I/O Topologies

ATPESC 2018

Jialin Liu

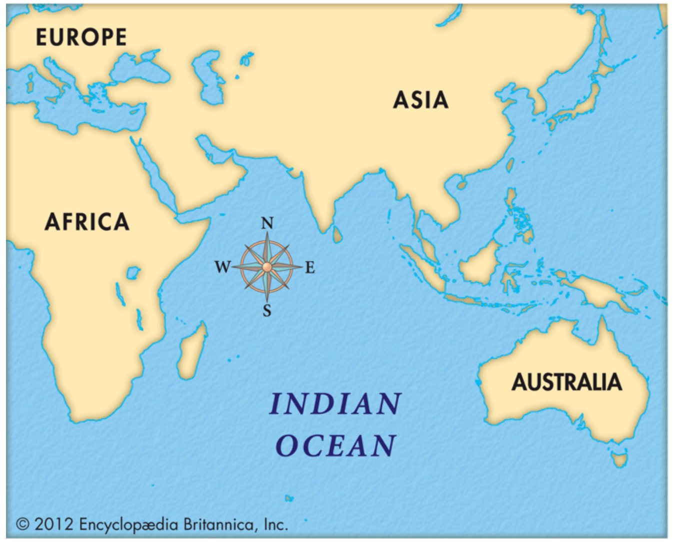
National Energy Research Scientific Computing Center

Q Center, St. Charles, IL (USA)
July 29 – August 10, 2018

Outline

- Common IO Issues in HPC I/O Stack
- I/O Performance v.s. I/O Productivity
- Burst Buffer v.s. Lustre (on HDD)

IO



- github.io – Rank: 521
- codepen.io – Rank: 3271
- soup.io – Rank: 3910
- filecloud.io – Rank: 7022
- intercom.io – Rank: 14449
- binbox.io – Rank: 15369
- laravel.io – Rank: 16374
- redis.io – Rank: 16418
- pen.io – Rank: 17889
- put.io – Rank: 21159
- icomoon.io – Rank: 21373
- imm.io – Rank: 23274
- purecss.io – Rank: 23484
- media.io – Rank: 27293
- ubiquity.io – Rank: 27481
- emmet.io – Rank: 28681
- galleria.io – Rank: 30307
- c9.io – Rank: 30347
- torquemag.io – Rank: 33209
- gamechanger.io – Rank: 34510
- plan.io – Rank: 37505
- brackets.io – Rank: 37508
- filepicker.io – Rank: 39491
- kraken.io – Rank: 41207
- sidebar.io – Rank: 42285
- dashboard.io – Rank: 46205

Common I/O Issues

1. Bandwidth
   - ➢ *"The peak bandwidth is 700 GB/s, why I only got 7 MB/sec?"*
   - ➢ *"Can you tell me how many OSTs should I use?"*
2. Metadata
   - ➢ *"ls is too slow"*
3. KNL v.s. Haswell
   - ➢ *"I have used more IO processes on KNL, why the performance is still bad"*
4. Pain of Productivity
   - ➢ *"I like to use Python/Spark/Tensorflow, but how can I load in the HDF5 data"*

Complex HPC I/O Stack

Productive Interface builds a thin layer on top of existing high performance I/O library for productive big data analytics

Python, Spark, TensorFlow

High Level I/O Libraries map application abstractions onto storage abstractions and provide data portability.

HDF5, Parallel netCDF, ADIOS

Parallel file system maintains logical file model and provides efficient access to data.

PVFS, PanFS, GPFS, Lustre

Application

Productive Interface

High-Level I/O Library

I/O Middleware

I/O Forwarding

Parallel File System

I/O Hardware

I/O Middleware organizes accesses from many processes, especially those using collective I/O.

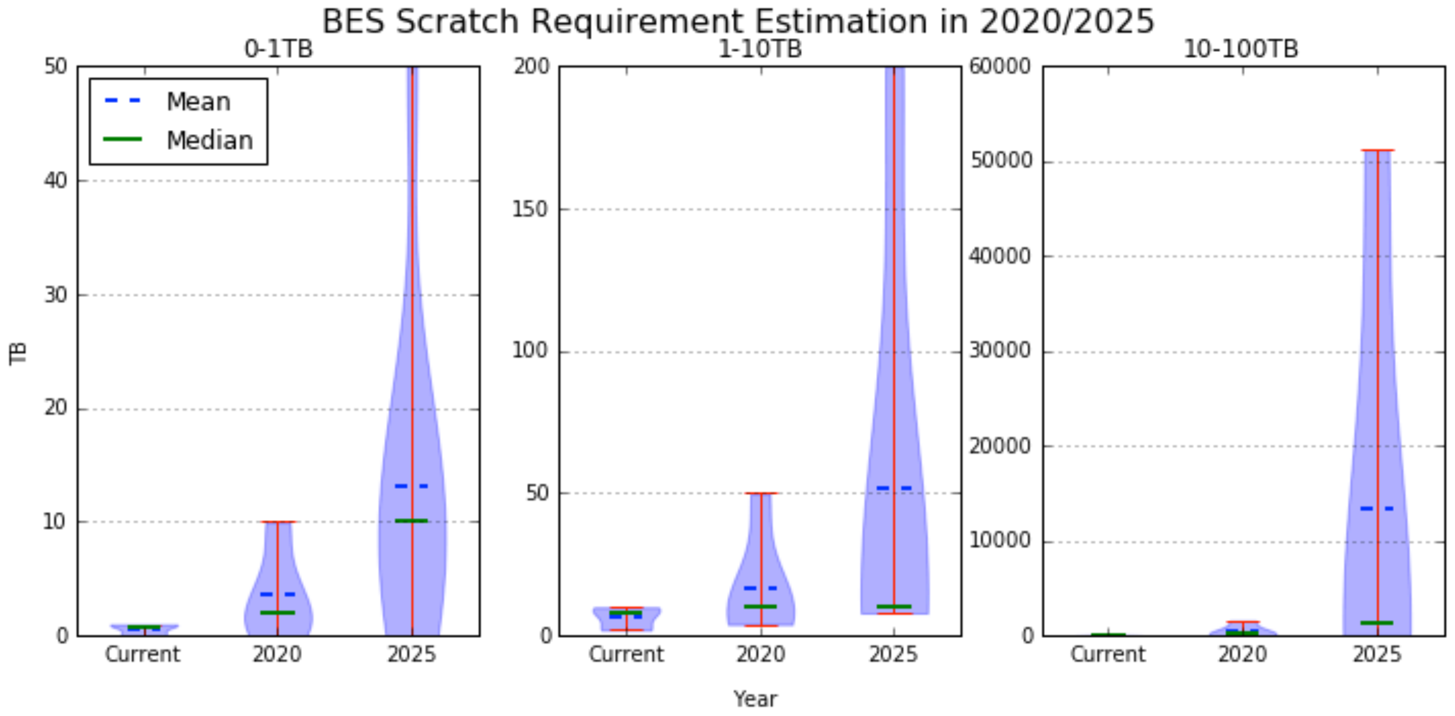
MPI-IO, GLEAN, PLFS

I/O Forwarding transforms I/O from many clients into fewer, larger request; reduces lock contention; and bridges between the HPC system and external storage. *IBM ciod, IOFSL, Cray DVS, Cray Datawarp*

Based on Philip Carns and Rob Ross' slide

I/O Challenges in 2020/2025

- Scientific applications/simulations generate massive quantities of data.
  - Example, BES: Basic Energy Science, Requirement Review, 2015
  - 19 projects review
  - Example projects: Quantum Materials, Soft Matters, Combustion

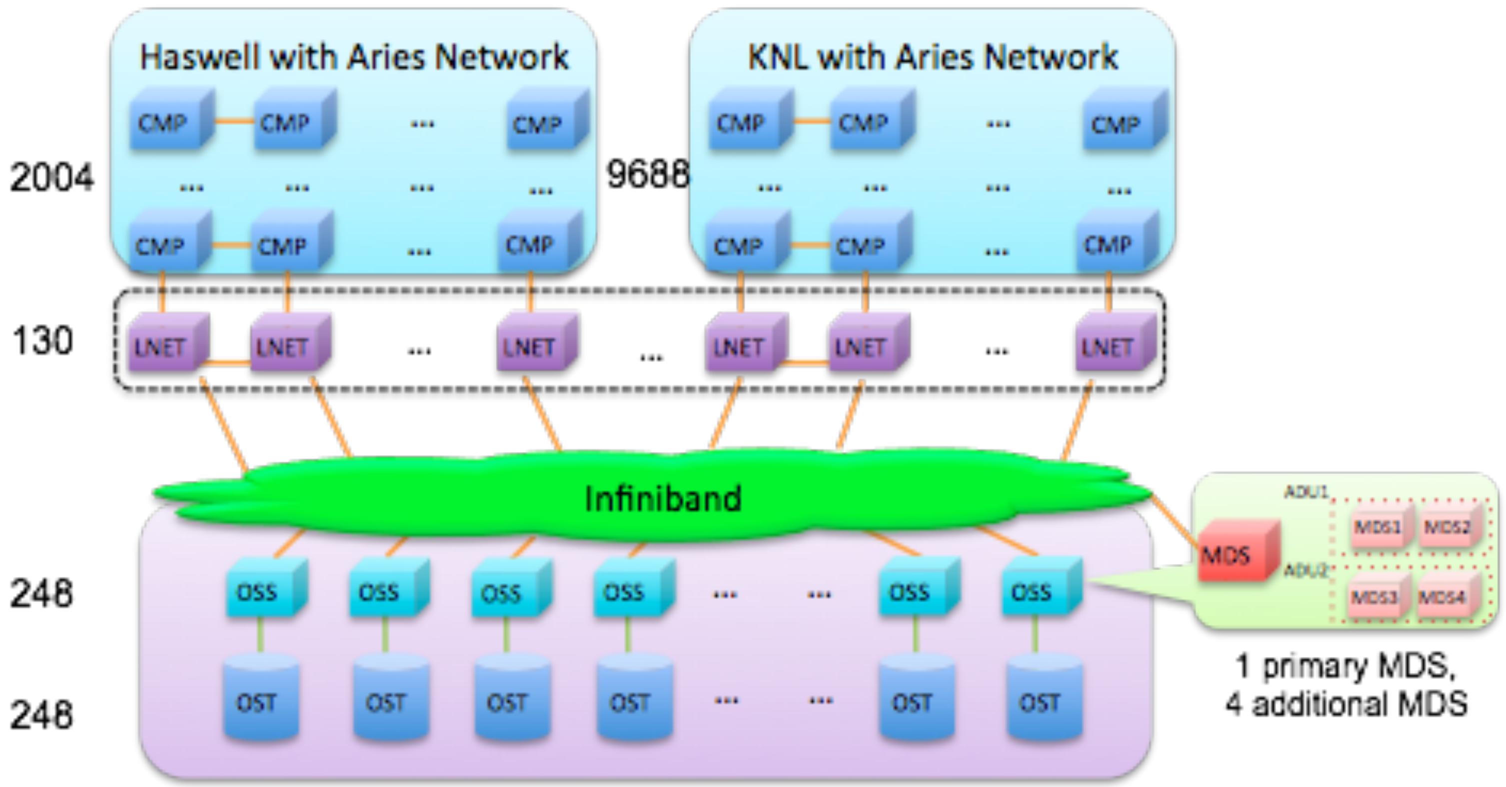


Current (TB)	2020/Current	2025/Current
0-1	7.5	15
1-10	2	4
10-100	3.5	22.5

Average Increasing Ratio

Storage 2020 White Paper
http://www.nersc.gov/news-publications/nersc-news/nersc-center-news/2017/new-storage-2020-report-outlines-future-hpc-storage-vision/

Parallel File System: Cori Scratch



- Edison and Cori
- 7000+ users across the world
- Data floods in every second

Now

- *“The peak bandwidth is 700 GB/s, why I only got 7 MB/sec?”*
- *“Can you tell me how many OSTs should I use?”*

Issue 1

“I noticed that the reading speed of these data is only about 500MB/s. I use MPIIO and collective buffering. And the speed is similar for runs with 4 Haswell nodes and 64 Haswell nodes.”

what is the proper stripe count and size for 100TB data

Striping is helpful, but not always

- *Basics about Striping*
  - *700 GB/s is the aggregated bandwidth with large contiguous I/O pattern*
  - *Store your data on 1 OST or multiple OSTs*
  - *Control the granularity of each data block, e.g., 1MB, 4MB*
- *Questions*
  - *Million of small files, each file is 1MB*
  - *One giant file, 1TB*

Size of File	Command
< 1GB	Do Nothing. Use default
~1GB - ~10GB	stripe_small
~10GB - ~100GB	stripe_medium
~100GB - 1TB+	stripe_large

cd $SCRATCH
lfs getstripe .

[1] Lustre Striping Recommendation on Cori:
[2] I/O Auto-tuning: Taming Parallel I/O Complexity with Auto-Tuning, B. Behzad, etc, SC'13

- Metadata: ls is too slow

Issue 2

What does the user say?

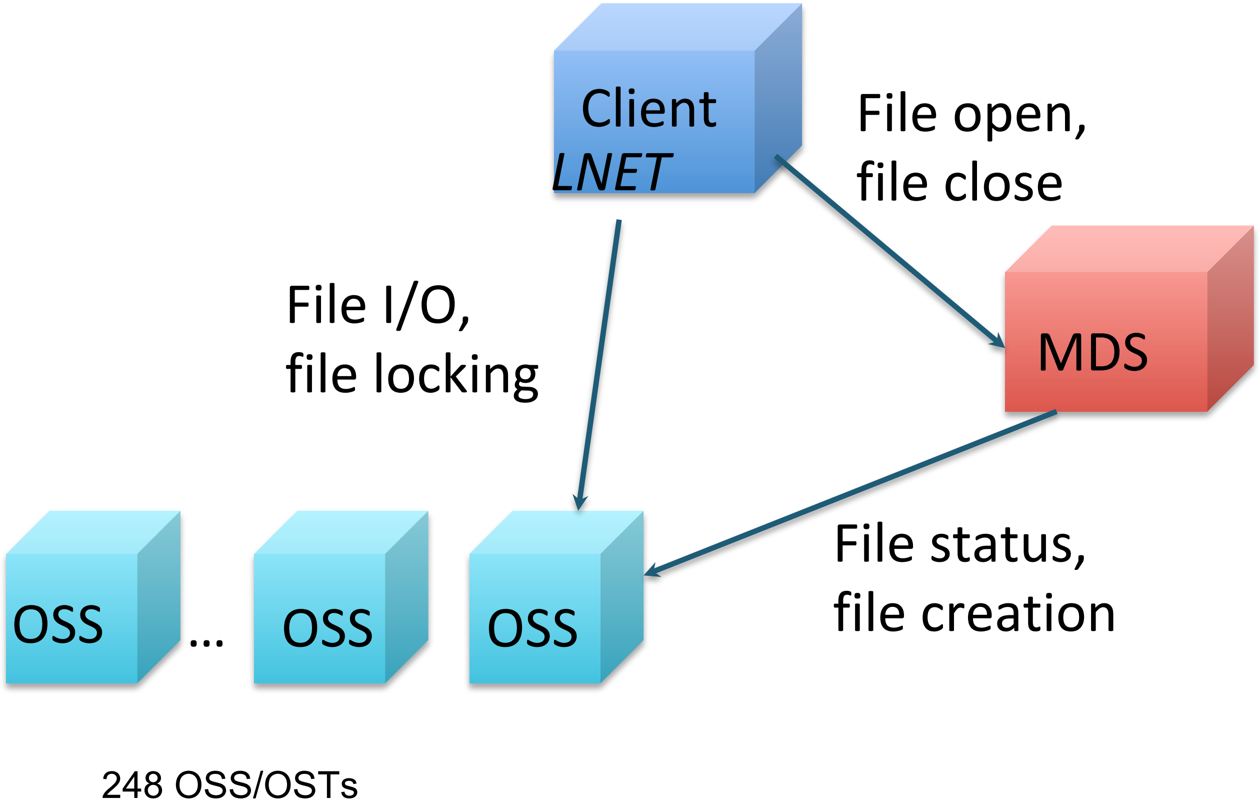
```
Harvey,

Note that the 'ls' command still has not returned as of 10:32 MST.

best wishes,
gil
```

```
I noticed that I am unable to do "ls" on certain files stored in my /global/cscratch1 area. I
can see them if I simply do "ls" in the directory, but "ls -ltr" is hanging on a particular
file (according to "strace ls -ltr"). I tried doing "rm" on the file, but this also hangs.
```

ls on Cori

A typical metadata operation path: Lustre client->LNET router-> MDS ->OSS

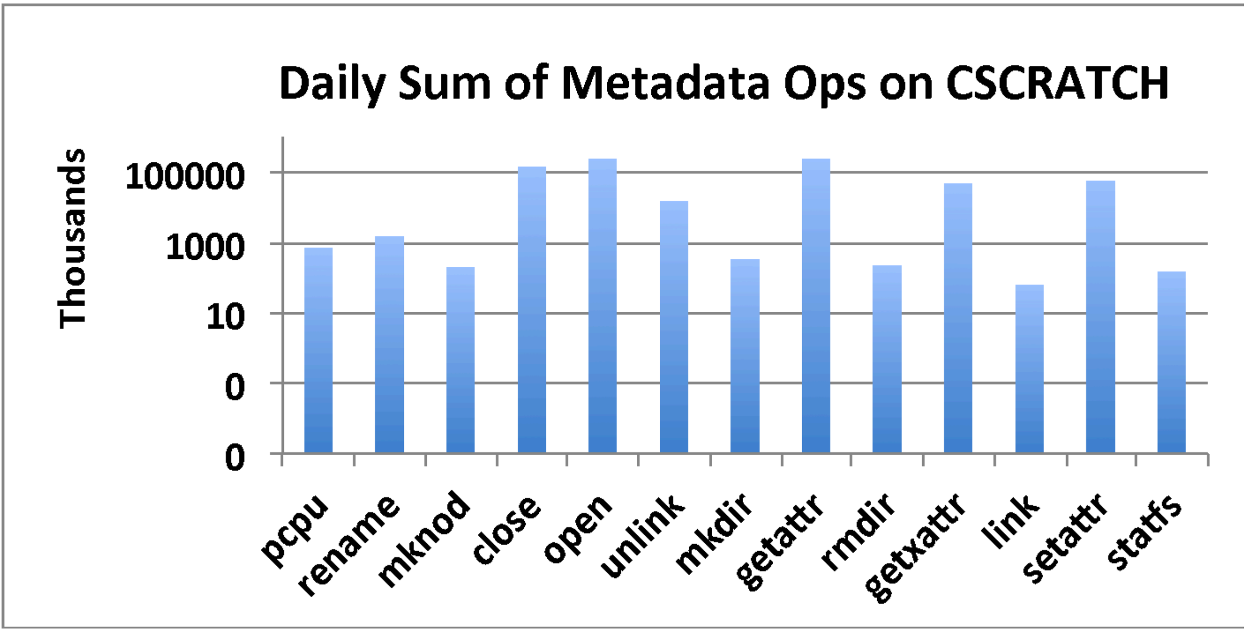
- CPU intensive
- Random and small I/O intensive



Cmd	Stripe	Cost (s)
ls	72	0.8
ls -l	72	134
ls -l	1	90

ls 60k files on Cori Scratch, Sep. 2016

ls on Cori



Measured by Glenn K. Lockwood

Metadata Operations	Daily Sum (millions)	Peak (Kilo opts/s)
open	245	255
close	143	139
getattr	234	211
getxattr	48	64
setattr	57	36

ls

ls -l

Top Five Metadata Operations on CSCRATCH

Check the 'weather' before ls

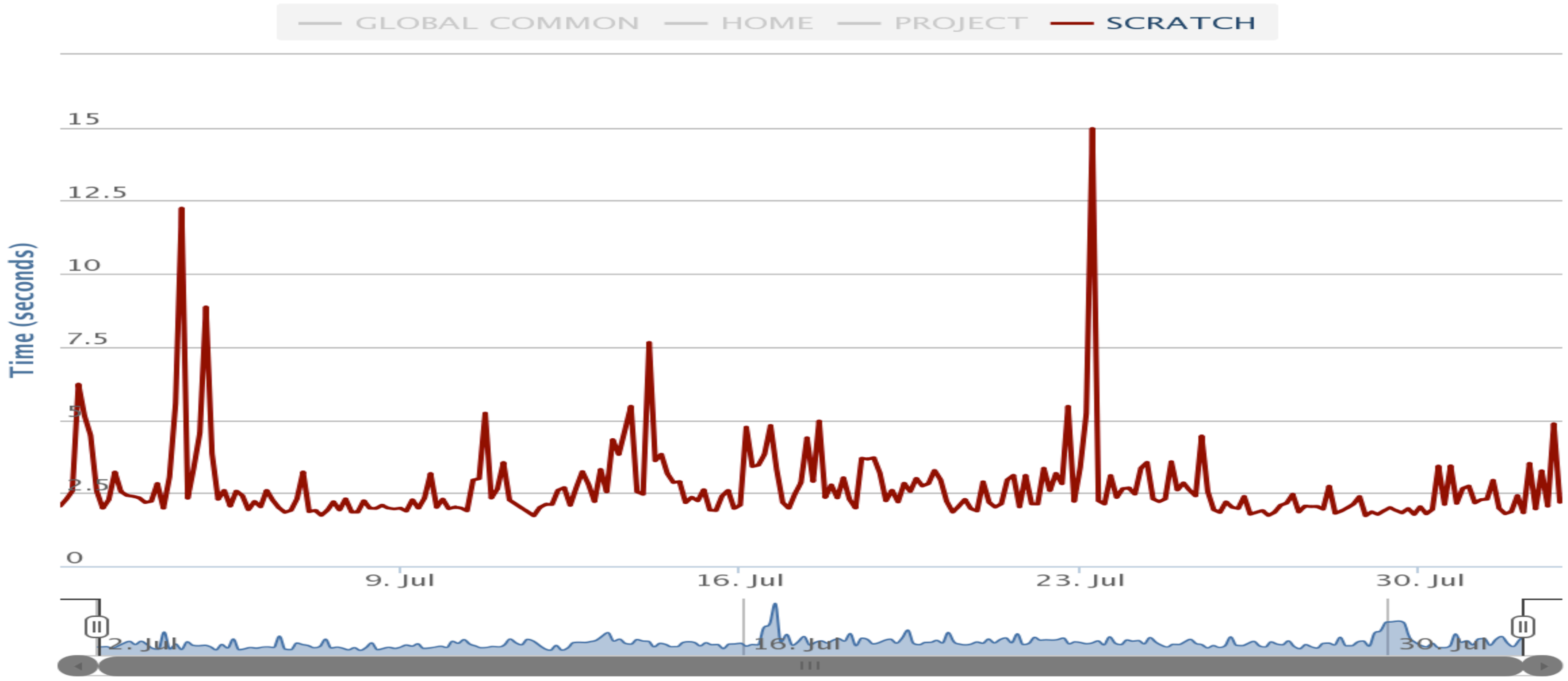
4 ADU have been added to split the workload on MDU

Check `weather' on MyNERSC website

https://my.nersc.gov/filesystems-cs.php

Weather Info:

1. MOTD
2. File system monitor
3. Benchmark
4. Data dashboard
5. Weekly email

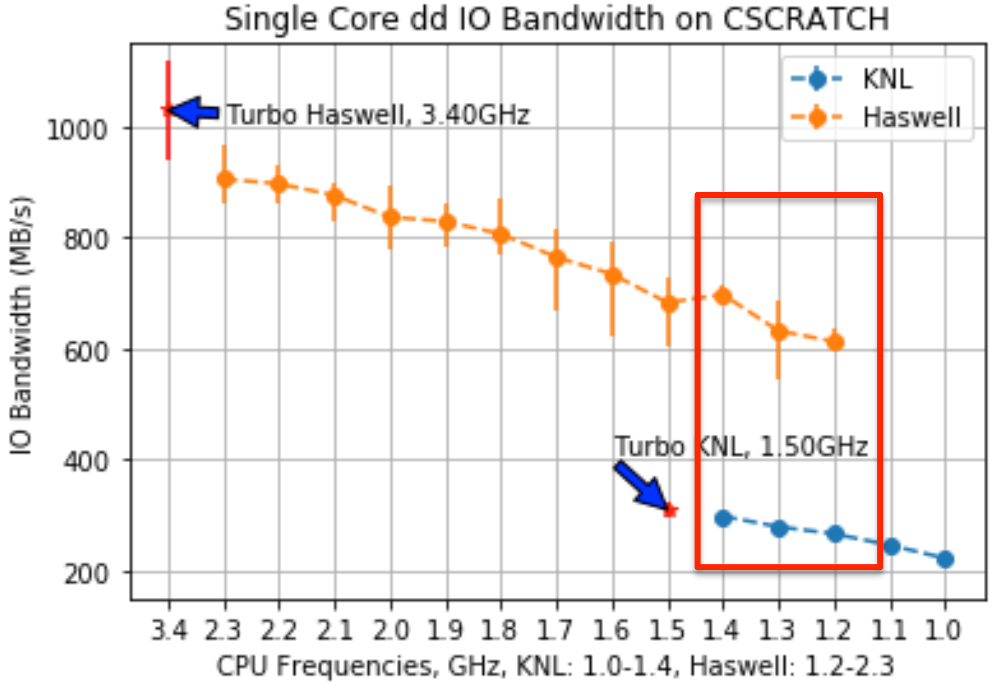


- KNL v.s. Haswell
  - *"I have used more IO processes on KNL, why the performance is still bad"*

Issue 3

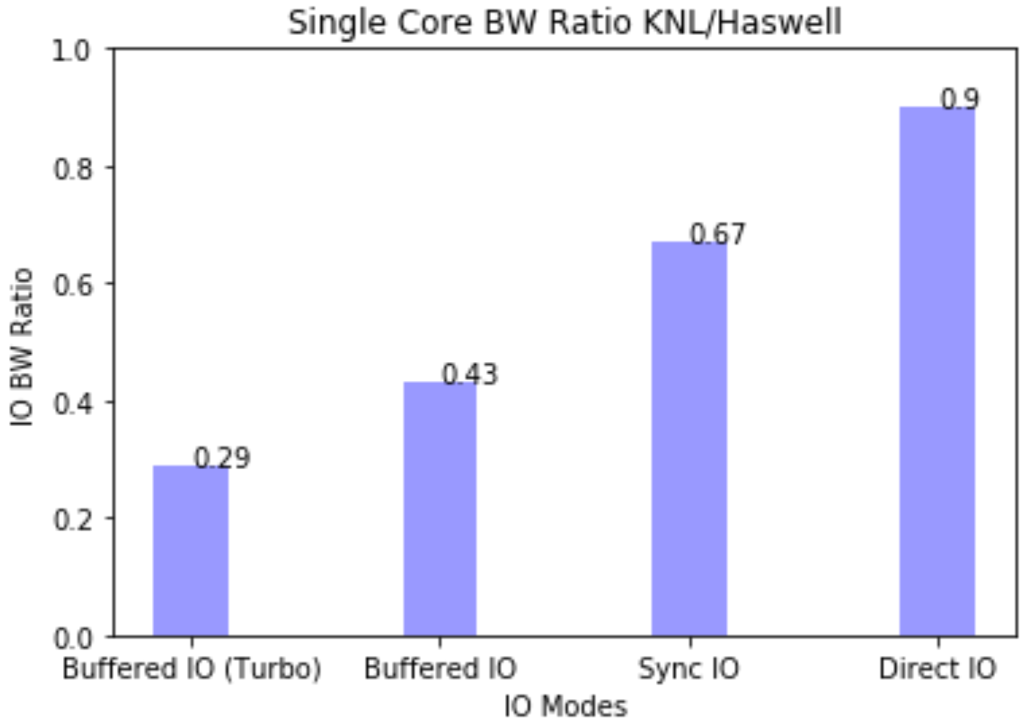
KNL v.s. Haswell

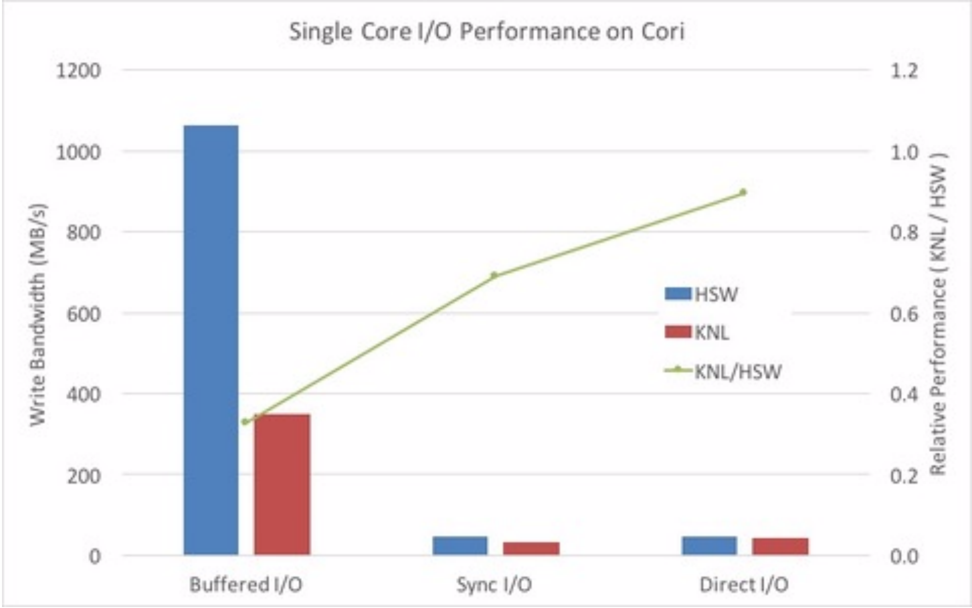
	KNL	Haswell
CPU	1.4GHz	2.3GHz
Memory	96 G DDR4, 16G HBM	128 G DDR4
Cache(L1, L2, L3)	64K, 1M	64K, 256K, 40M
Node	68 core, single socket	32 core, two sockets
Capacity	9688 nodes	2388 nodes
Hyperthreading	4 per core	2 per core



- Bandwidth Ratio Haswell / KNL = **2.30** (at same CPU freq)
  = **3.46** (Turbo)

Haswell vs KNL





- Core specialization
- Process affinity
- Threading
- etc

Note that the absolute performance number is not revealed in this plot, **Buffered IO** typically deliver **10X performance speedup in write**

http://www.nersc.gov/users/storage-and-file-systems/i-o-resources-for-scientific-applications/optimizing-io-on-cori-knl/

- Pain of Productivity
  - *"I like to use Python/Spark/Tensorflow, how can I load in the HDF5 data"*

Issue 4

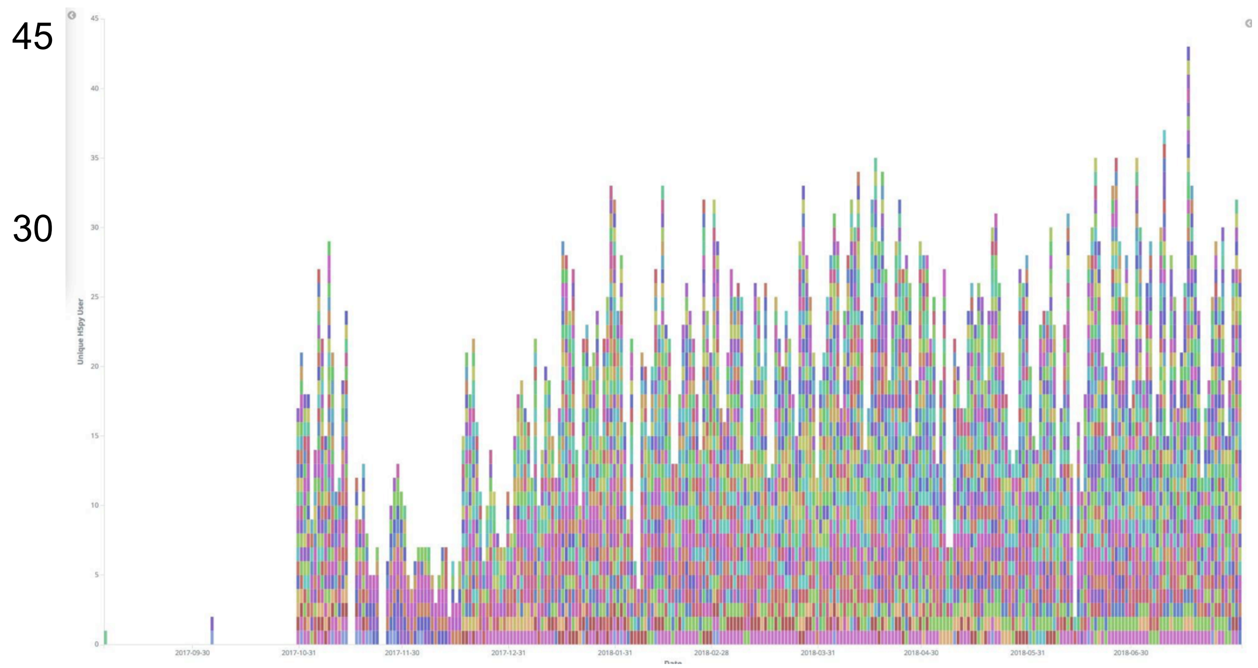
Productive I/O Interface

- Big Data Analytics Framework
  - Spark
  - Tensorflow
- Science data needs to be loaded efficiently into the engine.
  - H5py
  - H5Spark
  - Tensorflow_IO

module load python
module load h5py-parallel

H5Py: http://www.nersc.gov/users/data-analytics/data-management/i-o-libraries/hdf5-2/h5py/
H5Spark: http://www.nersc.gov/users/data-analytics/data-management/i-o-libraries/hdf5-2/h5spark/
TensorFlow_IO: https://www.tensorflow.org/api_guides/python/reading_data
TensorFlow_HDF5: Dataset API + H5py

MODS shows more



Daily H5py Unique Users from Sep 2017 to Aug 2018

H5py

361 Total H5py Users

19.594 Average H5py Users Per Day

NetCDF4 Python

132 Total netCDF4 Users

7.753 Average netCDF4 Users Per Day

H5py

- **Parallel H5py**

```
from mpi4py import MPI
import h5py
fx=h5py.File('output.h5','w',driver='mpio',comm=MPI.COMM_WORLD)
```

```
dset[start:end,:]=temp
```

Independent IO

```
with dset.collective:
 dset[start:end,:]=temp
```

Collective IO

Productivity: H5Py >> HDF5

```python
from mpi4py import MPI
import numpy as np
import h5py
import time
import sys
comm =MPI.COMM_WORLD
nproc = comm.Get_size()
comm.Barrier()
timefstart=MPI.Wtime()
f = h5py.File(filename, 'w', driver='mpio', comm=MPI.COMM_WORLD)
rank = comm.Get_rank()
dset = f.create_dataset('test', (length_x,length_y), dtype='f8')
comm.Barrier()
timefend=MPI.Wtime()
f.atomic = False
length_rank=length_x / nproc
length_last_rank=length_x -length_rank*(nproc-1)
comm.Barrier()
timestart=MPI.Wtime()
start=rank*length_rank
end=start+length_rankL
if rank==nproc-1: #last rank
    end=start+length_last_rank
temp=np.random.random((end-start,length_y))
comm.Barrier()
timemiddle=MPI.Wtime()
if colw==1:
        with dset.collective:
                dset[start:end,:] = temp
else :
        dset[start:end,:] = temp
comm.Barrier()
timeend=MPI.Wtime()
f.close()
```

```c
#include "stdlib.h"
#include "hdf5.h"
dataspace_id2 = H5Screate_simple(2, dims2, NULL);
dset_id2 = H5Dcreate(file_id2,dataset, H5T_NATIVE_DOUBLE, ...)
H5Sclose(dataspace_id2);
MPI_Barrier(comm);
double t00 = MPI_Wtime();
result_offset[1] = 0;
result_offset[0] =  (dims_x / mpi_size) * mpi_rank;
result_count[0] = dims_x / mpi_size;
result_count[1] = dims_y;
if(mpi_rank==mpi_size-1)
result_count[0] = dims_x / mpi_size + dims_x % mpi_size;
result_space = H5Dget_space(dset_id2);
H5Sselect_hyperslab(result_space, H5S_SELECT_SET, result_offset, ...);
result_memspace_size[0] = result_count[0];
result_memspace_size[1] = result_count[1];
result_memspace_id = H5Screate_simple(2, result_memspace_size, NULL);
  else{
   H5Dwrite(dset_id2, H5T_NATIVE_DOUBLE, result_memspace_id,...);
  }
  MPI_Barrier(comm);

  double t1 = MPI_Wtime()-t0;
  free(data_t);
  double tclose=MPI_Wtime();
  H5Sclose(result_space);
  H5Sclose(result_memspace_id);
  H5Dclose(dset_id2);
  H5Fclose(file_id2);
  tclose=MPI_Wtime()-tclose;
  MPI_Finalize();
}
```

Spark and TensorFlow

- Spark
  - H5Spark (H5py)
- Tensorflow
  - Dataset API + H5py

```
val sc = new SparkContext()
val rdd = h5read (sc, f, d, p)
sc.stop()
```

=

```
MPI_Init(&argc, &argv);
MPI_Comm_size(comm, &mpi_size);
MPI_Comm_rank(comm, &mpi_rank);
hid_t fapl = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(fapl, comm, info);
file= H5Fopen(f, H5F_ACC_RDONLY, fapl);
dataset= H5Dopen(file, v, H5P_DEFAULT);
hid_t dataspace = H5Dget_space(dataset);
hsize_t offset[rank];
hsize_t count[rank];
hsize_t rest = dims_out[0] % mpi_size;
if(mpi_rank != (mpi_size -1)){
  count[0] = dims_out[0]/mpi_size;
}else{
  count[0] = dims_out[0]/mpi_size + rest;
}
offset[0] = dims_out[0]/mpi_size * mpi_rank;
for(i=1; i<rank; i++){
 offset[i] = 0;
 count[i] = dims_out[i];
}
hid_t hyperid=H5Sselect_hyperslab(dataspace,
                              H5S_SELECT_SET, offset, NULL, count, NULL);
hsize_t rankmemsize=1;
for(i=0; i<rank; i++)   rankmemsize*=count[i];
hid_t memspace = H5Screate_simple(rank,count,NULL);
double * data_t=(double *)malloc(sizeof(double)*rankmemsize);
H5Dread(dataset, H5T_NATIVE_DOUBLE, memspace,
             dataspace, H5P_DEFAULT, data_t);
MPI_Finalize()
```

– https://github.com/valiantljk/h5spark

MPI Parallel Read

Performance Tradeoff: H5py ---> HDF5

H5Py Performance / HDF5 Performance

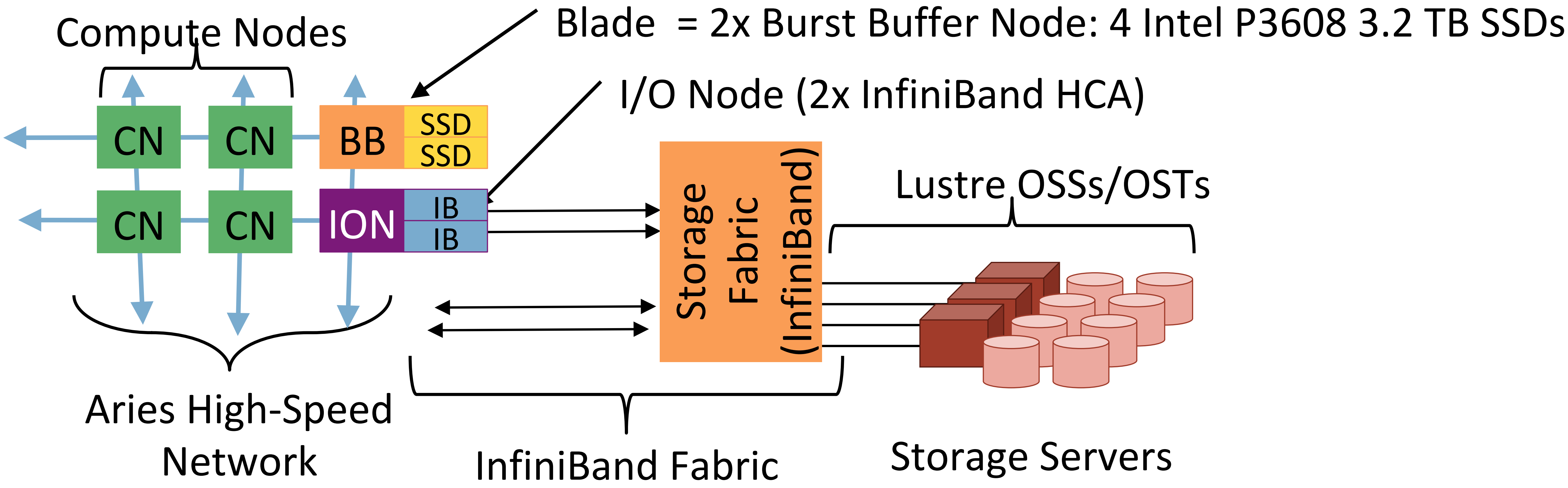
Questions: When you gain the productivity, how much performance you can afford to lose?

		Single Node	Multi-nodes
Metadata	1k File Creation	63.8%	
	1k Object Scanning	60.0%	
Independent I/O	Weak Scaling	97.8%	100%
	Strong Scaling	100%	97.1%
Collective I/O	Weak Scaling	100%	90%
	Strong Scaling	98.6%	87%

HDF5 vs. H5py: http://www.nersc.gov/users/data-analytics/data-management/i-o-libraries/hdf5-2/h5py/

Burst Buffer v.s. Lustre (on HDD)

Burst Buffer Architecture

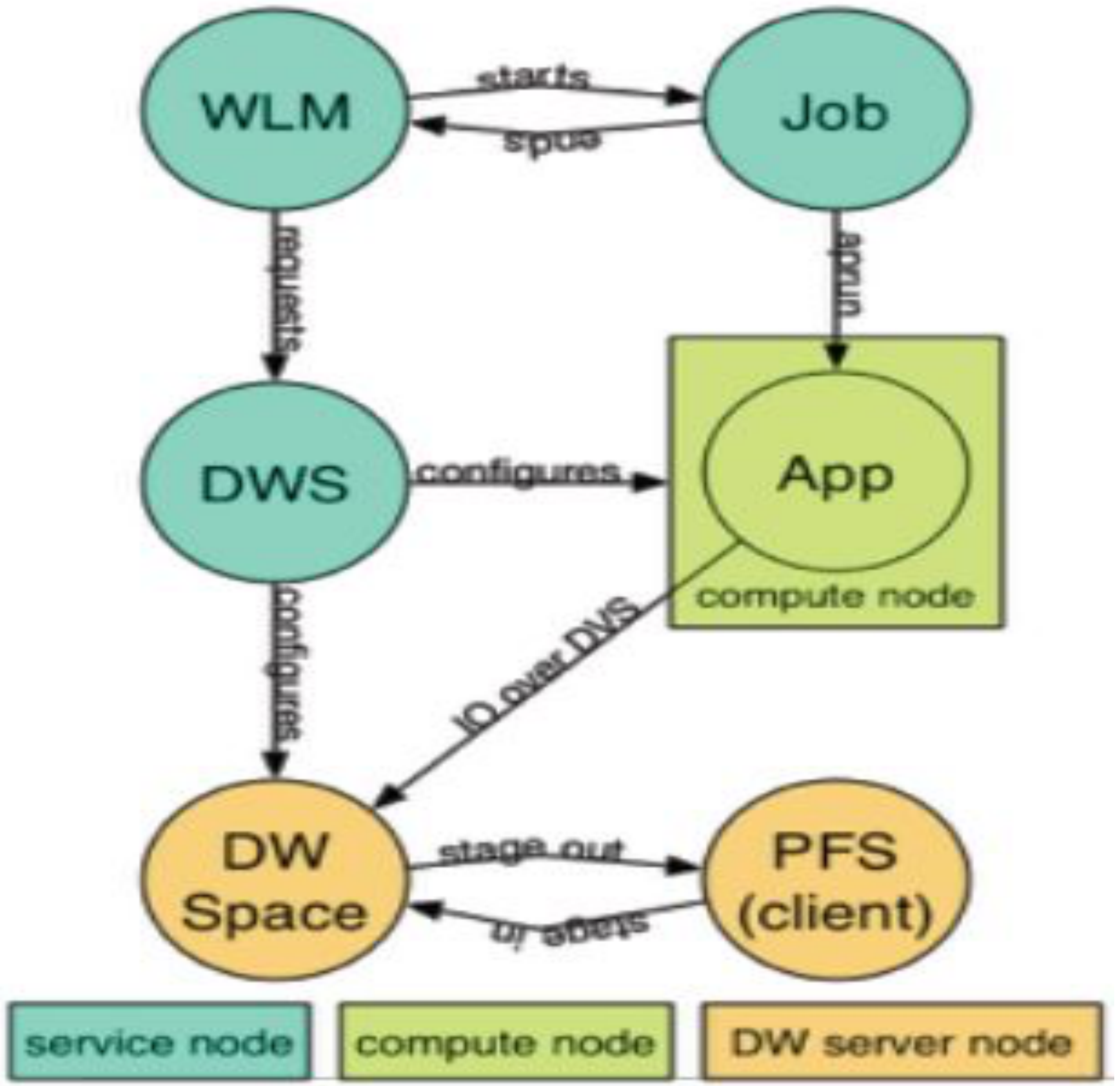


- ➢ DataWarp software (integrated with SLURM WLM) allocates portions of available storage to users
- ➢ Users see a POS
- ➢ Filesystem can b ... tion size requested)

- ~1.8PiB of SSDs over 288 nodes(6.4TB each)
- Accessible from both HSW and KNL nodes

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

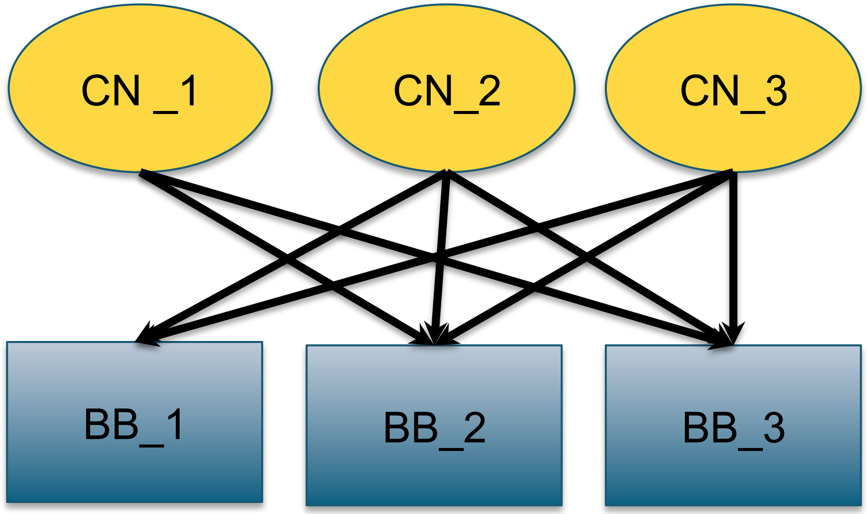


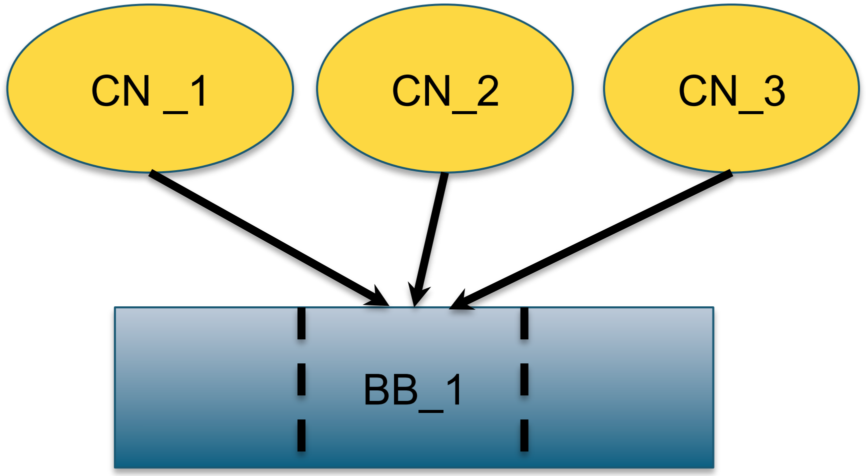
Two kinds of DataWarp Instances

- **"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

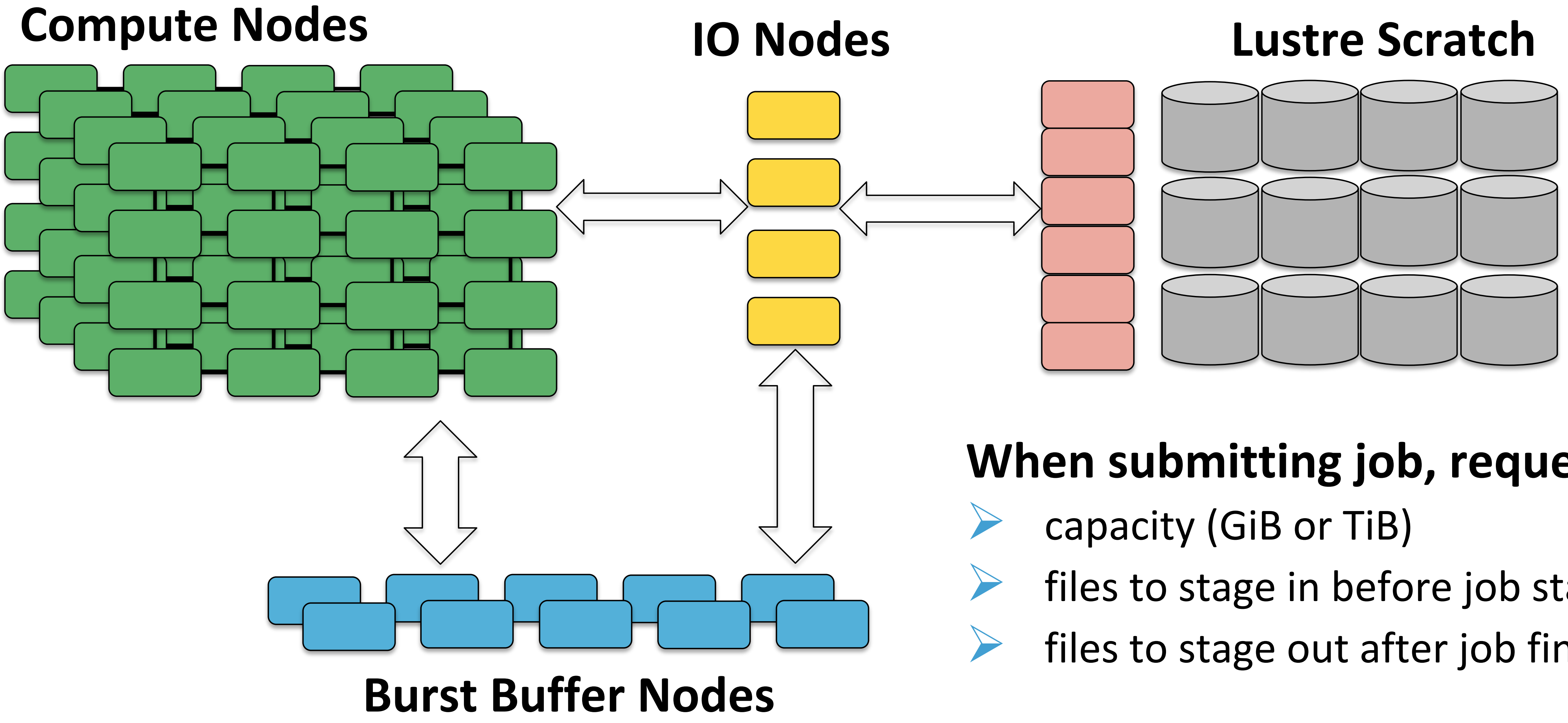




Striping, Granularity and Pools

- DataWarp nodes are configured to have granularity
    - Minimum amount of data that will land on one node
- Default pool: wlm_pool
    - granularity: 20GiB
    - #DW jobdw capacity=1000GiB access_mode=striped type=scratch pool=wlm_pool
    - For example, 50 BB nodes in wlm_pool
    - No guarantee that allocation will be spread evenly over SSDs - may see >1 "grain" on a single node

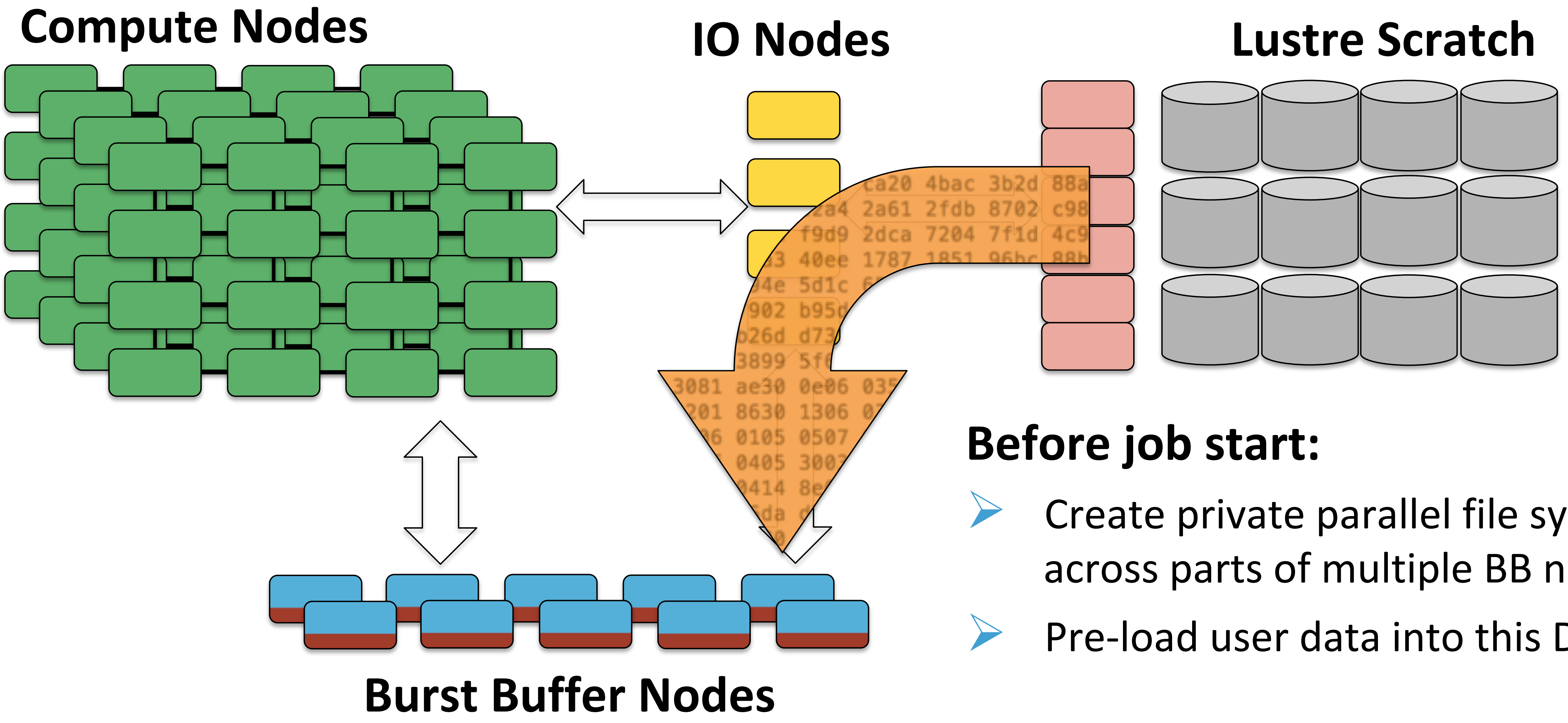
Cori's Data Paths



When submitting job, request:

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

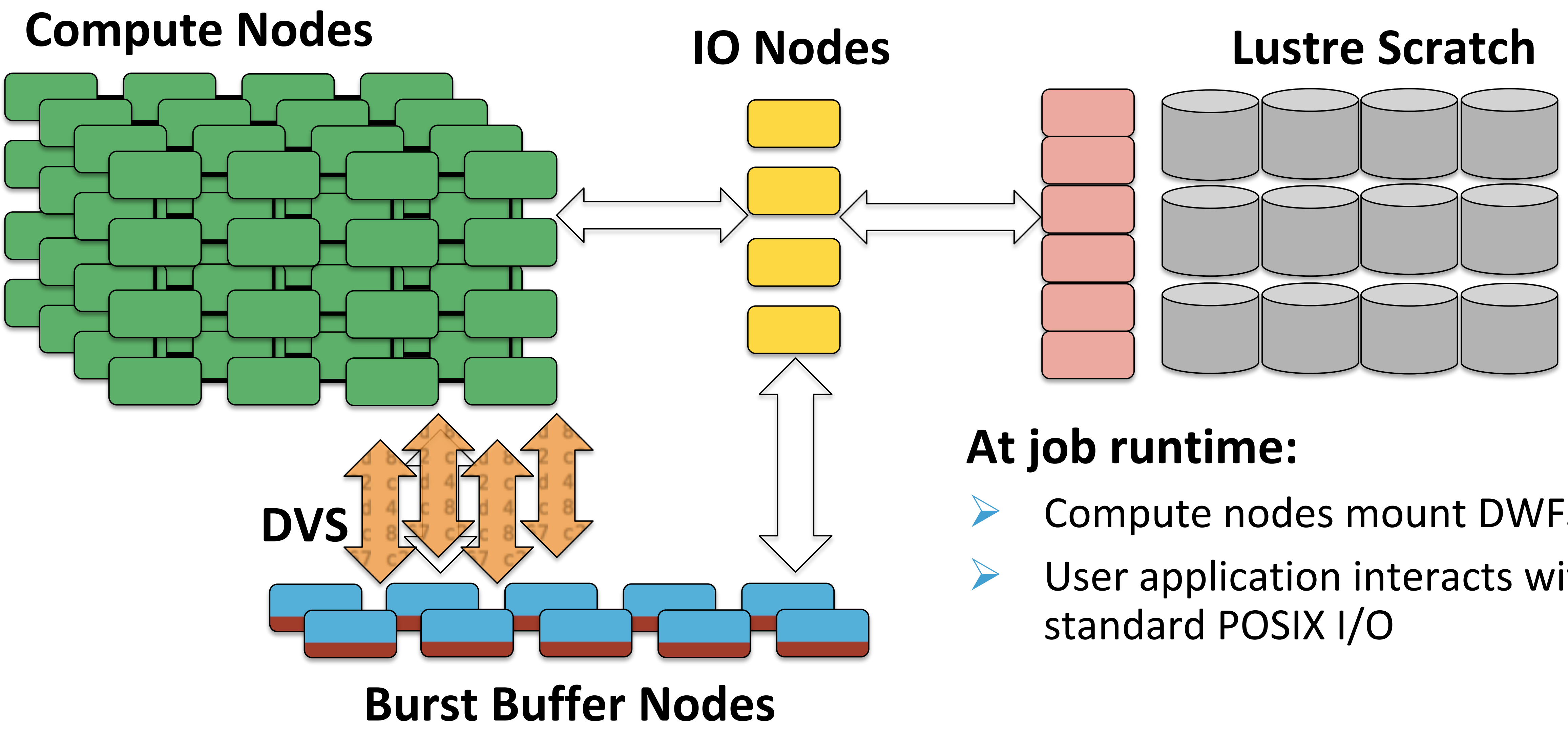
Cori's Data Paths



Before job start:

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

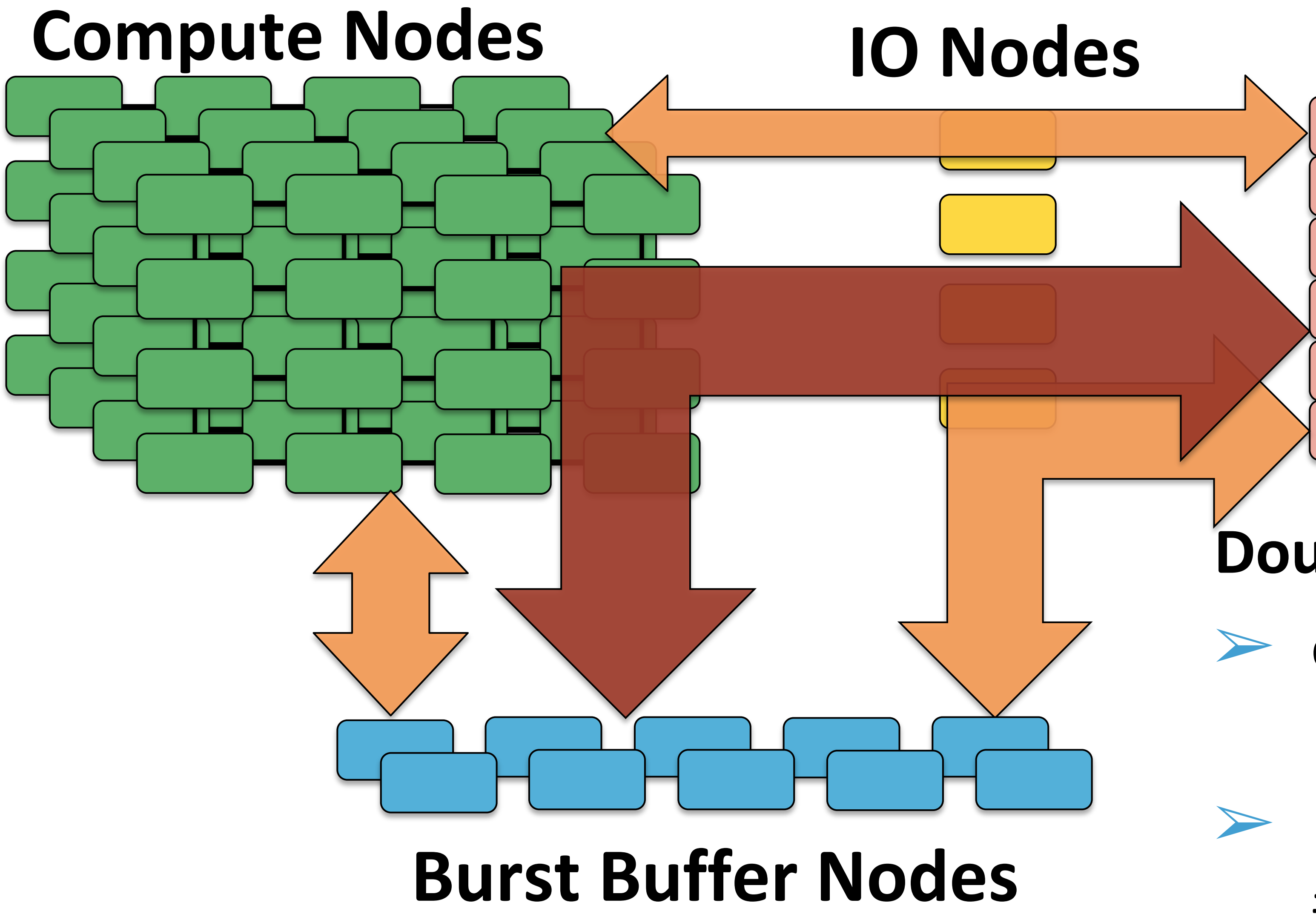
Cori's Data Paths



At job runtime:

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

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

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

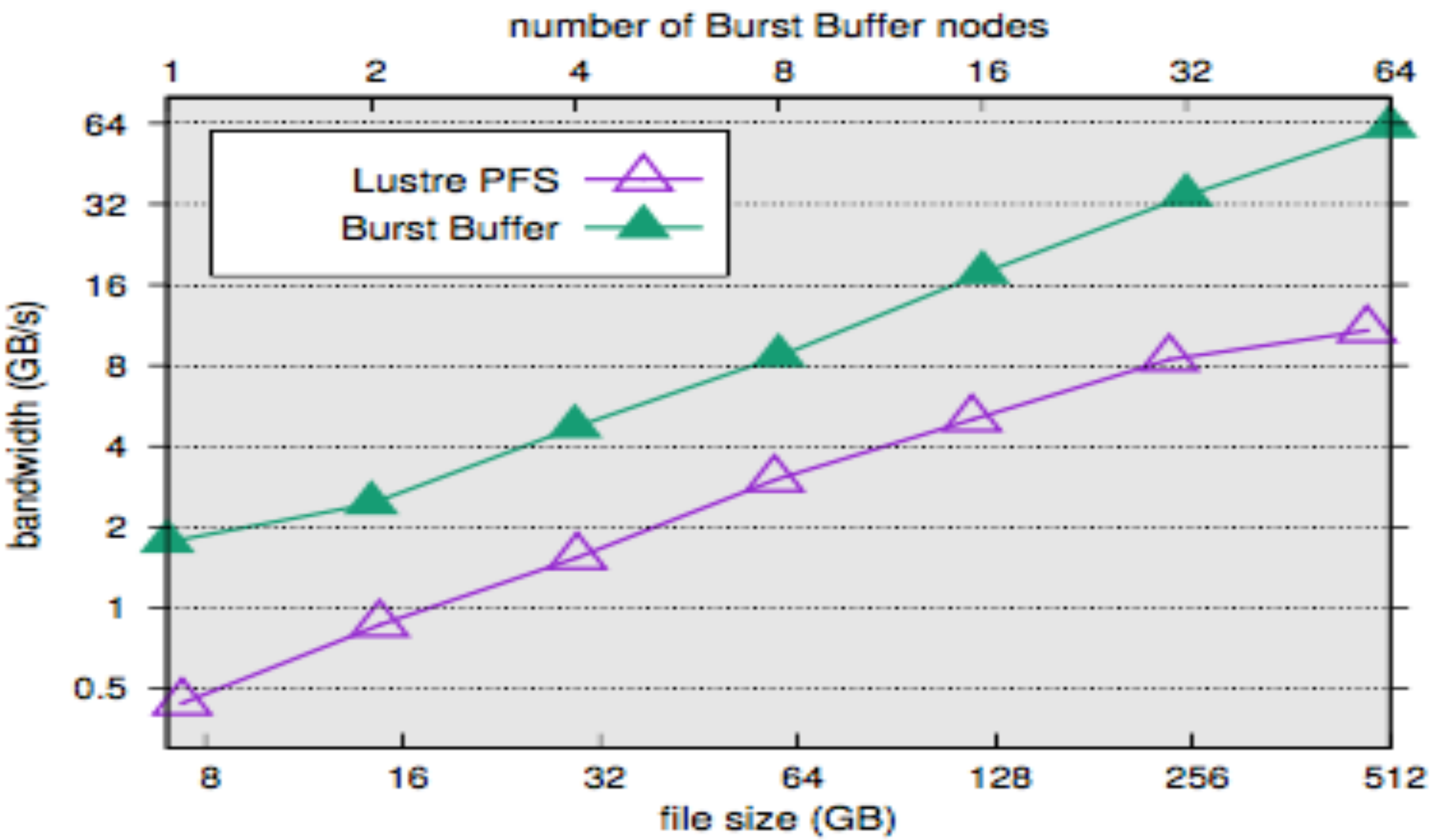
- **Burst Buffer is now doing very well against benchmark performance targets**
  - Out-performs Lustre significantly
  - (probably the) fastest IO system in the world!

	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*

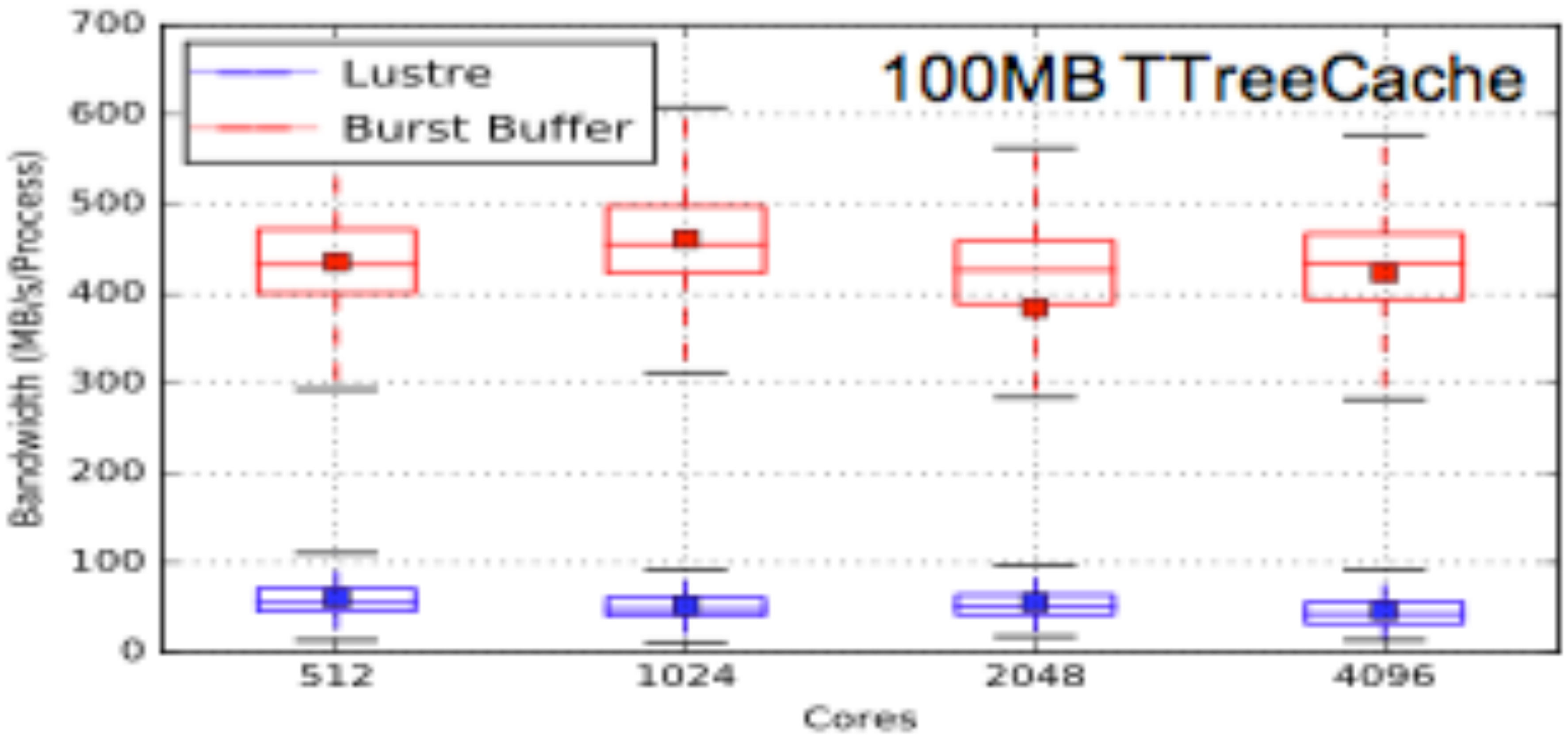
Workflows Use Case: ChomboCrunch + VisIT

- 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

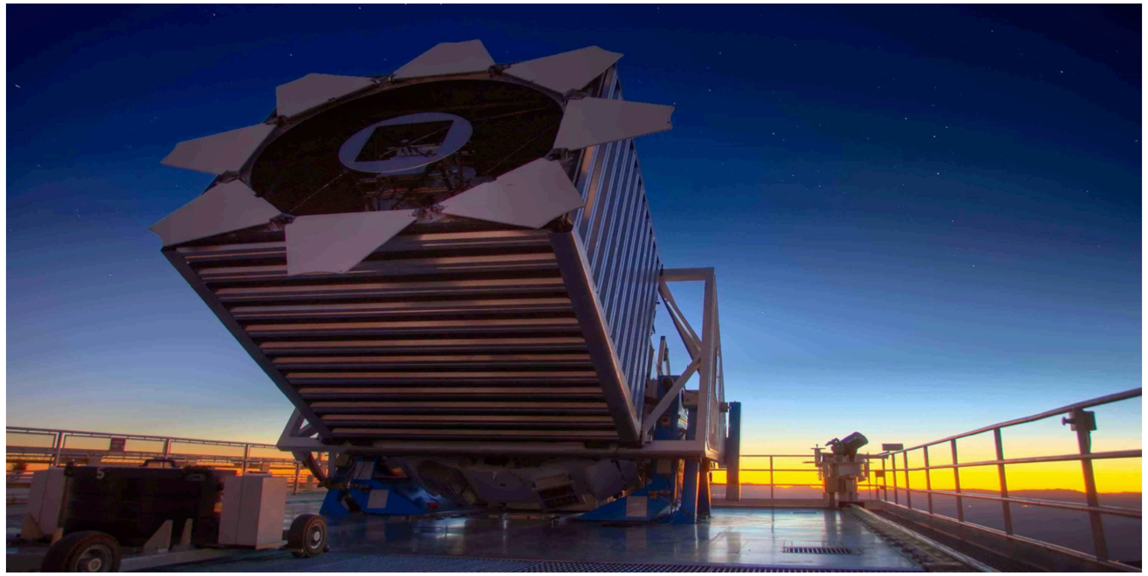


- **IOPS-heavy Data analysis**
  - Random reads from large numbers of data files
  - Used 50TB of BB space
  - ~9x faster I/O compared to Scratch.

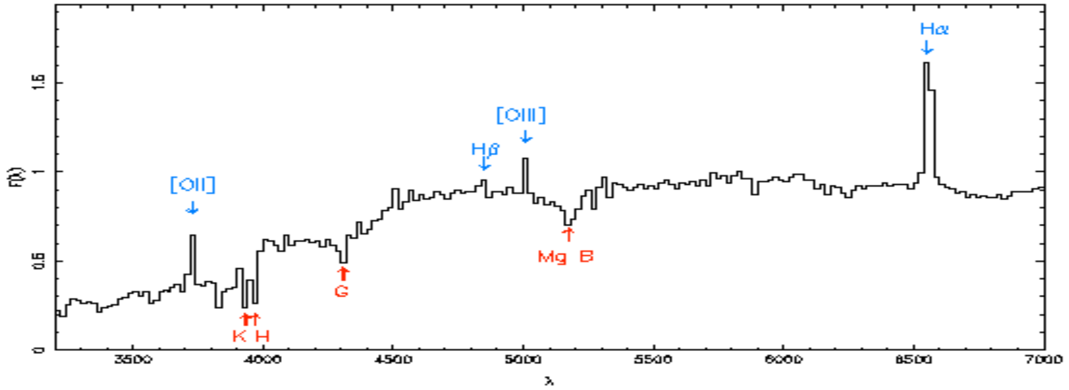
Vakho Tsulaia, Steve Farrell, Wahid Bhimji

Challenging IO use case: Astronomy data

- **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**



Thank you!

consult@nersc.gov

jalnliu@lbl.gov

Thanks Debbie Bard, Phil Carns, Rob Ross, Wahid Bhimji, Glenn Lockwood, Quincey Koziol, etc, for slides materials.