HDF5 Chunking and Compression

Performance tuning

Goal

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

Problem

- 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)	No compression 1MB (default) or 3MB cache	**Compressed with GZIP level 6 3MB cache**
~345 seconds	~0.09 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 1x1x1`x717	No compression Selection 1x1x1`x717	**No compression** **Selection 4x9x30x717**	**Compressed with GZIP level 6 Selection 4x9x30x717**
~345 seconds	~0.1 seconds	**~0.04 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	No compression Selection 4x9x30x717	**No compression** **chunk size 1x9x30x717**	**Compressed with GZIP level 6 chunk size 1x9x30x717**
~345 seconds	~0.04 seconds	**~0.08 seconds**	**~0.36 seconds**

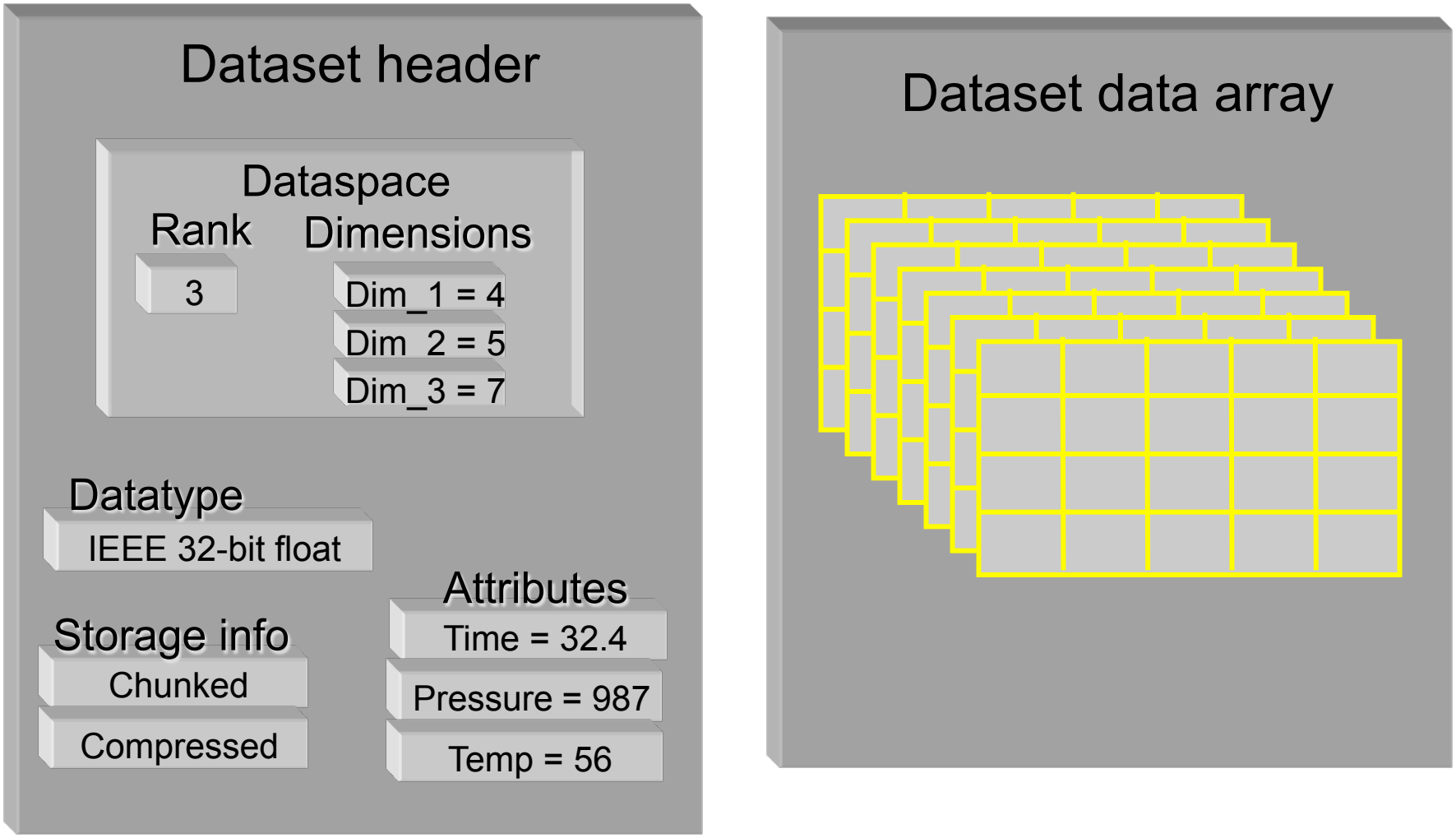
Outline

- HDF5 chunking overview
- HDF5 chunk cache
- Case study or how to avoid performance pitfalls
- Other considerations
  - Compression methods
  - Memory usage

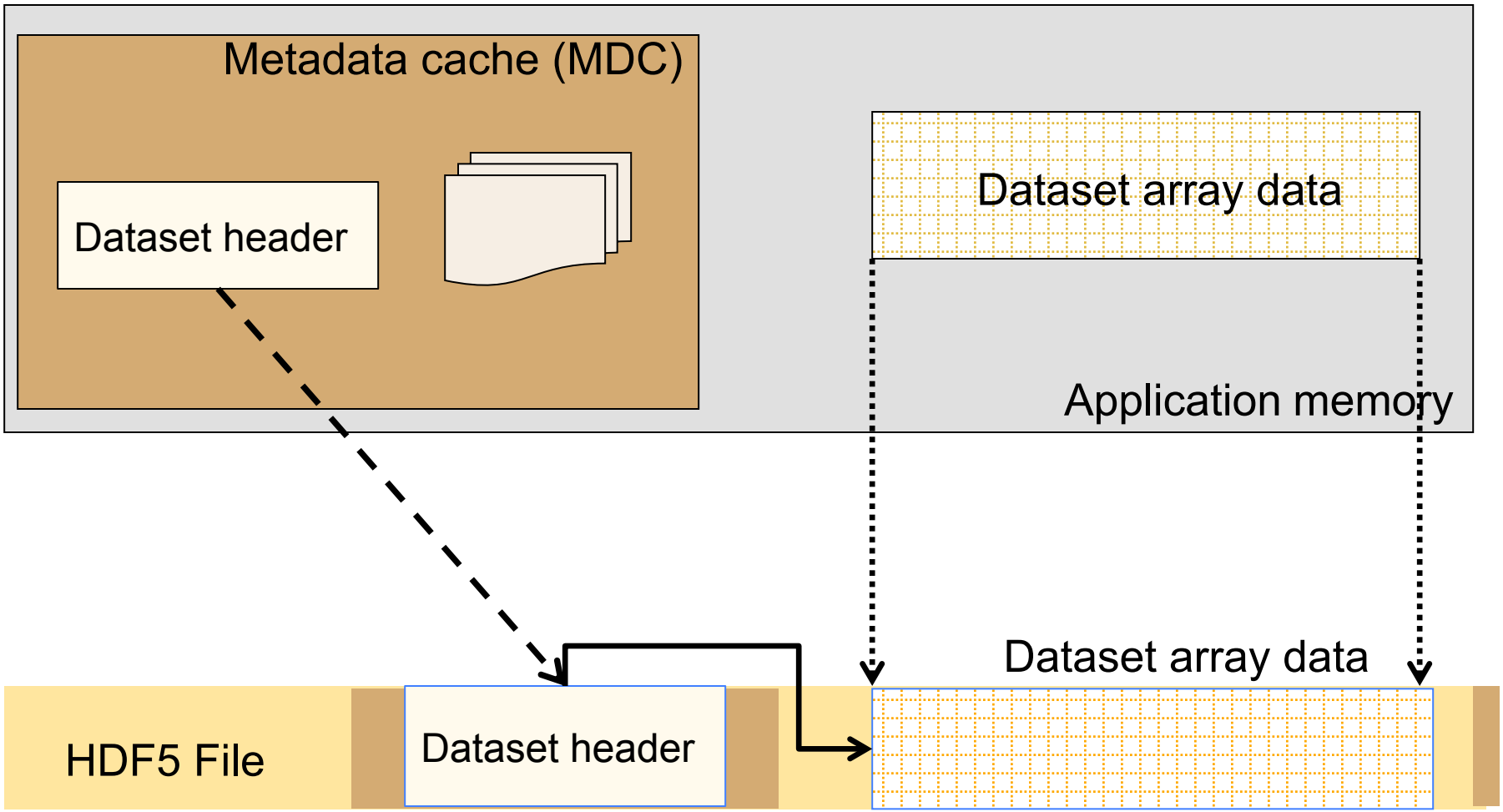
Reminder

HDF5 CHUNKING OVERVIEW

HDF5 Dataset Components



Contiguous storage layout

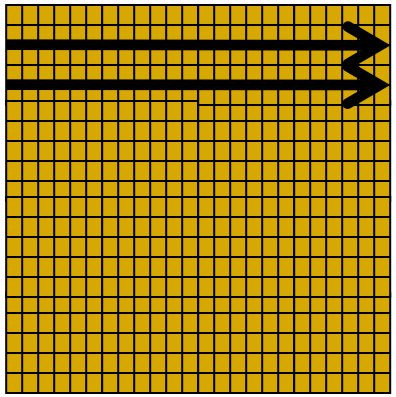


Raw data is stored in one contiguous block in HDF5 file

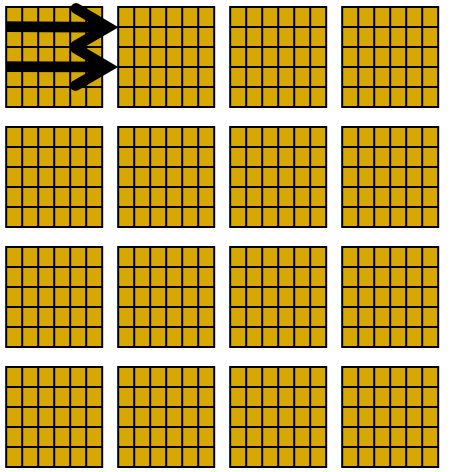
What is HDF5 chunking?

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

Contiguous



Chunked



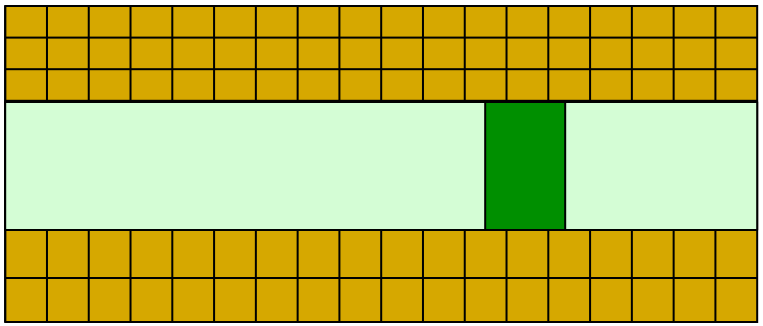
Why HDF5 chunking?

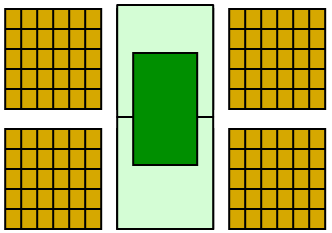
- Chunking is required for several HDF5 features
  - Expanding/shrinking dataset dimensions and adding/"deleting" data
  - Applying compression and other filters like checksum
- Example of the sizes with applied compression for our example file

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



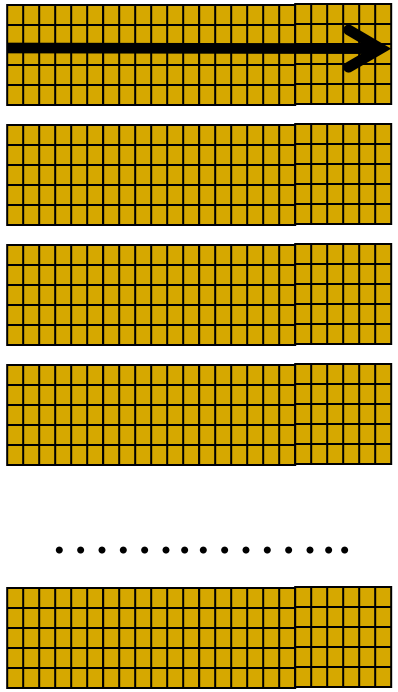


Only two chunks are involved in I/O

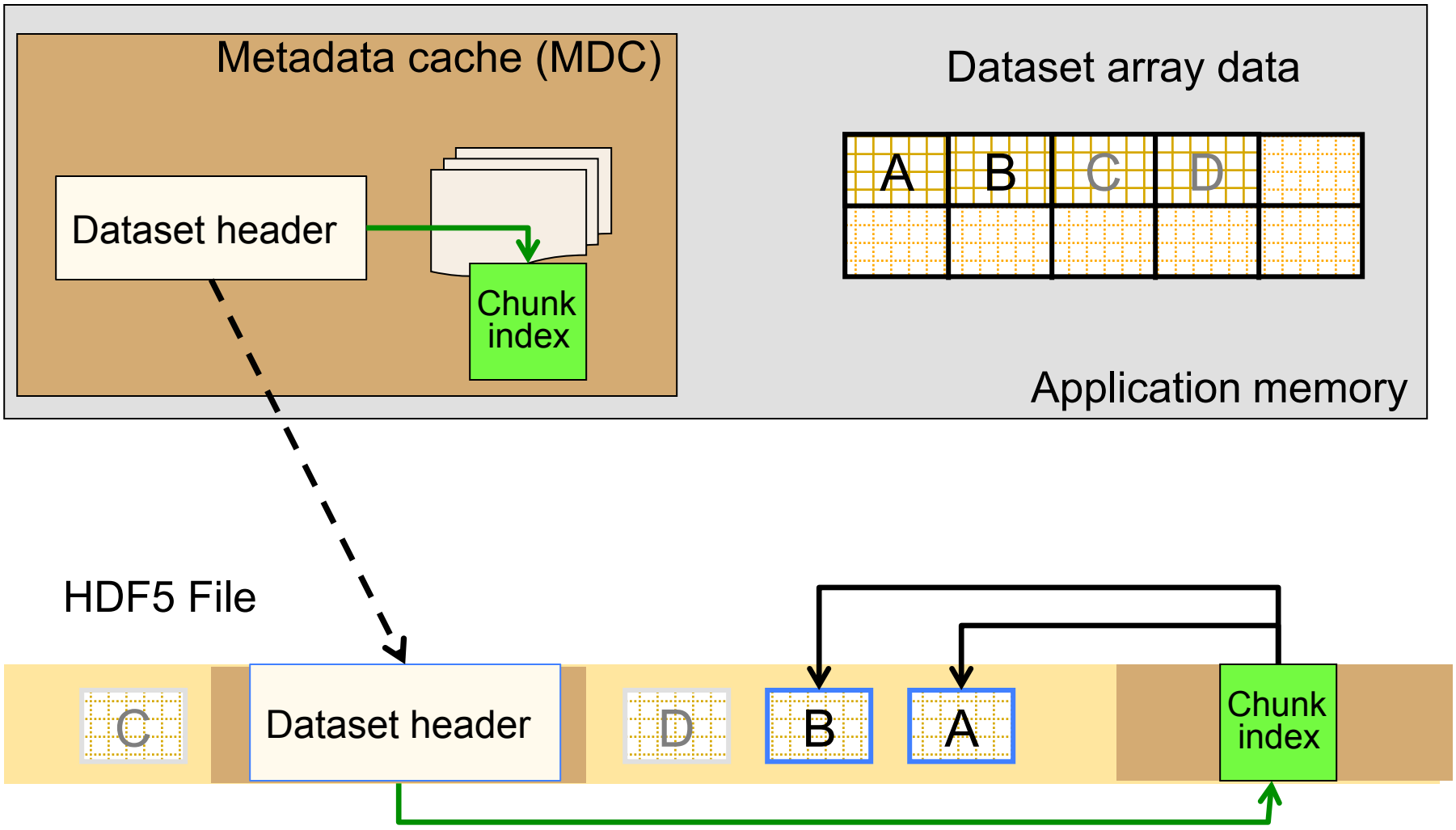
JPSS chunking strategy

- JPSS uses granule size as chunk size

  “ES_ImaginaryLW” is stored using 15 chunks with the size 4x30x9x717



Chunked storage layout



Raw data is stored in separate chunks in HDF5 file

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

HDF5 CHUNK CACHE

HDF5 chunk cache documentation

http://www.hdfgroup.org/HDF5/doc/Advanced.html

Advanced Topics in HDF5

The following topics are broadly applicable in HDF5 applications.

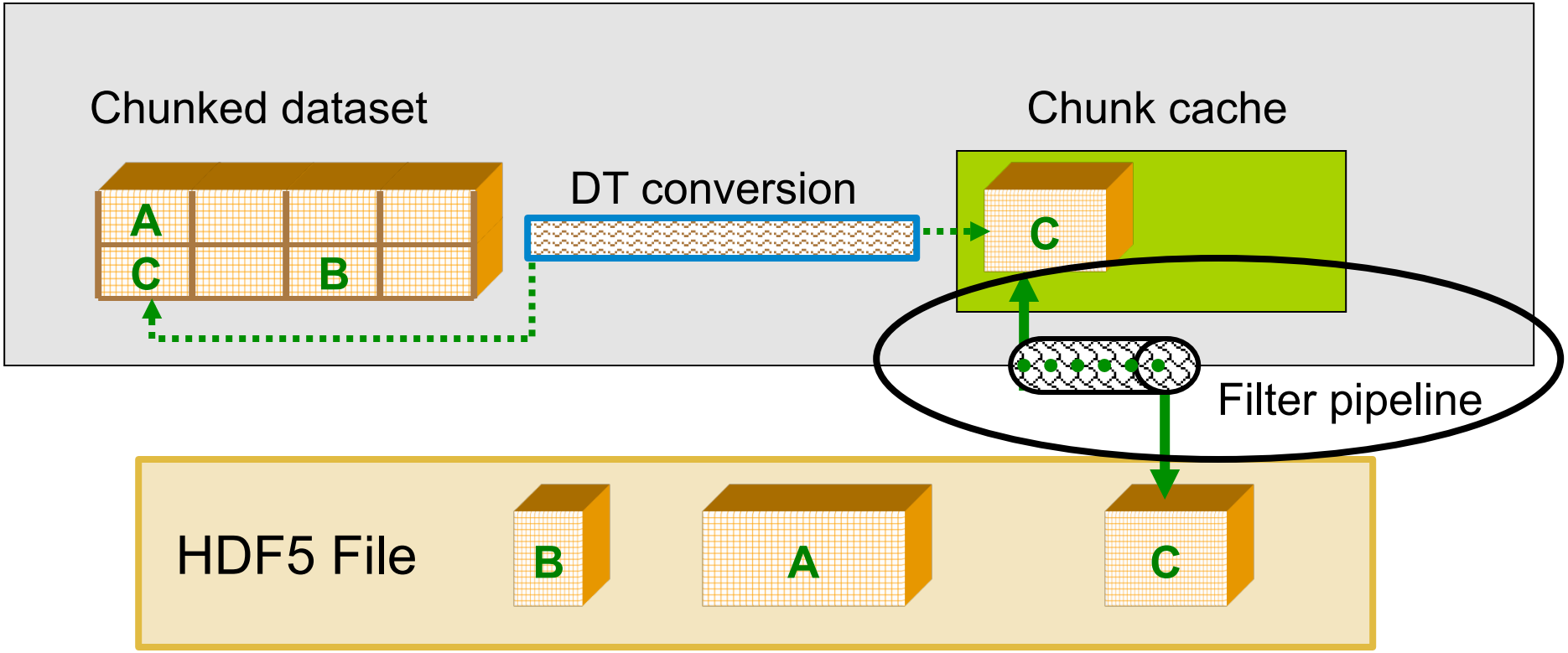
Chunking in HDF5	Provides background and guidance to assist in structuring and tuning the use of chunking.
Using the Direct Chunk Write Function	Describes another way that chunks can be written to datasets. *(PDF only)*
Copying Committed Datatypes with H5Ocopy	Describes how to copy to another file a dataset that uses a committed datatype or an object with an attribute that uses a committed datatype so that the committed datatype in the destination file can be used by multiple objects. *(PDF only)*
Using Identifiers	Describes how HDF5 identifiers behave and how they should be treated.
Using UTF-8 Encoding in HDF5 Applications	Describes the use of UTF-8 Unicode character encodings in HDF5 applications.
HDF5 Metadata	Provides a comprehensive overview of the types of metadata used in HDF5.
HDF5 Dynamically Loaded Filters	Describes how an HDF5 application can apply a filter that is not registered with the HDF5 Library. *(PDF only)*
HDF5 Data Flow Pipeline for H5Dread	Describes data flow when reading raw data from an HDF5 dataset. *(PDF only)*

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)

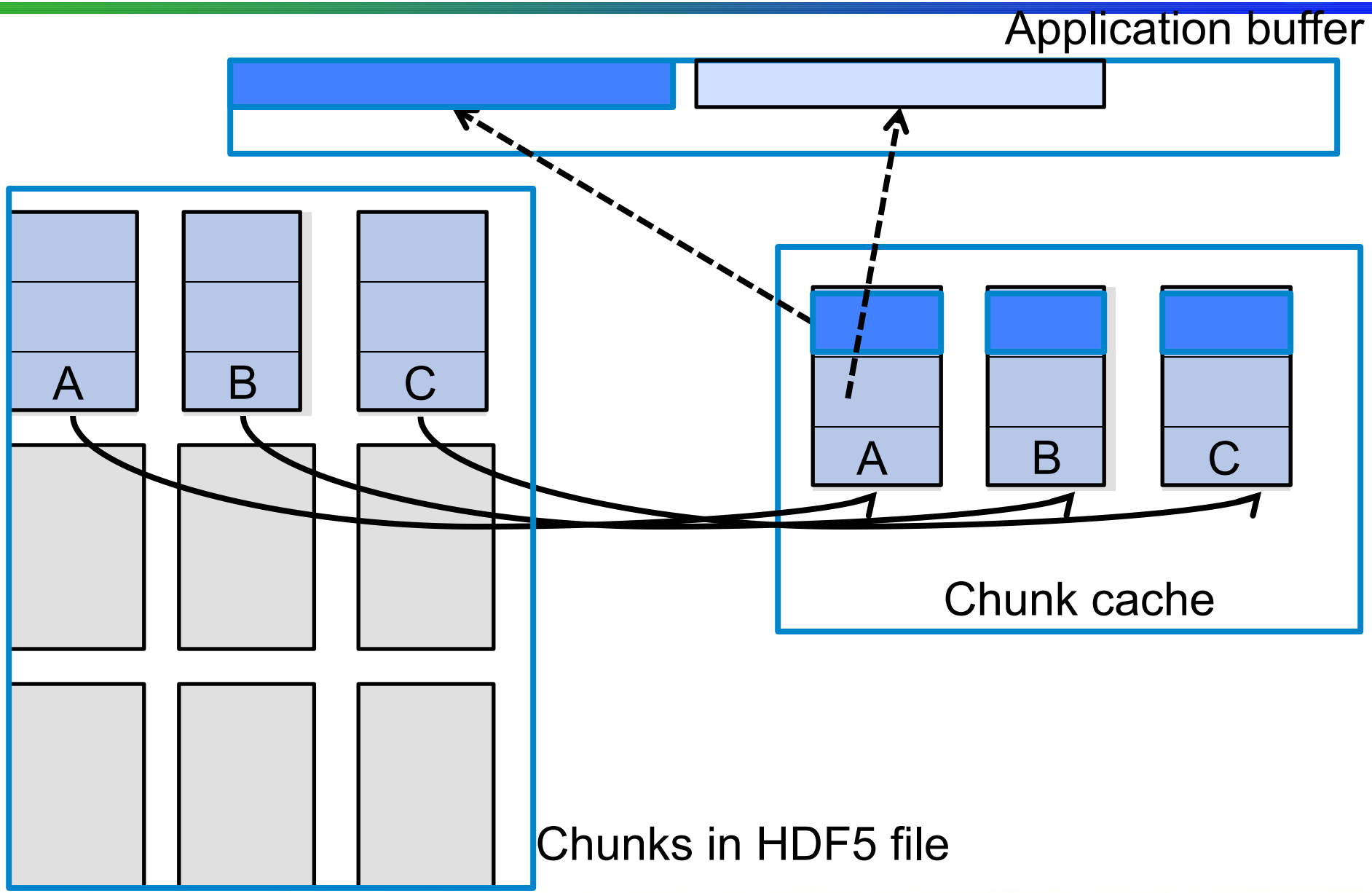
Chunked Dataset I/O

Application memory space



Datatype conversion is performed before chunked placed in cache on write
Datatype conversion is performed after chunked is placed in application buffer
Chunk is written when evicted from cache
Compression and other filters are applied on eviction or on bringing chunk into cache

Example: reading row selection



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 4 -7.
  - Increasing chunk cache size
  - Changing access pattern
  - Changing chunk size

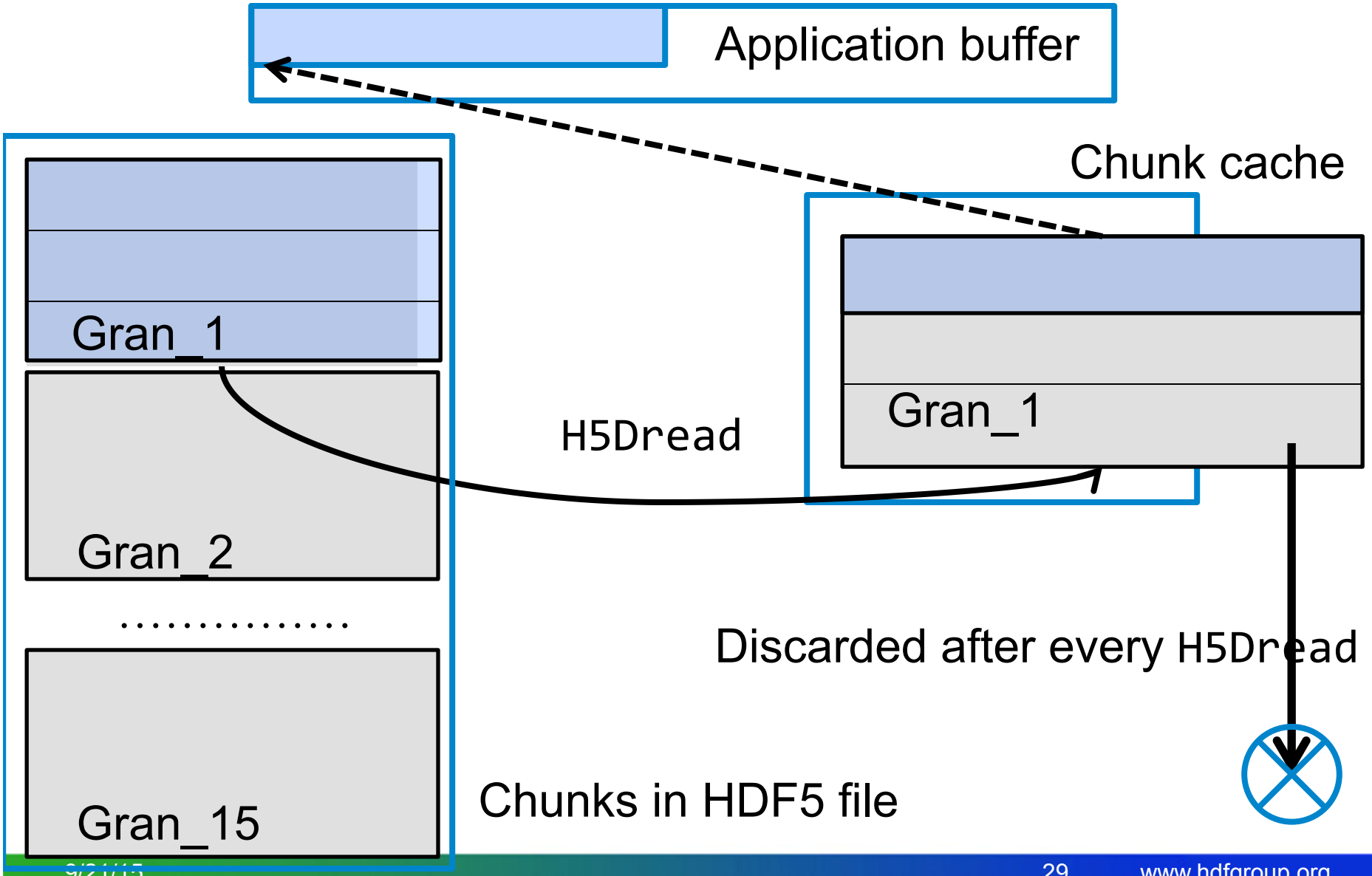
When chunk doesn’t fit into chunk cache

CHUNK CACHE SIZE

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



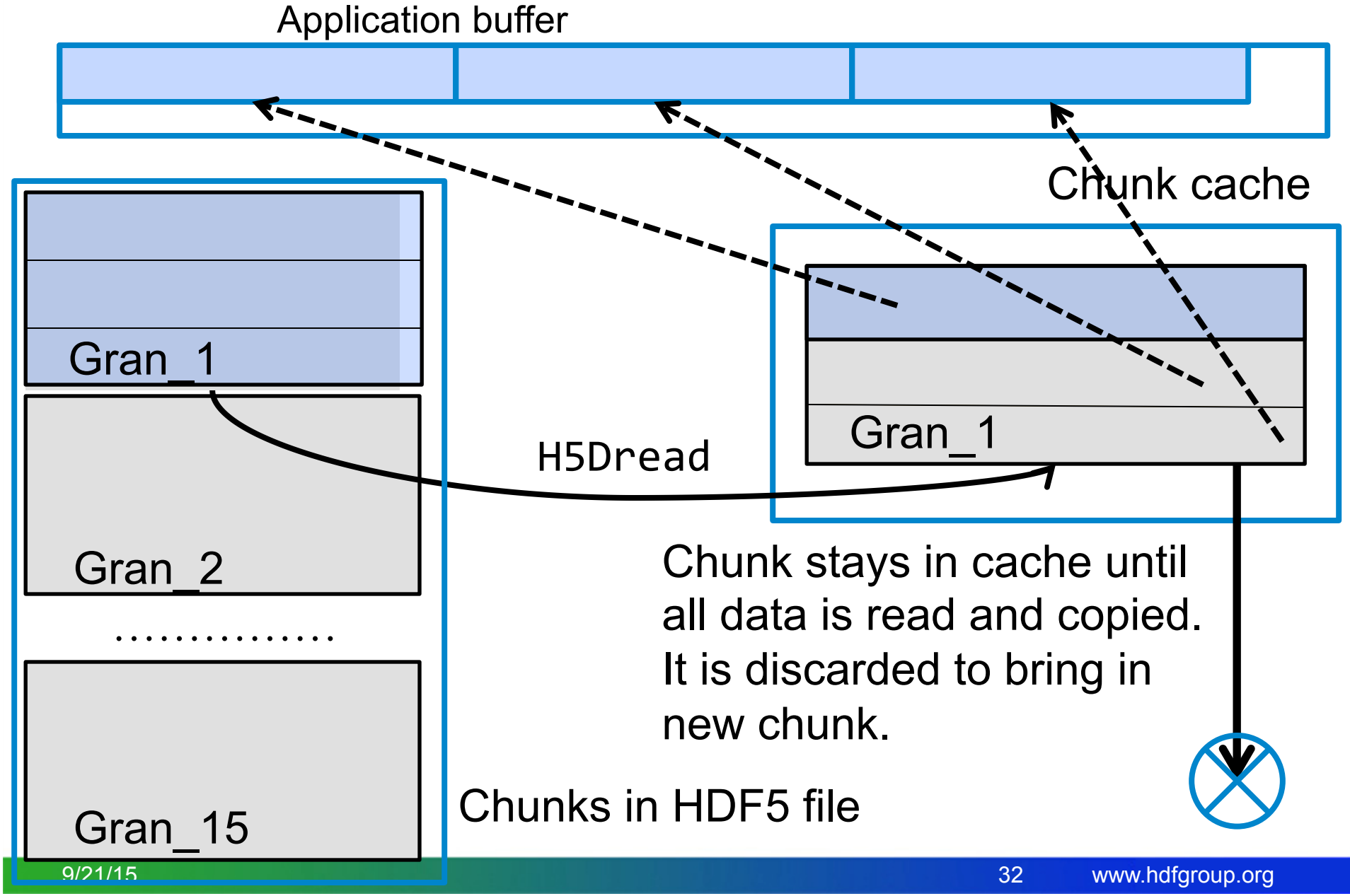
What happens in our case?

- *When chunk **doesn't fit** into chunk cache:*
  - Chunk size is 2.95MB and cache size is **1**MB
  - 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 **3**MB
  - 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



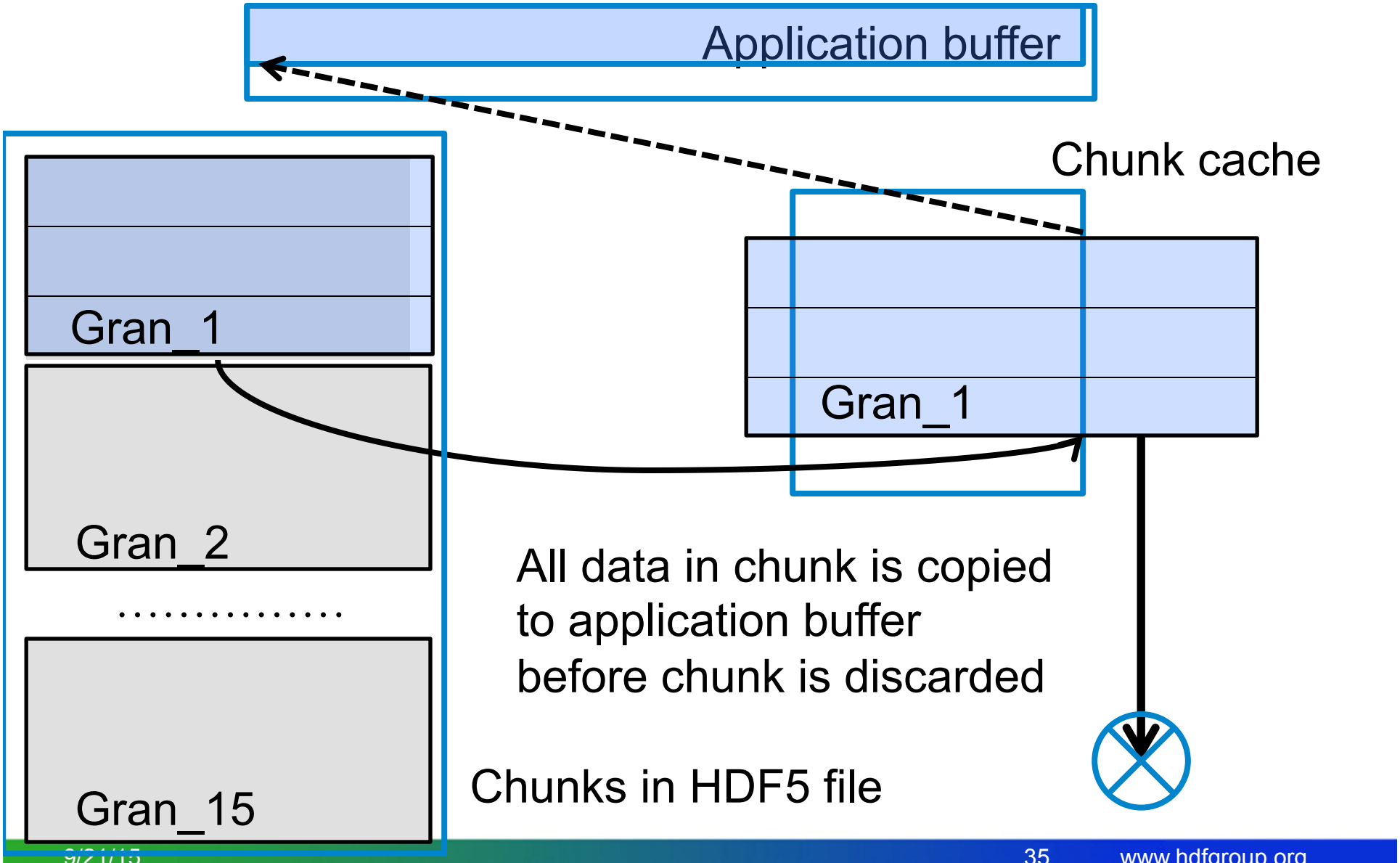
What else can be done except changing the chunk cache size?

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



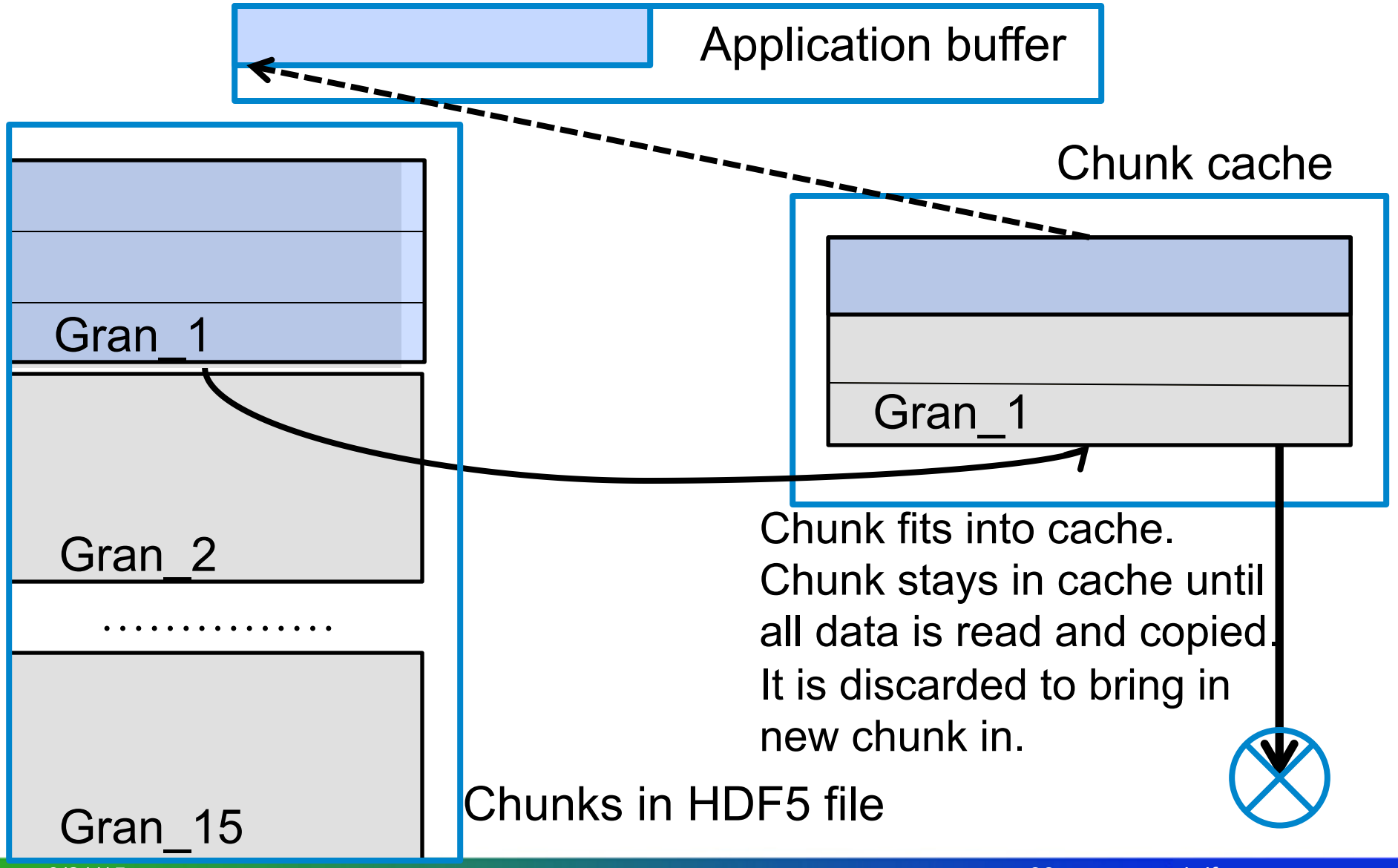
Can I create an “application friendly” data file?

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



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

OTHER CONSIDERATIONS

Compression methods

- Choose compression method appropriate for your data
- HDF5 compression methods
  - GZIP, SZIP, n-bit, scale-offset
  - Can be used with the shuffle filter to get a better compression ratio; for example for “ES_NEdNSW” dataset (uncomp/comp) ratio

“ES_NEdNSW” compressed with GZIP level 6	“ES_NEdNSW” compressed with shuffle and GZIP level 6
15.5	19.1

  - Applied for all datasets the total file size changes from 196.6MB to 138.2MB (see slide 13)

Word of caution

- Some data cannot be compressed well. Find an appropriate method or don't use compression at all to save processing time.
- Let's look at compression ratios for the datasets in our example file.

Example: Compression ratios

Use `h5dump -pH filename.h5` to see compression information

Compression ratio = uncompressed size/compressed size

Dataset name	Compression ratio with GZIP level 6	Compression ratio with shuffle and GZIP level 6
ES_ImaginaryLW	1.076	1.173
ES_ImaginaryMW	1.083	1.194
ES_ImaginarySW	1.079	1.174
ES_NedNLW	1.17	**18.589**
ES_NedNMW	**14.97**	**17.807**
ES_NedNSW	**15.584**	**19.097**
ES_RealLW	1.158	1.485
ES_RealMW	1.114	1.331
ES_RealSW	1.1.42	1.341

Memory considerations for applications

- HDF5 allocates metadata cache for each open file
  - 2MB default size; may grow to 32MB depending on the working set
  - Adjustable (see HDF5 User's Guide, Advanced topics chapter); minimum size 1K
- HDF5 allocates chunk cache for each open dataset
  - 1MB default
  - Adjustable (see reference on slide 31); can be disabled
- Large number of open files and datasets increases memory used by application

Acknowledgement

This work was supported by SGT under Prime Contract No. NNG12CR31C, funded by NASA.

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of SGT or NASA.

Thank You!

Questions?