# Data Management

*Parallel Filesystems*

Dr David Henty
HPC Training and Support
d.henty@epcc.ed.ac.uk
+44 131 650 5960

# Overview

- Lecture will cover
  - Why is IO difficult
  - Why is parallel IO even worse
  - Lustre
  - GPFS
  - Performance on ARCHER (Lustre)

# Why is IO hard?

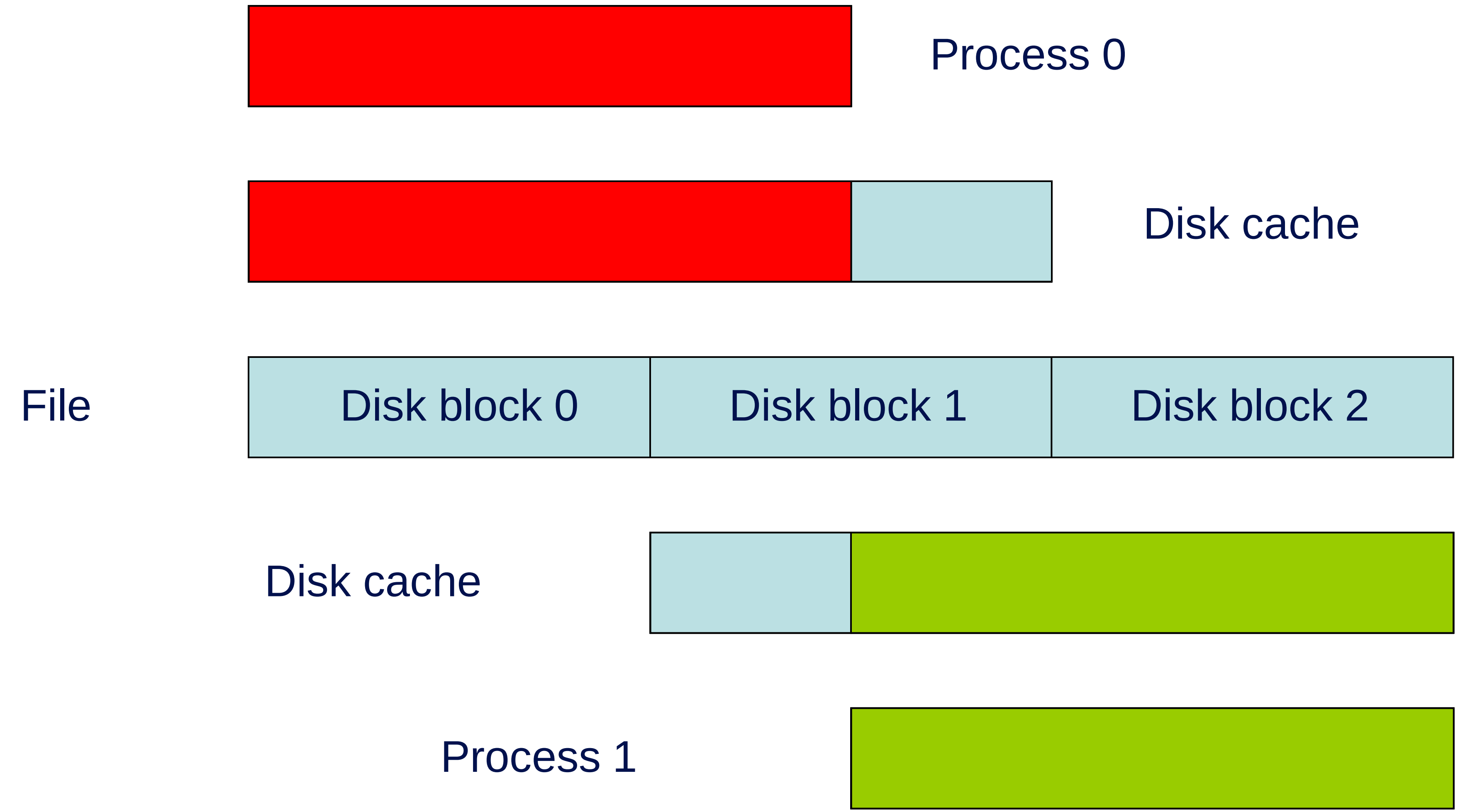
- Breaks out of the nice process/memory model
  - data in memory has to physically appear on an external device
- Files are very restrictive
  - linear access probably implies remapping of program data
  - just a string of bytes with no memory of their meaning

- Many, many system-specific options to IO calls
- Different formats
  - text, binary, big/little endian, Fortran unformatted, ...
- Disk systems are very complicated
  - RAID disks, many layers of caching on disk, in memory, ...
- IO is the HPC equivalent of printing!

# Why is Parallel IO Harder?

- Cannot have multiple processes writing a single file
  - Unix generally cannot cope with this
  - data cached in units of disk blocks (eg 4K) and is *not coherent*
  - not even sufficient to have processes writing to distinct parts of file
- Even reading can be difficult
  - 1024 processes opening a file can overload the filesystem (fs)
- Data is distributed across different processes
  - processes do not in general own contiguous chunks of the file
  - cannot easily do linear writes
  - local data may have halos to be stripped off

# Simultaneous Access to Files



# Parallel File Systems

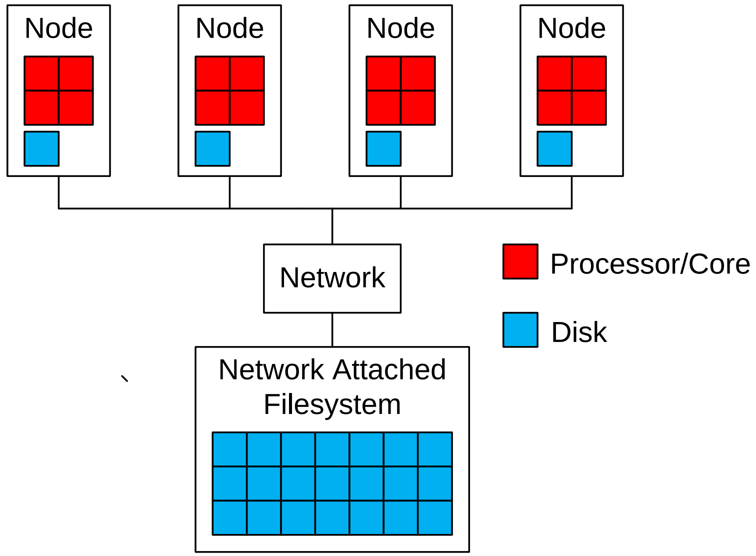
- Parallel computer
  - constructed of many processors
  - each processor not particularly fast
  - performance comes from using many processors at once
  - requires distribution of data and calculation across processors
- Parallel file systems
  - constructed from many standard disk
  - performance comes from reading / writing to many disks
  - requires many *clients* to read / write to different disks at once
  - data from a single file must be *striped* across many disks
- Must appear as a single file system to user
  - typically have a single *MedaData* Server (MDS)
  - can become a bottleneck for performance

# Performance

| Interface | Throughput Bandwidth (MB/s) |
| --- | --- |
| PATA (IDE) | 133 |
| SATA | 600 |
| Serial Attached SCSI (SAS) | 600 |
| Fibre Channel | 2,000 |

# HPC/Parallel Systems

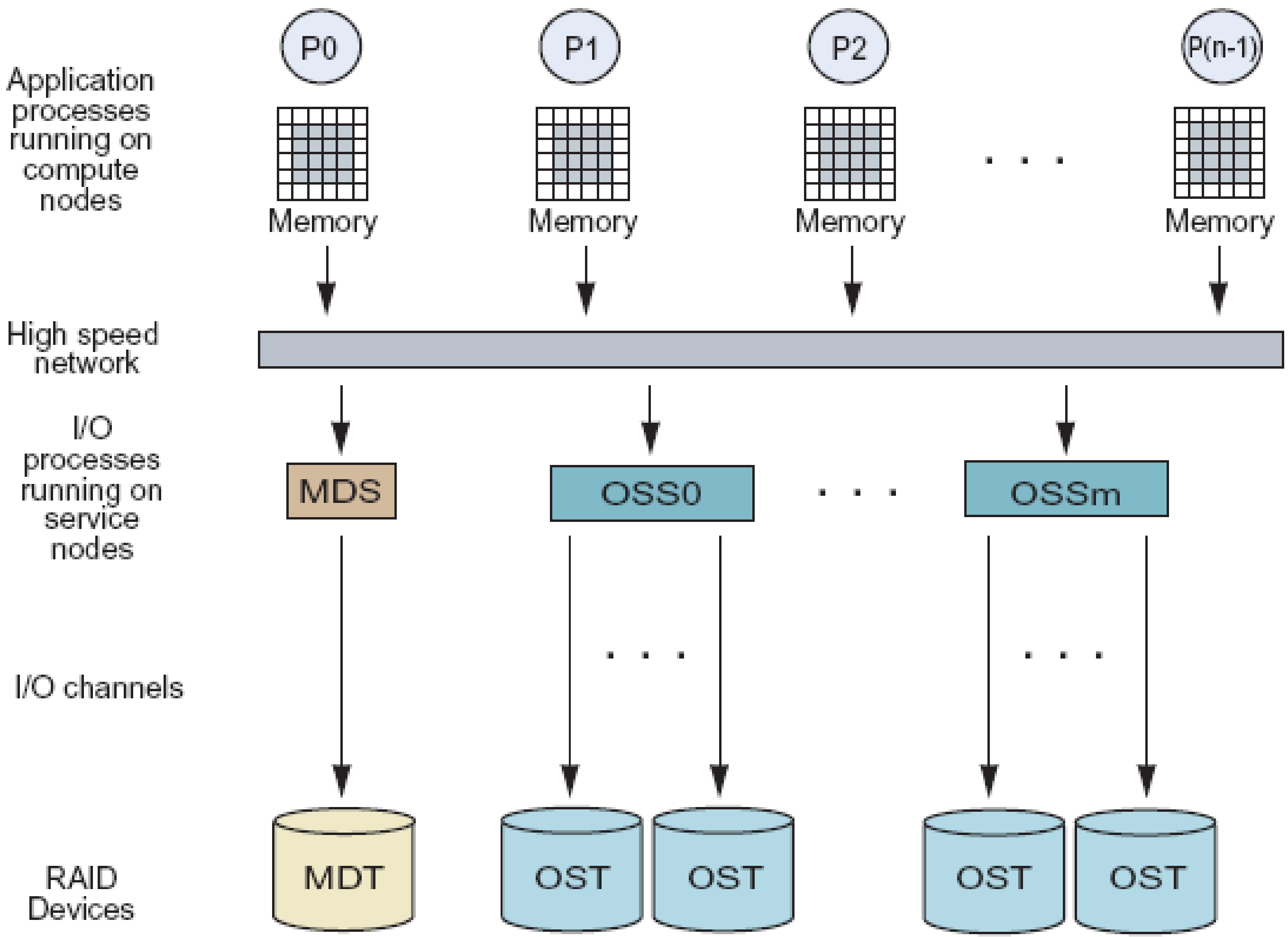
- Basic cluster
  - Individual nodes
  - Network attached filesystem
  - Local scratch disks
- Multiple I/O systems
  - Home and work
  - Optimised for production or for user access
- Many options for optimisations
  - Filesystem servers, caching, etc…



# Parallel File Systems

- Allow multiple IO processes to access same file
  - increases bandwidth
- Typically optimised for bandwidth
  - not for latency
  - e.g. reading/writing small amounts of data is very inefficient
- Very difficult for general user to configure and use
  - need some kind of higher level abstraction
  - allow user to focus on data layout across user processes
  - don’t want to worry about how file is split across IO servers

# Parallel File Systems: Lustre



# ARCHER’s Cray Sonexion Storage

1

**MMU:** ***Metadata Management Unit***

**Lustre MetaData Server**

- Contains server hardware and storage

2

**SSU:** ***Scalable Storage Unit***


2 x OSSs and 8 x OSTs (Object Storage Targets)

– Contains Storage controller, Lustre server, disk controller and RAID engine

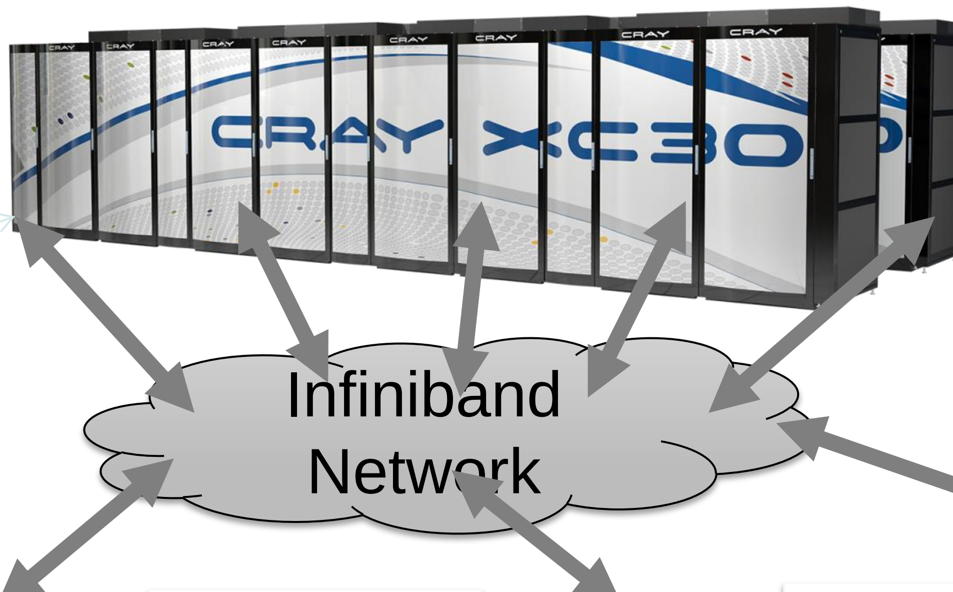
– Each unit is 2 OSSs each with 4 OSTs of 10 (8+2) disks in a RAID6 array

3


**Multiple SSUs are combined to form storage racks**

# ARCHER’s File systems

Connected to the Cray XC30 via LNET router service nodes.



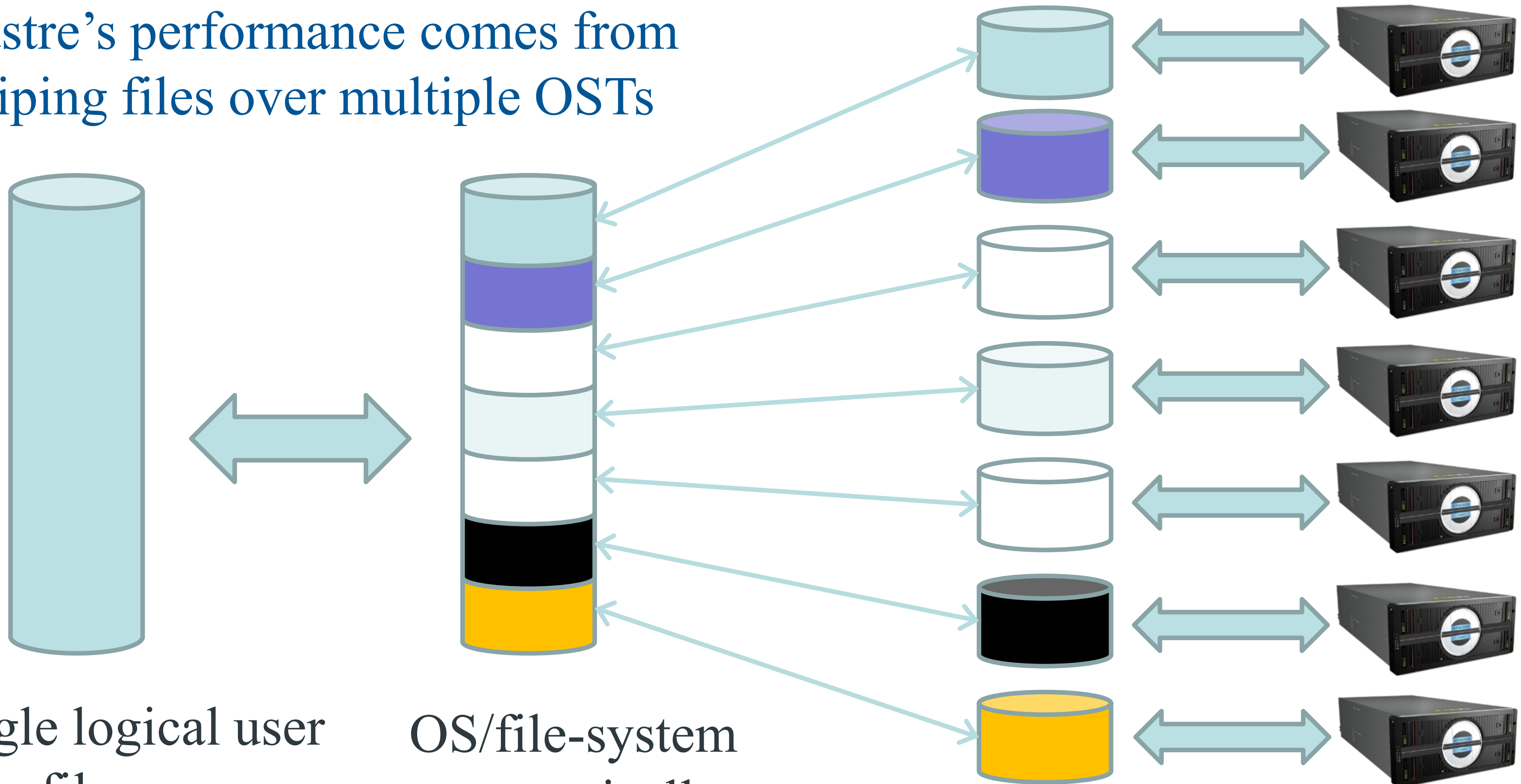

**/fs2**
6 SSUs
12 OSSs
48 OSTs
480 HDDs
4TB per HDD
1.4 PB Total

**/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

# Configuring Lustre

- Main control is the number of OSTs a file is striped across
  - default 4 stripes (i.e. file is stored across 4 OSTs) in 1 Mb chunks
  - under control of user
  - easiest to set this on a per-directory basis

- lfs setstripe –c <stripecount> directory

  - stripecount = 4 is default
  - stripecount = 1 is appropriate for many small files
  - stripecount = -1 sets maximum striping (i.e. around 50 OSTs)
    - appropriate for collective access to a single large file

- Can investigate this in practical exercise

# Lustre on ARCHER

- See white paper on I/O performance on ARCHER:

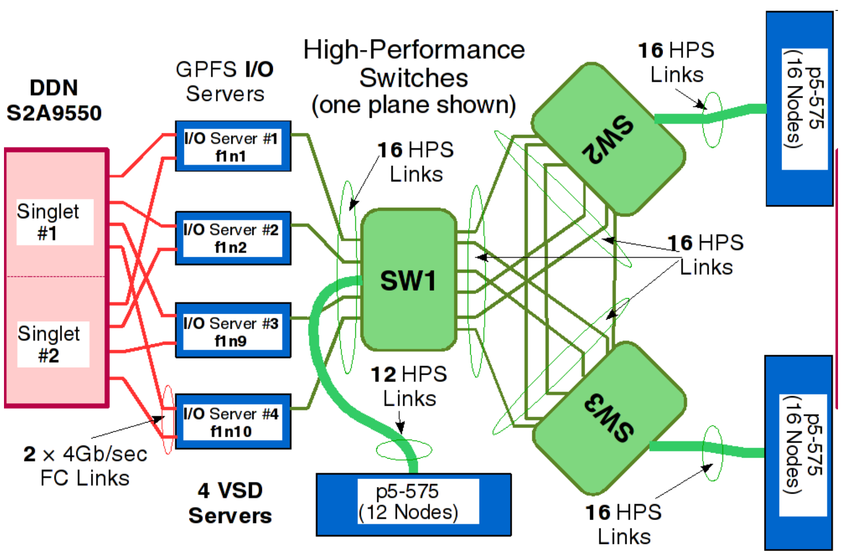
- http://www.archer.ac.uk/documentation/white-papers/parallelIO/ARCHER_wp_parallelIO.pdf

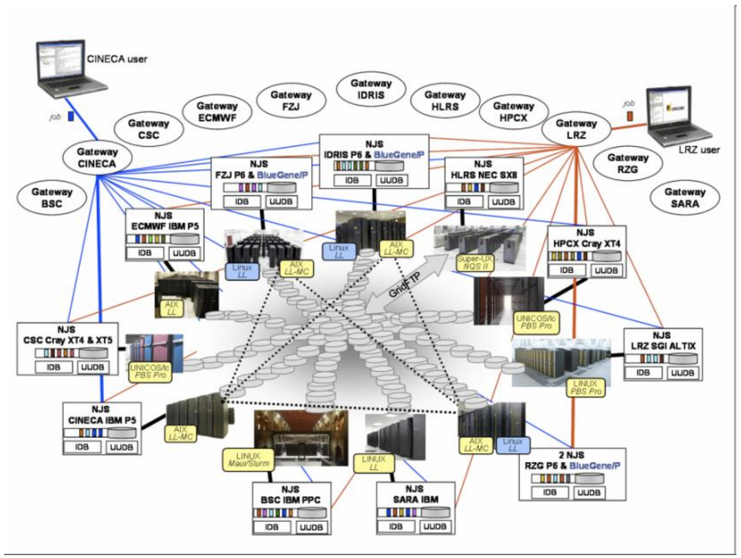
# GPFS (Spectrum Scale)

- IBM **G**eneral **P**urpose **F**ile**s**ystem
  - Files broken into blocks, striped over disks
  - Distributed metadata (including dir tree)
  - Extended directory indexes
  - Failure aware (partition based)
  - Fully POSIX compliant
- Storage pools and policies
  - Groups disks
  - Tiered on performance, reliability, locality
  - Policies move and manage data
  - Active management of data and location
  - Supports wide range of storage hardware
- High performance

# GPFS cont…

- Configuration
  - Shared disks (i.e. SAN attached to cluster)
  - Network Shared disks (NSD) using NSD servers
  - NSD across clusters (higher performance NFS)



# Configuring GPFS

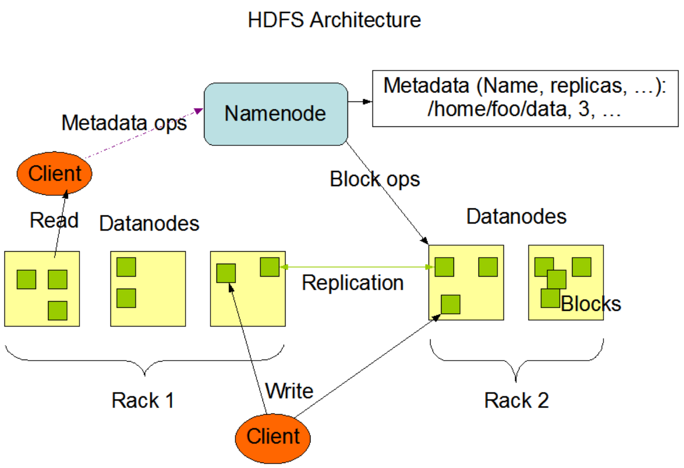
- Little experience so far of GPFS performance on DAC
  - MPI jobs limited to a single node
  - not clear what tuning can be done
- Previous experience from BlueGene/Q
  - performance seems to scale well with number of processors
  - no equivalent of tuning Lustre striping is required

# AFS

- Andrews Filesystem
  - Large/wide scale NFS
  - Distributed, transparent
  - Designed for scalability
- Server caching
  - File cached local, read and writes done locally
  - Servers maintain list of open files (callback coherence)
  - Local and shared files
- File locking
  - Doesn’t support large databases or updating shared files
- Kerberos authentication
  - Access control list on directories for users and groups

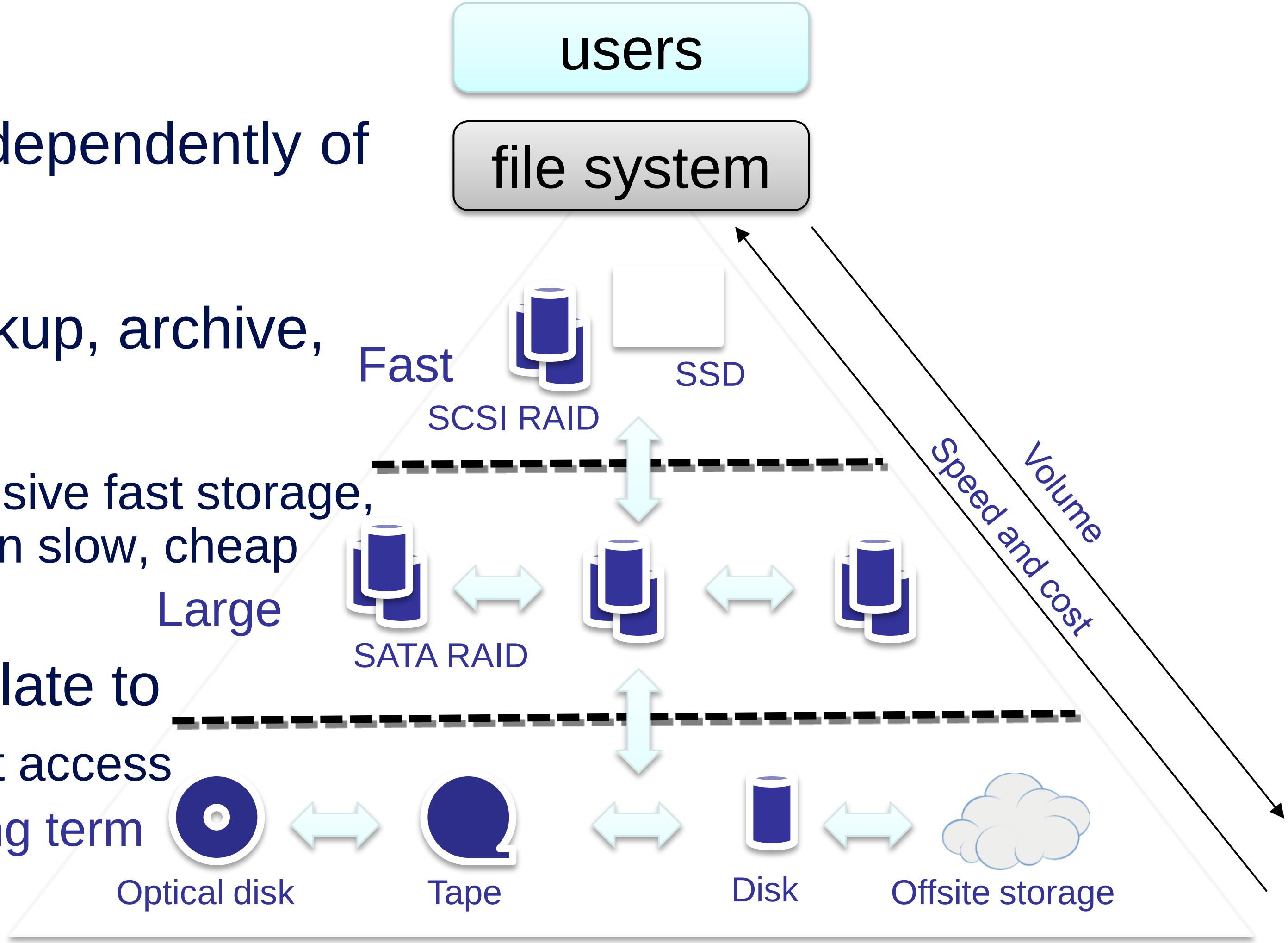
# HDFS

- Hadoop distributed file system
  - Distributed filesystem with built in fault tolerance
  - Relaxed POSIX implementation to allow data streaming
  - Optimised for large scale
- Java based implementation
  - Separate data nodes and metadata functionality
  - Single NameNode performs filesystem name space operations
  - Similar to Lustre decomposition
    - Namenode -> MDS server
- Block replication undertaken
  - Namenode “RAIDs” data
  - Namenode copes with DataNode failures
  - Heartbeat and status operations

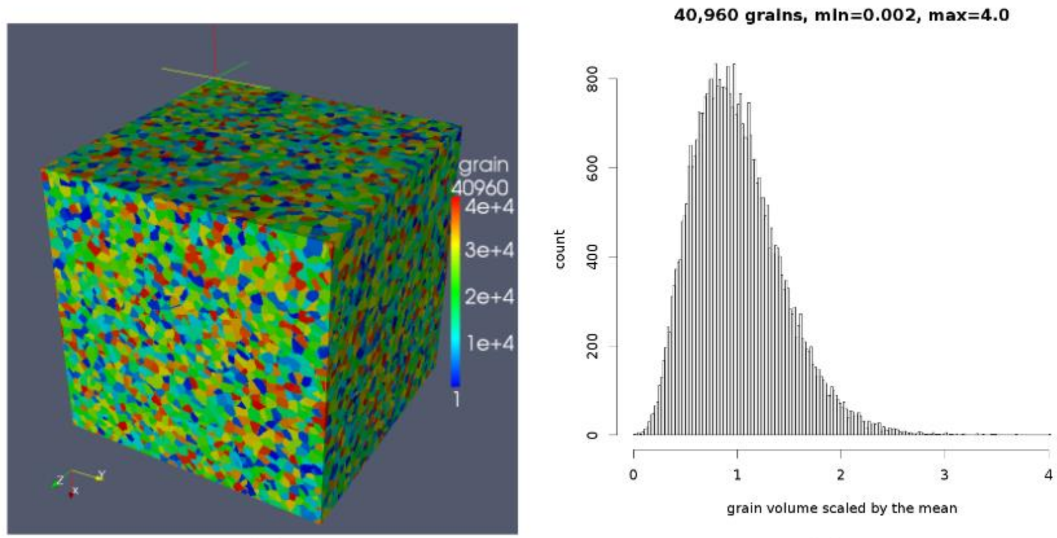


# Hierarchical storage management

- HSM moves data between storage levels based on policies
- Data moved independently of users
- May be for backup, archive, staging
  - Manage expensive fast storage, maintain data in slow, cheap storage
- Policies may relate to
  - Time since last access
  - Fixed time
  - Events



# Cellular Automaton Model




(a) (b)

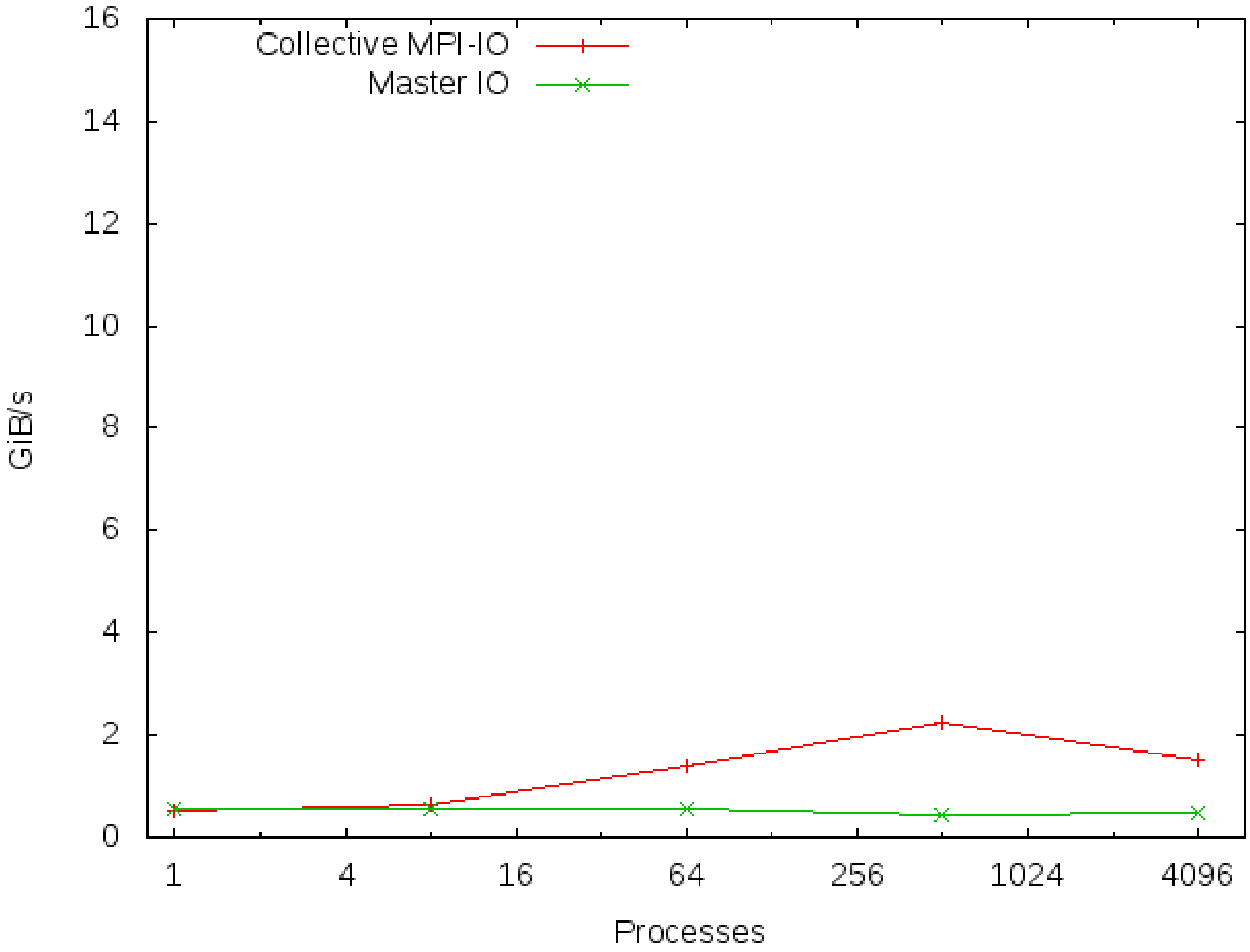
Figure 1: A $4.1 \times 10^9$ cell, 40,960 grain equiaxed microstructure model, showing (a) grain arrangement with colour denoting orientation; (b) grain size size (volume) histogram.

- *Fortran coarray library for 3D cellular automata microstructure simulation*, Anton Shterenlikht, proceedings of 7th International Conference on PGAS Programming Models, 3-4 October 2013, Edinburgh, UK.

# Benchmark

- Distributed regular 3D dataset across 3D process grid
  - set up for weak scaling
    - fixed local arrays, e.g. 128x128x128
    - replicated across processes
  - implemented in Fortran and MPI-IO, HDF5, NetCDF

# Parallel vs serial IO, default Lustre



# Results on ARCHER