“Extendable type-safe, thread-safe, asynchronous APIs for Neutron Science Data using modern C++ on top of HDF5”

William F Godoy, Addi Malviya Thakur, Steven E Hahn

Computer Science and Mathematics Division

Oak Ridge National Laboratory

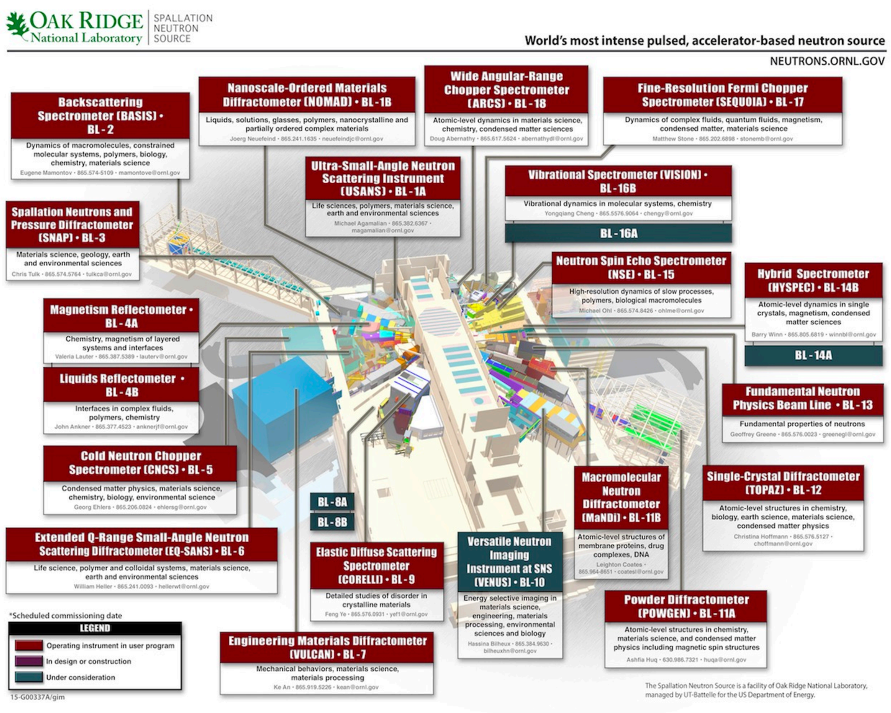
Prepared for: HDF5 Users Group Meeting HUG2021 10/14/2021

Contents

- Background: Data at ORNL neutron science facilities, SNS/HFIR
- Challenges: bottlenecks, implementations, sustainability
- Proposed long-term solution:
  - Type-safe, threaded API on top of HDF5 using modern C++
- Future?

Background

ORNL neutron facilities, SNS and HFIR, fill us with interesting data
www.neutrons.ornl.gov



Event-based Raw Neutron Data

- Saved to HDF5 files using the standard NeXus schema https://www.nexusformat.org/ capturing metadata annotations required for each instrument. (2,000 ~ 3,000 entries ….or more)

- < 5M events /s /instrument ~ 60 MB/s/instrument of raw data on the stream. Stored for 3 years at https://analysis.sns.gov/ 1.2 TB/day, Grand Total of 1.6 PB as of 2020. Single Intel Xeon "nodes" for processing.

- Mantid https://github.com/mantidproject/mantid processes raw-event data into in-memory "workspaces" using generic loader used by several instrument data reduction workflows. Used across several neutron facilities

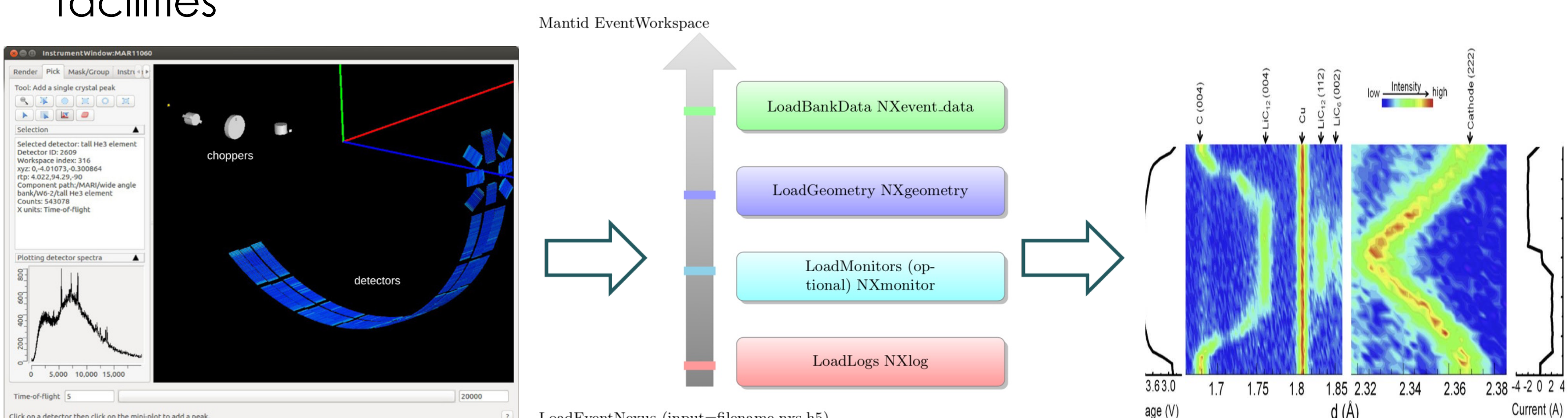
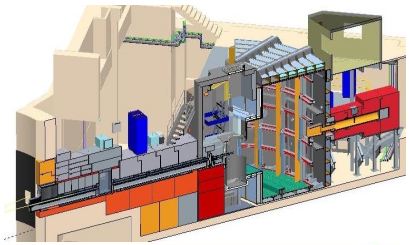


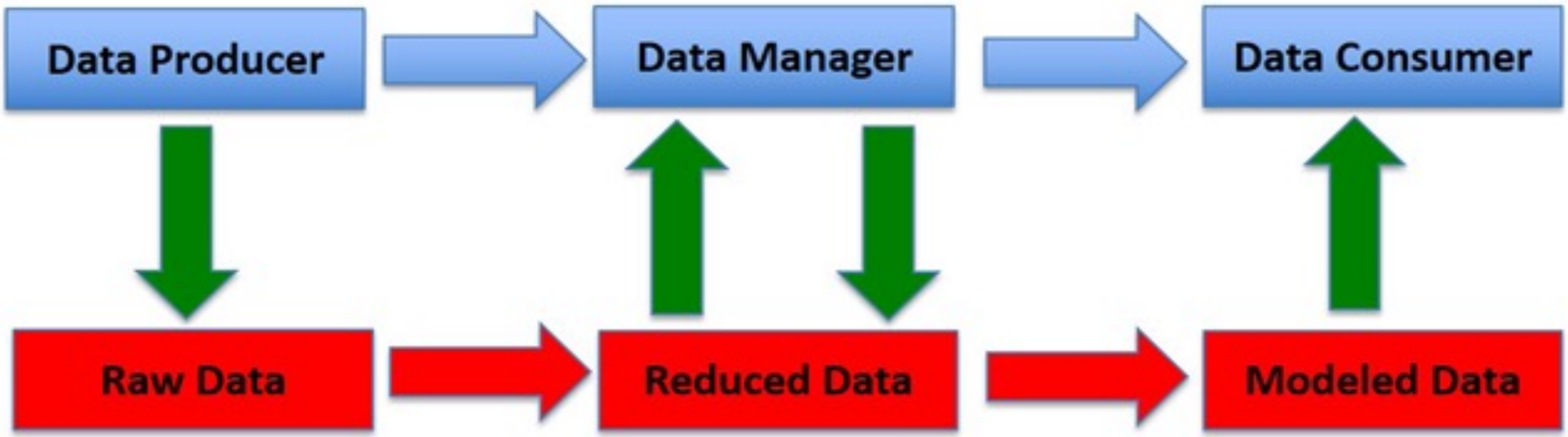
Fig. 2: Mantid's LoadEventNexus algorithm steps for processing entries of an input NeXus file generating a Mantid EventWorkspace data structure.

Schematic Overview of Data Flows



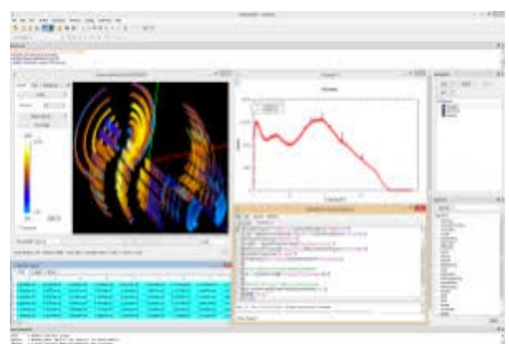
https://github.com/mantidproject/mantid

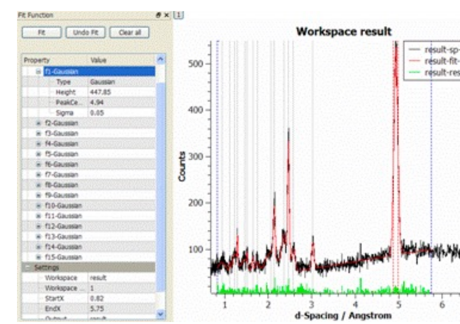
Remote Analysis Service



NeXus

Data Type	Entry Name
group	/entry
attribute	/entry/NX_class
	...
group	/entry/DASlogs
attribute	/entry/DASlogs/NX_class (NXlog)
group	/entry/DASlogs/BL6:CS:DataType
attribute	/entry/DASlogs/BL6:CS:DataType/NX_class
dataset	/entry/DASlogs/BL6:CS:DataType/average_value
dataset	/entry/DASlogs/BL6:CS:DataType/average_value_error
	...
group	/entry/bank5_events
attribute	/entry/bank5_events/NX_class (NXevent_data)
dataset	/entry/bank5_events/event_id
dataset	/entry/bank5_events/event_index





Donaldson, D.R., Martin, S. and Proffen, T., 2017. Understanding Perspectives on Sharing Neutron Data at Oak Ridge National Laboratory. Data Science Journal http://doi.org/10.5334/dsj-2017-035

Challenges

Several metadata indexing, data, memory challenges were identified on Mantid:

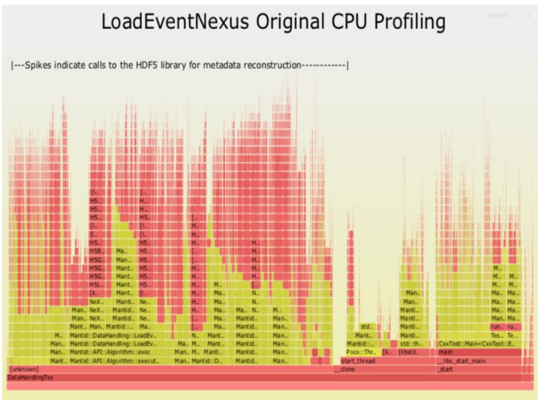
- Currently several I/O "glue-layers" to HDF5 including the defunct NeXus API library: https://github.com/nexusformat/code
- Inefficient data access, current APIs on top of HDF5 not designed with performance in mind balancing computation, memory, I/O → appropriate "in-memory" index for processing, memory hogs for indexing
- Threading opens several HDF5 descriptors (1 per thread) and locks I/O operations
- Single files are becoming "too large"…multiple files API? Few MB to 100 GB

Godoy W.F., Peterson P.F., Hahn S.E., Hetrick J., Doucet M., Billings J.J. (2020) Performance Improvements on SNS and HFIR Instrument Data Reduction Workflows Using Mantid. Smoky Mountains Conference 2020. https://doi.org/10.1007/978-3-030-63393-6_12

Short term improvements on Mantid Loader

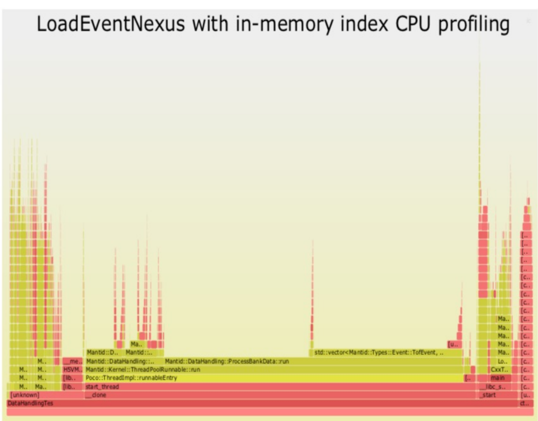
- Introduced a new in-memory indexing methodology. Facility Time == $$$$

Before



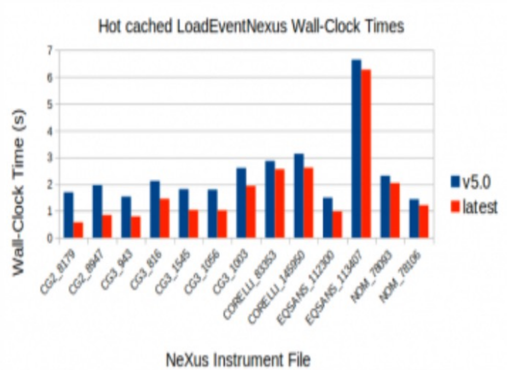
(a)

After



(b)

Flamegraph profiles, x is sampling per function, y is stack call!



Comparison of Mantid's "LoadEventNexus" wall-clock times for Mantid v5.0 release and our proposed strategy on Mantid's latest implementation. Results are shown for "hot" cached files (accessed several times) showing universal improvements across different ORNL SNS/HFIR instrument generated raw NeXus files (CG2 is GP-SANS, CG3 is BIO-SANS, NOM is NOMAD).

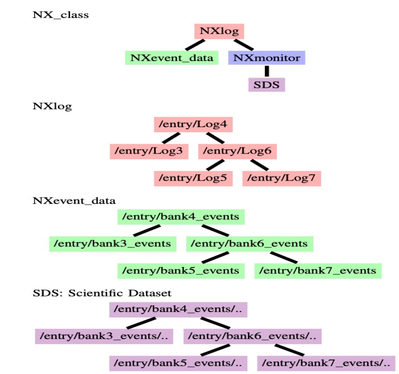


Fig. 4: Schematic representation of the efficient binary-tree in-memory index metadata for NeXus files entries classified by NX_class types at the top level. Each NX_class node (NXlog, NXevent_data, SDS) is a binary-tree on its own.

Impact on SNS/HFIR users

Instrument Workflow	Wall-clock time current index(s)	Wall-clock time improved index(s)	Speed up
GP-SANS	58.9	41.8	29%
Bio-SANS	100.2	80.9	19%
EQ-SANS	99.0	88.0	11%

Table 2: Overall wall-clock times comparison and speed up from applying the proposed in-memory index data structure on production data reduction workflows for SNS and HFIR instruments.

W. F. Godoy, P. F. Peterson, S. E. Hahn and J. J. Billings, "Efficient Data Management in Neutron Scattering Data Reduction Workflows at ORNL," *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 2674-2680, https://doi.org/10.1109/BigData50022.2020.9377836

Proposed long-term solution

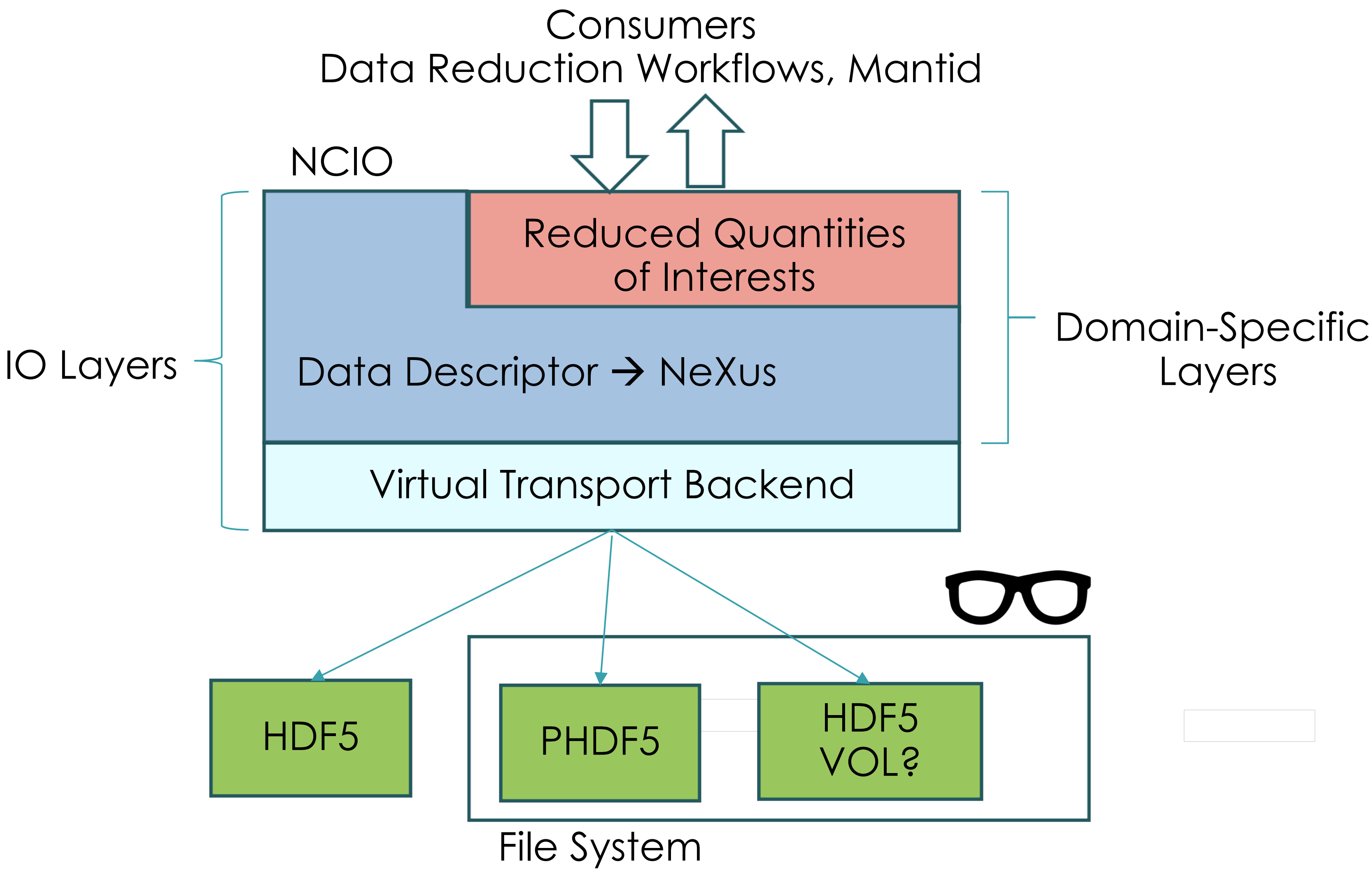
- No Cost I/O: NCIO (sorry for the pretentious name) Exploratory work: https://github.com/ORNL/ncio

- Domain specific API with the right level of abstraction on top of HDF5 (without doing a DSL approach):
  - NCIO: NeXus entry, bankID, histogram, log, instrument

- Different API levels:
  - Low-level "performance" API: pointers, deferred/lazy evaluation, key/value options, threaded? for backends
  - High-level: workflows on top of low-level APIs, bindings for end-users

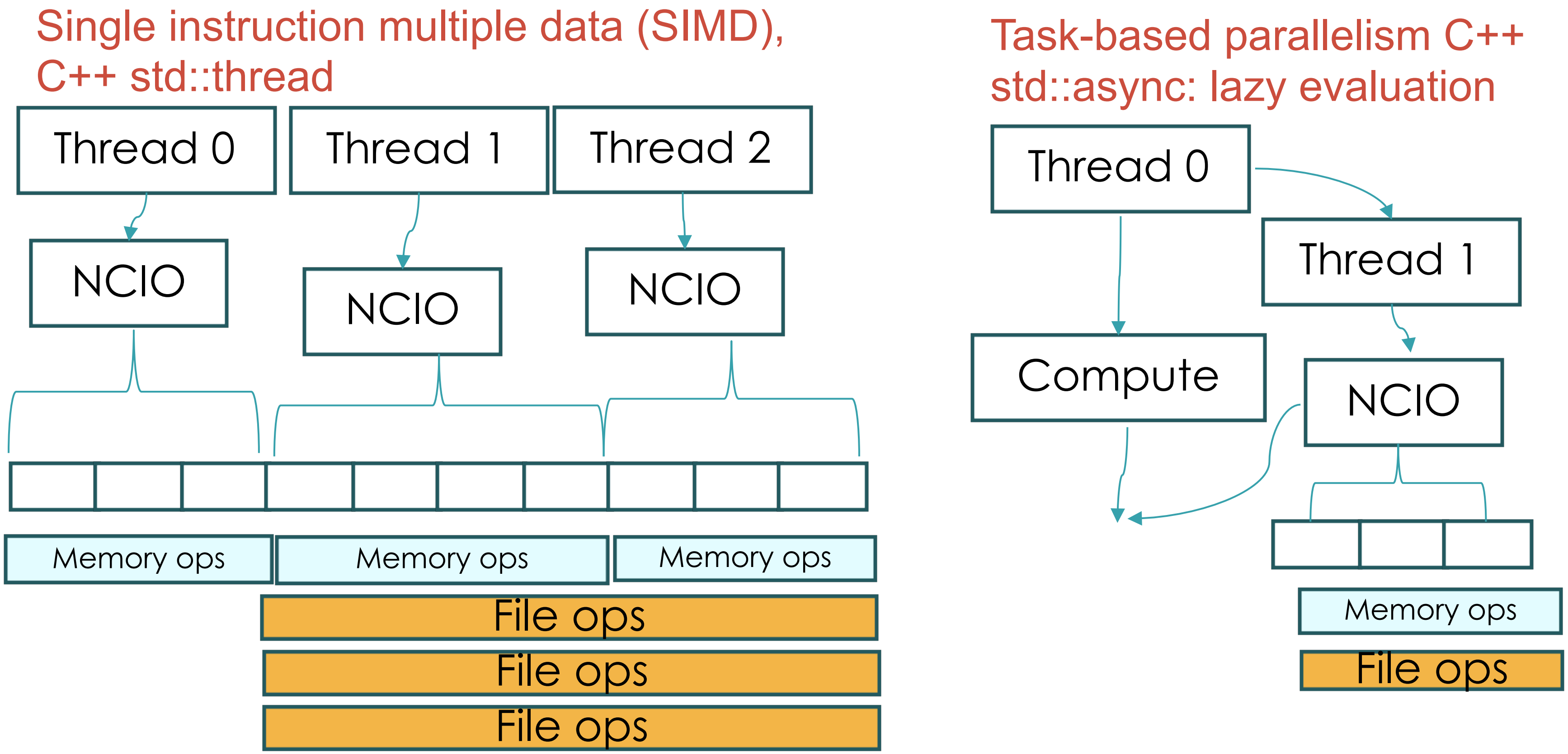
NCIO Pluggable Architecture

- NCIO should leverage HDF5 API features: VOL, compression, chunking



NCIO Multithreaded API

- Type-safe: as close as possible to a domain of science semantics
- Take advantage of modern C++ (auto, threads)
- Thread-safe and truly-threaded (if/when backends allow)

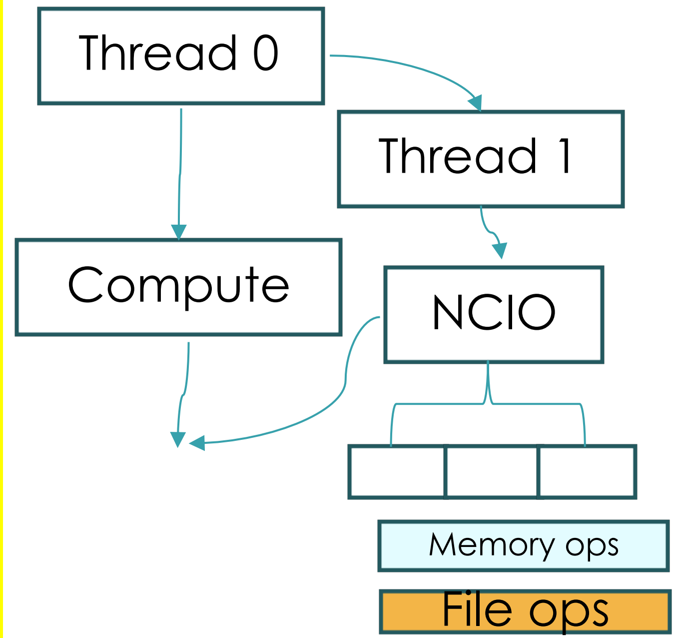


Type-safe using C++ templates, thread-safe API

- Concurrent I/O, compute API

```
ncio::DataDescriptor fr = ncio.Open("data_async.h5",
ncio::OpenMode::read);

// Get is type-safe lazy evaluation, ref and pointer based
fr.Get<ncio::schema::nexus::entry::bank1_events::total_counts>
(totalCounts);
 fr.Get<ncio::schema::nexus::entry::bank1_events::event_index>
(eventIndex.data(), ncio::BoxAll);
 // automatic reallocation when executing
 std::vector<double> eventTimeZero;
 fr.Get<ncio::schema::nexus::entry::bank2_events::event_time_
zero>(eventTimeZero, ncio::BoxAll);

// HDF5 action happens in the background
std::future future = fr.ExecuteAsync(std::launch::async);
do_some_interesting_compute();  //overlap compute + I/O
future.get(); // data is available
fr.Close();
```

Task-based parallelism
C++ std::async

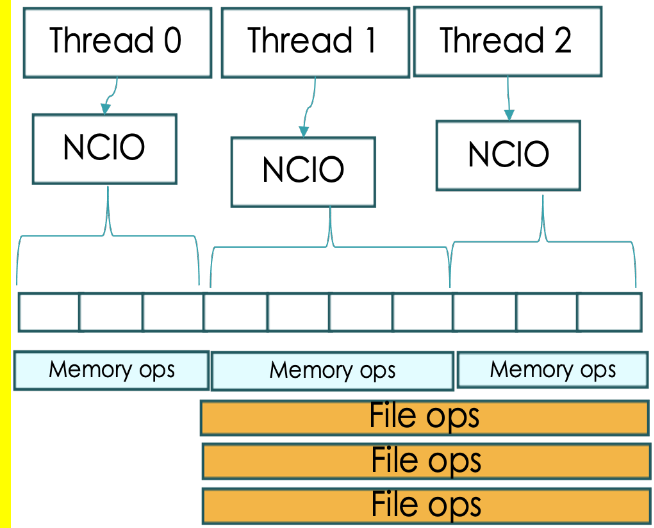


Type-safe using C++ templates, thread-safe API

- SIMD thread API (always pre-allocate memory)

```
// any callable
auto lf_ReadChunkThread = [](...){
 // start, count  = f(threadID);
fw.Get<ncio::schema::nexus::entry::bank1_events::event_time_offset>(&eventTimeOffset[start], {{start}, {count}}, threadID); }

// thread-safe handler
ncio::DataDescriptor fr = ncio.Open("data_threads.h5",
ncio::OpenMode::read);

// C++11 threads or OpenMP
std::vector<std::thread> threads;
threads.reserve(nThreads);

// launch thread task
for (auto threadID = 0; threadID < nThreads; ++threadID)
   threads.emplace_back(lf_ReadChunkThread, threadID,
nThreads, std::ref(eventTimeOffset), std::ref(totalCounts),
std::ref(fr));

for (auto &thread : threads) thread.join();
// data is available
fr.Close();
```

Single instruction multiple data (SIMD), C++ std::thread



“Appropriate” type-safe in-memory index API

- Your favorite IDE would pick up these types (in case the user forgets)…

```
ncio::DataDescriptor fr = ncio.Open(fileName, ncio::OpenMode::read);

const auto nxClassIndex =
    fr.GetMetadata<ncio::schema::nexus::index::model1,
                   ncio::schema::nexus::model1_t>();        You, 8 months ago • Test to v
```

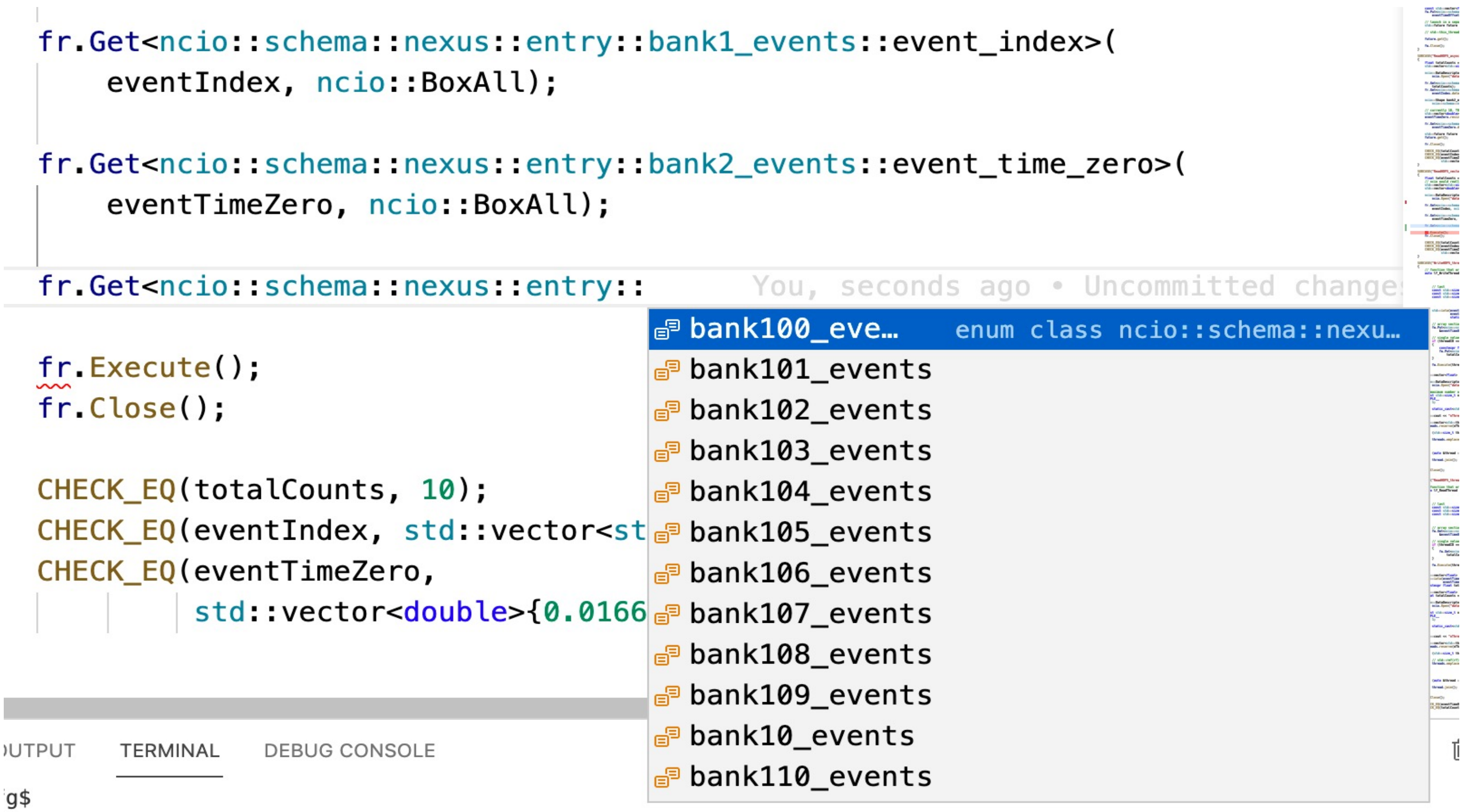
ncioTypesSchemaNexus.h ~/workspace/ncio/source/ncio/schema/nexus - Definitions (1)

```
enum class index
{
    model0,
    model1,
    model2,
    model3,
};

using model0_t = std::set<std::string>;
using model1_t = std::map<std::string, std::set<std::string>>;
using model2_t = std::map<std::string, std::string>;
using model3_t = std::unordered_map<std::string, std::unordered_set<std::string>>;

```

W. Zhang, S. Byna, C. Niu, and Y. Chen ,“Exploring metadata search essentials for scientific data management,” in 2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC), 2019, pp. 83–92.

“Appropriate” type-safe data entry API

- Your favorite IDE would pick up these types (in case the user forgets)…
- Catch errors way before runtime (or before compile time if using IDEs)
- Possible with C++17 auto template deduction (maps hierarchical entries nicely)



Summary

- Tackling (array-based) data as well as “in-memory” metadata index access is essential for reduction workflows at ORNL neutron science facilities SNS/HFIR.

- Current data access implementations on top of HDF5 serve specific purposes and they map 1-to-1 to HDF5 API calls

- We present a “extendable” thread-safe (concurrent and SIMD), type-safe, lazy API on top of HDF5 using modern C++ features (template auto, std::thread, std::async)

- https://github.com/ORNL/ncio (still exploratory, but running nightly regression with actual NeXus HDF5 data)

Future?

- More data is being produced that won't fit in memory: https://neutrons.ornl.gov/sts Second Target Station

- Might need current high-performance computing (HPC), MPI, parallel file system, NVRAM, etc.

- Extension to high-level languages (Python, Julia, R) for the end-user have its own challenges:

  - "Just-in-time" type-safety
  - Python's GIL, Global Interpreter Lock

- Some of these ideas need operational "patron" support…
  "quality software = large investment"

ACKNOWLEDGEMENT

Work at Oak Ridge National Laboratory was sponsored by the Division of Scientific User Facilities, Office of Basic Energy Sciences, US Department of Energy, under Contract no. DE-AC05-00OR22725 with UT-Battelle, LLC

Thanks to the HUG2021 organization, The HDF5 Group and HDF5 stakeholders

Thanks to the audience