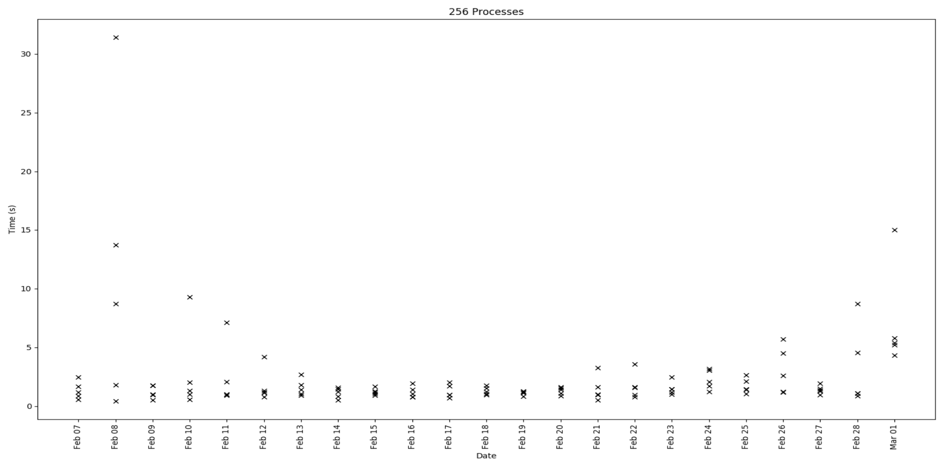
Persistent Memory for I/O

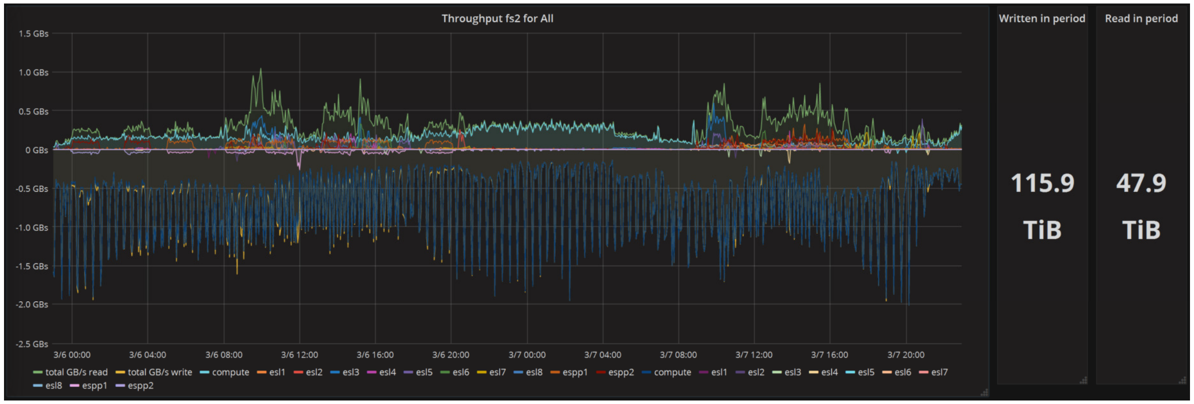
Adrian Jackson, Iakovos Panourgias
(EPCC, The Unversity of Edinburgh)
Ramon Nou, Alberto Miranda
(BSC)

@adrianjhpc
a.jackson@epcc.ed.ac.uk
http://www.nextgenio.eu

I/O Performance – Large writes

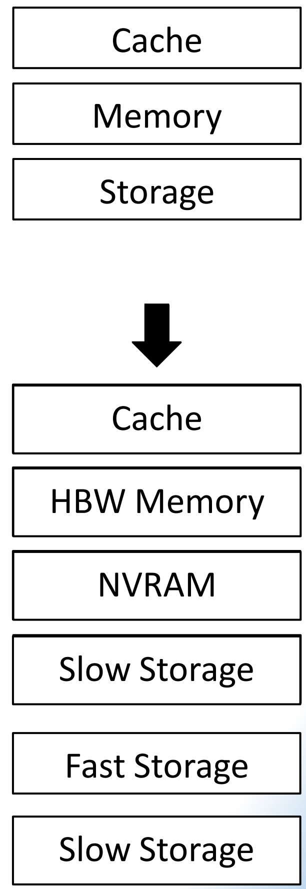
- Plot of run times of individual I/O regions for checkpoint I/O
  - Same code executed for all runs
- Checkpoint I/O less frequent but much quicker
  - Much higher data volumes



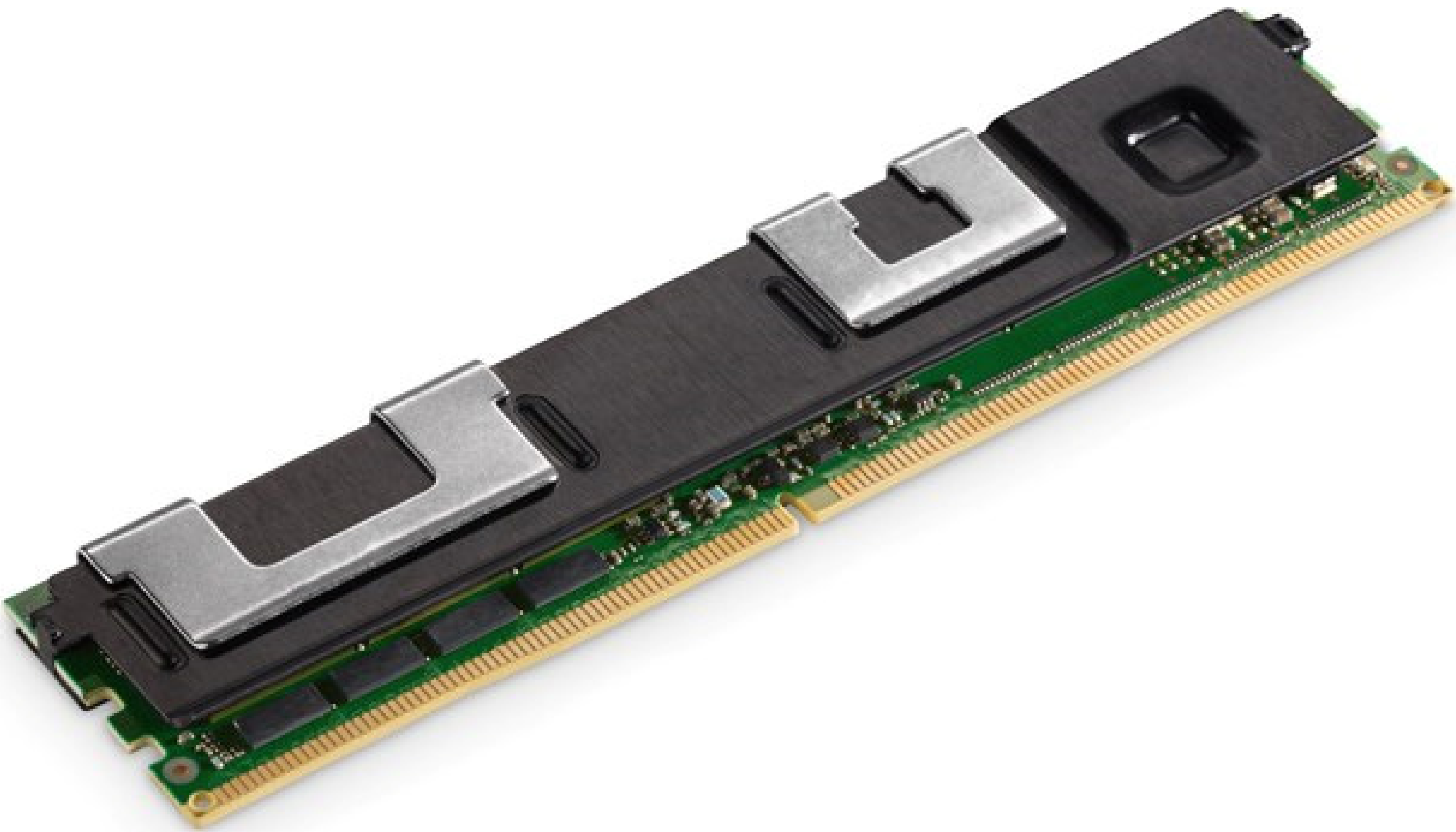


New Memory Hierarchies

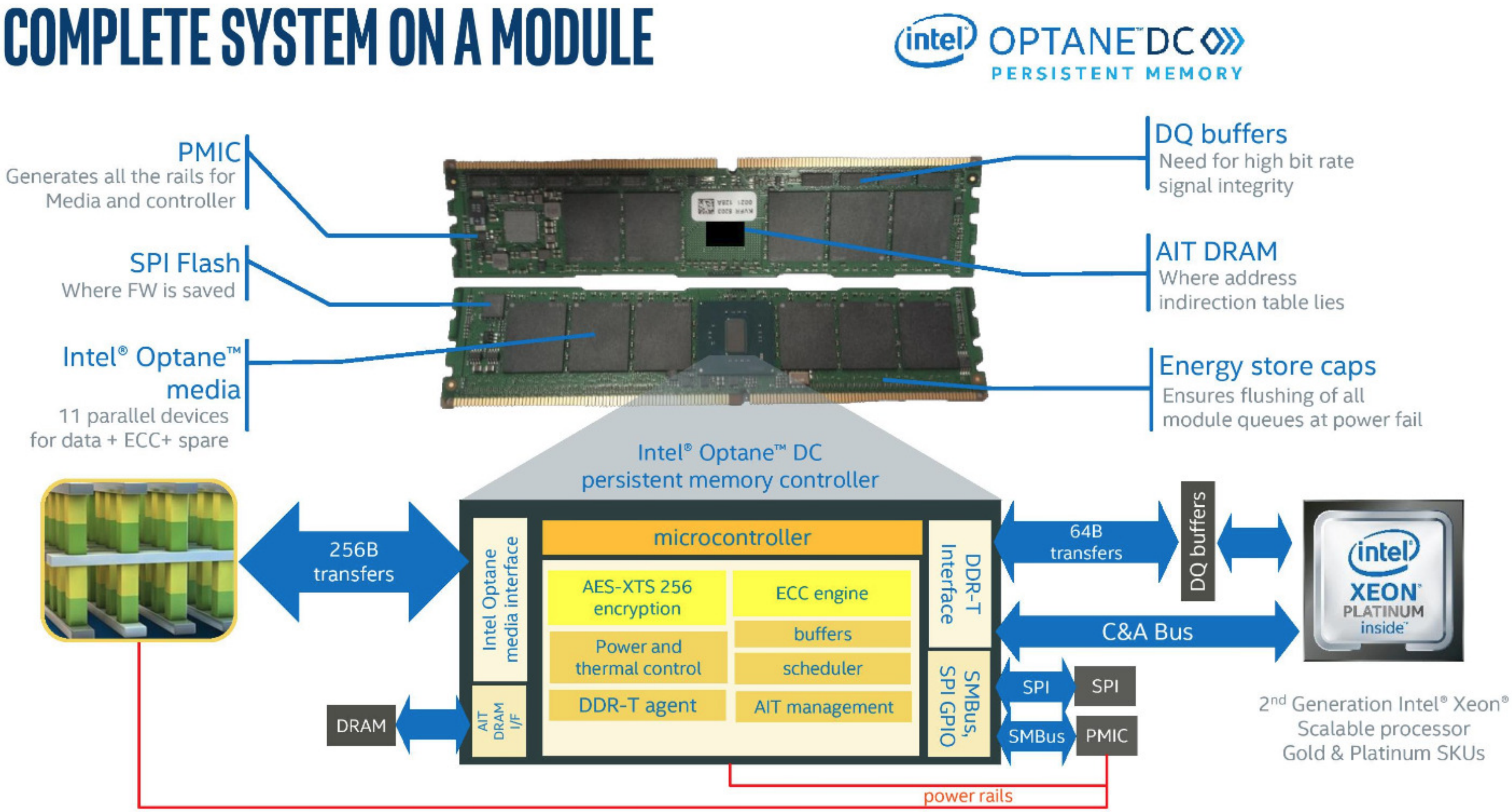
- High bandwidth, on processor memory
  - Large, high bandwidth cache
  - Latency cost for individual access may be an issue
- Main memory
  - DRAM
  - Costly in terms of energy, potential for lower latencies than high bandwidth memory
- Byte-addressable Persistent Memory (B-APM)
  - High capacity, ultra 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



NVRAM / B-APM



Optane DCPMM



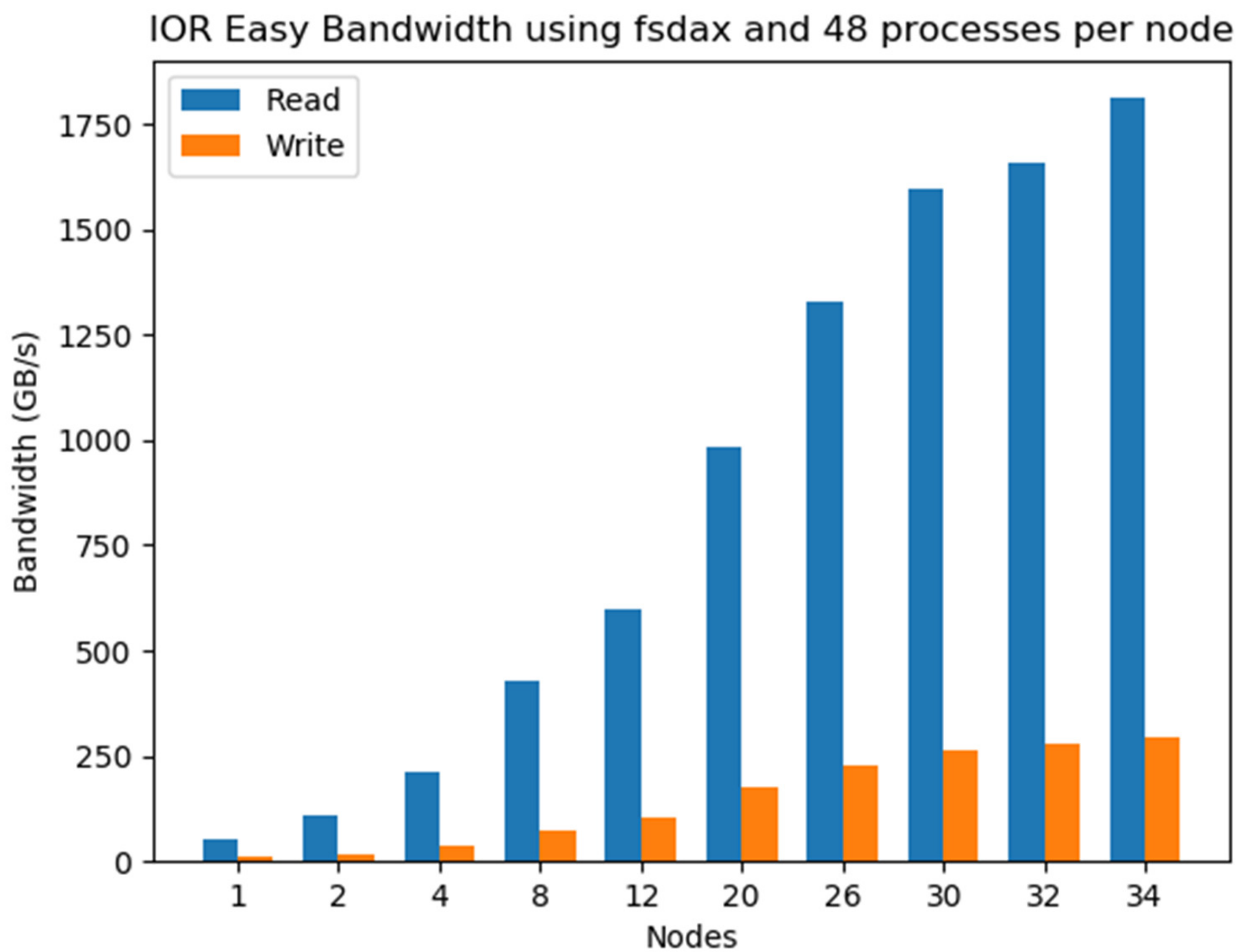
Performance - STREAM

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

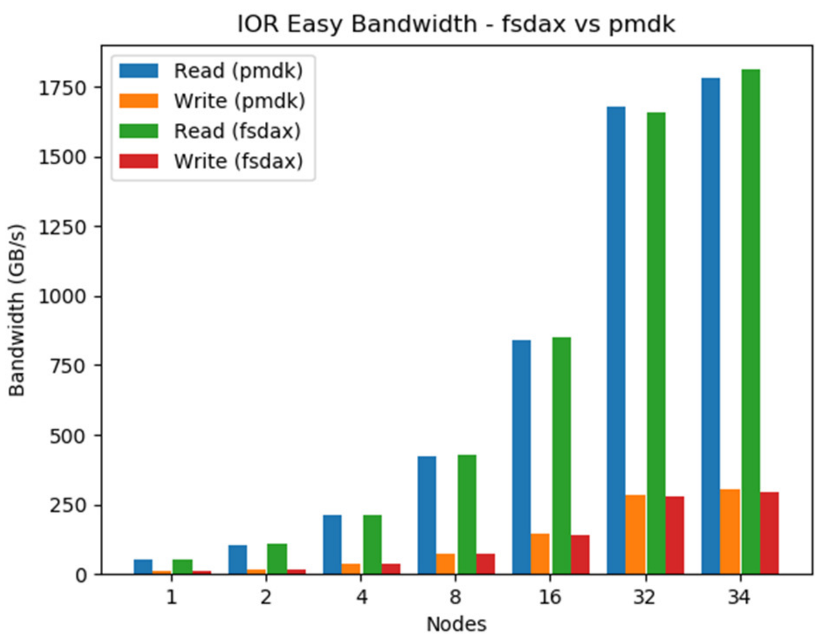
```
STREAM_TYPE     *a, *b, *c;
pmemaddr = pmem_map_file(path, array_length,
                         PMEM_FILE_CREATE|PMEM_FILE_EXCL,
                         0666, &mapped_len, &is_pmem)
a = pmemaddr;
b = pmemaddr + (*array_size+OFFSET)*BytesPerWord;
c = pmemaddr + (*array_size+OFFSET)*BytesPerWord*2;

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

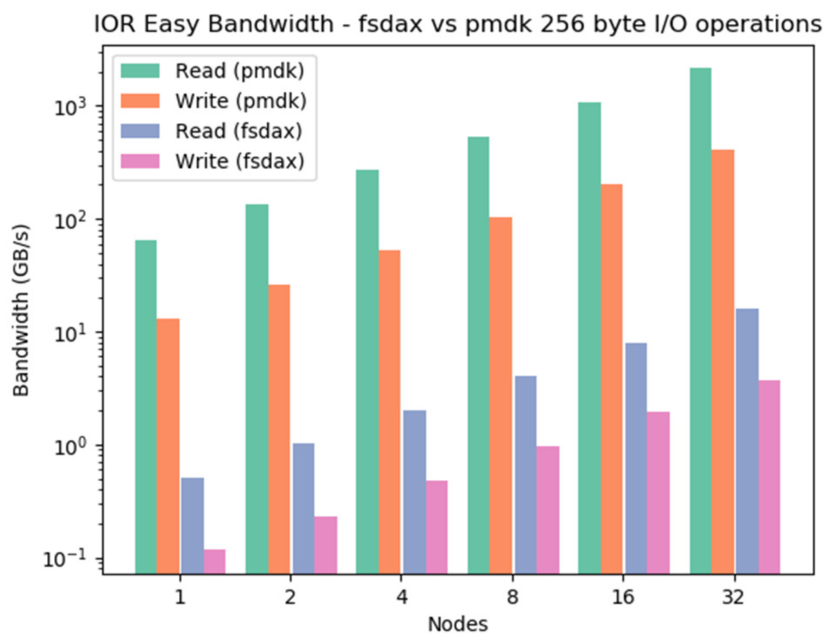
I/O Performance



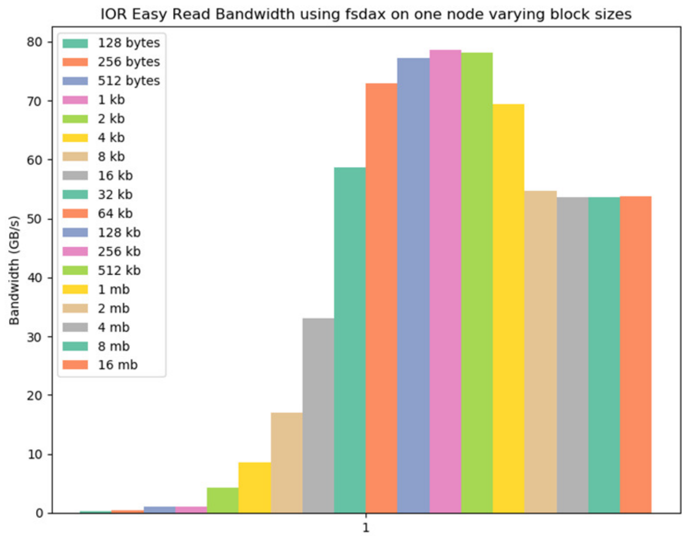
Move from I/O to Data

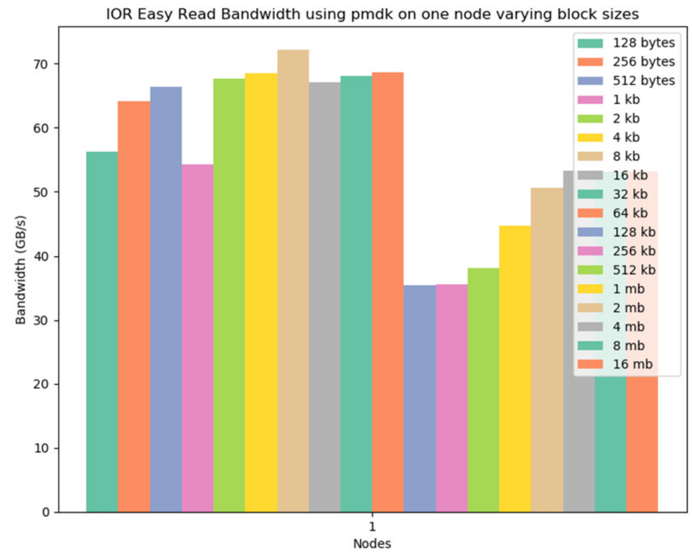
- Biggest potential for B-APM (to me) is removing the I/O interface
- Removing file (and block) operations

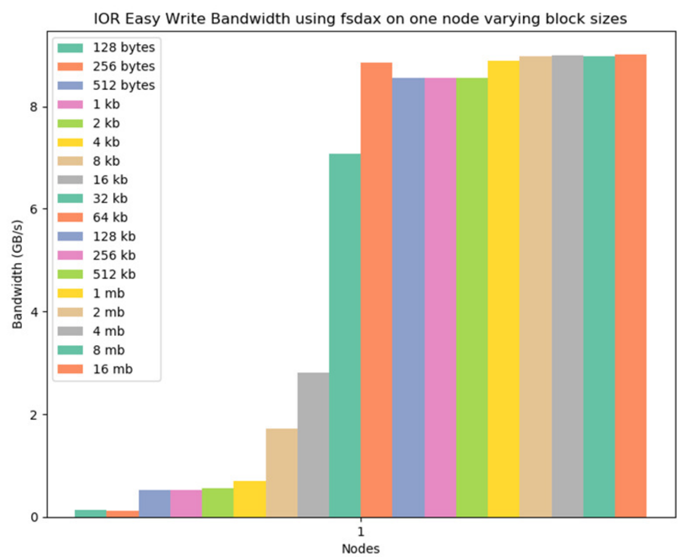


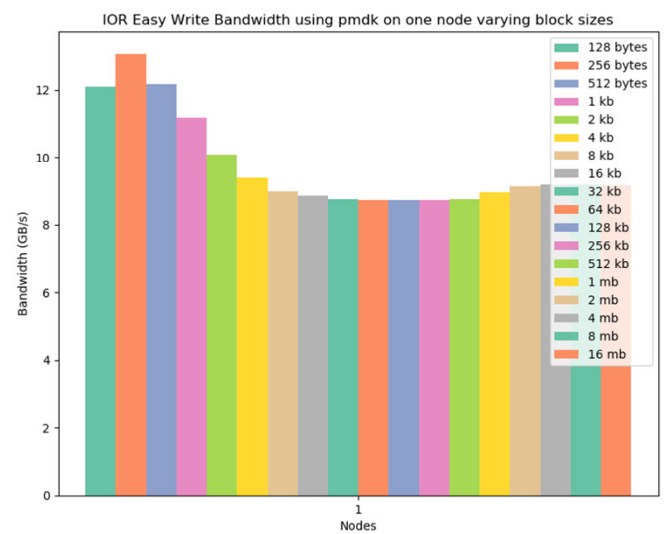


IOR - Data block sizes









Multi-level memory exploitation

```
address = (int **) malloc(nx*sizeof(int *) + nx*ny*sizeof(int));
fuzzy = int2D(nx, ny, address);
```

```
pmemaddr1 = pmem_map_file(filename, array_size,PMEM_FILE_CREATE|PMEM_FILE_EXCL,
                          0666, &mapped_len1, &is_pmem)
fuzzy =  int2D(nx, ny, pmemaddr1);
```

```
int **int2D(int nx, int ny, int **idata){
  int i;
  idata[0] = (int *) (idata + nx);

  for(i=1; i < nx; i++){
      idata[i] = idata[i-1] + ny;
    }

  return idata;
}
```

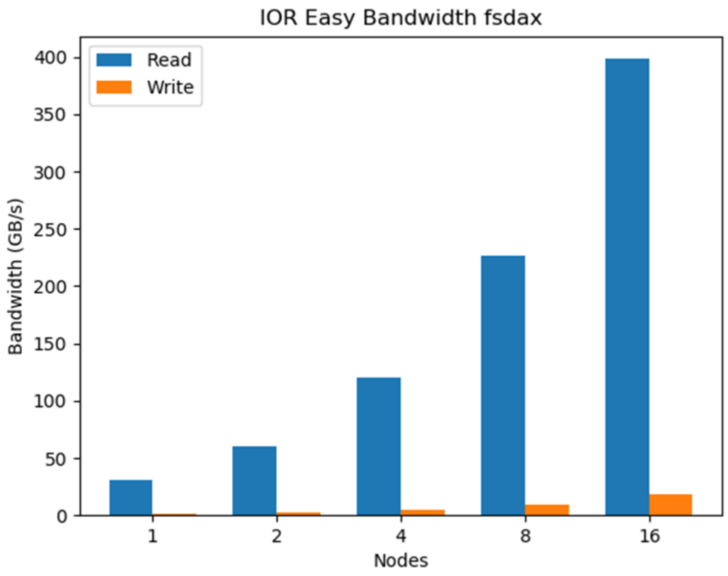
- Read-only data in DRAM

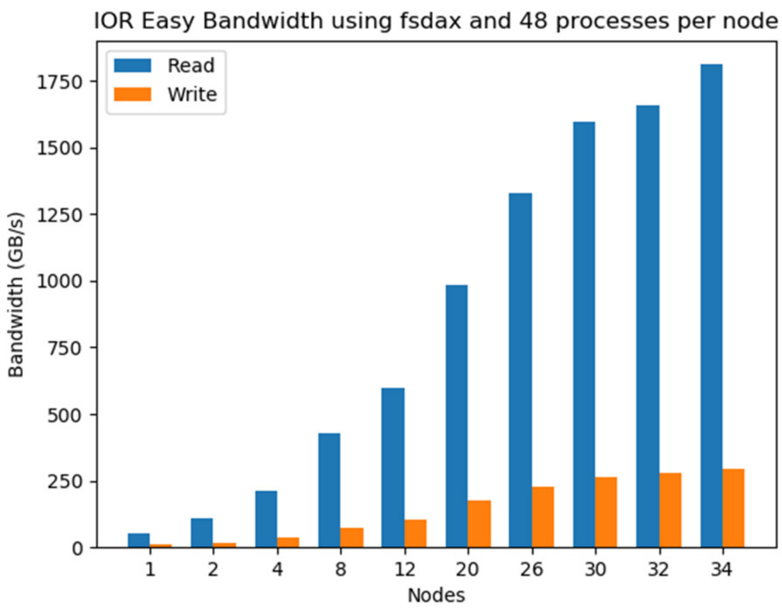
```
Calculation time was 14.269555 seconds
Overall run time was 16.397619 seconds
```

- Read-only data in B-APM

```
Calculation time was 14.250425 seconds
Overall run time was 16.800046 seconds
```

NUMA regions





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), "/");
```

Performance - workflows

Synthetic workflow runtime (Lustre vs NVM)		
Component	Lustre	NVM
Producer	96 secs	64 secs
Consumer	74 secs	30 secs
Total	170 secs	94 secs

Sequential data producer/consumer
Working set: 100GiB data
2 configurations:
write/read to Lustre, separate nodes
write/read to NVM, same node

44.70% faster

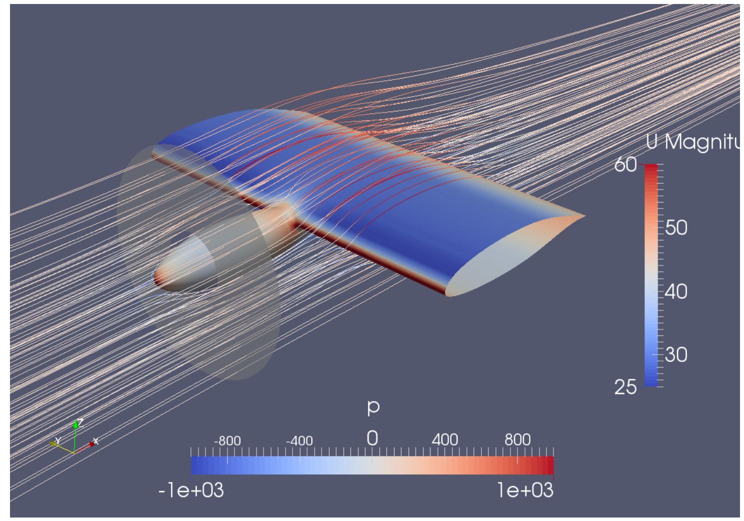
High Performance Conjugate Gradient (HPCG) Benchmark
Profile: CPU and memory-bound
Targets: Single node

Performance impact on HPCG due to concurrent data staging	
Component	Runtime
HPCG (no staging)	122 secs
HPCG + stage in	142 secs
HPCG + stage out	137 secs

12.29% slower

Performance – workflows

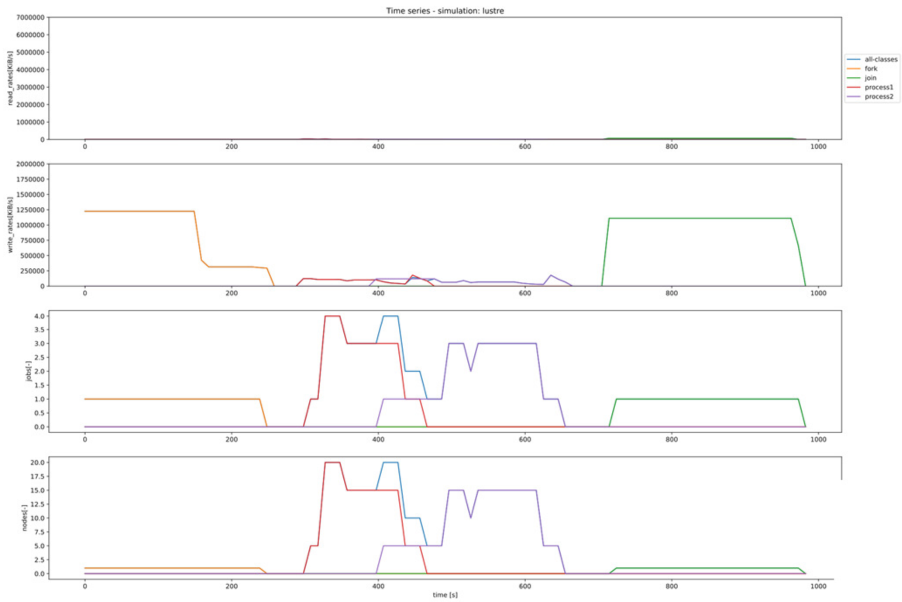
OpenFOAM simulation: *low-Reynolds number laminar turbulent transition modeling*
Input: mesh with ≈43M points
Stages: linear decomposition, parallel solver
768 MPI processes, 16 nodes
2 configurations:
① read/write to Lustre
② stage in, read/write on NVM, stage out

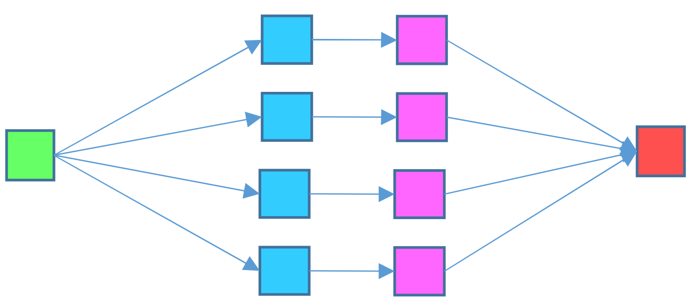


Performance benefits of data staging on OpenFOAM workflow

	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

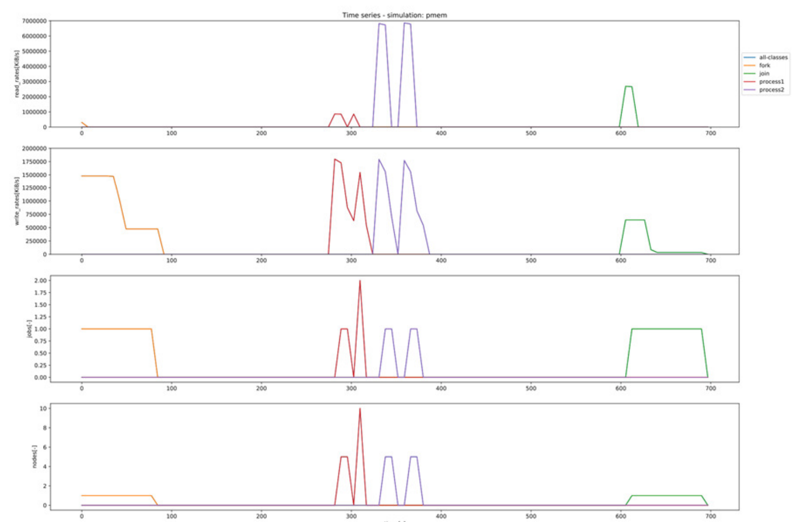




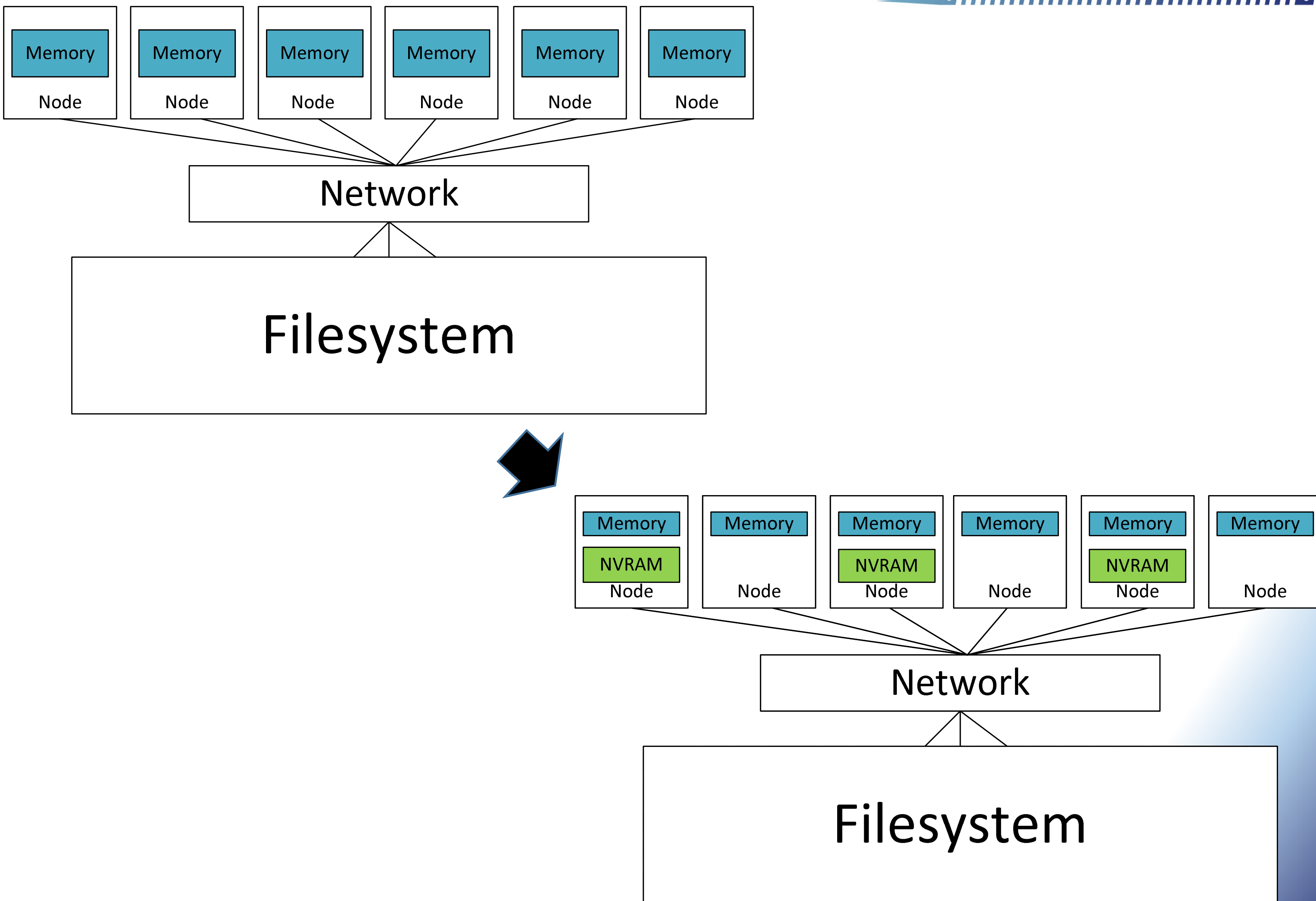
1 node
4 processes
4 files

20 nodes
80 processes
80 files

1 node
4 processes
80 files



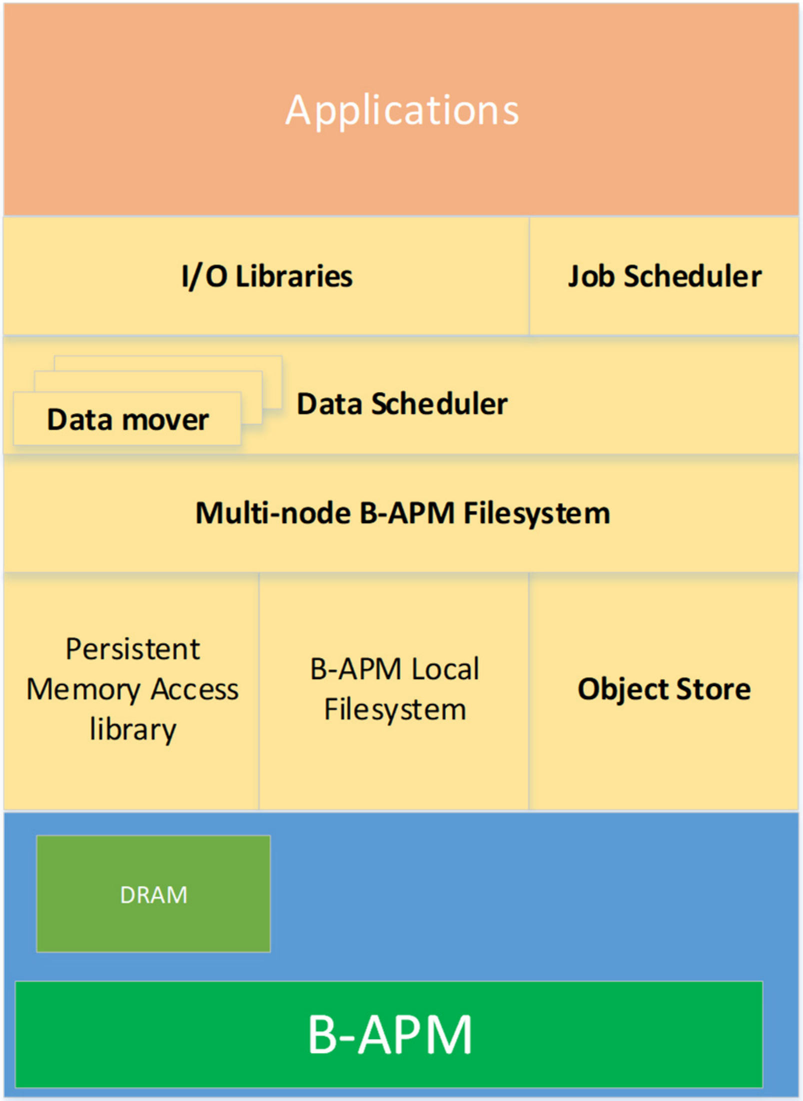
Exploiting distributed storage



Optimising data usage

- Reducing data movement
  - Time and associated energy cost for moving data too and from external parallel filesystems
  - Move compute to data
- Considering full scientific workflow
  - Data pre-/post-processing
  - Multi-physics/multi-application simulations
  - Combined simulation and analytics
- Enable scaling I/O performance with compute nodes

Systemware architecture



SLURM extensions

New options for srun, sbatch, salloc:

- SLURM tracks all workflow jobs; updating the prior- and post-dependencies and making sure they run in order
- If a workflow job fails; then all subsequent jobs fail (are deleted). Currently running jobs are not terminated

Option for job definition	Description
`#SBATCH --workflow-start`	Indicate that job starts a workflow
`#SBATCH --workflow-prior-dependency=JOBID+`	Make job depend on completion of prior jobs
`#SBATCH --workflow-end`	Indicate that job finalizes workflow
`#SBATCH --workflow-same-nodes`	Indicate the job should use the same nodes assigned to its prior dependent job

SLURM extensions

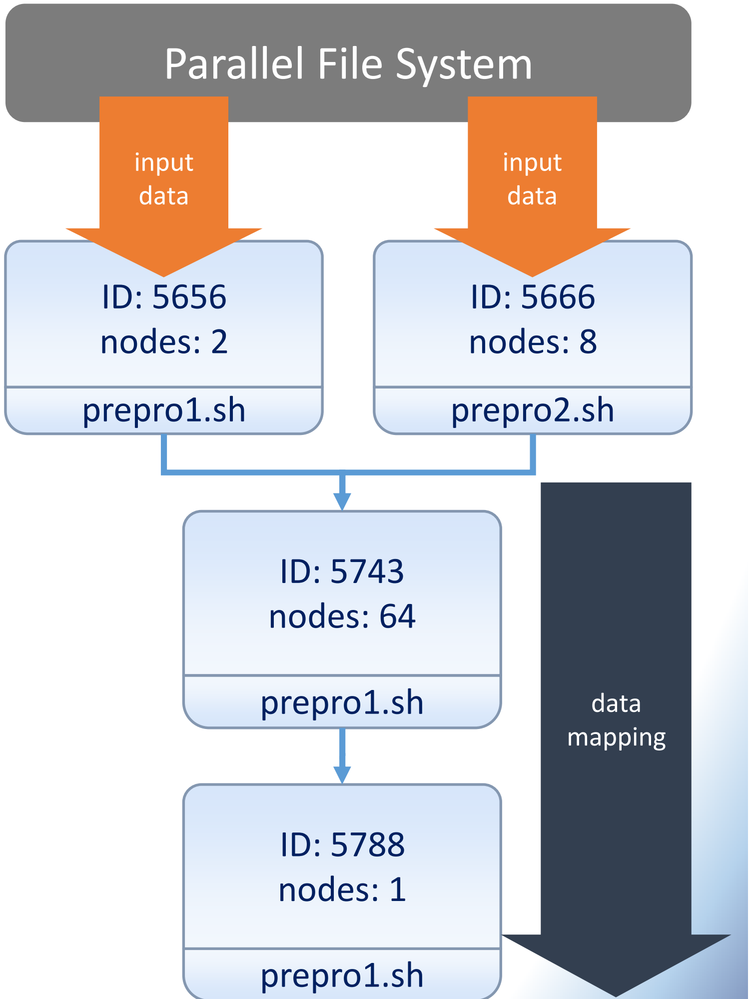
New options for data management:

- SLURM captures the dependencies and initiates the appropriate NORNS tasks to fulfill the transfers requested by users

Option for job definition	Description
`#NORNS stage_in origin destination mapping`	Stage in data from ORIGIN dataspace into DESTINATION according to a predefined MAPPING
`#NORNS stage_out origin destination mapping`	Stage out data from ORIGIN dataspace into DESTINATION according to a predefined MAPPING
`#NORNS persist [store|delete|share|unshare] location user`	Allow jobs to store data in node-local storage so that it can be shared among workflow jobs

Example: Definition of a workflow

```
[ bsc15455@mn1.bsc.es: ~ ] $
sbatch --nodes=2 \
       --workflow-start prepro1.sh

Job ID: 5656

[ bsc15455@mn1.bsc.es: ~ ] $
sbatch --nodes=8 \
       --workflow-start prepro2.sh

Job ID: 5666

[ bsc15455@mn1.bsc.es: ~ ] $
sbatch \
  --nodes=64 \
  --workflow-prior-dependency=5656,5666 \
  simulation.sh

Job ID: 5743

[ bsc15455@mn1.bsc.es: ~ ] $
sbatch --nodes=1 \
       --workflow-prior-dependency=5743 \
       --workflow-end postpro.sh

Job ID: 5788
```



Example: Resource mappings

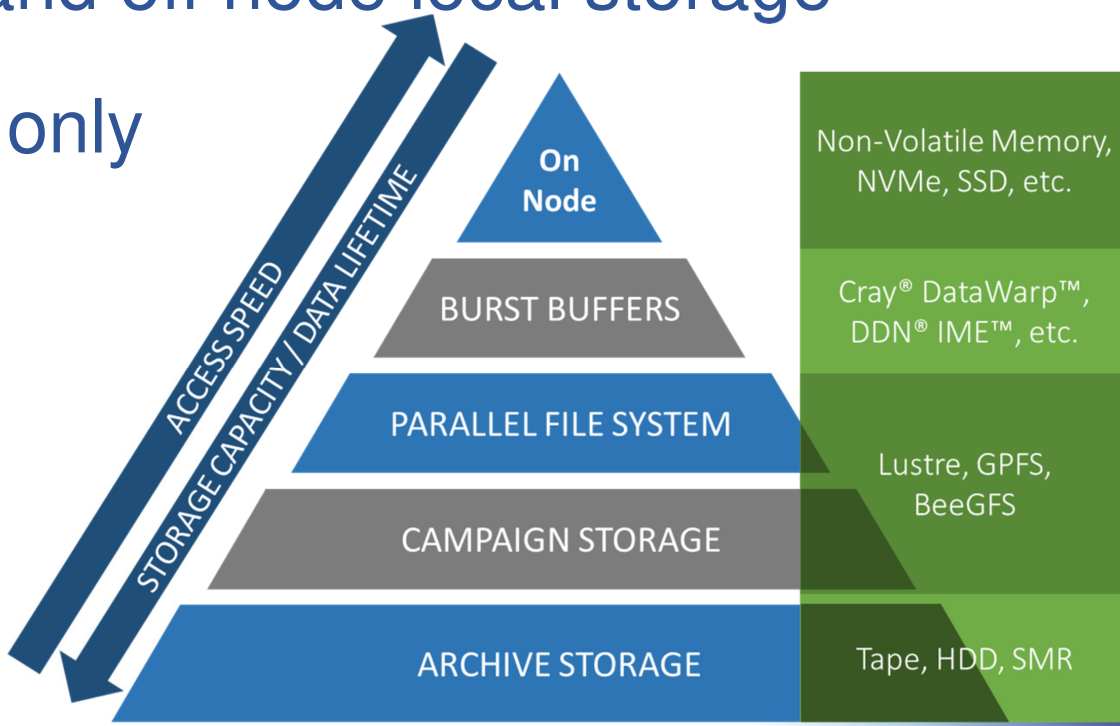
mapping.dat

```
### INPUT/OUTPUT FILE MAPPINGS ###
['lustre://${HOME}/file/dir/checkpoint%[0-9]+%.out'];
0;pmdk0://;0,5
1;pmdk0://;1,6
2;pmdk0://;2,7
3;pmdk0://;3,8
4;pmdk0://;4,9,11
```

```
[ bsc15455@ngio-login1: ~ ] $ dsh -f nodes.list -c ls /mnt/pmdk0
ngio-cn00: checkpoint0.out  checkpoint5.out
ngio-cn01: checkpoint1.out  checkpoint6.out
ngio-cn02: checkpoint2.out  checkpoint7.out
ngio-cn03: checkpoint3.out  checkpoint8.out
ngio-cn04: checkpoint4.out  checkpoint9.out  checkpoint11.out
```

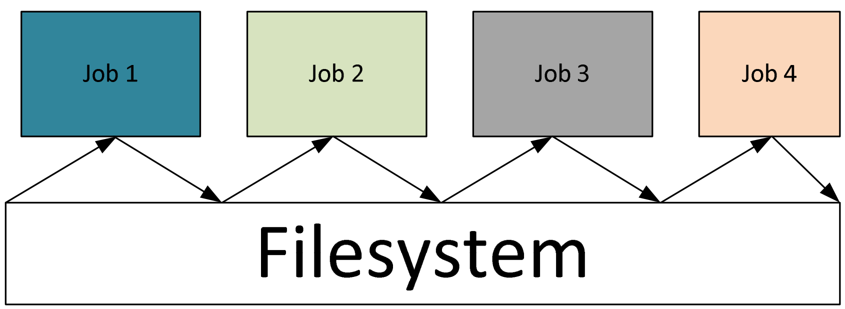
Challenges of compute local storage

- No single namespace
- Enabling workflow jobs to run on same set of nodes
- Moving data on and off node local storage
- Ensuring data is only accessible by authorised users
- Understanding application performance



Using distributed storage

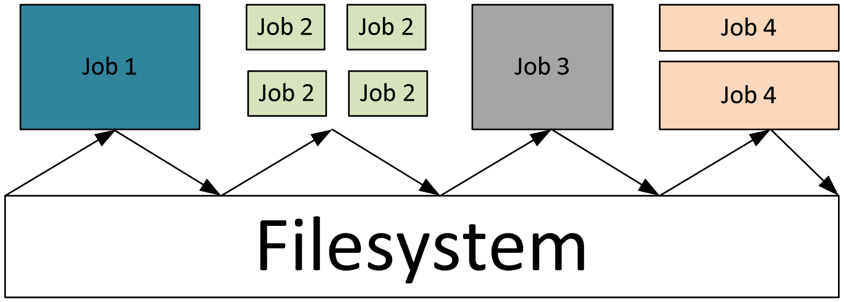
- New usage models
  - Resident data sets
    - Sharing preloaded data across a range of jobs
    - Data analytic workflows
    - How to control access/authorisation/security/etc….?
  - Workflows
    - Producer-consumer model



    - Remove filesystem from intermediate stages

Using distributed storage

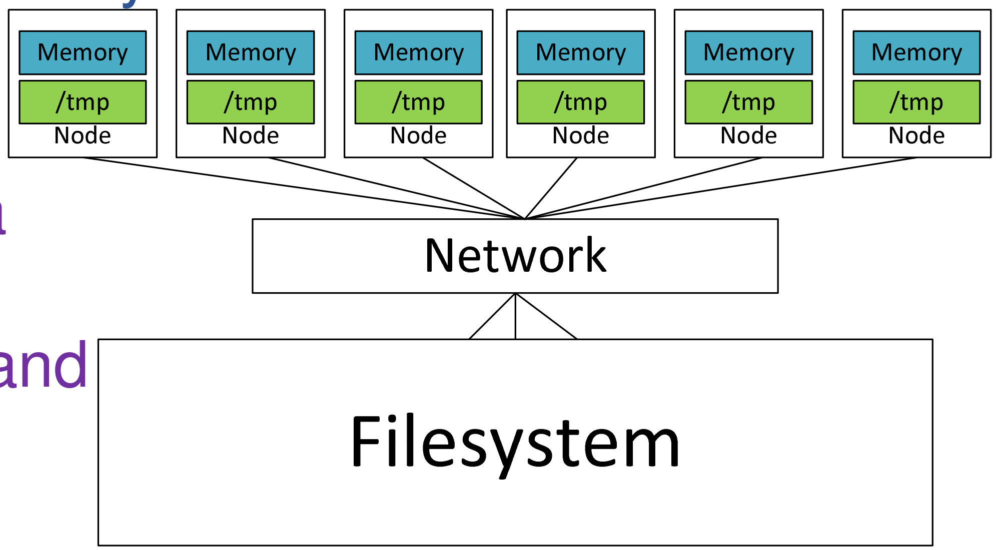
- Workflows
  - How to enable different sized applications?



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

Using distributed storage

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



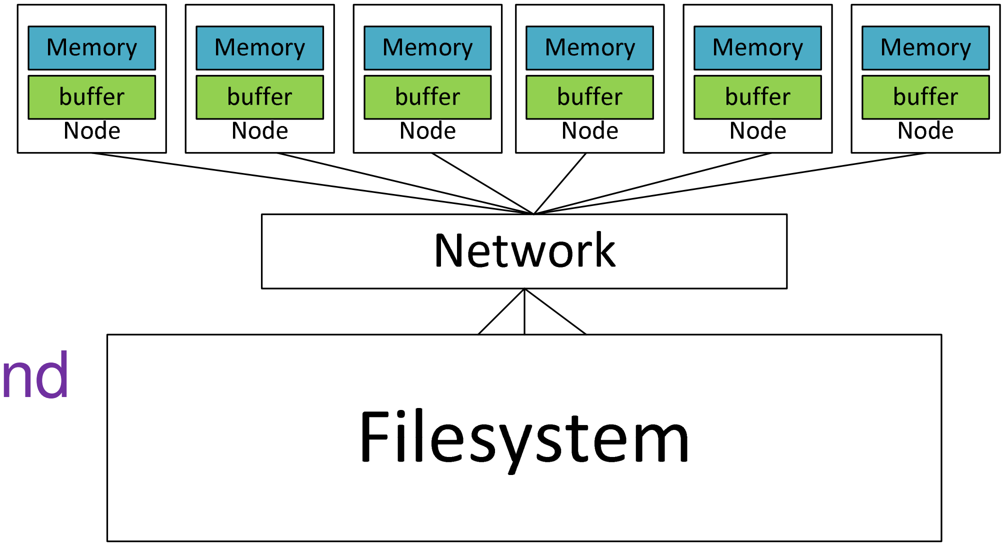
NGIO Data Scheduler (NORNS) and Slurm integration

- 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

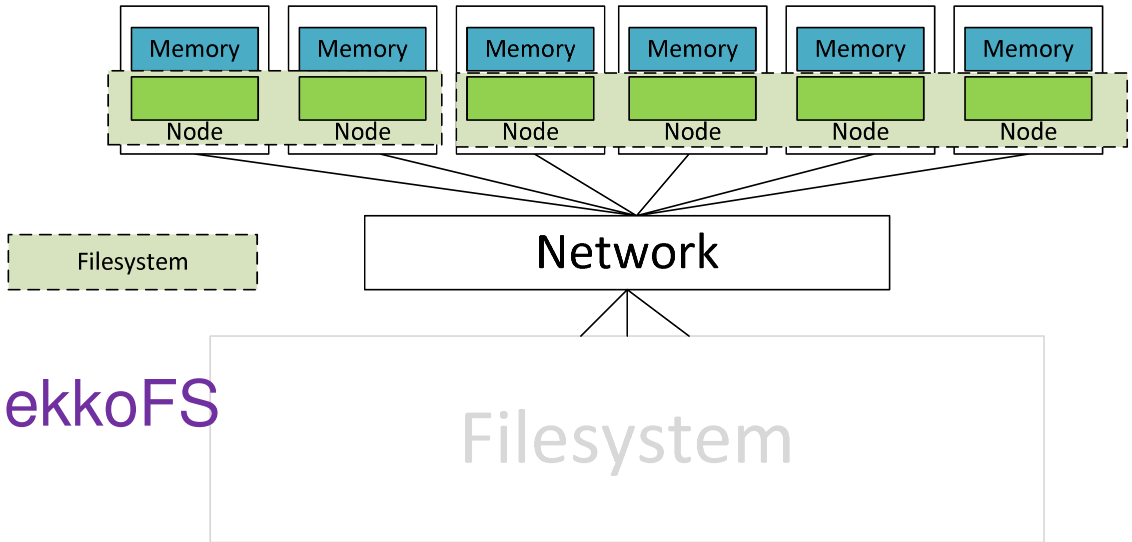
NGIO Data Scheduler (NORNS) and Slurm integration



- 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

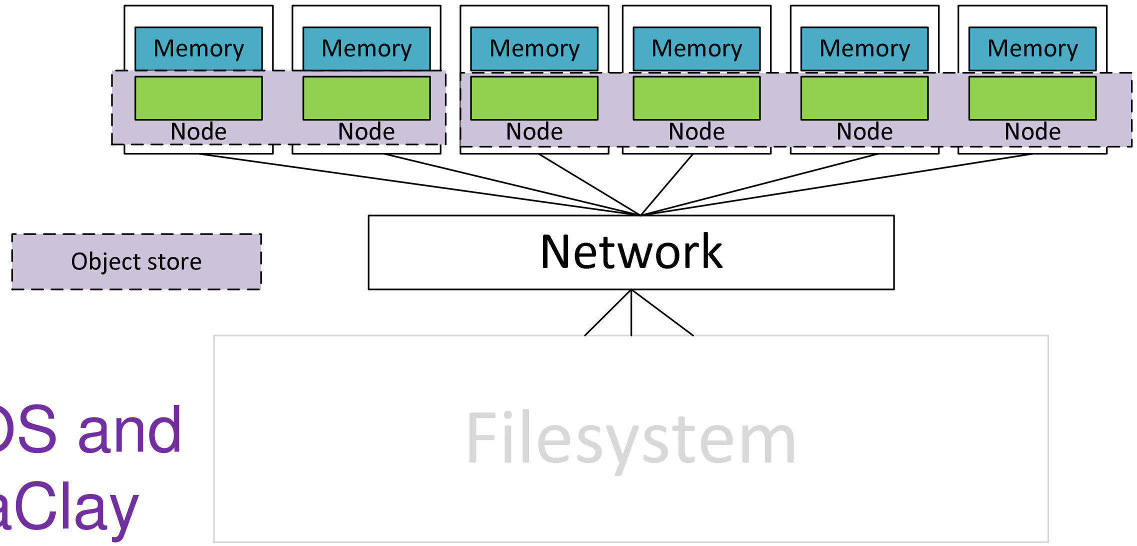


NGIO GekkoFS

  - 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

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
```

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

Summary

- Enabling new technologies with HPC systems requires systemware support
- Transparently handling data for applications requires integration with job schedulers and data storage targets
- Tools are essential to allow exploitation of new hardware without requiring code change
- Tools very useful to evaluate design decision and approaches for applications and systems
- In-node B-APM is potentially very powerful for performance, but will require some changes to use efficiently (either at the systemware level or the application level)