Accelerating your application I/O with UnifyFS

ATPESC 2020
July 31, 2020

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

ORNL: SARP ORAL (ORNL-PI), HYOGI SIM, FEIYI WANG, MIKE BRIM, SWEN BOEHM

NCSA: CELSO MENDES, CRAIG STEFFEN

LLNL-PRES-812833

What is UnifyFS?

- Simply put, it's a 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*

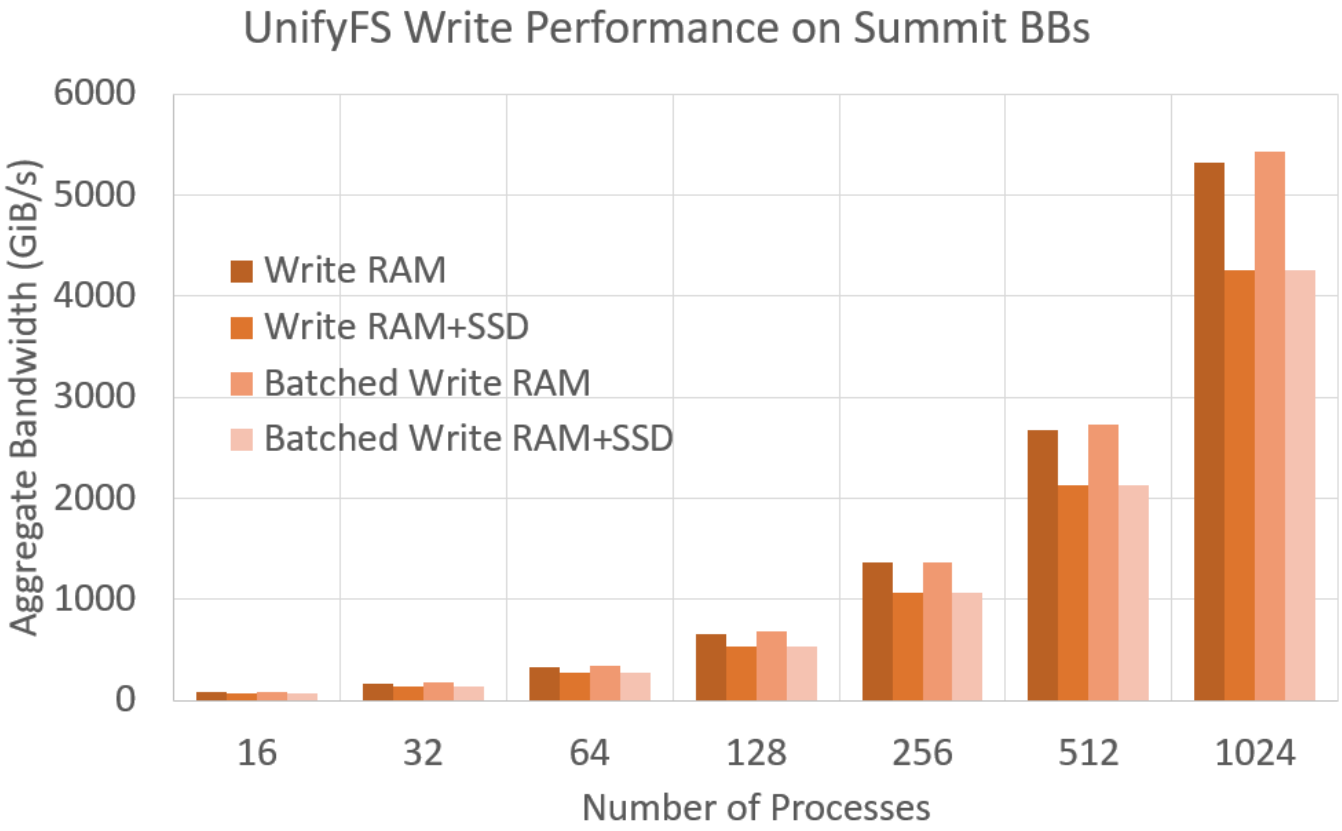
```
int main(int argc, char **argv) {
 MPI_Init(argc, argv);

 for (t = 0; t < TIMESTEPS; t++) {

  /* do work ... */

  checkpoint();
 }

 MPI_Finalize();

 return 0;
}
```

```
void checkpoint(void) {
 int rank;

 MPI_Comm_rank(MPI_COMM_WORLD, &rank);

 // file = "/pfs/shared.chpt";
 file = "/unifyfs/shared.ckpt";

 File *fs = fopen(file, "w");

 if (rank == 0)
  fwrite(header, ..., fs);

 long offset = header_size +
               rank*state_size;
 fseek(fs, offset, SEEK_SET);
 fwrite(state, ..., fs);
 fclose(fs);
}
```

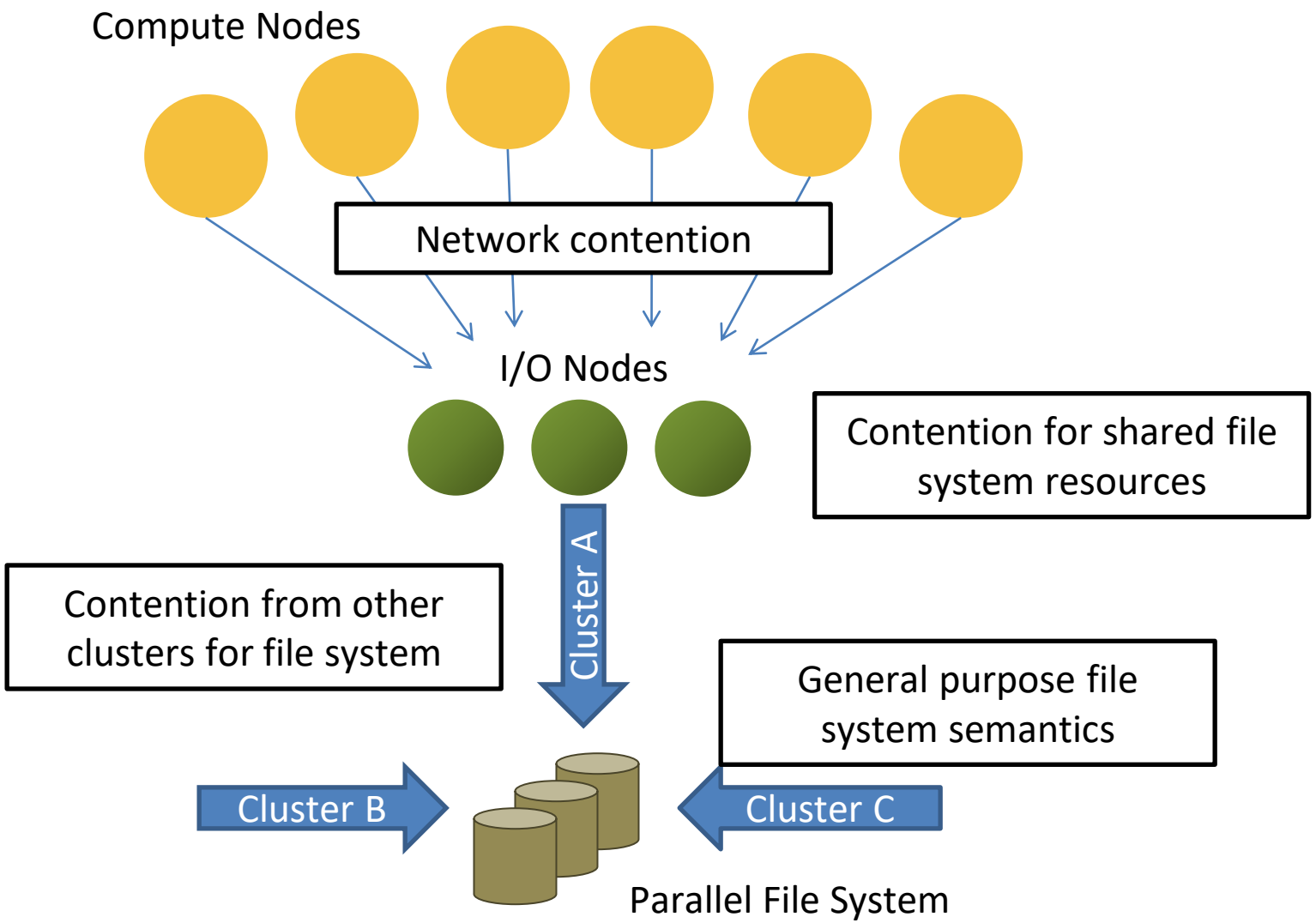
The only required change is to use **/unifyfs** instead of /pfs

What is UnifyFS?

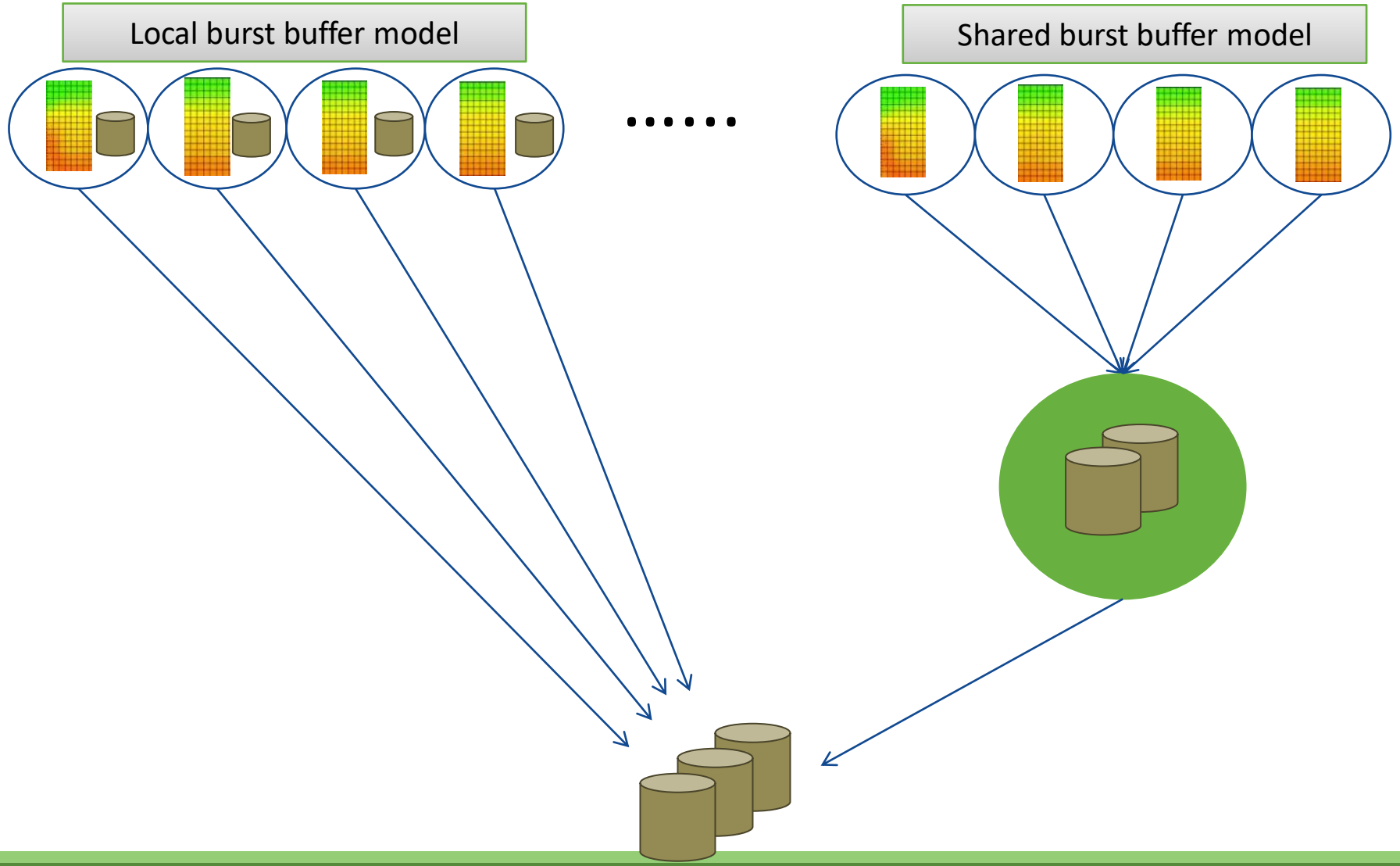
- Simply put, it's a file system for burst buffers
- Our goal is to make using burst buffers on exascale systems as *easy* as writing to the parallel file system and orders of magnitude *faster*
- Results on Summit show scalable write performance for UnifyFS with shared files on burst buffers



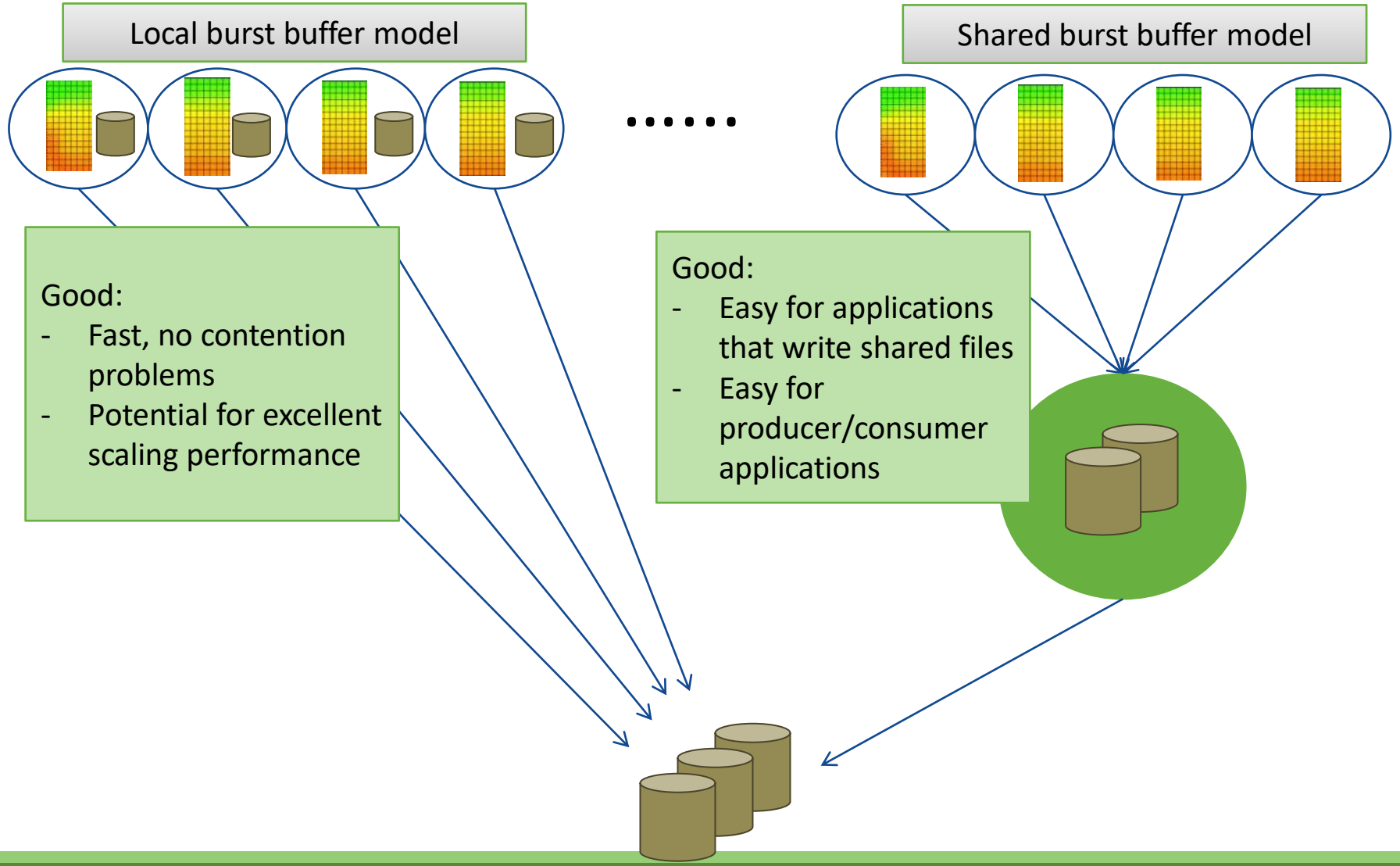
Writing data to the parallel file system is expensive



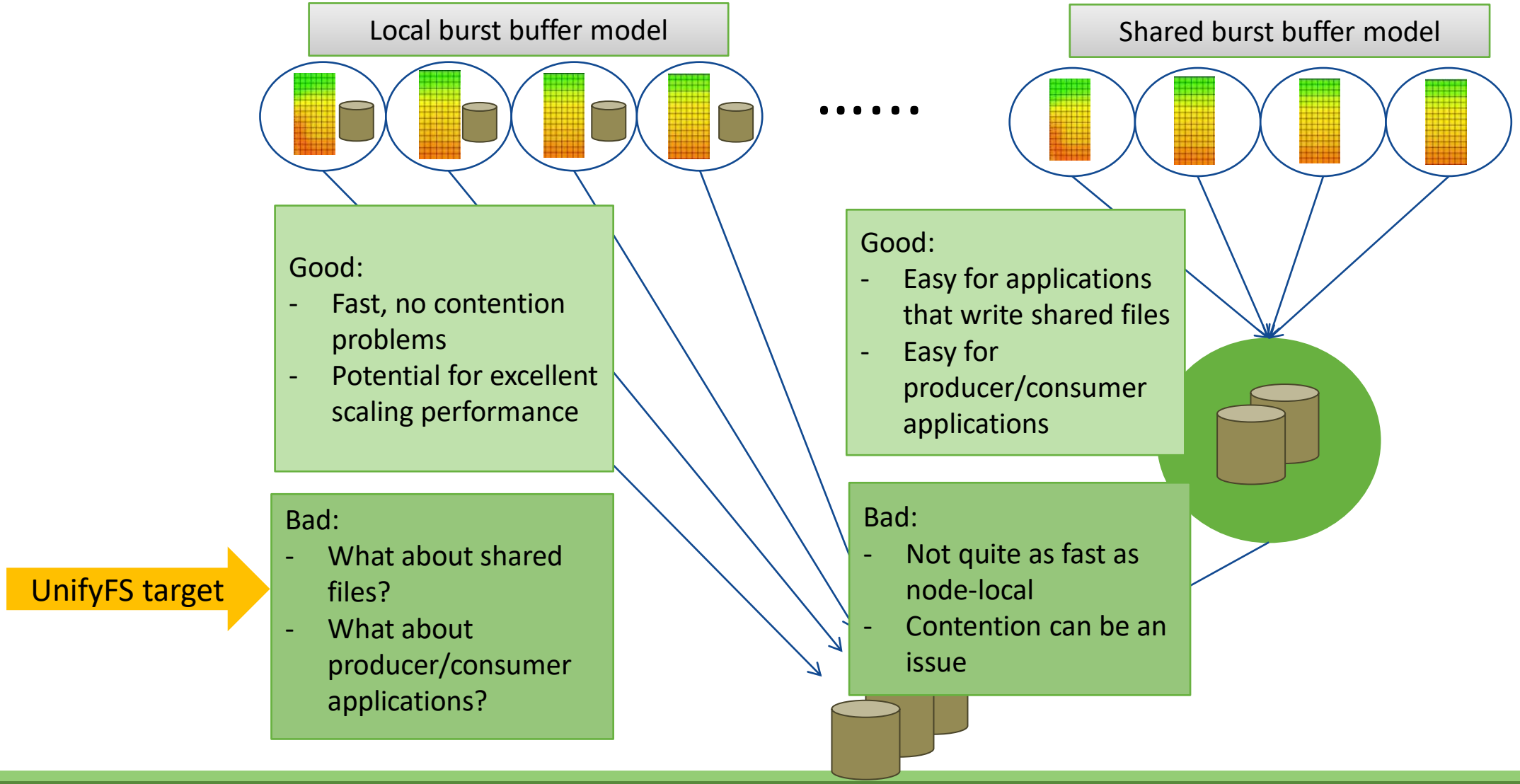
HPC Storage is becoming more complex



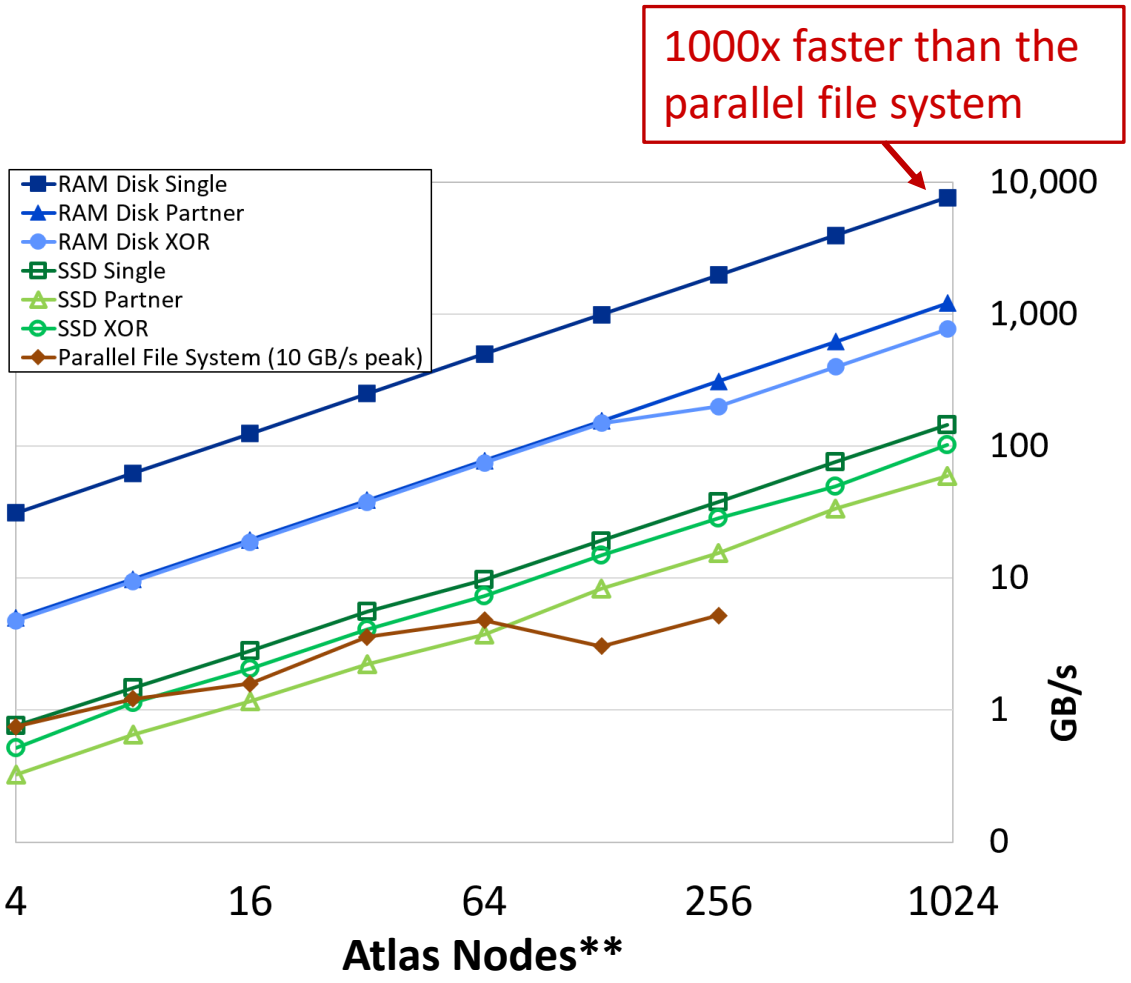
HPC Storage is becoming more complex

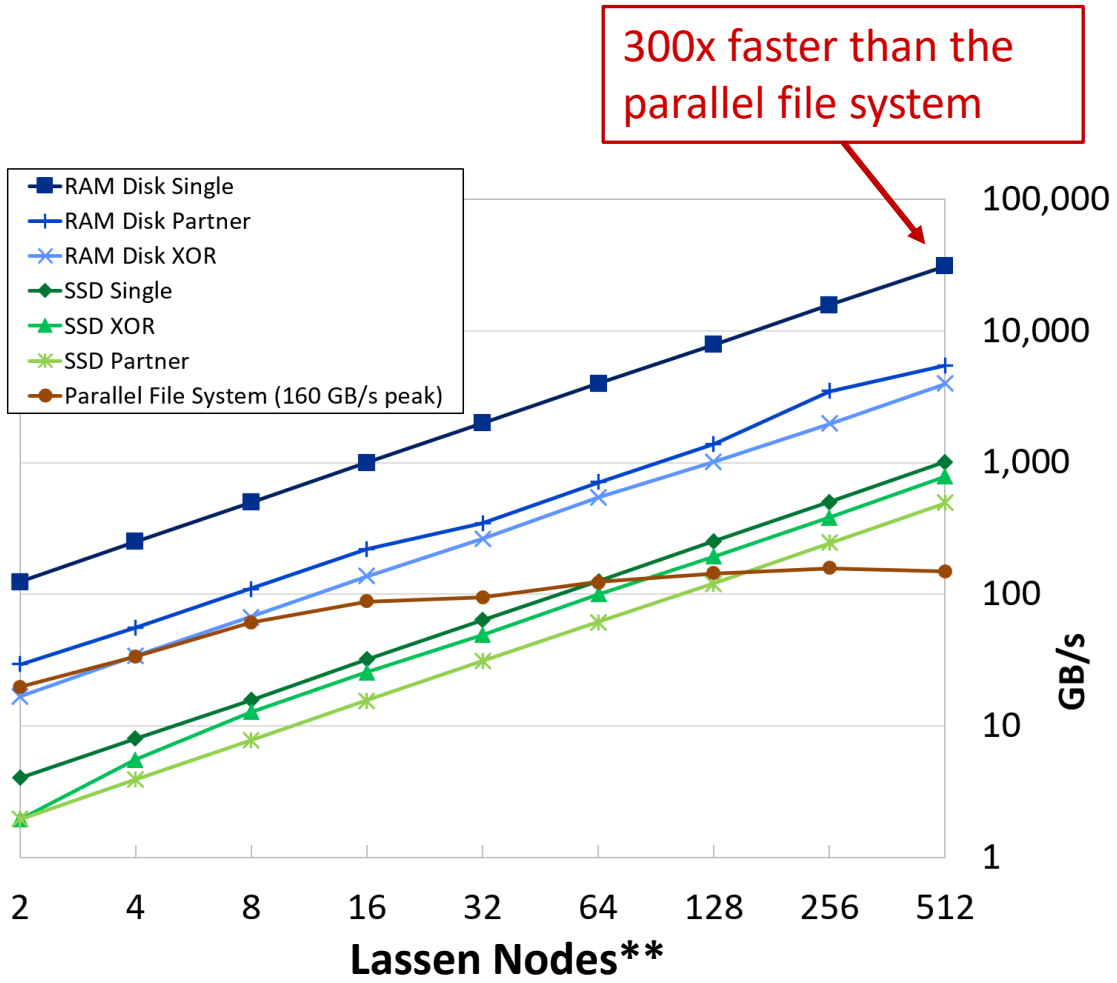


HPC Storage is becoming more complex



UnifyFS targets local burst buffers because they are fast and scalable

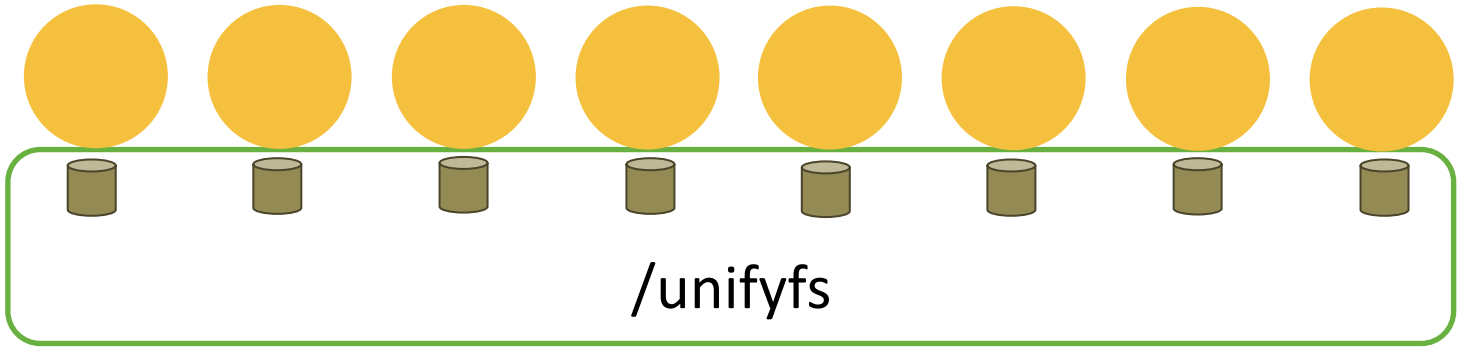




**Measurements of local storage performance taken with SCR (The Scalable Checkpoint/Restart Library) https://github.com/llnl/scr

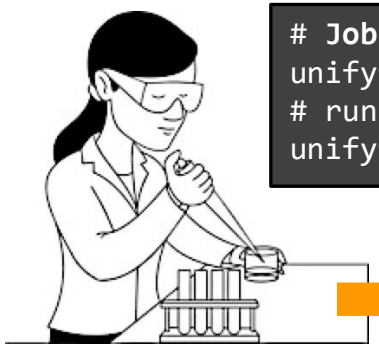
UnifyFS makes sharing files on node-local burst buffers easy and fast

- Sharing files on node-local burst buffers is not natively supported



- UnifyFS makes sharing files **easy**
  - UnifyFS presents a shared namespace across distributed storage
  - 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 contention with other jobs on the system

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



```
# Job Script Integration
unifyfs start -mount=/unifyfs
# run applications and tasks...
unifyfs terminate
```

Batch Job Manager

Dynamic File System for Each Job

HPC Application
(N-1, N-N Checkpointing, Collective I/O with Scientific I/O Libraries)

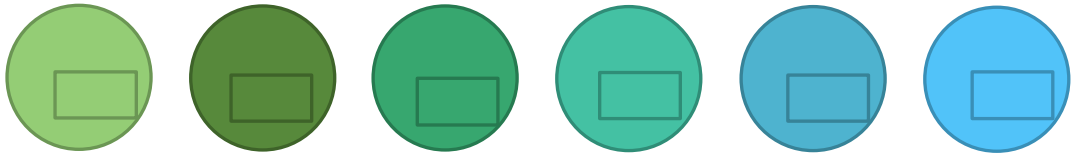
Checkpoint/Restart Lib: SCR, VeloC	Scientific I/O Lib: HDF5, MPI-IO, …	Standard POSIX I/O

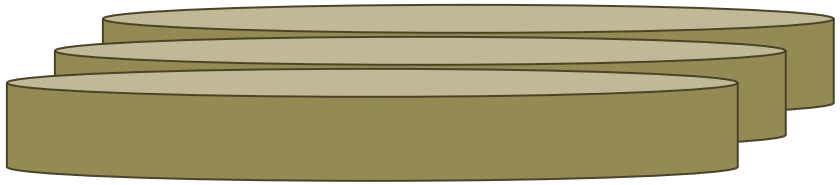
UnifyFS

User level I/O interception	Fast data writes to node local storage
Data visible globally after "lamination"	Transer API to move data to/from parallel file system

UnifyFS optimizes on typical HPC application I/O behavior

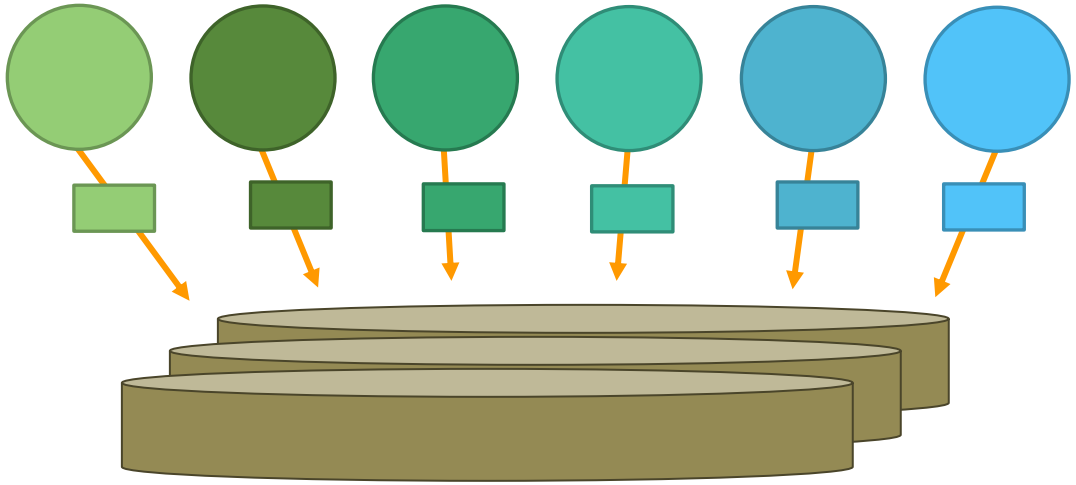
```
int main(int argc, char* argv[]) {
  MPI_Init(argc, argv);

  read_input();

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

      write_output();

  }

  MPI_Finalize();
  return 0;
}
```





UnifyFS optimizes on typical HPC application I/O behavior

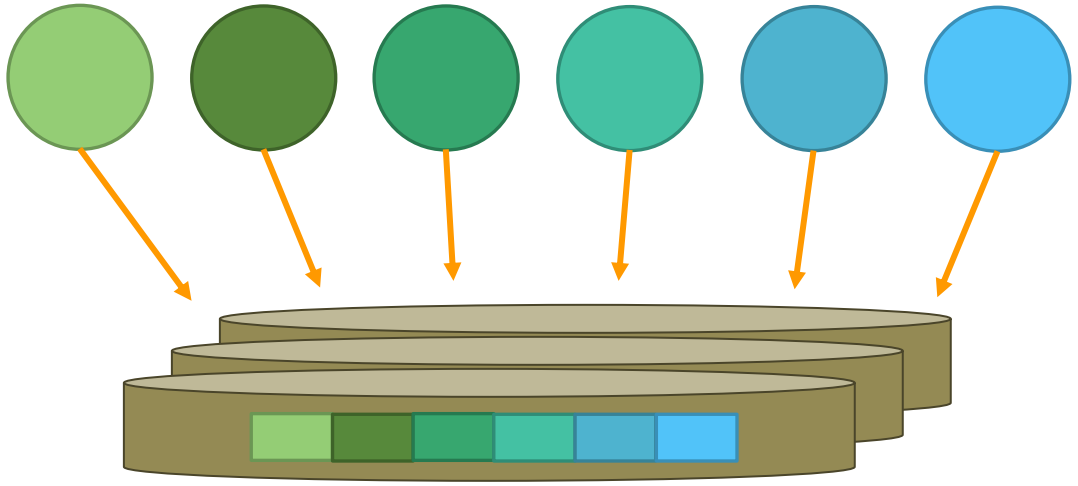
```
int main(int argc, char* argv[]) {
  MPI_Init(argc, argv);

  read_input();

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

      write_output();

  }

  MPI_Finalize();
  return 0;
}
```



- Reads and writes occur in distinct phases

UnifyFS optimizes on typical HPC application I/O behavior

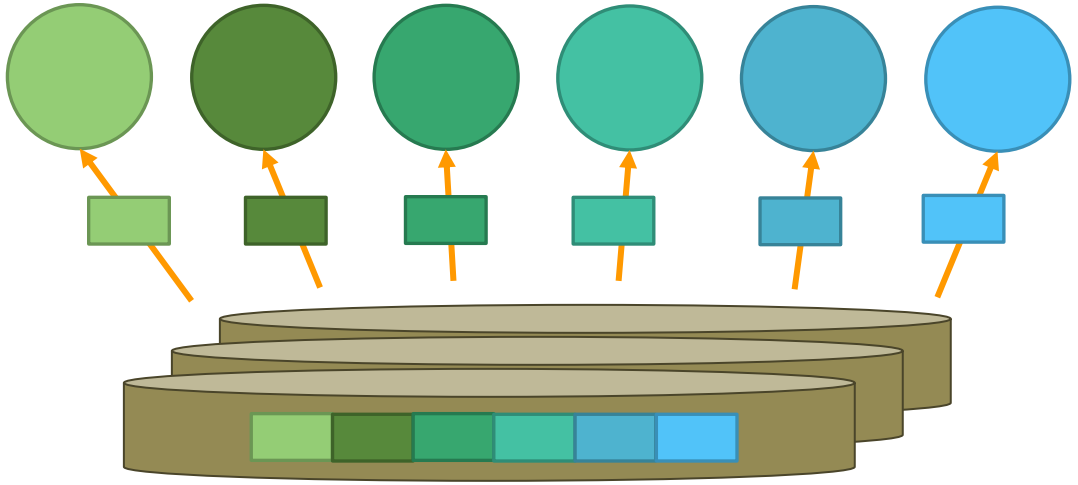
```
int main(int argc, char* argv[]) {
  MPI_Init(argc, argv);

  read_input();

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

      write_output();

  }

  MPI_Finalize();
  return 0;
}
```



- Reads and writes occur in distinct phases
- Reads and writes are performed to regular offsets in files (not random writes or reads)

UnifyFS optimizes on typical HPC application I/O behavior

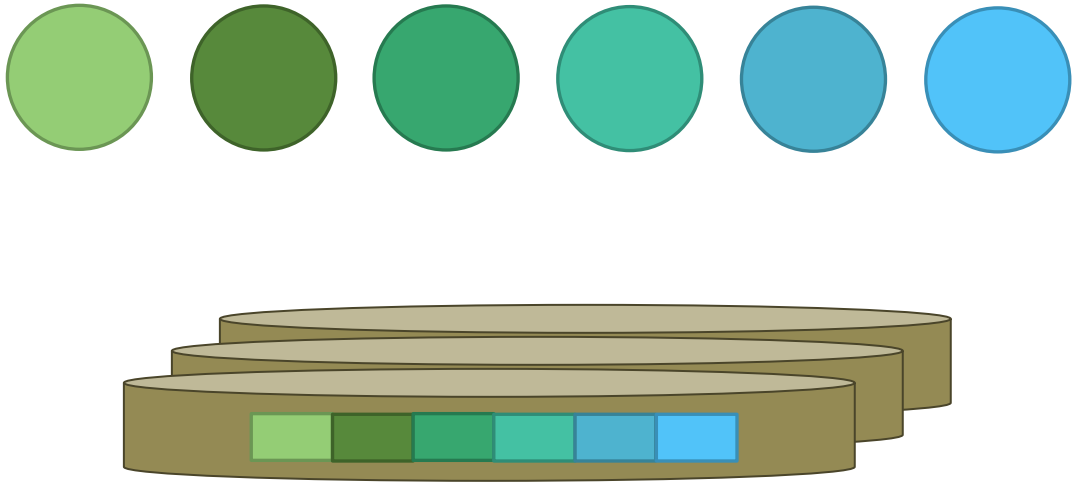
```
int main(int argc, char* argv[]) {
  MPI_Init(argc, argv);

  read_input();

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

      write_output();

  }

  MPI_Finalize();
  return 0;
}
```



- Reads and writes occur in distinct phases
- Reads and writes are performed to regular offsets in files (not random writes or reads)
- When reading, a process will most likely access data it wrote

UnifyFS optimizes on typical HPC application I/O behavior

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

  read_input();

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

      write_output();

  }

  MPI_Finalize();
  return 0;
}
```



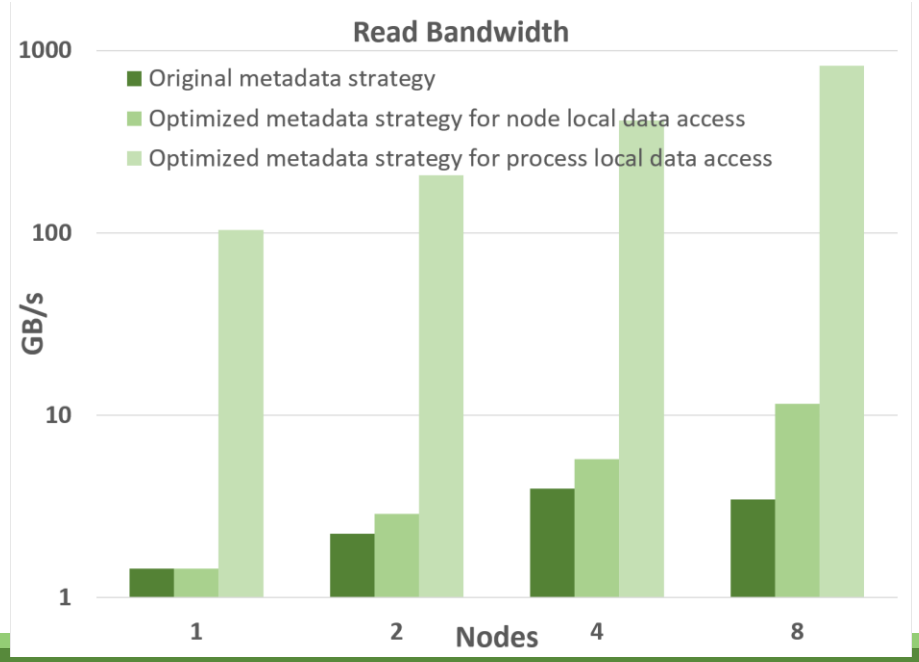
- Reads and writes occur in distinct phases
- Reads and writes are performed to regular offsets in files (not random writes or reads)
- When reading, a process will most likely access data it wrote
- These behaviors are typical of common HPC workloads
  - Checkpoint/restart, periodic output, producer/consumer

UnifyFS files are globally visible after “lamination”

- Based on typical I/O behavior in HPC applications we utilize “lamination semantics”
- Before lamination processes may not see updates written by processes on another node
- Once processes are done modifying a file, they initiate lamination
  - The lamination process renders your file read-only and synchronizes file data across nodes in your job
  - Now any process on any node can read the final state of the file
  - And you can transfer data to external storage

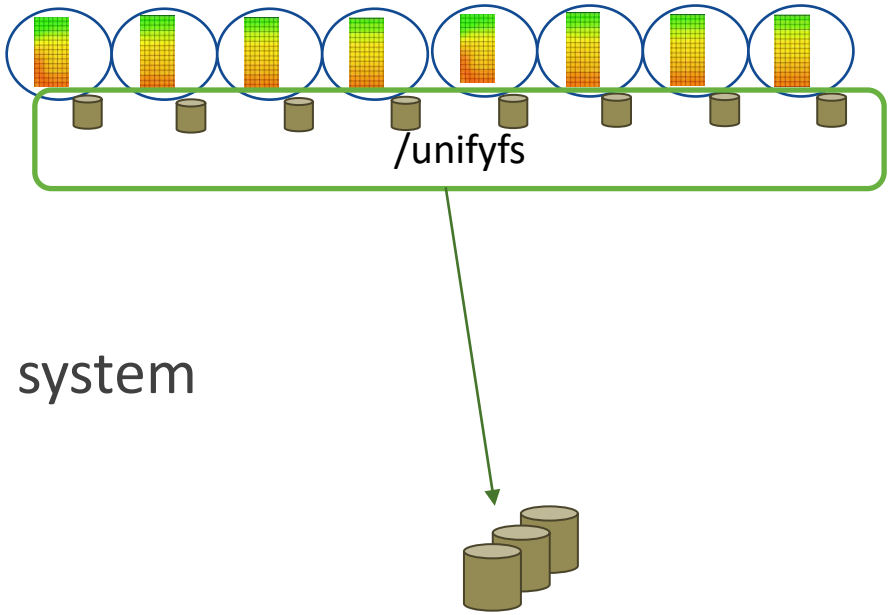
UnifyFS files are globally visible after “lamination”

- Based on typical I/O behavior in HPC applications we utilize “lamination semantics”
- Before lamination processes may not see updates written by processes on another node
- Once processes are done modifying a file, they initiate lamination
  - The lamination process renders your file read-only and synchronizes file data across nodes in your job
  - Now any process on any node can read the final state of the file
  - And you can transfer data to external storage



Now for a tutorial…

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



How do I get and build UnifyFS?

Got Spack?

```
$ spack install unifyfs

$ spack load unifyfs
```

https://github.com/spack/spack

How do I modify my MPI application for UnifyFS?

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

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





     //clean up
    return 0;
}
```

How do I modify my MPI application for UnifyFS?

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

     //perform I/O
    fp = fopen("/unifyfs/out.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/out.txt", 0444);

     //clean up
    return 0;
}
```

How do I modify my MPI application for UnifyFS?

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

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

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

     //clean up
    return 0;
}
```

- "chmod" is currently supported method for lamination
- (Near-)future methods we plan to support
  - Laminate on close()
  - Laminate on unmount() (either explicit API call or auto-mount)
  - Laminate with API call, e.g., unifyfs_laminate
- Expect to be able to specify which method you want when you start up UnifyFS

How do I modify my **serial** application for UnifyFS?

```
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/out.txt", 0444);

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

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

```
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
- We are partnered with HDF5 in ECP so we test it the most
- Build and run your application with UnifyFS, change the path

```
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);
  chmod(file, 0444);
  return;
}
```

How does UnifyFS intercept I/O calls?

- Static Linking

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

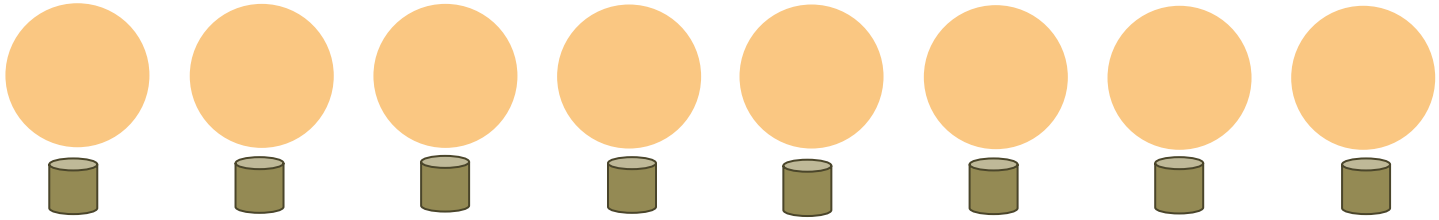
- Dynamic Linking (Recommended method)
  - Using Spack makes this very easy! (just “spack load unifyfs”)
  - We use the Gotcha library for dynamic interception

```
$ mpicc -o hello -L<unifyfs>/lib hello.c -lunifyfs_gotcha
```

How do I run my code with UnifyFS?

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

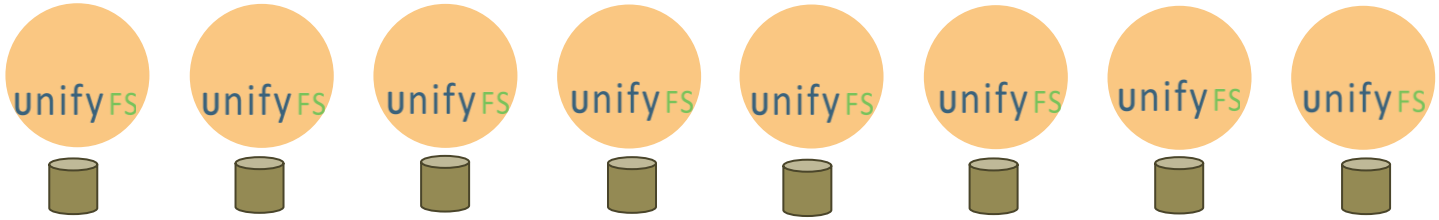
```
### allocate nodes and options for resource manager
#BSUB –nnodes 8 …
```



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

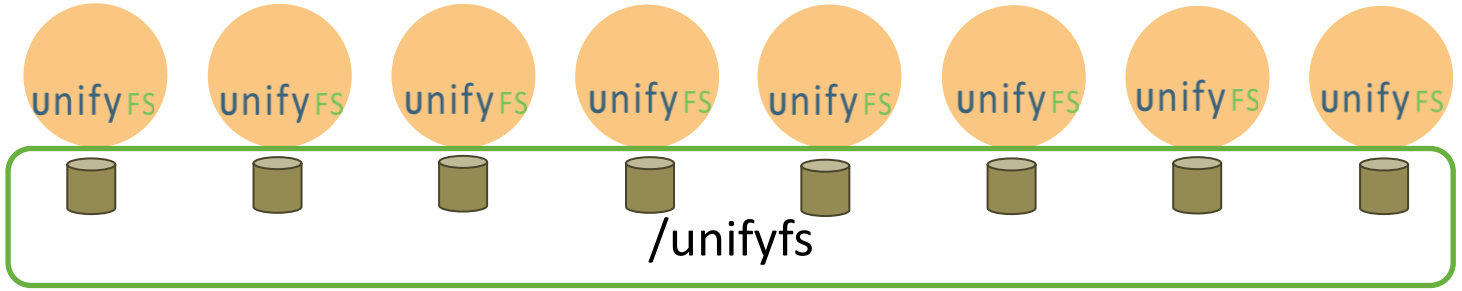
```
### allocate nodes and options for resource manager
#BSUB –nnodes 8 …

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



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

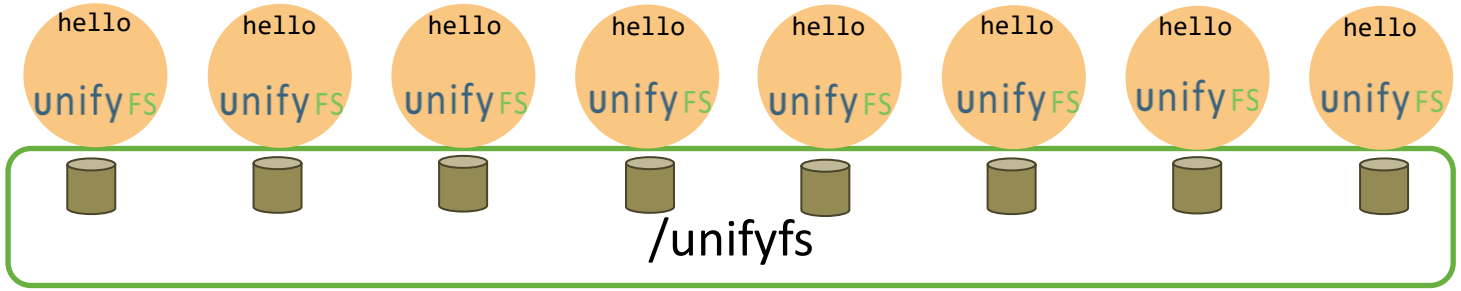
```
### allocate nodes and options for resource manager
#BSUB –nnodes 8 …

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



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

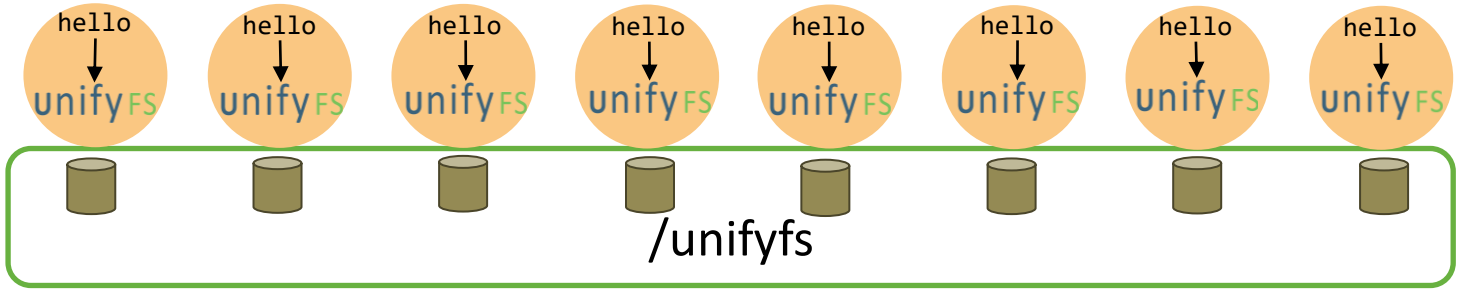
```
### allocate nodes and options for resource manager
#BSUB -nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path --mount=/unifyfs
jsrun -p 4096 ./hello
```



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

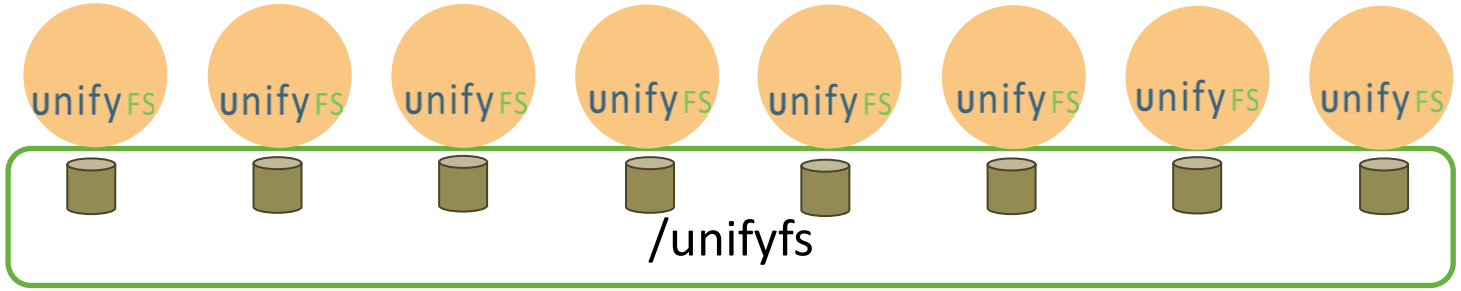
```
### allocate nodes and options for resource manager
#BSUB -nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path --mount=/unifyfs
jsrun -p 4096 ./hello
```



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

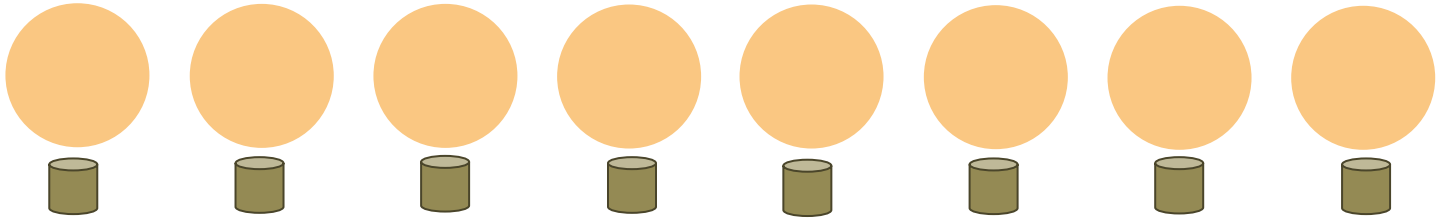
```
### allocate nodes and options for resource manager
#BSUB -nnodes 8 …

### shell command portion of batch script
unifyfs start --share-dir=/shared/file/system/path --mount=/unifyfs
jsrun -p 4096 ./hello
unifyfs terminate
```



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 --mount=/unifyfs
jsrun –p 4096 ./hello
unifyfs terminate
```

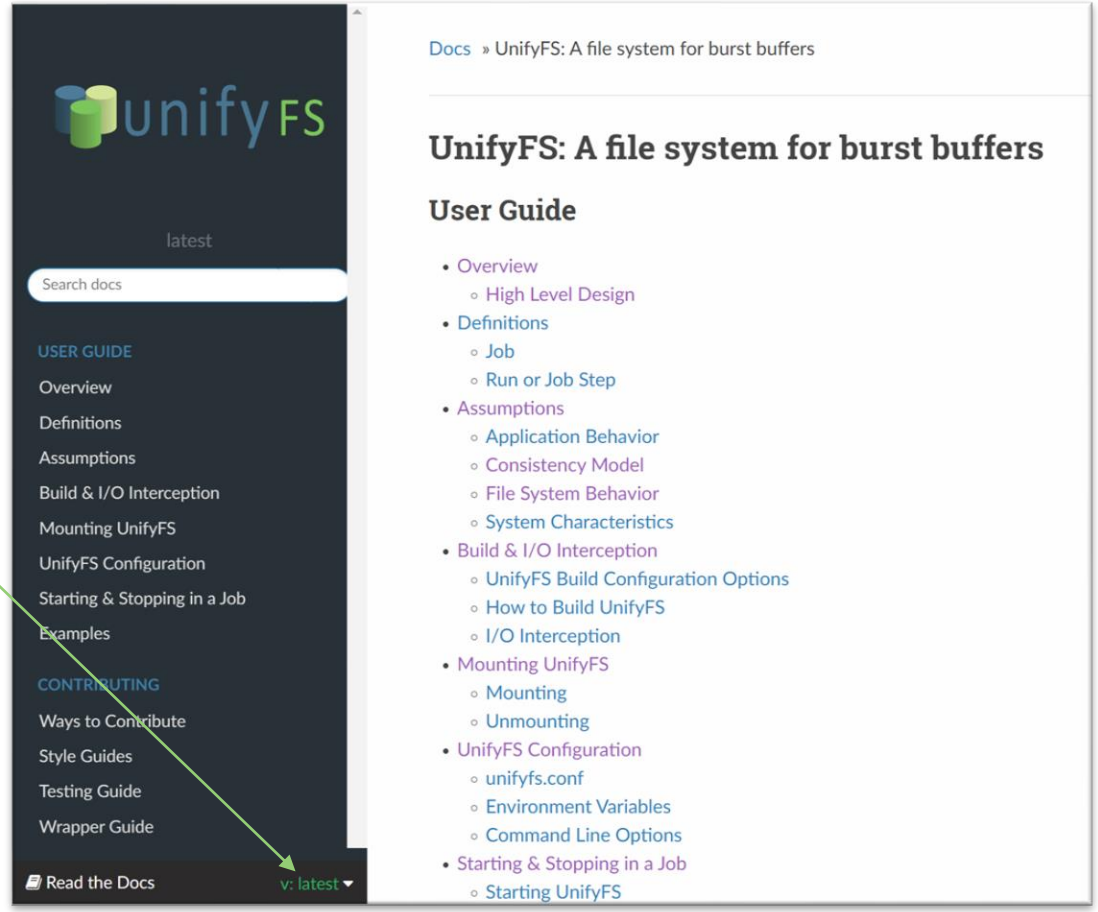


How do I move my data into and out of UnifyFS?

- Three ways to move data between UnifyFS and the parallel file system
- UnifyFS transfer program
  - `jsrun –r1 transfer /unifyfs/file1 /scratch/mydir/file1`
- UnifyFS transfer API
  - `unifyfs_transfer_file_parallel("/unifyfs/out.txt", "/scratch/out.txt");`
- Stage in and out options with UnifyFS command "unifyfs start"
  - `unifyfs start --stage-in=/path/to/parallel/file/system/inputs --stage-out=/path/to/parallel/file/system/outputs`

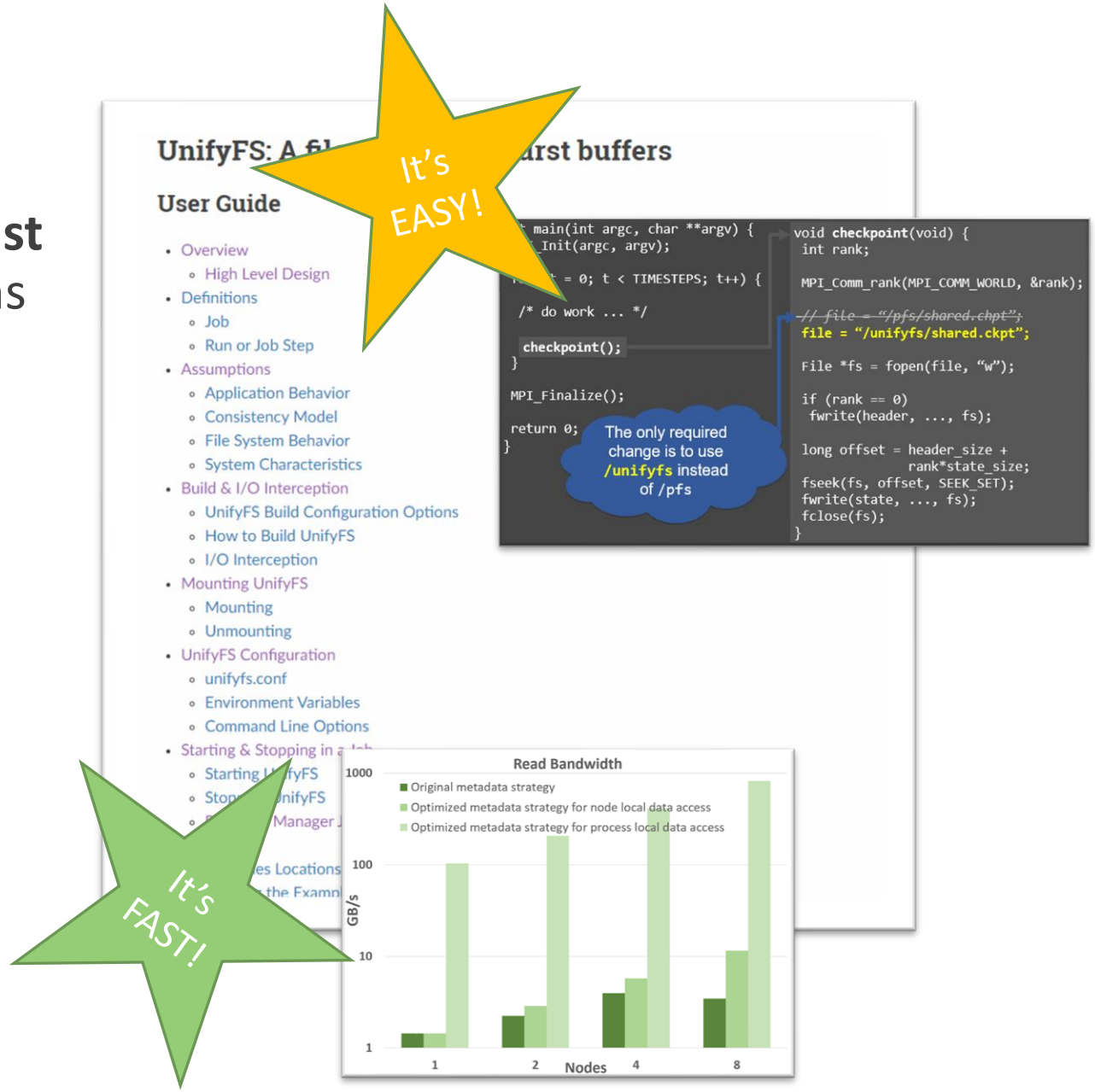
Wanna learn more?

- New features and improvements being added all the time
- Documentation and user support
  - User Guide: http://unifyfs.readthedocs.io
    - Down at the bottom, where it says “Read the Docs” chose “v: latest” to get the latest information
  - unifyfs@exascaleproject.org
- Example programs available in the “examples” directory in the source code



We need you!

- Our goal is to provide **easy, portable,** and **fast** support for burst buffers for ECP applications
- We need early users
- What features are most important to you
- Available on github: https://github.com/LLNL/UnifyFS
  - MIT license
- Documentation and user support
  - User Guide: http://unifyfs.readthedocs.io
  - unifyfs@exascaleproject.org



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 document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

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.

Tutorial: 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;
}
```

Tutorial: 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);
  chmod(file, 0444);
  VeloC_Checkpoint_end();
  return;
}
```