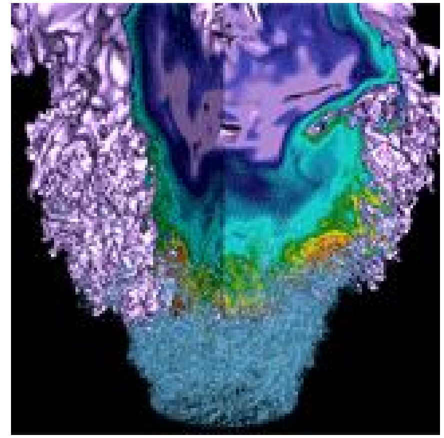
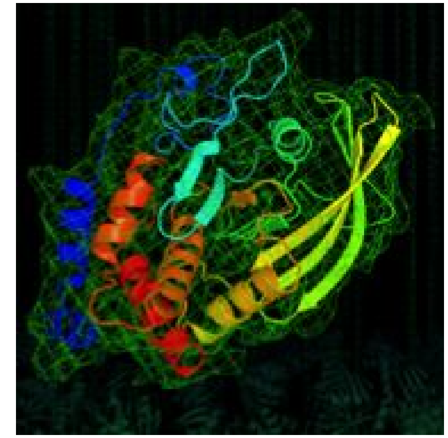
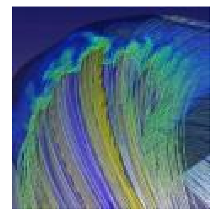
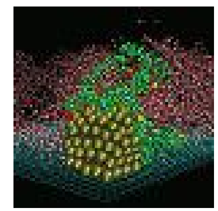
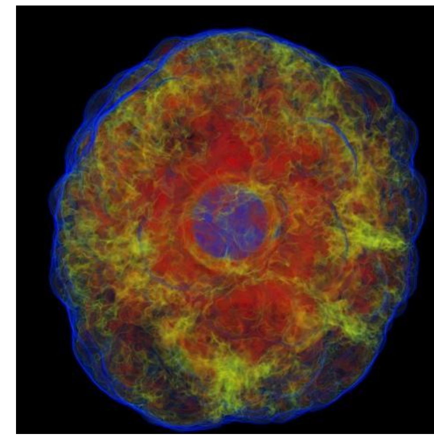
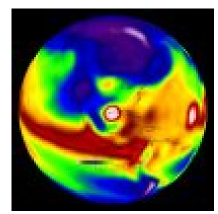
Accelerate your IO with the Burst Buffer


Debbie Bard
Data and Analytics Services
NERSC
ATPSEC IO day

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

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)

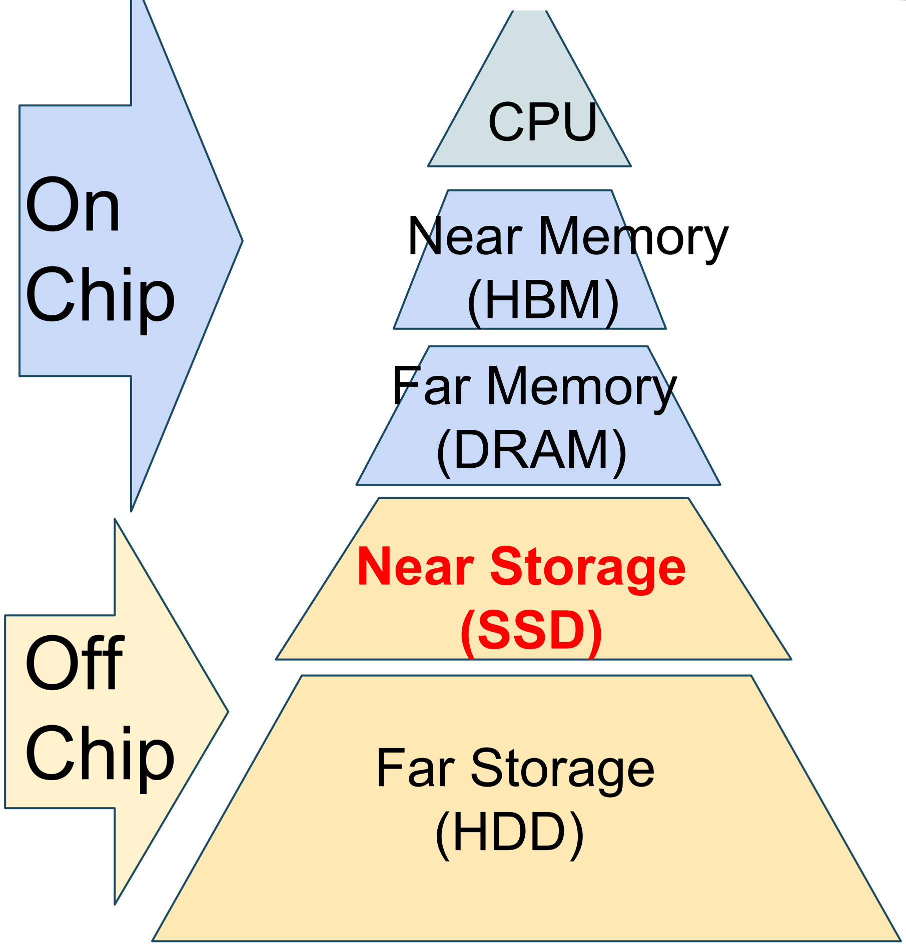
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

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*

Future

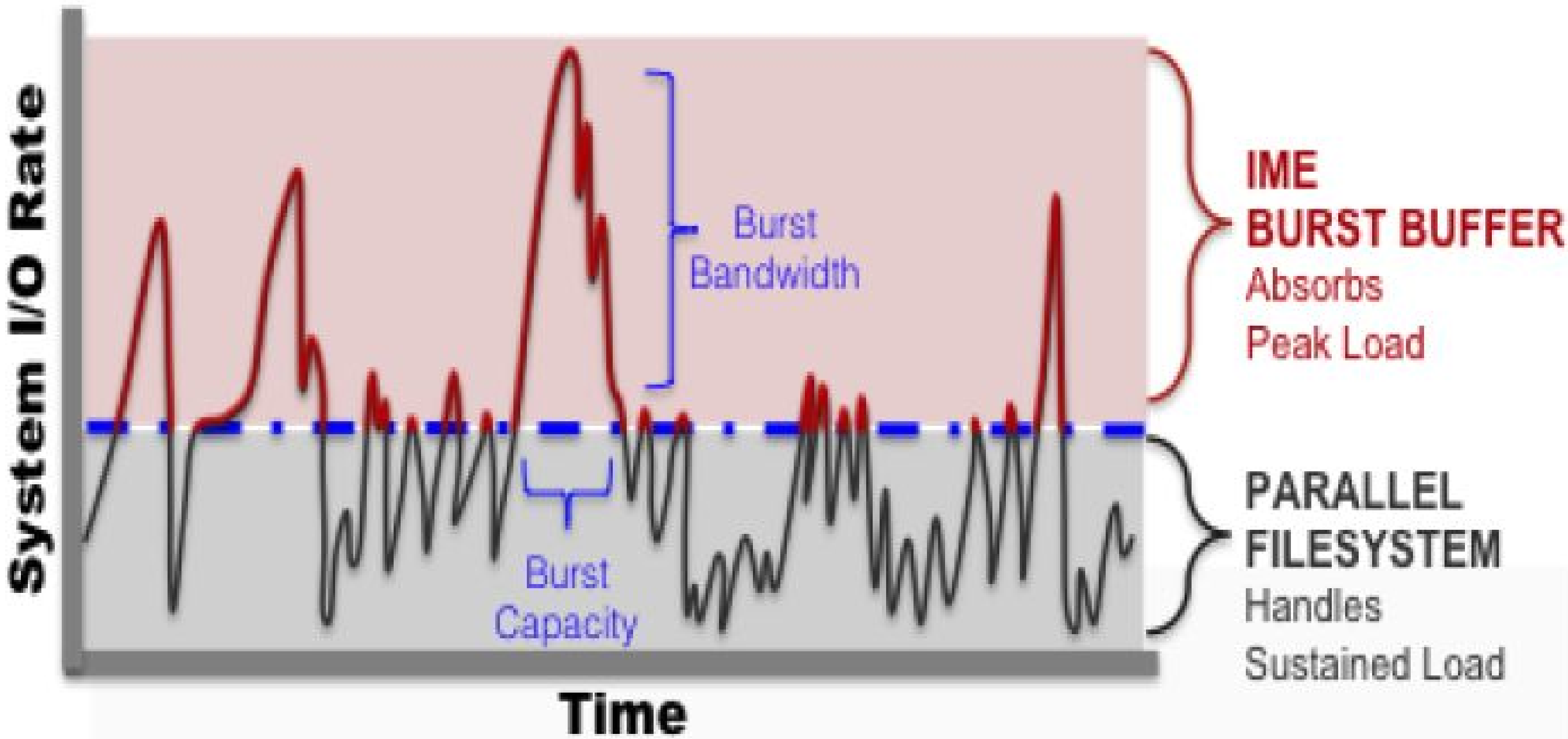


SSD vs HDD

- **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.

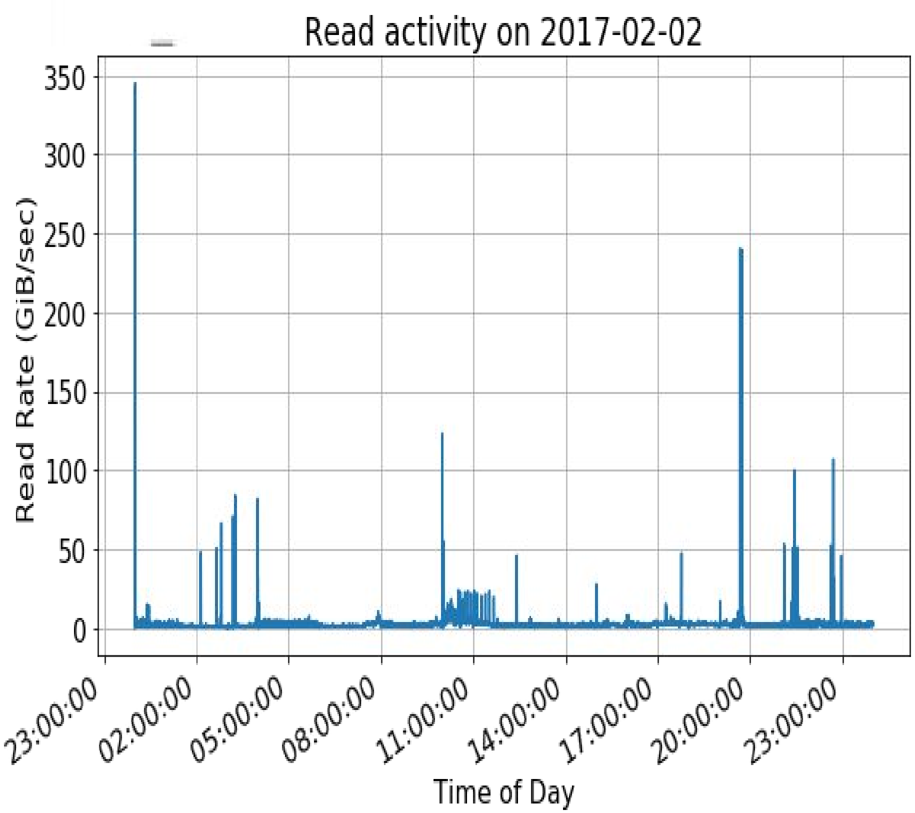
Why an SSD Burst Buffer?

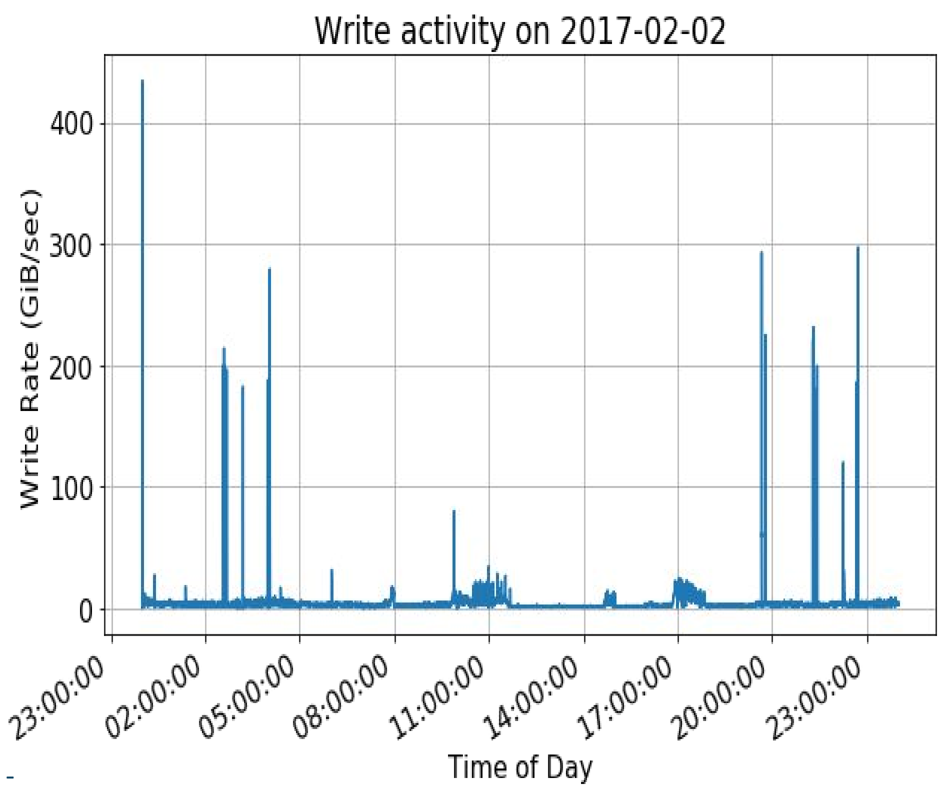
- **Motivation: Handle spikes in I/O bandwidth requirements**
  - Reduce overall application run time
  - Compute resources are idle during I/O bursts



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



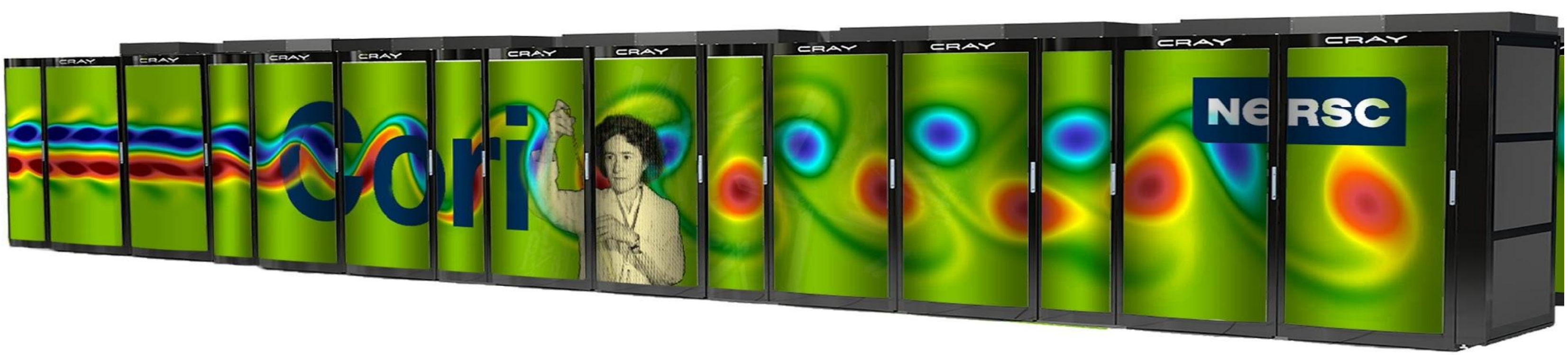


Why an SSD Burst Buffer?

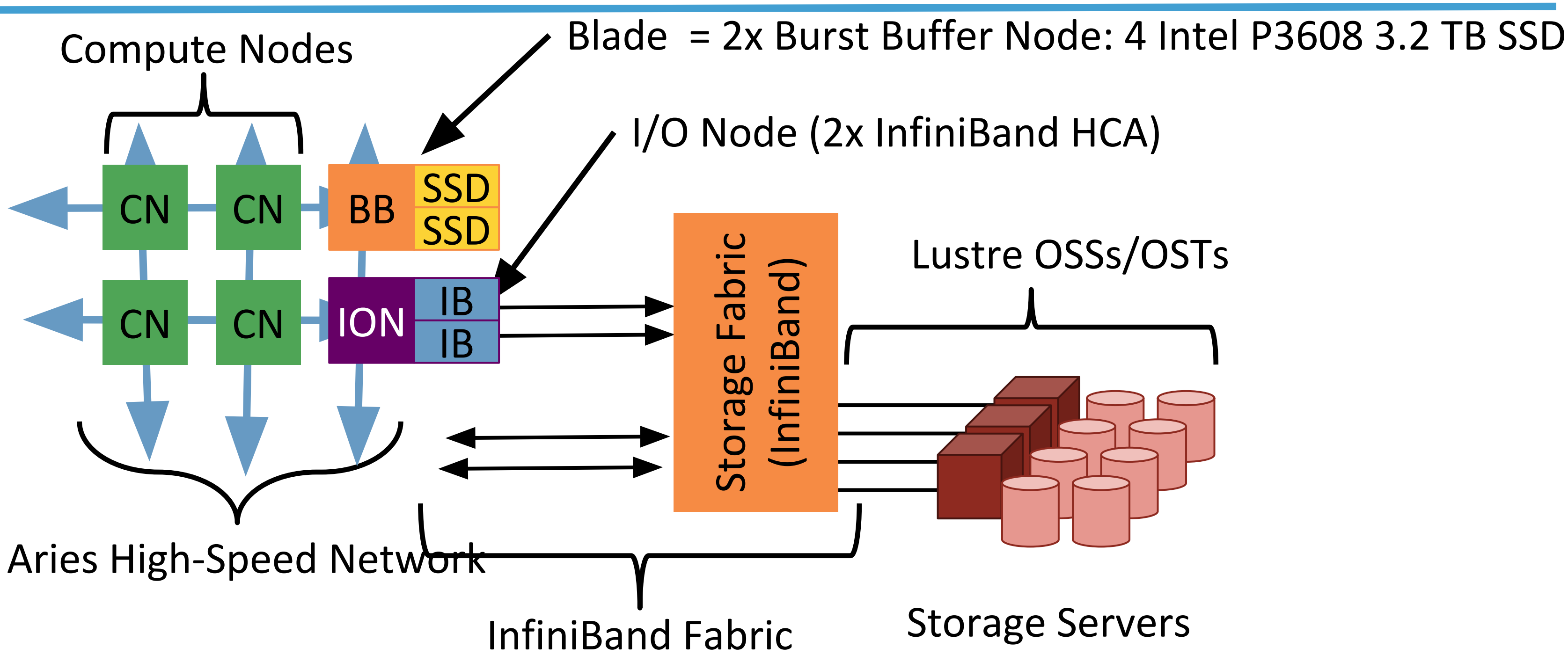
- **Motivation: Handle spikes in I/O bandwidth requirements**
  – Reduce overall application run time
  – Compute resources are idle during I/O bursts
- **Some user applications have challenging I/O patterns**
  – High IOPs, random reads, different concurrency… perfect for SSDs
- **Cost rationale: Disk-based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

Cori @ NERSC

- **NERSC at LBL, production HPC center for DoE**
  - >6000 diverse users across all DoE science domains
- **Cori – NERSCs Newest Supercomputer – Cray XC40**
  - 2,388 Intel Haswell dual 16-core nodes
  - 9,688 Intel Knights Landing Xeon Phi nodes, 68 cores
- **Cray Aries high-speed “dragonfly” topology interconnect**
- **Lustre Filesystem: 27 PB ; 248 OSTs; 700 GB/s peak performance**
- **1.8PB of Burst Buffer**

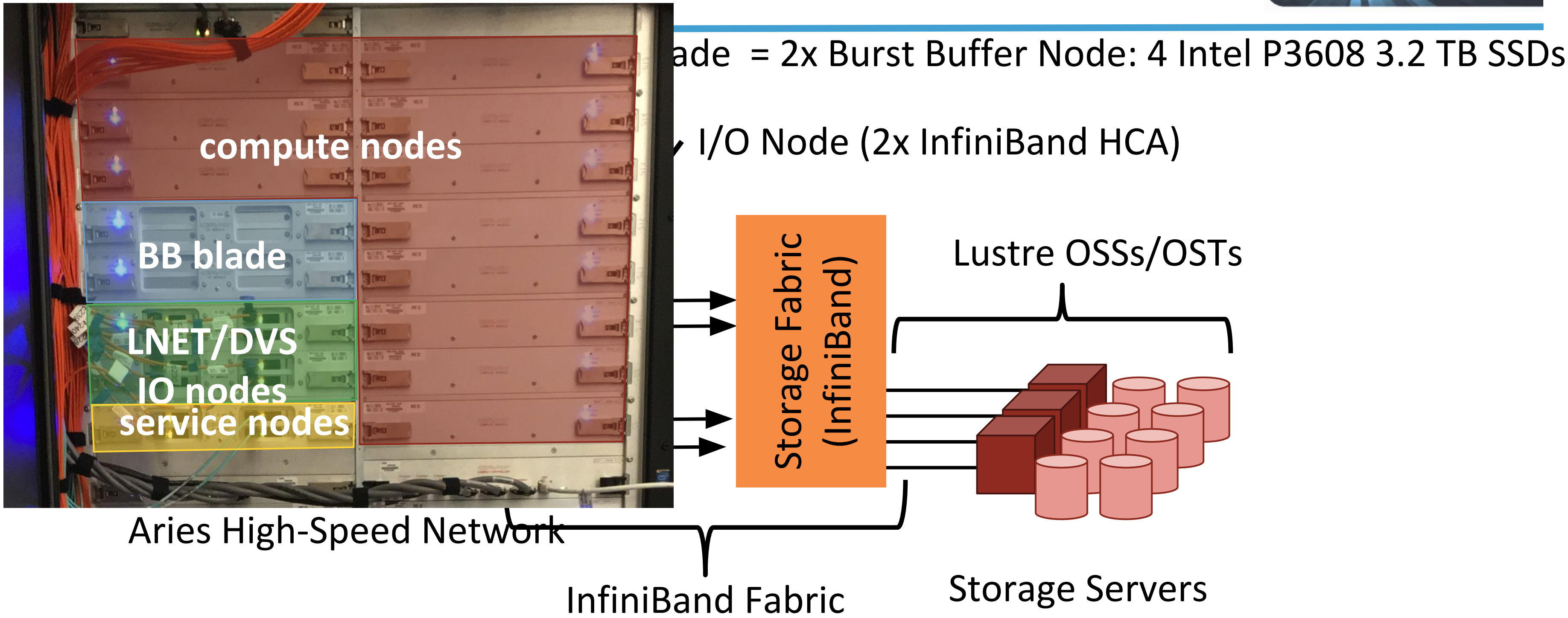


Burst Buffer Architecture



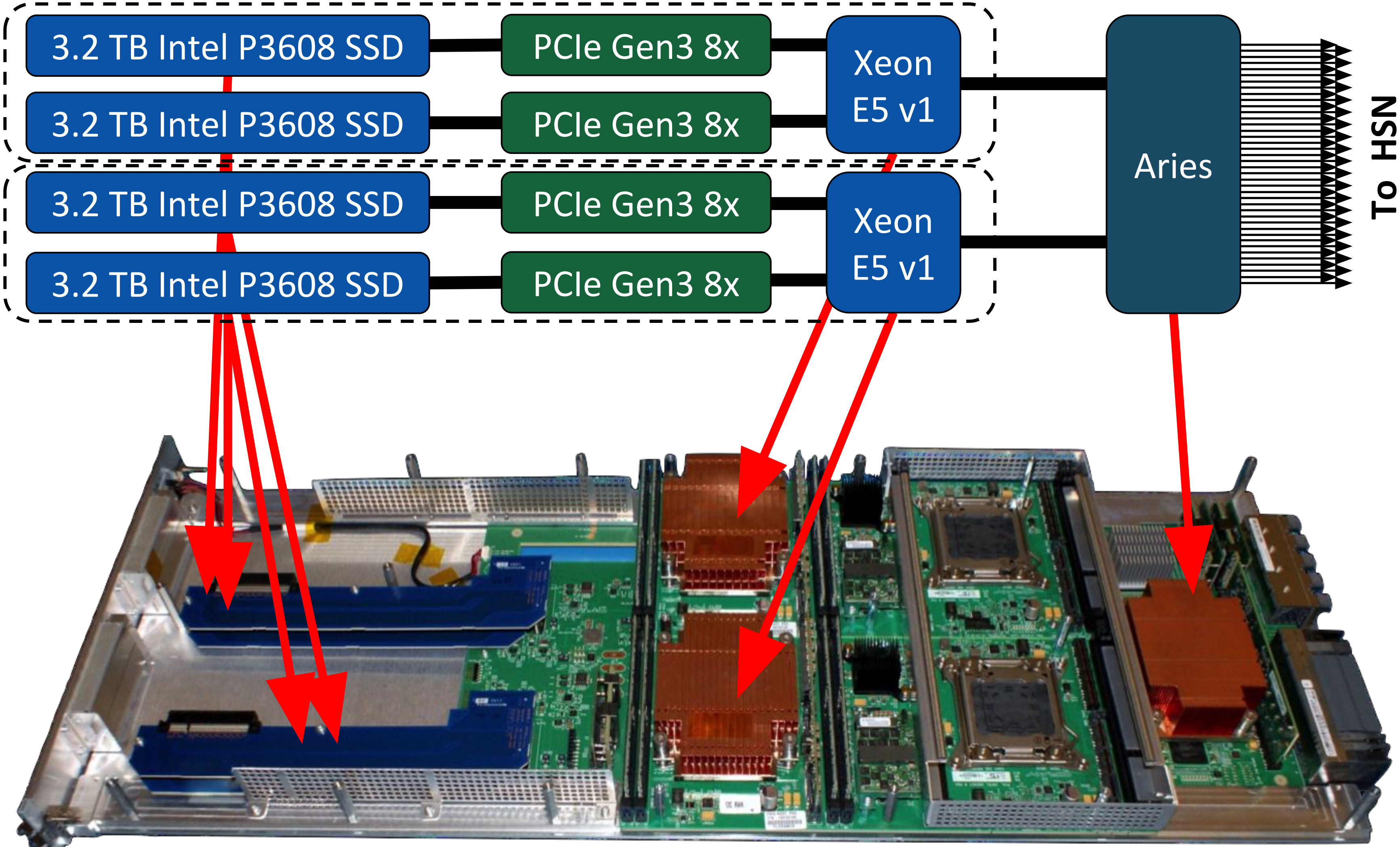
- DataWarp software (integrated with SLURM WLM) allocates portions of available storage to users per-job (or ‘persistent’).
- Users see a POSIX filesystem
- Filesystem can be striped across multiple BB nodes (depending on allocation size requested)

Burst Buffer Architecture

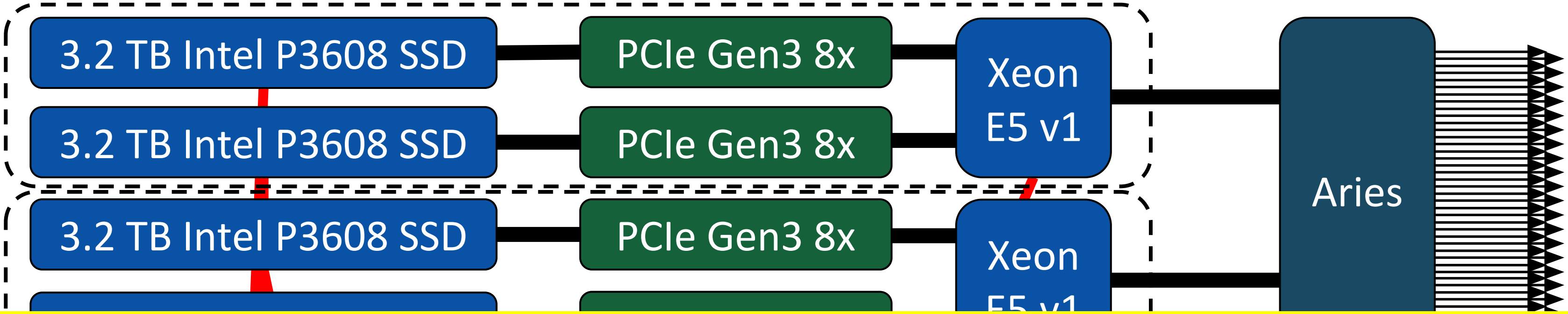


- DataWarp software (integrated with SLURM WLM) allocates portions of available storage to users per-job (or 'persistent').
- Users see a POSIX filesystem
- Filesystem can be striped across multiple BB nodes (depending on allocation size requested)

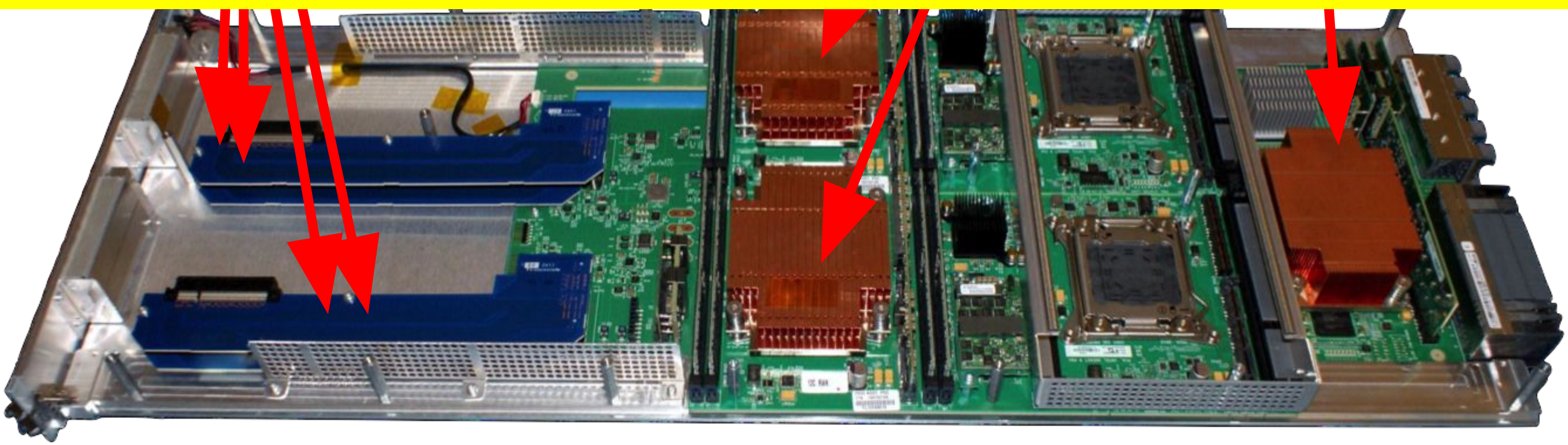
Burst Buffer Blade = 2xNodes



Burst Buffer Blade = 2xNodes



- ~1.8PiB of SSDs over 288 nodes
- Accessible from both HSW and KNL nodes

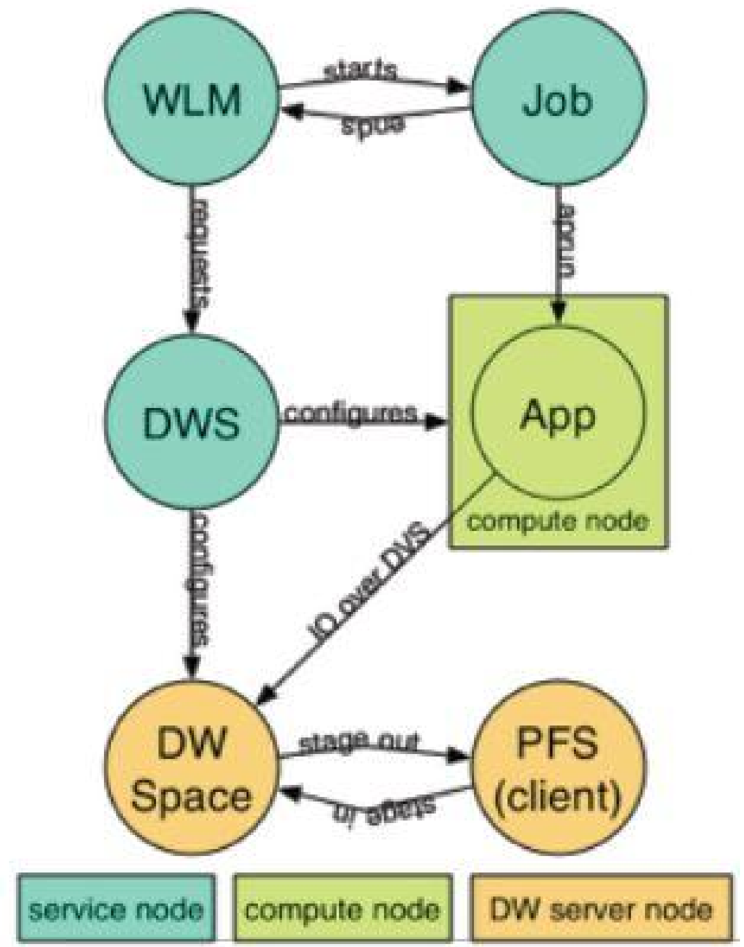


Why not node-local SSDs?

- **Average >1000 jobs running on Cori at any time**
- **Diverse workload**
  - –Many NERSC users are IO-bound
  - –Small-scale compute jobs, large-scale IO needs
- **Persistent BB reservations enable medium-term data access without tying up compute nodes**
  - –Multi-stage workflows with differing concurrencies can simultaneously access files on BB.
- **Easier to stream data directly into BB from external experiment**
- ***Configurable BB makes sense for our user load***

DataWarp: Under the hood

- Workload 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

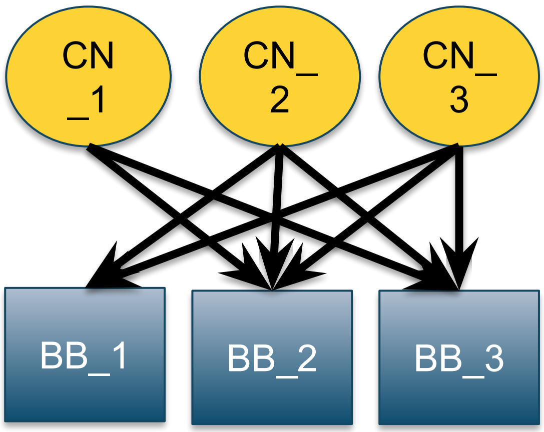


Two kinds of DataWarp Instances

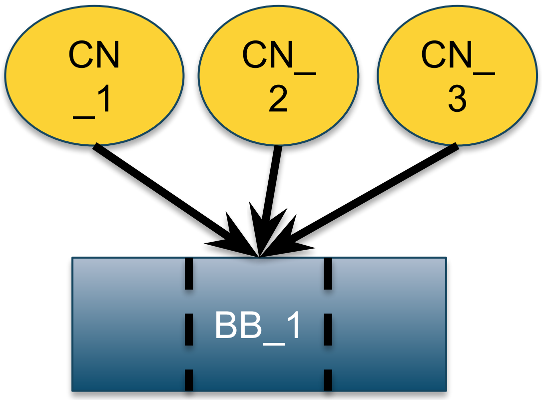
- **"Instance": an allocation on the BB**
- **Can it be shared? What is its lifetime?**
  - **Per-Job Instance**
    - Can only be used by job that creates it
    - Lifetime is the same as the creating job
    - Use cases: PFS staging, application scratch, checkpoints
  - **Persistent Instance**
    - 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
    - ***NOT for long-term storage of data!***

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 metadata server (MDS)



- **Private**
  - Files are assigned to one or more DataWarp node (can chose to stripe)
  - File are visible to *only the compute node that created them*
  - Each DataWarp node is an MDS for one or more compute nodes



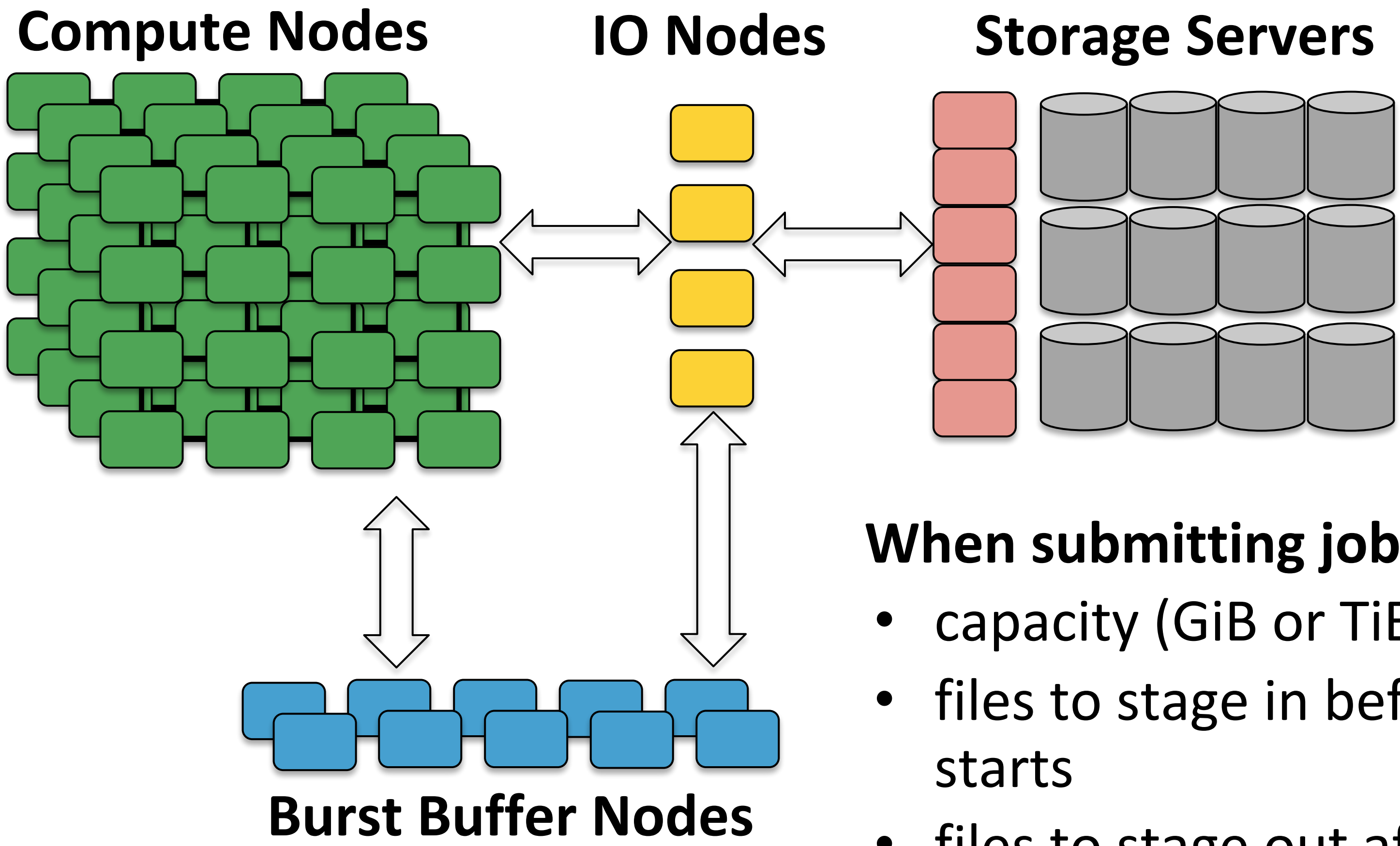
Striping, granularity and pools

- **DataWarp nodes are configured to have "granularity"**
  - Minimum amount of data that will land on one node
- **Two "pools" of DataWarp nodes, with different granularity**
  - wlm_pool (default): 82GiB
    - `#DW jobdw capacity=1000GB access_mode=striped type=scratch pool=wlm_pool`
  - sm_pool: 20.14 GiB
    - `#DW jobdw capacity=1000GB access_mode=striped type=scratch pool=sm_pool`
- **For example, 1.2TiB will be striped over 15 BB nodes in wlm_pool, but over 60 BB nodes in sm_pool**
  - No guarantee that allocation will be spread evenly over SSDs - may see >1 "grain" on a single node

I/O PFS ↔ BB

- **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***: if not “private, striped”, each DW node has an entire file, one PFS client per node
- **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***

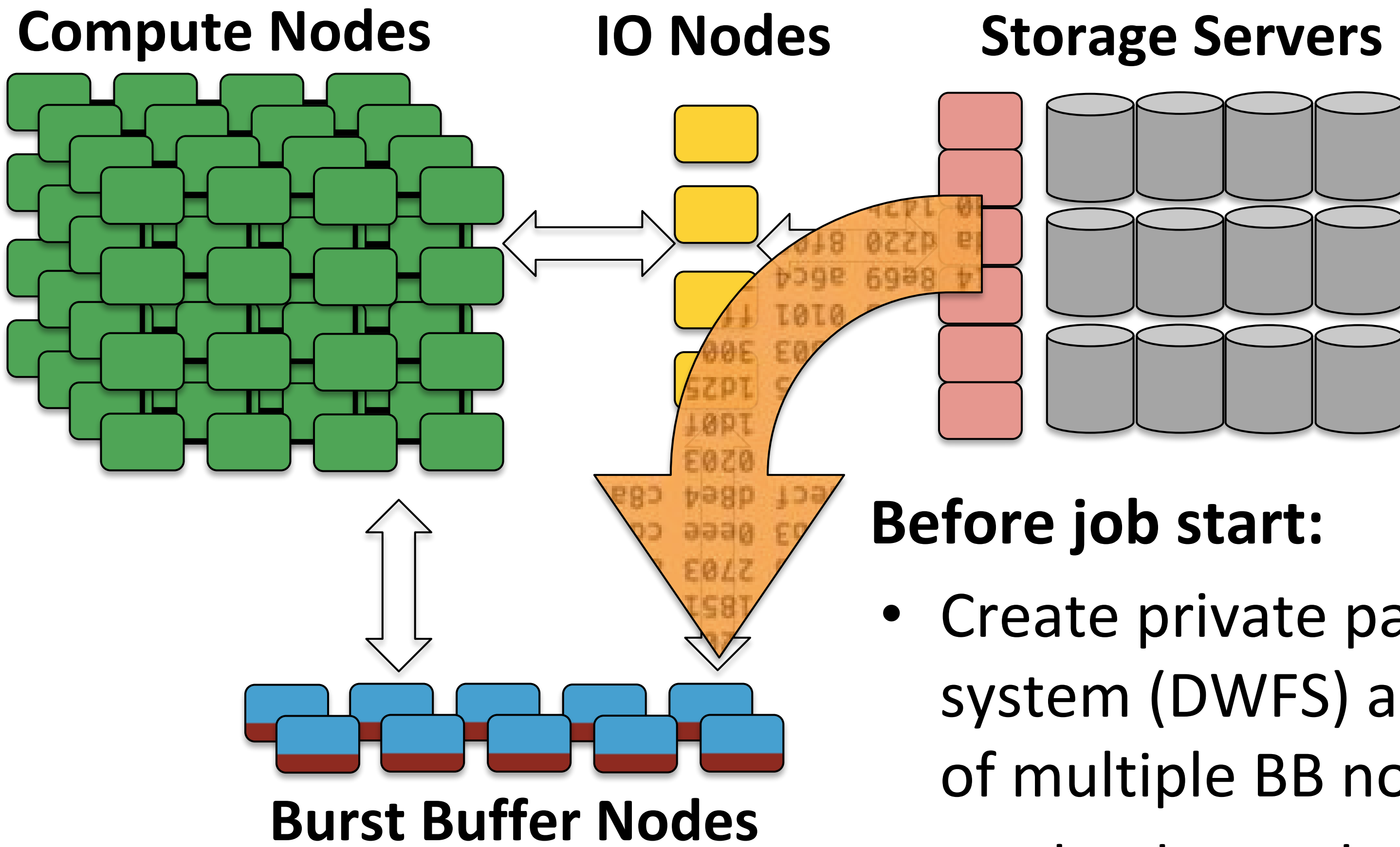
Cori's Data Paths



When submitting job, request:

- capacity (GiB or TiB)
- files to stage in before job starts
- files to stage out after job finishes

Cori's Data Paths

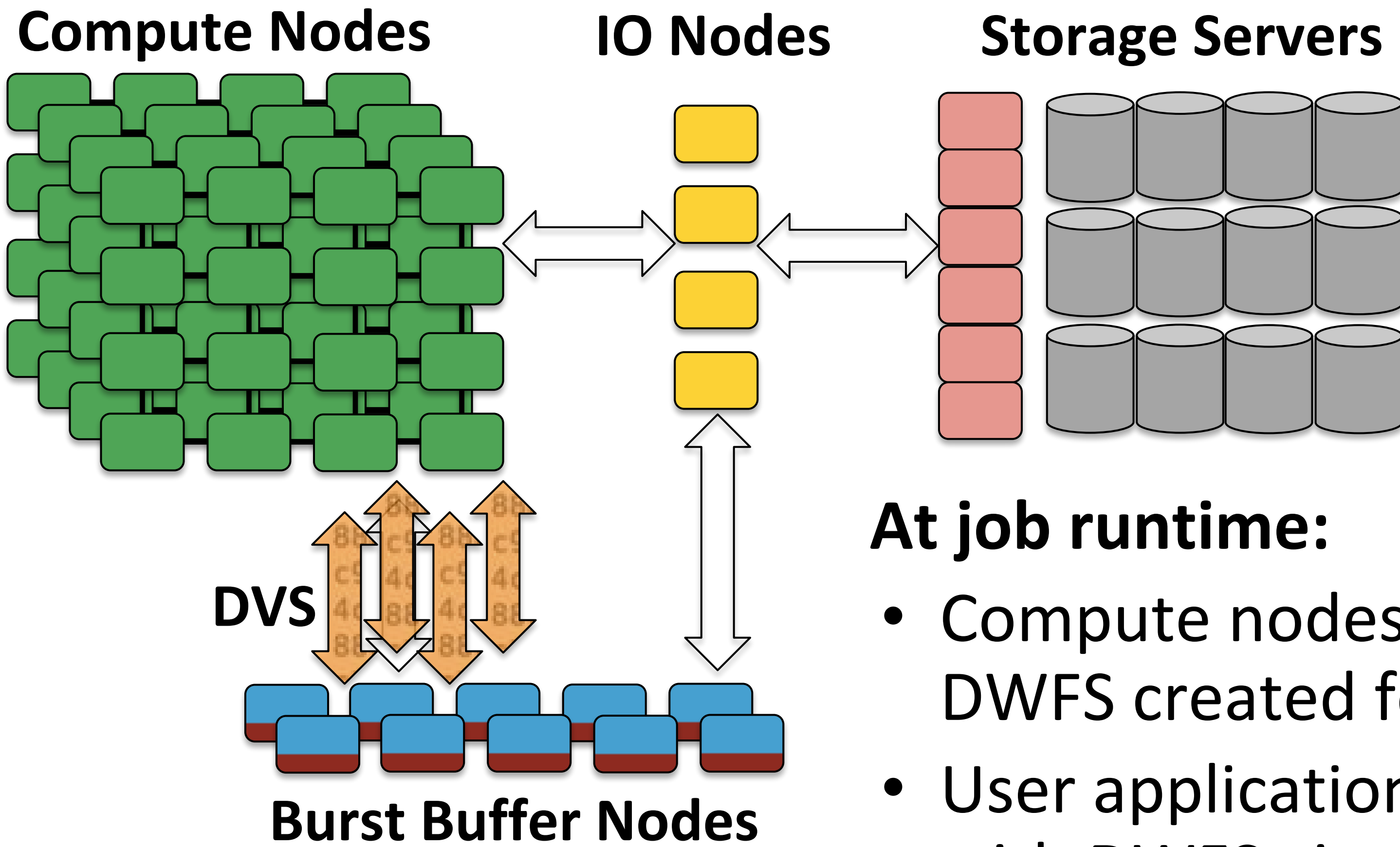


Before job start:

- Create private parallel file system (DWFS) across parts of multiple BB nodes
- Pre-load user data into this DWFS

Slides from Glenn Lockwood, NERSC

Cori's Data Paths

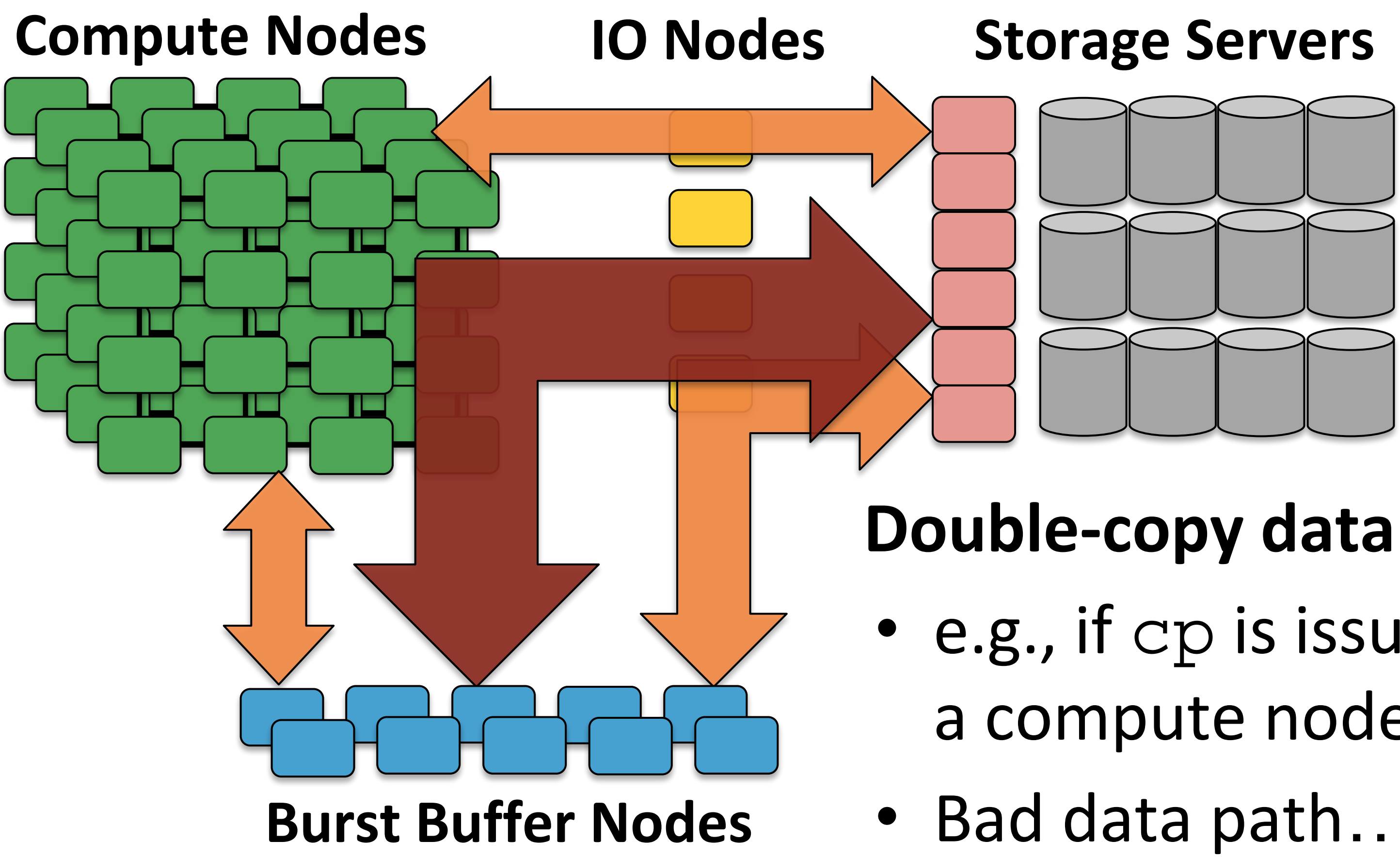


At job runtime:

- Compute nodes mount DWFS created for job
- User application interacts with DWFS via standard POSIX I/O

Slides from Glenn Lockwood, NERSC

Cori's Data Paths



Double-copy data path

- e.g., if `cp` is issued from a compute node
- Bad data path…except when #CN >> #BBNs

Slides from Glenn Lockwood, NERSC

How to use DataWarp

- **Principal 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`
- **We'll go through this in more detail later….**
- **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

Benchmark Performance on Cori

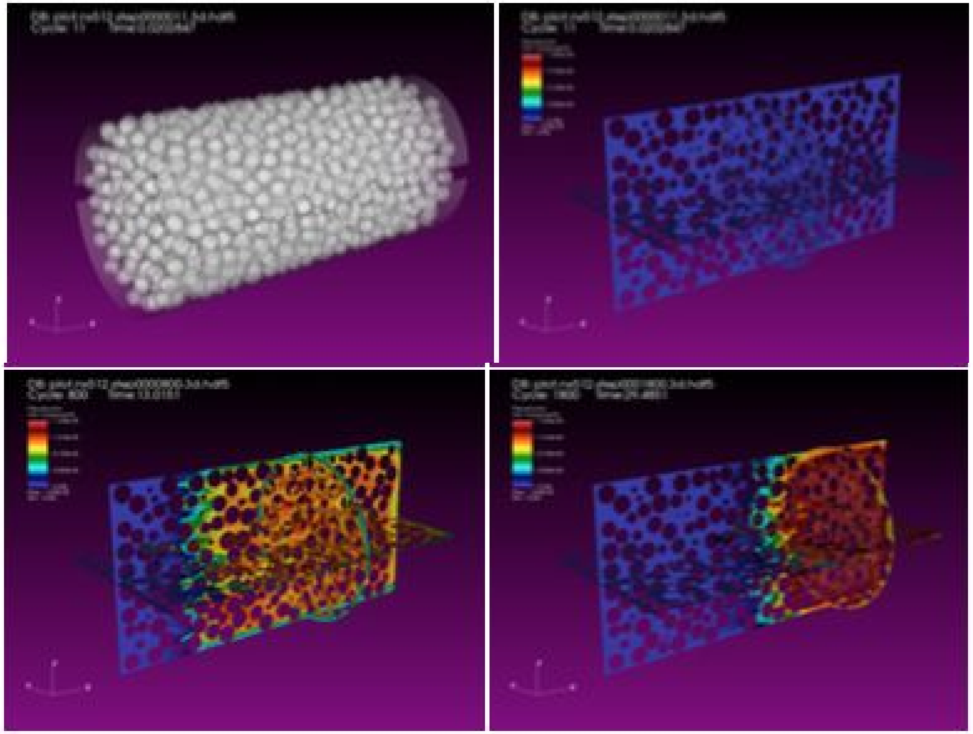
- **Burst Buffer is now doing very well against benchmark performance targets**
  - Out-performs Lustre significantly
  - (probably the) fastest IO system in the world!

	IOR Posix FPP		IOR MPIO Shared File		IOPS	
	Read	**Write**	**Read**	**Write**	**Read**	**Write**
Best Measured (287 Burst Buffer Nodes : 11120 Compute Nodes; 4 ranks/node)*	1.7 TB/s	1.6 TB/s	1.3 TB/s	1.4 TB/s	28M	13M

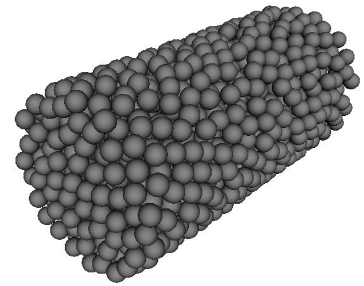
**Bandwidth tests: 8 GB block-size 1MB transfers IOPS tests: 1M blocks 4k transfer*

Burst Buffer enables Workflow coupling and visualization

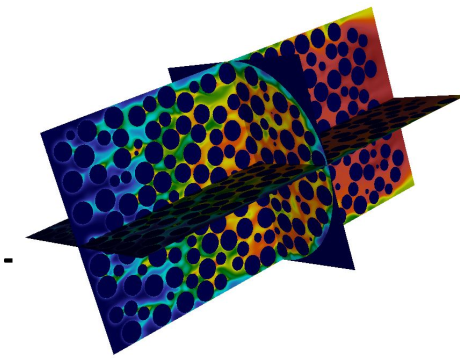
- **Success story: Burst Buffer can enable new workflows that were difficult to orchestrate using Lustre alone.**



Workflows Use Case: ChomboCrunch + VisIT



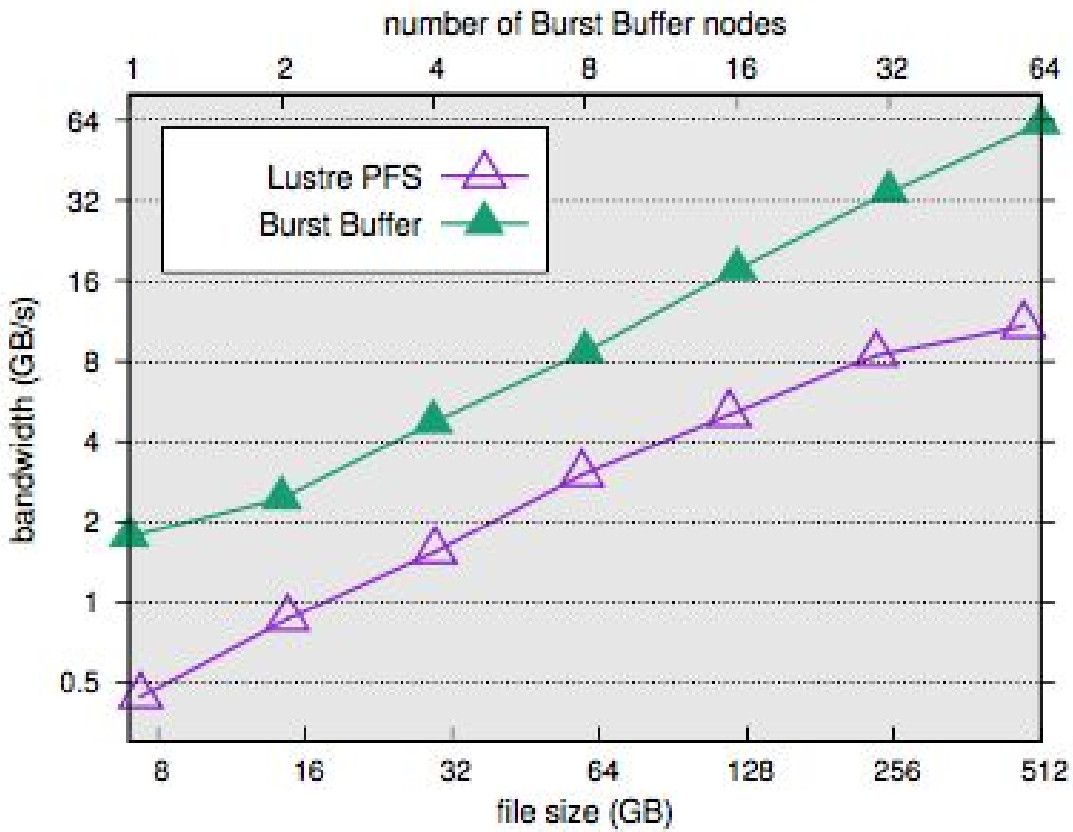
- **ChomboCrunch simulates pore-scale reactive transport processes associated with carbon sequestration**
  - –Flow of liquids through ground layers
  - –All MPI ranks write to single shared HDF5 '.plt' file.
  - –Higher resolution -> more accurate simulation - more data output (O(100TB))



- **VisIT – visualisation and analysis tool for scientific data**
  - Reads '.plt' files produces '.png' for encoding into movie
- **Before: used Lustre to *store* intermediate files.**

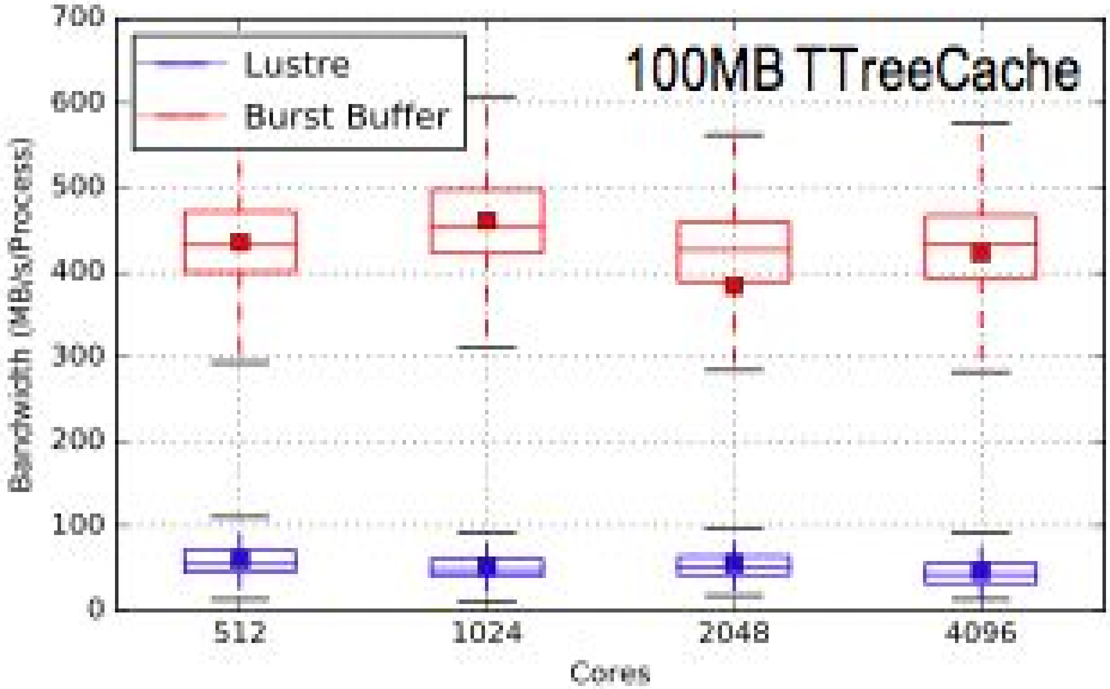
Workflows Use Case: ChomboCrunch + VisIT

- **Burst Buffer significantly out-performs Lustre for this application at all resolution levels**
  - Did not require any additional tuning!
- **Bandwidth achieved is around a quarter of peak, scales well.**



Compute node/BB node scaled: 16/1 to 1024/ 64
Lustre results used a 1MB stripe size and a stripe count of 72 OSTs

Success story: ATLAS

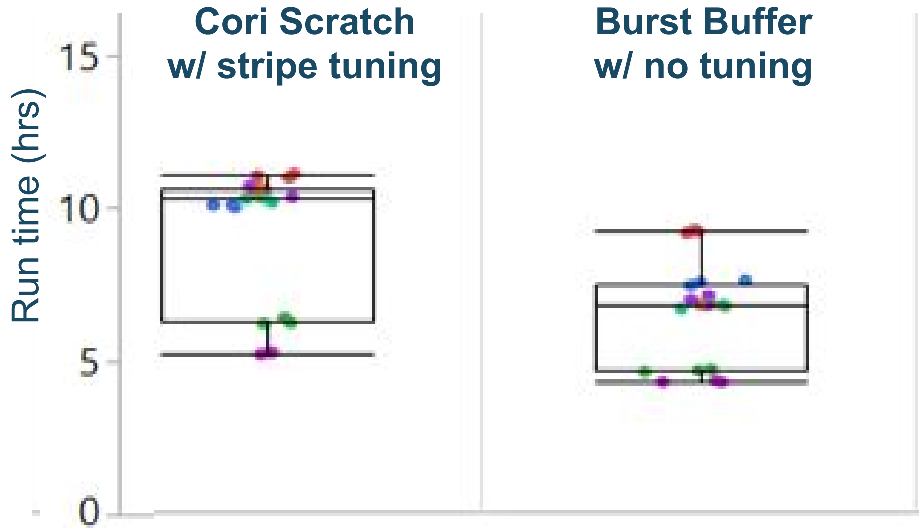


- **IOPS-heavy Data analysis**
  - Random reads from large numbers of data files
  - Used 50TB of BB space
  - ~9x faster I/O compared to Scratch.

Vakho Tsulaia, Steve Farrell, Wahid Bhimji

Success story: JGI

- **Metagenome assembly algorithm metaSPAdes**
  - Lots of small, random reads.
  - I/O is a significant bottleneck.



- **Using the Burst Buffer gains factor of 2 in I/O performance out of the box, compared to heavily tuned Lustre.**
- ***Users not part of the early user program!***

Detailed look at high IOPs: SQLite

- **A library which implements an SQL database engine**
- **No separate server process like there is in other database engines, e.g. MySQL, PostgreSQL, Oracle**
- **Database is stored in a single cross-platform file**
- **Installed on many supercomputers**
- ***"SQLite does not compete with client/server databases. SQLite competes with fopen()"* (https://sqlite.org/whentouse.html)**

SQLite benchmark

- **Inserts 2500 records into an SQLite database**
- **Written in C and optionally parallelized with MPI**
  - In parallel runs each MPI rank writes 2500 records to its own uniquely named database file

```
INSERT INTO 'pts1' ('I', 'DT', 'F1', 'F2')
VALUES ('1', CURRENT_TIMESTAMP, '6758',
'9844343722998287');
```

- **Anatomy of insert transaction:**
  - Dozens of I/O system calls are required for each SQLite transaction

System call	Count
fdatasync	4
read	2
write	10
lseek	12
fcntl	9
open	2
close	2
unlink	1
fstat	5
stat	2
access	2

Many I/O ops for 1 DB insert!

1. Create a new journal file (fd=5)

```
open("./benchmark-0.db-journal",\
  O_RDWR|O_CREAT|O_CLOEXEC,0644) = 5
```

2. Write data to the journal file: many small accesses, e.g. 4b, 512b, 1KiB (only a subset shown). Sync. to save journal file to storage

```
write(5, ""..., 512) = 512
lseek(5, 512, SEEK_SET) = 512
write(5, "", 4) = 4
lseek(5, 516, SEEK_SET) = 516
write(5, "", 1024) = 1024
fdatasync(5) = 0
```

3. Sync. parent directory to ensure file system has created the directory entry for the journal file

```
open(".", O_RDONLY|O_CLOEXEC) = 6
fdatasync(6) = 0
close(6) = 0
```

4. Write the number of records to the journal file and sync. to save journal file to storage

```
lseek(5, 0, SEEK_SET) = 0
write(5, "", 12) = 12
fdatasync(5) = 0
```

5. Write data to the database. Sync. to save database to storage

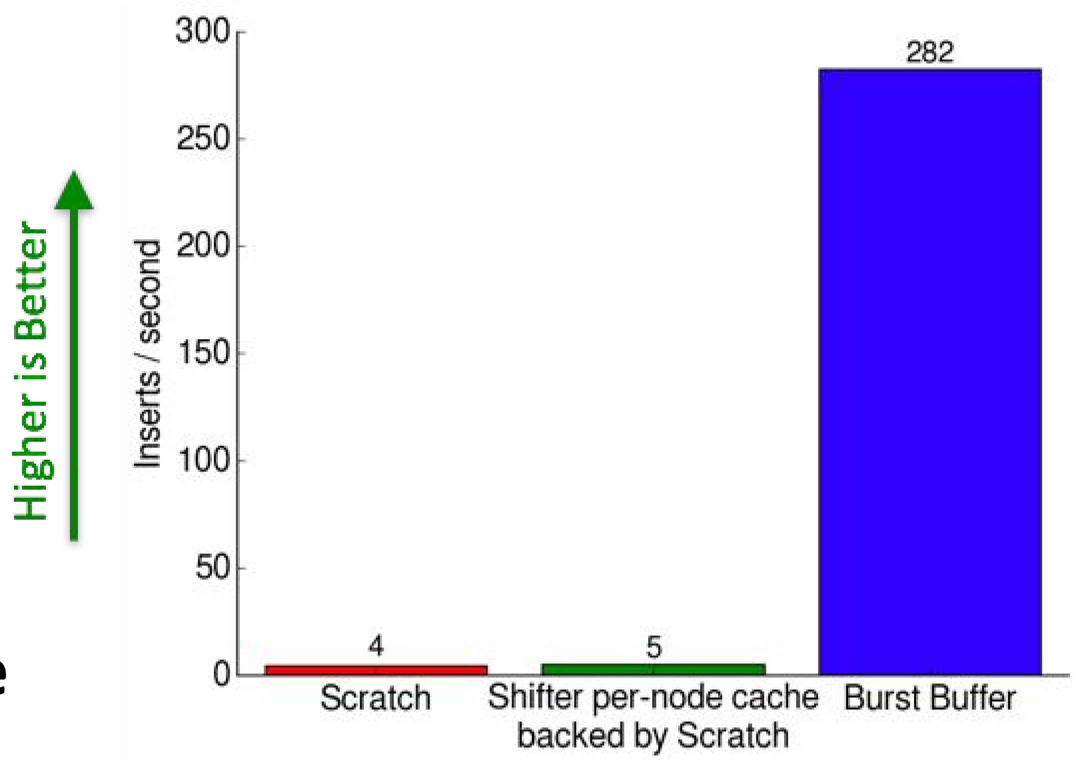
```
lseek(4, 0, SEEK_SET) = 0
write(4, "", 1024) = 1024
lseek(4, 1024, SEEK_SET) = 1024
write(4, ""..., 1024) = 1024
fdatasync(4) = 0
```

6. Close and delete the journal file

```
close(5) = 0
unlink("./benchmark-0.db-journal") = 0
```

~50x faster on the BB!

- **Benchmark run with 1 MPI rank**
- **Scratch configuration uses 1 OST**
- **Burst Buffer configuration uses 1 granule of storage**



Frequent synchs perform badly on Lustre

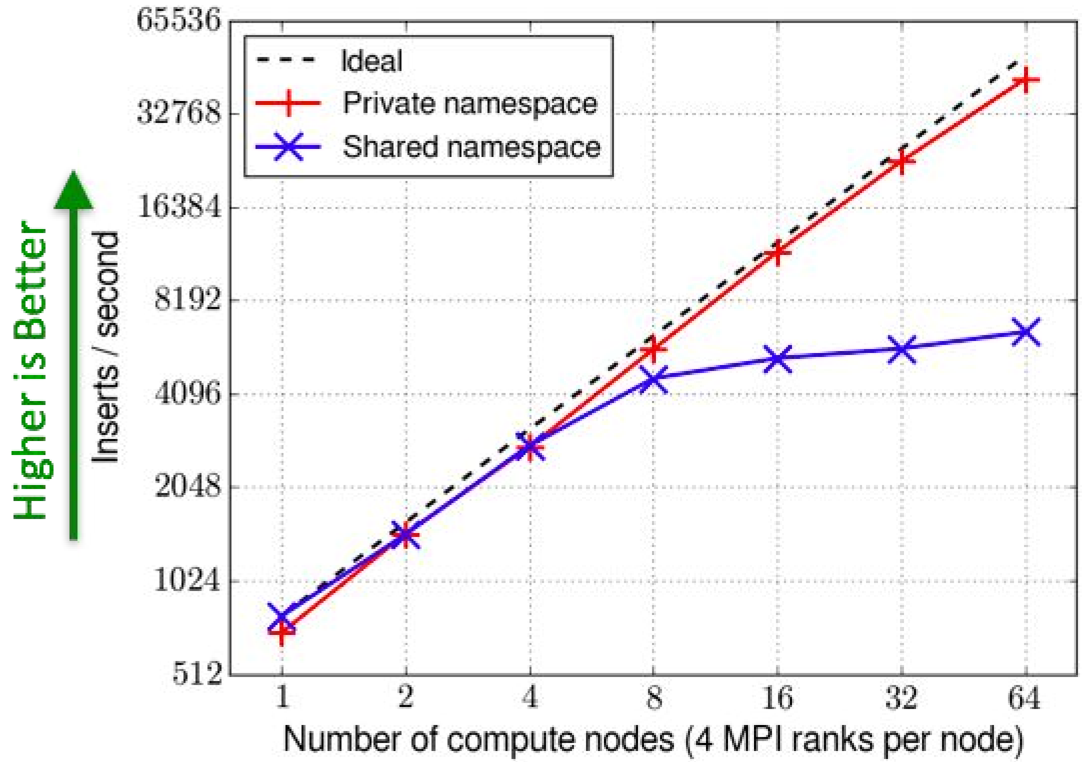
- **98% of wall time!**
- **1 synchronization every 2.5 writes gives no opportunity for the kernel to buffer the writes**

#	[time]	[count]	<%wall>
# fdatasync	**577.59**	**10004**	**97.83**
# unlink	2.39	2511	0.41
# write	**2.09**	**25036**	**0.35**
# __fxstat64	1.33	10016	0.22
# open64	1.32	5004	0.22
# close	1.29	5004	0.22
# __xstat64	0.93	10008	0.16
# read	0.06	5003	0.01
# lseek64	0.02	30038	0.00
# __lxstat64	0.00	1	0.00
# fflush	0.00	1	0.00

- ***The data transfer is limited by the write latency of spinning disk***

MD performance scales well in private mode

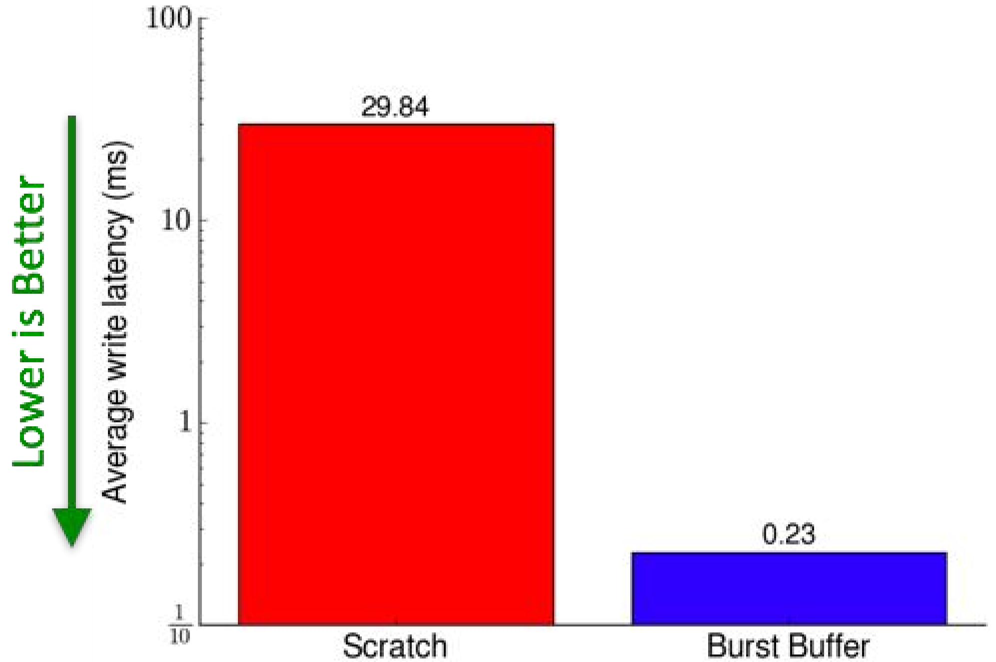
- **Private mode enables scalable metadata performance as we add compute nodes**
  – 1 metadata server per compute node



(All runs use 64 BB granules)

MD in IOR benchmark

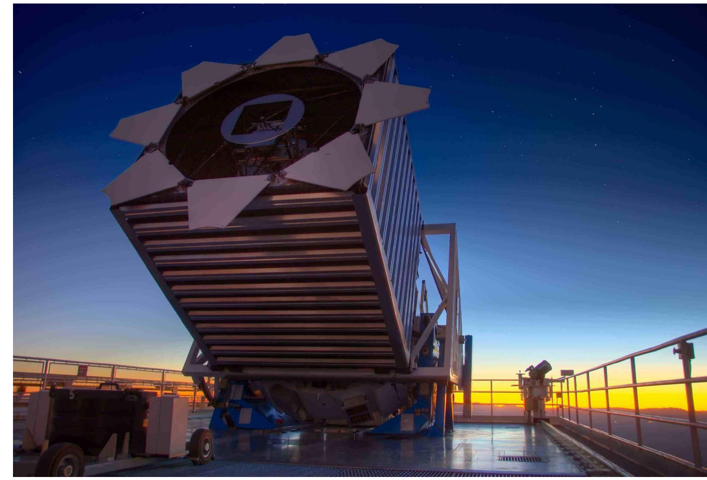
- **Single-stream IOR with a data synchronization after every POSIX write (-Y flag)**
- **Average write latency < 1 millisecond on BB**
  - two orders of magnitude faster than disk!



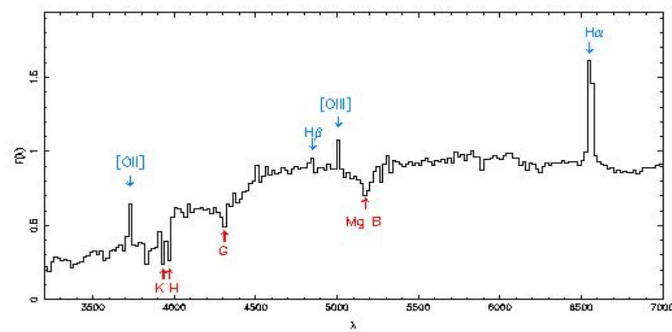
Challenging IO use case: Astronomy data

Jialin Liu and Debbie Bard

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



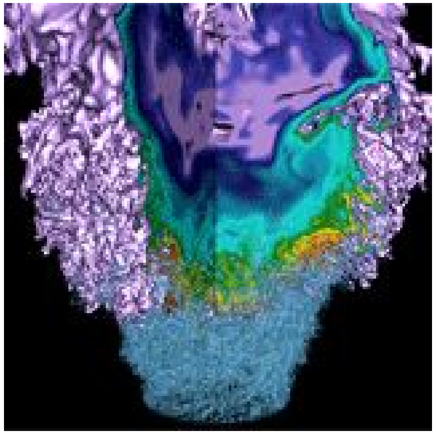
Time taken to extract 1000 random spectra	**From one hdf5 file**	**From individual fits files**
From Lustre	44.1s	160.3s
From BB	1.3s	44.0s
Speedup:	**33x**	**3.6x**



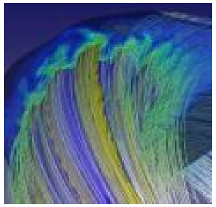
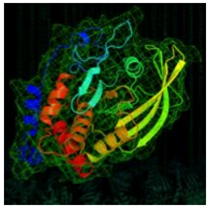
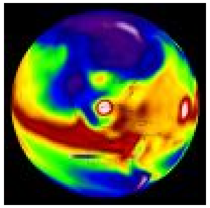
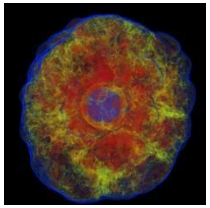
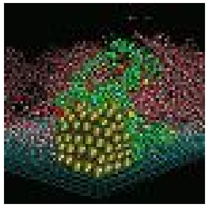
Summary

- **NERSC has the first Burst Buffer for open science in the USA**
- **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**
- **Performance for metadata-intensive operations is particularly excellent**

Extra slides



NERSC

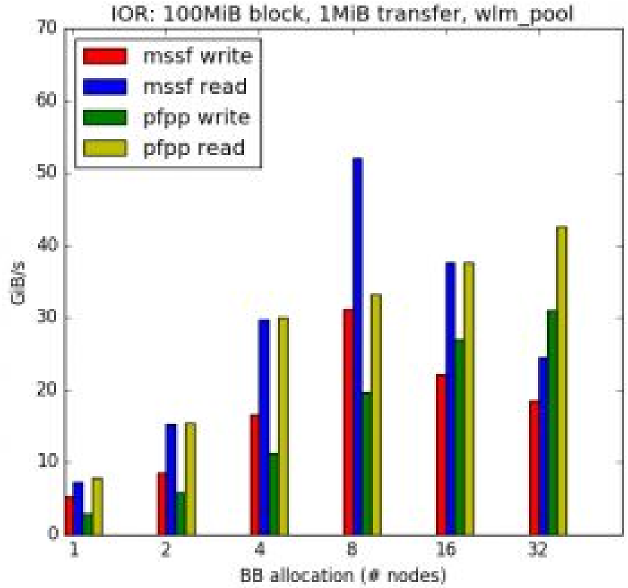






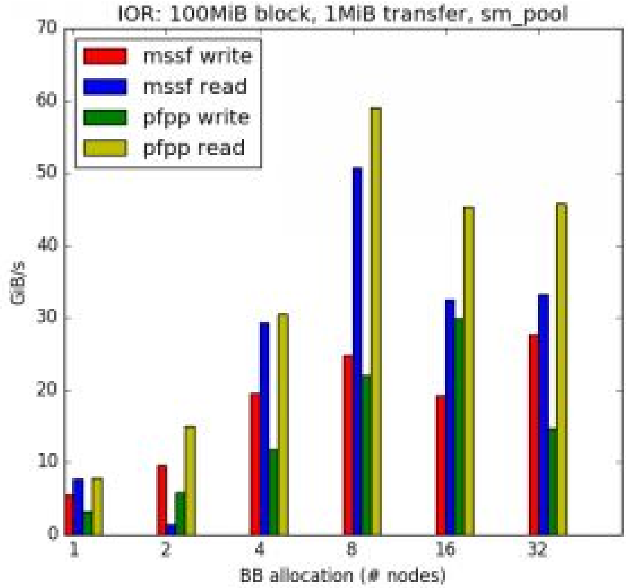
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Performance tips

- **Stripe your files across multiple BB servers**
  - To obtain good scaling, need to drive IO with sufficient compute - scale up # BB nodes with # compute nodes





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

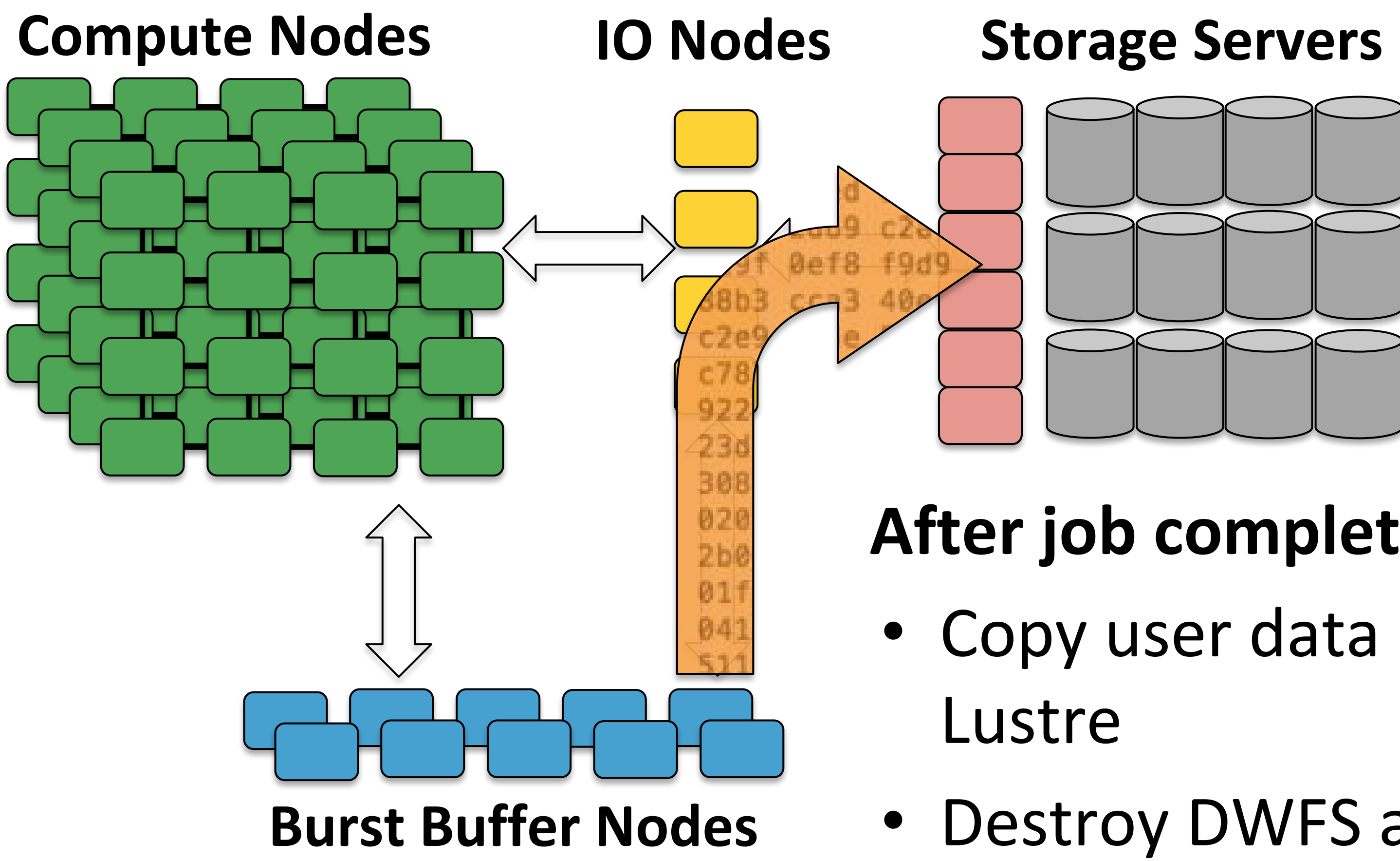
SSD write protection

- **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)

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Cori's Data Paths



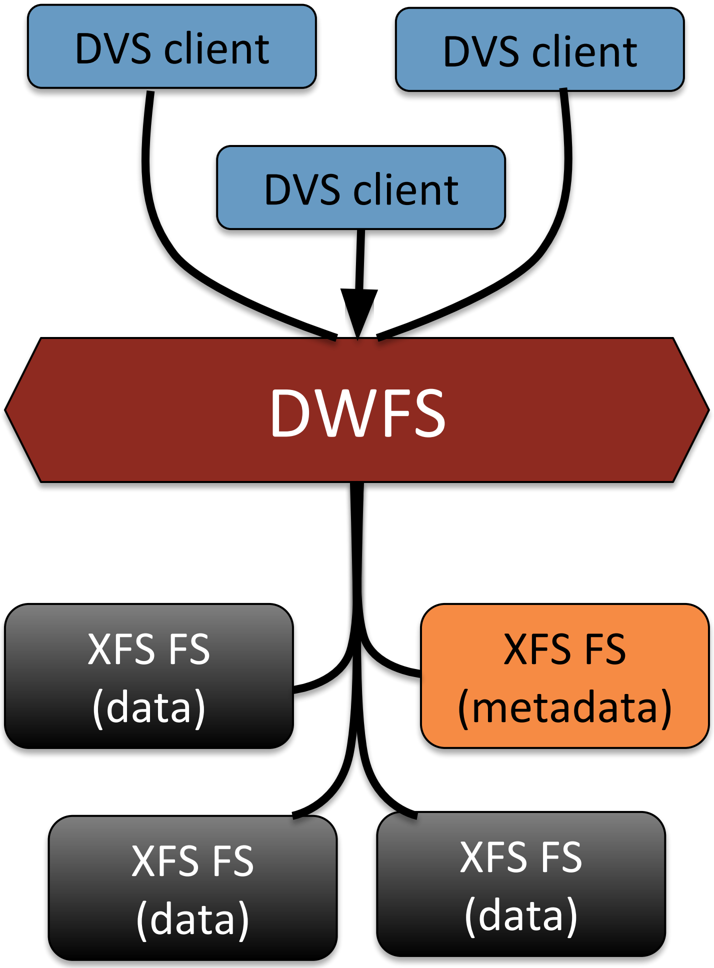
After job completes:

- Copy user data back to Lustre
- Destroy DWFS associated with job

Slides from Glenn Lockwood, NERSC

DataWarp File System (DWFS)

- **File system built on Wrapfs that glues together**
  - Cray DVS for client-server RPCs
  - many XFS file systems for data (called "fragments")
  - one XFS file system for metadata
- **Conceptually very simple**
  - No DLM
    - rely on server-side VFS file locking
    - no client-side page cache (yet)
  - Data placement determined by deterministic hash of inode, offset
  - Stubbed XFS file system encodes most file metadata

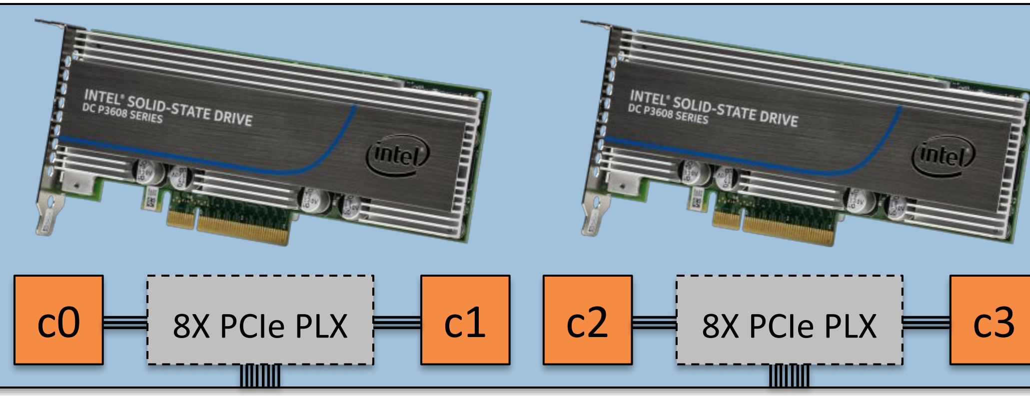


Slides from Glenn Lockwood, NERSC

DWFS Storage Substrate

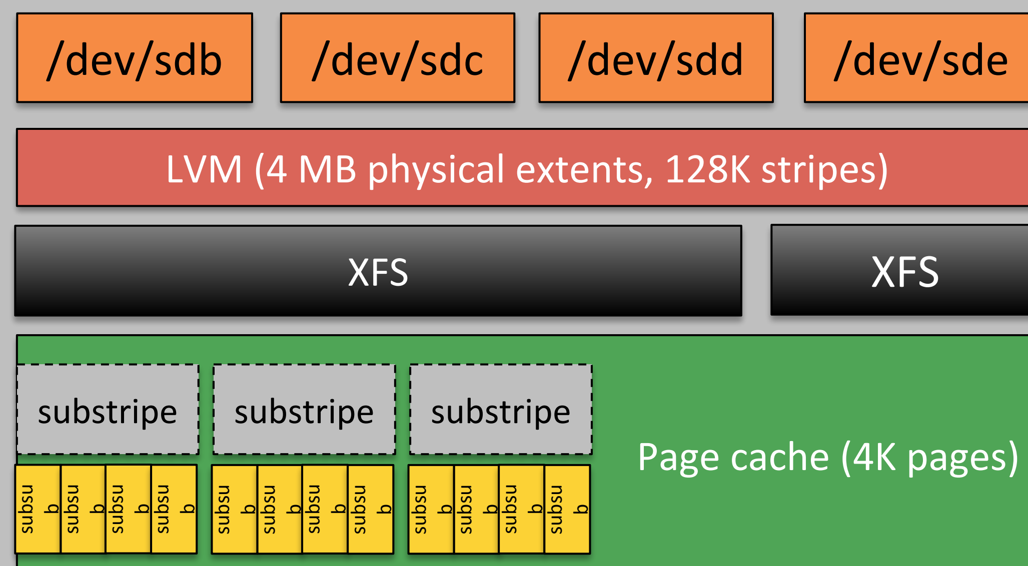
Physical Node

- 1x Sandy Bridge E5, 8-core
- 64 GB DDR3
- 2x Intel P3608 (3.2 TB ea.)
- 4x Intel P3600 controllers



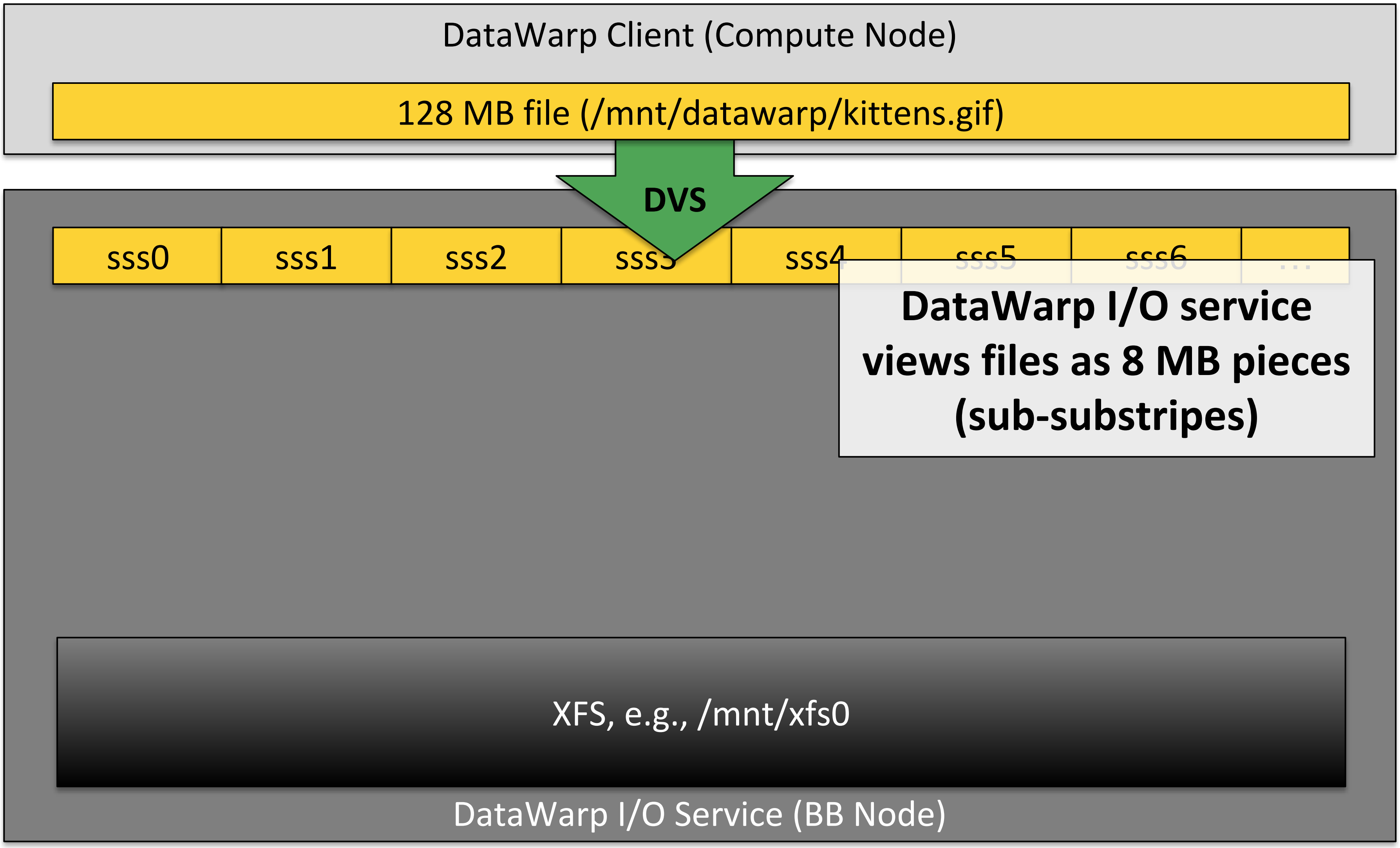
Linux OS

- Logically four block devices
- LVM aggregates block devices
- Linux vol group and XFS fs
- 3 substripes per file per BB node
- 8 MB sub-substripes in substripe

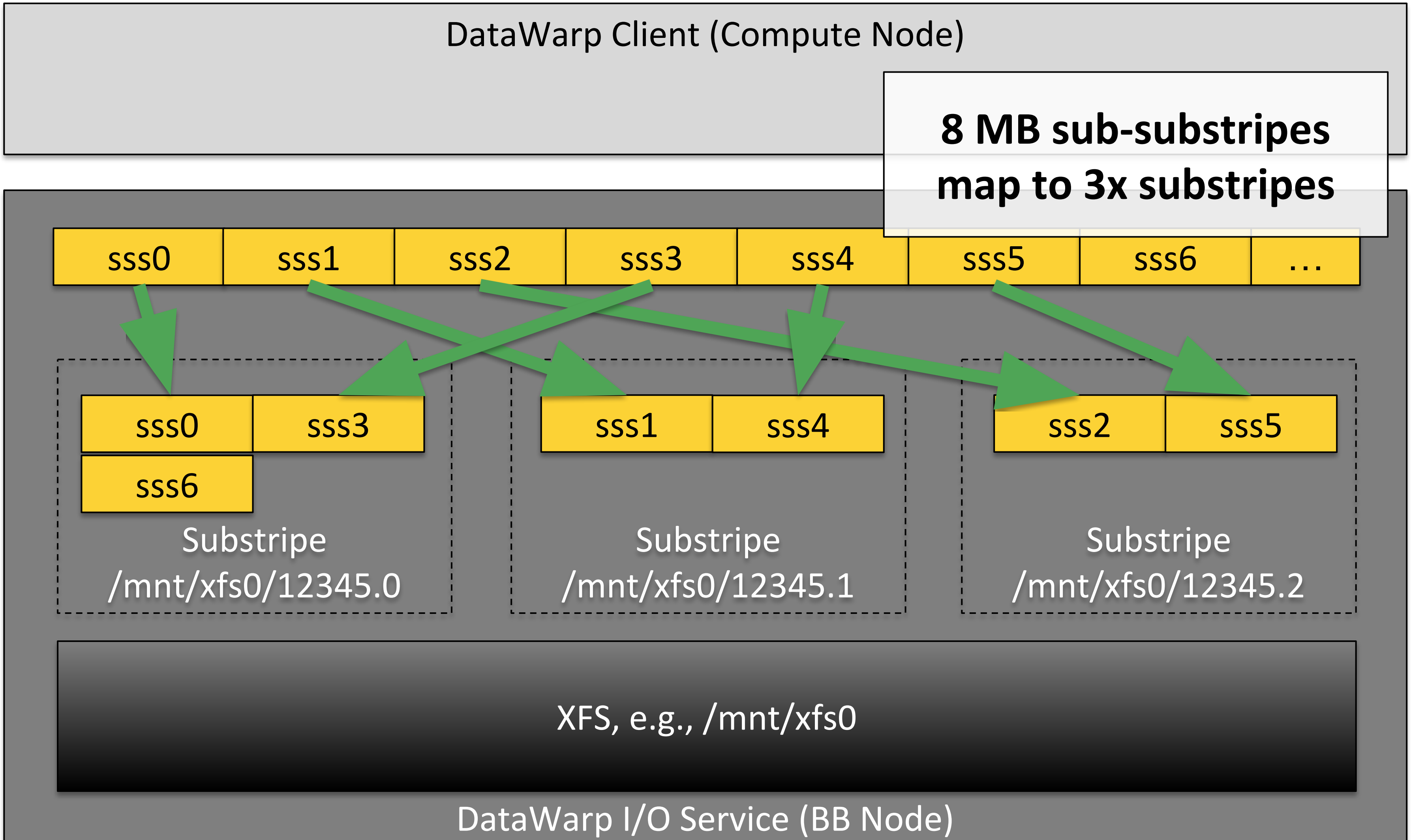


Slides from Glenn Lockwood, NERSC

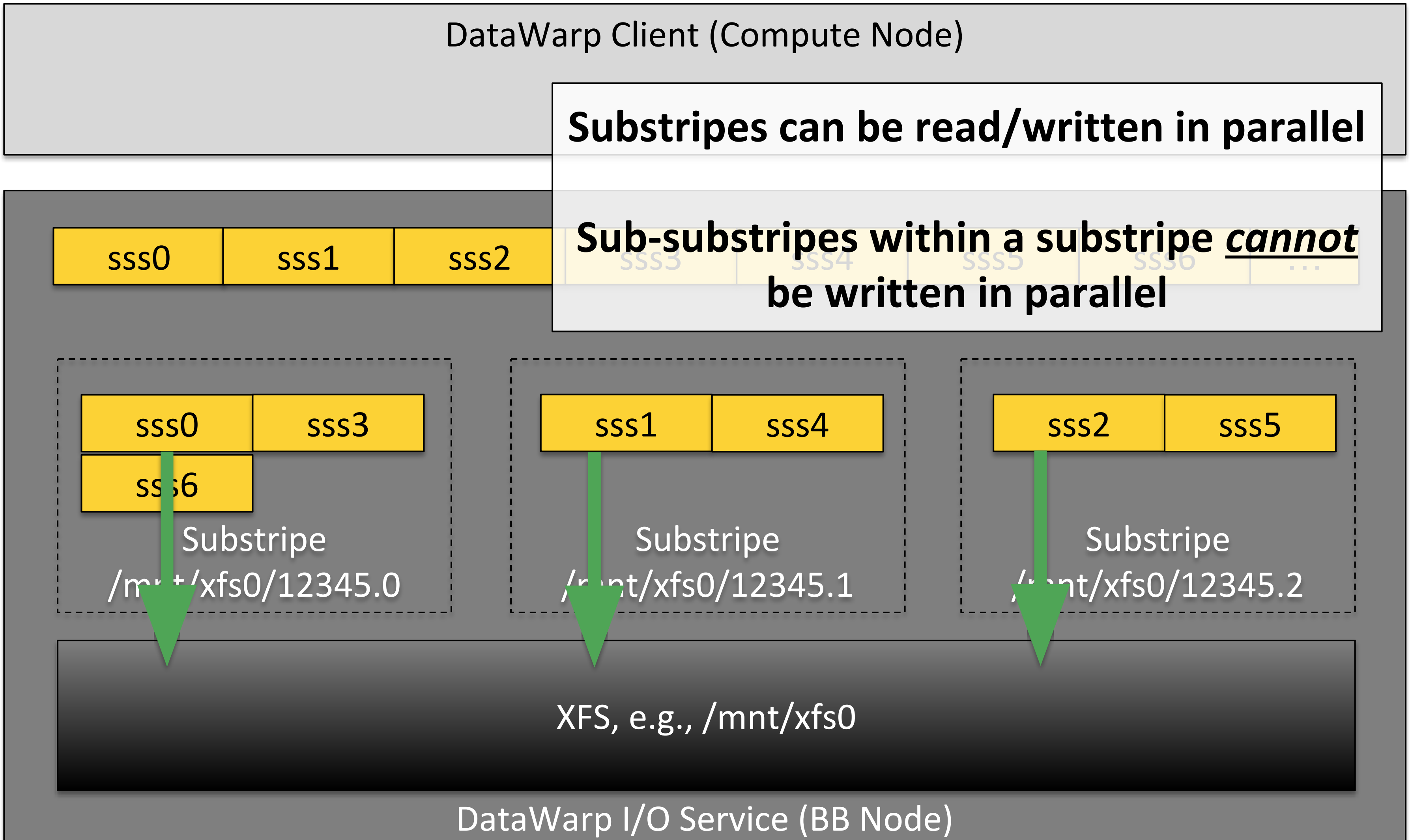
Data Layout: Simple Case (1 BB node)



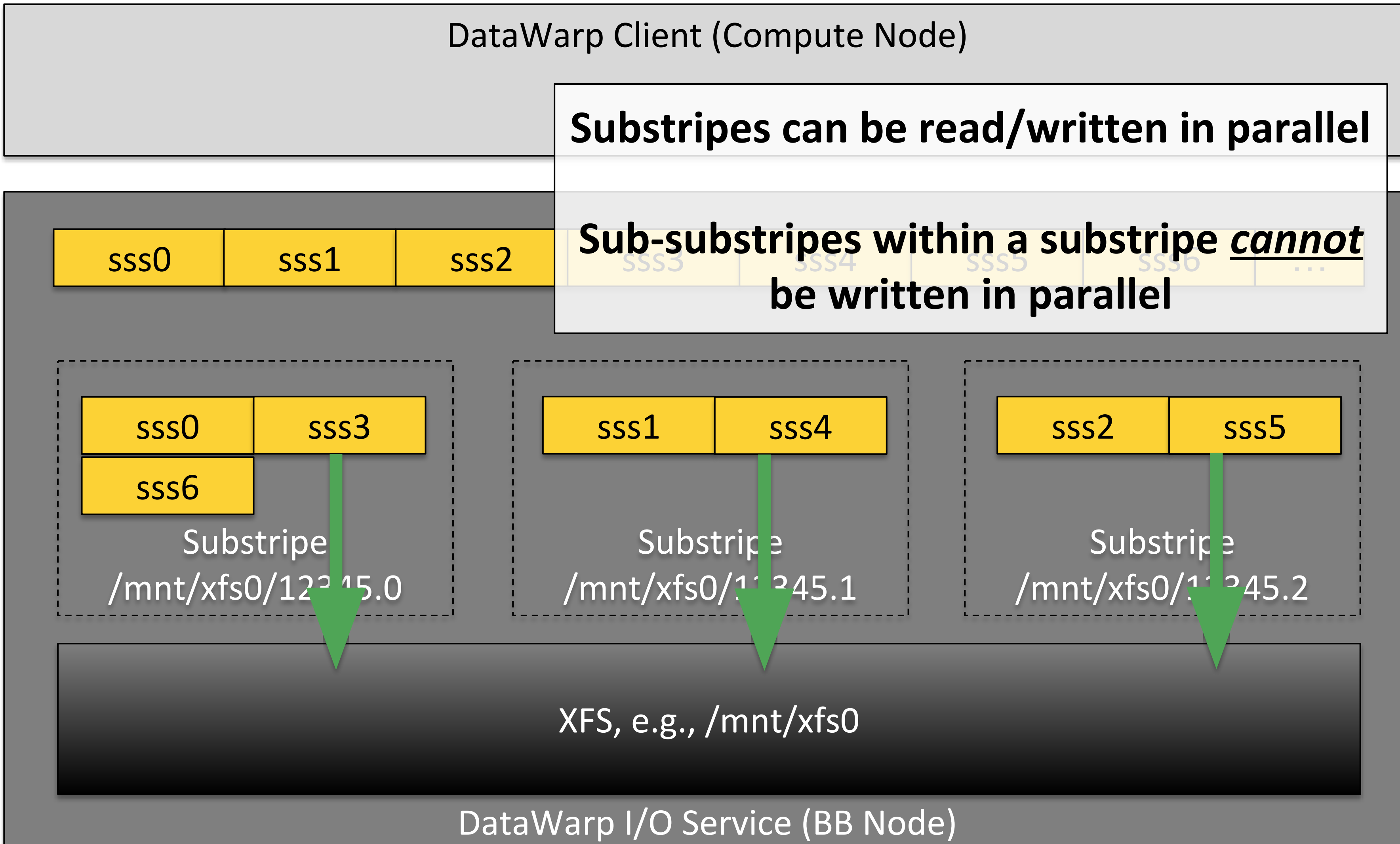
Data Layout: Simple Case (1 BB node)



Data Layout: Simple Case (1 BB node)

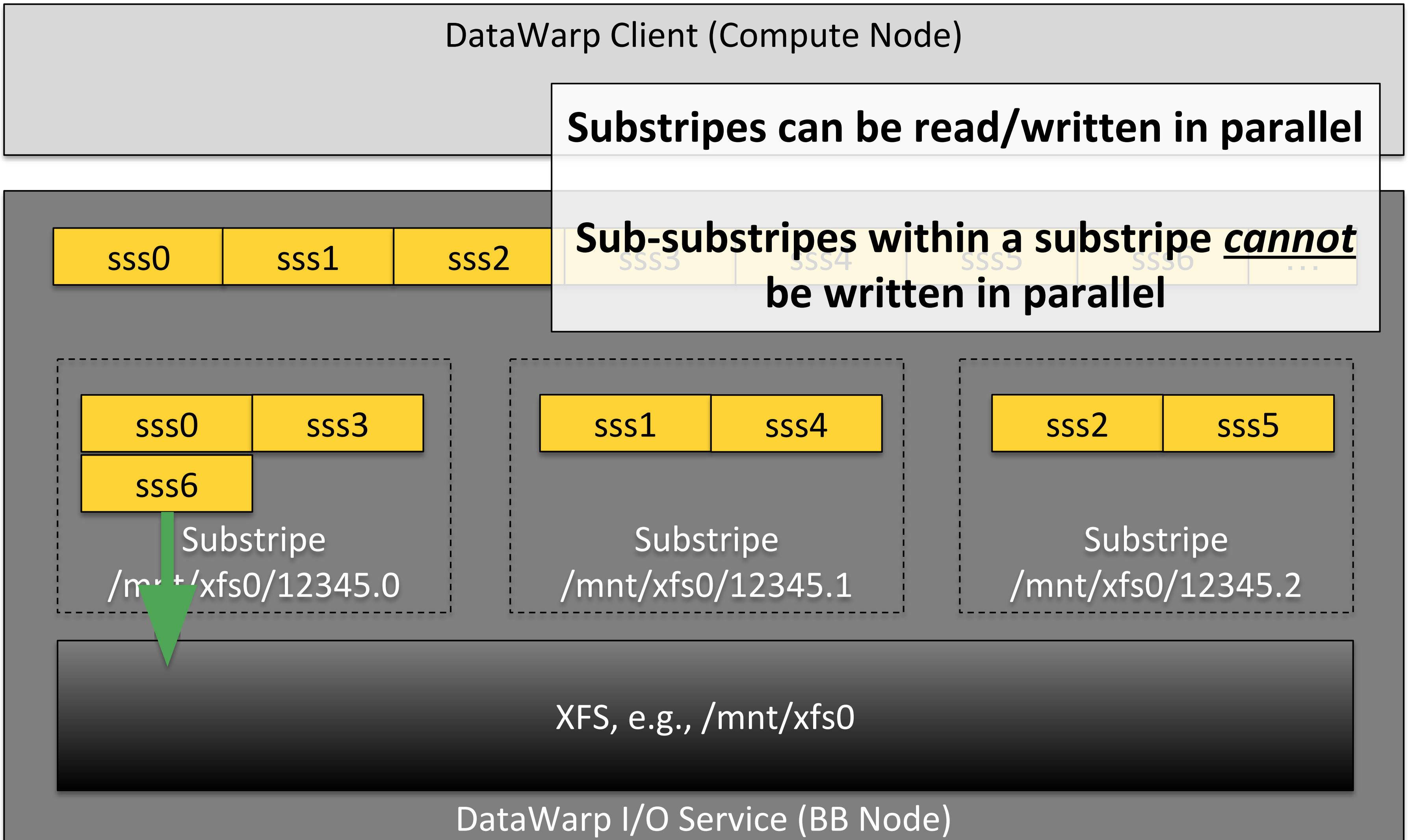


Data Layout: Simple Case (1 BB node)

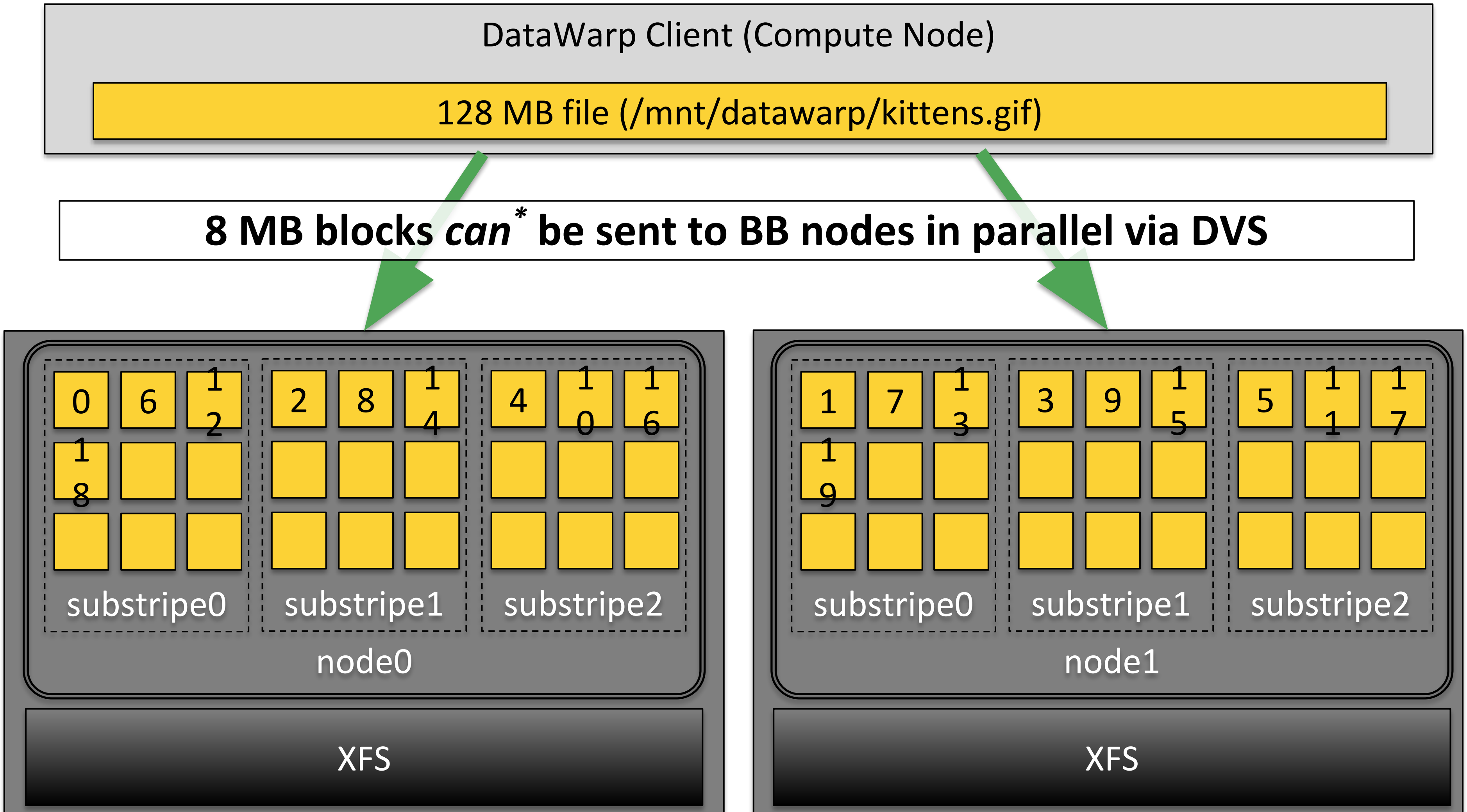


Slides from Glenn Lockwood, NERSC

Data Layout: Simple Case (1 BB node)



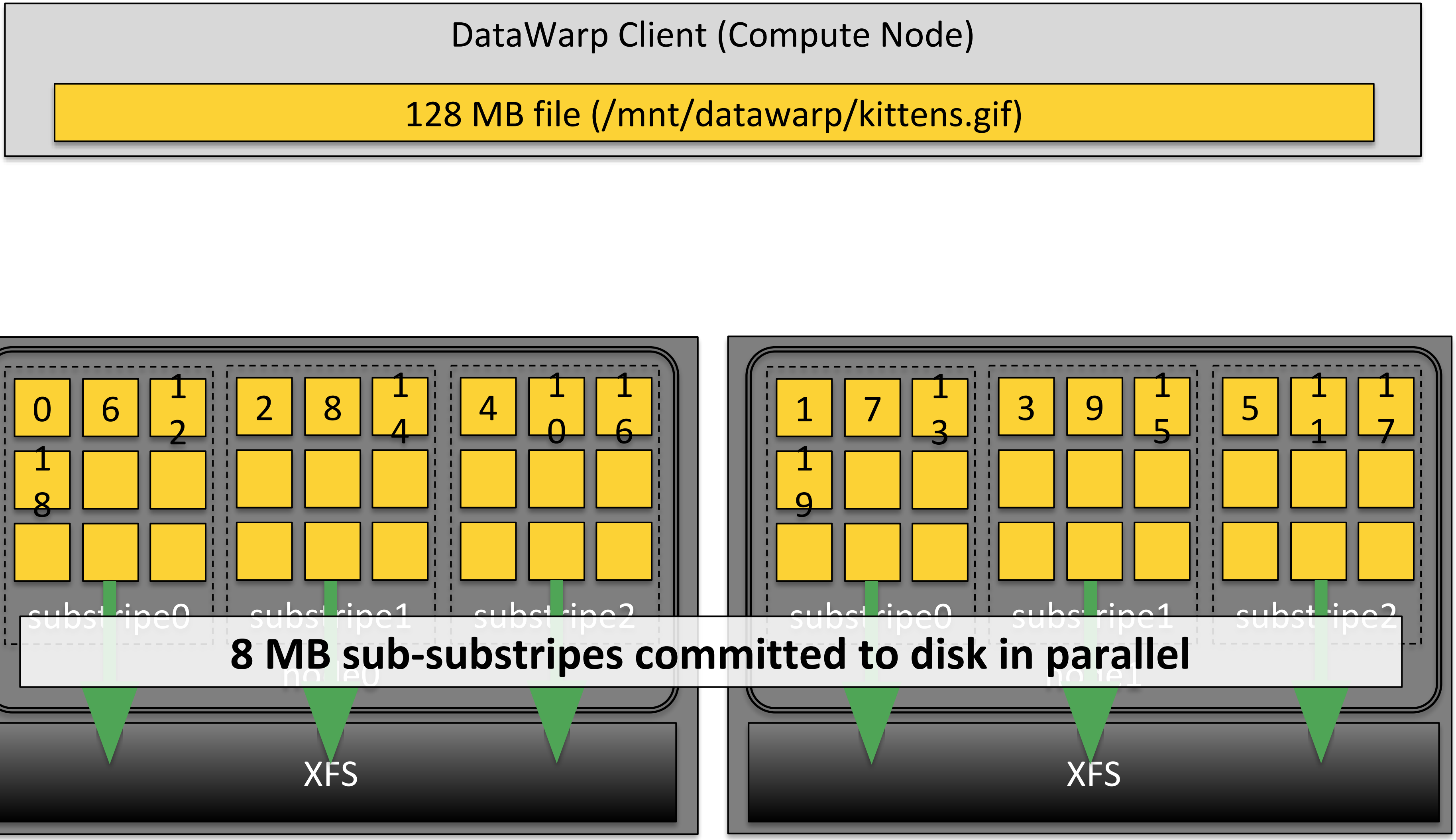
Data Layout: 2 BB nodes



* under certain conditions

Slides from Glenn Lockwood, NERSC

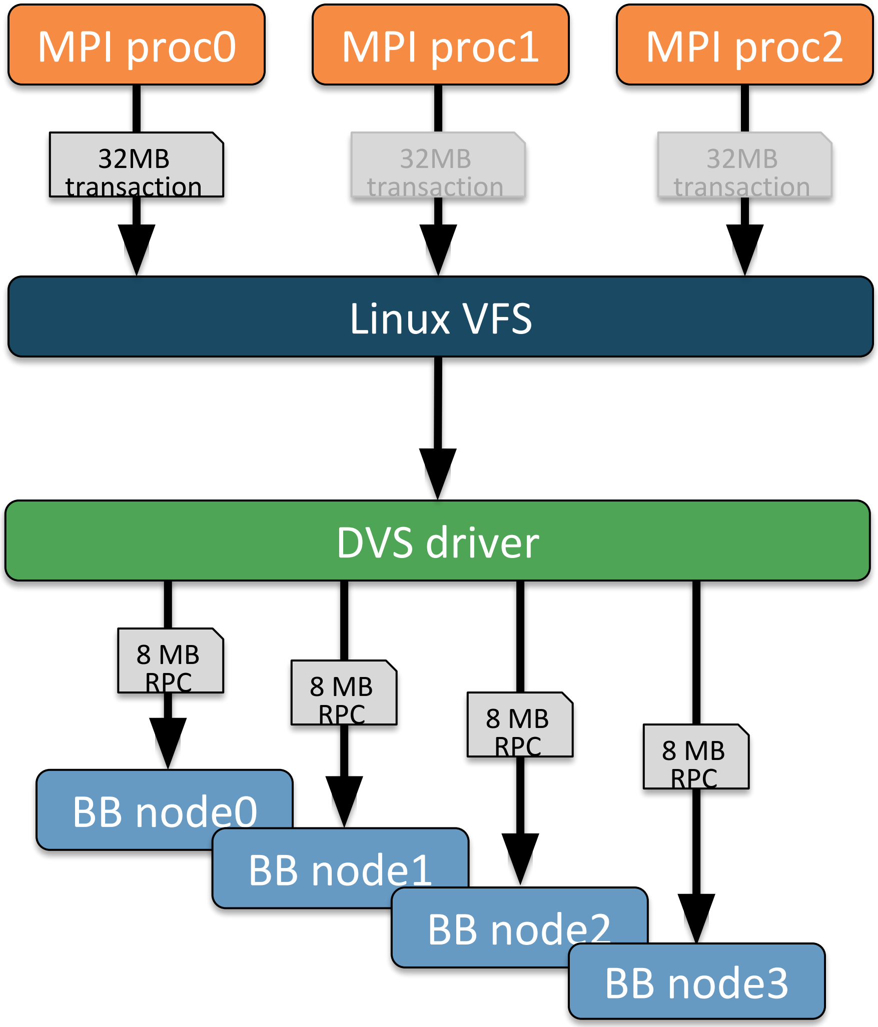
Data Layout: 2 BB nodes



Slides from Glenn Lockwood, NERSC

DWFS Data Path - Client

- **No page cache for write-back**
- **Shared-file writes are serialized by VFS**
- **DVS can parallelize very large transactions**



Slides from Glenn Lockwood, NERSC

Hierarchical Parallelism

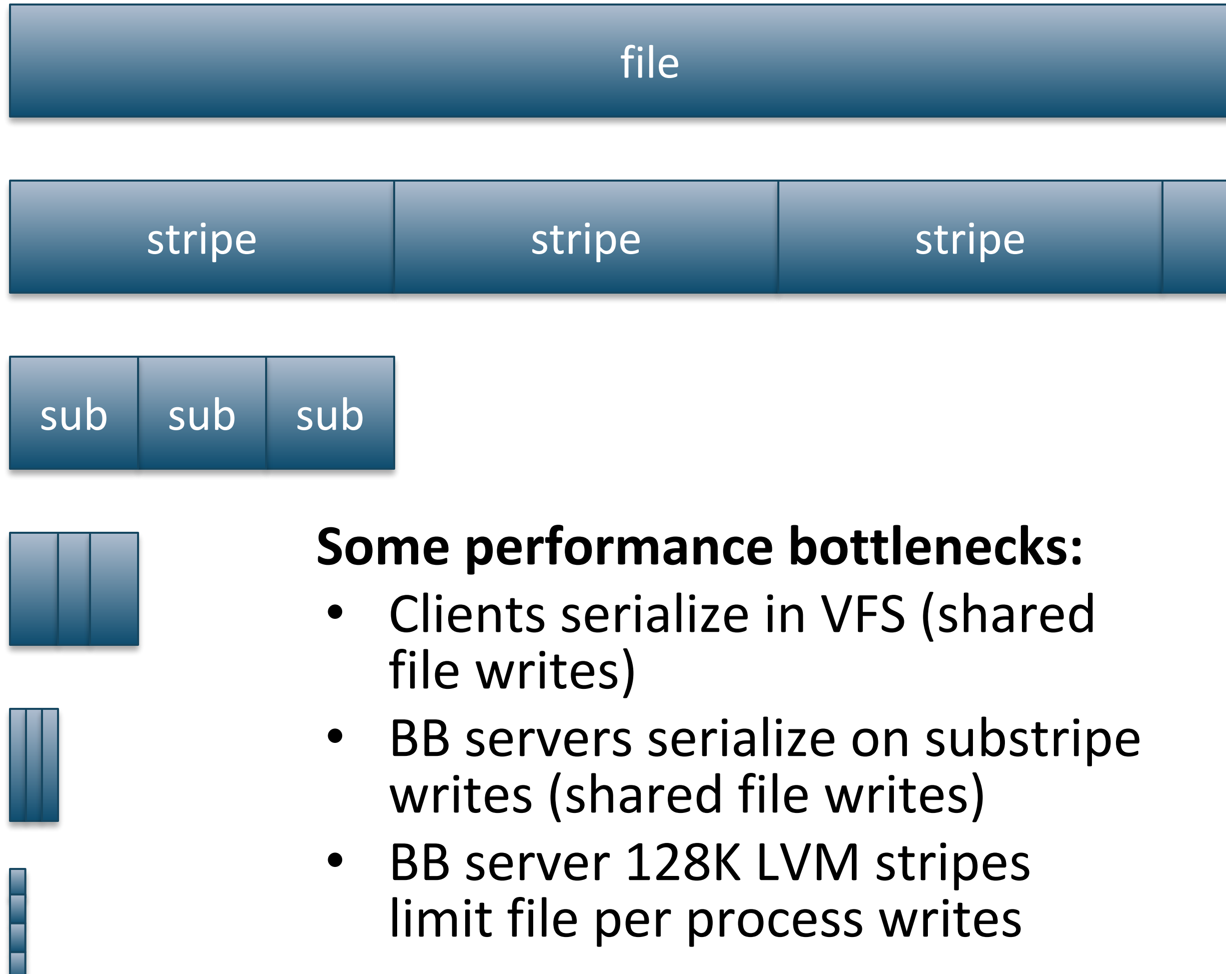
Stripes
8MB across N servers

Substripes
8MB across
3-14 substripes

XFS AGs
variable size
across 4 AGs

LVM PEs
4MB across 4 devices

LVM stripes
128K across 4 PEs



Some performance bottlenecks:

- Clients serialize in VFS (shared file writes)
- BB servers serialize on substripe writes (shared file writes)
- BB server 128K LVM stripes limit file per process writes

Slides from Glenn Lockwood, NERSC