HDF5 Chunking and Compression

Performance issues

The HDF Group

Goal

- To help you with understanding how HDF5 chunking and compression works, so you can efficiently store and retrieve data from HDF5 files

Case study

- SCRIS_npp_d20140522_t0754579_e0802557_b13293_c20140522142425734814_noaa_pop.h5

```
DATASET "/All_Data/CrIS-SDR_All/ES_ImaginaryLW" {
   DATATYPE  H5T_IEEE_F32BE
   DATASPACE  SIMPLE { ( 60, 30, 9, 717 ) / ( H5S_UNLIMITED,
H5S_UNLIMITED, H5S_UNLIMITED, H5S_UNLIMITED ) }
   STORAGE_LAYOUT {
      CHUNKED ( 4, 30, 9, 717 )
      SIZE 46461600
   }
```

- Dataset is read once, by contiguous 1x1x1x717 selections, i.e., 717 elements 16200 times. The time it takes to read the whole dataset is in the table below:

Compressed with GZIP level 6	**No compression**
~345 seconds	**~0.1 seconds**

Solutions

- Performance may depend on many factors such as I/O access patterns, chunk sizes, chunk layout, chunk cache, memory usage, compression, etc.
- Solutions discussed next are oriented for a particular use case and access patterns:
  - Reading entire dataset once by a contiguous selection along the fastest changing dimension(s) for a specified file.
- The troubleshooting approach should be applicable to a wider variety of files and access patterns.

Solution (Data Consumers)

- **Increase chunk cache size**
  - Tune application to use appropriate HDF5 chunk cache size for each dataset to read
  - For our example dataset, we increased chunk cache size to 3MB - big enough to hold one 2.95 MB chunk

Compressed with GZIP level 6 1MB cache (default)	**Compressed with GZIP level 6 3MB cache**
~345 seconds	**~0.37 seconds**

Solution (Data Consumers)

- **Change access pattern**
  - Keep default cache size (1MB)
  - Tune the application to use an appropriate HDF5 access pattern
  - We read our example dataset using a selection that *corresponds* to the *whole chunk* 4x9x30x717

Compressed with GZIP level 6 Selection 1x1x1x717	**Compressed with GZIP level 6 Selection 4x9x30x717**
~345 seconds	**~0.36 seconds**

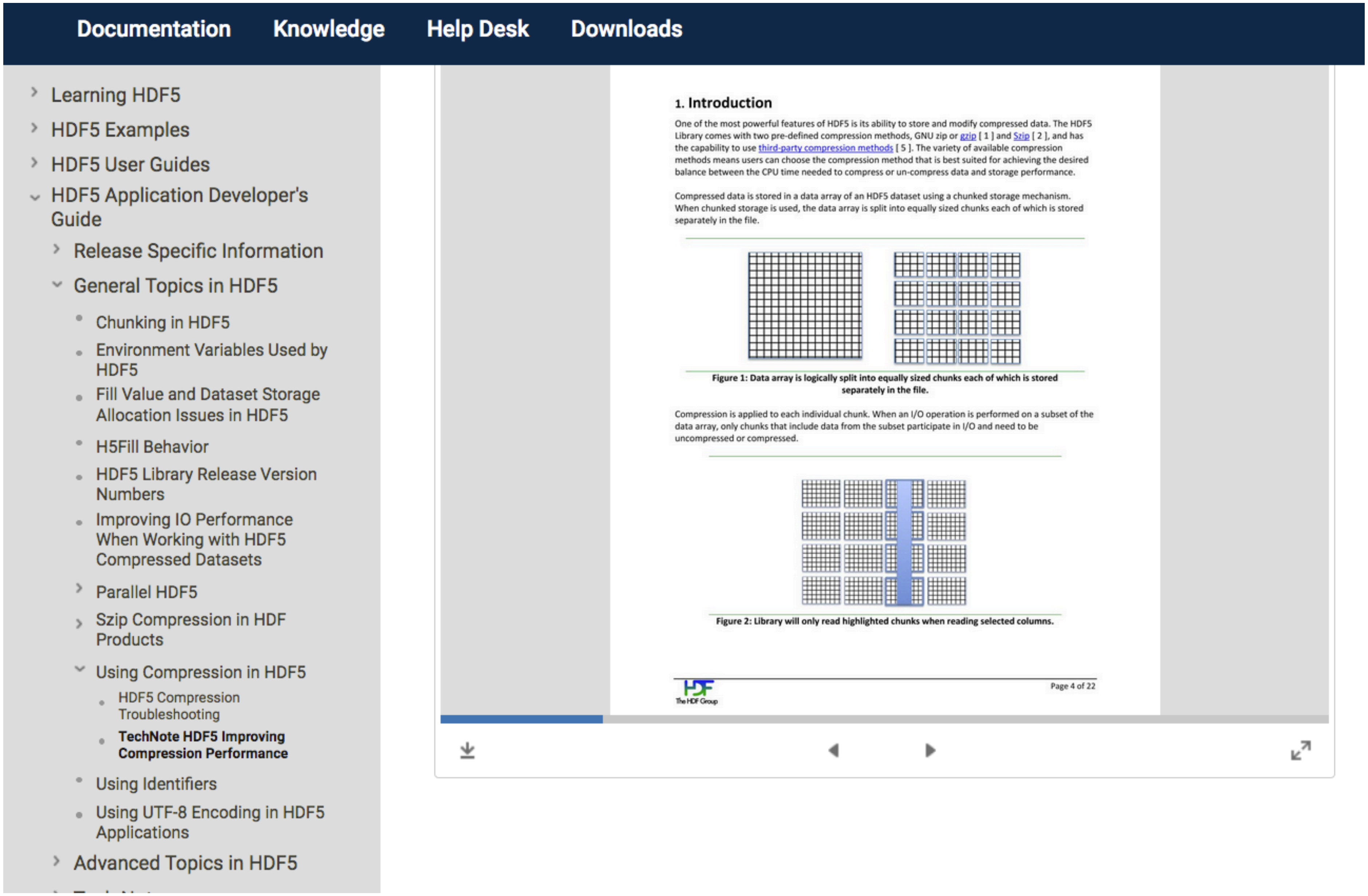
Solution (Data Providers)

- **Change chunk size**
  - Write original files with the smaller chunk size
  - We recreated our example dataset using chunk size 1x30x9x717 (~0.74MB)
  - We used default cache size 1MB
  - Read by 1x1x1x717 selections 16200 times
  - Performance improved 1000 times

Compressed with GZIP level 6 chunk size 4x9x30x717	**Compressed with GZIP level 6 chunk size 1x9x30x717**
~345 seconds	**~0.36 seconds**

HDF5 CHUNKING OVERVIEW

HDF5 Chunking and Compression documentation


Documentation
Knowledge
Help Desk
Downloads
Learning HDF5
HDF5 Examples
HDF5 User Guides
HDF5 Application Developer's Guide
Release Specific Information
General Topics in HDF5
Chunking in HDF5
Environment Variables Used by HDF5
Fill Value and Dataset Storage Allocation Issues in HDF5
H5Fill Behavior
HDF5 Library Release Version Numbers
Improving IO Performance When Working with HDF5 Compressed Datasets
Parallel HDF5
Szip Compression in HDF Products
Using Compression in HDF5
HDF5 Compression Troubleshooting
TechNote HDF5 Improving Compression Performance
Using Identifiers
Using UTF-8 Encoding in HDF5 Applications
Advanced Topics in HDF5

1. Introduction

One of the most powerful features of HDF5 is its ability to store and modify compressed data. The HDF5 Library comes with two pre-defined compression methods, GNU zip or gzip [1] and Szip [2], and has the capability to use third-party compression methods [5]. The variety of available compression methods means users can choose the compression method that is best suited for achieving the desired balance between the CPU time needed to compress or un-compress data and storage performance.

Compressed data is stored in a data array of an HDF5 dataset using a chunked storage mechanism. When chunked storage is used, the data array is split into equally sized chunks each of which is stored separately in the file.

Figure 1: Data array is logically split into equally sized chunks each of which is stored separately in the file.

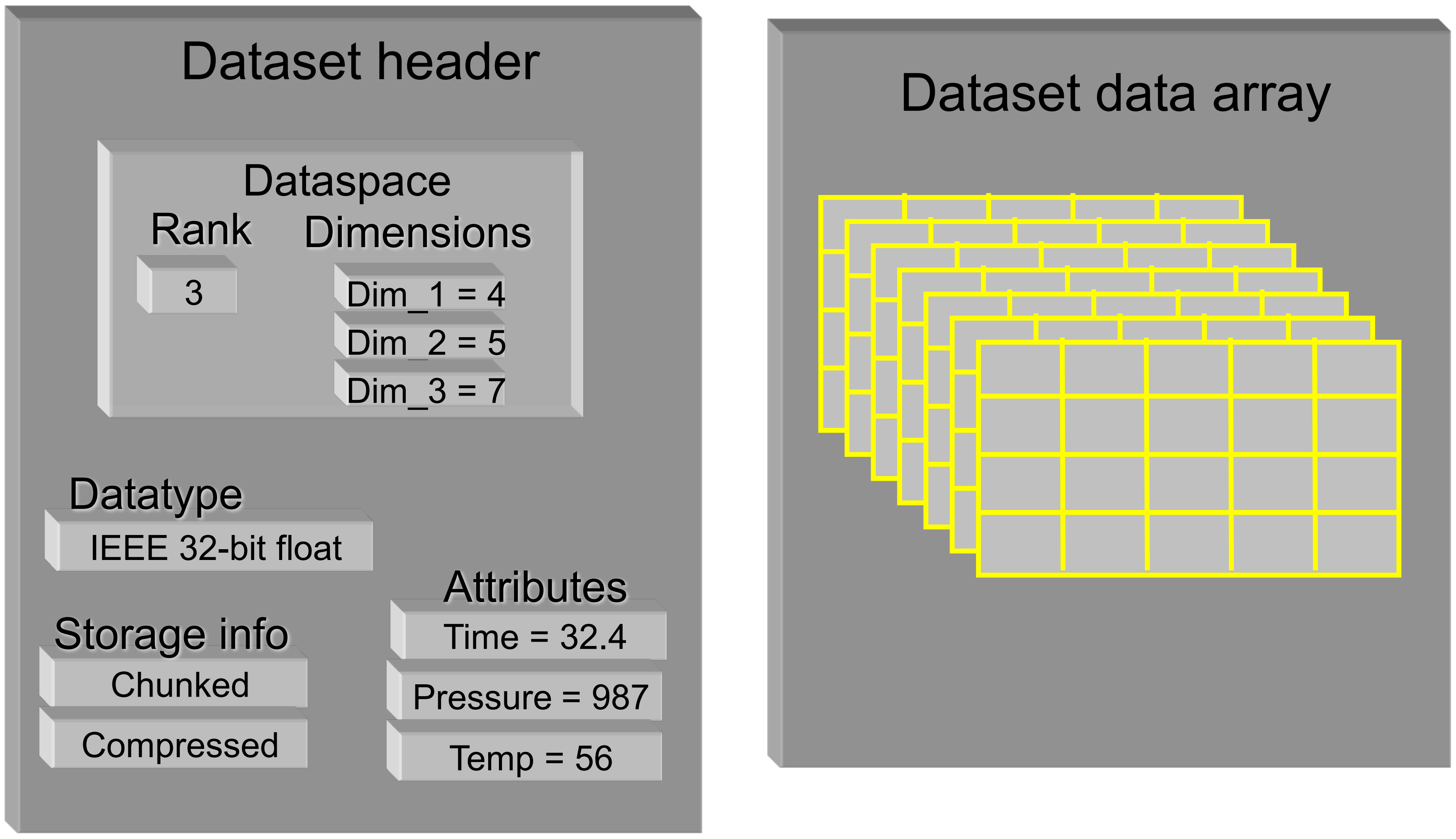
Compression is applied to each individual chunk. When an I/O operation is performed on a subset of the data array, only chunks that include data from the subset participate in I/O and need to be uncompressed or compressed.

Figure 2: Library will only read highlighted chunks when reading selected columns.

The HDF Group

Page 4 of 22

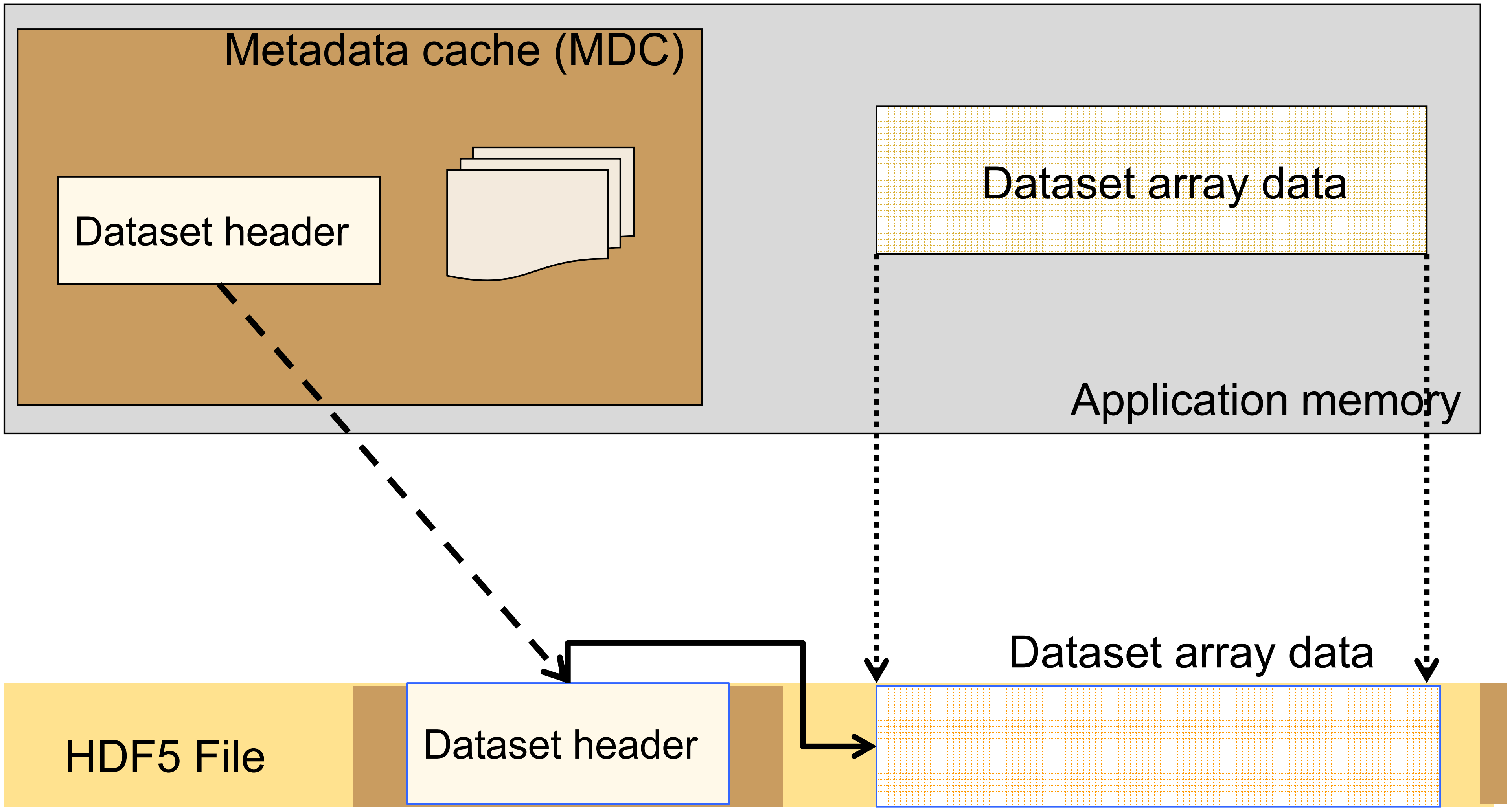
HDF5 Dataset Components



Metadata

Raw data

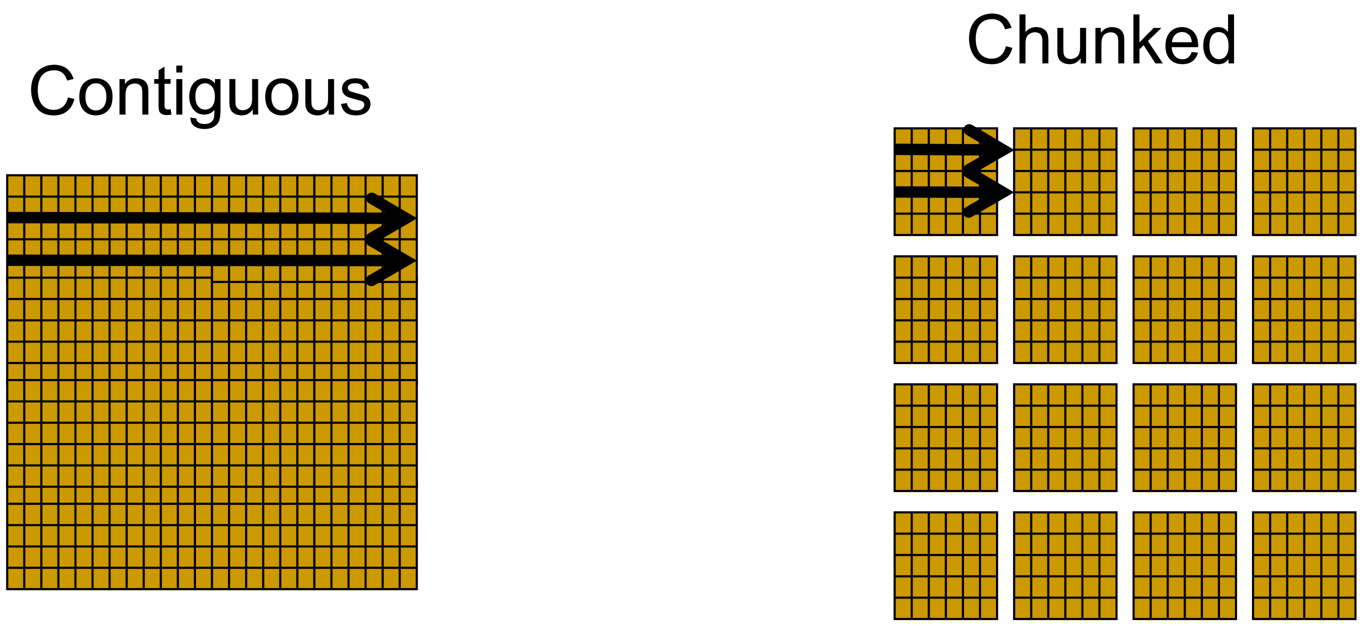
Contiguous storage layout



Raw data is stored in one contiguous block in HDF5 file

What is HDF5 chunking?

- Data is stored in a file in chunks of predefined size



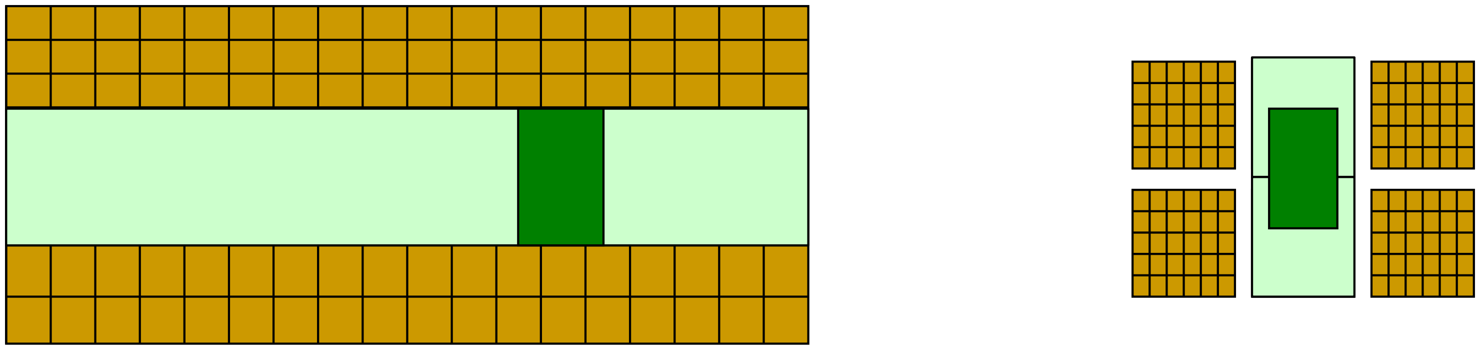
Why HDF5 chunking?

- Chunking is required for several HDF5 features
  - Expanding/shrinking dataset dimensions and adding/”deleting” data
  - Applying compression and other filters like checksum
- Example of the sizes with applied compression for our example file

Original size	**GZIP level 6**	**Shuffle and GZIP level 6**
256.8 MB	196.6 MB	**138.2 MB**

Why HDF5 chunking?

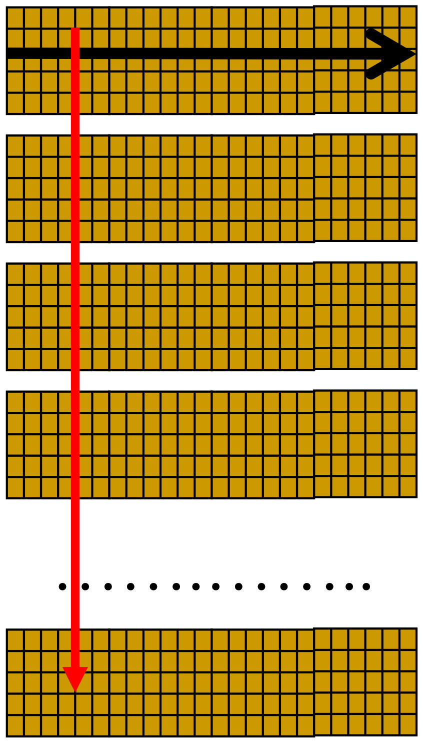
- If used appropriately chunking improves partial I/O for big datasets



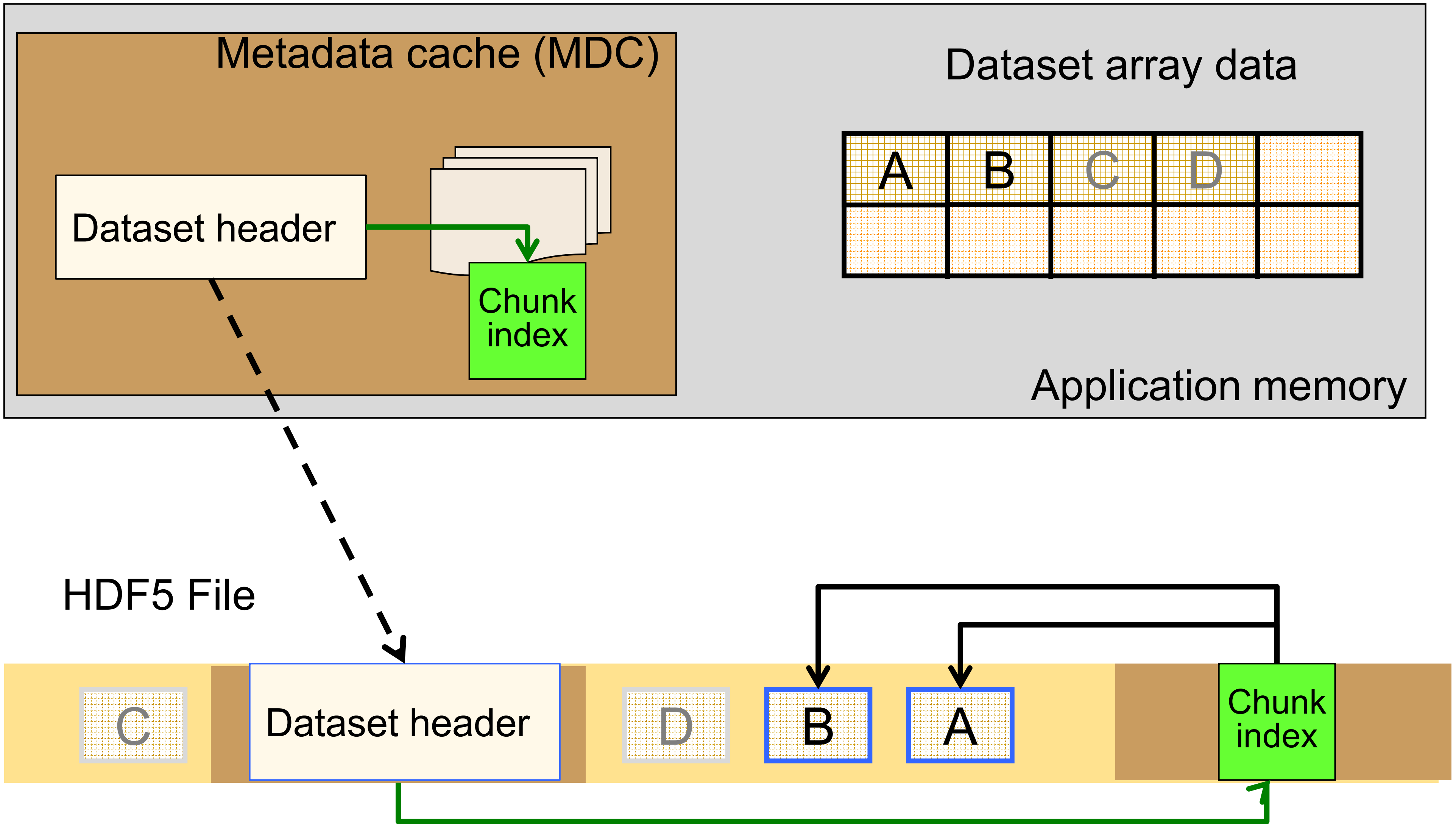
Only two chunks are involved in I/O

Chunking strategy

- **Extremely important**
- “ES_ImaginaryLW” is stored using 15 chunks
- with the size 4x30x9x717
- Chuck corresponds to “granule” size (a buffer created by an instrument)
- Affects performance when reading by columns since all data has to be read in order to read a column

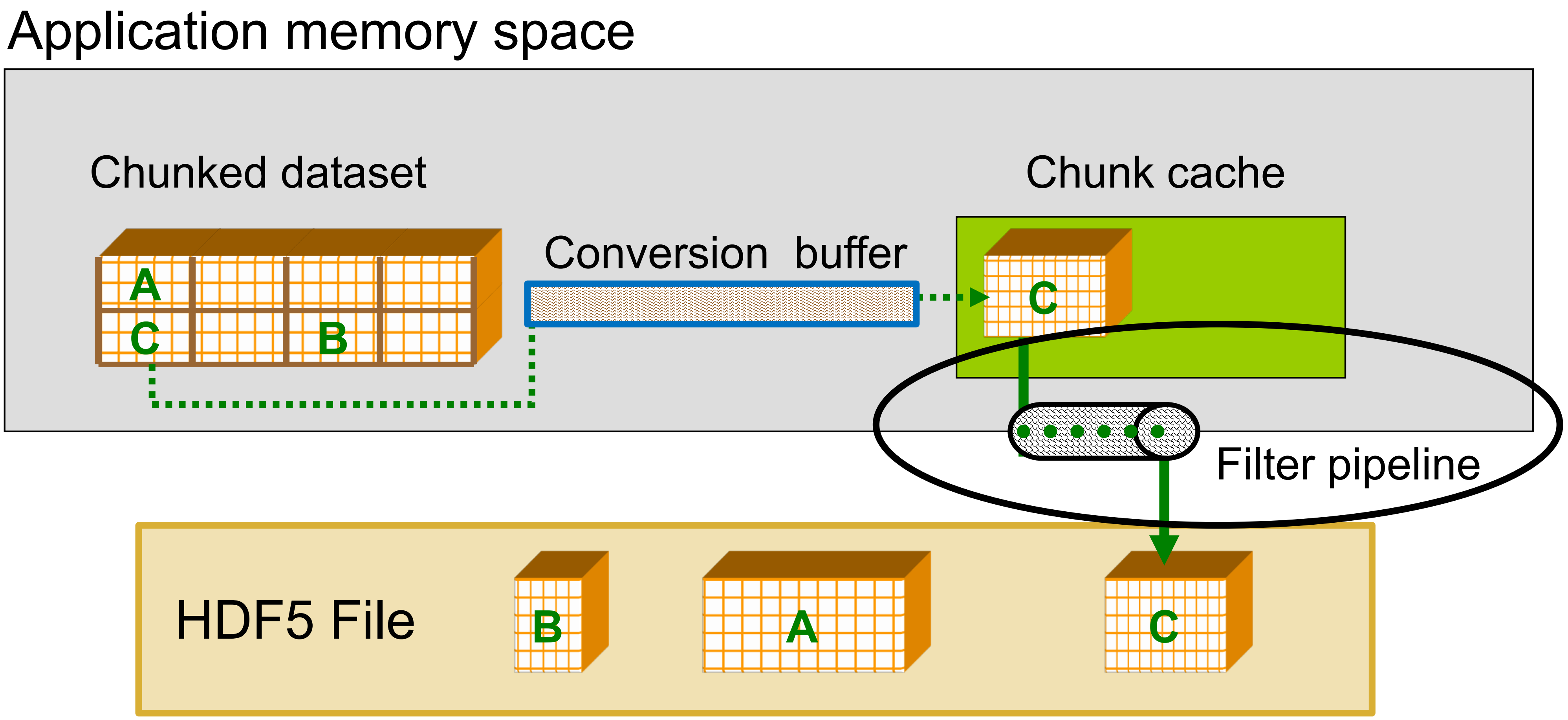


Chunked storage layout



Raw data is stored in separate chunks in HDF5 file

Writing chunked dataset



Datatype conversion is performed before chunked placed in cache
Chunk is written when evicted from cache
Compression and other filters are applied on eviction

FAQ

- Can one change chunk size after a dataset is created in a file?
  - No; use h5repack to change a storage layout or chunking/compression parameters
- How to choose chunk size?
  - *Next slide…*

Pitfall – chunk size

- Chunks are too small
  - File has too many chunks
  - Extra metadata increases file size
  - Extra time to look up each chunk
  - More I/O since each chunk is stored independently
  - Larger chunks results in fewer chunk lookups, smaller file size, and fewer I/O operations

Pitfall – chunk size

- Chunks are too large
  - Entire chunk has to be read and uncompressed before performing any operations
  - Great performance penalty for reading a small subset
  - Entire chunk has to be in memory and may cause OS to page memory to disk, slowing down the entire system

Better to See Something Once Than Hear About it Hundred Times

CASE STUDY

Case study

- We now look more closely into the solutions presented on the slides 5 -7.
  - Increasing chunk cache size
  - Changing access pattern
  - Changing chunk size

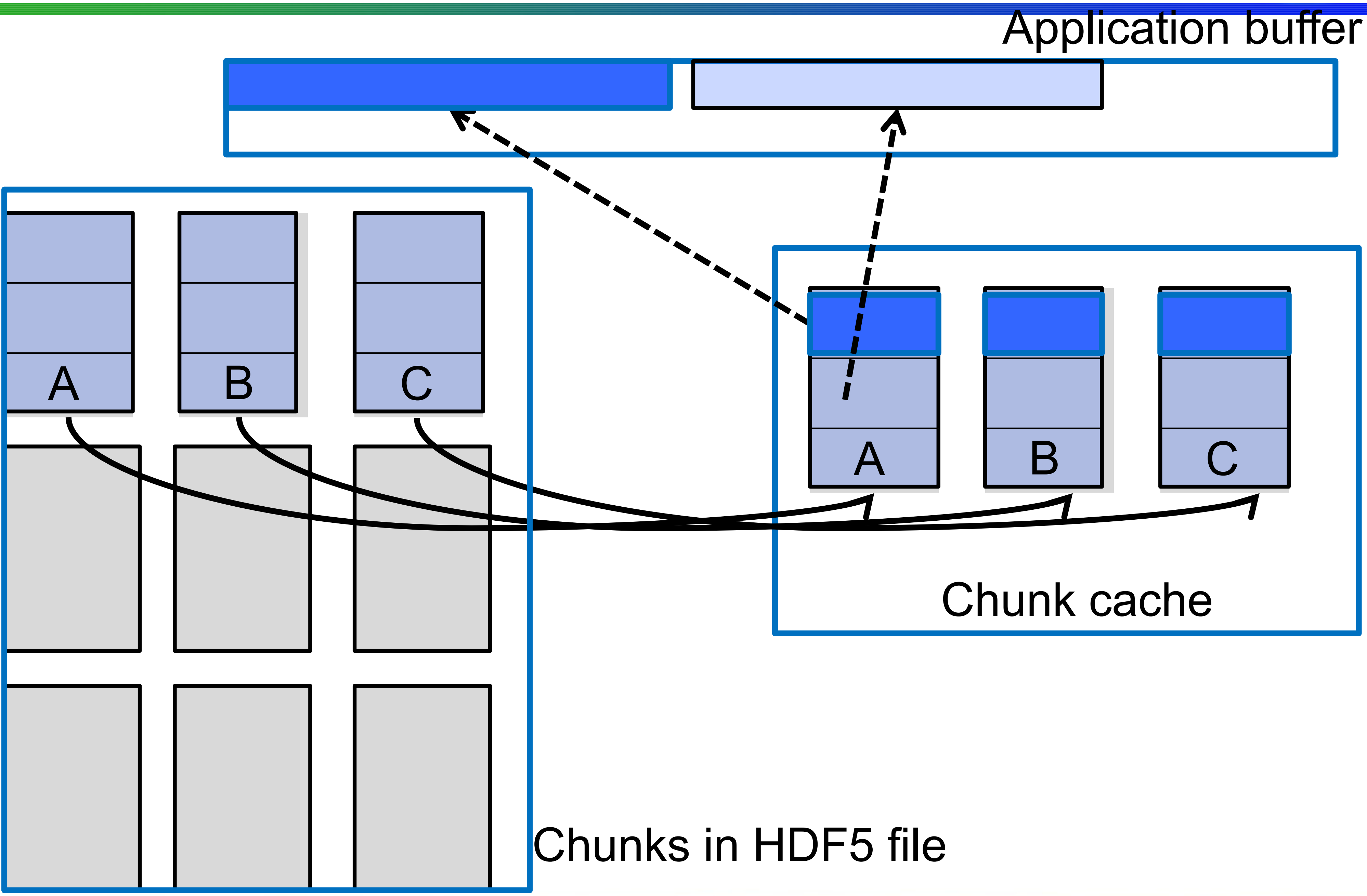
When chunk doesn’t fit into chunk cache

CHUNK CACHE SIZE

HDF5 raw data chunk cache

- The only raw data cache in HDF5
- Chunk cache is per dataset
- Improves performance whenever the same chunks are read or written multiple times (see next slide)
- Default size is 1MB
- Memory consumption grows if many chunked datasets are open
- Can be disabled

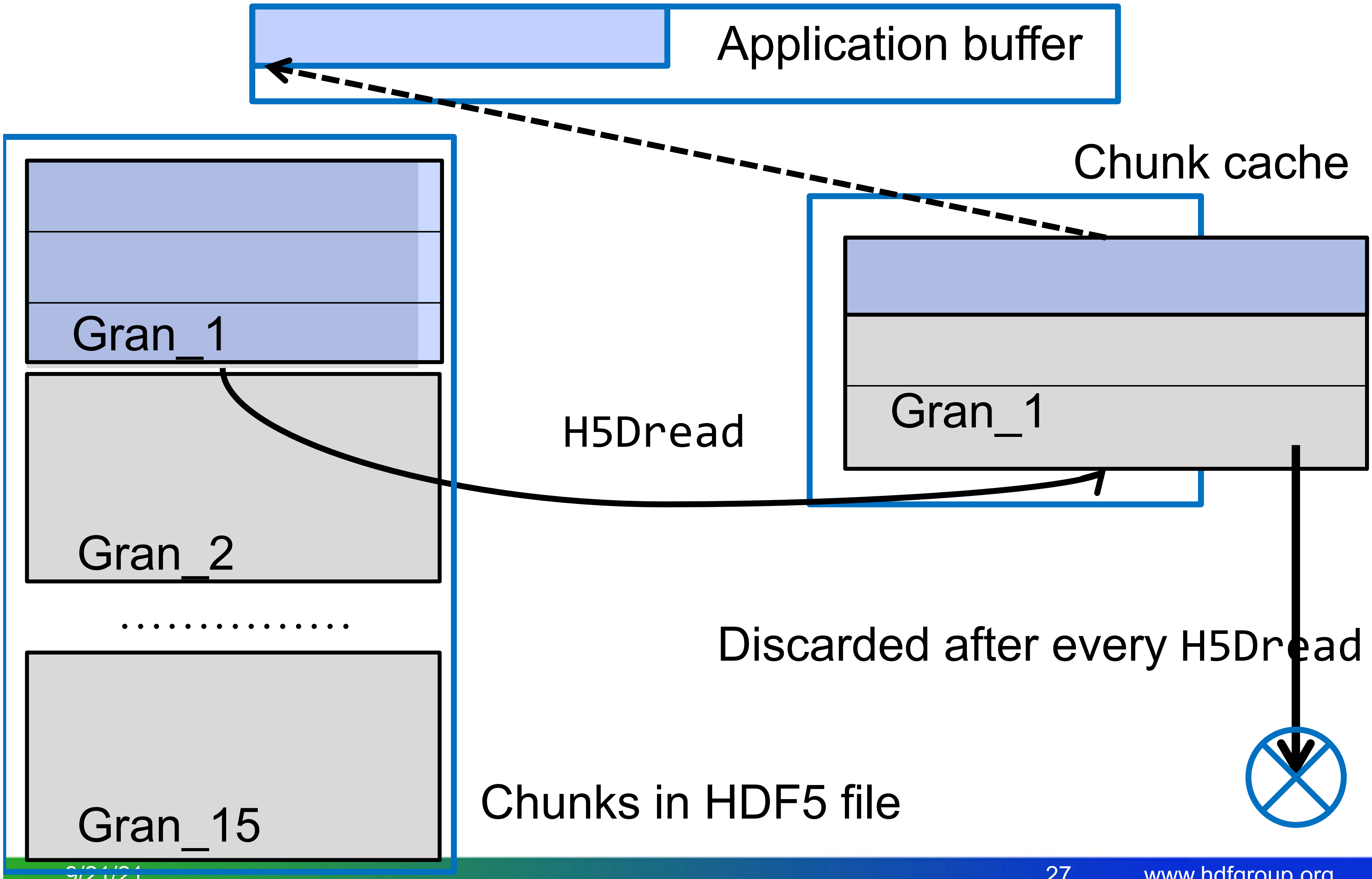
Example: reading row selection



HDF5 library behavior

- *When chunk doesn’t fit into chunk cache:*
  - Chunk is read, uncompressed, selected data converted to the memory datatype and copied to the application buffer.
  - Chunk is discarded.

Chunk cache size case study: Before



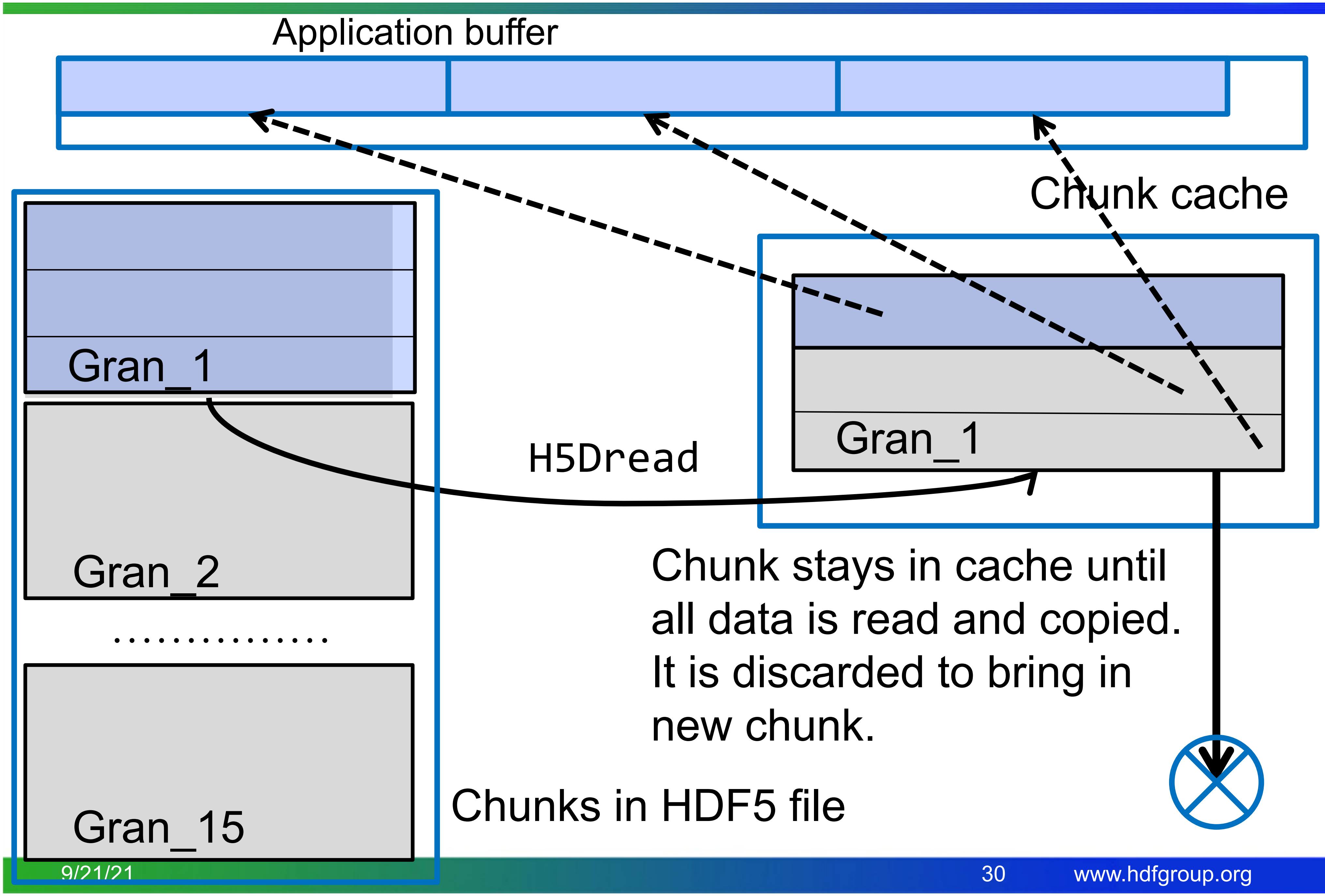
What happens in our case?

- *When chunk doesn't fit into chunk cache:*
  - Chunk size is 2.95MB and cache size is 1MB
  - If read by (1x1x1x717) selection, chunk is read and uncompressed 1080 times. For 15 chunks we perform 16,200 read and decode operations.
- *When chunk does fit into chunk cache:*
  - Chunk size is 2.95MB and cache size is 3MB
  - If read by (1x1x1x717) selection, chunk is read and uncompressed only *once*. For 15 chunks we perform 15 read and decode operations.
- How to change chunk cache size?

HDF5 chunk cache APIs

- `H5Pset_chunk_cache` sets raw data chunk cache parameters for a dataset
  - `H5Pset_chunk_cache (dapl, …);`
- `H5Pset_cache` sets raw data chunk cache parameters for all datasets in a file
  - `H5Pset_cache (fapl, …);`
- Other parameters to control chunk cache
  - *nbytes* – total size in bytes (1MB)
  - *nslots* – number of slots in a hash table (521)
  - *w0* – preemption policy (0.75)

Chunk cache size case study: After



Chunk stays in cache until all data is read and copied. It is discarded to bring in new chunk.

Chunks in HDF5 file

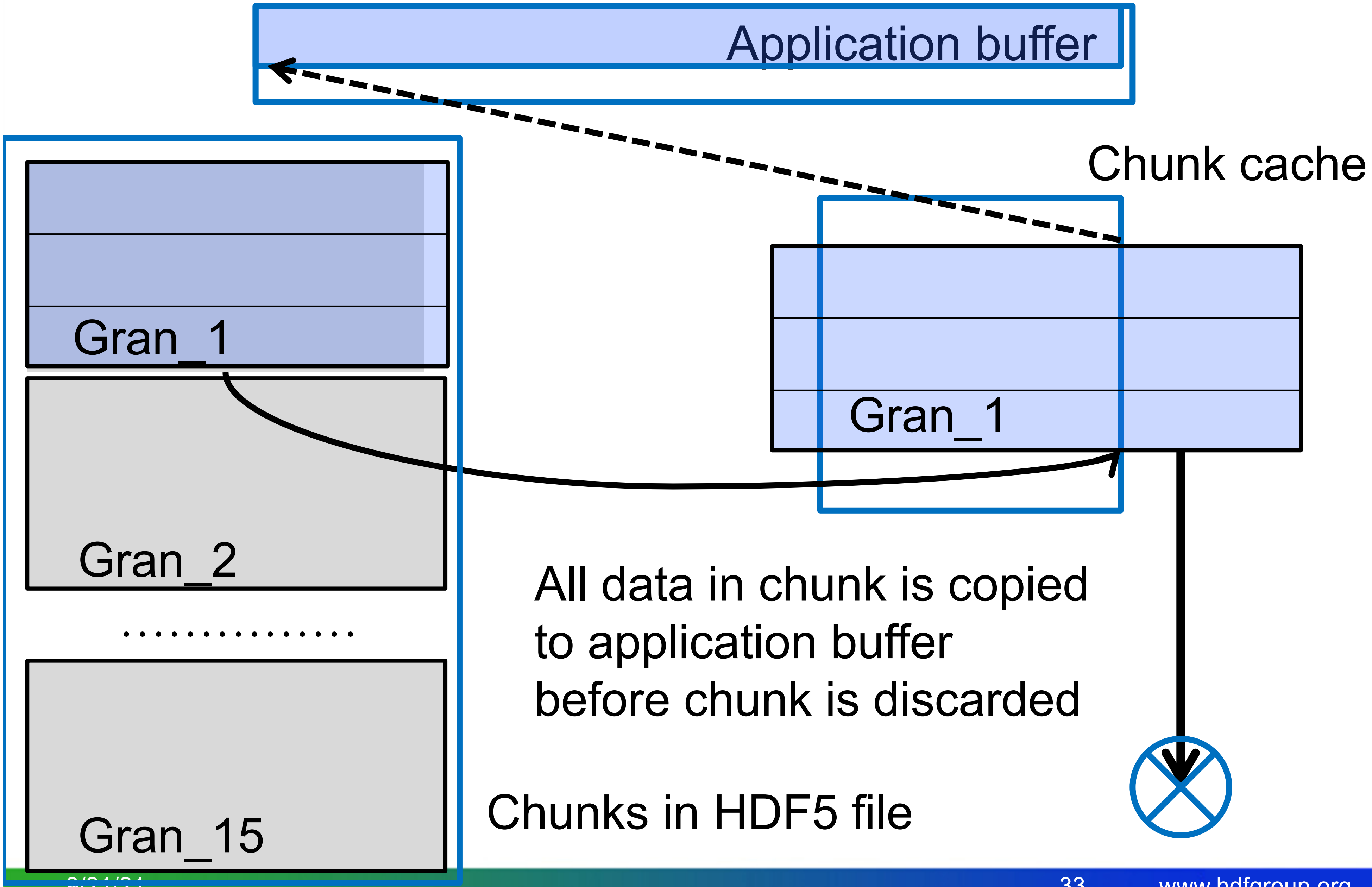
What else can be done except changing the chunk cache size?

ACCESS PATTERN

HDF5 Library behavior

- *When chunk doesn’t fit into chunk cache but selection is a whole chunk:*
  - If applications reads by the whole chunk (4x30x9x717) vs. by (1x1x1x717) selection, chunk is read and uncompressed once. For 15 chunks we have only 15 read and decode operations (compare with 16,200 before!)
  - Chunk cache is “ignored”.

Access pattern case study



All data in chunk is copied to application buffer before chunk is discarded

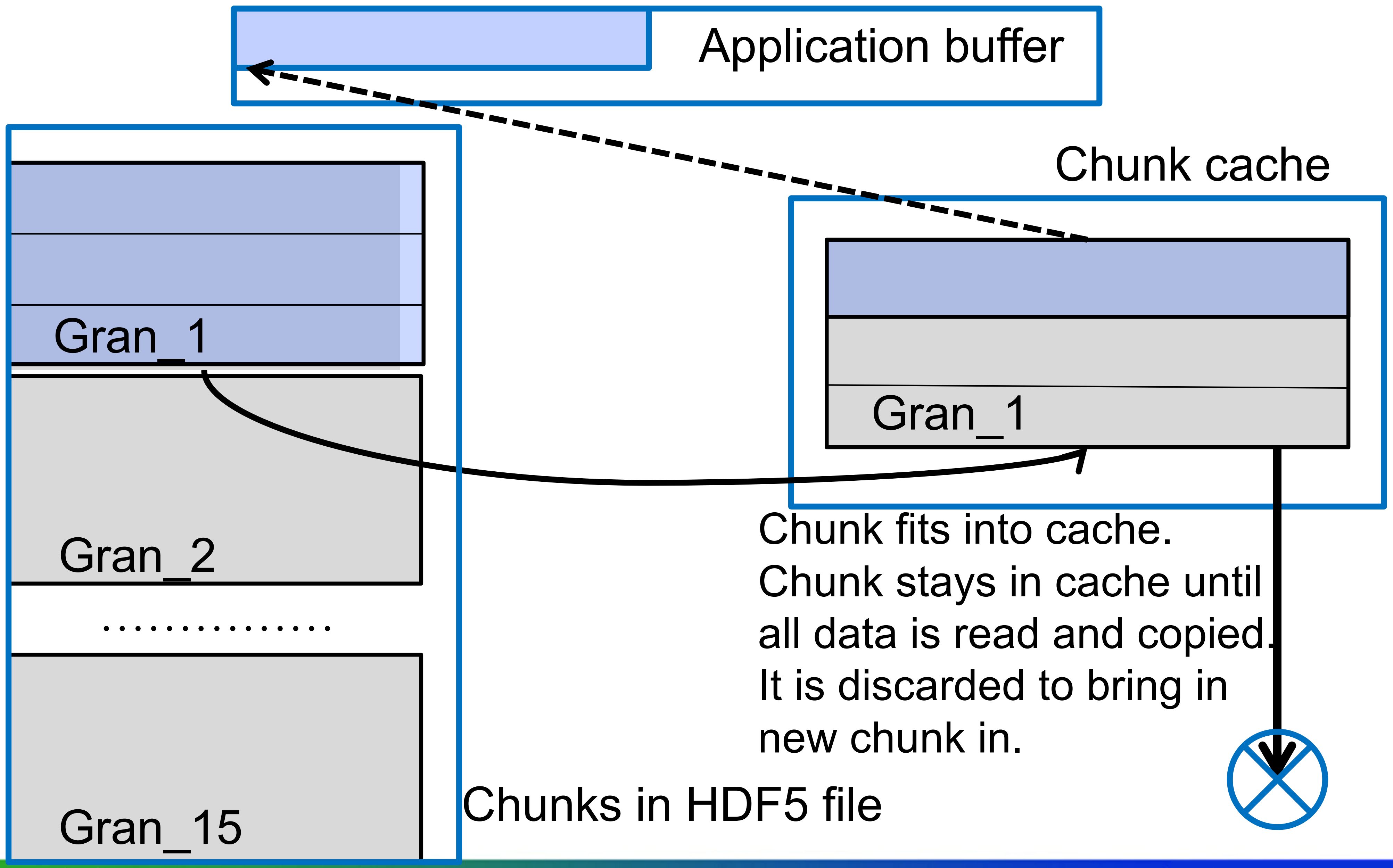
Can I create an “application friendly” data file?

CHUNK SIZE

HDF5 Library behavior

- *When datasets are created with chunks < 1MB*
  - Chunk fits into default chunk cache
  - No need to modify reading applications!

Small chunk size case study



Chunk fits into cache.
Chunk stays in cache until all data is read and copied.
It is discarded to bring in new chunk in.

Points to remember for data consumers and data producers

SUMMARY

Effect of cache and chunk sizes on read

- When compression is enabled, the library must always read entire chunk once for each call to `H5Dread` unless it is in cache.
- When compression is disabled, the library's behavior depends on the cache size relative to the chunk size.
  - If the chunk fits in cache, the library reads entire chunk once for each call to `H5Dread` unless it is in cache.
  - If the chunk does not fit in cache, the library reads only the data that is selected
    - More read operations, especially if the read plane does not include the fastest changing dimension
    - Less total data read

Effect of cache and chunk sizes on write

- If chunk doesn’t fit in cache and compression is enabled and not the whole chunk is written at once
  - All writes to the dataset must be immediately written to disk
  - With compression, the entire chunk must be read and rewritten every time a part of the chunk is written to
    - Data must also be decompressed and recompressed each time
    - Non sequential writes could result in a larger file

Thank You!

Questions?