MAE 5032 High Performance Computing: Methods and Practices

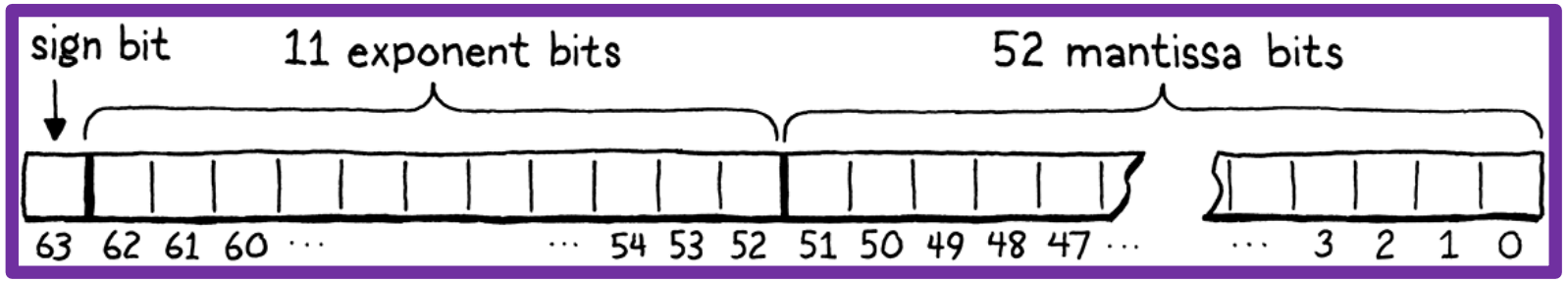
Lecture 11: I/O with HDF5

Ju Liu
Department of Mechanics and Aerospace Engineering
liuj36@sustech.edu.cn

Motivation

- Most of us need to read and write data at some point
  - Read geometry configurations for simulation driven research
  - Data mining algorithms
  - Post-process generated data (simple statistics or complicated visualizations)
  - Continuing from a restart file

- There are three main aspects of datt files that affect portability:
  - floating point representation
    - Some machines such as Cray employ their own representation for floating point numbers (17-bit exponent and 47 bit mantissa); most machines now use the IEEE 754 representation (11-bit exponent and 52 bit mantissa)
  - byte-ordering
  - I/O formating



Motivation

- Byte ordering
  - a **byte** is the lowest addressable storage unit
  - There are two different ways that the individual bytes of a number are stored in disk and memory. This ordering has nothing to do with the individual bits in each byte, only the order of how the bytes are stored
  - These two orderings are called **little** and **big endian** storage order.

Little vs Big Endian storage

- Little endian: the bytes are stored from the least significant byte to the highest, beginning at the lowest address. The word is stored as “little end first”.

- Big endian: the bytes are stored from the most significant byte to the lowest, beginning at the lowest address. The word is stored as “big end first”.

Example: to write 1234abcd to memory starting from 0x0000:

address	big-endian	little-endian
0x0000	0x12	0xcd
0x0001	0x34	0xab
0x0002	0xab	0x34
0x0003	0xcd	0x12

Little vs Big Endian storage

- Examples of ordering in common architectures

Processor	Endianess
Pentium 4	little-endian
Itanium 2	little-endian
Athlon	little-endian
Opteron	little-endian
PC970 (G5 - MAC)	big-endian
Power3/4/5 (AIX)	big-endian
MIPS (IRIX)	big-endian
Cray (Unicos)	big-endian

I/O formatting

- Fortran sequential unformatted files are formatted to contain additional information to aid in file seeks.
- For each Fortran write, a 4-byte integer is inserted at the beginning and end of the data sequence being written to disk.
- The two 4-byte additions are identical integers containing the number of data bytes.
- If we read big endian files from a little endian platform, we will have to swap the record header and footer, and the individual elements.

Platform independent binary files

- NETCDF: Unidata Netword Common Data Form
  - Common format used in Climate/Weather/Ocean applications
  - https://www.unidata.ucar.edu/software/netcdf/
  - `module load netcdf; man netcdf`

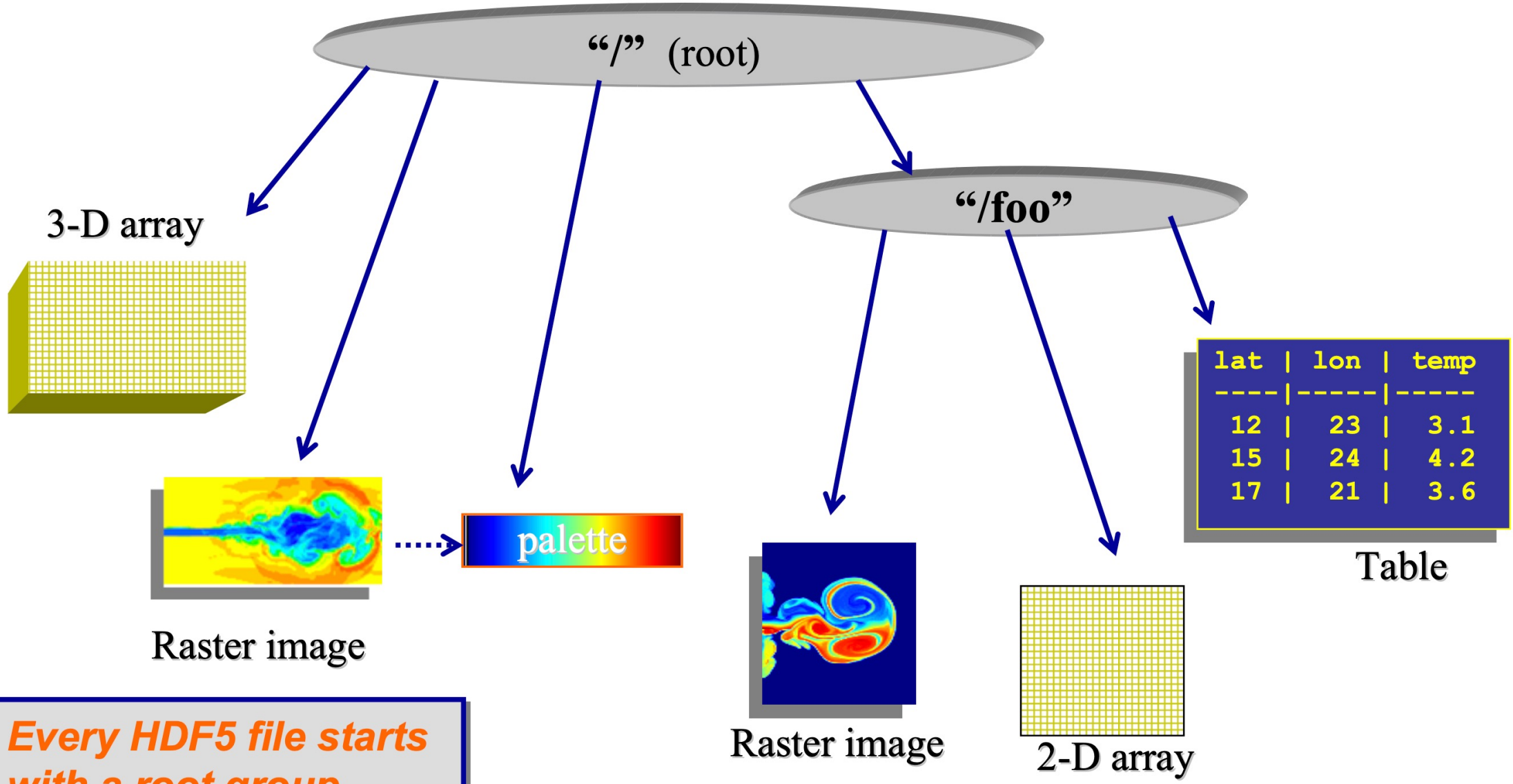
- HDF: Hierarchical Data Format developed by NCSA
  - https://www.hdfgroup.org/solutions/hdf5/
  - format and software for storing, transfer, and distribution of datasets
  - can store images, multidimensional arrays, tables, etc.
  - emphasis on storage and I/O efficiency
  - free and widely used in engineering and sciences

A little history

- HDF4 – based on original 1988 version of HDF
  - backward compatible with all earlier versions
  - 6 basic objects: raster image, multidimensional arrays, palette, group, table, annotation
  - limits on file size (<2GB) and number of objects (<20K)

- HDF5 – First released in 1998
  - new formats and library are NOT compatible with HDF4
  - includes only 2 basic primitive objects (**groups** and **datasets**)
    - No limit on the HDF files size or number of objects in the file
    - HDF5 file is portable across **all** computing platforms

HDF file



Every HDF5 file starts with a root group

HDF5 data model

- Dataset
  -- multidimensional array of elements together with supporting metadata

- Group
  -- directory-like structure containing datasets, groups.

HDF5 dataset

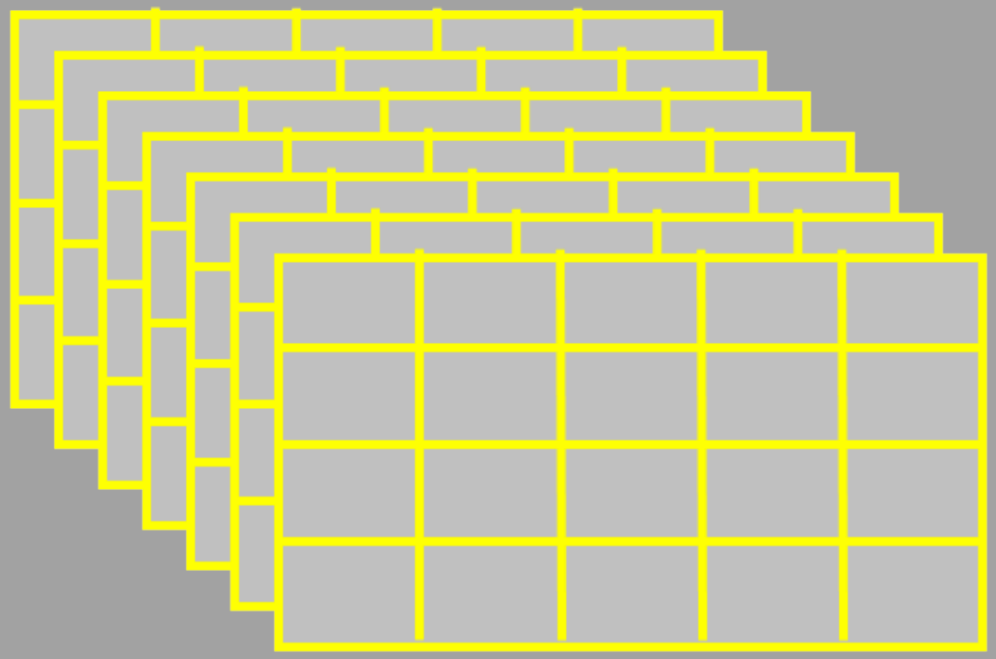
HDF5 Datatype

Integer 32bit LE

HDF5 Dataspace

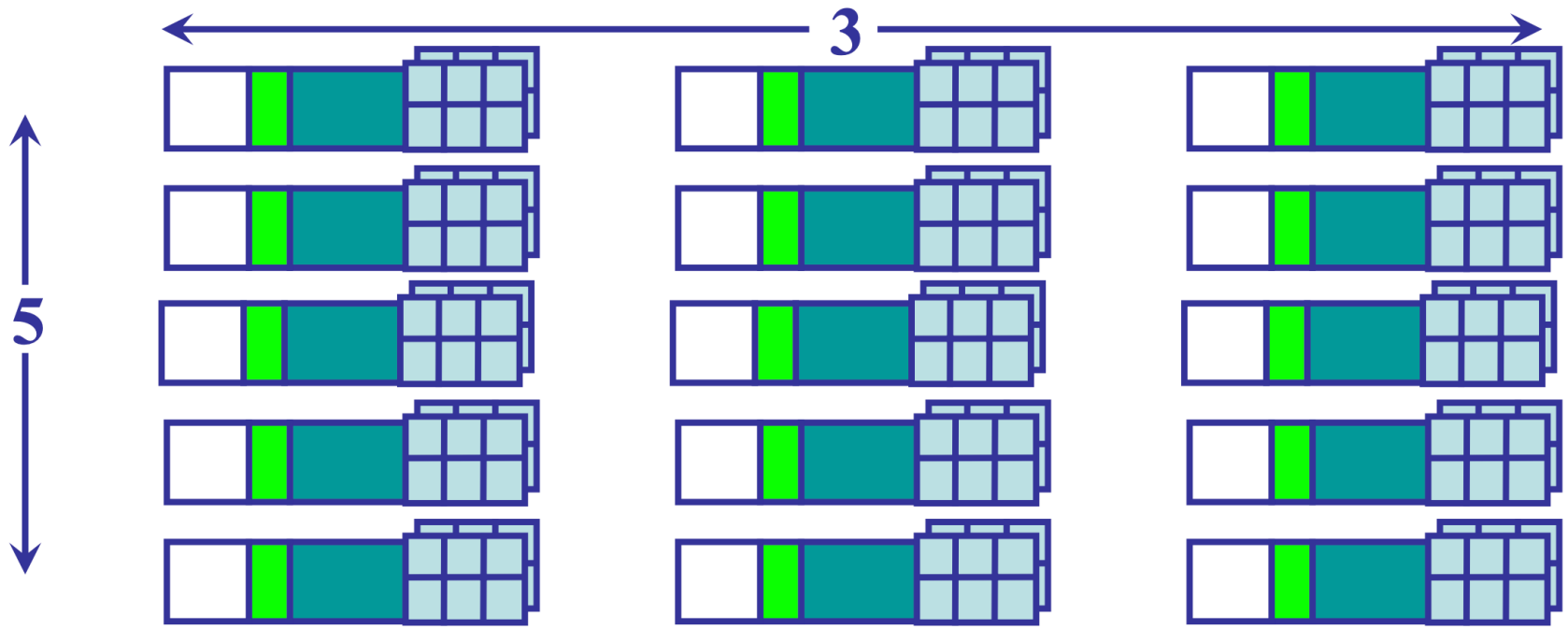
Rank	Dimensions
3	Dim_0 = 4
	Dim_1 = 5
	Dim_2 = 7

Specifications for single data element and array dimensions



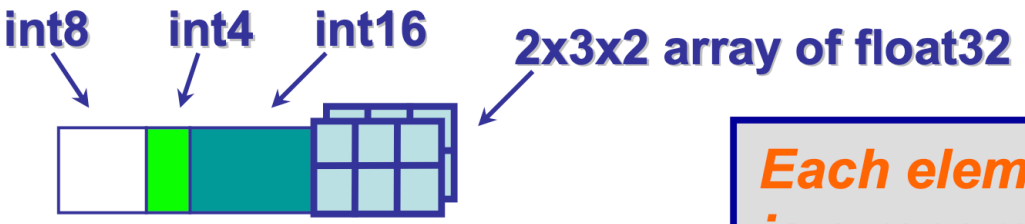
Multi-dimensional array of identically typed data elements

HDF5 dataset with compound datatype



Dimensionality: 5 x 3

Datatype:



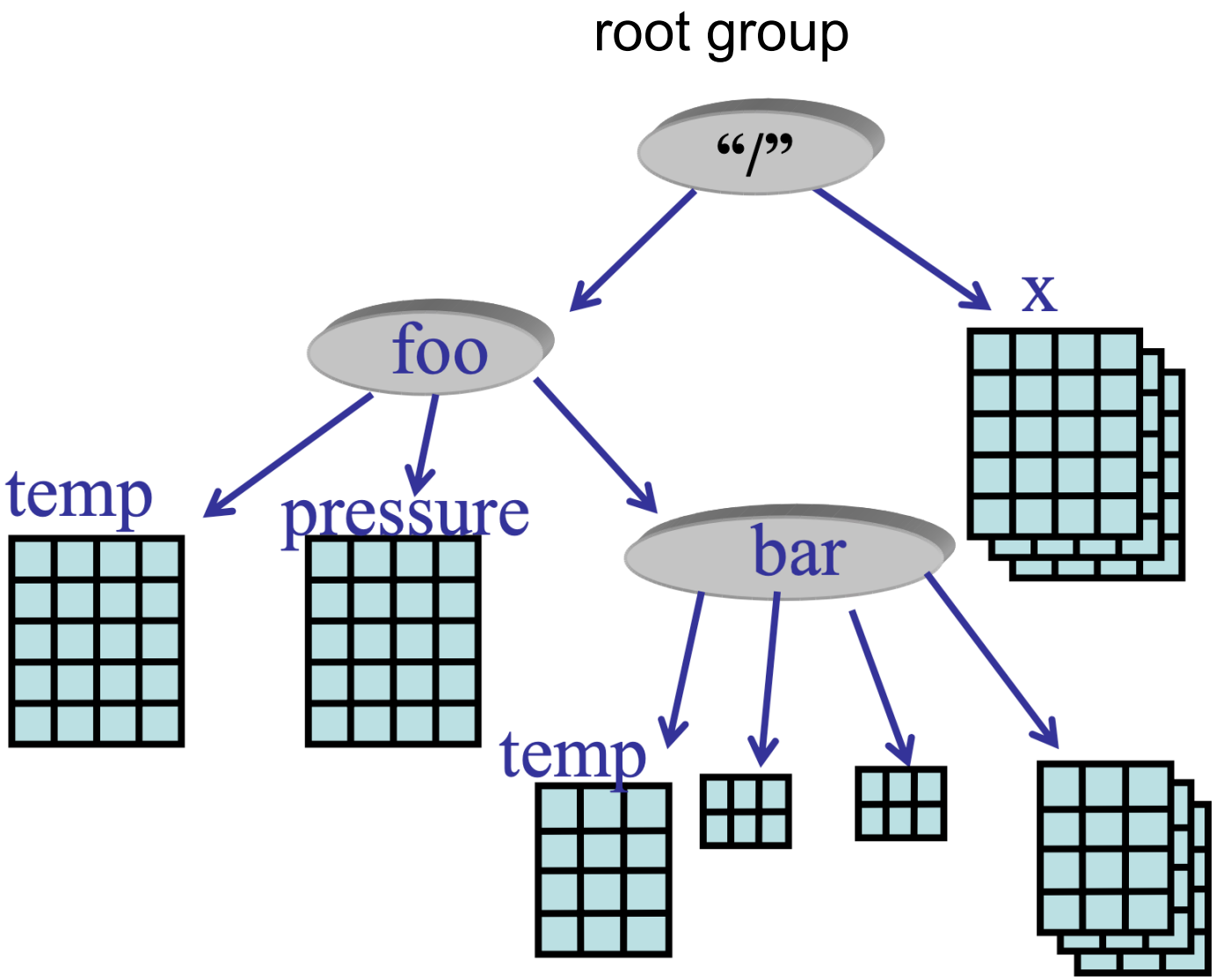
Record

Each element in 3D array is a record with 4 values

HDF5 dataset are located by their pathname/Group

similar to UNIX
directory structure

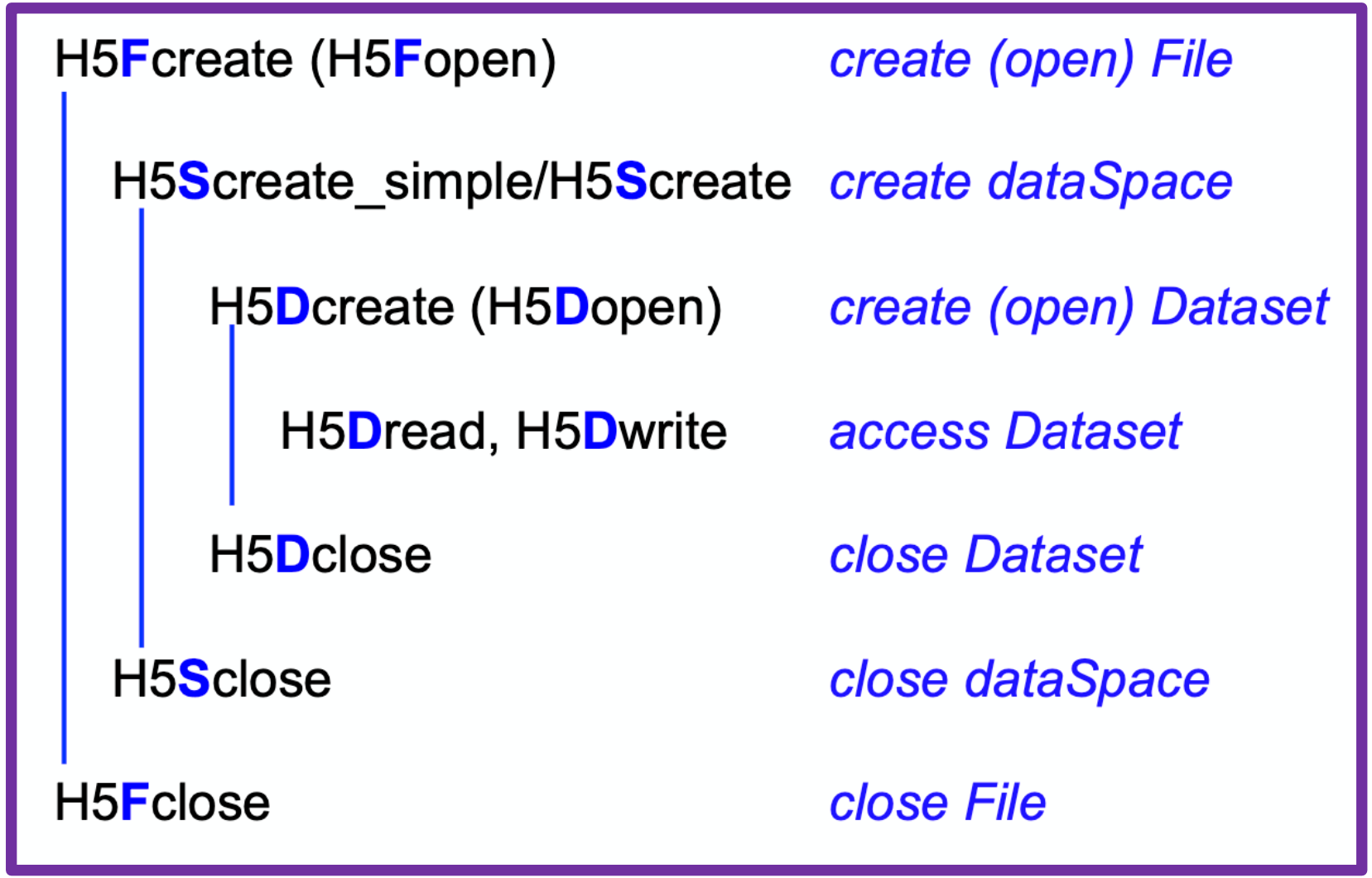
/ (root)
/x
/foo
/foo/temp
/foo/pressure
/foo/bar/temp



General programming paradigm

- Object (file, group, data) is opened or created
- Object is accessed
- Object is closed

Example functions
H5D : Dataset interface,
e.g. H5Dread
H5F : File interface,
e.g. H5Fopen
H5S : dataspace interface,
e.g. H5Sclose



HDF5 compilations

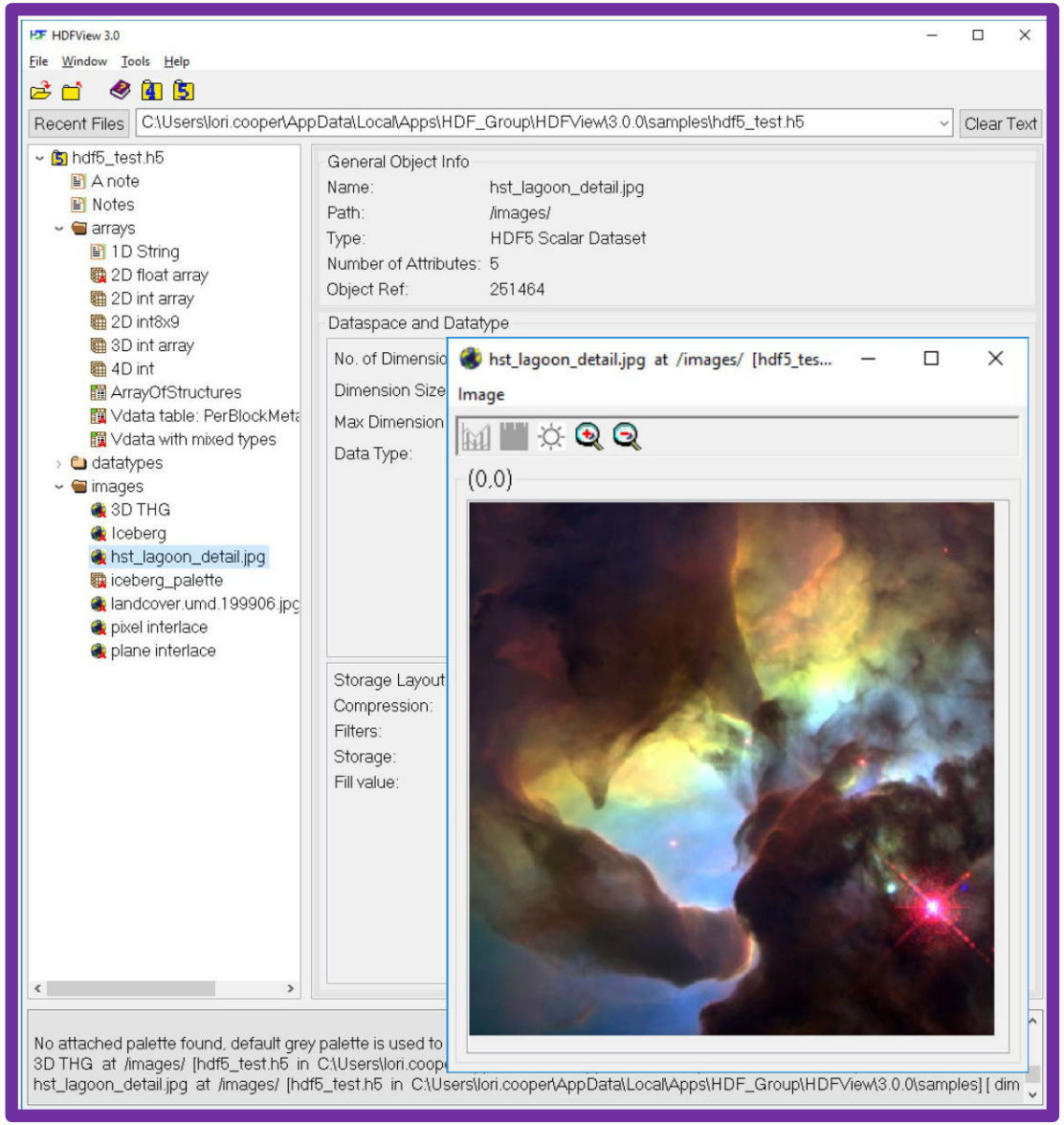
- Once you start using the HDF API, you will have to link your code against the library.
- You need to include the header file hdf5.h in your code.
- Example:

```
icc -I/Users/lib/hdf5/hdf5-1.6.5/include -
L/Users/lib/hdf5/hdf5-1.6.5/lib main.c -lhdf5
```

Some useful binaries

- These binaries are included in your hdf5 installation
  - ➢ h5ls: list contents of HDF5 file
  - ➢ h5dump: higher level view of the HDF5 file
  - ➢ h5diff: show the difference between two HDF5 files

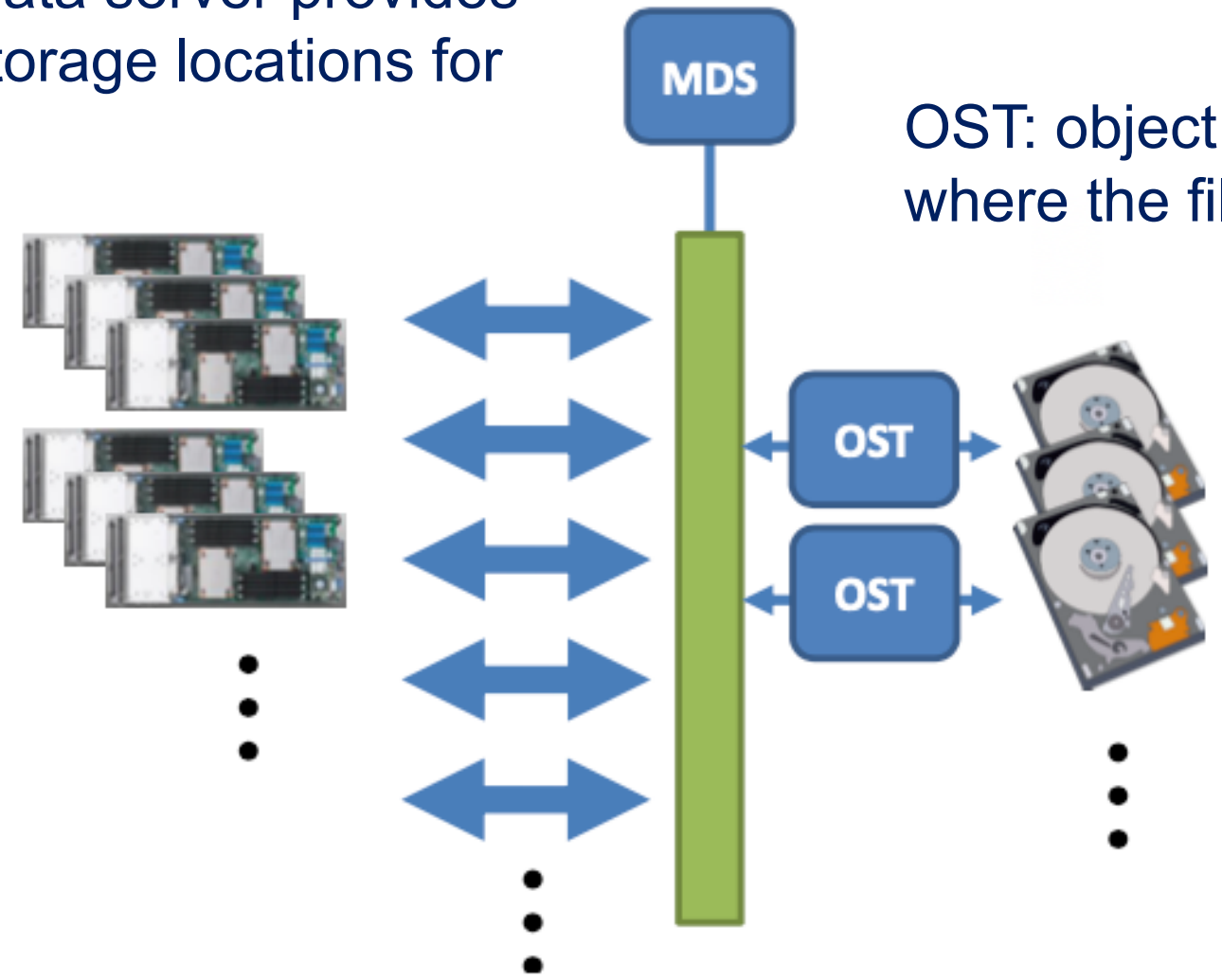
- There is also a HDFView software that provides a GUI interface for file inspection.



Additional I/O topics

File system

MDS: medadata server provides associated storage locations for your file



OST: object storage target is where the files are stored

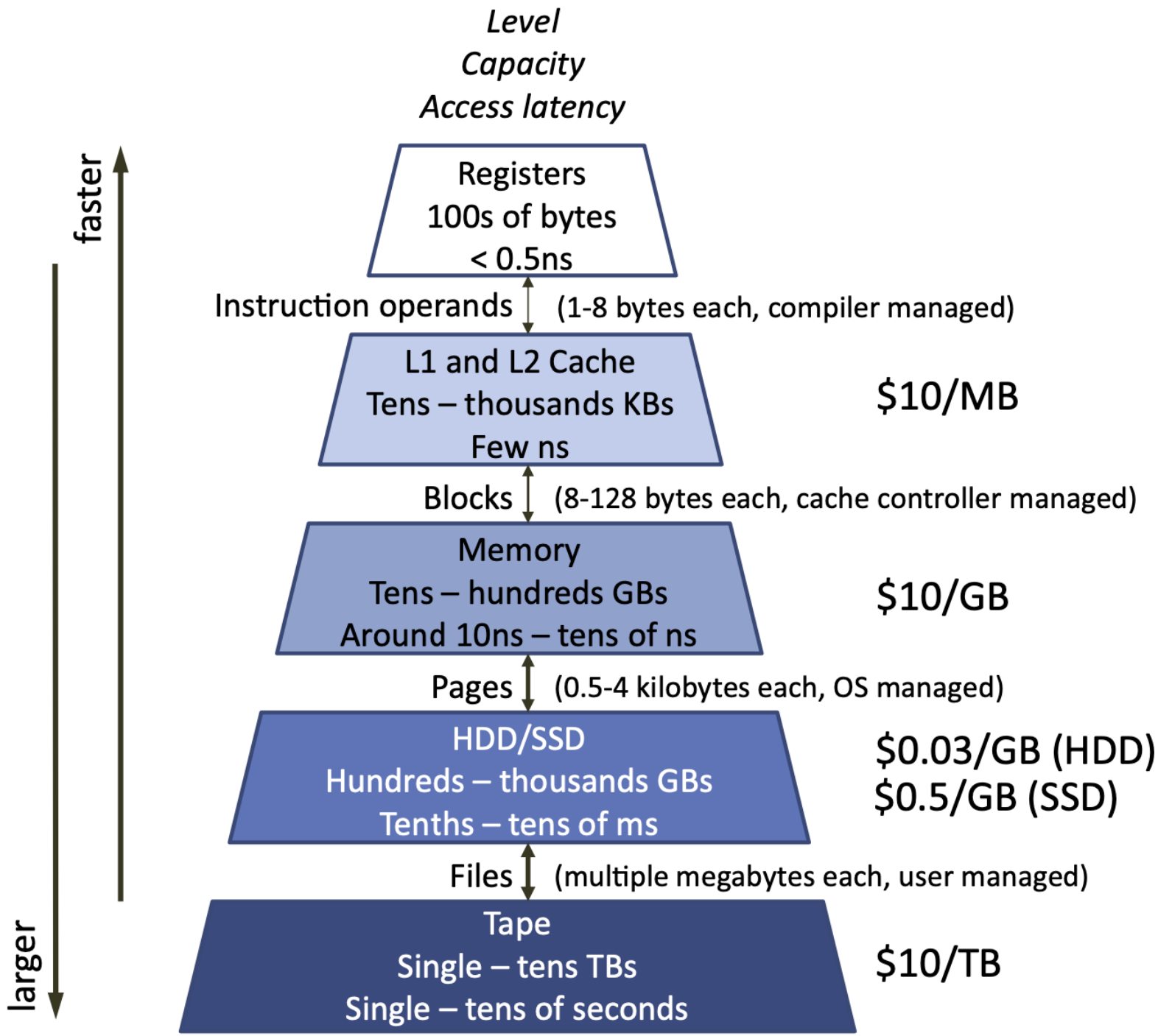
a few hundred disks

hundreds of thousands of processors

File system

User view of file – contiguous storage

File as actually stored across multiple disks


Level
Capacity
Access latency
faster
larger
Registers
100s of bytes
< 0.5ns
Instruction operands
(1-8 bytes each, compiler managed)
L1 and L2 Cache
Tens – thousands KBs
Few ns
$10/MB
Blocks
(8-128 bytes each, cache controller managed)
Memory
Tens – hundreds GBs
Around 10ns – tens of ns
$10/GB
Pages
(0.5-4 kilobytes each, OS managed)
HDD/SSD
Hundreds – thousands GBs
Tenths – tens of ms
$0.03/GB (HDD)
$0.5/GB (SSD)
Files
(multiple megabytes each, user managed)
Tape
Single – tens TBs
Single – tens of seconds
$10/TB

Bad I/O strategies

- Open and close the same file every few milliseconds
  - stresses the MDS
- Too often and too many
  - stresses the MDS and OSTs
- Write large files to $WORK or $DATA
  - $SCRATCH has more OSTs
- `ls' in a crowded directory
  - ls stresses MDS
- Create thousands of files in the same directory
  - a directory is a file managed by the MDS

Good I/O strategies

- Write large files to $SCRATCH
- Write one global file instead of multiple task files
- Use HDF5 or netCDF
- Use parallel I/O
  - Parallel HDF5 i.e. PHDF5
  - MPI I/O