unifyFS Tutorial

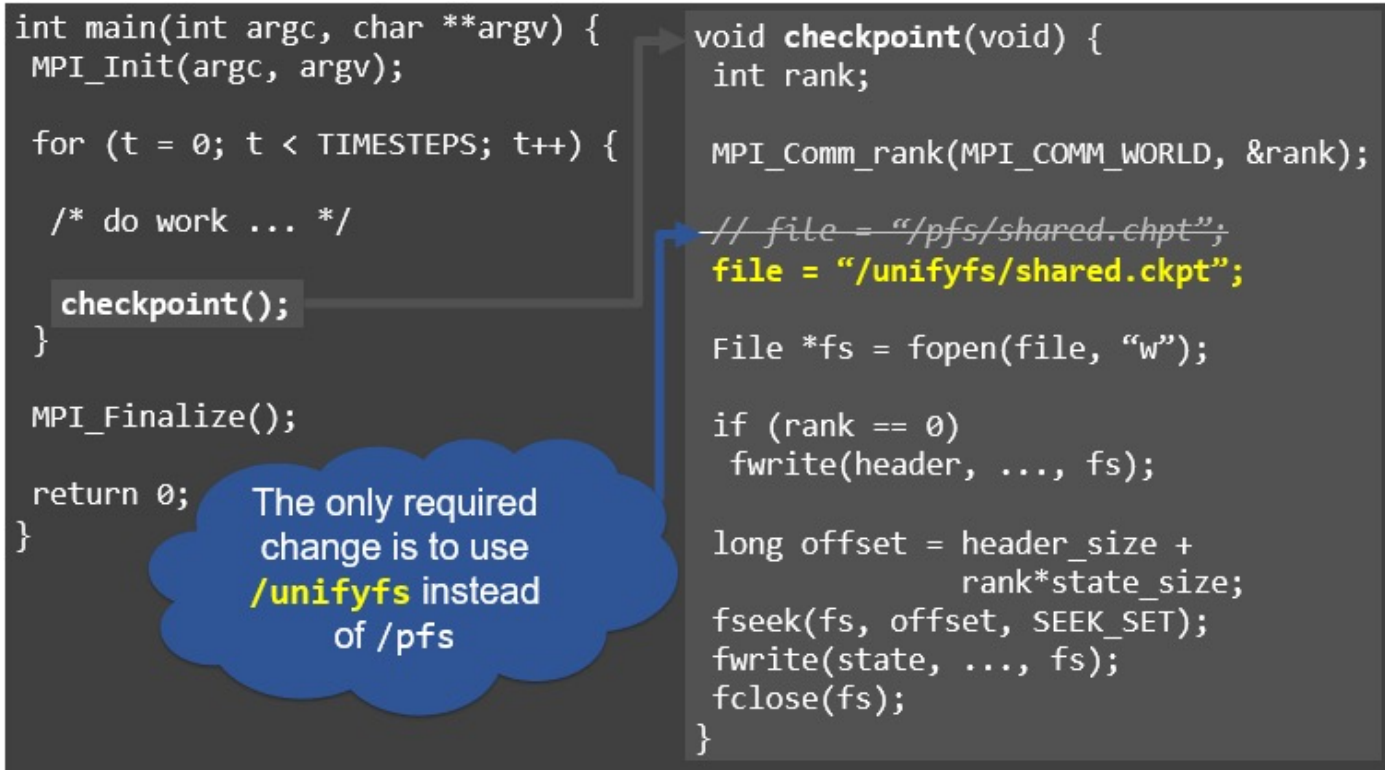
ECP 6th Annual Meeting
May 4th (be with you), 2022

LLNL: KATHRYN MOHROR (PI), ADAM MOODY, **CAMERON STANAVIGE**

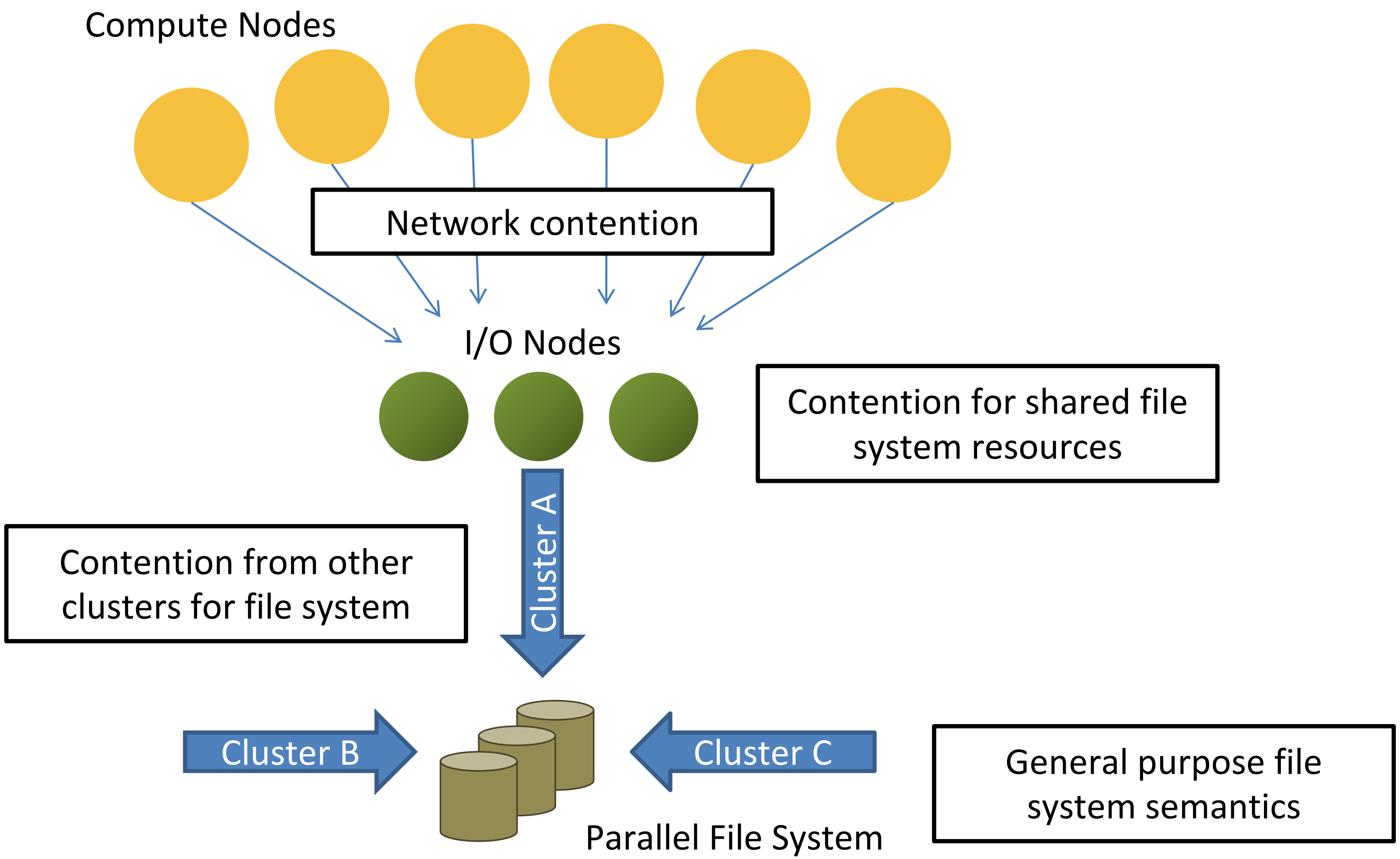
ORNL: SARP ORAL (ORNL-PI), SWEN BOEHM, **MICHAEL BRIM**, SEUNG-HWAN LIM, ROSS MILLER

What is UnifyFS?

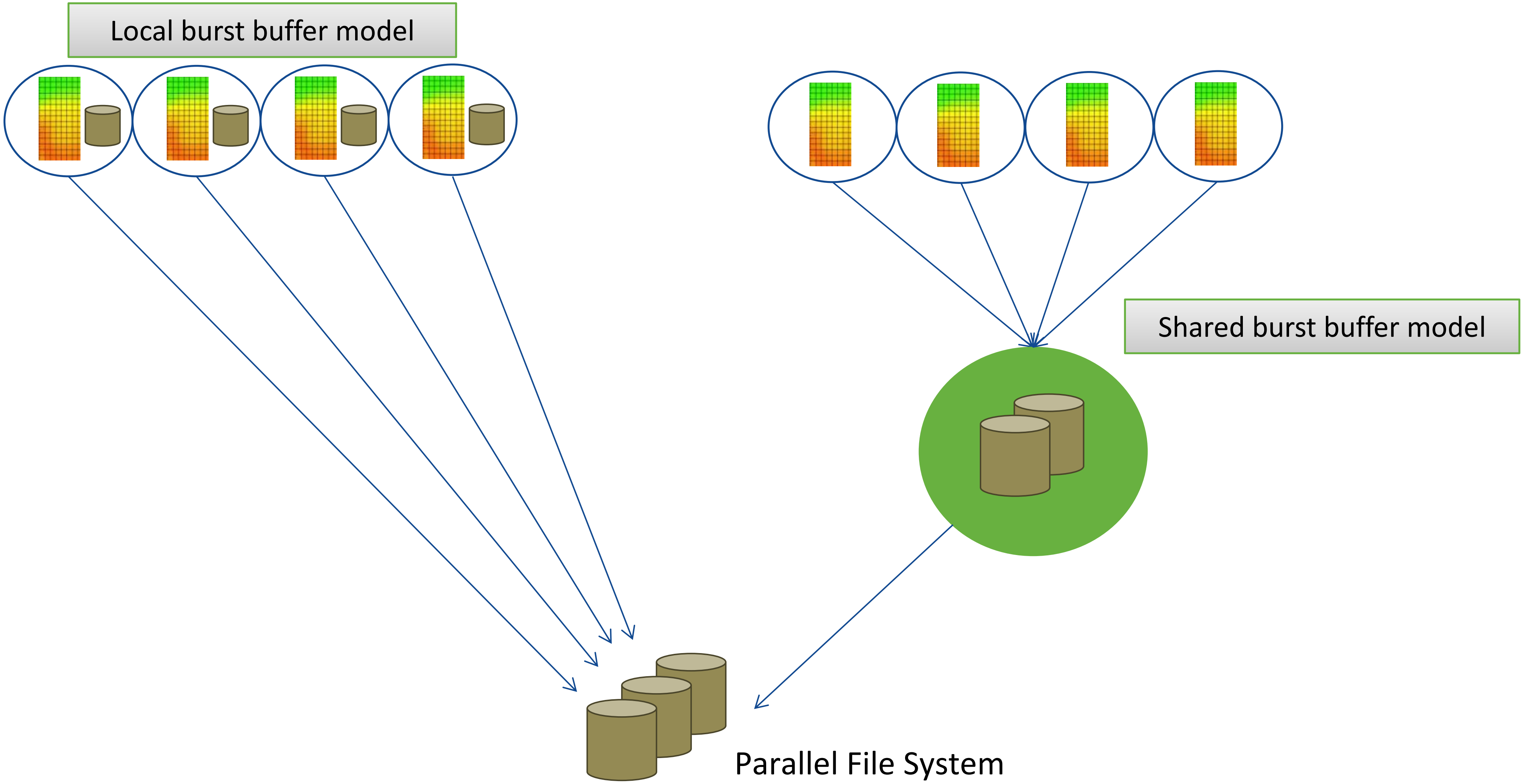
- An ephemeral, user-level shared file system for burst buffers
- Our goal is to make using burst buffers on exascale systems as *easy* as writing to the parallel file system and orders of magnitude *faster*



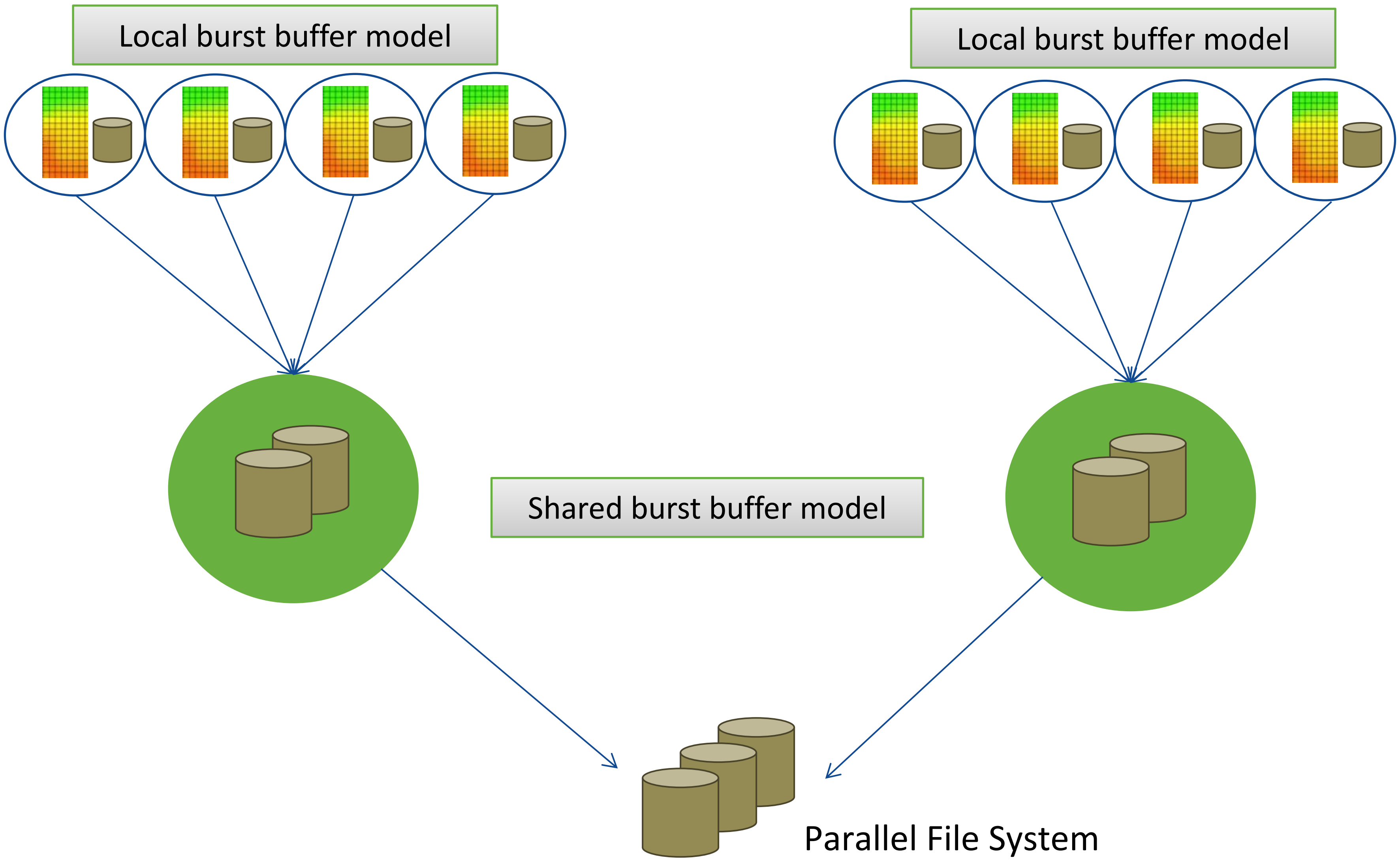
Writing data to the parallel file system is expensive



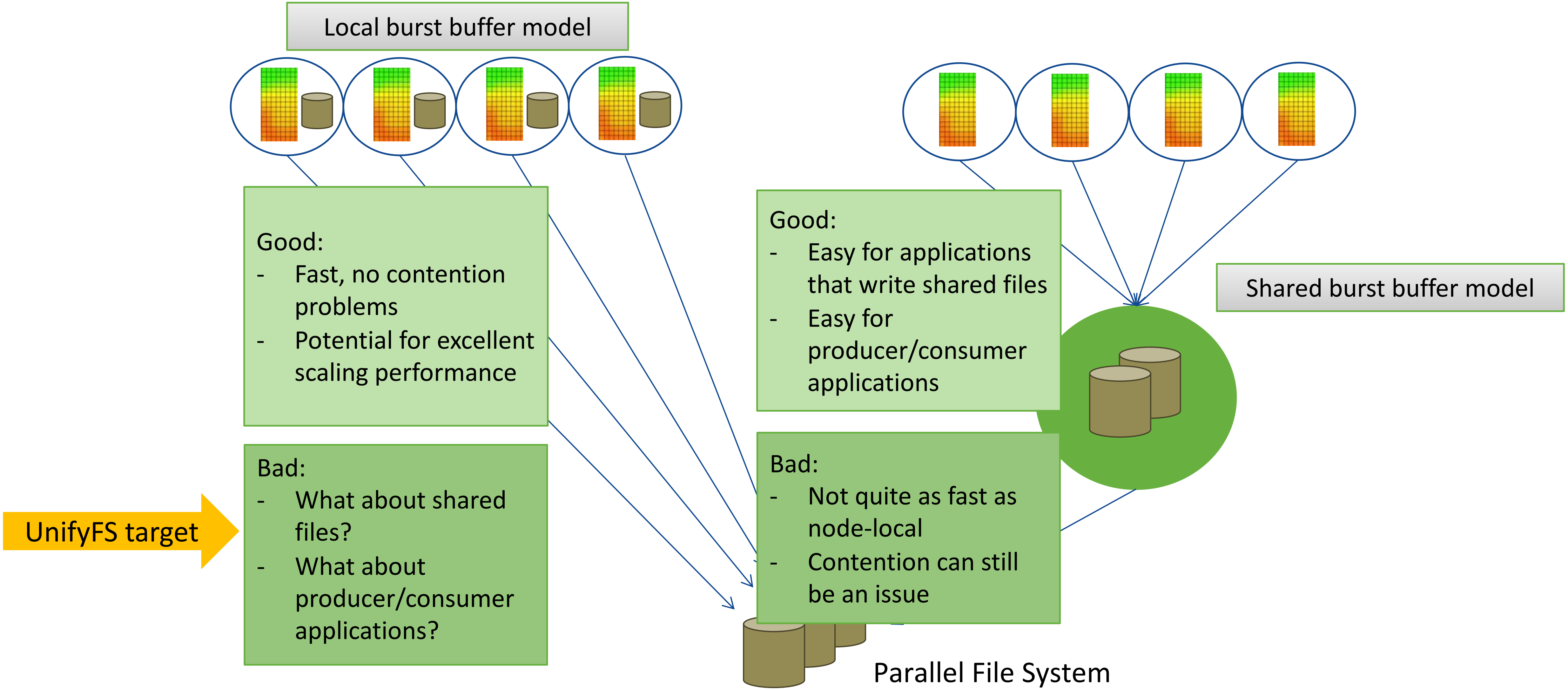
HPC Storage is becoming more complex



HPC Storage is becoming more complex



HPC Storage is becoming more complex



Local burst buffer model

Good:
- Fast, no contention problems
- Potential for excellent scaling performance

Bad:
- What about shared files?
- What about producer/consumer applications?

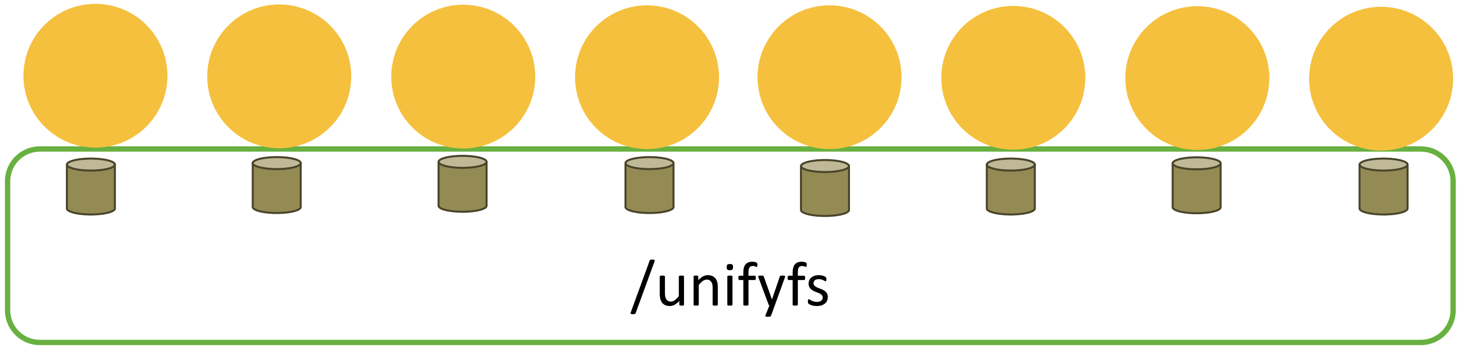
Shared burst buffer model

Good:
- Easy for applications that write shared files
- Easy for producer/consumer applications

Bad:
- Not quite as fast as node-local
- Contention can still be an issue

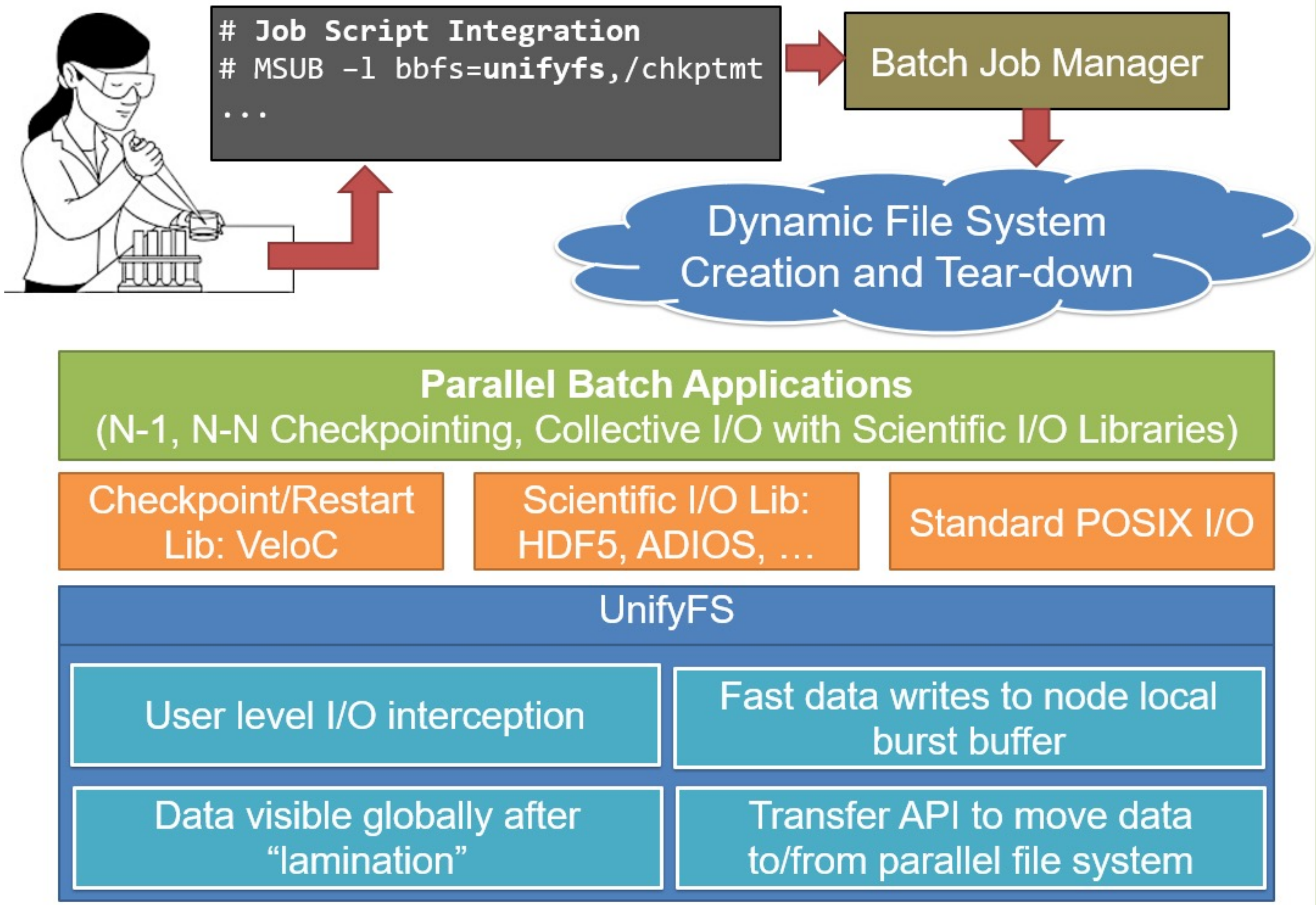
UnifyFS makes sharing files on node-local storage easy and fast

- **Problem:** Sharing files on node-local storage is not natively supported



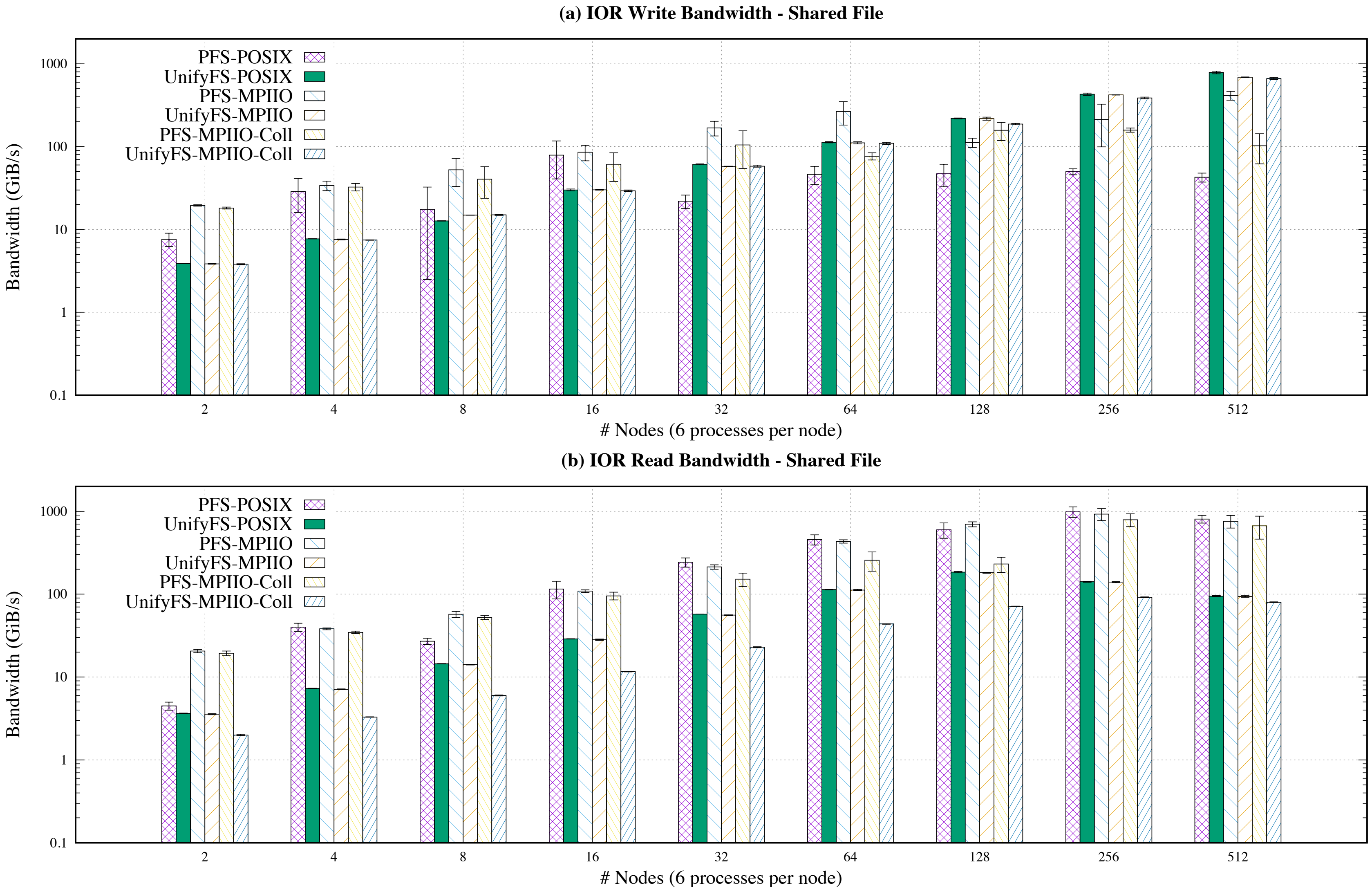
- UnifyFS makes sharing files **easy**
  - UnifyFS presents a shared namespace across distributed, independent storage devices
  - Used directly by applications or indirectly via higher level libraries like VeloC, MPI-IO, HDF5, PnetCDF, ADIOS, etc.
- UnifyFS is **fast**
  - Tailored for specific HPC workloads, e.g., checkpoint/restart, visualization output
  - Each UnifyFS instance exists only within a single job, no I/O contention with other jobs on the system
  - UnifyFS can use a combination of memory-backed and file-backed local storage

UnifyFS is designed to work completely in user space for a single job



UnifyFS targets local burst buffers because they are fast and scalable

- IOR v3.3 shared-file scaling on OLCF Summit
- UnifyFS (v1.0c)
  - All write data stored in NVMe (not using memory storage)
    - NVMe provides peak 2 GiB/s write and 5 GiB/s read per node
  - Write performance scaling well
    - up to 128 nodes, follows the the cumulative theoretical throughput of the node-local burst buffers
    - fairly consistent performance regardless of I/O method
  - Read performance (without metadata caching) scales less well
- Alpine parallel file system (PFS) performance is highly variable due to contention
  - MPI-IO has better write scaling performance than POSIX-IO
  - GPFS read caching works well

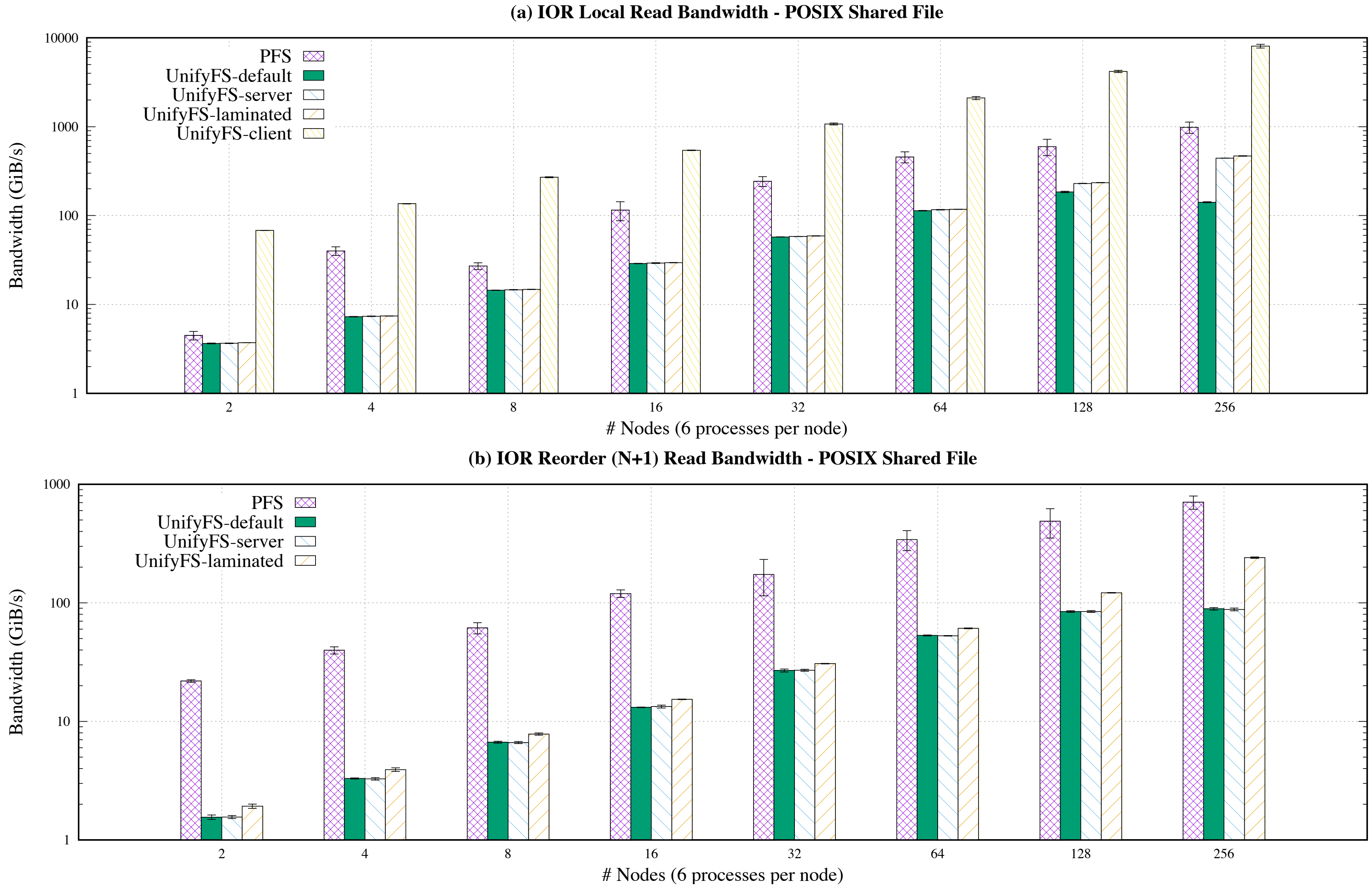


UnifyFS offers customizable file system semantics to meet varied application requirements

- By default, UnifyFS makes simplifying assumptions about how you access your data
  - **Assumptions meet common use cases for HPC I/O: checkpointing, output, producer/consumer**
  - I/O occurs in phases (except in limited circumstances, e.g., reads by a process of the data it wrote)
  - No two processes write to the same byte/offset concurrently
  - Without explicit synchronization, processes may not see updates written by processes on another node
  - Go here for more information: https://unifyfs.readthedocs.io/en/latest/assumptions.html
- The default semantics are compatible with MPI-IO and HDF5 parallel-independent I/O
  - For POSIX-IO or HDF5 parallel-collective I/O semantics, enable ”`client.write_sync`” mode
- Once you are done modifying a file, you may initiate “lamination”
  - The lamination process renders your file read-only and synchronizes file metadata across nodes in your job
  - Now any process on any node can read the final state of the file

UnifyFS customizable behavior can boost read performance

- IOR v3.3 shared-file scaling on OLCF Summit
- UnifyFS (v1.0c)
  - All write data stored in NVMe
- Four extent metadata caching configurations
  1. **default** (no metadata caching)
  2. default with **lamination**
  3. **server**-local metadata caching
  4. **client**-local metadata caching



Can I use UnifyFS if I use an I/O library?

- Yes! UnifyFS works with HDF5 I/O as well as other I/O libraries (e.g., MPI-IO)
  - We are partnered with HDF5 in ECP ExaIO so we test it the most

```
int main(int argc, char* argv[]) {
  MPI_Init(argc, argv);

  for(int t = 0; t < TIMESTEPS; t++)
  {
    /* ... Do work ... */


        checkpoint(dset_data);
  }

  MPI_Finalize();
  return 0;
}
```

```
void checkpoint(dset_data) {
  int rank; char file[256];
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  sprintf(file, “/lustre/shared.ckpt”);

  file_id = H5Fopen(file, …);
  dset_id = H5Dopen2(file_id, “/dset”, …);

  H5DWrite(dset_id, …, dset_data);
  H5Dclose(dset_id);
  H5Fclose(file_id);
  return;
}
```

Can I use UnifyFS if I use an I/O library?

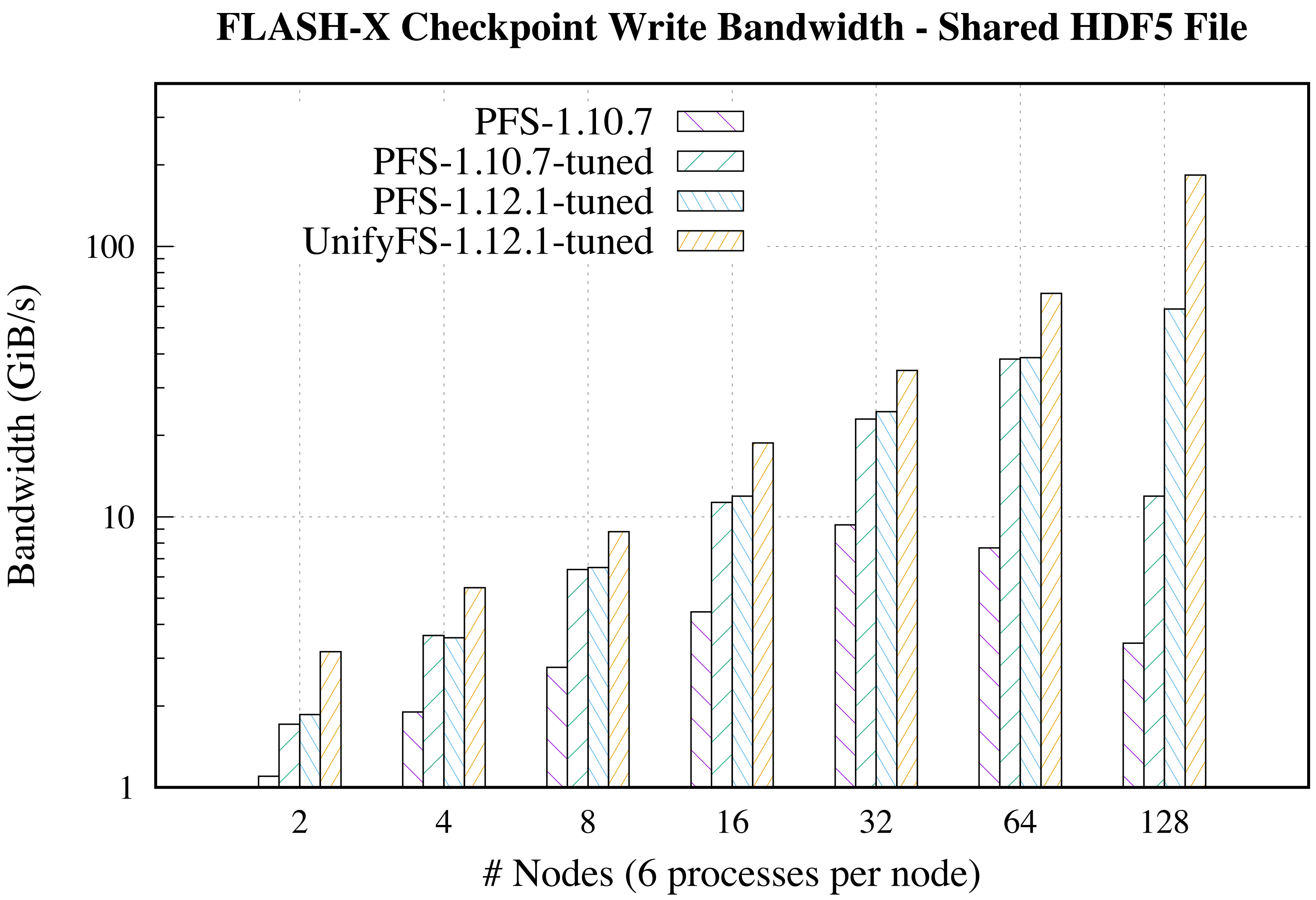
- Yes! UnifyFS works with HDF5 I/O as well as other I/O libraries (e.g., MPI-IO)
  - We are partnered with HDF5 in ECP ExaIO so we test it the most
- Build and run your application with UnifyFS, change the file path(s)

```
int main(int argc, char* argv[]) {
  MPI_Init(argc, argv);

  for(int t = 0; t < TIMESTEPS; t++)
  {
    /* ... Do work ... */


        checkpoint(dset_data);
  }

  MPI_Finalize();
  return 0;
}
```

```
void checkpoint(dset_data) {
  int rank; char file[256];
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  sprintf(file, "/unifyfs/shared.ckpt");

  file_id = H5Fopen(file, …);
  dset_id = H5Dopen2(file_id, "/dset", …);

  H5DWrite(dset_id, …, dset_data);
  H5Dclose(dset_id);
  H5Fclose(file_id);
  return;
}
```

HDF5 Example Application: FLASH-IO on OLCF Summit

- FLASH-X Astrophysics code
  - https://flash-x.org/
  - FLASH-IO benchmark configuration writes checkpoint and plot files
    - ~ 72 GB of checkpoint data per node
    - ~ 220 MB of plot data per node
- "PFS" is Parallel File System
  - OLCF Alpine (IBM Spectrum Scale FS)
- UnifyFS uses only node-local NVMe devices (2 GiB/s peak write bandwidth)
- HDF5 versions
  - v1.10.7 is Summit default
  - v1.12.1 includes recent improvements
- "tuned" application includes two optimizations:
  1. a good MPI-IO configuration for Alpine
  2. elimination of a redundant `H5Fflush()` call per write operation



unifyFS

Can I use UnifyFS with VeloC?

** Initial testing done at the time of this tutorial. More evaluation needed for production use.

- Yes! UnifyFS works with the VeloC checkpointing library**

```
int main(int argc, char* argv[]) {
  MPI_Init(argc, argv);
  VeloC_Init();
  for(int t = 0; t < TIMESTEPS; t++)
  {
    /* ... Do work ... */

     if (VeloC_Checkpoint_need())
        checkpoint(dset_data);
  }
  VeloC_Finalize();
  MPI_Finalize();
  return 0;
}
```

```
void checkpoint(dset_data) {
  int rank; char file[256];
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  VeloC_Checkpoint_begin();
  sprintf(file, “/lustre/shared.ckpt”);

  VeloC_Route_file(file, new_file);
  file_id = H5Fopen(new_file, …);
  dset_id = H5Dopen2(file_id, “/dset”, …);

  H5DWrite(dset_id, …, dset_data);
  H5Dclose(dset_id);
  H5Fclose(file_id);
  VeloC_Checkpoint_end();
  return;
}
```

Can I use UnifyFS with VeloC?

** Initial testing done at the time of this tutorial. More evaluation needed for production use.

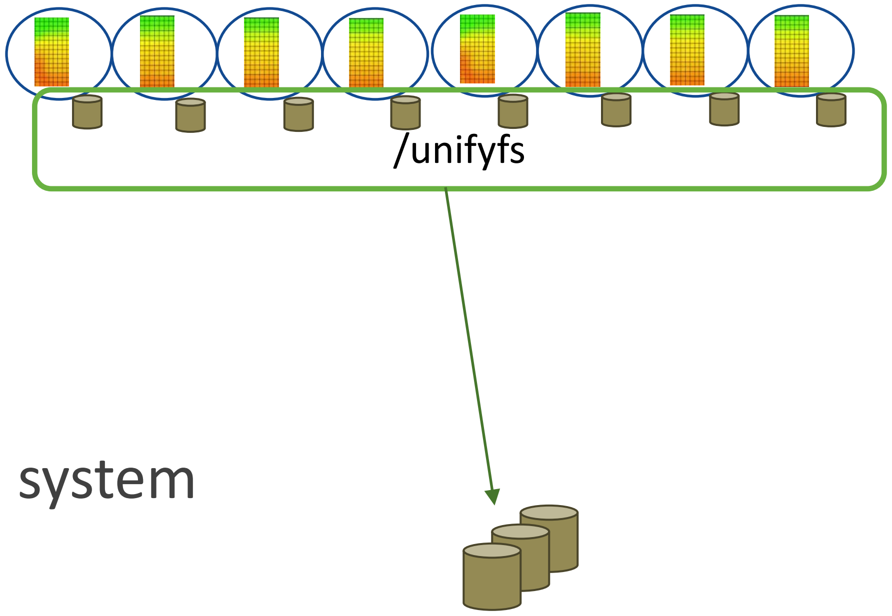
- Yes! UnifyFS works with the VeloC checkpointing library**
- Build and run your app with UnifyFS, change the path that VeloC uses for 1st level store

```
int main(int argc, char* argv[]) {
  MPI_Init(argc, argv);
  VeloC_Init();
  for(int t = 0; t < TIMESTEPS; t++)
  {
    /* ... Do work ... */

     if (VeloC_Checkpoint_need())
        checkpoint(dset_data);
  }
  VeloC_Finalize();
  MPI_Finalize();
  return 0;
}
```

```
void checkpoint(dset_data) {
  int rank; char file[256];
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  VeloC_Checkpoint_begin();
  sprintf(file, “/lustre/shared.ckpt”);

  VeloC_Route_file(file, new_file);
  file_id = H5Fopen(new_file, …);
  dset_id = H5Dopen2(file_id, “/dset”, …);

  H5DWrite(dset_id, …, dset_data);
  H5Dclose(dset_id);
  H5Fclose(file_id);
  VeloC_Checkpoint_end();
  return;
}
```

Walkthrough Tutorial

- get and build UnifyFS
- build your application to use UnifyFS
- set up your application environment to use UnifyFS
- run your application with UnifyFS
- move your data between UnifyFS and the parallel file system



Tutorial: How do I get and build UnifyFS?

OLCF Summit

- Recent pre-1.0-release versions are already installed for a few compilers

```
$ module use /sw/summit/unifyfs/modulefiles

$ module avail unifyfs

$ module load unifyfs/<version>/mpi-mount-<compiler>
```

Got Spack?

```
$ spack install unifyfs

$ spack load unifyfs
```

unifyFS

Tutorial: How do I get and build UnifyFS?

UnifyFS variants for Spack installation

```
$ spack info unifyfs
⋮
Variants:
    Name [Default]     Allowed values    Description
    ===============    ==============    ==============================

    auto-mount [True]  True, False       Enable automatic mount/unmount in
                                         MPI_Init/Finalize
    fortran [False]    True, False       Build with gfortran support
    pmi [False]        True, False       Enable PMI2 build options
    pmix [False]       True, False       Enable PMIx build options
    spath [True]       True, False       Normalize relative paths
```

Tutorial: How do I get and build UnifyFS?

Installing UnifyFS with a variant using Spack

```
$ spack install unifyfs +fortran ~auto-mount

$ spack load unifyfs
```

Tutorial: How do I get UnifyFS without Spack?

- Not using Spack?
- UnifyFS can be found on GitHub
  - https://github.com/LLNL/UnifyFS

```
$ git clone https://github.com/LLNL/UnifyFS.git
```

Tutorial: How do I build UnifyFS without Spack?

- Build and install UnifyFS's dependencies
  - GOTCHA: https://github.com/LLNL/GOTCHA
  - Margo: https://github.com/mochi-hpc/mochi-margo
    - Mercury: https://github.com/mercury-hpc/mercury
      - libfabric: https://github.com/ofiwg/libfabric and/or bmi: https://github.com/radix-io/bmi/
    - Argobots: https://github.com/pmodels/argobots
    - JSON-C: https://github.com/json-c/json-c
  - SPath: https://github.com/ecp-veloc/spath
  - Run our bootstrap script to automatically download and install our dependencies

```
$ cd UnifyFS
$ ./bootstrap
```

Tutorial: How do I build UnifyFS without Spack?

- Then build and install UnifyFS

```
$ export PKG_CONFIG_PATH=$INSTALL_DIR/lib/pkgconfig:
  $INSTALL_DIR/lib64/pkgconfig:$PKG_CONFIG_PATH
$ export LD_LIBRARY_PATH=$INSTALL_DIR/lib:$INSTALL_DIR/lib64:$LD_LIBRARY_PATH

$ ./configure --prefix=$INSTALL_DIR --enable-mpi-mount --enable-pmix
  --enable-fortran --with-gotcha=$INSTALL_DIR --with-spath=$INSTALL_DIR

$ make
$ make install
```

Tutorial: How do I modify my MPI application for UnifyFS?

- Example MPI application without UnifyFS (using native file system)
  - Simple application that writes "Hello World" to Lustre at `/lustre/dset.txt`

```
int main(int argc, char * argv[]) {
    FILE *fp;
     // program initialization
     // MPI setup

     // perform I/O
    fp = fopen("/lustre/dset.txt", "w");
    fprintf(fp, "Hello World! I'm rank %d", rank);
    fclose(fp);

     // clean up

    return 0;
}
```

Tutorial: How do I modify my MPI application for UnifyFS?

- Example MPI application
  - To use UnifyFS, change the file path(s) to point to the UnifyFS mount point at `/unifyfs`

```
int main(int argc, char * argv[]) {
    FILE *fp;
     // program initialization
     // MPI setup

     // perform I/O
    fp = fopen("/unifyfs/dset.txt", "w");
    fprintf(fp, "Hello World! I'm rank %d", rank);
    fclose(fp);

    // "laminate" the file to indicate to UnifyFS you
       are done modifying this file
    chmod("/unifyfs/dset.txt", 0444);

     // clean up
    return 0;
}
```

Tutorial: How do I modify my MPI application for UnifyFS?

- Example MPI application
  - To use UnifyFS, change the file path(s) to point to the UnifyFS mount point at /unifyfs

```
int main(int argc, char * argv[]) {
    FILE *fp;
     // program initialization
     // MPI setup

     // perform I/O
    fp = fopen("/unifyfs/dset.txt", "w");
    fprintf(fp, "Hello World! I'm rank %d"
    fclose(fp);

    // "laminate" the file to indicate to
       are done modifying this file
    chmod("/unifyfs/dset.txt", 0444);

     // clean up
    return 0;
}
```

- Current support for lamination:
  - chmod() file to read-only
  - unifyfs_laminate() API call
  - unifyfs_laminate command-line utility
- Future lamination methods we are considering as mount options:
  - Laminate on close()
  - Laminate on unmount

Tutorial: How do I modify my serial application for UnifyFS?

- Example serial (no MPI) application
  - To use UnifyFS, mount/unmount via API calls and change file path(s) to point to the mount point

```
#include <unifyfs.h>
int main(int argc, char * argv[]) {
    FILE *fp;
     // program initialization
    unifyfs_mount("/unifyfs");

     // perform I/O
    fp = fopen("/unifyfs/dset.txt", "w");
    fprintf(fp, "Hello World!");
    fclose(fp);
    chmod("/unifyfs/dset.txt", 0444);

     // clean up
    unifyfs_unmount();
    return 0;
}
```

Tutorial: How does UnifyFS intercept I/O calls?

- Static Linking
  - To intercept I/O calls using a static link you'll need to add flags to your link line.
  - UnifyFS installs a `unifyfs-config` script that returns those flags:

```
$ mpicc -o hello hello.c `<unifyfs>/bin/unifyfs-config \
  --pre-ld-flags` `<unifyfs>/bin/unifyfs-config --post-ld-flags`
```

  - Won't see syscalls in MPI implementations that dynamically load MPI-IO libraries

- Dynamic Linking (Recommended method)
  - We use the LLNL GOTCHA library for dynamic interception

```
$ mpicc -o hello hello.c \
    -L<unifyfs_install>/lib -lunifyfs_mpi_gotcha
```

Note: <unifyfs_install> is the install path of UnifyFS. With spack "spack location –i unifyfs" will give you the path.

Tutorial: What happens when I run my code **without** UnifyFS?

1. user application calls “fopen”

```
fopen(“/<path>/dset.txt”)
```

```
fopen()
```

(GOT/PLT)

Symbol Name	Symbol Location
fopen	fopen@glibc
fprintf	fprintf@glibc
fclose	fclose@glibc

Tutorial: What happens when I run my code **without** UnifyFS?

1. user application calls "fopen"

```
fopen("/<path>/dset.txt")
```

2. Address to glibc fopen is looked up via GOT/PLT

(GOT/PLT)

Symbol Name	Symbol Location
fopen	fopen@glibc
fprintf	fprintf@glibc
fclose	fclose@glibc

```
fopen()
```

Tutorial: What happens when I run my code **without** UnifyFS?

1. user application calls "fopen"

```
fopen("/<path>/dset.txt")
```

2. Address to glibc fopen is looked up via GOT/PLT

(GOT/PLT)

Symbol Name	Symbol Location
fopen	fopen@glibc
fprintf	fprintf@glibc
fclose	fclose@glibc

3. fopen@glibc is executed

```
fopen()
```

Tutorial: What happens when I run my code **with** UnifyFS and Gotcha?

1. user application calls "fopen"

```
fopen("/<path>/dset.txt")
```

(libunifyfs_gotcha)

```
UNIFYFS_WRAP(fopen)
{
 if(intercept(path){
   …

  //using UnifyFS

   …
  } else {
   __real_fopen(path)
  }
}
```

(glibc)

```
fopen()
```

(GOT/PLT)

Symbol Name	Symbol Location
fopen	fopen@unifyfs_gotcha
fprintf	fprintf@unifyfs_gotcha
fclose	fclose@unifyfs_gotcha

Tutorial: What happens when I run my code **with** UnifyFS and Gotcha?

1. user application calls “fopen”

```
fopen(“/<path>/dset.txt”)
```

2. Address to glibc fopen is rewritten to UnifyFS’s “fopen” in GOT/PLT

(libunifyfs_gotcha)

```
UNIFYFS_WRAP(fopen)
{
 if(intercept(path){
   …

  //using UnifyFS

   …
  } else {
   __real_fopen(path)
  }
}
```

(glibc)

```
fopen()
```

(GOT/PLT)

Symbol Name	Symbol Location
fopen	fopen@unifyfs_gotcha
fprintf	fprintf@unifyfs_gotcha
fclose	fclose@unifyfs_gotcha

Tutorial: What happens when I run my code **with** UnifyFS and Gotcha?

1. user application calls "fopen"

```
fopen("/<path>/dset.txt")
```

2. Address to glibc fopen is rewritten to UnifyFS's "fopen" in GOT/PLT

(GOT/PLT)

Symbol Name	Symbol Location
fopen	fopen@unifyfs_gotcha
fprintf	fprintf@unifyfs_gotcha
fclose	fclose@unifyfs_gotcha

3. fopen call is re-directed to UnifyFS library for processing

(libunifyfs_gotcha)

```
UNIFYFS_WRAP(fopen)
{
 if(intercept(path){
   …

  //using UnifyFS

   …
  } else {
   __real_fopen(path)
  }
}
```

(glibc)

```
fopen()
```

Tutorial: What happens when I run my code **with** UnifyFS and Gotcha?

1. user application calls “fopen”

```
fopen(“/<path>/dset.txt”)
```

2. Address to glibc fopen is rewritten to UnifyFS’s “fopen” in GOT/PLT

(GOT/PLT)

Symbol Name	Symbol Location
fopen	fopen@unifyfs_gotcha
fprintf	fprintf@unifyfs_gotcha
fclose	fclose@unifyfs_gotcha

(libunifyfs_gotcha)

```
UNIFYFS_WRAP(fopen)
{
 if(intercept(path){
   …

  //using UnifyFS

   …
  } else {
   __real_fopen(path)
  }
}
```

3. fopen call is re-directed to UnifyFS library for processing

(glibc)

```
fopen()
```

4. If path is not under UnifyFS, then the wrapper calls glibc fopen

Tutorial: How do I set up my code to run with UnifyFS?

- **UnifyFS provides the following ways to set configuration settings**:
  - Configuration file: `$INSTALL_PREFIX/etc/unifyfs/unifyfs.conf`
  - Environment variables
  - Command line options to `unifyfs start`
    - Available for a subset of config options
  - When defined via multiple methods, the priority order is as follows:
    - command line options, environment variables, and finally the configuration file.
- **Link to detailed breakdown of all UnifyFS configuration options: https://unifyfs.readthedocs.io/en/dev/configuration.html**

Tutorial: How do I set up my code to run with UnifyFS?

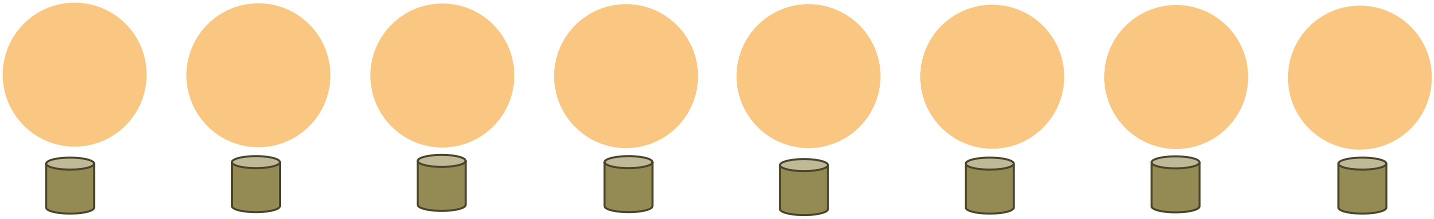
- Example configuration file using `unifyfs.conf`
  - Located in installation directory under `etc/unifyfs/` and in `extras/` directory in the source repository.

```
# unifyfs.conf
# SECTION: top-level configuration
[unifyfs]
mountpoint = /unifyfs ; (i.e., prefix path)
# SECTION: top-level configuration
[log]
dir  = /tmp ; log file directory path
verbosity  = 5 ; logging verbosity level (default: 0)
# SECTION: log-based I/O configuration (NOTE: values are per-client)
[logio]
shmem_size = 536870912 ; max size of data in shared memory data (default: 256MB)
spill_size = 2147482548 ; max size of data in spillover file (default: 1GiB)
spill_dir  = /mnt/ssd ; directory path for data spillover
```

Tutorial: How do I run my code with UnifyFS?

- Easiest method: Start & stop UnifyFS in your batch script

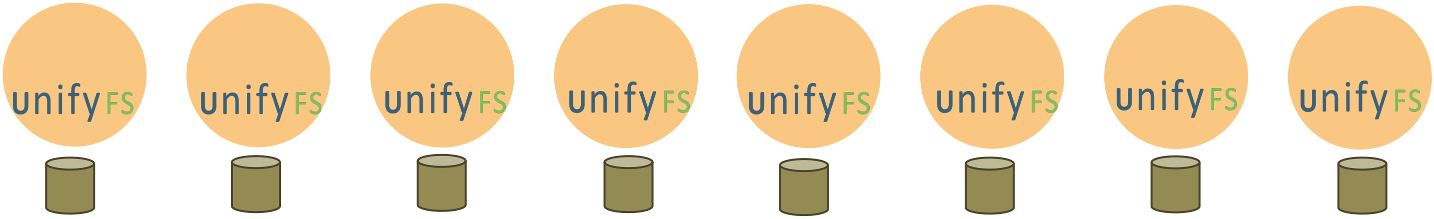
```
### allocate nodes and options for resource manager
#BSUB -nnodes 8 …
```



Tutorial: How do I run my code with UnifyFS?

- Easiest method: Start & stop UnifyFS in your batch script
  - Command ‘unifyfs start’ launches UnifyFS for your job and sets up the file system

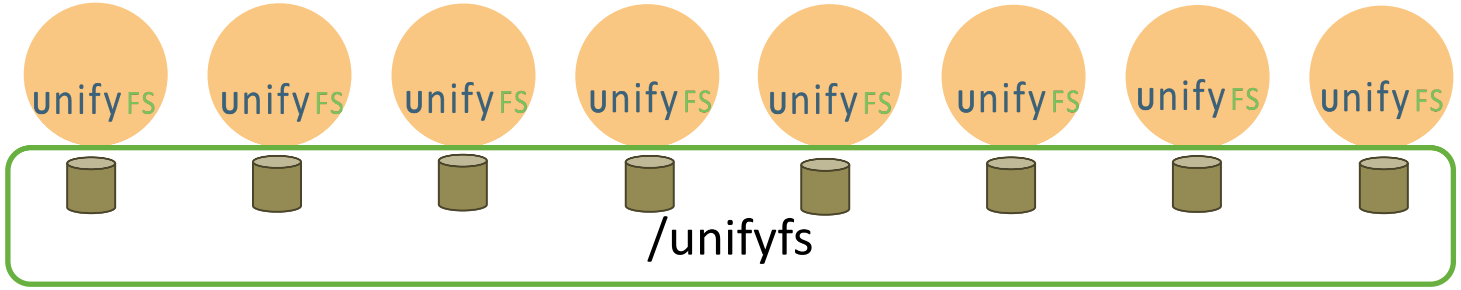
```
### allocate nodes and options for resource manager
#BSUB –nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path
```



Tutorial: How do I run my code with UnifyFS?

- Easiest method: Start & stop UnifyFS in your batch script
  - Command ‘unifyfs start’ launches UnifyFS for your job and sets up the file system

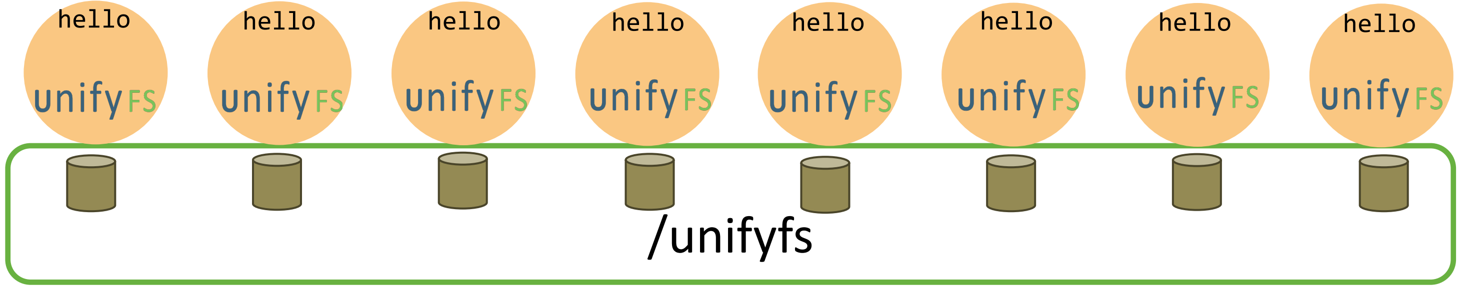
```
### allocate nodes and options for resource manager
#BSUB –nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path
```



Tutorial: How do I run my code with UnifyFS?

- Easiest method: Start & stop UnifyFS in your batch script
  - Command 'unifyfs start' launches UnifyFS for your job and sets up the file system
  - Run your command as usual and use the path /unifyfs to direct data to UnifyFS

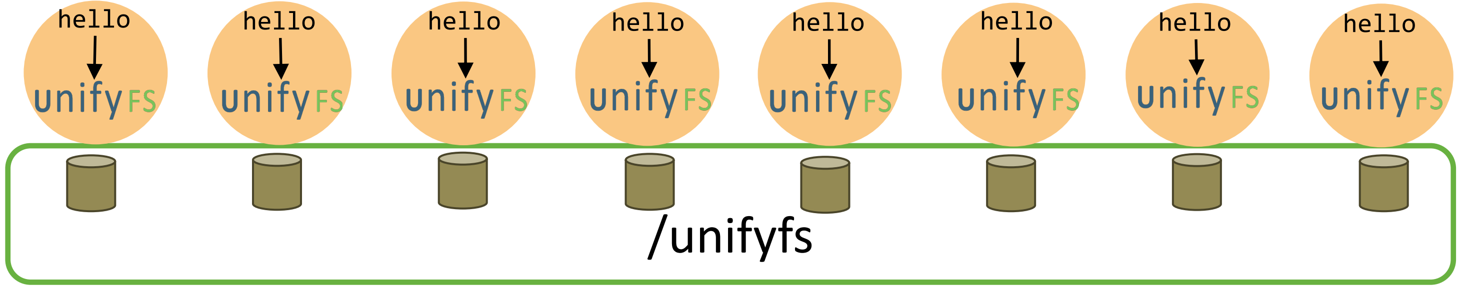
```
### allocate nodes and options for resource manager
#BSUB –nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path
jsrun –p 256 ./hello
```



Tutorial: How do I run my code with UnifyFS?

- Easiest method: Start & stop UnifyFS in your batch script
  - Command ‘unifyfs start’ launches UnifyFS for your job and sets up the file system
  - Run your command as usual and use the path /unifyfs to direct data to UnifyFS

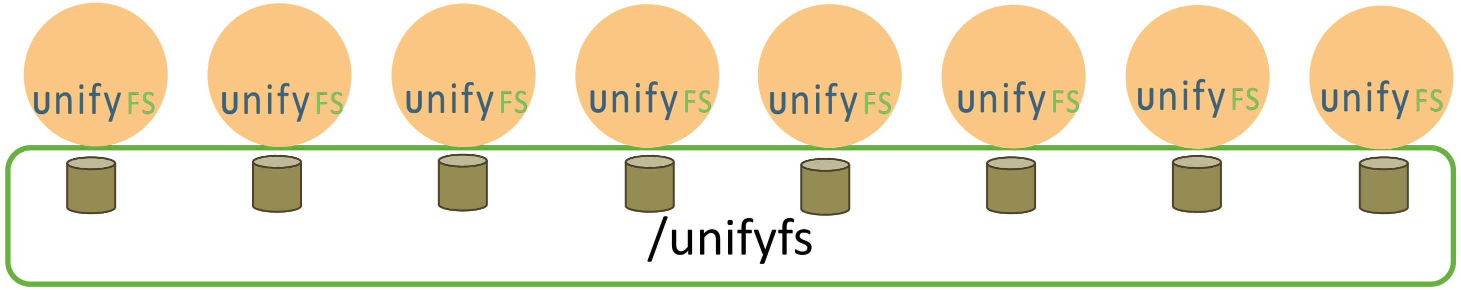
```
### allocate nodes and options for resource manager
#BSUB -nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path
jsrun -p 256 ./hello
```



Tutorial: How do I run my code with UnifyFS?

- Easiest method: Start & stop UnifyFS in your batch script
  - Command 'unifyfs start' launches UnifyFS for your job and sets up the file system
  - Run your command as usual and use the path /unifyfs to direct data to UnifyFS
  - Command 'unifyfs terminate' cleans up the UnifyFS file system and tears it down

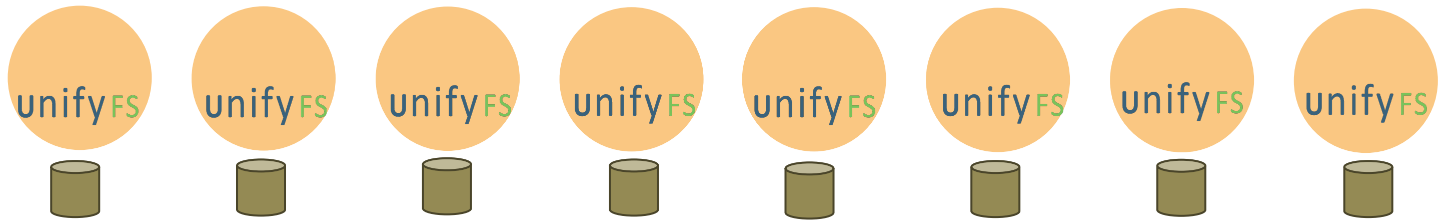
```
### allocate nodes and options for resource manager
#BSUB -nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path
jsrun -p 256 ./hello
unifyfs terminate
```



Tutorial: How do I run my code with UnifyFS?

- Easiest method: Start & stop UnifyFS in your batch script
  - Command ‘unifyfs start’ launches UnifyFS for your job and sets up the file system
  - Run your command as usual and use the path /unifyfs to direct data to UnifyFS
  - Command ‘unifyfs terminate’ cleans up the UnifyFS file system and tears it down

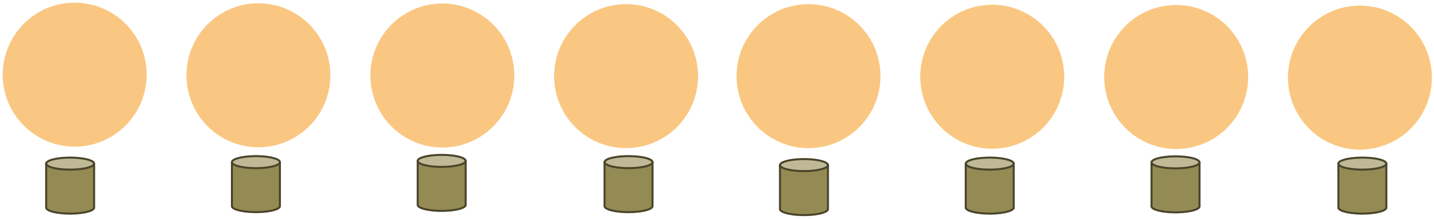
```
### allocate nodes and options for resource manager
#BSUB -nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path
jsrun -p 256 ./hello
unifyfs terminate
```



Tutorial: How do I run my code with UnifyFS?

- Easiest method: Start & stop UnifyFS in your batch script
  - Command 'unifyfs start' launches UnifyFS for your job and sets up the file system
  - Run your command as usual and use the path /unifyfs to direct data to UnifyFS
  - Command 'unifyfs terminate' cleans up the UnifyFS file system and tears it down

```
### allocate nodes and options for resource manager
#BSUB –nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path
jsrun –p 256 ./hello
unifyfs terminate
```



Tutorial: How do I move my data into and out of UnifyFS?

- Three ways to move data between UnifyFS and the parallel file system
- UnifyFS transfer API
  - `unifyfs_transfer_file_parallel(“/unifyfs/out.txt”, “/scratch/out.txt”);`
- UnifyFS transfer program
  - `jsrun –r1 unifyfs-stage --parallel $MY_MANIFEST_FILE`
- Stage in and out options with UnifyFS commands “unifyfs start” & “unifyfs terminate”
  - `unifyfs start --stage-in=$MY_INPUTS_MANIFEST_FILE`
  - `unifyfs terminate --stage-out=$MY_OUTPUTS_MANIFEST_FILE`

VerifyIO: Is my application compatible with UnifyFS?

- Recorder
  - Tracing framework that can capture I/O function calls at multiple levels of the I/O stack, including HDF5, MPI-IO, and POSIX I/O
  - Actively being developed by Chen Wang from UIUC
  - GitHub: https://github.com/uiuc-hpc/Recorder/

- VerifyIO
  - Recorder tool that takes application traces and determines whether I/O synchronization is correct based on the underlying file system semantics (e.g., posix, commit) and synchronization semantics (e.g., posix, MPI)
  - Use "commit" semantics to check compatibility with UnifyFS
  - GitHub: https://github.com/uiuc-hpc/Recorder/tree/pilgrim/tools/verifyio

We need you!

- Our goal is to provide **easy, portable,** and **fast** support for burst buffers for ECP applications
- We need early users
- v1.0 pre-releases available on Summit
- What features are most important to you
- Available on GitHub: https://github.com/LLNL/UnifyFS
  - Latest release: version 0.9.2 March 2021
  - Next release: version 1.0 (Summer 2022)
  - MIT license
- Documentation and user support
  - User Guide: http://unifyfs.readthedocs.io
  - ecp-unifyfs@exascaleproject.org
- VerifyIO
  - https://github.com/uiuc-hpc/Recorder/tree/pilgrim/tools/verifyio

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.