I/O Hardware and Software

Adrian Jackson
adrianj@epcc.ed.ac.uk
@adrianjhpc

I/O

- I/O essential for all applications/codes
  - Some data must be read in or produced
  - Instructions and Data
- Basic hierarchy
  - CPU – Cache – Memory – Devices (including I/O)
- Often “forgotten” for HPC systems
  - Linpack not I/O bound
  - Not based on CPU clock speed or memory size
- Often “forgotten” in program
  - Start and end so un-important
  - Just assumed overhead

I/O

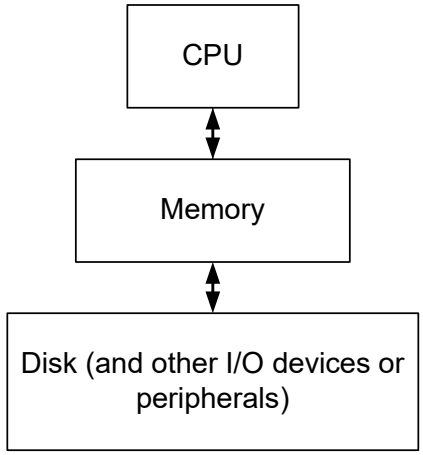
- Small parallel programs (i.e. under 1000 processors)
  - Cope with I/O overhead
- Large parallel programs (i.e. tens of thousand processors)
  - Can completely dominate performance
  - Exacerbate by poor functionality/performance of I/O systems
- Any opportunity for program optimisation important
  - Improve performance without changing program

Challenges of I/O

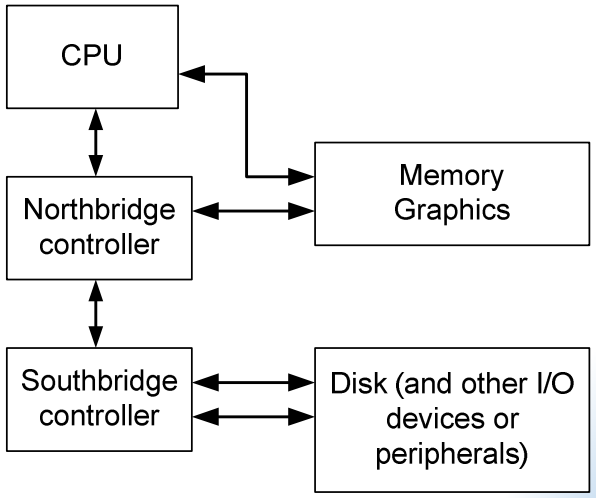
- Moves beyond process-memory model
  - data in memory has to physically appear on an external device
- Files are very restrictive
  - Don’t often map well to common program data structures (i.e. flat file/array)
  - Often no description of data in file
- I/O libraries or options system specific
  - Hardware different on different systems
- Lots of different formats
  - text, binary, big/little endian, Fortran unformatted, ...
  - Different performance and usability characteristics
- Disk systems are very complicated
  - RAID disks, caching on disk, in memory, I/O nodes, network, etc…

Challenges of I/O

- Standard computer hardware
  - Possibly multiple disks
  - PATA, SATA, SCSI (SAS)
- Optimisations
  - RAID (striping and replication)
  - Fast disks (SSD or server)
- HPC/Server/SAN hardware
  - Many disks
  - SCSI (SAS), Fibre channel
- Optimisations
  - Striped
  - Multiple adapters and network interfaces
- Network filesystems
  - Provide access to data from many machines and for many users



Abstract Hardware Hierarchy



Actual Hardware Hierarchy

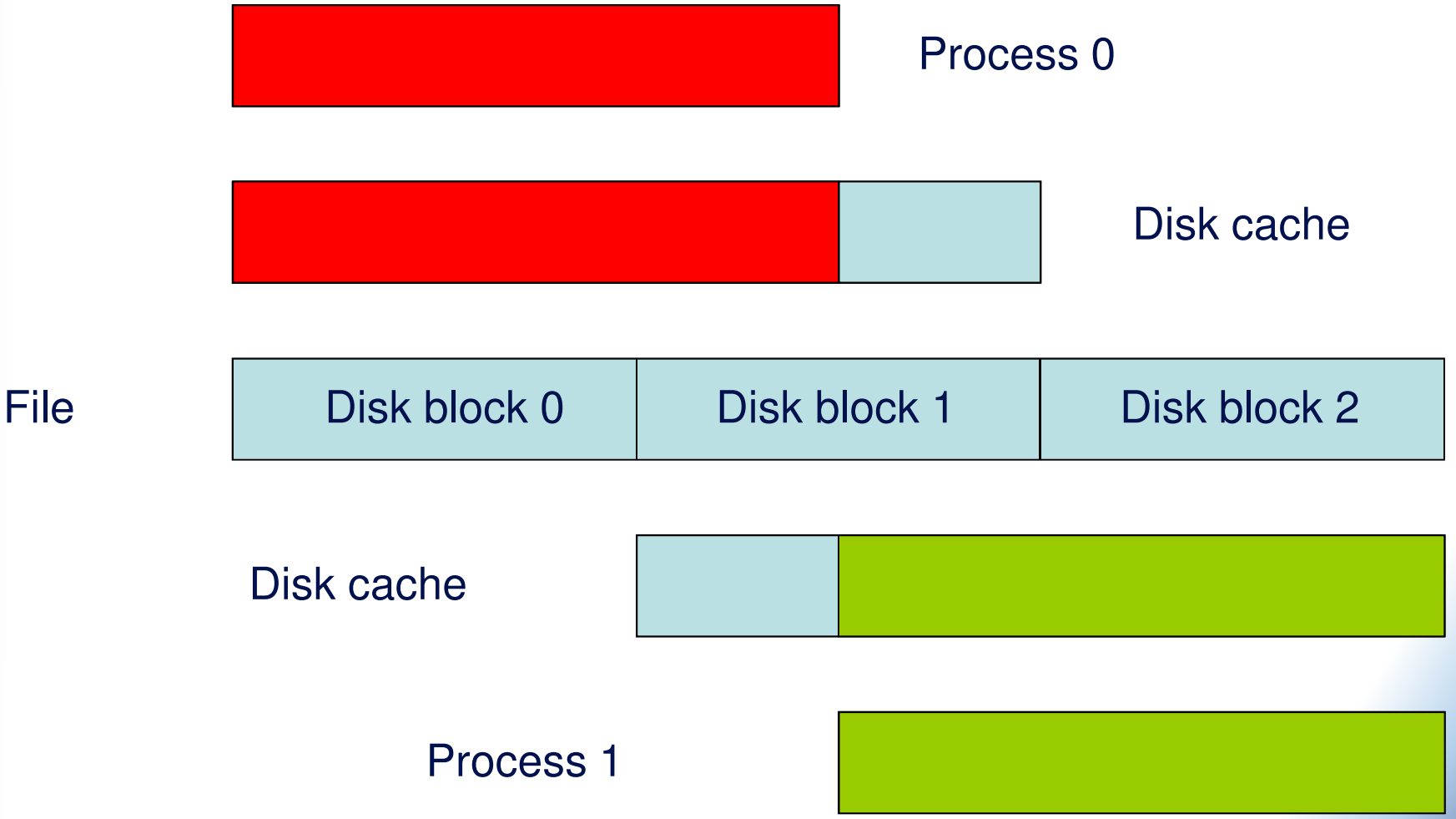
Performance

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

High Performance or Parallel I/O

- Lots of different methods for providing high performance I/O
- Hard to support multiple processes writing to same file
  - Basic O/S does not support
  - 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 limit on file handles etc….
- Data is distributed across different processes
  - Dependent on number of processors used, etc…
- Parallel file systems may allow multiple access
  - but complicated and difficult for the user to manage

Simultaneous access to a file

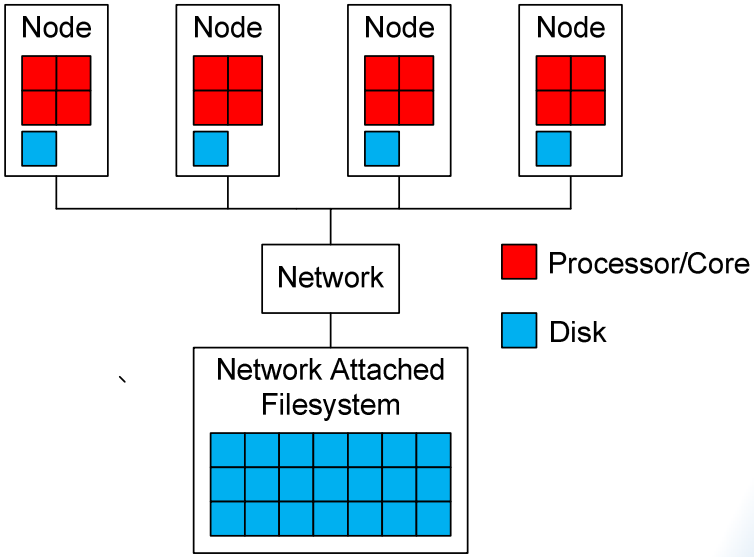


Parallel File Systems

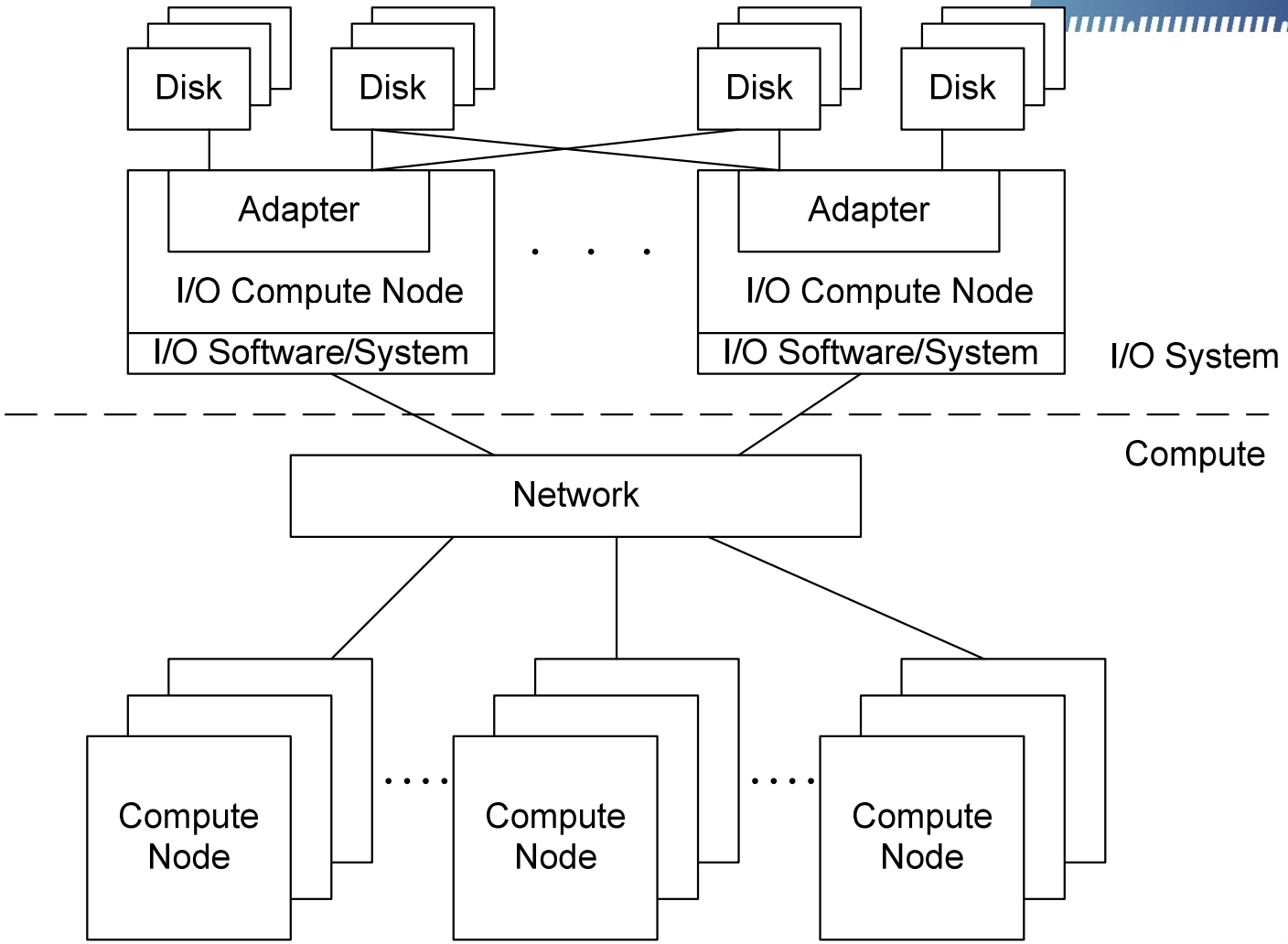
- Parallel computer
  - constructed of many processors
  - performance comes from using many processors at once
  - requires distribution of data and calculation across processors
- Parallel file systems
  - constructed from many standard disk
  - performance comes from reading / writing to many disks
  - 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 *MetaData* Server (MDS)
  - can become a bottleneck for performance

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…



Hierarchy


Disk
Disk
Disk
Disk
Adapter
Adapter
I/O Compute Node
I/O Compute Node
I/O Software/System
I/O Software/System
I/O System
Compute
Network
Compute Node
Compute Node
Compute Node

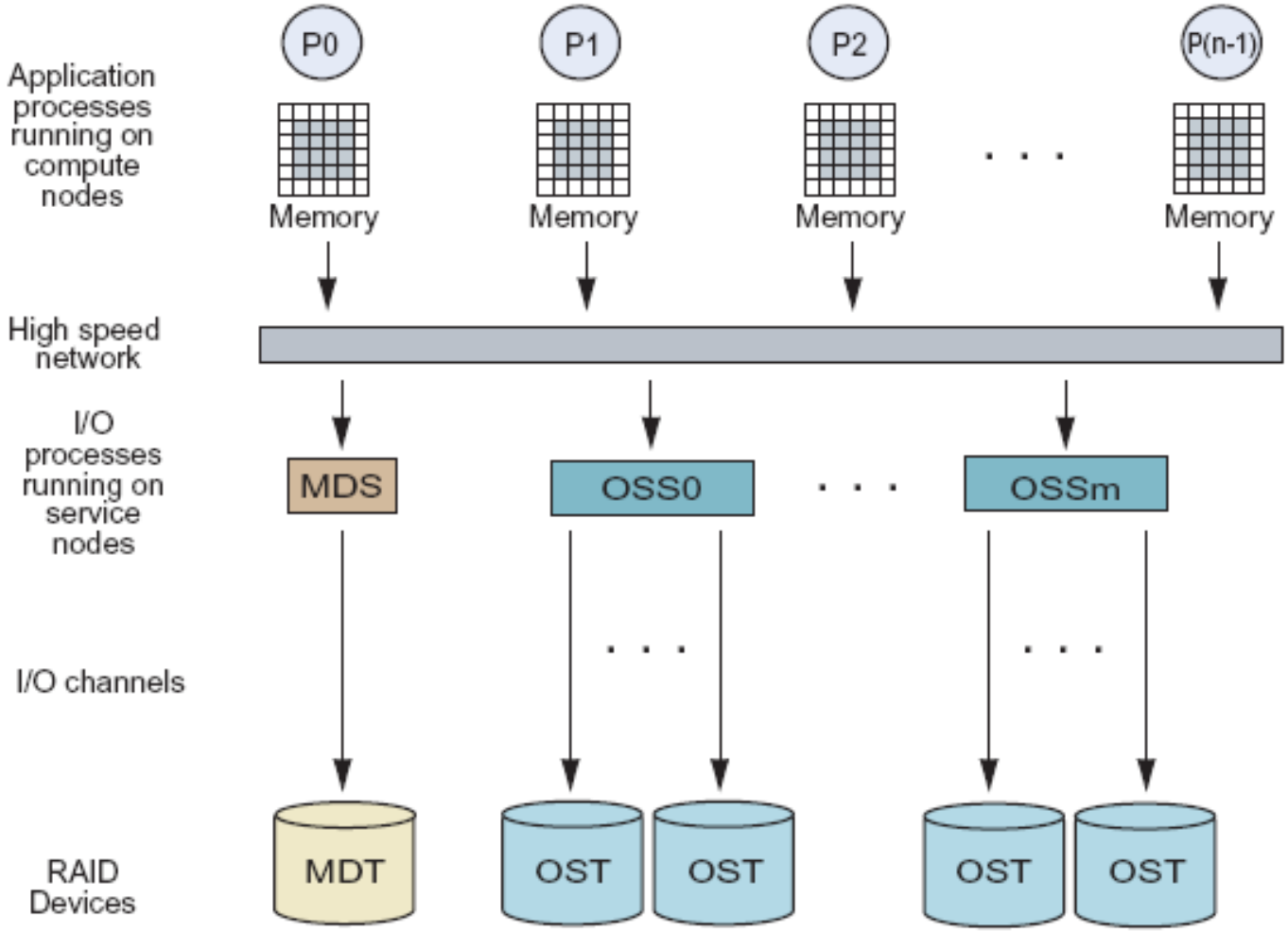
POSIX I/O

- Standard interface to files
  - Unix/Linux approach
  - Based on systems with single filesystem
  - open, close, write, read, etc…
- Does not support parallel or HPC I/O well
  - Many NFS don't fully implement it for performance reasons
- Some work on extending for HPC

Lustre

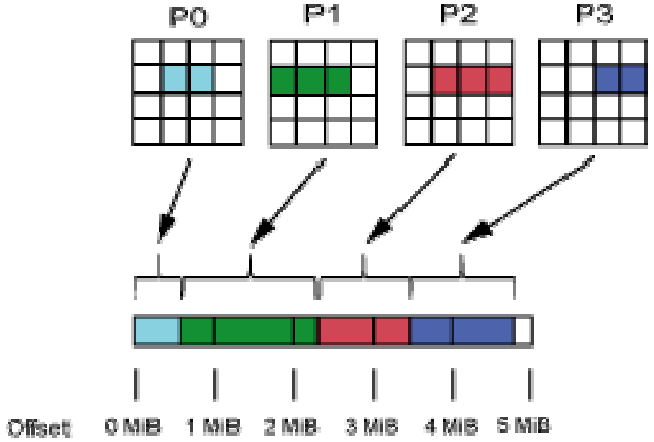
- Three functional units
  - Object Storage Servers (OSS)
    - Store data on one or more Object Storage Targets (OST)
    - The **OST** handles interaction between client data request and underlying physical storage
    - An **OSS** typically serves 2-8 targets, each target a local disk system. The capacity of the Lustre file system is the sum of the capacities provided by the targets
    - The **OSS** operate in parallel, independent of one another
  - Metadata Target (MDT)
    - One per filesystem, storing all metadata: filenames, directories, permissions, file layout
    - Stored on Metadata Server (MDS)
  - Clients
    - Supports standard POSIX access

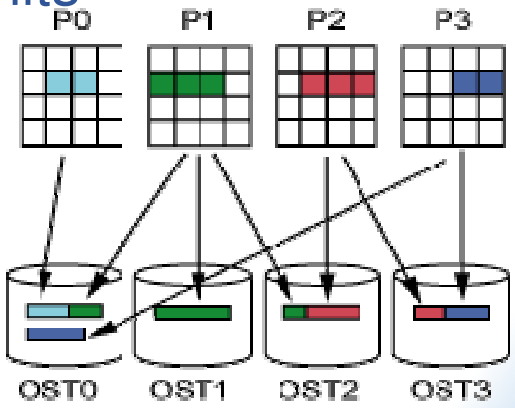
Lustre cont.


P0
P1
P2
P(n-1)
Application processes running on compute nodes
Memory
Memory
Memory
Memory
High speed network
I/O processes running on service nodes
MDS
OSS0
OSSm
I/O channels
RAID Devices
MDT
OST
OST
OST
OST

Lustre cont…

- Supports different networks
  - Infiniband, Ethernet, Myrinet, Quadrics
- Striping
  - Data striped across OSTs (round robin)
  - File split into units
  - Simultaneous read/write to different units





Lustre commands

- Striping cont.
  - Improves bandwidths, overall performance available, and maximum file size
  - Incurs communication overhead and contention potentials including serialisation if multiple processes accessing same units
- lfs command for more information and configuration

```
adrianj@nid16958:~>lfs df -h
```

(query number of OSTs)

```
adrianj@nid16958:~> lfs getstripe dirname
```

(query `stripe count, stripe size`)

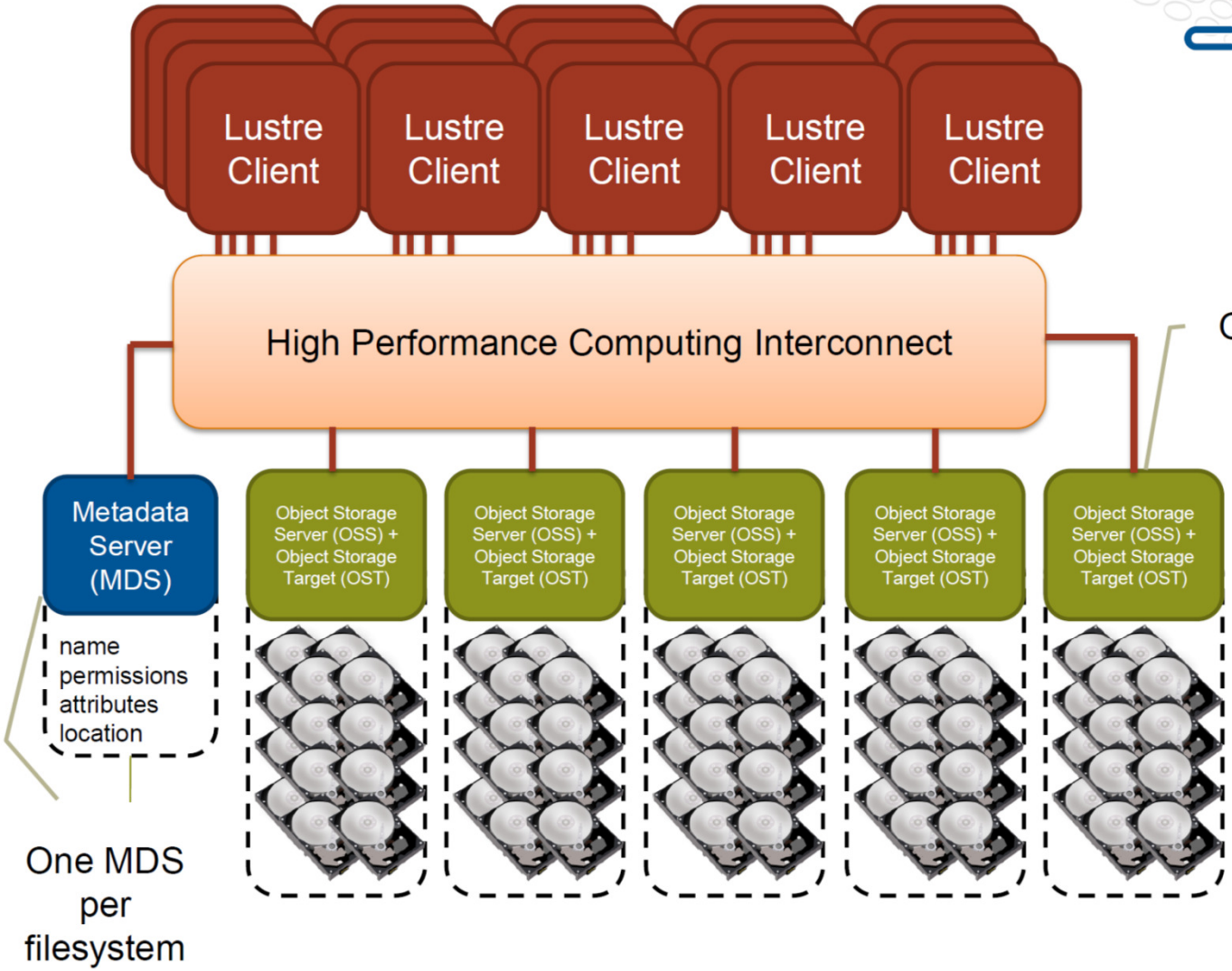
```
adrianj@nid16958:~> lfs setstripe dirname 0 -1 -1
```

(set large file `stripe size, start index, stripe count`)

```
adrianj@nid16958:~> lfs setstripe dirname 0 -1 1
```

(set lots of files `stripe size, start index, stripe count`)

Example I/O hardware: ARCHER



Lustre on ARCHER

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

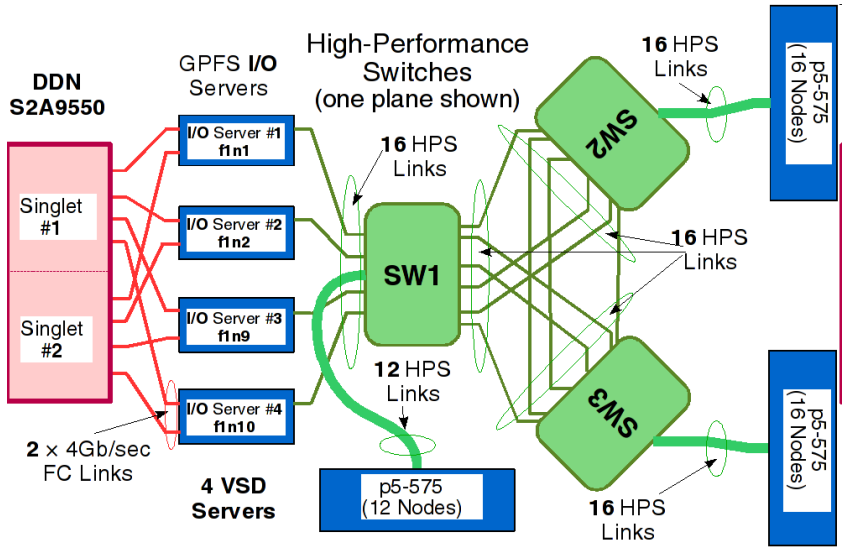
- http://www.archer.ac.uk/documentation/white-papers/parallelIO/ARCHER_wp_parallelIO.pdf

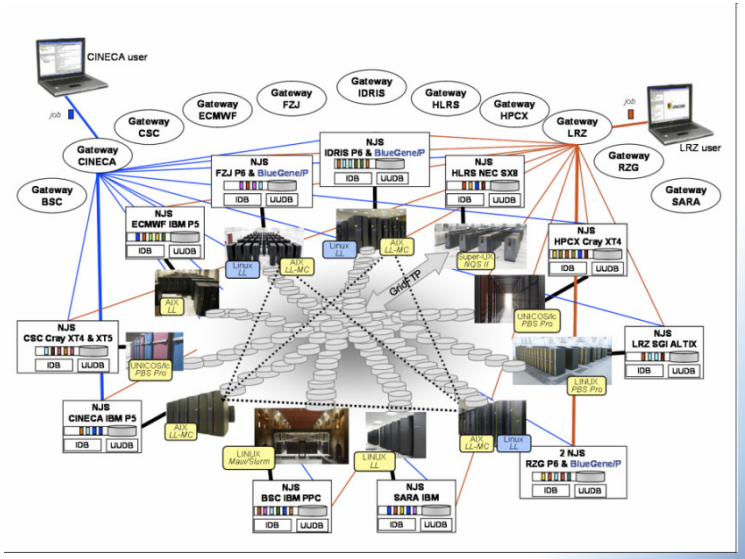
GPFS

- 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
- 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)





AFS

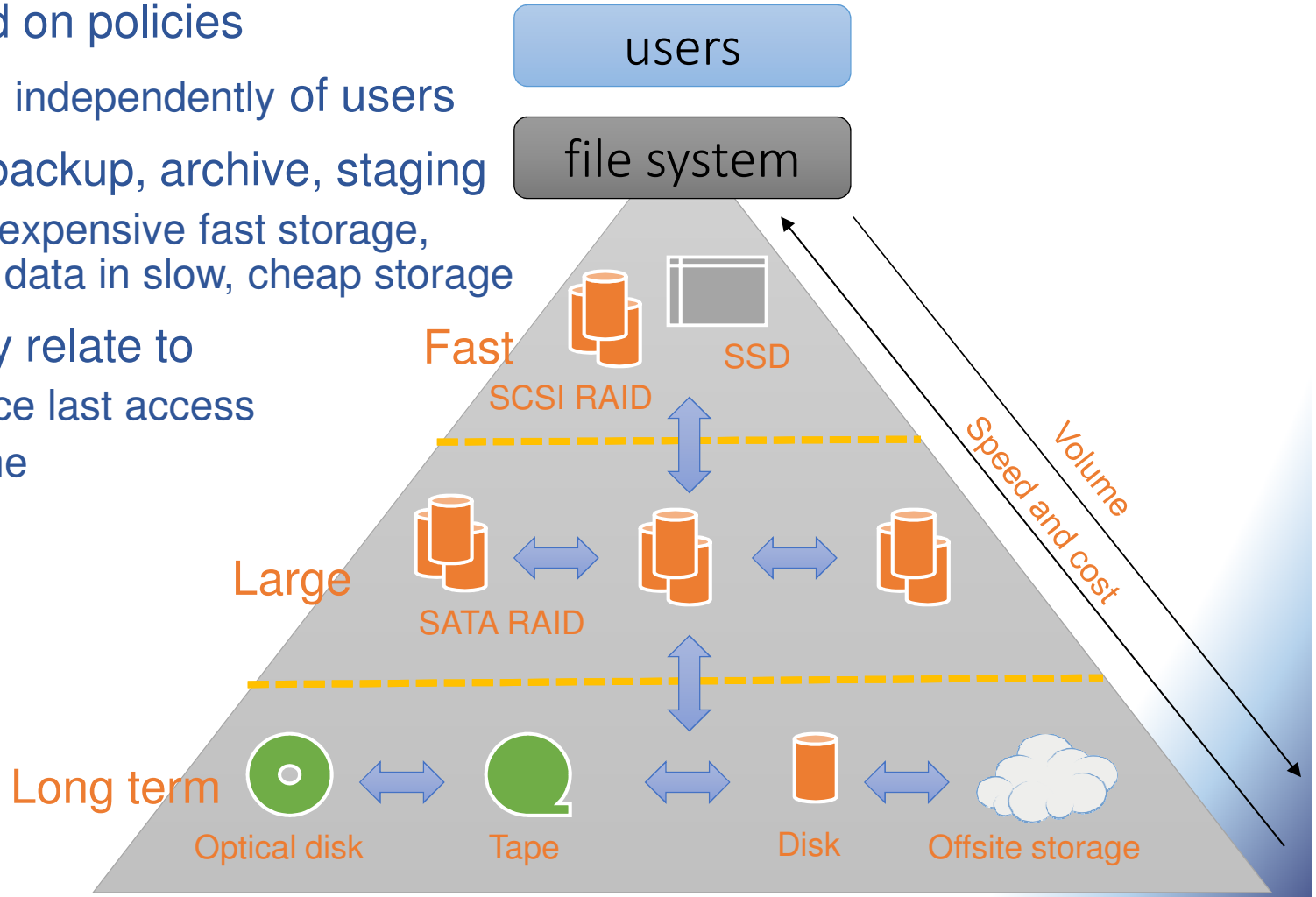
- Andrews Filesystem
  - Large/wide scale NFS
  - Distributed, transparent
  - Designed for scalability
- Server caching
  - File cached local, read and writes done locally
  - Servers maintain list of open files (callback coherence)
  - Local and shared files
- File locking
  - Doesn’t support large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

zfs

- Filesystem and logical volume manager
  - Focus on data integrity
  - 128-bit filesystem (very large potential volume)
  - Encryption built in (if desired)
  - Storage pools for different types of storage
  - Cache-like storage hierarchy
- Data integrity
  - Blocks are checksummed
  - Checksums are also checksummed up the file system tree
  - Enables detection of corruption of checksums as well as corruption of block data
  - Full filesystem integrity
- Sun developed
  - Now supported by OpenZFS
  - Not deployed by vendors

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



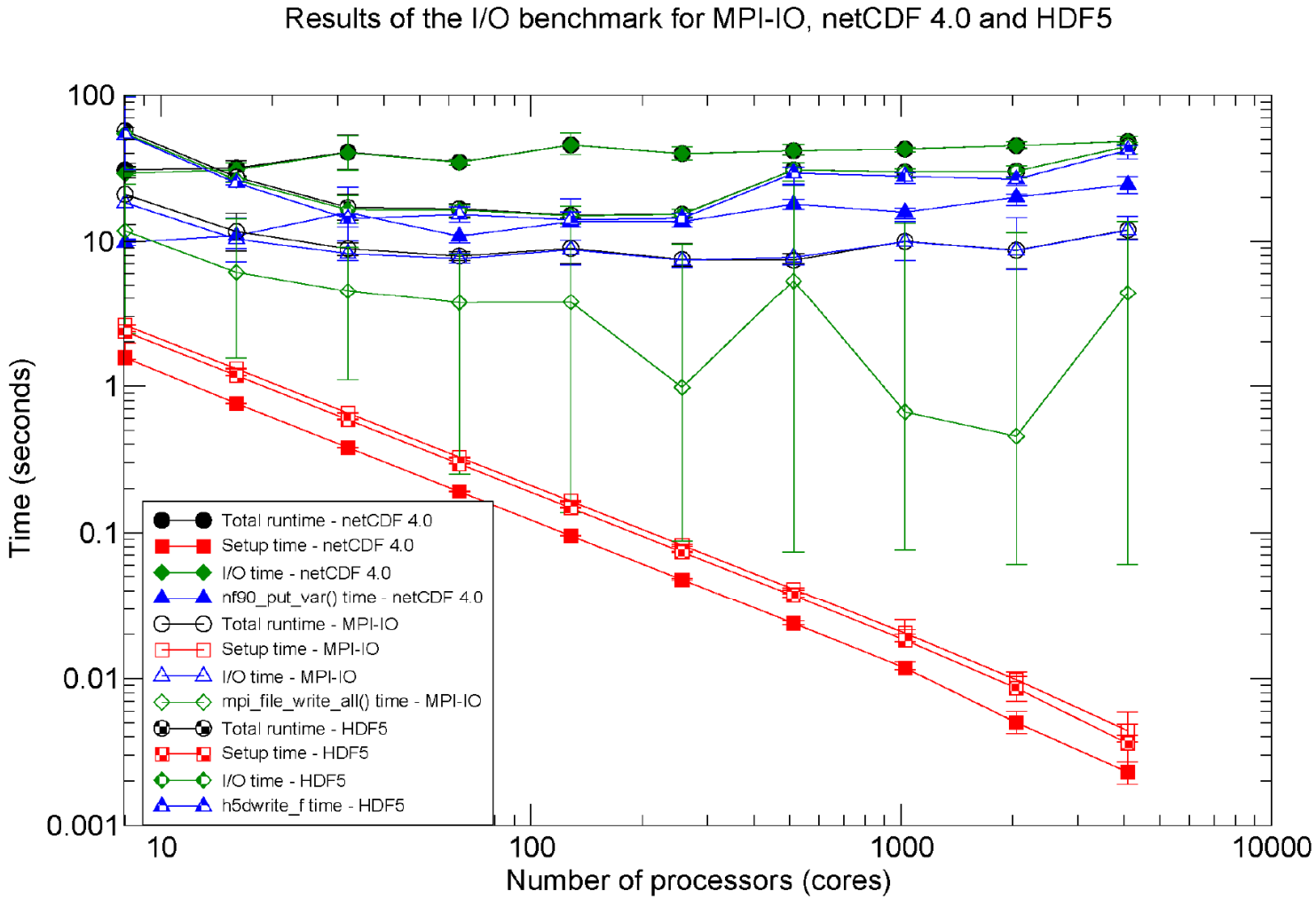
Software layers

- MPI-I/O
  - Fast direct parallel access to files in parallel (multiple reads and writers allowed)
- HDF5
  - Can build on MPI-I/O
  - Gives metadata operations (allows addition of structure and metadata to files)
- netCDF
  - Can build on HDF5 (and therefore MPI-I/O)
  - Specific structures/interfaces/tools for modelling communities (climate/earth science)
- ….
  - Other similar formats designed for specific communities

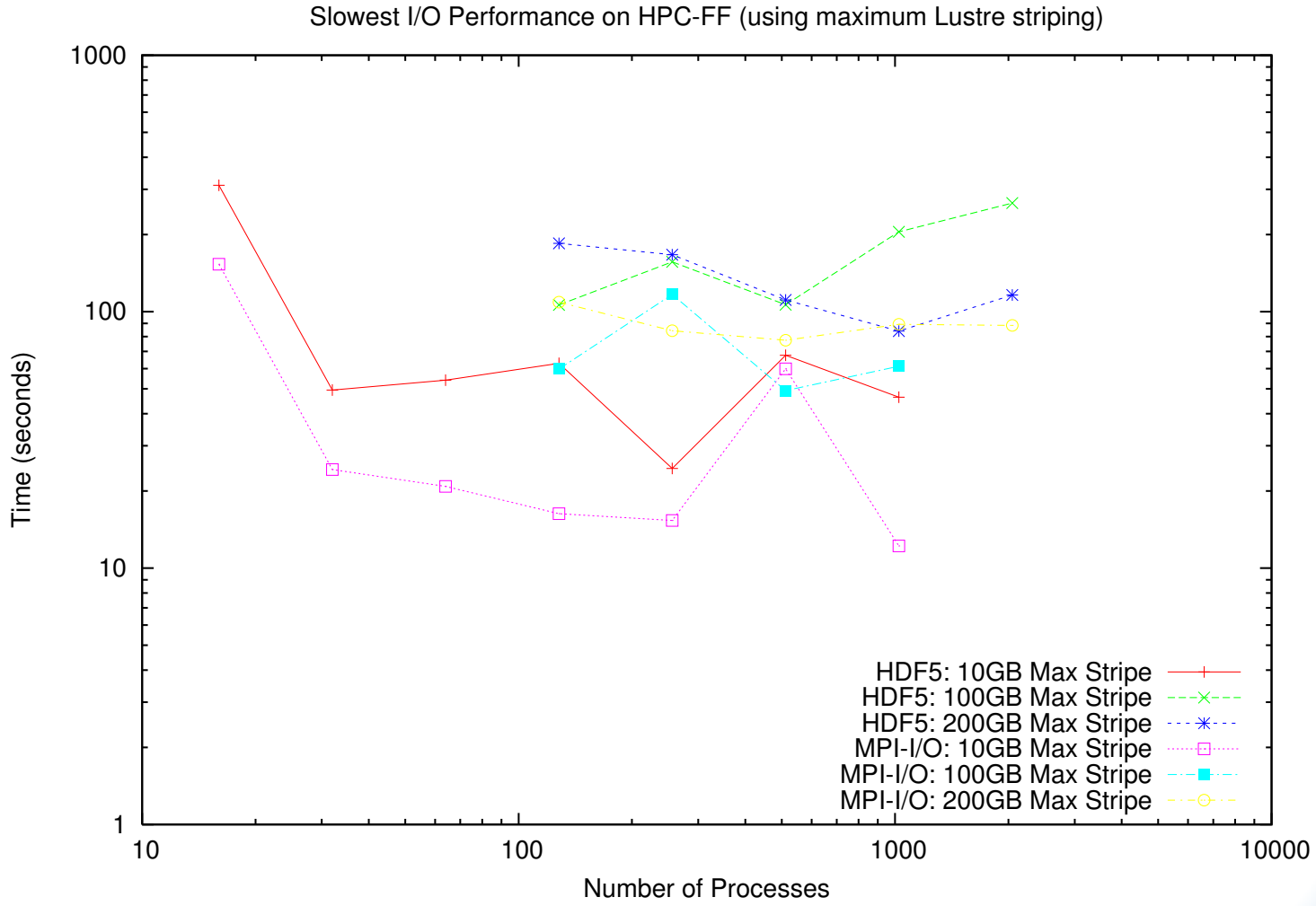
None Filesystem based data

- Database storage
  - Traditional SQL databases don't scale to very large distributed system
- NoSQL
  - Relax ACID ((Atomicity, Consistency, Isolation, Durability) transaction guarantees (i.e. eventual consistency rather than immediate consistency)
  - None-relational or tabular, i.e. key-value, graph, object storage, etc…
  - Support querying, storage, and retrieval of data in SQL-like formats
- NewSQL databases
  - Scale database to large, distributed, systems
  - Enable SQL querying
  - Support ACID guarantees
  - Custom SQL engines, data sharding infrastructure, distributed cluster nodes…

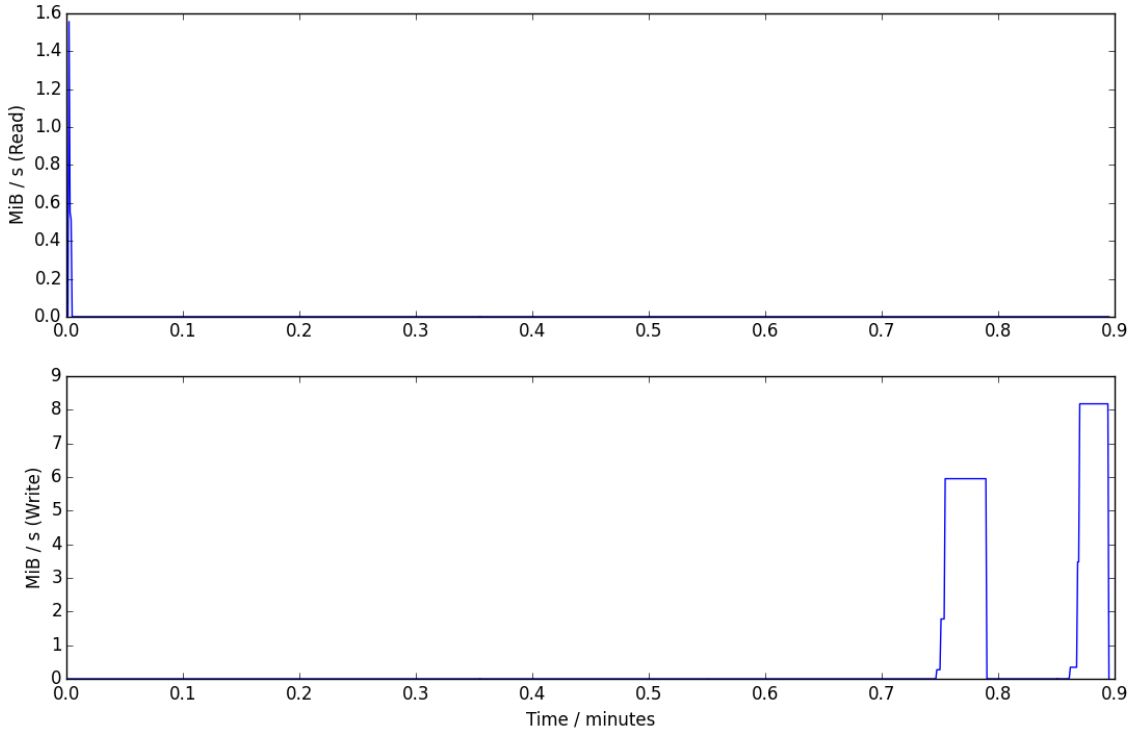
Performance



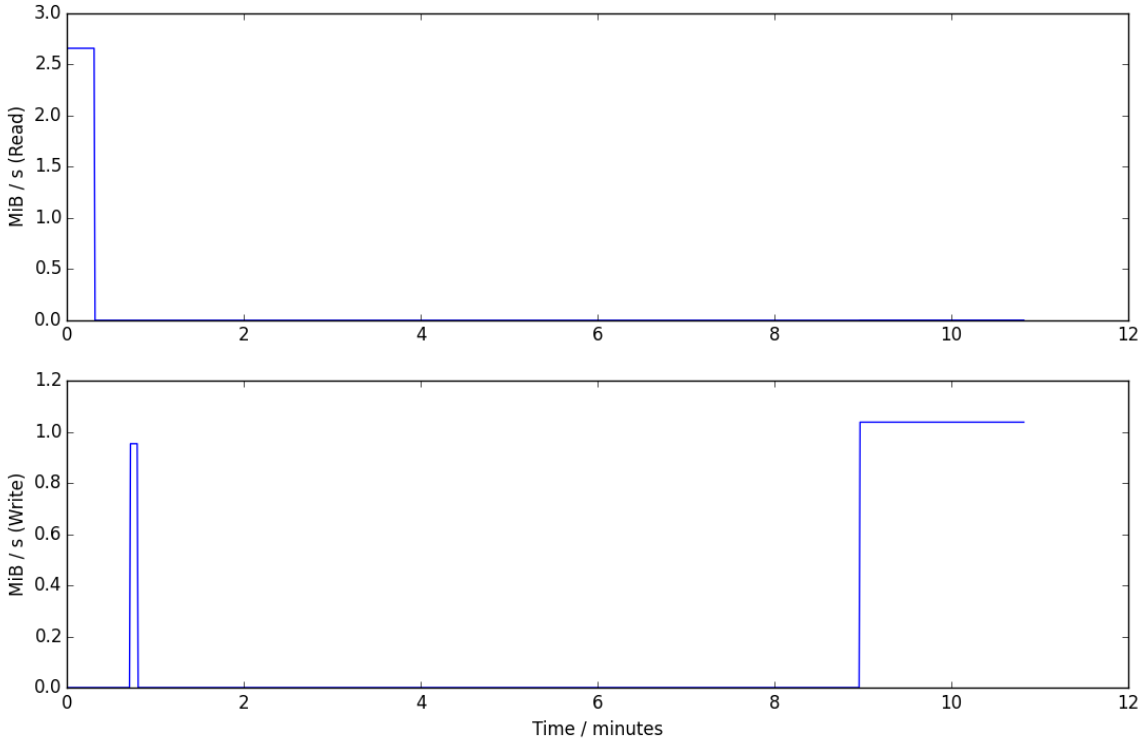
Performance – HDF5 vs MPI-I/O



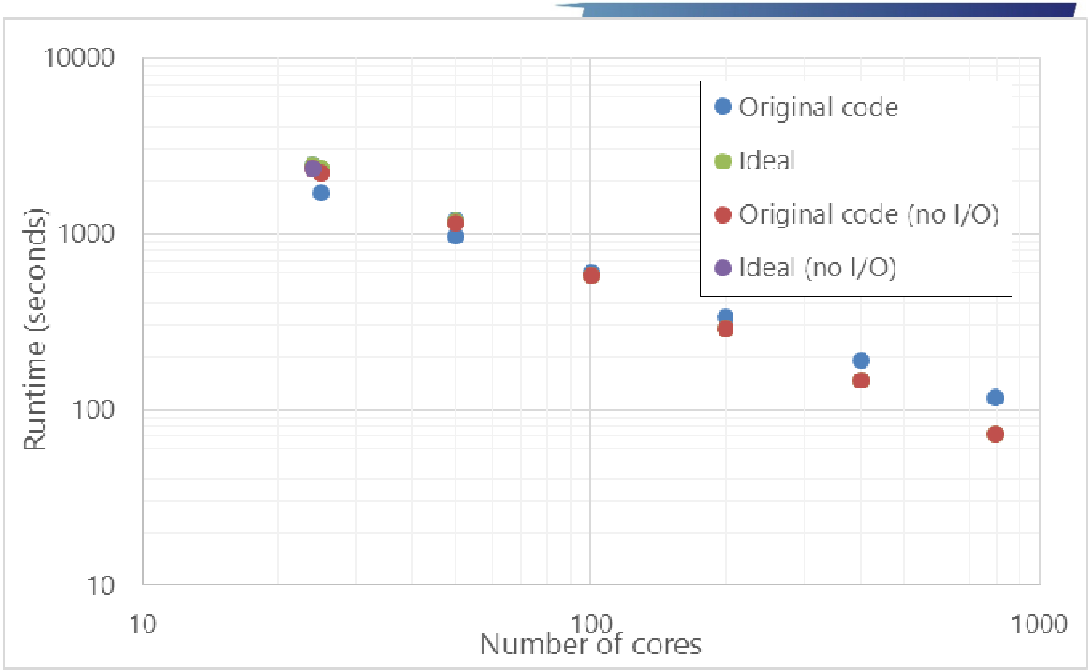
I/O

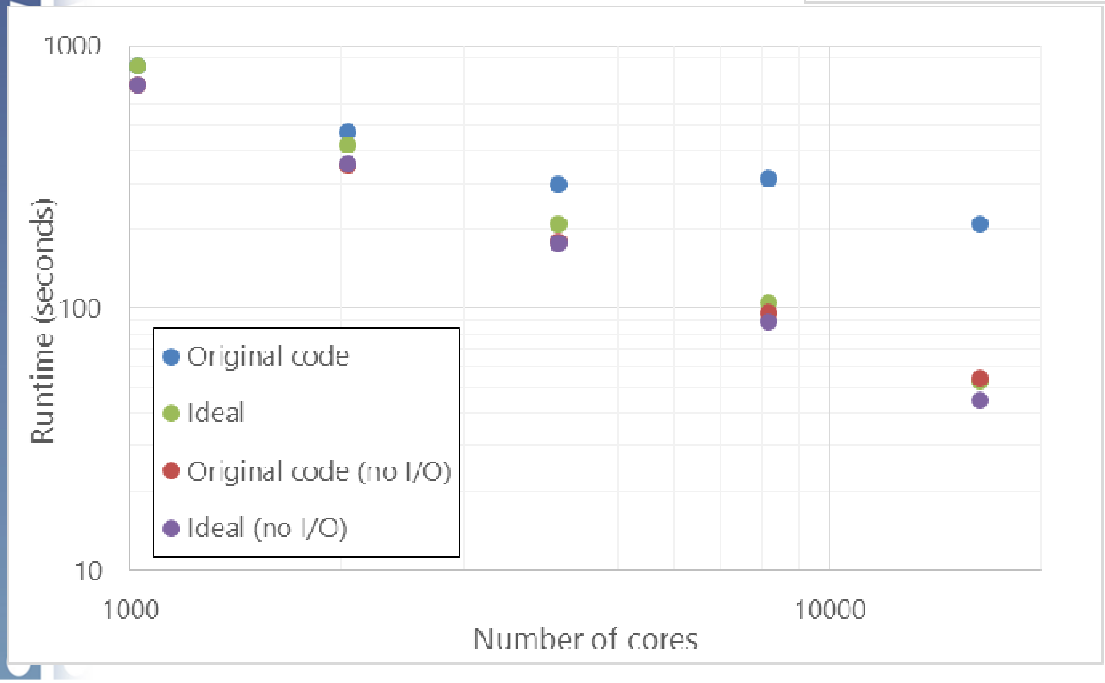

MiB / s (Read)
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
MiB / s (Write)
9
8
7
6
5
4
3
2
1
0
Time / minutes

I/O


MiB / s (Read)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
12
MiB / s (Write)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Time / minutes

I/O Performance





Performance

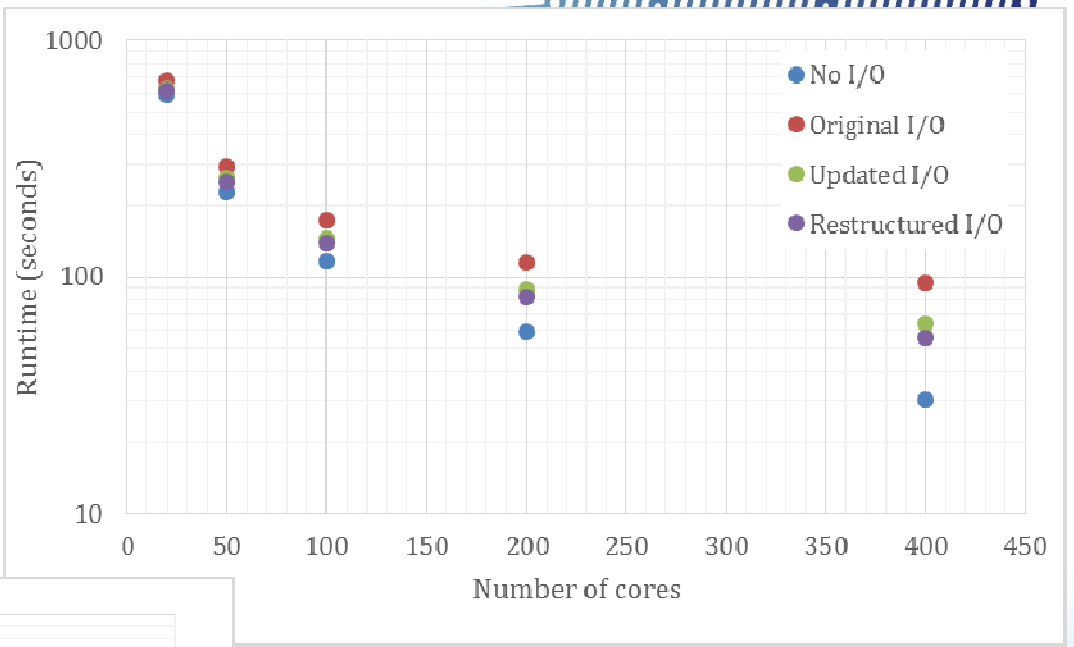
100 processes

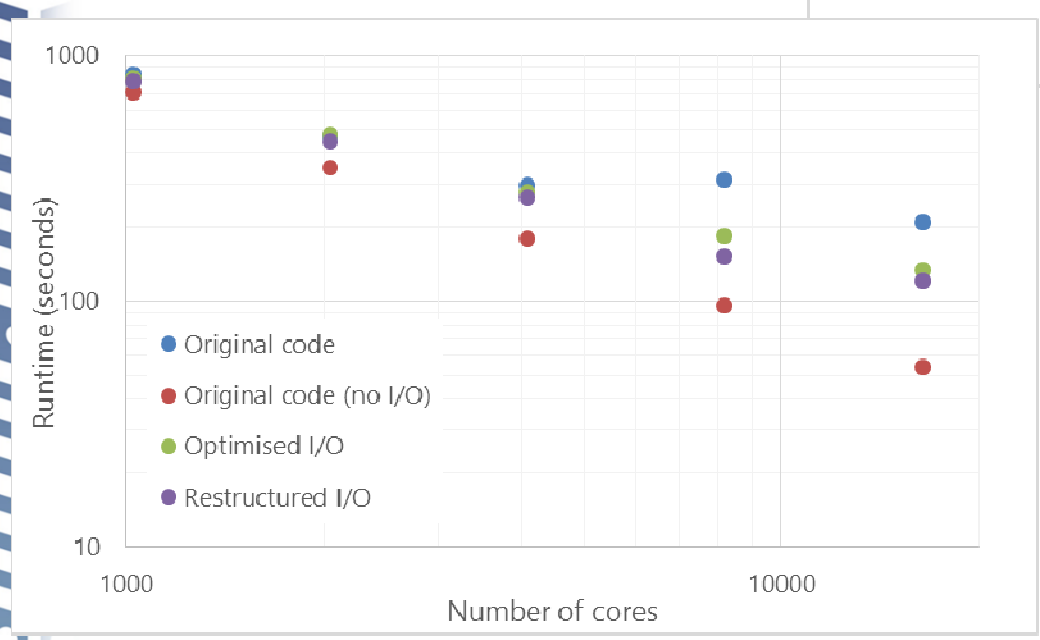
Samp%	Samp	Imb. Samp	Imb. Samp%	Group Function PE=HIDE
100.0%	61,775.2	--	--	Total
78.1%	48,217.5	--	--	USER
19.0%	11,747.3	3,485.7	23.1%	vflux_
8.8%	5,418.5	857.5	13.8%	roflux_
6.1%	3,764.0	1,434.0	27.9%	muscl_
4.7%	2,909.9	1,479.1	34.0%	q_face_
4.1%	2,555.1	364.9	12.6%	tridi_
3.9%	2,380.3	1,327.7	36.2%	bresid_
3.7%	2,302.4	1,445.6	39.0%	muscl_bi_
3.2%	1,972.9	1,146.1	37.1%	rtst_
19.0%	11,742.3	--	--	MPI
11.1%	6,831.8	33,844.2	84.0%	mpi_waitany
5.3%	3,262.0	910.0	22.0%	MPI_FILE_WRITE

800 processes

Samp%	Samp	Imb. Samp	Imb. Samp%	Group Function PE=HIDE
100.0%	10,461.0	--	--	Total
56.3%	5,889.8	--	--	USER
13.1%	1,375.1	464.9	25.3%	vflux_
6.5%	683.7	132.3	16.2%	roflux_
4.5%	469.1	199.9	29.9%	muscl_
3.4%	355.9	192.1	35.1%	q_face_
3.1%	320.7	82.3	20.4%	tridi_
2.8%	293.7	193.3	39.7%	bresid_
41.4%	4,333.4	--	--	MPI
14.2%	1,482.2	1,111.8	42.9%	MPI_FILE_WRITE
10.5%	1,093.8	4,369.2	80.1%	mpi_waitany
8.4%	877.7	885.3	50.3%	mpi_file_open
7.0%	730.0	802.0	52.4%	MPI_BARRIER

Optimised I/O performance





Summary

- Many different *working parts*
- Most of them are shared
- A lot of complexity going on behind the scenes
- Applications also self censoring
  - Or moving the problem somewhere else
- Full workflow optimisation needs to take this into account
  - Making production I/O faster at expense of man hours might not be a good trade