ARGONNE
ATPESC2024
EXTREME-SCALE COMPUTING

HPC Storage Systems

Kevin Harms
Performance Engineer / ALCF-4 Technical Director, ANL

Argonne NATIONAL LABORATORY

HPC Storage Universe

- Wide variety of storage solutions exist for HPC and AI
  - File systems
    - Parallel
    - NFS
  - Object Storage (many flavors)
  - Node Local Storage
  - Tape
- The specific storage solution will have a significant on I/O performance of your workload
- Understanding the storage system and how it is connected and configured will enable
  - Improved performance
  - Improved reliability
- Tuning will be system specific and software specific
  - Unfortunate but true, you don’t compile your I/O
  - Tractable problem, you can solve it with the help of your local HPC contacts

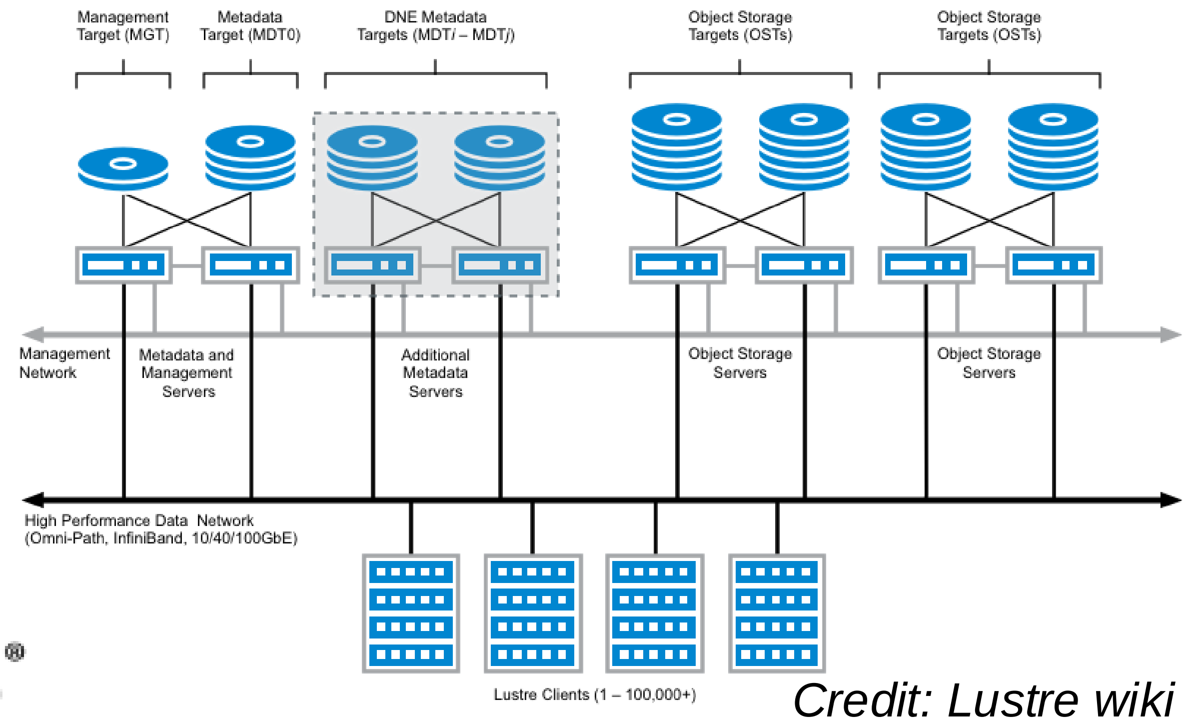
Types of Storage

- Parallel File Systems
  - Provide strict (or almost) POSIX compliance in parallel at scale
  - Most commonly available solution?
  - POSIX semantics (esp. locking) causes issues with scalability
- NFS
  - Provides support of POSIX APIs
  - Does not enforce POSIX compliance when accessed in parallel
  - Widely used and works for some workloads but fails for others
- Object Storage
  - Provide non-POSIX APIs, potential support for POSIX APIs
  - Provide limited to no POSIX-like semantics
  - Close-to-open-like semantics commonly supported
- Node Local
  - Non-volatile storage on the compute node
  - Often not treated as non-volatile
  - Accessible only by the local processes
  - Usually standard file system like ext4, xfs, …
- Tape
  - Provide proprietary interface to store bulk data
  - May have disk cache to improve performance
  - Typically not accessible by compute nodes

Parallel File Systems

- Provide POSIX API and POSIX semantics
  - Equivalent to Linux, Mac, Windows file systems
  - File and directories
- Built as client software running on compute nodes
- Server software running on storage servers with many storage device connected to each server
- Files and directories split across storage resources
  - How data is distributed is key part of performance

IBM Storage Scale



Credit: Lustre wiki

Lustre

<u>Vocabulary of Lustre</u>

Client = Lustre software running on compute node

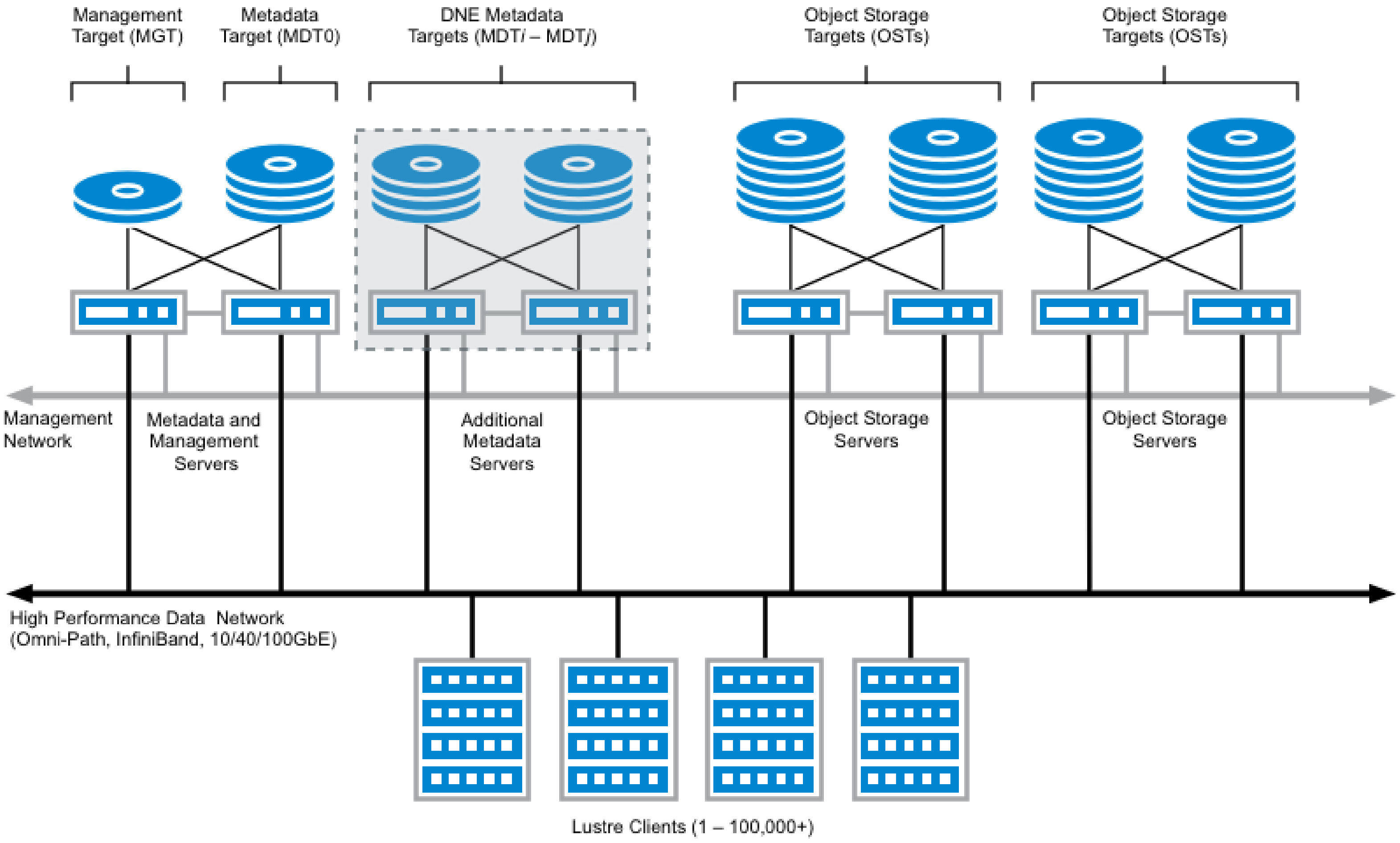
LNET = Lustre Network Router, I/O forwarding node

MDS = Metadata Server, manages metadata

MDT = Metadata Target, metadata storage

OSS = Object Storage Server, manages data

OST = Object Storage Target, data storage



Lustre - Hardware

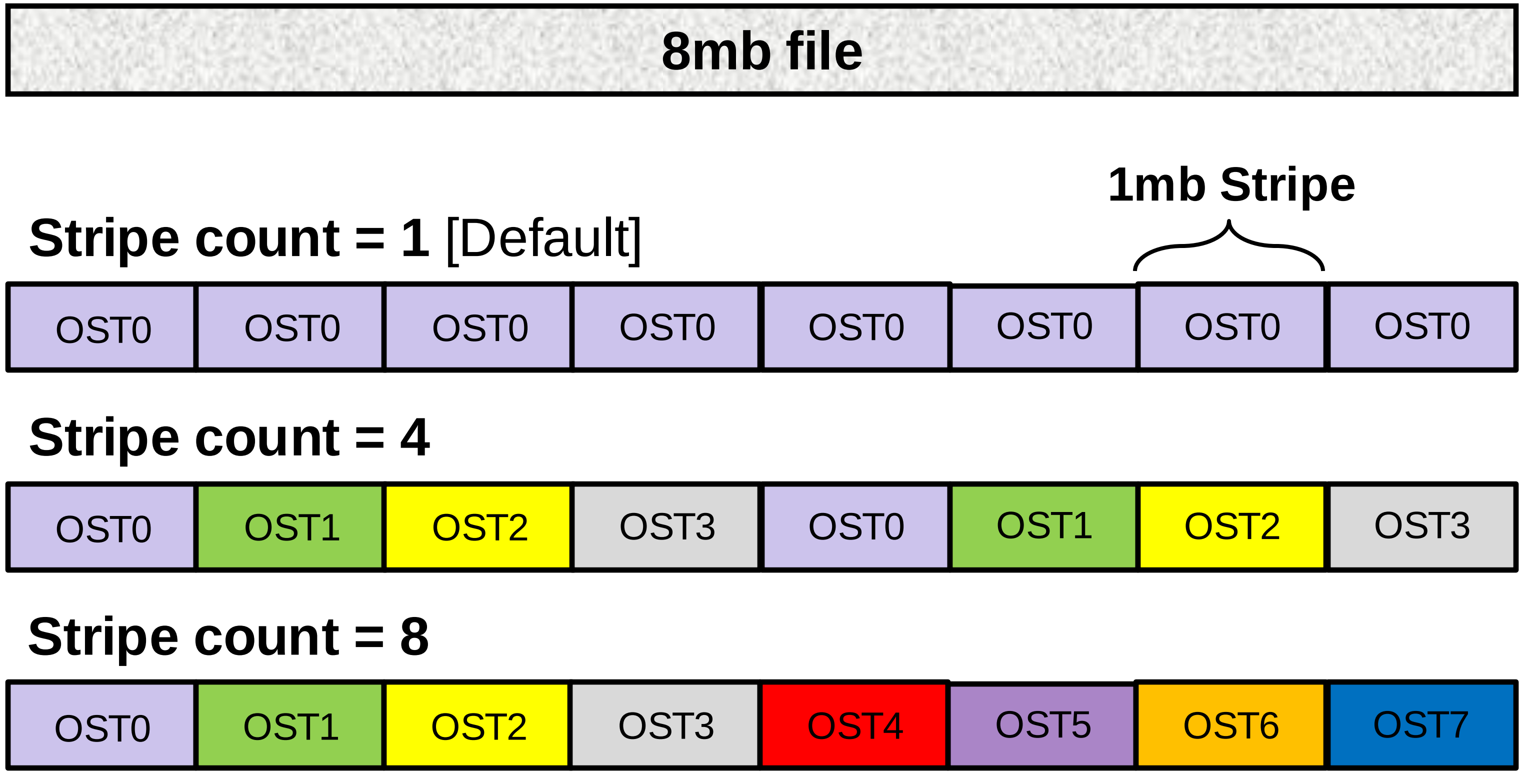
- Lustre can be built on many forms of storage
  - Most common is hardware storage array with hard disks or flash-based disk
- Storage arrays provide data protection (not Lustre)
  - RAID6 data protection
  - Error correcting code which contains 8 pieces of data and 2 pieces of parity
    - Reconstruct missing data as long as 8 of 10 are available
  - Declustered parity is a common feature where these 8+2 stripes are logically distributed over pools of larger groups of disks, 50-100 disks
  - One of these declustered parity groups is an OST
- The more MDT and OST, the more throughput is available

Lustre Software

- Striping is a key parameter to understanding performance that is obtainable
  - No one setting is optimal for all conditions

- Data is written by the client to a specific OST
- Files are distributed over OSTs based on the configuration of the file
- Every Lustre system has a default for the number of “stripes” or OSTs that are used for a given file
  - 1 stripe for a file will limit the total bandwidth that reading or writing that file is possible
  - Many stripes will allow large files to be read and written in parallel
  - The size of the stripe is also configuration but Lustre defaults to 1MiB

Example: Consider a single **8mb file** with **1mb stripe size…**



Files and Directories

lfs getstripe <name>
lfs setstripe -c N <name>

The setstripe command will not change an existing file.

- Everybody likes files, just because you can have 100M files doesn't mean you should
- As scale increase file-per-process can become a bottleneck
  - Cost of meta-data operations relative to cost of data operations increases
    - Files get smaller
    - Number of files get larger
- File-per-process
  - Typically scales well to O(10K)
  - Lustre default is often 1 stripe or 2 stripes per file with suits FPP
  - Risk of becoming "that guy" who is overloading the MDS and causing all other users problems
- Single-shared-file
  - Need to configure stripe unit for larger count
    - -1 will use all, not always the best setting
  - Scaling limited by lock contention
  - Using MPI-IO can help optimize access to single-shared-files

Lustre Stripe Examples

Stripe parameters are inherited from the directory if none are used when creating the file.

Set the stripe of a directory as a way to set stripe configurations of files

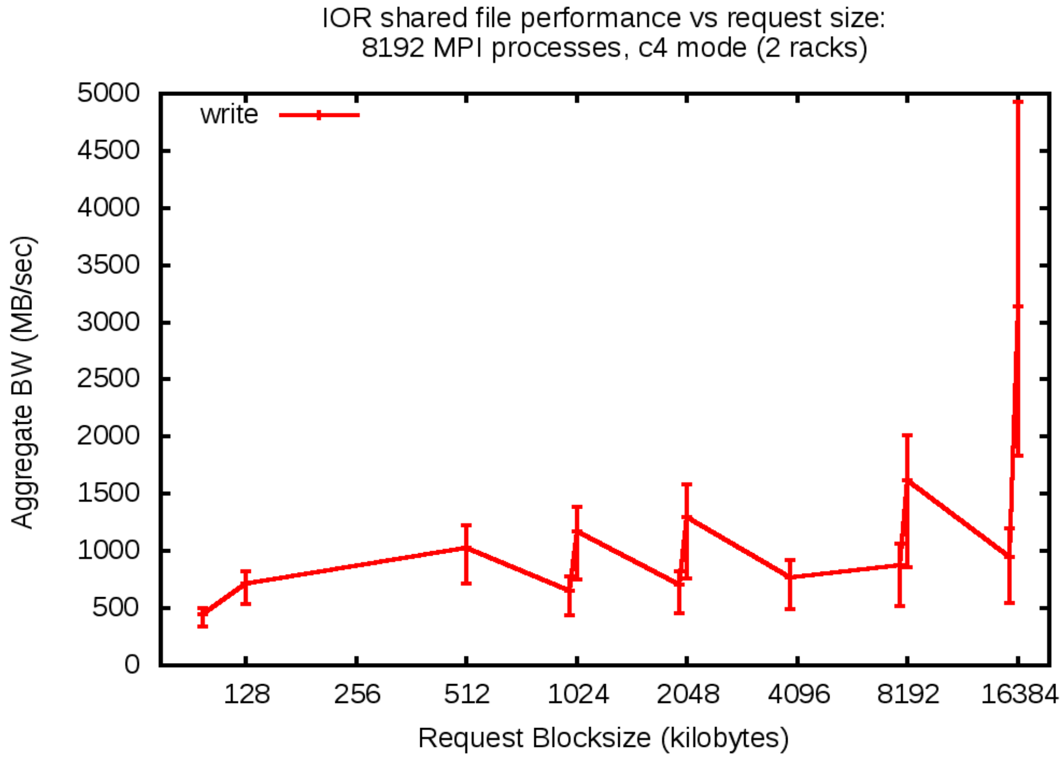
```
harms@aurora-uan-0011:~> lfs getstripe test.txt
test.txt
 lcm_layout_gen:   2
 lcm_mirror_count: 1
 lcm_entry_count:  2
  lcme_id:             1
  lcme_mirror_id:      0
  lcme_flags:          init,prefer
  lcme_extent.e_start: 0
  lcme_extent.e_end:   536870912000
   lmm_stripe_count:  1
   lmm_stripe_size:   1048576
   lmm_pattern:       raid0
   lmm_layout_gen:    0
   lmm_stripe_offset: 109
   lmm_pool:          ddn_ssd
   lmm_objects:
   - 0: { l_ost_idx: 109, l_fid: [0x600000bd3:0x342679:0x0] }

  lcme_id:             2
  lcme_mirror_id:      0
  lcme_flags:          0
  lcme_extent.e_start: 536870912000
  lcme_extent.e_end:   EOF
   lmm_stripe_count:  2
   lmm_stripe_size:   1048576
   lmm_pattern:       raid0
   lmm_layout_gen:    0
   lmm_stripe_offset: -1
   lmm_pool:          ddn_hdd
```

```
zamora@thetalogin6:~> mkdir stripecount4size8m
zamora@thetalogin6:~> lfs setstripe -c 4 -S 8m stripecount4size8m/.
zamora@thetalogin6:~> lfs getstripe stripecount4size8m
stripecount4size8m
stripe_count:   4 stripe_size:    8388608 stripe_offset:  -1
```

Lustre Request Size and Alignment

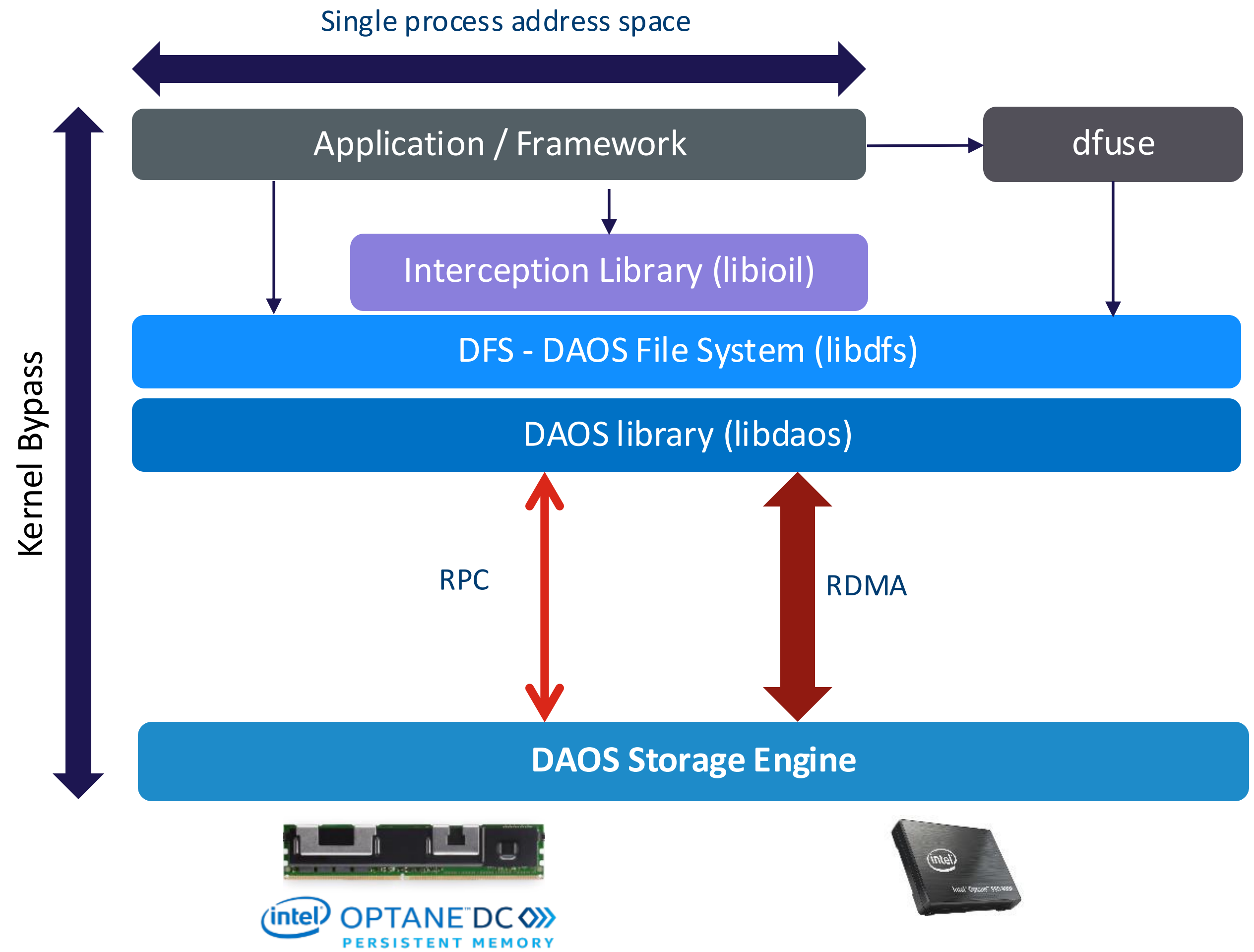
- The access size of I/O requests is an important component of I/O performance
  - The larger the request, generally the better performance (up to some limit)
    - Take advantage of the bandwidth and minimize the impact of latency
- Accesses should be on block boundaries
  - The default block (stripe) size of Lustre is 1MiB
    - Confirm the stripe size with lfs getstripe
  - GPFS storage systems often have blocks sizes between 4-16 MiB
    - Can be confirmed with statfs
  - When not on block boundaries
    - Performance suffers because different nodes may request/hold locks to access the particular block
    - Serialization of access negates parallel operations



Object Storage

- Provide non-POSIX interface
  - S3 is common
  - Many others are custom or proprietary
- Usually simplified put/get with limited to no-overwrite of data
  - New data is appended to existing data
- No files or directories
  - Named blobs of data
  - Simplified if any hierarchy of data
- May support POSIX API wrapping with emulation of files and directories
- Removal of POSIX semantics allows for far greater scalability and performance
- Different models of data protection
  - Often replication is preferred
  - Other schemes are available

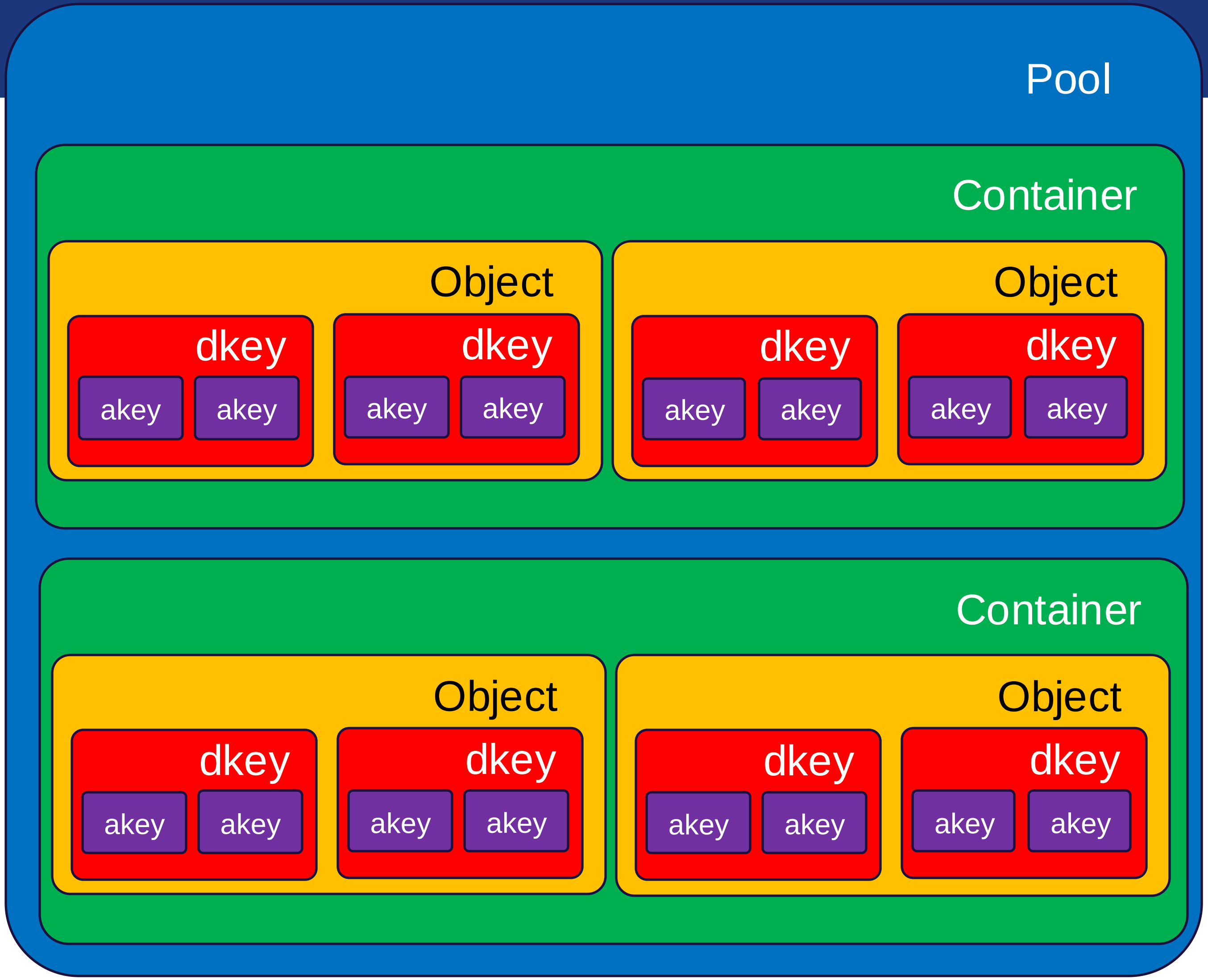
DAOS



- User space DFS library with an API like POSIX.
  - Requires application changes (new API)
  - Kernel Bypass, no client cache
- DFUSE plugin to support POSIX API
  - No application changes
  - Fuse Kernel Supports data (wb and ra) & metadata caching (stat, open, etc.)
- DFUSE + IL
  - No application changes, runtime LD_PRELOAD
  - Kernel Bypass for raw data IO only.

DAOS Pools and Containers

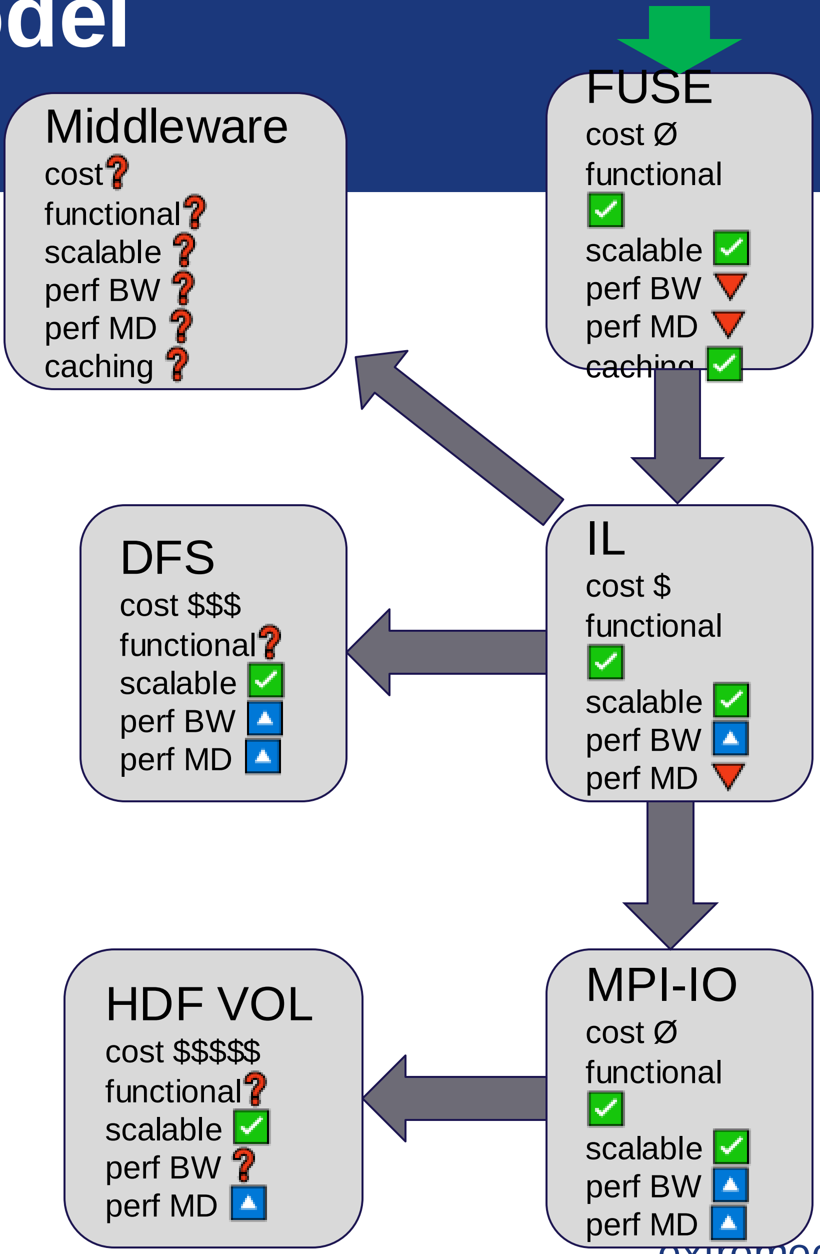
- Pools
  - A system contains *hundreds*
  - Physically allocated storage
    - Decided at pool creation time
  - Equal storage allocated per storage target
  - Contain list of Access Control Lists (ACLs)
  - Contains default parameters for containers
- Containers
  - A pool contains *thousands* of containers
  - Basic unit of storage from user perspective
  - Containers have a type (POSIX, HDF5, …)
  - POSIX containers can have many *millions* of files/directory/data
  - Configuration for object class/redundancy, checksums, cell size, etc.



DAOS Porting Model

Other middleware with custom DAOS backends...

dFuse provides no effort path. Performance will be suspect, but does provide caching and buffering which can be positive for performance.



DFS API very similar to POSIX but requires porting all your I/O code. Allows ability for low level control of objects.

Interception Library low effort with large performance upside on BW. Potential issue with functionality if more esoteric interfaces used.

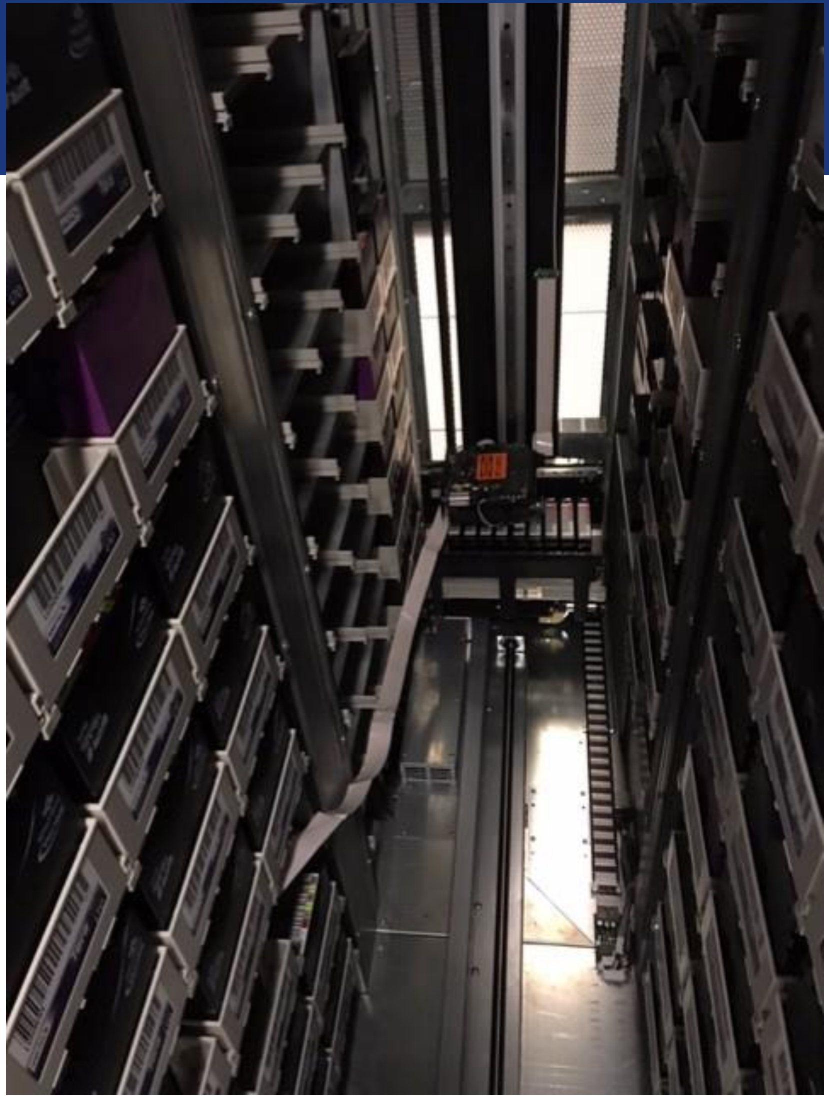
HDF VOL usage will require (potentially) significant rework. Need to understand usage before making changes

MPI-IO with DAOS ADIO Should be transparent and provide best possible performance.

Node Local

- Compute nodes can be equipped with non-volatile storage
  - Flash SSD
  - Persistent Memory
- Low-latency but low-bandwidth
  - Using more compute nodes scales the bandwidth horizontally
- NVM is more often used as “volatile”
  - If the node reboots or jobs ends, data is removed
- File system like ext4 or xfs is used
- Data is only accessible by the node which wrote it
- Used for storing temporary/intermediate data which may be larger than nodes memory
- Can be used to store data quickly, then copy the data to durable storage in background
- Might be used in workflow model where data persists through executions of a different phases of computation

Tape



- Tape is an older storage format but still in wide use today
- Used for archival purposes
  - – long term storage with no power cost
- Extremely slow random access
  - Can takes minutes, hours, days to fetch data
- Large robots have thousands of tapes and tens of drives
  - Physically move tapes in and out of drives to read and write
- Typically have proprietary interface
  - HPSS is a common solution
- Specific commands to copy data to and from tape
- Normally only accessible from login node or perhaps Globus
- Not directly accessible by compute nodes

DOE Storage Systems

- Not discussed here
  - All systems have a home file system
    - Not intended for high performance
  - All systems have a tape solution
  - Some systems have a read-only volumes for software

NERSC Storage

https://docs.nersc.gov/filesystems/

System	Capacity	Performance	System
Perlmutter Scratch (Lustre)	35 PB @ RAID6 ▪ 16 MDT ▪ 298 OST	≥ 5 TB/s Read & Write - Stripe = 1OST	Perlmutter
Community (GPFS)	106 PB @ GNR (EC8+2) • 28 NSD • 11,872 disks	200 GB/s Read & Write	All NERSC systems
Node Local (tmpfs)	System memory	System Memory BW	Any - /dev/shm

OLCF Storage

Storage systems are categorized by access privileges

https://docs.olcf.ornl.gov/data/index.html

System	Capacity	Performance	System
Orion (Lustre)	679 PB @ RAID6 ▪ 500 OSS	10 TB/s Read & Write • Performance Tier 5.5 TB/s (rd) & 4.6 TB/s (wr) • Disk Tier	Frontier
Alpine2 (GPFS)	50 PB @ GNR (EC8+2) ▪ 16 ESS	< 800 GB/s Read & Write • Based on network performance	Summit
Project Home (NFS)			Frontier / Summit / others
Node Local (xfs)	3.8 TB/node • ~32 PB agg	~ 8 GB/s (rd) & 4 GB/s (wr) per node 75 TB/s (rd) & 37TB/s (wr) aggregate	Frontier

Orion File Layout

https://www.depts.ttu.edu/hpcc/events/LUG24/slides/Day1/LUG_2024_Talk_07-Utilization_Trends_and_IO_Patterns_in_the_Orion-Lustre_Filesystem.pdf

- ORNL Orion Lustre has a complex file layout
  - Takes advantage of different storage hardware within Orion system
  - Get performance from files by using flash layer
  - Large files end up on disk components

lfs setstripe -E 256K -L mdt -E 8M -c 1 -S 1M –z 64M -p performance -E 128G -c 1 -S 1M -z 16G -p capacity -E -1 -c 8 -S 1M –z 256G -p capacity <name>

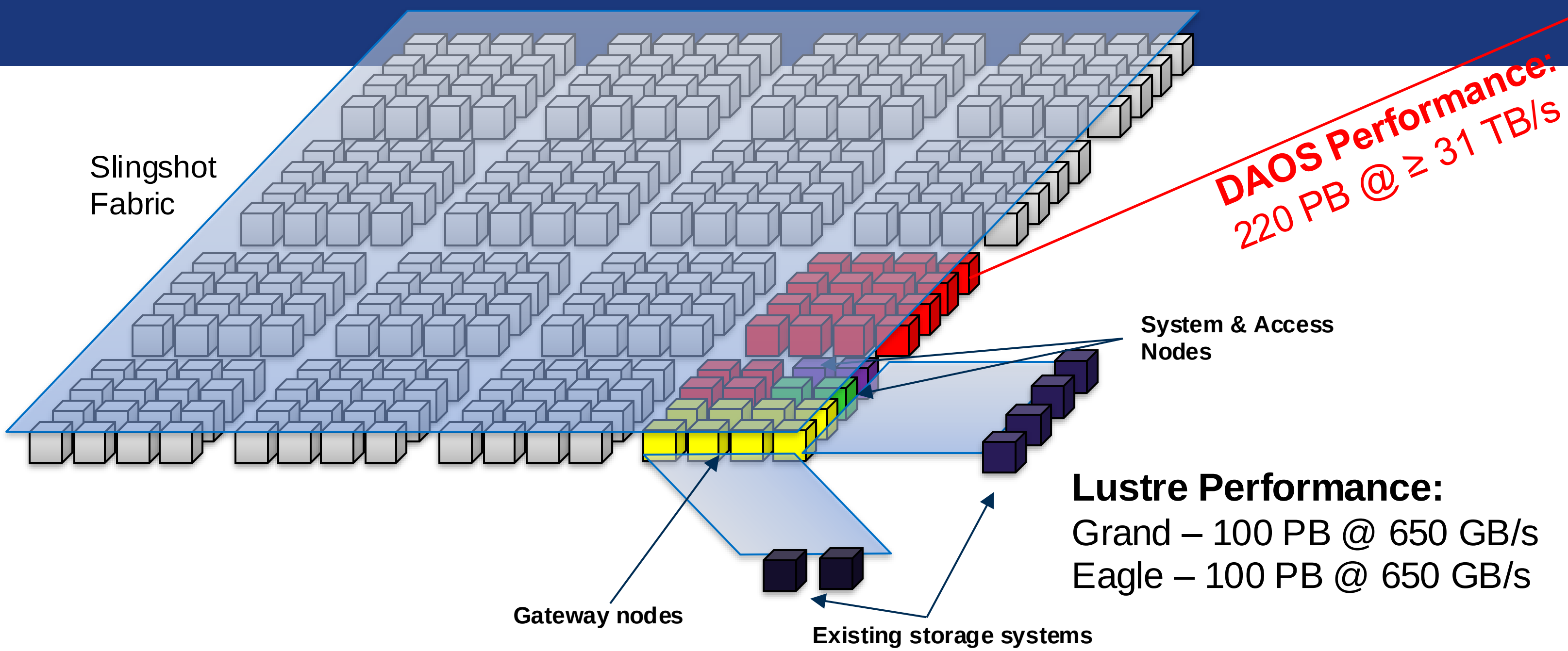
Tier	Component Length	Stripe Size	Stripe Count	Extension Size
Metadata	256 KiB	256 KiB	1	N/A
Performance	8 MiB	1 MiB	1	64 MiB
Capacity	128 GiB	1 MiB	1	16 GiB
Capacity	∞	1 MiB	8	256 GiB

ALCF Storage

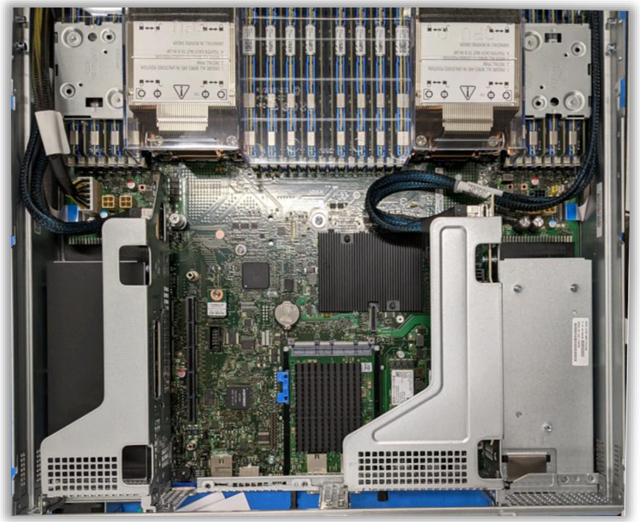
https://docs.alcf.anl.gov/data-management/filesystem-and-storage/file-systems/

System	Capacity	Performance	System
DAOS	220 PB @ EC16+2 ▪ 250 PB NVMe ▪ 16384 SSD ▪ 8 PB Optane PMEM	≥ 25 TB/s Read & Write	Aurora
Eagle (Lustre)	100 PB @ RAID6 ▪ 8480 HDD ▪ 56 Lustre OST ▪ 40 Lustre MDT	> 650 GB/s Read & Write	Polaris
~~Grand~~ Flare (Lustre)	100 PB @ RAID6 ▪ 8480 HDD ▪ 56 Lustre OST ▪ 40 Lustre MDT	> 650 GB/s Read & Write	Aurora
Local (ext4)	3.2 TB/node • 1.8 PB agg	~ 3 GB/s Read & Write per node 1.7 TB/s aggregate	Polaris

Aurora Storage Architecture



Lustre Performance:
Grand – 100 PB @ 650 GB/s
Eagle – 100 PB @ 650 GB/s



The Aurora open-source storage strategy strongly favors cooperation:

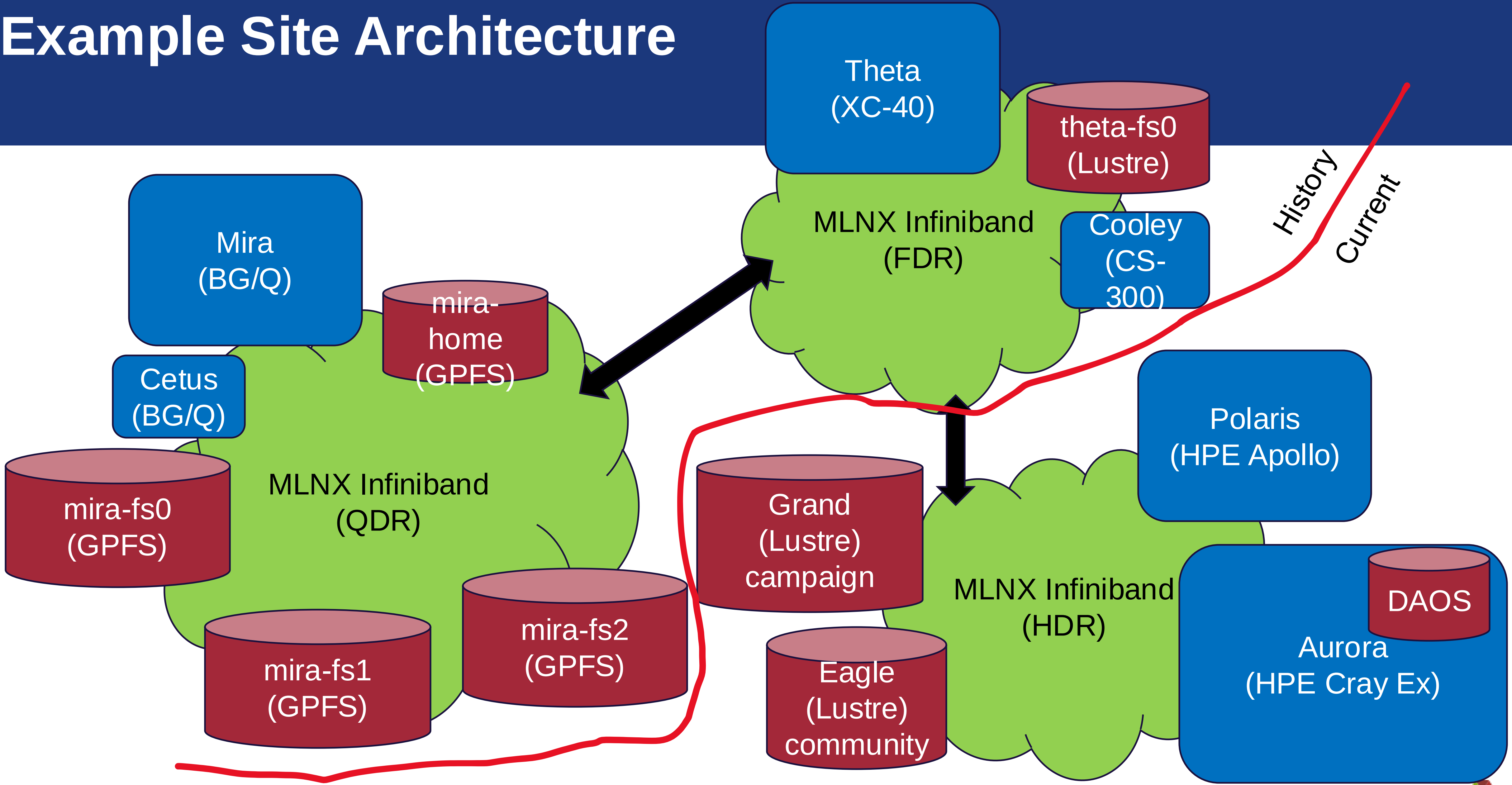
- DAOS: object storage system for in-fabric high-performance platform storage (the first of its kind on a DOE leadership system!)
- Lustre: parallel file systems for facility-wide access and data sharing

Namespace integration will make it easier for users to manage data.

1024 DAOS server nodes, each with:

16 x 512GB persistent memory
16 x 15.3TB NVMe drives
2 x HPE Slingshot NICs
Dual CPU with 512 GB RAM

Example Site Architecture



Conclusion

- I/O is not something you *have* to do, but is something you *want* to do
- Understand the storage system(s) your HPC has available
  - Select the storage based on your needs
  - Select configuration parameters to get the best performance from this system
- Profit!