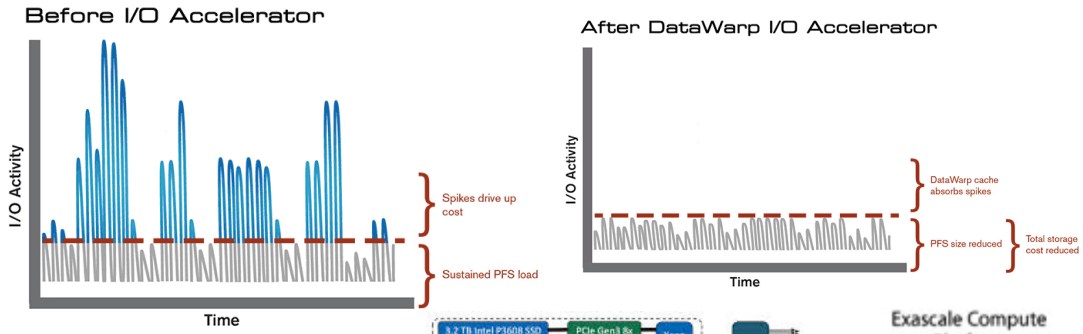
Non-volatile Memory

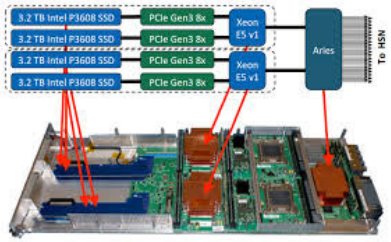
The changing face of HPC technology

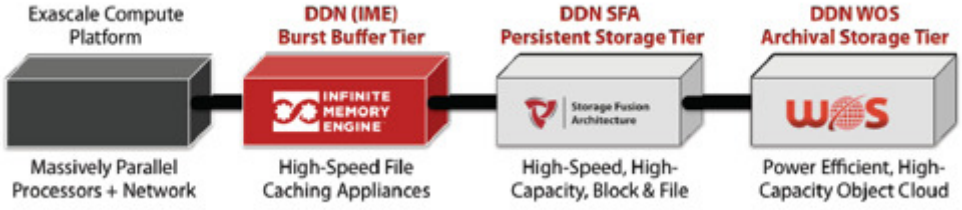
Adrian Jackson
Research Architect
EPCC, The University of Edinburgh
a.jackson@epcc.ed.ac.uk
@adrianjhpc

Burst Buffer

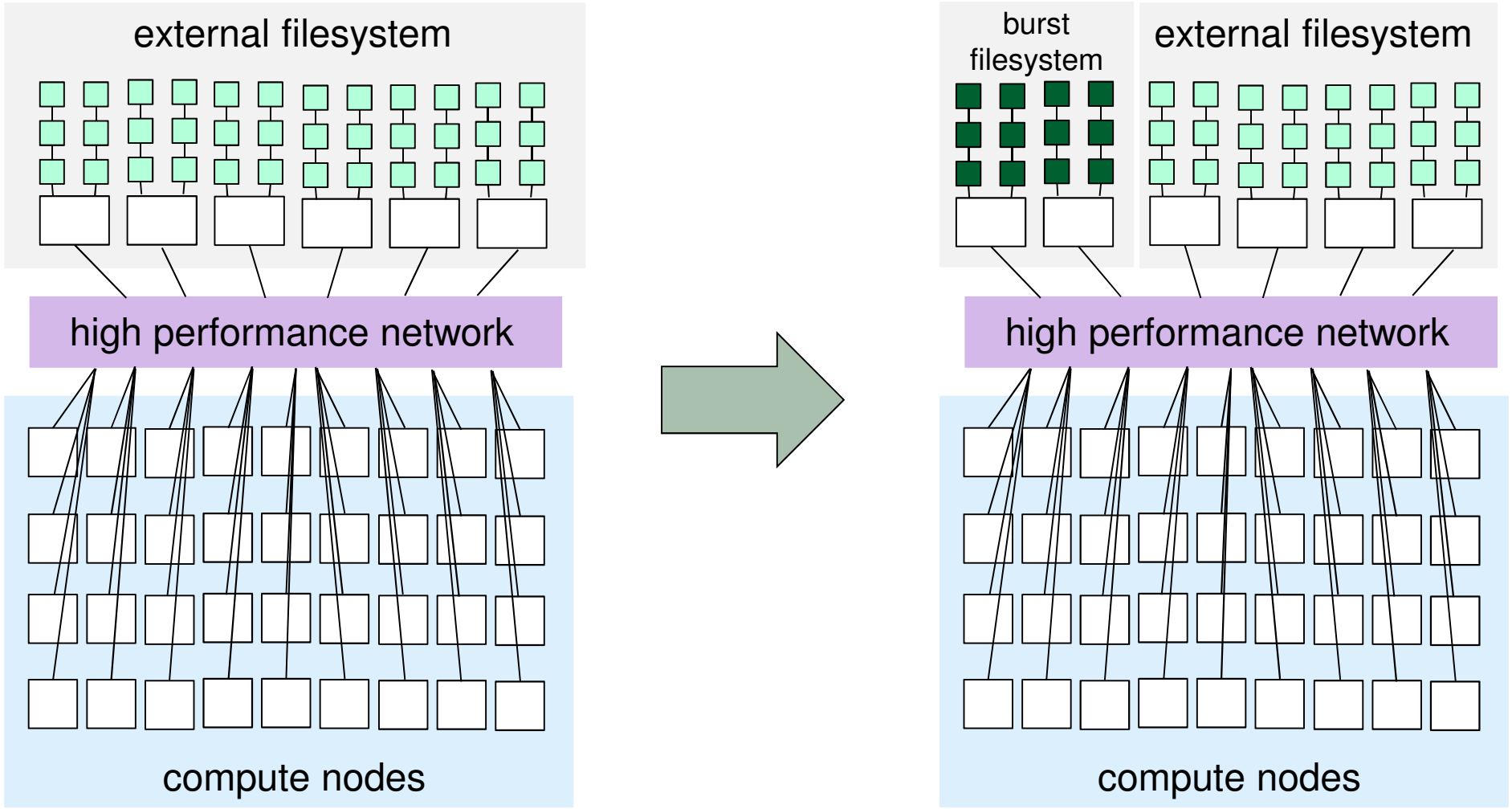
- Non-volatile already becoming part of HPC hardware stack
- SSDs offer high I/O performance but at a cost
  - How to utilise in large scale systems?
- Burst-buffer hardware accelerating parallel filesystem
  - Cray DataWarp
  - DDN IME (Infinite Memory Engine)







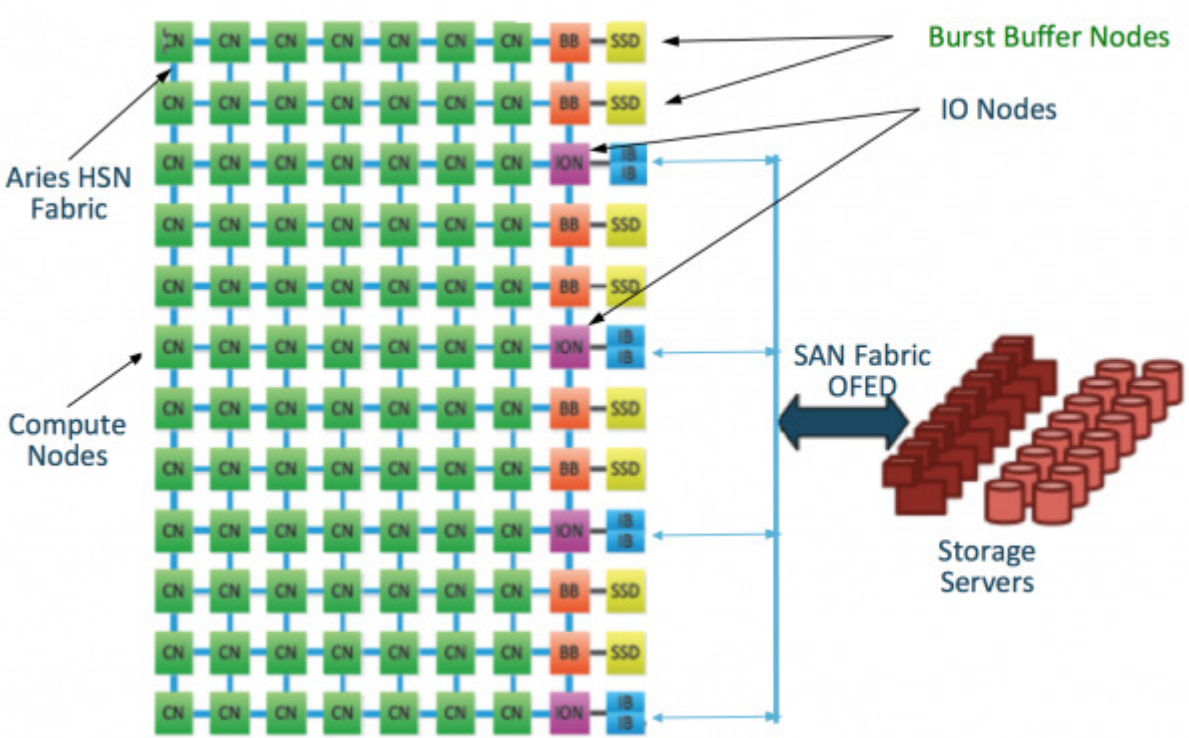
Burst buffer



Burst buffer

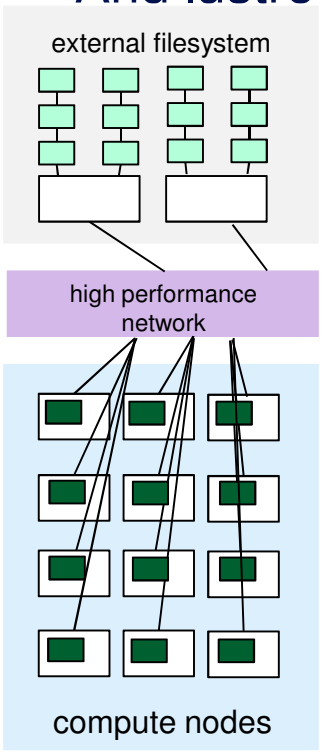
- Burst buffer is (generally) separate resource to filesystem
- Managing usage/allocation/data movement is user responsibility
- Storage has compute
- Resource is external
- Scheduling can be separate
  - Usage not required

NERSC Cori system



Moving beyond burst buffer

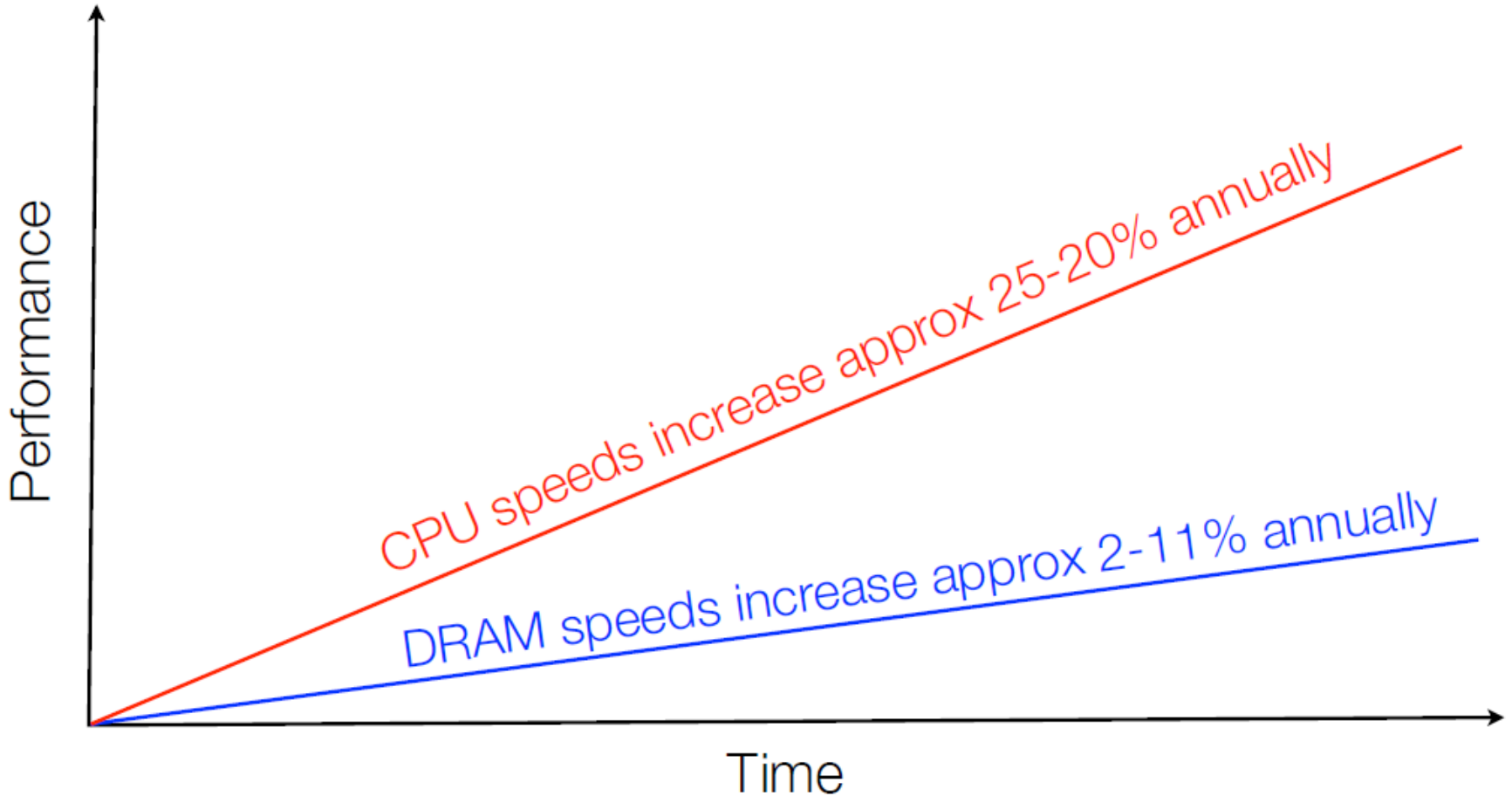
- Non-volatile is coming to the node rather than the filesystem
- Argonne Theta machine has 128GB SSD in each compute node
  - And lustre



Non-volatile memory

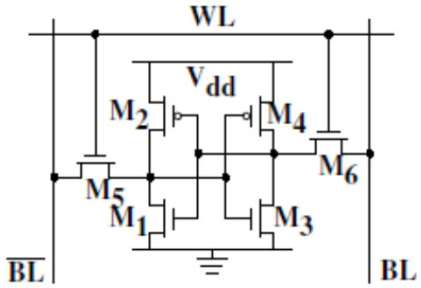
- Non-volatile RAM
  - 3D XPoint technology
  - STT-RAM
- Much larger capacity than DRAM
  - Hosted in the DRAM slots, controlled by a standard memory controller
- Slower than DRAM by a small factor, but significantly faster than SSDs
- STT-RAM
  - Read fast and low energy
  - Write slow and high energy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

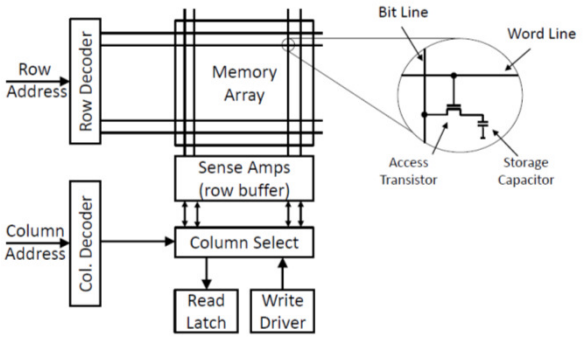
Performance Trend

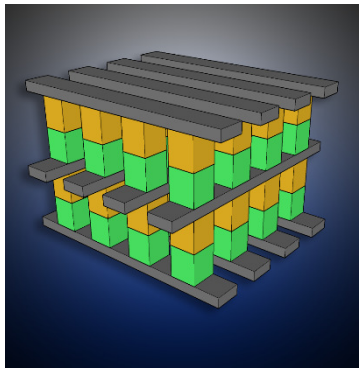


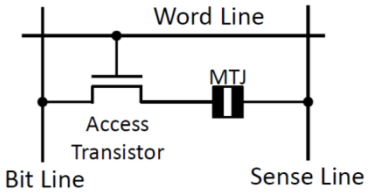
SRAM vs NVRAM

- 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









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

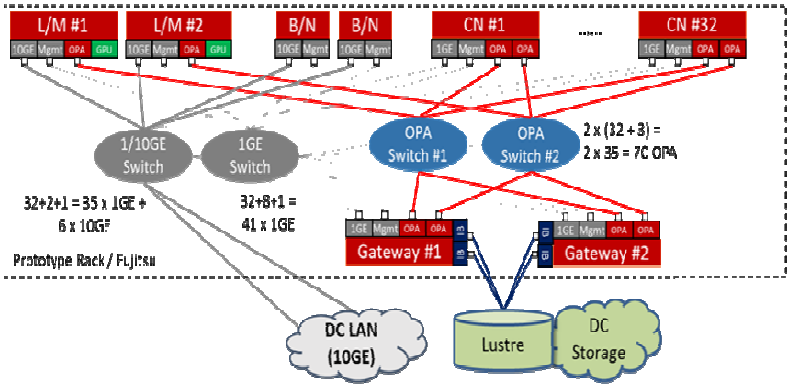
Prototype

2 Ethernet Switches

2 Boot Nodes

2 Login Nodes

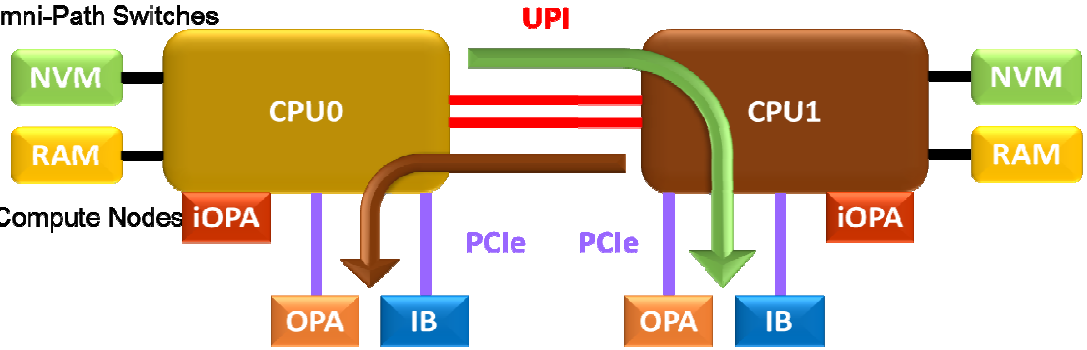
2 Gateway Nodes



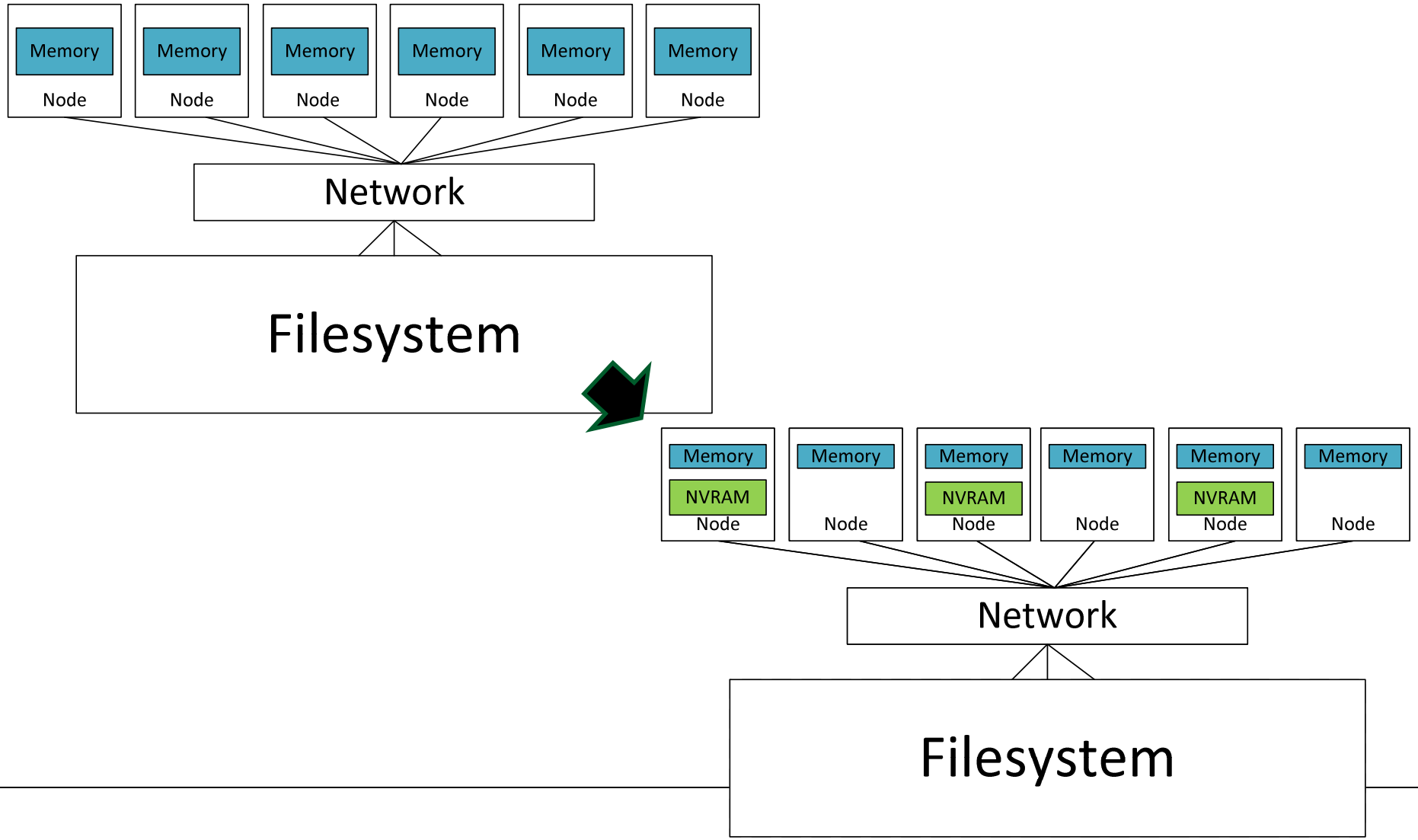
16 Compute Nodes

2 Omni-Path Switches

16 Compute Nodes



Exploiting distributed storage



NVRAM

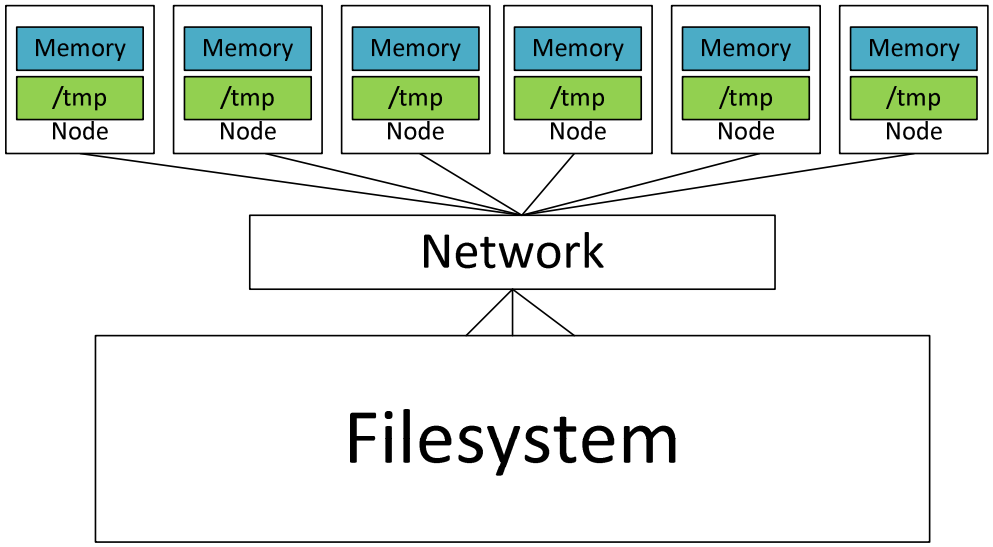
- The “memory” usage model allows for the extension of the main memory
  - The data is volatile like normal DRAM based main memory
- The “storage” usage model which supports the use of NVRAM like a classic block device
  - E.g. like a very fast SSD
- The “application direct” usage model maps persistent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

Slide 12

WM1 Can we talk about this? If so, can we use the diagrams from D6.1 (Figures 5, 6 & 7)?
WEILAND Michele, 31/10/2016

Using distributed storage

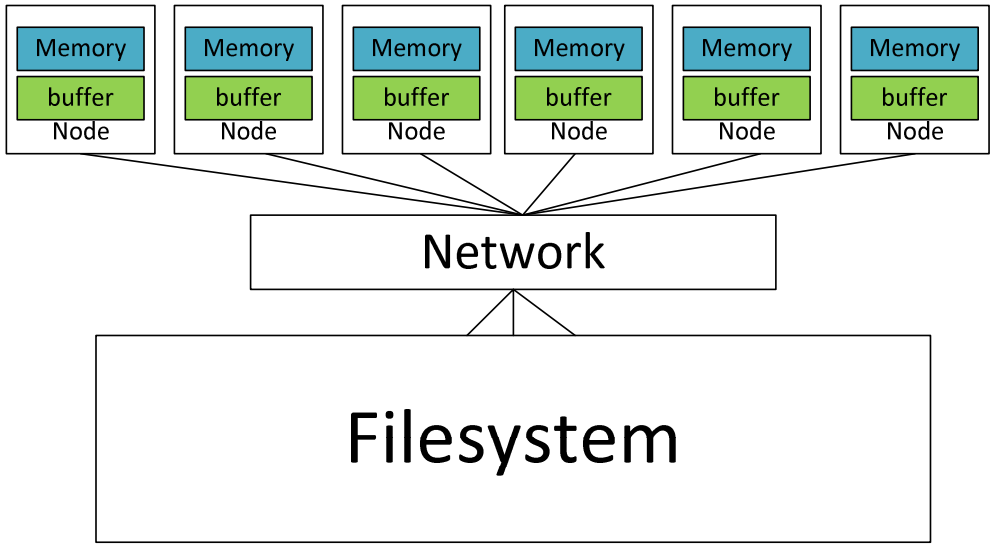
- Without changing applications
  - Large memory space/in-memory database etc…
  - Local filesystem



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

Using distributed storage

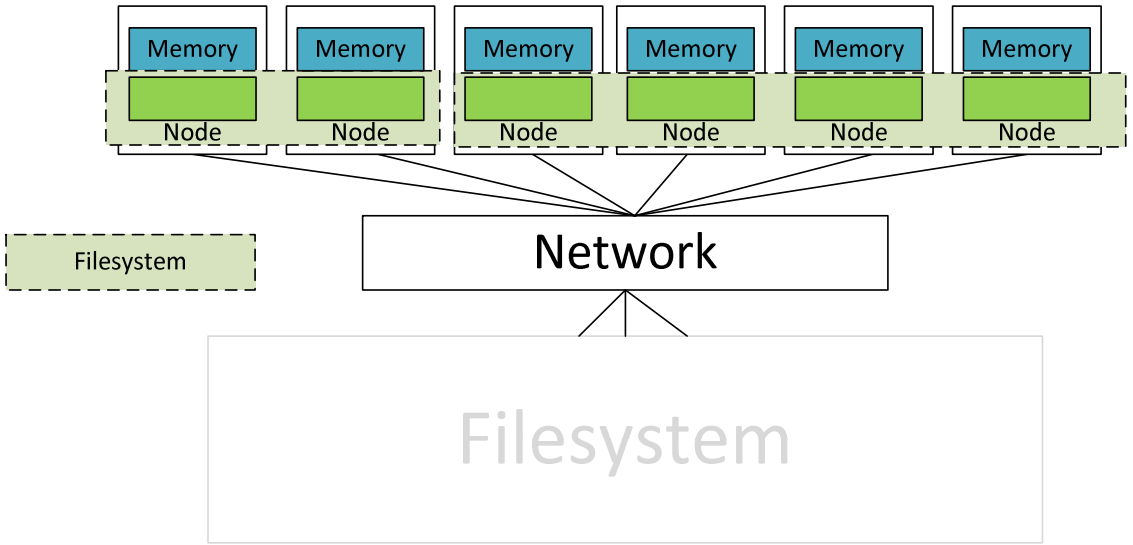
- Without changing applications
  - Filesystem buffer



  - 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

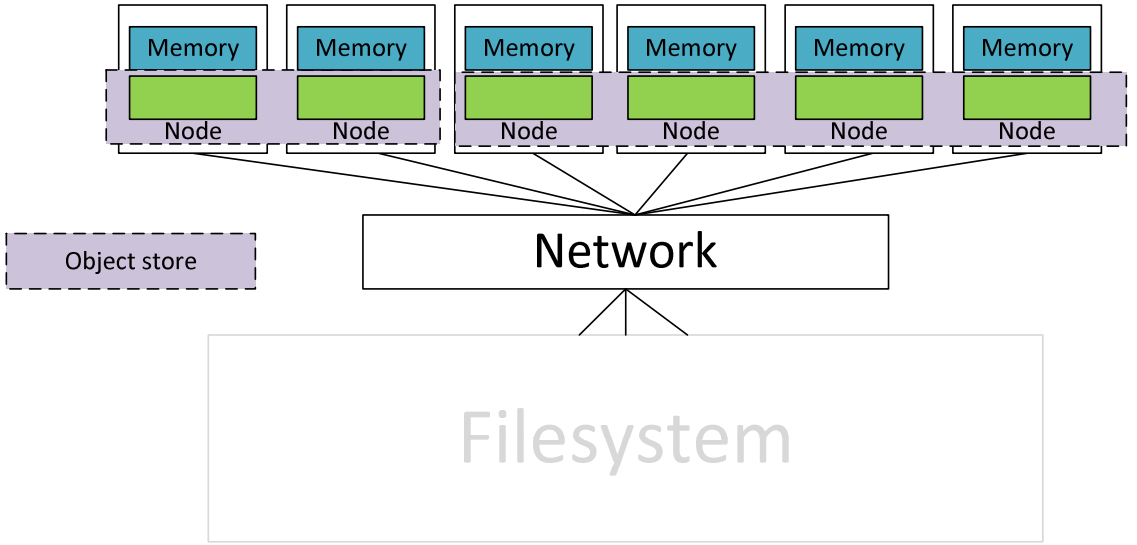
- Without changing applications
  - Global filesystem



  - 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



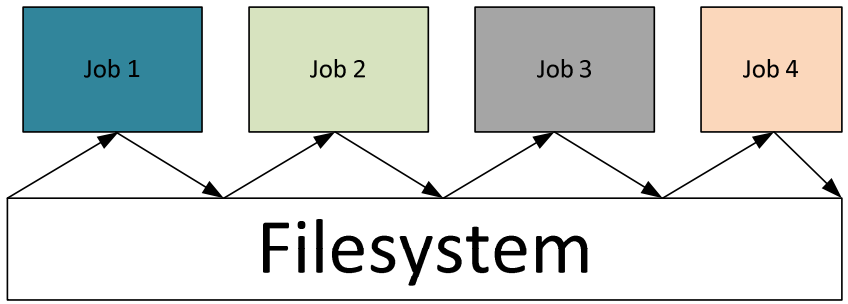
  - Needs same functionality as global filesystem
  - Removes need for POSIX, or POSIX-like functionality

Using distributed storage

- Without changing applications
  - Automatic checkpointing
    - Resiliency
    - Local checkpointing without hitting the filesystem
  - Pause and restart
    - Just-in-time scheduling/high priority jobs
    - Waiting for something else to happen…

Using distributed storage

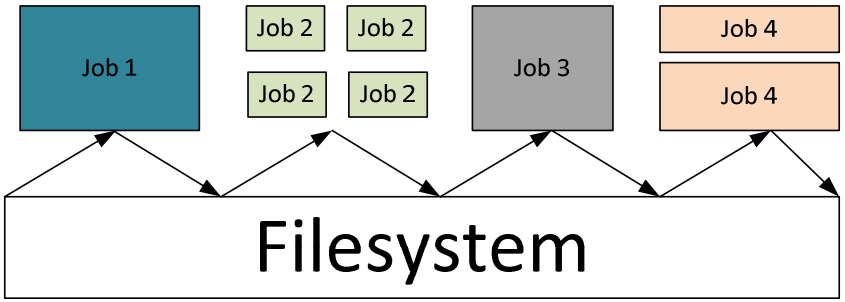
- New usage models
  - Resident data sets
    - Sharing preloaded data across a range of jobs
    - Data analytic workflows
    - How to control access/authorisation/security/etc….?
  - Workflows
    - Producer-consumer model



    - 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

epcc

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

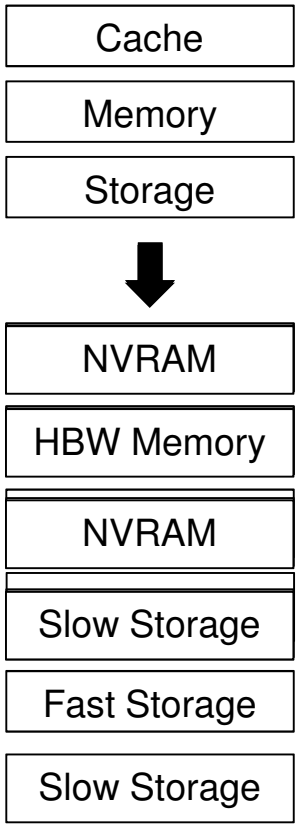
Potential solutions

- Large memory space
- Burst buffer
- Filesystem across NVRAM in nodes
  - echofs
- HSM functionality
  - Data scheduling based on hints from applications
- Object store across nodes
  - DAOS etc…
- Checkpointing and I/O libraries
  - Transparent I/O stack management


Diagram Legend
Complete Compute Nodes

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
- NVRAM main memory
  - High capacity, ultra fast storage
  - Low energy (when at rest) but still slower than DRAM



Implications for applications

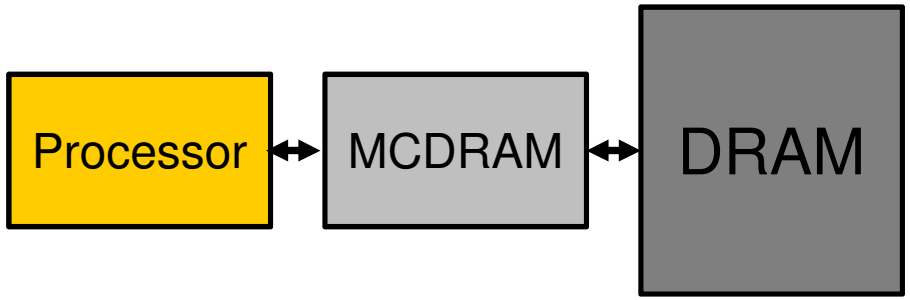
- Future looks bright for traditional high Flop/s, caching compatible applications
  - On processor memory used for large scale caching
  - On node NVRAM for large resident sets
  - Moving back to Flop rather than Bandwidth bound performance
- What about latency bound/sparse access pattern applications
  - High bandwidth memory doesn't help
  - NVRAM on node may bring some benefits
  - Possibly look for adaptive memory on processor
  - Variable NVRAM rather than high bandwidth memory
  - Tune memory usage for applications?
- What's important, performance or throughput?

Memory

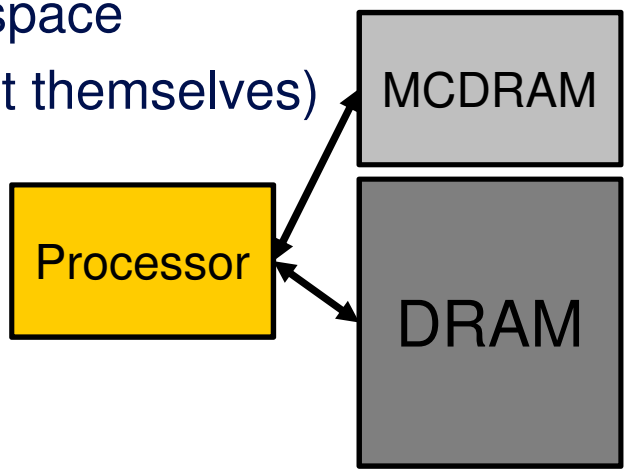
- Two levels of memory for KNL
  - Main memory
    - KNL has direct access to all of main memory
    - Similar latency/bandwidth as you'd see from a standard processors
    - 6 DDR channels
  - MCDRAM
    - High bandwidth memory on chip: 16 GB
    - Slightly higher latency than main memory (~10% slower)
    - 8 MCDRAM controllers/16 channels

Memory Modes

- Cache mode
  - MCDRAM cache for DRAM
  - Only DRAM address space
  - Done in hardware (applications don't need modified)
  - Misses more expensive (DRAM and MCDRAM access)



- Flat mode
  - MCDRAM and DRAM are both available
  - MCDRAM is just memory, in same address space
  - Software managed (applications need to do it themselves)
- Hybrid – Part cache/part memory
  - 25% or 50% cache



numactl in flat mode

- Example code:
  - Check available memory

```
[adrianj@esknl1 ~]$ aprun -n 1 numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170
171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212
213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254
255
node 0 size: 98178 MB
node 0 free: 81935 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15924 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

numactl in flat mode

- Specifying memory policy
  - Mandating the memory to use
    - -m or --membind=
    - Fails if exhausts memory

```
mpirun –n 64  numactl –m 1 ./castep.mpi forsterite
```

  - Setting the preferred memory
    - -p or --preferred=
    - Tries to used preferred memory, falls back if exhausts memory

```
mpirun –n 64  numactl –p 1 ./castep.mpi forsterite
```

Memkind and hbwmalloc

- Open source software available that can manage MCDRAM
  - memkind and hbwmalloc
  - https://github.com/memkind
- memkind
  - built on jemalloc
  - heap manager
  - allows defining memory “kind”s
- hbwmalloc
  - Implements memory model for knl using memkind
  - Predefines kind options for easy knl usage
  - Can have selective fallback behaviour
  - Can target different page sizes (2MB – 1GB)

hbwmalloc for C and C++

- Allocate arrays in C:

```
#include<hbwmalloc.h>
…
double *a = (double *) malloc(sizeof(double)*2048);
double *b = (double *) hbw_malloc(sizeof(double)*2048);
```

  - Need to link to memkind library
  - Automatic variables/arrays are in main memory
    - May need to restructure data to deal with this
  - Also need to free with:
    - `hbw_free`
  - Also `hbw_calloc`
- Fallback
  - If not enough memory available it will fallback
  - Three different options
    - HBW_POLICY_BIND
      - Fail
    - HBW_POLICY_PREFERRED (default)
      - Fallback to main memory
    - HBW_POLICY_INTERLEAVE
      - Interleave pages between MCDRAM and DDR
  - `hbw_set_policy()` sets the current fallback policy
  - `hbw_get_policy()` gets the current fallback policy

hbwmalloc for C and C++

- Can check for hbw from code

```
hbw_check_available()
```

- Can specify page size

```
hbw_posix_memalign_psize()
HBW_PAGESIZE_4KB
HBW_PAGESIZE_2MB
HBW_PAGESIZE_1GB
```

- Allocating in C++
  - STL allocator

```
#include<hbw_allocator.h>
…
std::vector<double, hbw::allocator<double> > a(100);
```

- Need to link to memkind for this too

MCDRAM from Fortran

- Intel directive for allocatable arrays

```
!DIR$ ATTRIBUTES FASTMEM :: …
```

- i.e.

```
real, allocatable(:) :: a, b
 …
!dir$ attributes fastmem :: a
…
allocate(a(1000))
allocate(b(20))
```

- Need to link with memkind otherwise will still use main memory and **not tell you**
- Only works for allocatable variables
- Also some restrictions on where it can be used (this may change/may have changed)
  - Not on components of derived types (might now be available in v17)
  - Not in common blocks

MCDRAM from Fortran

- Intel
  - `FASTMEM` is Intel directive
  - Only works for allocatable arrays
- Cray CCE:

```
!dir$ memory(attributes)
#pragma memory(attributes)
```

- Placed before allocation and deallocation statements
  - Fortran: allocate, deallocate (optional)
  - C/C++: malloc, calloc, realloc, posix_memalign, free
  - C++: new, delete, new[], delete[]
- Directive on deallocation must match (C/C++ only)
- Converts normal allocation to high-bandwidth memory
- The `bandwidth` attribute maps to MCDRAM

Cray CCE MCDRAM Support

```
integer, dimension(:,:), allocatable :: A
!dir$ memory(bandwidth) B
integer :: B(N)

!dir$ memory(bandwidth)
allocate(A(N,N))
```

- Allocation will fail (in above examples) if MCDRAM is unavailable/exhausted
- More general form can deal with this:

  `!dir$ memory([fallback,] attributes)`
  - i.e. `!dir$ memory(fallback,bandwidth)` will fall back to DDR if MCDRAM isn't available

autohbw

- Automatic allocator, part of memkind
  - Intercepts heap allocates and performs them on MCDRAM
  - Can be tuned for different size limits
  - Link to the library and set environment variables:

```
export AUTO_HBW_SIZE=1K:5K
export AUTO_HBW_SIZE=10K
```

  - Needs to be linked before system libraries, i.e. LD_PRELOAD or `equivalent`