CSE 5449: Intermediate Studies in Scientific Data Management

Lecture 7: HDF5 Hyperslabs and Parallel I/O with MPI-IO

Dr. Suren Byna

The Ohio State University

E-mail: byna.1@osu.edu

https://sbyna.github.io

01/31/2023

Today's class

- Class project – Revised execution plan presentation
- Homework – h5bench runs and PnetCDF basic tests
- HDF5 hyperslabs
- MPI-IO

Class projects

5. Performance comparison of sub-filing in HDF5 and PnetCDF

- Background: Sub-filing is an approach to split a very large file into smaller files. However, there are pros / cons with the approach on how the data is organized.
- Question
  - Which of the HDF5 and PnetCDF sub-filing approaches are best?
  - What better strategies for sub-filing are there?
- Deliverable: A short paper describing
- Resources
  - Tuning HDF5 subfiling performance on parallel file systems https://escholarship.org/content/qt6fs7s3jb/qt6fs7s3jb.pdf
  - Using Subfiling to Improve Programming Flexibility and Performance of Parallel Shared-file I/O https://ieeexplore.ieee.org/document/5362452
  - Scalable Parallel I/O on a Blue Gene/Q Supercomputer Using Compression, Topology-Aware Data Aggregation, and Subfiling https://ieeexplore.ieee.org/document/6787260
  - HDF5 Subfiling presentation:
    - https://www.hdfgroup.org/wp-content/uploads/2022/09/HDF5-Subfiling-VFD.pdf
    - https://www.youtube.com/watch?v=psl2iZmP2SY
  - PnetCDF subfiling
    - http://cucis.eecs.northwestern.edu/projects/PnetCDF/subfiling.html

Homework

- h5bench runs
  - write and read benchmarks
- PnetCDF basic runs

How to write a subset of an array?

```
$ h5dump file.h5

HDF5 "file.h5" {
GROUP "/" {
   DATASET "A" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 4, 6 ) / ( 4, 6 ) }
      DATA {
      (0,0): 0, 0, 0, 0, 0, 0,
      (1,0): 1, 2, 3, 4, 5, 6,
      (2,0): 0, 0, 0, 0, 0, 0,
      (3,0): 0, 0, 0, 0, 0, 0
      }
   }
   GROUP "B" {
   }
}
}
```

How to Describe a Subset in HDF5?

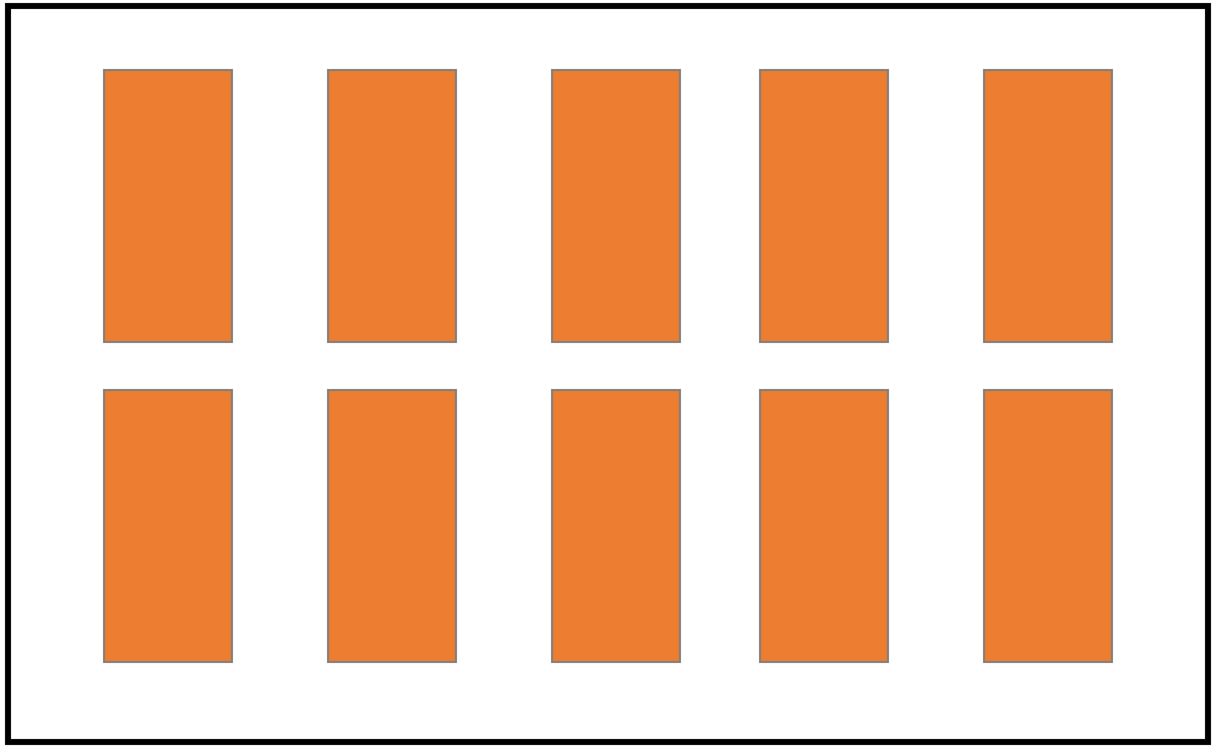
- Before writing and reading a subset of data one must describe it to the HDF5 Library
- HDF5 APIs and documentation refer to a subset as a “selection” or “hyperslab selection”
- If specified, HDF5 library will perform I/O on a selection *only* and not on all elements of a dataset.

Types of Selections in HDF5

- Two types of selections
  - Hyperslab selection
    - Regular hyperslab
    - Simple hyperslab
    - Result of set operations on hyperslabs (union, difference, …)
  - Point selection

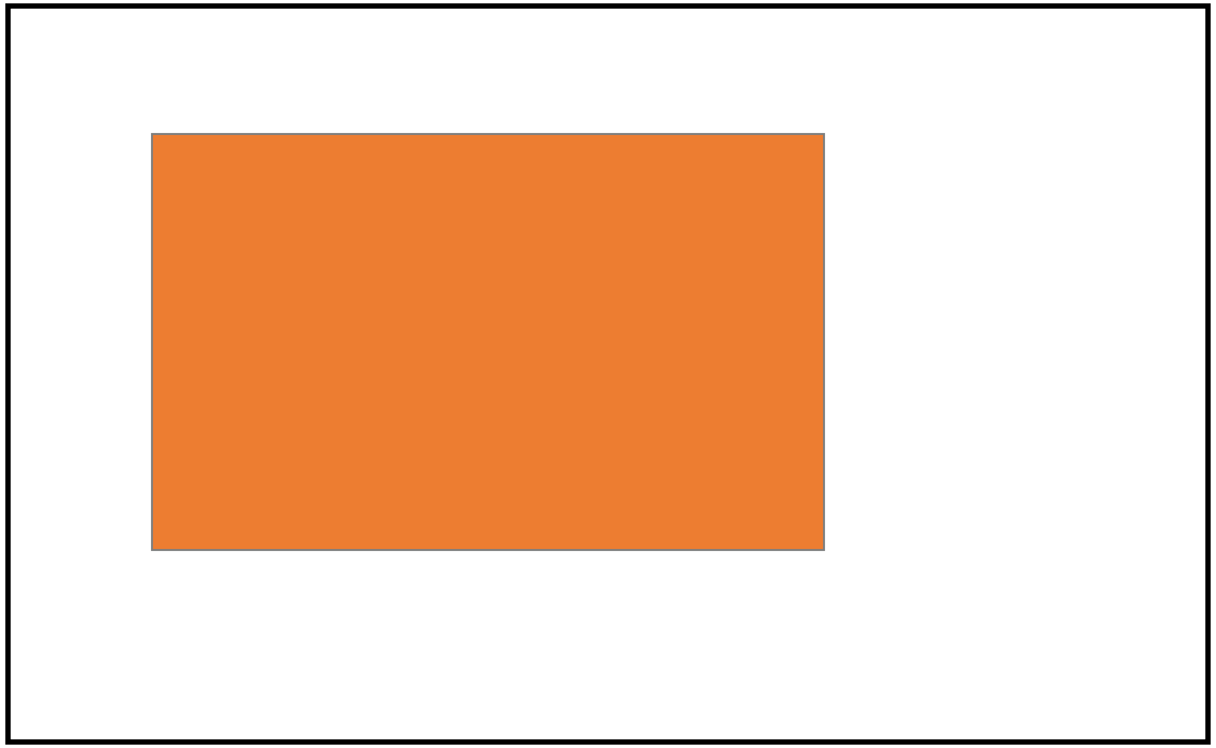
- Hyperslab selection is especially important for doing parallel I/O in HDF5

Regular Hyperslab



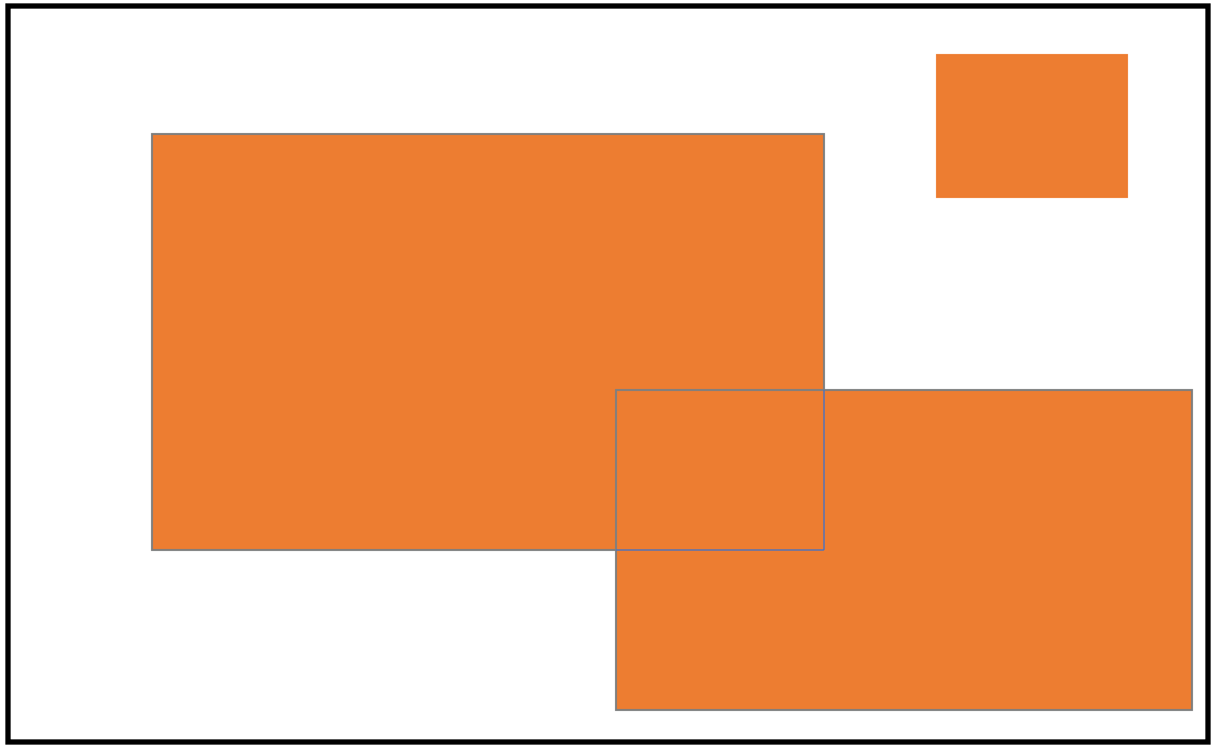
Collection of regularly spaced blocks of equal size

Simple Hyperslab



Contiguous subset or sub-array

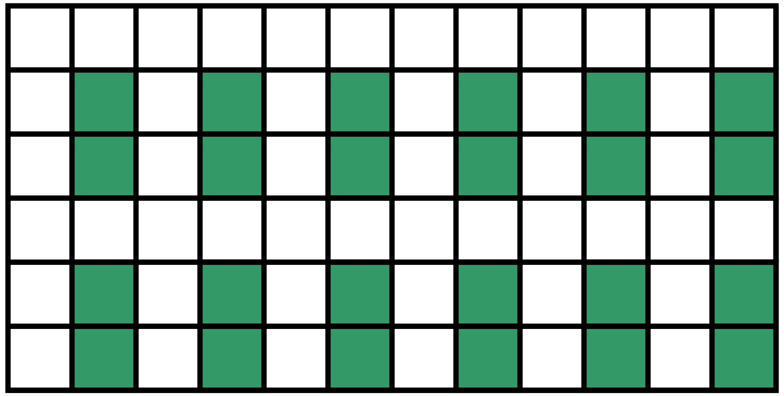
Hyperslab Selection



Result of union operation on three simple hyperslabs

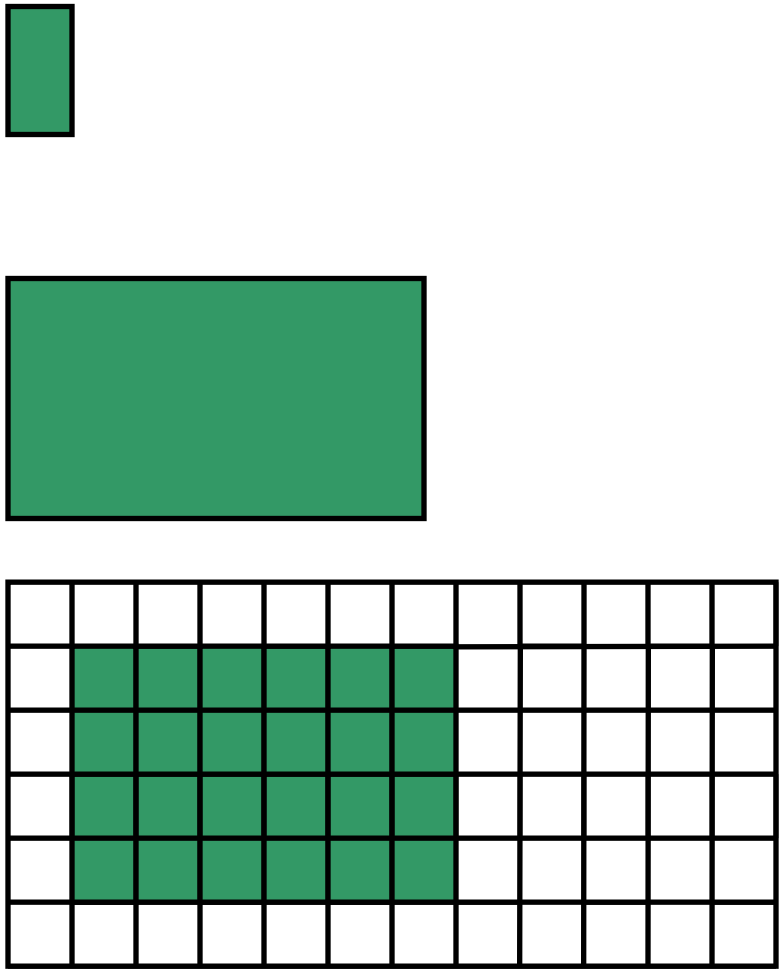
HDF5 Hyperslab Description

- *Everything is "measured" in number of elements*
- A hyperslab is defined with 4 properties
- Start - starting location of a hyperslab (1,1)
- Stride - number of elements that separate each block (3,2)
- Count - number of blocks (2,6)
- Block - block size (2,1)



Simple Hyperslab Description

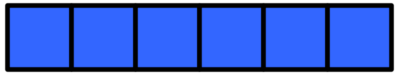
- Two ways to describe a simple hyperslab
- As *several* blocks
  - **Stride – (1,1)**
  - **Count – (2,6)**
  - Block – (2,1)
- As *one* block
  - Stride – (1,1)
  - Count – (1,1)
  - **Block – (4,6)**



No performance penalty for one way or another

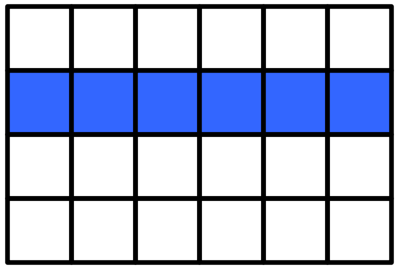
Writing a row

- Memory space selection is 1-dim array of size 6



- File space selection

start = {1,0}, stride = {1,1}, count = {1,6}, block = {1,1}



Number of elements selected in memory should be the same as selected in the file

Modified slides from The HDF Group (Scot Breitenfeld, Quincey Koziol, et al.)

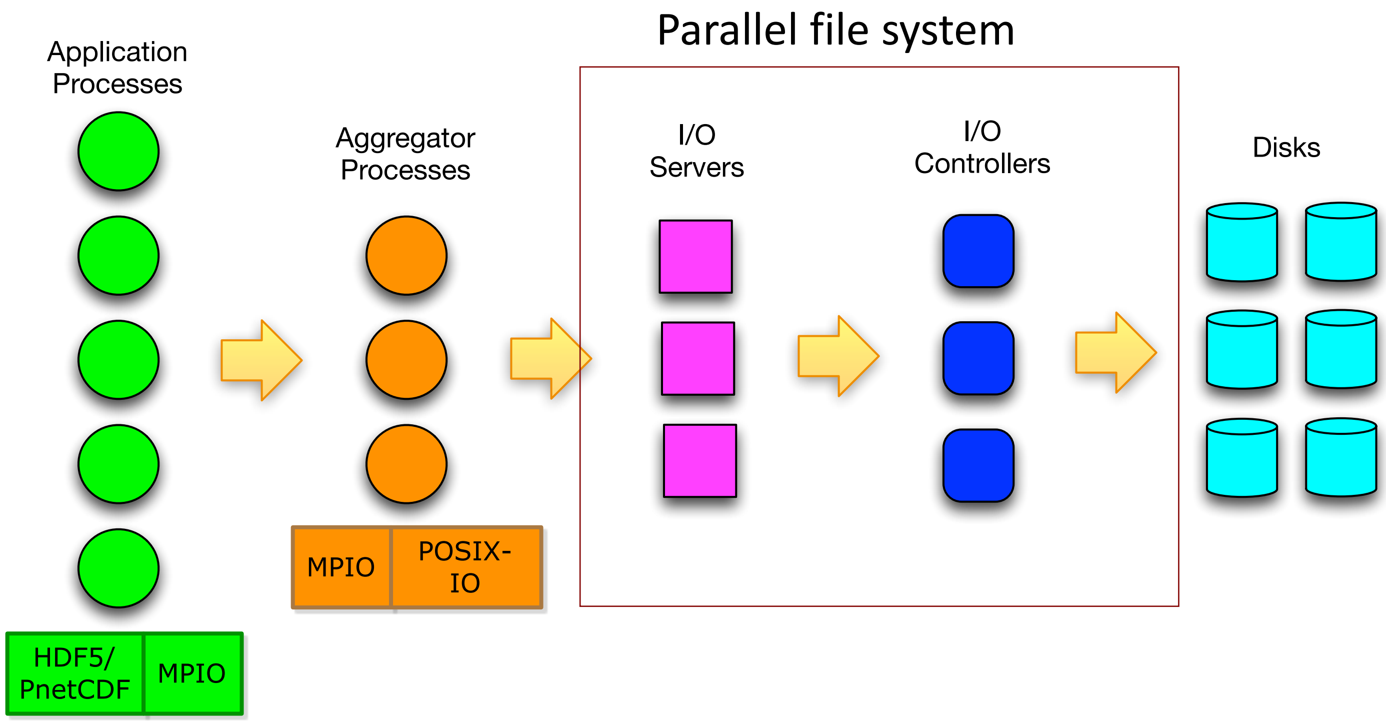
Writing a row

```
hid_t       mspace_id, fspace_id;
hsize_t     dims[1] = {6};
hsize_t     start[2], count[2];
…..
/* Create memory dataspace */
mspace_id = H5Screate_simple(1, dims, NULL);

/* Get file space identifier from the dataset */
fspace_id = H5Dget_space(dataset_id);

/* Select hyperslab in the dataset to write too */
start[0] = 1;
start[1] = 0;
count[0] = 1;
count[1] = 6;
status = H5Sselect_hyperslab(fspace_id, H5S_SELECT_SET,
         start, NULL, count, NULL);
H5Dwrite(dataset_id, H5T_NATIVE_INT, mspace_id, fspace_id,
         H5P_DEFAULT, wdata);
```

Parallel I/O with MPI-IO

Reminder: Parallel I/O software stack

- Multiple layers of software libraries and hardware
- High-level libraries (HDF5, PnetCDF, etc.), middleware (MPI-IO), parallel file system (Lustre, GPFS, etc.)

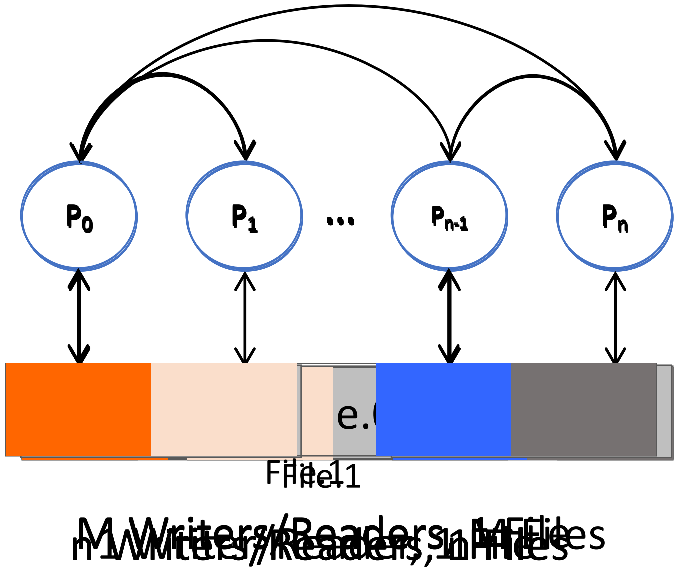


MPI-IO

- A lower-level interface than HDF5, PnetCDF (high-level I/O libraries)
- A convenient interface for enabling parallel I/O
  - Used by high-level I/O libraries as well as application developers
- What does MPI-IO offer?
  - Provides mechanism for performing synchronization
  - Syntax for data movement
  - Optimizations – collective buffering, data sieving, etc.
  - Allows definitions of non-contiguous data layout in files (MPI derived datatypes)

Parallel I/O – Application view

- **Types of parallel I/O**
  - 1 writer/reader, 1 file
  - N writers/readers, N files (File-per-process)
  - N writers/readers, 1 file
  - M writers/readers, 1 file
    - Aggregators
    - Two-phase I/O
  - M aggregators, M files (file-per-aggregator)
    - Variations of this mode



Parallel I/O in MPI-IO

- Program level
  - Multiple processes concurrently perform I/O (read / write) operations to a **common** file

- System level
  - A parallel file system and storage hardware that support concurrent accesses to a **common** file

Independent I/O in MPI-IO

- Common operations (in POSIX-IO)
  - Open the file
  - Read / Write data from / to the file
  - Close the file

- In MPI-IO
  - Open the file: MPI_File_open
  - Write to the file: MPI_File_write
  - Close the file: MPI_File_close

Adapted from Prof. Bill Gropp's "Introduction to MPI-IO" talk

MPI-IO: Independent I/O example

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
        MPI_File fh;
        int buf [1000], rank;
        MPI_Init (0,0);
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
        MPI_File_open (MPI_COMM_WORLD, "mpi-ind-file.out",
                MPI_MODE_CREATE|MPI_MODE_WRONLY,
                MPI_INFO_NULL, &fh);
        if (rank == 0)
                MPI_File_write (fh, buf, 1000, MPI_INT, MPI_STATUS_IGNORE);
        MPI_File_close (&fh);
        MPI_Finalize();
        return 0;
}
```

Adapted from Prof. Bill Gropp's "Introduction to MPI-IO" talk

MPI-IO: Independent I/O example

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
        MPI_File fh;
        int buf [1000], rank;
        MPI_Init (0,0);
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
        MPI_File_open (MPI_COMM_WORLD, "mpi-ind-file.out",
                MPI_MODE_CREATE|MPI_MODE_WRONLY,
                MPI_INFO_NULL, &fh);
        if (rank == 0)
                MPI_File_write (fh, buf, 1000, MPI_INT, MPI_STATUS_IGNORE);
        MPI_File_close (&fh);
        MPI_Finalize();
        return 0;
}
```

MPI_File_open → Collective operation

MPI_File_write → Independent operation

MPI_File_close → Collective operation

MPI_File_write ()
MPI_File_write_at ()

MPI_MODE_WRONLY / MPI_MODE_RDONLY / MPI_MODE_RDWR / MPI_MODE_CREATE have to be passed to MPI_File_open ()

Adapted from Prof. Bill Gropp's "Introduction to MPI-IO" talk

MPI-IO: Independent I/O example

```
#include "mpi.h"
MPI_Status status;
MPI_File fh;
MPI_Offset offset;

MPI_File_open (MPI_COMM_WORLD, "file.bin",
        MPI_MODE_RDONLY, MPI_INFO_NULL, &fh)
nints = FILESIZE / (nprocs*INTSIZE);
offset = rank * nints * INTSIZE;
MPI_File_read_at (fh, offset, buf, nints, MPI_INT, &status);
MPI_Get_count (&status, MPI_INT, &count);
printf ("process % d read %d ints\n", rank, count);
MPI_File_close (&fh);
```

MPI_File_write ()
MPI_File_write_at ()

MPI_MODE_WRONLY / MPI_MODE_RDONLY / MPI_MODE_RDWR / MPI_MODE_CREATE must be passed to MPI_File_open ()

MPI_File_seek
MPI_File_read
MPI_File_write

MPI_File_read_at
MPI_File_write_at

} Combines seek + I/O for thread safety

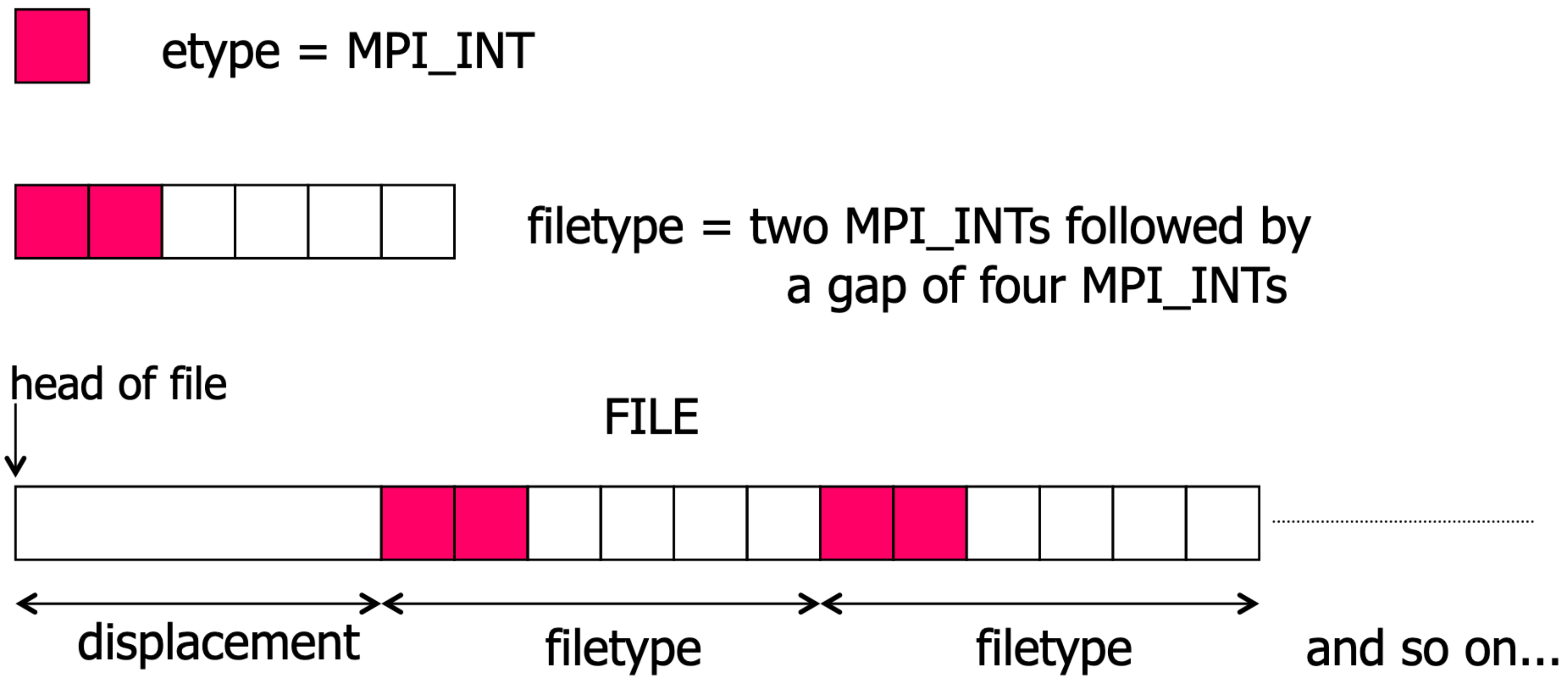
Adapted from Prof. Bill Gropp's "Introduction to MPI-IO" talk

File views for non-contiguous accesses

- Each process describes the part of the file, i.e., file view, for which it is responsible
- Only the part of the file described by the file view is visible to the process; reads and writes access these locations
- Specified by a triplet (displacement, etype, and filetype) passed to MPI_File_set_view
    - displacement = number of bytes to be skipped from the start of the file
    - etype = basic unit of data access (can be any basic or derived datatype)
    - filetype = specifies which portion of the file is visible to the process
- HDF5 → Similar to Hyperslabs, hyperslabs provide more flexibility

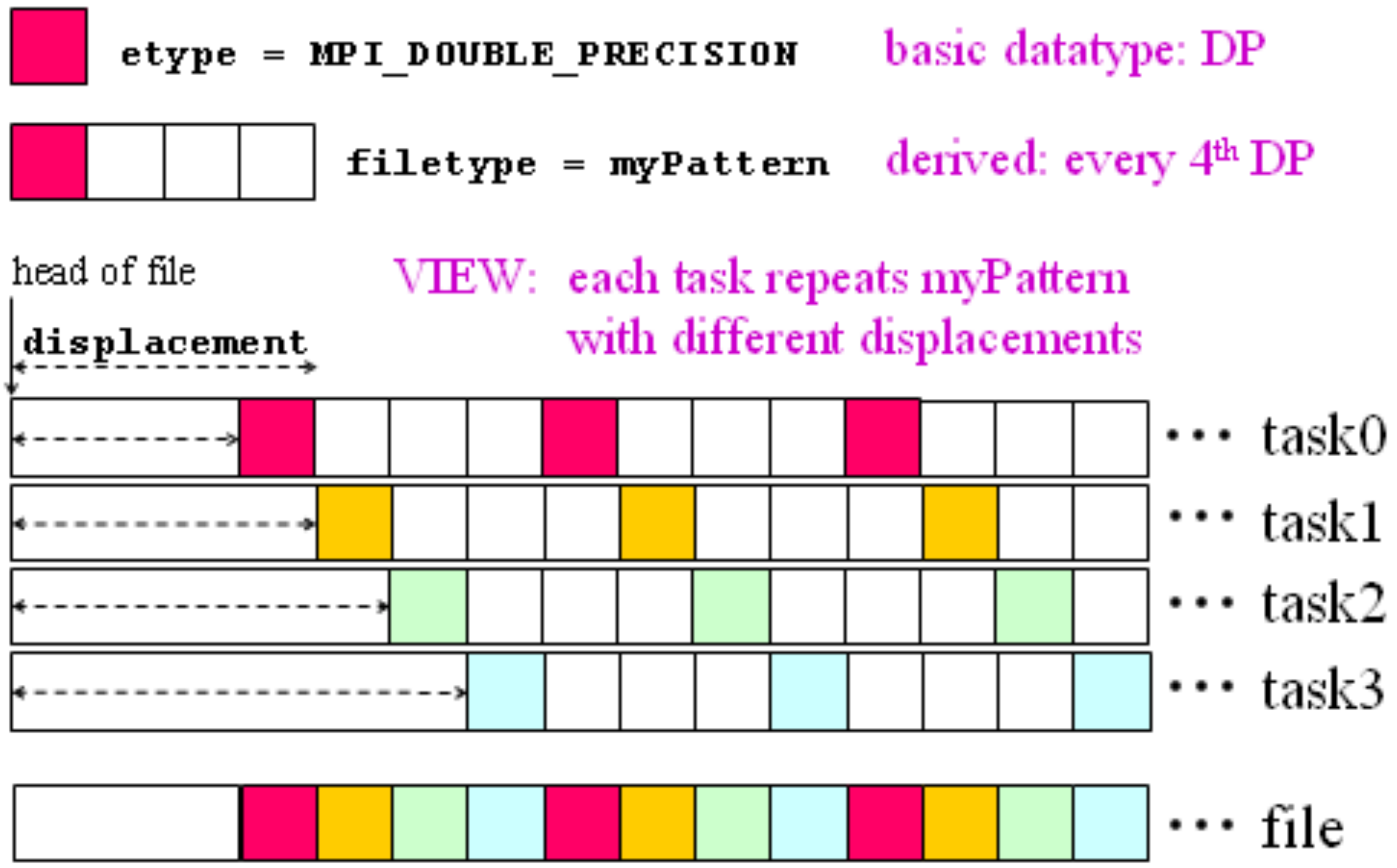
Adapted from Prof. Bill Gropp's "Introduction to MPI-IO" talk

Simple non-contiguous file view



Adapted from Prof. Bill Gropp's "Introduction to MPI-IO" talk

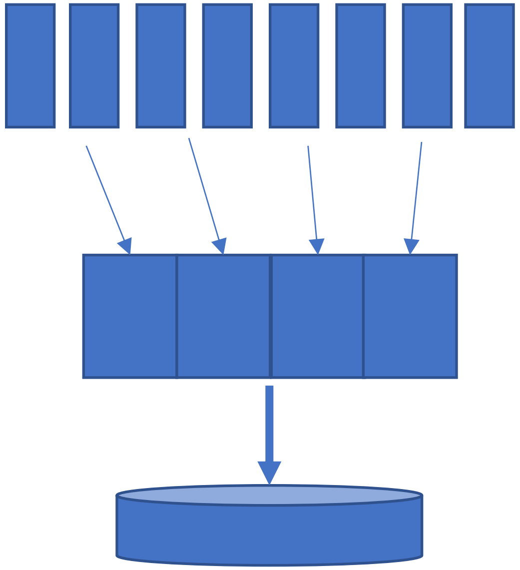
Simple non-contiguous file view – from multiple processes



https://cvw.cac.cornell.edu/ParallelIO/fileviews

Example code: https://cvw.cac.cornell.edu/ParallelIO/fileviewex

Collective I/O in MPI



- All processes must call the collective I/O function
- Aggregating large blocks so that the reads / writes to the I/O system would be large

- `MPI_File_write_at_all ()`
    - _all → all processes in the communicator are participating
    - _at → provides thread-safety and avoids a separate seek
- `MPI_File_seek`
    - `MPI_File_read_all`
    - `MPI_File_write_all`
    - `MPI_File_read_at_all`
    - `MPI_File_write_at_all`

When to use independent and collective?

- Independent
  - A small number of large I/O requests from processes
  - Load imbalance among processes that need to wait for too long in a collective call
- Collective
  - A large number of small I/O requests from processes → aggregation is beneficial
  - Load on all processes is approximately the same

Summary of today's class

- Class project
- HDF5 Hyperslabs
- MPI-IO basics
- Next Class –
  - A few more details on MPI-IO
  - PnetCDF and ADIOS

Collective buffering