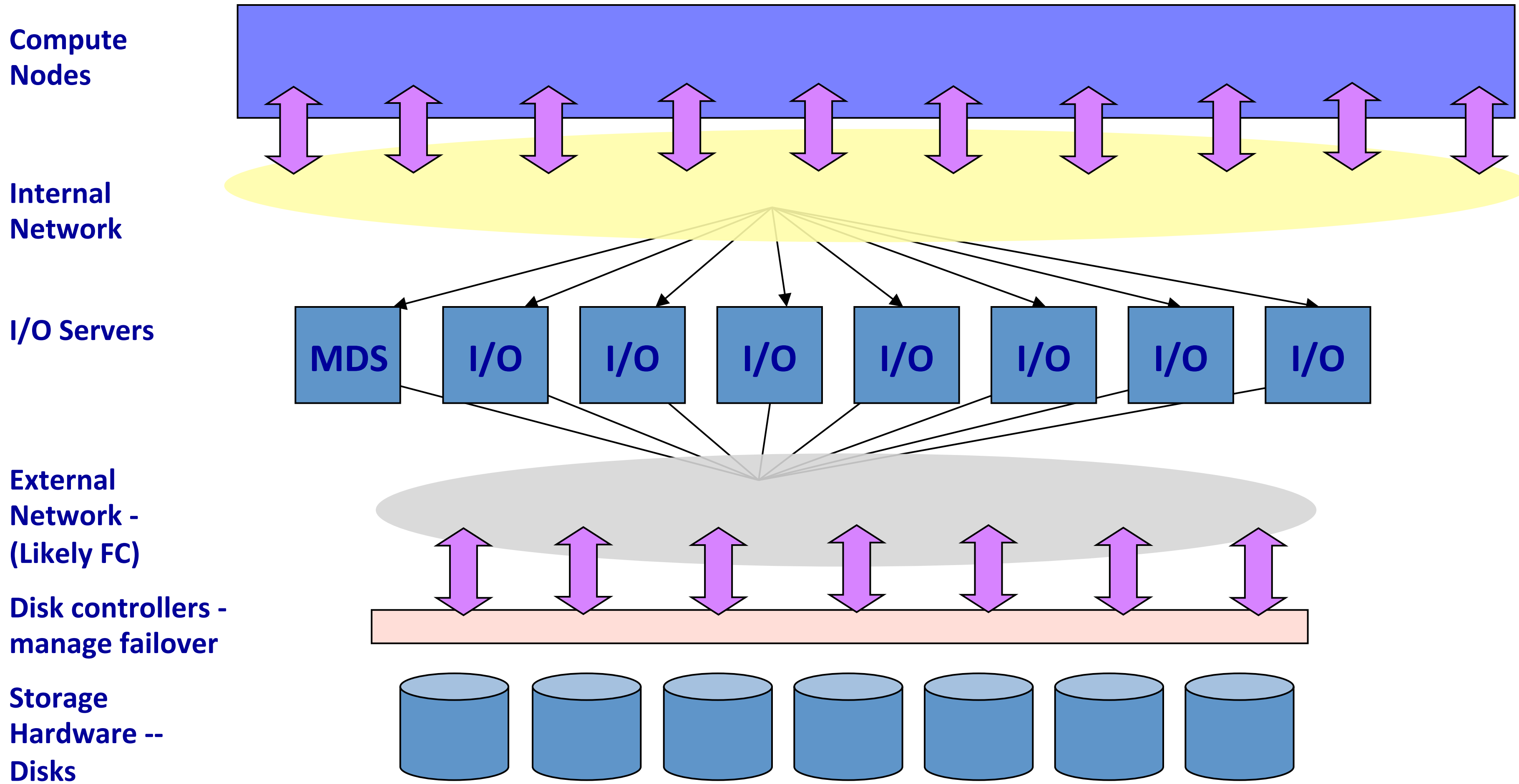
Disks are added in parallel in a format called "RAID"

Redundant Array of Independent Disks

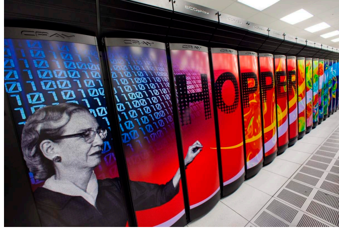
Flash or NVRAM could provide a better balance between bandwidth and storage that could benefit HPC applications

Questions
• What are the use cases for NVRAM in a supercomputer? • How much of the NERSC workload could benefit from NVRAM in an HPC system? • What are the cost/benefit trade-offs in terms of disk capacity, I/O bandwidth and compute? • At what level should NVRAM sit? (compute node? I/O node? Disk?) • What software development would be needed to allow productive use of NVRAM on NERSC-8?

A high performing parallel file system efficiently manages concurrent file access and scales to support huge HPC systems



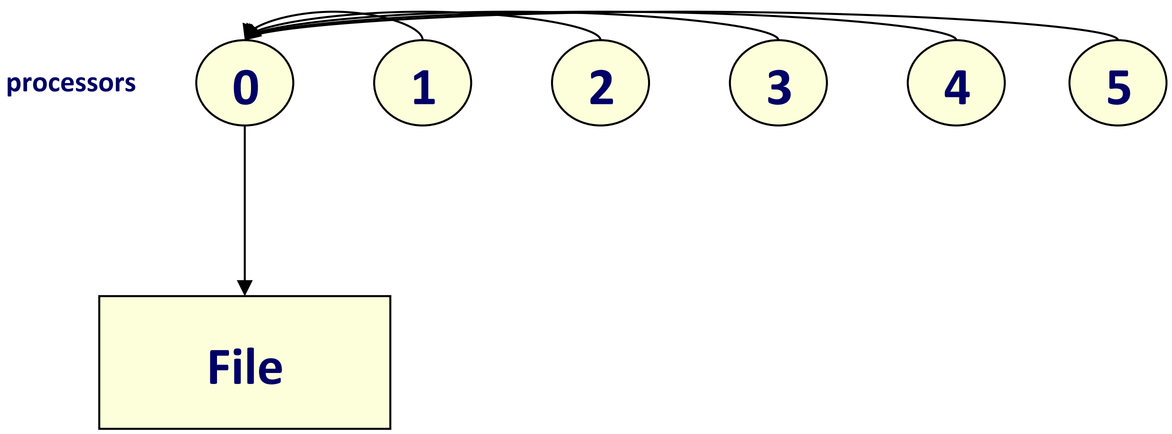
What's the best file system for your application to use on Hopper?



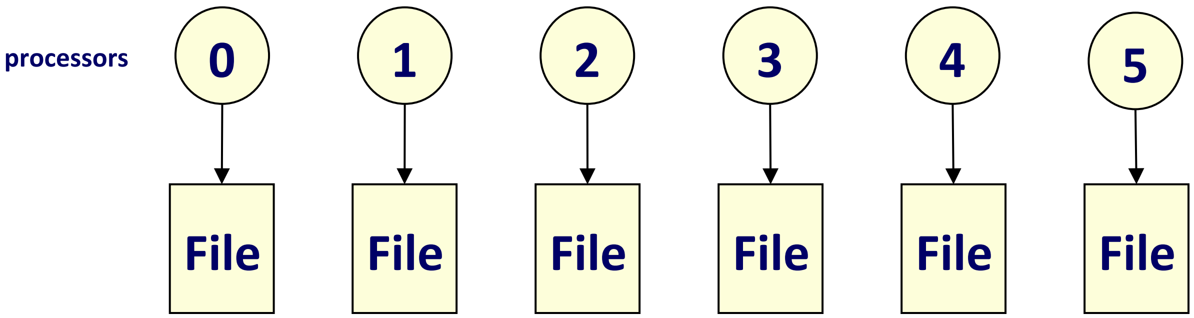
	PEAK	PURPOSE	PROS	CONS
$HOME	Low	Store application code, compile files	Backed up, not purged	Low performing; Low quota
$SCRATCH/ $SCRATCH2	35 GB/sec	Large temporary files, checkpoints	Highest performing	Data not available on other NERSC systems Purged
$PROJECT	15GB/sec	For groups needing shared data access	Data available on all NERSC systems	Shared file performance
$GSCRATCH	12 GB/sec	Alternative scratch space	Data available on almost all NERSC systems	Shared file performance Purged

Serial I/O and file-per-processor I/O are used by many NERSC users

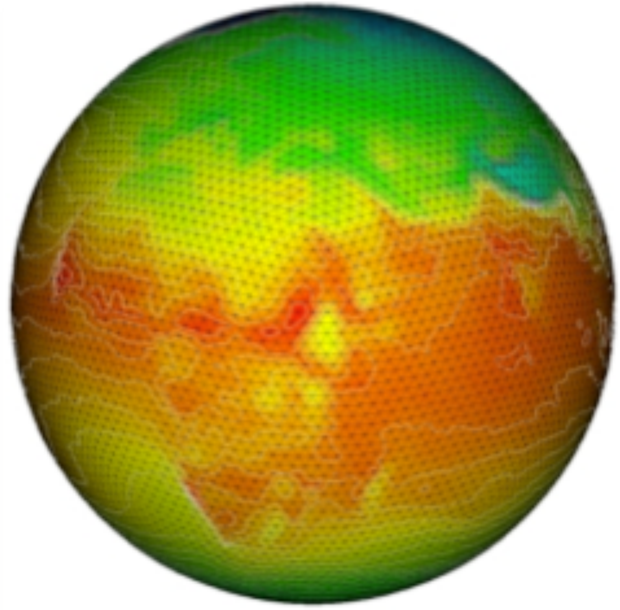
Serial I/O

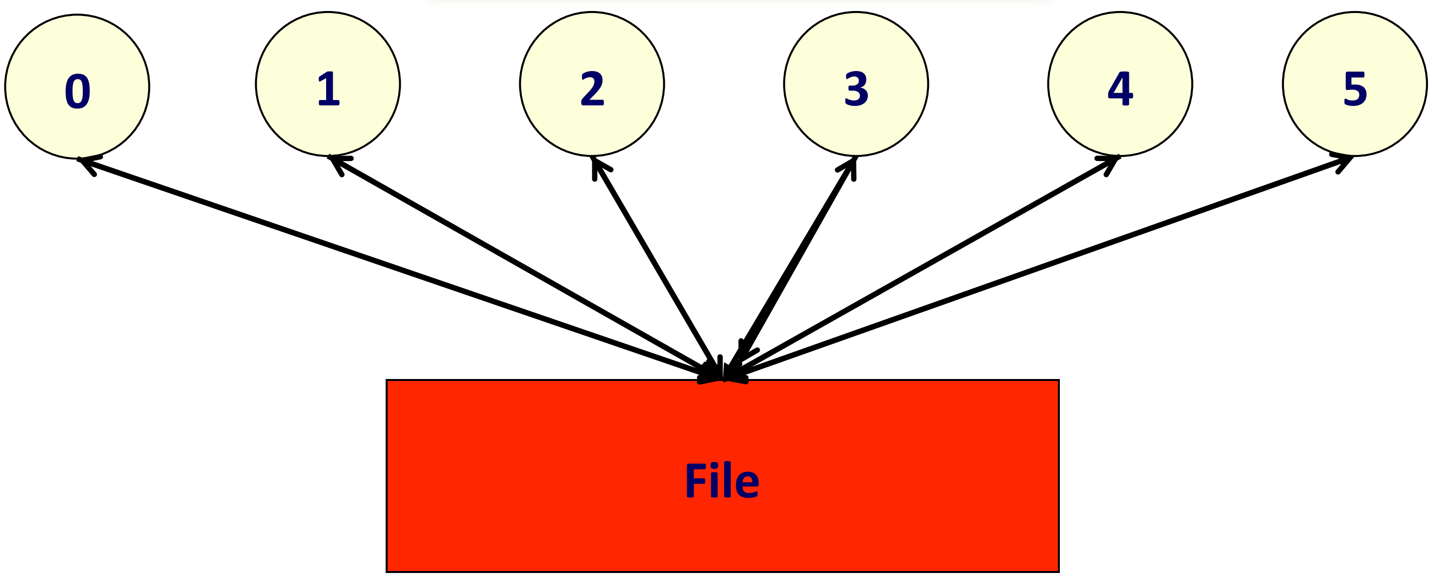


File per Processor I/O

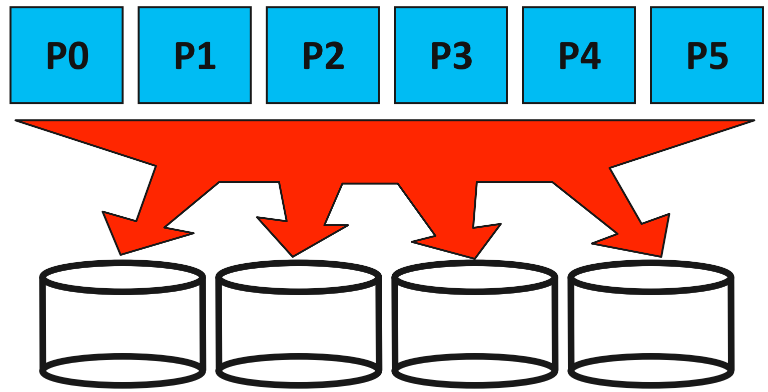


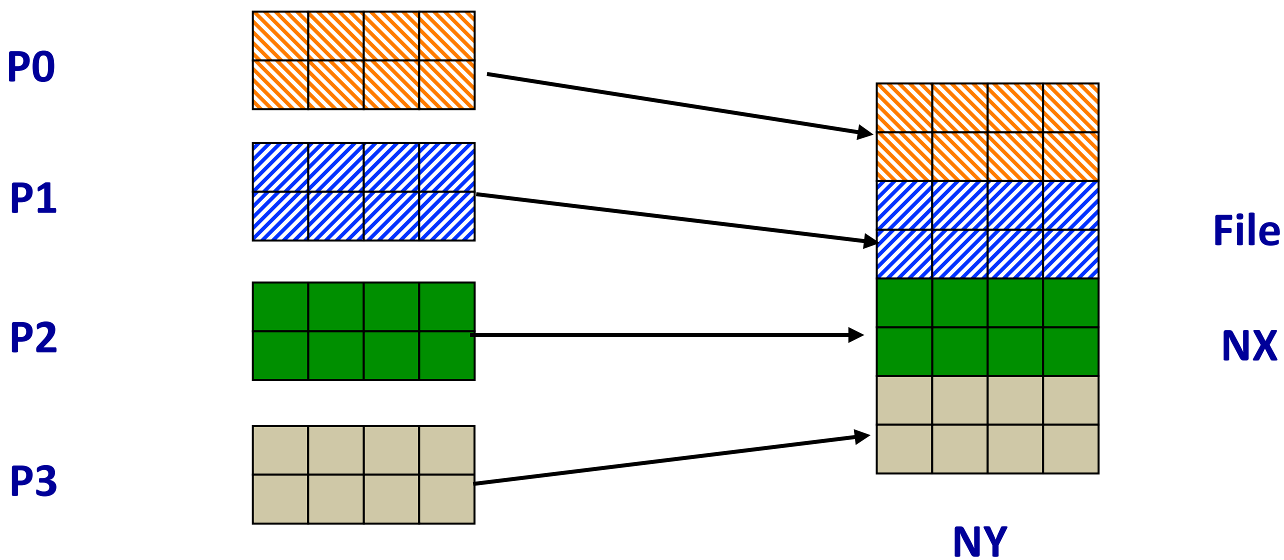
Shared file IO leads to better data management and more natural data layout for scientific applications





MPI/IO and high level parallel I/O libraries like HDF5 allow users to write shared files with a simple interface



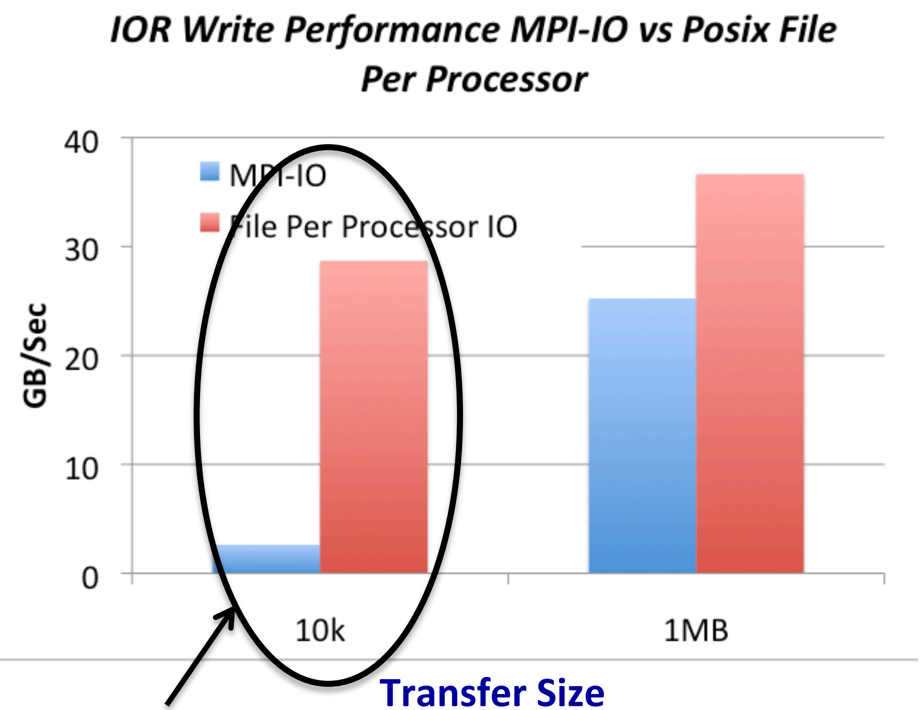


But how does shared file I/O perform?

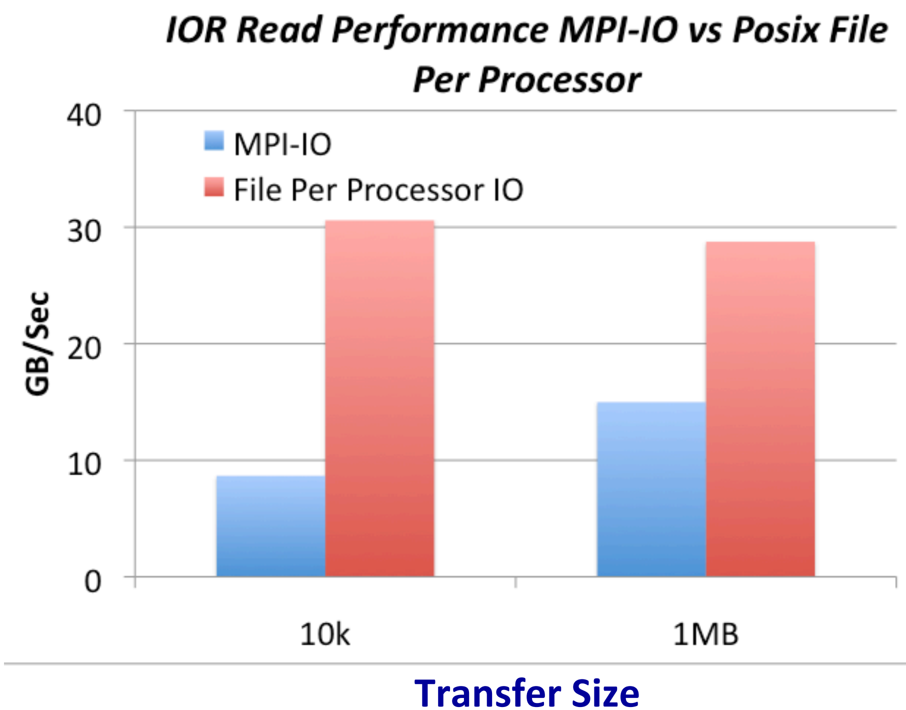
This talk doesn't give details on using MPI-IO or HDF5. See online tutorials.

Shared file I/O depends on the file system, MPI-IO layer and the data access pattern

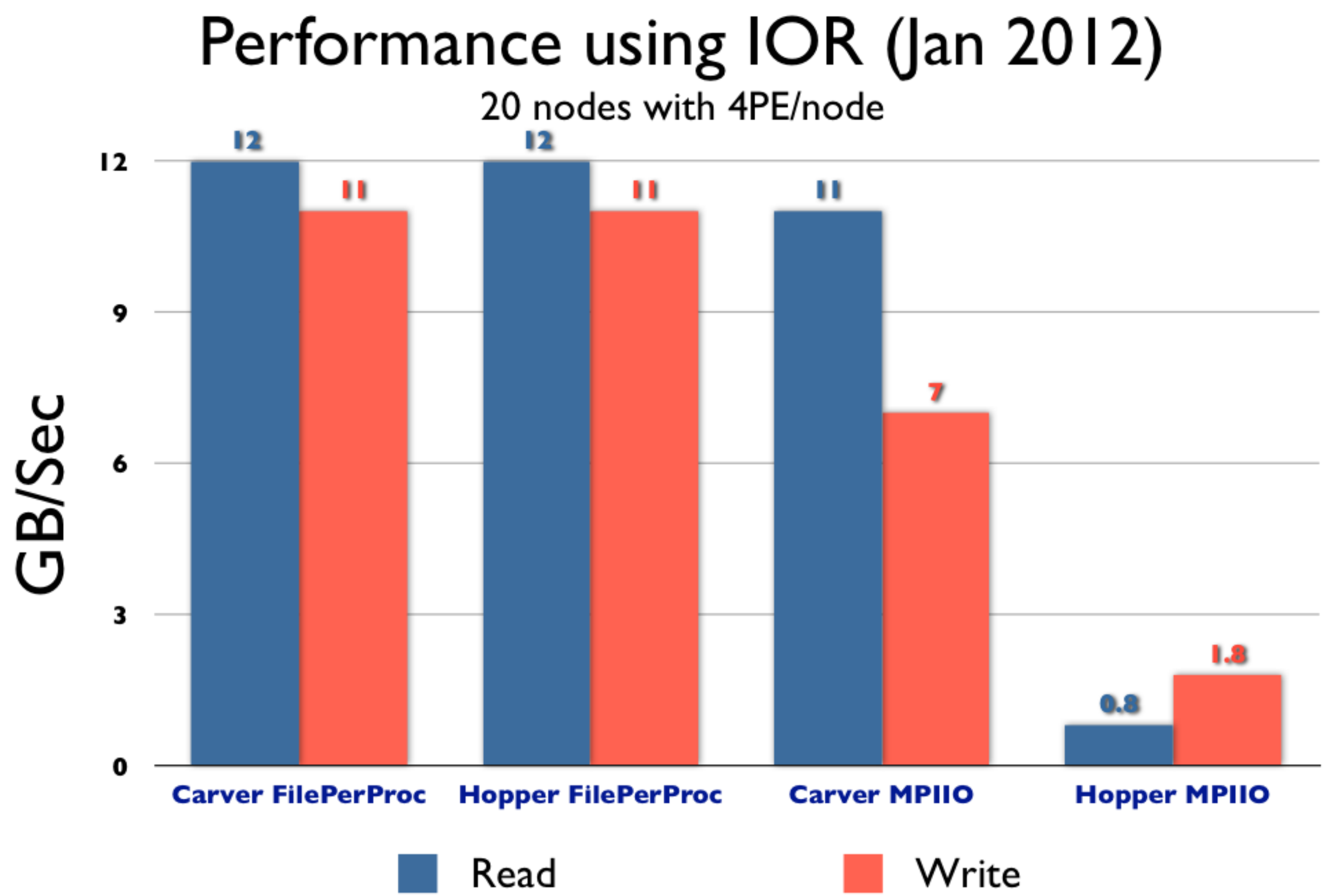
IO Test using IOR benchmark on 576 cores on Hopper with Lustre file system



Hard sell to users

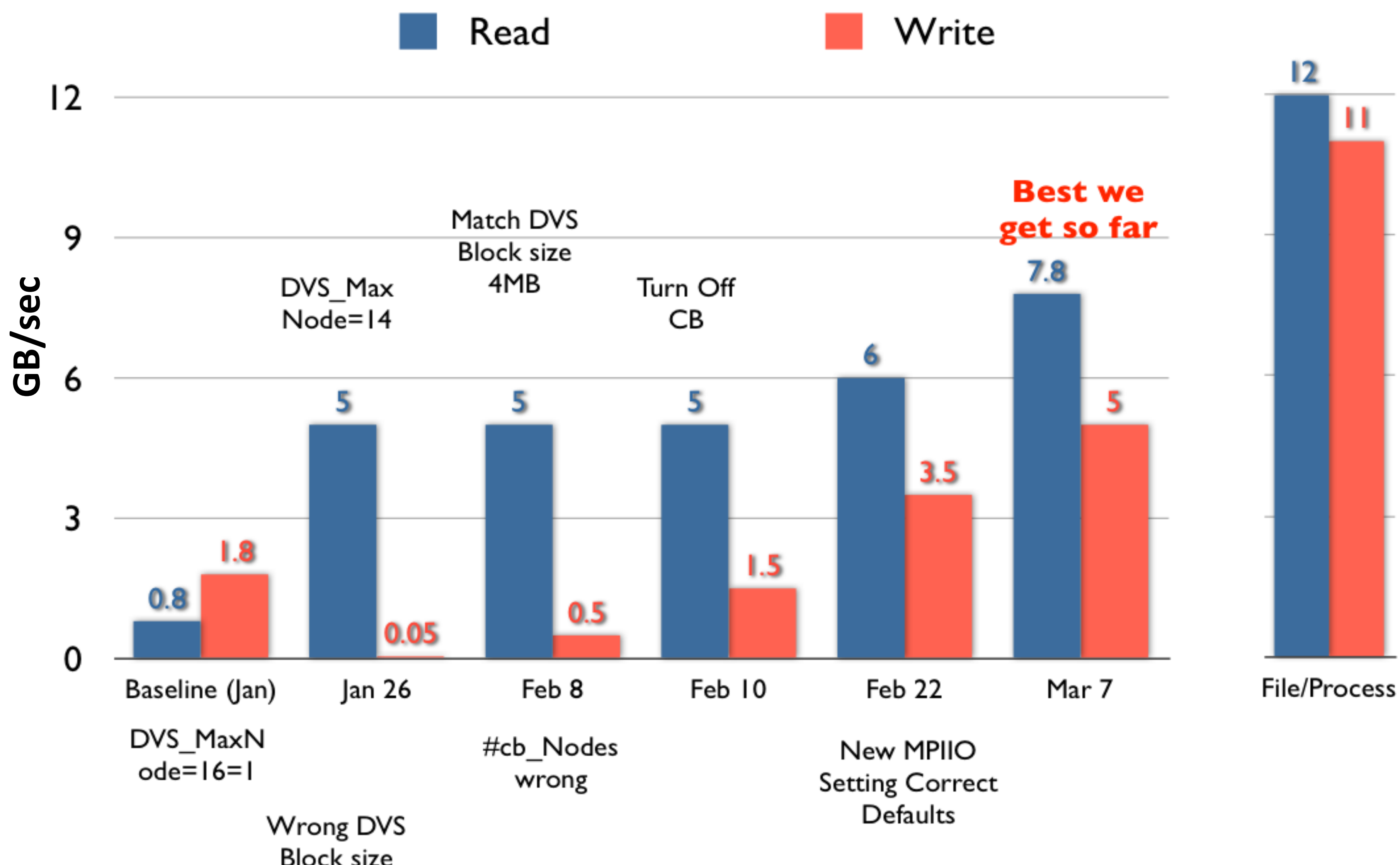


We initiated a collaboration with Cray to improve MPI-IO performance on Hopper to /project and /gscratch

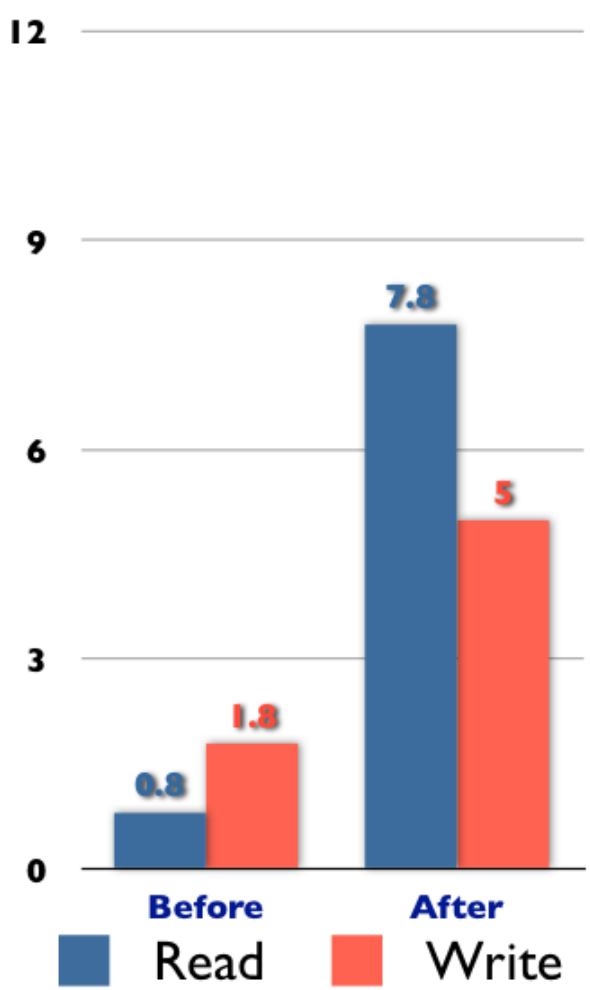


Consider combining smaller write requests into larger ones and limiting the number of writers per node

Progress over time …

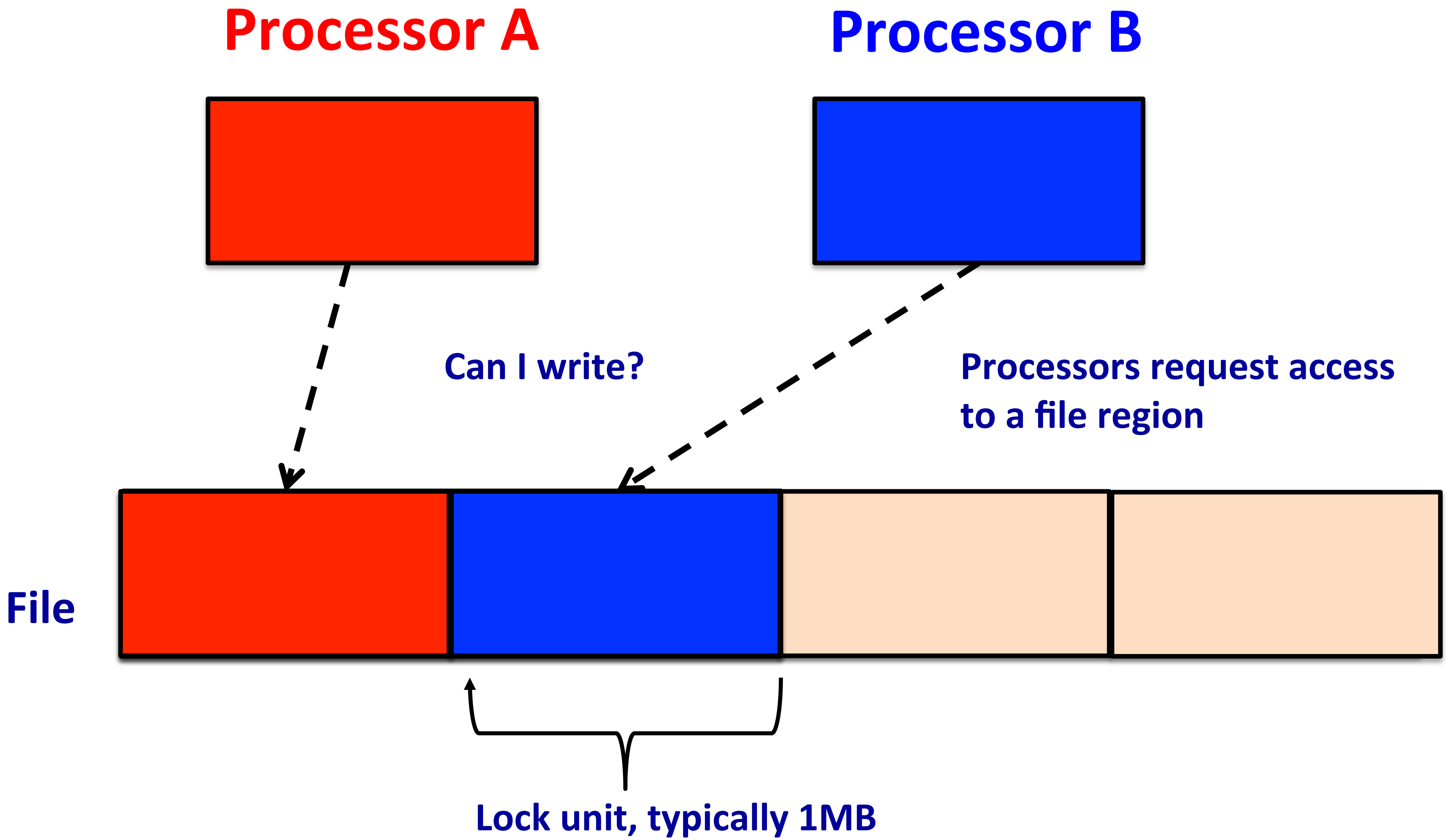


Conclusion

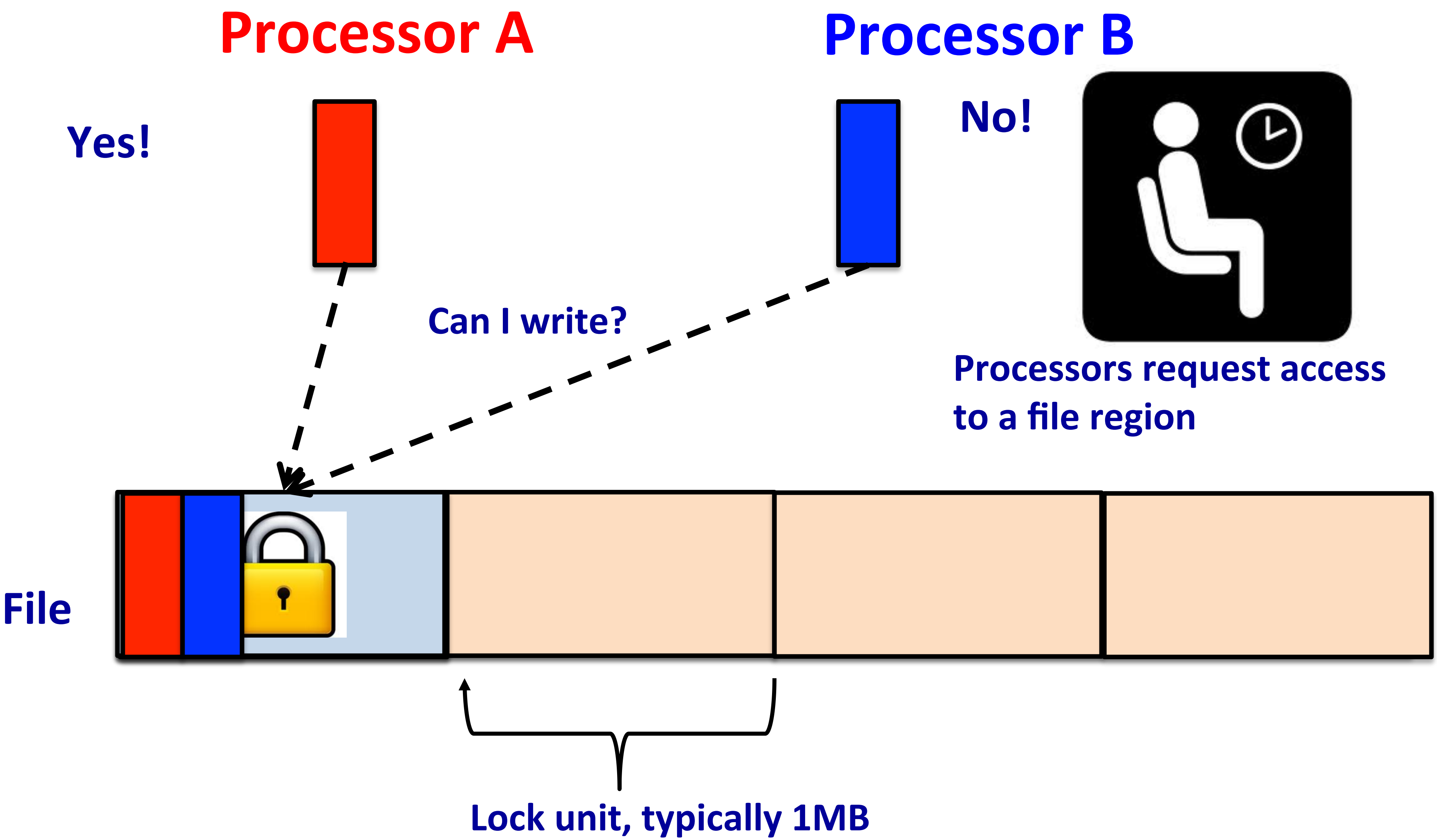


- 10 X performance improvement on read, 3X write, after changing both run setup and MPIIO library
- Setting DEFAULT values for users so that they can get best performance (in most cases) automatically

Files are broken up into lock units, which the file system uses to manage concurrent access to a region in a file



Files are broken up into lock units, which the file system uses to manage concurrent access to a region in a file

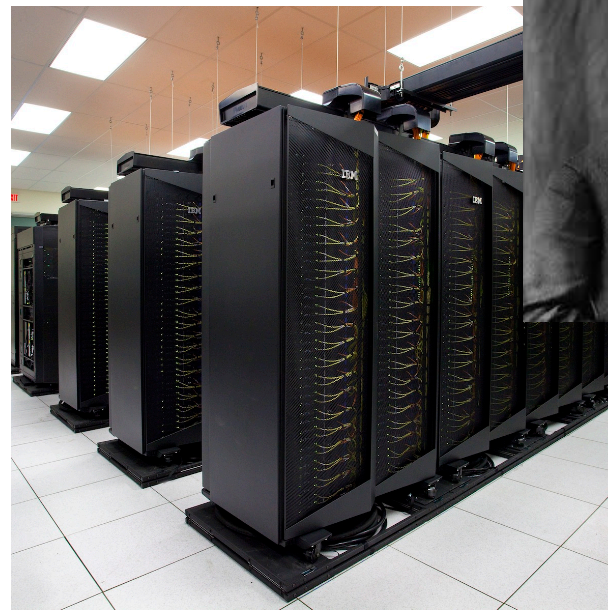


Users can (and should) adjust striping parameters on Lustre file systems

User controlled striping on Lustre filesystems, $SCRATCH, $SCRATCH2

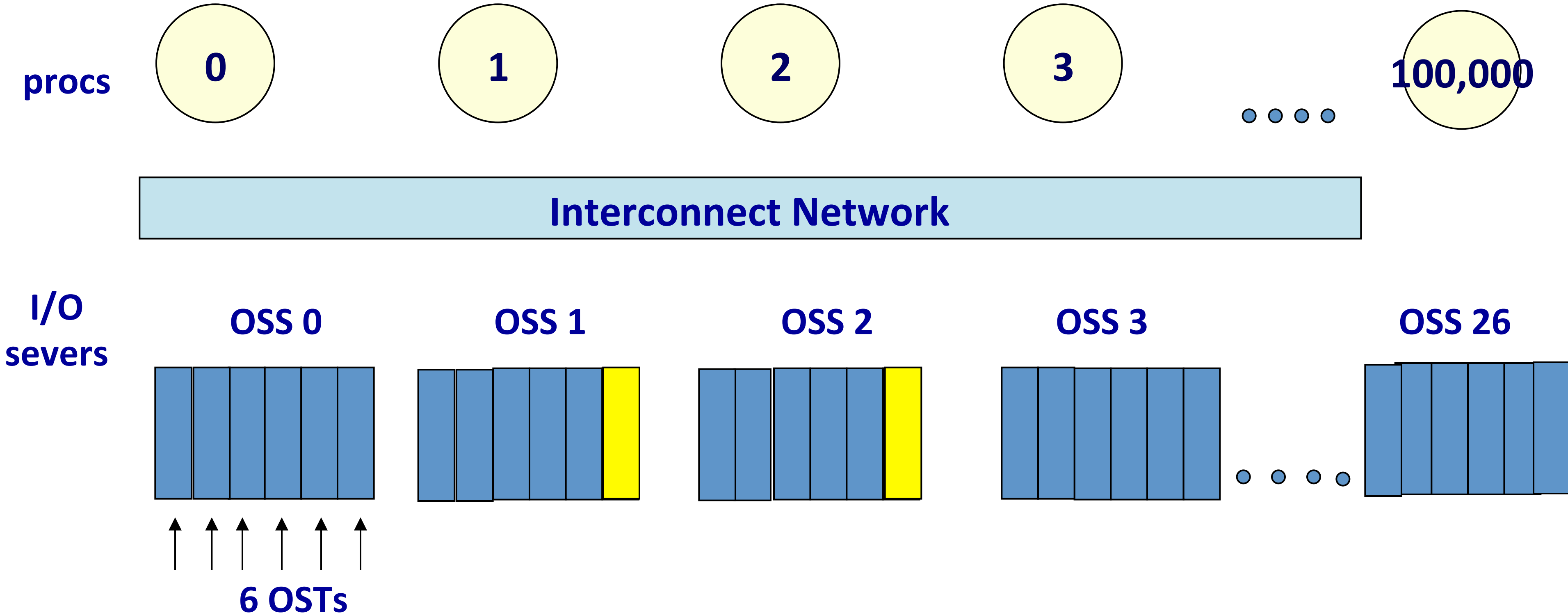
X NO User controlled striping on GPFS filesystems, $GSCRATCH, $HOME, $PROJECT

Carver



X No user controlled striping, (ALL GPFS filesystems)

There are three parameters that characterize striping on a Lustre file system, the stripe count, stripe size and the offset



- **Stripe count: Number of OSTs file is spilt across: Default 2**
- **Stripe size: Number of bytes to write on each OST before cycling to the next OST: Default 1MB**
- **OST offset: Indicates starting OST: Default round robin**

Striping can be set at the file or directory level.

When striping set on a directory: all files created in that directory will inherit striping set on the directory

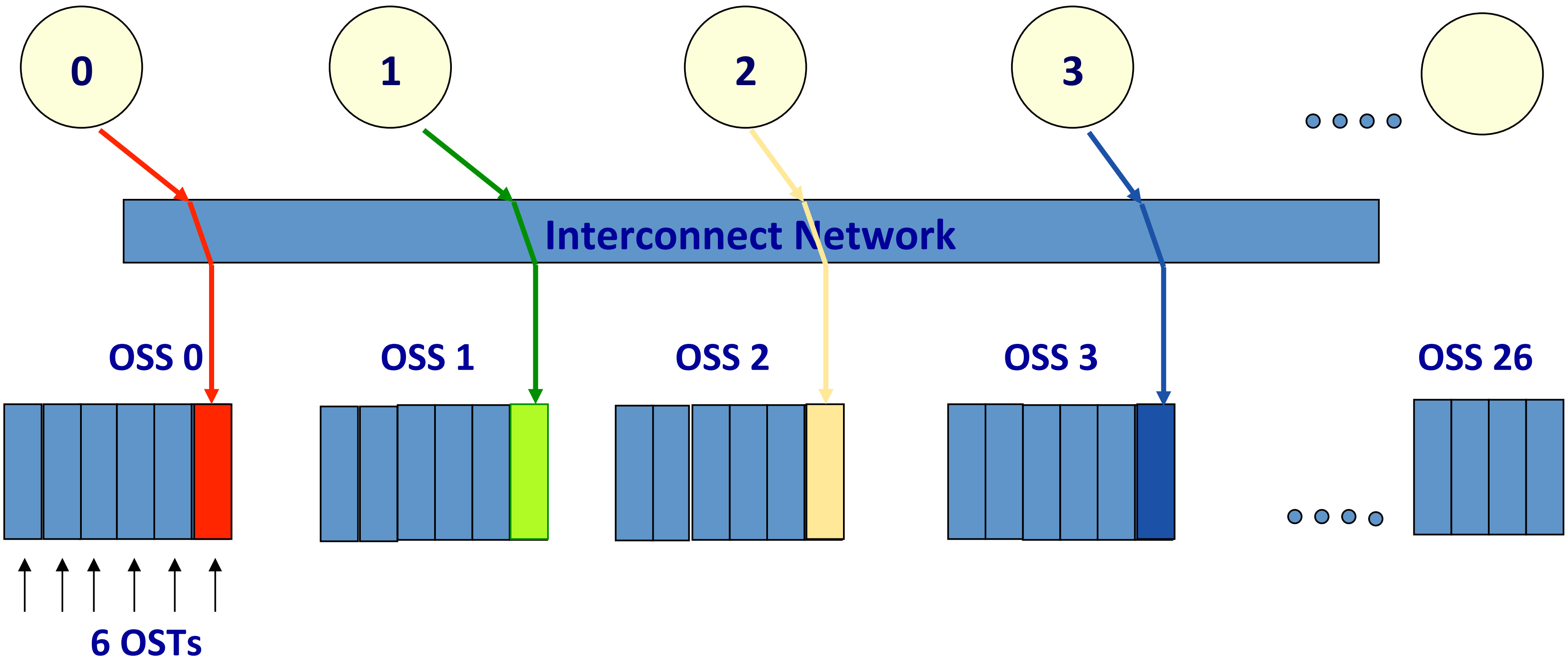
lfs setstripe <directory|file> -c stripe-count

Stripe count - # of OSTs file is split across

Example: change stripe count to 10

lfs setstripe mydirectory -c 10

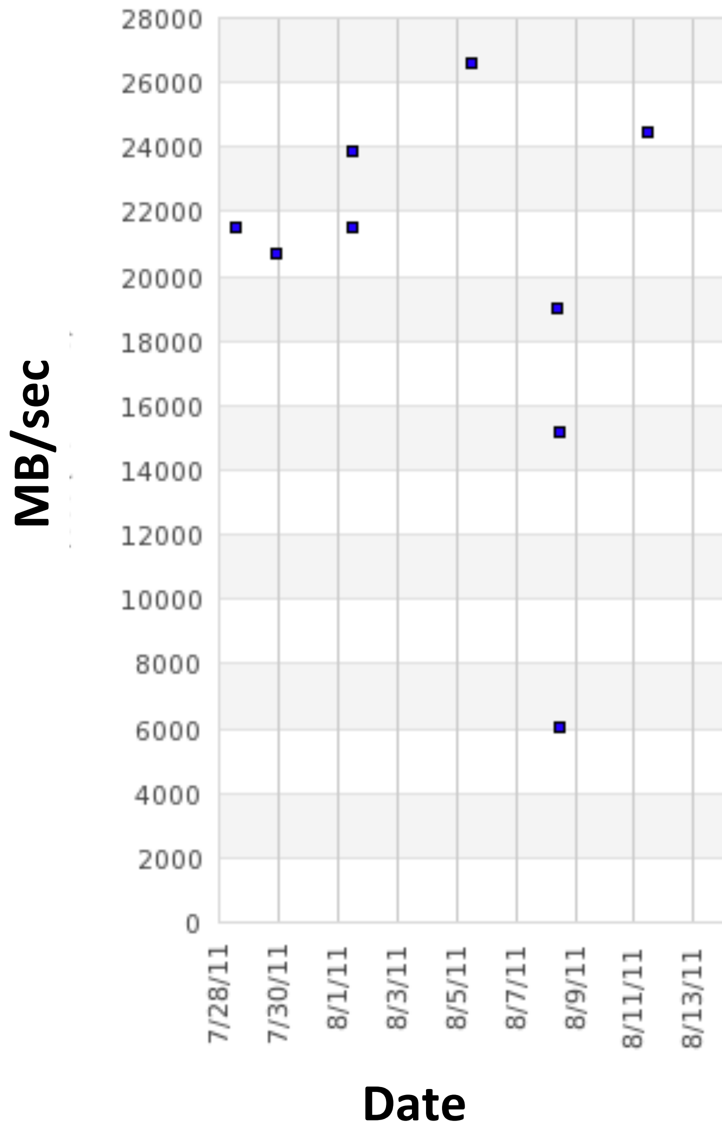
For one-file-per-processor workloads set the stripe count to 1 for maximum bandwidth and minimal contention



Striping guidelines for Hopper

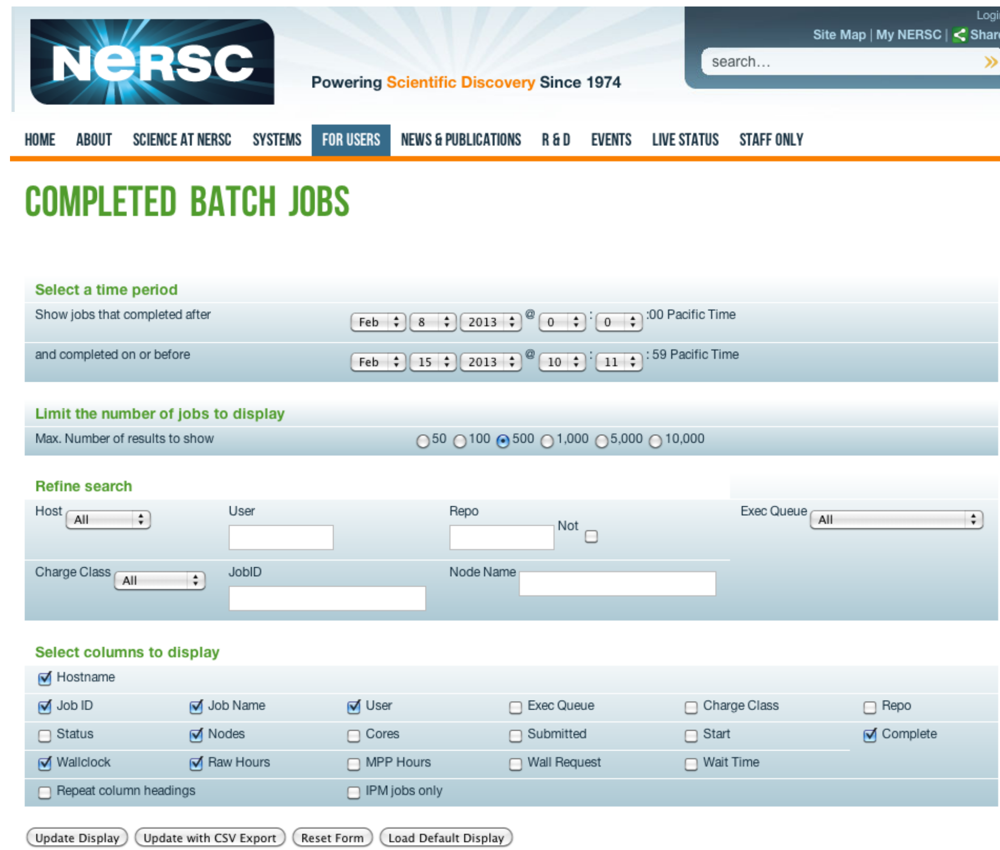
- One File-Per-Processor I/O or shared files < 10 GB
  - Keep default or stripe count 1
- Medium shared files: 10GB – 100sGB
  - Set stripe count ~4-20
- Large shared files > 1TB
  - Set stripe count to 20 or higher, maybe all OSTs?
- You' ll have to experiment a little

I/O resources are shared between all users on the system so you may often see some variability in performance



You can learn more about your application's I/O pattern using an I/O profiling tool like Darshan

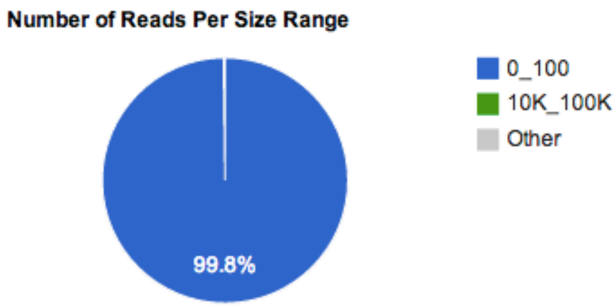
- How to use Darshan
- Compile your application as you normally would
- Run your application as you normally would
- Look at NERSC's 'Completed Jobs' page for I/O statistics

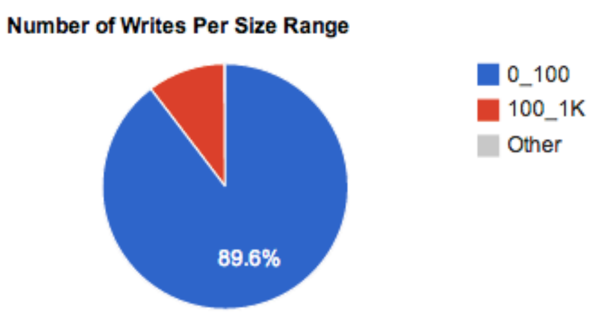


On the completed jobs webpage you will see output like the below image

IO Summary from Darshan

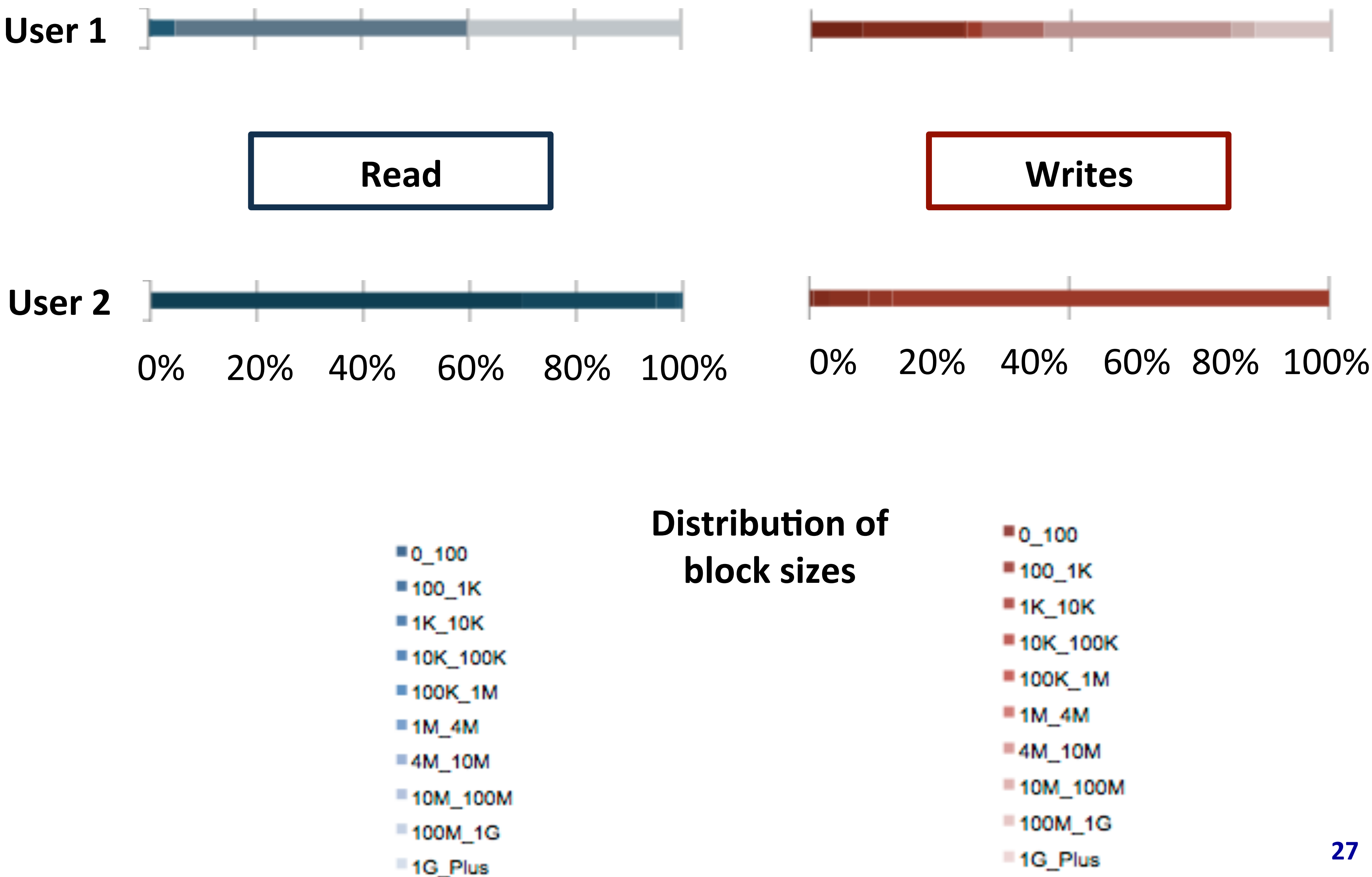
Exec. Runtime	MB Read	MB Written	Read Time (s)	Write Time (s)	Read Rate (MB/s)	Write Rate (MB/s)
12-13 04:03:39 - 12-13 17:21:39	16203033.3	314607.21	1.29026e+06	510309	12.56	0.62



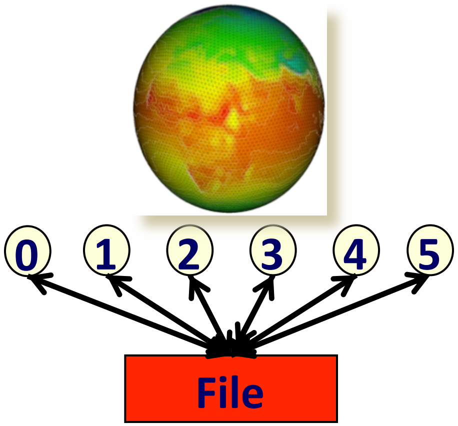


- When an application is compiled against the Darshan library, I/O calls are intercepted and recorded in a central logfile
- Examine distribution of write sizes
- Measure I/O rates

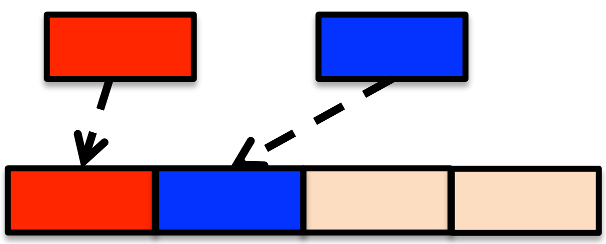
Comparing the I/O patterns of two users



In summary, think about the big picture in terms of your simulation, output and visualization needs



Determine your I/O priorities:
Performance? Data Portability? Ease of analysis?



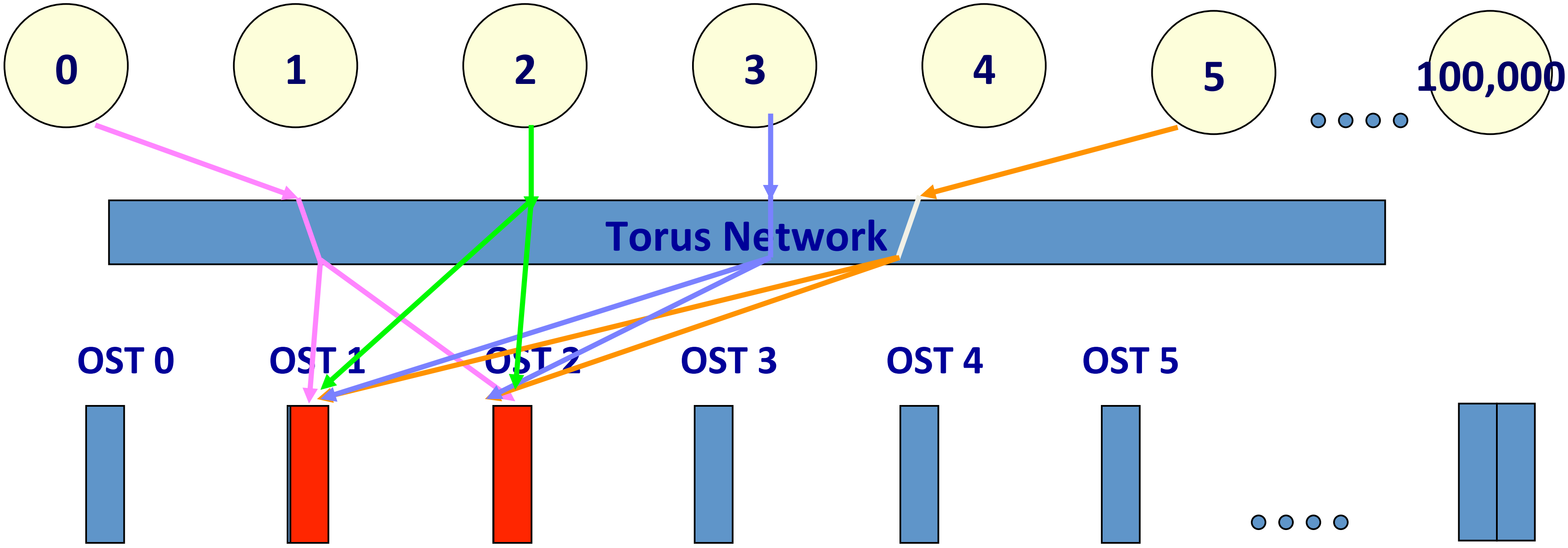
Write large blocks of I/O

Understand the type of file system you are using and make local modifications

THE END

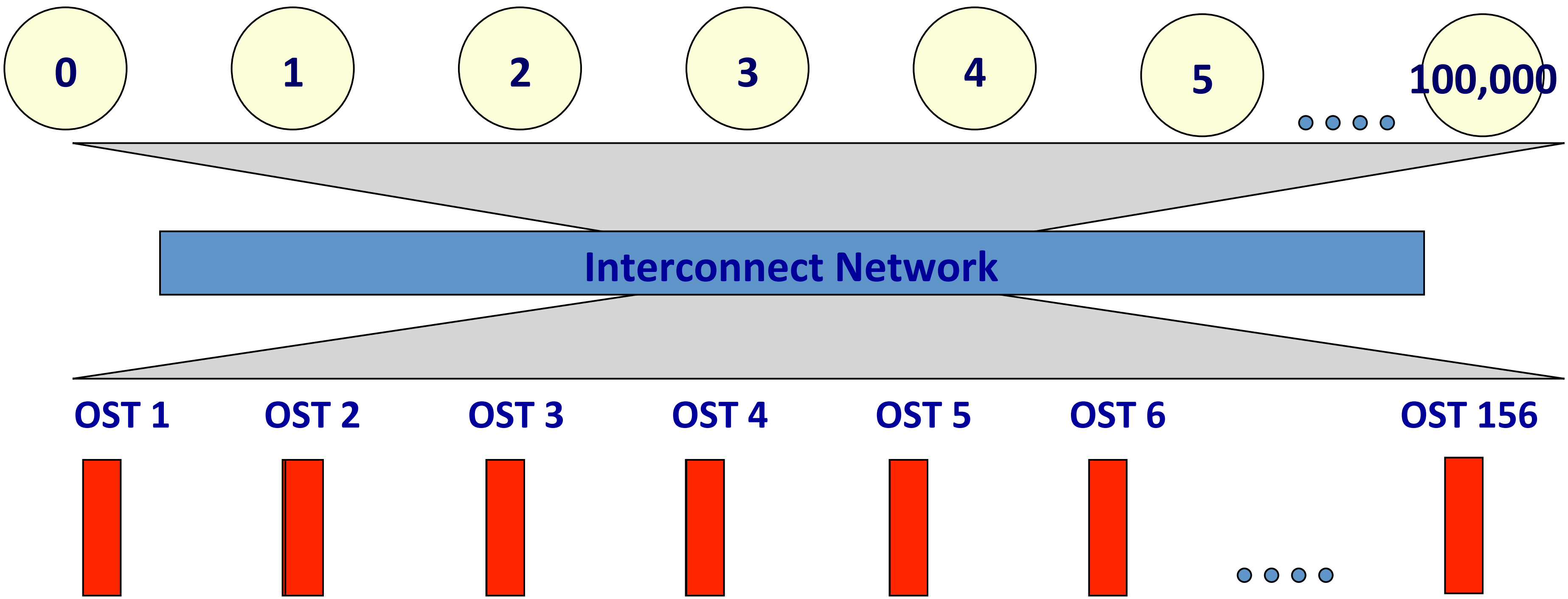
A simulation writing a shared file, with a stripe count of 2, will achieve a maximum write performance of ~800 MB/sec

No matter how many processors are used in the simulation



For large shared files, increase the stripe count

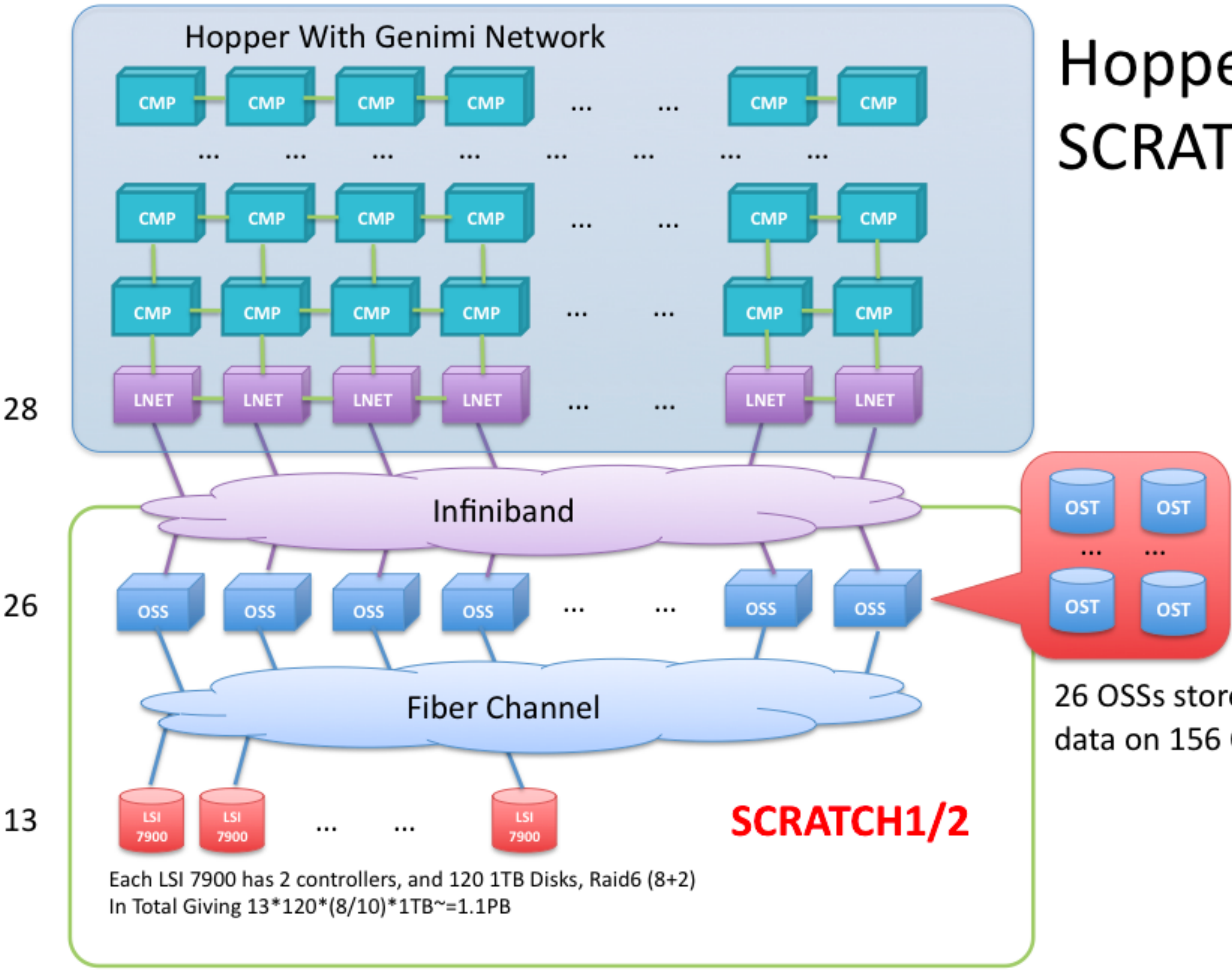
Striping over all OSTS increases bandwidth available to application



The next table gives guidelines on setting the stripe count

Lustre file system on Hopper

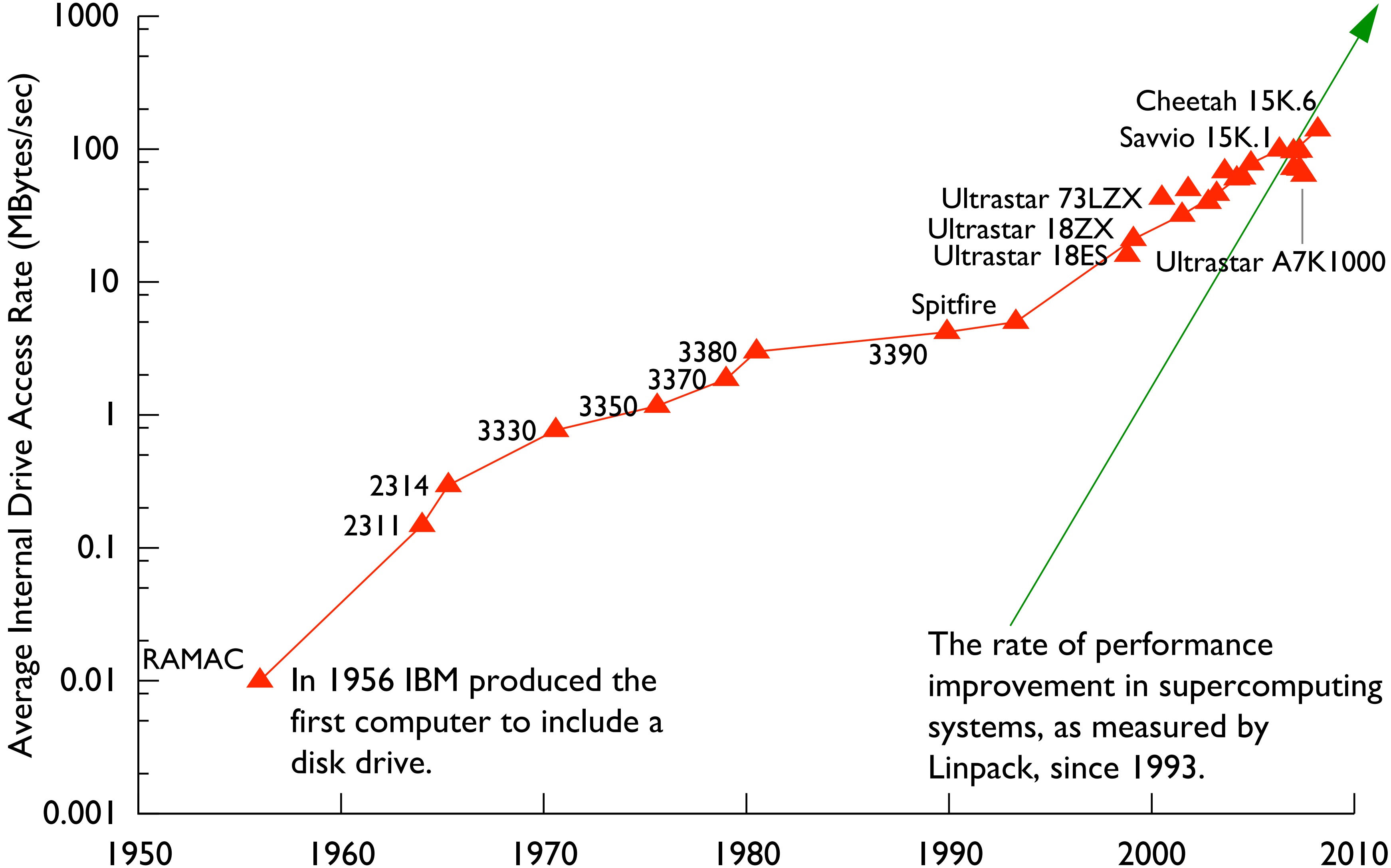
Hopper SCRATCH



26 OSSs store data on 156 OSTs

Note: SCRATCH1 and SCRATCH2 have identical configurations.

Disk rates are improving, but not nearly as fast as compute performance



Source: R. Freitas of IBM Almaden Research Center

A file system is a software layer between the operating system and storage device

Files

Directories

Access permissions

memory

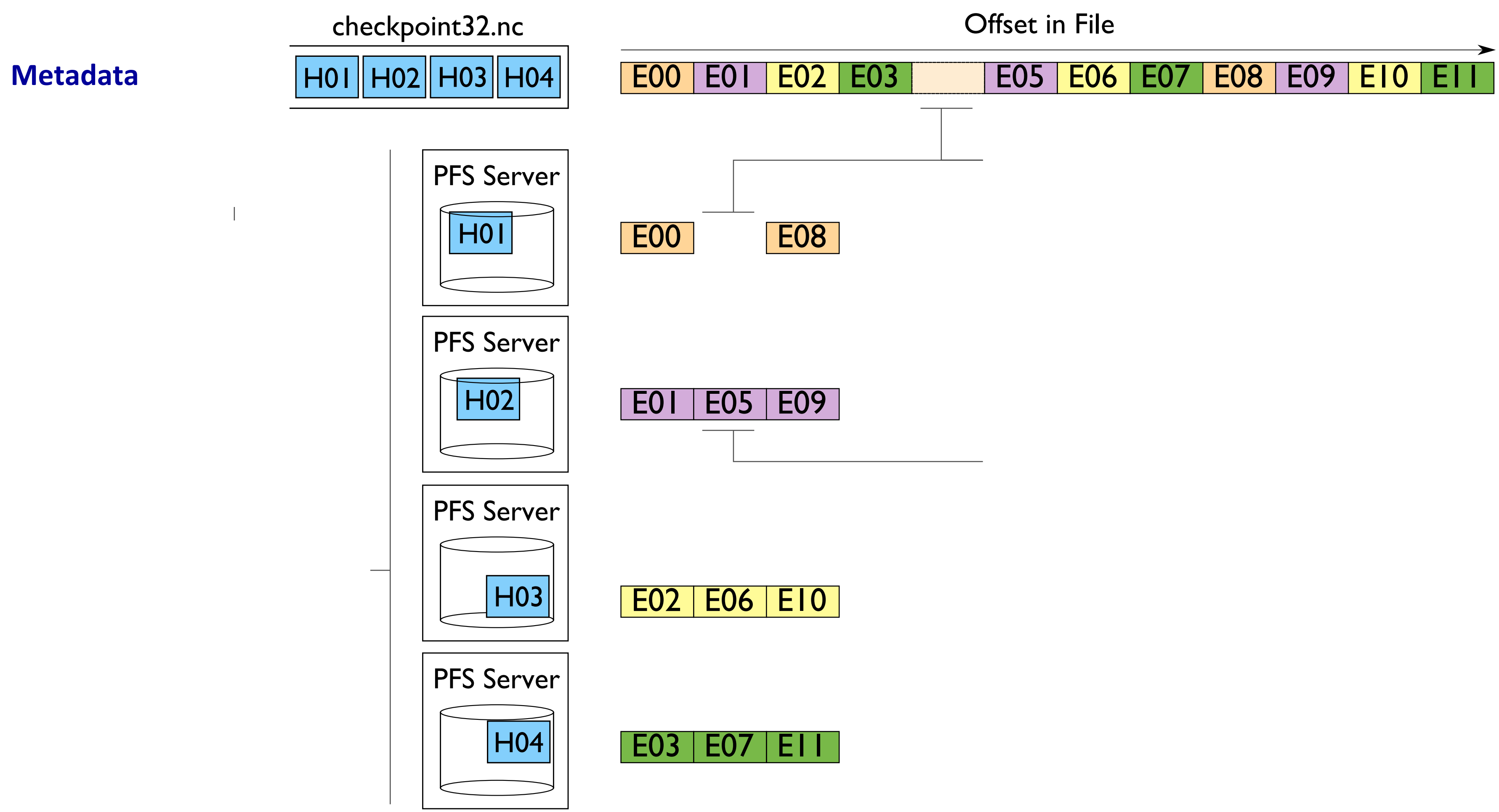
1010010101110100

File System

Storage device

Source: J.M. May "Parallel IO for High Performance Computing, techCrunch, howstuffworks.com

File striping is a technique used to increase I/O performance by simultaneously writing/reading data from multiple disks



Slide Rob Ross, Rob Latham at ANL