# 平行計算與程式 補充

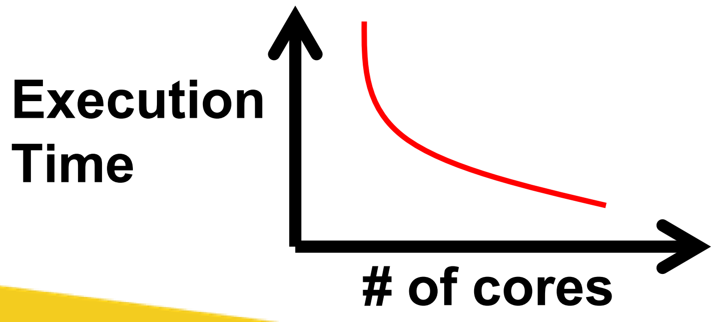
國家高速網路與計算中心
National Center for High-performance Computing

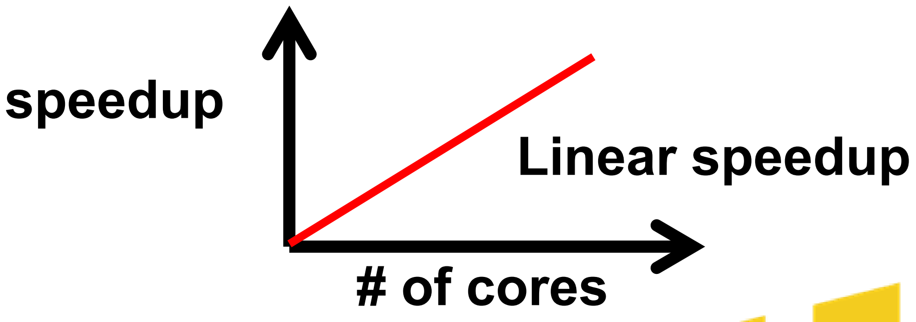
國立清華大學叢集電腦競賽團隊
Student Cluster Competition Team of NTHU

# Strong Scalability vs Weak Scalability

# Strong Scaling

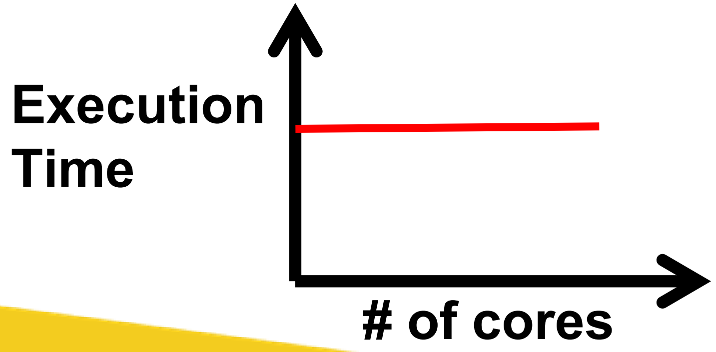
- The problem size stays fixed but the number of processing elements are increased.
- It is used to find a "sweet spot" that allows the computation to complete in a reasonable amount of time, yet does not waste too many cycles due to parallel overhead.
- Linear scaling is achieve if the speedup is equal to the number of processing elements.

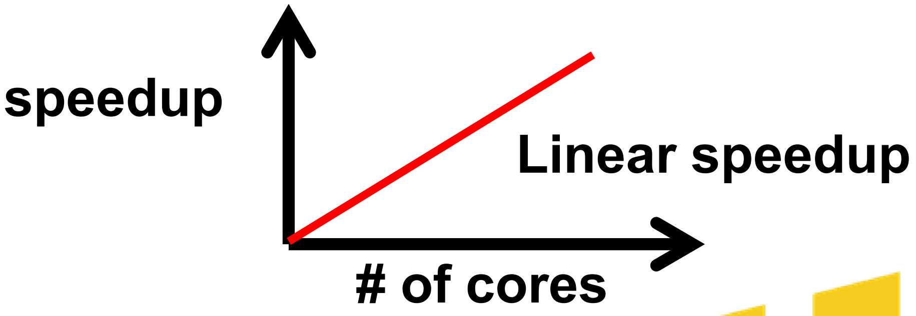


## Weak Scaling

- The problem size (workload) assigned to each processing element stays fixed and additional processing elements are used to solve a larger total problem
- It is a justification for programs that take a lot of memory or other system resources (e.g., a problem wouldn't fit in RAM on a single node)
- Linear scaling is achieved if the run time stays constant while the workload is increased



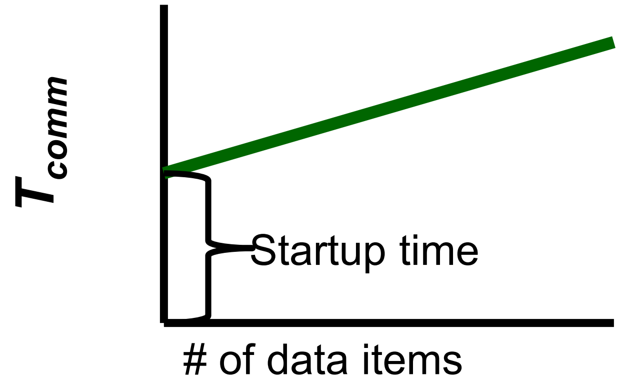
# Strong Scaling vs. Weak Scaling

- Strong scaling
  - Linear scaling is harder to achieve, because of the communication overhead may increase proportional to the scale
- Weak scaling
  - Linear scaling is easier to achieve because programs typically employ nearest-neighbor communication patterns where the communication overhead is relatively constant regardless of the number of processes used

# Time Complexity Analysis

# Time Complexity Analysis

- $T_p = T_{comp} + T_{comm}$
  - $T_p$: Total execution time of a parallel algorithm
  - $T_{comp}$: Computation part
  - $T_{comm}$: Communication part
- $T_{comm} = q\ (T_{startup} + n\ T_{data})$
  - $T_{startup}$: Message latency (assumed constant)
  - $T_{data}$: Transmission time to send one data item
  - n: Number of data items in a message
  - q: Number of message

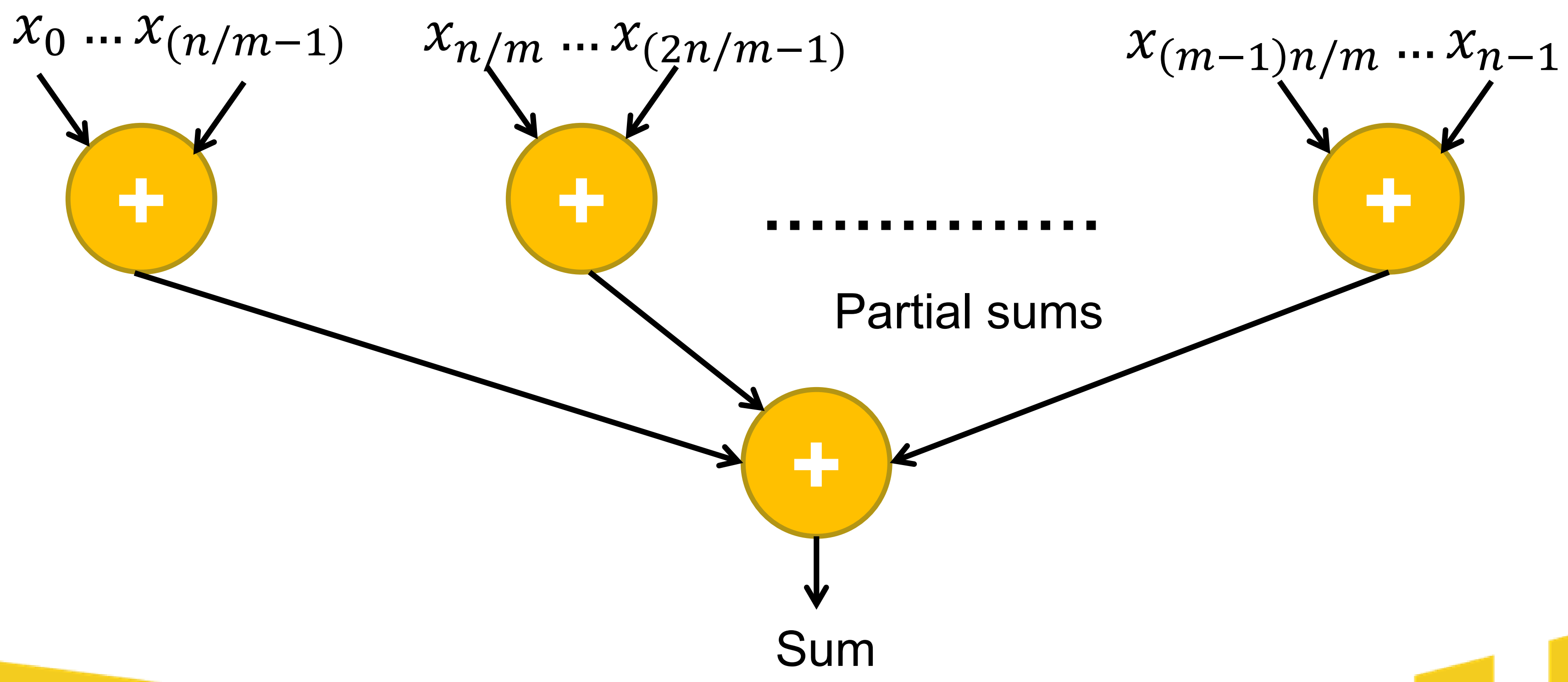


# Time Complexity Example 1

- Algorithm phase:
  1. Computer 1 sends $n/2$ numbers to computer 2
  2. Both computers add $n/2$ numbers simultaneously
  3. Computer 2 sends its partial result back to computer 1
  4. Computer 1 adds the partial sums to produce the final result
- Complexity analysis:
  - Computation (for step 2 & 4):
    - $T_{comp} = n/2 + 1 = O(n)$
  - Communication (for step 1 & 3):
    - $T_{comm} = (T_{startup} + n/2 \times T_{data}) + (T_{startup} + T_{data})$
      $= 2T_{startup} + (n/2 + 1)\ T_{data} = O(n)$
  - Overall complexity: O(n)

# Time Complexity Example 2

- Adding *n* numbers using m processes
    - Evenly partition numbers to processes



# Time Complexity Example 2

Adding *n* numbers using ***m*** processes

- Sequential: $O(n)$
- Parallel:
  - Phase1: Send numbers to slaves

$$t_{comm1} = m(t_{startup} + (n/m)t_{data})$$

  - Phase2: Compute partial sum

$$t_{comp1} = n/m - 1$$

  - Phase3: Send results to master

$$t_{comm2} = m(t_{startup} + t_{data})$$

  - Phase4: Compute final accumulation

$$t_{comp2} = m - 1$$

  - Overall:

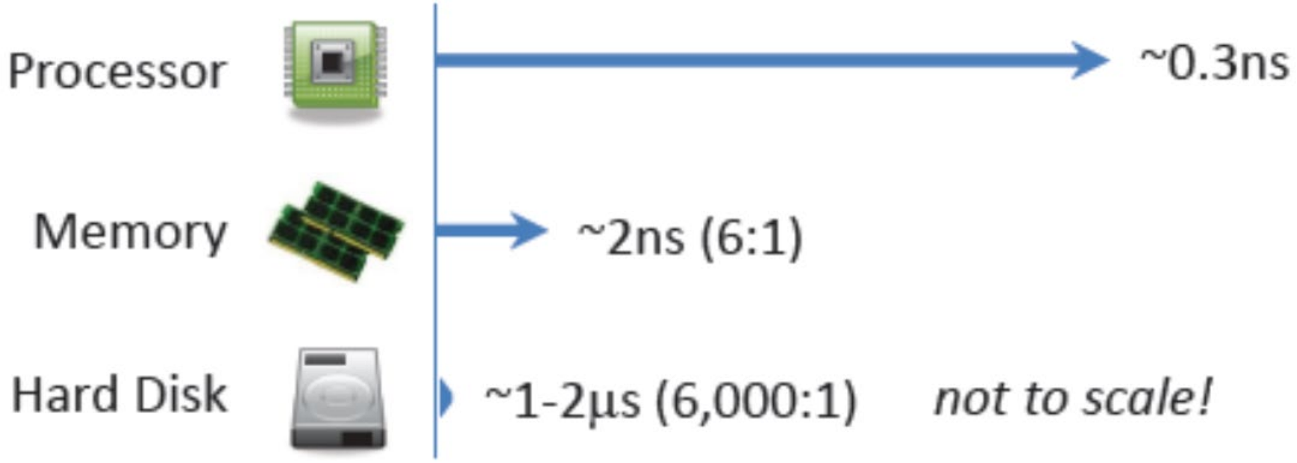
$$t_p = 2mt_{startup} + (n+m)t_{data} + m + \frac{n}{m} - 2 = O(m + n/m)$$

Tradeoff between **computation** & **communication**

# MPI & Parallel IO

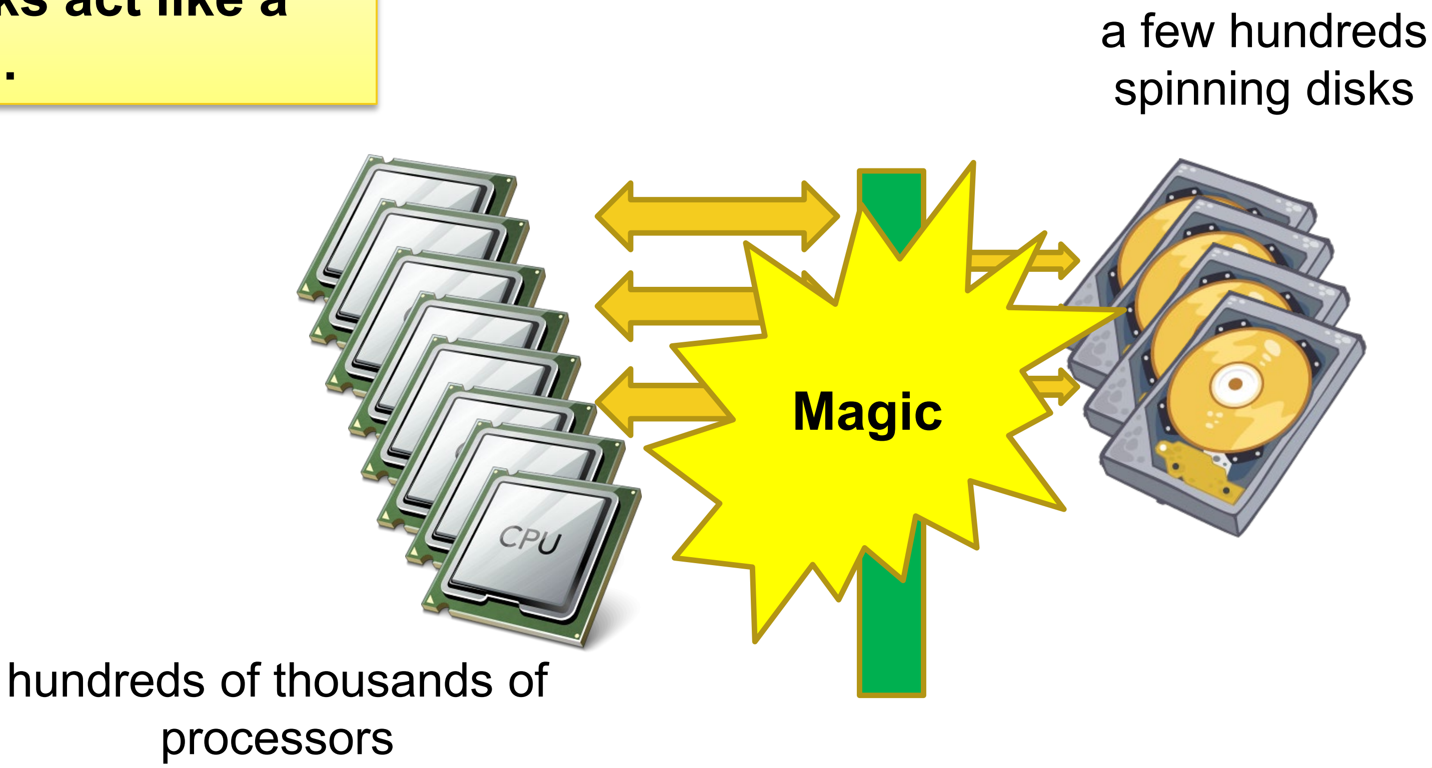
# Relative Speed of Components in HPC Platform

- An HPC platform's I/O  subsystems are typically slow as compared to its other parts
- The I/O gap between memory speed and average disk access stands at roughly $10^{-3}$



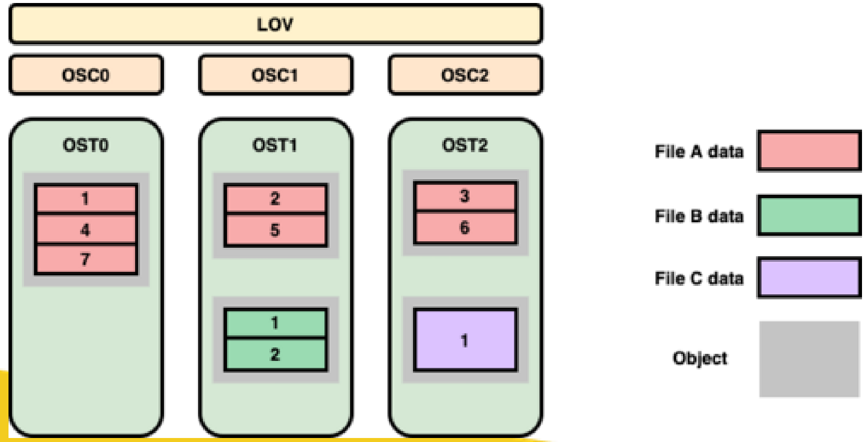
# Concurrent Data Access in a Cluster

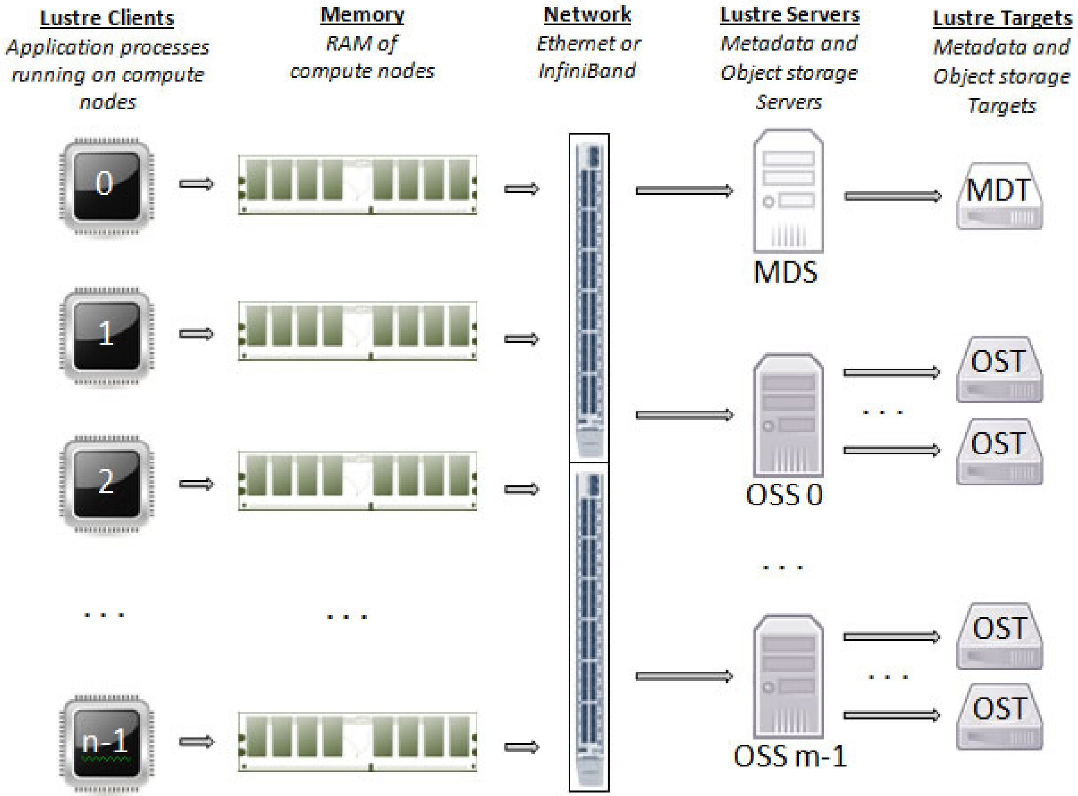
**We need some magic to make the collection of spinning disks act like a single disk …**



# Parallel File Systems: Lustre

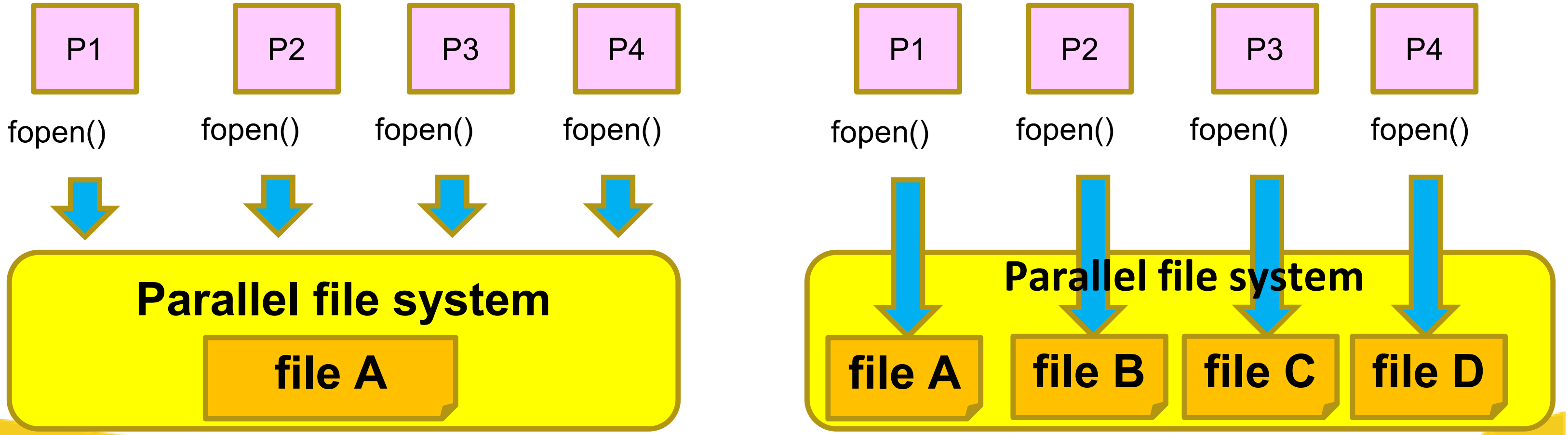
- Separate control plan (metadata) and data plan (data)
- Distributed system architecture
  - Multiple MDS and OSS servers
- Simplify the task of storage node
- Parallel I/O on a single file
  - Files are chunked & stripped
  - Stipe size & count is configured by users



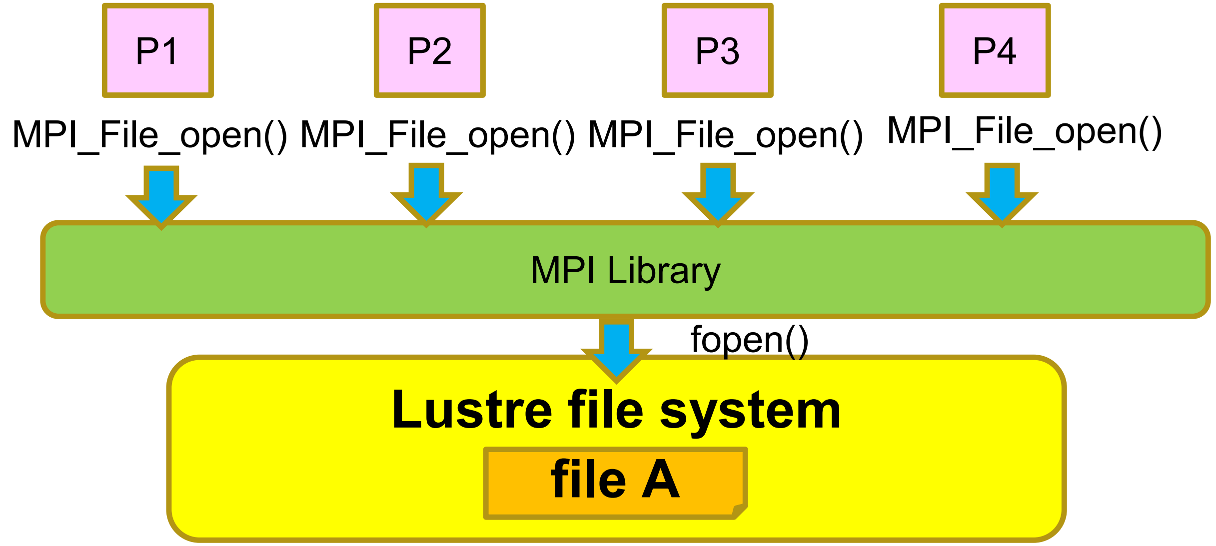
# POSIX File Access Operations

- POSIX file system call “fopen()”:
  - The same is opened by each processes ➔ multiple file handlers across your MPI processes
  - Open the same file with read permission is OK
  - But can’t open with write permission together due file system locking mechanism ➔ data inconsistency
  - To write simultaneously must create multiple files (can’t take advantage of parallel file system & hard to manage)



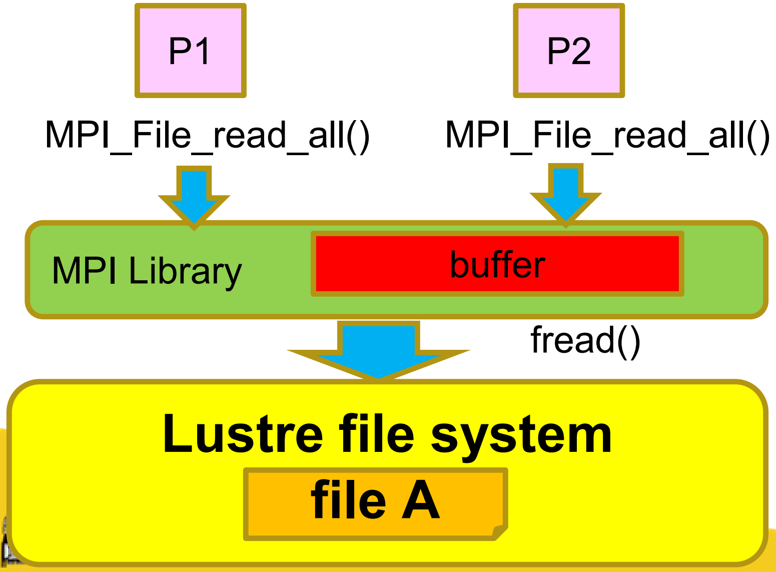
# MPI-IO File Access Operations

- MPI-IO call "MPI_File_open()"
  - File is opened only once in a collective manner
  - MPI library will share and synchronize with each other to use the same file handler
  - Can handle both read and write together

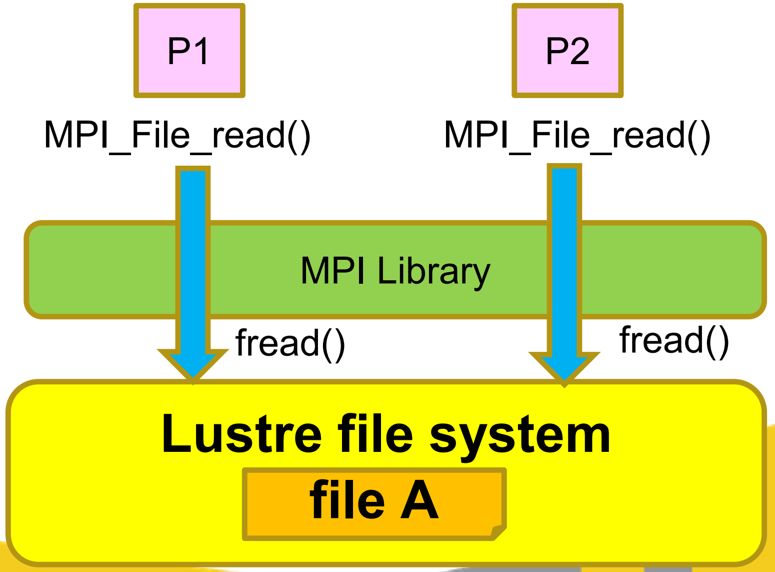


# MPI-IO Independent/Collective I/O

- Collective I/O
  - Read/write to a shared memory buffer, then issue ONE file request
  - Reduce #I/O request
    ➔ Good for small I/O
  - Require synchronization




- Independent I/O
  - Read/write individually
  - Prevent synchronization
  - One request per process
  - Request is serialized if access the same OST
  - Good for large I/O



# MPI-IO API

- MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh)
    - Open a file
- MPI_File_close(MPI_File *fh)
    - Close a file
- MPI_File_read/write(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)
    - Independent read/write using individual file pointer
- MPI_File_read/write_all(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)
    - Collectively read/write using individual file pointer
- MPI_File_sync(MPI_File fh)
    - Flush all previous writes to the storage device

# Scientific Data Format: NetCDF & HDF5

- What is a Scientific Data Format?
  - A data format for scientist to store, access & operate their data easily and efficiently
- Key requirements:
  - Self-Describing: A file includes information about the data it contains.
  - Portable: A file can be accessed by computers with different ways of storing integers, characters, and floating-point numbers.
  - Scalable: Small subsets of large datasets in various formats may be accessed efficiently through file interfaces, even from remote servers.
  - Appendable: Data may be appended to a properly structured file without copying the dataset or redefining its structure.
  - Sharable: One writer and multiple readers may simultaneously access the same file.
  - Archivable: Access to all earlier forms of data will be supported by current and future versions of the software.

# Scientific Data Format: NetCDF & HDF5

- What is a Scientific Data Format?
  - A data format for scientist to store, access & operate their data easily and efficiently
- Key requirements:
  - Se ns.
  - Po f storing
    in
  - Sc accessed
    ef
  - Ap vithout copying
    the dataset or redefining its structure.
  - Sharable: One writer and multiple readers may simultaneously access the same file.
  - Archivable: Access to all earlier forms of data will be supported by current and future versions of the software.

Can you achieve these with a complex doc file or a plain text file?

# Scientific Data Format: NetCDF & HDF5

- Key features
  - A file contains its own directory (tree) structure
  - Each dataset is a multi-dimensional array
    - The dimension and size can be configured & changed
  - Each file entity (group & dataset) is self-describe
    - By its own metadata & attributes
  - The mapping between the dataset and disk layout can be controlled
    - Column or low major
- Visualization tools
  - HDFView
  - https://www.hdfgroup.org/downloads/hdfview/