Exploiting Non-volatile memory for HPC I/O

Adrian Jackson, a.Jackson@epcc.ed.ac.uk

EPCC, The University of Edinburgh

Warning!

- Terminology will be annoying:
  - NVDIMM
  - NVRAM
  - SCM
  - B-APM
  - …….

- My fault, but people argue which is the most appropriate
  - So using them all to annoy as many people as possible ☺

NEXTGenIO summary

Project

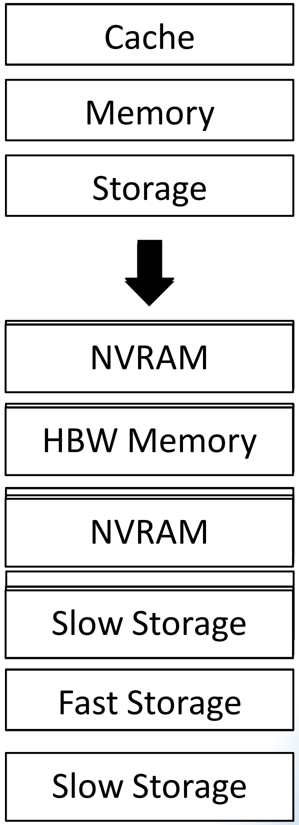
- Research & Innovation Action
- 36 month duration
- €8.1 million
- Approx. 50% committed to hardware development

Partners

- EPCC
- INTEL
- FUJITSU
- BSC
- TUD
- ALLINEA
- ECMWF
- ARCTUR

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

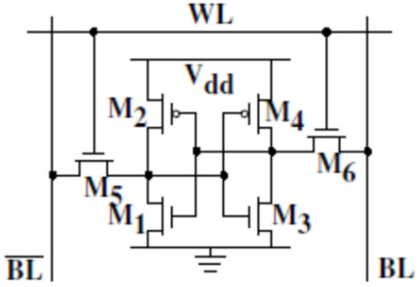


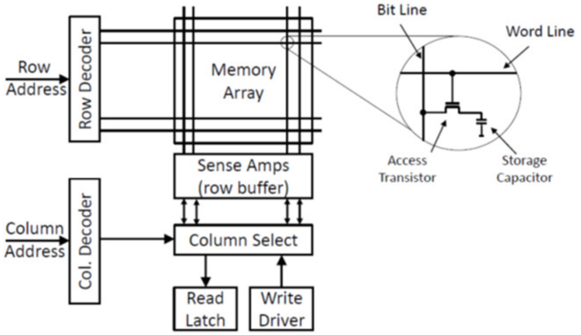
Non-volatile memory

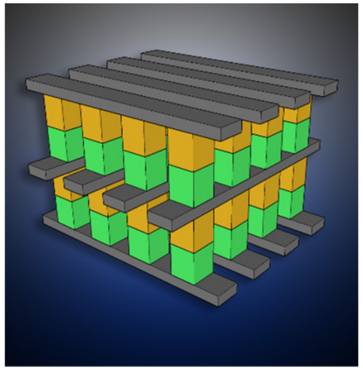
- Non-volatile RAM
  - 3D Xpoint (Intel/Micron)
  - STT-RAM
- Much larger capacity than DRAM
  - Hosted in the DRAM slots, controlled by a standard memory controller
- Slower than DRAM by a small factor, but significantly faster than SSDs
- STT-RAM
  - Read fast and low energy
  - Write slow and high energy
    - Trade off between durability and performance
    - Can sacrifice data persistence for faster writes

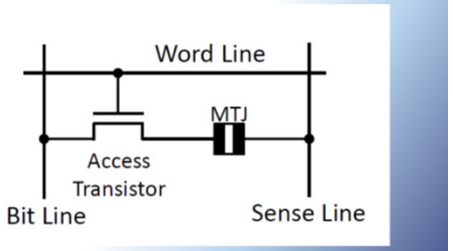
SRAM vs NVRAM

- SRAM used for cache
- High performance but costly
  - Die area
  - Energy leakage
- DRAM lower cost but lower performa
  - 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)







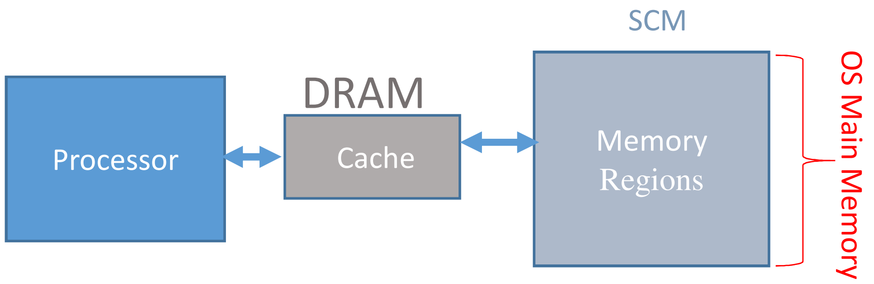


NVDIMMs

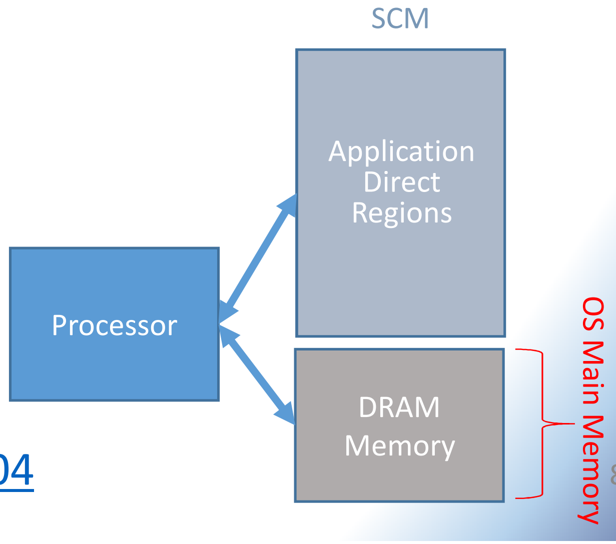
- Non-volatile memory already exists
  - NVDIMM-N:
    - DRAM with NAND Flash on board
    - External power source (i.e super capacitors)
    - Data automatically moved to flash on power failure with capacitor support, moved back when power restored
    - Persistence functionality with memory performance (and capacity)
  - NVDIMM-F:
    - NAND Flash in memory form
    - No DRAM
    - Accessed through block mode (like SSD)
  - NVDIMM-P:
    - Combination of N and F
    - Direct mapped DRAM and NAND Flash
    - Both block and direct memory access possible
- 3D Xpoint -> Intel Optane DC Persistent Memory
  - NVDIMM-P like (i.e. direct memory access and block)
  - But no DRAM on board
  - Likely to be paired with DRAM in the memory channel
  - Real differentiator (from NVDIMM-N) likely to be capacity and cost

Memory levels

- B-APM in general is likely to have different memory modes* (like MCDRAM on KNL):
  - Two-level memory (2LM)



  - One-level memory (1LM)



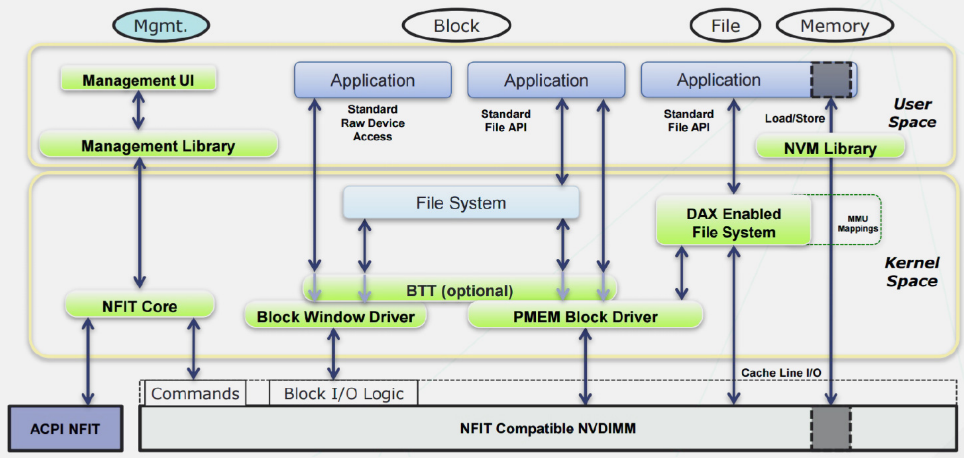
*https://www.google.com/patents/US20150178204

Byte-Addressable Persistent Memory

- The “memory” usage model allows for the extension of the main memory
  - The data is volatile like normal DRAM based main memory
  - Potential for very large memory spaces at reduced cost (capital and recurrent) compared to DRAM
- 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

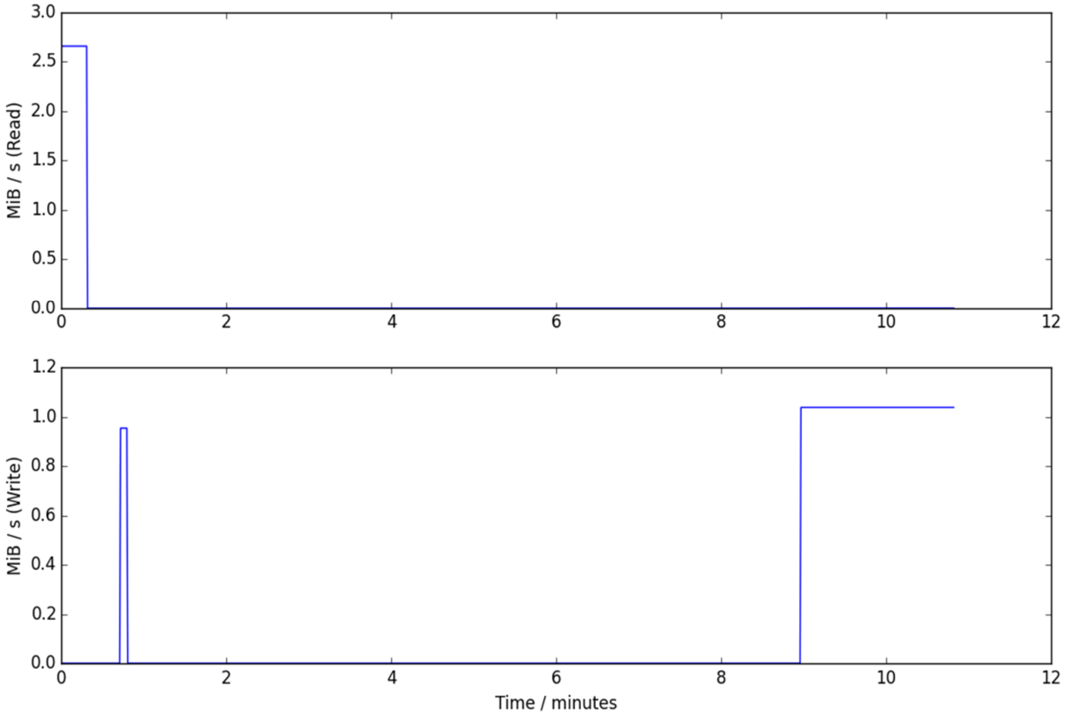
Programming B-APM

- 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
  - NVM library
    - pmem.io/PMDK
    - Persistent load/store
    - memory mapped file like functionality

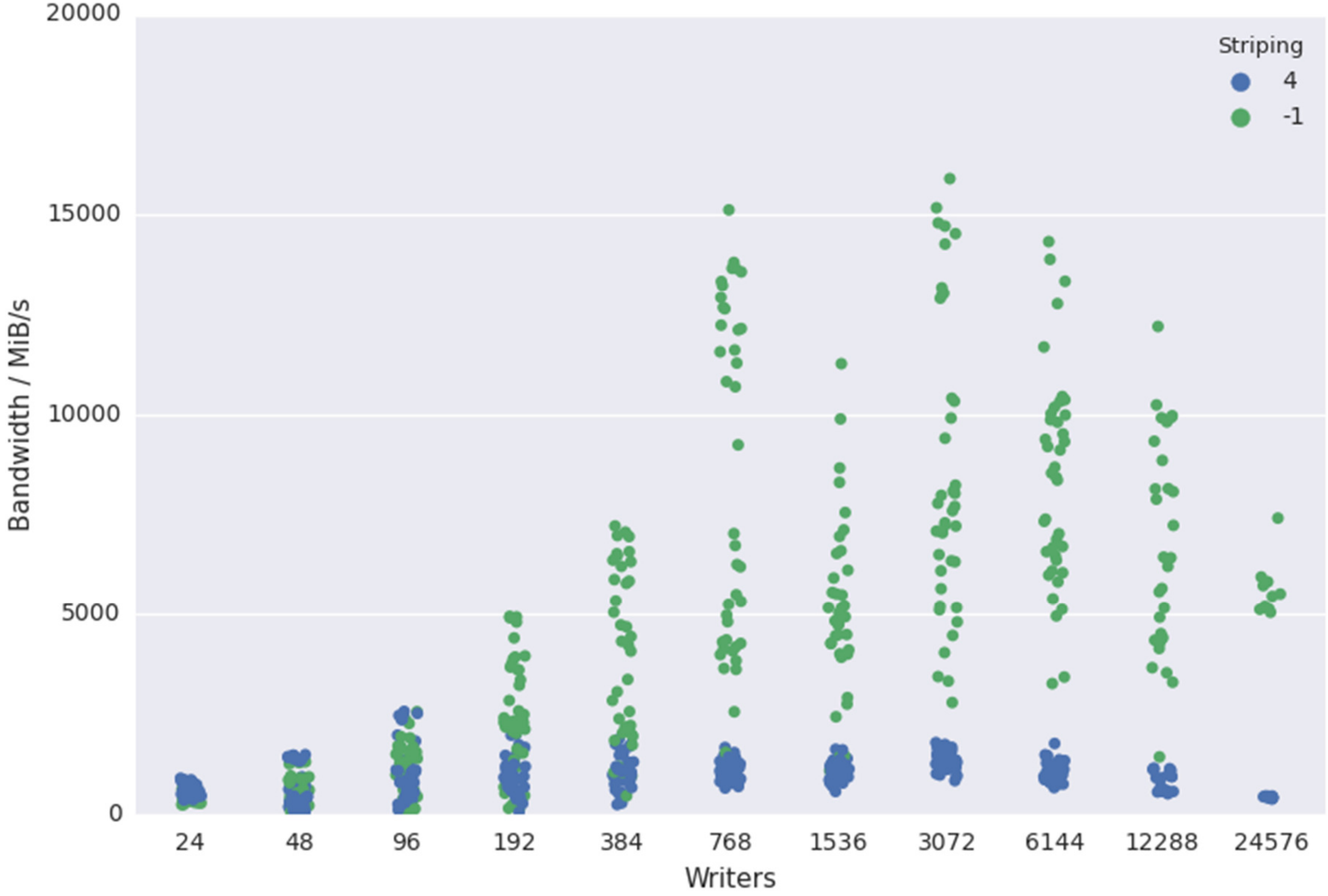


source: http://pmem.io/documents/NVDIMM_Namespace_Spec.pdf

I/O


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

I/O Performance

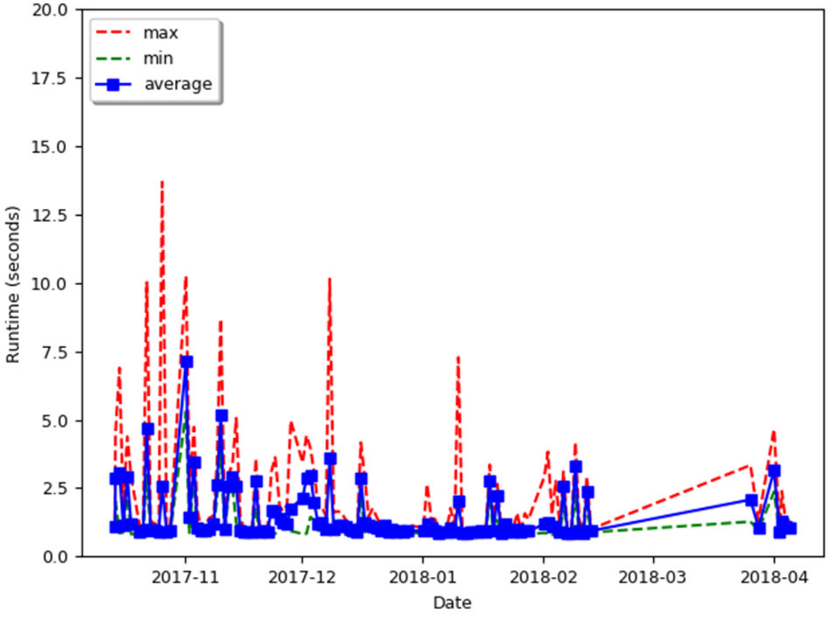


- https://www.archer.ac.uk/documentation/white-papers/parallelIO-benchmarking/ARCHER-Parallel-IO-1.0.pdf

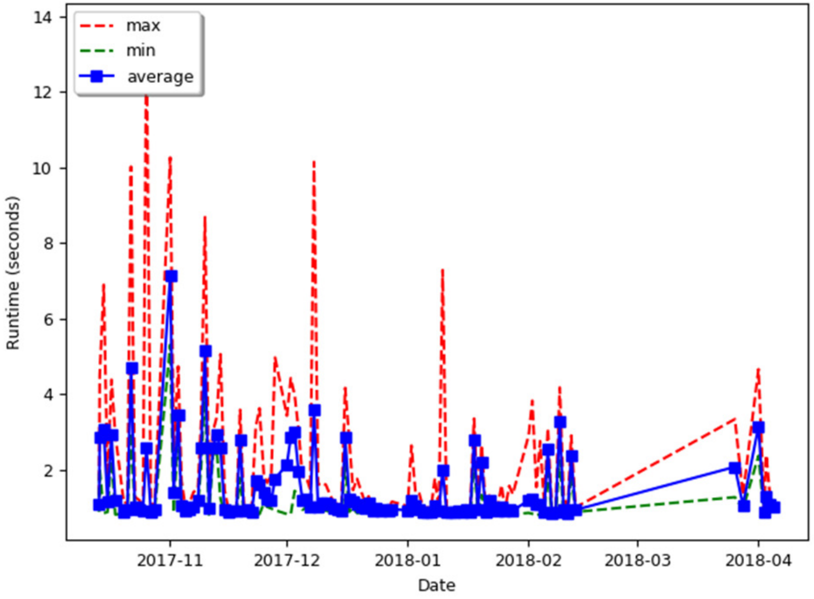
I/O Performance

MPI I/O 768 processes, 12GB data set

Lustre: 4 stripes



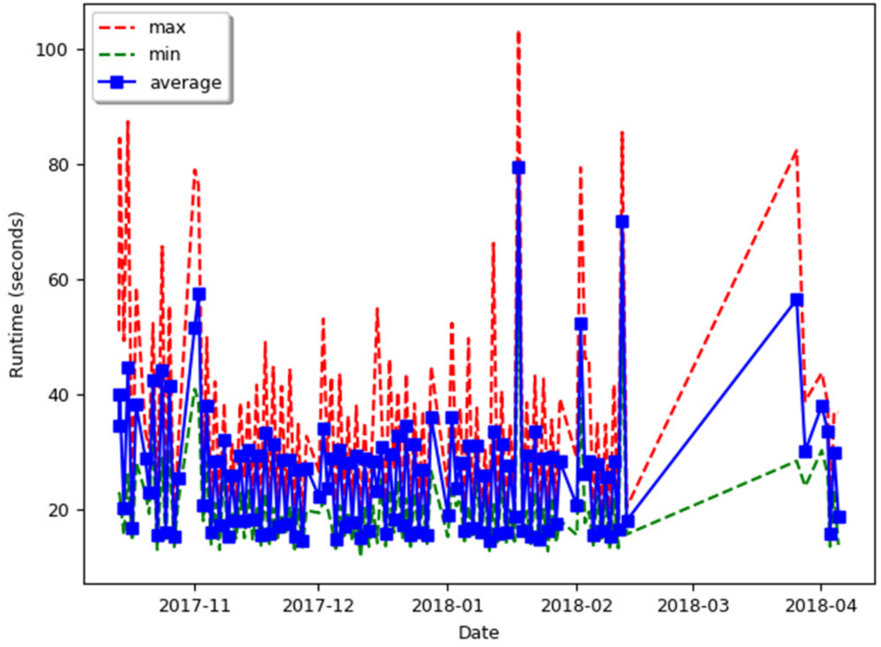
Lustre: max stripes



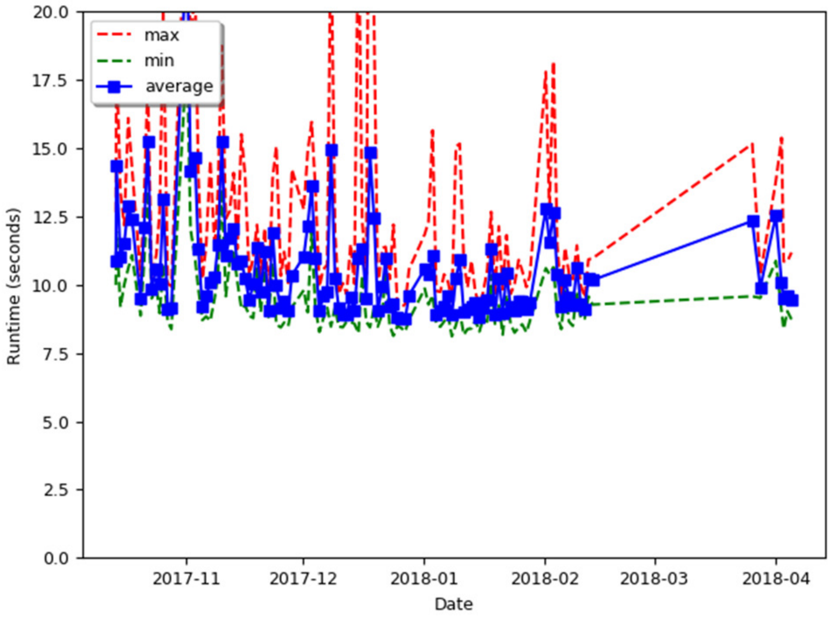
I/O Performance

HDF5 on MPI-I/O 768 processes, 12GB data set

Lustre: 4 stripes



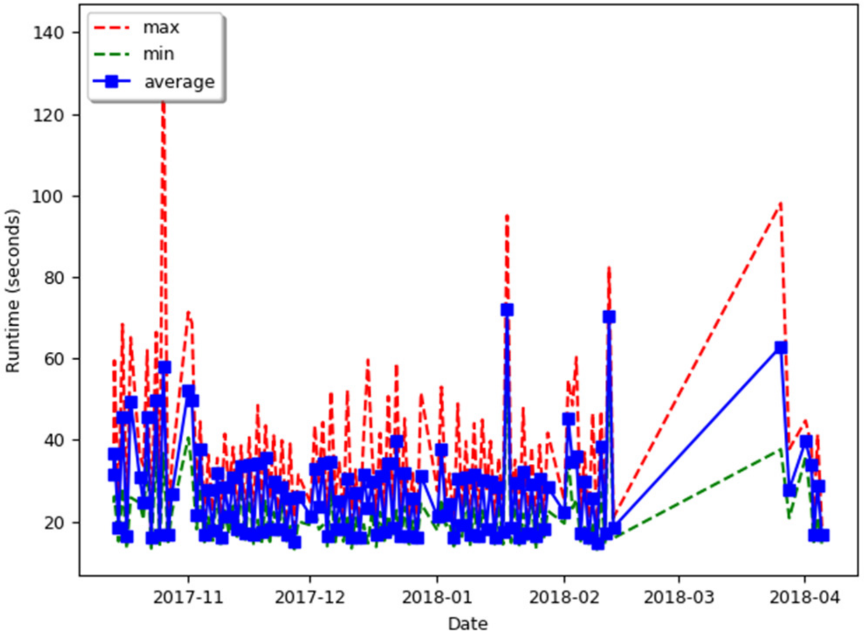
Lustre: max stripes



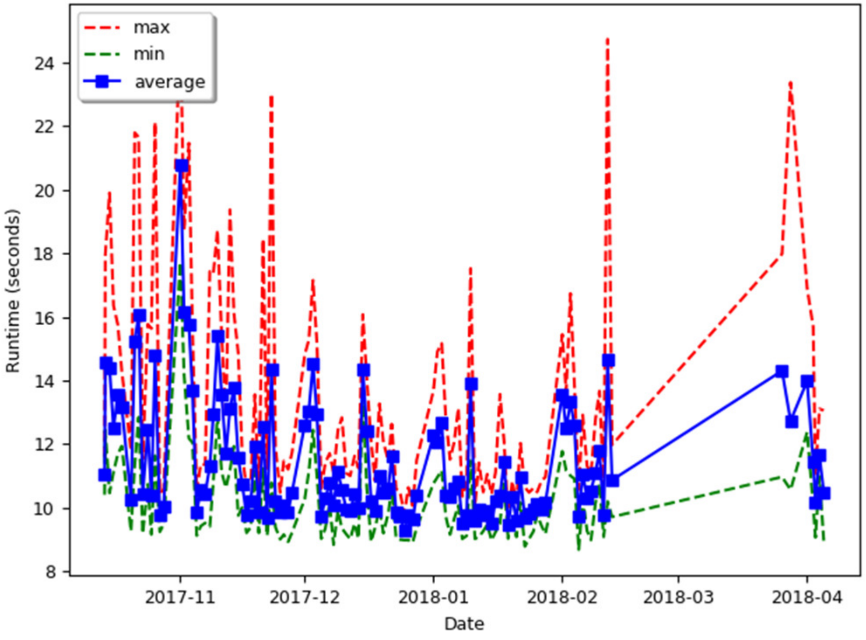
I/O Performance

NetCDF on HDF5 768 processes, 12GB data set

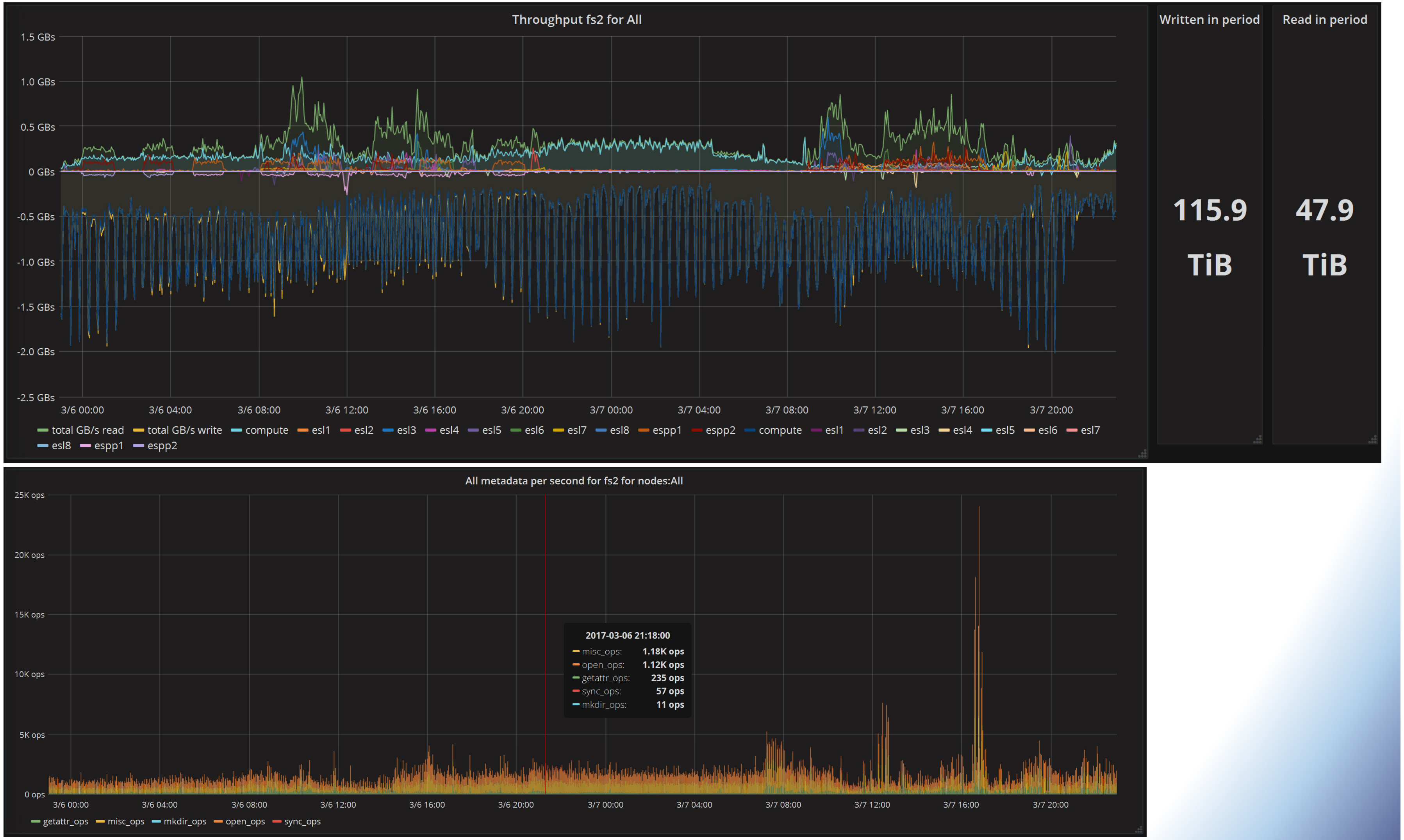
Lustre: 4 stripes



Lustre: max stripes



ARCHER workload



Written in period

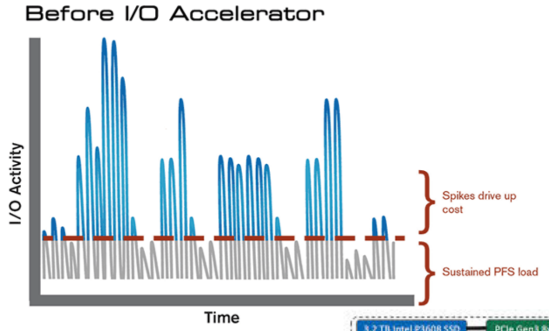
115.9 TiB

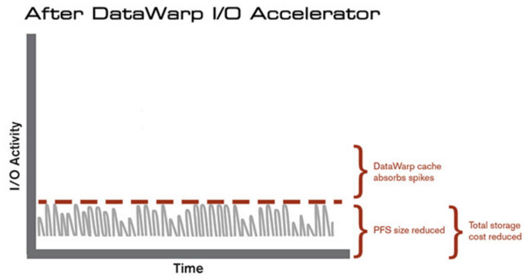
Read in period

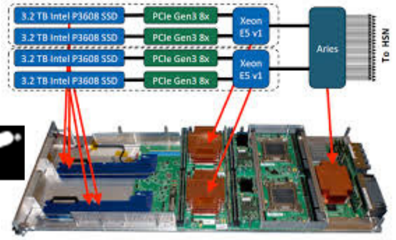
47.9 TiB

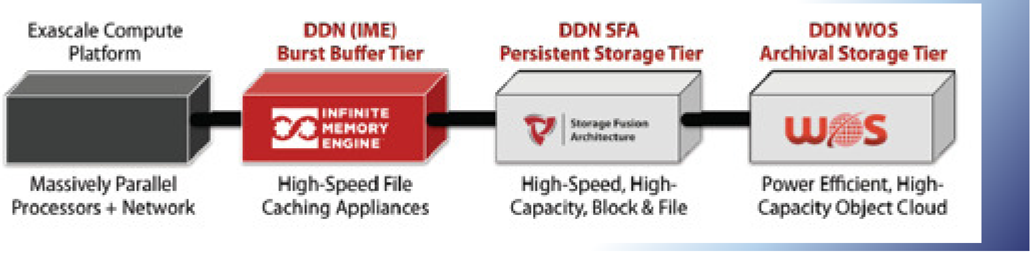
Burst Buffer

- 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)

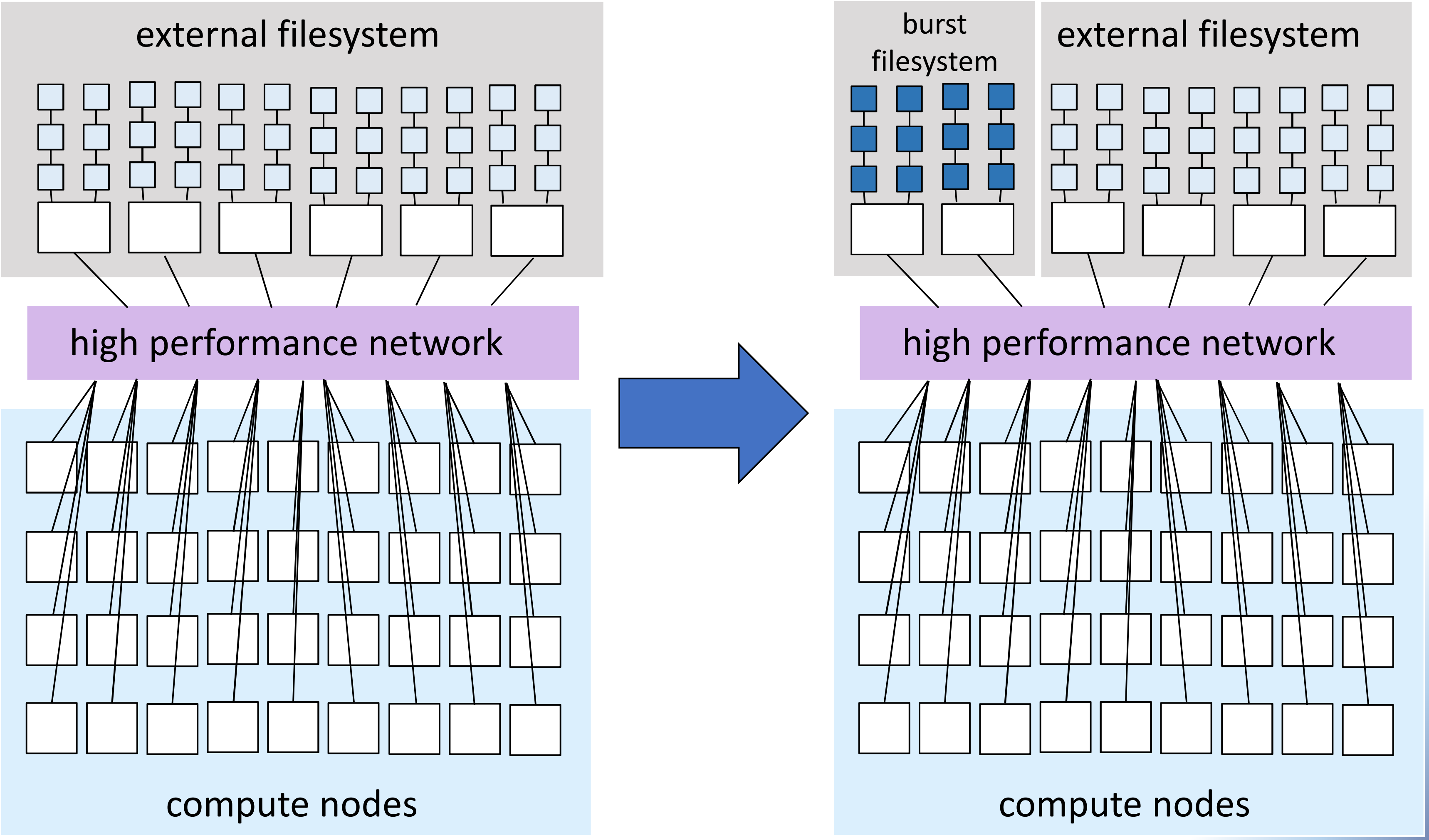




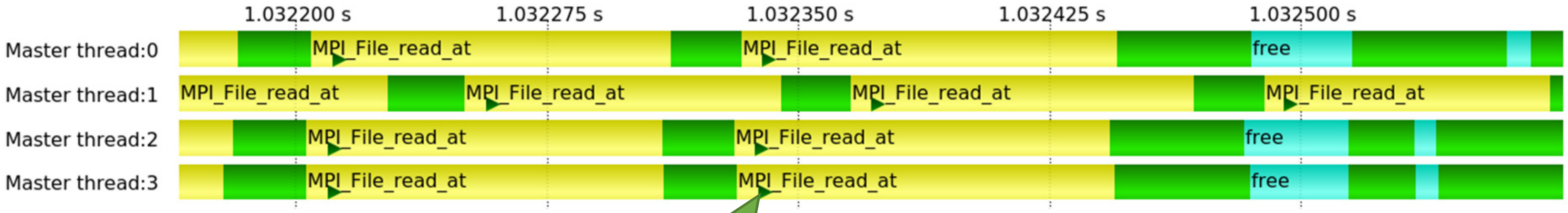



Exascale Compute Platform
DDN (IME) Burst Buffer Tier
DDN SFA Persistent Storage Tier
DDN WOS Archival Storage Tier
INFINITE MEMORY ENGINE
Storage Fusion Architecture
WOS
Massively Parallel Processors + Network
High-Speed File Caching Appliances
High-Speed, High-Capacity, Block & File
Power Efficient, High-Capacity Object Cloud

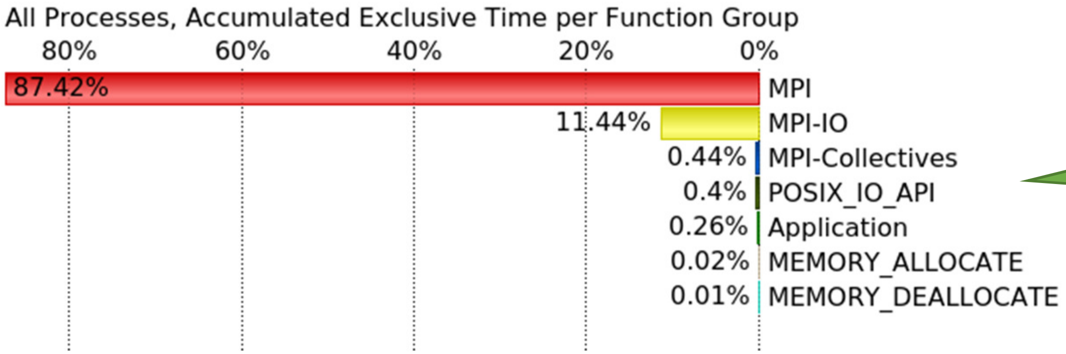
Burst buffer



I/O application patterns

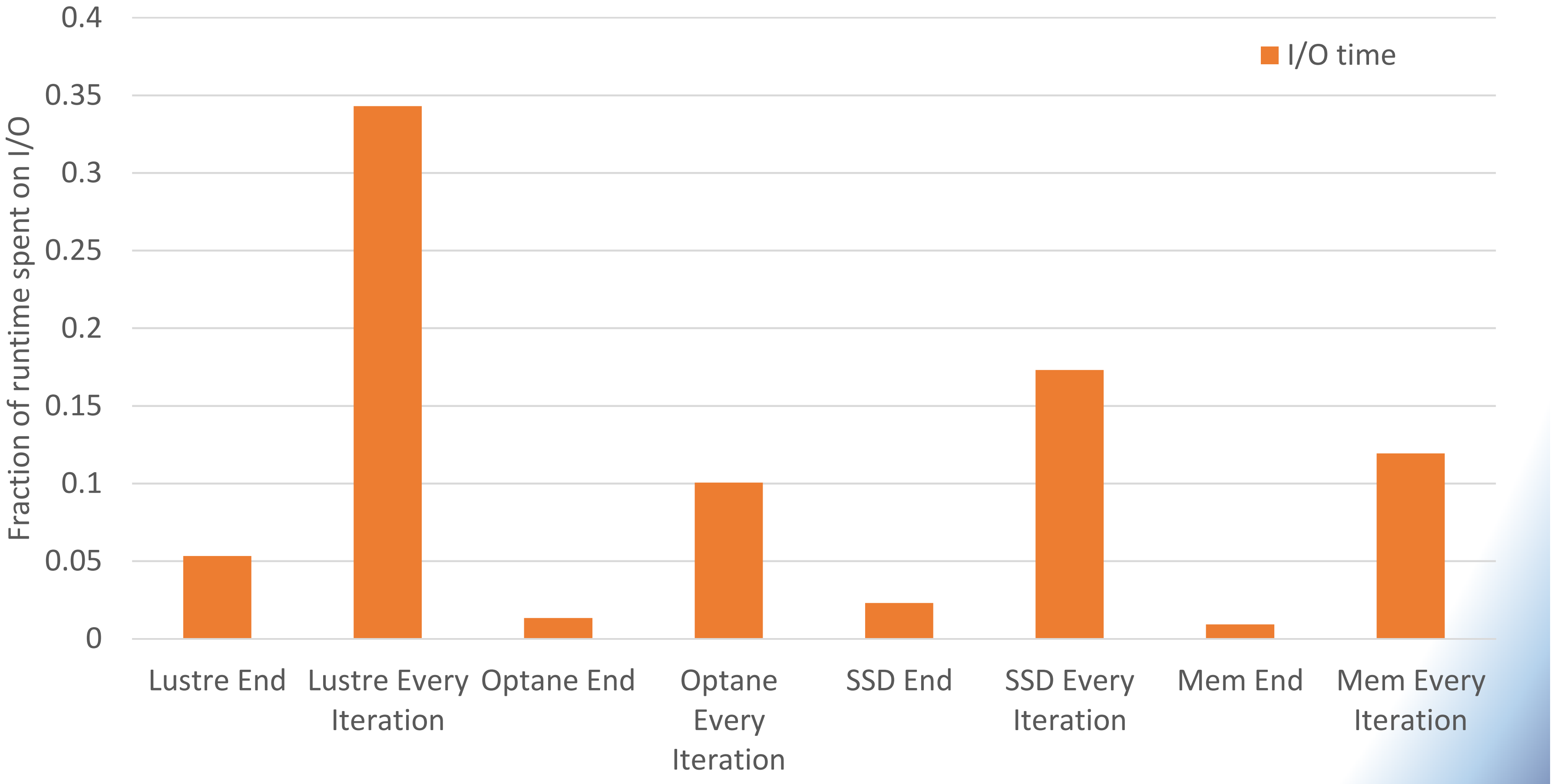


Individual I/O Operation

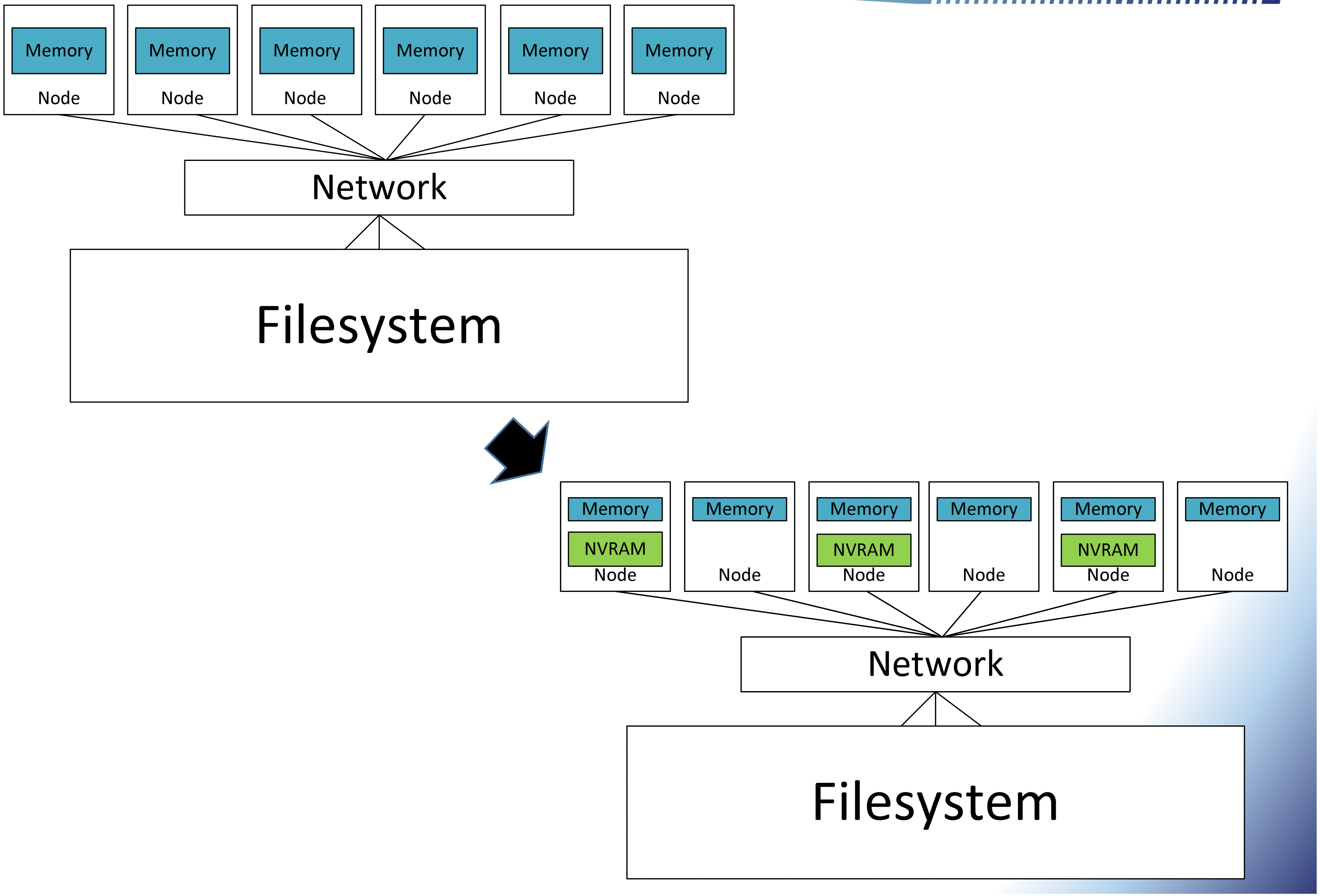


I/O Runtime Contribution

Enabling new I/O



Exploiting distributed storage



Memory Node | Memory Node | Memory Node | Memory Node | Memory Node | Memory Node

Network

Filesystem

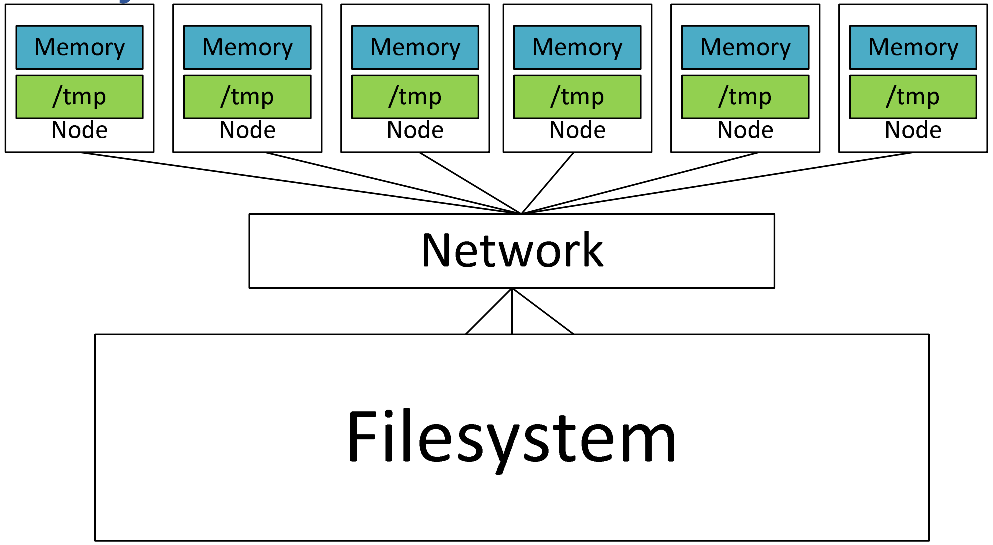
Memory NVRAM Node | Memory Node | Memory NVRAM Node | Memory Node | Memory NVRAM Node | Memory Node

Network

Filesystem

Using distributed storage

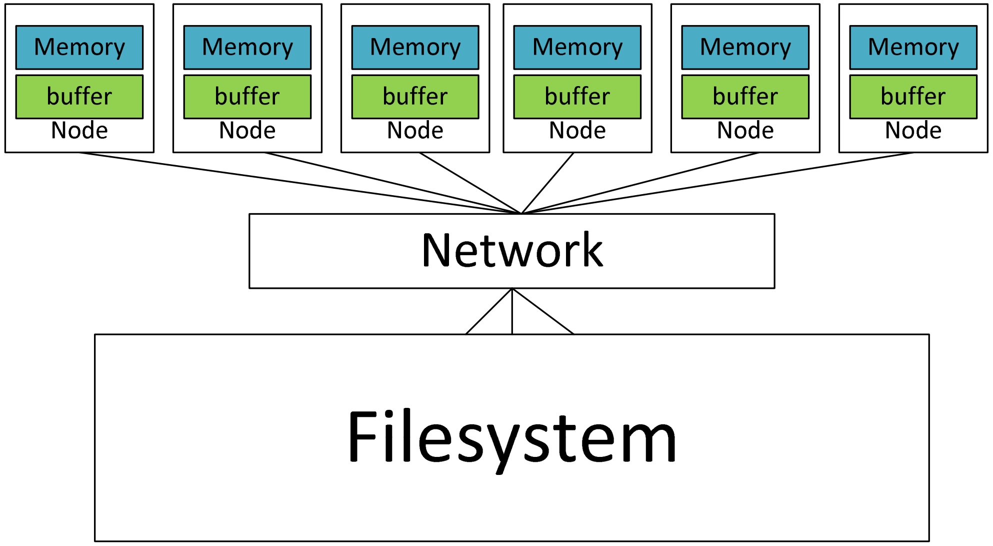
- Without changing applications
  - Large memory space/in-memory database etc…
  - Local filesystem



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

Using distributed storage

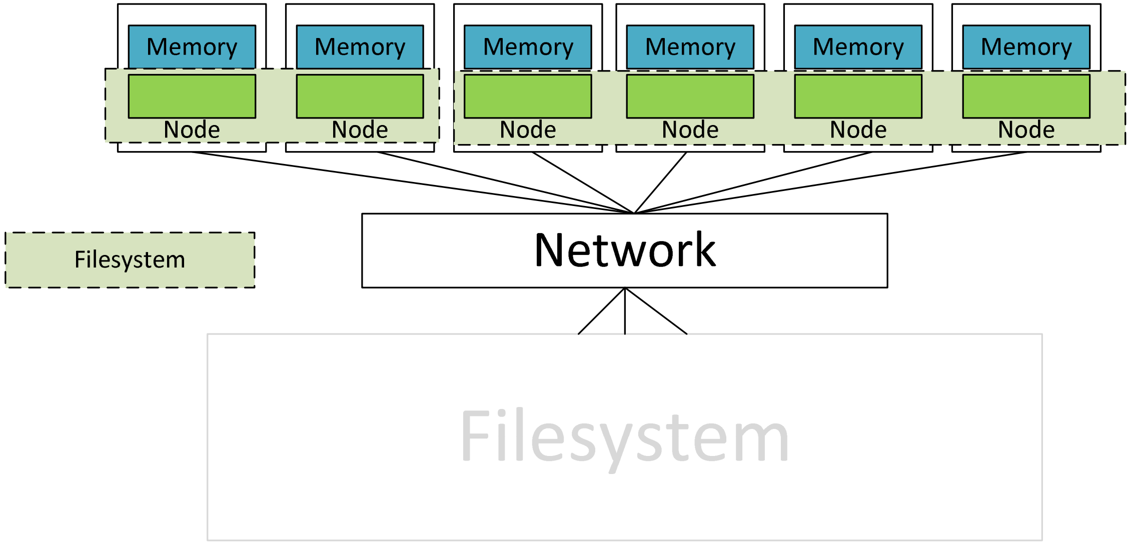
- Without changing applications
  - Filesystem buffer



- 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

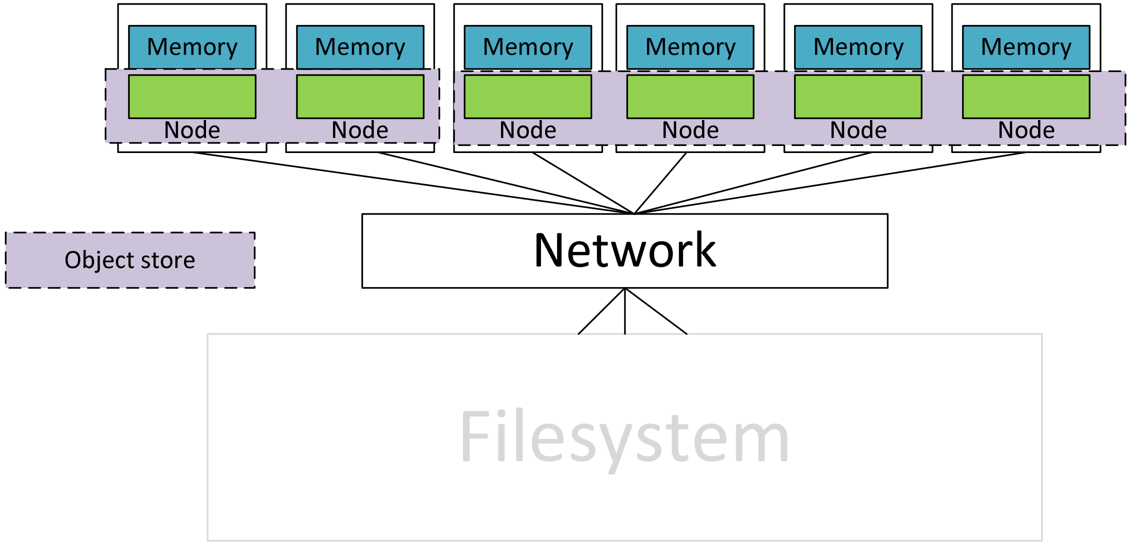
- Without changing applications
  - Global filesystem



  - 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

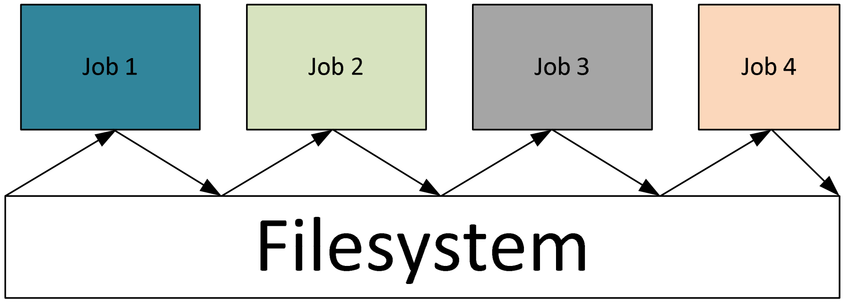
- With changes to applications
  - Object store



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

Using distributed storage

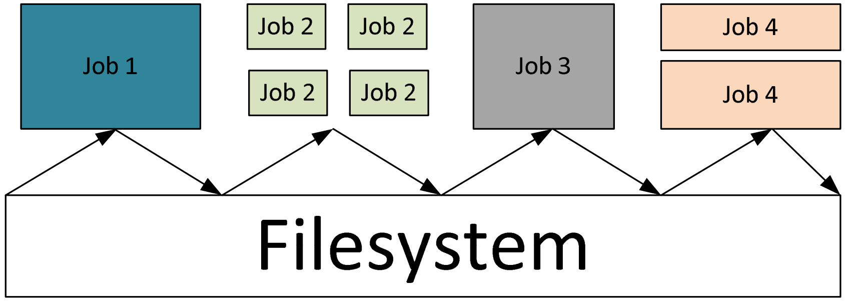
- New usage models
  - Resident data sets
    - Sharing preloaded data across a range of jobs
    - Data analytic workflows
    - How to control access/authorisation/security/etc….?
  - Workflows
    - Producer-consumer model



    - 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?

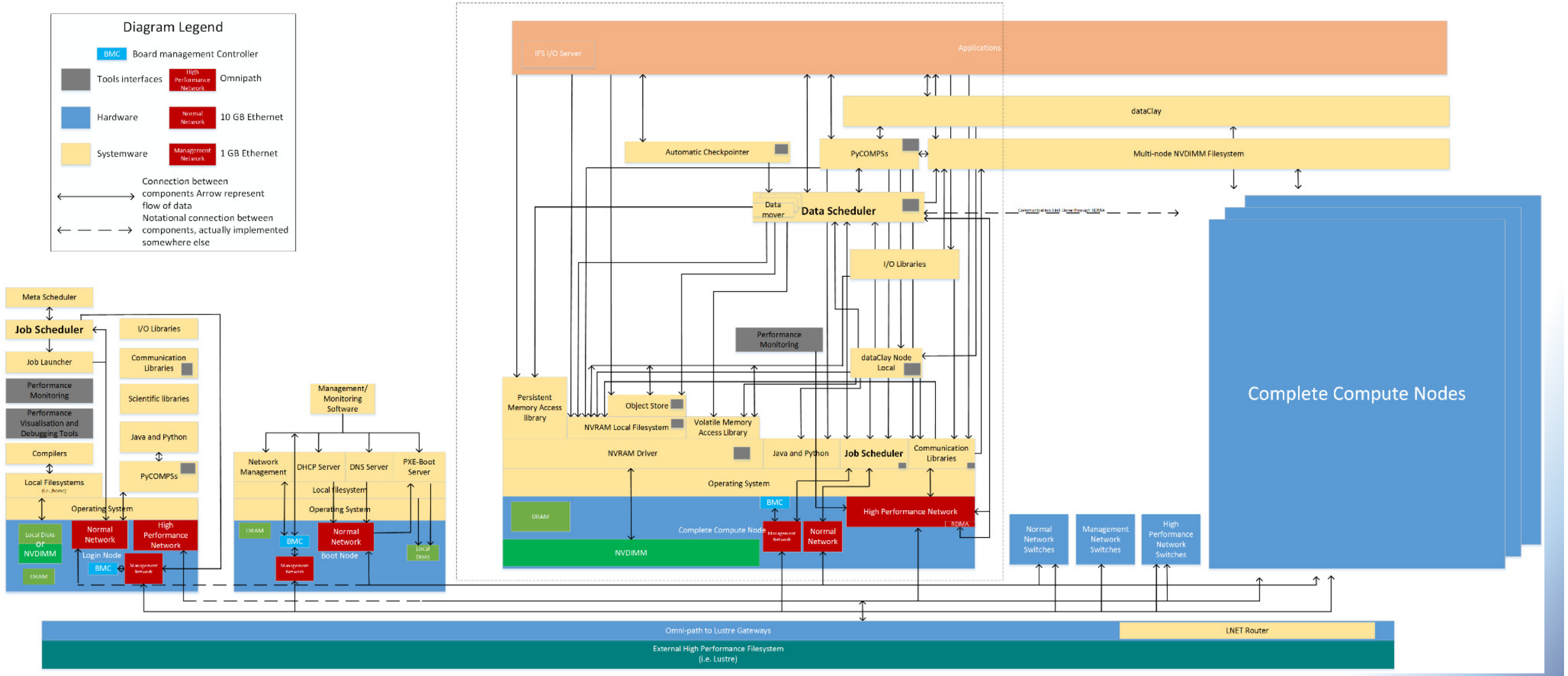
The challenge of distributed storage

- Enabling all the use cases in multi-user, multi-job environment is the real challenge
  - Heterogeneous scheduling mix
  - Different requirements on the SCM
  - 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?

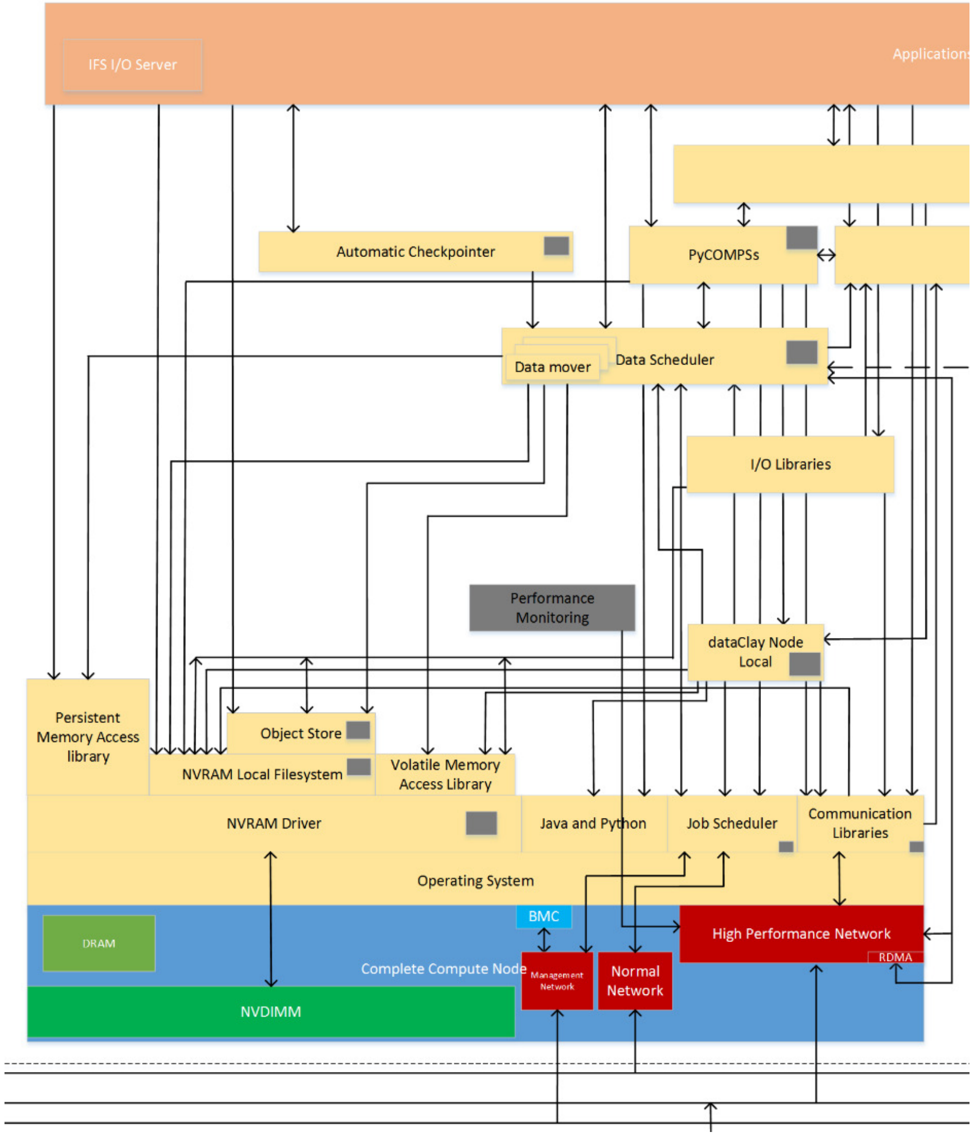
Potential solutions

- Large memory space
- Burst buffer
- Filesystem across NVRAM in nodes
- HSM functionality
- Object store across nodes
- Checkpointing and I/O libraries
- Much of the above require active systemware
  - Integration with the job scheduler
  - Data scheduler for “automatic” data movement
  - Namespace provision and distributed -> single dataview management

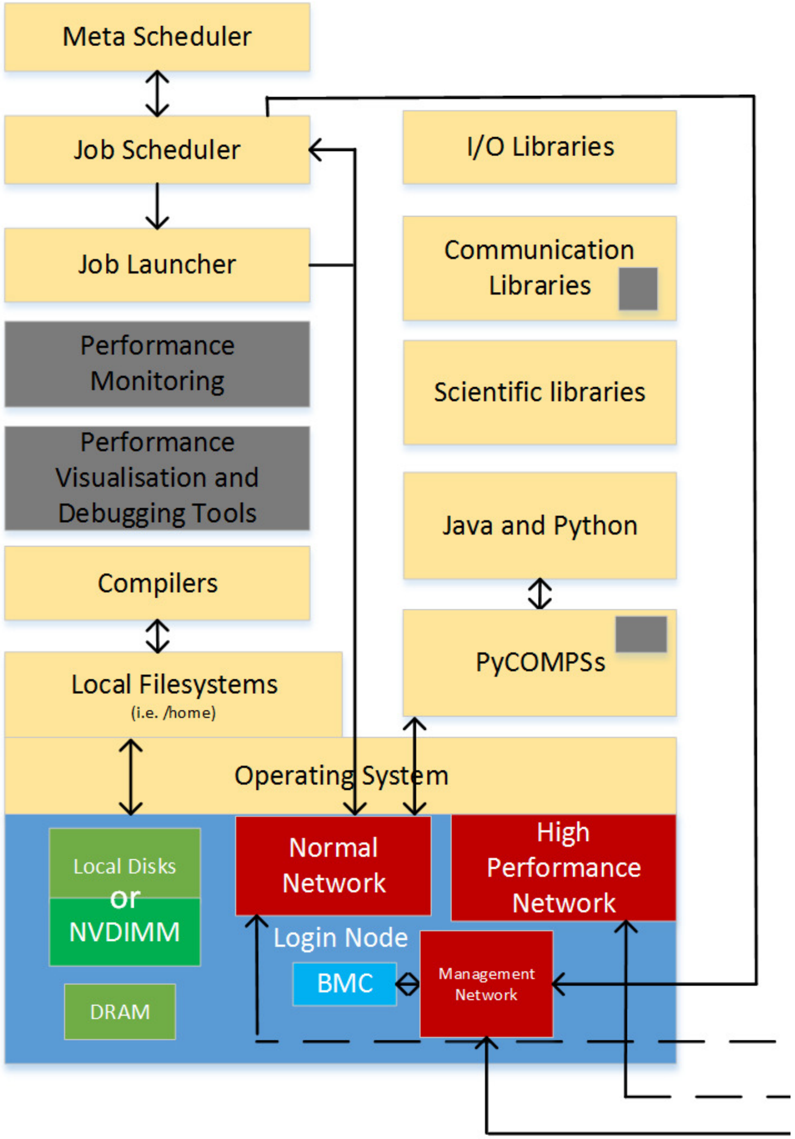
NEXTGenIO Systemware



Compute node systemware



User node systemware



Summary

- Byte-Addressable Persistent Memory is here
  - Price and capacity remains to be seen, but initial indications are interesting (large, cheaper than DRAM on a per GB)
  - 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…

Further reading

- http://www.nextgenio.eu
- Architectures for High Performance Computing and Data Systems using Byte-Addressable Persistent Memory
  http://arxiv.org/abs/1805.10041