Introduction to Parallel Processing

Lecture 17 : MPI I/O

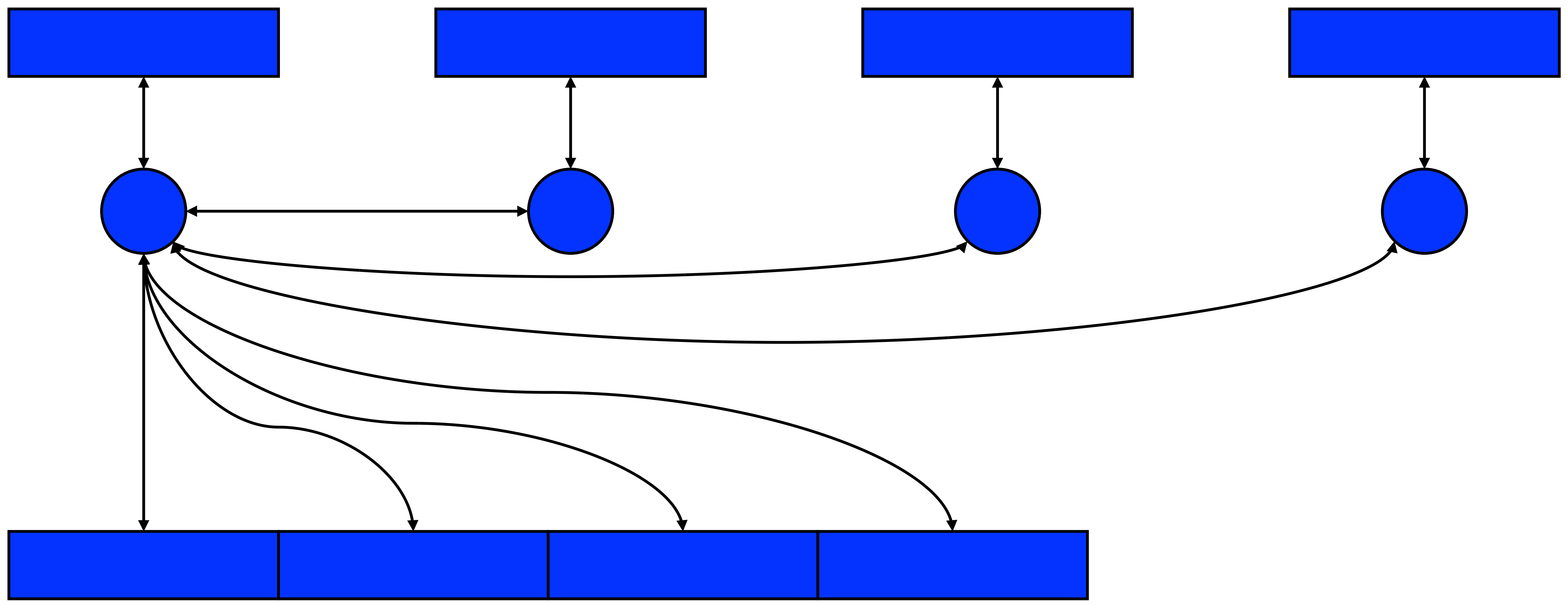
10/21/2022

Professor Amanda Bienz

Why MPI IO?

- MPI allows for parallel performance when performing I/O
- Can write to a single file from all processes, instead of one file per process
- MPI provides a lot of helpful I/O methods

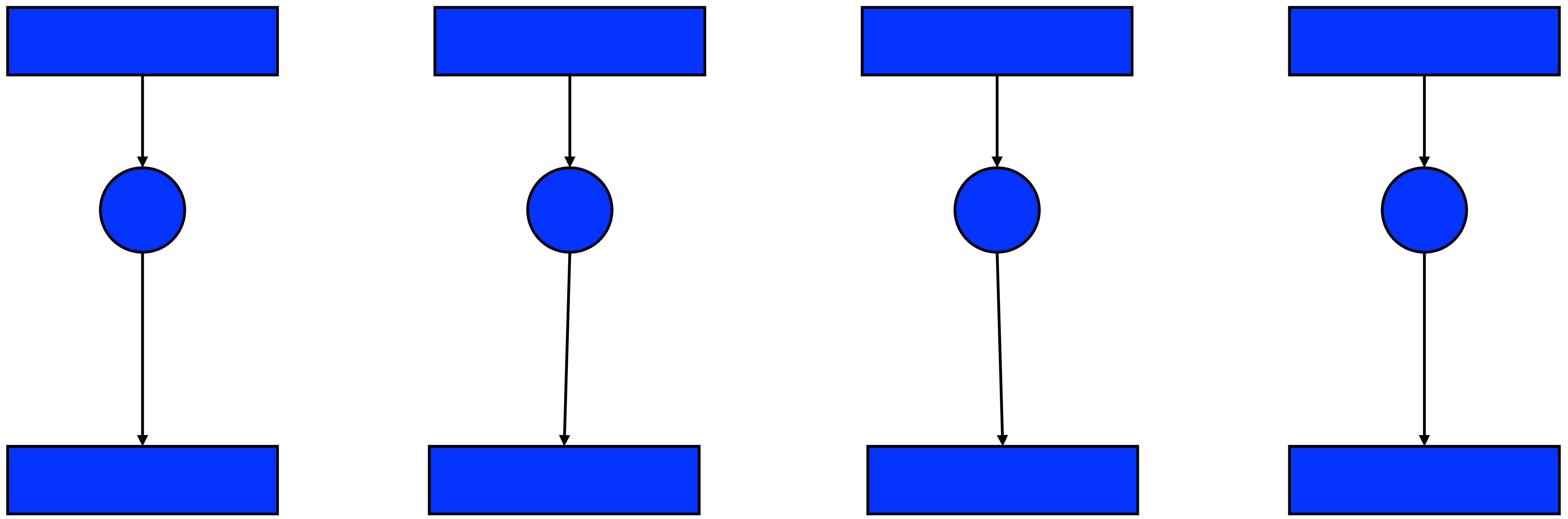
Non-Parallel I/O



- Non-parallel
- Performance worse than sequential
- Legacy from before application was parallelized
- Either MPI or not

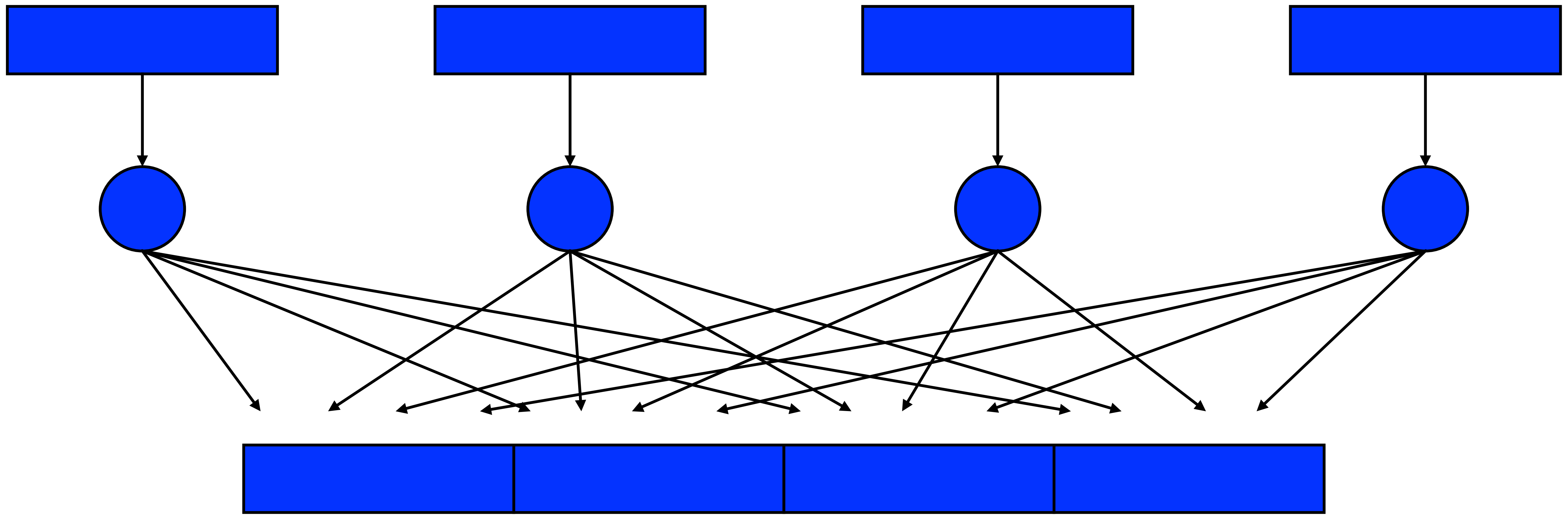
Independent Parallel I/O

- Each process writes to a separate file



- Pro: parallelism
- Con: lots of small files to manage
- Legacy from before MPI
- MPI or not

Cooperative Parallel I/O



- Parallelism
- Can only be expressed in MPI
- Natural once you get used to it

Why MPI is a natural fit for I/O

- Writing is similar to sending messages
- Reading is similar to receiving messages
- Parallel I/O needs:
  - Collective operations
  - User-defined datatypes for both memory and file layouts
  - Communicators to separate application level message passing from I/O related message passing
  - Non-blocking operations

Parallel I/O

- What do we mean by parallel I/O?
  - Concurrent reads to multiple processes from a common file
  - Concurrent writes from multiple processes to a common file
  - A parallel file system and hardware that support these concurrent accesses

Simple IO Process

- Standard I/O:
  - Open the file
  - Read or write data
  - Close the file
- Parallel I/O with MPI:
  - Open the file : MPI_File_open
  - Write to the file: MPI_File_write
  - Close the file: MPI_File_close

File Open

- Collective over communicator
- Used to support collective I/O, which is important for performance
- Modes are similar to Unix open
- MPI_Info variable can provide additional hints for performance

File write

- Independent (example shows this… only called write from one rank)
- Use MPI_File_write or MPI_File_write_at
- Use MPI_MODE_WRONLY or MPI_MODE_RDWR
- If the file doesn’t previously exist, must also use MPI_MODE_CREATE
- Can pass multiple flags with ‘|’ in C or ‘+’ in Fortran

Ways to Access a Shared File

- **`MPI_File_seek`**
- **`MPI_File_read`**
- **`MPI_File_write`**
- **`MPI_File_read_at`**
- **`MPI_File_write_at`**
- **`MPI_File_read_shared`**
- **`MPI_File_write_shared`**

like Unix I/O

combine seek and I/O
for thread safety

use shared file pointer

PARALLEL@ILLINOIS

File close

- Like open, close is also a collective operation
- This operation is similar to MPI_Comm_free

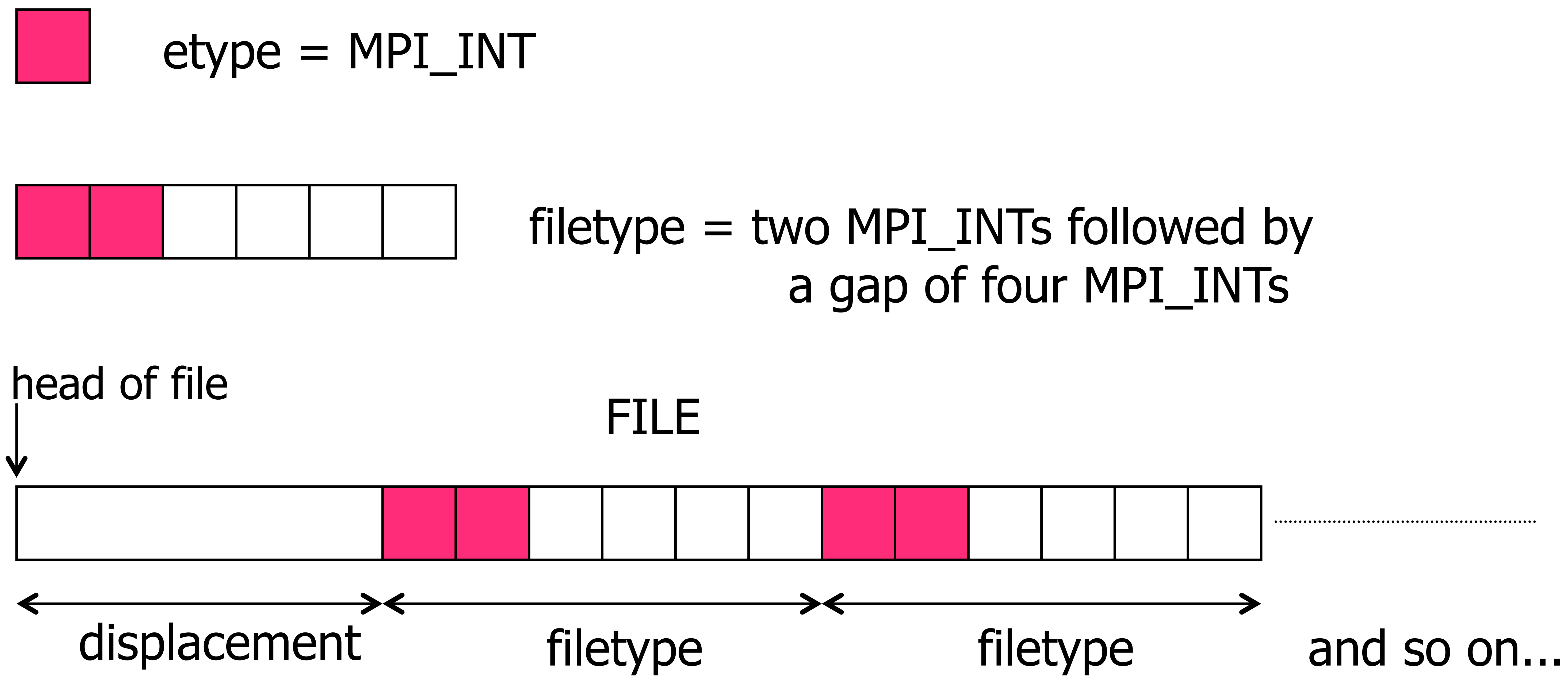
Why independent I/O?

- Collectives add synchronization… if all processes are highly unsynchronized during I/O, collectives could add high cost
- The overhead of collective calls (especially for small messages) can out weight their benefits

Noncontiguous I/O

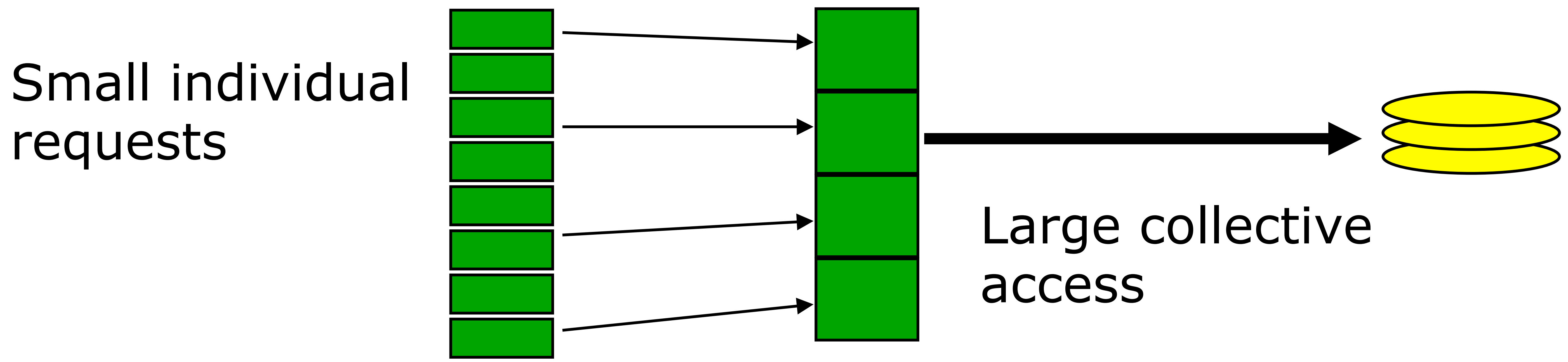
- Each process describes the part of the file for which it is responsible
- Only the part of the file described by the file view is visible to the process; read and write will access these locations
  - 'File view' created with MPI_File_set_view
    - Displacement : number of bytes to be skipped from start of file (e.g. to skip a file header)
    - Etype : basic unit of data access (basic or derived datatype)
    - Filetype : which portion of the file is visible to the process?

A Simple Noncontiguous File View Example



Collective I/O

- Allows communication of the big picture to a file system
- Basic idea : build large blocks so that reads and writes will be large
  - Requests from different processes may be merged together
  - Particularly effective when the accesses of different processes are noncontiguous and interleaved



PARALLEL@ILLINOIS

Collective I/O

- MPI_File_write_at_all
    - _all says that all processes in the communicator (which was passed to MPI_File_open) will call this function
- Each process specifies its own information for the write
    - Calls the same information as if writing without a collective, but tells MPI to aggregate data

The Other Collective I/O Calls

- **MPI_File_seek**
- **MPI_File_read_all**
- **MPI_File_write_all**

like Unix I/O

- **MPI_File_read_at_all**
- **MPI_File_write_at_all**

combine seek and I/O for thread safety

- **MPI_File_read_ordered**
- **MPI_File_write_ordered**

use shared file pointer

File Systems

- NFS (Network File System) : distributed file system
  - Allows a user on a computer to access files from some storage over a computer network
  - Typically only uses a single node

Parallel File Systems

- PNFS (Parallel Network File System):
  - Scalable parallel access to files distributed among multiple servers
  - Separates data and metadata
  - Moves metadata server out of the data path