



(SOME) EUROPEAN EXASCALE PROJECTS

Adrian Jackson

@adrianjhpc

a.jackson@epcc.ed.ac.uk





Exascale I/O with Storage Class Memory

Warning!



- Terminology will be annoying:
 - NVDIMM
 - NVRAM
 - SCM
 -
- 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

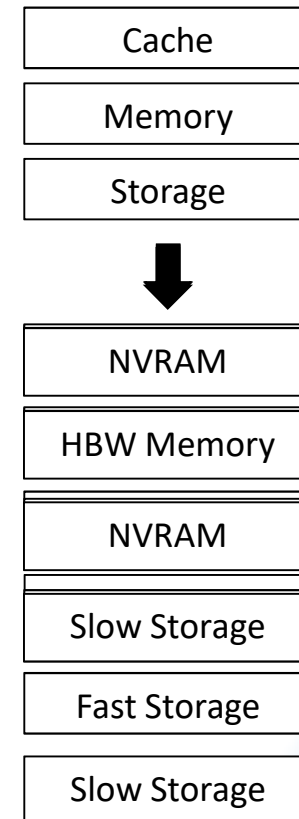
- EPCC
- INTEL
- FUJITSU
- BSC
- TUD
- ALLINEA
- ECMWF
- ARCTUR



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
- Storage class 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

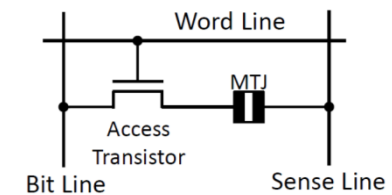
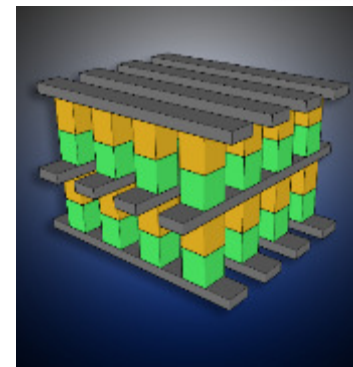
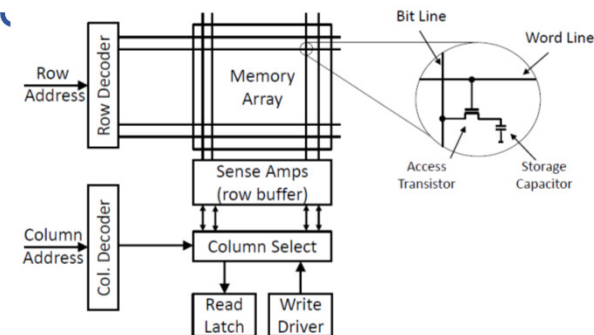
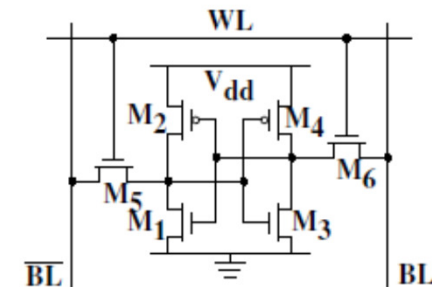


- Non-volatile RAM
 - 3D XPoint technology
 - STT-RAM
- Much larger capacity than DRAM
 - Hosted in the DRAM slots, controlled by a standard memory controller
- Slower than DRAM by a small factor, but significantly faster than SSDs
- STT-RAM
 - Read fast and low energy
 - Write slow and high energy
 - 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)



NVDIMMs

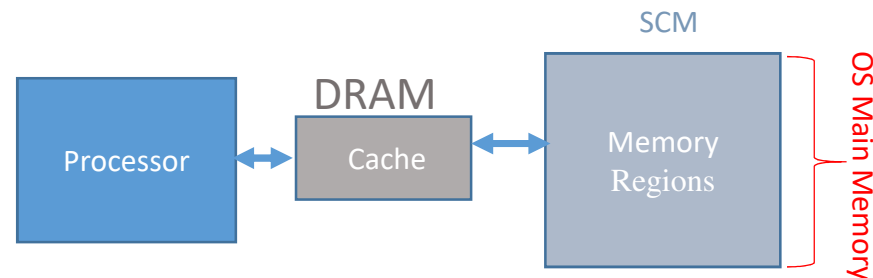


- Non-volatile memory already exists
 - NVDIMM-N:
 - DRAM with NAND Flash on board
 - External power source (i.e super capacitors)
 - Data automatically moved to flash on power failure with capacitor support, moved back when power restored
 - Persistence functionality with memory performance (and capacity)
 - 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, when it comes
 - 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

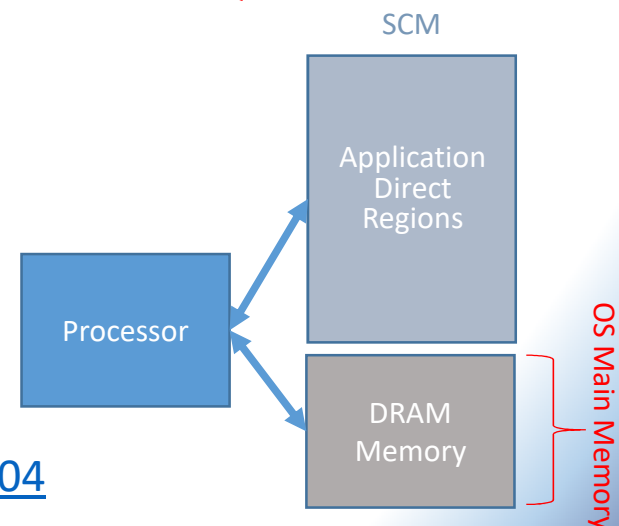
Memory levels



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



- One-level memory (1LM)



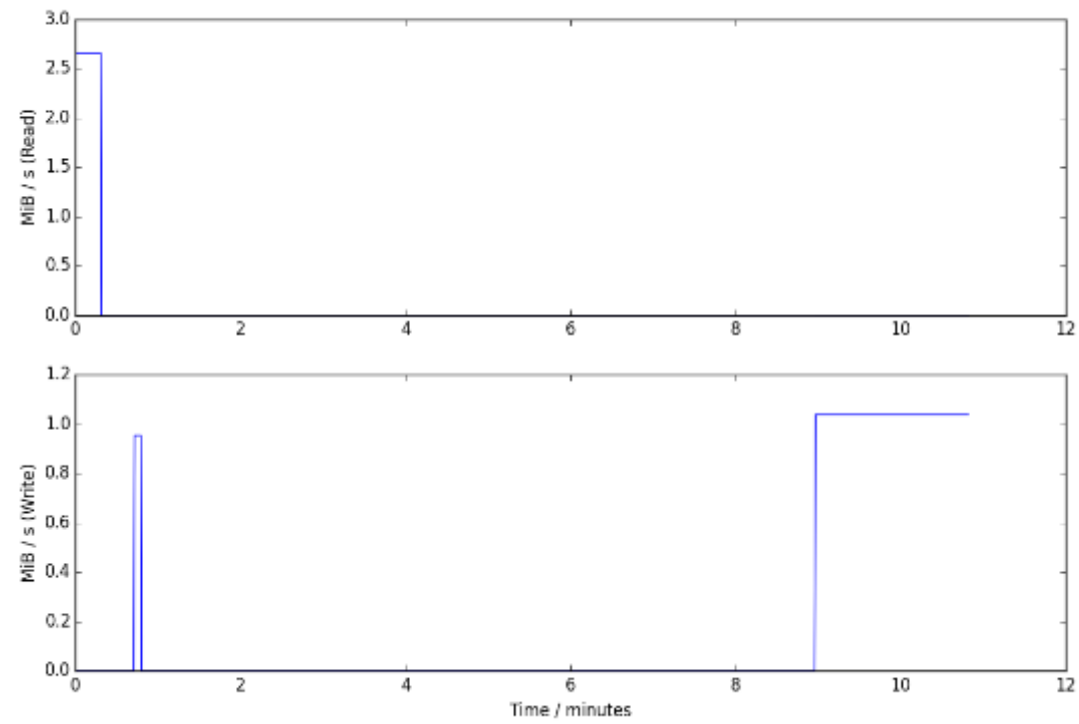
*<https://www.google.com/patents/US20150178204>

Storage Class Memory

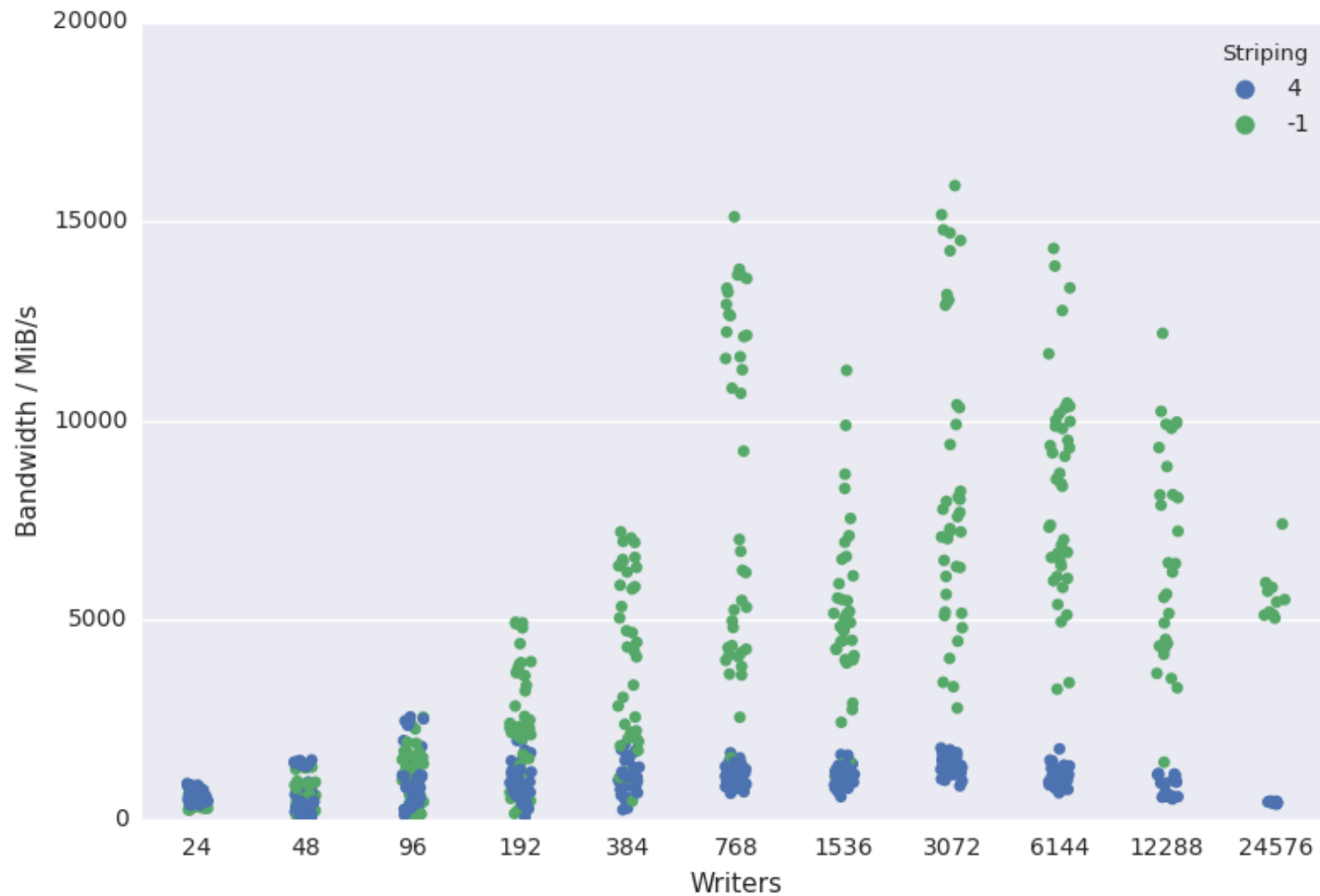


- The “memory” usage model allows for the extension of the main memory
 - The data is volatile like normal DRAM based main memory
- The “storage” usage model which supports the use of NVRAM like a classic block device
 - E.g. like a very fast SSD
- The “application direct” (DAX) usage model maps persistent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

I/O

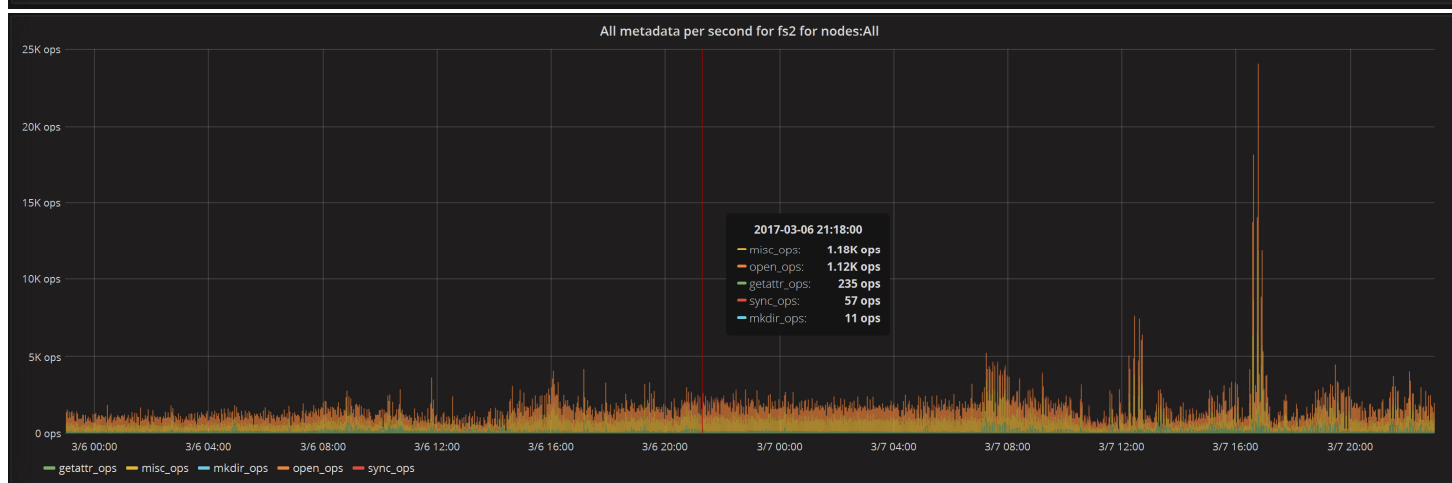
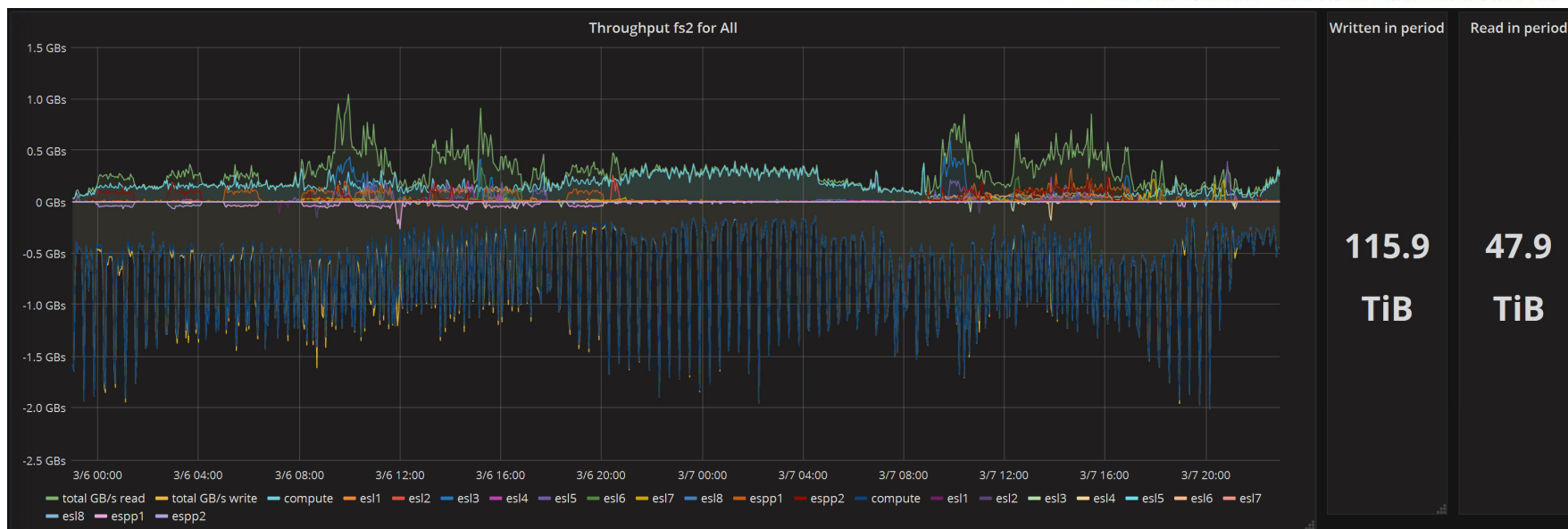


I/O Performance



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

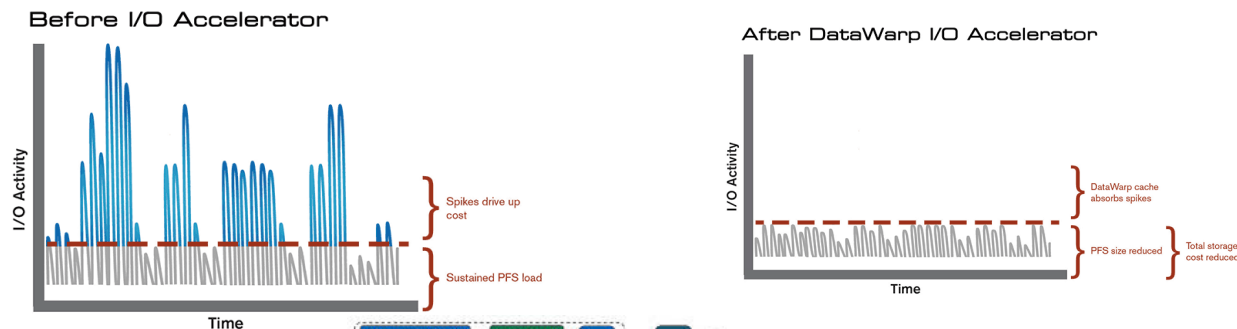
ARCHER workload



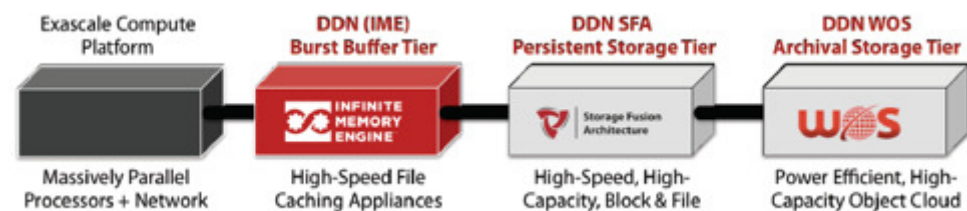
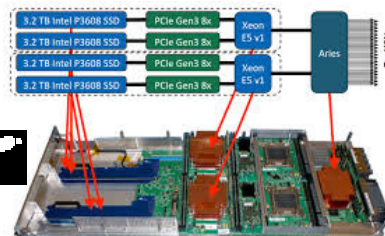
Burst Buffer



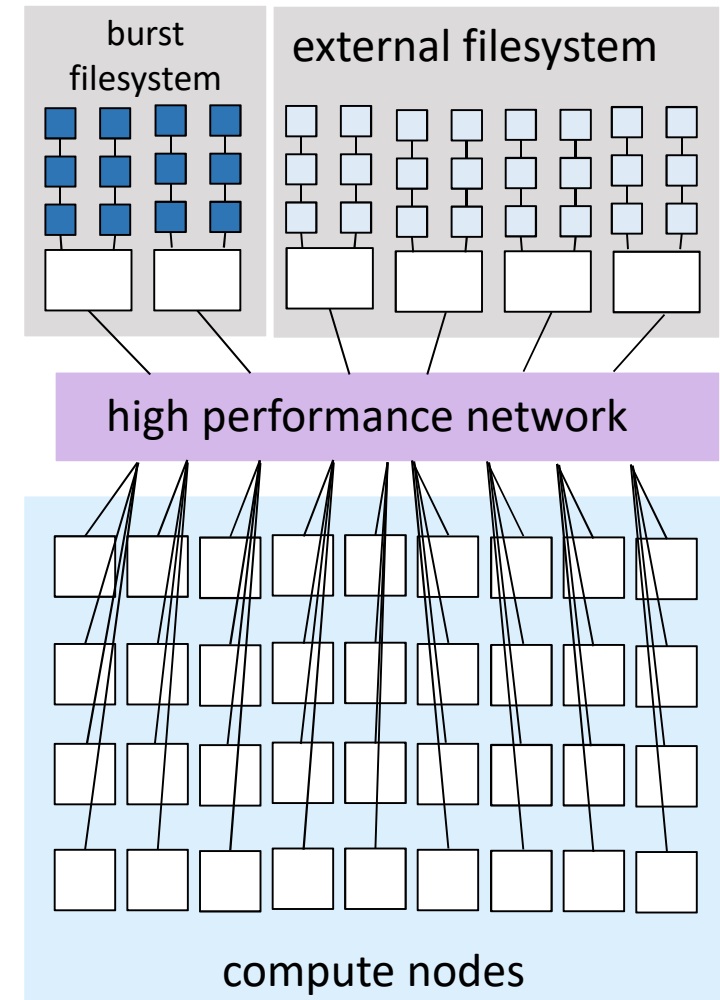
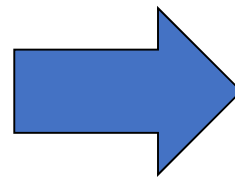
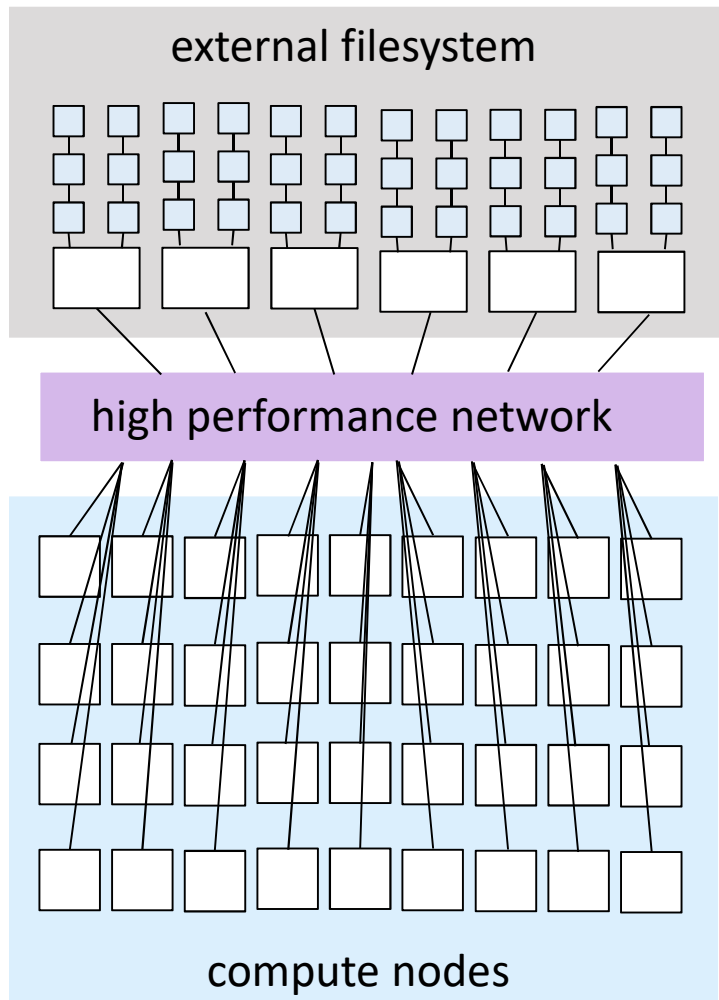
- Non-volatile already becoming part of HPC hardware stack
- SSDs offer high I/O performance but at a cost
 - How to utilise in large scale systems?
- Burst-buffer hardware accelerating parallel filesystem
 - Cray DataWarp
 - DDN IME (Infinite Memory Engine)



CRAY



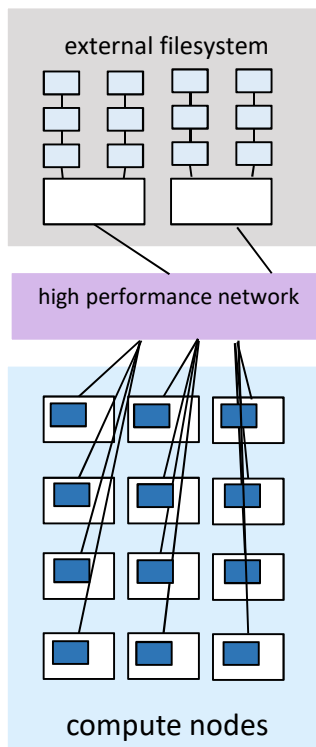
Burst buffer



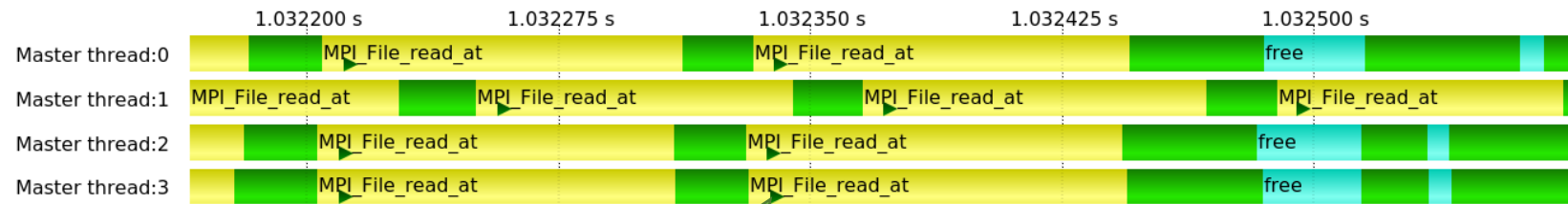
Moving beyond burst buffer



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

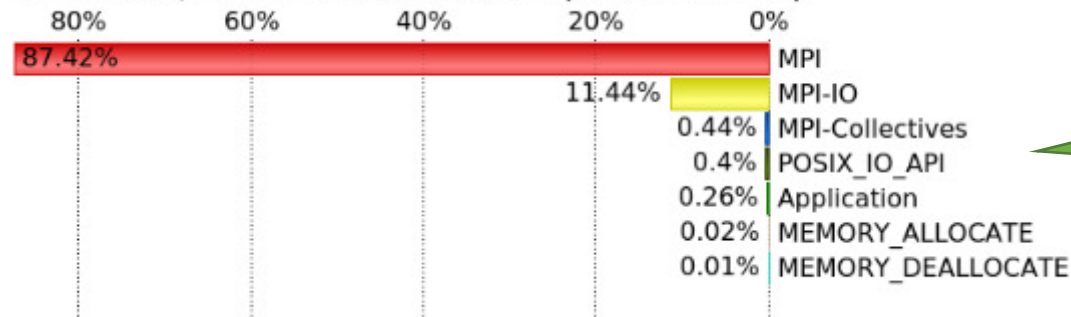


I/O application patterns



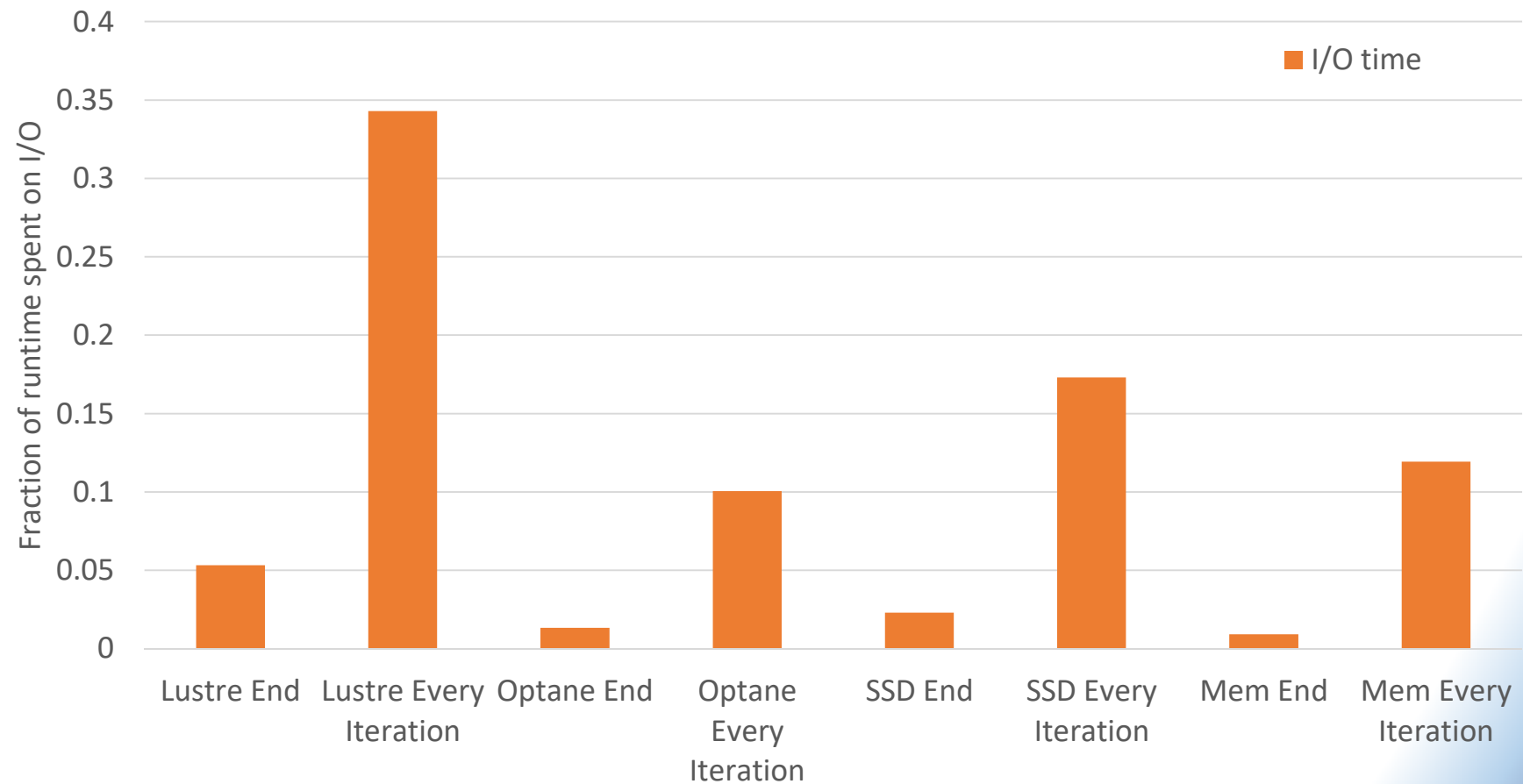
Individual I/O Operation

All Processes, Accumulated Exclusive Time per Function Group

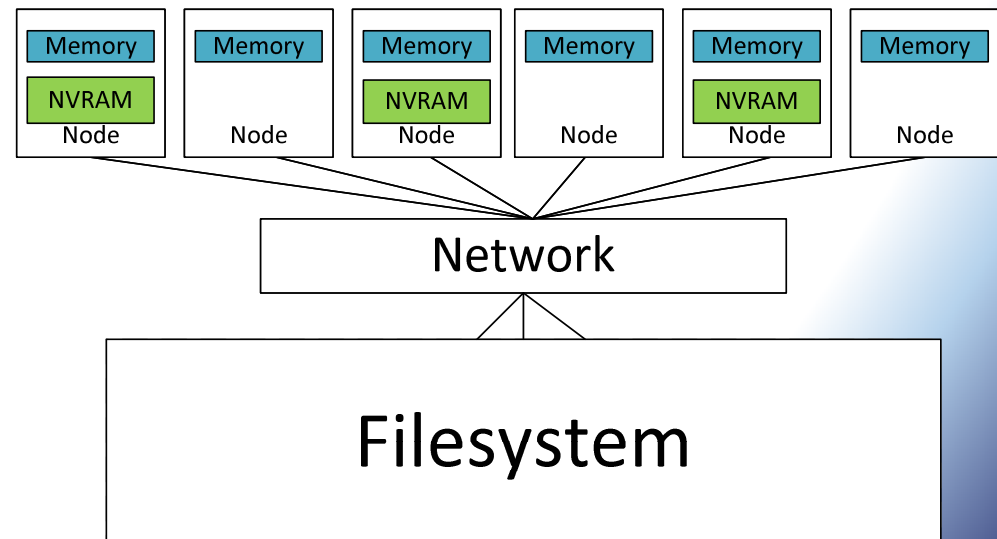
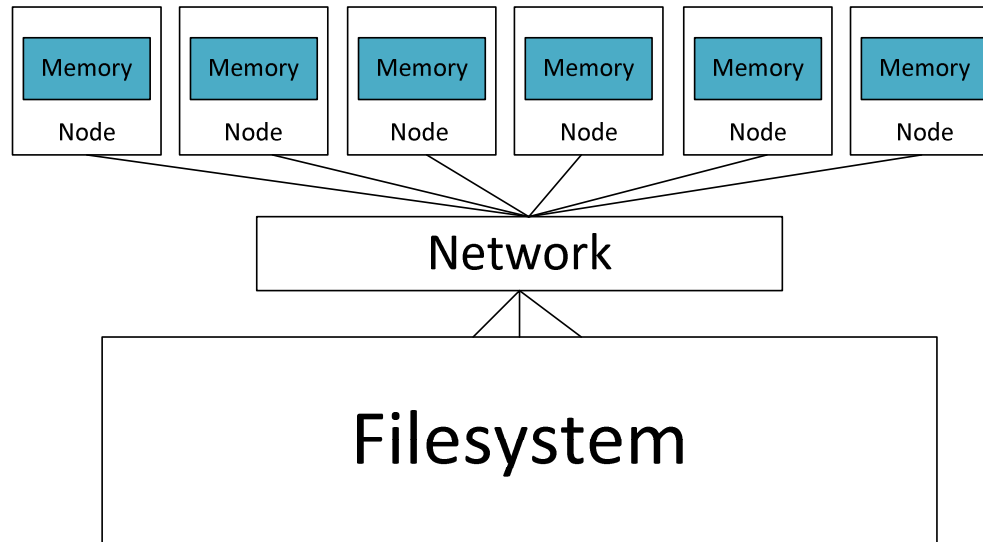


I/O Runtime Contribution

Enabling new I/O



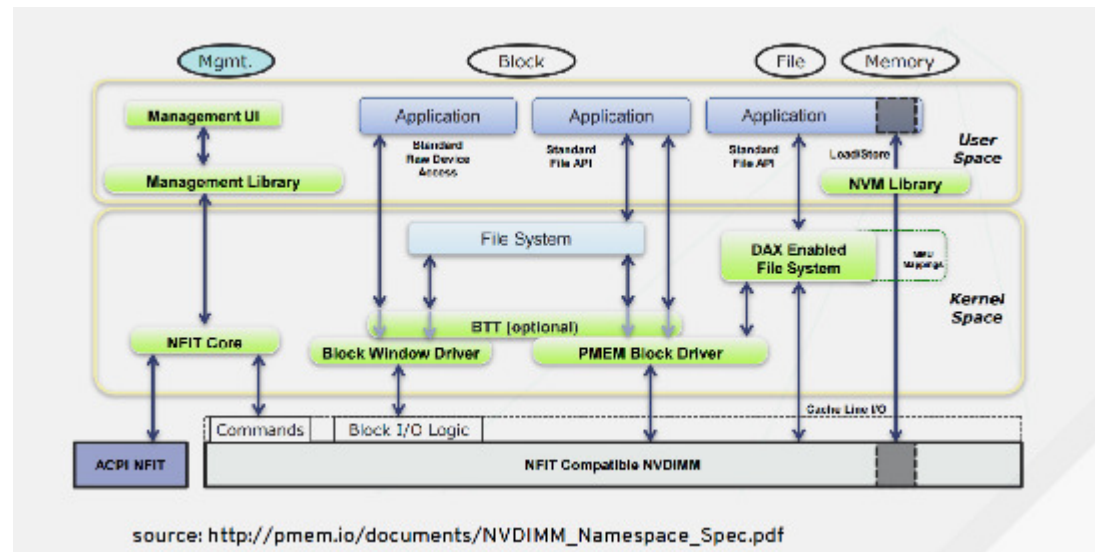
Exploiting distributed storage



Programming SCM



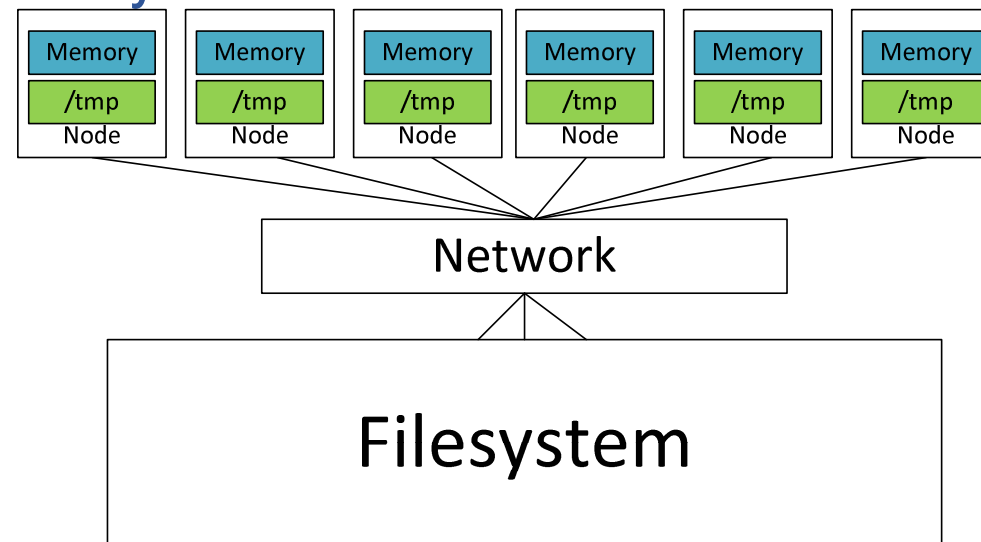
- Block memory mode
 - Standard filesystem api's
 - Will incur block mode overheads (not byte granularity, kernel interrupts, etc...)
- App Direct/DAX mode
 - Volatile memory access can use standard load/store
 - NVM library
 - pmem.io
 - Persistent load/store
 - memory mapped file like functionality



Using distributed storage



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

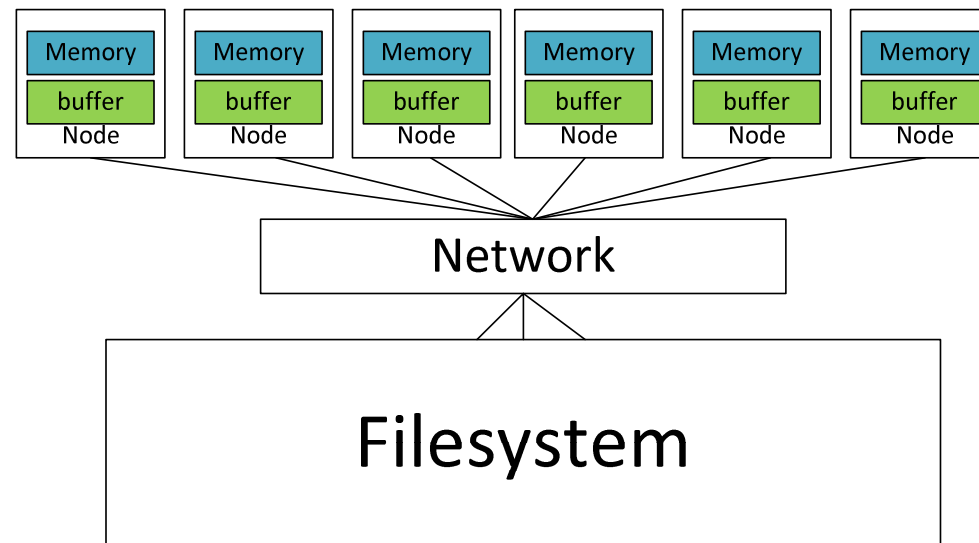


- Users manage data themselves
- No global data access/namespaces, large number of files
- Still require global filesystem for persistence

Using distributed storage



- Without changing applications
 - Filesystem buffer

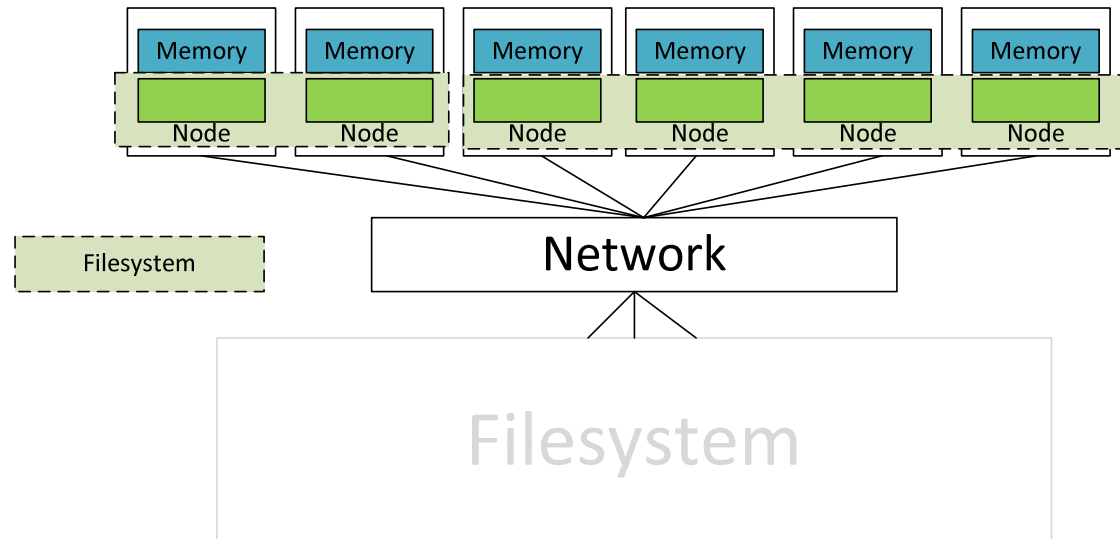


- Pre-load data into NVRAM from filesystem
- Use NVRAM for I/O and write data back to filesystem at the end
- Requires systemware to preload and postmove data
- Uses filesystem as namespace manager

Using distributed storage



- Without changing applications
 - Global filesystem

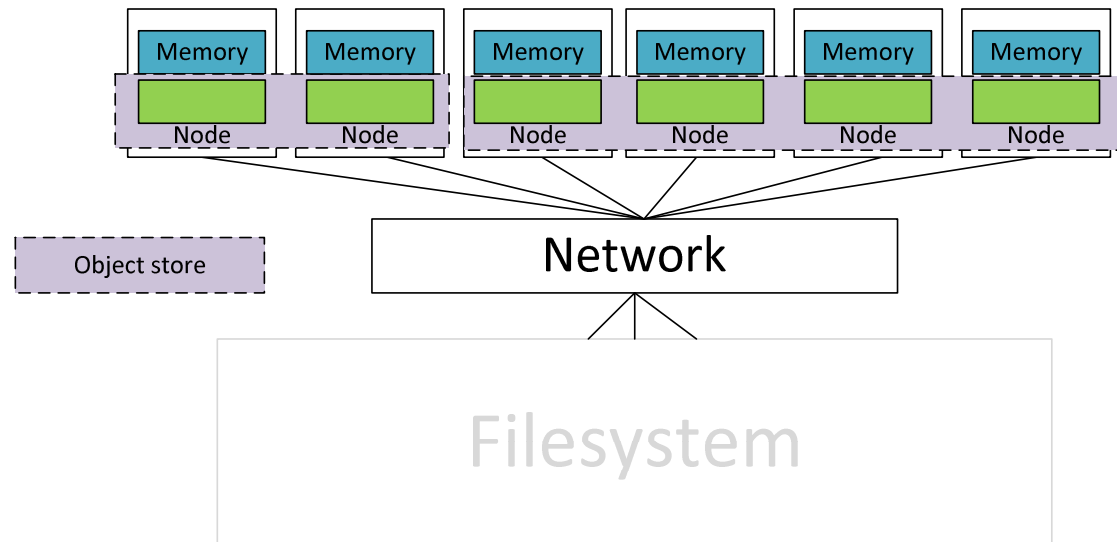


- Requires functionality to create and tear down global filesystems for individual jobs
- Requires filesystem that works across nodes
- Requires functionality to preload and postmove filesystems
- Need to be able to support multiple filesystems across system

Using distributed storage



- With changes to applications
 - Object store

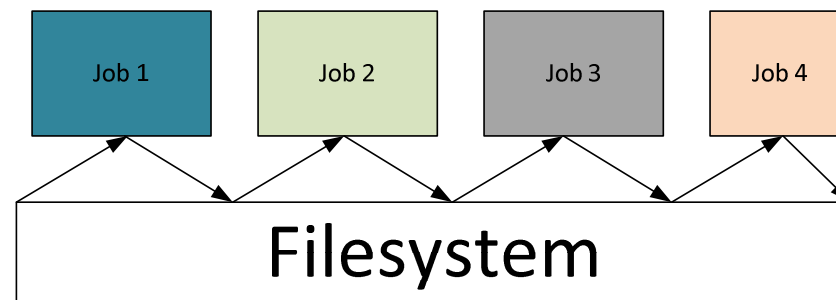


- Needs same functionality as global filesystem
- Removes need for POSIX, or POSIX-like functionality

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



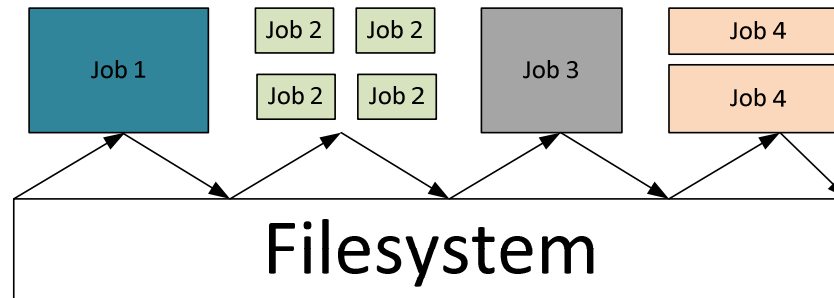
- Remove filesystem from intermediate stages

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

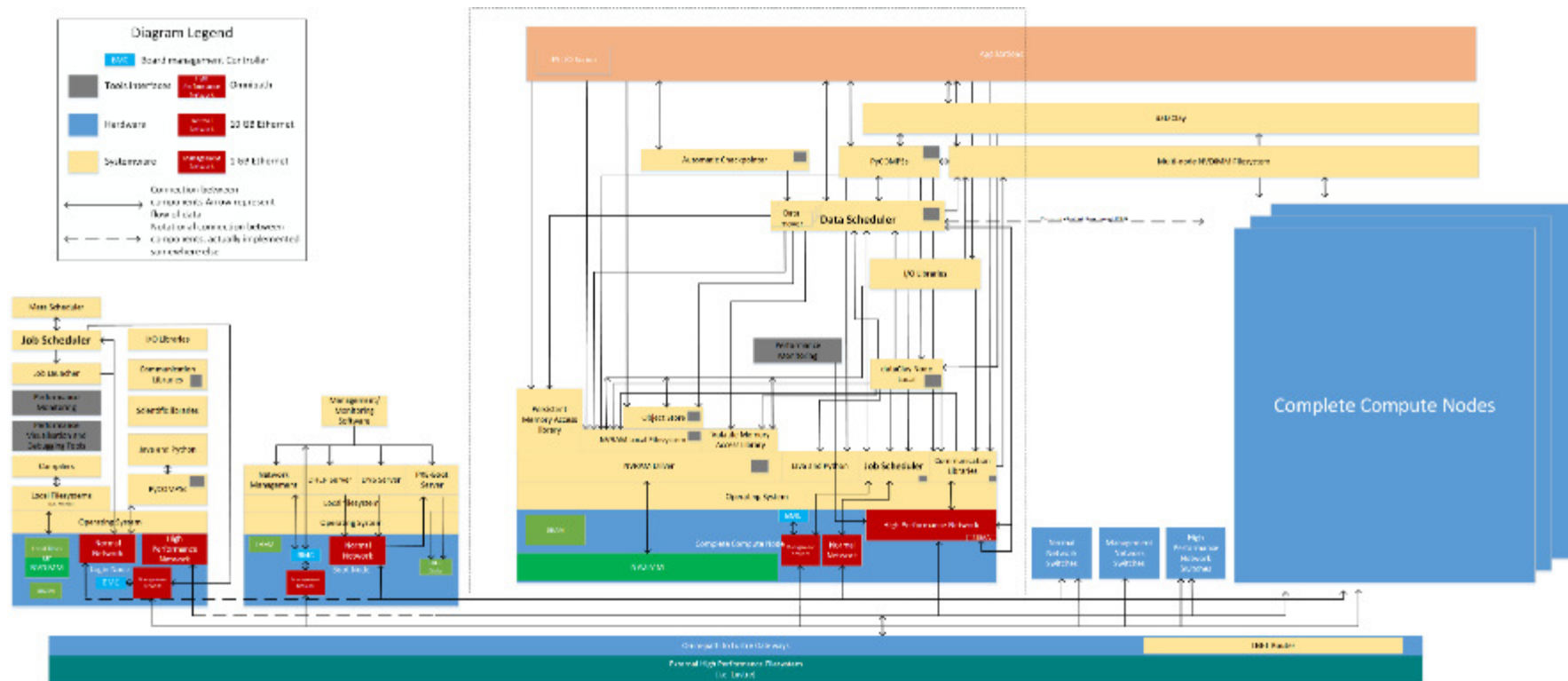


Potential solutions

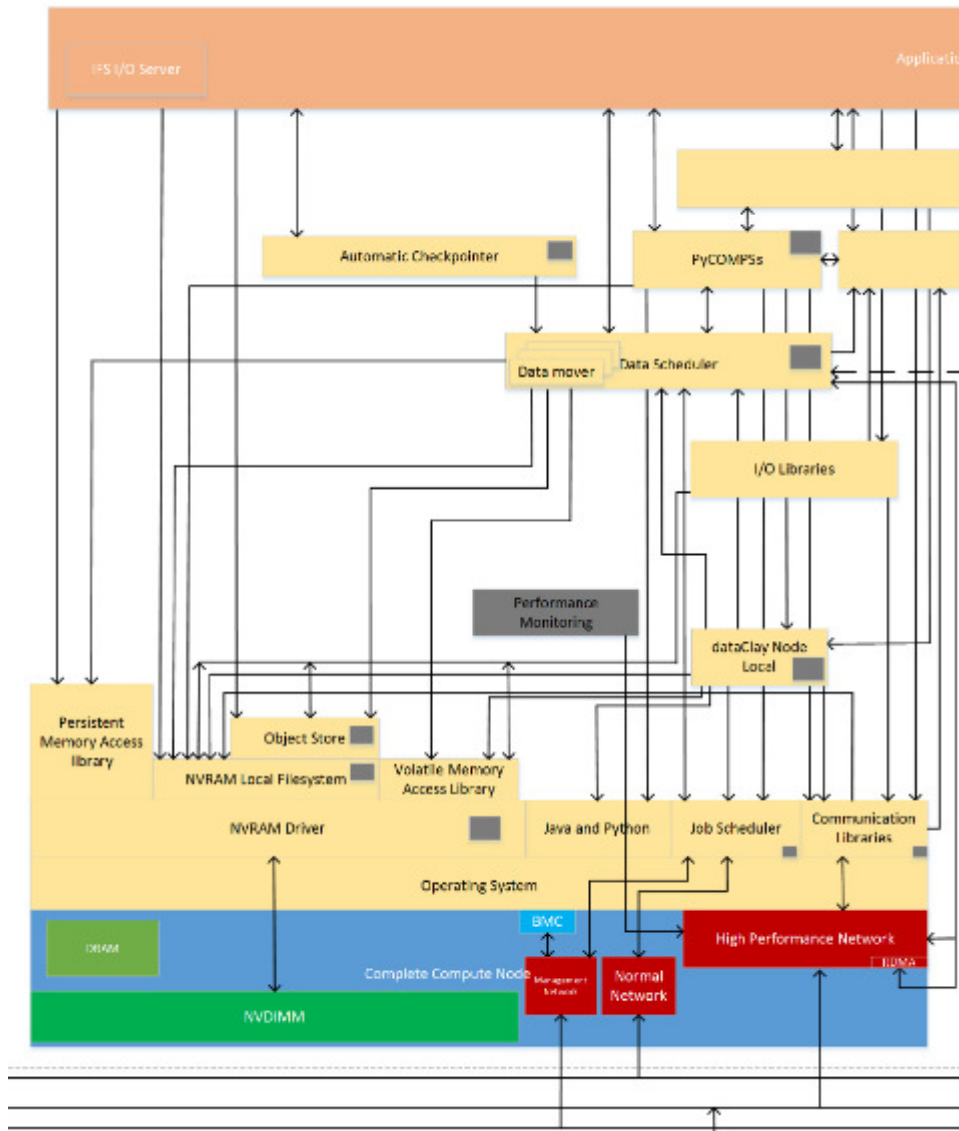


- Large memory space
 - Burst buffer
 - Filesystem across NVRAM in nodes
 - HSM functionality
 - Object store across nodes
 - Checkpointing and I/O libraries
- 

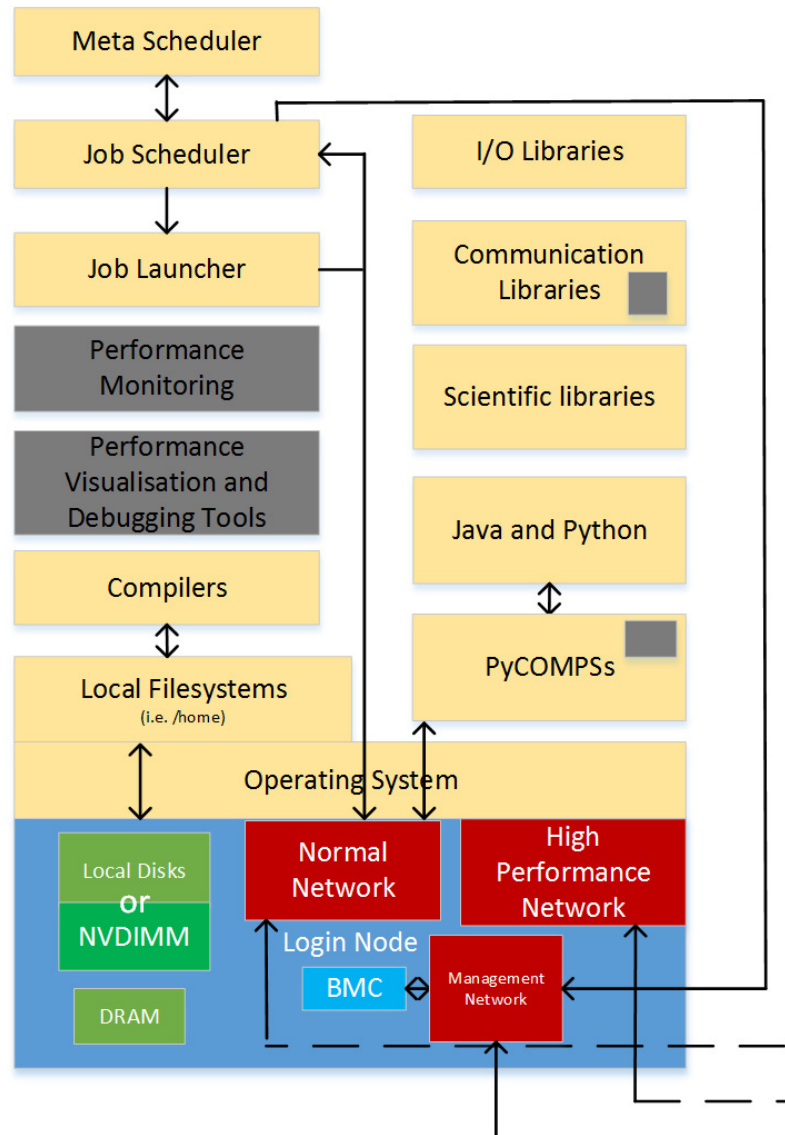
NEXTGenIO Systemware



Compute node systemware



User node systemware



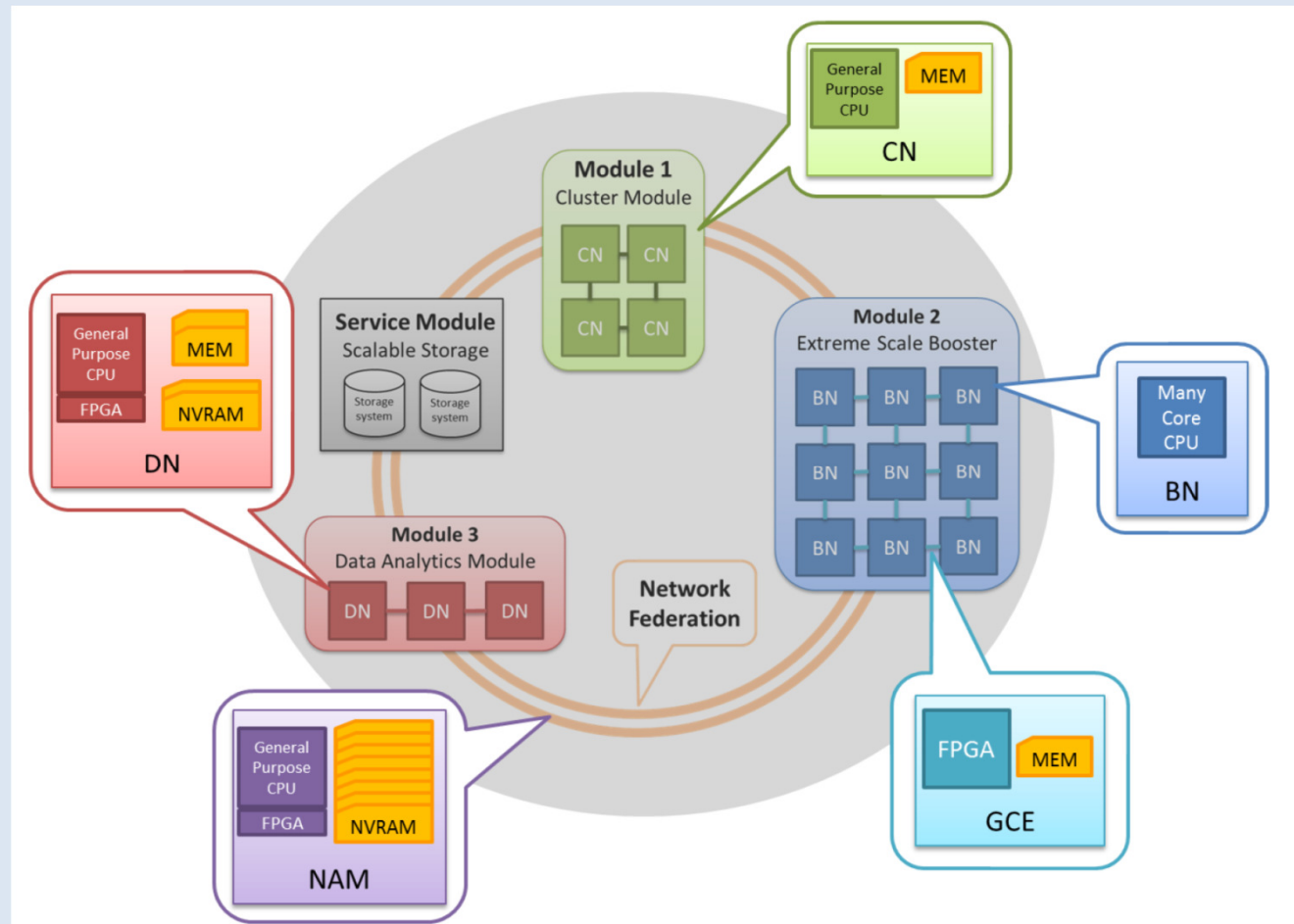
Summary



- Storage class memory is coming
 - 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...

DEEP-EST: Extreme Scale Projects

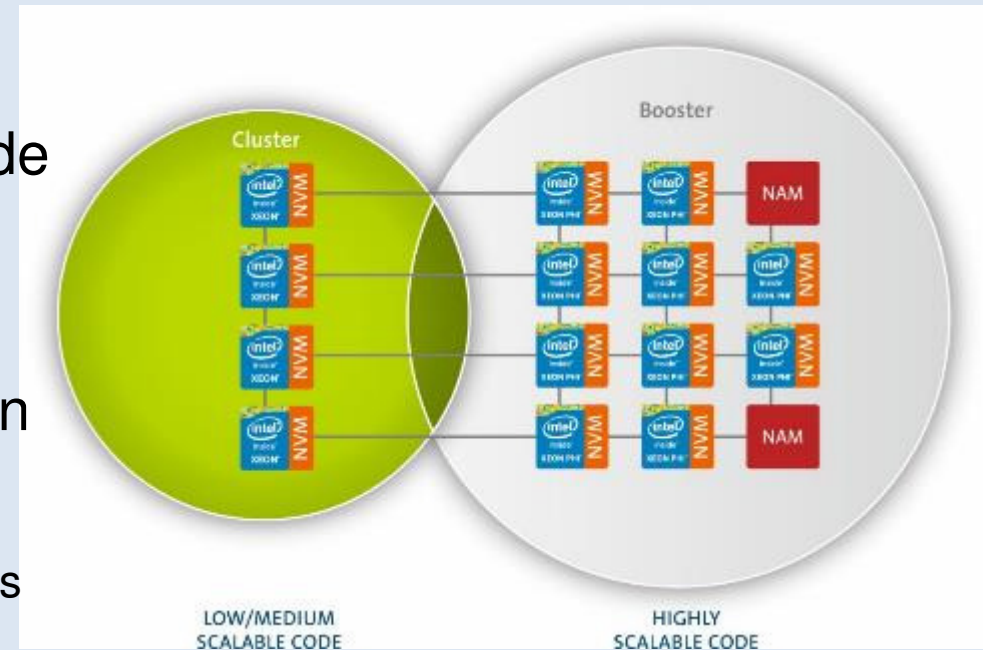
- EU Horizon2020 funded: 754304
- €15M 3 year project
- Create a first version of the Modular Supercomputer Architecture (MSA) defined in DEEP and DEEP-ER
- 15 partners



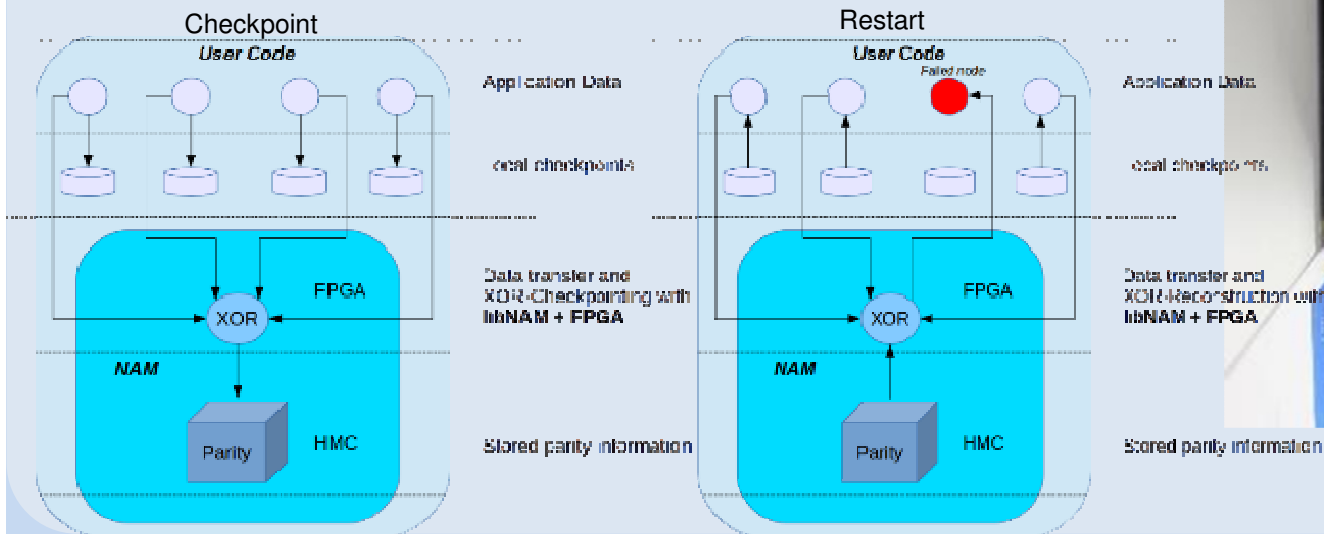
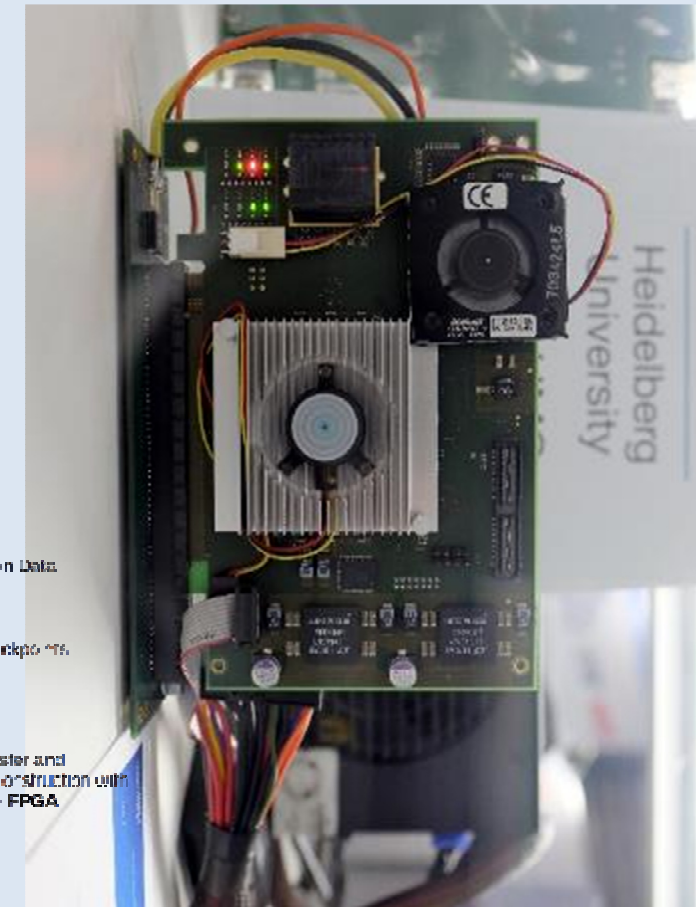


- Schedule across heterogeneous hardware setup
- Monitor power, energy, and performance
- Support single filesystem and storage target from different modules
- Develop/extend programming environments to use new modules
 - Data analytics modules

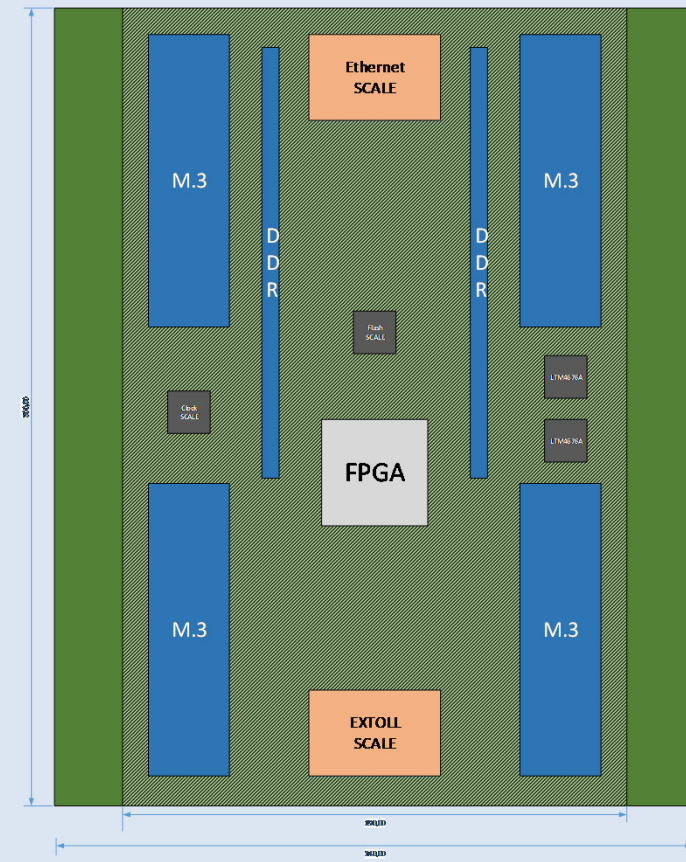
- NAM is a physical circuit board with memory/storage, network interfaces, and an FPGA
- A NAM is treated as regular node
 - Attaches to the same network (as any other node)
- Every process in the system can
 - Write to it
 - Read from it
 - Use application specific functions
- Access rights and security managed by a central instance: NAM Manager



- Xilinx Virtex 7 FPGA
- Hybrid Memory Cube (HMC)
 - DRAM based
 - 2 GB capacity
 - 40 GB/s read/write
- 2* Network 12x connections
 - ~20GB/s read/write in DEEP-ER



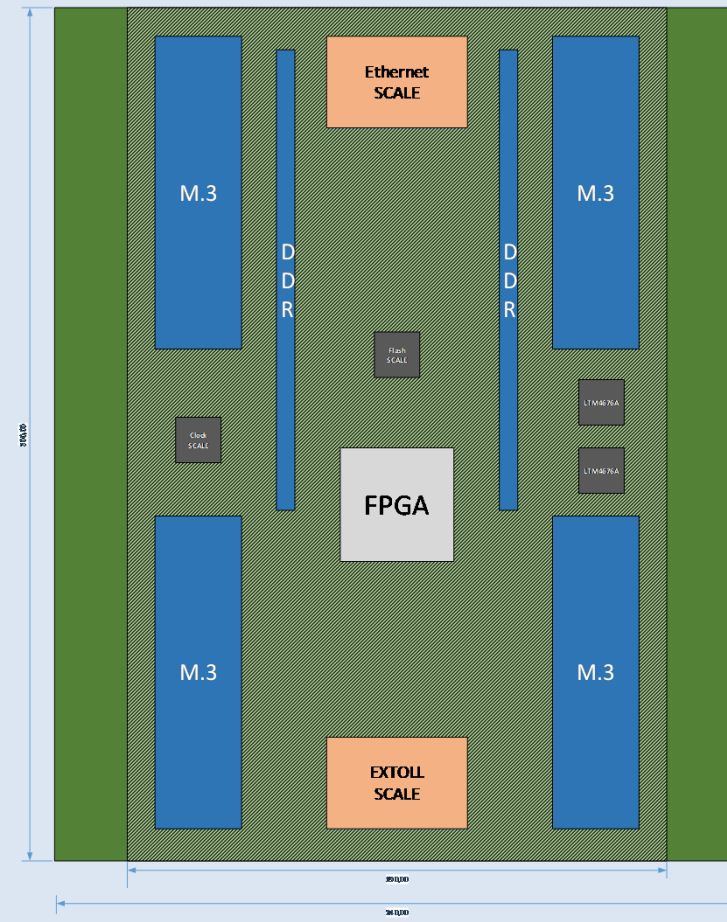
- Memory/Storage directly connected to network through FPGA
- NAM is integrated in the Fabri³
- Per NAM:
 - 16 GB/s write + 16 GB/s read
 - Up to 64 TByte persistent storage (NAND, 3DXpoint)
 - (up to 32 GB DDR4)
- Fabri³
 - Maximum 8 NAMs per Fabri³
 - 128 GB/s write + 128 GB/s read
 - 512 TByte persistent storage
- Lower latency and higher bandwidth than writing to or reading from a regular remote node memory or the PFS





- Checkpointing target
 - Use existing libNAM functions (e.g. with SIONlib)
- Parallel File System
 - Use file operations (fopen, fwrite, ...) as if the NAM was the PFS
 - Use as intermediate file-system as BeeOND extension
- Burst buffer alternative
- Fast, global shared memory
- MPI one-sided shared

- Accelerate collective calls inside the network
- Use the same physical board as the NAM
- Use FPGA to implement collectives





- Jobs **register** groups of participants for a collective group
 - In MPI context this reflects generation of a communicator
- Jobs **de-register** groups of participants
 - In MPI context this reflects destroying a communicator
- Registering a collective group will trigger allocation of resources on the GCE
 - E.g. State & scratchpad memory allocation



- Jobs send **request** messages describing the collective and their parameters
 - Source & destination buffers, size, operand type, collective operation, compute operation in case of reduce,...
- GCE **performs** the actual work completely asynchronously to the actual compute job
- GCE **distributes data back** to the participating processes and **notifies** them



- **Fully concurrent** to actual compute code
- Supports the idea of **non-blocking** collectives ideally
- **Optimize bandwidth** requirements in the network
- **Reduces latency**, since no synchronous intermediate full stack send & receive messages to be processed by compute nodes