JÜLICH
FORSCHUNGSZENTRUM

# Parallel I/O and Portable Data Formats
## HDF5

Sebastian Lührs
s.luehrs@fz-juelich.de
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH

Jülich, March 14th, 2017

# Outline

- Introduction
  - Structure of the HDF5 library
  - Terms and definitions
- HDF5 - programming model and API
  - Creating/opening HDF5 files
  - Closing HDF5 files and other objects
  - HDF5 predefined datatypes
  - Creating dataspaces
  - Creating datasets
  - Writing/reading data
  - Row major / column major
  - Partial I/O
- Parallel HDF5

# What is HDF5?

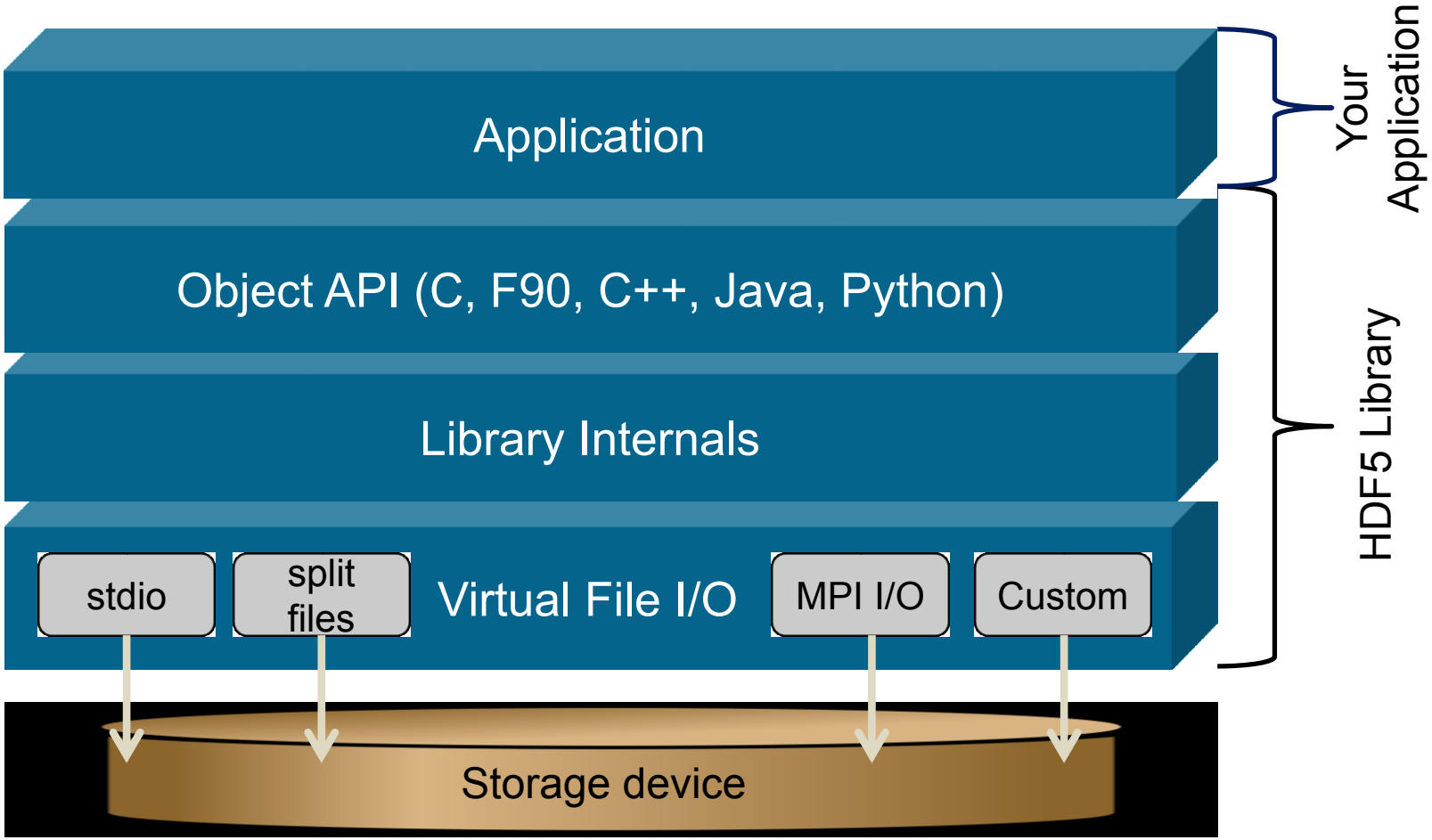
Hierarchical Data Format

- API, data model and file format for I/O management
- Tools suite for accessing data in HDF5 format

# HDF5 - Features

- Supports parallel I/O
- Self describing data model which allows the management of complex data sets
- Portable file format
- Available on a variety of platforms
- Supports C, C++, Fortran 90 and Java
  - Pythonic interfaces also available
- Provides tools to operate on HDF5 files and data

# Layers of the HDF5 Library



# File organization

- HDF5 file structure corresponds in many respects to a Unix/Linux file system (fs)

| HDF5 | | Unix/Linux fs |
|---|---|---|
| Group | ↔ | Directory |
| Data set | ↔ | File |

/DataSet1

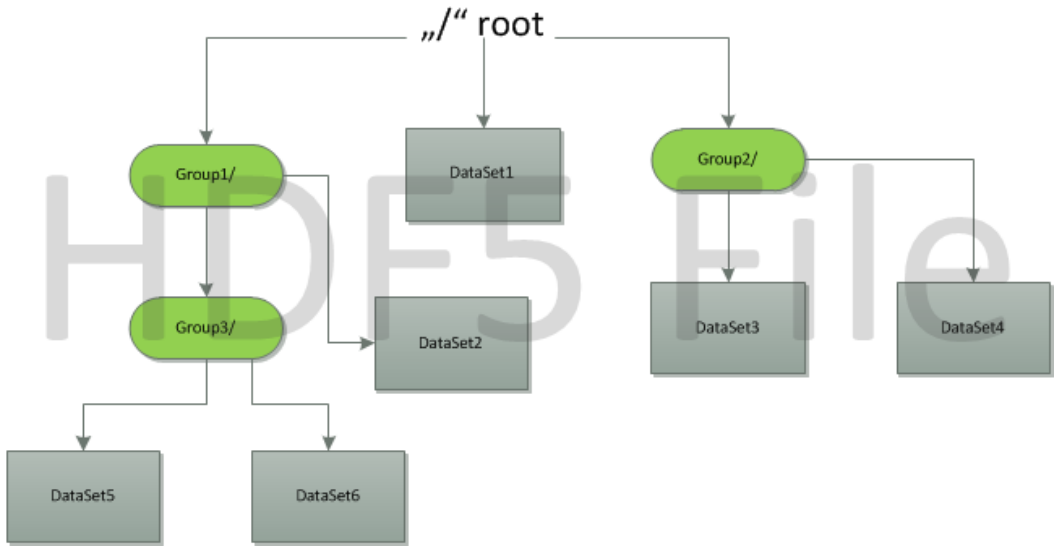
/Group1/DataSet2

/Group1/Group3/DataSet5

/Group1/Group3/DataSet6

/Group2/DataSet3

/Group2/Dataset4



# Terminology

**File**

Container for storing data

**Group**

Structure which may contain HDF5 objects, e.g. datasets, attributes, datasets

**Attribute**

Can be used to describe datasets and is attached to them

**Dataspace**

Describes the dimensionality of the data array and the shape of the data points respectively, i.e. it describes the shape of a dataset

**Dataset**

Multi-dimensional array of data elements

# Library specific types

**C**

```
#include hdf5.h
hid_t      Object identifier
herr_t     Function return value
hsize_t    Used for dimensions
hssize_t   Used for coordinates and dimensions
hvl_t      Variable length datatype
```

**Fortran**

```
use hdf5
INTEGER(HID_T)      Object identifier
INTEGER(HSIZE_T)    Used for dimensions
INTEGER(HSSIZE_T)   Used for coordinates and dimensions
```

- Defined types are integers of different size
- Own defined types ensure portability

# Fortran HDF5 open

- The HDF5 library interface needs to be initialized (e.g. global variables) by calling `H5OPEN_F` before it can be used in your code and closed (`H5CLOSE_F`) at the end.

Fortran

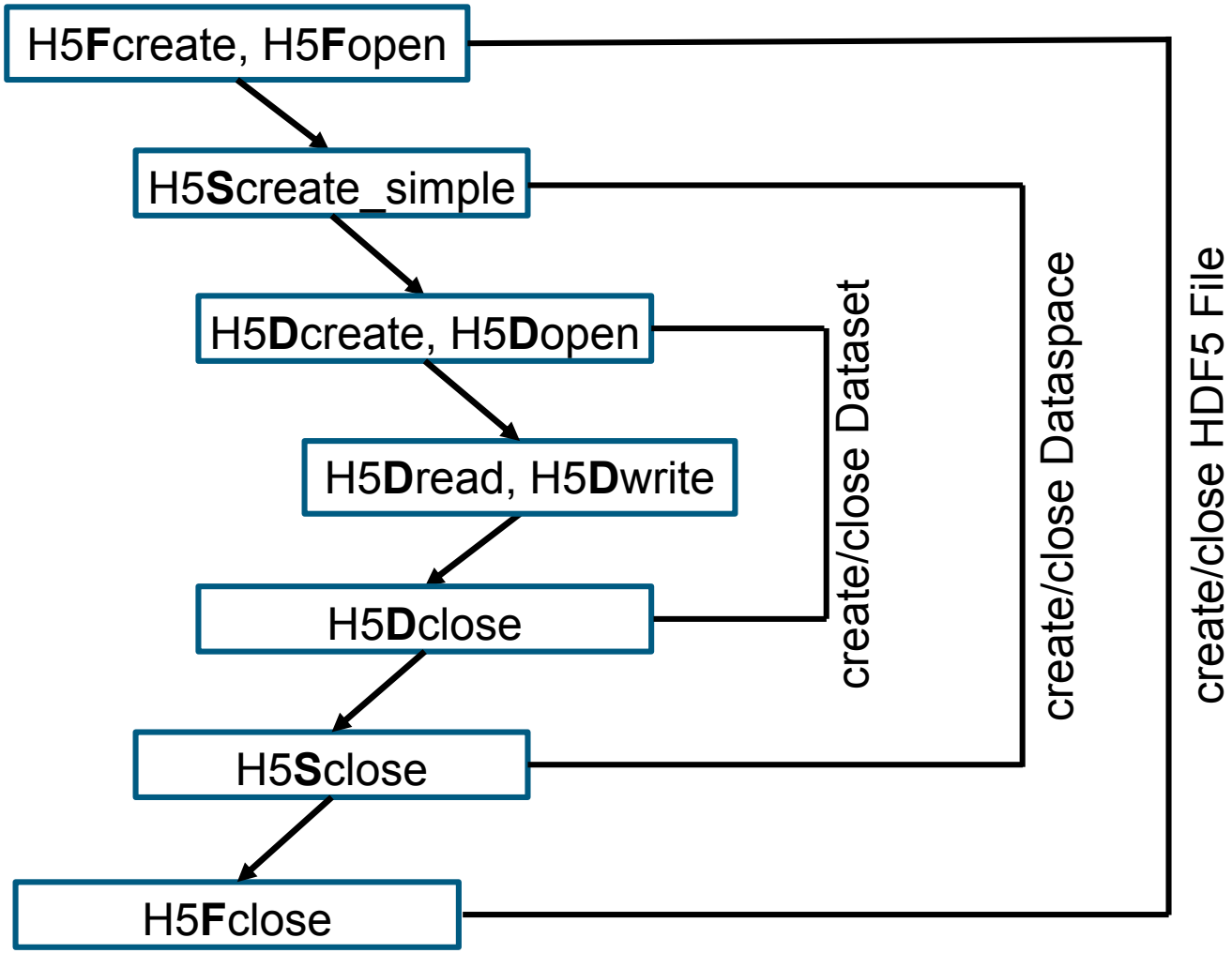
```
H5OPEN_F(STATUS)
  INTEGER, INTENT(OUT) :: STATUS

H5CLOSE_F(STATUS)
  INTEGER, INTENT(OUT) :: STATUS
```

- `status` returns `0` if successful

# API naming scheme (excerpt)

- `H5`
  - Library functions: general-purpose functions
- `H5D`
  - Dataset interface: dataset access and manipulation routines
- `H5G`
  - Group interface: group creation and manipulation routines
- `H5F`
  - File interface: file access routines
- `H5P`
  - Property list interface: object property list manipulation routines
- `H5S`
  - Dataspace interface: dataspace definition and access routines

# General Procedure



# Creating an HDF5 file

C

```
hid_t H5Fcreate(const char *name, unsigned
                access_flag, hid_t creation_prp,
                hid_t access_prp)
```

Fortran

```
H5FCREATE_F(NAME, ACCESS_FLAGS, FILE_ID, HDFERR,
            CREATION_PRP, ACCESS_PRP)
  CHARACTER(*), INTENT(IN) :: NAME
  INTEGER, INTENT(IN) :: ACCESS_FLAGS
  INTEGER(KIND=HID_T), INTENT(OUT) :: FILE_ID
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(KIND=HID_T), OPTIONAL, INTENT(IN) ::
    CREATION_PRP, ACCESS_PRP
```

- `name`: Name of the file
- `access_flags`: File access flags
- `creation_prp` and `access_prp`: File creation and access property list, `H5P_DEFAULT[_F]` if not specified
- Fortran uses `file_id` as return value

# Opening an existing HDF5 file

C

```
hid_t H5Fopen(const char *name, unsigned flags,
              hid_t access_prp)
```

Fortran

```
H5FOPEN_F(NAME, FLAGS, FILE_ID, HDFERR,
          ACCESS_PRP)
  CHARACTER(*), INTENT(IN) :: NAME
  INTEGER, INTENT(IN) :: FLAGS
  INTEGER(KIND=HID_T), INTENT(OUT) :: FILE_ID
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(KIND=HID_T), OPTIONAL, INTENT(IN) ::
    ACCESS_PRP
```

- `name`: Name of the file
- `access_prp`: File access property list, `H5P_DEFAULT[_F]` if not specified
- Fortran uses `file_id` as return value
- Avoid multiple opens of the same file

# Access modes

- `H5F_ACC_TRUNC[_F]`: Create a new file, overwrite an existing file
- `H5F_ACC_EXCL[_F]`: Create a new file, `H5Fcreate` fails if file already exists
- `H5F_ACC_RDWR[_F]`: Open file in read-write mode, irrelevant for `H5Fcreate[_f]`
- `H5F_ACC_RDONLY[_F]`: Open file in read-only mode, irrelevant for `H5Fcreate[_f]`
- More specific settings are controlled through file creation property list (`creation_prp`) and file access property lists (`access_prp`) which defaults to `H5P_DEFAULT[_F]`
- `creation_prp` controls file metadata
- `access_prp` controls different methods of performing I/O on files

# Group creation

C

```
hid_t H5Gcreate(hid_t loc_id, const char *name,
                hid_t lcpl_id, hid_t gcpl_id,
                hid_t gapl_id )
```

Fortran

```
H5GCREATE_F(LOC_ID, NAME, GRP_ID, HDFERR,
            SIZE_HINT, LCPL_ID, GCPL_ID, GAPL_ID)
  INTEGER(KIND=HID_T), INTENT(IN) :: LOC_ID
  CHARACTER(LEN=*), INTENT(IN) :: NAME
  INTEGER(KIND=HID_T), INTENT(OUT) :: GRP_ID
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(KIND=SIZE_T), OPTIONAL, INTENT(IN) ::
    SIZE_HINT
  INTEGER(KIND=HID_T), OPTIONAL, INTENT(IN) ::
    LCPL_ID, GCPL_ID, GAPL_ID
```

- `loc_id`: Can be the `file_id` or another `group_id`
- `name` can be an absolute or relative path
- `lcpl_id, gcpl_id, gapl_id`: Property lists for link/group
- use `H5Gclose[_f]` to finalize group access

# Closing an HDF5 file

C

```
herr_t H5Fclose(hid_t file_id)
```

Fortran

```
H5FCLOSE_F(FILE_ID, HDFERR)
  INTEGER(KIND=HID_T), INTENT(IN) :: FILE_ID
  INTEGER, INTENT(OUT) :: HDFERR
```

# Exercise

**Exercise 1 – HDF5 hello world**

- Write a serial program in C or Fortran which creates and closes an HDF5 file
- Create a group “data” inside of this file

Check the resulting file using:
`h5dump`

```
module load intel-para
module load HDF5/1.8.16
mpicc helloworld_hdf5.c -lhdf5

module load intel-para
module load HDF5/1.8.16
mpif90 helloworld_hdf5.f90 -lhdf5_fortran
```

- Solutions and templates: `/work/hpclab/train001`

# HDF5 pre-defined datatypes (excerpt)

| C | C type | HDF5 file type (pre-defined) | HDF5 memory type (native) |
|---|---|---|---|
| | `int` | `H5T_STD_I32[BE,LE]` | `H5T_NATIVE_INT` |
| | `float` | `H5T_IEEE_F32[BE,LE]` | `H5T_NATIVE_FLOAT` |
| | `double` | `H5T_IEEE_F64[BE,LE]` | `H5T_NATIVE_DOUBLE` |

| Fortran | F type | HDF5 file type (pre-defined) | HDF5 memory type (native) |
|---|---|---|---|
| | `integer` | `H5T_STD_I32[BE,LE]` | `H5T_NATIVE_INTEGER` |
| | `real` | `H5T_IEEE_F32[BE,LE]` | `H5T_NATIVE_REAL` |

- Native datatype might differ from platform to platform
- HDF5 file type depends on compiler switches and underlying platform
- Native datatypes are not in an HDF file but the pre-defined ones which are referred to by native datatypes appear in the HDF5 files.

# Dataspace

- The dataspace is part of the metadata of the underlying dataset
- Metadata are:
  - Dataspace
  - Datatype
  - Attributes
  - Storage info
- The dataspace describes the size and shape of the dataset

**Simple dataspace**

```
rank: int
current_size: hsize_t[rank]
maximum_size: hsize_t[rank]
```

rank = `2`, dimensions = `2x5`

# Creating a dataspace

C

```
hid_t H5Screate_simple(int rank,
                       const hsize_t *current_dims,
                       const hsize_t *maximum_dims)
```

Fortran

```
H5SCREATE_SIMPLE_F(RANK, DIMS, SPACE_ID, HDFERR,
                   MAXDIMS)
  INTEGER, INTENT(IN) :: RANK
  INTEGER(KIND=HISIZE_T)(*), INTENT(IN) :: DIMS
  INTEGER(KIND=HID_T), INTENT(OUT) :: SPACE_ID
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(KIND=HISIZE_T)(*), OPTIONAL,
    INTENT(OUT) :: MAXDIMS
```

- `rank`: Number of dimensions
- `maximum_dims` may be `NULL`. Then `maximum_dims` and `current_dims` are the same
- `H5S_UNLIMITED[_F]` can be used as `maximum_dims` to set dimensions to “infinite” size
- use `H5Sclose[_f]` to finalize dataspace access

# Creating a dataspace

C

```
hid_t H5Screate(H5S_class_t type)
```

Fortran

```
H5SCREATE_F(CLASSTYPE, SPACE_ID, HDFERR)
  INTEGER, INTENT(IN) :: CLASSTYPE
  INTEGER(HID_T), INTENT(OUT) :: SPACE_ID
  INTEGER, INTENT(OUT) :: HDFERR
```

- `classtype`: `H5S_SCALAR[_F]` or `H5S_SIMPLE[_F]`

# Creating an Attribute

C

```
hid_t H5Acreate(hid_t loc_id, const char *attr_name,
                hid_t type_id, hid_t space_id,
                hid_t acpl_id, hid_t aapl_id)
```

Fortran

```
H5ACREATE_F(LOC_ID, NAME, TYPE_ID, SPACE_ID,
            ATTR_ID, HDFERR, ACPL_ID, AAPL_ID)
  INTEGER(KIND=HID_T), INTENT(IN) :: LOC_ID
  CHARACTER(LEN=*), INTENT(IN) :: NAME
  INTEGER(KIND=HID_T), INTENT(IN) :: TYPE_ID,
    SPACE_ID
  INTEGER(KIND=HID_T), INTENT(OUT) :: ATTR_ID
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(KIND=HID_T), OPTIONAL, INTENT(IN) ::
    ACPL_ID, AAPL_ID
```

- `loc_id` may be any HDF5 object identifier (group, dataset, or committed datatype) or an HDF5 file identifier
- `ACPL_ID, AAPL_ID`: `H5P_DEFAULT[_F]` if not specified
- use `H5Aclose[_f]` to finalize the attribute access

# Writing an Attribute

C

```
herr_t H5Awrite(hid_t attr_id, hid_t mem_type_id,
                const void *buf)
```

Fortran

```
H5AWRITE_F(ATTR_ID, MEMTYPE_ID, BUF, DIMS, HDFERR)
    INTEGER(KIND=HID_T), INTENT(IN) :: ATTR_ID
    INTEGER(KIND=HID_T), INTENT(IN) :: MEMTYPE_ID
    TYPE, INTENT(IN) :: BUF
    INTEGER(KIND=HSIZE_T)(*), INTENT(IN):: DIMS
    INTEGER, INTENT(OUT) :: HDFERR
```

- Fortran: `DIMS` array to hold corresponding dimension sizes of data buffer `buf` (new since 1.4.2)

# Writing an Attribute

- StringType Example (C):

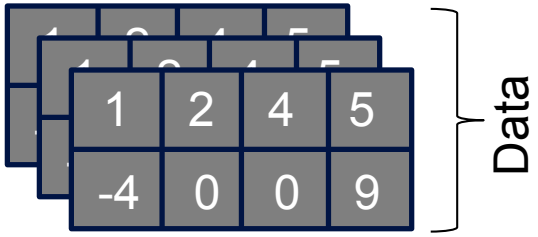
```
atype = H5Tcopy(H5T_C_S1);
H5Tset_size(atype, 5);
H5Tset_strpad(atype,H5T_STR_NULLTERM);
…
H5Tclose(atype);
```

alternative:
H5T_STR_SPACEPAD
H5T_STR_NULLPAD

- StringType Example (Fortran):

```
call H5Tcopy_f(H5T_C_S1,atype,status)
call H5Tset_size_f(atype,
                   int(5,HSIZE_T),status)
call H5Tset_strpad_f(atype,H5T_STR_NULLTERM_F)
…
call H5Tclose_f(atype, status)
```

# Dataset (metadata + data)




Metadata

Dataspace
- rank = 3
- dim[0] = 2
- dim[1] = 4
- dim[2] = 3

Datatype
- Integer

Attributes
- Time = 2.1
- Temp = 122

Storage
- Contiguous

# Creating a Dataset

C

```
hid_t H5Dcreate(hid_t loc_id, const char *name,
                hid_t dtype_id, hid_t space_id,
                hid_t lcpl_id, hid_t dcpl_id,
                hid_t dapl_id)
```

Fortran

```
H5DCREATE_F(LOC_ID, NAME, TYPE_ID, SPACE_ID,
          DSET_ID, HDFERR, DCPL_ID, LCPL_ID, DAPL_ID)
  INTEGER(KIND=HID_T), INTENT(IN) :: LOC_ID
  CHARACTER(LEN=*), INTENT(IN) :: NAME
  INTEGER(KIND=HID_T), INTENT(IN) :: TYPE_ID,
    SPACE_ID
  INTEGER(KIND=HID_T), INTENT(OUT) :: DSET_ID
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(KIND=HID_T), OPTIONAL, INTENT(IN) ::
    DCPL_ID, LCPL_ID, DAPL_ID
```

- use `H5Dclose[_f]` to finalize the dataset access

# Creating a Dataset

- `type_id`: Datatype identifier
- `space_id`: Dataspace identifier
- `dcpl_id`: Dataset creation property list
- `lcpl_id`: Link creation property list
- `dapl_id`: Dataset access property list

# Property Lists

- Property lists (`H5P`) can be used to change the internal data handling in HDF5
- Default: `H5P_DEFAULT[_F]`
- Creation properties
  - Whether a dataset is stored in a compact, contiguous, or chunked layout
  - Specify filters to be applied to a dataset (e.g. gzip compression or checksum evaluation)
- Access properties
  - The driver used to open a file (e.g. MPI-I/O or Posix)
  - Optimization settings in specialized environments
- Transfer properties
  - Collective or independent I/O

# Recipe: Creating an empty dataset

1. Get identifier for dataset location
2. Specify datatype (integer, composite etc.)
3. Define dataspace
4. Specify property lists (or `H5P_DEFAULT[_F]`)
5. Create dataset
6. Close all opened objects

# Exercise

**Exercise 2 – HDF5 metadata handling**

- Extend your serial program
- Create inside the “data” group an empty dataset which should be a two dimensional array (5x20 elements) of integer values
- Add a string attribute connected to this dataset
- Write a string value into this attribute

Check the resulting file using:

```
h5dump
```

# Writing to a dataset

C

```
herr_t H5Dwrite(hid_t dataset_id, hid_t mem_type_id,
                hid_t mem_space_id, hid_t
                file_space_id, hid_t xfer_plist_id,
                const void * buf )
```

Fortran

```
H5DWRITE_F(DSET_ID, MEM_TYPE_ID, BUF, DIMS, HDFERR,
           MEM_SPACE_ID, FILE_SPACE_ID, XFER_PRP)
  INTEGER(HID_T), INTENT(IN) :: DSET_ID, MEM_TYPE_ID
  TYPE, INTENT(IN) :: BUF
  DIMENSION(*), INTEGER(HSIZE_T), INTENT(IN) :: DIMS
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(HID_T), OPTIONAL, INTENT(IN) ::
    MEM_SPACE_ID, FILE_SPACE_ID, XFER_PRP
```

- `H5S_ALL[_F]` can be used to specify no special `mem_space` or `file_space` identifier
- `xfer_plist_id/xfer_prp` is a transfer property (e.g. to specify collective or independent parallel I/O)

# Writing to a dataset

| mem_space_id | file_space_id | Behaviour |
|---|---|---|
| dataspace id | dataspace id | use dataspaces as is |
| H5S_ALL | dataspace id | use given file_space dataspace also for mem_space dataspace (including the selection) |
| dataspace id | H5S_ALL | use *all* selection for default file_space |
| H5S_ALL | H5S_ALL | use default file_space also for mem_space, set *all* selection for both |

# Open a existing dataset

C

```
hid_t H5Dopen(hid_t loc_id, const char *name, hid_t
              dapl_id)
```

Fortran

```
H5DOPEN_F(LOC_ID, NAME, DSET_ID, HDFERR)
  INTEGER(HID_T), INTENT(IN) :: LOC_ID
  CHARACTER(LEN=*), INTENT(IN) :: NAME
  INTEGER(HID_T), INTENT(OUT) :: DSET_ID
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: DAPL_ID
```

- `dapl_id`: Dataset access property list

# Dataspace inquiry

C
```
hid_t H5Dget_space(hid_t dataset_id)
```

Fortran
```
H5DGET_SPACE_F(DATASET_ID, DATASPACE_ID, HDFERR)
  INTEGER(HID_T), INTENT(IN) :: DATASET_ID
  INTEGER(HID_T), INTENT(OUT) :: DATASPACE_ID
  INTEGER, INTENT(OUT) :: HDFERR
```

- Returns an identifier for a copy of the dataspace for a dataset.
- `H5Sget_simple_extent_ndims` and `H5Sget_simple_extent_dims` can be used to extract dimension information

# Reading a dataset

C

```
herr_t H5Dread(hid_t dataset_id, hid_t mem_type_id,
               hid_t mem_space_id, hid_t
               file_space_id, hid_t xfer_plist_id,
               void * buf)
```

Fortran

```
H5DREAD_F(DSET_ID, MEM_TYPE_ID, BUF, DIMS, HDFERR,
          MEM_SPACE_ID, FILE_SPACE_ID, XFER_PRP)
  INTEGER(HID_T), INTENT(IN) :: DSET_ID, MEM_TYPE_ID
  TYPE, INTENT(IN) :: BUF
  DIMENSION(*), INTEGER(HSIZE_T), INTENT(IN) :: DIMS
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(HID_T), OPTIONAL, INTENT(IN) ::
    MEM_SPACE_ID, FILE_SPACE_ID, XFER_PRP
```

- `H5S_ALL[_F]` can be used to specify no special `mem_space` or `file_space` identifier
- `xfer_plist_id/xfer_prp` is a transfer property (e.g. to specify collective or independent parallel I/O)

# Exercise

**Exercise 3 – HDF5 write data**

- Extend your serial program
- Create a two dimensional array with values 1 up to 100

```
 1  2  3  4  5  6  7 …
21 22 23 24 25 26 27 …
41 42 43 44 45 46 47 …
…
```

- Write this array into the existing empty HD5 dataset

Check the resulting file using:

```
h5dump
```

# Excursion: row-major / column-major order

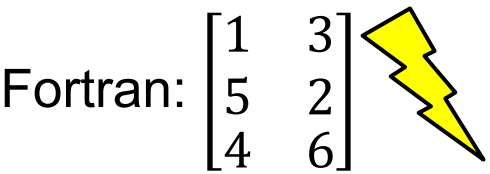
"Logical" data view:

$$M[i,j] = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

| Adress | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value C | 1 | 2 | 3 | 4 | 5 | 6 |
| Value Fortran | 1 | 3 | 5 | 2 | 4 | 6 |

Storing data in a `3x2` dimensional HDF5 dataset:

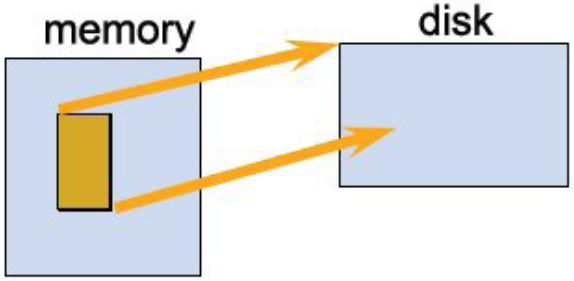
$$\text{C: } \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad \text{Fortran: } \begin{bmatrix} 1 & 3 \\ 5 & 2 \\ 4 & 6 \end{bmatrix}$$



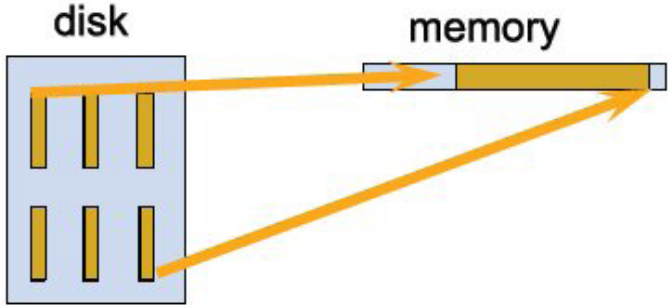
Storing data in a `2x3` dimensional dataset:

$$\text{Fortran: } \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

# Partial I/O - Hyperslabs

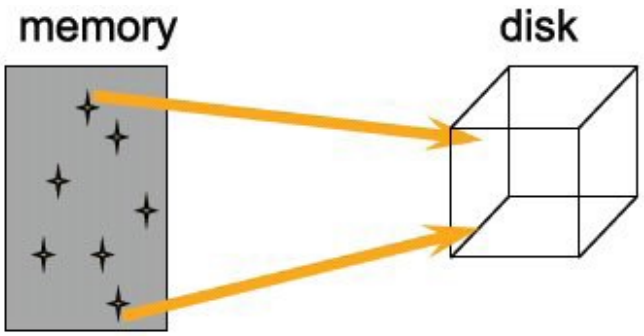



(a) Hyperslab from a 2D array to the corner of a smaller 2D array

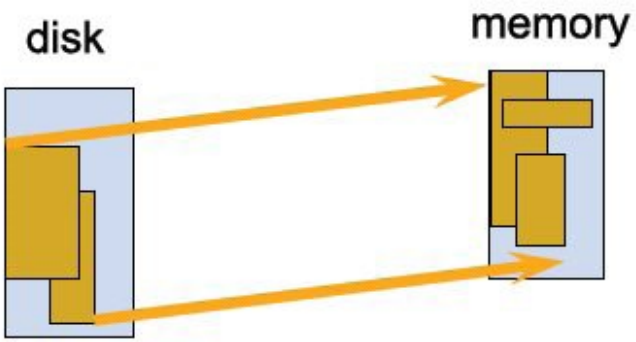



(b) Regular series of blocks from a 2D array to a contiguous sequence at a certain offset in a 1D array

# Partial I/O - Hyperslabs




(c) A sequence of points from a 2D array to a sequence of points in a 3D array.



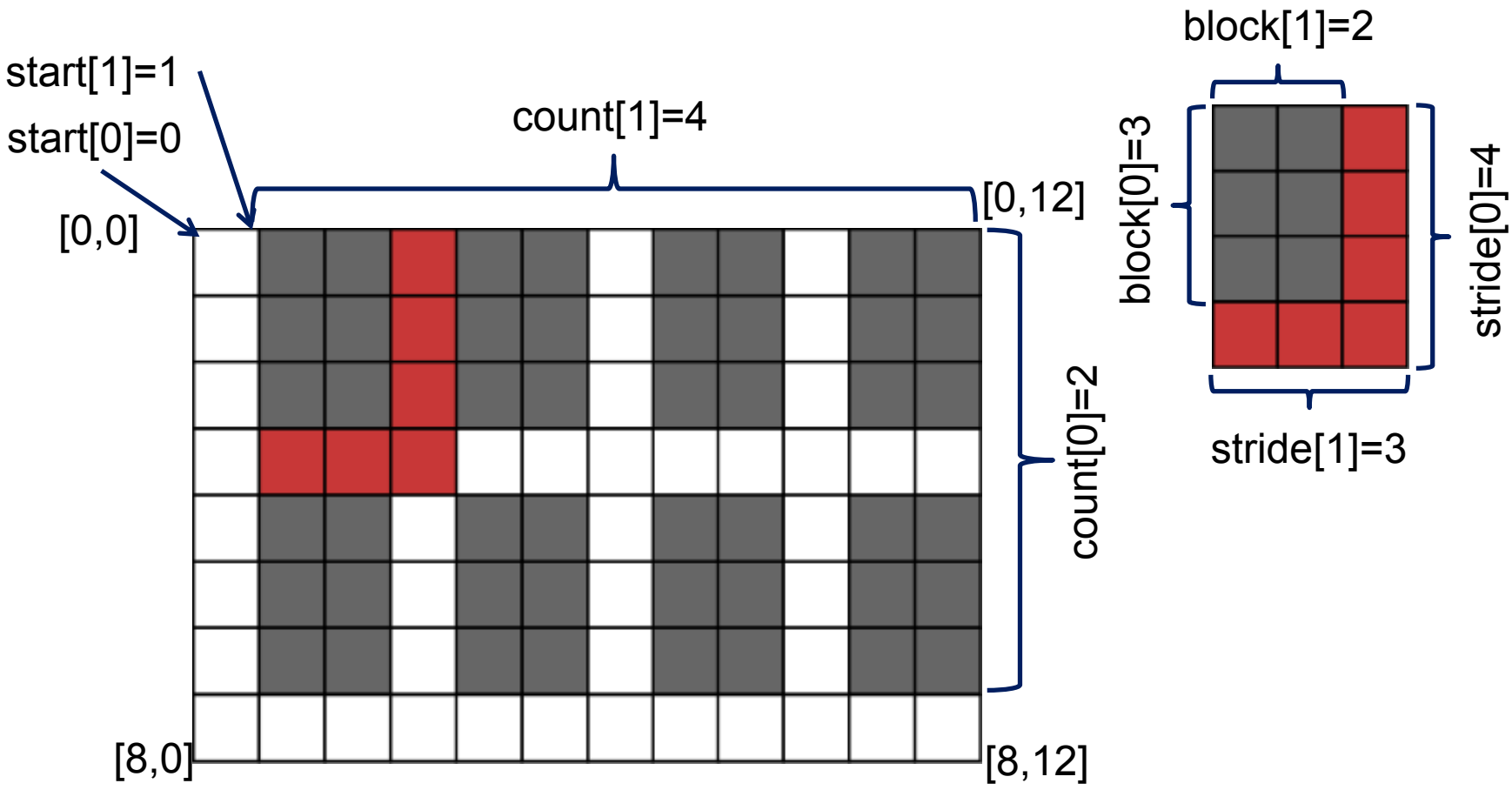

(d) Union of hyperslabs in file to union of hyperslabs in memory.

# Partial I/O - Hyperslabs

- Hyperslabs are portions of datasets
  - Contiguous collection of points in a dataspace
  - Regular pattern of points in a dataspace
  - Blocks in a dataspace

- Hyperslabs are described by four parameters:
  - start: (or offset): starting location
  - stride: separation blocks to be selected
  - count: number of blocks to be selected
  - block: size of block to be selected from dataspace
  - **Dimension of these four parameters corresponds to dimension of the underlying dataspace**

# Hyperslab example



# Creating hyperslabs

C

```
herr_t H5Sselect_hyperslab(hid_t space_id,
            H5S_seloper_t op, const hsize_t *start,
            const hsize_t *stride, const hsize_t
            *count, const hsize_t *block )
```

Fortran

```
H5SSELECT_HYPERSLAB_F(SPACE_ID, OPERATOR, START,
          COUNT, HDFERR, STRIDE, BLOCK)
  INTEGER(HID_T), INTENT(IN) :: SPACE_ID
  INTEGER, INTENT(IN) :: OP
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(IN) ::
    START, COUNT
  INTEGER, INTENT(OUT) :: HDFERR
  INTEGER(HSIZE_T), DIMENSION(*), OPTIONAL,
    INTENT(IN) :: STRIDE, BLOCK
```

# Creating hyperslabs

The following operators (`op`) are supported to combine old and new selections:

- `H5S_SELECT_SET[_F]`: Replaces the existing selection with the parameters from this call. Overlapping blocks are not supported with this operator.
- `H5S_SELECT_OR[_F]`: Adds the new selection to the existing selection.
- `H5S_SELECT_AND[_F]`: Retains only the overlapping portions of the new selection and the existing selection.
- `H5S_SELECT_XOR[_F]`: Retains only the elements that are members of the new selection or the existing selection, excluding elements that are members of both selections.
- `H5S_SELECT_NOTB[_F]`: Retains only elements of the existing selection that are not in the new selection.
- `H5S_SELECT_NOTA[_F]`: Retains only elements of the new selection that are not in the existing selection.

JÜLICH
FORSCHUNGSZENTRUM

# Parallel I/O and Portable Data Formats
## Parallel HDF5

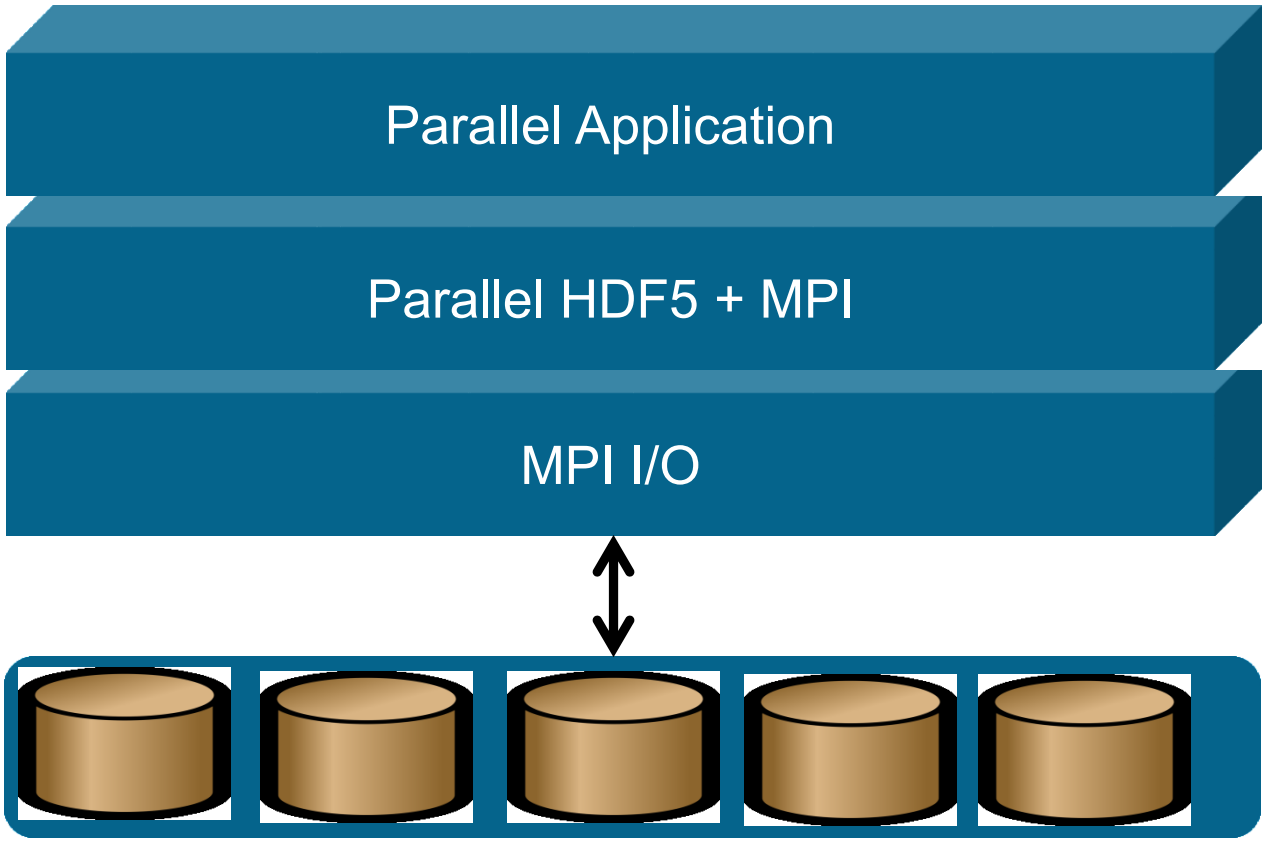
Sebastian Lührs
s.luehrs@fz-juelich.de
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH

Jülich, March 14th, 2017

# Factoids

- Supports MPI programming
- PHDF5 files compatible with serial HDF5 files
  - Shareable between different serial or parallel platforms
- Single file image to all processes
  - One file per process design is undesirable
- A standard parallel I/O interface must be portable to different platforms.

# Implementation layers




User Application

HDF5 library

Parallel I/O layer

Parallel file systems
(GPFS, Lustre, …)

# Important to know

- Most functions of the PHDF5 API are collectives
  - i.e. all processes of the communicator must participate
- PHDF5 opens a parallel file with a communicator
  - Returns a file-handle
  - Future access to the file via the file-handle
  - Different files can be opened via different communicators
- After a file is opened by the processes of a communicator
  - All parts of file are accessible by all processes
  - All objects in the file are accessible by all processes
  - Multiple processes may write to the same data array
  - Each process may write to an individual data array

# MPI-IO access template

C

```
hid_t H5Pcreate(hid_t cls_id);
herr_t H5Pset_fapl_mpio(hid_t fapl_id, MPI_Comm
                        comm, MPI_Info info)
```

Fortran

```
H5PCREATE_F(CLASSTYPE, PRP_ID, HDFERR)
  INTEGER, INTENT(IN) :: CLASSTYPE
  INTEGER(HID_T), INTENT(OUT) :: PRP_ID
  INTEGER, INTENT(OUT) :: HDFERR
H5PSET_FAPL_MPIO_F(PRP_ID, COMM, INFO, HDFERR)
  INTEGER(HID_T), INTENT(IN) :: PRP_ID
  INTEGER, INTENT(IN) :: COMM
  INTEGER, INTENT(IN) :: INFO
  INTEGER, INTENT(OUT) :: HDFERR
```

- `cls_id/classtype` must be `H5P_FILE_ACCESS[_F]`
- Property is used during file creation/access
- Each process of the MPI communicator creates an access template and sets it up with MPI parallel access information

# Dataset transfer property

C

```
hid_t H5Pcreate(hid_t cls_id);
herr_t H5Pset_dxpl_mpio(hid_t dxpl_id,
                        H5FD_mpio_xfer_t xfer_mode )
```

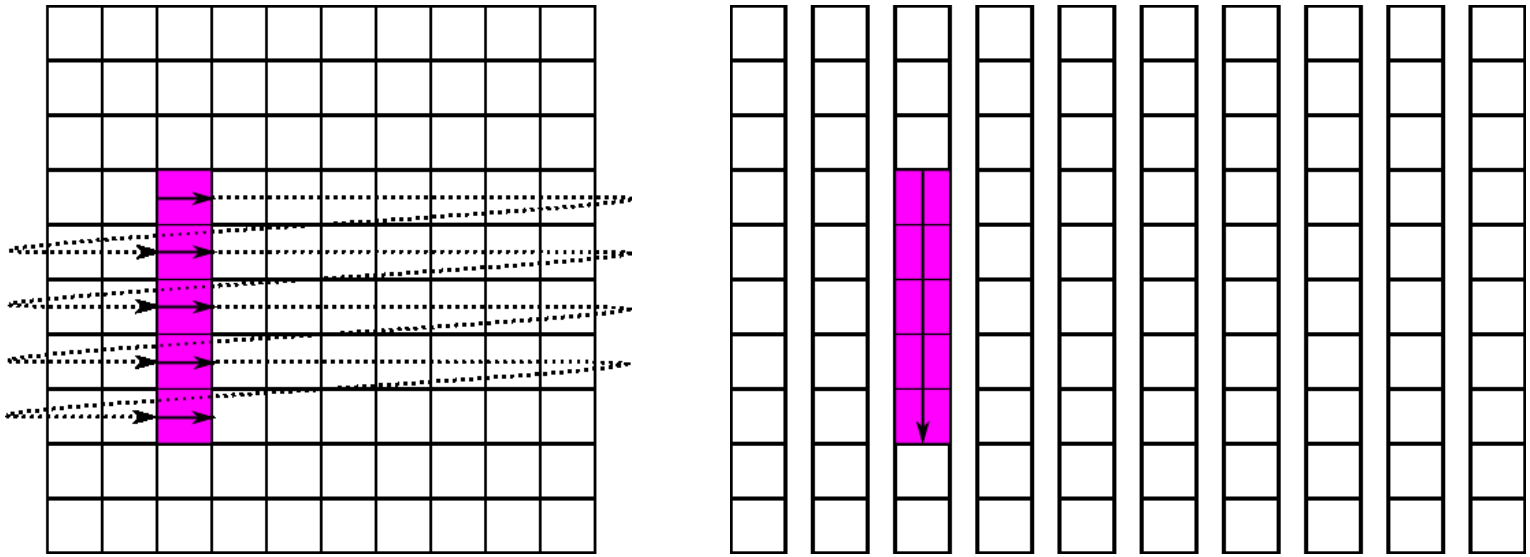
Fortran

```
H5PCREATE_F(CLASSTYPE, PRP_ID, HDFERR)
  INTEGER, INTENT(IN) :: CLASSTYPE
  INTEGER(HID_T), INTENT(OUT) :: PRP_ID
  INTEGER, INTENT(OUT) :: HDFERR
H5PSET_DXPL_MPIO_F(PRP_ID, DATA_XFER_MODE, HDFERR)
  INTEGER(HID_T), INTENT(IN) :: PRP_ID
  INTEGER, INTENT(IN) :: DATA_XFER_MODE
  INTEGER, INTENT(OUT) :: HDFERR
```

- `cls_id/classtype` must be `H5P_DATASET_XFER[_F]`
- `xfer_modes:`
  - `H5FD_MPIO_INDEPENDENT[_F]`: Use independent I/O access (default)
  - `H5FD_MPIO_COLLECTIVE[_F]`: Use collective I/O access

# Performance hints

- **Chunking:** Contiguous datasets are stored in a single block in the file, chunked datasets are split into multiple chunks which are all stored separately in the file.
- Additional chunk cache is possible



C

```
dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_chunk(dcpl_id, 2, chunk_dims);
```

*https://www.hdfgroup.org/HDF5/doc/Advanced/Chunking/*

# Performance hints

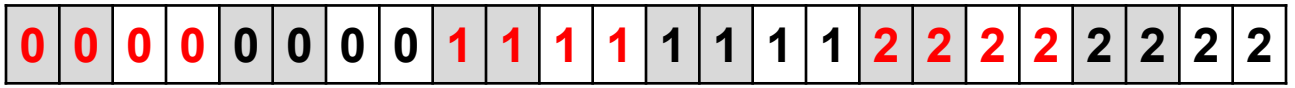
**h5perf**

- Simple HDF5 I/O-benchmark application
- 1D or 2D dataset
- Part of the standard HDF5 installation
- Contiguous or interleaved access pattern
- Independent and collective I/O
- Chunking
- Example Options (`h5perf -h`):
  - 1D / 2D (`-g`)
  - Bytes per Process (`-e`)
  - Block size (`-B`)
  - Transfer size (`-x / -X`)
  - Number of datasets (`-d`)

# Performance hints

Example (1D):

- `num-processes = 3`
- `bytes-per-process = 8`
- `block-size = 2`
- `transfer-buffer-size = 4`
- `contiguous`




1 write operation per transfer

- `interleaved`


2 write operations per transfer

*https://www.hdfgroup.org/HDF5/doc/Tools/h5perf_parallel/h5perf_parallel.pdf*

# Performance hints

Example (2D):

- `num-processes = 2`
- `bytes-per-process = 4`
- `block-size = 2`
- `transfer-buffer-size = 8`

`contiguous`

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

1 write operation per transfer

`interleaved`

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

8 write operations per transfer

*https://www.hdfgroup.org/HDF5/doc/Tools/h5perf_parallel/h5perf_parallel.pdf*

# Exercise

**Exercise 4 – parallel HDF5**

- Extend your serial program to a parallel program
- Fill your two dimensional array with the rank number
- Create a combined dataset of all processes involved
- Logical view:

```
0 0 0 0 0 0 0 0 …
0 0 0 0 0 0 0 0 …
…
1 1 1 1 1 1 1 1 …
1 1 1 1 1 1 1 1 …
…
```

#cores x 5

- Write the data collectively into the file
- Check the resulting file using: `h5dump`
- Template is available: `/work/hpclab/train001`

- Time left? Try to run a `h5perf` benchmark (use `h5perf -h` to see all available options) or run the Mandelbrot hdf5 implementation