An Architecture for High Performance Computing and Data Systems using Byte-Addressable Persistent Memory

Adrian Jackson

@adrianjhpc

a.jackson@epcc.ed.ac.uk

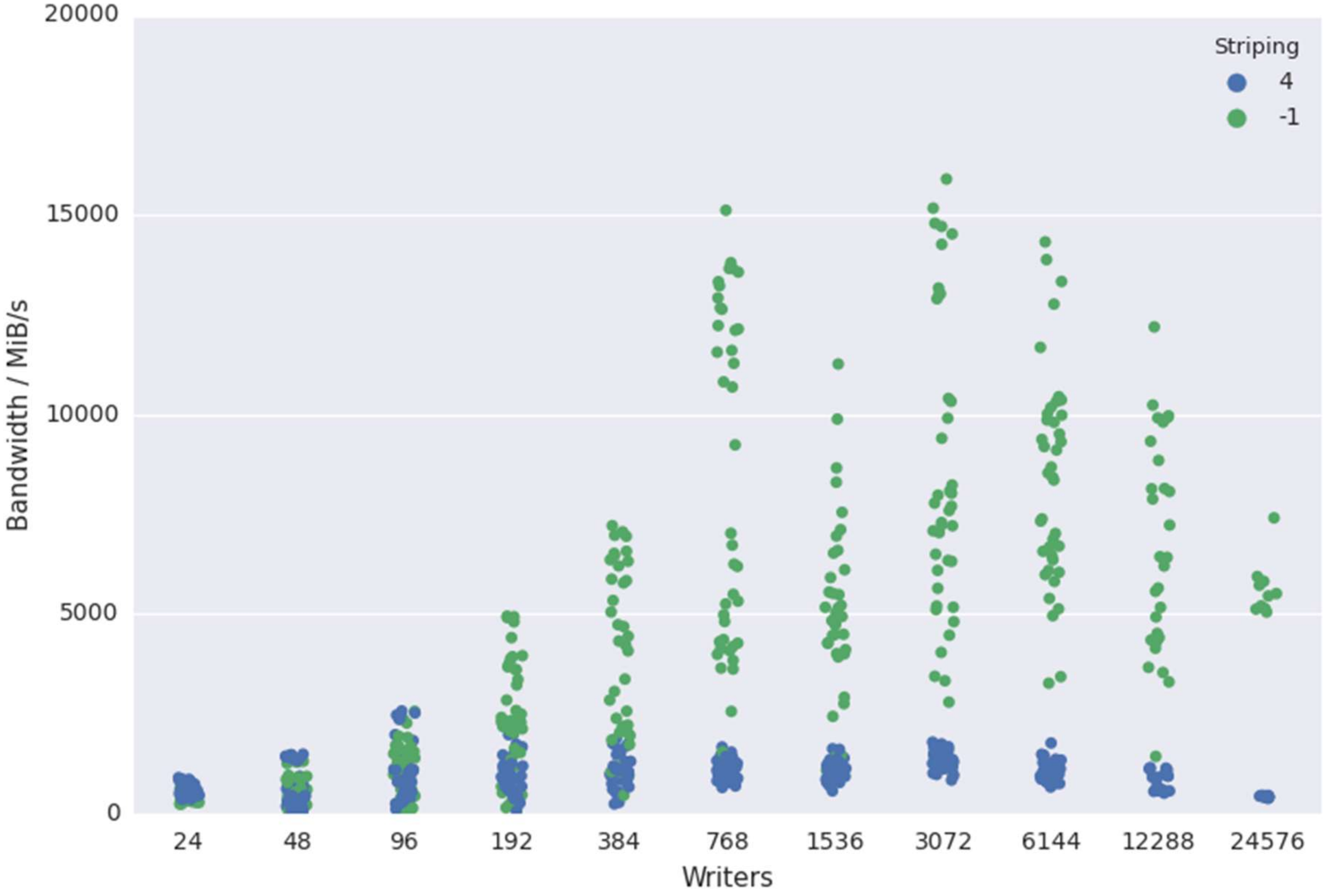
EPCC, The University of Edinburgh

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 ☺

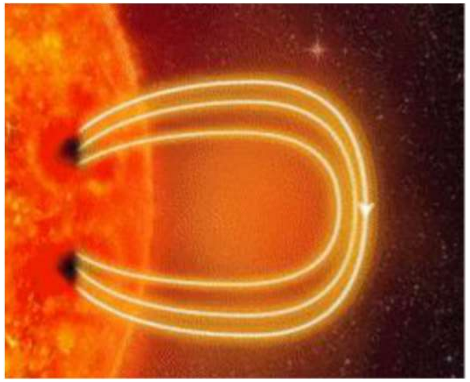
I/O Performance

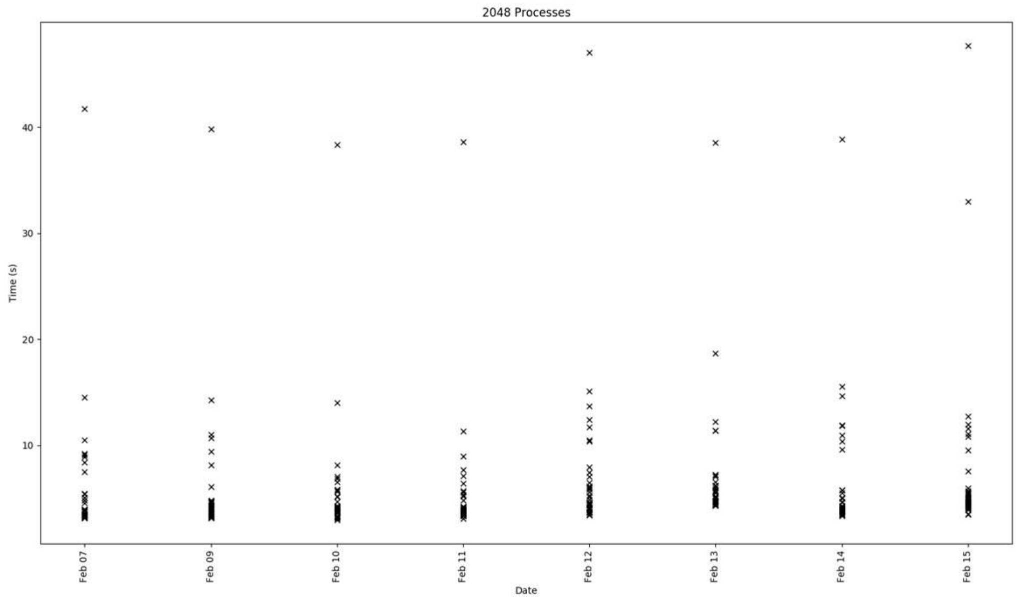


- https://www.archer.ac.uk/documentation/white-papers/parallelIO-benchmarking/ARCHER-Parallel-IO-1.0.pdf

I/O Performance – Small writes

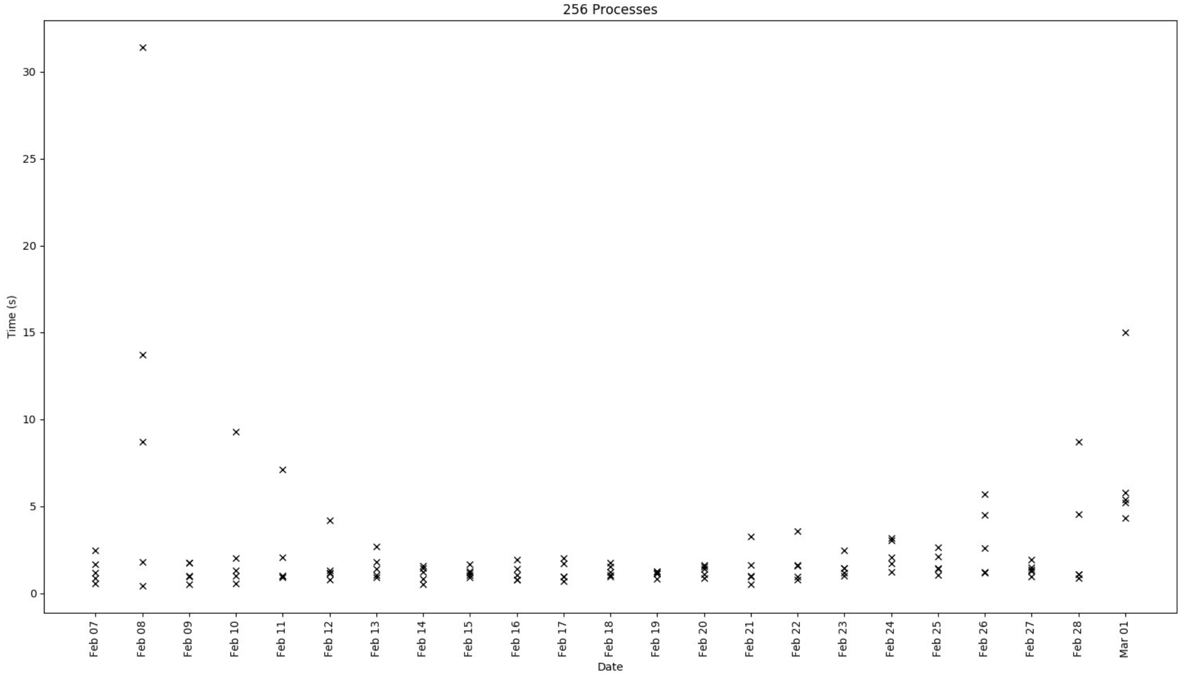
- Plot of average (across processes) run times of individual I/O regions for visualisation I/O
  - Same code executed for all runs
- I/O varies significantly in some cases:
  - Worst case ~12x
  - Best case ~2x



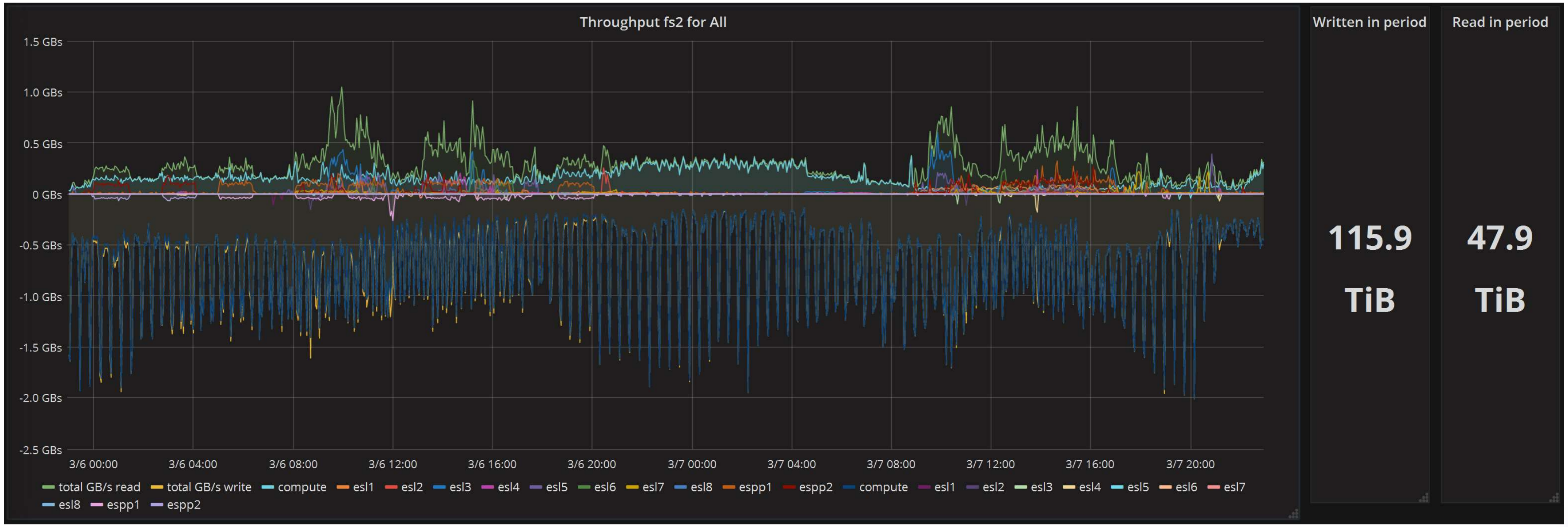


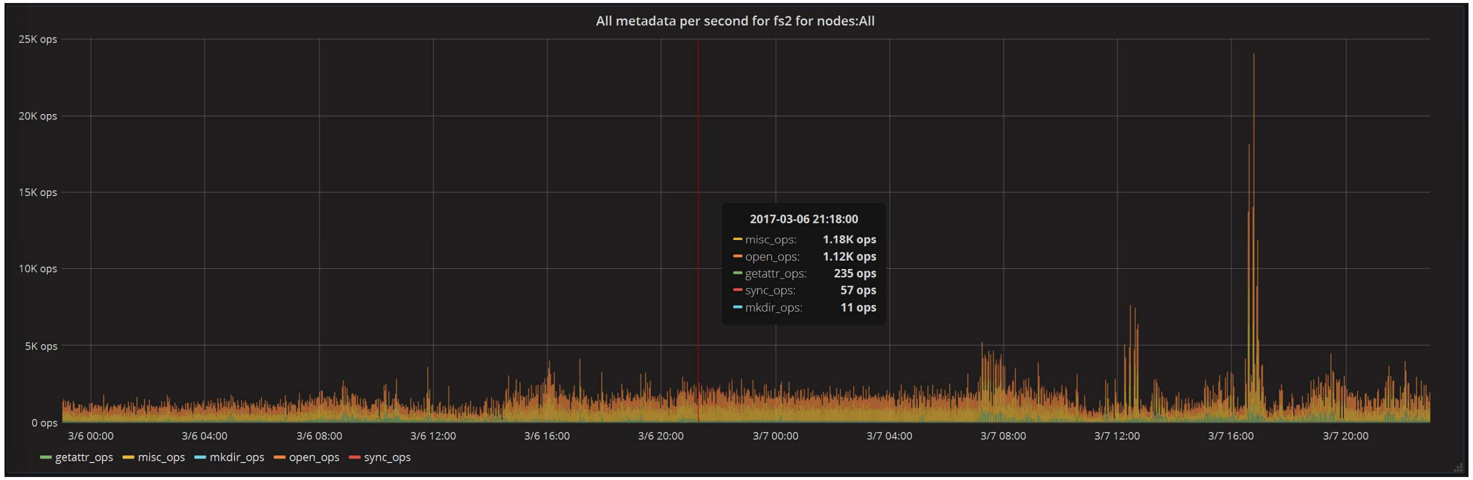
I/O Performance – Large writes

- 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

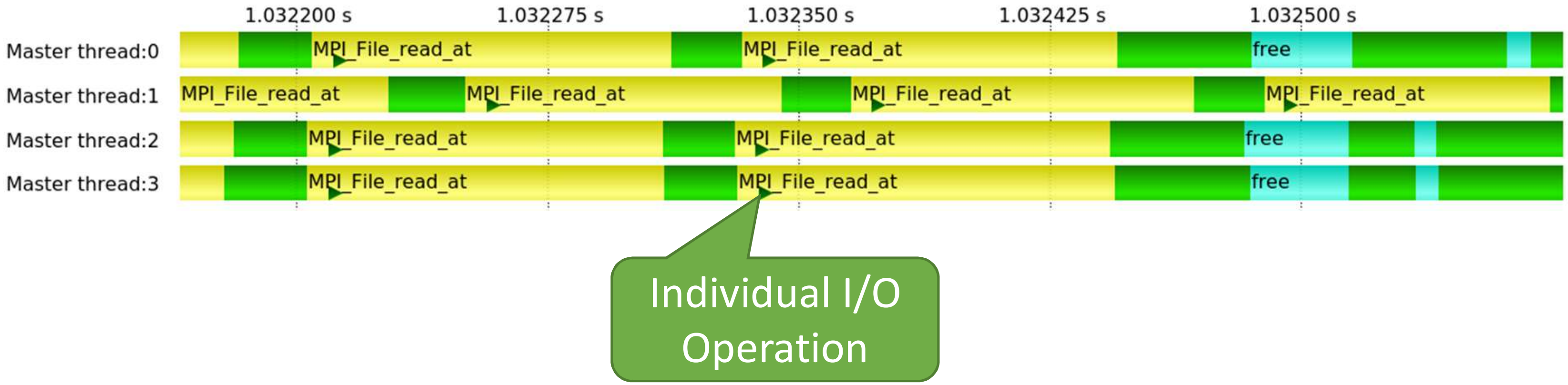


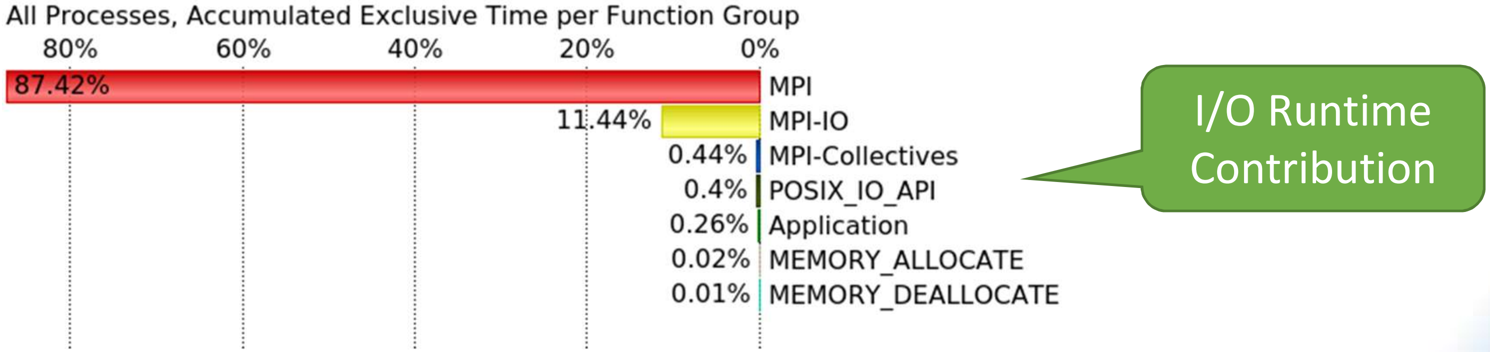
I/O Performance





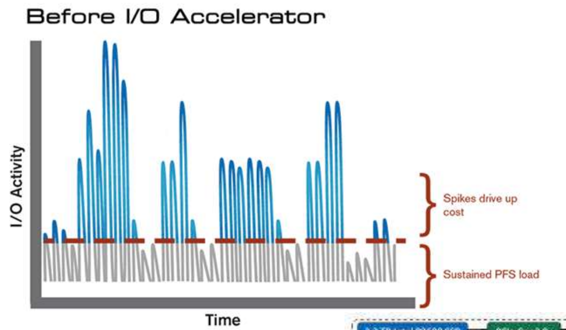
Application I/O patterns

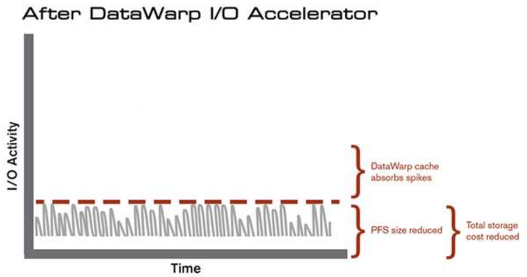


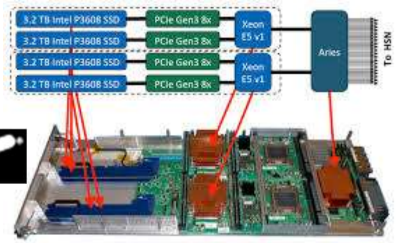


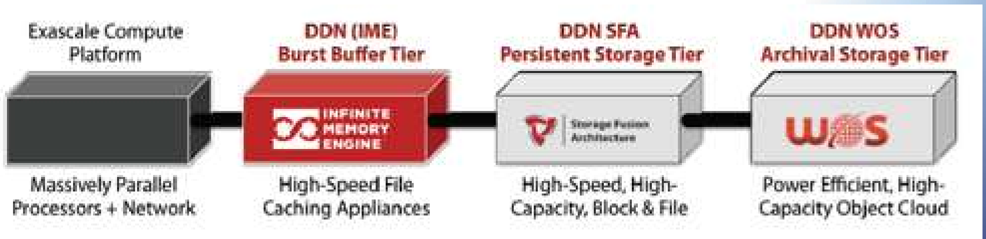
Burst Buffer

- 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)

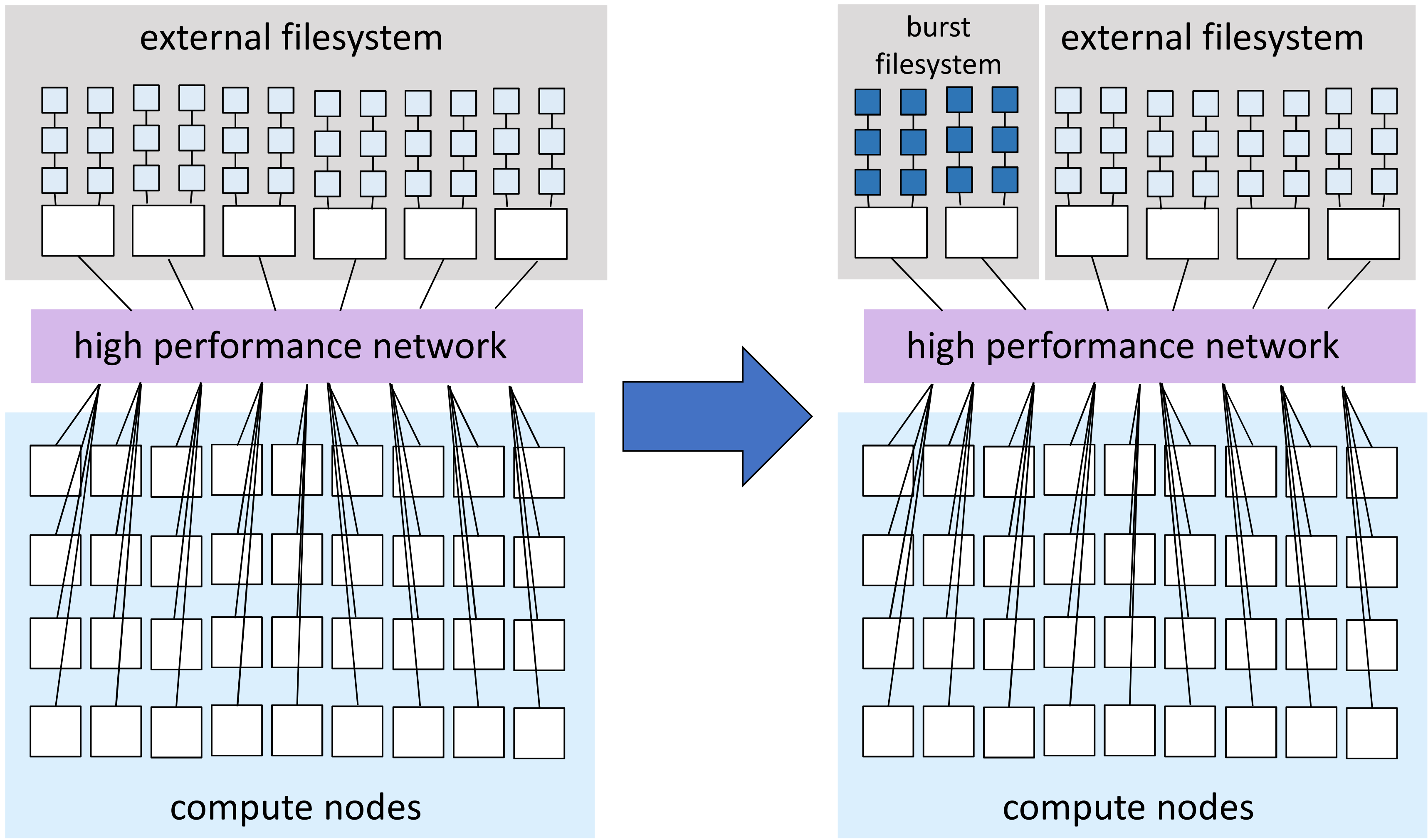






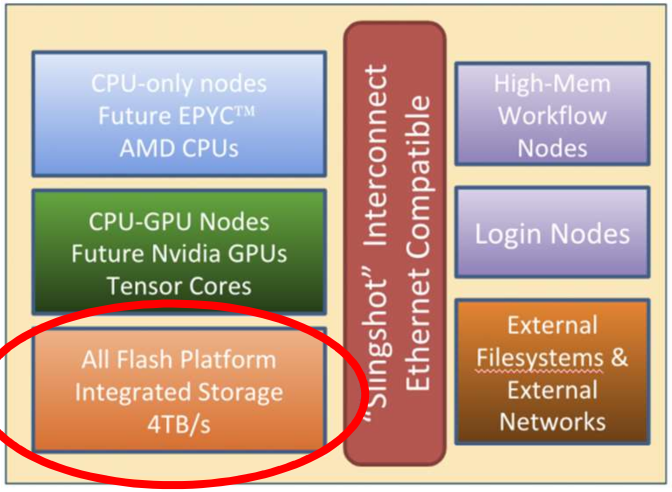


Burst buffer



Future storage

Perlmutter

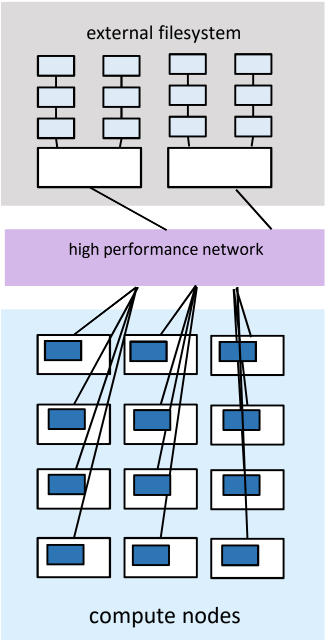


All-flash scratch filesystem

- 30-petabyte Lustre filesystem
- 4 TB/sec

Moving beyond burst buffer

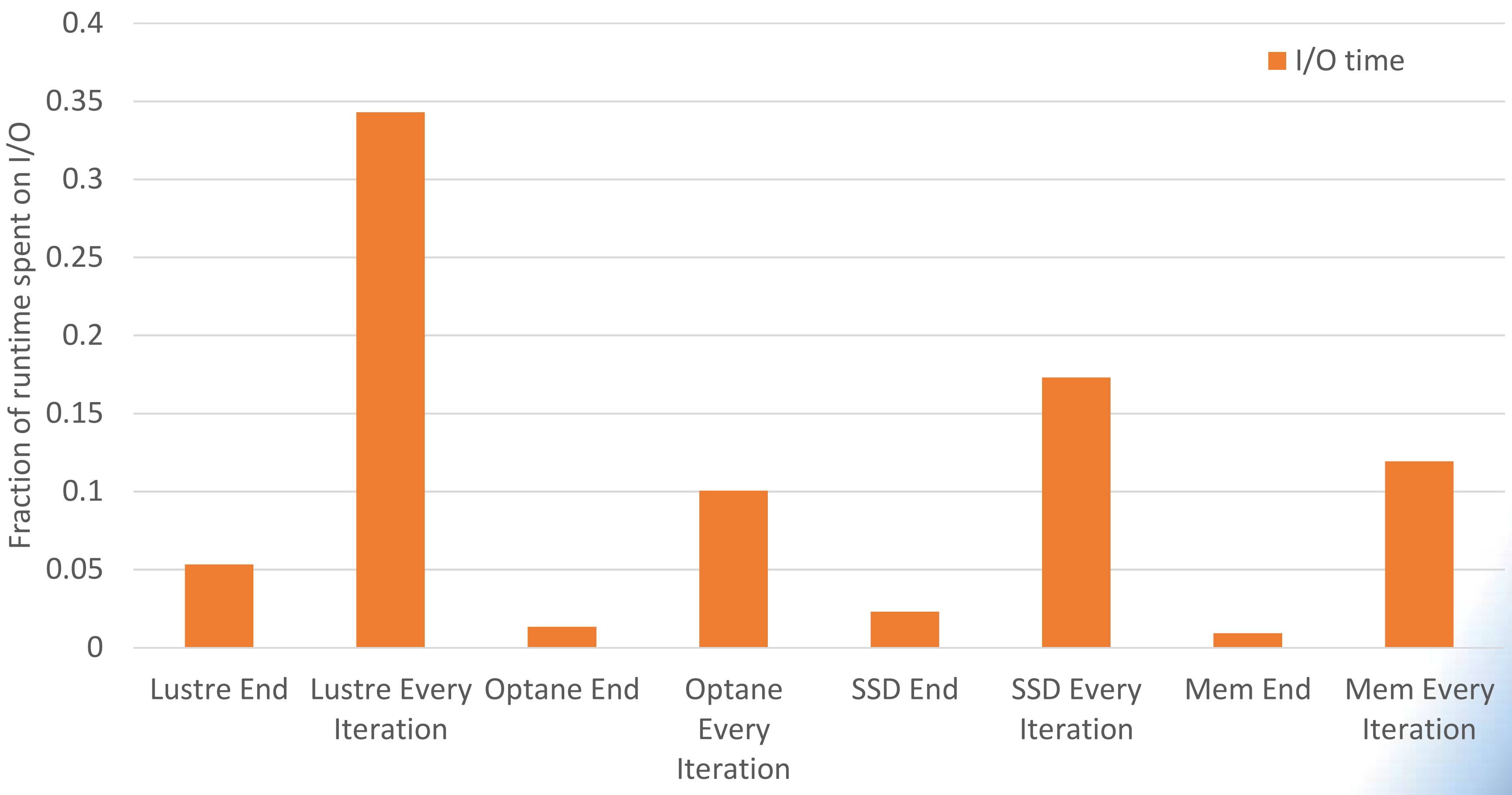
- Storage is moving to the node rather than the filesystem
- Argonne Theta machine has 128GB SSD in each compute node



Moving beyond burst buffer

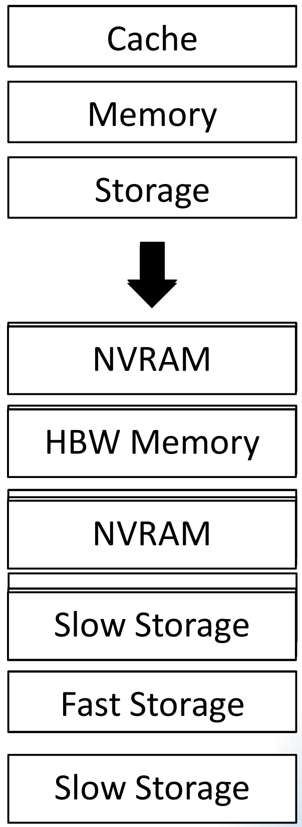
- Aurora will feature next generation Intel DPCMM

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

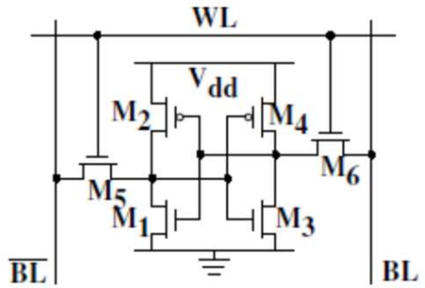


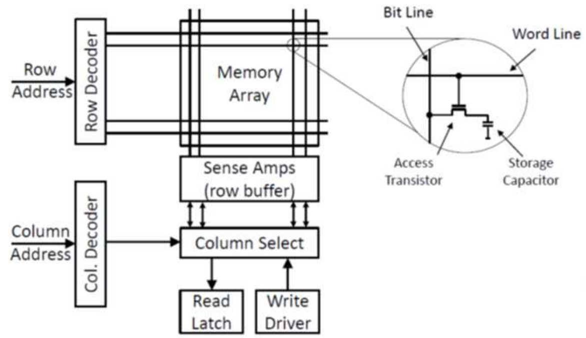
Non-volatile memory

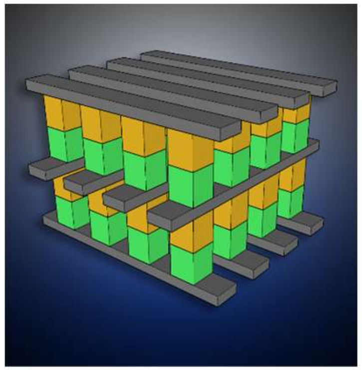
- Non-volatile RAM
  - Intel DCPMM technology
  - STT-RAM
- Much larger capacity than DRAM
  - Hosted in the DRAM slots, controlled by a standard memory controller
- Slower than DRAM by a small factor, but significantly faster than SSDs
- STT-RAM
  - Read fast and low energy
  - Write slow and high energy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

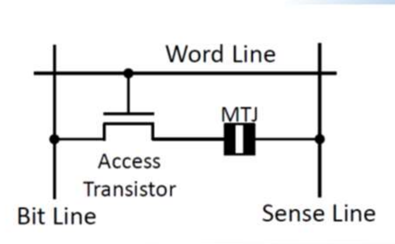
SRAM vs NVRAM

- 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)



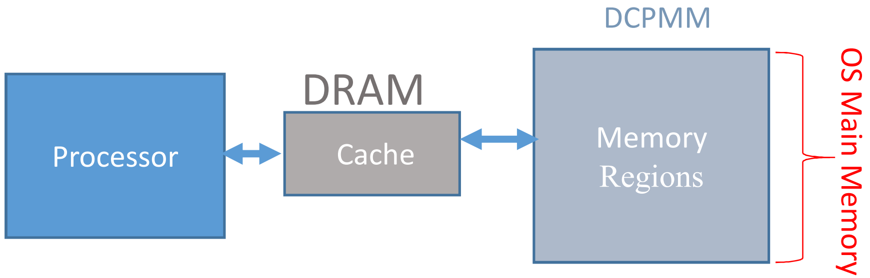




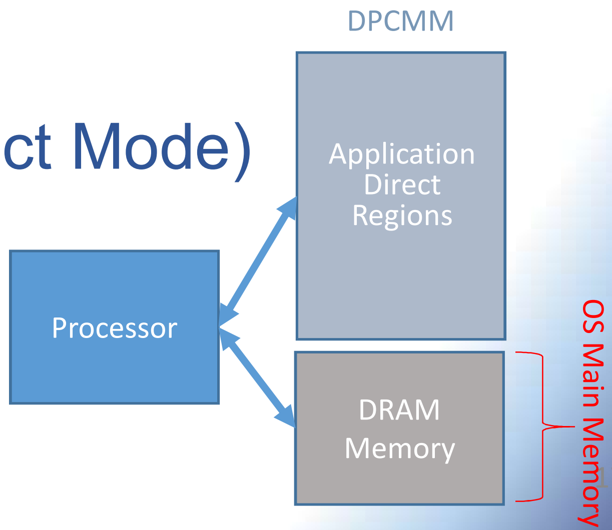


Memory levels

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



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



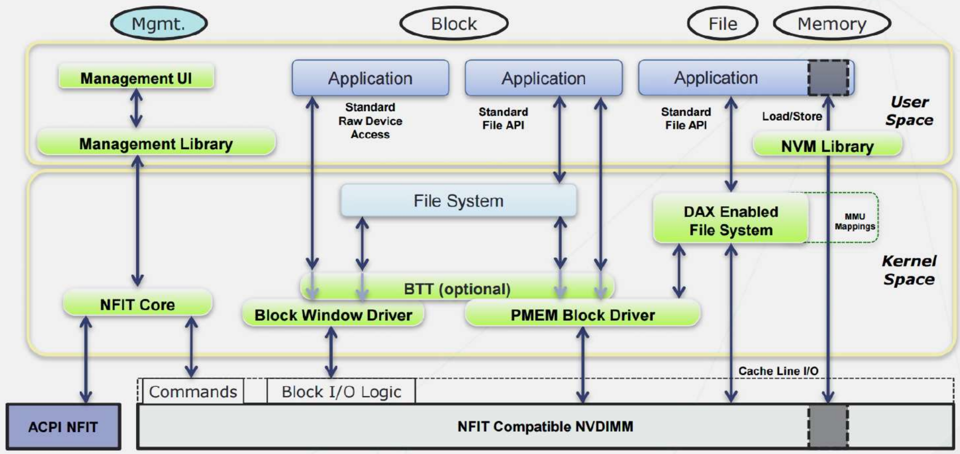
*https://www.google.com/patents/US20150178204

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

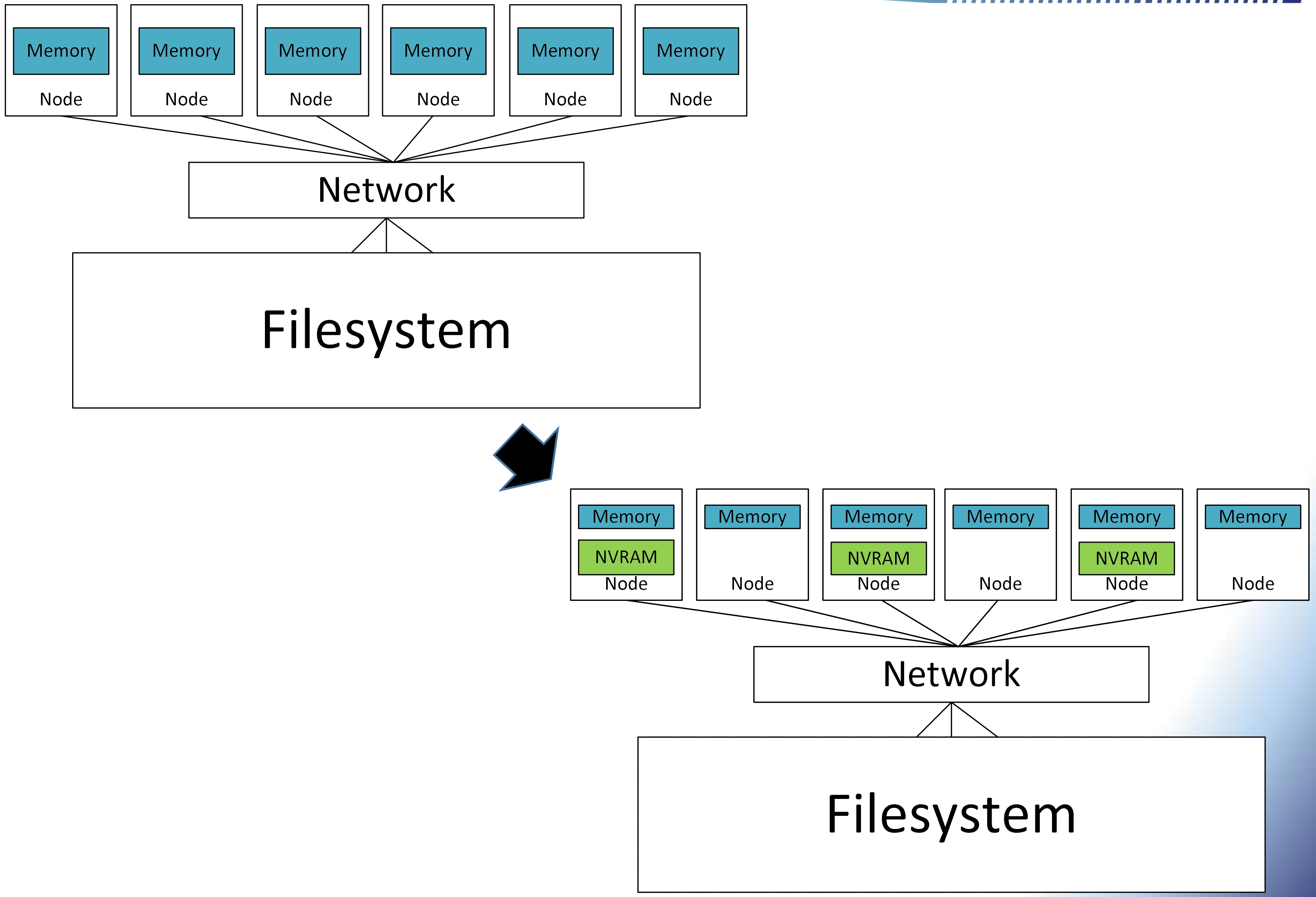
Programming DCPMM

- 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



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

Exploiting distributed storage



NEXTGenIO

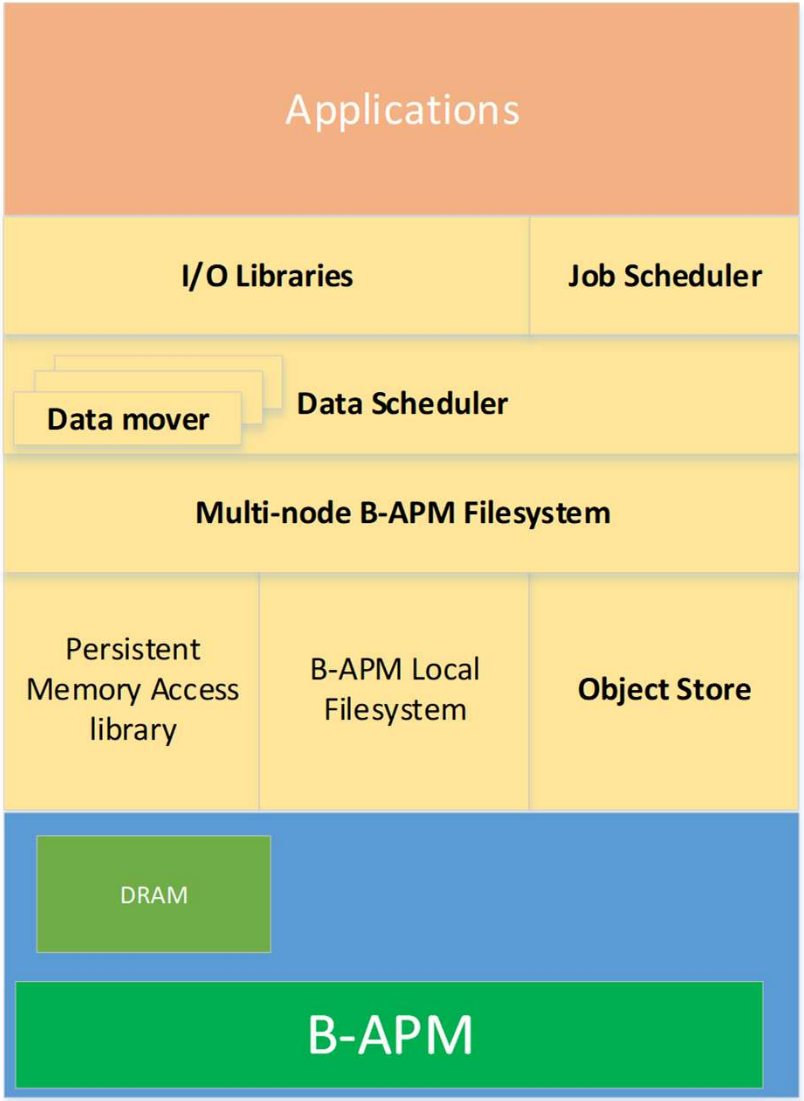
Project
- European Funded Research & Innovation Action
- 42 month duration
- €8.1 million
- Approx. 50% committed to hardware development

Partners
- EPCC
- INTEL
- FUJITSU
- BSC
- TUD
- ARM/ALLINEA
- ECMWF
- ARCTUR

NGIO

- Whole ecosystem development
  - Support hardware and software, support users in porting and optimising application
- Hardware development
  - Fujitsu motherboard and BIOS work
  - Intel memory and processor hardware
- Software development
  - Applications
  - Scheduler
  - Filesystems
  - Data scheduler
  - Profilers and debuggers

Systemware architecture

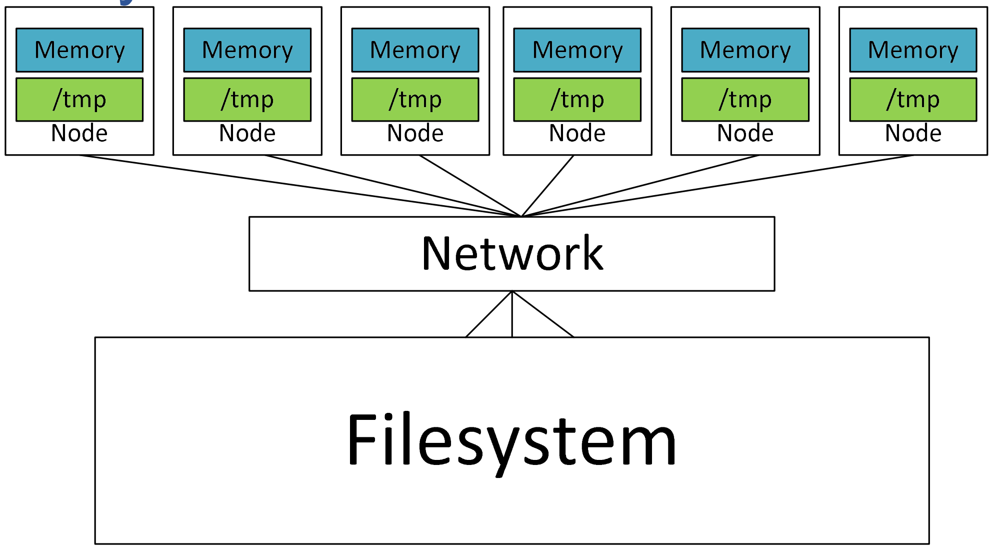


NGIO Prototype

- 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

Using distributed storage

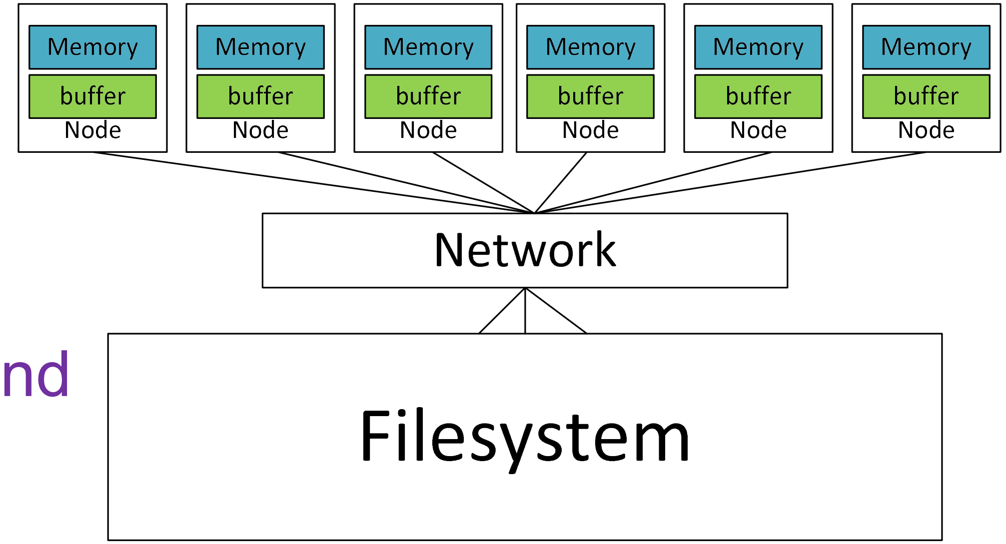
- Without changing applications
  - Large memory space/in-memory database etc…
  - Local filesystem



- 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

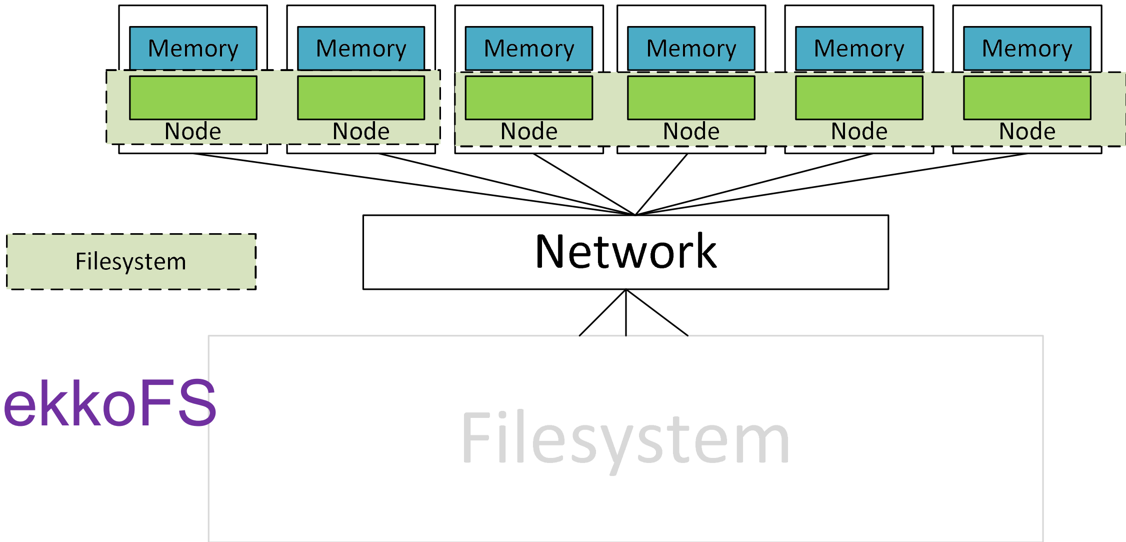


NGIO Data Scheduler (NORNS) and Slurm integration

- 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

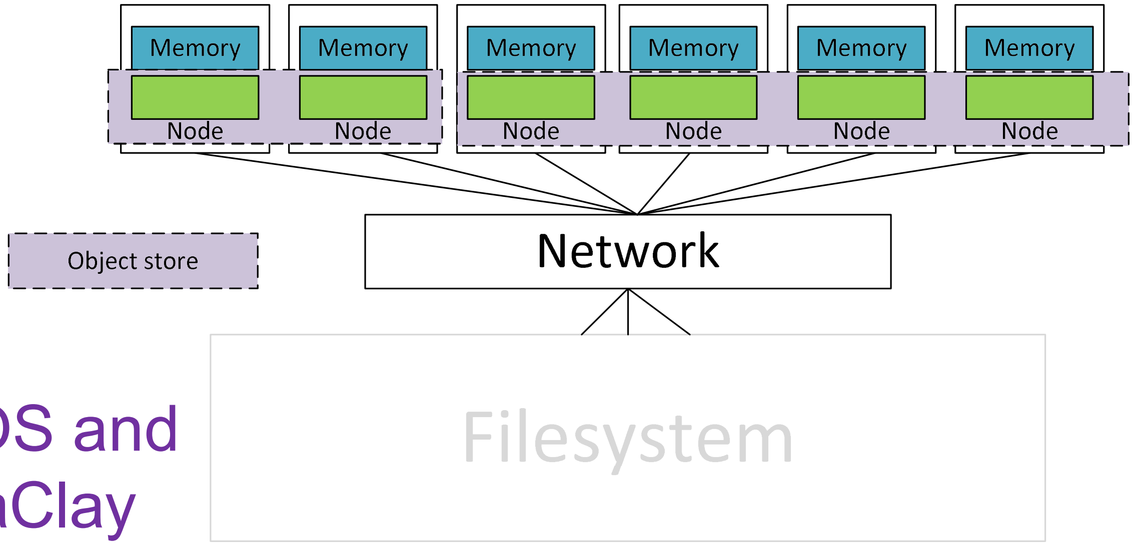


NGIO GekkoFS

  - 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

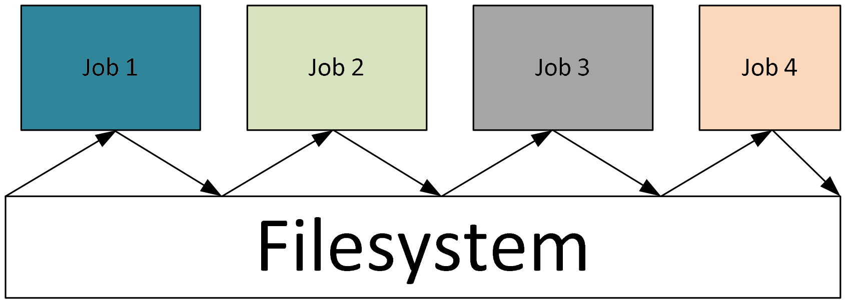


Intel DAOS and
BSC dataClay

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

Using distributed storage

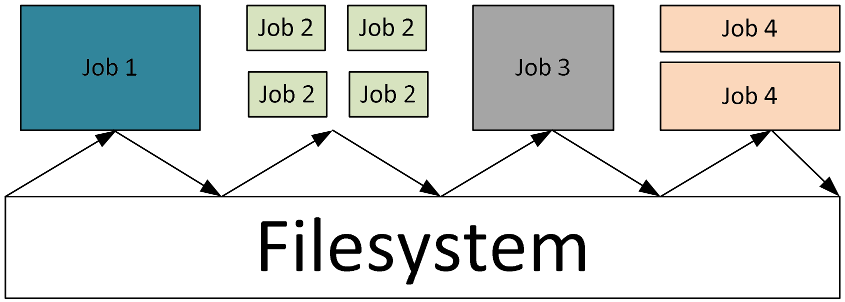
- New usage models
  - Resident data sets
    - Sharing preloaded data across a range of jobs
    - Data analytic workflows
    - How to control access/authorisation/security/etc….?
  - Workflows
    - Producer-consumer model



    - 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?

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

Performance - workflows

1 node

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

1 node

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

16 nodes

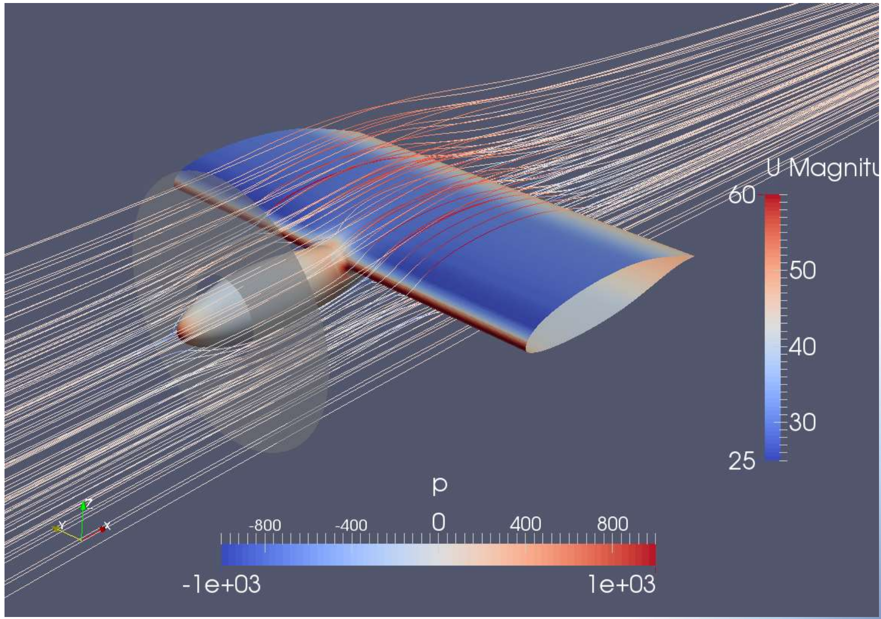
OPENFOAM WORKFLOW BENCHMARK USING LUSTRE VS NVMS + DATA STAGING

Workflow phase	Lustre	NVMs
decomposition	1191	1105
data-staging	–	32
solver	123	66

- Ext4 filesystem on each socket
- Standard file access

20 nodes

Workflow phase	Lustre	B-APM
Decomposition	1841	1453
Data-staging		330
Solver	664	78



Performance – IO-500

- GekkoFS filesystem
  - GekkoFS only using TCP/IP. Optimisations to be done to utilise the Omnipath network
  - Only using a single rail
  - Only using a single sockets worth of memory

- Lots of optimisation scope

Performance – IO-500

- Ten nodes

```
[RESULT] BW   phase 1            ior_easy_write            22.566 GB/s : time 334.77 seconds
[RESULT] IOPS phase 1         mdtest_easy_write           293.677 kiops : time 365.91 seconds
[RESULT] BW   phase 2            ior_hard_write             3.063 GB/s : time 309.71 seconds
[RESULT] IOPS phase 2         mdtest_hard_write            34.665 kiops : time 318.85 seconds
[RESULT] IOPS phase 3                      find          1245.860 kiops : time  94.33 seconds
[RESULT] BW   phase 3             ior_easy_read            21.625 GB/s : time 349.33 seconds
[RESULT] IOPS phase 4          mdtest_easy_stat           758.889 kiops : time 143.15 seconds
[RESULT] BW   phase 4             ior_hard_read             9.804 GB/s : time  96.78 seconds
[RESULT] IOPS phase 5          mdtest_hard_stat           768.476 kiops : time  17.48 seconds
[RESULT] IOPS phase 6        mdtest_easy_delete           441.682 kiops : time 248.24 seconds
[RESULT] IOPS phase 7          mdtest_hard_read           159.821 kiops : time  71.86 seconds
[RESULT] IOPS phase 8        mdtest_hard_delete            37.775 kiops : time 293.52 seconds
[SCORE] Bandwidth 11.0028 GB/s : IOPS 258.151 kiops : TOTAL 53.2953
```

- Twenty nodes

```
[RESULT] BW   phase 1            ior_easy_write            45.689 GB/s : time 326.58 seconds
[RESULT] IOPS phase 1         mdtest_easy_write           398.313 kiops : time 348.71 seconds
[RESULT] BW   phase 2            ior_hard_write             3.827 GB/s : time 310.10 seconds
[RESULT] IOPS phase 2         mdtest_hard_write            48.792 kiops : time 315.29 seconds
[RESULT] IOPS phase 3                      find          2645.500 kiops : time  57.71 seconds
[RESULT] BW   phase 3             ior_easy_read            48.452 GB/s : time 307.96 seconds
[RESULT] IOPS phase 4          mdtest_easy_stat          1040.100 kiops : time 133.82 seconds
[RESULT] BW   phase 4             ior_hard_read            13.438 GB/s : time  88.32 seconds
[RESULT] IOPS phase 5          mdtest_hard_stat          1063.020 kiops : time  16.73 seconds
[RESULT] IOPS phase 6        mdtest_easy_delete           592.988 kiops : time 239.39 seconds
[RESULT] IOPS phase 7          mdtest_hard_read           239.824 kiops : time  66.02 seconds
[RESULT] IOPS phase 8        mdtest_hard_delete            41.083 kiops : time 374.58 seconds
[SCORE] Bandwidth 18.3687 GB/s : IOPS 367.42 kiops : TOTAL 82.1525
```

Performance - STREAM

https://github.com/adrianjhpc/DistributedStream.git

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

```
STREAM_TYPE     *a, *b, *c;
pmemaddr = pmem_map_file(path, array_length,
                         PMEM_FILE_CREATE|PMEM_FILE_EXCL,
                         0666, &mapped_len, &is_pmem)
a = pmemaddr;
b = pmemaddr + (*array_size+OFFSET)*BytesPerWord;
c = pmemaddr + (*array_size+OFFSET)*BytesPerWord*2;

#pragma omp parallel for
for (j=0; j<*array_size; j++){
   a[j] = b[j]+scalar*c[j];
}
pmem_persist(a, *array_size*BytesPerWord);
```

Performance - STREAM

```
unsigned long get_processor_and_core(int *socket, int *core){
  unsigned long a,d,c;
  __asm__ volatile("rdtscp" : "=a" (a), "=d" (d), "=c" (c));
  *socket = (c & 0xFFF000)>>12;
  *core = c & 0xFFF;
   return ((unsigned long)a) | (((unsigned long)d) << 32);;
}


strcpy(path,"/mnt/pmem_fsdax");
sprintf(path+strlen(path), "%d", socket/2);
sprintf(path+strlen(path), "/");
```

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 performance of Intel DCPMM can be very significant