
# Exploiting persistent memory for workflows and computational simulation

Adrian Jackson

@adrianjhpc

a.jackson@epcc.ed.ac.uk

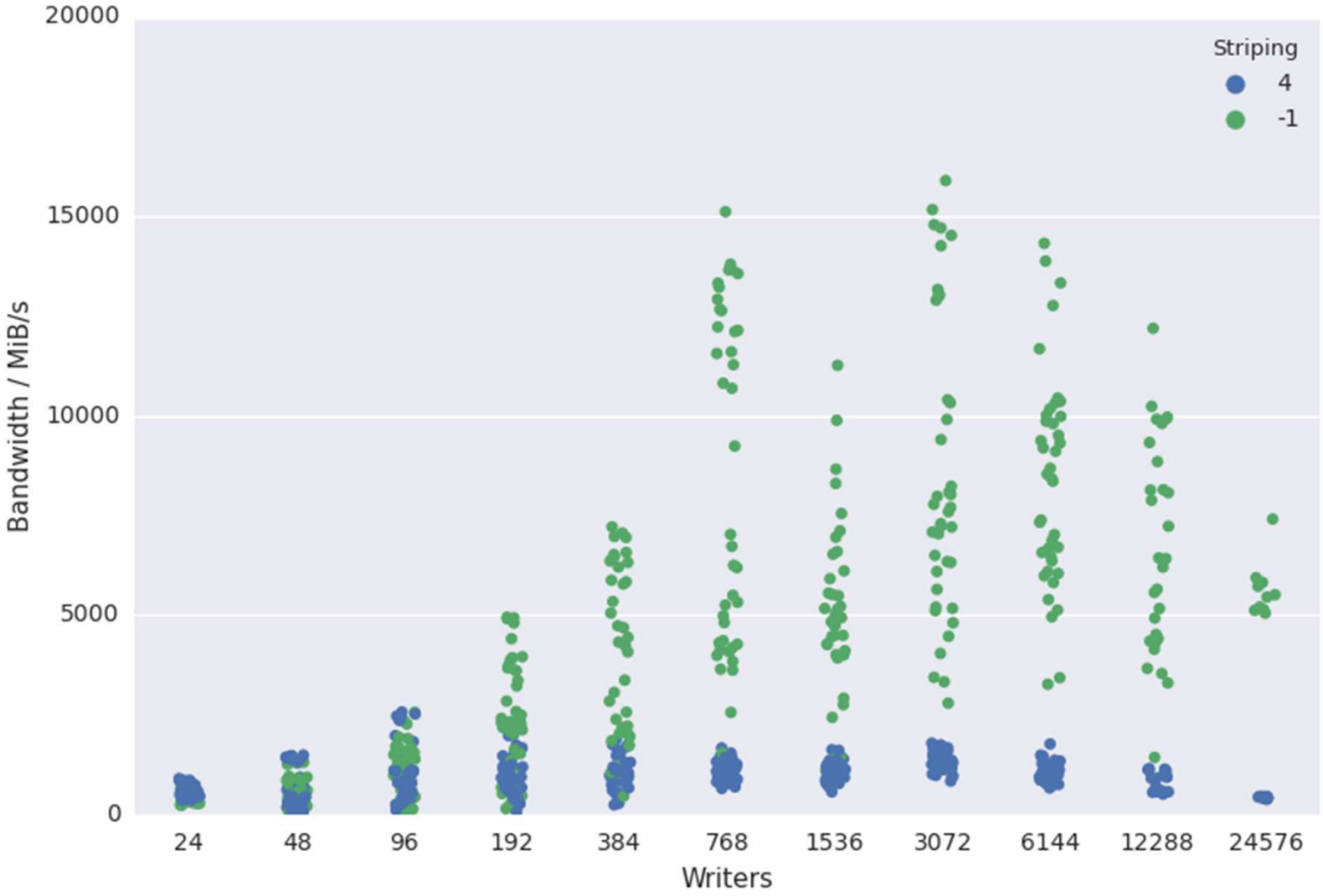
EPCC, The University of Edinburgh

http://www.nextgenio.eu

# Warning!


- Terminology might be annoying:
  - NVDIMM
  - NVRAM
  - PM (Persistent Memory)
  - SCM (Storage Class Memory (people get upset about this term))
  - B-APM (Byte-Addressable Persistent Memory (my favourite))
- My fault, but people will argue which is the most appropriate
  - So using them all to annoy as many people as possible ☺

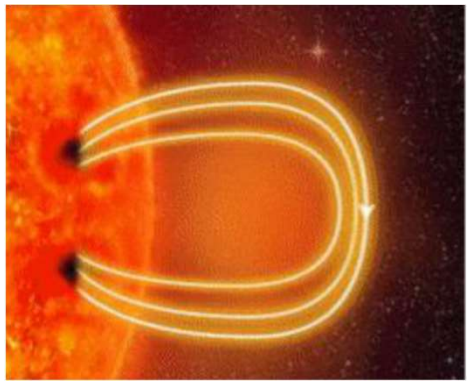
# I/O Performance

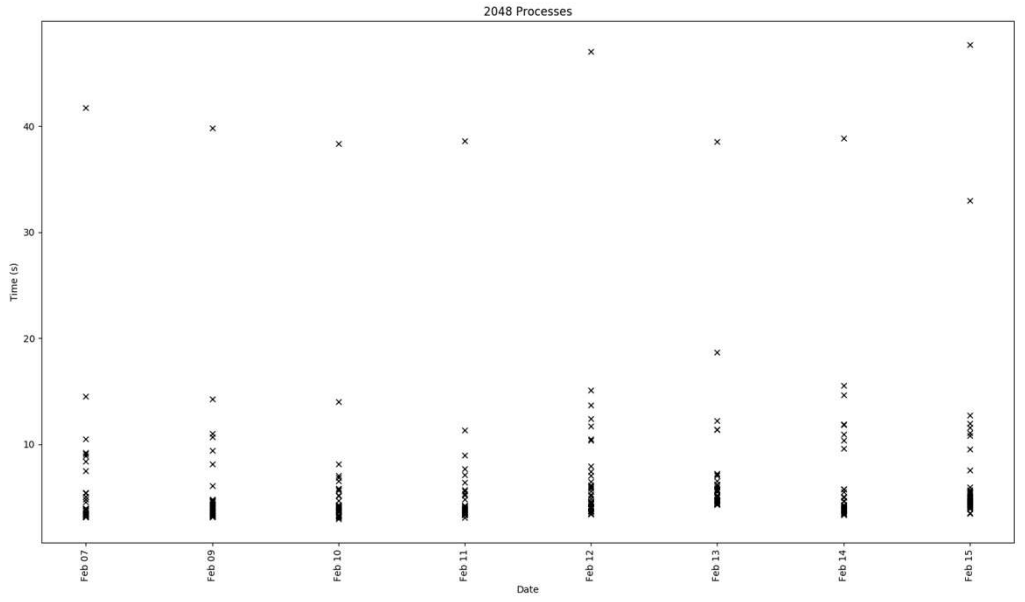



- https://www.archer.ac.uk/documentation/white-papers/parallelIO-benchmarking/ARCHER-Parallel-IO-1.0.pdf

# I/O Performance – Small writes

- Plot of average (across processes) run times of individual I/O regions for visualisation I/O
  - Same code executed for all runs
- I/O varies significantly in some cases:
  - Worst case ~12x
  - Best case ~2x

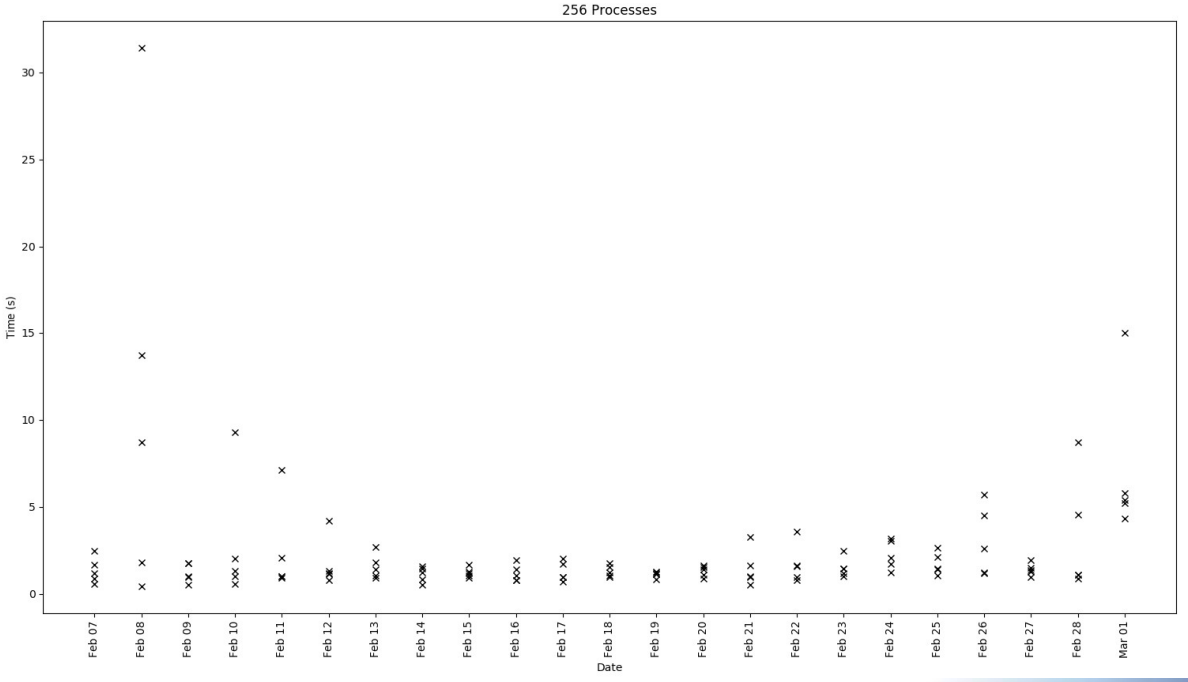




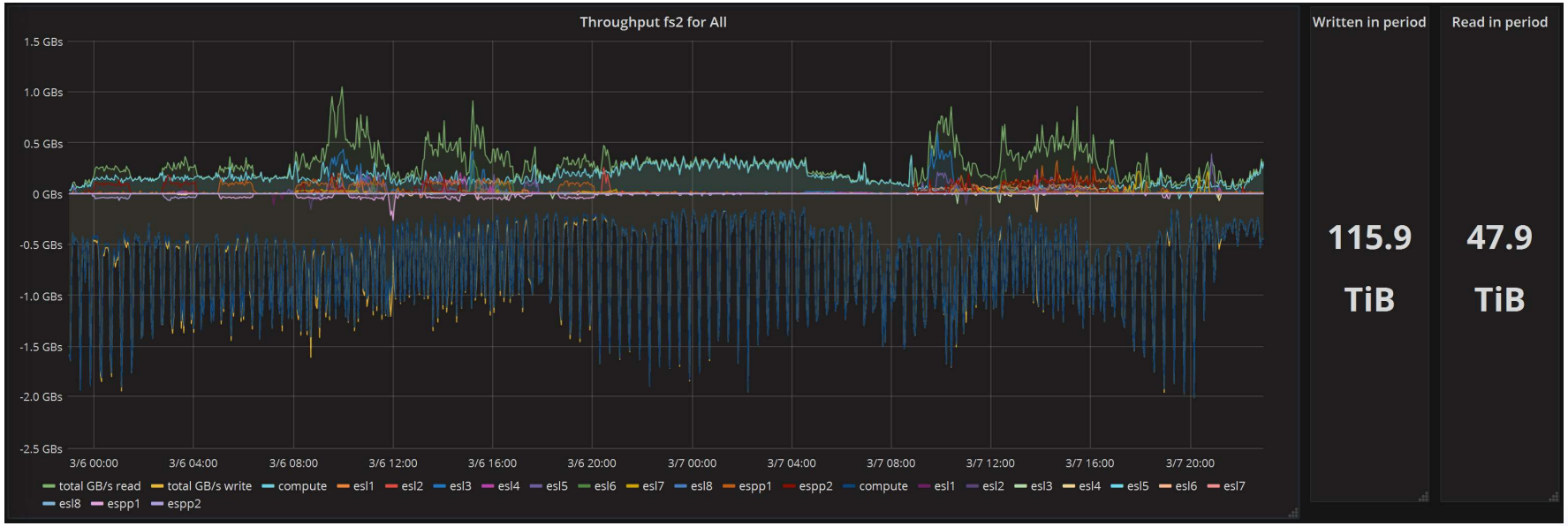
# I/O Performance – Large writes

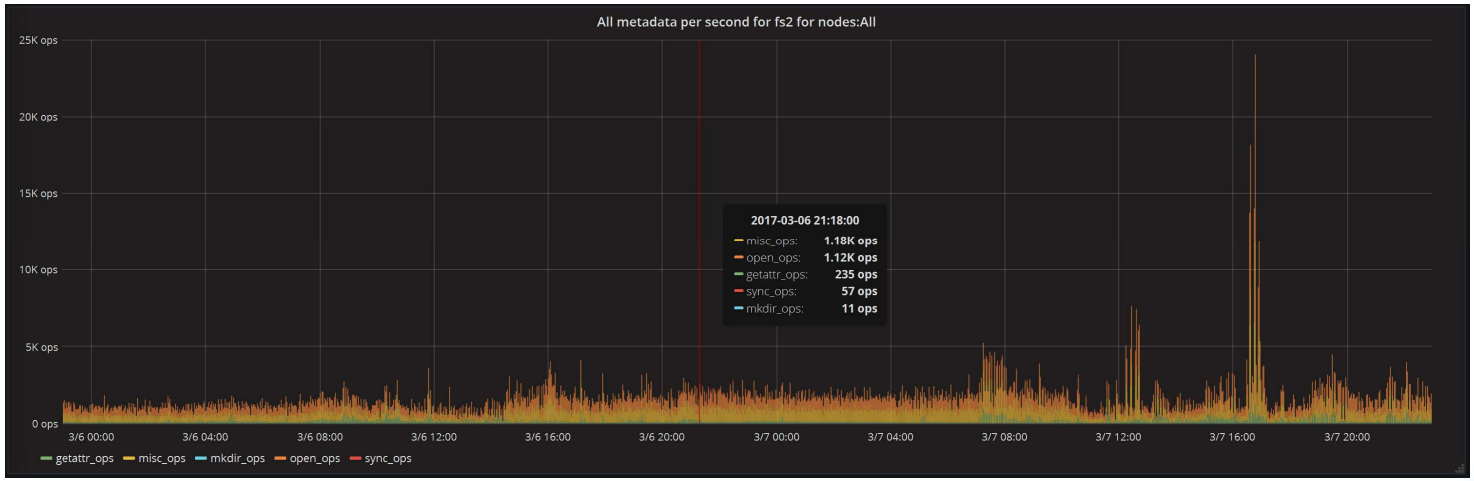

- Plot of run times of individual I/O regions for checkpoint I/O
  - Same code executed for all runs
- I/O varies in a similar pattern to the visualisation I/O
  - Variation more extreme (fastest is faster)
  - Average more consistent

- Checkpoint I/O less frequent but much quicker
  - Much higher data volumes

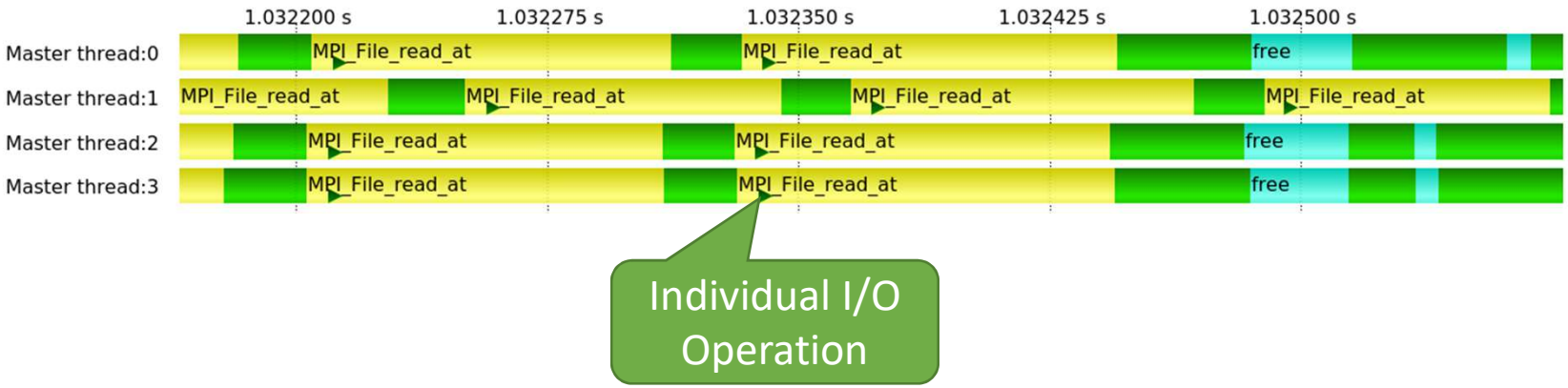


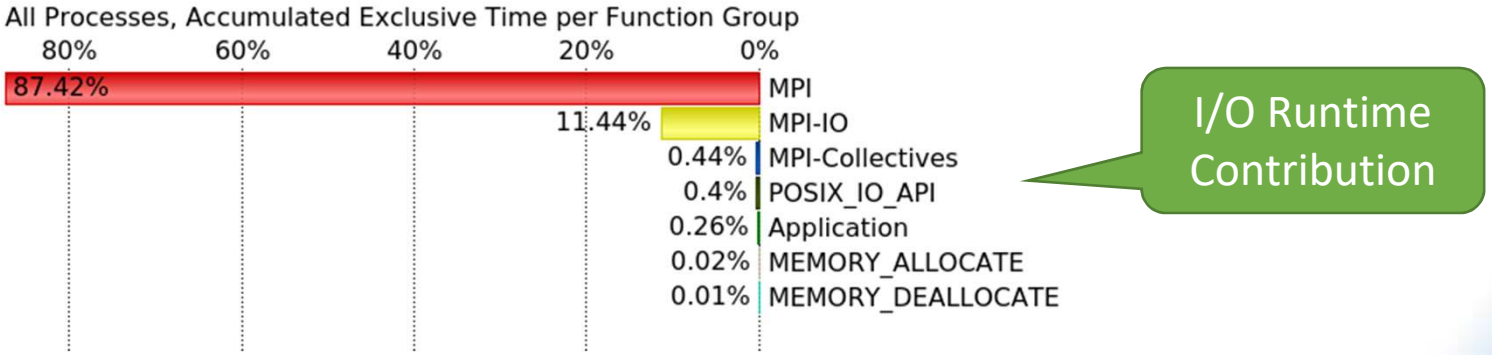
# I/O Performance



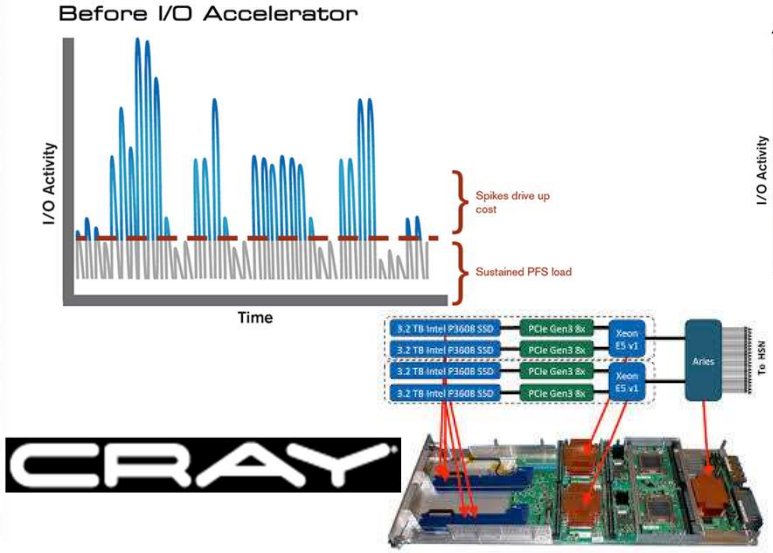
# Application I/O patterns

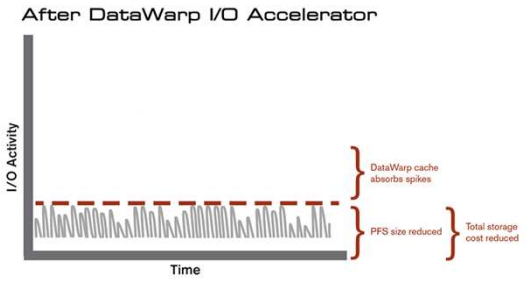


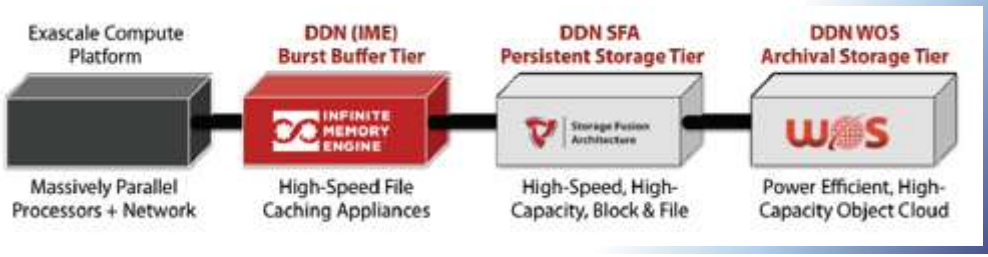



# Burst Buffer

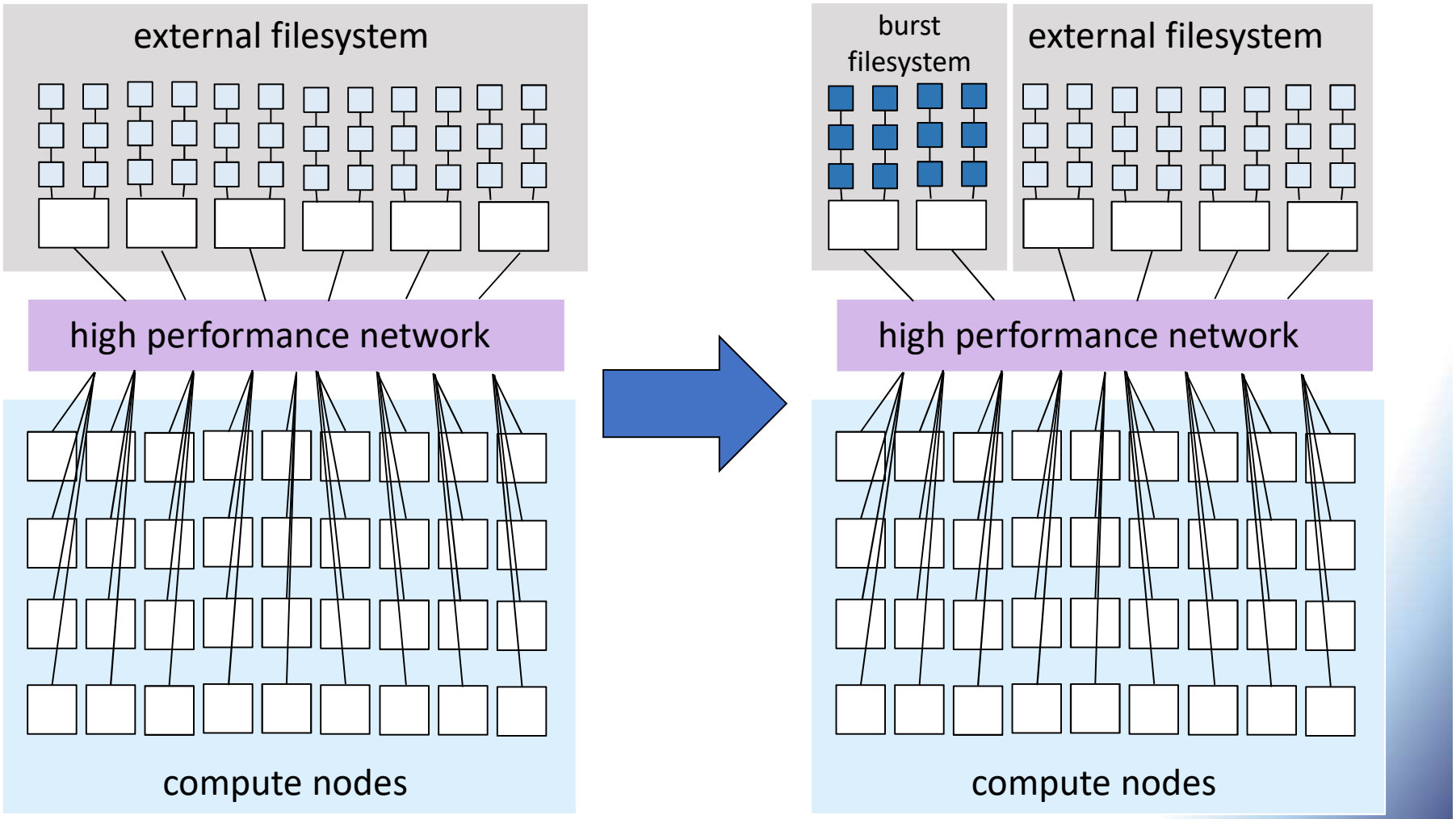
- Non-volatile already becoming part of HPC hardware stack
- SSDs offer high I/O performance but at a cost
  - How to utilise in large scale systems?
- Burst-buffer hardware accelerating parallel filesystem
  - Cray DataWarp
  - DDN IME (Infinite Memory Engine)



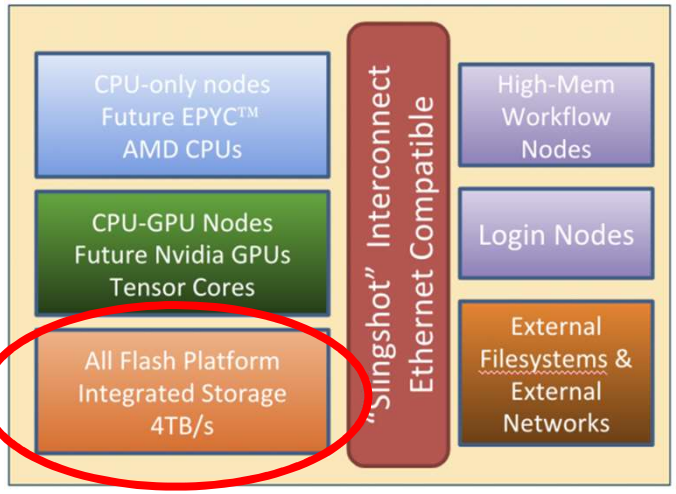



# Burst buffer



# Future storage

## Perlmutter

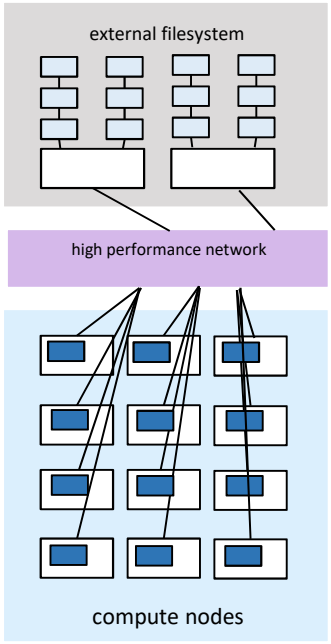



All-flash scratch filesystem

- 30-petabyte Lustre filesystem
- 4 TB/sec

# Moving beyond burst buffer

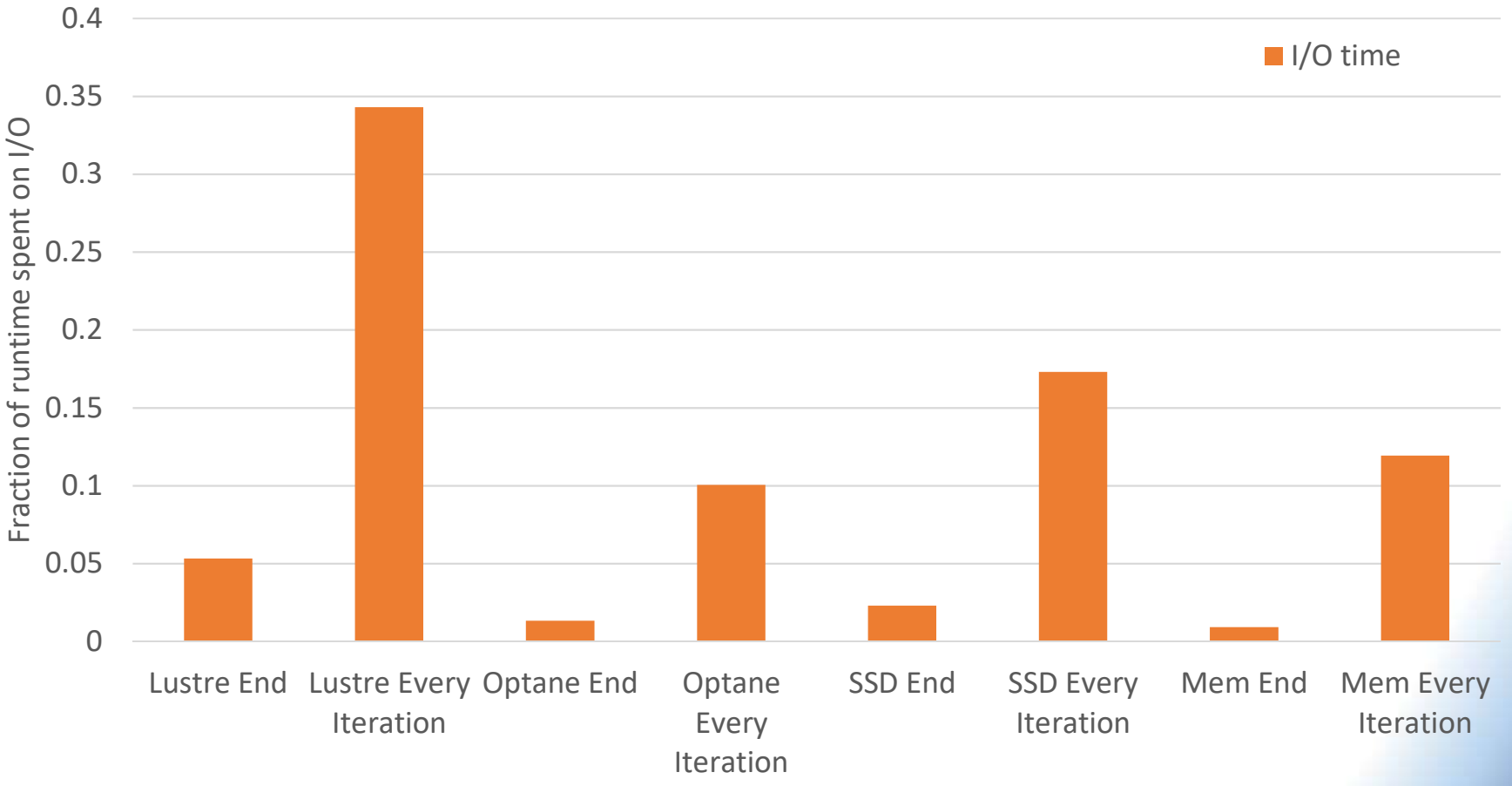
- Storage is moving to the node rather than the filesystem
- Argonne Theta machine has 128GB SSD in each compute node




# Moving beyond burst buffer

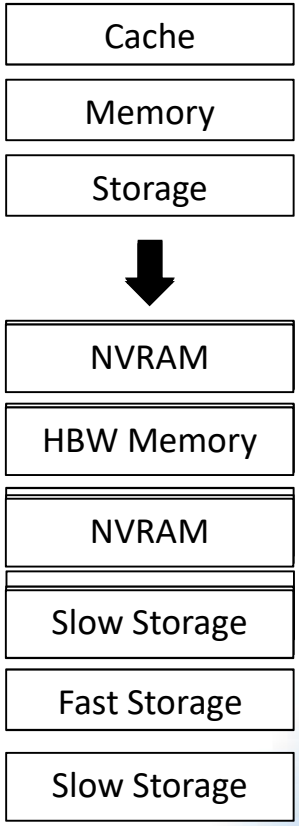

- Aurora will feature next generation Intel DPCMM

# Enabling new I/O



# New Memory Hierarchies


- High bandwidth, on processor memory
  - Large, high bandwidth cache
  - Latency cost for individual access may be an issue
- Main memory
  - DRAM
  - Costly in terms of energy, potential for lower latencies than high bandwidth memory
- Byte-addressable Persistent Memory
  - High capacity, ultra fast storage
  - Low energy (when at rest) but still slower than DRAM
  - Available through same memory controller as main memory, programs have access to memory address space

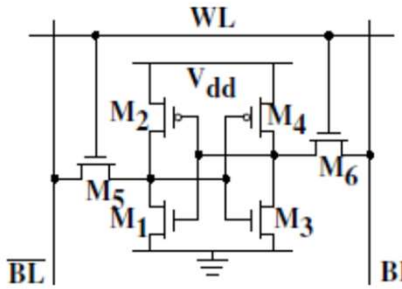


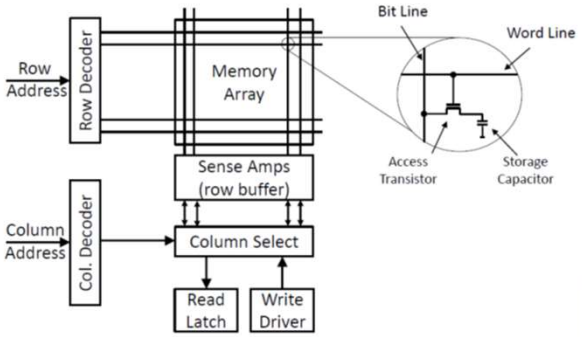
# Non-volatile memory

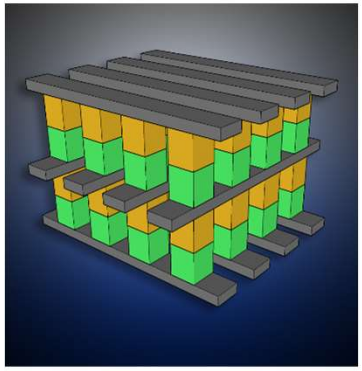

- Non-volatile RAM
  - Intel DCPMM technology
  - STT-RAM
- Much larger capacity than DRAM
  - Hosted in the DRAM slots, controlled by a standard memory controller
- Slower than DRAM by a small factor, but significantly faster than SSDs
- STT-RAM
  - Read fast and low energy
  - Write slow and high energy
    - Trade off between durability and performance
    - Can sacrifice data persistence for faster writes

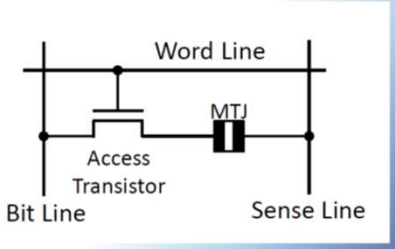
# SRAM vs NVRAM

- SRAM used for cache
- High performance but costly
  - Die area
  - Energy leakage
- DRAM lower cost but lower performance
  - Higher power/refresh requirement
- NVRAM technologies offer
  - Much smaller implementation area
  - No refresh/ no/low energy leakage
  - Independent read/write cycles
- NVDIMM offers
  - Persistency
  - Direct access (DAX)



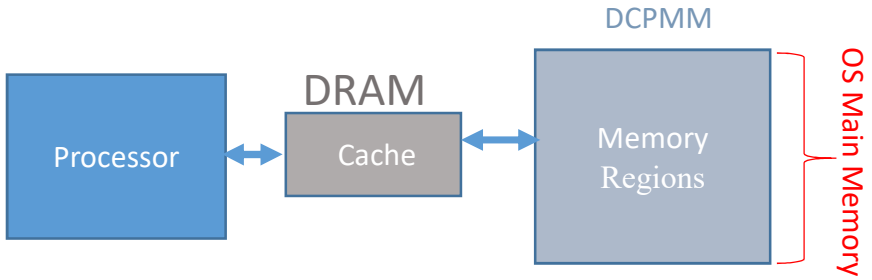



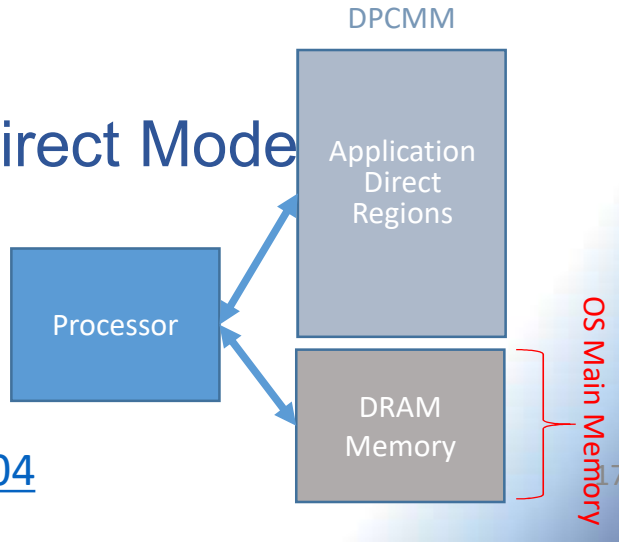


# Memory levels

- Intel DCPMM has different memory modes* (like MCDRAM on KNL):
  - Two-level memory (2LM) (Memory Mode)




  - One-level memory (1LM) (App Direct Mode

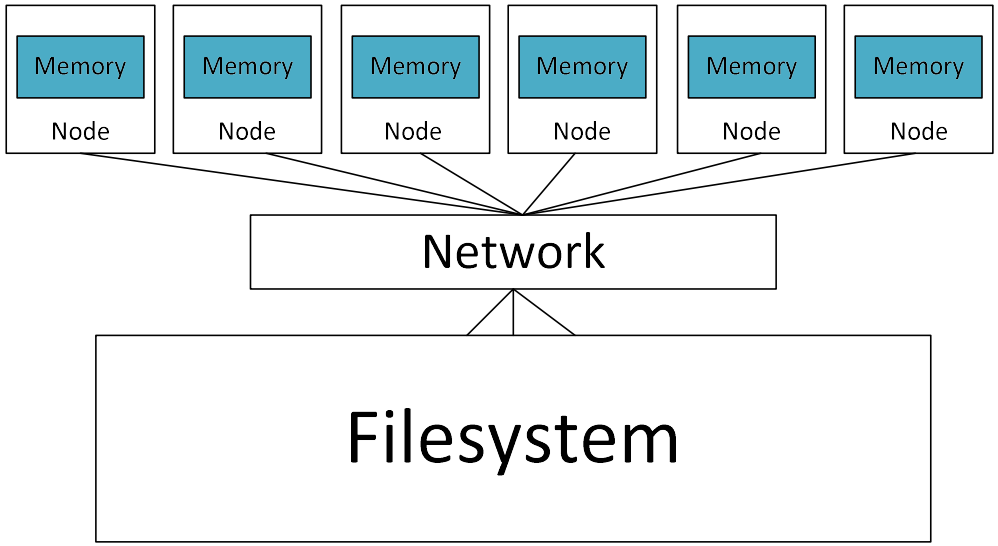


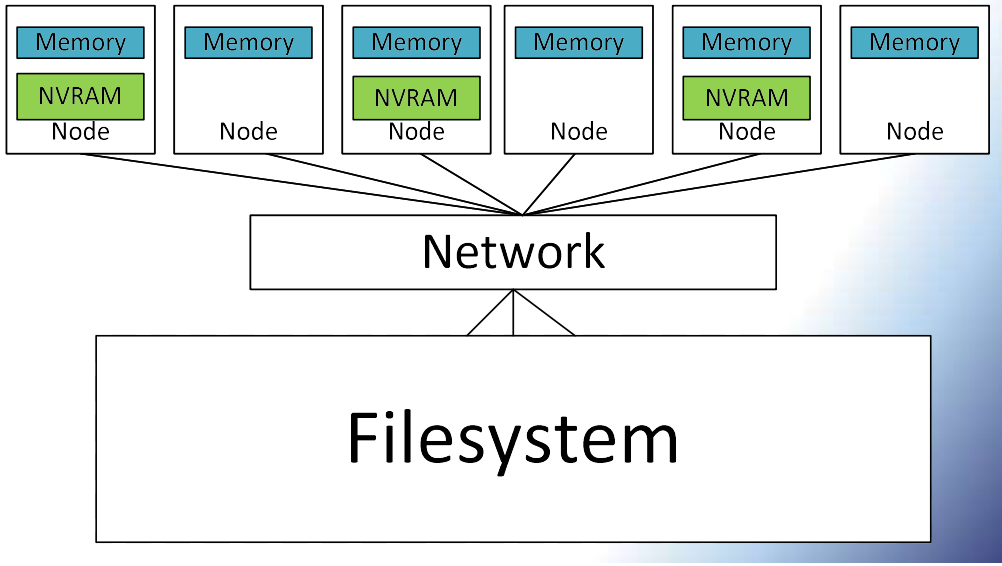

*https://www.google.com/patents/US20150178204

# Intel DCPMM


- The “memory” usage model allows for the extension of the main memory
  - The data is volatile like normal DRAM based main memory
- The “storage” usage model which supports the use of NVRAM like a classic block device
  - E.g. like a very fast SSD
- The “application direct” (DAX) usage model maps persistent storage from the NVRAM directly into the main memory address space
  - Direct CPU load/store instructions for persistent main memory regions

# Exploiting distributed storage

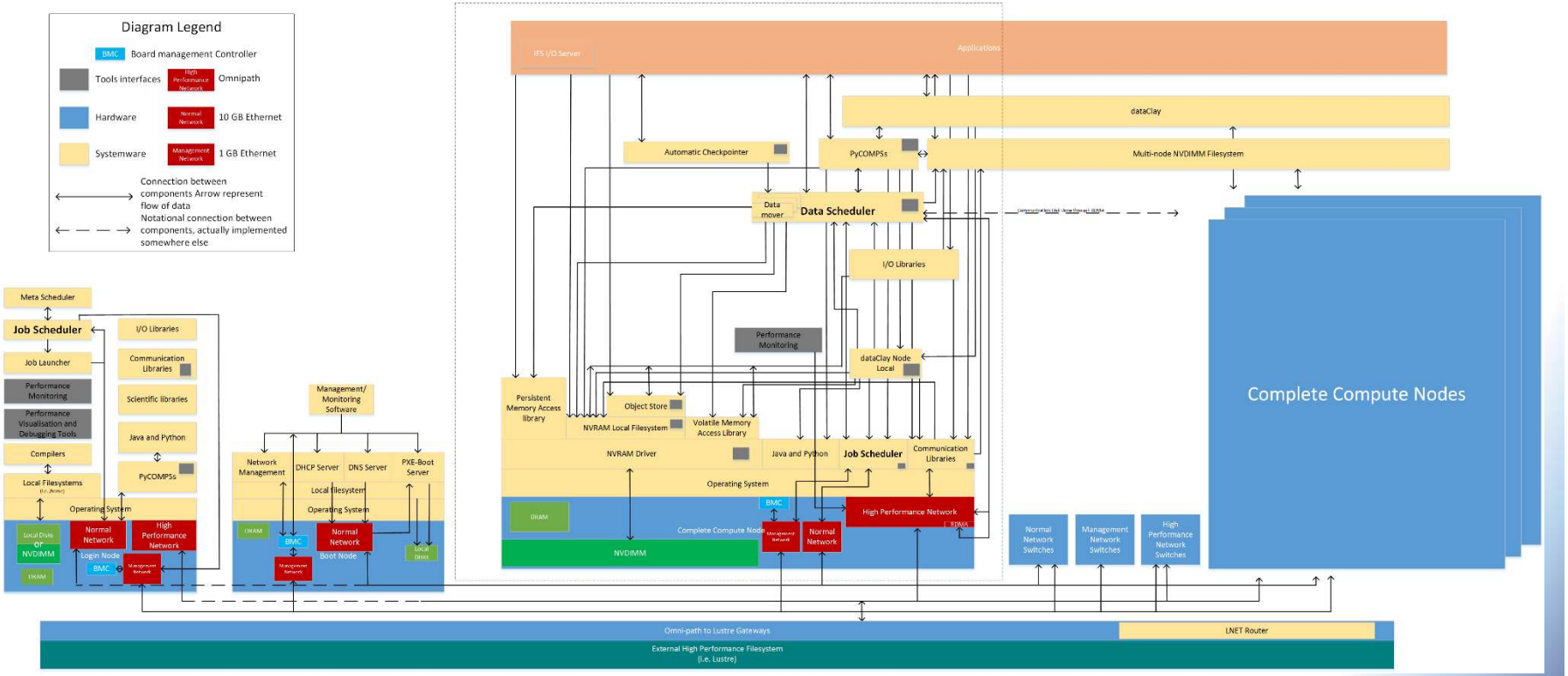


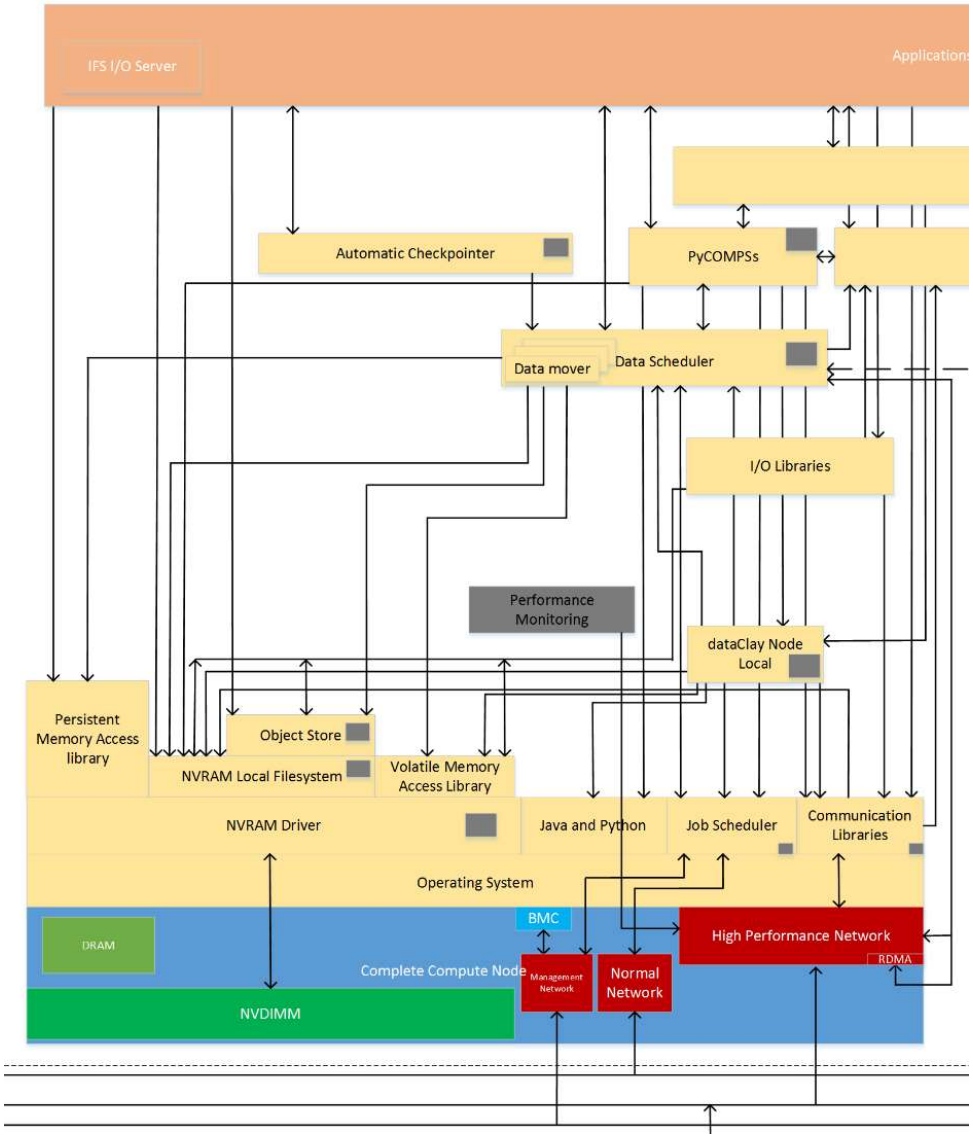
# The challenge of distributed storage


- Enabling all the use cases in multi-user, multi-job environment is the real challenge
  - Heterogeneous scheduling mix
  - Different requirements on the B-APM
  - Scheduling across these resources
  - Enabling sharing of nodes
  - Not impacting on node compute performance
  - etc….
- Enabling applications to do more I/O
  - Large numbers of our applications don't heavily use I/O at the moment
  - What can we enable if I/O is significantly cheaper

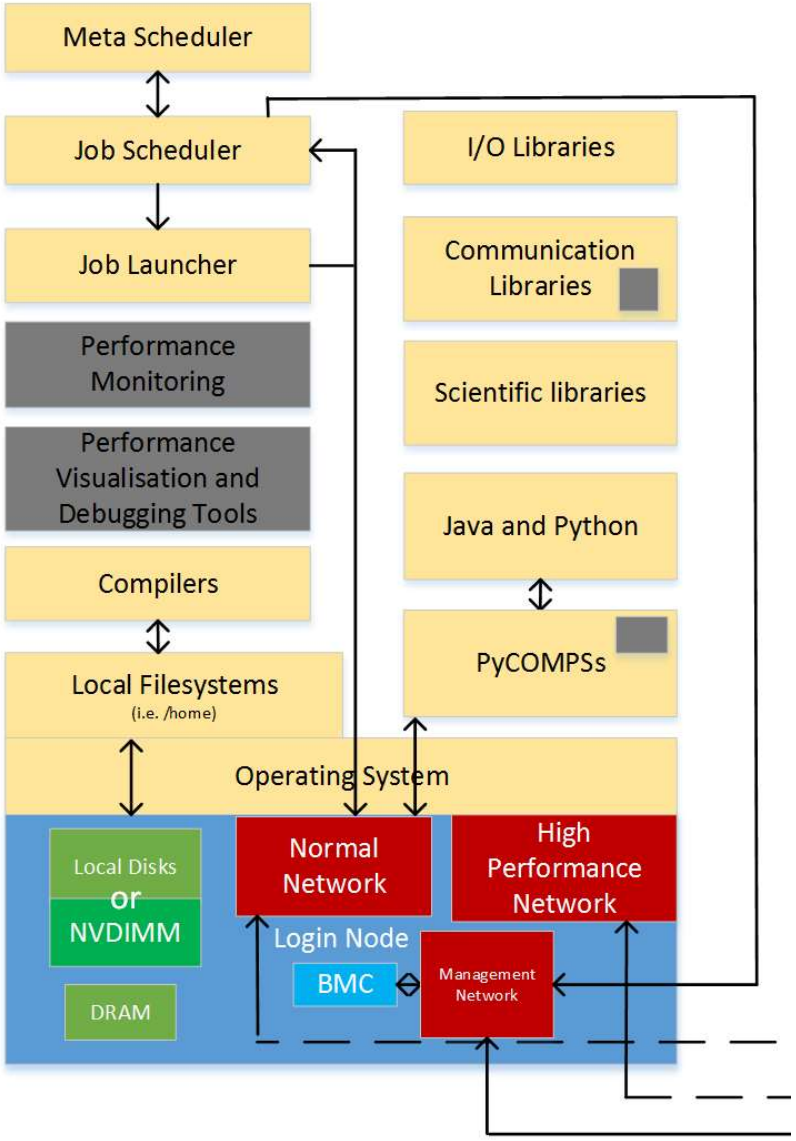
# NEXTGenIO Systemware



# Compute node systemware

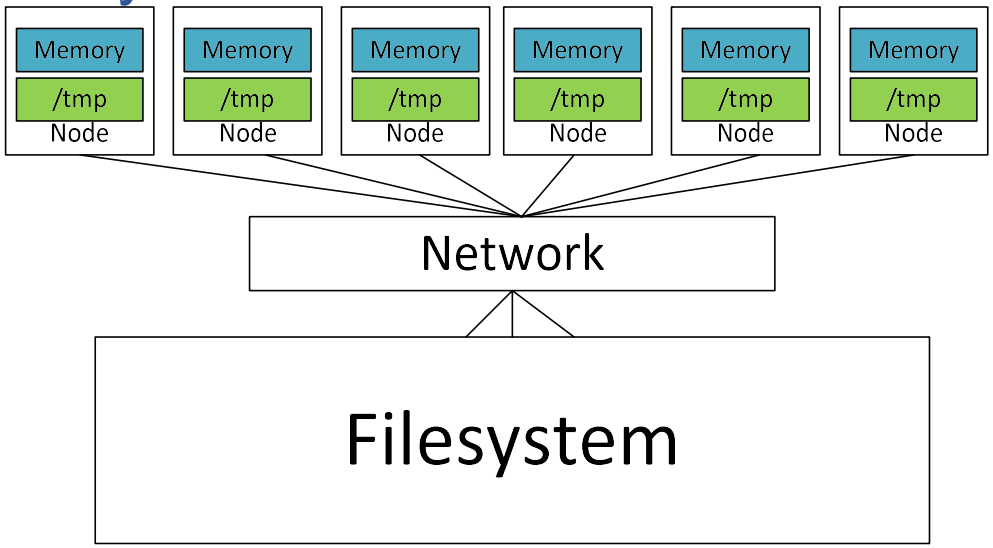


# User node systemware



# Using distributed storage

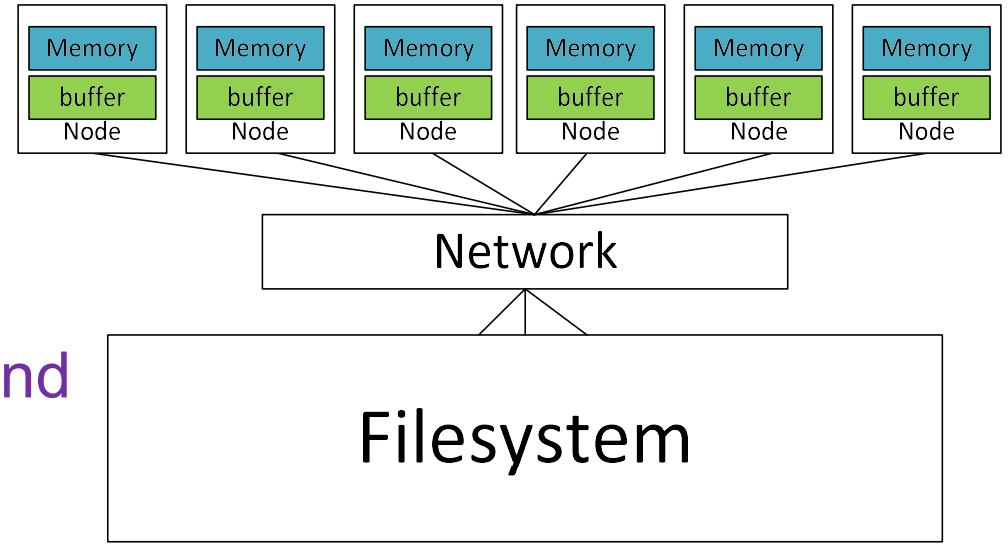

- Without changing applications
  - Large memory space/in-memory database etc…
  - Local filesystem




  - Users manage data themselves
  - No global data access/namespace, large number of files
  - Still require global filesystem for persistence

# Using distributed storage

- Without changing applications
  - Filesystem buffer

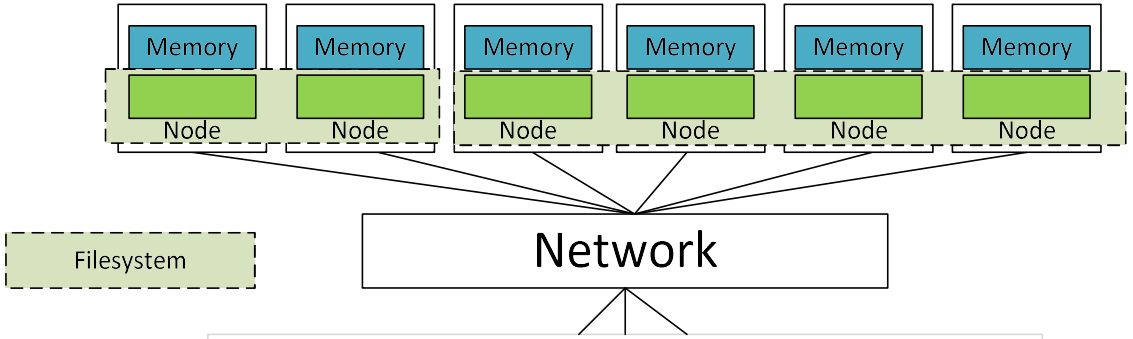



NGIO Data Scheduler (NORNS) and Slurm integration

- Pre-load data into NVRAM from filesystem
- Use NVRAM for I/O and write data back to filesystem at the end
- Requires systemware to preload and postmove data
- Uses filesystem as namespace manager

# Using distributed storage


- Without changing applications
  - Global filesystem



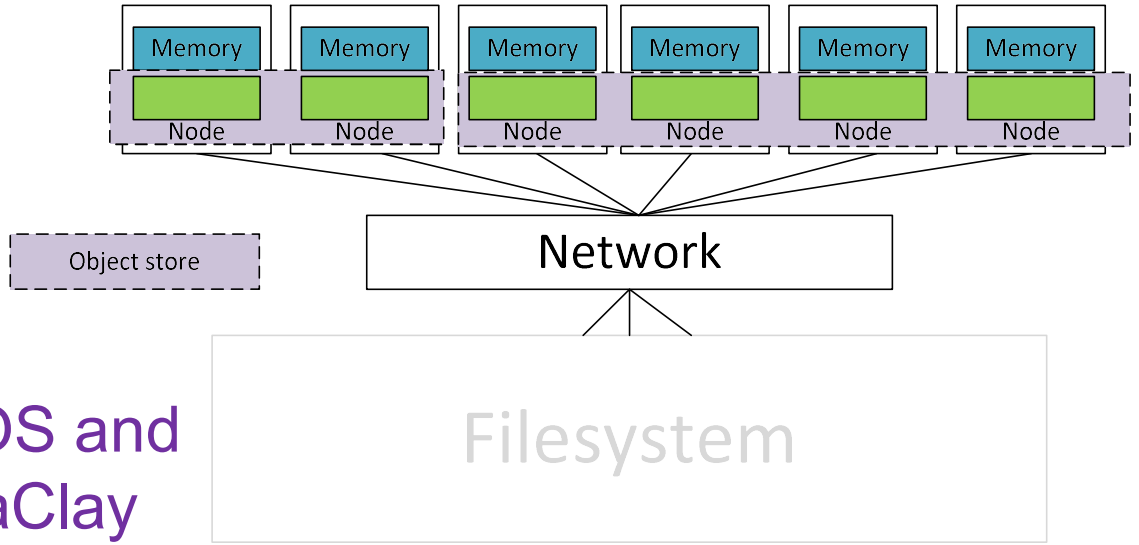

NGIO GekkoFS

Filesystem

- Requires functionality to create and tear down global filesystems for individual jobs
- Requires filesystem that works across nodes
- Requires functionality to preload and postmove filesystems
- Need to be able to support multiple filesystems across system

# Using distributed storage


- With changes to applications
  - Object store

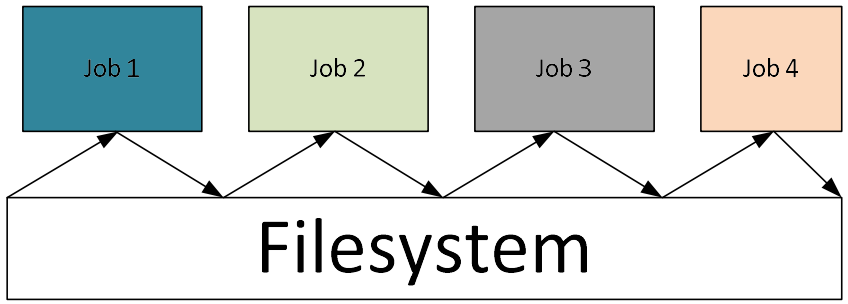



Intel DAOS and
BSC dataClay

- Needs same functionality as global filesystem
- Removes need for POSIX, or POSIX-like functionality

# Using distributed storage

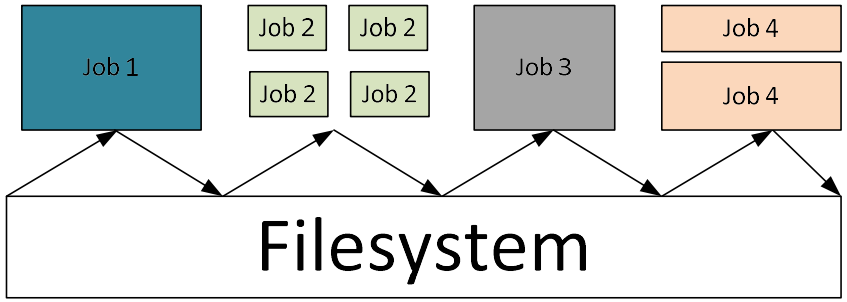

- New usage models
  - Resident data sets
    - Sharing preloaded data across a range of jobs
    - Data analytic workflows
    - How to control access/authorisation/security/etc….?
  - Workflows
    - Producer-consumer model




    - Remove filesystem from intermediate stages

# Using distributed storage

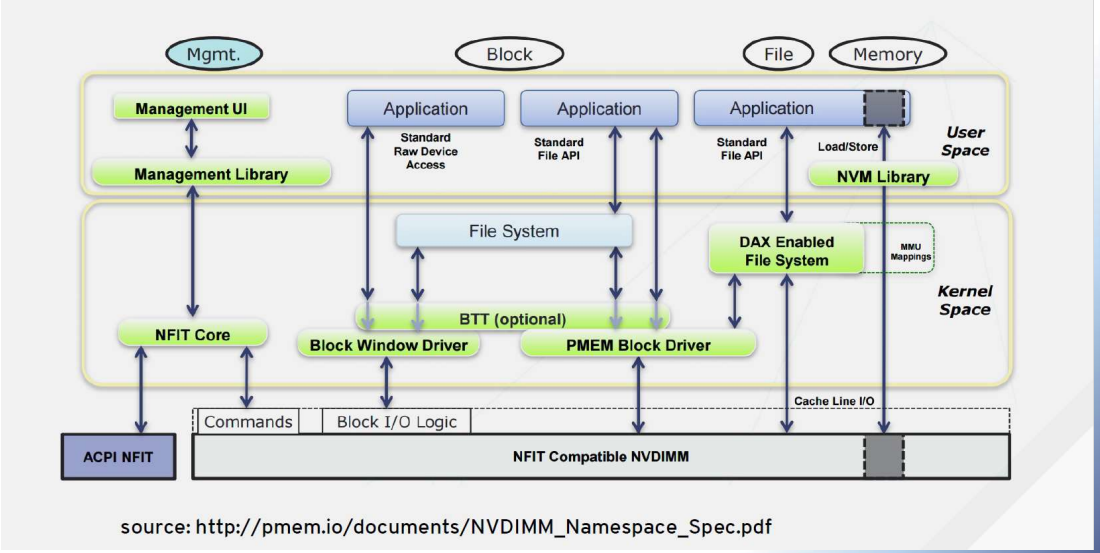

- Workflows
  - How to enable different sized applications?




  - How to schedule these jobs fairly?
  - How to enable secure access?

# Programming DCPMM

- Block memory mode
  - Standard filesystem api's
  - Will incur block mode overheads (not byte granularity, kernel interrupts, etc…)
- App Direct/DAX mode
  - Volatile memory access can use standard load/store
  - PMDK
    - pmem.io
    - Persistent load/store
    - memory mapped file like functionality

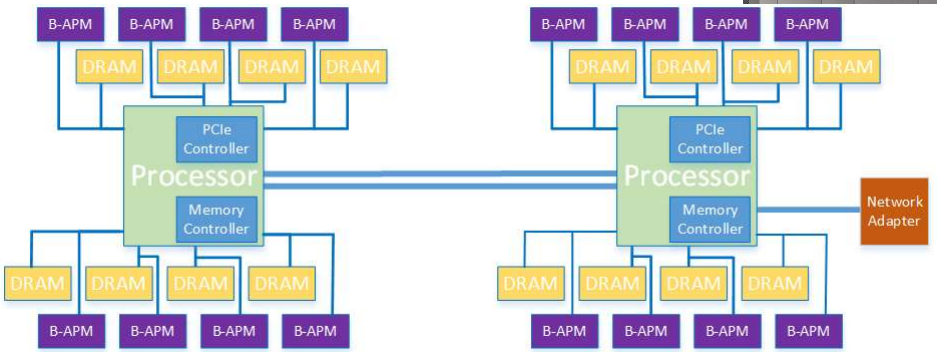


# Potential solutions


- Tiered memory performance/programming
- Large memory space
- Burst buffer
- Filesystem across NVRAM in nodes
- HSM functionality
- Object store across nodes
- Checkpointing and I/O libraries

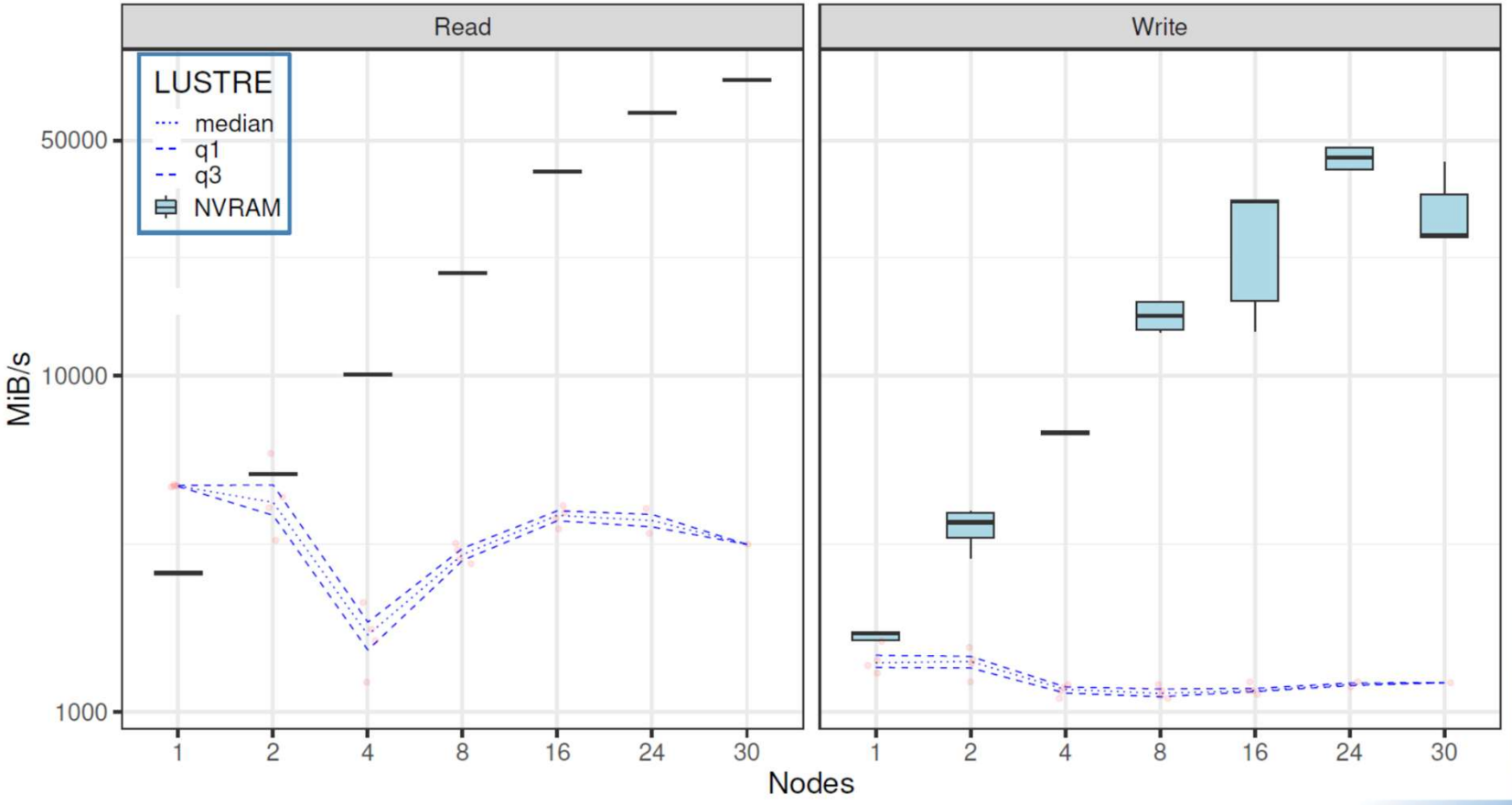
# NGIO Prototype

- 34 node cluster with 3TB of Intel DCPMM per node
  - 2 CPUS per node, each with 1.5TB of DCPMM and 96GB of DRAM
- External Lustre filesystem

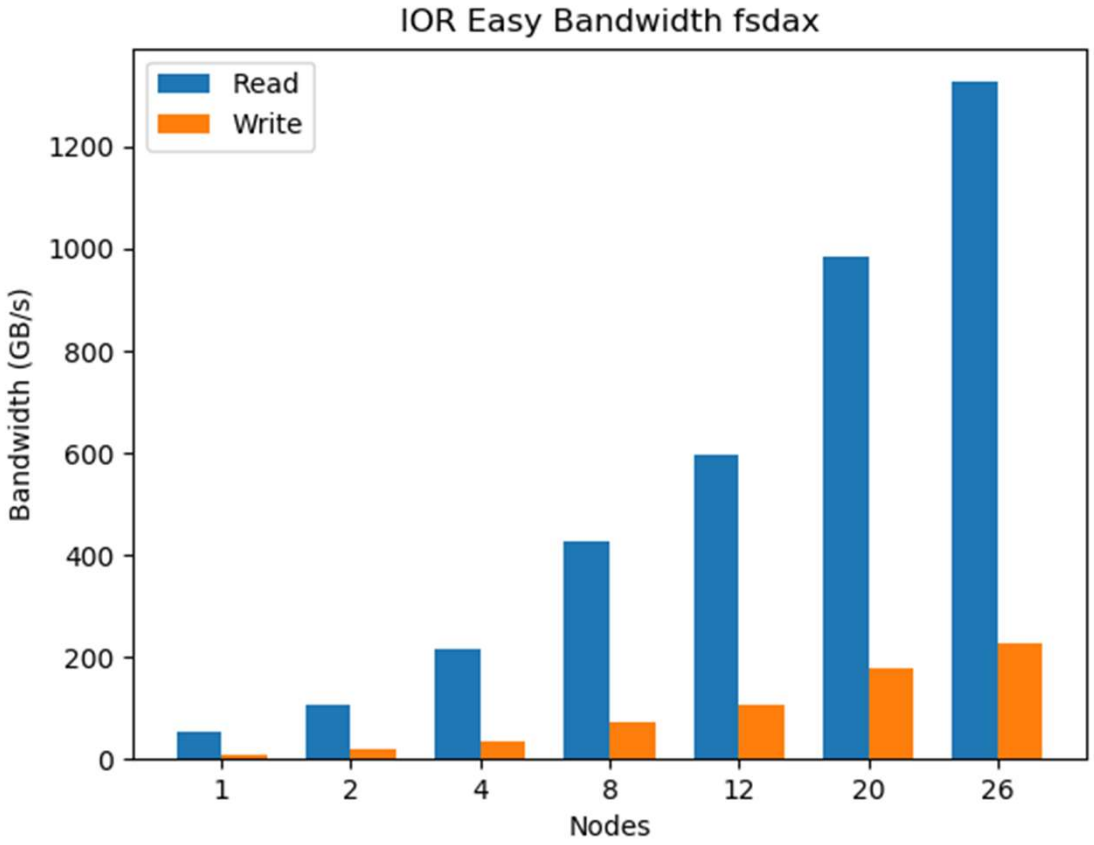


# Performance – IOR Easy

- File per process

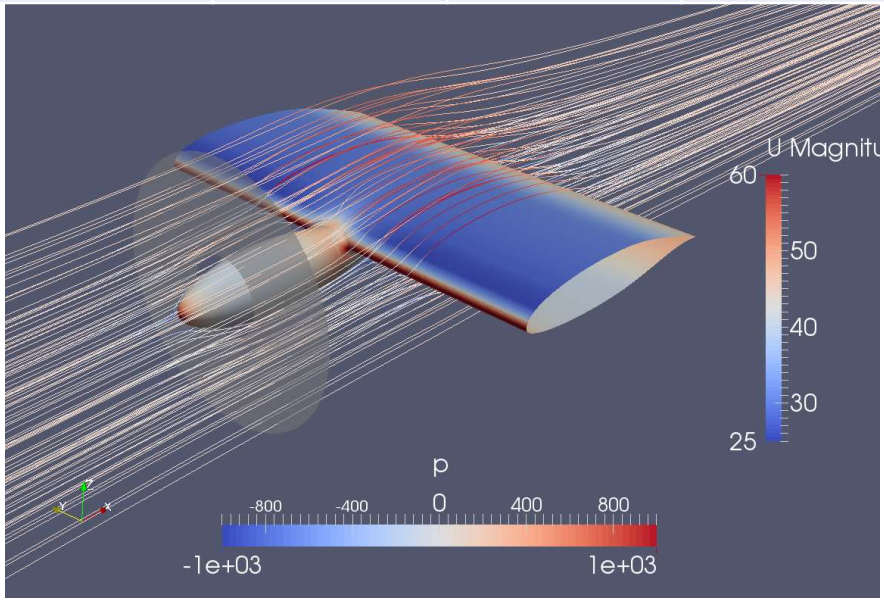


# Performance – IOR Easy



# Performance - workflows

| Performance benefits of data staging on OpenFOAM workflow | | | | | | |
|---|---|---|---|---|---|---|
| | 16 nodes, 768 MPI procs | | | 20 nodes, 960 MPI procs | | |
| Stage | Lustre | NVM | Benefit | Lustre | NVM | Benefit |
| decomposition | 1191 secs | 1105 secs | – | 1841 secs | 1453 secs | – |
| data staging | – | 32 secs | – | – | 330 secs | – |
| solver | 123 secs | 66 secs | 46% faster | 664 secs | 78 secs | 88% faster |
| Total | 1314 secs | 1203 secs | 8% faster | 2505 secs | 1861 secs | 25% faster |



# Performance - workflows


SYNTHETIC WORKFLOW BENCHMARK USING LUSTRE AND/OR NVMS IN A COMPUTE NODE

| Component | Target | Runtime (seconds) |
|---|---|---|
| Producer | Lustre | 96 |
| Consumer | Lustre | 74 |
| Producer | NVM | 64 |
| Consumer | NVM | 30 |

SYNTHETIC WORKFLOW BENCHMARK WITH DATA STAGING

| Component | Runtime (seconds) |
|---|---|
| Producer | 64 |
| Consumer | 30 |
| HPCG stage out | 137 |
| HPCG stage in | 142 |
| HPCG no activity | 122 |

# Performance – IO-500


- Ten client nodes, GekkoFS filesystem
  - GekkoFS only using TCP/IP. Optimisations to be done to utilise the Omnipath network

```
[RESULT] BW   phase 1          ior_easy_write 43.908 GB/s : time 342.83 seconds
[RESULT] BW   phase 2          ior_hard_write 4.449 GB/s : time 305.85 seconds
[RESULT] BW   phase 3          ior_easy_read 28.391 GB/s : time 530.21 seconds
[RESULT] BW   phase 4          ior_hard_read 21.788 GB/s : time  62.46 seconds
[RESULT] IOPS phase 1          mdtest_easy_write1 799.460 kiops : time 271.16 seconds
[RESULT] IOPS phase 2          mdtest_hard_write 140.924 kiops : time 325.54 seconds
[RESULT] IOPS phase 3          find 606.030 kiops : time 866.72 seconds
[RESULT] IOPS phase 4          mdtest_easy_stat 1844.630 kiops : time 264.39 seconds
[RESULT] IOPS phase 5          mdtest_hard_stat 1773.000 kiops : time  29.92 seconds
[RESULT] IOPS phase 6          mdtest_easy_delete 904.720 kiops : time 549.28 seconds
[RESULT] IOPS phase 7          mdtest_hard_read 435.259 kiops : time 112.96 seconds
[RESULT] IOPS phase 8          mdtest_hard_delete 40.490 kiops : time 1122.59 seconds
[SCORE] Bandwidth 18.6447 GB/s : IOPS 546.993 kiops : TOTAL 100.988
```

# Performance IOR

- For comparison: Summit at ORNL

- **504** compute nodes, **2** MPI processes per node

| | | |
|---|---|---|
| [RESULT] BW phase 1 | ior_easy_write | **2158.700** GB/s : time 362.34 seconds |
| [RESULT] BW phase 2 | ior_hard_write | **0.572** GB/s : time 462.76 seconds |
| [RESULT] BW phase 3 | ior_easy_read | 1788.320 GB/s : time 437.39 seconds |
| [RESULT] BW phase 4 | ior_hard_read | **27.403** GB/s : time 9.66 seconds |
| [RESULT] IOPS phase 1 | mdtest_easy_write | 3071.260 kiops : time 352.36 seconds \| |
| [RESULT] IOPS phase 2 | mdtest_hard_write | 24.375 kiops : time 327.20 seconds |
| [RESULT] IOPS phase 3 | find | **21780.030** kiops : time 46.35 seconds |
| [RESULT] IOPS phase 4 | mdtest_easy_stat | **28769.400** kiops : time 52.52 seconds |
| [RESULT] IOPS phase 5 | mdtest_hard_stat | 560.886 kiops : time 19.36 seconds |
| [RESULT] IOPS phase 6 | mdtest_easy_delete | 2699.000 kiops : time 398.34 seconds |
| [RESULT] IOPS phase 7 | mdtest_hard_read | 15354.700 kiops : time 17.02 seconds |
| [RESULT] IOPS phase 8 | mdtest_hard_delete | 26.504 kiops : time 181.79 seconds |

[SCORE] Bandwidth **88.2049** GB/s : IOPS **1522.68** kiops : TOTAL **366.48**

# Performance - STREAM


| Mode | Min BW (GB/s) | Median BW (GB/s) | Max BW (GB/s) |
|---|---|---|---|
| App Direct (DRAM) | 142 | 150 | 155 |
| App Direct (DCPMM) | 32 | 32 | 32 |
| Memory mode | 144 | 146 | 147 |
| Memory mode | 12 | 12 | 12 |

# Summary


- B-APM is here
  - In-node persistent storage likely to come to (maybe some) HPC and HPDA systems shortly
  - Applications can program directly but….
  - …potentially systemware can handle functionality for applications, at least in transition period
- Interesting times
  - Convergence of HPC and HPDA (maybe)
  - Different data usage/memory access models may become more interesting
  - Certainly benefits for single usage machines, i.e. bioinformatics, weather and climate, etc…
- When used efficiently the performance of persistent memory can be very significant