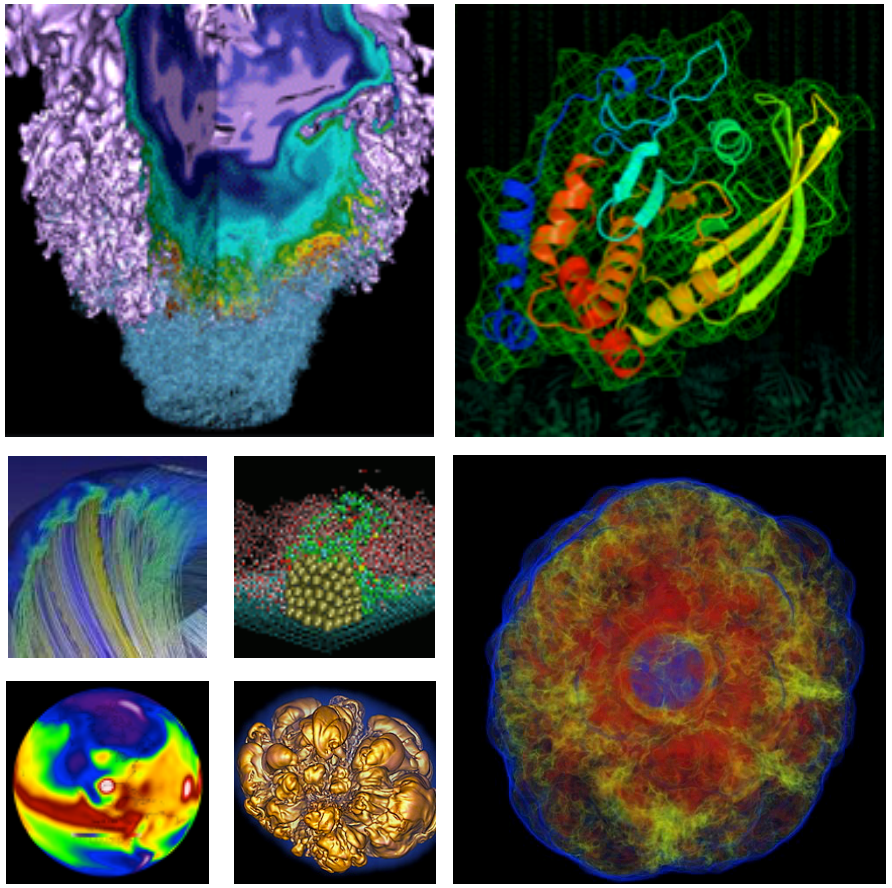
Accelerate your IO with the NERSC Burst Buffer



Debbie Bard
Data and Analytics Services
NERSC

Our users are demanding

- **Who wants more compute?**

Our users are demanding

- **Who wants more compute?**
  - Sorry, you'll have to talk to your allocation manager

- **Who wants better IO?**

Our users are demanding

- **Who wants more compute?**
  - Sorry, you'll have to talk to your allocation manager
- **Who wants better IO?**
  - We can help with that…

HPC memory hierarchy

Past

On Chip

Off Chip

CPU

Memory (DRAM)

Storage (HDD)

Future

On Chip

Off Chip

CPU

Near Memory (HBM)

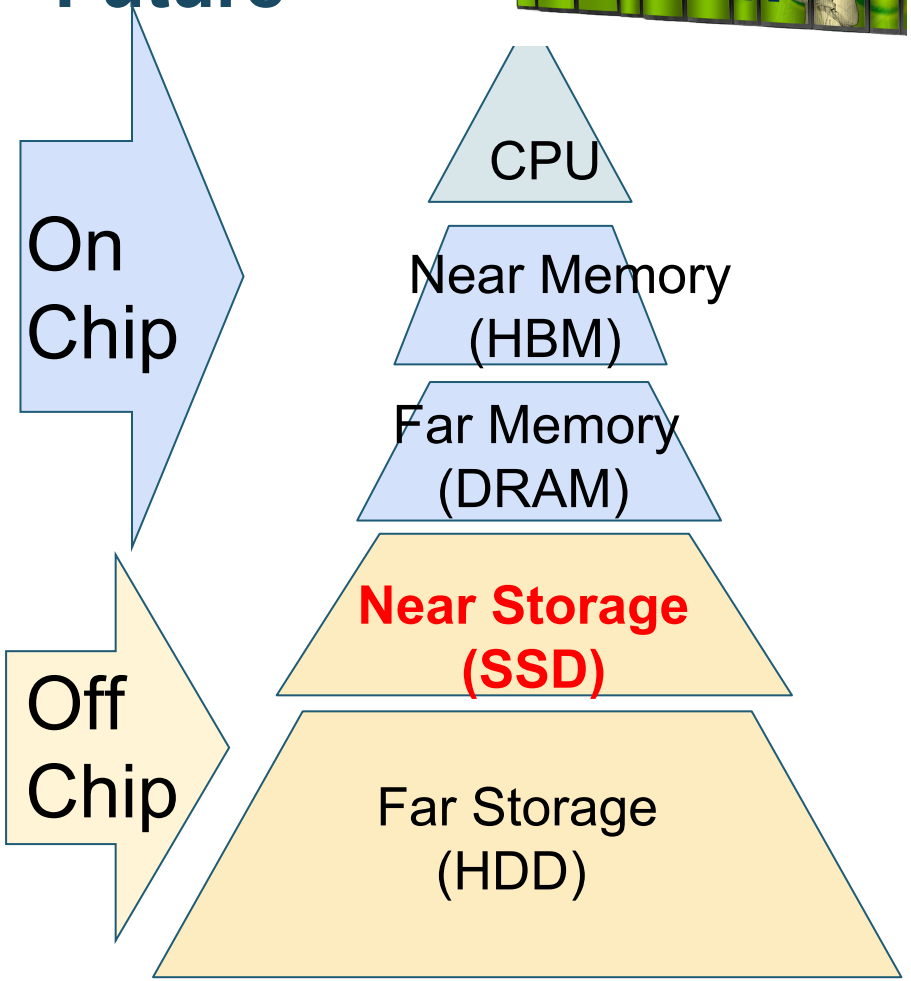
Far Memory (DRAM)

Near Storage (SSD)

Far Storage (HDD)

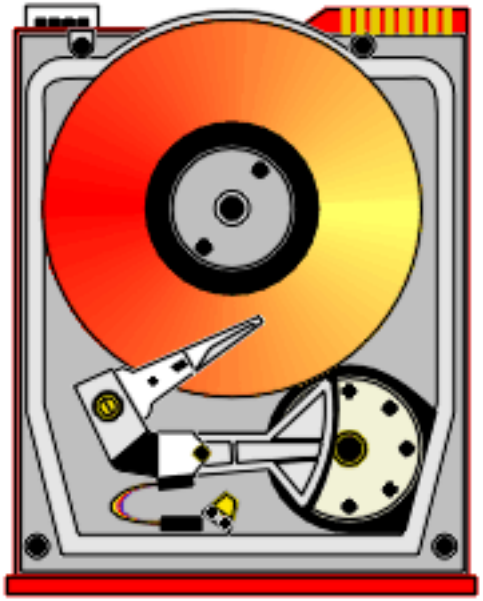
HPC memory hierarchy

- ***Silicon and system integration***
- **Bring everything – storage, memory, interconnect – closer to the cores**
- **Raise center of gravity of memory pyramid, and make it fatter**
  - *–Enable faster and more efficient data movement*
  - *–Scientific Big Data: Addressing Volume, Velocity*



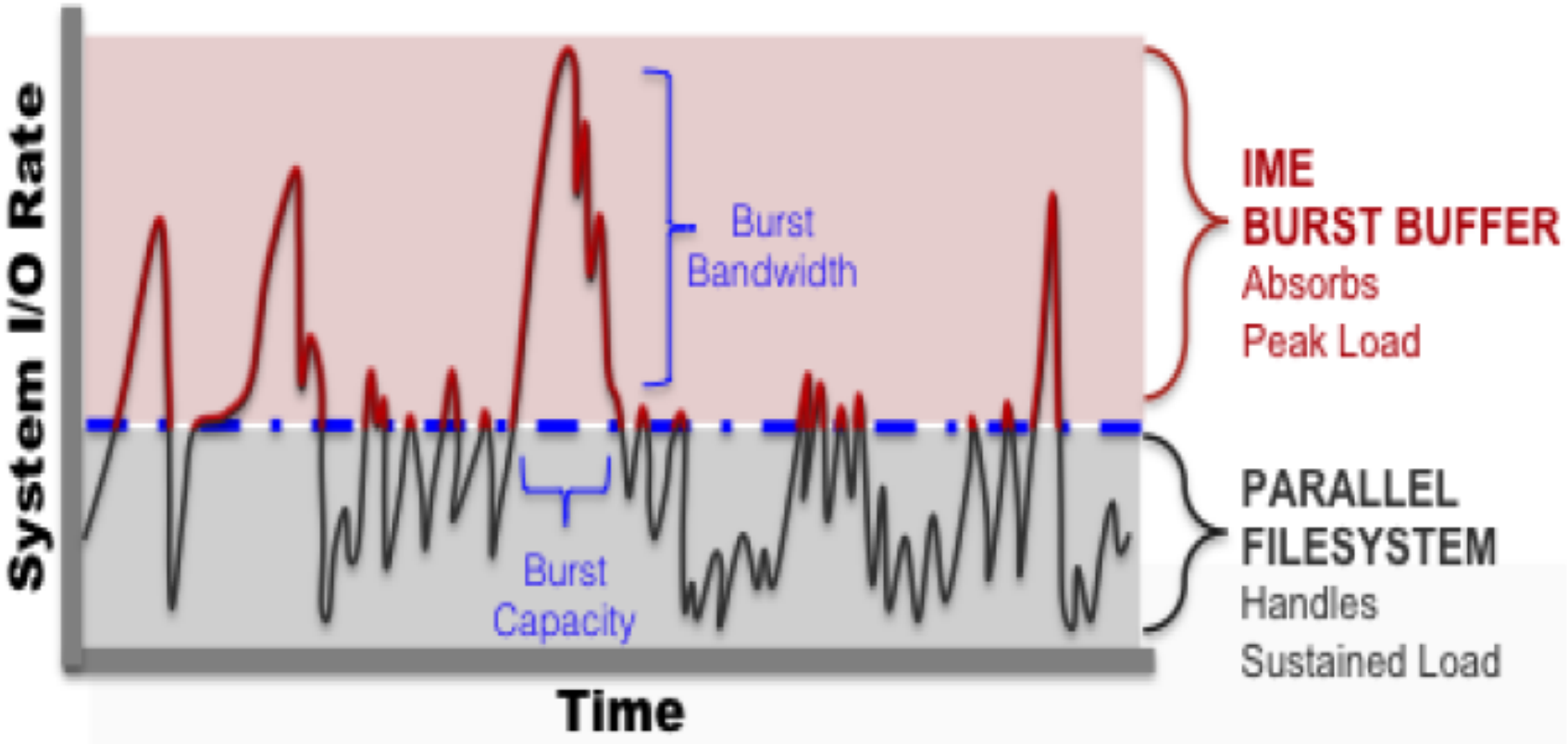
SSD vs HDD

- **Spinning disk has mechanical limitation in how fast data can be read from the disk**
  - –SSDs do not have the physical drive components so will always read faster
  - –Problem exacerbated for small/random reads
  - –But for large files striped over many disks e.g. via Lustre, HDD still performs well.
- **But SSDs are *expensive*!**
- **SSDs have limited RWs – the memory cells will wear out over time**
  - –This is a real concern for a data-intensive computing center like NERSC.



Why an SSD Burst Buffer?

- **Motivation: Handle spikes in I/O bandwidth requirements**
  - Reduce overall application run time
  - Compute resources are idle during I/O bursts

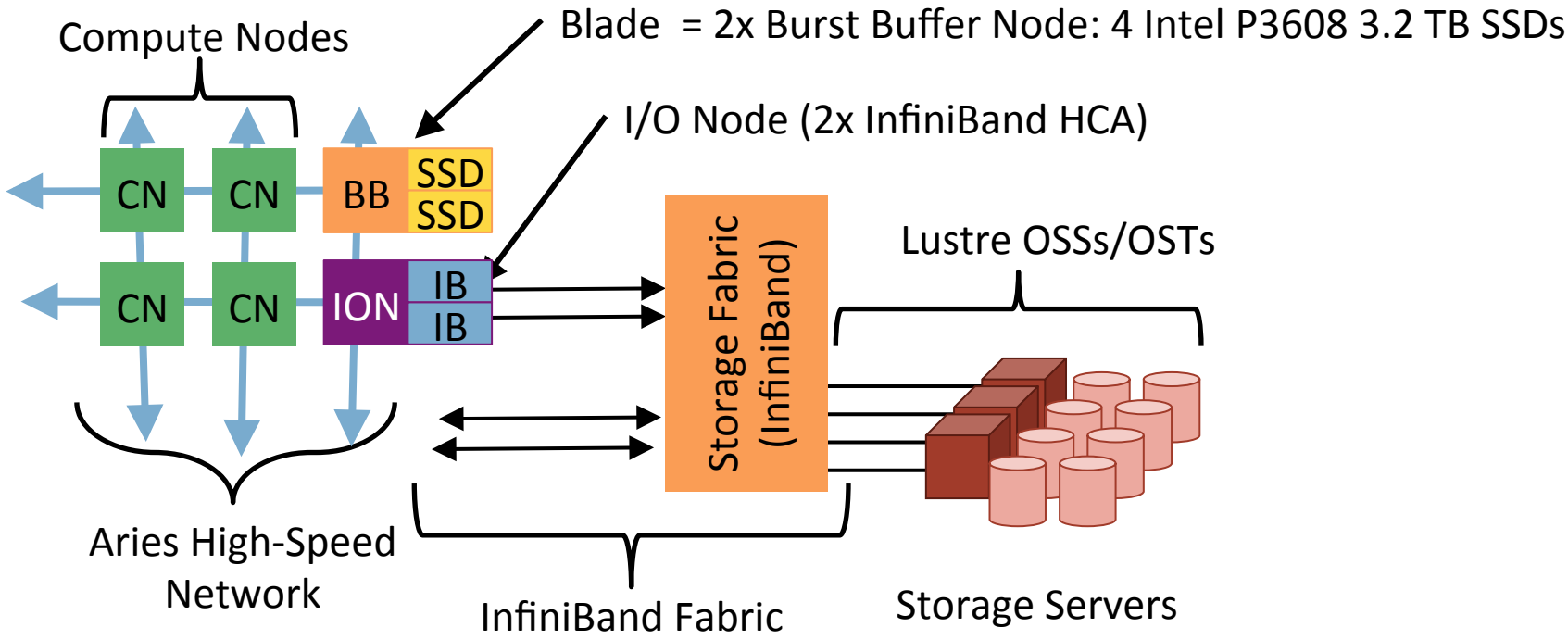


Why an SSD Burst Buffer?

- **Motivation: Handle spikes in I/O bandwidth requirements**
  - Reduce overall application run time
  - Compute resources are idle during I/O bursts
- **Some user applications have challenging I/O patterns**
  - High IOPs, random reads, different concurrency… perfect for SSDs
- **Cost rationale: Disk-based PFS bandwidth is expensive**
  - Disk capacity is relatively cheap
  - SSD *bandwidth* is relatively cheap

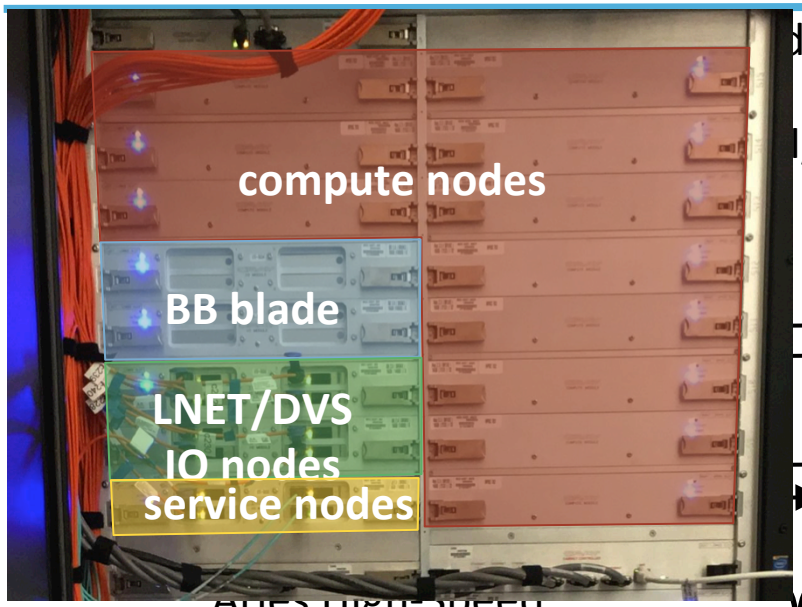
    =>Separate bandwidth and spinning disk
    - Provide high BW without wasting PFS capacity
    - Leverage Cray Aries network speed

NERSC/Cray Architecture



- DataWarp software (integrated with SLURM WLM) allocates portions of available storage to users per-job (or 'persistent').
- Users see a POSIX filesystem
- Filesystem can be striped across multiple BB nodes (depending on allocation size requested)

NERSC/Cray Architecture



de = 2x Burst Buffer Node: 4 Intel P3608 3.2 TB SSDs

/O Node (2x InfiniBand HCA)

Storage Fabric (InfiniBand)

Lustre OSSs/OSTs

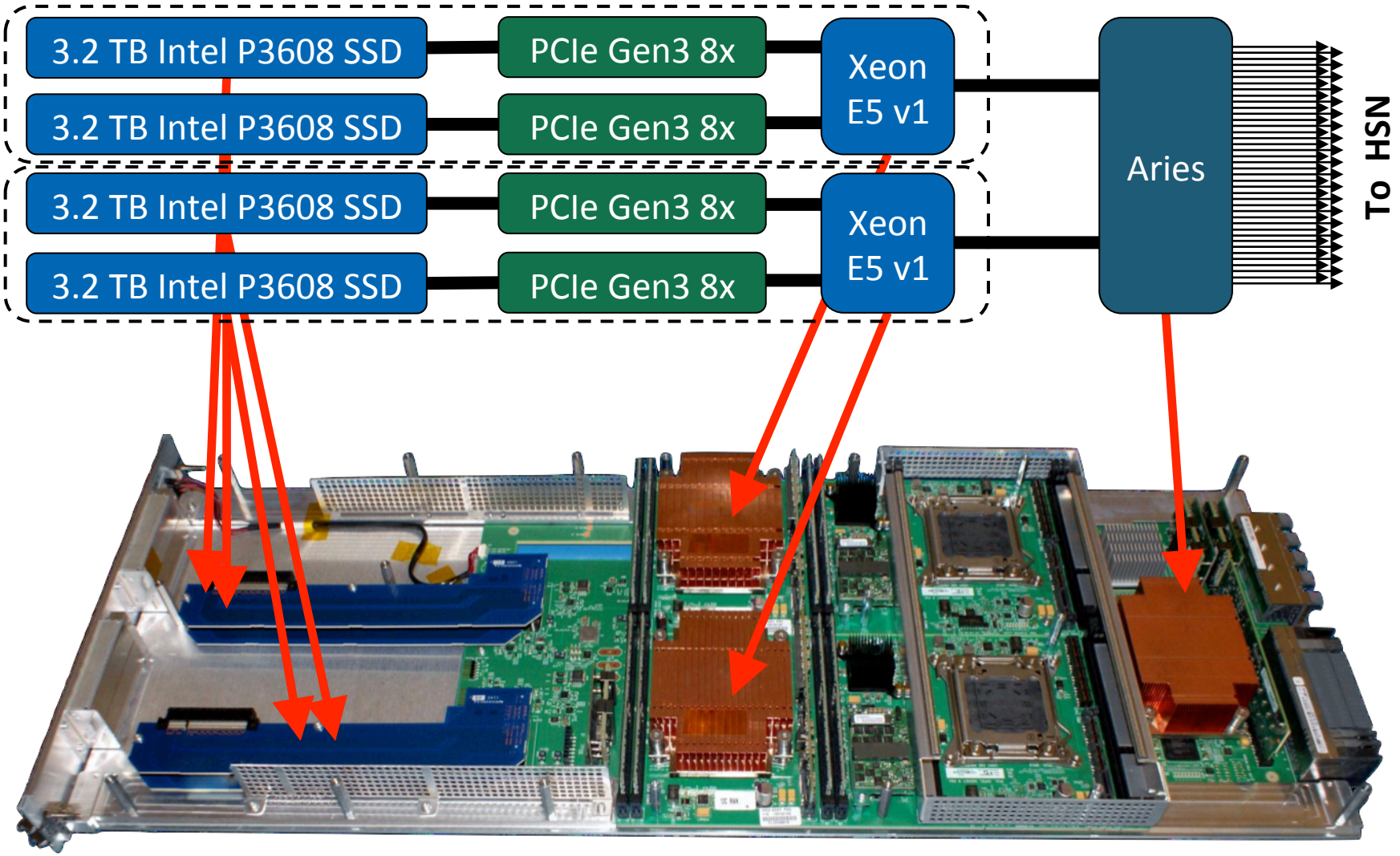
Aries High-Speed Network

InfiniBand Fabric

Storage Servers

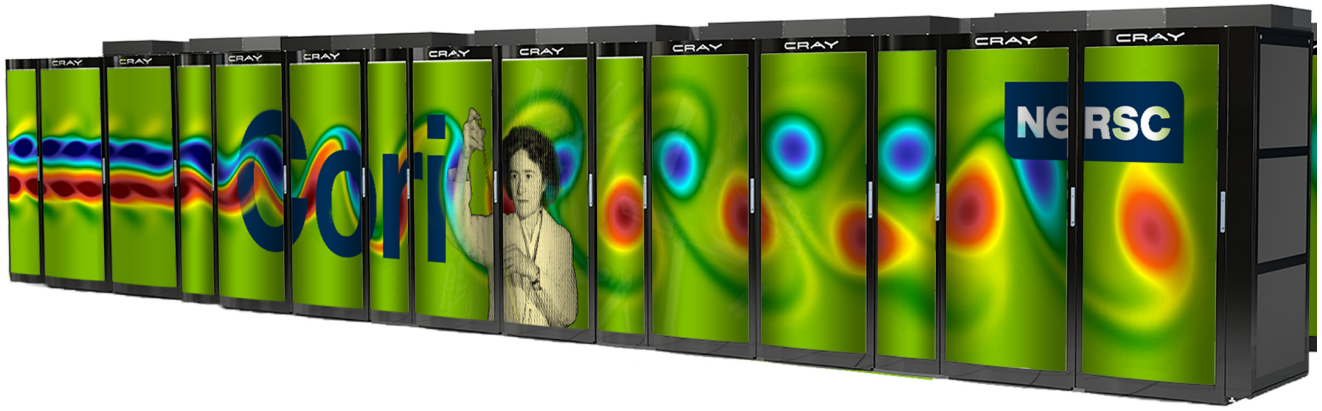
- DataWarp software (integrated with SLURM WLM) allocates portions of available storage to users per-job (or 'persistent').
- Users see a POSIX filesystem
- Filesystem can be striped across multiple BB nodes (depending on allocation size requested)

Burst Buffer Blade = 2xNodes



Cori, a Cray XC40 system

- **Cori Phase 1: partition to support data intensive applications**
  - –1630 Intel Haswell nodes: Two 16-core processors and 128 GB DDR4 /node,
- **Cori Phase 2: >9,300 Intel Knights Landing compute nodes**
- **Lustre Filesystem: 27 PB ; 248 OSTs;**
  - –700 GB/s peak performance
- **Cray Aries high-speed "dragonfly" topology interconnect**
- **Burst Buffer: ~2PB on 188 BB nodes;**
  - 900GB/s peak performance, 12.5M IOP/s on Phase 1 (144 BB nodes)

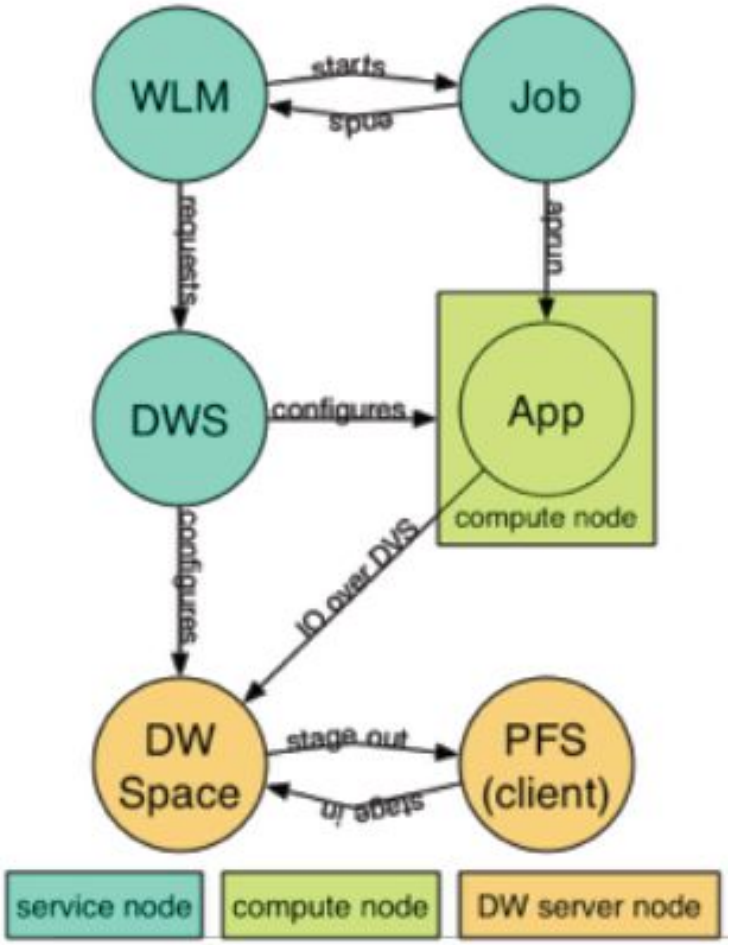


Cori, a Cray XC40 system

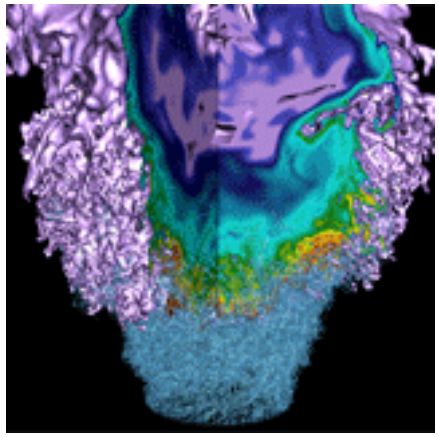
- **Cori Phase 1: partition to support data intensive applications**
  - –1630 Intel Haswell nodes: Two 16-core processors and 128 GB DDR4 /node,
- **Cori Phase 2: >9,300 Intel Knights Landing compute nodes**

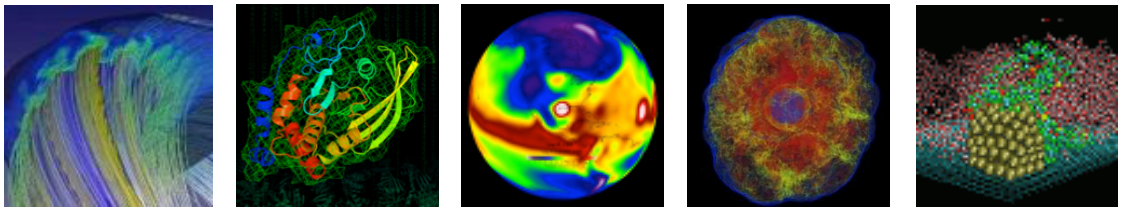
DataWarp: Under the hood

- Wokload Manager (Slurm) schedules job in the queue on Cori
- DataWarp Service (DWS) configures DW space and compute node access to DW
- DataWarp Filesystem handles stage interactions with PFS (Parallel File System, i.e. scratch)
- Compute nodes access DW via a mount point



Using Cray DataWarp at NERSC





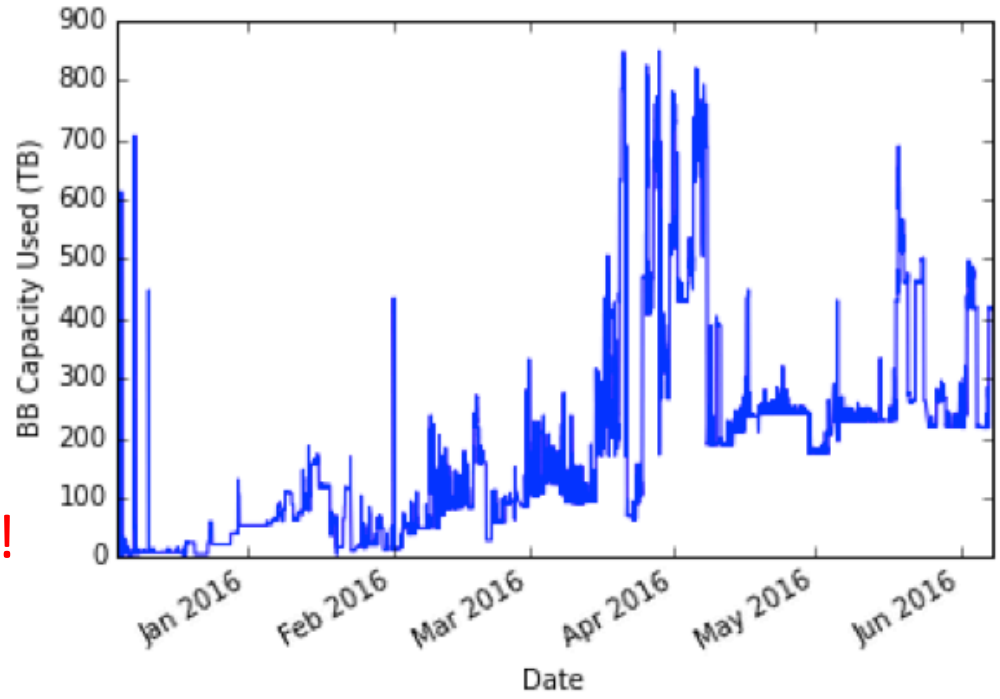
http://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/

Burst Buffer Early Users

- **Early User period: ~60 active users**
- **Access available upon request**
  - Email consult@nersc.gov to request BB access
  - USE CAUTION!

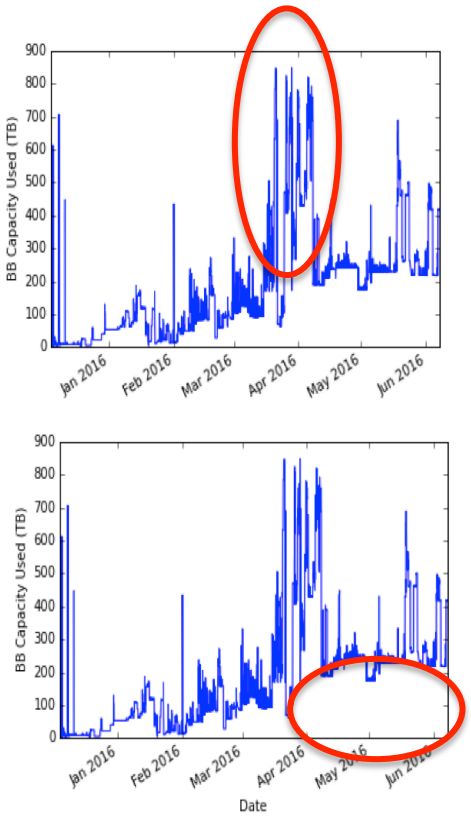
- Many bugs still being uncovered
- Lots of quirks of usage
- No guarantee of data integrity
- No guarantee of performance improvement…
- Read our "known issues" page!

https://www.nersc.gov/users/computational-systems/cori/burst-buffer/known-issues/



Two kinds of DataWarp Instances

- **"Instance": an allocation on the BB**
- **Can it be shared? What is its lifetime?**
  - **Per-Job Instance**
    - Can only be used by job that creates it
    - Lifetime is the same as the creating job
    - Use cases: PFS staging, application scratch, checkpoints
  - **Persistent Instance**
    - Can be used by any job (subject to UNIX file permissions)
    - Lifetime is controlled by creator
    - Use cases: Shared data, PFS staging, Coupled job workflow

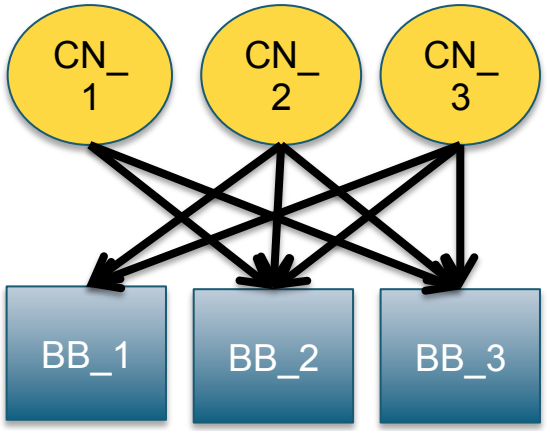


PFS I/O path

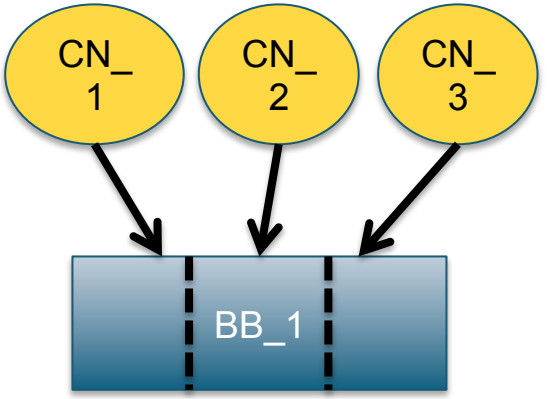
- **Each DataWarp node separately manages all PFS I/O for the files or stripes it contains**
  - –***Striped***: each DW node has a stripe of a file, multiple PFS clients per file
  - –***Private***: each DW node has an entire file, one PFS client per file
- **So I/O to PFS from DW is automatically done in parallel**
  - –Note that at present, can only access PFS (i.e. $CSCRATCH) from BB
- **Compute nodes are not involved with this PFS I/O**

Two DataWarp Access Modes

- **Striped ("Shared")**
  - –Files are striped across all DataWarp nodes
  - –Files are visible to all compute nodes
    Aggregates both capacity and BW per file
  - –One DataWarp node elected as the MDS



- **Private**
  - –Files are assigned to *one* DataWarp node
  - –File are visible to *only the compute node that created them*
  - –Each DataWarp node is an MDS for one or more computes
  - –Private,striped also available (but untested…)



How to use DataWarp

- **Principle user access: SLURM Job script directives: `#DW`**
  - –Allocate job or persistent DataWarp space
  - –Stage files or directories in from PFS to DW; out DW to PFS
  - –Access BB mount point via `$DW_JOB_STRIPED`, `$DW_JOB_PRIVATE`, `$DW_PERSISTENT_STRIPED_name`
- **User library API – `libdatawarp`**
  - –Allows direct control of staging files asynchronously
  - –C library interface
  - –https://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/#toc-anchor-8
  - –https://github.com/NERSC/BB-unit-tests/tree/master/datawarpAPI

Integration with SLURM

```
#!/bin/bash
#SBATCH –p regular –N 10 –t 00:10:00
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=/lustre/inputs destination=$DW_JOB_STRIPED/inputs \
type=directory
#DW stage_in source=/lustre/file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/outputs destination=/lustre/outputs \
type=directory
srun my.x --indir=$DW_JOB_STRIPED/inputs --infile=$DW_JOB_STRIPED/file.dat \
--outdir=$DW_JOB_STRIPED/outputs
```

- **'type=scratch' – duration just for compute job (i.e. not 'persistent')**
- **'access_mode=striped' – visible to all compute nodes (i.e. not 'private') and striped across multiple BB nodes**
  - –Actual distribution across BB Nodes is in units of (configurable) granularity (currently 200 GB at NERSC so 1000 GB would normally be placed on 5 BB nodes)
- **Data 'stage_in' before job start and 'stage_out' after**

Integration with SLURM

```
#!/bin/bash
#SBATCH -p regular -N 10 -t 00:10:00
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=/lustre/inputs destination=$DW_JOB_STRIPED/inputs \
type=directory
#DW stage_in source=/lustre/file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/outputs destination=/lustre/outputs \
type=directory
srun my.x --indir=$DW_JOB_STRIPED/inputs --infile=$DW_JOB_STRIPED/file.dat \
--outdir=$DW_JOB_STRIPED/outputs
```

- **'type=scratch' – duration just for compute job (i.e. not 'persistent')**
- **'access_mode=striped' – visible to all compute nodes (i.e. not 'private') and striped across multiple BB nodes**
  - –Actual distribution across BB Nodes is in units of (configurable) granularity (currently 200 GB at NERSC so 1000 GB would normally be placed on 5 BB nodes)
- **Data 'stage_in' before job start and 'stage_out' after**

Integration with SLURM

```
#!/bin/bash
#SBATCH –p regular –N 10 –t 00:10:00
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=/lustre/inputs destination=$DW_JOB_STRIPED/inputs \
type=directory
#DW stage_in source=/lustre/file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/outputs destination=/lustre/outputs \
type=directory
srun my.x --indir=$DW_JOB_STRIPED/inputs --infile=$DW_JOB_STRIPED/file.dat \
--outdir=$DW_JOB_STRIPED/outputs
```

- **'type=scratch' – duration just for compute job (i.e. not 'persistent')**
- **'access_mode=striped' – visible to all compute nodes (i.e. not 'private') and striped across multiple BB nodes**
  - –Actual distribution across BB Nodes is in units of (configurable) granularity (currently 200 GB at NERSC so 1000 GB would normally be placed on 5 BB nodes)
- **Data 'stage_in' before job start and 'stage_out' after**

Integration with SLURM

```
#!/bin/bash
#SBATCH –p regular –N 10 –t 00:10:00
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=/lustre/inputs destination=$DW_JOB_STRIPED/inputs \
type=directory
#DW stage_in source=/lustre/file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/outputs destination=/lustre/outputs \
type=directory
srun my.x --indir=$DW_JOB_STRIPED/inputs --infile=$DW_JOB_STRIPED/file.dat \
--outdir=$DW_JOB_STRIPED/outputs
```

- **'type=scratch' – duration just for compute job (i.e. not 'persistent')**
- **'access_mode=striped' – visible to all compute nodes (i.e. not 'private') and striped across multiple BB nodes**
  - –Actual distribution across BB Nodes is in units of (configurable) granularity (currently 200 GB at NERSC so 1000 GB would normally be placed on 5 BB nodes)
- **Data 'stage_in' before job start and 'stage_out' after**

Integration with SLURM

- **Using a *persistent* DataWarp instance**
  - –Lifetime different from the batch job
  - –Usable by any batch job
  - –`name=xyz:` Name of persistent instance to use

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB create_persistent name=myBBname capacity=10GB access=striped type=scratch
```

Delete

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB destroy_persistent name=myBBname
```

Use in another job

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#DW persistentdw name=myBBname
mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```

Integration with SLURM

- **Using a *persistent* DataWarp instance**
  - Lifetime different from the batch job
  - Usable by any batch job
  - `name=xyz`: Name of persistent instance to use

```
1 #!/bin/bash
2 #SBATCH -p debug
3 #SBATCH -N 1
4 #SBATCH -t 00:05:00
5 #BB create_persistent name=myBBname capacity=10GB access=striped type=scratch
```

Delete

```
1 #!/bin/bash
2 #SBATCH -p debug
3 #SBATCH -N 1
4 #SBATCH -t 00:05:00
5 #BB destroy_persistent name=myBBname
```

Use in another job

```
1 #!/bin/bash
2 #SBATCH -p debug
3 #SBATCH -N 1
4 #SBATCH -t 00:05:00
5 #DW persistentdw name=myBBname
6 mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
7 srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```

Integration with SLURM

- **Using a *persistent* DataWarp instance**
  - Lifetime different from the batch job
  - Usable by any batch job
  - `name=xyz:` Name of persistent instance to use

```
1 #!/bin/bash
2 #SBATCH -p debug
3 #SBATCH -N 1
4 #SBATCH -t 00:05:00
5 #BB create_persistent name=myBBname capacity=10GB access=striped type=scratch
```

Delete

```
1 #!/bin/bash
2 #SBATCH -p debug
3 #SBATCH -N 1
4 #SBATCH -t 00:05:00
5 #BB destroy_persistent name=myBBname
```

Use in another job

```
1 #!/bin/bash
2 #SBATCH -p debug
3 #SBATCH -N 1
4 #SBATCH -t 00:05:00
5 #DW persistentdw name=myBBname
6 mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
7 srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```

Integration with SLURM

- **Using a *persistent* DataWarp instance**
  - Lifetime different from the batch job
  - Usable by any batch job
  - `name=xyz:` Name of persistent instance to use

```
1 #!/bin/bash
2 #SBATCH -p debug
3 #SBATCH -N 1
4 #SBATCH -t 00:05:00
5 #BB create_persistent name=myBBname capacity=10GB access=striped type=scratch
```

Delete

```
1 #!/bin/bash
2 #SBATCH -p debug
3 #SBATCH -N 1
4 #SBATCH -t 00:05:00
5 #BB destroy_persistent name=myBBname
```

Use in another job

```
1 #!/bin/bash
2 #SBATCH -p debug
3 #SBATCH -N 1
4 #SBATCH -t 00:05:00
5 #DW persistentdw name=myBBname
6 mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
7 srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```

Resources

- NERSC Burst Buffer Web Pages

http://www.nersc.gov/users/computational-systems/cori/burst-buffer/

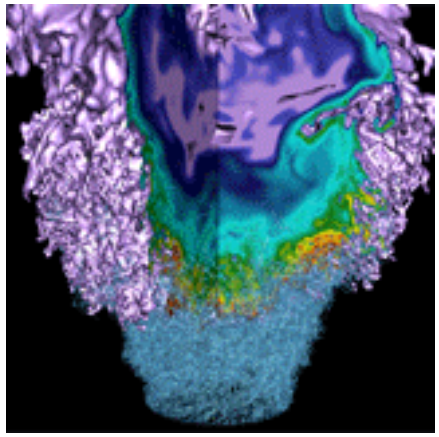
- Example batch scripts

http://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/

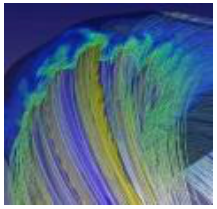
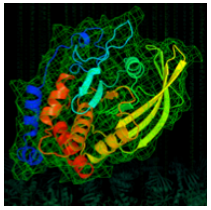
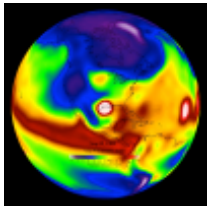
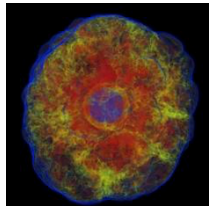
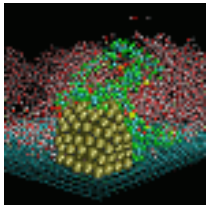
- Burst Buffer Early User Program Paper

http://www.nersc.gov/assets/Uploads/Nersc-BB-EUP-CUG.pdf

Science Use Cases



NERSC

Or,
Real code ≠ benchmarks

Benchmark Performance

- **Burst Buffer is doing well against benchmark performance targets**
  - Out-performs Lustre (in tests using half the full Burst Buffer and only a fraction of the full Cori compute load)

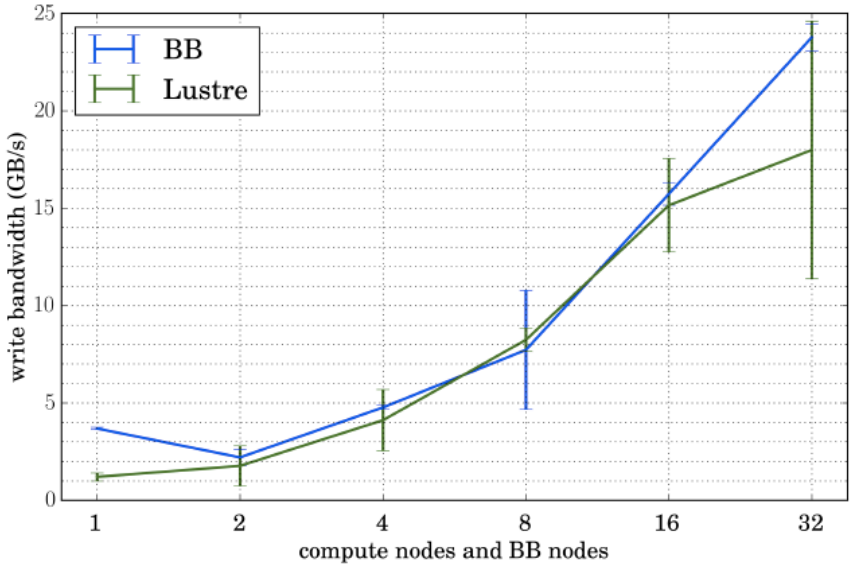
	IOR Posix FPP		IOR MPIO Shared File		IOPS	
	Read	**Write**	**Read**	**Write**	**Read**	**Write**
Best Measured (140 Burst Buffer Nodes : 1120 Compute Nodes; 4 ranks/node)*	905 GB/s	873 GB/s	803 GB/s	351GB/s (since improved)	12.6 M	12.5 M
Lustre (peak – 24 OSTs: 930 compute nodes, 4 ranks/ node; 4 MB transfer)	708 GB/s	751 GB/s	573 GB/s	223 GB/s	-	-

**Bandwidth tests: 8 GB block-size 1MB transfers IOPS tests: 1M blocks 4k transfer*

I/O R/W use case: Nyx/Boxlib

Brian Friesen and Vince Beckner

- **Cosmology simulation code**
  - Writing out large data files at regular time steps
- **Able to achieve max BW to single BB node**
  - tweaking # MPI writers, transfer size.
- **BB performance scales up as you increase # BB nodes in allocation**
  - Aggregate BW of all BB nodes
- **BB performance matches Lustre…**

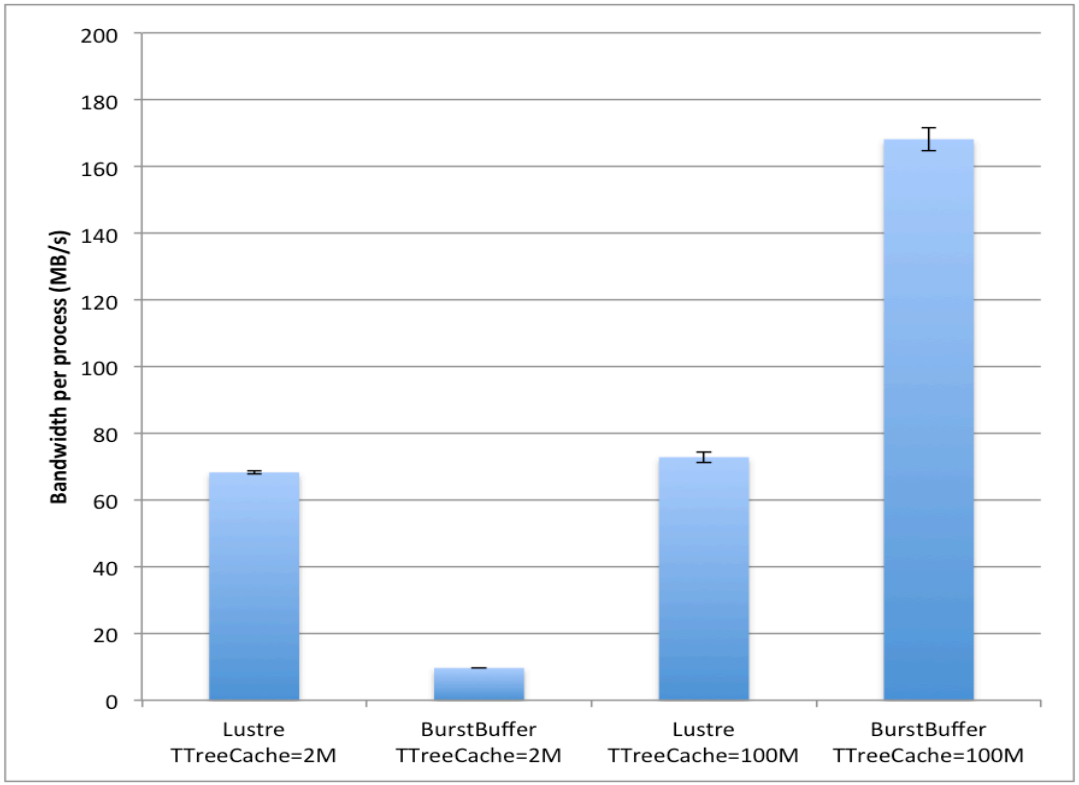


- **Note that this does not necessarily correspond to optimal Nyx *compute* configuration!**

Challenging IO use case: ATLAS data

Steve Farrell and Wahid Bhimji

- Initial study of I/O intensive data processing
- Reading 475 GB dataset in custom ROOT format
- 32 forked processes per node, FPP R/W
- Initial result: BB performs poorly compared to Lustre.

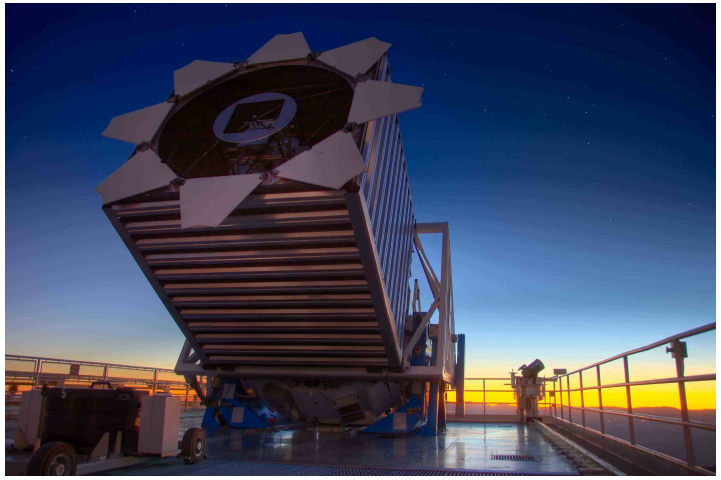


- Increase application memory cache to 100 M
  - Less reads – > 17x performance boost on BB

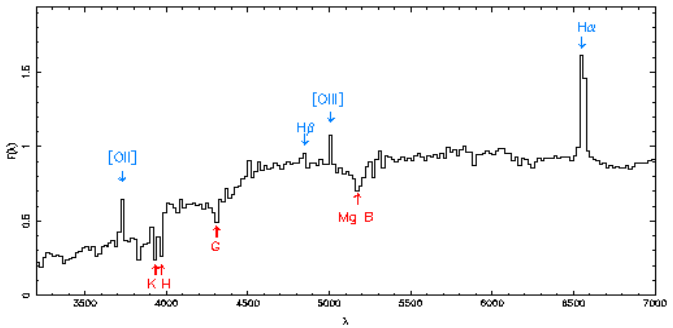
Challenging IO use case: Astronomy data

Jialin Liu and Debbie Bard

- **Selecting subsets of galaxy spectra from a large dataset**
  - –Small, random memory accesses
  - –Typical web query for SDSS dataset



Time taken to extract 1000 random spectra	From one hdf5 file	From individual fits files
From Lustre	44.1s	160.3s
From BB	1.3s	44.0s
Speedup:	**33x**	**3.6x**



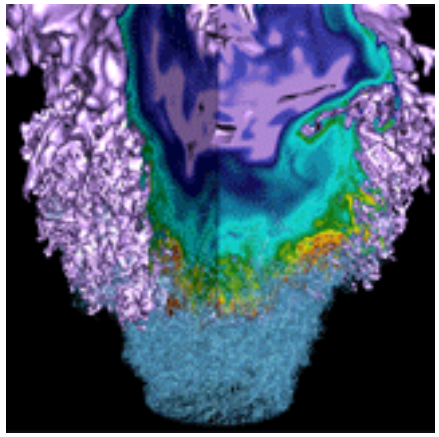
Summary

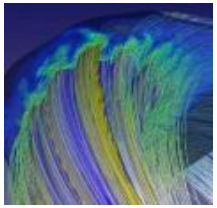
- **NERSC has the first Burst Buffer for open science in the USA**
  - –And the first in the world that is being tested for real use cases!
- **Users are able to take advantage of SSD performance**
  - –Some tuning may be required to maximise performance
- **Many bugs now worked through**
  - –But care is needed when using this new technology!
- **User experience today is generally good**
  - –Let us know if you're interested in using the Burst Buffer…

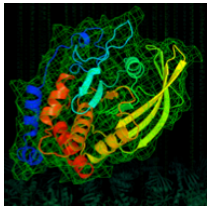
More information…

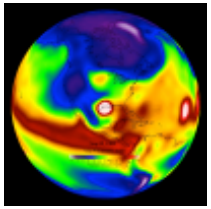
- https://www.nersc.gov/users/computational-systems/cori/burst-buffer/
- **CUG Best Paper award:** https://www.nersc.gov/assets/Uploads/Nersc-BB-EUP-CUG.pdf
- **Salishan 2016 talk:** http://salishan.ahsc-nm.org/uploads/4/9/7/0/49704495/2016-bard.pdf

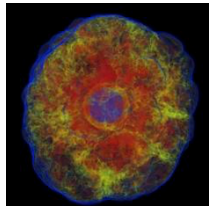
Extra slides

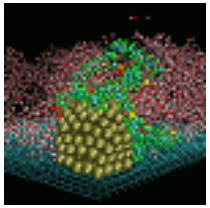










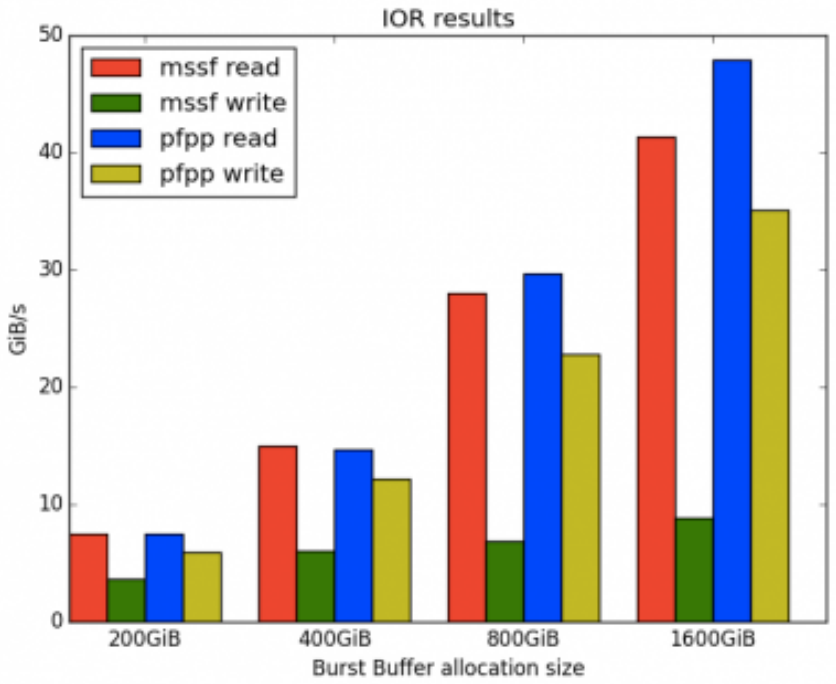


SSD write protection

- **SSDs support a set amount of write activity before they wear out**
- **Runaway application processes may write an excessive amount of data, and therefore, "destroy" the SSDs**
- **Three write protection policies**
  - –Maximum number of bytes written in a period of time
  - –Maximum size of a file in a namespace
  - –Maximum number of files allowed to be created in a namespace
- **Log, error, log and error**
  - –-`EROFS` (write window exceeded)
  - –-`EMFILE` (maximum files created exceeded)
  - –-`EFBIG` (maximum file size exceeded)

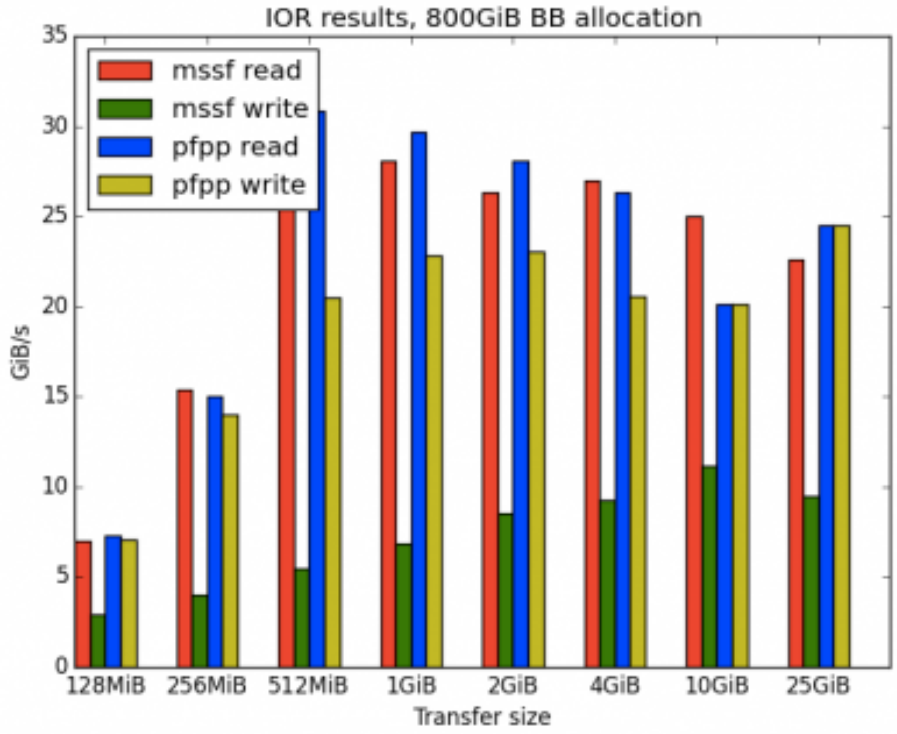
Performance Tips - striping

- **Request larger allocation to stripe over more BB nodes**
- Current granularity is 213GB so requesting more space will stripe
- This granularity will likely be lowered after Rhine/Redwood upgrade



Transfer sizes

- **Use larger transfer sizes**
  - –>~ 512KB
- **Consider Cray iobuf library for smaller transfer sizes**
  - –module load iobuf
  - –man iobuf
  - –Have to recompile
- DVS client-side caching will improve low transfer performance later in the year



MPI-IO

- **Disable MPI-IO Collective buffering (for now)**

```
export MPICH_MPIIO_HINTS="*:romio_cb_write=disable
```

- **If do use collective buffering then number of aggregators important:**