Future Enhancements to the HDF5 Library

Outline

- HDF5 1.10.0 new features
- Possible enhancements

NEW FEATURES IN 1.10.0

HDF5 problems addressed by 1.10.0 release

- Unused space in HDF5
  - Persistent free file space tracking
- Non-optimal chunking storage and performance
  - Scalable chunk indexing
- Data access to file being written
  - Single Writer/Multiple Reader access
- Viewing data stored in multiple datasets
  - Virtual Dataset
- Improvements to parallel HDF5

Reusing free file space in a file

PERSISTENT FILE FREE SPACE TRACKING

Unused space in HDF5 file

- HDF5 library currently only tracks free space while file is open
  - Space from deleted objects
  - Space from resized compressed chunks
- Free space in the file is “lost” after file is closed
- h5repack is used to remove “holes” in the file
- New function H5Pset_file_space
  - Sets a property to track free space in the file that can be reused when file is reopened
  - Allows fine tuning space tracking

Improving performance and saving space

SCALABLE CHUNK INDEXING

Optimizing chunking storage and performance

- HDF5 has an ability to add more data to existing datasets (data arrays)
- Special storage mechanism – chunked storage
- B-trees are used to index chunks in the file
  - O(log n) lookup time
- HDF5 takes advantage of the access pattern and properties of the datasets
  - O(1) lookup time
  - File space savings when storing HDF5 metadata

Optimizing chunking storage and performance

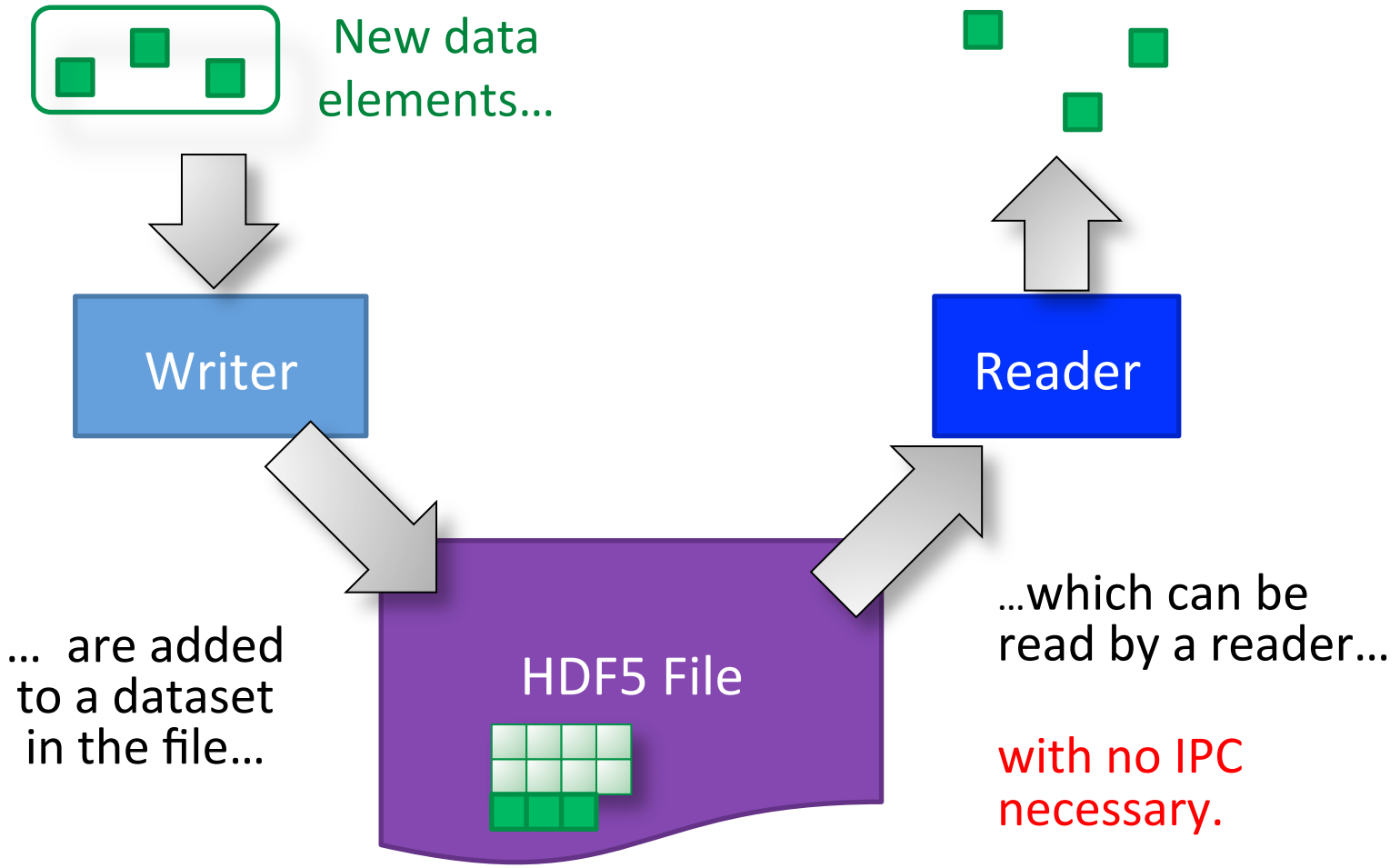
- B-tree implementation was reworked to use less space in the file
  - Used for datasets with more than one unlimited dimension
- New indexing structures were introduced to achieve O(1) performance and storage savings in special cases

Optimizing chunking storage and performance

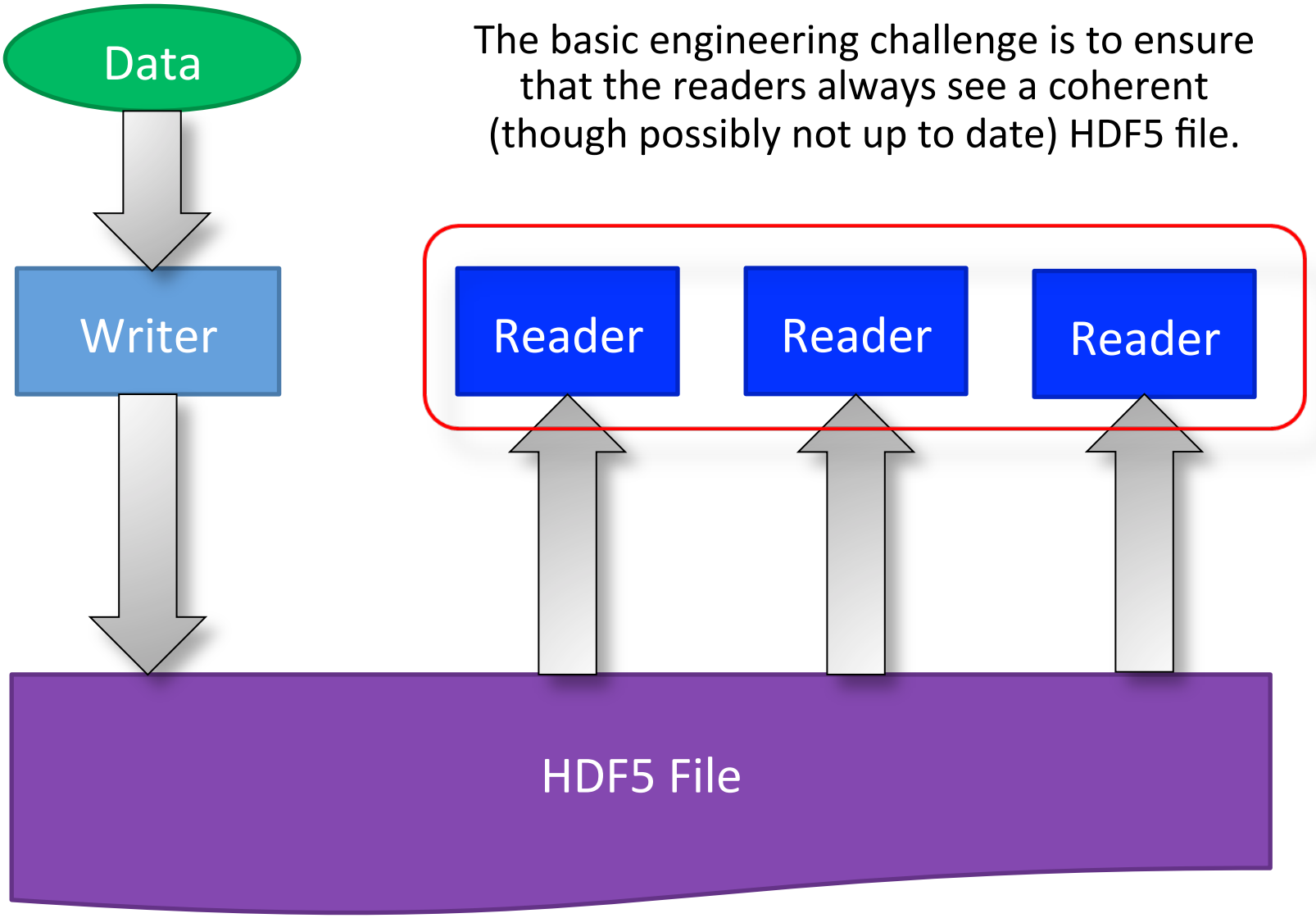
- Examples of O(1) lookup access:
  - Fixed-size chunked dataset with no compression filters
    - Algorithmic lookup
  - Fixed-size chunked dataset with compression filters
    - Array to index chunks
  - Fixed-size dataset stored in one chunk (i.e., we now allow compression for contiguous dataset)
    - No index
  - Dataset with one unlimited dimension
    - Extensible array to index chunks

CONCURRENCY: SINGLE-WRITER/MULTIPLE-READER

Data access to file being written



The Challenge



The basic engineering challenge is to ensure that the readers always see a coherent (though possibly not up to date) HDF5 file.

Data access to file being written

- Implemented for raw data "append only" scenario
  - No creation or deletion of the datasets, groups, and attributes is allowed at this time
- Works on GPFS, Lustre, Linux Ext3, Ext4, FreeBSD USF2, OS X HDFS+
- Does not work on NFS
- Documentation http://www.hdfgroup.org/HDF5/docNewFeatures/
- Source ftp://ftp.hdfgroup.uiuc.edu/pub/outgoing/SWMR/
- Testers are needed!

VDS

PROGRAMMING MODEL

Data access to file being written

Requirement: easy to set up; no major changes to applications

Writer

- Call H5Fopen or create using the H5F_ACC_SWMR_WRITE flag.

Reader

- Call H5Fopen using the H5F_ACC_SWMR_READ flag.

Data access to file being written

Writer

- Write data to the HDF5 file.

Reader

- Poll, checking the size of the dataset to see if there is new data available for reading.
- Read new data, if any.

Side affect of SWMR access

- Fault tolerance

Example of SWMR writer

```
//Create the file using the latest file format property as
shown
fapl = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_libver_bounds(fapl, H5F_LIBVER_LATEST,
H5F_LIBVER_LATEST);
fid = H5Fcreate(filename, H5F_ACC_TRUNC, H5P_DEFAULT, fapl);
//Create file objects such as datasets and groups.
// Close attributes and named datatypes objects. Groups and
// datasets may remain open before starting SWMR access to
// them.
// Start SWMR access the file
status = H5Fstart_swmr_write(fid);
// Reopen datasets and start writing
H5Dwrite(dset_id);
H5Dflush)dset_id); // periodically to flush the data for a
particular dataset.
```

Example of SWMR reader

```
// Open the file using SWMR read flag
fid = H5Fopen(filename, H5F_ACC_RDONLY |
H5F_ACC_SWMR_READ, H5P_DEFAULT);
// Open the dataset, poll dimensions, read new data
and refresh; repeat.
dset_id = H5Dopen(…);
space_id = H5Dget_space;
while (…) {
   H5Dread(…); // read if any new data arrives
   H5Drefresh;
   H5Dget_space(…);
}
```

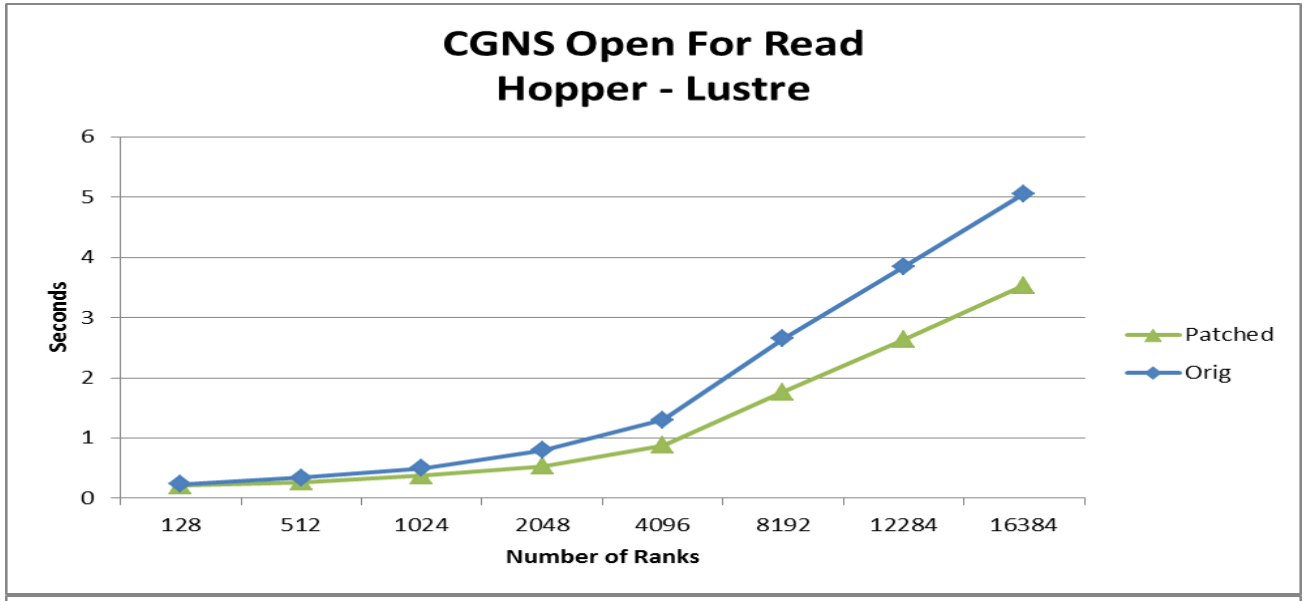
Parallel HDF5

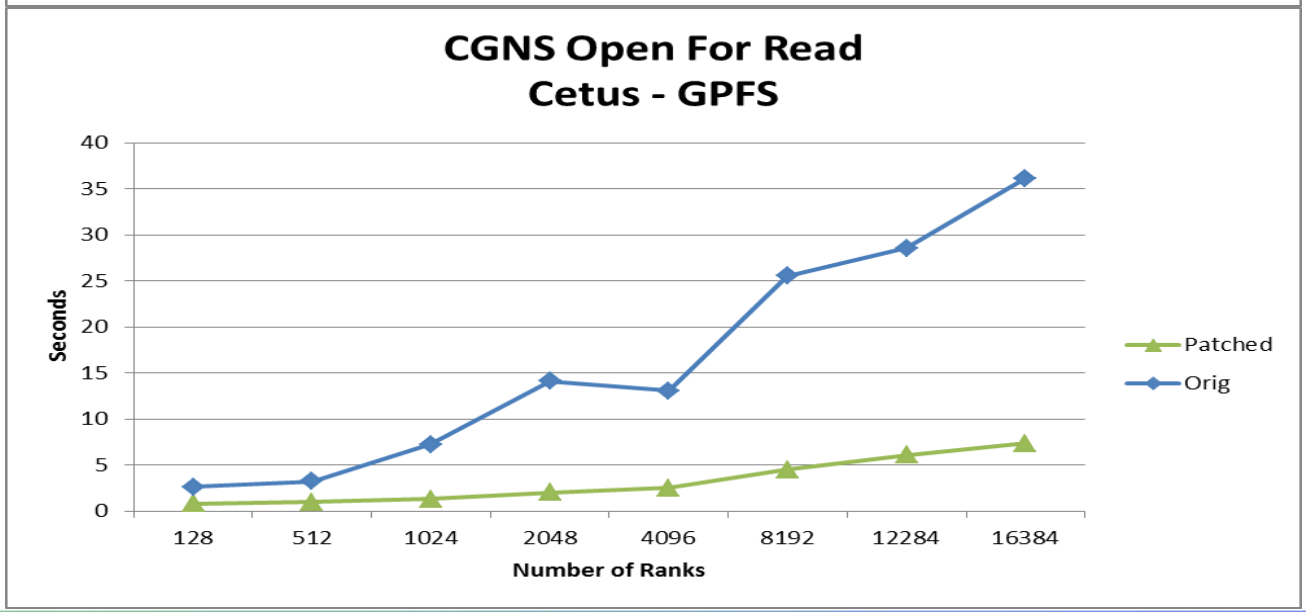
ENHANCEMENTS TO PARALLEL HDF5

Metadata Cache Optimizations

- Avoiding the Metadata Read "Storm"
  - Affects large MPI jobs when using, for example, H5Dopen
  - Issuing many small I/O requests to read metadata overwhelms the system

Performance with CGNS

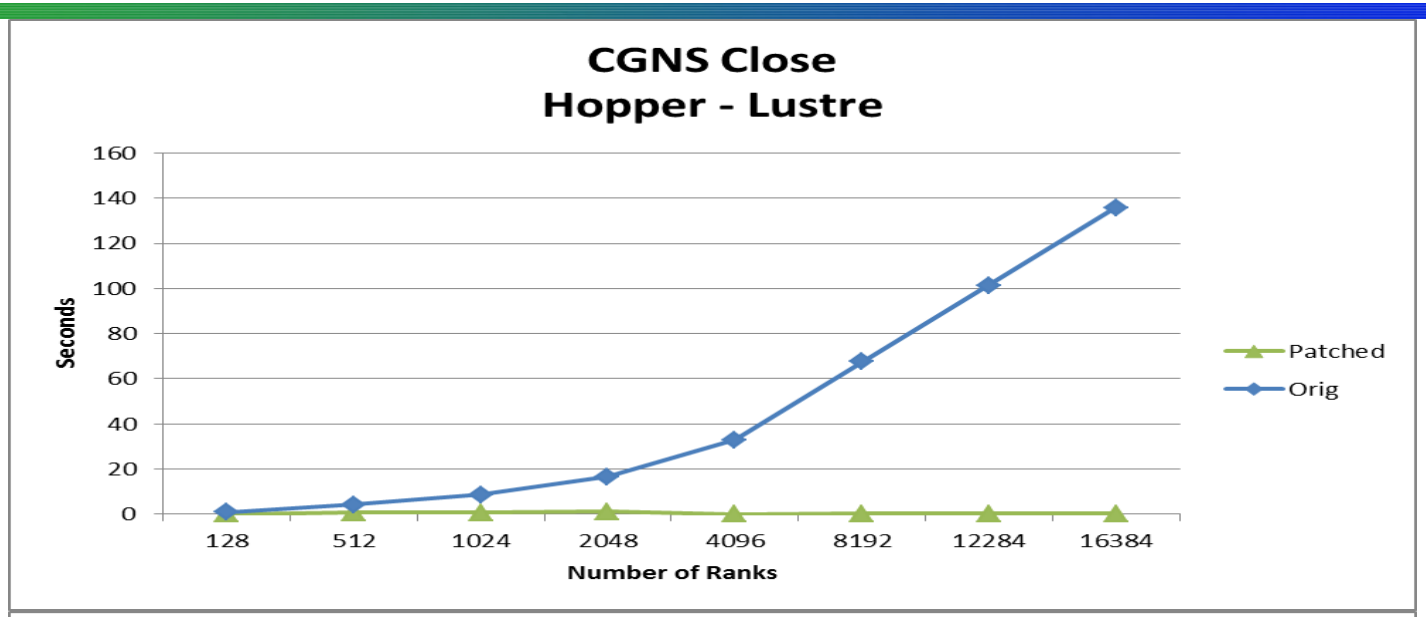


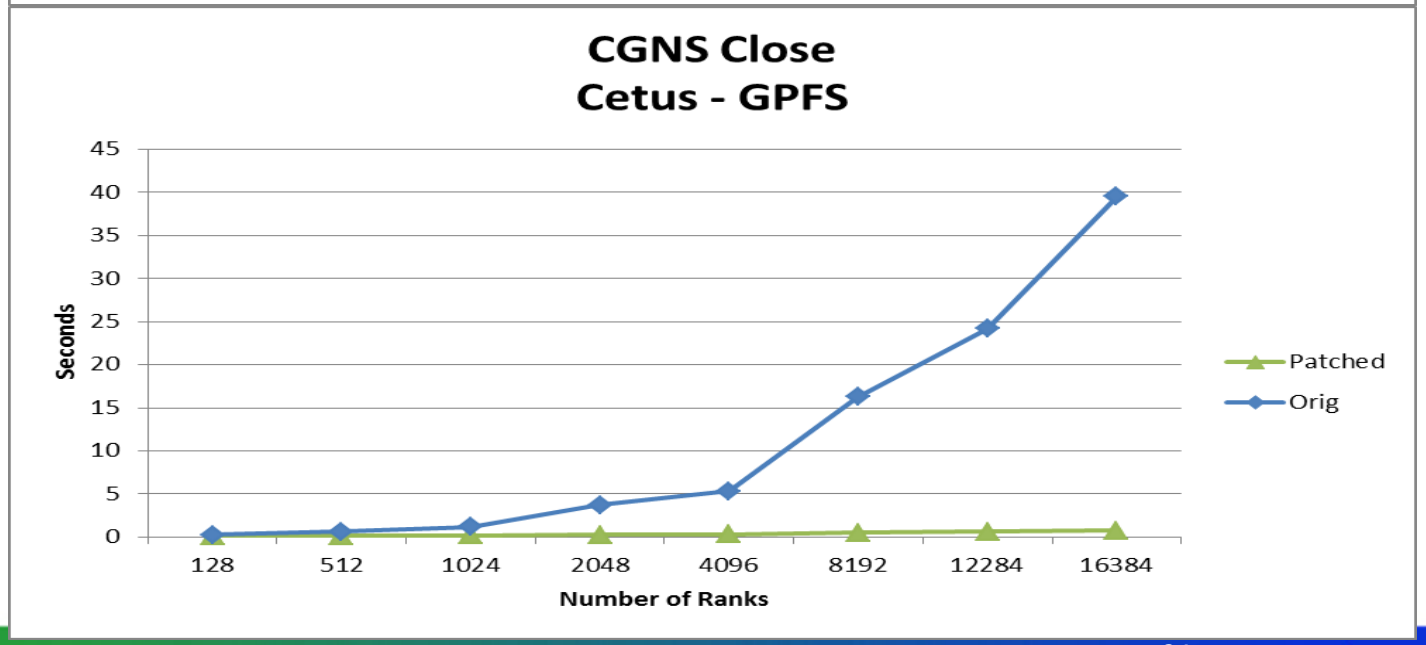


Metadata Cache Optimizations

- Collective Metadata Writes
  - H5Fclose is slow since it is writing small metadata items independently
  - Parallel FS don't like it

Performance with CGNS





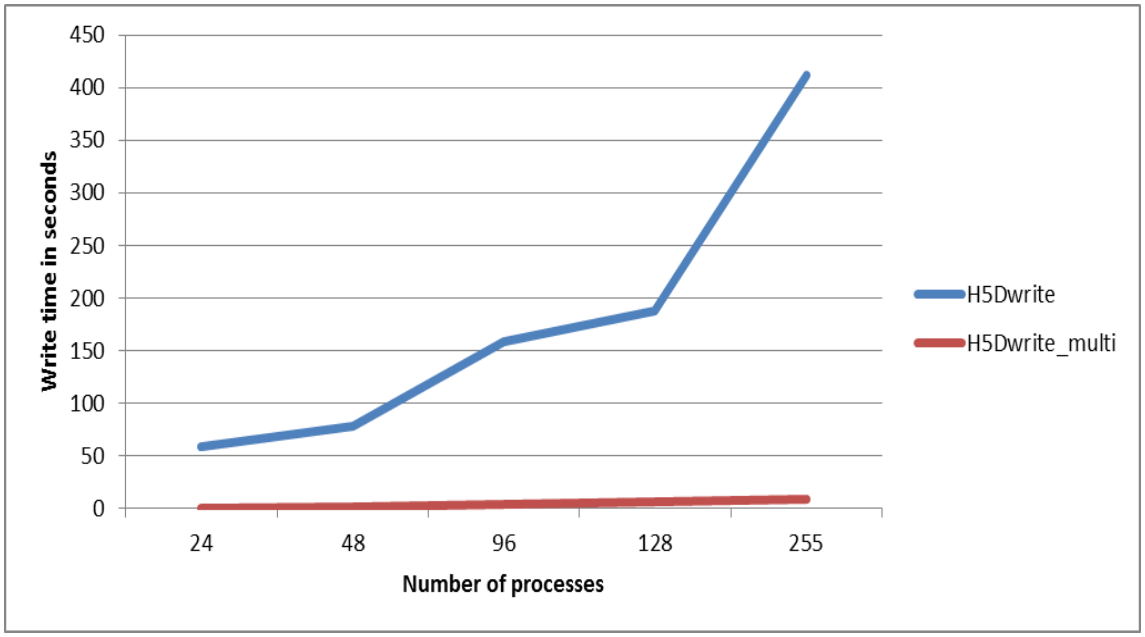
Multi-Dataset I/O - Motivation

- Many HPC applications access data in multiple datasets, but access to each dataset could be small.
  - This translates to multiple I/O calls to the underlying file system.
- Performance can be improved by allowing users to do more I/O per a single call to HDF5.

Multi-Dataset APIs

- Two New API routines:
  - H5Dread_multi()
    - Read data from multiple datasets in a HDF5 file with a single call
  - H5Dwrite_multi()
    - Write data to multiple datasets in a HDF5 file with a single call

Performance Results



On Hopper (Cray XE-6 with Lustre file system), for example, the plot above shows the performance difference between using a single H5Dwrite() multiple times and using H5Dwrite_multi () once on 30 chunked datasets:

Run	Code	Nodes	Cores	File Size	Stripe Size	Write Time	Total HDF Time	Throughput
Max Throughput	Multiple Dataset	320	5,120	5 TB	1 GB	91.08 s	167.78 s	56.21 GB/s
Code Comparison	Single Dataset	320	5,120	5 TB	128 MB	116.94 s	117.37 s	43.78 GB/s
Hero Run	Single Dataset	9,314	298,048	291 TB	1 GB	5,763.14 s	5,779.89 s	51.81 GB/s

TABLE I
COMPARISON OF VPIC-IO KERNEL PARAMETERS AND OBSERVED IO THROUGHPUT.

Parallel HDF5

METADATA CACHE ENHANCEMENTS

Current Handling of HDF5 Metadata

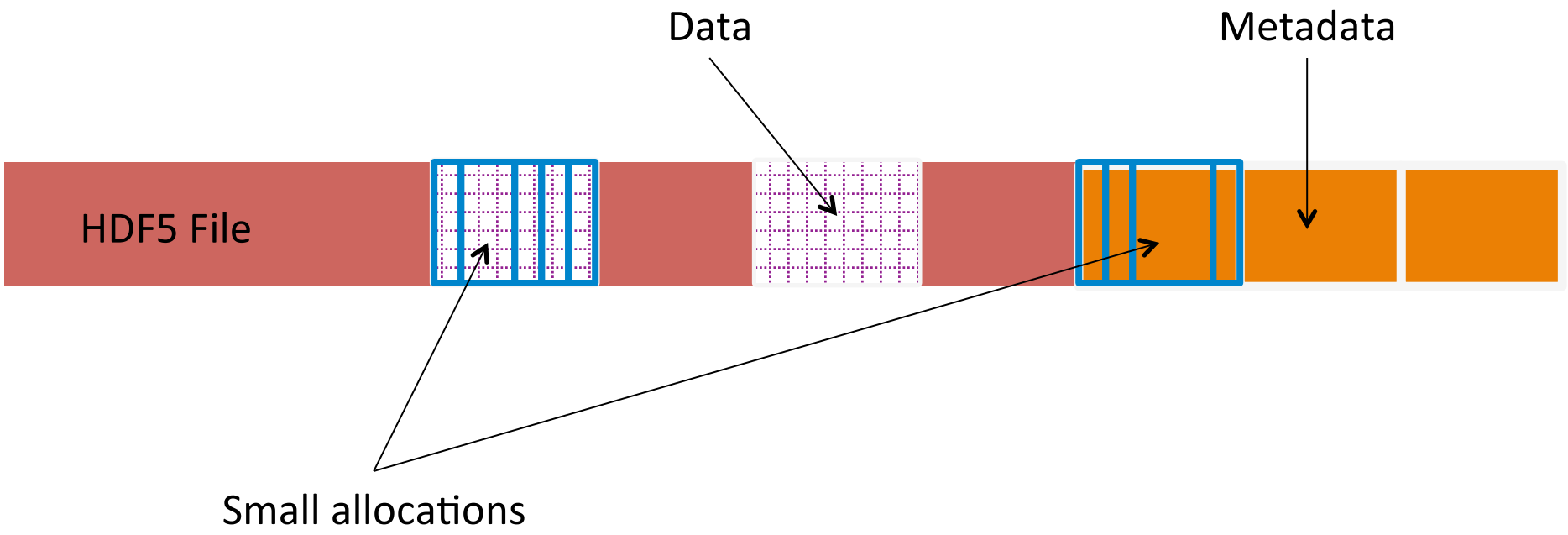
- Problems that affect metadata I/O
  - Size of metadata items aggregation varies and is not stored in the file
    - Library cannot take an advantage of reading metadata block since it doesn't know the length of the block
  - Metadata blocks are not aligned to the block size of the underlying file system and do not have size of some multiple of the file system block size

The Approach/Solution

Aggregate and align metadata and small data, perform I/O in aligned pages

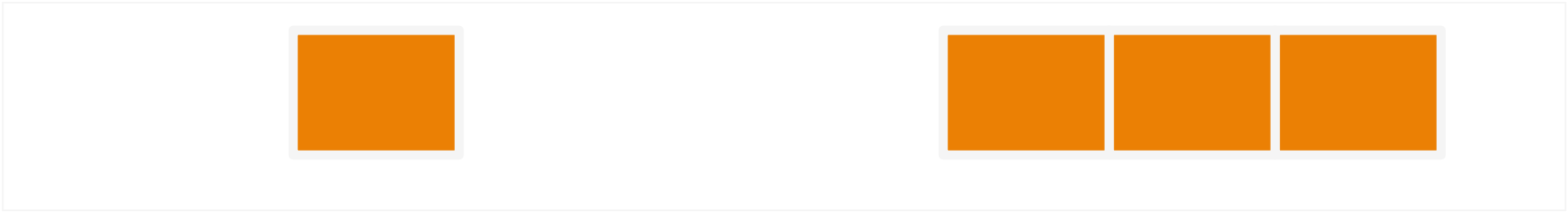
Data and Metadata Aggregators

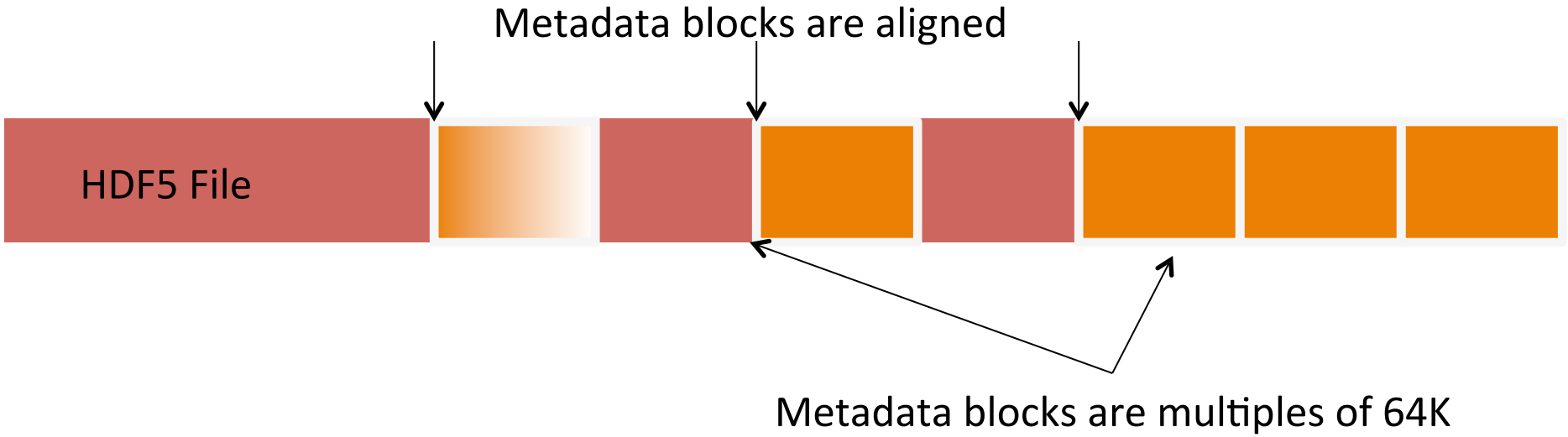
The new aggregators pack small raw data and metadata allocations into aligned blocks which work with the page buffer.



HDF5 Page Buffering

Page buffer contains MD pages (L2 cache)





HDF5 1.10.0

BACKWARD/FORWARD COMPATIBILITY ISSUES

Backward/Forward compatibility issues

- HDF5 1.10.0 will always read files created by the earlier versions
- HDF5 1.10.0 by default will create files that can be read by HDF5 1.8.*
- HDF5 1.10.0 will create files incompatible with 1.8 version if new features are used
- Tools to “downgrade” the file created by HDF5 1.10.0
  - h5format_convert (SWMR files; doesn’t rewrite raw data)
  - h5repack (VDS, SWMR and other; does rewrite data)

POSSIBLE ENHANCEMENTS

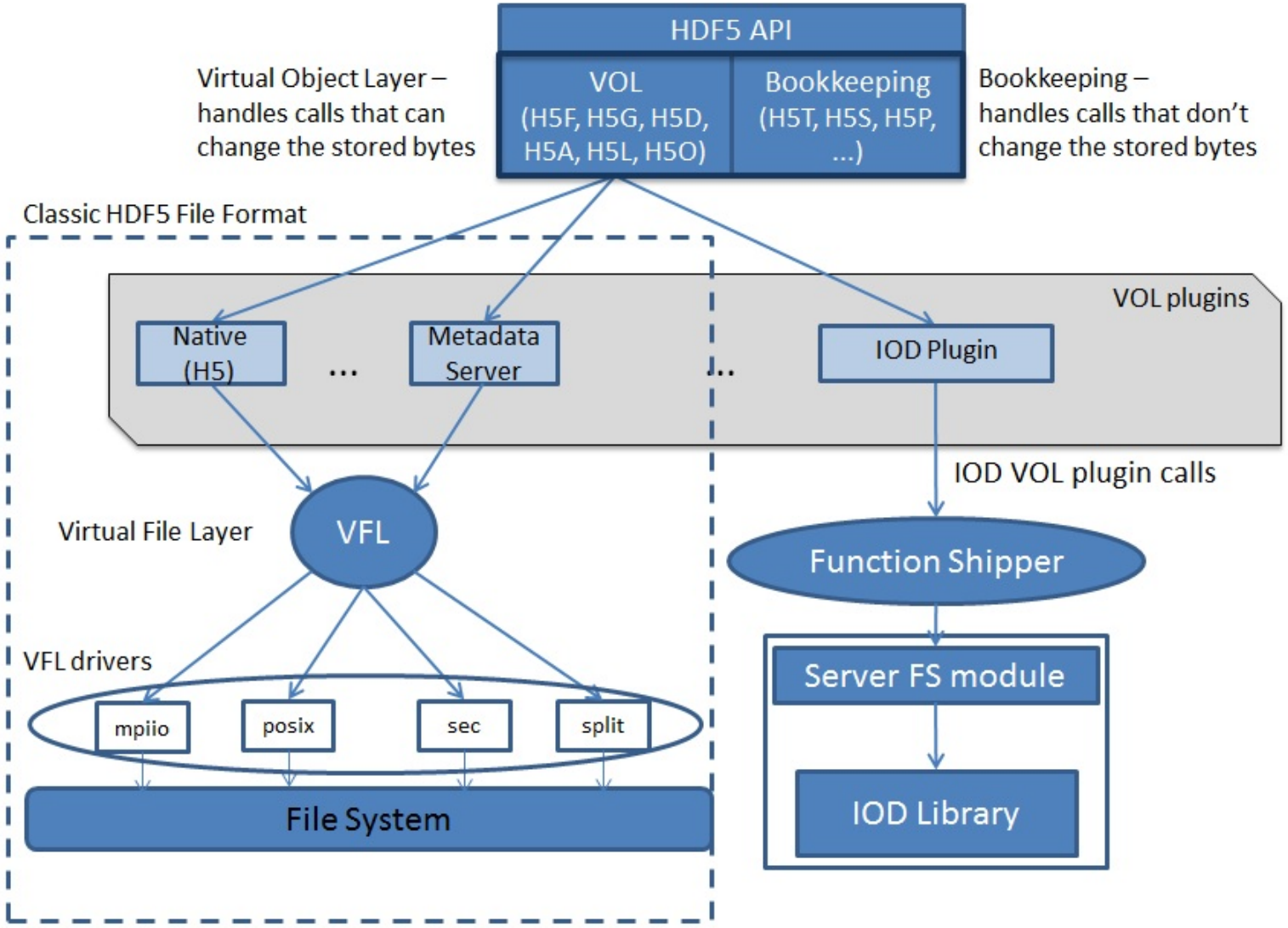
Features we are investigating

VIRTUAL OBJECT LAYER

Virtual Object Layer

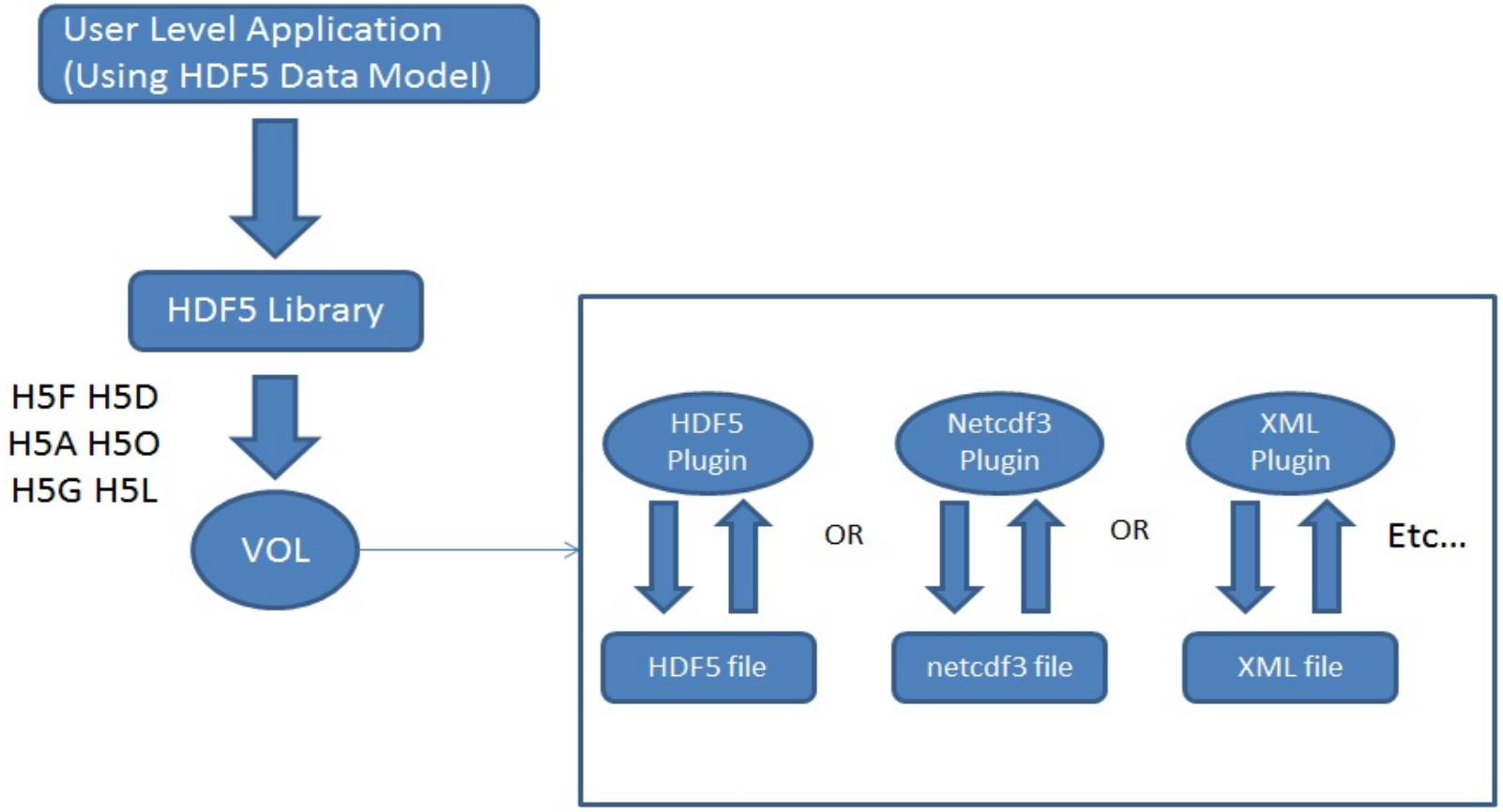
- Goal
  - Provide an application with the HDF5 data model and API, but allow different underlying storage mechanisms
- New layer below HDF5 API layer
  - Intercepts all API calls that potentially could touch the data on disk and route them to a Virtual Object Driver
- Potential Object Drivers (or plugins):
  - Native HDF5 driver (writes to HDF5 file)
  - Raw driver (maps groups to file system directories and datasets to files in directories)
  - Remote driver (the file exists on a remote machine)

VOL Architecture



Sample Plugins

- Different File Formats plugins:



Features we are investigating

DATA INDEXING

Indexing and HDF5

- New APIs for indexing and querying of both structure and contents of HDF5 file
- H5Q API defines query to apply to a file

Create/combine queries (OR, AND)

  - Basic operators supported ($\leq$, $\geq$,=, $\neq$) on either dataset/attribute values, link/attribute names
- HDF5V API retrieves data
- HDF5X API adds third-party indexing plugins

Example: Find specific links

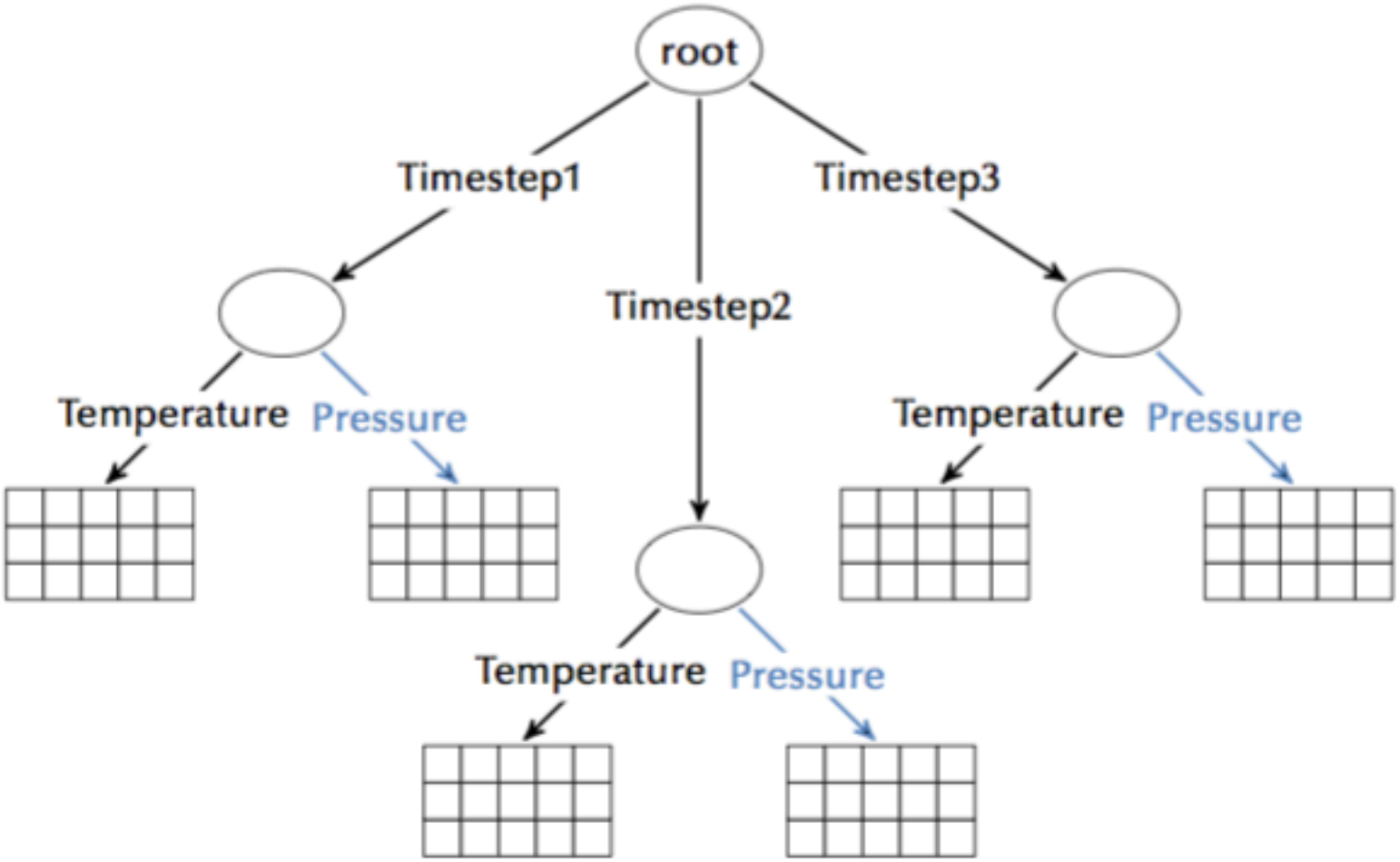


Figure 2 HDF5 container example with query $link\_name = Pressure$ applied.

Example: Find an element

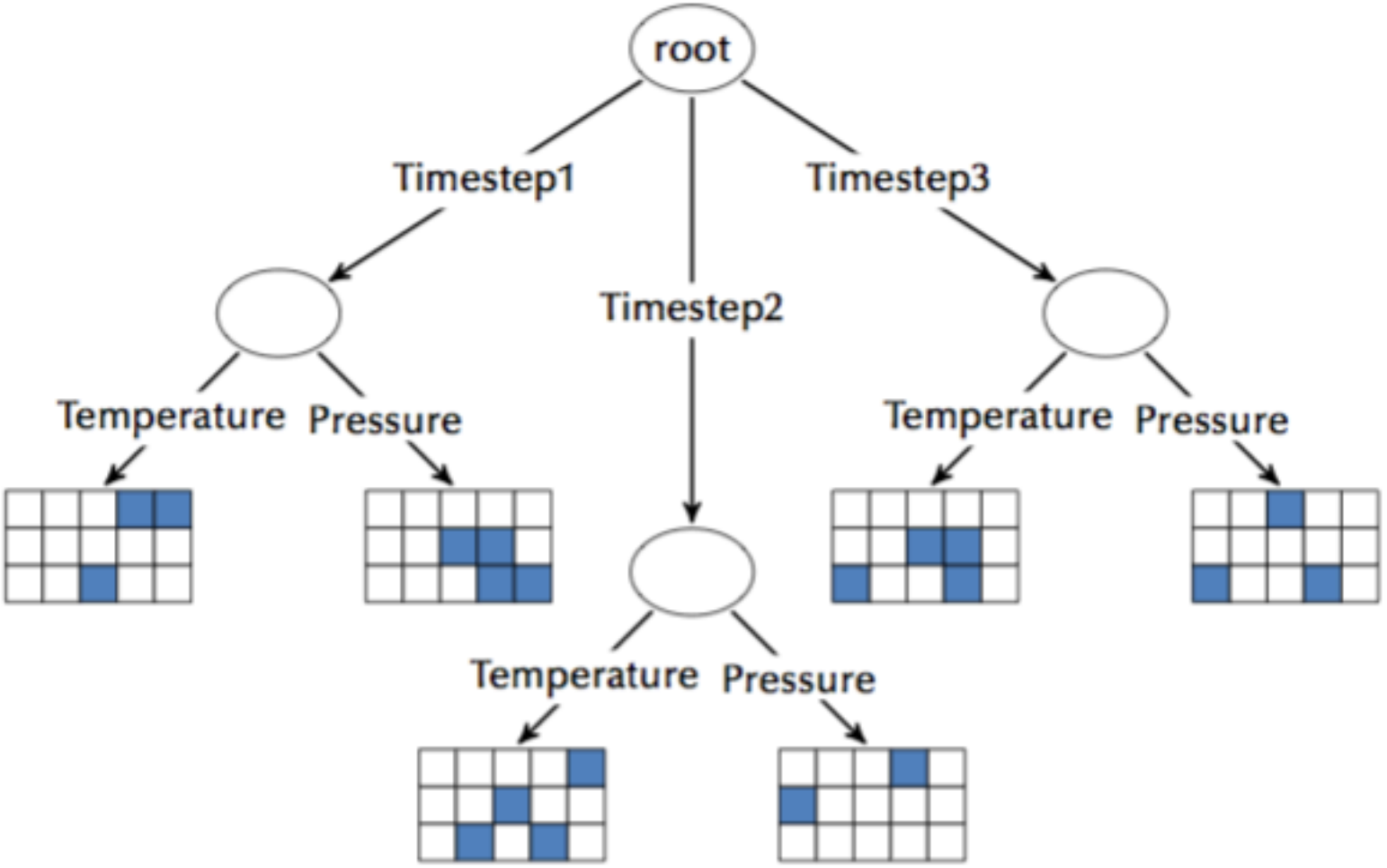


Figure 3 HDF5 container example with query *data_element* = 17 applied.

Example: Combined query

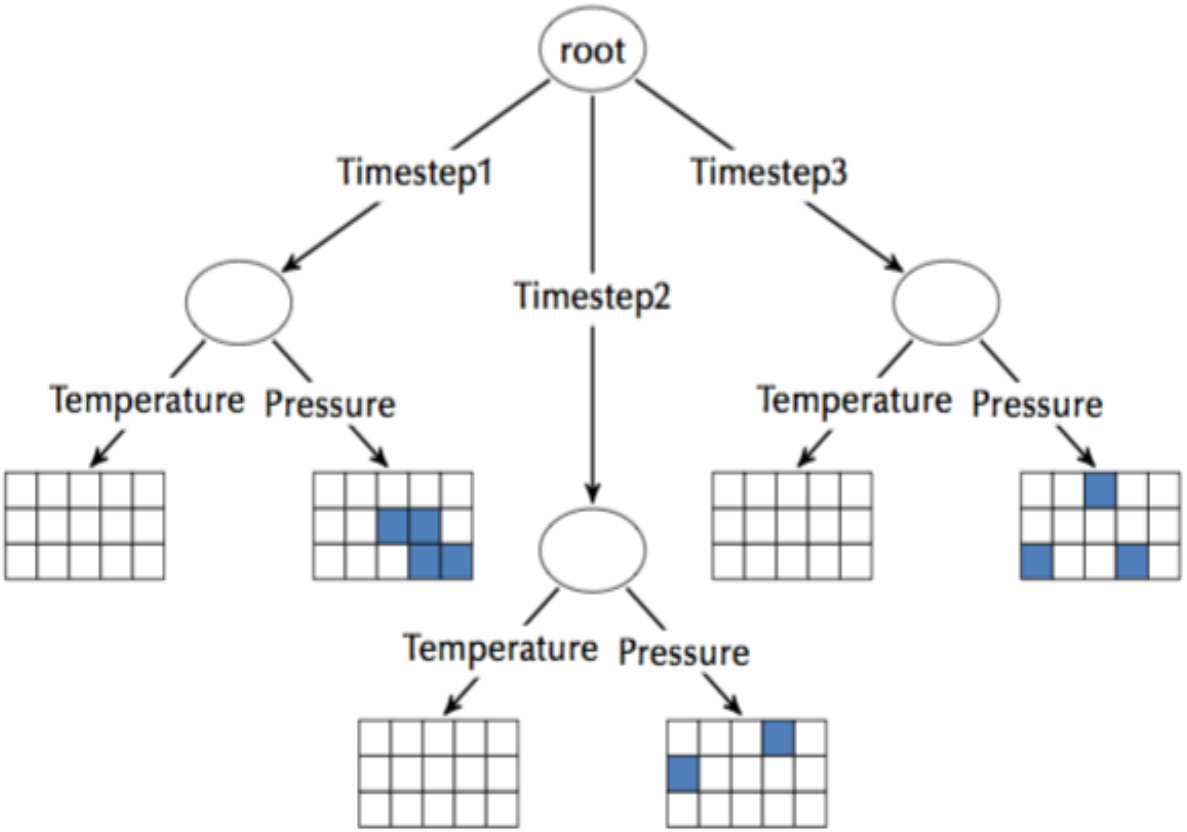


Figure 4 HDF5 container example with query $(link\_name = Pressure) \wedge (data\_element = 17)$ applied.

- Plugin mechanism
  - FastBit https://sdm.lbl.gov/fastbit/
  - ALACRITY from NCSU (Analytics-Driven Lossless Data Compression for Rapid In-Situ Indexing, Storing and Querying
  - Ability of third-party indexing packages to access HDF5 datasets when creating indices
  - Ability of third-party indexing packages to create, write and read index information within the HDF5 file
  - Ability of applications to register and remove indexing plugins, query available indexing packages and choose which package(s) to use when creating indices

Questions?

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?