Data Management

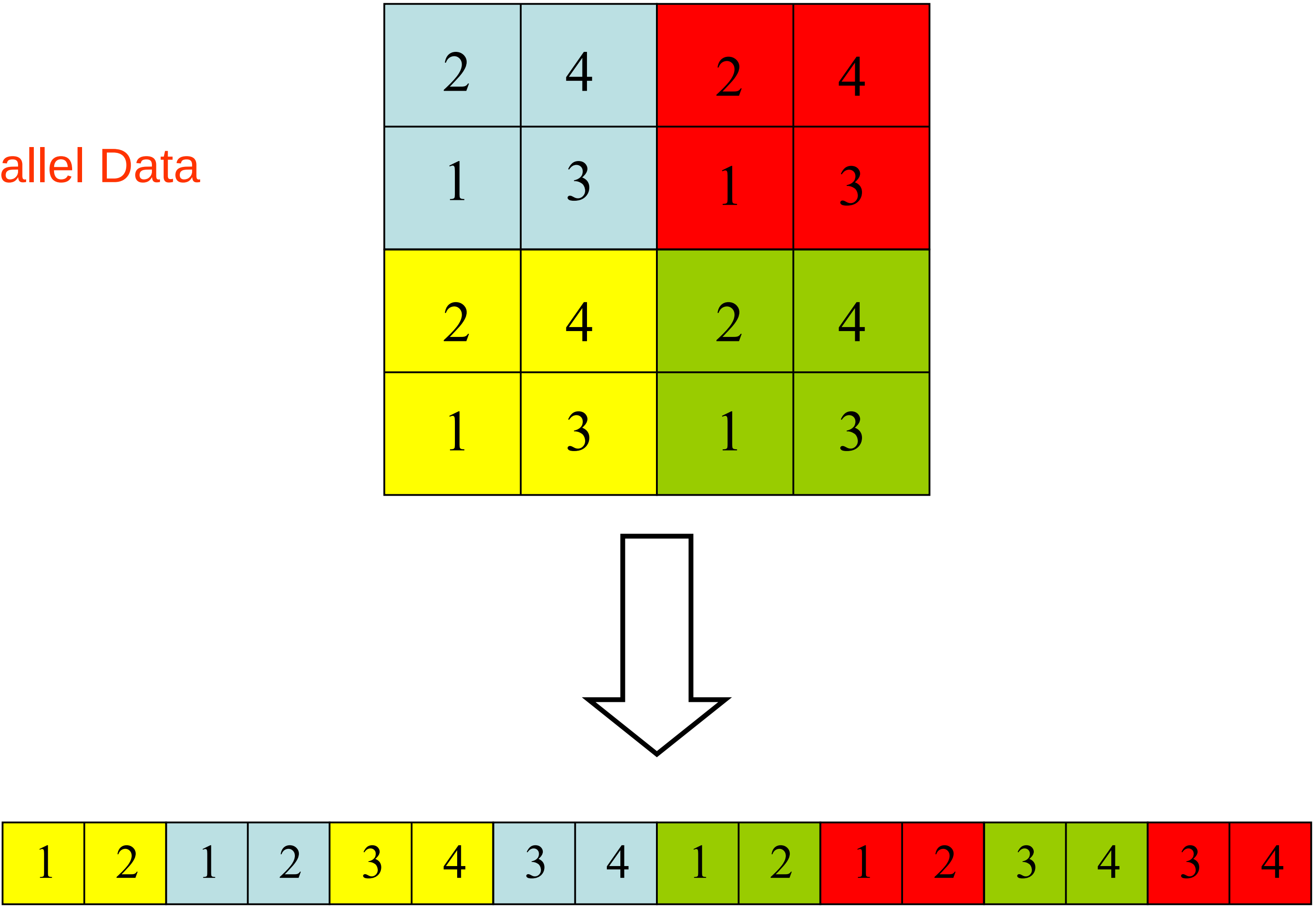
Parallel IO Libraries

Dr David Henty
HPC Training and Support
d.henty@epcc.ed.ac.uk
+44 131 650 5960

Overview

- Lecture will cover
  - general parallel IO challenge
  - describing data layout
  - MPI-IO
  - collective IO
  - parallel HDF5 and NetCDF

4x4 array on 2x2 Process Grid



Parallel Data

2	4	2	4
1	3	1	3
2	4	2	4
1	3	1	3

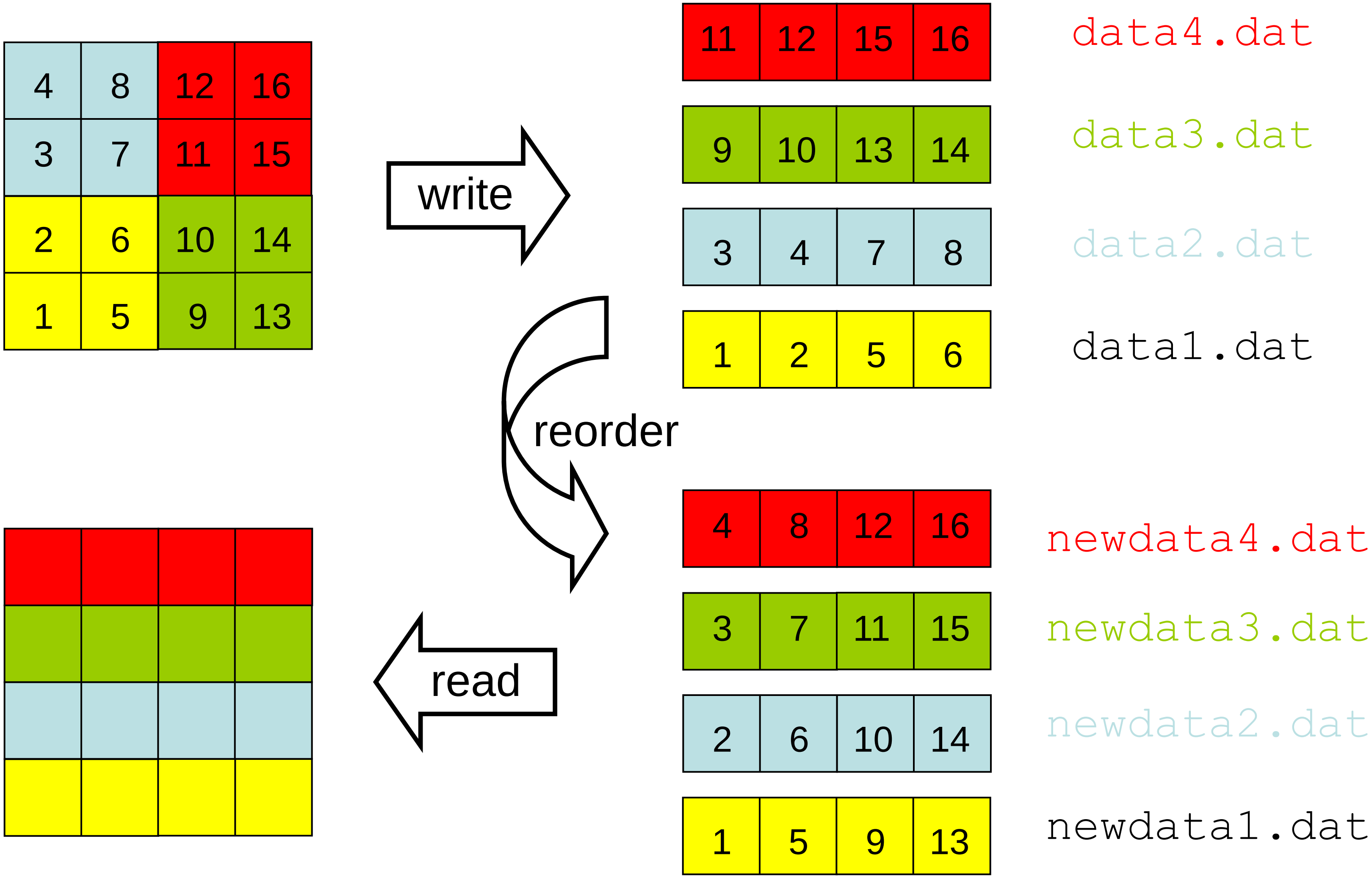
File

1	2	1	2	3	4	3	4	1	2	1	2	3	4	3	4

Message Passing: Naive Solutions

- Master IO
  - send all data to/from master and write/read a single file
  - quickly run out of memory on the master
    - or have to write in many small chunks
  - does not benefit from a parallel fs that supports multiple write streams
- Separate files
  - each process writes to a local fs and user copies back to home
  - or each process opens a unique file (dataXXX.dat) on shared fs
- Major problem with separate files is reassembling data
  - file contents dependent on number of CPUs and decomposition
  - pre / post-processing steps needed to change number of processes
  - but at least this approach means that reads and writes are in parallel
- But may overload filesystem for many processes
  - e.g. MDS cannot keep up with requests

2x2 to 1x4 Redistribution



4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

11	12	15	16	data4.dat
9	10	13	14	data3.dat
3	4	7	8	data2.dat
1	2	5	6	data1.dat

4	8	12	16	newdata4.dat
3	7	11	15	newdata3.dat
2	6	10	14	newdata2.dat
1	5	9	13	newdata1.dat

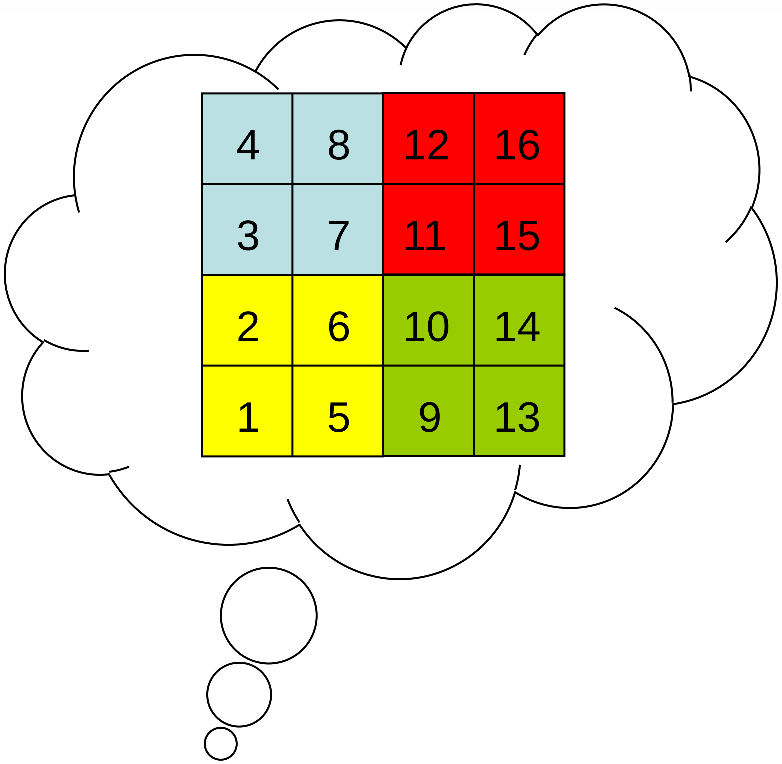
Information on Data Layout

- What does the IO system need to know about the data?
  - how the local arrays should be stitched together to form the file

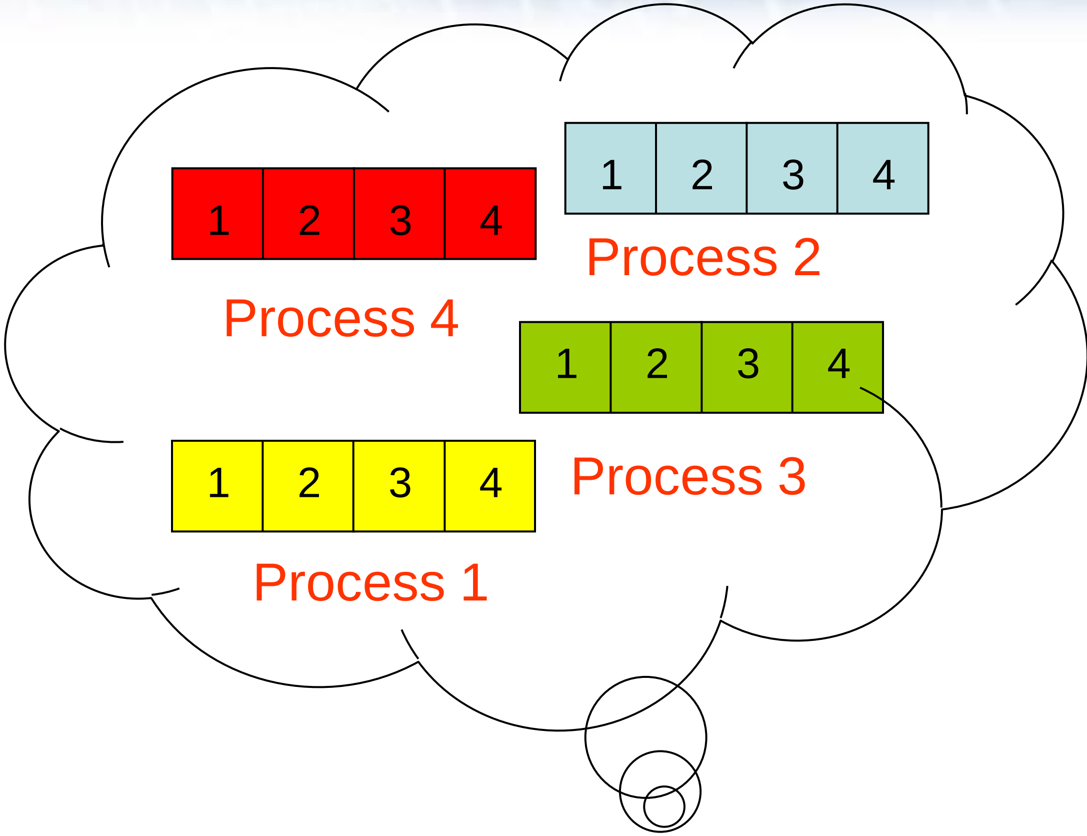
- But ...
  - mapping from local data to the global file is only in the mind of the programmer!
  - the program does not know that we imagine the processes to be arranged in a 2D grid

- How do we describe data layout to the IO system
- This is the crucial step in parallel IO, e.g.
  - MPI-IO used derived datatypes
  - HDF5 uses hyperslabs

Programmer View vs Machine View



4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13



1	2	3	4

Process 2

1	2	3	4

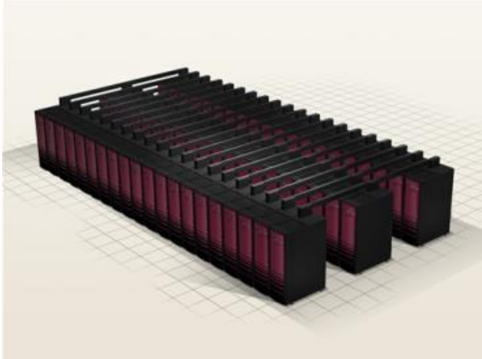
Process 4

1	2	3	4

Process 3

1	2	3	4

Process 1



Files vs Arrays

- Think of the file as a large array
  - forget that IO actually goes to disk
  - imagine we are recreating a single large array on a master process

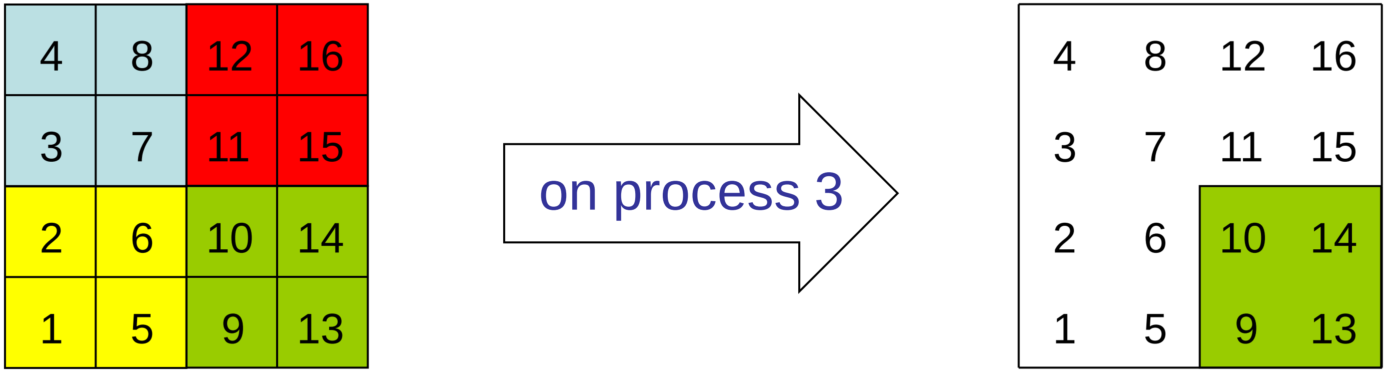
- The IO system must create this array and save to disk
  - without running out of memory
    - never actually creating the entire array
    - ie without doing naive master IO
  - and by doing a small number of large IO operations
    - merge data to write large contiguous sections at a time
  - utilising any parallel features
    - doing multiple simultaneous writes if there are multiple IO nodes
    - managing any coherency issues re file blocks

MPI-IO Approach

- MPI-IO is part of the MPI standard
  - http://www.mpi-forum.org/docs/docs.html

- Each process needs to describe what subsection of the global array it holds
  - it is entirely up to the programmer to ensure that these do not overlap for write operations!
- Programmer needs to be able to pass system-specific information
  - pass an **`info`** object to all calls

Data Sections



4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

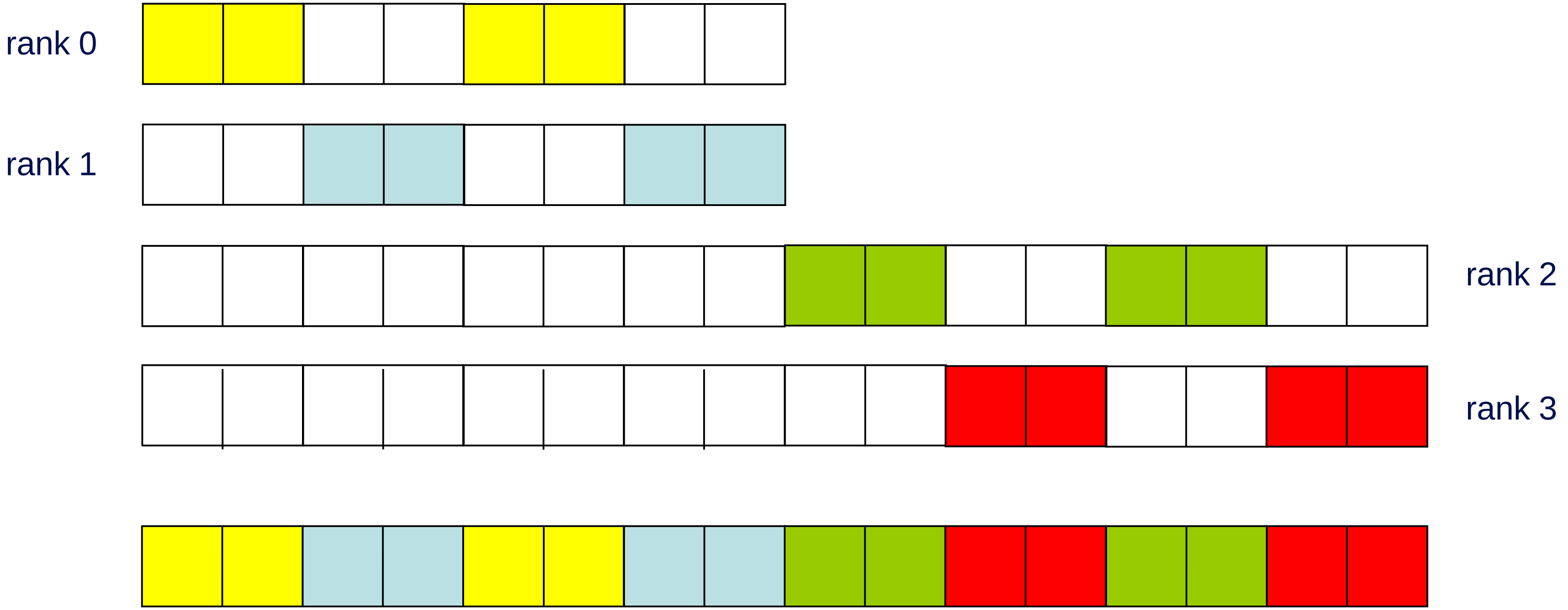
- Describe 2x2 subsection of 4x4 array
- Using standard MPI derived datatypes
- A number of different ways to do this
  - e.g. MPI vector types or subarray types

- This is called the *filetype*

Filetypes Should Tile the File

4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

rank 1 (0,1)	rank 3 (1,1)
rank 0 (0,0)	rank 2 (1,0)



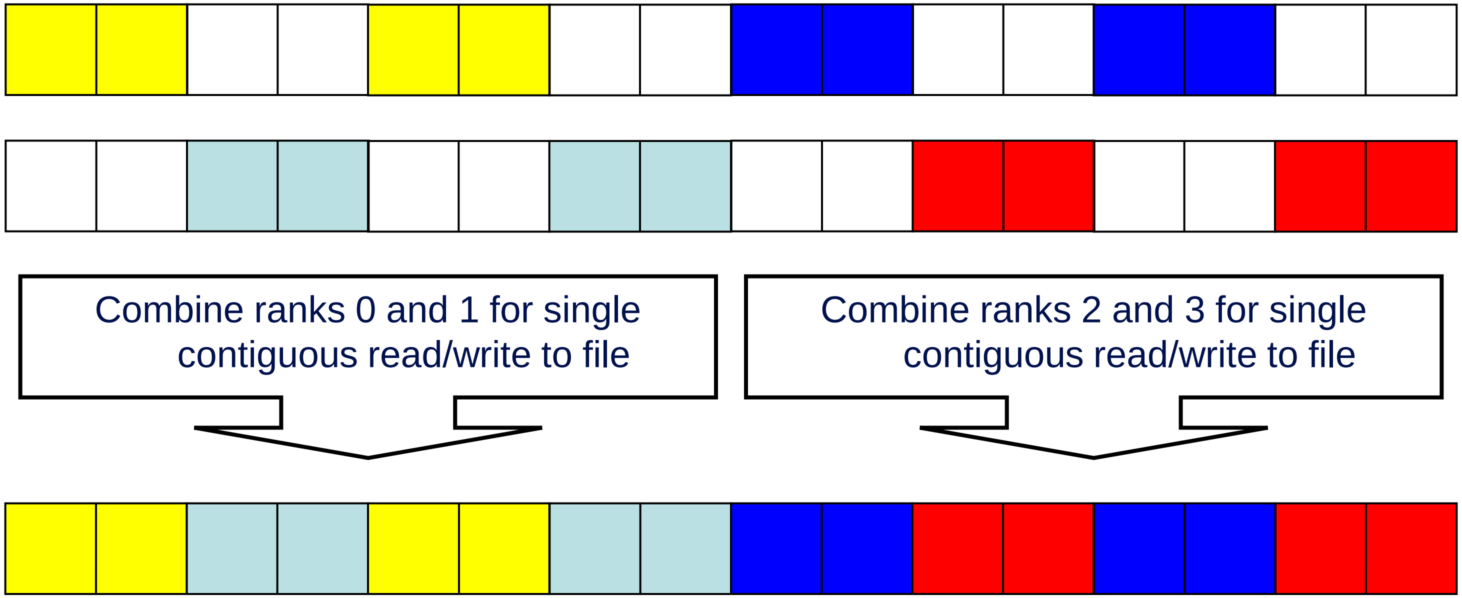
Performance

- If each IO transaction is performed individually ...
  - very slow performance
  - large number of small IO transactions
  - lots of activity on Meta Data Server (open, close, lock, seek, ...)

- Key is to do Collective IO
  - common feature of all parallel libraries

Collective MPI-IO

- For read and write, “`_all`” means operation is collective
  - all processes attached to the file are taking part
- Other MPI-IO routines exist which are individual
  - functionality is the same but performance will be slower
  - collective routines can aggregate reads/writes for better performance



Combine ranks 0 and 1 for single contiguous read/write to file

Combine ranks 2 and 3 for single contiguous read/write to file

MPI-IO Issues

- Files are raw bytes
  - no header information
  - storage is architecture-specific (e.g. big / little-endian floating-point)
- Difficult to cope with in other codes downstream
  - user must write their own post-processing tools
  - c.f. cioview / fioview with “metadata” encoded in file name!
- But ...
  - it can be very fast!

Solution

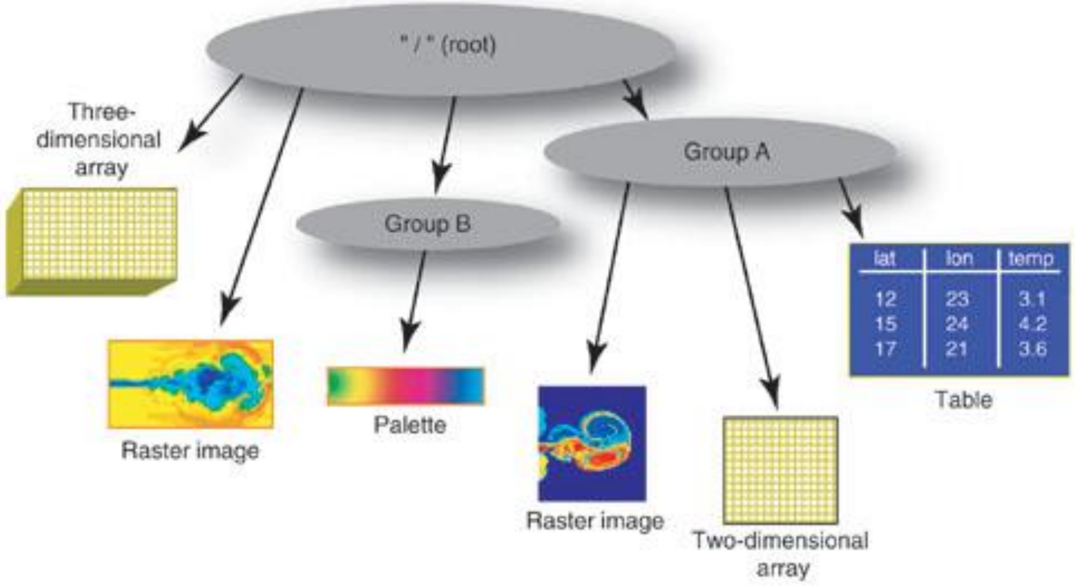
- For functionality
  - define higher-level formats
  - include metadata, e.g. “this is a 4x5x7 array of doubles”
  - enables standard data converters, browsers, viewers etc.

- For performance
  - layer on top of MPI-IO

- Many real applications use higher-level formats
  - understanding MPI-IO will enable you to get performance as well

HDF5

- “**Hierarchical Data Format** (**HDF**) is a set of file formats (**HDF4**, **HDF5**) designed to store and organize large amounts of data.” (Wikipedia)
  - data arranged like a Unix file system
  - self-describing
  - hierarchical
  - can use MPI-IO



lat	lon	temp
12	23	3.1
15	24	4.2
17	21	3.6

image taken from www.hdfgroup.org

Parallel HDF5 (Fortran)

- Approach much like MPI-IO
  - describe global dataset
  - s describes its local portion(s) of the glo

```
CALL h5sselect_hyperslab_f(filespace, &
          H5S_SELECT_SET_F, offset, &
          count, error)
```

MPI_ORDER_FORTRAN

global data, encodes sizes

starts

subsizes

- Then call collective write
  - hyperslabs can be merged to create global file
  - actual file IO done through MPI-IO
  - important to choose collective IO

NetCDF: Network Common Data Form

- "a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data.." (Wikipedia)
  - more restricted than HDF5
  - common in certain communities
    - climate research
    - oceanography
    - GIS ...

- Rich set of tools
  - data manipulation
  - visualisation
  - ...

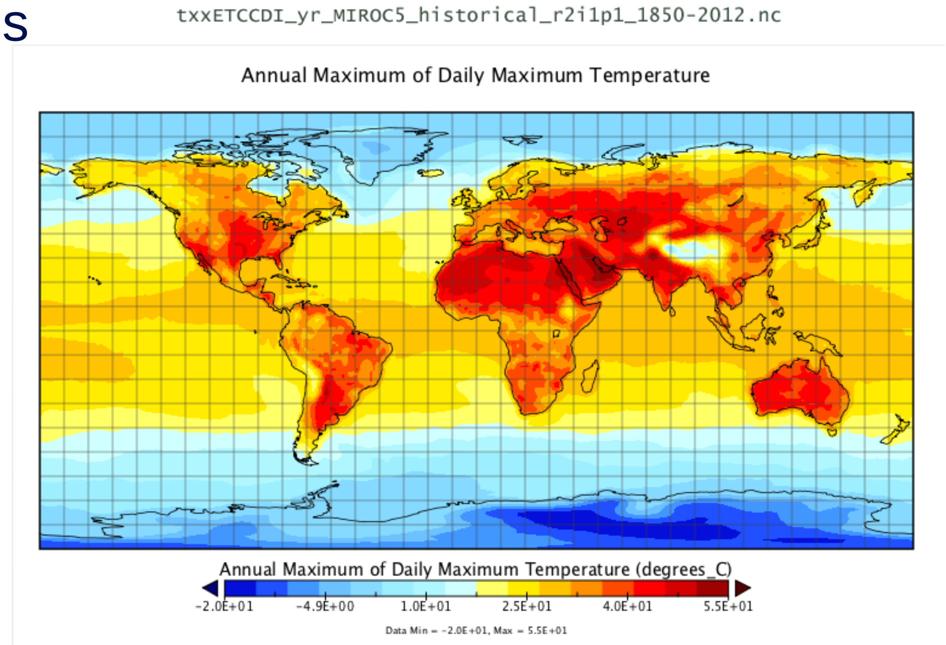
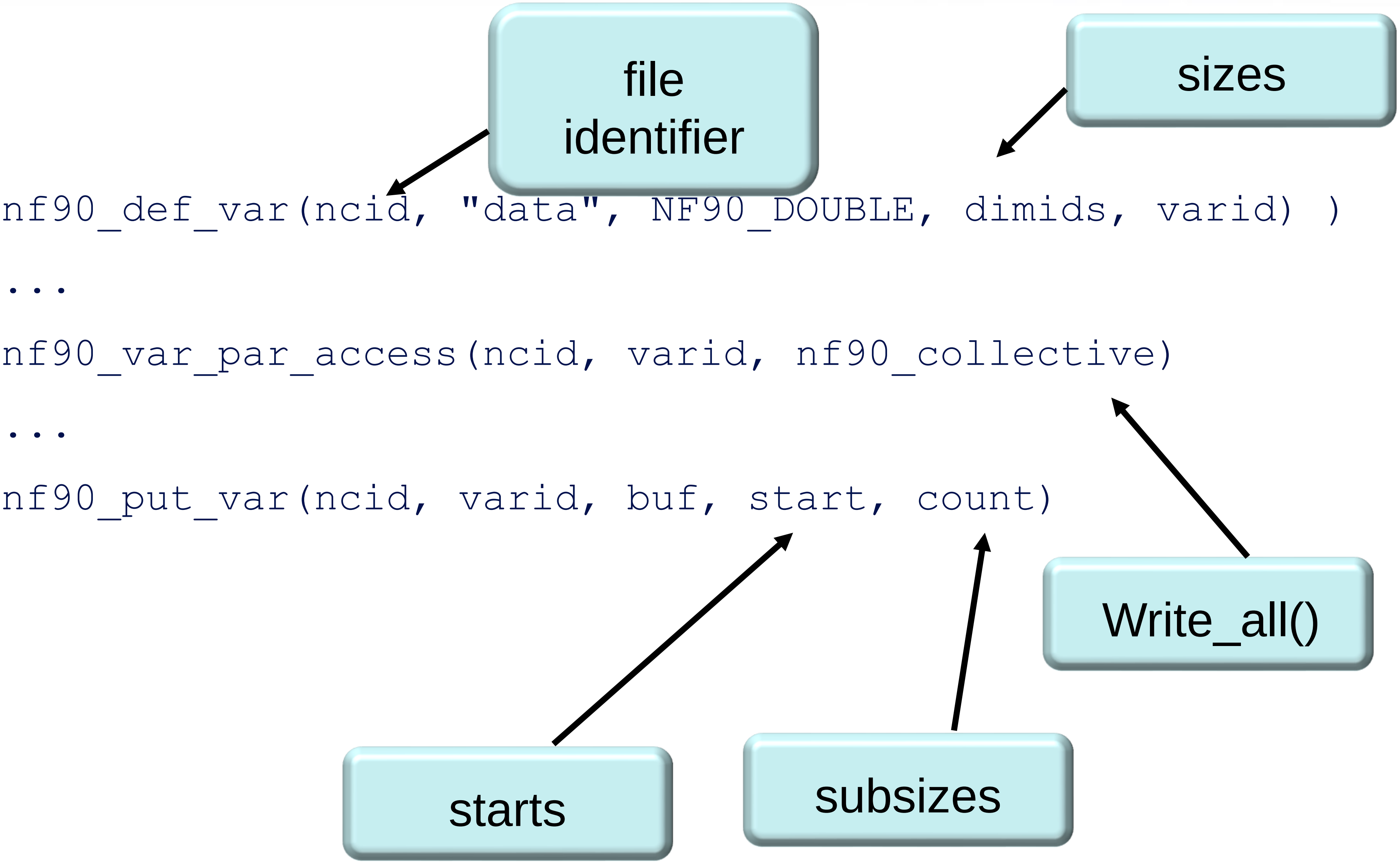

txxETCCDI_yr_MIROC5_historical_r2i1p1_1850-2012.nc
Annual Maximum of Daily Maximum Temperature
Annual Maximum of Daily Maximum Temperature (degrees_C)
-2.0E+01
-4.9E+00
1.0E+01
2.5E+01
4.0E+01
5.5E+01
Data Min = -2.0E+01, Max = 5.5E+01

image taken from http://live.osgeo.org

Parallel NetCDF (Fortran)



file identifier

sizes

```
nf90_def_var(ncid, "data", NF90_DOUBLE, dimids, varid) )

...

nf90_var_par_access(ncid, varid, nf90_collective)

...

nf90_put_var(ncid, varid, buf, start, count)
```

Write_all()

starts

subsizes

Accessing libraries

- ARCHER
  - HDF5

    -
      ```
      user@archer:~> module load cray-hdf5-parallel
      ```
    - interfaces to Cray MPI-IO
  - NetCDF

    -
      ```
      user@archer:~> module load cray-netcdf-hdf5parallel
      ```
    - interfaces to HDF5 ...
    - ... which interfaces to Cray MPI-IO
- DAC
  - HDF5
    - should compile by default
  - NetCDF
    - module load netcdf-hdf5parallel
    - mpicc `pkg-config --cflags --libs netcdf`

Summary

- MPI-IO may seem a little low-level
  - but is building block of parallel IO on ARCHER and DAC
- Higher-level formats layer on top of MPI-IO
  - to benefit from performance work by Cray, Lustre etc.
- Common formats are HDF5 and NetCDF
  - both supported on ARCHER and DAC
- Understanding MPI-IO performance is key to getting good performance for HDF5 and NetCDF
  - collective IO is crucial to obtain performance