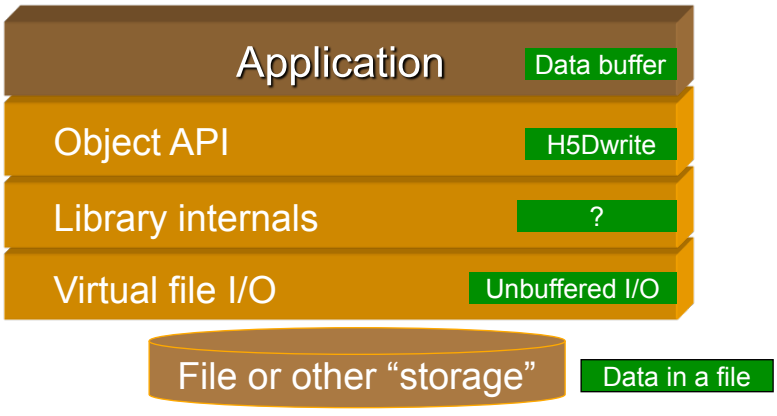
Caching and Buffering in HDF5

Software stack

- Life cycle: What happens to data when it is transferred from application buffer to HDF5 file and from HDF5 file to application buffer?



Goals

- Understanding of what is happening to data inside the HDF5 library will help to write efficient applications
- Goals of this talk:
  - Describe some basic operations and data structures, and explain how they affect performance and storage sizes
  - Give some “recipes” for how to improve performance

Topics

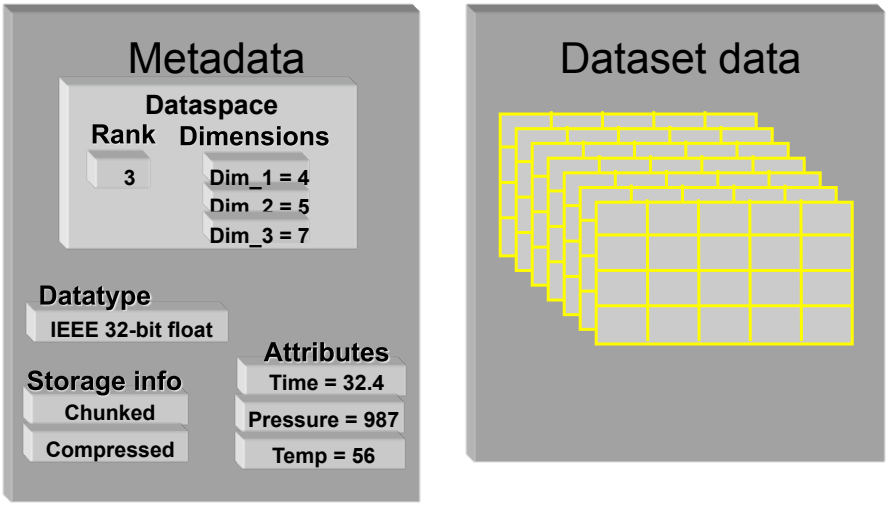
- Dataset metadata and array data storage layouts
- Types of dataset storage layouts
- Factors affecting I/O performance
  - I/O with compact datasets
  - I/O with contiguous datasets
  - I/O with chunked datasets
  - Variable length data and I/O

HDF5 dataset metadata and array data storage layouts

HDF5 Dataset

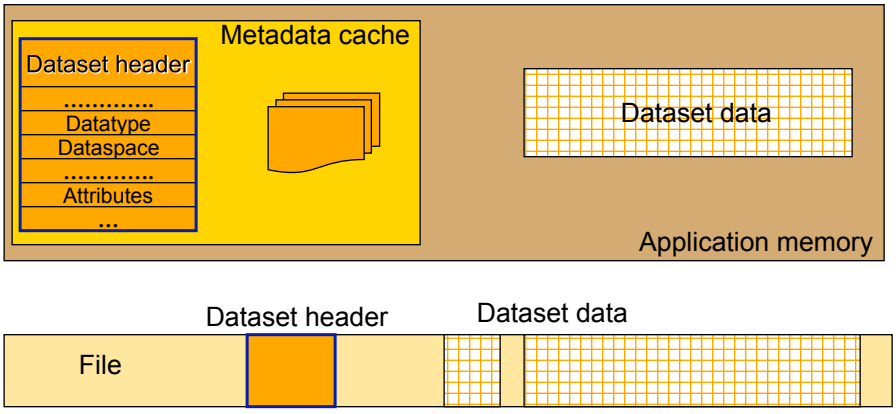
- Data array
  - Ordered collection of identically typed data items distinguished by their indices
- Metadata
  - Dataspace: Rank, dimensions of dataset array
  - Datatype: Information on how to interpret data
  - Storage Properties: How array is organized on disk
  - Attributes: User-defined metadata (optional)

HDF5 Dataset



Metadata cache and dataset data

- Dataset data typically kept in application memory
- Dataset header in separate space – metadata cache



Metadata and metadata cache

- HDF5 metadata
  - Information about HDF5 objects used by the library
  - Examples: object headers, B-tree nodes for group, B-Tree nodes for chunks, heaps, super-block, etc.
  - Usually small compared to raw data sizes (KB vs. MB-GB)

Metadata and metadata cache

- Metadata cache
  - Space allocated to handle pieces of the HDF5 metadata
  - Allocated by the HDF5 library in application's memory space
  - Cache behavior affects overall performance
  - Metadata cache implementation prior to HDF5 1.6.5 could cause performance degradation for some applications

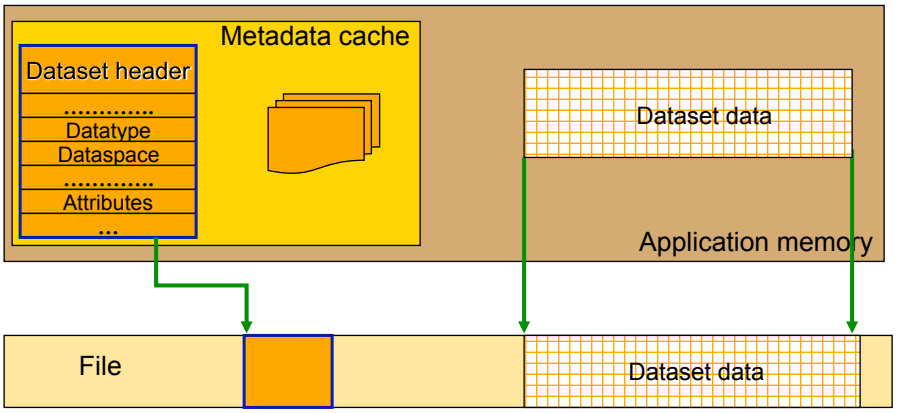
Types of data storage layouts

HDF5 datasets storage layouts

- Contiguous
- Chunked
- Compact

Contiguous storage layout

- Metadata header separate from dataset data
- Data stored in one contiguous block in HDF5 file

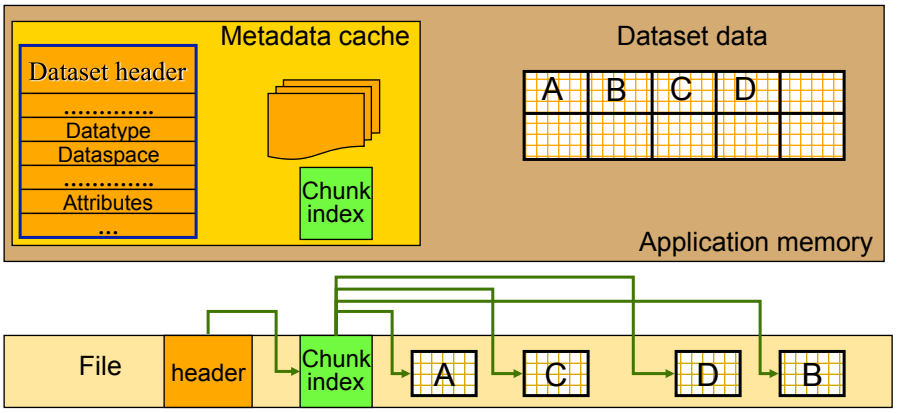


Chunked storage

- Chunking – storage layout where a dataset is partitioned in fixed-size multi-dimensional tiles or chunks
- Used for extendible datasets and datasets with filters applied (checksum, compression)
- HDF5 library treats each chunk as atomic object
- Greatly affects performance and file sizes

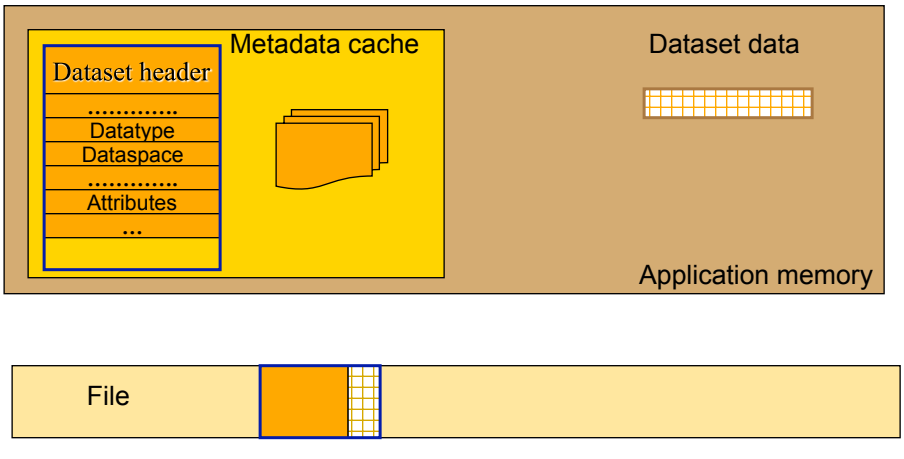
Chunked storage layout

- Dataset data divided into equal sized blocks (chunks)
- Each chunk stored separately as a contiguous block in HDF5 file



Compact storage layout

- Dataset data and metadata stored together in the object header



Factors affecting I/O performance

What goes on inside the library?

- Operations on data inside the library
  - Copying to/from internal buffers
  - Datatype conversion
  - Scattering - gathering
  - Data transformation (filters, compression)
- Data structures used
  - B-trees (groups, dataset chunks)
  - Hash tables
  - Local and Global heaps (variable length data: link names, strings, etc.)
- Other concepts
  - HDF5 metadata, metadata cache
  - Chunking, chunk cache

Operations on data inside the library

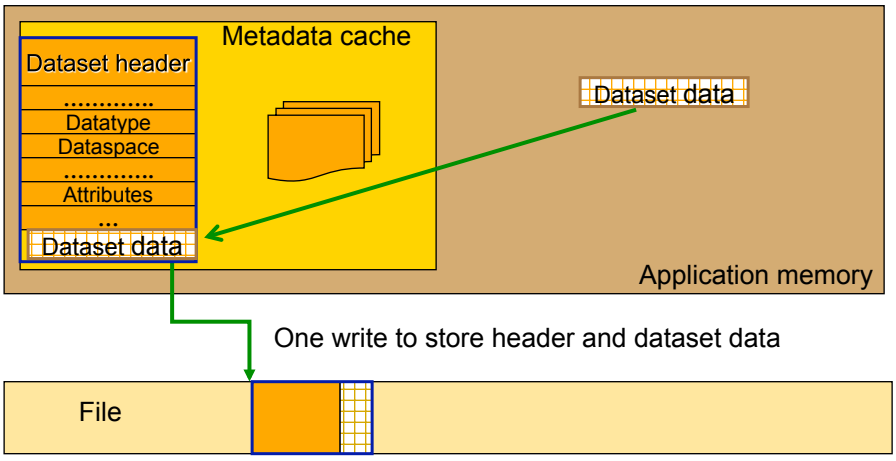
- Copying to/from internal buffers
- Datatype conversion, such as
  - float → integer
  - Little-endian → big-endian
  - 64-bit integer to 16-bit integer
- Scattering - gathering
  - Data is scattered/gathered from/to application buffers into internal buffers for datatype conversion and partial I/O
- Data transformation (filters, compression)
  - Checksum on raw data and metadata (in 1.8.0)
  - Algebraic transform
  - GZIP and SZIP compressions
  - User-defined filters

I/O performance

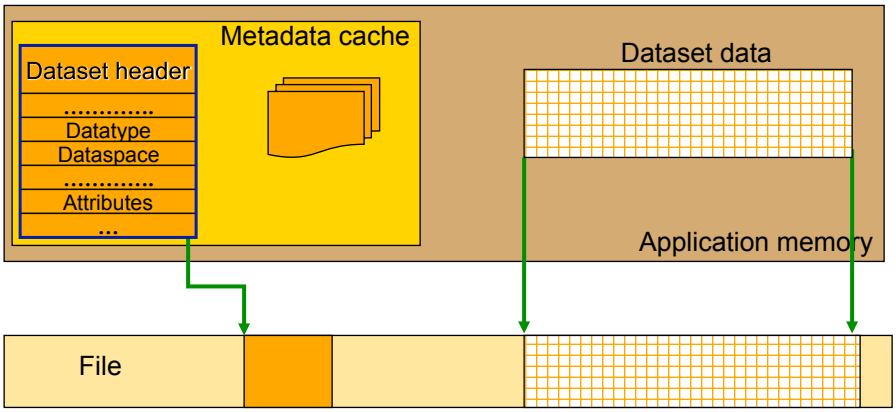
- I/O performance depends on
  - Storage layouts
  - Dataset storage properties
  - Chunking strategy
  - Metadata cache performance
  - Datatype conversion performance
  - Other filters, such as compression
  - Access patterns

I/O with different storage layouts

Writing a compact dataset

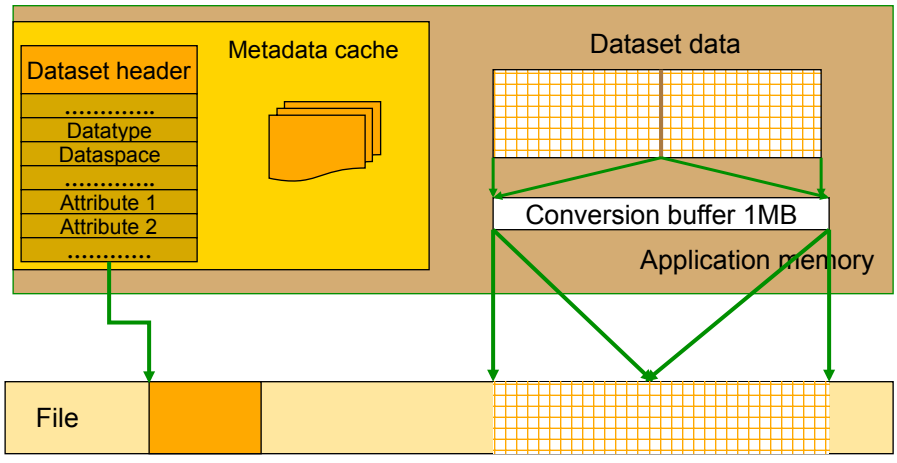


Writing contiguous dataset – no conversion



No sub-setting in memory or a file is performed

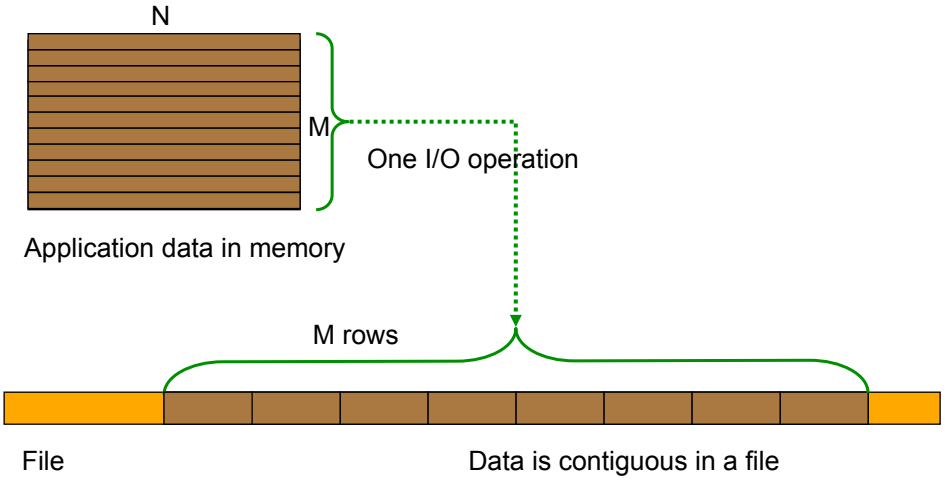
Writing a contiguous dataset with datatype conversion



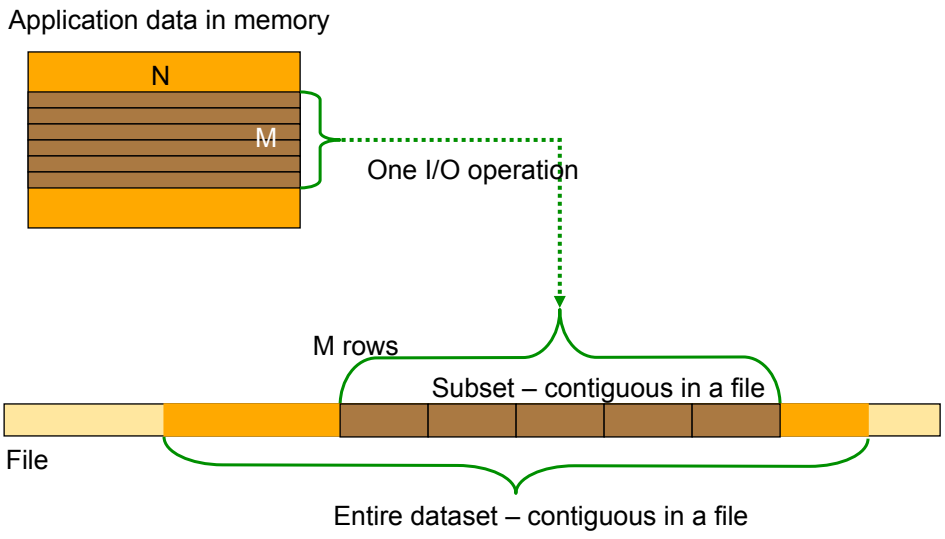
No sub-setting in memory or a file is performed

Partial I/O with contiguous datasets

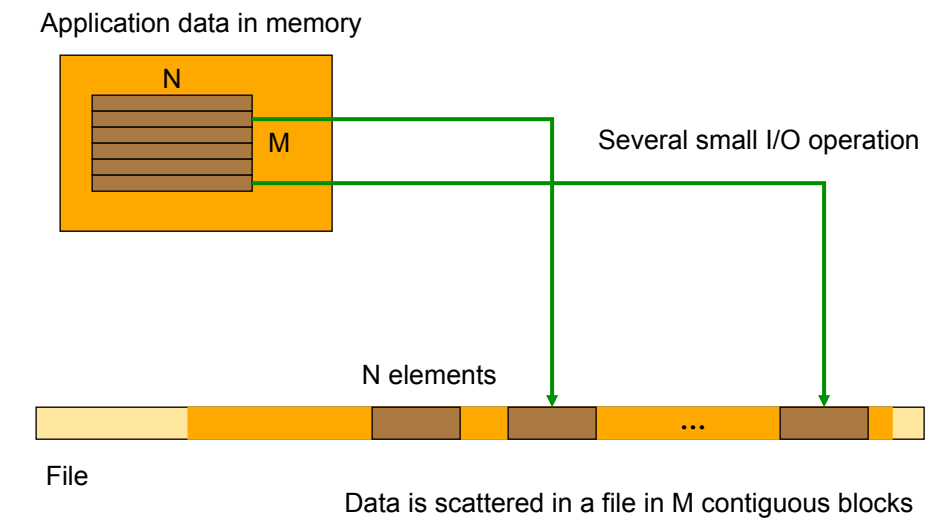
Writing whole dataset – contiguous rows



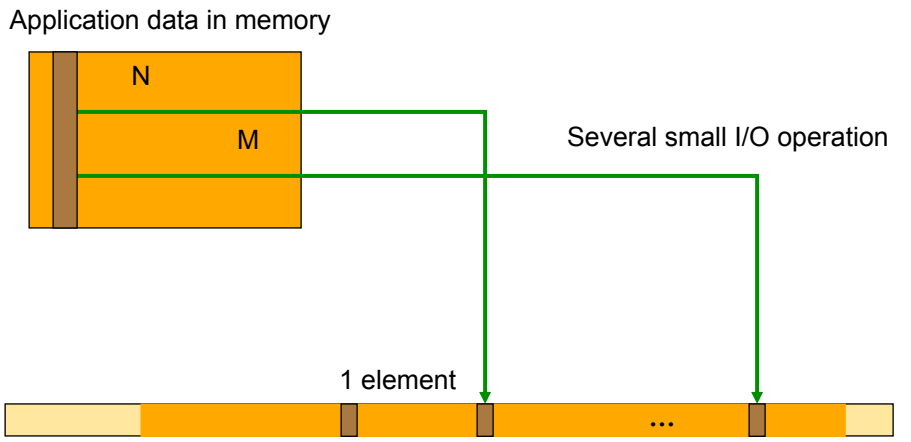
Sub-setting of contiguous dataset
Series of adjacent rows



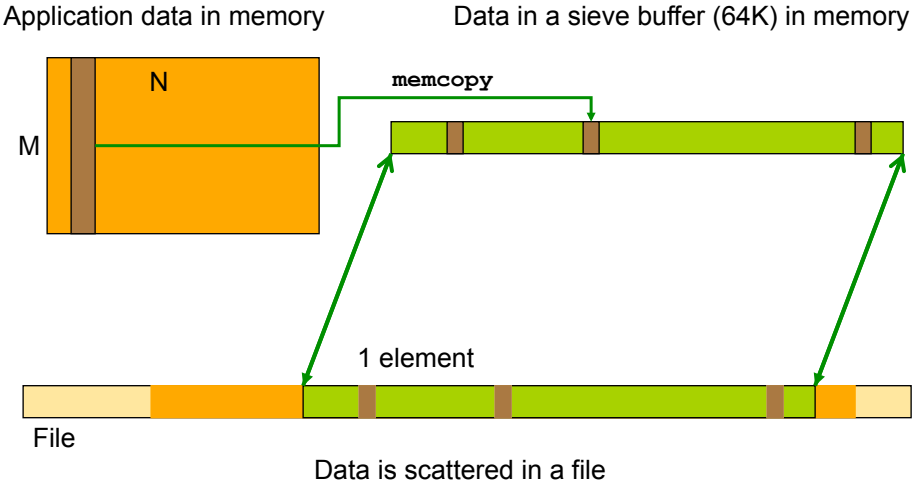
Sub-setting of contiguous dataset
Adjacent, partial rows



Sub-setting of contiguous dataset
Extreme case: writing a column



Sub-setting of contiguous dataset
Data sieve buffer



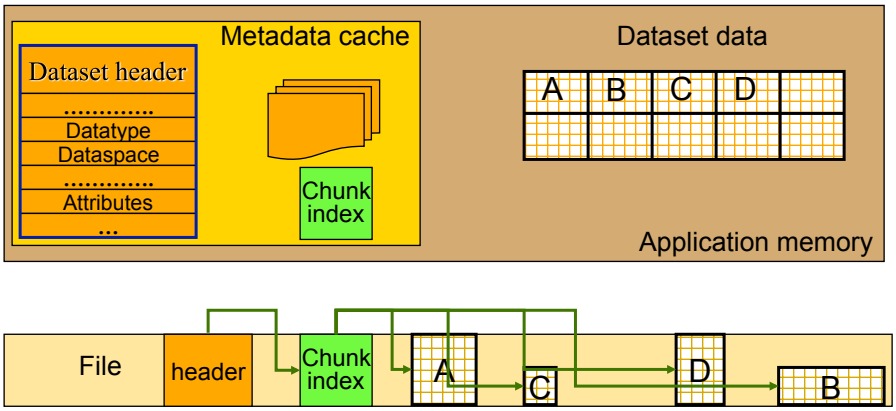
Performance tuning for contiguous dataset

- Datatype conversion
  - Avoid for better performance
  - Use **`H5Pset_buffer`** function to customize conversion buffer size
- Partial I/O
  - Write/read in big contiguous blocks
  - Use **`H5Pset_sieve_buf_size`** to improve performance for complex subsetting

I/O with Chunking

Chunked storage layout

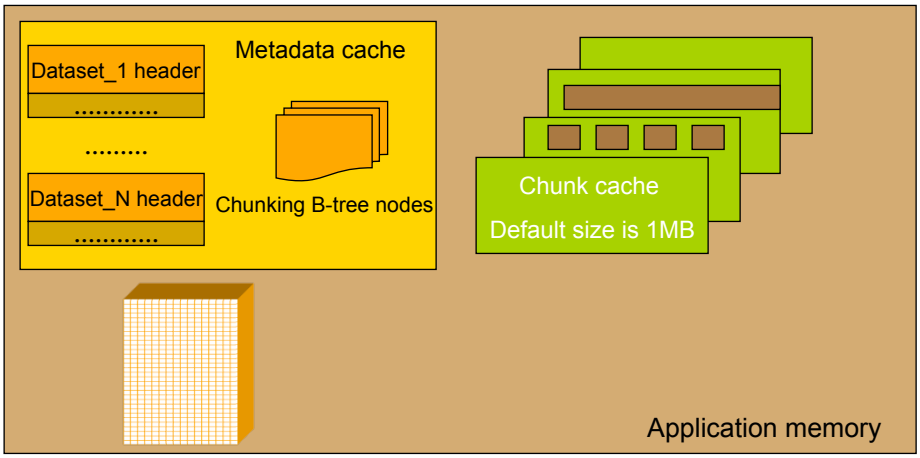
- Raw data divided into equal sized blocks (chunks)
- Each chunk stored separately as a contiguous block in a file



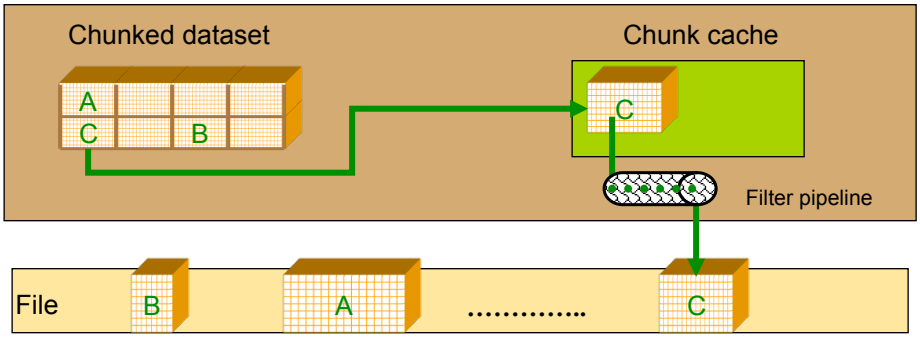
Information about chunking

- HDF5 library treats each chunk as atomic object
  - Compression and other filters are applied to each chunk
  - Datatype conversion is performed on each chunk
- Chunk size greatly affects performance
  - Chunk overhead adds to file size
  - Chunk processing involves many steps
- Chunk cache
  - Caches chunks for better performance
  - Size of chunk cache is set for file (default size 1MB)
  - Each chunked dataset has its own chunk cache
  - Chunk may be too big to fit into cache
  - Memory may grow if application keeps opening datasets

Chunk cache



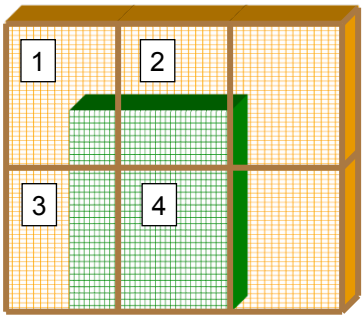
Writing chunked dataset



Filters including compression are applied when chunk is evicted from cache

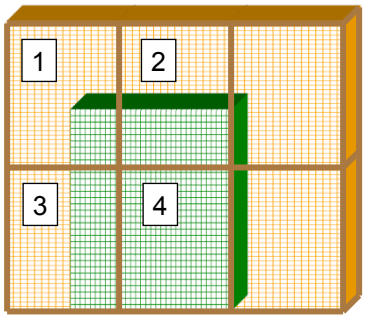
Partial I/O with Chunking

Partial I/O for chunked dataset



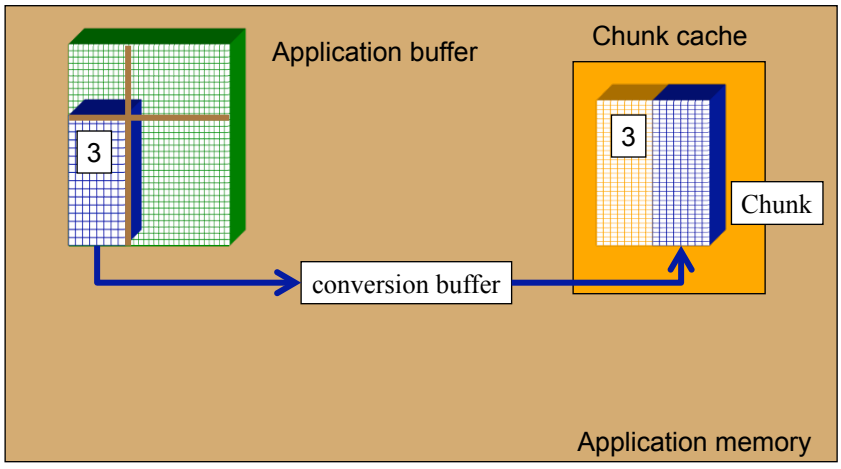
- ✓ Example: write the green subset from the dataset , converting the data
- ✓ Dataset is stored as six chunks in the file.
- ✓ The subset spans four chunks, numbered 1-4 in the figure.
- ✓ Hence four chunks must be written to the file.
- ✓ But first, the four chunks must be read from the file, to preserve those parts of each chunk that are not to be overwritten.

Partial I/O for chunked dataset



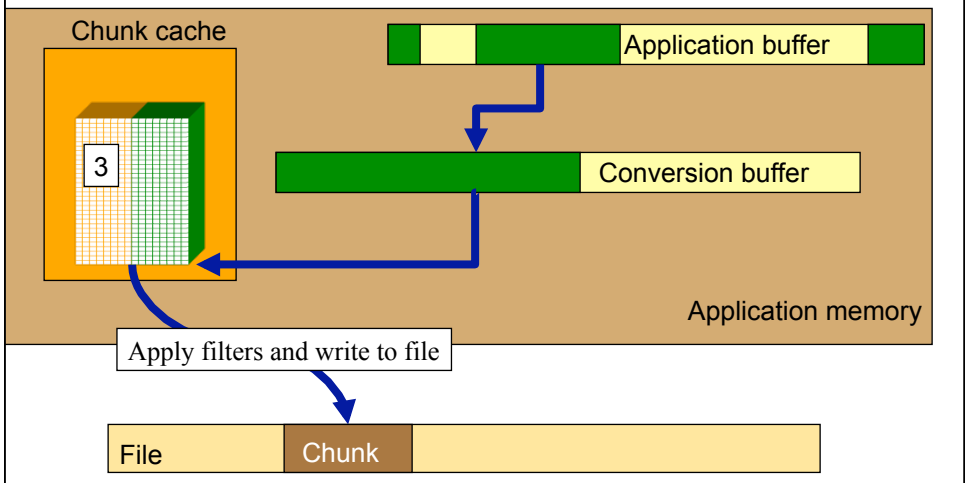
- For each of four chunks on writing:
  - Read chunk from file into chunk cache, unless it's already there
  - Determine which part of the chunk will be replaced by the selection
  - Move those elements from application buffer to conversion buffer
  - Perform conversion
  - Replace that part of the chunk in the cache with the corresponding elements from the conversion buffer
  - Apply filters (compression) when chunk is flushed from chunk cache
- *For each element 3 (or more) memcopy operations are performed*

Partial I/O for chunked dataset



Elements participating in I/O are gathered into corresponding chunk after going through conversion buffer

Partial I/O for chunked dataset



The HDF Group

Variable length data and I/O

Examples of variable length data

- String

  A[0] "*the first string we want to write*"

  ……

  A[N-1] *"the N-th string we want to write"*

- Each element is a record of variable-length

  A[0] (1,1,0,0,0,5,6,7,8,9) [length = 10]

  A[1] (0,0,110,2005) [length = 4]

  ……..

  A[N] (1,2,3,4,5,6,7,8,9,10,11,12,….,M) [length = M]

Variable length data in HDF5

- Variable length description in HDF5 application

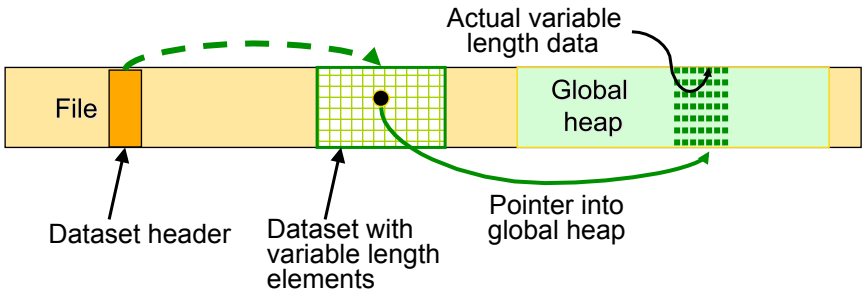
```
typedef struct {
        size_t length;
        void   *p;
}hvl_t;
```

  - Base type can be any HDF5 type

    `H5Tvlen_create(base_type)`

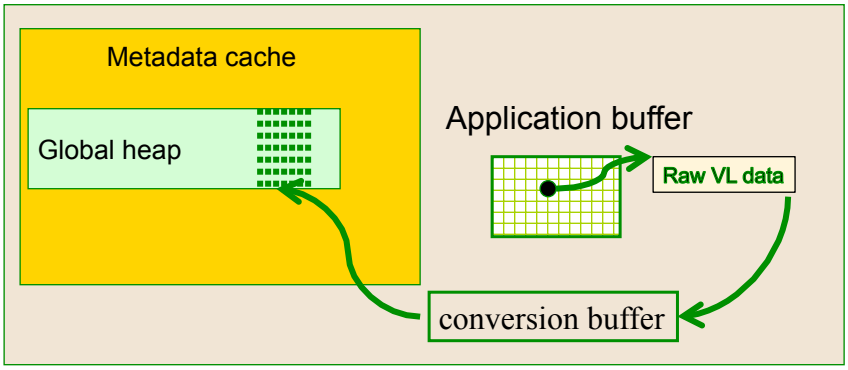
  - ~ 20 bytes overhead for each element
  - Data cannot be compressed

Variable length data storage in HDF5

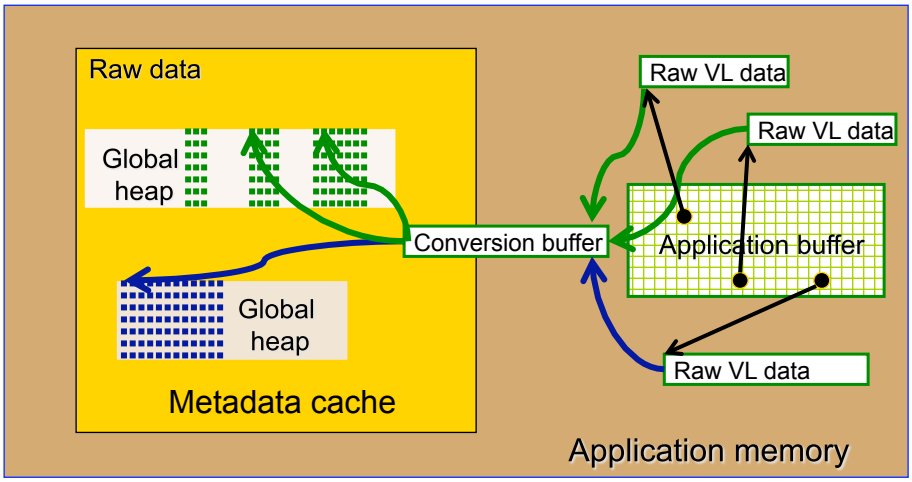


Variable length datasets and I/O

- When writing variable length data, elements in application buffer *always go through conversion* and are copied to the global heaps in a metadata cache before ending in a file

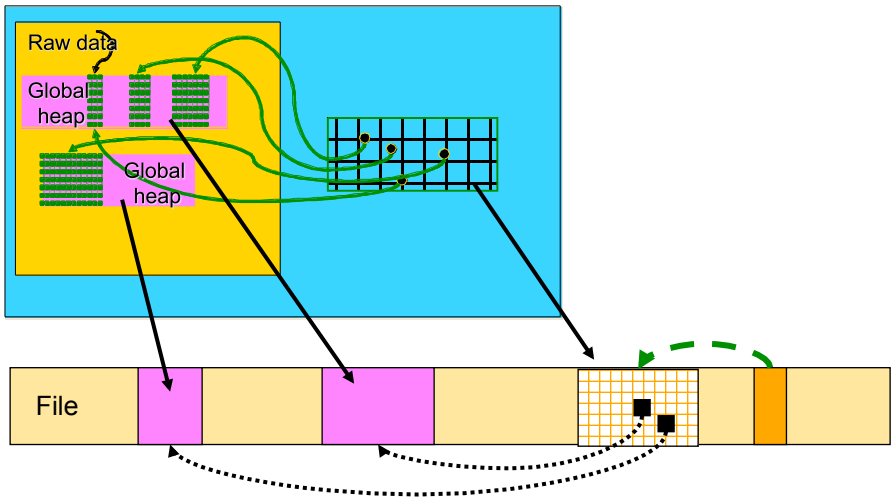


There may be more than one global heap

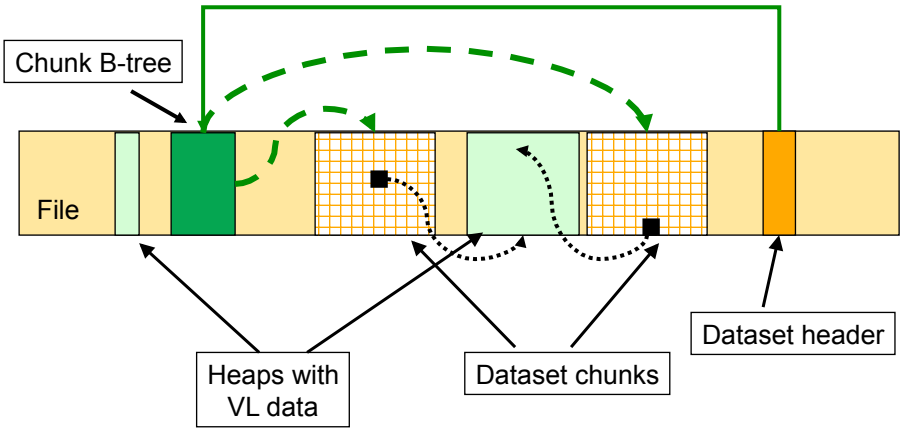


On a write request, VL data goes through conversion and is written to a global heap; elements of the same dataset may be written to different heaps.

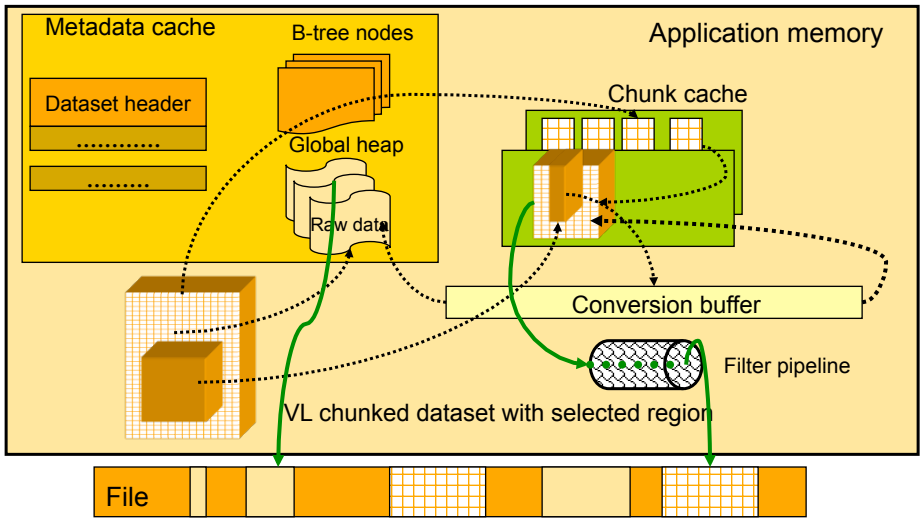
Variable length datasets and I/O



VL *chunked* dataset in a file



Writing chunked VL datasets



Hints for variable length data I/O

- Avoid closing/opening a file while writing VL datasets
  - Global heap information is lost
  - Global heaps may have unused space
- Avoid alternately writing different VL datasets
  - Data from different datasets will go into to the same heap
- If maximum length of the record is known, consider using fixed-length records and compression

Questions?