# RFC: HDF5 Virtual Dataset

**Quincey Koziol (koziol@hdfgroup.org)**
**Elena Pourmal (epourmal@hdfgroup.org)**
**Neil Fortner (nfortne2@hdfgroup.org)**

This document introduces Virtual Datasets (VDS) for HDF5 and summarizes the use cases, requirements, and programming model for them. The document also outlines VDS implementation and other changes to HDF5 software.

## Table of Contents

# 1 Introduction

There have been several requests to The HDF Group to implement a new feature called “Virtual Datasets” (VDS). While requests have come from different communities such as the synchrotron community and the earth sciences community, they have a common requirement: to view data stored across HDF5 files as a single unified HDF5 dataset on which normal I/O operations can be performed.

This document formalizes the notion of the “Virtual Dataset” and documents its requirements and use cases.

## 2 HDF5 Virtual Datasets

Our goal is to introduce and implement a new way of constructing and accessing collections of HDF5 datasets in the same way as if the data was stored in a single dataset in one HDF5 file.

An HDF5 Virtual Dataset is an HDF5 dataset with the following properties:

1. Data elements for a virtual dataset are stored in source datasets in one or more source HDF5 files.
2. There is a mapping from a set of elements in each source dataset to a set of elements in the virtual dataset.

For example, the diagram below shows mapping portions of four source datasets to a single virtual dataset:

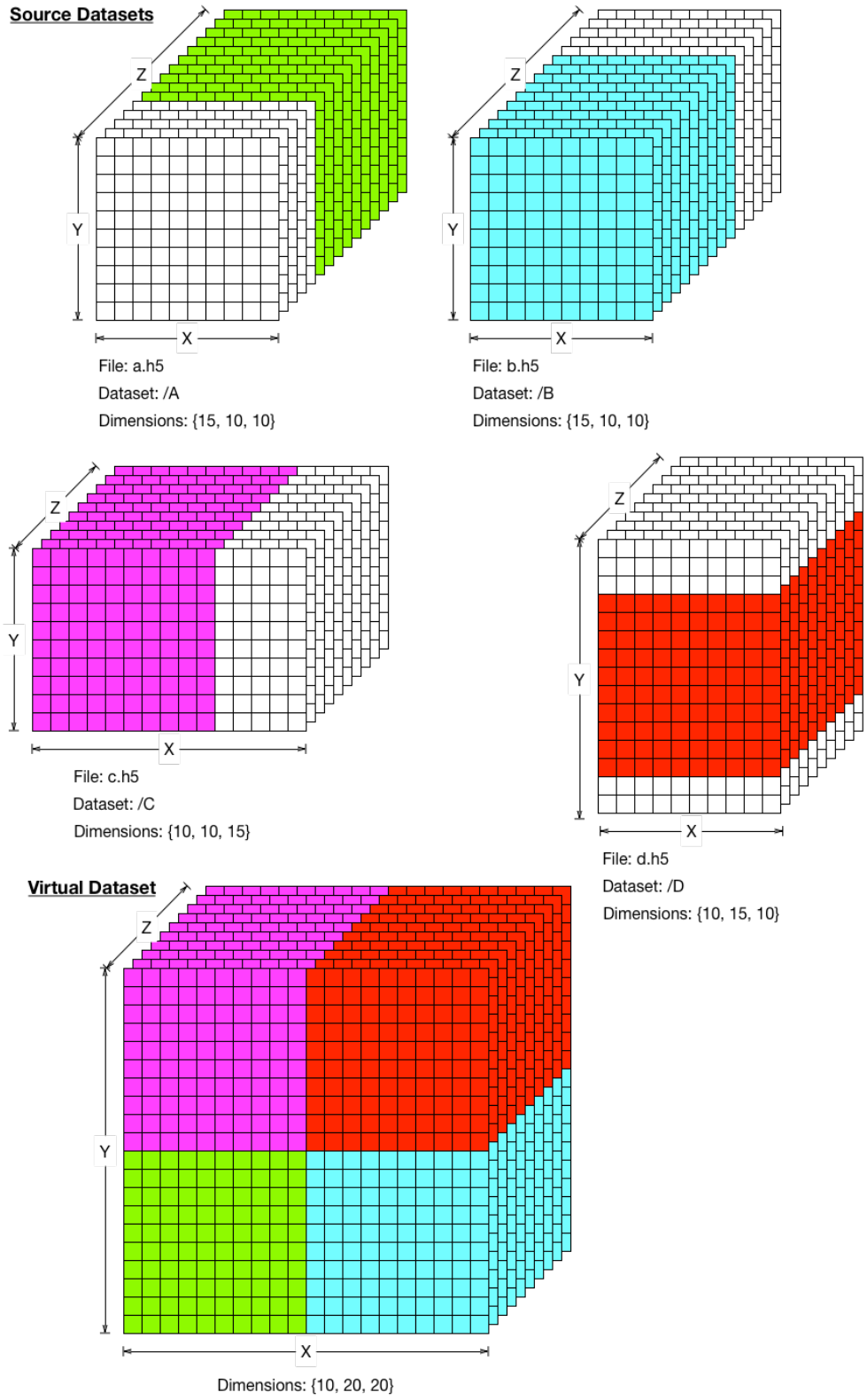


**Figure 1: Mapping Source Datasets to Virtual Dataset**

After a virtual dataset has been created, its data elements can be accessed using typical HDF5 API routines such as H5Dread and H5Dwrite with the actual data elements being retrieved from the corresponding locations in the source datasets.

Furthermore, virtual datasets can have unlimited dimensions and map to source datasets with unlimited dimensions, allowing a virtual dataset to grow over time, as its underlying source datasets change in size. Extending the idea of source datasets growing over time (and therefore the virtual dataset growing) to the realm of simultaneous writing to and reading from those datasets (in other words, the single-writer/multiple-reader (SWMR) feature in HDF5) shows that a virtual dataset can be employed to create a single, coherent view of a set of source datasets that are being concurrently written while the virtual dataset is read from, something otherwise not possible with HDF5.

# 3 VDS Use Cases

In this section we discuss several use cases for VDS.

## 3.1 DLS Use Case "Excalibur"

The Excalibur detector is operated by Diamond Light Source (DLS). For more information, see the "References" chapter on page 58. A series of images taken from the detector are stored in six HDF5 files as shown in Figure 2.

At every time step, data for a 2D image is divided into six sub-images *A*, *B*, *C*, *D*, *E,* and *F*. The sub-images *A*, *C,* and *E* have size k x M, and the sub-images *B*, *D,* and *F* have size n x M. The size of the full image is (3k +3n) x M.

A time series for each sub-image *A*, *B*, *C*, *D*, *E,* and *F* is stored in a 3D dataset in a corresponding HDF5 file; for example, time series data for sub-images *A* are stored in a.h5 in the dataset A, time series data for sub-images *B* are stored in b.h5 in the datasets B, and so on

Each 3D source dataset may have its own chunking and compression properties. For example, datasets A, C, and E have chunk size (1 x k x M), while datasets B, D, and F have chunk size (1 x n x M).

The slowest changing dimension of each 3D dataset corresponds to the "time" axis while a plane orthogonal to the axis represents the sub-image at the given time step. The sub-images are written in the order they are taken. Independent processes write sub-images, and there is no synchronization between the processes. Therefore, at any particular moment the number of the planes in the sub-image datasets may be different.

If an application needs to read a full image at some particular time step, it can read corresponding sub-images and then combine them in an application buffer, but the application (or user) has to know how to find sub-images in the HDF5 files.

A virtual dataset VDS in the file VDS.h5 (see Figure 2) provides a seamless access to the sub-images without explicit knowledge by the application of where the actual data is stored. As with sub-image datasets, a VDS represents the series of full images. The slowest changing dimension of the VDS corresponds to the "time" axis, and the planes orthogonal to the axis correspond to the full images. In Figure 2 the highlighted planes in datasets A, B, C, D, and E are sub-images of the image "stored" in the VDS at time step $t_2$.

If some of the sub-images at a particular time step have not been written yet, for example, highlighted images in b.h5, d.h5, and f.h5, and if the second plane in VDS is read, the corresponding sub-images *B*, *D*, and *F* in the full image will be filled with the fill values, or the read may fail (depending on the access properties for the I/O operation).

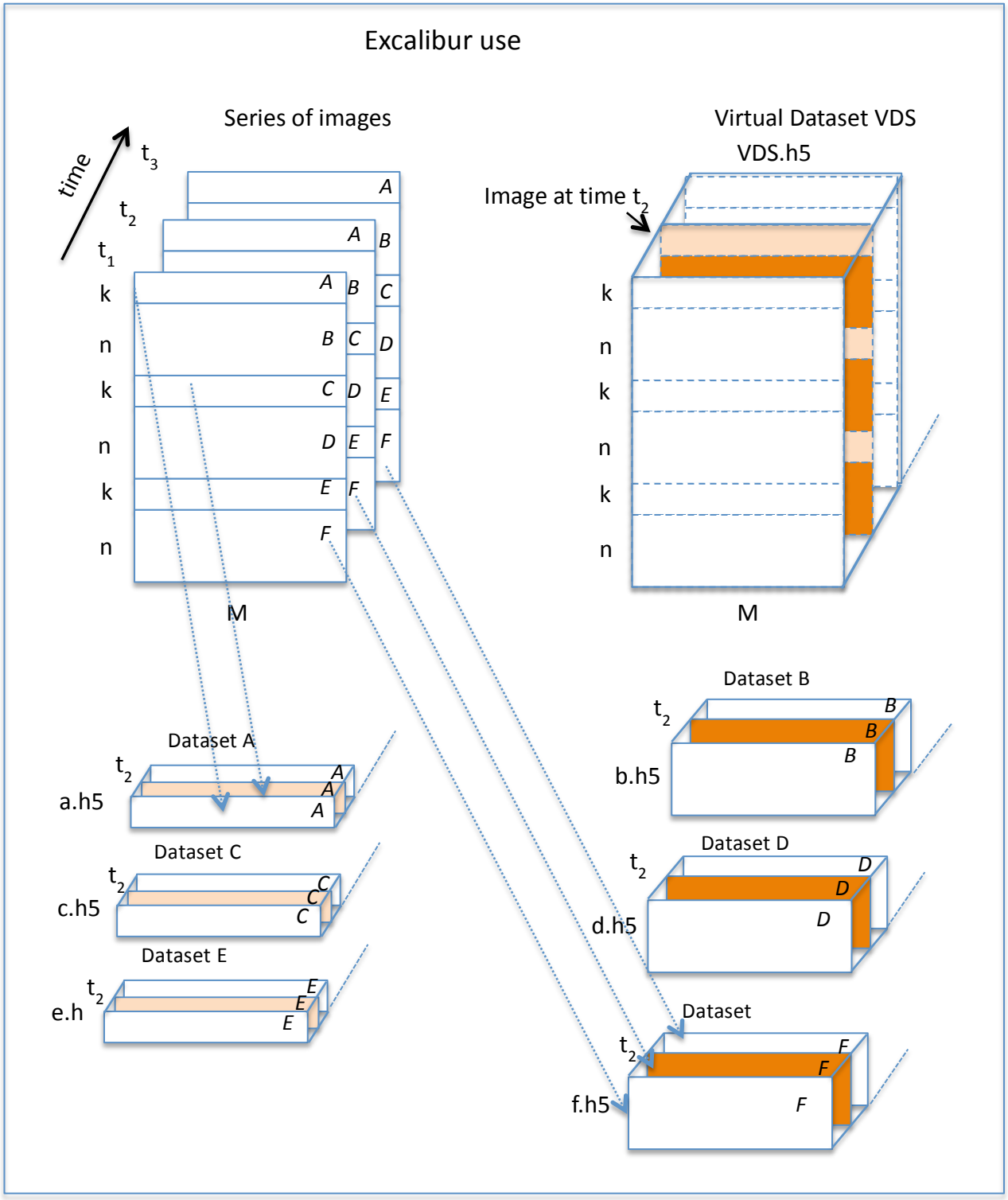


**Figure 2: Excalibur Use Case**

The series of each sub-image is stored in a 3D dataset with unlimited dimension corresponding to time. A virtual dataset presents the series of full images comprised of sub-images.

## 3.2 DLS Use Case “Dual PCO Edge”

The “Dual PCO” detector is also operated by Diamond Light Source (DLS). For more information, see the “References” chapter on page 58. Data for a full image consists of 5 sub-images as showed in Figure 3. Each sub-image is stored as in the Excalibur use case, use case 3.1, with one difference: each dataset is written by a selection that contains several consecutive sub-images. To optimize writing of compressed data, the slowest changing dimension for a chunk can be set to a number of sub-images desired to be compressed and written with one HDF5 write call.

When written, the data is comprised of two groups of HDF5 files. Each group contains data for an image taken over time from one camera. Sub-images are stored in the corresponding datasets as in the Excalibur use case, 3.1.

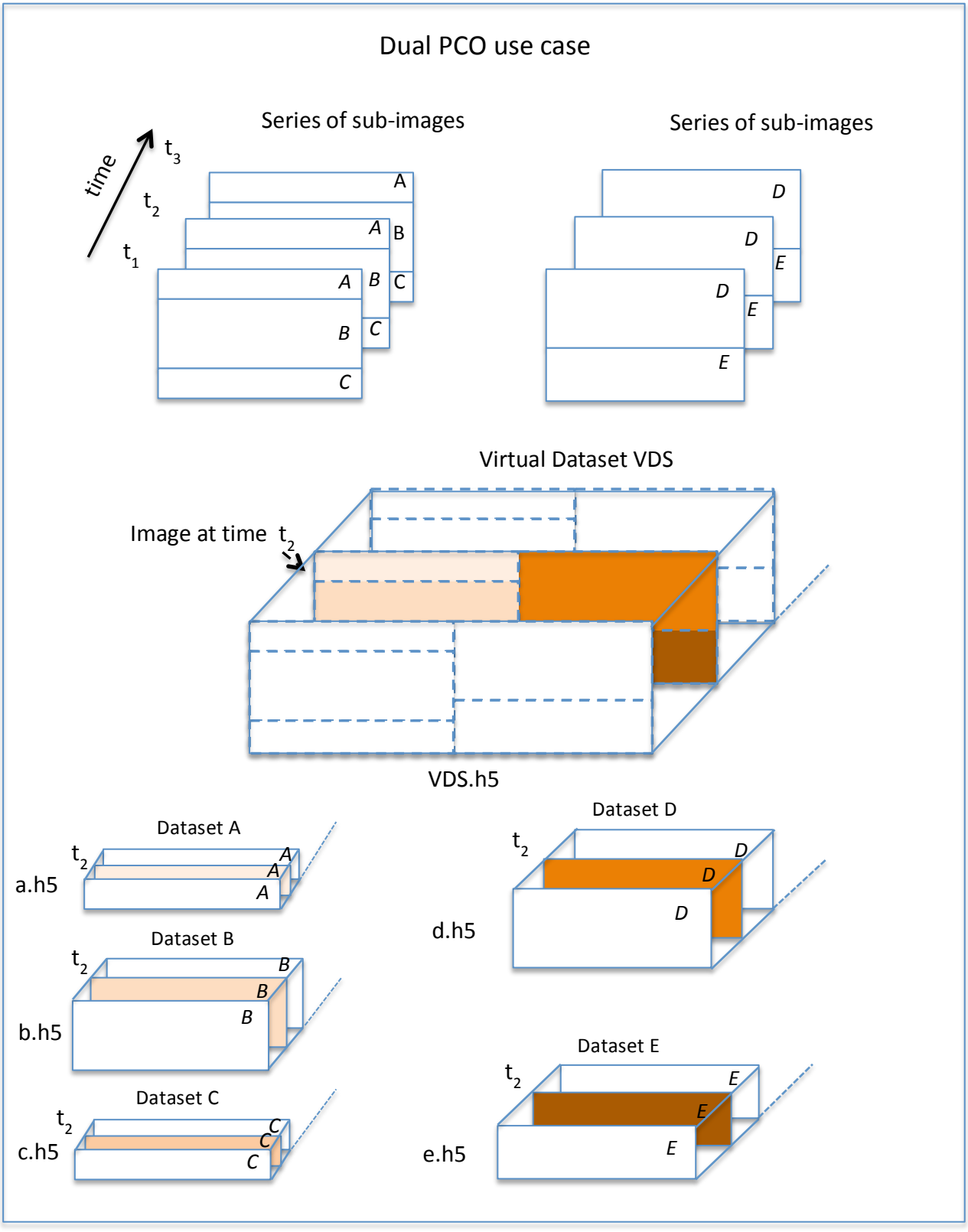


**Figure 3: Dual PCO use case: sub-images are taken from two different cameras**

The storage layout is similar to the Excalibur use case. The properties of each 3D dataset may be set to have "thicker" chunks for data buffering.

### 3.3 Dealing with Gaps

Both “Excalibur” and “Dual PCO Edge” have to accommodate gaps between sub-images. This can be accomplished by allowing “gaps” in the virtual dataset as shown in Figure 4. When a plane is read from VDS, “gaps” are filled with the fill values defined by the virtual dataset or with private data for the VDS.

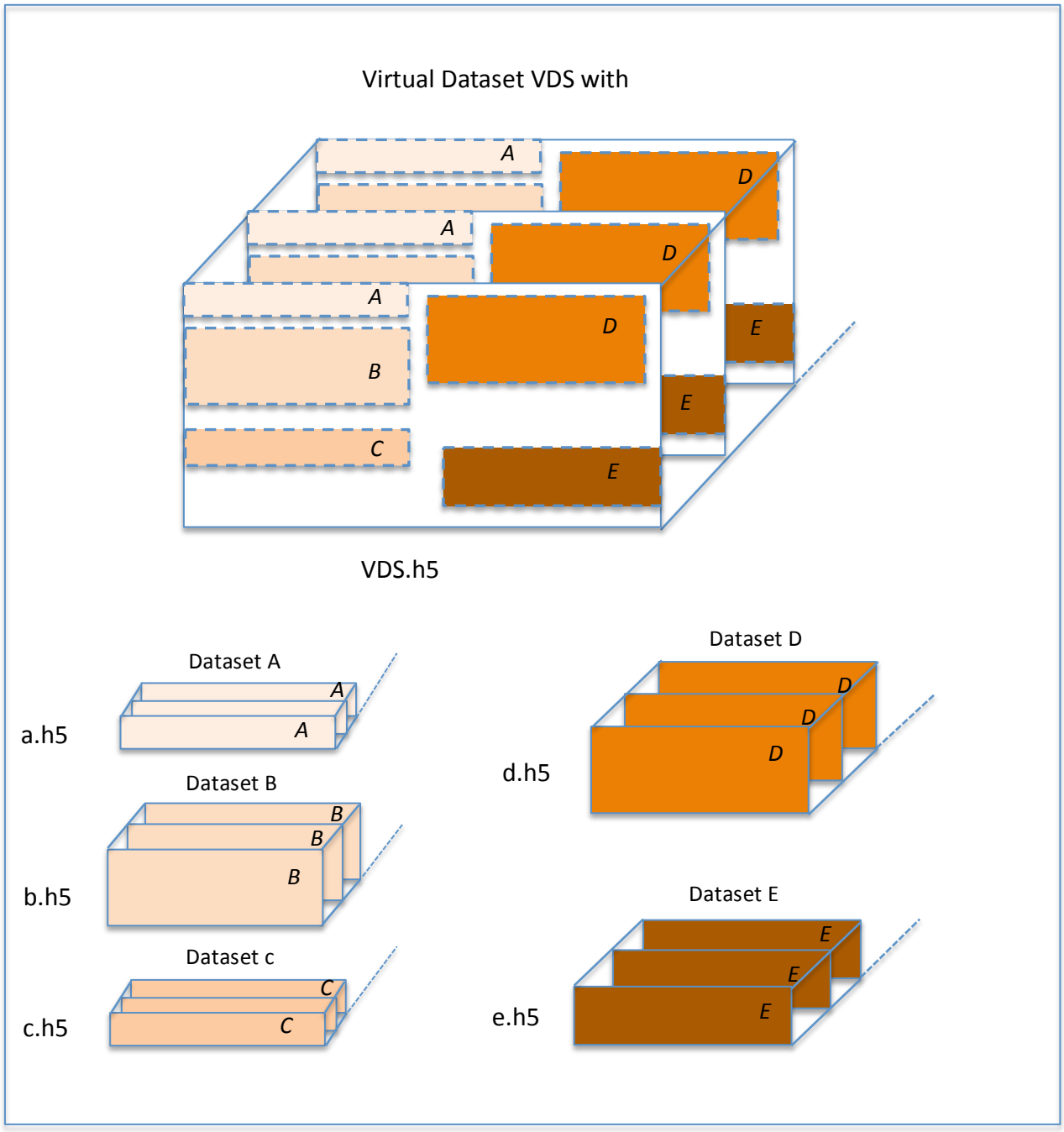


**Figure 4: Virtual datasets VDS contains “gaps” shown in white**

### 3.4 DLS Use Case “Eiger”

The “Eiger” detector is another machine operated by Diamond Light Source (DLS). For more information, see the “References” chapter on page 58.

Images from every M time steps are written to a 3D dataset in a separate HDF5 file. The slowest changing dimension of the 3D dataset has size M. When written, the data comprises N HDF5 files, each with M consequent images, except the last dataset that may have a fewer images as shown in Figure 5. Chunking and compression parameters for each 3D dataset are the same; for example, chunk size can be chosen to be the same as of the size of the image. The virtual dataset has a slowest changing dimension that represents the “time” axis. Each plane orthogonal to the “time” axis represents an image.

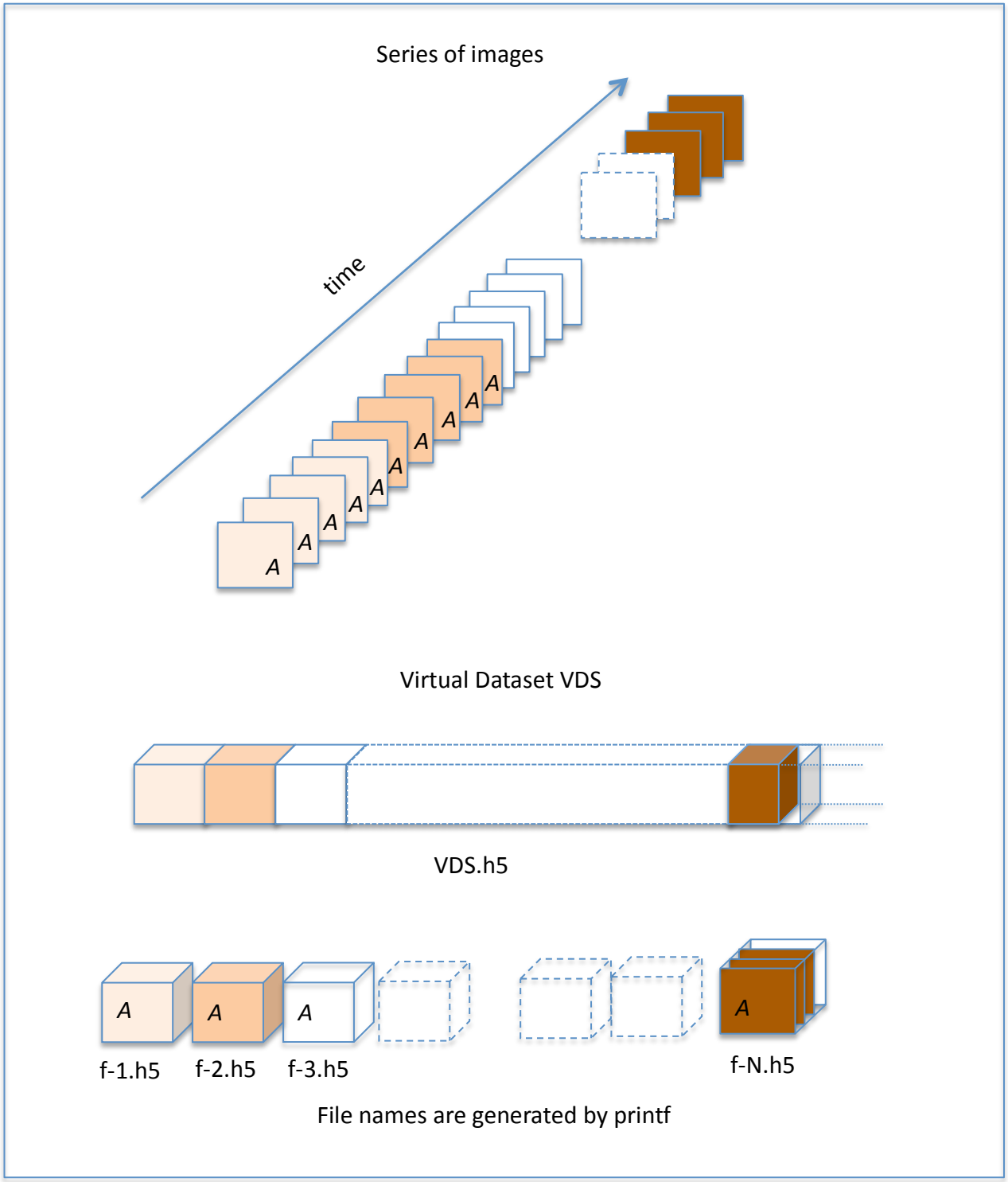


**Figure 5: Each HDF5 file stores M consequent images**

## 3.5 DLS Use Case “Percival Frame Builder”

The “Percival Frame Builder” detector is another machine operated by Diamond Light Source (DLS). For more information, see the “References” chapter on page 58.

An image taken by the detector is stored as a 2D plane in a 3D dataset in an HDF5 file. The images are added to the dataset along the slowest changing dimension, which represents time. Indices of the images added to the same file are monotonically increasing and the value `mod(<image index>, <number of the HDF5 files>)` is constant. For example, as shown in Figure 6 every fourth image is stored in dataset A starting with the first image, every fourth image is stored in dataset B starting with the second image, and so on.

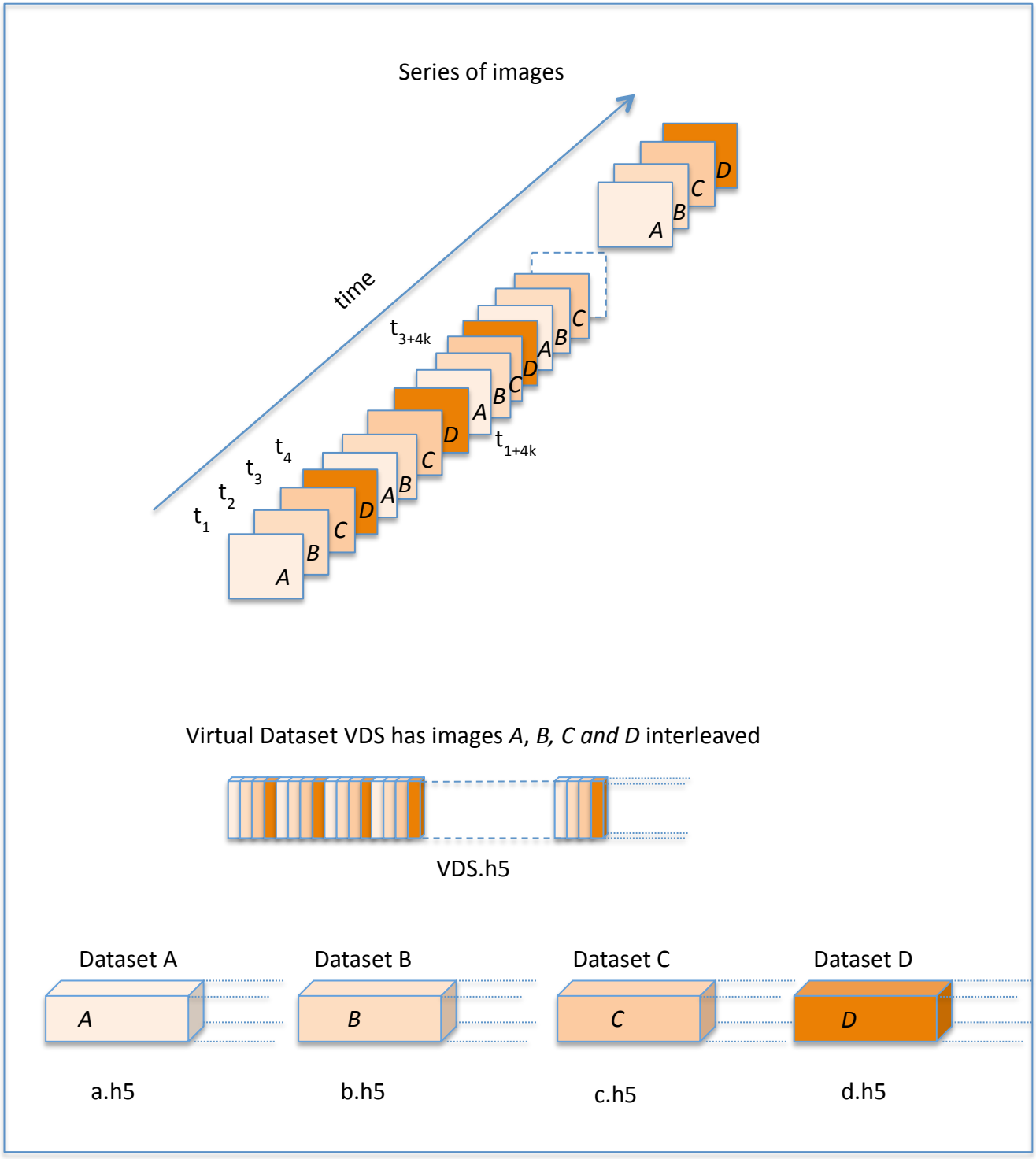


**Figure 6: Images are interleaved in the virtual dataset**

Each of the 3D datasets contains images taken at the time steps indexed with the fixed stride and a different offset.

# 4 HDF5 Virtual Dataset Requirements and Constraints

Based on the use cases above and some reasonable extrapolations from them, this section describes requirements and constraints for virtual datasets.

## 4.1 Requirements

1. There will be no changes to the HDF5 programming model when accessing a VDS.
    a. Dataset create, write, read, sub-set, query, and close operations will work for VDS.
    b. Virtual datasets will have a new storage type: 'virtual' (storage types for non-virtual datasets currently include chunked, external, contiguous, and compact).
2. The mapping between source dataset elements and VDS elements will be described at VDS creation time using the HDF5 dataspace selection mechanism by specifying elements in a source dataset and corresponding elements in the VDS.
    a. Current selection mechanism should be extended to allow unlimited hyperslab "count" values to accommodate unbounded selections in source datasets that can be extended or have unlimited dimensions.
    b. There will be a query mechanism to discover the mapping for a VDS.
3. A source dataset and VDS's datatypes must be "convertible" by the current algorithms available within the HDF5 library (which could include user-registered conversion routines).
4. A source dataset may have different rank and dimension sizes than the VDS. However, if a source dataset has an unlimited dimension, it must be the slowest-changing dimension, and the virtual dataset must be the same rank and have the same dimension as unlimited.
5. A source dataset may be a VDS itself.
6. A VDS may have its own "private" data. In other words, some elements of a VDS may not be mapped to source dataset elements and may instead be stored locally to the VDS.
7. A filter can be applied to VDS "private" data (for example, to compress it), although this will require that the VDS private data be stored in chunked layout.
8. There will be a mechanism in the HDF5 Library to search for source files at run time (similar to external links).
9. There will be a "printf-like" capability to generate source dataset and file names based on some or all of an element's coordinates in the VDS, enabling a dynamic mapping of source datasets into a VDS at runtime. If a virtual dataset has an unlimited dimension that is not the slowest changing dimension, it must use printf formatting for mapping to source datasets.
10. There will be a mechanism within the HDF5 Library to validate a mapping when a VDS is created. For example, selections in VDS mapping cannot overlap one another including unbounded selections.
11. If there is an element of a VDS that does not have a corresponding element in a source dataset or private VDS data, the element will be filled with the VDS fill value for I/O operations.

12. There will be a mechanism to override a "source" dataset's fill value with the VDS fill value, when retrieving data from unallocated portions of a source dataset.
13. Source datasets may be written to with the SWMR feature, and the virtual dataset may be refreshed to reflect the current size and data of those datasets.

## 4.2 Constraints and Assumptions

The following constraints and assumption have been identified:

1. There will be no UUIDs to identify source datasets and files (although they could be added, if desired).
2. It will be the user's responsibility to maintain the consistency of a VDS. If source files are unavailable, the library will report an error or use fill values when doing I/O on the VDS elements mapped to the elements in the missing source files.
3. Since the source datasets may be out-of-sync with one another's dimension sizes, the VDS needs to present a coherent view of the data in the source datasets (possibly a restating of assumption 2, above).
4. Users should assume that source datasets will not shrink in size.

## 4.3 Derived Design Aspects

The following additional complexities were identified during feature design:

1. How should the HDF5 Library detect 'gaps' in the sequence of files or datasets when printf formatting is used to identify source data for a virtual dataset (otherwise, the library could keep looking for the 'next' defined file or dataset indefinitely)?
   - Solution: Add a dataset access property for virtual datasets that specifies the largest 'gap' in the sequence of printf formatting to allow, defaulting to something sensible like '10'. This will allow applications to control the behavior, but put some upper limit in by default. This new property is described in a section below (link: H5Pset_virtual_printf_gap).
2. What should the HDF5 Library report for the size of a virtual dataset when it has an unlimited dimension(s) (since the underlying source datasets are could be different lengths along that dimension)?
   - Solution: Add a dataset access property for virtual datasets that specifies whether to take the largest source dataset dimension or the smallest, defaulting to the largest. This will allow applications to choose, but default to including all the possible data. This new property is described in a section below (link: H5Pset_virtual_dataspace_bounds).
3. How should the HDF5 Library handle source and virtual datasets located in different directories?
   - Solution: The HDF5 Library should allow for alternate paths to source datasets or files to be specified with both an environment variable and a data access property at dataset open time. This new property is described in a section below (link: H5Pset_virtual_source_path).

# 5 VDS Programming Model

This section describes the programming model for creating and accessing HDF5 virtual datasets.

The model is similar to the programming model for a regular HDF5 dataset and is summarized below. Please notice an additional step 2.d that is required for creating a VDS. The steps 2.a – 2.e are described in detail in the sub-sections below.

Programming model for a VDS:

1. Create datasets that comprise the VDS (the source datasets) (optional)
2. Create the VDS
    a. Define a datatype
    b. Define a dataspace
    c. Define the dataset creation property list
    d. Map elements from the source datasets to the elements of the VDS
        i. Iterate over the source datasets:
            1. Select elements in the source dataset (source selection)
            2. Select elements in the virtual dataset (destination selection)
            3. Map destination selections to source selections with a dataset creation property list call
        ii. End iteration
    e. Call `H5Dcreate` using the properties defined above
3. Access the VDS as a regular HDF5 dataset
4. Close the VDS when finished

## 5.1 Create Datasets that Comprise the VDS (Optional)

Users should follow the current HDF5 data model to create a source dataset if the dataset uses one of the current storage layouts (contiguous, chunked, compact, external) or the programming model for a VDS if a source dataset is a virtual dataset.

This step is optional. If a source dataset does not exist, users can specify the behavior by setting a special property: fill values may be used when performing I/O on a VDS, or I/O operations fail with appropriate error messages.

## 5.2 Create the Virtual Dataset

### 5.2.1 Define the Datatype

A virtual dataset may have any datatype, but it should be “compatible” with the datatypes of the source datasets. Two HDF5 datatypes are called compatible if the HDF5 Library can convert an element of one datatype to an element with another datatype. An example is, an element with an integer datatype can be converted to an element with a floating point datatype.

### 5.2.2 Define the Dataspace

The dataspace of a virtual dataset is defined in the same way as the dataspace for a regular dataset. The VDS's rank and dimension sizes do not depend on the dimensionality of the source datasets. For example, the `H5Screate_simple` function can be used to describe the VDS's dataspace as shown below.

```
vspace_id = H5Screate_simple(vRANK, vdims, vmaxdims);
```

A VDS may have fixed or unlimited dimensions.

### 5.2.3 Define the Creation Property

To define the creation property for a virtual dataset, an application must use a new API routine to indicate that the dataset will be a virtual dataset as shown below:

```
dcpl_id = H5Pcreate (H5P_DATASET_CREATE);
H5Pset_virtual(dcpl_id);
```

Other creation properties can be applied if the VDS will have "private" data. For example, contiguous, chunked, external or compact storage layouts can be used for storing VDS data that is not part of the source datasets. A compression method can be applied to the VDS's "private" data that is different from the compression methods used with the source datasets.

### 5.2.4 Map Elements from the Source Datasets to the Virtual Dataset

If the source and virtual datasets have fixed-size dimensions, one can use the current HDF5 hyperslab selection mechanisms to select elements for the mapping. As in the case of partial I/O, the number of elements selected in the source dataset for mapping must be the same as the elements selected in the virtual dataset.

The current hyperslab selection mechanism cannot be used if the dimensions of the source or virtual datasets are unlimited (as in several of the use cases described above) or repetition of a more sophisticated pattern is desired.

This section introduces an "unlimited" selection that overcomes a limitation of the current selection mechanism: a hyperslab pattern can be repeated only a finite number of times along each dimension (see the `count` parameter in the `H5Sselect_hyperslab` function).

#### *5.2.4.1 Unlimited Selections*

To create an unlimited selection one should use the following model:

1. Define a dataspace using the `H5Screate_simple` function. Two restrictions are applied:
    a. The rank of the dataspace should be the same as the rank of the dataset to which the unlimited selection will be applied (see the `H5Pset_virtual_mapping` function below).
    b. The maximum dimensions should be set to H5S_UNLIMITED in the same dimension as desired for unlimited selections:

```
hsize_t dims[RANK] = {10, 10, 1};
hsize_t max_dims[RANK] = {10, 10, H5S_UNLIMITED};
space_id = H5Screate_simple(RANK, dims, max_dims);
```

2. Select an unlimited region in the dataspace as shown:

```
hsize_t start[RANK] = {0, 0, 0};
hsize_t stride[RANK] = {0, 0, 1};
hsize_t block[RANK] = {10, 10, 1};
hsize_t count[RANK] = {1, 1, H5S_UNLIMITED};
H5Sselect_hyperslab(space_id, H5S_SELECT_SET, start, stride, block,
      count);
```

Setting `count[2]` to `H5S_UNLIMITED` indicates that the selection is unlimited. Although it is technically feasible to allow more complicated operations between selections that have unlimited count values, that capability is out of scope for this project, and the `'op'` parameter to `H5Sselect_hyperslab` must always be `H5S_SELECT_SET` when an unlimited count value is used.

#### *5.2.4.2 Select Elements in a Source and Virtual Dataset for Mapping*

The selection mechanism described above should be used to select the elements for mapping in both source and virtual datasets. If the mapping requires multiple source files, the selection in a source dataset may be a fixed-size selection, and the `printf` form of mapping should be used (described below).

#### *5.2.4.3 Mapping Elements*

After selections are defined on the source and virtual datasets, the mapping is accomplished by calling the `H5Pset_virtual_mapping` function:

```
status = H5Pset_virtual_mapping(dcpl_id, vspace_id, src_file_name, src_dset_name,
src_space_id);
```

*Parameters:*

`dcpl_id` - The identifier of the dataset creation property list that will be used when creating the virtual dataset.

`vspace_id` - The dataspace identifier with the selection within the virtual dataset applied, possibly an unlimited selection.

`src_file_name` - The name of the HDF5 file where the source dataset is located. The file might not exist yet. The name can be specified using a C-style `printf` statement. See the "C-style printf Formatting Notes " section on page 21 for more information.

`src_dset_name` - The path to the HDF5 dataset in the file specified by `src_file_name`. The dataset might not exist yet. The dataset name can be specified using a C-style `printf` statement, as described below.

`src_space_id` - The source dataset's dataspace identifier with a selection applied, possibly an unlimited selection.

*Returns*: negative on error, and positive on success.

When a selection with unlimited dimensions is used for the source dataset, the selection in the virtual dataset must also be an unlimited selection with the same number of unlimited dimensions. If fixed-size selections are used, the number of elements in the source dataset selection must be the same as the number of elements in the virtual dataset selection.

C-style printf Formatting Notes

C-style `printf` formatting allows a pattern to be specified in the name of a source file or dataset. Strings for the file and dataset names are treated as literals, except for the following substitutions:

"%%" - Replaced with a single '%' character.

"%<*d*>*b*" - Where <d> is the virtual dataset dimension axis (0-based) and 'b' indicates that the block count of the selection in that dimension should be used. The full expression (for example, "%0b") is replaced with a single numeric value when the mapping is evaluated at VDS access time. Example code for many source and virtual dataset mappings is available in the "Examples of Source to Virtual Dataset Mapping" chapter beginning on page 33.

If the `printf` form is used for the source file or dataset names, the selection in the source dataset's dataspace must be fixed-size.

#### *5.2.4.4 Create Virtual Dataset*

After all of the required elements are mapped from the source datasets to the virtual dataset, the virtual dataset can be created as every dataset is created:

```
vdset_id = H5Dcreate(loc_id, VDS_name, datatype, vspace_id, lcpl_id,
      dcpl_id, dapl_id);
```

If the mapping of source datasets to the virtual dataset's dataspace does not entirely cover the virtual dataset's dataspace elements, the virtual dataset will have "private" data elements for those elements. These elements are stored in the virtual dataset's file as any other dataset would be and are accessed transparently along with the source dataset elements with H5Dread and H5Dwrite. It is strongly recommended that if a virtual dataset has private data elements, chunked storage should be used for its storage layout to take advantage of the sparse data storage available with chunked

datasets. Chunked storage is required for virtual datasets with unlimited dimensions or that have compression filters applied.

## 5.3 Performing I/O on a Virtual Dataset

Once a virtual dataset has been created, it may be accessed as with any other HDF5 dataset using the H5Dread or H5Dwrite API calls. Elements mapped to source datasets are transparently accessed from those datasets and private elements for the virtual dataset are accessed from the virtual dataset's storage without further actions from the user application. Currently, there are no new data transfer properties defined for virtual datasets, but they may be defined in the future.

## 5.4 Closing a Virtual Dataset

Closing a virtual dataset occurs with H5Dclose as with any other HDF5 dataset. Any cached information about the source datasets is released transparently to the application.

## 5.5 Opening a Virtual Dataset

Once a virtual dataset has been created, it may be opened as with any other HDF5 dataset using the H5Dopen API call. The following dataset access properties may be set when opening (or creating) a virtual dataset to control various aspects of library behavior.

### 5.5.1 Choose the Dataspace Size Reported for Virtual Datasets

Virtual datasets can rely on multiple source datasets that may have different dimension sizes if they have unlimited dimensions (for example, see examples 3-7 in the "Examples of Source to Virtual Dataset Mapping" chapter beginning on page 33). By default, the dimension sizes of the virtual dataset reported with the H5Dget_space API routine will be the maximum of all the underlying source datasets' dimensions. If the application would prefer to have H5Dget_space report the minimum of all underlying source datasets' dimensions, the following property can be set on a dataset access property list:

```
status = H5Pset_virtual_dataspace_bounds(dapl_id, bounds_option);
```

*Parameters:*

`dapl_id` - The identifier for the dataset access property list for the VDS.

`bounds_option` - This parameter can be either H5D_VDS_MAX or H5D_VDS_MIN to choose the maximum of all underlying source datasets' dimensions or the minimum, respectively, when a virtual dataset's dimensions are queried with H5Dget_space.

*Returns*: negative on error, and positive on success.

### 5.5.2 Choose the Gap Size for printf-Formatted Source Dataset or File Names

Virtual datasets can rely on source datasets which have names that are determined by a printf-formatted string when the dataset is created. However, when those datasets or files need to be accessed later, some method must be used to determine that there are no more datasets or files in the sequence generated by the formatting. By default, the library will search for up to 10 more matches to the printf-formatting pattern after the last match of the dataset or file pattern. The number of matches can be controlled with the following property set on a dataset access property list:

```
status = H5Pset_virtual_printf_gap(dapl_id, gap_size);
```

*Parameters:*

`dapl_id` - The identifier for the dataset access property list for the VDS.

`gap_size` - This parameter indicates the number of printf-formatted datasets or files which can be missing before the library stops searching for more datasets or files. A value of 0 indicates that no gaps are tolerated. The default value is a gap size of 10.

*Returns*: negative on error, and positive on success.

### 5.5.3 Choose the Path to Search for Locating Source Datasets or Files

Virtual datasets can rely on source datasets whose containing files may not be located in the same directory as the virtual dataset. A path to locate files for source datasets may be specified in one of two ways: an environment variable (“HDF5_VDS_SOURCE_PATH”) or with the following property set on a dataset access property list:

```
status = H5Pset_virtual_source_path(dapl_id, source_path);
```

*Parameters:*

`dapl_id` - The identifier for the dataset access property list for the VDS.

`source_path` - This parameter is a string containing a colon-separated list of paths to search for files containing source datasets. The default value is the null string (“”). The paths specified by the “HDF5_VDS_SOURCE_PATH” environment variable are searched before the paths specified by this property.

*Returns*: negative on error, and positive on success.

# 6 Use Case Implementations

This section will provide pseudo-code examples to show how the programming model introduced in the "VDS Programming Model" section on page 18 can be applied to create a VDS for the use cases described in the "VDS Use Cases" section beginning on page 7.

## 6.1 The Excalibur Use Case

For this use case, source datasets do not exist at the time the VDS is created. The source HDF5 files and datasets will be written by independent processes and will be available for applications that access the VDS.

The sample code below starts with step 2 of the programming model - defining creation properties and a dataspace for the VDS. Next the code will select elements in each of the source datasets and the corresponding elements in the VDS. Finally, the code shows the mapping of elements before creating the VDS.

```
/* Set creation property to be used in mapping definition */
vdcpl_ id = H5Pcreate (H5P_DATASET_CREATE);
H5Pset_virtual(vdcpl_id);
/* Create a dataspace for virtual dataset VDS */
N = 3 * (k + n); /* See Figure in section 3.1 for the dimensions of full image */
vdims[] = {1, N, M};
vmaxdims = {H5S_UNLIMITED, N, M};
vspace_id = H5Screate_simple(3, vdims, vmaxdims);
/* Start the loop over the source datasets to define mapping */
for each DATASET in (A, B, C, D, E, F) {
       /* First, describe dataspace for a source DATASET */
      src_dims[] = {1, i, M}; /* i is k for A, C, E and n for B, D, F */
      src_maxdims[] = {H5S_UNLIMITED, i, M}; /* i is k for A, C, E and n for B, D,
F */
      src_dataspace_id = H5Screate_simple(3, src_dims, src_maxdims);
       /* Define unlimited selection for the source dataset */
       start = {0, 0, 0};
       stride = {1, 1, 1};
       count = {H5S_UNLIMITED, 1, 1};
       block = {1, i, M}; /* i is k for A, C, E and n for B, D, F */
       H5Sselect_hyperslab(src_dataspace_id, H5S_SELECT_SET, start, stride, count,
                block);
       /* Select elements in VDS that will be mapped with the elements in DATASET */
       vstart = {0, p, 0}; /*The value of p is
                              0 for A,
```

```
                                n for B,
                                n+k for C,
                                n+2k for D,
                                2n+2k for E,
                                3k+2n for F */
        vstride = {1, 1, 1};
        vcount = {H5S_UNLIMITED, 1, 1};
        vblock[] = {1, i, M}; /* i is k for A, C, E and n for B, D, F */
        H5Sselect_template(vspace_id, H5S_SELECT_SET, vstart, vstride,
                  vcount, vblock);
        /* Map selections */
        H5Pset_virtual_mapping(vdcpl_id, vspace_id, FILE, DATASET, src_dataspace_id);
} /* End loop over the source datasets */
/* Create virtual dataset using regular HDF5 call */
VDS_id = H5Dcreate(…, “VDS”,…, vspace_id, …, vdcpl_id,…);
```

## 6.2 Generalized Excalibur Use Case (Dealing with Gaps)

The only difference between the Excalibur use case and the Dual PCO Edge and Eiger use cases is the selection in the VDS dataspace. See the “VDS Use Cases” section beginning on page 7 for more information.

```
        vstart = {0, n_SUB-IMAGE, m_SUB-IMAGE};
```

Values $n_{SUB\text{-}IMAGE}$ and $m_{SUB\text{-}IMAGE}$ indicate the offset (in the number of elements to skip) of sub-images in the full image. For example, $n_D$ and $m_D$ are offsets for sub-image *D* as shown in Figure 7 below.

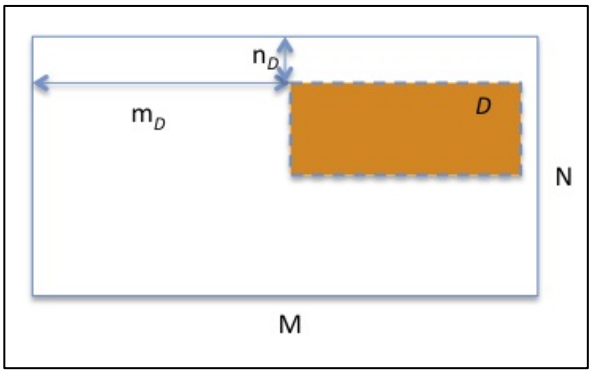


**Figure 7: Offsets for sub-image D in the full image**

The programming model can easily accommodate gaps as shown in Figure 4.

This use case is the same as the base Excalibur use case described in the section above (see page 25).

## 6.3 Eiger Use Case

The Eiger use case is different from the use cases discussed above. There is no template selection in the source datasets. The whole dataspace for each dataset in a files `f-i.h5` will be selected. The loop over the source datasets corresponds to the call to `H5Pset_virtual_mapping` using "`printf`" formatting for file names. For more information, see the "C-style printf Formatting Notes" section on page 21.

```
/* Set creation property to be used in mapping definition */
vdcpl_ id = H5Pcreate (H5P_DATASET_CREATE);
H5Pset_virtual(vdcpl_id);
/* Create a dataspace for virtual dataset VDS */
vdims[] = {M, N, K};
vmaxdims = {H5S_UNLIMITED, N, K};
vspace_id = H5Screate_simple(3, vdims, vmaxdims);
/* No loop over the source datasets to define mapping; we can use the
   same source dataspace as shown on the next two lines. The dataspace contains M
   images, and the size of each image is NxK */
dims[] = {M, N, K};
src_id = H5Screate_simple(3, dims, NULL);
/* Select elements in VDS that will be mapped with the elements in DATASET */
vstart = {0, 0, 0};
vstride = {M, 1, 1};
vcount = {H5S_UNLIMITED, 1, 1};
vblock = {M, N, K};
H5Sselect_hyperslab(vspace_id, H5S_SELECT_SET, vstart, vstride, vcount, vblock);
```

```
/* Map selections */
H5Pset_virtual_mapping(vdcpl_id, vspace_id, “f-%0b.h5”, “/A”, src_id);
/* Finally create virtual dataset using regular HDF5 call */
VDS_id = H5Dcreate(…, “VDS”,…, vspace_id, …, vdcpl_id,…);
```

Notice that the files “f-%0b.h5” and the datasets in them may not exist.

## 6.4 Percival Frame Builder Use Case

This example will use offset and stride when selecting templates in the VDS dataspace. The rest of the example follows the Excalibur use case. See page 25.

```
/* Set creation property to be used in mapping definition */
vdcpl_ id = H5Pcreate (H5P_DATASET_CREATE);
H5Pset_virtual(vdcpl_id);
/* Second, create a dataspace for virtual dataset VDS */
vdims[] = {1, N, M};
vmaxdims = {H5S_UNLIMITED, N, M};
vspace_id = H5Screate_simple(3, vdims, vmaxdims);
/* Selection for all source datasets will be the same */
dims [] = {1, N, M};
maxdims [] = {H5S_UNLIMITED, N, M};
src_id = H5Screate_simple(3, dims_DATASET, maxdims);
/* Select elements in the source dataset */
start = {0, 0, 0};
stride = {1, 1, 1};
count = {H5S_UNLIMITED, 1, 1};
block = {1, N, M};
H5Sselect_hyperslab(src_id, H5S_SELECT_SET, start, stride, count, block);
/* Start the loop over the source datasets to define mapping */
for each DATASET in (A, B, C, D,) {
      /* Select elements in VDS that will be mapped with the elements in DATASET */
      vstart = {p, 0, 0}; /*The value of p is
                           0 for A,
                           1 for B,
                           2 for C,
                           3 for D */
      vstride = {4, 1, 1};
      vcount = {H5S_UNLIMITED, 1, 1};
      vblock = {1, N, M};
```

```
        /* Select elements in VDS */
        H5Sselect_hyperslab(vspace_id,  H5S_SELECT_SET,  vstart,  vstride,  vcount,
vblock);
        /* Map selections */
        H5Pset_virtual_mapping(vdcpl_id, vspace_id, FILE, DATASET, src_id);
}
/* Create virtual dataset using regular HDF5 call */
VDS_id = H5Dcreate(…, “VDS”,…, vspace_id, …, vdcpl_id,…);
```

# 7 Implementation Details

This section describes the changes to major components of the HDF5 software package needed for implementing the virtual dataset capability.

## 7.1 HDF5 C Library

This section describes the major modifications to the HDF5 C Library required for the implementation of the VDS feature.

### 7.1.1 Adding the VDS Storage Layout

The core of the virtual dataset feature will be in the implementation of the VDS dataset layout type. This consists of adding callback functions for the VDS `H5D_layout_t` structure and other auxiliary functions.

We will have to create a structure for the VDS layout type. This new layout type will allow the library to track persistent information about the VDS storage for the dataset that is not contained in the layout info message or that should be cached for efficiency. We will also need to create routines to initialize this structure when a dataset is opened and a routine to initialize the layout structure when the dataset is created.

We will have to create a layout callback to inform the library if space has been allocated for the dataset yet. We will need to decide whether to examine the source datasets to see if they have been allocated, or simply return “yes”, at least until the private data feature is implemented.

### 7.1.2 Modifications to I/O

We will need to write a function to initialize structures used during an I/O operation. This routine may need to make decisions about how to proceed with I/O and cache the results in the I/O structure, or it may defer those decisions to the read/write callbacks.

The translation between selections in the VDS and selections in the source datasets is one major task that could be done when we make decisions about how to proceed with I/O or later in read/write callbacks. The translation will use HDF5 hyperslab routines in order to transform the selections as necessary.

The other major task that could be performed in either place is the resolution of file and dataset names for each source dataset in the selection. This includes resolving the “printf” style names to strings that can be used to open the source dataset directly.

We will need to prototype solutions before finalizing the design of VDS and source datasets selections translations, and names resolutions.

Adding the VDS feature to HDF5 will require changes for parallel I/O. The H5Dread and H5Dwrite routines, used to perform actual I/O, will also need to determine if the I/O is a collective MPI operation, and if so, change it to independent, while adding a barrier to make it behave like a collective operation.

### 7.1.3 VDS, Source Dataset, and Fill Values

Special care should be taken to work with HDF5 fill values. The read routine will need to, when a selected part of the VDS does not have an associated source dataset, propagate the fill value to the relevant places in the application's read buffer.

### 7.1.4 VDS Flushing

We will need to write a VDS flush routine which will instruct all source datasets to flush; we will also need to write a routine that will release any resources used during I/O.

### 7.1.5 Modifications to the HDF5 Selection Mechanism

Changes will be required to the existing HDF5 hyperslab routines to support the “unlimited” selections for the VDS feature. Hyperslabs with unlimited selections may not be combined with any other selections. This will simplify the process of creating a hyperslab, as the library need only record the initial description of the hyperslab (`start/stride/count/block`). Other hyperslab functions such as the iterator functions will need to be updated to handle unlimited selections by clipping the selection to the extent of the dataspace. We will also need to update the serialized form of a dataspace to account for unlimited selections.

### 7.1.6 Modifications to the Dataset Layout Object Header Message

The dataset layout object header message will need to be updated to be able to describe virtual datasets. This includes both the in-memory structure and the serialized format written to the file. These will need to include the list of selections and the paths to source datasets required to construct the VDS. This work will use the updated hyperslab routines for serializing selections described above. We will also need to update region references to be able to handle unlimited selections, though this work should be minimal, as it should just use the same hyperslab routines.

### 7.1.7 VDS APIs and Changes to Existing APIs

Finally, the new API functions will need to be written and others will need to be updated to reflect the changes specified in this document.

The public API functions themselves will mostly check the parameters for validity and translate them into a form the internal library handles before calling internal functions.

The following are some other API-related changes that will need to be made:

- Add internal functions to handle the VDS layout on dataset creation property lists
- Add options to control the behavior of `H5Dget_storage_size`
- Update the search path for source files to dataset access property lists
- Update `H5Dget_space` to handle virtual datasets by checking to see which source datasets have been allocated and what sizes they are
- Update `H5Ocopy` to properly copy virtual datasets

## 7.2 HDF5 Library Testing

The bulk of the regression testing will test the virtual dataset feature directly. These tests will start from simple cases such as a single non-unlimited source dataset and build to complex configurations. Corner cases such as a source dataset with a `NULL` dataspace will also be tested. These tests will make sure that invalid inputs such as overlapping source datasets are rejected appropriately.

The following other tests will also need to be written:

- Tests similar to the hyperslab selection code to test that unlimited selections are handled correctly by all of the internal selection routines including the routines to serialize and de-serialize the selection
- A test to verify that region references to unlimited selections are handled appropriately
- A test to make sure that virtual datasets can be accessed in SWMR mode correctly
- An acceptance test suite to make sure we correctly handle the customer's requirements. We will work closely with the customer to develop these tests to ensure that they closely reflect the use cases, and the customer will sign off on the test suite before the project is complete.
- A performance benchmark to ensure that the virtual dataset feature's performance in the customer's use cases adequately meets their needs.
- A performance regression test to ensure that the performance of the VDS feature does not degrade in the future.

## 7.3 HDF5 Command-line Tools and Testing

As the changes to the command-line tools for HDF5 (for example, h5dump and h5repack) to support virtual datasets have not yet been designed, implementation details are not yet available. As the design for the command-line tools is finalized, this section will be updated with information about the implementation and testing details.

## 7.4 HDF5 Language Wrappers and Testing

As with the command-line tools, the language wrappers have not yet been designed, and this section will be updated at that time.

# 8 Examples of Source to Virtual Dataset Mapping

The following code examples show various combinations of the possible source and virtual dataset mappings, and elaborate on `printf` usage.

## 8.1 Fixed-size Source and Fixed-size Virtual Dataset Mapping, Blocked Pattern in the Slowest Dimension of a Virtual Dataset

The figure below shows a mapping of two source datasets with fixed-size selections of the entire source dataset to a single virtual dataset and with each source dataset occupying a block along the virtual dataset's slowest changing dimension.

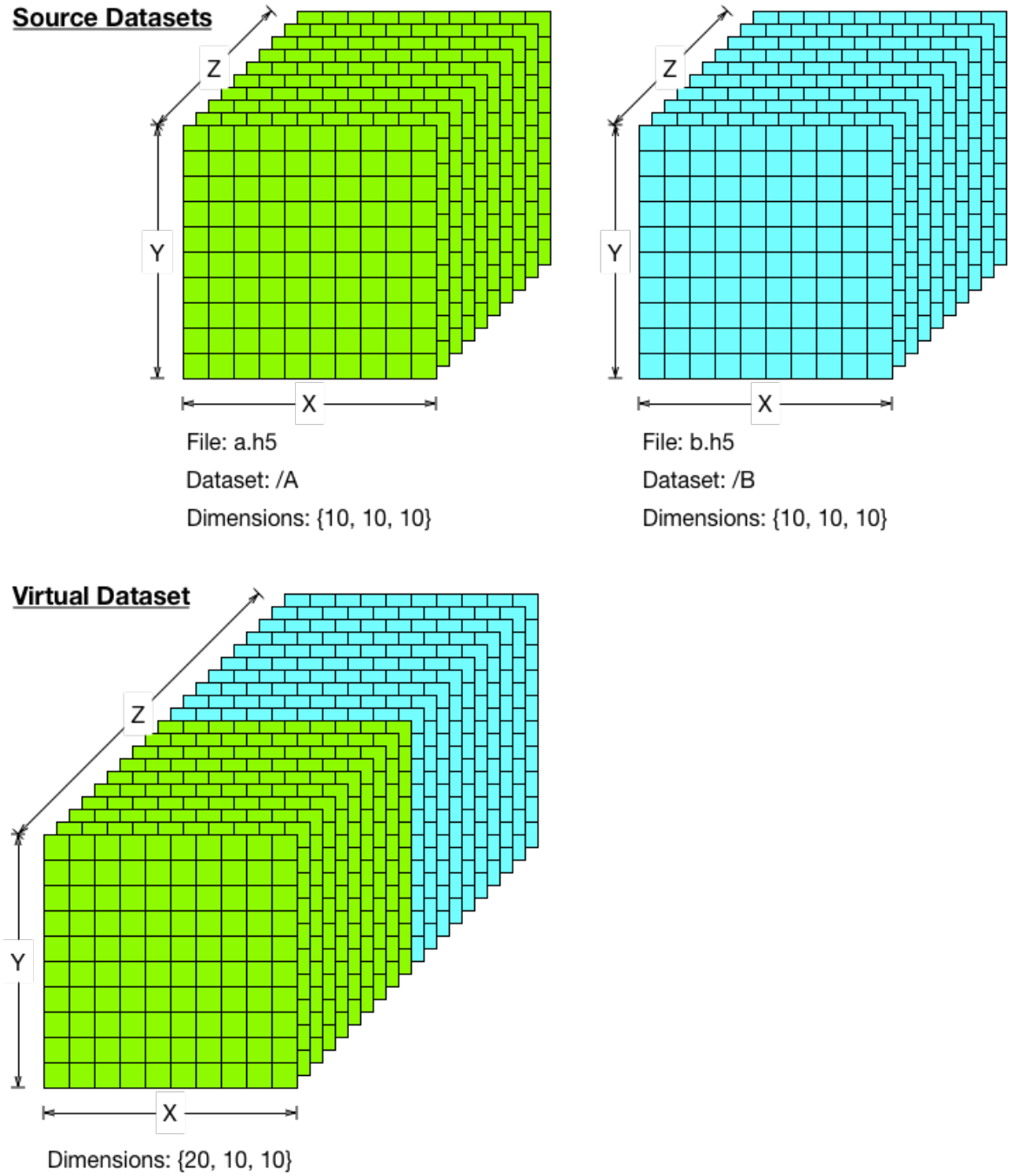


**Figure 8: Mapping fixed-size selections of two entire source datasets to blocks in the slowest changing dimension of virtual dataset**

The following pseudo-code shows how to create this mapping:

```
hsize_t src_dims[3], vds_dims[3];
hsize_t start[3], stride[3], block[3], count[3];
hid_t src_space_id, vds_space_id;
hid_t dcpl_id;

/* Create DCPL and set to virtual storage */
dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_virtual(dcpl_id);

/* Create source dataspace and set selection */
src_dims[3] = {/* Z: */ 10, /* Y: */ 10, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, NULL);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Create virtual dataspace and set first selection */
vds_dims[3] = {/* Z: */ 20, /* Y: */ 10, /* X: */ 10};
vds_space_id = H5Screate_simple(3, vds_dims, NULL);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between first source dataset and first block in virtual dataset */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/A”, src_space_id);

/* Set second selection in virtual dataset */
start[3] = {10, 0, 0};
```

```
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between second source dataset and second block in virtual dataset
*/
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “b.h5”, “/B”, src_space_id);

/* Close source dataspace */
H5Sclose(src_space_id);
```

## 8.2 Fixed-size Source and Fixed-size Virtual Dataset Mapping, Tiled Pattern in the Virtual Dataset, Partial Source Dataset Selected

The figure below shows a mapping of four source datasets with fixed-size selections of a portion of the source dataset to a single virtual dataset with each source dataset selection occupying a tiled portion of the virtual dataset.

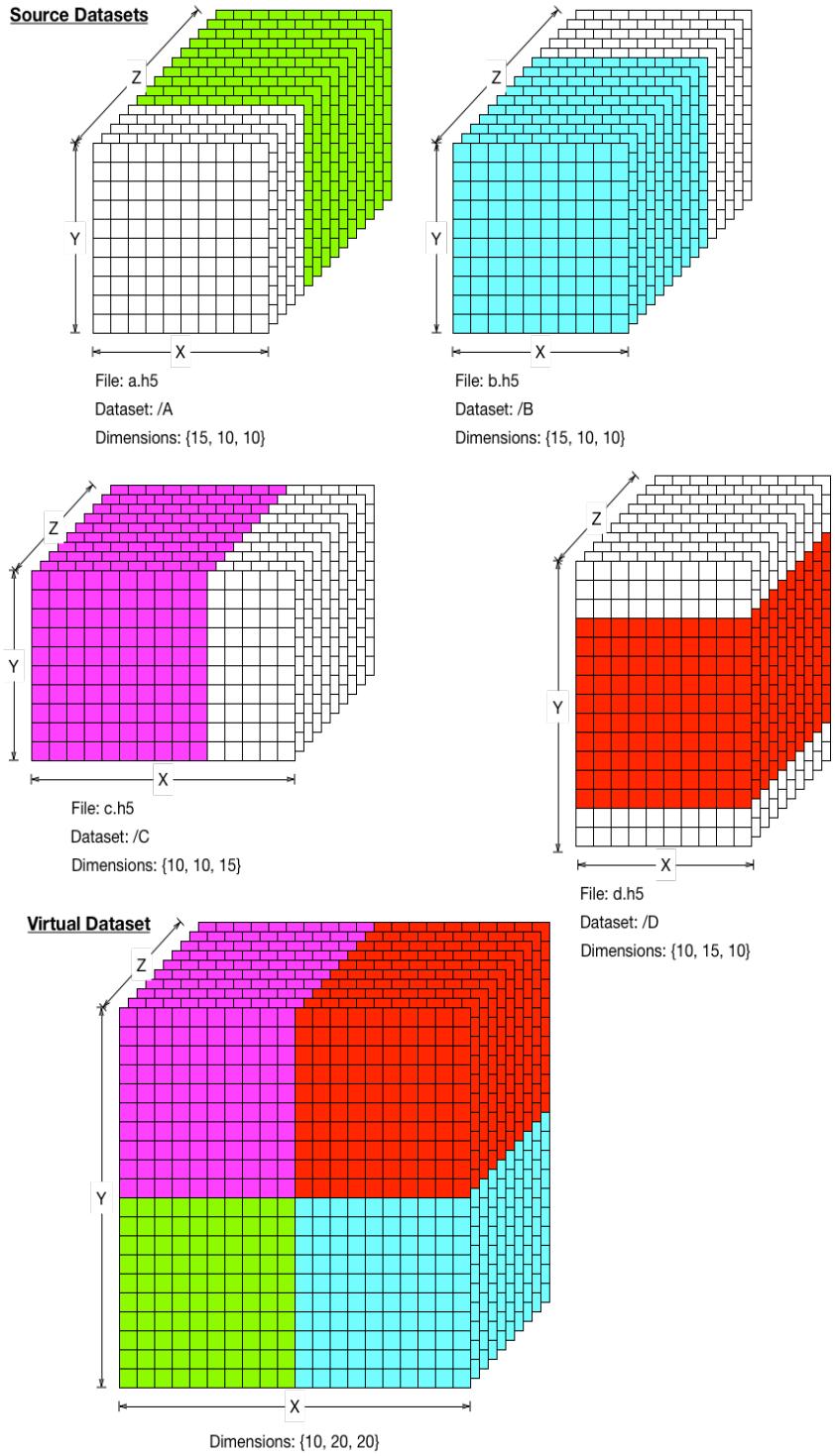


**Figure 9: Mapping fixed-size partial selections of four source datasets to tiles in a virtual dataset**

The following pseudo-code shows how to create this mapping:

```
hsize_t src_dims[3], vds_dims[3];
hsize_t start[3], stride[3], block[3], count[3];
hid_t src_space_id, vds_space_id;
hid_t dcpl_id;

/* Create DCPL and set to virtual storage */
dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_virtual(dcpl_id);

/* Create first source dataspace and set selection */
src_dims[3] = {/* Z: */ 15, /* Y: */ 10, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, NULL);
start[3] = {5, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Create virtual dataspace and set first selection */
vds_dims[3] = {/* Z: */ 10, /* Y: */ 20, /* X: */ 20};
vds_space_id = H5Screate_simple(3, vds_dims, NULL);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between first source dataset and first block in virtual dataset */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/A”, src_space_id);

/* Close first source dataspace */
```

```
H5Sclose(src_space_id);

/* Create second source dataspace and set selection */
src_dims[3] = {/* Z: */ 15, /* Y: */ 10, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, NULL);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set second selection in virtual dataset */
start[3] = {0, 0, 10};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between second source dataset and second block in virtual dataset
*/
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “b.h5”, “/B”, src_space_id);

/* Close second source dataspace */
H5Sclose(src_space_id);

/* Create third source dataspace and set selection */
src_dims[3] = {/* Z: */ 10, /* Y: */ 10, /* X: */ 15};
src_space_id = H5Screate_simple(3, src_dims, NULL);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
```

```
      count);

/* Set third selection in virtual dataset */
start[3] = {0, 10, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between third source dataset and third block in virtual dataset */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “c.h5”, “/C”, src_space_id);

/* Close third source dataspace */
H5Sclose(src_space_id);

/* Create fourth source dataspace and set selection */
src_dims[3] = {/* Z: */ 10, /* Y: */ 10, /* X: */ 15};
src_space_id = H5Screate_simple(3, src_dims, NULL);
start[3] = {0, 2, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set fourth selection in virtual dataset */
start[3] = {0, 10, 10};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {10, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);
```

```
/* Set mapping between fourth source dataset and fourth block in virtual dataset
*/
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “d.h5”, “/D”, src_space_id);


/* Close fourth source dataspace */
H5Sclose(src_space_id);
```

## 8.3 Unlimited-dimension Source and Unlimited-dimension Virtual Dataset Mapping, Tiled Pattern in the Virtual Dataset, Entire Source Dataset Selected

The figure below shows a mapping of four source datasets with unlimited selections of the entire source dataset to a single virtual dataset with unlimited dimensions with each source dataset selection occupying a tiled portion of the virtual dataset.

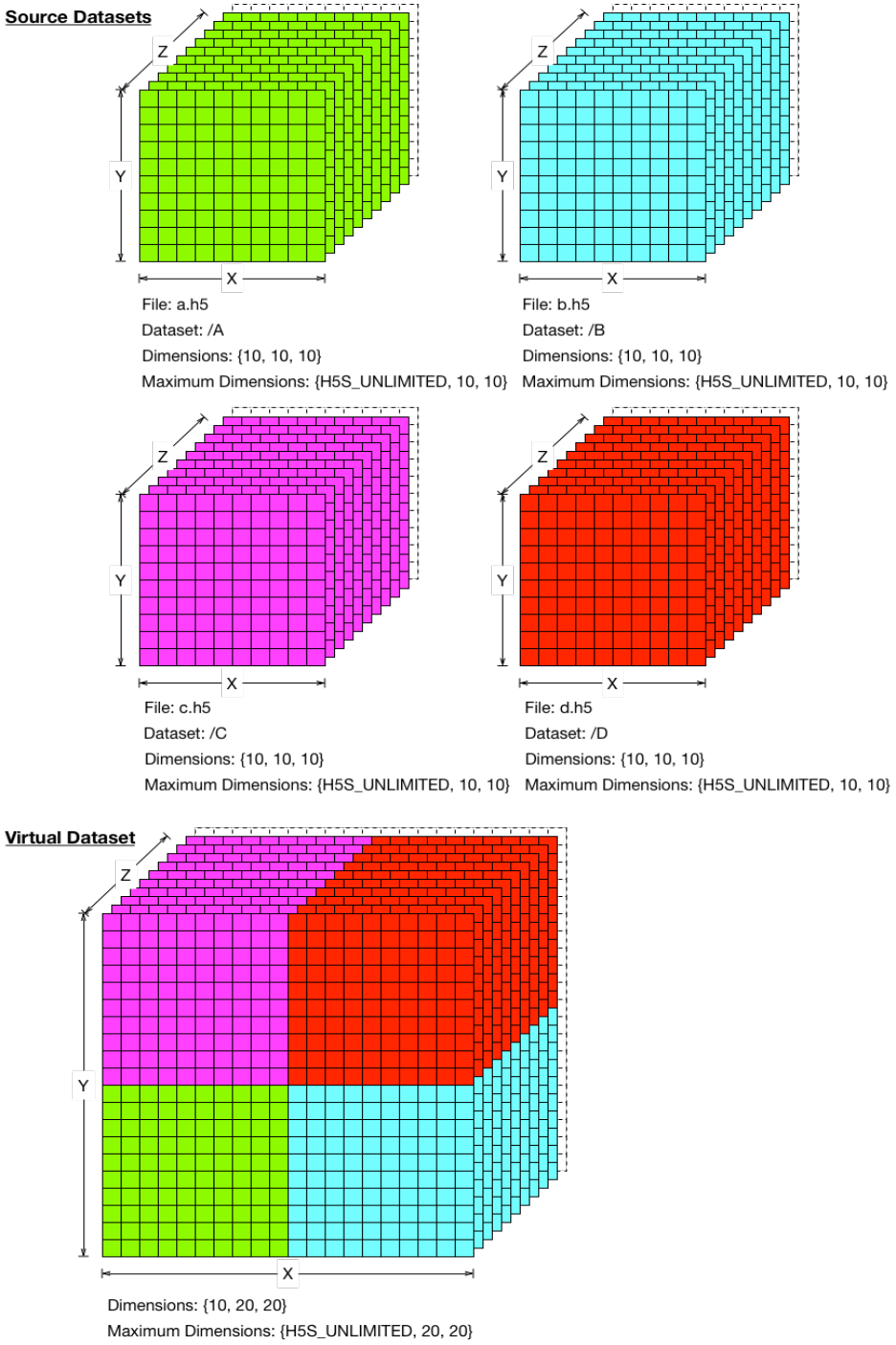


**Figure 10: Mapping unlimited full selections of four source datasets to tiles in a virtual dataset with an unlimited dimension**

The following pseudo-code shows how to create this mapping:

```
hsize_t src_dims[3], max_src_dims[3], vds_dims[3], max_vds_dims[3];
hsize_t start[3], stride[3], block[3], count[3];
hid_t src_space_id, vds_space_id;
hid_t dcpl_id;

/* Create DCPL and set to virtual storage */
dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_virtual(dcpl_id);

/* Create unlimited source dataspace and set selection */
src_dims[3] = {/* Z: */ 10, /* Y: */ 10, /* X: */ 10};
max_src_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 10, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, max_src_dims);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Create virtual dataspace and set first selection */
vds_dims[3] = {/* Z: */ 10, /* Y: */ 20, /* X: */ 20};
max_vds_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 20, /* X: */ 20};
vds_space_id = H5Screate_simple(3, vds_dims, max_vds_dims);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between first source dataset and first block in virtual dataset */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/A”, src_space_id);
```

```
/* Set second selection in virtual dataset */
start[3] = {0, 0, 10};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);
/* Set mapping between second source dataset and second block in virtual dataset
*/
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “b.h5”, “/B”, src_space_id);
/* Set third selection in virtual dataset */
start[3] = {0, 10, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);
/* Set mapping between third source dataset and third block in virtual dataset */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “c.h5”, “/C”, src_space_id);

/* Set fourth selection in virtual dataset */
start[3] = {0, 10, 10};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between fourth source dataset and fourth block in virtual dataset
*/
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “d.h5”, “/D”, src_space_id);

/* Close source dataspace */
H5Sclose(src_space_id);
```

## 8.4 Unlimited-dimension Source and Unlimited-dimension Virtual Dataset Mapping, Tiled and Interleaved Pattern in the Virtual Dataset, Partial and Strided Source Dataset Selected

The figure below shows a mapping of four source datasets with unlimited selections of partial and strided source dataset to a single virtual dataset with unlimited dimensions, with each source dataset selection occupying a tiled and interleaved portion of the virtual dataset.

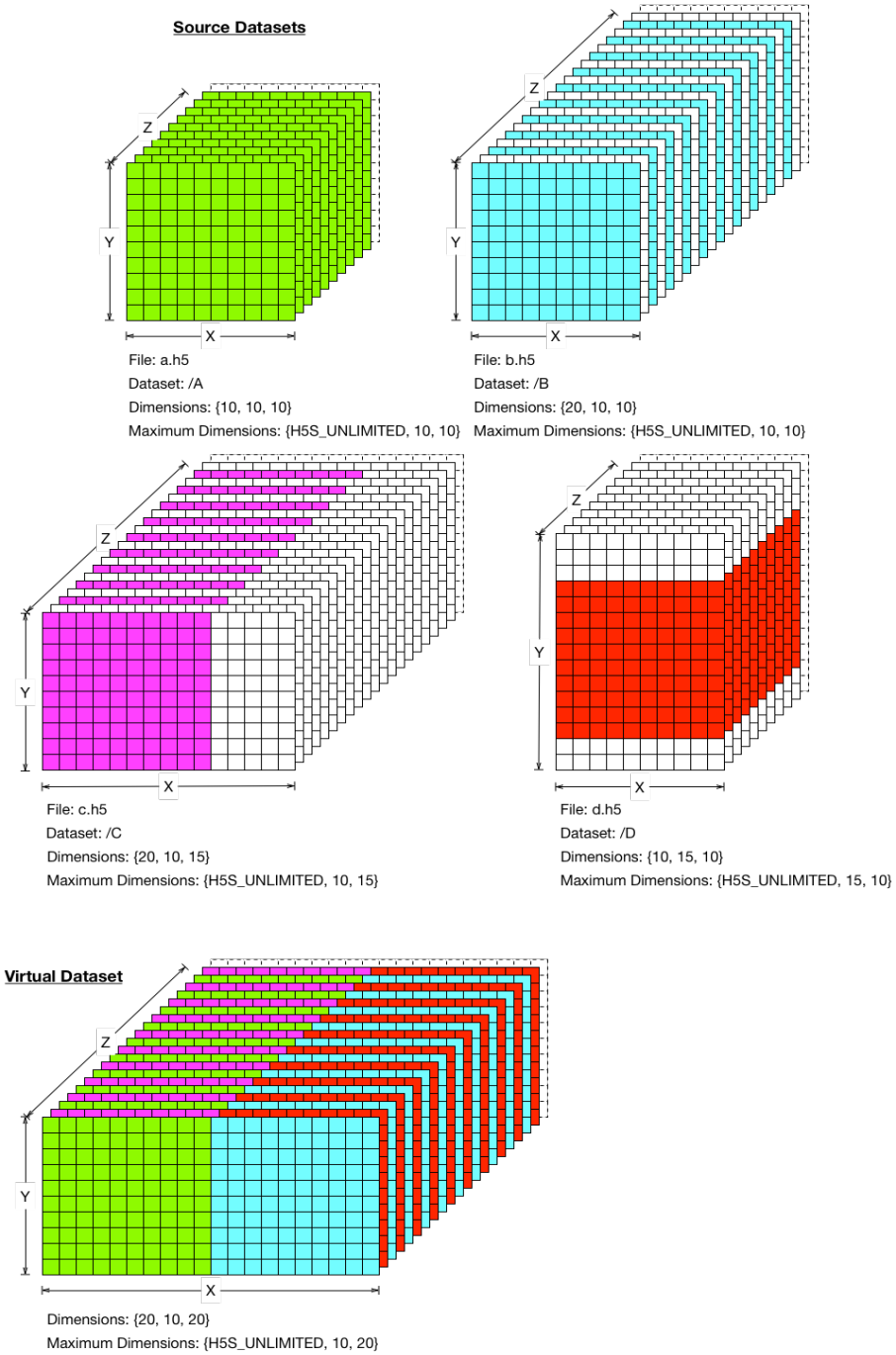


**Figure 11: Mapping unlimited partial and strided selections of four source datasets to strided tiles in virtual dataset with an unlimited dimension**

The following pseudo-code shows how to create this mapping:

```
hsize_t src_dims[3], max_src_dims[3], vds_dims[3], max_vds_dims[3];
hsize_t start[3], stride[3], block[3], count[3];
hid_t src_space_id, vds_space_id;
hid_t dcpl_id;

/* Create DCPL and set to virtual storage */
dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_virtual(dcpl_id);

/* Create first unlimited source dataspace and set selection */
src_dims[3] = {/* Z: */ 10, /* Y: */ 10, /* X: */ 10};
max_src_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 10, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, max_src_dims);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Create virtual dataspace and set first selection */
vds_dims[3] = {/* Z: */ 20, /* Y: */ 10, /* X: */ 20};
max_vds_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 10, /* X: */ 20};
vds_space_id = H5Screate_simple(3, vds_dims, max_vds_dims);
start[3] = {0, 0, 0};
stride[3] = {2, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between first source dataset and first unlimited, strided selection
in virtual dataset */
```

```
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/A”, src_space_id);

/* Close first source dataspace */
H5Sclose(src_space_id);

/* Create second unlimited source dataspace and set strided selection */
src_dims[3] = {/* Z: */ 20, /* Y: */ 10, /* X: */ 10};
max_src_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 10, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, max_src_dims);
start[3] = {0, 0, 0};
stride[3] = {2, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set second selection in virtual dataset */
start[3] = {0, 0, 10};
stride[3] = {2, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between second source dataset and second unlimited, strided
selection in virtual dataset */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “b.h5”, “/B”, src_space_id);

/* Close second source dataspace */
H5Sclose(src_space_id);

/* Create third unlimited source dataspace and set partial, strided selection */
src_dims[3] = {/* Z: */ 20, /* Y: */ 10, /* X: */ 15};
max_src_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 10, /* X: */ 15};
src_space_id = H5Screate_simple(3, src_dims, max_src_dims);
```

```
start[3] = {0, 0, 0};
stride[3] = {2, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set third selection in virtual dataset */
start[3] = {1, 0, 0};
stride[3] = {2, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between third source dataset and third unlimited, strided selection
in virtual dataset */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “c.h5”, “/C”, src_space_id);

/* Close third source dataspace */
H5Sclose(src_space_id);

/* Create fourth unlimited source dataspace and set partial selection */
src_dims[3] = {/* Z: */ 10, /* Y: */ 15, /* X: */ 10};
max_src_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 15, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, max_src_dims);
start[3] = {0, 2, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set fourth selection in virtual dataset */
start[3] = {1, 0, 10};
```

```
stride[3] = {2, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between fourth source dataset and fourth block in virtual dataset
*/
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “d.h5”, “/D”, src_space_id);

/* Close fourth source dataspace */
H5Sclose(src_space_id);
```

## 8.5 Fixed-dimension Full Source and Unlimited-dimension Virtual Dataset Mapping, Blocked Pattern in the Virtual Dataset, printf-named Source Datasets with an Unlimited Dimension

The figure below shows a mapping of a printf-named sequence of source datasets with fixed-dimensions and full selections to a single virtual dataset with unlimited dimensions with each source dataset selection occupying a sequentially blocked portion of the virtual dataset.

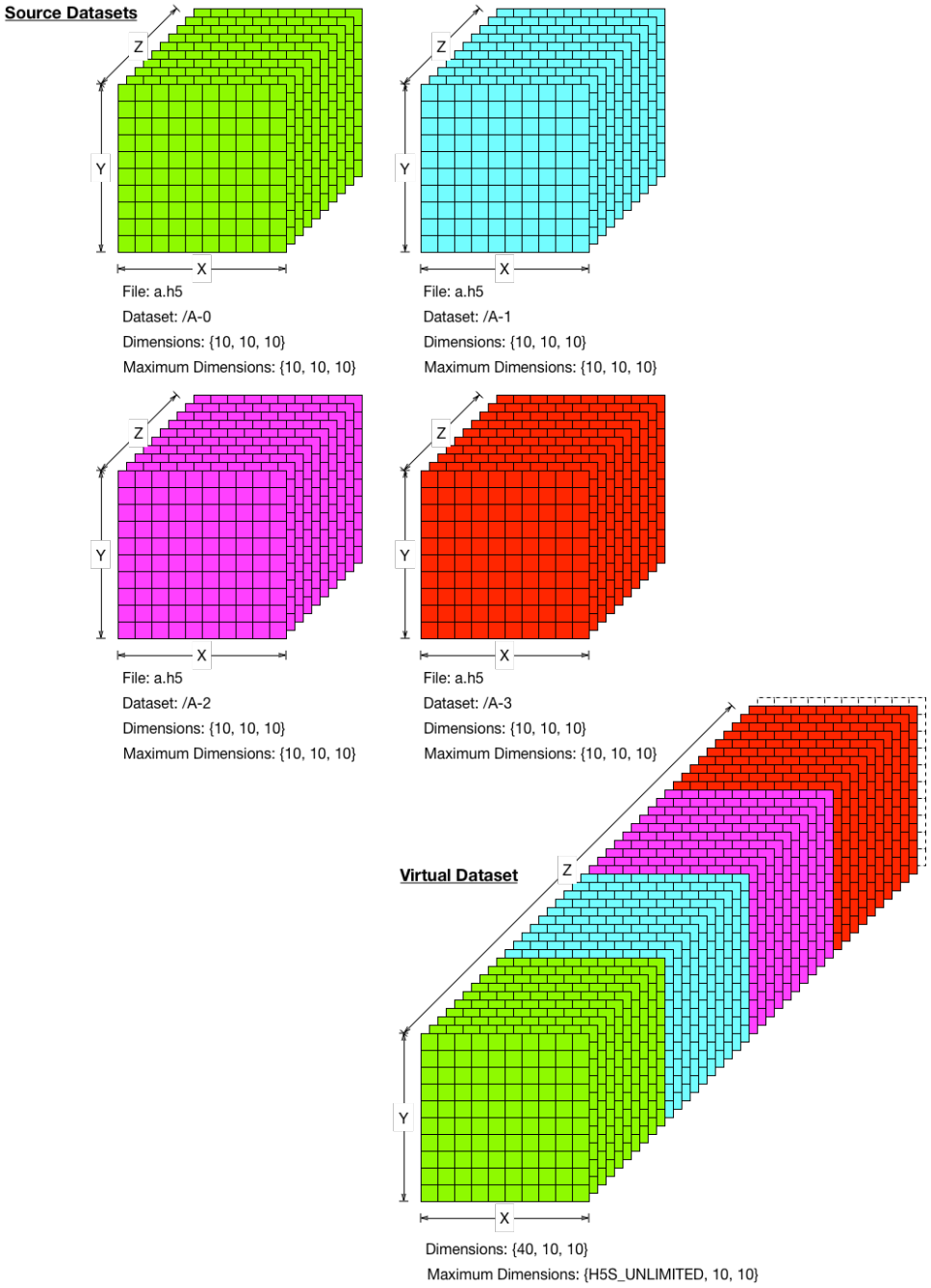


**Figure 12: Mapping full selections of four fixed-dimension source datasets chosen with printf-naming to blocks in a virtual dataset with an unlimited dimension**

The following pseudo-code shows how to create this mapping:

```
hsize_t src_dims[3], vds_dims[3], max_vds_dims[3];
hsize_t start[3], stride[3], block[3], count[3];
hid_t src_space_id, vds_space_id;
hid_t dcpl_id;

/* Create DCPL and set to virtual storage */
dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_virtual(dcpl_id);

/* Create source dataspace, defaults to “all” selection */
src_dims[3] = {/* Z: */ 10, /* Y: */ 10, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, NULL);

/* Create virtual dataspace and set first selection */
vds_dims[3] = {/* Z: */ 40, /* Y: */ 10, /* X: */ 10};
max_vds_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 10, /* X: */ 10};
vds_space_id = H5Screate_simple(3, vds_dims, max_vds_dims);
start[3] = {0, 0, 0};
stride[3] = {10, 10, 10};
block[3] = {10, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between first source dataset and first unlimited, strided selection
in virtual dataset, with printf-mapping */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/A-%0b”, src_space_id);

/* Close source dataspace */
H5Sclose(src_space_id);
```

## 8.6 Unlimited-dimension Full Source and Unlimited-Dimension Virtual Dataset Mapping, Tiled Pattern in the Virtual Dataset, printf-named Source Datasets with Non-unlimited Dimensions

The figure below shows a mapping of a printf-named array of source datasets with unlimited-dimensions and full selections to a single virtual dataset with unlimited dimensions with each source dataset selection occupying a sequentially tiled portion of the virtual dataset.

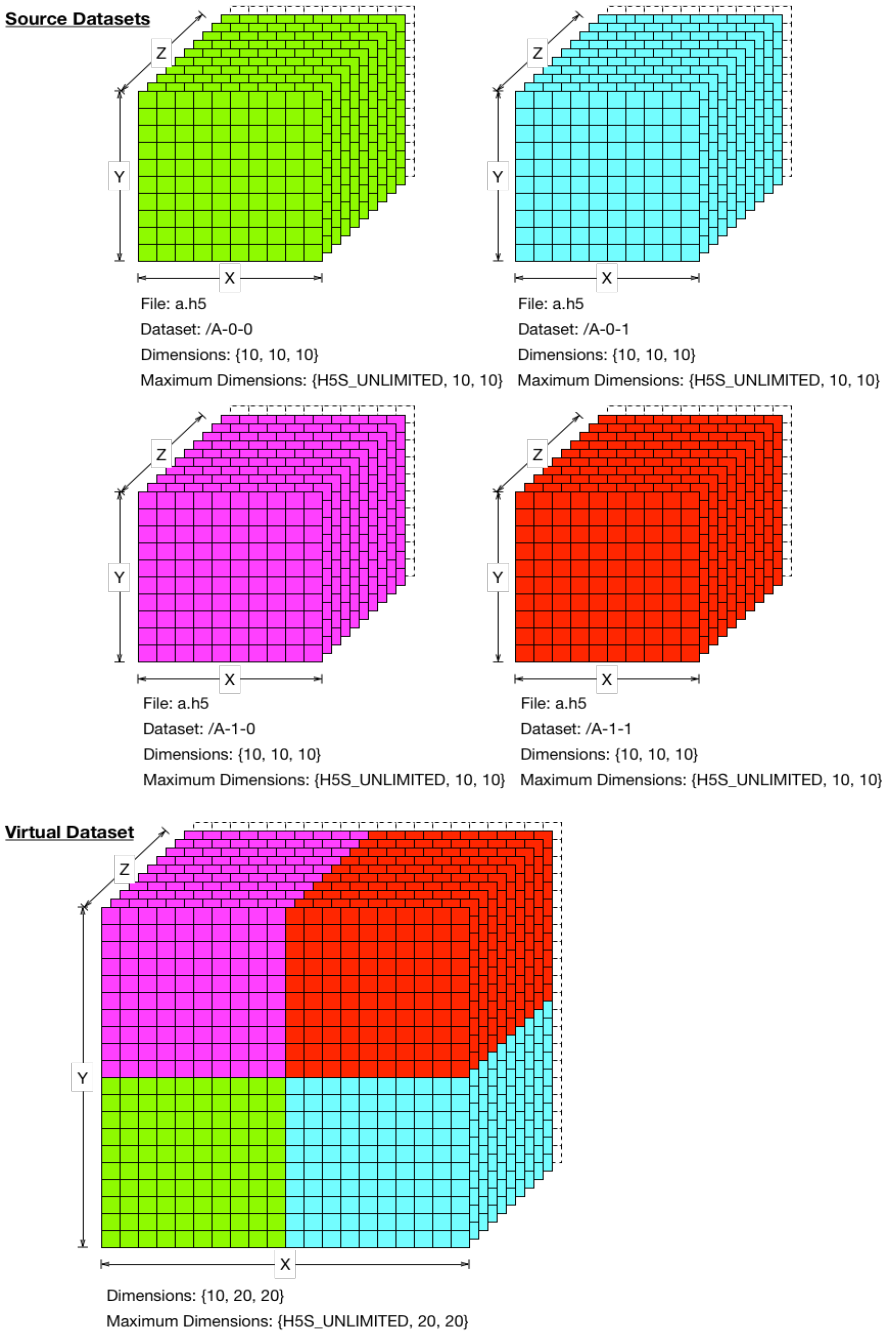


**Figure 13: Mapping full selections of four unlimited-dimension source datasets chosen with printf-naming to tiles in a virtual dataset with an unlimited dimension**

The following pseudo-code shows how to create this mapping:

```
hsize_t src_dims[3], max_src_dims[3], vds_dims[3], max_vds_dims[3];
hsize_t start[3], stride[3], block[3], count[3];
hid_t src_space_id, vds_space_id;
hid_t dcpl_id;

/* Create DCPL and set to virtual storage */
dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_virtual(dcpl_id);

/* Create source dataspace, select unlimited selection */
src_dims[3] = {/* Z: */ 10, /* Y: */ 10, /* X: */ 10};
max_src_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 10, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, max_src_dims);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(src_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Create virtual dataspace and set selection */
vds_dims[3] = {/* Z: */ 10, /* Y: */ 20, /* X: */ 20};
max_vds_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 20, /* X: */ 20};
vds_space_id = H5Screate_simple(3, vds_dims, max_vds_dims);
start[3] = {0, 0, 0};
stride[3] = {1, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between source datasets and unlimited, tiled selection in virtual
dataset, with printf-mapping */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/A-%1b-%2b”, src_space_id);
```

```
/* Close source dataspace */
H5Sclose(src_space_id);
```

## 8.7 Fixed-dimension Full Source and Unlimited-dimension Virtual Dataset Mapping, Interleaved Pattern in the Virtual Dataset, printf-named Source Datasets with an Unlimited Dimension

The figure below shows a mapping of a printf-named set of source datasets with fixed-dimensions and full selections to a single virtual dataset with unlimited dimensions with each source dataset selection occupying an interleaved portion of the virtual dataset.

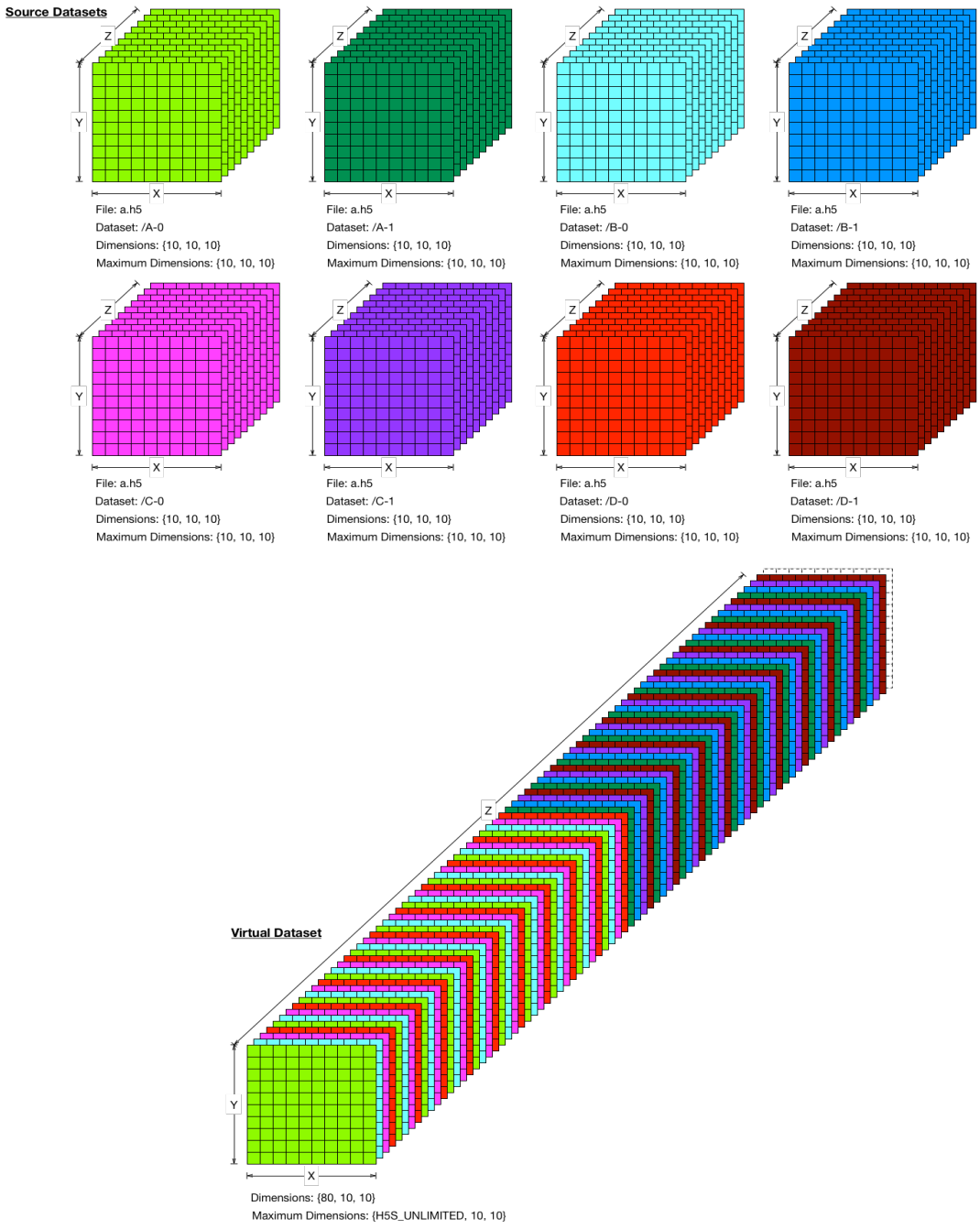


**Figure 14: Mapping full selections of fixed-dimension source datasets chosen with printf-naming to an interleaved portion of a virtual dataset with an unlimited dimension**

The following pseudo-code shows how to create this mapping:

```
hsize_t src_dims[3], vds_dims[3], max_vds_dims[3];
hsize_t start[3], stride[3], block[3], count[3];
hid_t src_space_id, vds_space_id;
hid_t dcpl_id;

/* Create DCPL and set to virtual storage */
dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_virtual(dcpl_id);

/* Create source dataspace, defaults to “all” selection */
src_dims[3] = {/* Z: */ 10, /* Y: */ 10, /* X: */ 10};
src_space_id = H5Screate_simple(3, src_dims, NULL);

/* Create virtual dataspace and set first selection */
vds_dims[3] = {/* Z: */ 10, /* Y: */ 10, /* X: */ 10};
max_vds_dims[3] = {/* Z: */ H5S_UNLIMITED, /* Y: */ 10, /* X: */ 10};
vds_space_id = H5Screate_simple(3, vds_dims, max_vds_dims);
start[3] = {0, 0, 0};
stride[3] = {4, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between first source dataset sequence and first unlimited, strided
selection in virtual dataset, with printf-mapping */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/A-%0b”, src_space_id);

/* Set second selection in virtual dataspace */
start[3] = {1, 0, 0};
stride[3] = {4, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
```

```
      count);

/* Set mapping between second source dataset sequence and second unlimited,
strided selection in virtual dataset, with printf-mapping */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/B-%0b”, src_space_id);

/* Set third selection in virtual dataspace */
start[3] = {2, 0, 0};
stride[3] = {4, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between third source dataset sequence and third unlimited, strided
selection in virtual dataset, with printf-mapping */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/C-%0b”, src_space_id);

/* Set fourth selection in virtual dataspace */
start[3] = {3, 0, 0};
stride[3] = {4, 1, 1};
block[3] = {1, 10, 10};
count[3] = {H5S_UNLIMITED, 1, 1};
H5Sselect_hyperslab(vds_space_id, H5S_SELECT_SET, start, stride, block,
      count);

/* Set mapping between fourth source dataset sequence and fourth unlimited,
strided selection in virtual dataset, with printf-mapping */
H5Pset_virtual_mapping(dcpl_id, vds_space_id, “a.h5”, “/D-%0b”, src_space_id);

/* Close source dataspace */
H5Sclose(src_space_id);
```

## References

1. “DLS RFC: HDF5 Parallel Compressed Writer Requirements and Use Cases”, Doc No: TDI-CTRL-REQ-034, Issue: 0.2, April 5, 2013.

## Acknowledgment

This paper is based upon work supported by Diamond Light Source Limited.

## Revision History

| | |
|---|---|
| *April 24, 2013:* | Version 1 prepared for the brain storming meeting by Elena |
| | Version 2: accepted Quincey's changes; sent to DLS |
| *August 20, 2013* | Version 3: drawing and programming model were added |
| *September 1, 2013* | Version 4: revised programming model |
| *September 3, 2013* | Version 5: created section 7; sent to DLS |
| *November 21, 2014* | Version 6: Many revisions to simplify programming model, added examples of how to use mapping function call; sent to DLS |
| *December 4, 2014* | Version 7: Moved programming examples to the "Examples of Source to Virtual Dataset Mapping" chapter, more cleanups and clarifications. |
| *December 9, 2014* | Version 8: Added the section on implementation and sent to DLS |
| *December 10, 2014* | Version 9: General editing. |