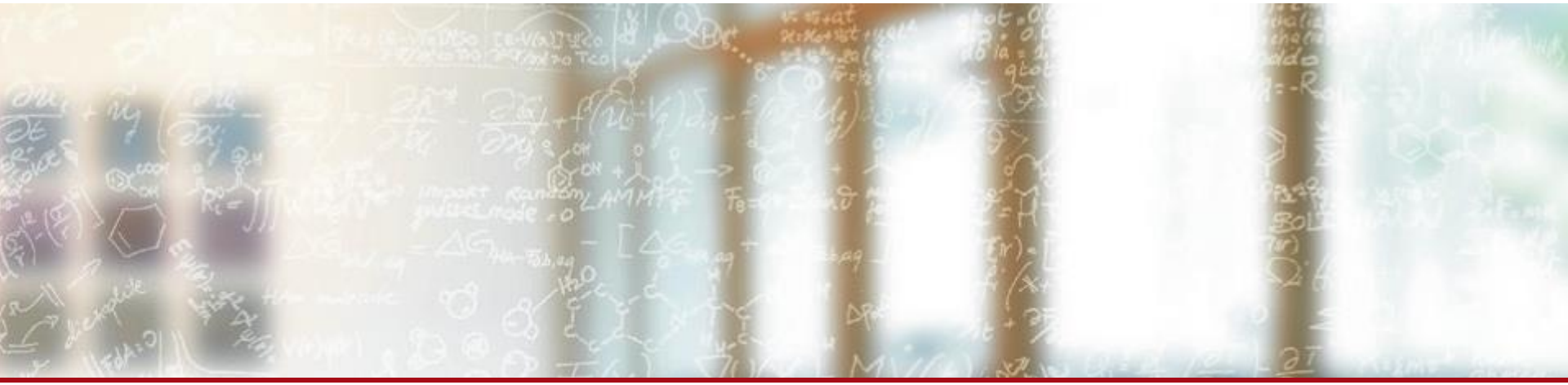
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

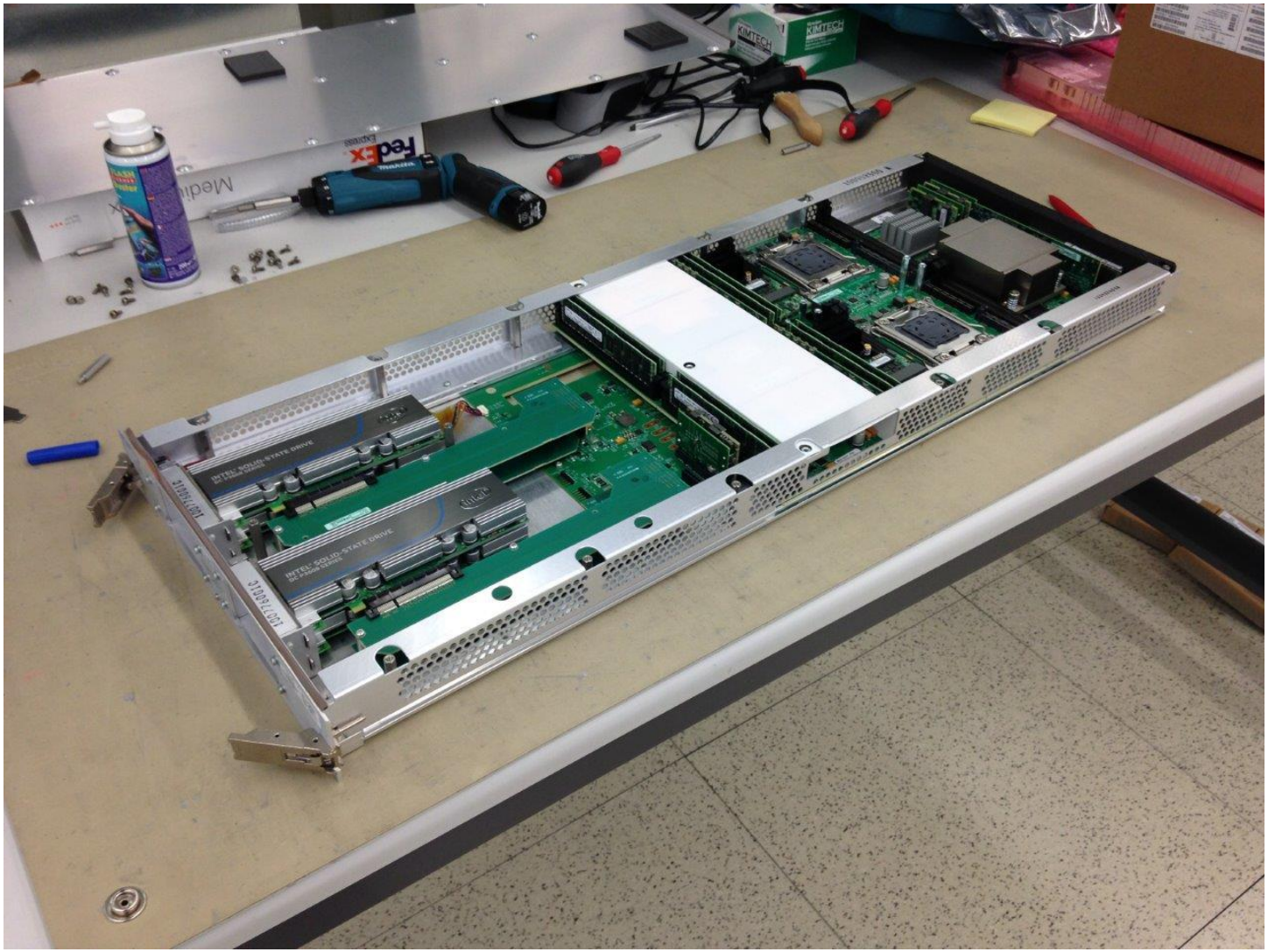


Introduction to DataWarp

Mario Valle, CSCS

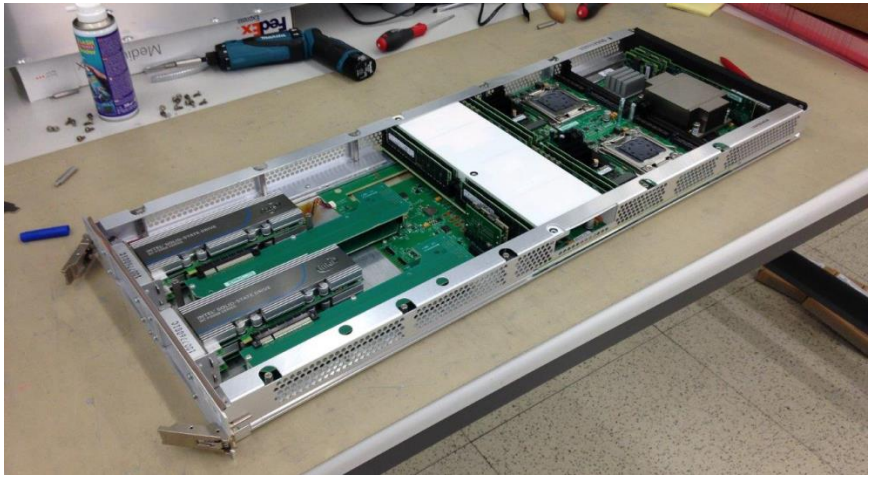
July 21, 2017

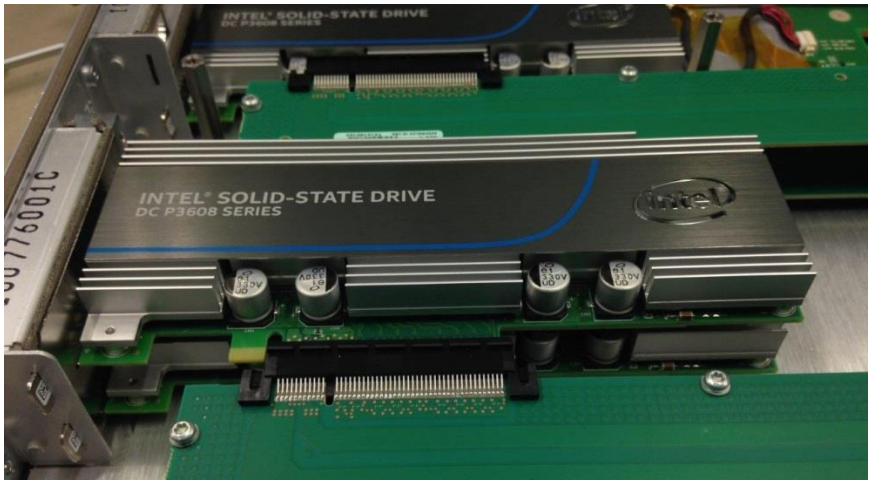
My name is Warp, DataWarp



All photos from James Brunson – Cray

DataWarp: what is it?

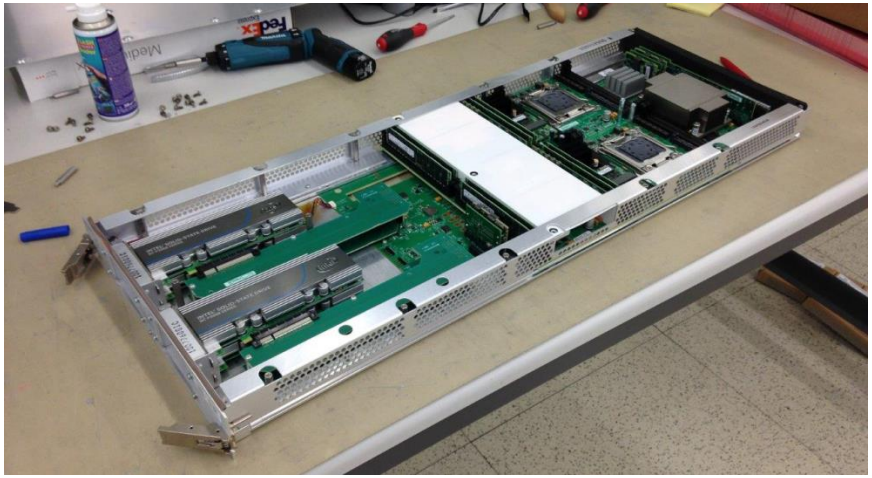


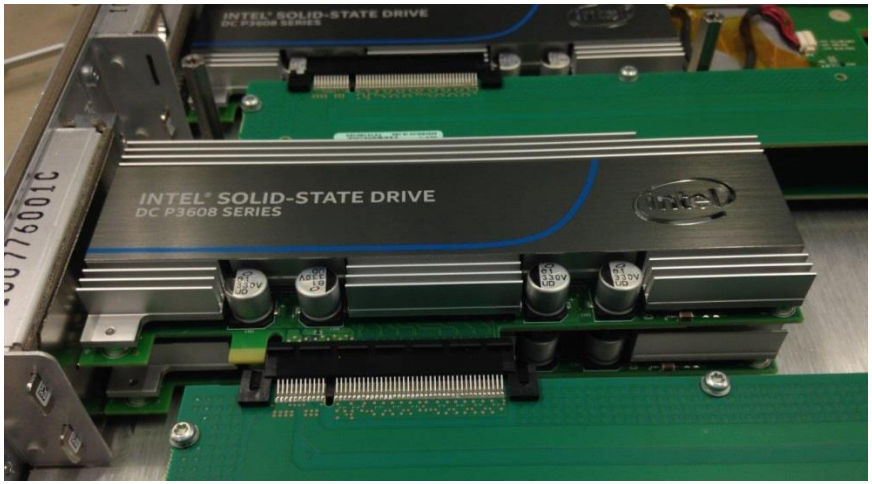


"The Cray® XC™ series DataWarp™ applications I/O accelerator technology delivers a balanced and cohesive system architecture *bla bla bla*…"

(from the product brochure)

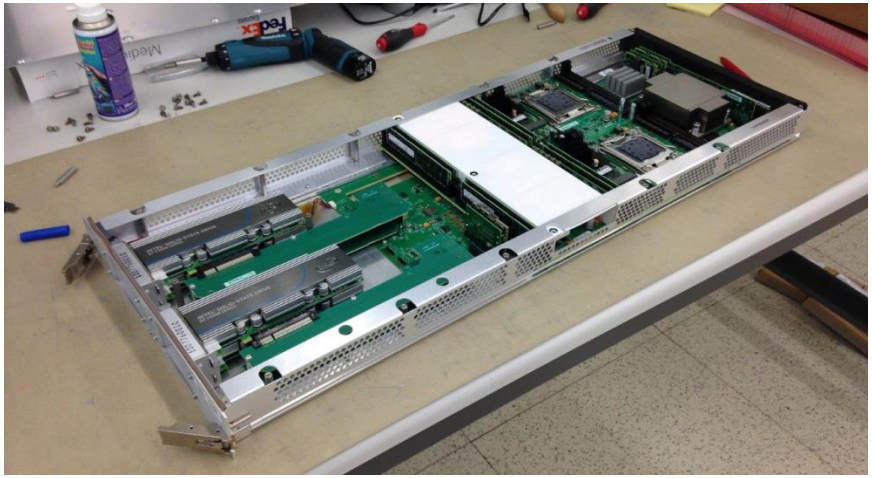
OK. But what really is DataWarp?

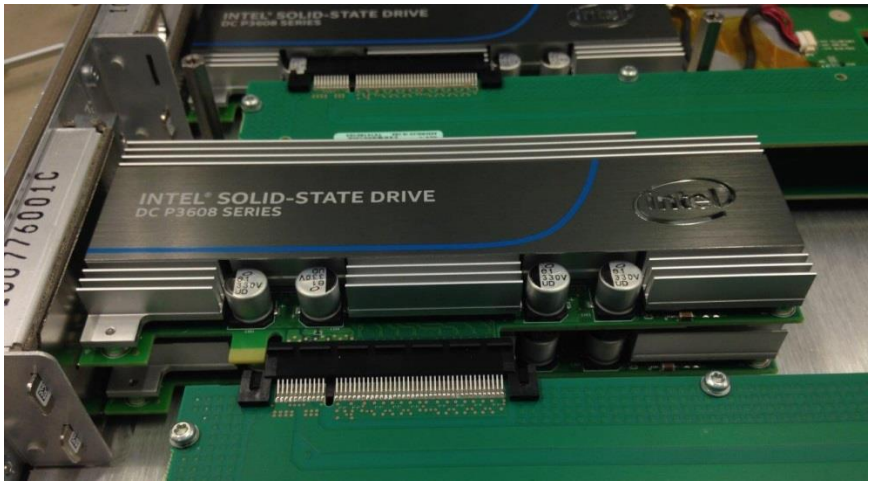




- The Cray® XC™ series DataWarp™ (call me: DataWarp or DW, thanks) …
- … provides an intermediate layer of high bandwidth, file-based storage to applications running on compute nodes. Indeed, it is a fast disk with few interesting features:
  - It sits on the Aries interconnect
  - It matches the Aries bandwidth
  - It is tightly integrated with SLURM
- Beside CSCS there are DW installations at NERSC and KAUST

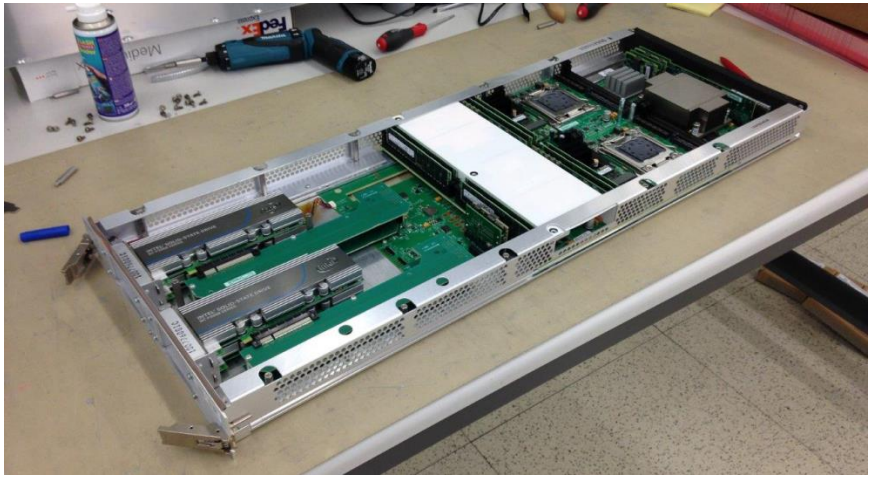
DataWarp is an hardware and software solution

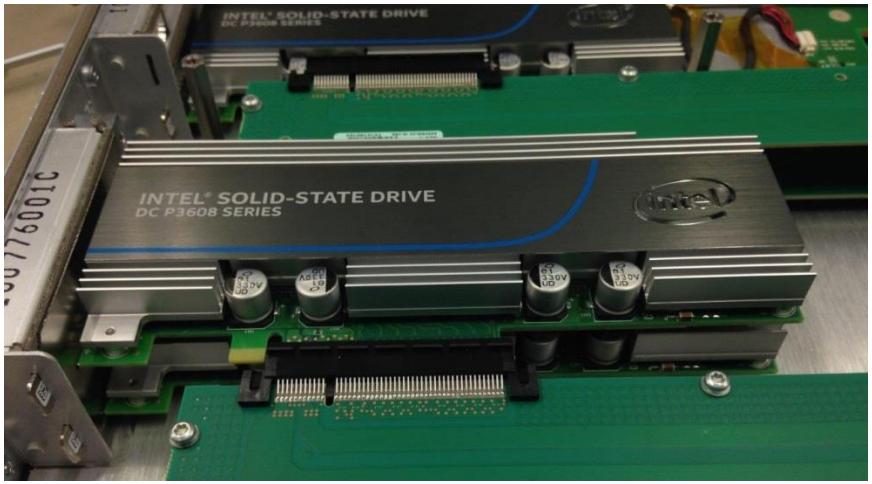




- DataWarp is composed by Intel SSD + dedicated service nodes …
- … and software …
- … tightly integrated with native SLURM
- So it is not simply a SSD

DataWarp is an hardware and software solution





- Currently DataWarp provides the following filesystem abstractions for the computing nodes:
  - Scratch storage (per job)
  - Shared storage (aka permanent)
- In a not so distant future:
  - File system cache (burst buffer)
  - Swap space for compute nodes

Agenda

- DW at CSCS
- Getting started: DW as scratch disk
- DW for shared storage (aka permanent instances)
- Data staging using SLURM and libdatawarp API
- DW and MPI-IO
- What DW cannot do
- Few CSCS experiences

Looking at what we have now at CSCS (on Piz Daint)

```
$ dwstat
    pool units quantity     free      gran
wlm_pool bytes 58.22TiB 57.77TiB 458.58GiB
$
$ dwstat nodes
    node     pool online drain  gran capacity insts activs
nid00005 wlm_pool   true false 16MiB  5.82TiB     0      0
nid00197 wlm_pool   true false 16MiB  5.82TiB     0      0
nid00325 wlm_pool   true false 16MiB  5.82TiB     0      0
nid00389 wlm_pool   true false 16MiB  5.82TiB     0      0
nid00390 wlm_pool   true false 16MiB  5.82TiB     0      0
nid00641 wlm_pool   true false 16MiB  5.82TiB     0      0
nid00774 wlm_pool   true false 16MiB  5.82TiB     0      0
nid00837 wlm_pool   true false 16MiB  5.82TiB     0      0
nid00897 wlm_pool   true false 16MiB  5.82TiB     1      0
nid01094 wlm_pool   true false 16MiB  5.82TiB     0      0
```

Size units: MiB (mebibyte) = 2^{20} bytes; TiB (tebibyte) = 2^{40} bytes
(Using –H output in SI units)

See DW through SLURM

```
$ scontrol show burst
Name=cray DefaultPool=wlm_pool Granularity=469584M
TotalSpace=61045920M FreeSpace=60576336M
UsedSpace=469584M
Flags=EnablePersistent,TeardownFailure
StageInTimeout=30 StageOutTimeout=30
ValidateTimeout=5 OtherTimeout=300
GetSysState=/opt/cray/dw_wlm/default/bin/dw_wlm_cli
Allocated Buffers:
    Name=my_database CreateTime=2017-03-09T16:24:55
Pool=(null) Size=469584M State=allocated
UserID=kleinm(23086)
  Per User Buffer Use:
    UserID=kleinm(23086) Used=469584M
```

Depending on your environment this could happen

```
$ dwstat
Traceback (most recent call last):
  File "/opt/cray/elogin/eswrap/2.0.11-2.2/bin/dwstat", line 11, in <module>
    import eswrap_main
  File "/opt/cray/elogin/eswrap/2.0.11-2.2/bin/eswrap_main.py", line 17, in <module>
    import subprocess
  File "/apps/daint/UES/jenkins/6.0.UP02/gpu/easybuild/software/Python/2.7.12-CrayGNU-
2016.11/lib/python2.7/subprocess.py", line 427, in <module>
    import select
ImportError: /usr/lib/python2.7/lib-dynload/select.so: wrong ELF class: ELFCLASS32
$
$ # The following is one way to solve the problem
$
$ module unload Python/2.7.12-CrayGNU-2016.11
$
$ dwstat
    pool units quantity     free      gran
wlm_pool bytes 58.22TiB 57.77TiB 458.58GiB
$
```

Getting started (DW as scratch storage)

```
#!/bin/bash
#SBATCH --job-name="DW-hello-world"
#SBATCH --nodes=1
#SBATCH --time=0:05:00
#DW jobdw access_mode=striped capacity=100GiB type=scratch
#--------------------------------------------------------
env | grep DW_
touch ${DW_JOB_STRIPED}/HelloWorld
mkdir ${DW_JOB_STRIPED}/SomeDir
ls -l ${DW_JOB_STRIPED}
```

```
$ sbatch –C mc ScriptAbove.sh
```

Trivial method to load needed data

```
#!/bin/bash
#SBATCH --job-name="DW-staging-files"
#SBATCH --nodes=1
#SBATCH --time=0:05:00
#DW jobdw access_mode=striped capacity=100GiB type=scratch
#-----------------------------------------
cp $SCRATCH/inputData.dat $DW_JOB_STRIPED/inputData.dat
srun ./do-something $DW_JOB_STRIPED/inputData.dat
cp $DW_JOB_STRIPED/results.dat $SCRATCH/results.dat
```

Getting started (stage files)

```
#!/bin/bash

#SBATCH --job-name="DW-staging-files"

#SBATCH --nodes=1

#SBATCH --time=0:05:00

#DW jobdw access_mode=striped capacity=100GiB type=scratch

#DW stage_in type=file ⮐
source=/scratch/snx3000/mvalle/inputData.dat ⮐
destination=$DW_JOB_STRIPED/inputData.dat

#DW stage_out type=file ⮐
source=$DW_JOB_STRIPED/results.dat ⮐
destination=/scratch/snx3000/mvalle/results.dat

#--------------------------------------------------

srun ./do-something $DW_JOB_STRIPED/inputData.dat
```

This is all you need to use DW as scratch storage

- Estimate and request the space needed (rounded up to pool granularity). See next slides.
- If you have files and directories you access frequently remember to stage them (and have $TMP points to DW).
- You can stage files (one or a list) and directories (recursive).
- The staging system has a few quirks: you should use absolute paths (on the non-DW side), you cannot create (sub)directories under $DW_JOB_STRIPED, on the SSD side you could refer only to $DW_JOB_STRIPED, and the filesystem should be visible from computing nodes.
- Remember to **`stage_out`** the results.
- **Beware of the "SSD protection from excess I/O activity mechanism". Your DW could turn suddenly read-only!**

Space allocation

```
$ dwstat pools nodes
    pool units quantity     free      gran
wlm_pool bytes 58.22TiB 57.77TiB 458.58GiB

    node     pool online drain  gran capacity insts activs
nid00005 wlm_pool   true false 16MiB  5.82TiB     0      0
...
```

Request the space needed (rounded up to pool granularity). Request more space to have striping benefits.

Access modes

Defines how the storage looks to the compute nodes. It can be either or both of the following:

- **`access_mode=striped`** Individual files are striped across multiple DataWarp nodes (aggregating both capacity and bandwidth *per file*) and are accessible by all compute nodes. The path to the storage is: **`$DW_JOB_STRIPED`**
- **`access_mode=private`** Each of the job's compute nodes has its own, private storage (like /tmp). individual files are also striped across multiple DataWarp nodes (also aggregating both capacity and bandwidth *per file*). The compute node path to the storage is: **`$DW_JOB_PRIVATE`**
- **`access_mode=striped,private`** Both together. The allocation of total size between the two modes is not clear.

Accessing $SCRATCH (using a script, method 1)

```
$ sbatch -C gpu <<\EOF
#!/bin/bash
#SBATCH --job-name="DW-staging-files"
#SBATCH --nodes=1
#SBATCH --time=0:05:00
#DW jobdw access_mode=striped capacity=100GiB type=scratch
#DW stage_in type=file ⮐
source=/scratch/snx3000/mvalle/example.dat ⮐
destination=$DW_JOB_STRIPED/example.dat
#--------------------------------------------------
srun ./do-something $DW_JOB_STRIPED/example.dat
EOF
$
```

Accessing $SCRATCH (using a script, method 2)

```
$ sbatch -C gpu <<EOF
#!/bin/bash
#SBATCH --job-name="DW-staging-files"
#SBATCH --nodes=1
#SBATCH --time=0:05:00
#DW jobdw access_mode=striped capacity=100GiB type=scratch
#DW stage_in type=file ⮐
source=$SCRATCH/example.dat ⮐
destination=\$DW_JOB_STRIPED/example.dat
#--------------------------------------------------
srun ./do-something \$DW_JOB_STRIPED/example.dat
EOF
$
```

Summarizing scratch and introducing persistent

- Per job / shared across job computing nodes

  ```
  access_mode=striped capacity=100GiB type=scratch
  ```

- Per job / private to each job computing node

  ```
  access_mode=private capacity=100GiB type=scratch
  ```

- Persistent / shared across jobs and across each job computing nodes

  ```
  access_mode=striped capacity=100GiB type=scratch
  ```

  Created and terminated by command

Creation / deletion of a permanent (or shared) instance

```
#!/bin/bash
#SBATCH --job-name="DW-create-permanent"
#SBATCH --nodes=1
#SBATCH --time=0:05:00
#SBATCH --output=/dev/null
#BB create_persistent name=mvalleBB ↵
capacity=400GiB access=striped type=scratch

#!/bin/bash
#SBATCH --job-name="DW-destroy-permanent"
#SBATCH --nodes=1
#SBATCH --time=0:05:00
#SBATCH --output=/dev/null
#BB destroy_persistent name=mvalleBB
```

Permanent instances list with the `dwstat` command

```
$ dwstat instances
inst state sess     bytes nodes             created expiration intact       label public confs
   1 CA---    1 458.58GiB     1 2017-03-09T16:24:56      never   true my_database   true     1
  55 CA---   67 458.58GiB     1 2017-06-14T09:43:32      never   true    mvalleBB   true     1
```

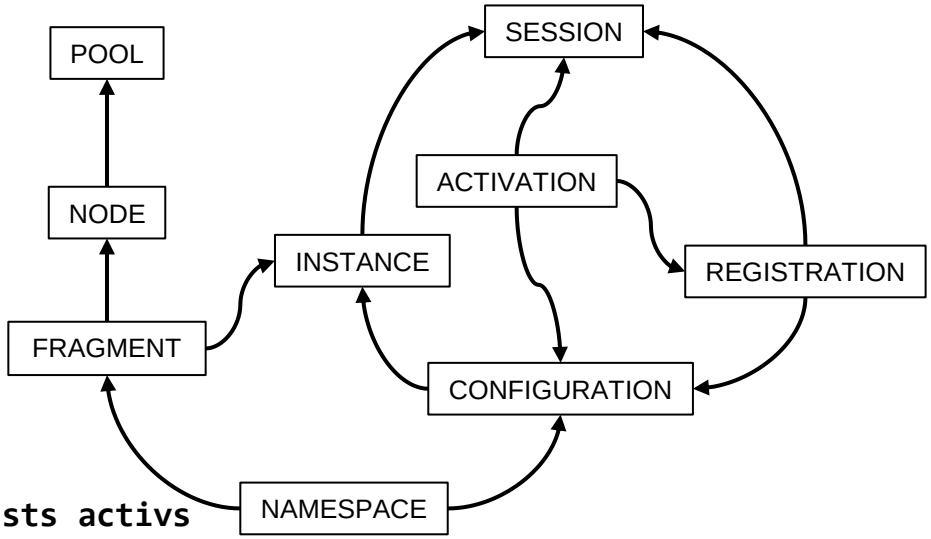
Instance

A specific subset of the storage space comprised of DataWarp *fragments*, where no two fragments exist on the same *node*. An instance is essentially raw space until there exists at least one DataWarp instance *configuration* that specifies how the space is to be used and accessed.

More later…

Permanent instances with the `dwstat all` command

```
$ dwstat all
    pool units quantity     free      gran
wlm_pool bytes 58.22TiB 57.32TiB 458.58GiB

sess state    token creator owner             created expiration nodes
  67 CA--- mvalleBB     CLI 20341 2017-06-14T09:43:32      never     0

inst state sess     bytes nodes             created expiration intact       label public confs
   1 CA---    1 458.58GiB     1 2017-03-09T16:24:56      never   true my_database   true     1
  55 CA---   67 458.58GiB     1 2017-06-14T09:43:32      never   true    mvalleBB   true     1

conf state inst    type activs
   1 CA---    1 scratch      0
  62 CA--    55 scratch      0

frag state inst  capacity     node
  13  CA--    1 458.58GiB nid00897
  73  CA--   55 458.58GiB nid00837

nss state conf frag span
  1  CA--    1   13    1
 54  CA--   62   73    1

    node     pool online drain  gran capacity insts activs
nid00005 wlm_pool   true false 16MiB  5.82TiB     0      0
...
```



Focusing on interesting info with `dwstat most`

```
$ dwstat most
    pool units quantity     free      gran
lhc_pool bytes 23.25TiB   448GiB     32GiB
wlm_pool bytes 34.93TiB 34.48TiB 458.58GiB

inst state sess     bytes nodes             created expiration intact       label public confs
   1 CA---    1 458.58GiB     1 2017-03-09T16:24:56      never   true my_database   true     1
  80 CA---  111   1.56TiB     4 2017-06-27T14:31:21      never   true    lhc_swap   true     1
  81 CA---  112   6.25TiB     4 2017-06-27T15:05:19      never   true   lhc_cvmfs   true     1
  82 CA---  113     15TiB     4 2017-06-30T10:54:44      never   true lhc_scratch   true     1

conf state inst    type activs
   1 CA---    1 scratch      0
  87 CA---   80    swap      1
  88 CA---   81 scratch      1
  89 CA---   82 scratch      1

activ state sess conf nodes ccache                          mount
   98 CA---  111   87    25     no                              -
   99 CA---  112   88    25     no                      /dws/cvmfs
  112 CA---  113   89    25     no /var/opt/cray/dws/lhc_scratch

did not find any sessions, cache configurations, registrations
```

Attach to a permanent instance

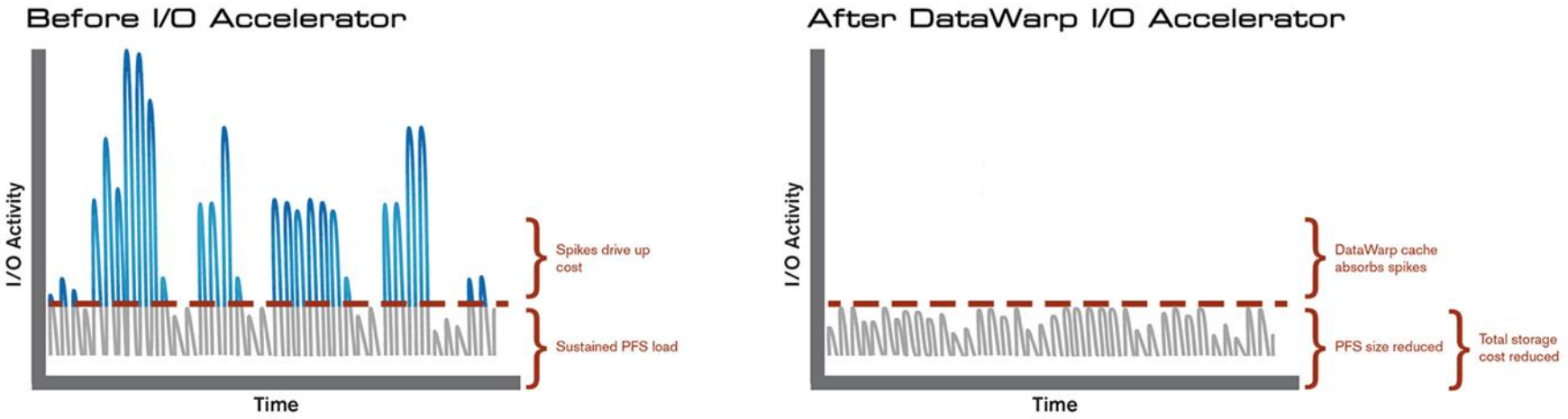
```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=03:00:00
#DW persistentdw name=mvalleBB
#DW stage_in type=file ⮐
source=/scratch/snx3000/mvalle/bmark.ttl ⮐
destination=$DW_PERSISTENT_STRIPED_mvalleBB/bmark.ttl
#DW stage_in type=directory ⮐
source=/scratch/snx3000/mvalle/apache-jena-3.2.0 ⮐
destination=$DW_PERSISTENT_STRIPED_mvalleBB/⮐
apache-jena-2.12.1
#-----------------------------------------

sh $DW_PERSISTENT_STRIPED_mvalleBB/⮐
apache-jena-3.2.0/bin/tdbloader --loc=${DB} ⮐
$DW_PERSISTENT_STRIPED_mvalleBB/bmark.ttl
```

This is all you need to use a DW permanent instance

- You need a job to allocate the instance and nothing more. In the allocation job you have no access to the newly allocated instance.
- Remember to destroy the instance when no more used (with another short job).
- It is not clear who can attach to your instance.
- Do not use a permanent instance as (permanent) storage! Also because SSD can and do fail…
- Access mode cannot be private.

DW as (non transparent) Burst Buffer

- Burst Buffer (or implicit caching for external PFS) absorbs I/O spikes

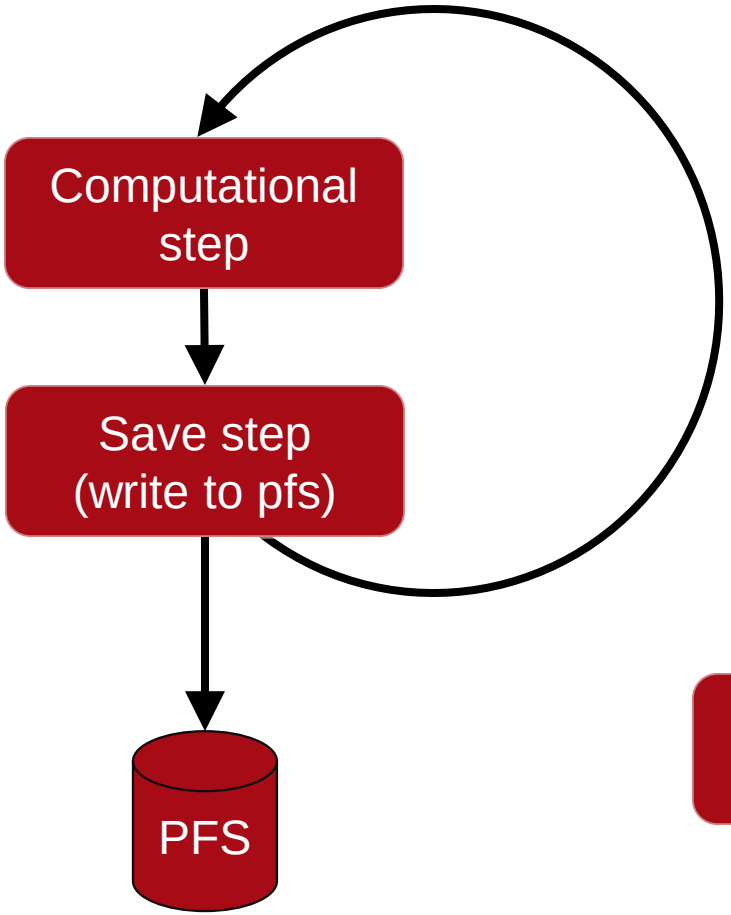


- Initial implicit caching features is in CLE6.0UPD4
- When caching will be implemented, it will be activated with:

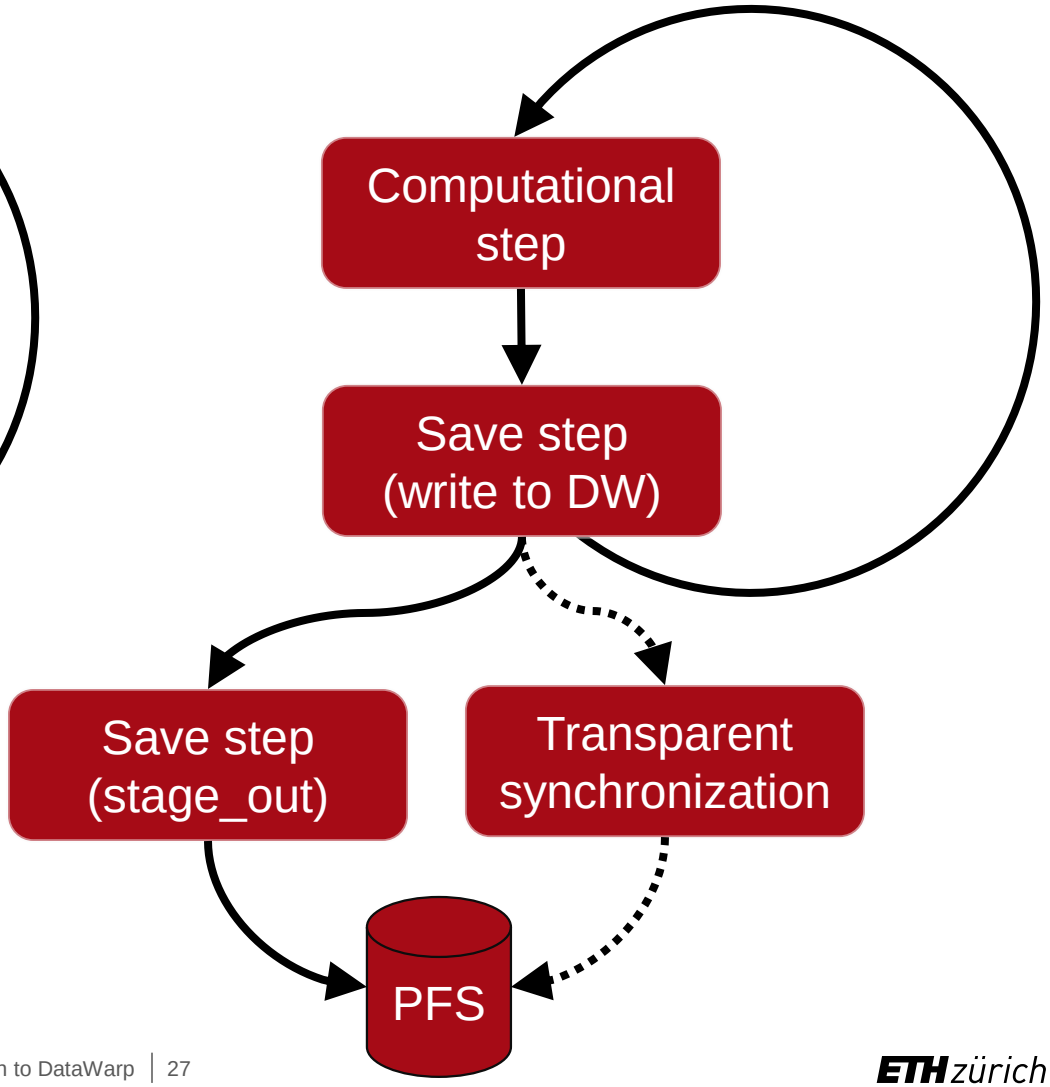
```
#DW jobdw type=cache access_mode=striped ⮐
pfs=/scratch/snx3000/mvalle capacity=100GiB
```

Non transparent Burst Buffer

Conventional multi-steps simulation



Non-transparent buffering & transparent caching



DW as (non transparent) Burst Buffer

- A Burst Buffer can be emulated by using **`libdatawarp`** API.
- Staging happens asynchronously on the DW nodes.
- API provides calls to wait for a file/directory staging to finish.
- To build:

```
$ module load datawarp

$ gcc -c `pkg-config cray-datawarp --cflags` \
datawarp_stager.c

$ gcc `pkg-config cray-datawarp --libs` \
-o datawarp_stager datawarp_stager.o
```

DW API provided by libdatawarp.a

```
#include <datawarp.h>

ret = dw_stage_file_in(target, source);

ret = dw_stage_file_out(target, destination,
                                DW_STAGE_IMMEDIATE);

ret = dw_query_file_stage(target,
          &complete, &pending, &deferred, &failed);

ret = dw_wait_file_stage(target);

ret = dw_set_stage_concurrency(source, nconcurrent);
```

e.g. target = $DW_JOB_STRIPED/file.dat
source = $SCRATCH/file.dat
environment variables should be expanded

Staging through dwcli

```
usage: dwcli stage in --session SESSION_ID --configuration
                      CONFIGURATION_ID --backing-path BACKING_PATH
                      [--file FILENAME | --dir DIRNAME]

required arguments:
  --session SESSION_ID, -s SESSION_ID
                        numeric session id
  --configuration CONFIGURATION_ID, -c CONFIGURATION_ID
                        numeric configuration id
  --backing-path BACKING_PATH, -b BACKING_PATH
                        file/dir to stage into dwfs
  --file FILENAME, -f FILENAME
                        name of the file to stage into dwfs
  --dir DIRNAME, -d DIRNAME
                        name of the directory to stage into dwfs
```

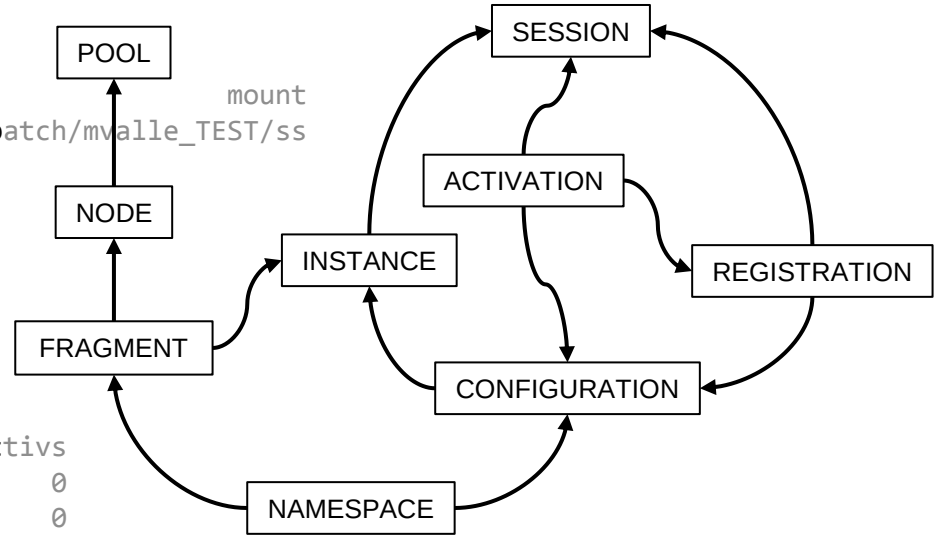
The `dwcli` command could be useful for scripts

```
$ dwcli stage in --session 67 --configuration 62 \
--backing-path \
/var/opt/cray/dws/mounts/batch/mvalleBB_striped_sc
ratch/exe-file \
--file /users/mvalle/dw/summer-school/4-mpi-
io/a.out
```

But before you even consider to look at **`dwcli`**, you should understand the intricacies of the DW data model

Output of the `dwstat all` command

```
$ dwstat all
    pool units quantity     free      gran
wlm_pool bytes 11.64TiB 11.14TiB 170.33GiB

sess state       token creator owner             created expiration nodes
 506 CA--- mvalle_TEST     CLI 20341 2016-04-19T04:33:59      never     0
 507 CA---      278823   SLURM 20341 2016-04-19T04:34:40      never     1

inst state sess     bytes nodes             created expiration intact       label public confs
 422 CA---  506 510.98GiB     2 2016-04-19T04:33:59      never   true mvalle_TEST   true     1

conf state inst    type access_type activs
 426 CA---  422 scratch      stripe      1

reg state sess conf wait
452 CA---  506  426 true
453 CA---  507  426 true

activ state sess conf nodes                                    mount
  406 CA---  507  426     1 /var/opt/cray/dws/mounts/batch/mvalle_TEST/ss

frag state inst  capacity gran     node
 789  CA--  422 340.66GiB 4MiB nid00018
 790  CA--  422 170.33GiB 4MiB nid00017

 ns state conf frag span
426  CA--  426  789    2

    node     pool online drain  gran capacity insts activs
nid00017 wlm_pool   true false 16MiB  5.82TiB     1      0
nid00018 wlm_pool   true false 16MiB  5.82TiB     1      0
```



Something about performances

`striping`

Allocate more space than needed to have fragments on different nodes (see CSCS tests results)

`I/O blocksize`

Sequential I/O is done in blocks of fragment granularity size

`client_cache=yes`

Although many workloads can benefit from client-side caching because it can reduce the frequency and necessity of network operations (e.g. Java I/O), others will be negatively affected. In some cases (e.g., many compute nodes modifying a specific file simultaneously with this access mode) data corruption can occur.

Add swap space (future)

```
#DW jobdw access_mode=striped capacity=total ⮐
type=scratch
```

```
#DW swap size
```

size in GiB of the swap space per computed node
total should be bigger than ***size*nodes***

Example:

```
#DW jobdw type=scratch access_mode=striped ⮐
capacity=100GiB
#DW swap 10GiB
#--------------------------------------

srun -N 10 big_memory_application
```

MPI-IO

- Nothing special to setup
- Point files to: **`$DW_JOB_STRIPED/file`**
- MPI-IO fine tuning is possible. The collective buffering default is to use 1 aggregator per DW node. You could increase the number of aggregators with:
  `export MPICH_MPIIO_HINTS="*out.dat:cb_nodes=8"`
- **`export MPICH_MPIIO_HINTS_DISPLAY=1`** could help

What DW cannot do

- Cannot provide memory mapped files.
- Cannot guarantee the filesystem never turns readonly.

  SSD protection from excess I/O activity mechanism that could abort your I/O. This mechanism can be tuned by putting these modifiers in parenthesis after the access mode.

  - **`MFS`** maximum size of any file
  - **`MFC`** maximum number of created files
  - **`write_window_multiplier`** Number of times *capacity* number of bytes may be written in a period defined by **`write_window_length`**
  - **`write_window_length`** Number of seconds to use when calculating the moving average of bytes written
- Cannot guarantee/impose a fair use of the resource.

Anything missing?

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

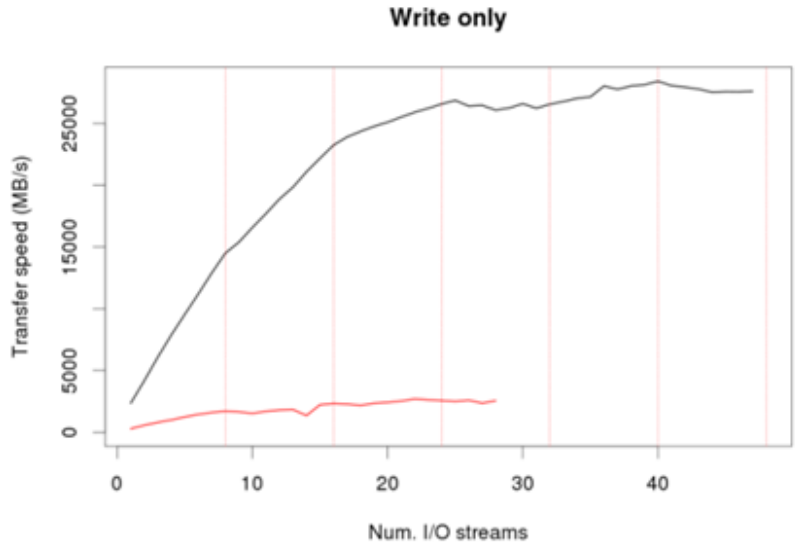
DataWarp testing at CSCS

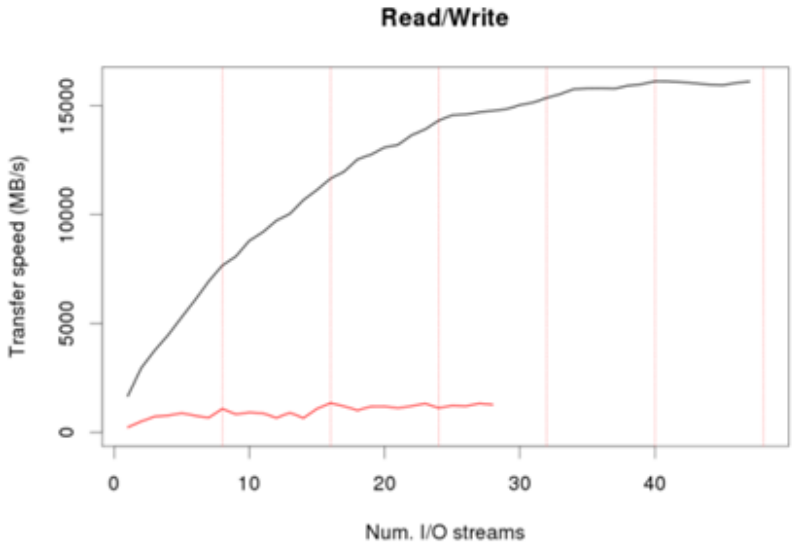
Run few kinds of tests

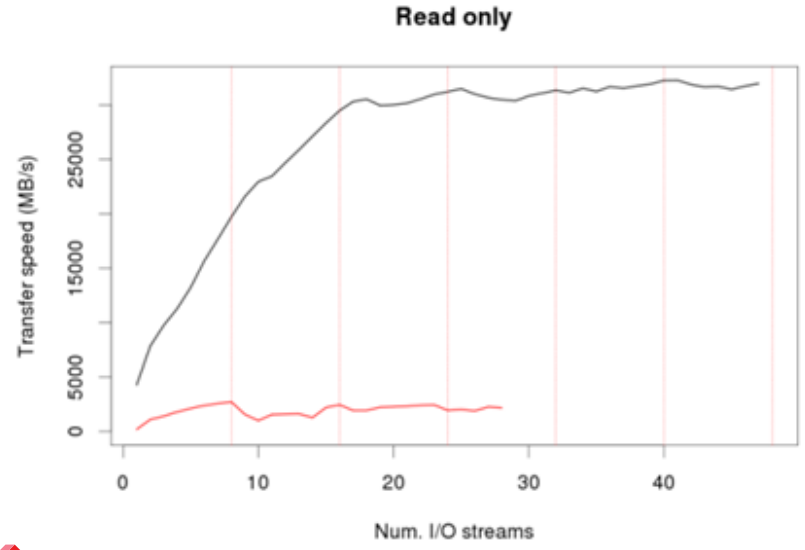
1. **Parallel I/O streams**
   Artificial sequential block I/O
2. **Astrophysical simulation code ENZO**
   An example of real scientific simulation job using HDF5 files
3. **Berlin SPARQL benchmark**
   Creating and accessing a triple-store database using Java I/O
4. **Random I/O using Vdbench**
   Accessing plain files in random read and write

Parallel sequential I/O

Parallel sequential I/O







Black: DataWarp

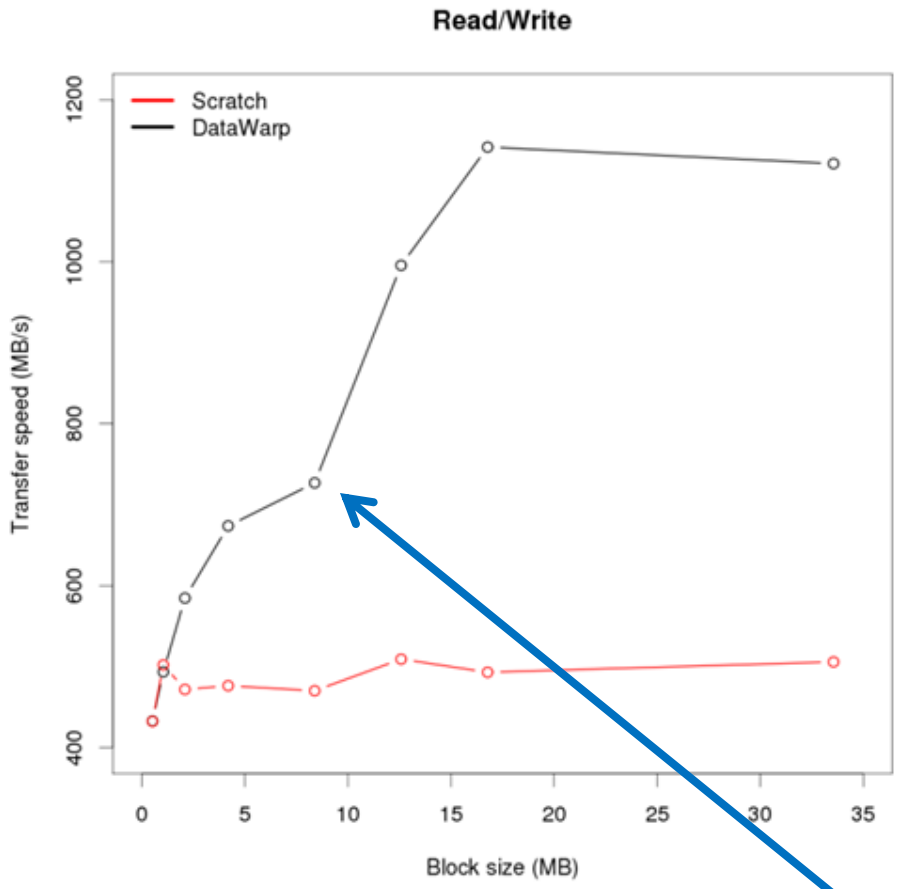
Red: Lustre

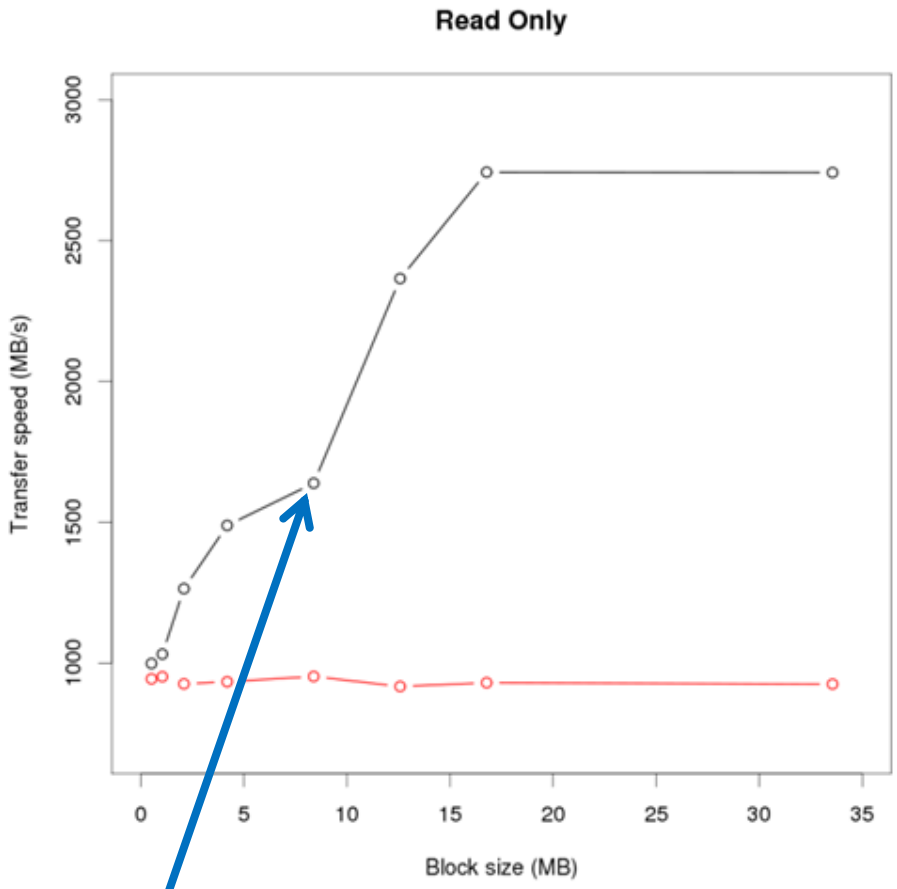
Summary of results (note: not on Piz Daint)

DataWarp		Lustre	
Write Only	28.4 (GiB/s) at 40 streams	Write Only	2.7 (GiB/s) at 22 streams
Read/Write	16.1 (GiB/s) at 40 streams	Read/Write	1.3 (GiB/s) at 16 streams
Read Only	32.3 (GiB/s) at 41 streams	Read Only	2.7 (GiB/s) at 8 streams

- Done with 8 computing nodes / 8 DataWarp nodes
- Simple **dd** I/O using a maximum of 6 tasks per node
- Using a blocksize of 16 MiB (more on this later)
- The throughput for a single DataWarp node is around 3.8 GiB/s. For Lustre, it is 2.7 GiB/s.
- The throughput for a single stream saturates between 700 and 800 MiB/s. Instead with Lustre it saturates between 100 and 300 MiB/s

Blocksize effect






```
    node     pool online drain  gran capacity insts activs
nid00017 wlm_pool   true false 16MiB  5.82TiB     1      0
nid00018 wlm_pool   true false 16MiB  5.82TiB     1      0
```

Astrophysical simulation code ENZO

Astrophysical simulation code ENZO

Time to Solution (sec)

DataWarp

Nodes\task per node	2	4	8	16
2		6,282	3,432	1,838
4	6,197	3,152	1,594	921
8	3,067	1,462	804	465
16	1,441	736	407	312
32	725	372	275	530

Lustre

Nodes\task per node	2	4	8	16
2		6,431	3,555	1,949
4	6,377	3,285	1,712	1,026
8	3,193	1,596	925	561
16	1,582	921	542	426
32	905	501	432	663

Time to solution reduction using DataWarp instead of Lustre

Nodes\task per node	2	4	8	16
2		-2.3%	-13.7%	-5.7%
4	-2.8%	-4.1%	-6.9%	-10.2%
8	-4.0%	-8.4%	-13.1%	-17.1%
16	-8.9%	-20.1%	-24.9%	-26.8%
32	-19.9%	-25.7%	**-36.3%**	-20.1%

Astrophysical simulation code ENZO

Solution writing only throughput (MiB/s)

DataWarp

Nodes\task per node	2	4	8	16
2		2,832	2,638	5,214
4	1,736	3,649	5,299	7,949
8	3,063	6,764	8,369	8,816
16	5,650	9,096	8,188	8,112
32	8,501	8,669	8,372	9,035

Lustre

Nodes\task per node	2	4	8	16
2		1,158	1,304	1,741
4	998	1,424	2,202	2,225
8	1,493	2,658	2,431	2,733
16	2,270	1,750	1,991	3,139
32	3,110	4,014	2,283	15,844

Throughput comparison (ratio DataWarp over Lustre)

Nodes\task per node	2	4	8	16
2		2.45	2.02	2.99
4	1.74	2.56	2.41	3.57
8	2.05	2.54	3.44	3.23
16	2.49	**5.20**	4.11	2.58
32	2.73	2.16	3.67	**0.57**

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Berlin SPARQL benchmark (on Apache Jena DB)

Berlin SPARQL benchmark (on Apache Jena DB)

Synthetic data creation (50M triples for 4.3 GB)

	Lustre	DataWarp	Cached DataWarp
real	1' 52"	5' 52"	1’ 38”
user	4' 30"	4' 56"	4’ 37”
sys	0' 31"	0' 48"	0’ 10”

Cached DataWarp Enables DW client caching with: **`client_cache=yes`**

Database creation (final size 9.6 GB)

	Total speed (triples/sec)	Stall time (sec)
DataWarp	988	2,904
Cached DataWarp	1,631	39
Lustre (direct)	3,237	254
Lustre (mapped)	13,974	7

Database creation is essentially a single thread process

Queries (50 warmup + 500 real)

	Query mixes per hour (1 client)	Query mixes per hour (4 clients)	Query mixes per hour (8 clients)	Query mixes per hour (12 clients)
DataWarp	271	320	320	312
Cached DataWarp	3,077	6,171	6,045	6,090
Lustre (direct)	1,627	2,577	2,680	2,673
Lustre (mapped)	1,943	3,015	3,002	2,999

Why plain DataWarp performances so bad?

- If not enabled with: **`client_cache=yes`** DW lacks any caching & buffering mechanism…
- Synthetic data creation phase reads parameter files `titlewords.txt` and `givennames.txt` **one byte at a time!**
- Seems the other inefficiencies and stalls have similar origin
- We could blame equally DataWarp & Java.
- Beware! Although many workloads can benefit from client-side caching because it can reduce the frequency and necessity of network operations (e.g. Java I/O), others will be negatively affected. **In some cases (e.g., many compute nodes modifying a specific file simultaneously with this access mode) data corruption can occur.**

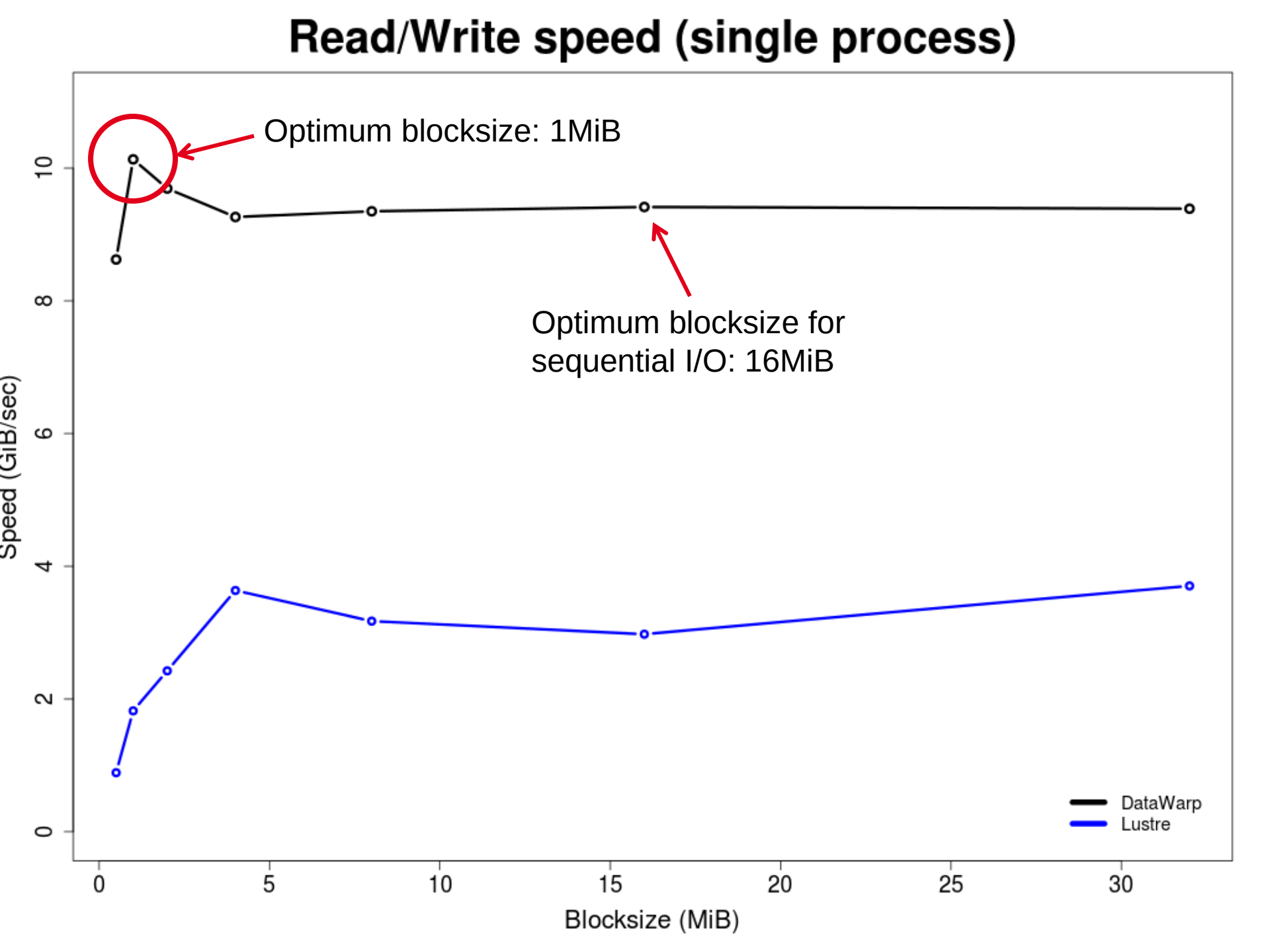
Random I/O using Vdbench

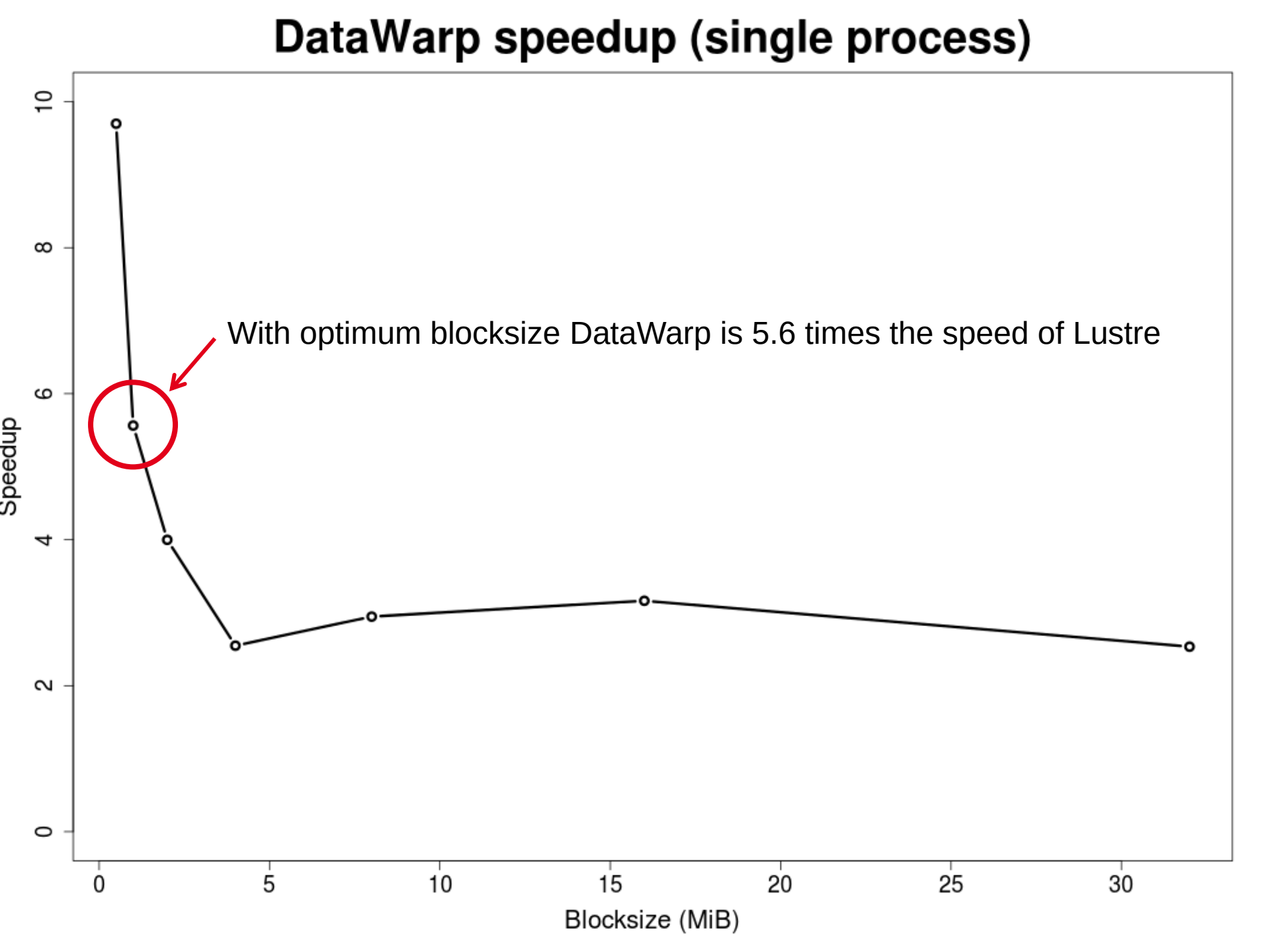
Vdbench

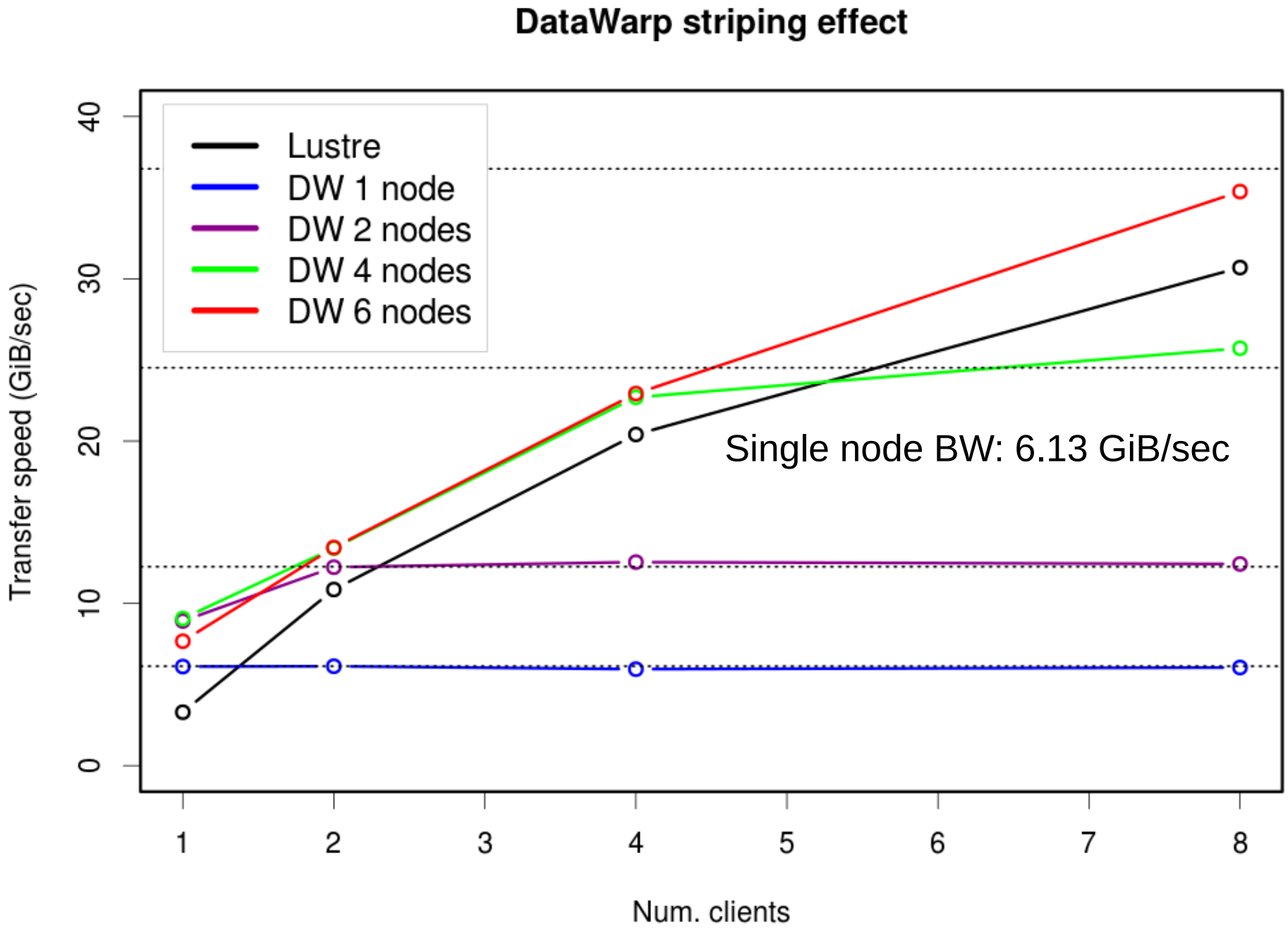
Vdbench is a command line utility specifically created to generate disk I/O workloads to be used for validating storage performance and storage data integrity.

http://www.oracle.com/technetwork/server-storage/vdbench-downloads-1901681.html

```
fsd=fsd1,anchor=$dir,depth=1,width=1,files=1000,size=256M

fwd=default,xfersize=$bs,fileio=random,fileselect=random,threads=12
fwd=fwd1,fsd=fsd1,operation=read
fwd=fwd2,fsd=fsd1,operation=write

rd=rd1,fwd=fwd*,fwdrate=max,format=yes,elapsed=720,interval=60, ↵
warmup=120
```


Read/Write speed (single process)
Optimum blocksize: 1MiB
Optimum blocksize for
sequential I/O: 16MiB
Speed (GiB/sec)
0
2
4
6
8
10
0
5
10
15
20
25
30
Blocksize (MiB)
DataWarp
Lustre


DataWarp speedup (single process)
With optimum blocksize DataWarp is 5.6 times the speed of Lustre
Speedup
0
2
4
6
8
10
0
5
10
15
20
25
30
Blocksize (MiB)


DataWarp striping effect
Lustre
DW 1 node
DW 2 nodes
DW 4 nodes
DW 6 nodes
Single node BW: 6.13 GiB/sec
Transfer speed (GiB/sec)
0
10
20
30
40
1
2
3
4
5
6
7
8
Num. clients

Lessons learned and open questions

Lessons learned (if you want DW speed)

1. Don’t expect too much…

   Normally I/O is a fraction of the total running time

2. Use applications with well buffered I/O

   Or set: **`client_cache=yes`** after checking its usage and limitations

3. Use correct I/O blocksize

   If possible change the I/O blocksize in the code

4. Use well parallelized I/O distributing the same number of tasks on more computing nodes.

   Simply it is not possible to saturate the Burst Buffer bandwidth using a single process on a single node

5. Enable striping by allocating more space than needed to have fragments on different SSD

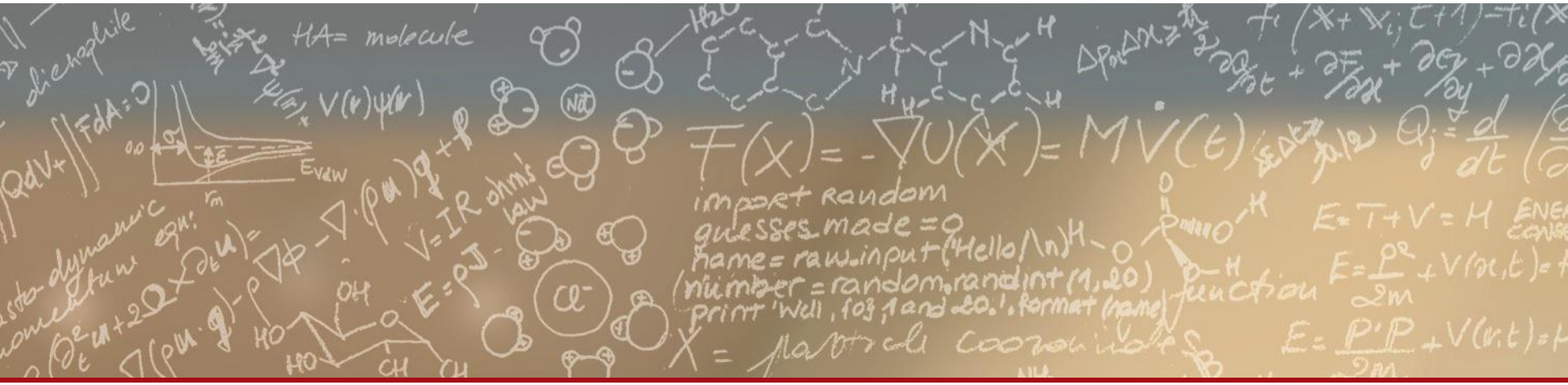
Open questions

- What happens if I have million of files?
  (Hello QuantumExpresso…)
- What if two users create two permanent instances with the same name?
- How could I protect a permanent instance?
- How to impose rules for fair use of the resource?

Conclusion

Conclusion

- Start experimenting, collect experiences, share results
- Set the correct expectations (BTW, the testing has not covered every possible situation)
- CUG 2016 paper *"Architecture and Design of Cray DataWarp"*
  - Note that it describes things not yet implemented in DataWarp
- User guide (UPD2 is the CLE currently installed on Daint): https://pubs.cray.com/content/S-2558/CLE%206.0.UP02/xctm-series-datawarptm-user-guide-cle-60up02-s-2558
- Other DW documents: https://pubs.cray.com/browse/datawarp/software



Thank you for your attention.