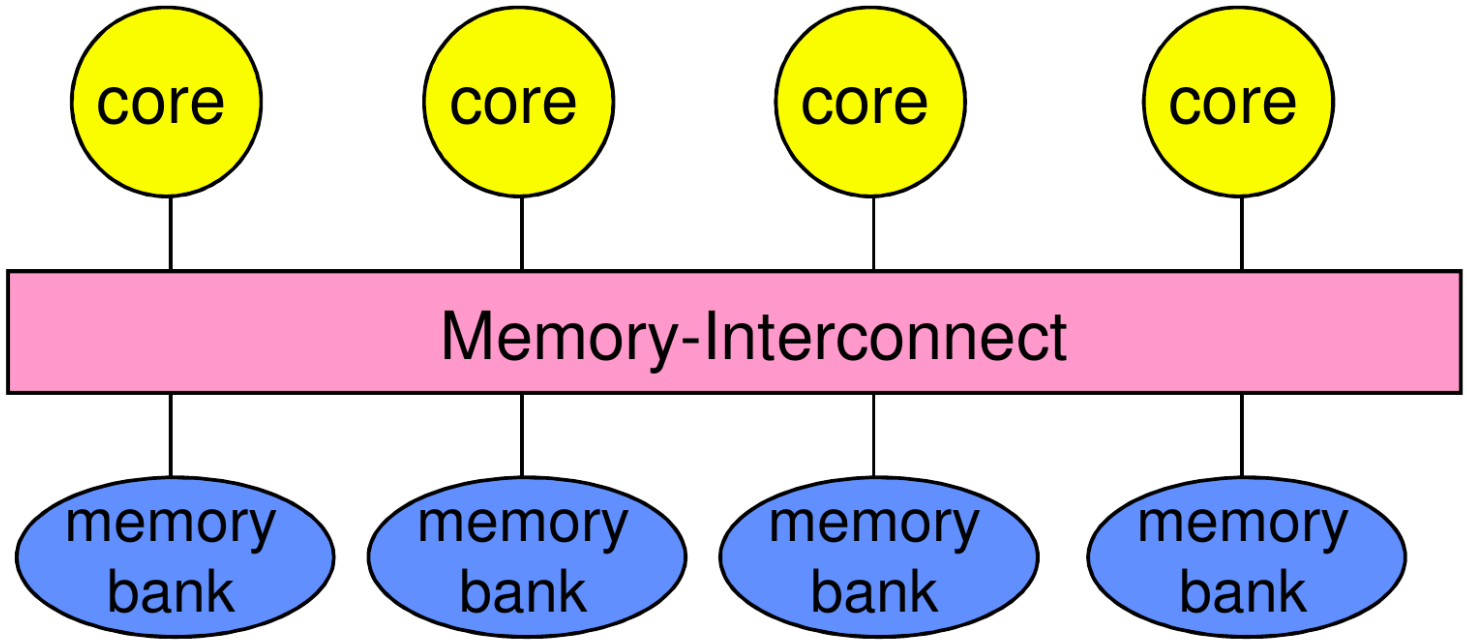
Intro to Parallel Programming and HDF5

Ángel de Vicente

November 19, 2019

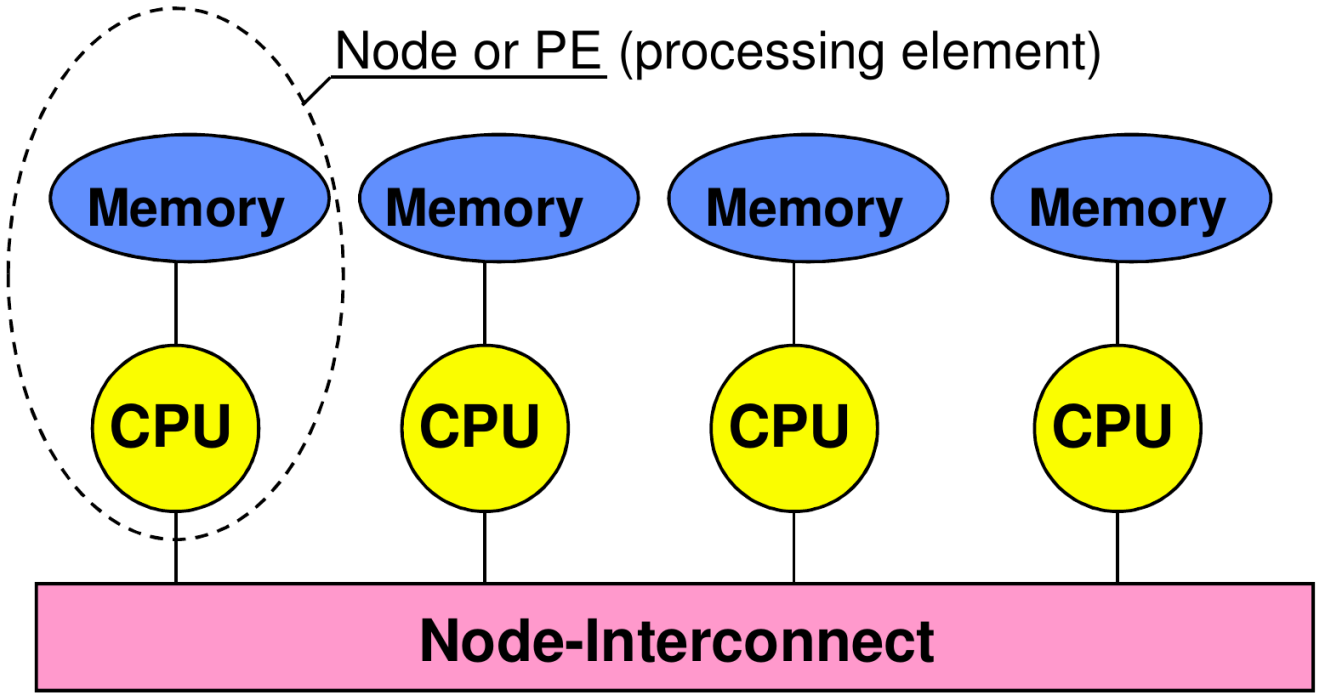
Topic

Multiprocessors (shared memory)



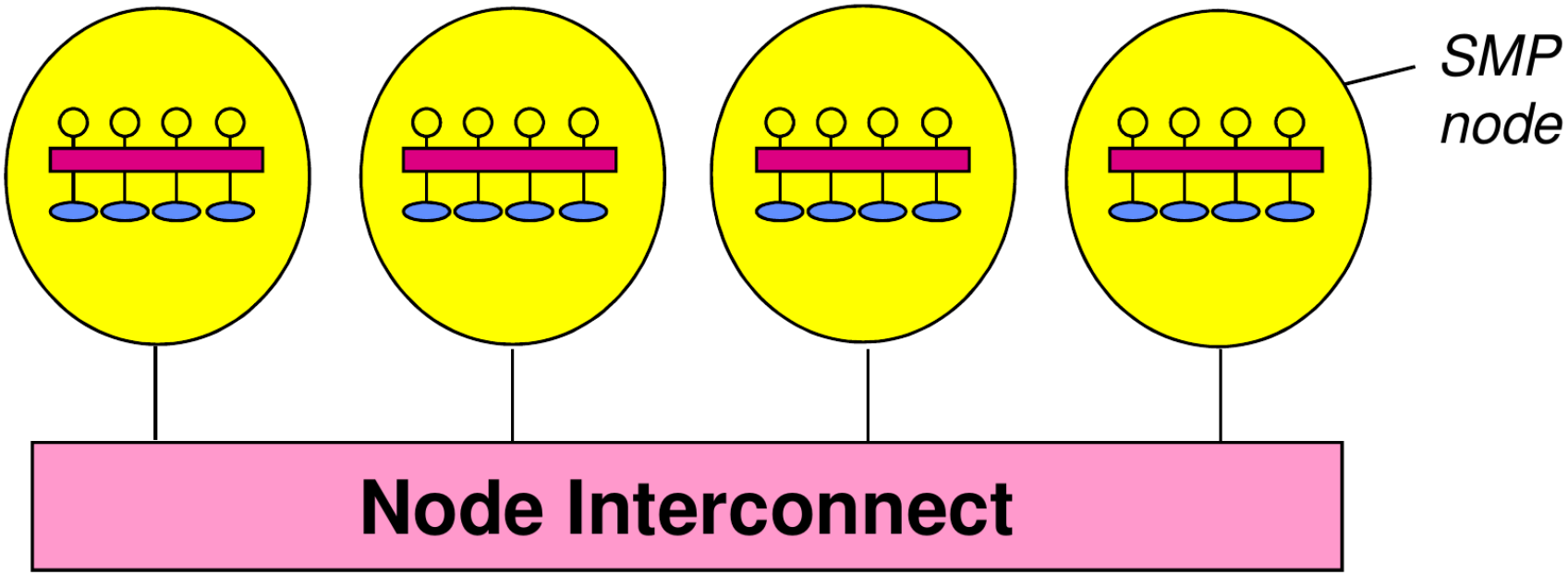
- All cores have access to all memory at the same speed
- Symmetric multiprocessor (SMP)
- These days basically all your workstations and laptops.

Multicomputers (distributed memory)



- Each node
  - direct access to its own memory
  - slow (and indirect) access to other CPU's memories
- Nodes connected via a node-interconnect
  - Ethernet, InfiniBand, Omni-Path
  - BUS, torus, dragonfly, crossbar, etc.

Hybrid architectures



- Modern HPC systems (supercomputers) are almost always hybrid architectures: clusters of SMP nodes.
  - SMP within each node
  - Distributed memory between nodes

LaPalma3 Supercomputer

http://research.iac.es/sieinvens/siepedia/pmwiki.php?n=HOWTOs.LaPalma3Intro

83,85 TFlops
4032 cores
7,88 TBytes mem
346 TBytes disco

- 252 IBM dx360 M4 compute nodes.
- Each node:
  - 16 cores (Intel E5-2670) @ 2.6 GHz
  - 32 GB of RAM (2 GB per core) and 500GB of local disk storage.
- 346 TB of disk storage via Lustre Parallel File System.
- Interconnection networks:
  - Infiniband FDR10 40G Network: used by parallel applications communications.
  - Gigabit Network: Ethernet network for Lustre file system and for admin

MareNostrum4 (BSC) Supercomputer

https://www.bsc.es/marenostrum/marenostrum

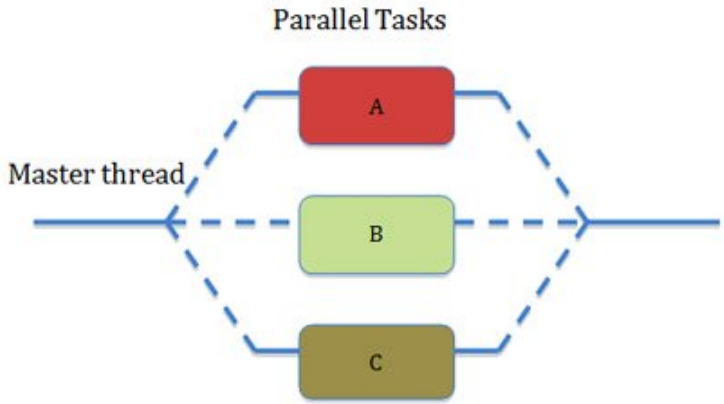
- Most Beautiful Datacenter in the World (2017)
- Rank 30 in the November 2019 TOP 500 list (https://www.top500.org/)
- 48 racks with 3,456 nodes.
  - each node: two Intel Xeon Platinum, each with 24 processors.
- 165,888 processors, 390 TB, 11.15 Petaflops

Main Programming Models

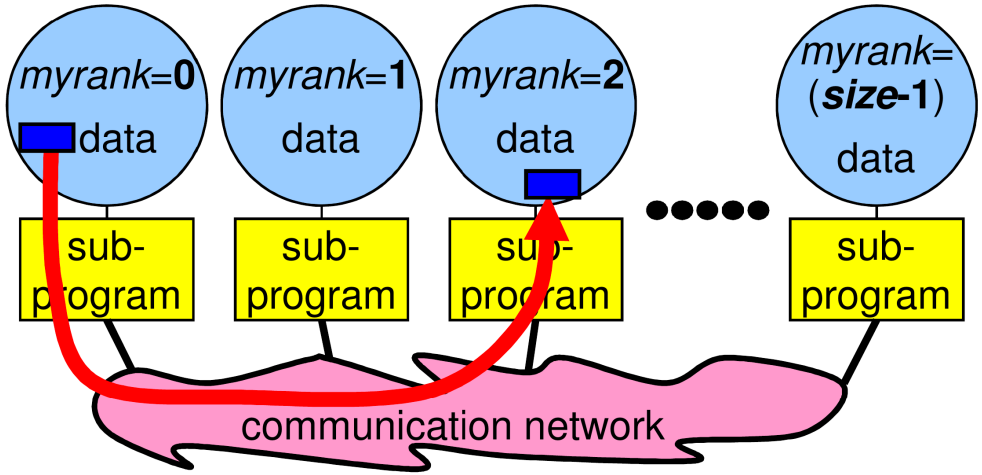
- OpenMP
  - Works on shared memory systems
  - Better performance on SMP systems than MPI
  - Shared memory directives
  - Easier (in principle) to program and debug
- MPI (Message Passing Interface)
  - Works on both shared memory and distributed memory systems
  - Worse performance on SMP systems than OpenMP
  - User specifies:
    - how work & data is distributed
    - how and when communication is done
    - calling MPI library routines
  - More flexible and scalable

Sample code (C)

```c
int main(int argc, char** argv) {
    int partial_Sum, total_Sum;

    #pragma omp parallel private(partial_Sum
        ) shared(total_Sum)
    {
        partial_Sum = 0;
        total_Sum = 0;

        #pragma omp for
        {
            for(int i = 1; i <= 1000; i++){
                partial_Sum += i;
            }
        }
    }
    return 0;
}
```

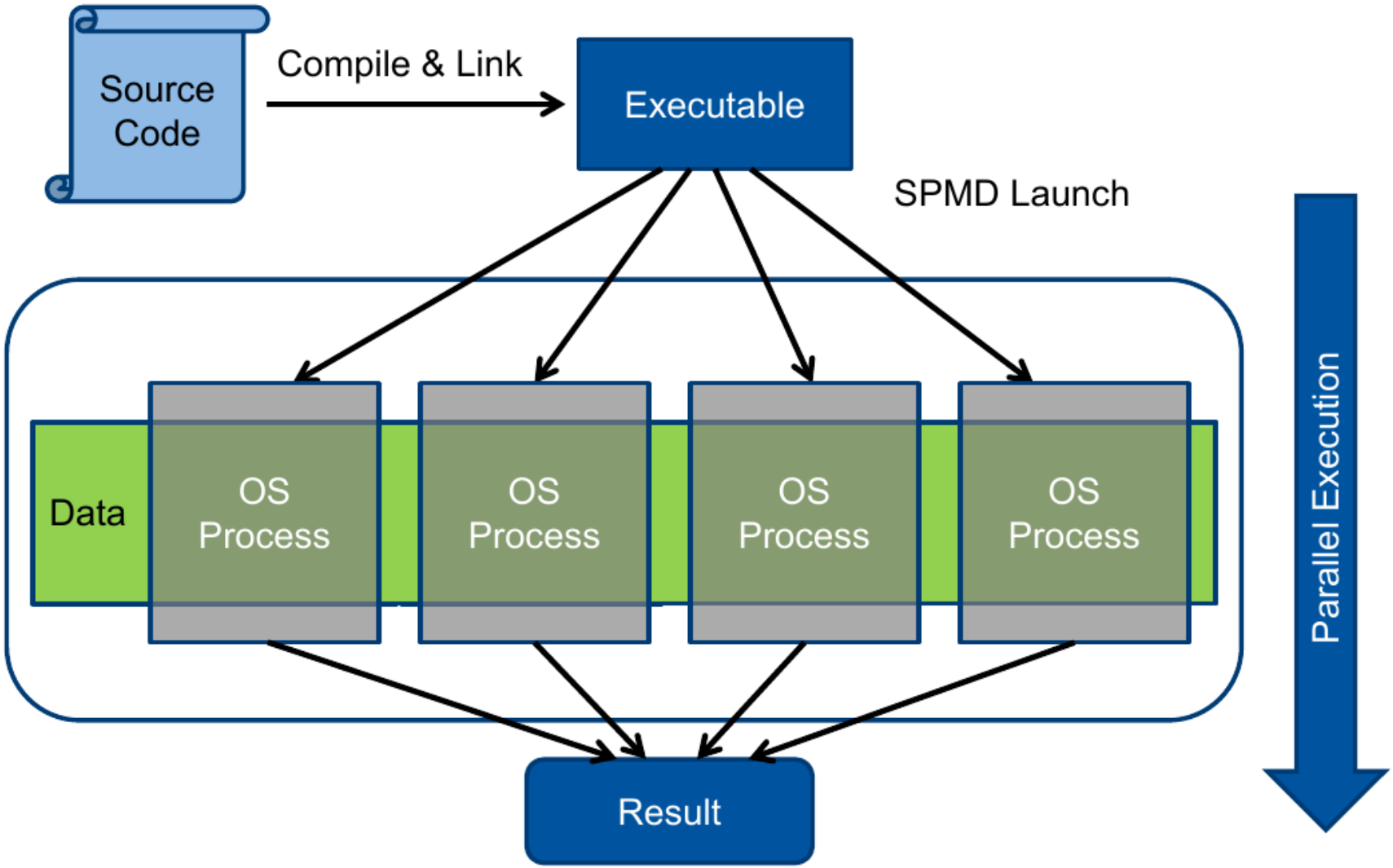


MPI Basics



- Each processor runs a sub-program
  - usually the same on each process
  - written in C/Fortran/etc.
- All parallel distribution (work and data) based on rank
  - rank given by the MPI library
- All communication via MPI library routines (send/receive messages)

Single Program Multiple Data (SPMD) lifecycle



Launch an MPI job (1)

mpirun (Single Program Multiple Data - SPMD)

```
$ mpirun  -tag-output -display-map -np 4 uname -a
 Data for JOB [27996,1] offset 0 Total slots allocated 6

 ========================   JOB MAP   ========================

 Data for node: comer        Num slots: 6        Max slots: 0        Num procs: 4
        Process OMPI jobid: [27996,1] App: 0 Process rank: 0 Bound: UNBOUND
        Process OMPI jobid: [27996,1] App: 0 Process rank: 1 Bound: UNBOUND
        Process OMPI jobid: [27996,1] App: 0 Process rank: 2 Bound: UNBOUND
        Process OMPI jobid: [27996,1] App: 0 Process rank: 3 Bound: UNBOUND

 =============================================================
[1,0]<stdout>:Linux comer 5.3.8-arch1-1 #1 SMP PREEMPT @1572357769 x86_64 GNU/Linux
[1,1]<stdout>:Linux comer 5.3.8-arch1-1 #1 SMP PREEMPT @1572357769 x86_64 GNU/Linux
[1,2]<stdout>:Linux comer 5.3.8-arch1-1 #1 SMP PREEMPT @1572357769 x86_64 GNU/Linux
[1,3]<stdout>:Linux comer 5.3.8-arch1-1 #1 SMP PREEMPT @1572357769 x86_64 GNU/Linux
```

- Common pattern: "Embarrassingly parallel" codes
  - don't need to feel embarrassed, sometimes also called "delightfully parallel" https://en.wikipedia.org/wiki/Embarrassingly_parallel

Launch an MPI job (2)

mpirun (Multiple Program Multiple Data - MPMD)

```
$ mpirun -tag-output -display-map -np 1 hostname : -np 1 date : -np 1 uname : -np 1 pwd
 Data for JOB [33576,1] offset 0 Total slots allocated 6

 ========================   JOB MAP   ========================

 Data for node: comer        Num slots: 6        Max slots: 0        Num procs: 4
        Process OMPI jobid: [33576,1] App: 0 Process rank: 0 Bound: UNBOUND
        Process OMPI jobid: [33576,1] App: 1 Process rank: 1 Bound: UNBOUND
        Process OMPI jobid: [33576,1] App: 2 Process rank: 2 Bound: UNBOUND
        Process OMPI jobid: [33576,1] App: 3 Process rank: 3 Bound: UNBOUND

 =============================================================
[1,0]<stdout>:comer
[1,1]<stdout>:Sat  9 Nov 22:33:09 WET 2019
[1,2]<stdout>:Linux
[1,3]<stdout>:/scratch/Winter_School_2019
```

Emulating MPMD with SPMD (1)

"Greetings" in MPI (Fortran)

```
program greetings
  use mpi

  [...]

  if(rank.eq.0) then
     print*, "greetings from", rank
  else if (rank.eq.1) then
     print*, "hi from", rank
  else
     print*, "bye from", rank
  end if

  call mpi_finalize(err)
end program greetings
```

"Greetings" execution

```
$ mpirun -np 4 greetings
 greetings from           0
 bye from                 3
 bye from                 2
 hi from                  1
```

Emulating MPMD with SPMD (2)

- The previous simple program is as if we had written 3 programs
- Each process executes its own copy of the code
- Synchronization / communication only done if explicitly asked by calling MPI routines
- Note the order of the messages in the "Greetings" output

Process 0

```
program greetings
  [...]
  print*, "greetings␣
      from", rank

end program greetings
```

Process 1

```
program greetings
  [...]
  print*, "hi␣from",
      rank

end program greetings
```

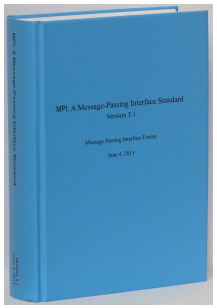
Other processes

```
program greetings
  [...]
  print*, "bye␣from",
      rank

end program greetings
```

Topic

What is MPI?

- MPI (Message-Passing Interface) is a message-passing library interface specification:
  - MPI addresses primarily the message-passing parallel programming model, in which data is moved from the address space of one process to that of another
  - MPI is a specification, not an implementation; there are multiple implementations of MPI (OpenMPI, MPICH, Intel MPI, MVAPICH, etc.)
  - This specification is for a library interface; MPI is not a language, and all MPI operations are expressed as functions, subroutines, or methods.
  - The standard has been defined through an open process by a community of parallel computing vendors, computer scientists, and application developers.



from: "MPI: A Message-Passing Interface Standard, Version 3.1 (2015)", https://www.mpi-forum.org/docs/ (**868 pages!**)

MPI history

- Version 1.0 (1994): FORTRAN 77 and C bindings
- Version 1.1 (1995): Minor corrections and clarifications
- Version 1.2 (1997): Further corrections and clarifications
- Version 2.0 (1997): MPI-2 – Major extensions
  - One-sided communication & Parallel I/O
  - Dynamic process creation
  - Fortran 90 and C++ bindings
  - Language interoperability
- Version 2.1 (2008): Merger of MPI-1 and MPI-2
- Version 2.2 (2009): Minor corrections and clarifications
  - C++ bindings deprecated
- Version 3.0 (2012): Major enhancements
  - Non-blocking collective operations
  - Modern Fortran 2008 bindings
  - C++ deleted from the standard
- Version 3.1 (2015): Corrections and clarifications
  - Portable operation with address variables
  - Non-blocking collective I/O

General Structure of an MPI program

Fortran

```
PROGRAM example
  USE mpi
  INTEGER :: rank, numberOfProcs, ierr
  … some code …
  CALL MPI_Init(ierr)
  … other code …
1 CALL MPI_Comm_size(MPI_COMM_WORLD,&
       numberOfProcs, ierr)
2 CALL MPI_Comm_rank(MPI_COMM_WORLD,&
       rank, ierr)
  … computation  & communication …
  CALL MPI_Finalize(ierr)
  … wrap-up …
END PROGRAM example
```

1. How many processes are there in total? This will set *numberOfProcs* to the number of processes (ranks) in the MPI program
2. Who am I? This will set *rank* to the identity of the calling process within the MPI program (*starting from 0*)

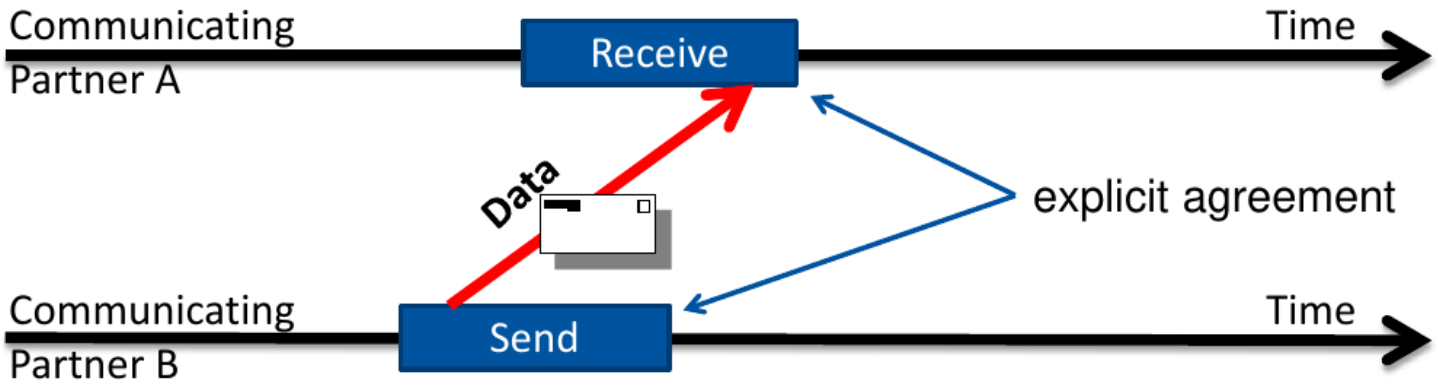
"Greetings" with MPI - Fortran

Our first complete MPI code

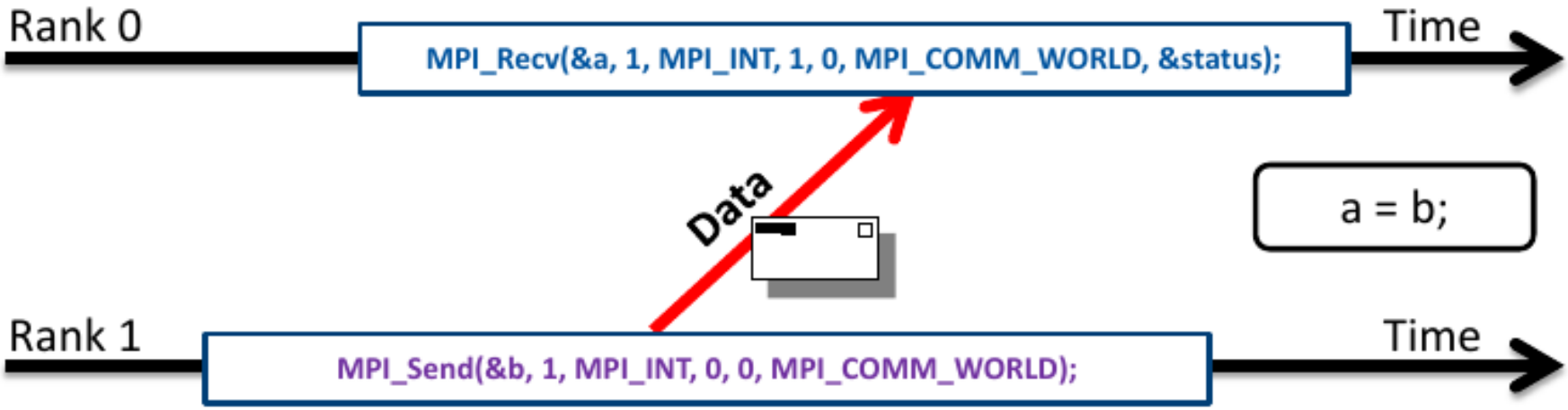
```
program greetings
  use mpi
  implicit none
  integer err,rank,size

  call mpi_init(err)
  call mpi_comm_rank(MPI_COMM_WORLD,rank,err)
  call mpi_comm_size(MPI_COMM_WORLD,size,err)

  if(rank.eq.0) then
     print*, "greetings from", rank
  else if (rank.eq.1) then
     print*, "hi from", rank
  else
     print*, "bye from", rank
  end if

  call mpi_finalize(err)
end program greetings
```

Where are the "messages"?

- The goal of MPI is to be able to communicate between processes that do not share the memory space, which will require:
  - Send and receive routines
  - Addresses of sender and receiver
  - Specification of data to be sent/received



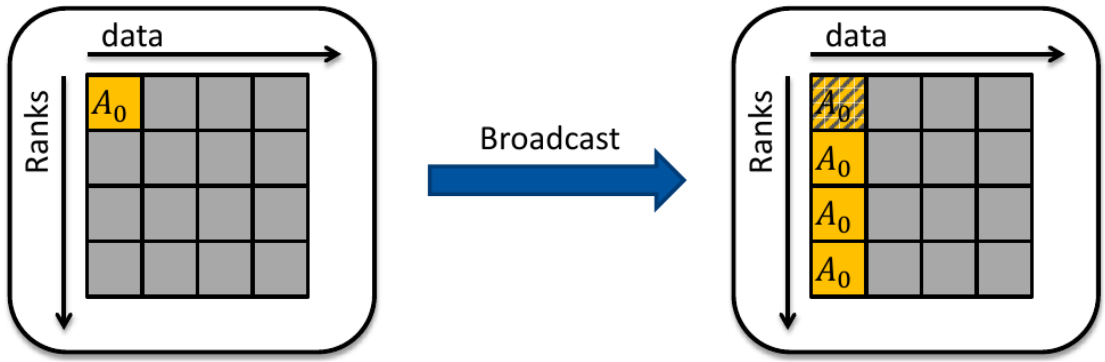
Basic Message passing - Point-to-Point operations




```
if (rank == 0) {
  MPI_Recv(&a, 1, MPI_INT, 1, 0,
           MPI_COMM_WORLD, &status);
}
else if (rank == 1) {
  MPI_Send(&b, 1, MPI_INT, 0, 0,
           MPI_COMM_WORLD);
}
```

Basic Message passing - Collective operations

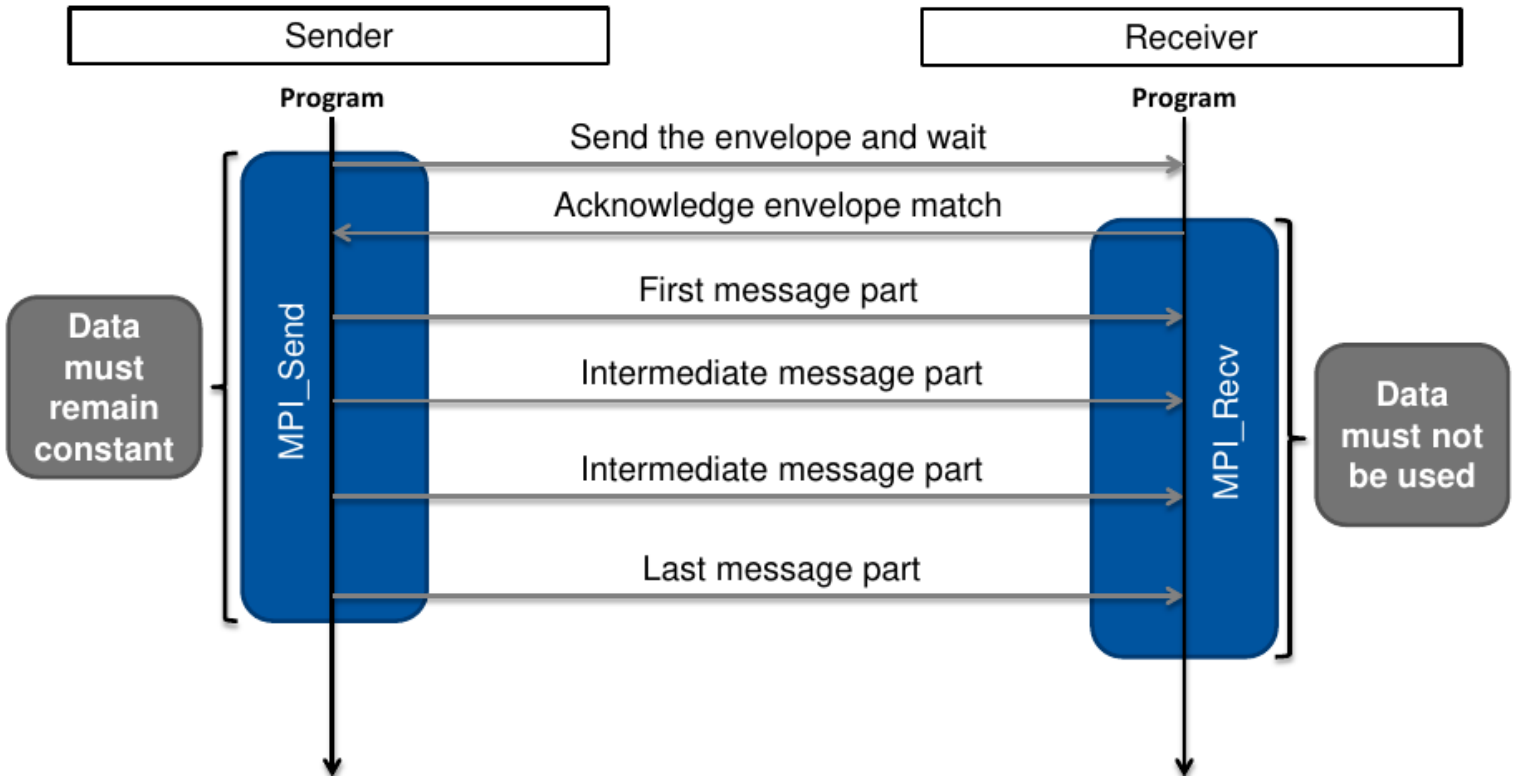
- Collective operations involve all ranks at the same time
- Collectives can be easily implemented with point-to-point primitives.
- . . . but usually collectives offer better performance.
- For example:
  - data replication to all ranks (*MPI_Bcast* - broadcast)



- data scatter: distribute chunks of data to all ranks (*MPI_Scatter*)
- data gather: collect chunks of data from all ranks into one (*MPI_Gather*)

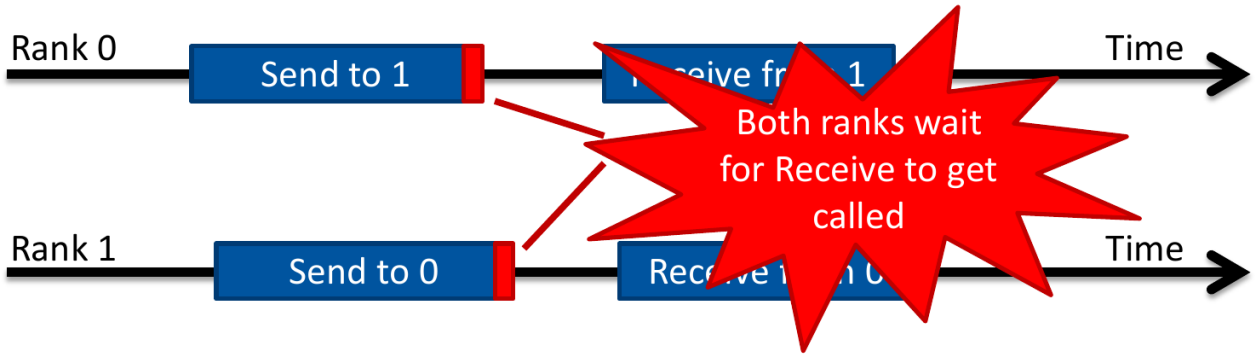
Blocking calls

- MPI operations complete when the message buffer is no longer used by MPI and is free for reuse.
- Blocking MPI calls (e.g. *MPI_Send* and *MPI_Recv*) only return once the operation has completed.



Deadlocks

- Since *MPI_Send* and *MPI_Recv* are blocking, deadlocks can happen.



- *MPI_Sendrecv* will avoid this type of deadlocks: it sends one message and receive one message (in any order) without deadlocking.

```
MPI_Sendrecv (void *senddata, int sendcount, MPI_Datatype sendtype,
              int dest, int sendtag, void *recvdata, int recvcount,
              MPI_Datatype recvtype, int source, int recvtag,
              MPI_Comm comm, MPI_Status *status)
```

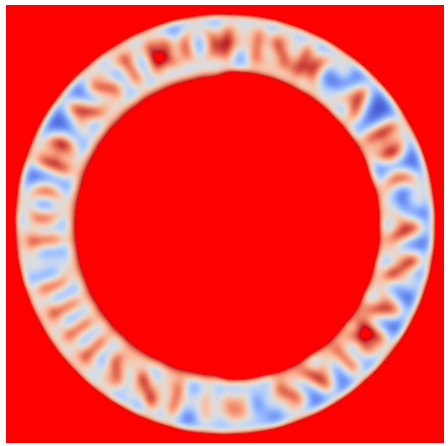
MPI Sample code - 2D Heat equation

- That's all there is to MPI! (well, I'm afraid *not quite!*)
- . . . but useful programs can be written just with this.
- Implementations in *Code-Heat2D* directory
- Serial version at *Code-Heat2D/serial*

```
$ make

$ ./heat iac.dat
Reference value at 5,5: 254.99999950963578
```

heat_0000.png

heat_0004.png



heat_0020.png

2D Heat equation

- Heat (or diffusion) equation is

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u \tag{1}$$

- u(x, y, t): temperature field; α: thermal diffusivity constant.
- The two dimensional Laplacian can be discretized with finite differences as:

$$\nabla^2 u = \frac{u(i-1,j) - 2u(i,j) + u(i+1,j)}{(\Delta x)^2} + \frac{u(i,j-1) - 2u(i,j) + u(i,j+1)}{(\Delta y)^2} \tag{2}$$

- Given an initial condition ($u(t=0) = u0$) we follow the time dependence of the temperature field with the explicit time evolution method:

$$u^{m+1}(i,j) = u^m(i,j) + \Delta t \alpha \nabla^2 u^m(i,j) \tag{3}$$

- *Note*: The algorithm is stable only when

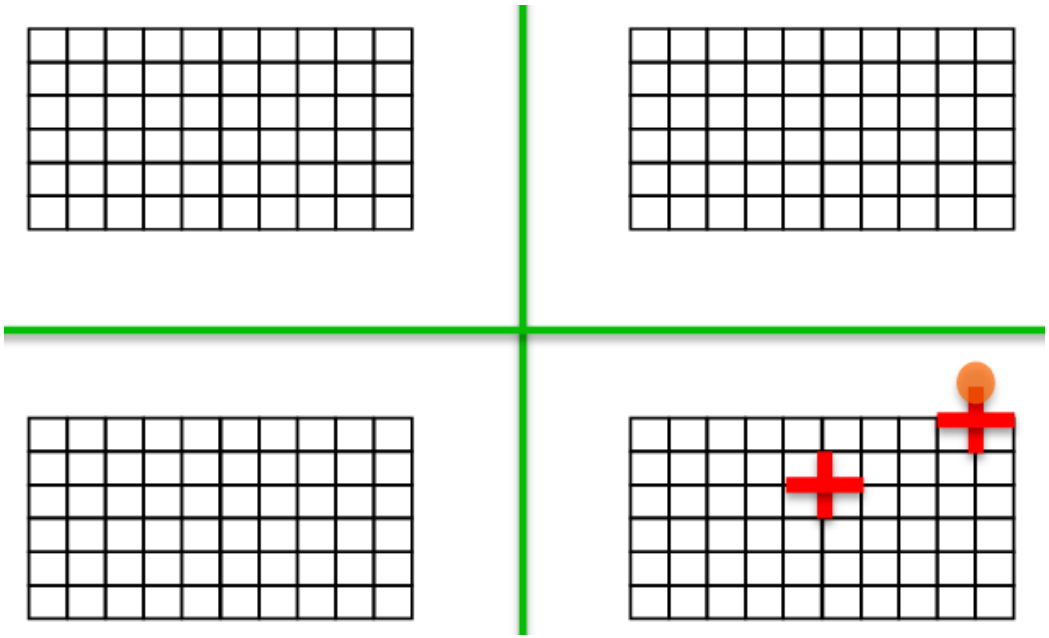
$$\Delta t < \frac{1}{2\alpha} \frac{(\Delta x \Delta y)^2}{(\Delta x)^2 (\Delta y)^2} \tag{4}$$

Heat2D MPI V.1

- Simplest MPI version (*Code-Heat2D/mpi-v1*)
  - Only uses:
    - *MPI_Init*, *MPI_Comm_Size*, *MPI_Comm_Rank*, *MPI_Abort*, *MPI_Finalize*
    - *MPI_Send*, *MPI_Recv*, *MPI_Sendrecv*
  - Hardcoded to 4 processes (2x2)
  - Run with

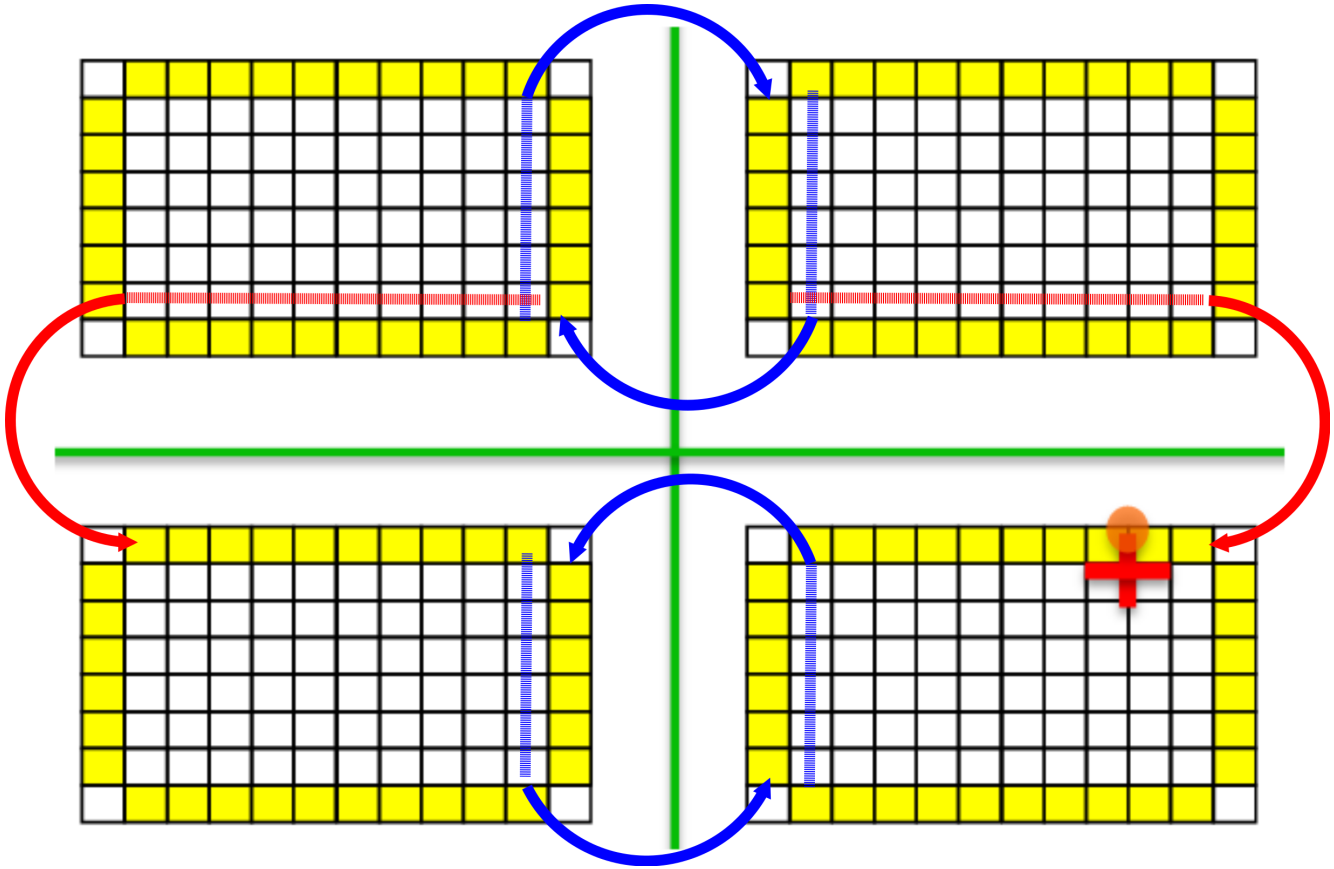
```
$ mpirun -np 4 ./heat_mpi iac.dat
Reference value at 5,5: 254.99999950963578
```

  - *Halo* exchange



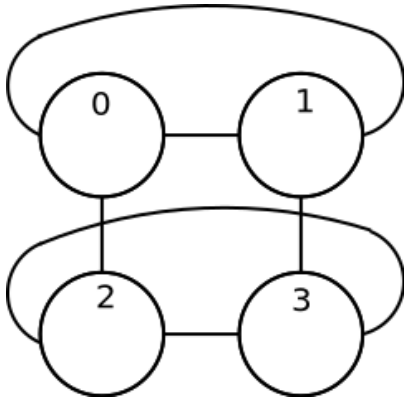
Heat2D MPI V.1 (2)

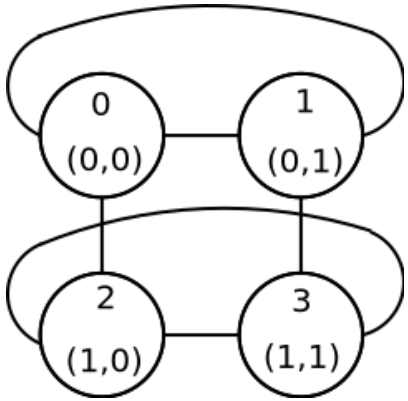
- *Halo* exchange (continued)



Heat2D MPI V.2 - with Cartesian Virtual Topology

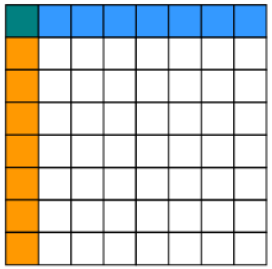
- Finding who my neighbours are and which part of the domain I deal with can get quite messy (specially for 3D). That is why *Heat2D/mpi-v1* was hardcoded to 2x2
- MPI Cartesian Virtual Topologies to the rescue (*Code-Heat2D/mpi-v2*)
  - MPI_Dims_create - Creates a division of processors in a Cartesian grid.
  - MPI_Cart_shift - Returns the shifted source and destination ranks, given a shift direction and amount.
  - MPI_Cart_get - Retrieves Cartesian topology information associated with a communicator.
  - In *mpi-v2* the topology is not hardcoded (for example, run with *mpirun -np 16 ...*)





Heat2D MPI V.3 - User defined datatypes

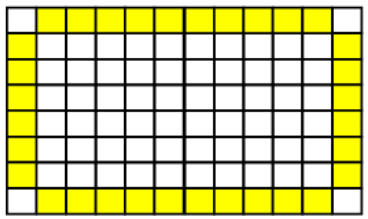
- Many times we want to send non-contiguous data.



  - Memory layout in C

  - Memory layout in Fortran

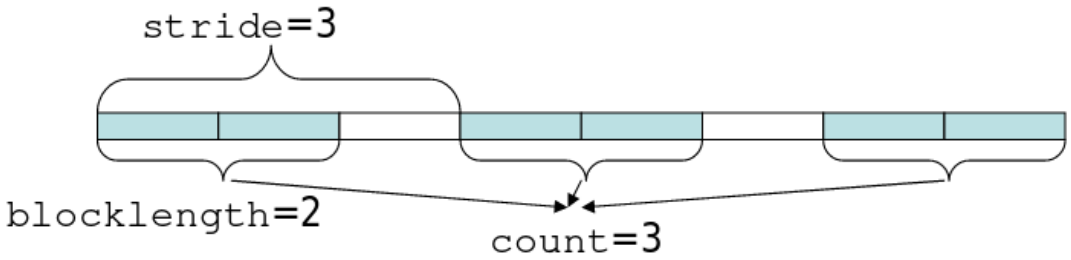
- ... even worse with *halo* data (*real* data to send to master process for saving file is non-contiguous in all languages)



Heat2D MPI V.3 - User defined datatypes (2)

- So MPI offers a few ways to describe non-contiguous data structures. In *Code-Heat2D/mpi-v3* we use:
  - MPI_Type_contiguous - Creates a contiguous datatype.
  - MPI_Type_vector - Creates a vector (strided) datatype.

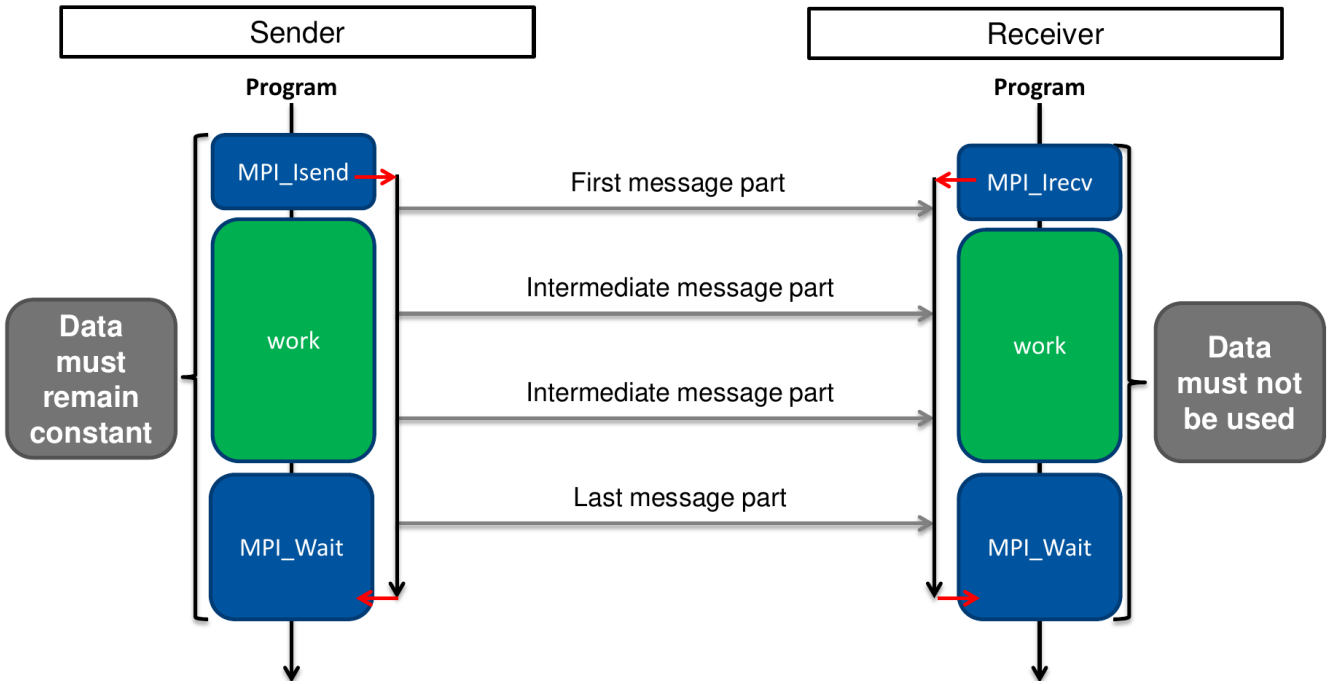
```
int MPI_Type_vector(int count, int blocklength, int stride,
    MPI_Datatype oldtype, MPI_Datatype *newtype)
```



  - MPI_Type_create_subarray - Creates a data type describing an n-dimensional subarray of an n-dimensional array.
- This way I/O of data and halo exchange become much easier to program.
- In Fortran the impact is not so big, because we have nice array syntax, but in C this makes it *a lot* easier.

Heat2D MPI V.4 - Non-blocking calls

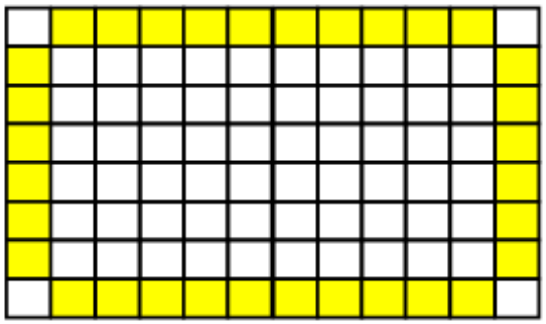
- Non-blocking calls (e.g. *MPI_Isend* and *MPI_Irecv*) return immediately, and communication continues in background.
- We can call *wait* MPI routines to block until they have finished, or *test* routines to only check if they have finished.
- Used to overlay communication and computation, which can greatly increase performance.



Heat2D MPI V.4 - Non-blocking calls (2)

- In *Code-Heat2D/mpi-v4* we use:
  - *MPI_Isend* - Starts a standard-mode, nonblocking send.
  - *MPI_Irecv* - Starts a standard-mode, nonblocking receive.
  - *MPI_Waitall* - Waits for all given communications to complete.
- Now, in the main loop of the code (*main.F90*), we can start evolving the interior cells without having to wait for the communication to finish.

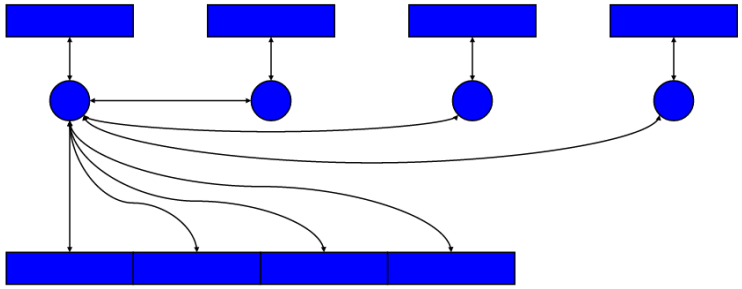
```
do iter = iter0, iter0 + nsteps
   call exchange_init(previous, parallelization)
   call evolve_interior(current, previous, a, dt)
   call exchange_finalize(parallelization)
   call evolve_edges(current,previous, a, dt)
```



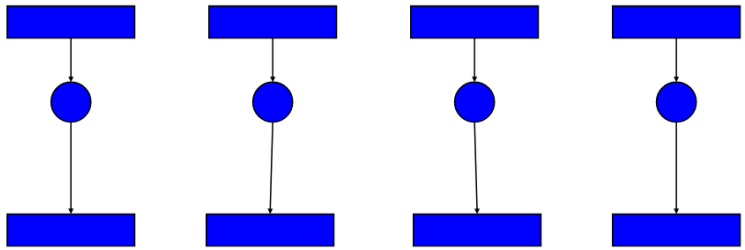
I/O in MPI programs

Several ways of dealing with I/O in MPI programs:

- non-parallel I/O
  - performance worse than sequential
  - all versions of *Code-Heat2D/mpi-v[1-4]* use this

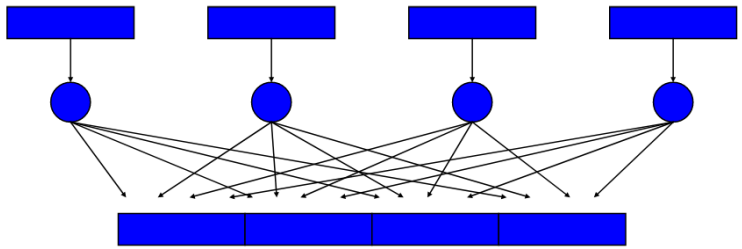


- independent parallel I/O
  - much better performance, parallelism!
  - . . . but lots of small files to manage



I/O in MPI programs (2)

- cooperative parallel I/O
  - very good performance (if using a Parallel File System, such as *Lustre*)
  - output as one file



- You can use MPI I/O routines to deal with parallel I/O directly:
  - you will have more control, but at the expense of being more difficult
  - and ending with a *home-made* file format.
- Better option to use HDF5+MPI!
  - so let's learn about HDF5 first.

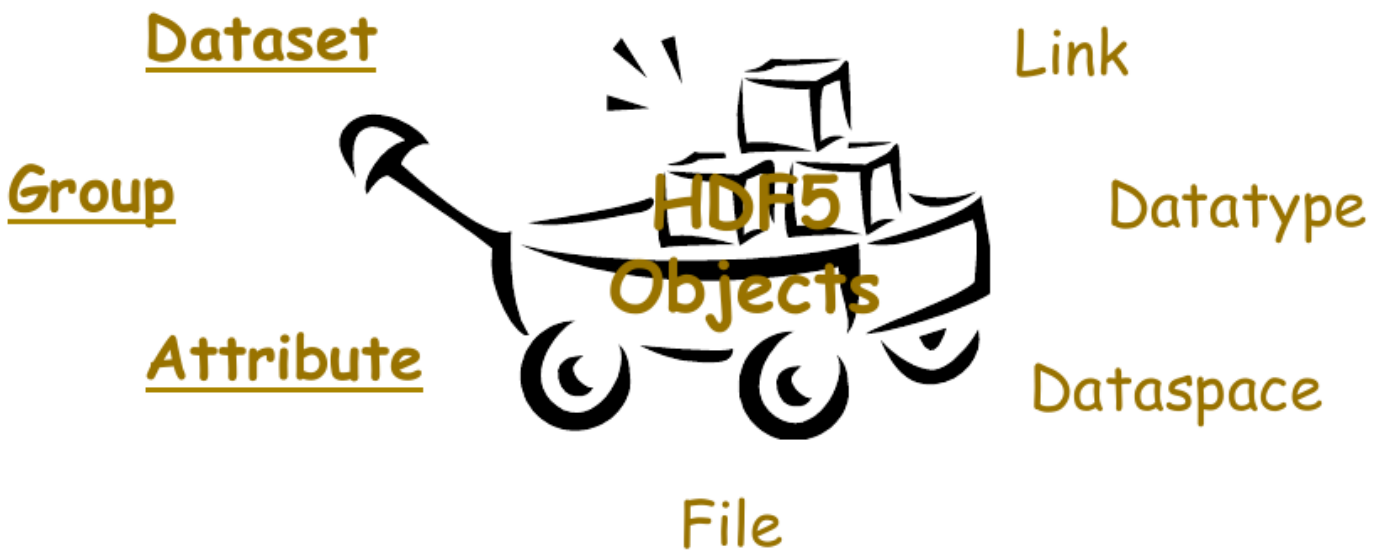
Topic

Basics of HDF5

- HDF5 is a high-performance technology suite that consists of:
  - an abstract data model
  - a library
  - file format (bit-level organization of a file. Users do not need to know the details of this: https://support.hdfgroup.org/HDF5/doc/H5.format.html)
- ... for storing and managing extremely large and/or complex data collections.
- Advantages of HDF5:
  - versatile data model (complex and heterogeneous data and metadata with unlimited variety of datatypes)
  - portable and extensible with no size limits
  - self-describing
  - robust software ecosystem of open source tools and applications to manage, manipulate, view and analyze data
  - architecture independent that runs with a wide range of programming languages
  - advanced performance features (compression, encryption, ...)

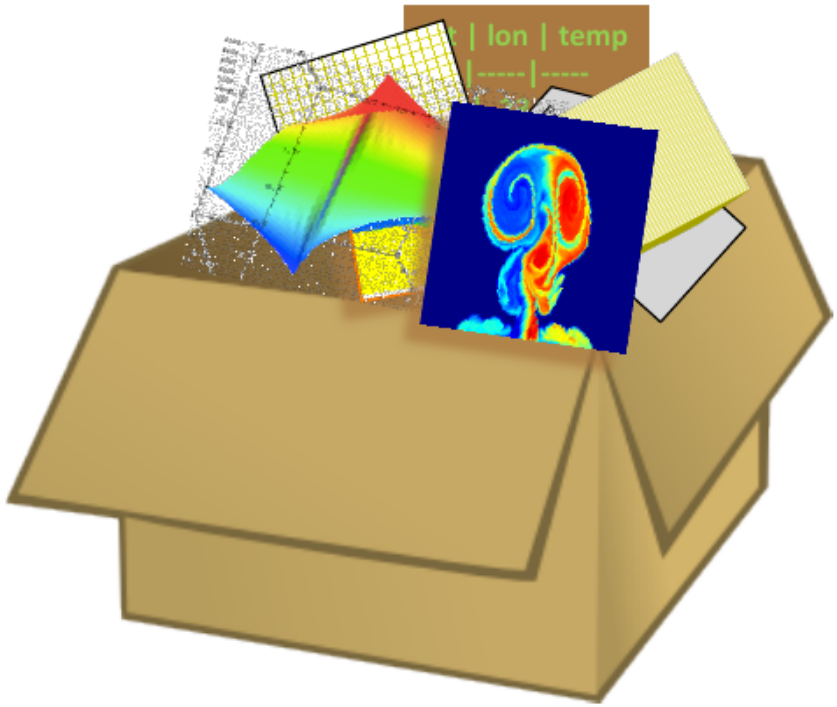
HDF5 data model

- The data model describes the "building blocks" for data organization and specification:
  - File
  - Dataset
    - Datatype
    - Dataspace
  - Attribute
  - Group and link



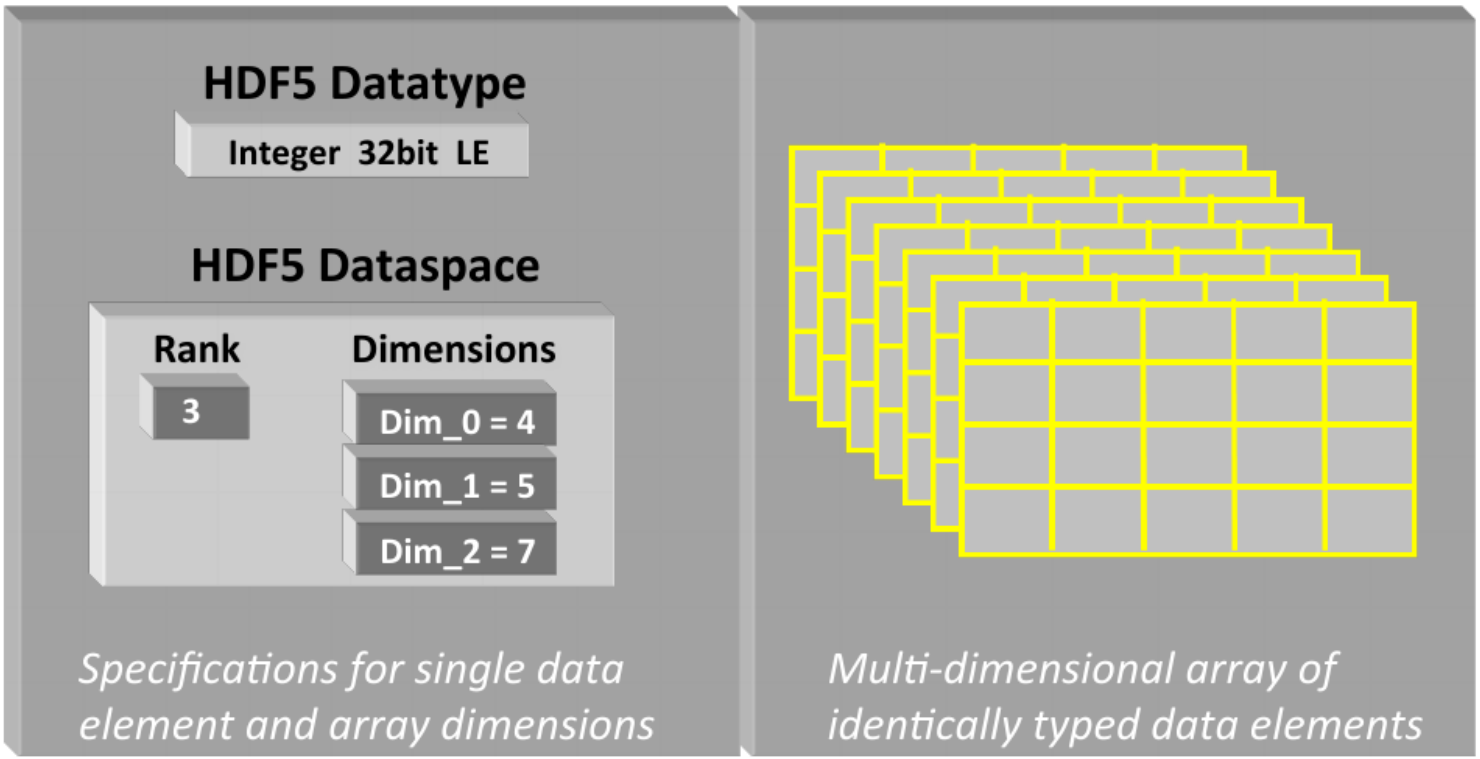
HDF5 data model - File

- An HDF5 file is a container that holds data objects



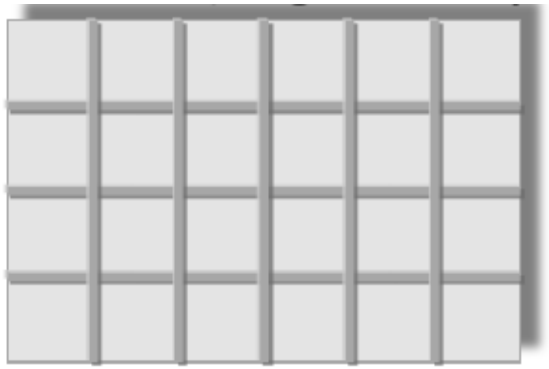
HDF5 data model - Dataset

- HDF5 datasets organize and contain "raw data values".
  - HDF5 datatypes describe individual data elements
  - HDF5 dataspaces describe the logical layout of the data elements. (*NULL*, scalar or simple array (*most common*).



HDF5 data model - Dataspace

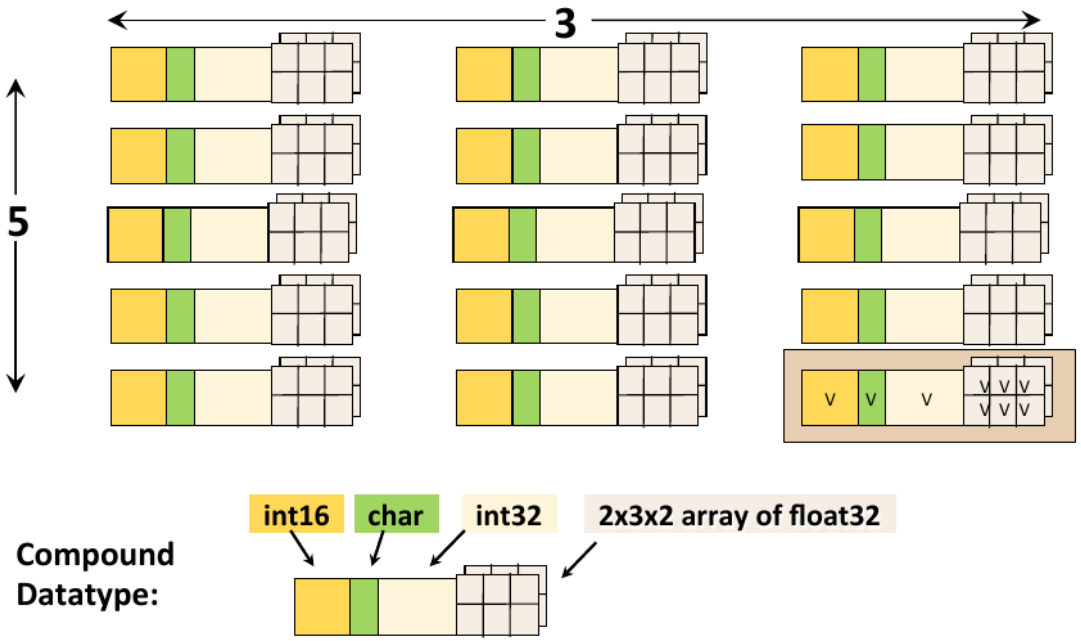
- A Dataspace can have two different roles:
  - spatial information about dataset stored in a file



  - partial I/O: application's data buffer and data elements participating in I/O (used for serial and parallel HDF5)

HDF5 data model - Datatype

- It describes individual elements in a dataset. Wide range supported:
  - Integer
  - Float
  - Compound
  - User-defined
  - ...

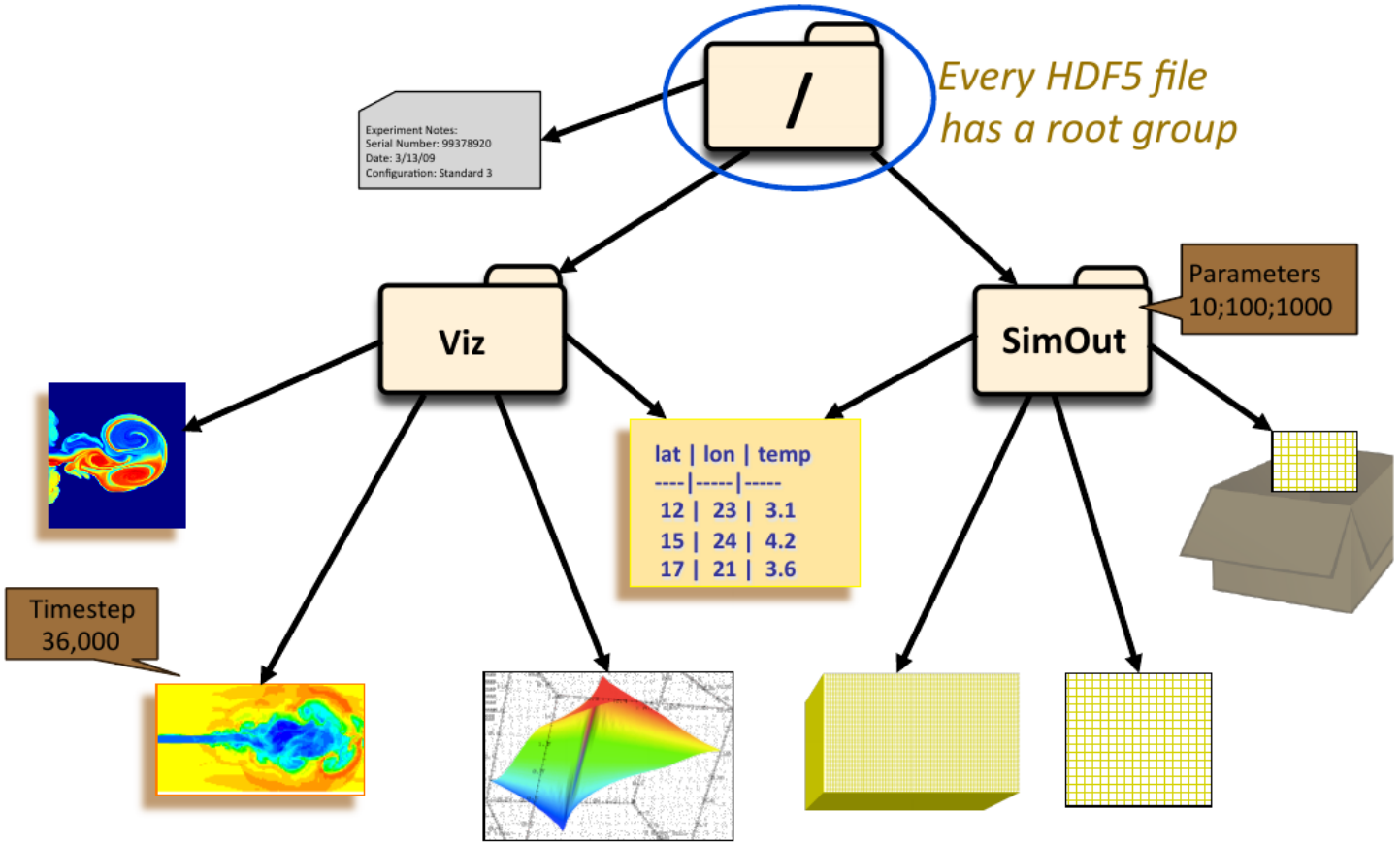


HDF5 data model - Attribute

- Associated with an HDF5 object
- Value is described by a datatype and dataspace (analogous to a dataset)
- Have a *name* and a *value*
- Usually small and typically contain user metadata

HDF5 data model - Groups and links

- HDF5 groups and links organize data objects.



Sample HDF5 files

Mancha3D

- *HDF5-samples/orszag_0051.h5*

```
+ Group: "/"
  + Attr: cellsize
  + Attr: counter
  + [...]
  + Dataset: bx
     + Dataype: H5T_IEEE_F64LE
     + Dataspace: ( 128, 128, 128 )
  + Dataset: by
     + Dataype: H5T_IEEE_F64LE
     + Dataspace: ( 128, 128, 128 )
  + [...]
```

PORTA

- *HDF5-samples/twolevel.h5*

```
+ Group: "/"
  + Attr: Azimuth_nodes
  + Attr: Creation_date
  + [...]
  + Attr: Z_Axis
  + Group: "Module"
    + Attr: Atom_mass
    + Attr: A_UL
    + [...]
    + Dataset: g_data
      + Datatype: H5T_COMPOUND
        + H5T_IEEE_F64LE "eps";
        + H5T_IEEE_F64LE "temp";
        + H5T_ARRAY { [3] H5T_IEEE_F64LE } "B";
        + [...]
      + Dataspace: (100, 3, 3)
    + Dataset: temp_bc
      + Datatype: H5T_IEEE_F64LE
      + Dataspace: ( 3, 3 )
```

HDF5 command-line tools

- A number of useful comand-line tools distributed and installed with HDF5 to examine or manage HDF5 files, convert between formats, etc.
- Documentation via command line (e.g. *h5dump –help*) or at: https://portal.hdfgroup.org/display/HDF5/HDF5+Command-line+Tools
- Most useful tools:
  - h5cc - h5pcc, h5fc - h5pfc, h5c++: Helper scripts for compiling an HDF5 application (*will see this one later*)
  - h5ls: Lists selected information about file objects in the specified format
  - h5dump: Enables the user to examine the contents of an HDF5 file and dump those contents to a file (either as ASCII text or as binary data)
  - h5diff: Compares two HDF5 files and reports the differences
  - h5repack: Copies an HDF5 file to a new file with or without compression/chunking
- Importing data, creating groups, etc. can also be done via command-line tools, but in the end more flexible and probably easier using, e.g. Python.

HDF5 Tools - *h5ls*

```
[HDF5-samples]$ h5ls -r twolevel.h5
/                        Group
/Module                  Group
/Module/g_data           Dataset {100, 3, 3}
/Module/temp_bc          Dataset {3, 3}

[HDF5-samples]$ h5ls -d twolevel.h5/Module/temp_bc
temp_bc                  Dataset {3, 3}
    Data:
        (0,0) 6000, 6000, 6000, 6000, 6450, 5550, 6000, 5550, 6450
```

HDF5 Tools - *h5dump*

```
[HDF5-samples]$ h5dump -H -d Module/g_data twolevel.h5
HDF5 "twolevel.h5" {
DATASET "Module/g_data" {
   DATATYPE  H5T_COMPOUND {
      H5T_IEEE_F64LE "eps";
      H5T_IEEE_F64LE "temp";
      [...]
   }
   DATASPACE  SIMPLE { ( 100, 3, 3 ) / ( 100, 3, 3 ) }
}
```

```
[HDF5-samples]$ h5dump -d Module/g_data -S "4,1,1" -c "2,1,1" -k "2,1,1" twolevel.h5
[...]
      DATA {
      (0,0,0): {
            0.0001,
            6000,
            1e+15,
            [...]
      (0,0,1): {
            0.0001,
```

HDF5 Tools - *h5diff*

```
[HDF5-samples]$ h5diff iac.h5 heat_0000.h5

[HDF5-samples]$ h5diff heat_0000.h5 heat_0001.h5
dataset: </T> and </T>
181667 differences found

[HDF5-samples]$ h5diff -p 1e1 heat_0000.h5 heat_0001.h5
dataset: </T> and </T>
2754 differences found

[HDF5-samples]$ h5diff -v -p 1e1 heat_0000.h5 heat_0001.h5
file1     file2
----------------------------------------
    x       x    /
    x       x    /T

dataset: </T> and </T>
size:           [460x460]           [460x460]
position        T                   T                   difference      relative
--------------------------------------------------------------------------------
[ 9 206 ]          2                   47.8046875      45.8046875      22.90234375
[ 9 207 ]          2                   44.80371094     42.80371094     21.40185547
[ 9 208 ]          1                   42.79882812     41.79882812     41.79882812
[...]
```

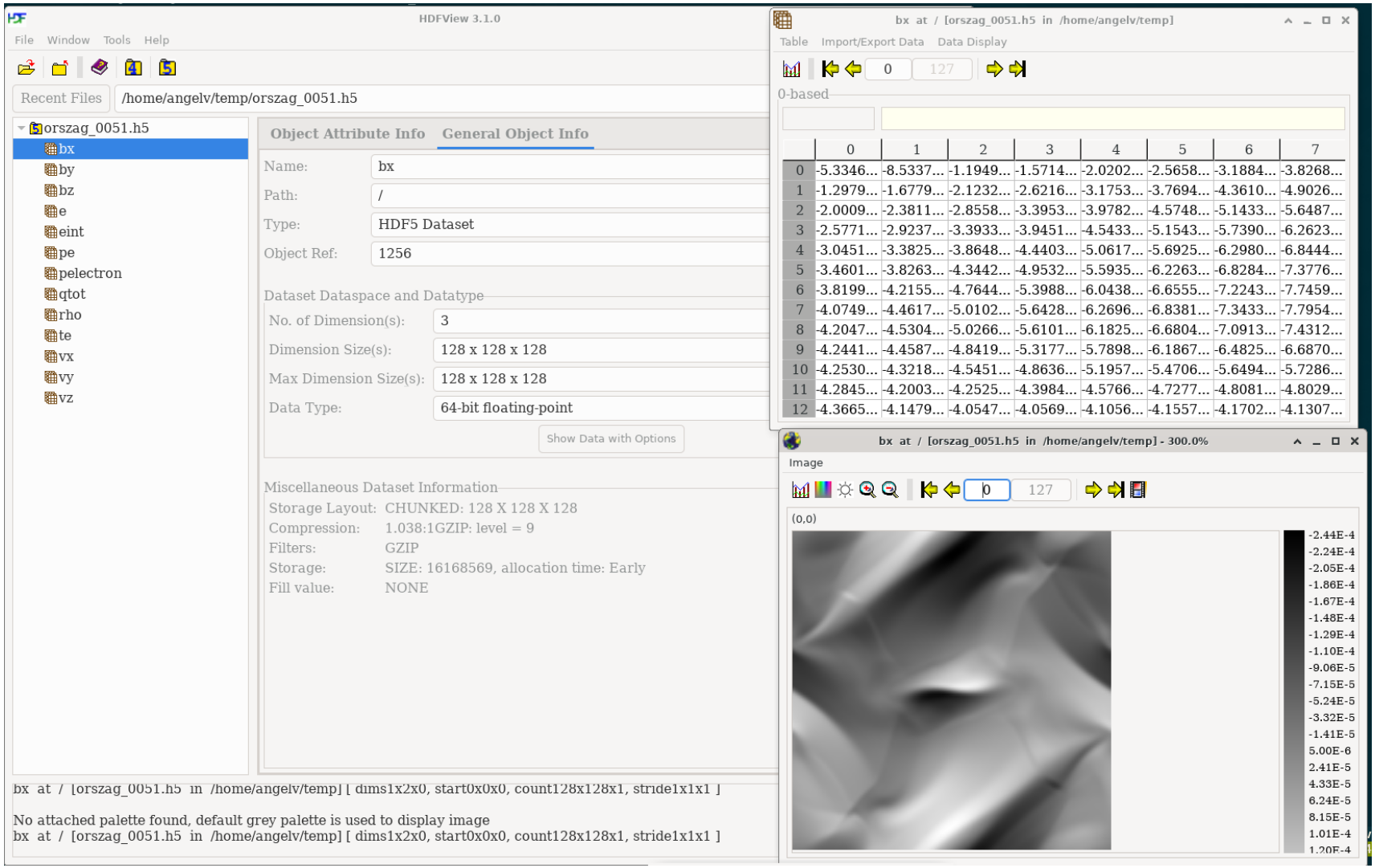
HDF5 Tools - *h5repack*

```
[HDF5-samples]$ h5repack -f GZIP=8 twolevel.h5 twolevel_gz8.h5

[HDF5-samples]$ h5ls -r twolevel_gz8.h5
/                        Group
/Module                  Group
/Module/g_data           Dataset {100, 3, 3}
/Module/temp_bc          Dataset {3, 3}

[HDF5-samples]$ ls -ltrh twolevel*
-rw-r--r-- 1 angelv angelv 307K Jun 24 09:27 twolevel.h5
-rw-r--r-- 1 angelv angelv  17K Nov 15 21:44 twolevel_gz8.h5

[HDF5-samples]$ h5stat twolevel.h5 | grep -A 10 -i filters
Dataset filters information:
        Number of datasets with:
                NO filter: 2
                GZIP filter: 0
                SHUFFLE filter: 0

[HDF5-samples]$ h5stat twolevel_gz8.h5 | grep -A 10 -i filters
Dataset filters information:
        Number of datasets with:
                NO filter: 0
                GZIP filter: 2
                SHUFFLE filter: 0
```

HDFView

- Download at: https://www.hdfgroup.org/downloads/hdfview/
  - *but probably already installable via your package manager*
- Visual tool for browsing and editing HDF4 and HDF5 files. You can:
  - view a file hierarchy in a tree structure
  - create new files, add or delete groups and datasets
  - view and modify the content of a dataset
  - add, delete and modify attributes

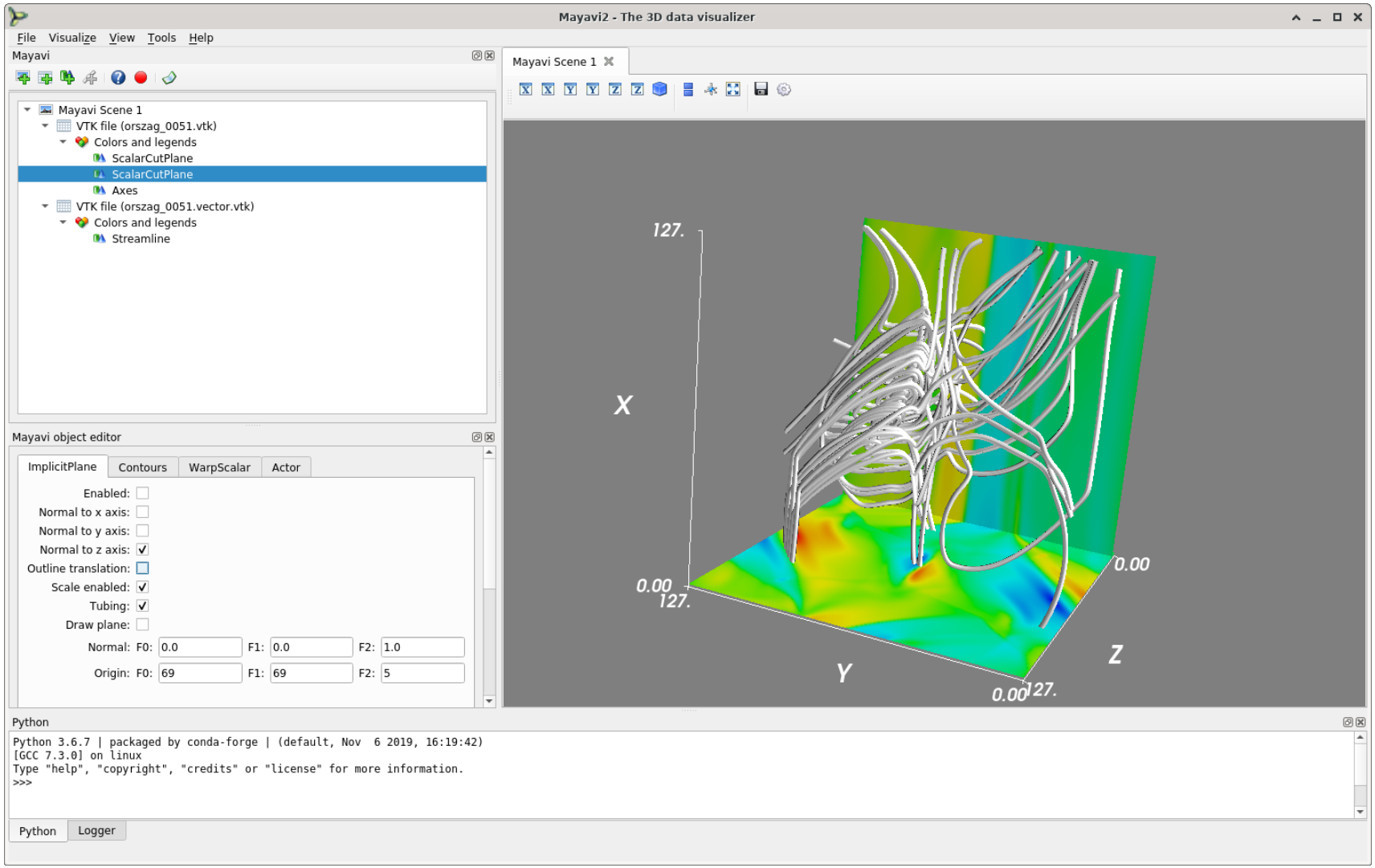
HDFView *sample*



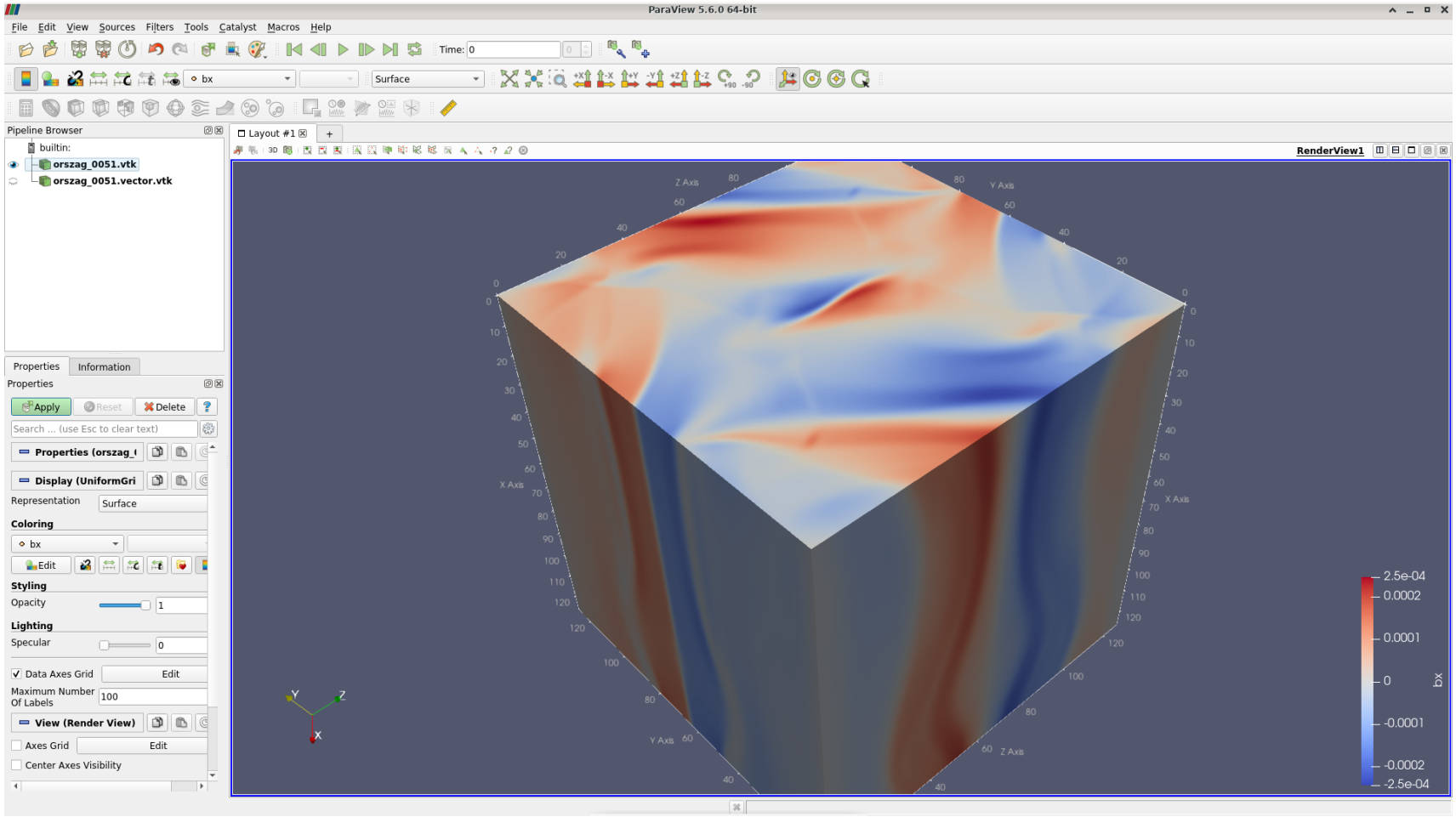
Other visualization/analysis tools

- Many tools can read HDF5 files. For a more detailed list, see https://portal.hdfgroup.org/display/support/Software+Using+HDF5
  - VisIt https://wci.llnl.gov/simulation/computer-codes/visit/
  - ParaView https://www.paraview.org/
  - Tecplot https://www.tecplot.com/
  - GNU Octave https://www.gnu.org/software/octave/
  - IDL https://www.harrisgeospatial.com/Software-Technology/IDL
  - MATLAB https://uk.mathworks.com/
- HDF5 is a generic container format, so form some of these tools writing a Plugin might be required, or describing the "heavy data" with, for example XDMF (http://www.xdmf.org/index.php/XDMF_Model_and_Format).
- Many specific formats (e.g. Chombo, netCDF-4, CGNS) actually use HDF5 file format, but following specific conventions (samples in *HDF5-samples*).
- And for other tools it might be possible to convert HDF5 to the required format.
  - Mayavi https://docs.enthought.com/mayavi/mayavi/ (previous conversion to VTK with, for example, https://github.com/NanoComp/h5utils)

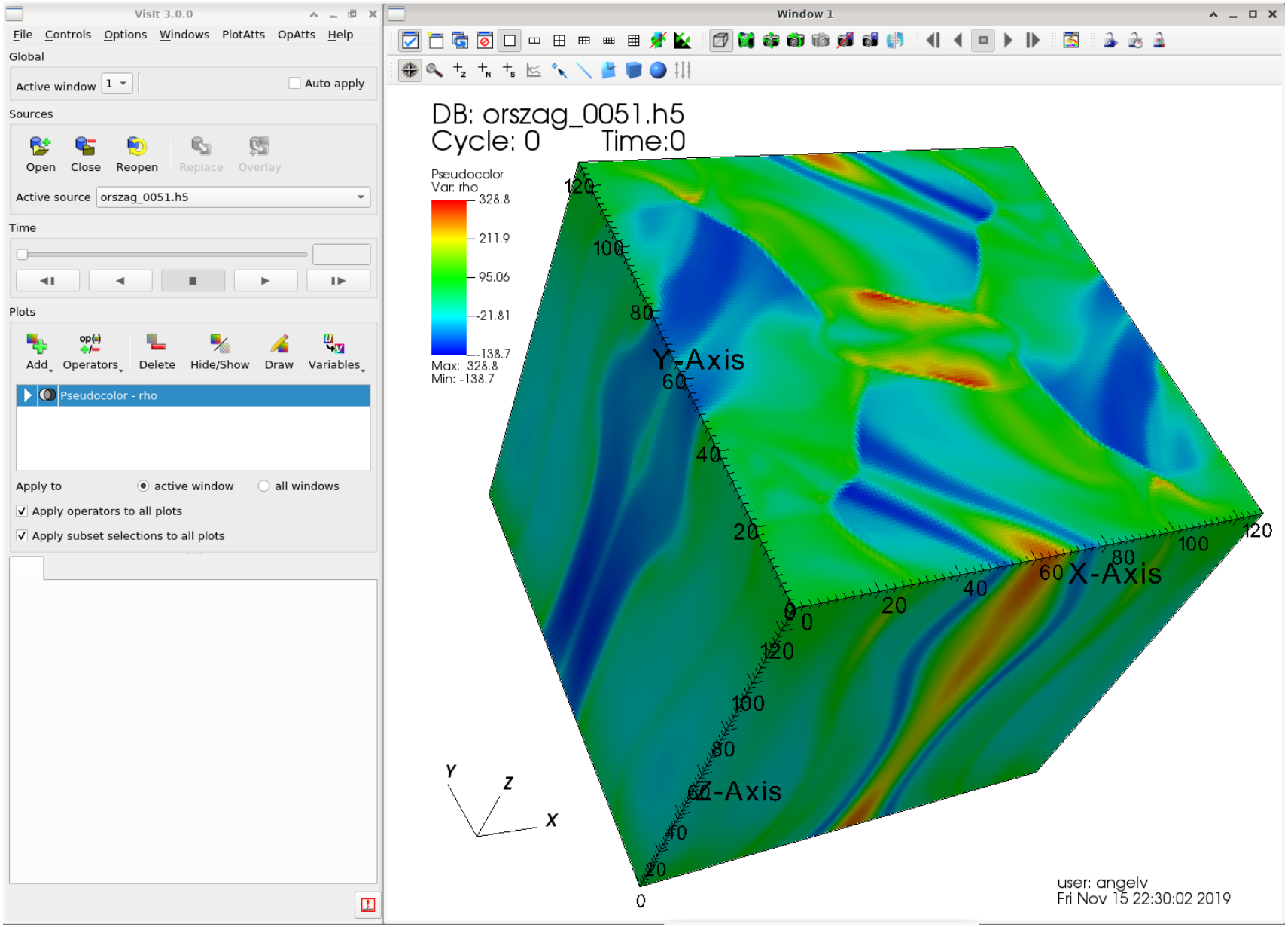
Visualization tools - Mayavi



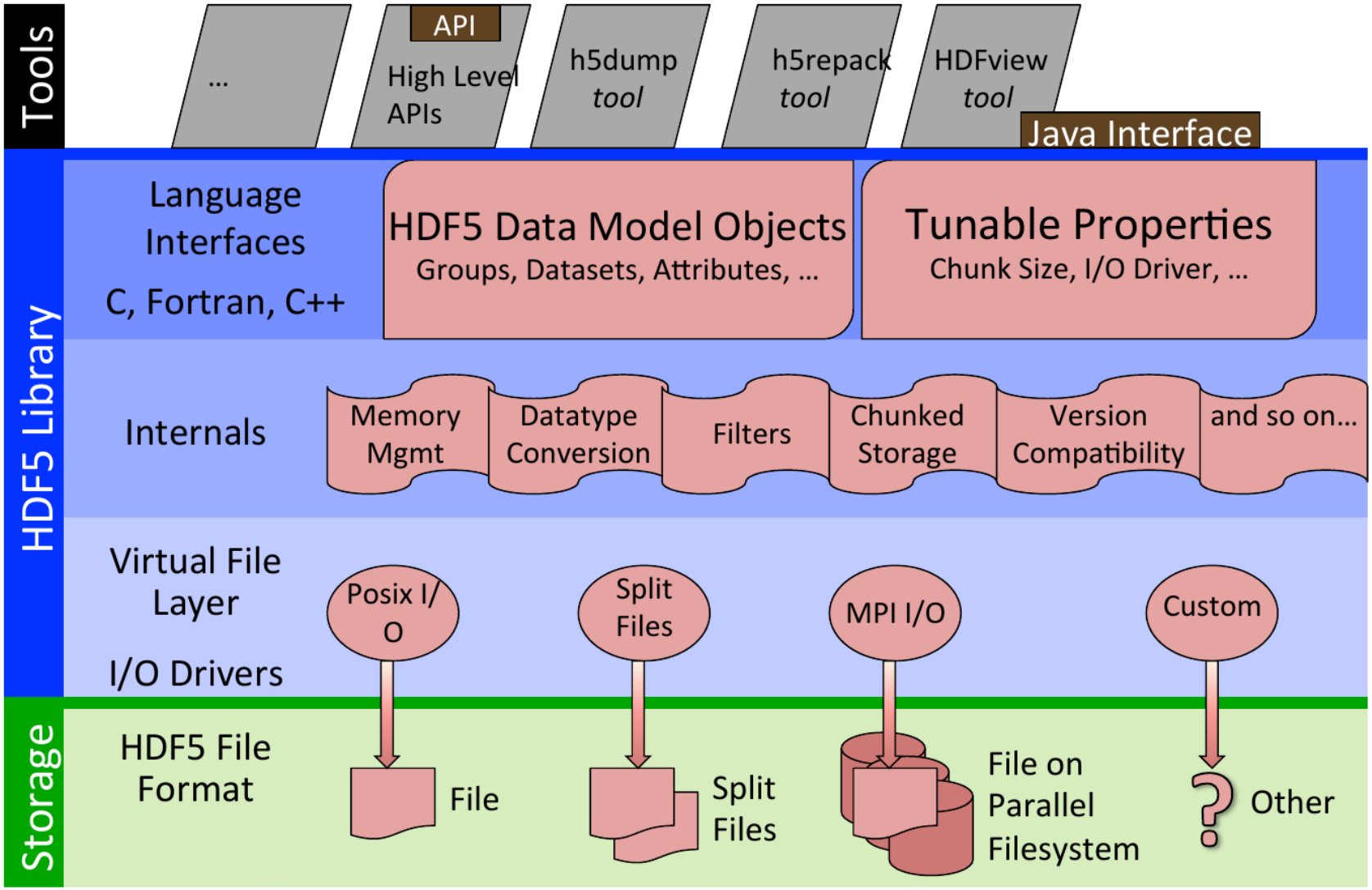
Visualization tools - ParaView



Visualization tools - VisIt



Application Programming Interface (API)



HDF5 Programming model

- Object is opened or created
  - Properties (creation, access) are *optionally* defined
- Object is accessed, possibly many times
- Object is closed
- An order is imposed on operations by argument dependencies
  - For example: the call to open a dataset requires a file handle, so the file must be opened first!
- Library has C, Fortran90, Java and C++ bindings.
  - More than 300+ functions, but only a few needed in most cases.
  - High-level libraries (e.g. *Lite*) available, which simplifies code a lot. Details at: https://tinyurl.com/v2u9e4s
  - C routines begin with prefix corresponding to the type of object they act on:
    - H5D_... : Dataset interface (e.g. *H5Dread*)
    - H5F_... : File interface (e.g. *H5Fopen*)
    - H5S_... : dataSpace interface (e.g. *H5Sclose*)

Sample code - create a dataset (C)

```c
#include "hdf5.h"

int main() {
    hid_t       file_id, dataset_id, dataspace_id;
    hsize_t     dims[2];
    herr_t      status;
    int         i, j, dset_data[4][6];

    for (i = 0; i < 4; i++)
        for (j = 0; j < 6; j++)
            dset_data[i][j] = i * 6 + j + 1;

    file_id = H5Fcreate("dset.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

    dims[0] = 4; dims[1] = 6;
    dataspace_id = H5Screate_simple(2, dims, NULL);

    dataset_id = H5Dcreate2(file_id, "/dset", H5T_STD_I32BE, dataspace_id,
                            H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

    status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
                      dset_data);

    status = H5Dclose(dataset_id);
    status = H5Sclose(dataspace_id);
    status = H5Fclose(file_id);
}
```

Sample code - create a dataset using High Level Library (C)

```c
#include "hdf5_hl.h"

int main() {
    hid_t       file_id, dataset_id, dataspace_id;
    hsize_t     dims[2] = {4,6};
    herr_t      status;
    int         i, j, dset_data[4][6];

    for (i = 0; i < 4; i++)
        for (j = 0; j < 6; j++)
            dset_data[i][j] = i * 6 + j + 1;

    file_id = H5Fcreate("dset_hl.h5", H5F_ACC_TRUNC, H5P_DEFAULT,
        H5P_DEFAULT);

    status = H5LTmake_dataset(file_id,"/dset",2,dims,H5T_STD_I32LE,
        dset_data);

    status = H5Fclose(file_id);
}
```

APIs (even higher level) in many languages

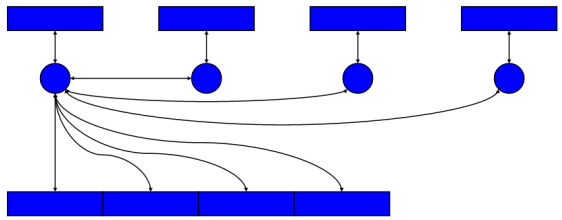
- Let's write Python sample code to:
  - read a simple HDF5 file with only the root group,
  - create another file with the same attributes,
  - for each dataset create another one with "_2" appended to the name, with the squared values of the original dataset.
  - at the same time, save the mean value of each dataset in a new attribute "mean_value".

Sample code: Python + h5py (https://www.h5py.org/)

```
import h5py
import numpy as np

def test(h5in,h5out):
    with h5py.File(h5in, 'r') as fin:
        with h5py.File(h5out,'w') as fout:
            for key in fin.attrs.keys():
                fout.attrs[key] = fin.attrs[key]
            for key in fin.keys():
                dat = fin[key][:]
                fout[key] = dat
                fout[key].attrs["mean_value"] = dat.mean()
                dat **= 2
                fout.create_dataset(key+"_2",data=dat)
                fout[key+"_2"].attrs["mean_value"] = dat.mean()
```

Heat2D MPI+HDF5 - V.5

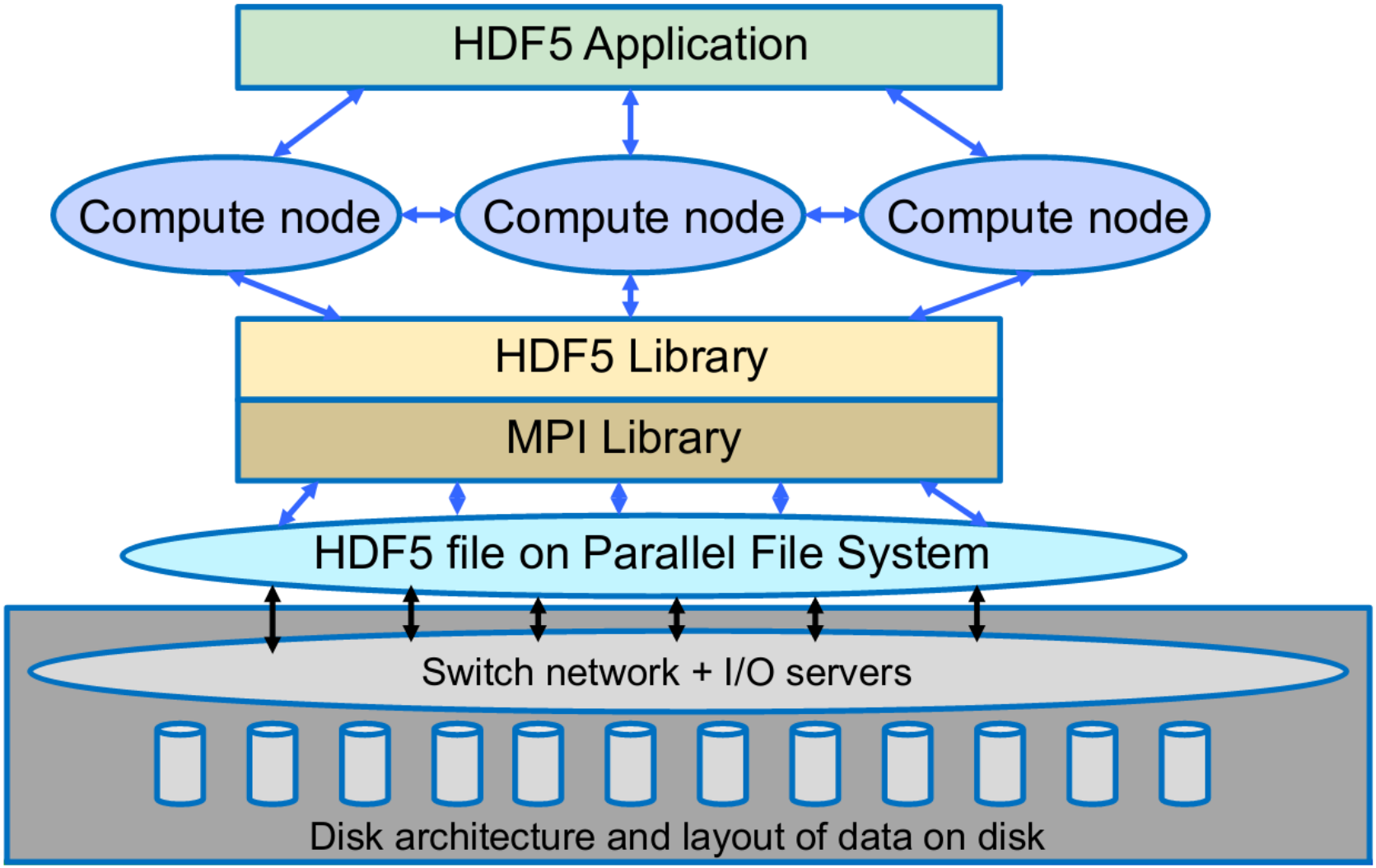
- In *Code-Heat2D/mpi-v5*
  - Python script to convert *iac.dat* to *iac.h5* (*dattoh5.py*)
  - need to compile it with MPI+PHDF5 libraries: use *h5pfc* wrapper, already modified in Makefile. (You can see the actual commands invoked by calling *h5pfc -show* and *mpif90 -show*)
  - we modify the Heat2D code to write output files in HDF5, but with non-parallel I/O.
  - this follows the pattern used in all versions of *Code-Heat2D/mpi-v[1-4]*
  - we use the following routines for writing the file (see *io.F90* file), and similar ones for reading an HDF5 file:
    - *h5open_f* - Opens HDF5 Fortran interface
    - *h5fcreate_f* - Creates an HDF5 file
    - *h5ltset_attribute_int_t* - Creates and writes an attribute.
    - *h5ltmake_dataset_f* - Creates and writes a dataset.
    - *h5fclose_f* - Closes an HDF5 file
    - *h5close_f* - Closes HDF5 Fortran interface



Parallel HDF5

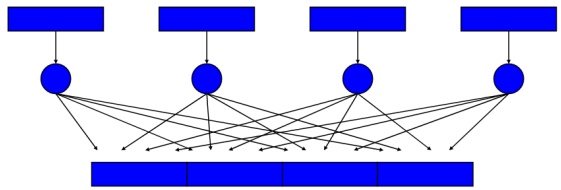
- Requirements
  - PHDF5 should allow multiple processes to do I/O to an HDF5 file at the same time (single file image for all processes).
  - Should use a standard parallel I/O interface.
  - Portable to different platforms
  - Support MPI programming
  - PHDF5 files compatible with serial HDF5 files (shareable between different serial or parallel platforms)
  - MPI with MPI-IO (MPICH, OpenMPI, etc.)
  - Parallel file system (IBM GPFS, Lustre, PVFS)

PHDF5 implementation layers



Heat2D MPI+PHDF5 - V.6

- In *Code-Heat2D/mpi-v6*
  - we modify the Heat2D code to write output files in HDF5, but with parallel I/O (PHDF5).
  - when using a Parallel File System the performance will be much better
  - much easier and portable than using direct MPI I/O calls
  - we use the following additional routines for writing the file (see *io.F90* file), and similar ones for reading an HDF5 file:
    - Routines to create and set properties for collective access to file and datasets: *h5pcreate_f*, *h5pset_fapl_mpio_f*, *h5pset_dxpl_mpio_f*
    - Routines to create dataspaces (for to the dimensions of the whole file and to the portion of each processor): *h5screate_simple_f*
    - Routines to specify partial I/O, to indicate where block of data has to be written in the file: *h5sselect_hyperslab_f*
    - Routines to create and write a dataset (here we cannot use the high-level API): *h5dcreate_f*, *h5dwrite_f*
    - Routines to close objects: *h5dclose_f*, *h5sclose_f*, *h5pclose_f*



Topic

LaPalma3 overview

- One of the 12 supercomputers of the Spanish Supercomputing Network (RES) https://www.res.es/en
  - User's Guide: https://tinyurl.com/t6y7d9l
- Hardware overview
  - 252 IBM dx360 M4 compute nodes.
  - Each node:
    - 16 cores (Intel E5-2670) @ 2.6 GHz
    - 32 GB of RAM (2 GB per core) and 500GB of local disk storage.
  - 346 TB of disk storage via Lustre Parallel File System.
  - Interconnection networks:
    - Infiniband FDR10 40G Network: used by parallel applications communications.
    - Gigabit Network: Ethernet network for Lustre file system and for admin
- File systems
  - Root filesystem: where the operating system resides
  - Lustre filesystem, *recommended for most jobs*
    - /storage/home - user home directories
    - /storage/projects - shared space for members of a project
    - /storage/scratch - for temporary files during execution
    - /storage/apps - apps installed in LaPalma
  - Local hard drive: /scratch directory in each node, used for temporary files

Connecting and transferring files

- Connection to LaPalma
  - Authentication is done via public/private key pair (you must send your public key via email to *res_support@iac.es*)
  - Use ssh to connect to *lapalma1.iac.es*. For example:

    ```
    ssh -l can30003 lapalma1.iac.es
    ```
- Transfer files
  - You can use scp, rsync, etc., but remember that you cannot make outgoing connections from LaPalma for security reasons, so all transfers must be initiated in your local machine. For example:

    ```
    rsync -azv -e ssh WS_tutorial can30003@lapalma1.iac.es:/storage/projects/can30/
    ```

Available software

- Software in LaPalma is managed via Environment Modules: http://modules.sourceforge.net/
- Most used Modules commands:
  - List all available modules: *module avail*
  - Load a specific module: *module load <module_name>* For example: `module load gnu/7.2.0`
  - List all loaded modules: *module list*
  - Unload a module: *module unload <module_name>* For example: `module unload python/2.6`
  - Unload all modules: *module purge*

Executing jobs

- LaPalma3 uses Slurm to manage users' job submissions. (Slurm quick start guide: https://slurm.schedmd.com/quickstart.html)
- Write a Slurm submission script (see details at https://tinyurl.com/up4nm59). For example:

```
#!/bin/bash
#############################
#SBATCH -J test_mpi
#SBATCH -n 64
#SBATCH -t 05:30:00
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH -D .
#############################

module purge
module load gnu openmpi/gnu

mpirun ./myprogram_mpi
```

Executing jobs (2)

- Submit jobs to the queue with the following command (see details at https://tinyurl.com/tuujv38):

  ```
  sbatch <script_file>
  ```

- A number of Slurm commands are specially useful:
  - *idlenodes* and *idlecores*: check number of idle nodes and cores, respectively. Useful to dimension your job submissions if you want them to be executed as soon as possible.
  - *squeue* : check status of jobs.
  - *jobtimes* : command to get an estimate about starting and ending time for your pending jobs; or total, used and remaining time of the running jobs.
  - *scancel <job_id>* : remove a job from the queue

Heat2D in LaPalma3

- In order to run the different versions of Heat2D in LaPalma3:
  - Transfer the directory *WS_Tutorial_PP_HDF5_deVicente* to LaPalma3. For example:

    ```
    rsync -avz -e ssh WS_Tutorial_PP_HDF5_deVicente
                      can30003@lapalma1.iac.es:/storage/scratch/can30/can30003/
    ```

  - In LaPalma3 and inside the WS_Tutorial_PP_HDF5 directory run the *png.sh* script (*this is only needed once, because the PNG library is not installed by default in LaPalma3*).

    ```
    .../can30003/WS_Tutorial_PP_HDF5_deVicente $ ./png.sh
    ```

  - In each of the directories for any of the seven available versions (*serial* and *mpi-v[1-6]*) do:
    - run the *lmods.sh* script so that the right software for compilation is loaded

      ```
      .../Code-Heat2D/serial $ . lmods.sh
      ```

    - compile this code version:

      ```
      .../Code-Heat2D/serial $ make clean ; make
      ```

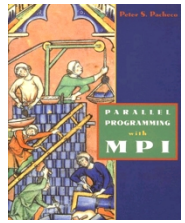
    - submit the job

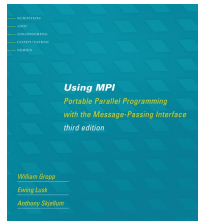
      ```
      .../Code-Heat2D/serial $ sbatch heat_submit.sh
      ```

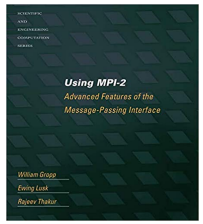
Topic

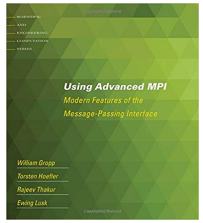
MPI References

- Parallel Programming with MPI. Peter Pacheco. Morgan Kaufmann, 1996
- Using MPI, Portable Parallel Programming with the Message-Passing Interface (3rd edition). William Gropp et. al. The MIT Press, 2014.
- Using MPI-2: Advanced Features of the Message-Passing Interface. William Gropp et. al. The MIT Press, 1999.
- Using Advanced MPI: Modern Features of the Message-Passing Interface. William Gropp et. al. The MIT Press, 2014.
- MPI: A Message-Passing Interface Standard, Version 3.1. Message Passing Interface Forum, 2015.









HDF5 References

- The HDF Group https://www.hdfgroup.org/
- Learning HDF5 resources: https://portal.hdfgroup.org/display/HDF5/Learning+HDF5
- Sample HDF5 programs in several languages: https://portal.hdfgroup.org/display/HDF5/HDF5+Examples
- Python and HDF5. Andrew Collette. O'Reilly Media, Inc., 2013: https://www.oreilly.com/library/view/python-and-hdf5/9781491944981/
- The HDF Group's Support website https://support.hdfgroup.org/ *Old, not maintained anymore, but useful information*

LaPalma3 References

- LaPalma3 webpage: http://research.iac.es/sieinvens/siepedia/pmwiki.php?n=HOWTOs.LaPalma3Intro
- Lustre filesystem: http://lustre.org/
- Slurm workload manager
  - Latest version: https://slurm.schedmd.com/
  - Version 18.08.7 (installed in LaPalma3): https://slurm.schedmd.com/archive/slurm-18.08.7/
    - Quick Start User Guide: https://slurm.schedmd.com/archive/slurm-18.08.7/quickstart.html

Credits

- Some of the images for this presentation are from:
  - Rolf Rabenseifner: https://fs.hlrs.de/projects/par/par_prog_ws/pw/2014-PAR/arch_models_which-2014-PAR.pdf
  - *Helloacm* website:

    https://helloacm.com/simple-tutorial-with-openmp-how-to-use-parallel-block-in-cc-using-openmp/
  - Hristo Iliev's presentation at PPCES'2016:

    https://doc.itc.rwth-aachen.de/display/VE/PPCES+2016
  - William Gropp: http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture25.pdf
  - Edgar Gabriel: http://www2.cs.uh.edu/~gabriel/courses/mpicourse_03_07/MPI_ddt.pdf
  - The HDF Group:
    - https://www.hdfgroup.org/wp-content/uploads/2017/12/HDF512-17.pdf
    - https://www.star.nesdis.noaa.gov/jpss/documents/HDF5_Tutorial_201509/1-2-Introduction%20to%20HDF5.ppt.pdf
  - Quincey Koziol and Scot Breitenfeld: https://www.alcf.anl.gov/files/Parallel_HDF5_1.pdf
- The Heat2D code is based on:

  https://repository.prace-ri.eu/git/CodeVault/training-material/parallel-programming/MPI