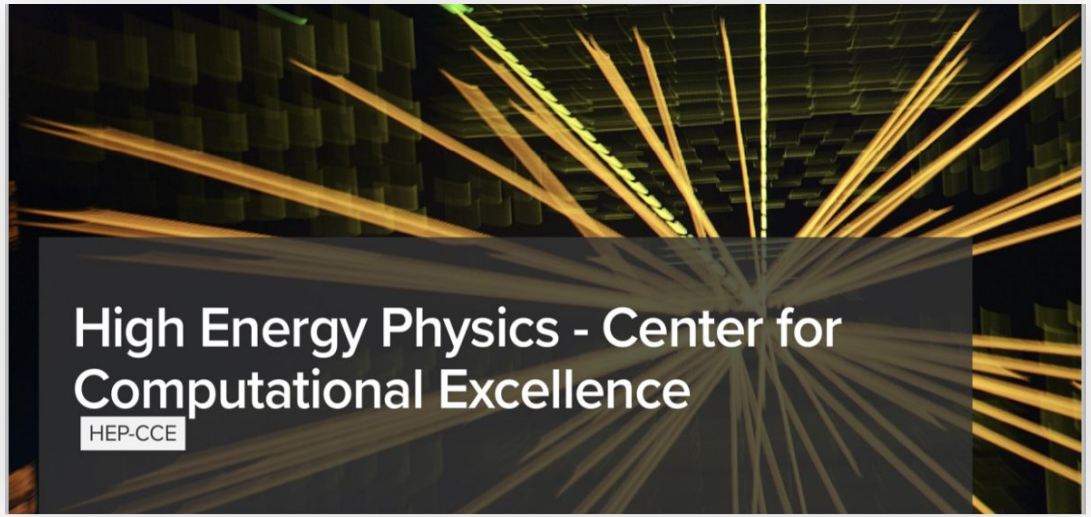
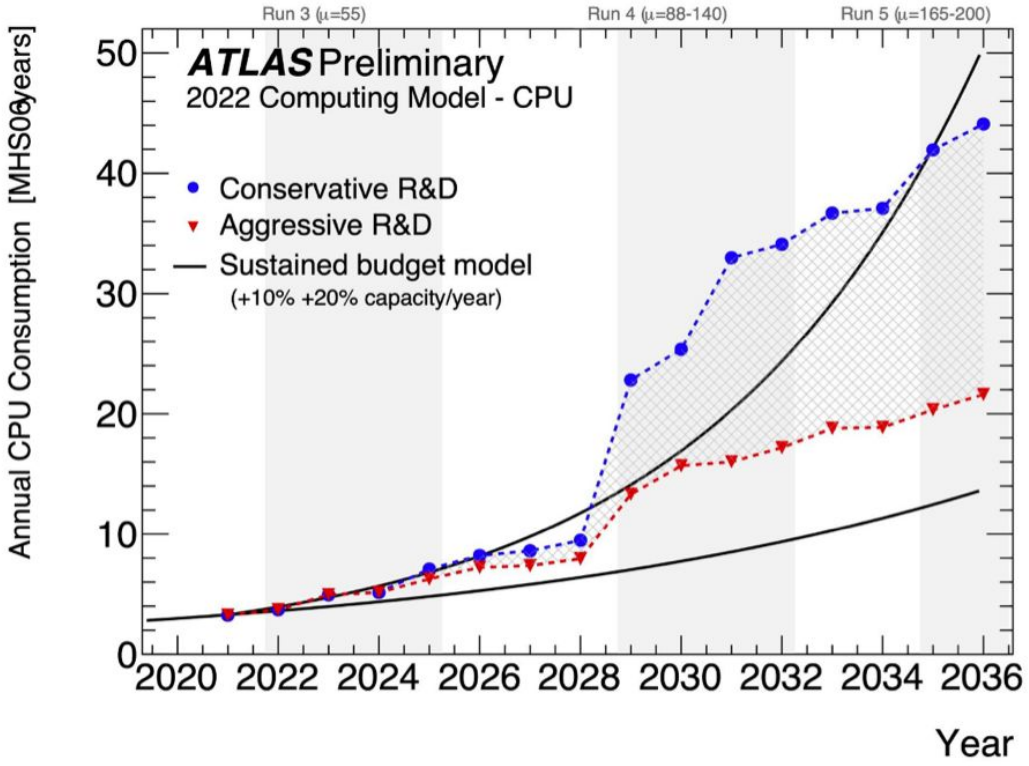
Parallel I/O libraries for managing HEP experimental Data

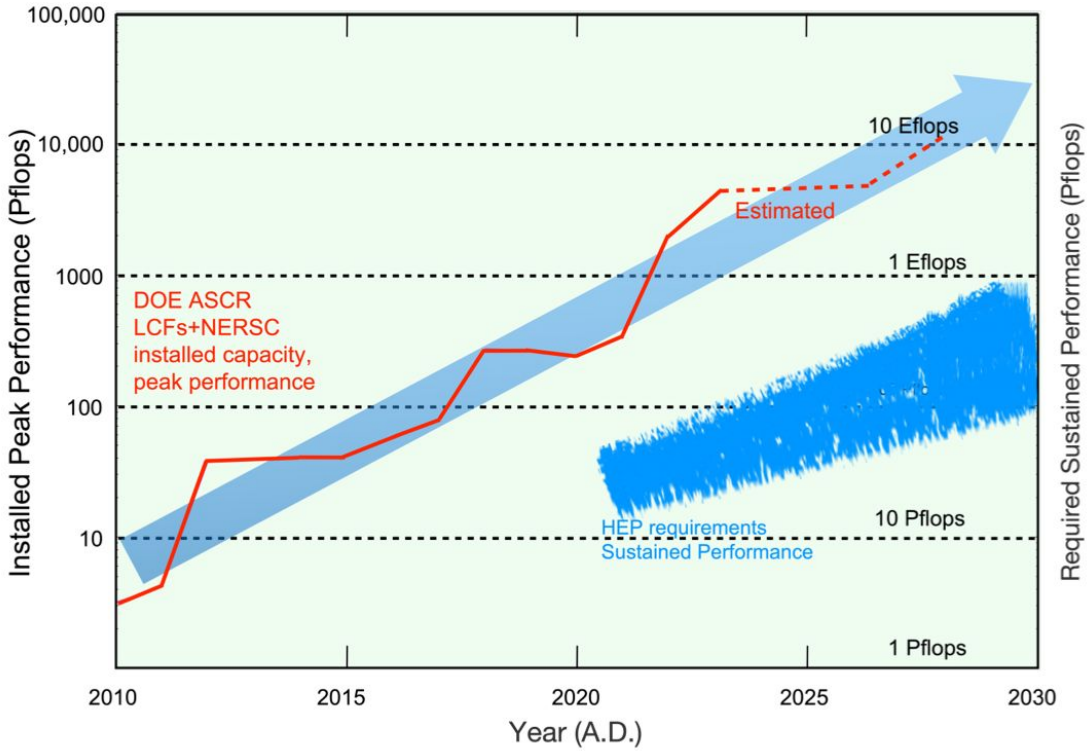
**Amit Bashyal*,Christopher Jones,Kyle Knoepfel,Patrick Gartung,
Peter Van Gemmeren, Saba Sehrish, Suren Byna
on behalf of HEP-CCE**



9th May, 2023

Computing Resources for Future HEP Experiments





CHALLENGE:
Increased computing requirements over coming years.

SOLUTION:
HPCs can fulfill the computing needs through the era of HL-LHC (Run 4) and DUNE.

See Charles Leggett's talk for more details.

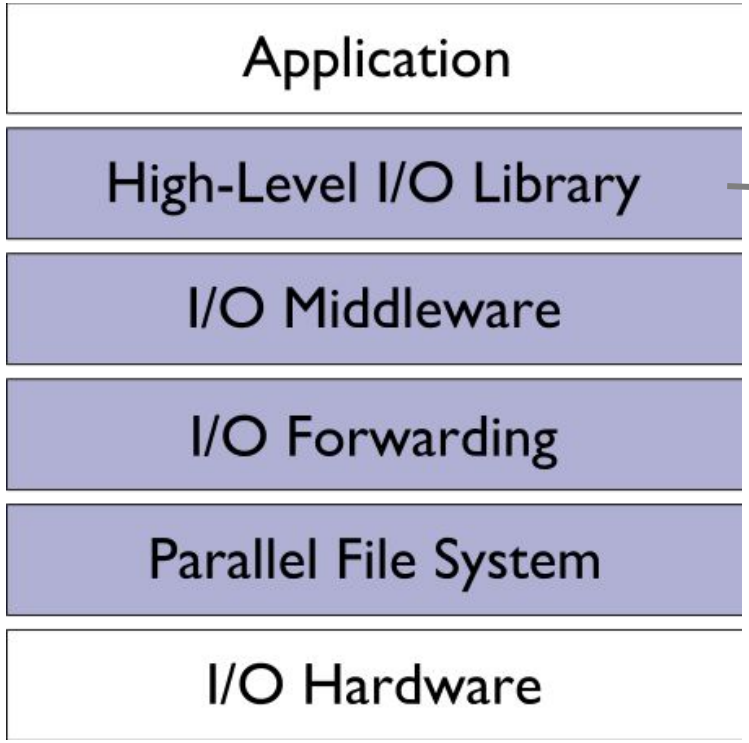
I/O and Storage in the HPCs

- Differences in **HTC** (High Throughput Computing) and **HPC** (High Performance Computing) resources → Cannot directly move HEP computing workflow into HPCs
- HEP-CCE I/O and Storage studies the HEP general computing framework in the HPCs.
  - **Storage**: Writing data in storage format supporting parallel I/O
  - **I/O**: Performing parallel I/O on HEP data with minimal changes on existing computing workflow
  - **Optimization**: Tuning of parallel libraries to optimize the performance
  - **Data Mapping**: I/O performance based on various ways data is written in HPC friendly format

- Test-framework development
  - **Experiment agnostic**: Should work for common HEP data models
  - **Parallel I/O** of the HEP data using MPI (Message Parsing Interface) and HDF5 libraries
  - **Multi-threading** using TBB libraries
  - **HDF5 and MPI parameters** tuning to optimize I/O and storage

HEP Requirements for the HPC Storage Systems

HPCs use **parallel file systems** for data-storage and access.

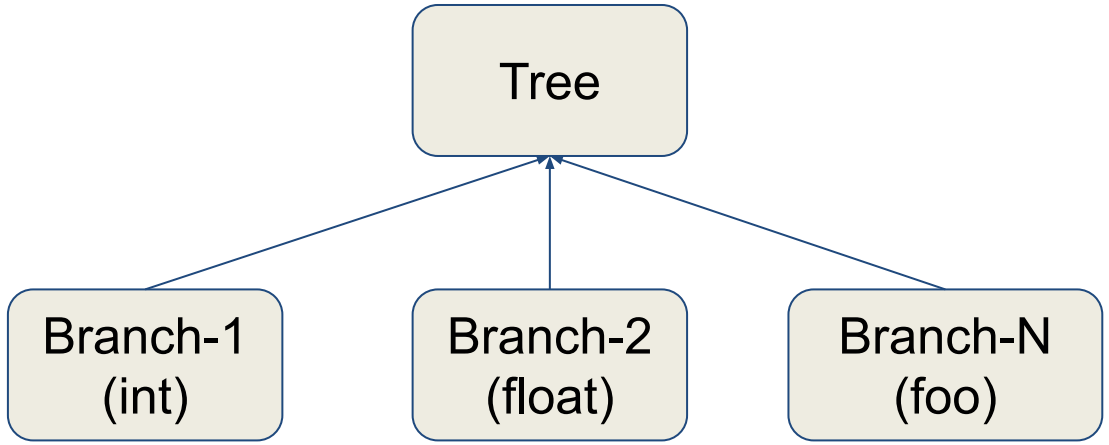


- High Level libraries can hook into the HPC I/O stack.
- Allows us to take advantages of optimizations such as collective I/O

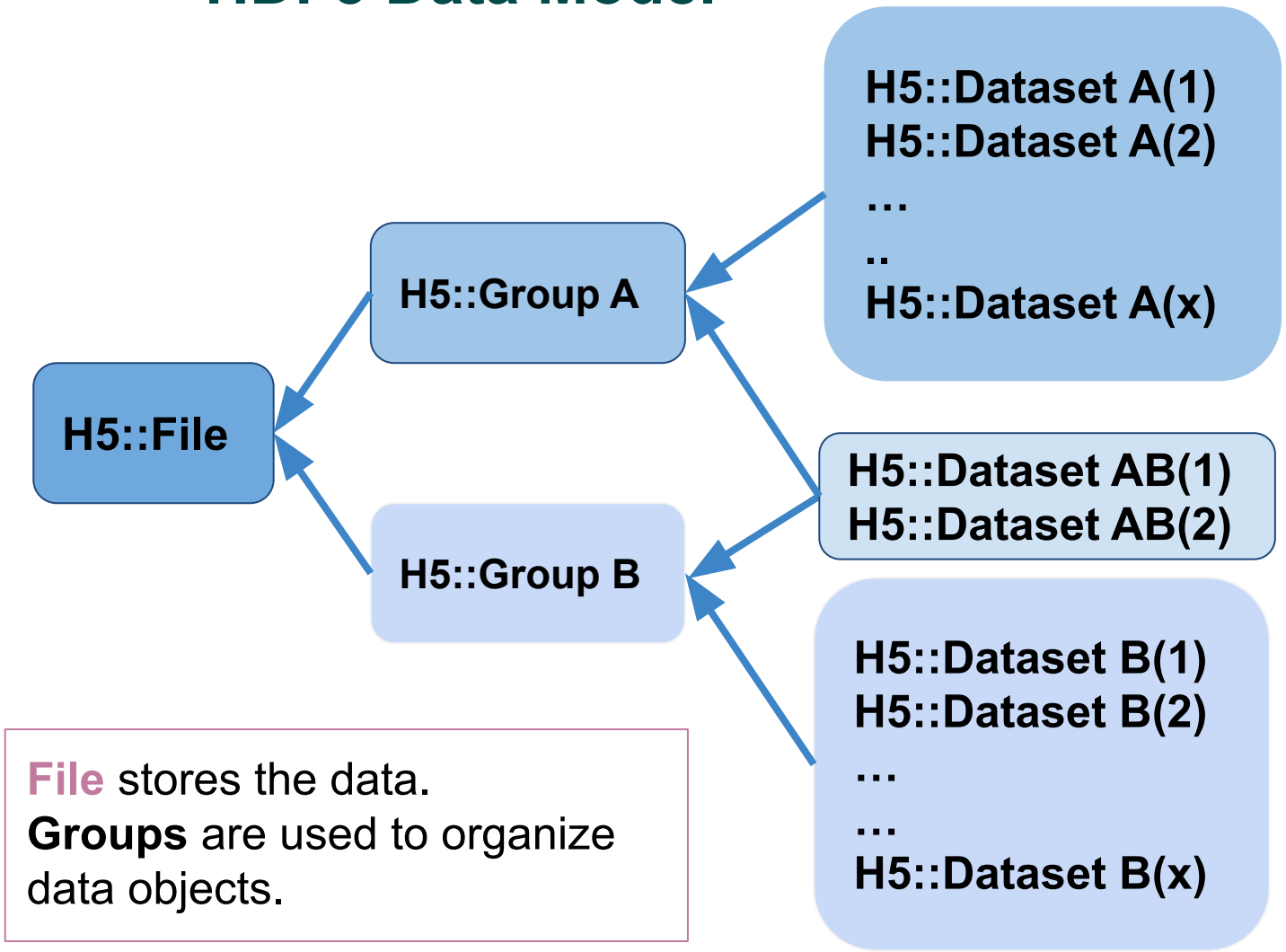
HPCs have an established I/O software stack used to support parallel file system

HEP Data and ROOT Data Model

- ROOT has been workhorse of HEP experiments
  - Data processing, storage and analysis
- HEP **data models are complex**
  - Using C++ language features: pointers, inheritance, polymorphism
- Use ROOT to read and write data into **ROOT::TTree**
  - TTrees store **data of any types** (TBranch)
  - Use of **internal libraries, metadata handlings and functionalities** for efficient and scalable I/O



HDF5 Data Model



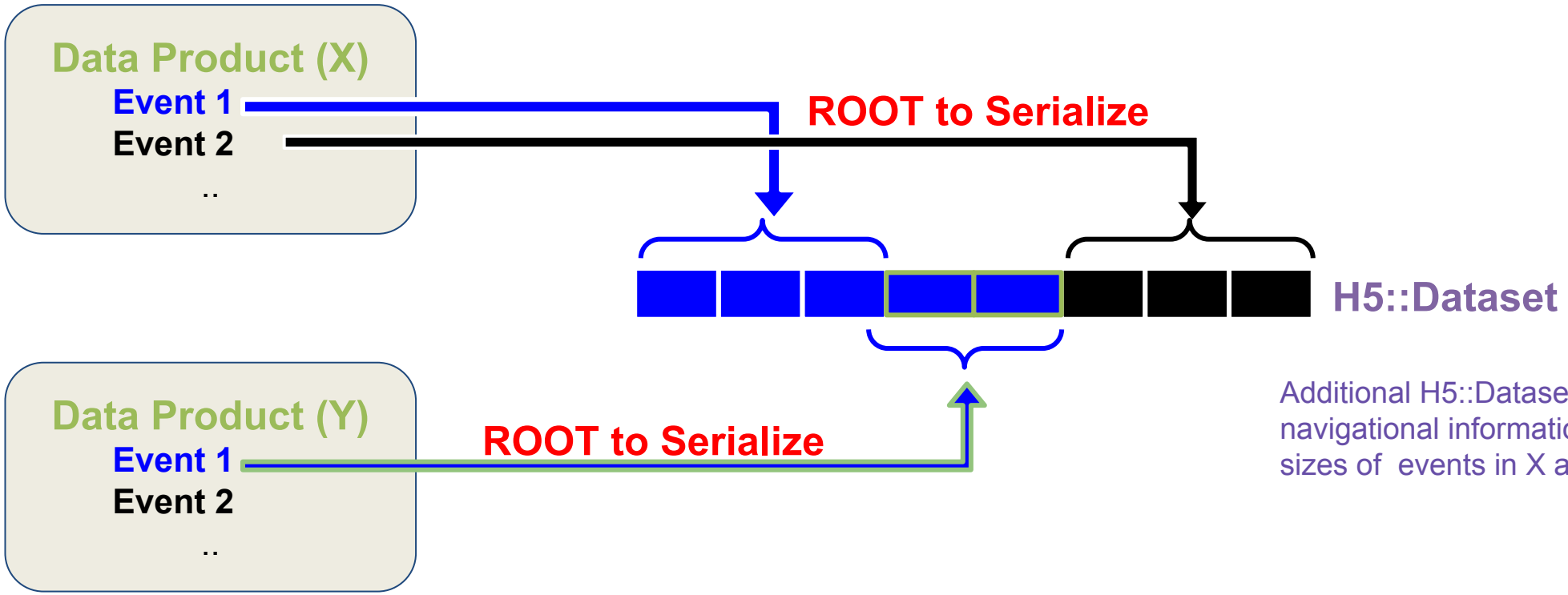
- Data written in **Datasets**.
- **Datasets** can be:
  - **Grouped** together to organize data objects
  - **Shared** among groups
- Store H5::Attributes for metadata

File stores the data.
Groups are used to organize data objects.

- **MPI libraries** implemented to perform **parallel I/O** on the HDF5::Datasets

HDF5 File needs to be opened with the MPI Flag to enable the parallel I/O.

HDF5 as Data Storage Format



Data Products are experiment specific C++ objects usually written in ROOT format.

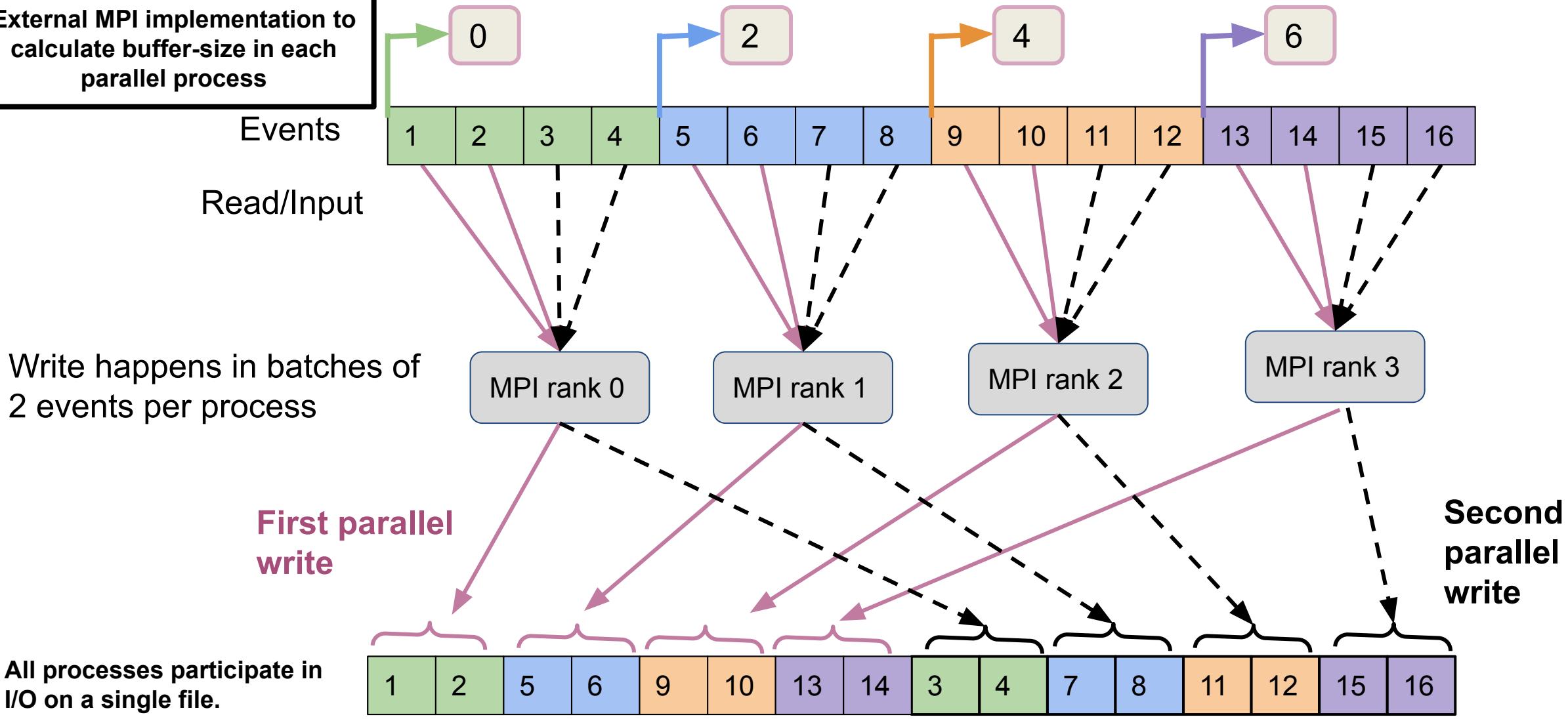
Use **ROOT** as common tool to serialize C++ objects into byte stream array buffers

HDF5 Datasets store serialized data products with mapping optimized for parallel I/O. Mapping is independent of experiments.

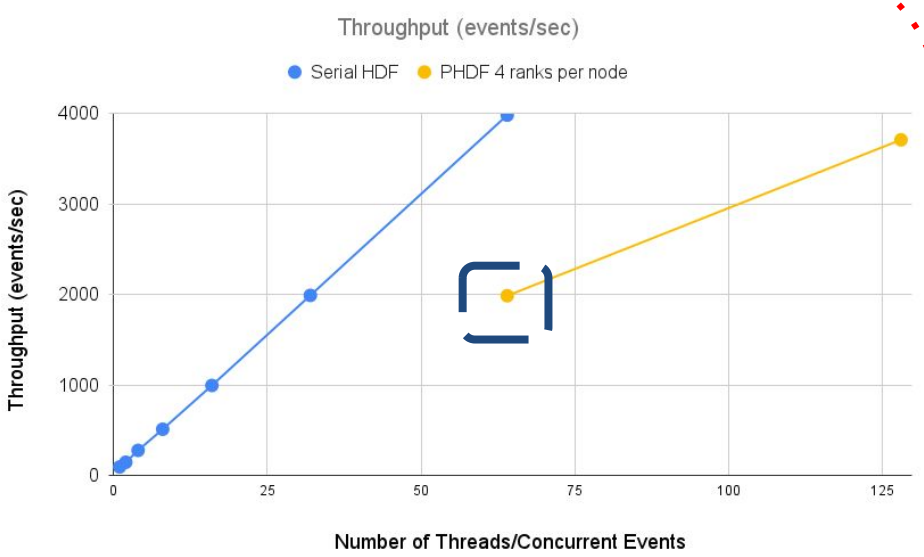
Leveraging HDF5 for parallel I/O

Parallel (Collective) I/O using HDF5

External MPI implementation to calculate buffer-size in each parallel process



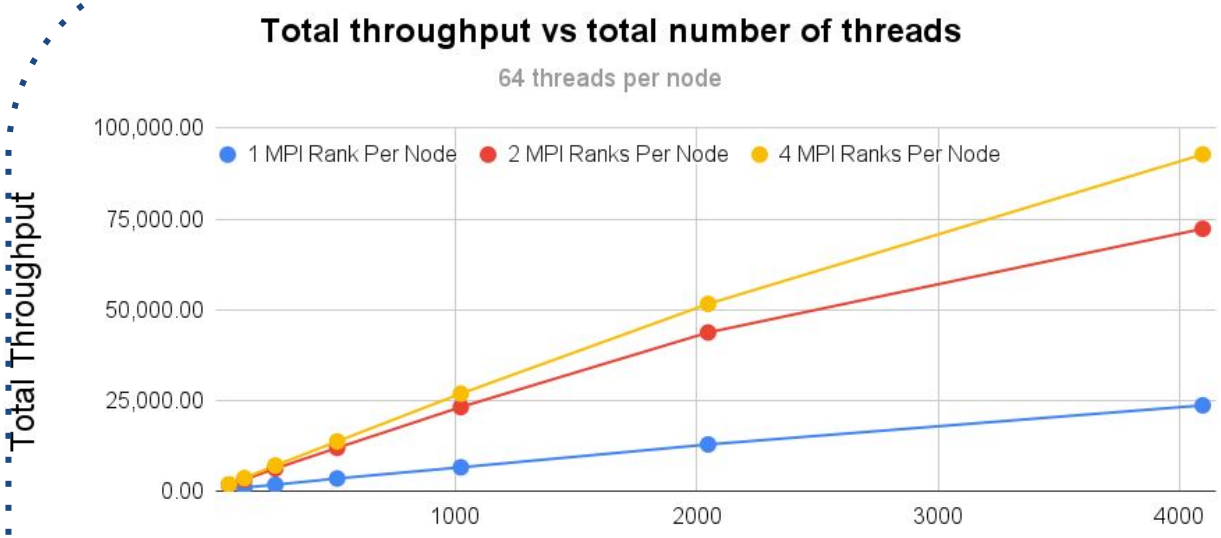
Parallel I/O with HDF5



Test done in a single node
Batch size of 100 events

Throughput = (Number of Events processed)/ (Application Run time)

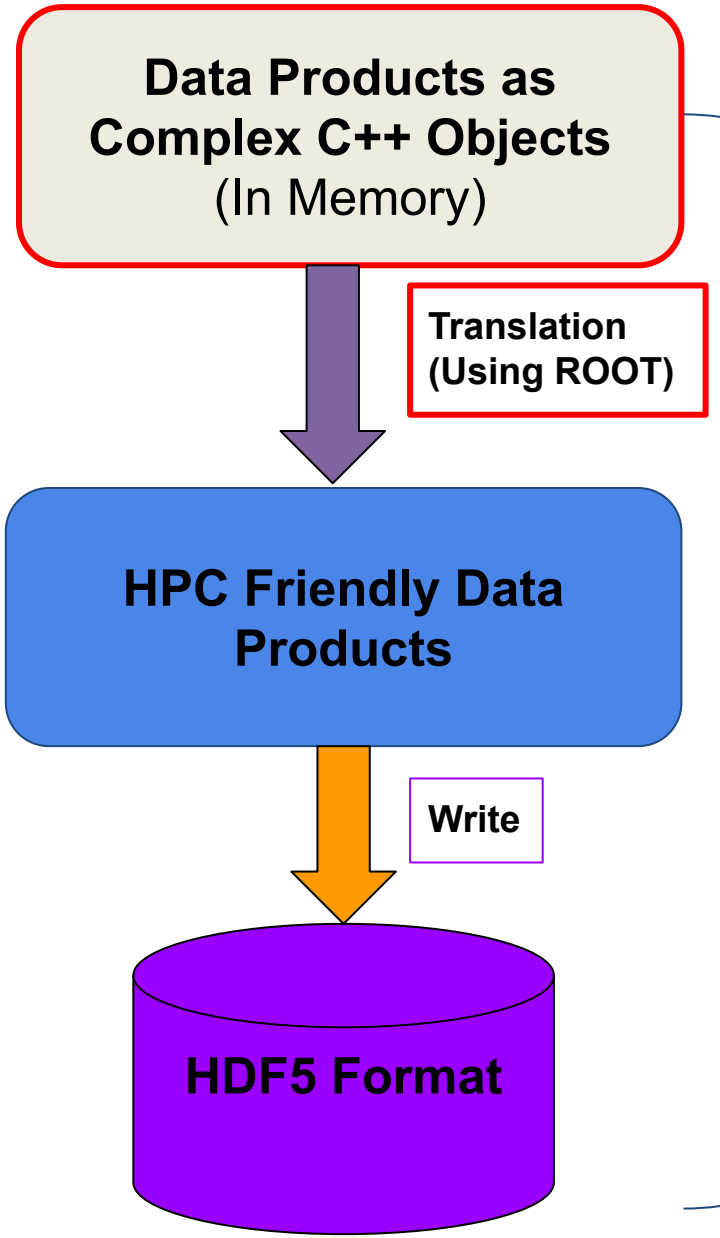
For Parallel I/O:
4 parallel processes
Threads per Rank: #Threads/4



- **Total Throughput**: (Throughput per rank) X(MPI Ranks)
- Test with **64 threads per node**.

I/O Calls	Fraction of Total I/O Time
MPI calls (external to HDF5)	14%
Write data into HDF5 file	32%
Other (including serialization)	54%

Extending the Test Framework



- Use ROOT to serialize HEP data products to make it HPC friendly.
- Collective writing of data into HDF5 file

- HPCs **rely on both CPU and GPUs** to achieve high computational capability.
- Fully utilizing HPC resources requires to use GPU resources as well.
- Serialized data cannot be offloaded into the GPUs directly.
- **Using GPUs might need different data organization**.

Extension to Direct GPU Offloading

Data Products as Complex C++ Objects (In Memory)

Translation (Using ROOT)

HPC Friendly Data Products

Write

HDF5 Format

HEP data needs to serialize/deserialize using ROOT.

Complex objects cannot be offloaded directly into the GPUs.

Offload into GPUs Directly

Design Data Model that is HPC (GPU) friendly

HPC Friendly Data Products (In Memory)

Write

GPU

HDF5 Format

Offload

Future Work

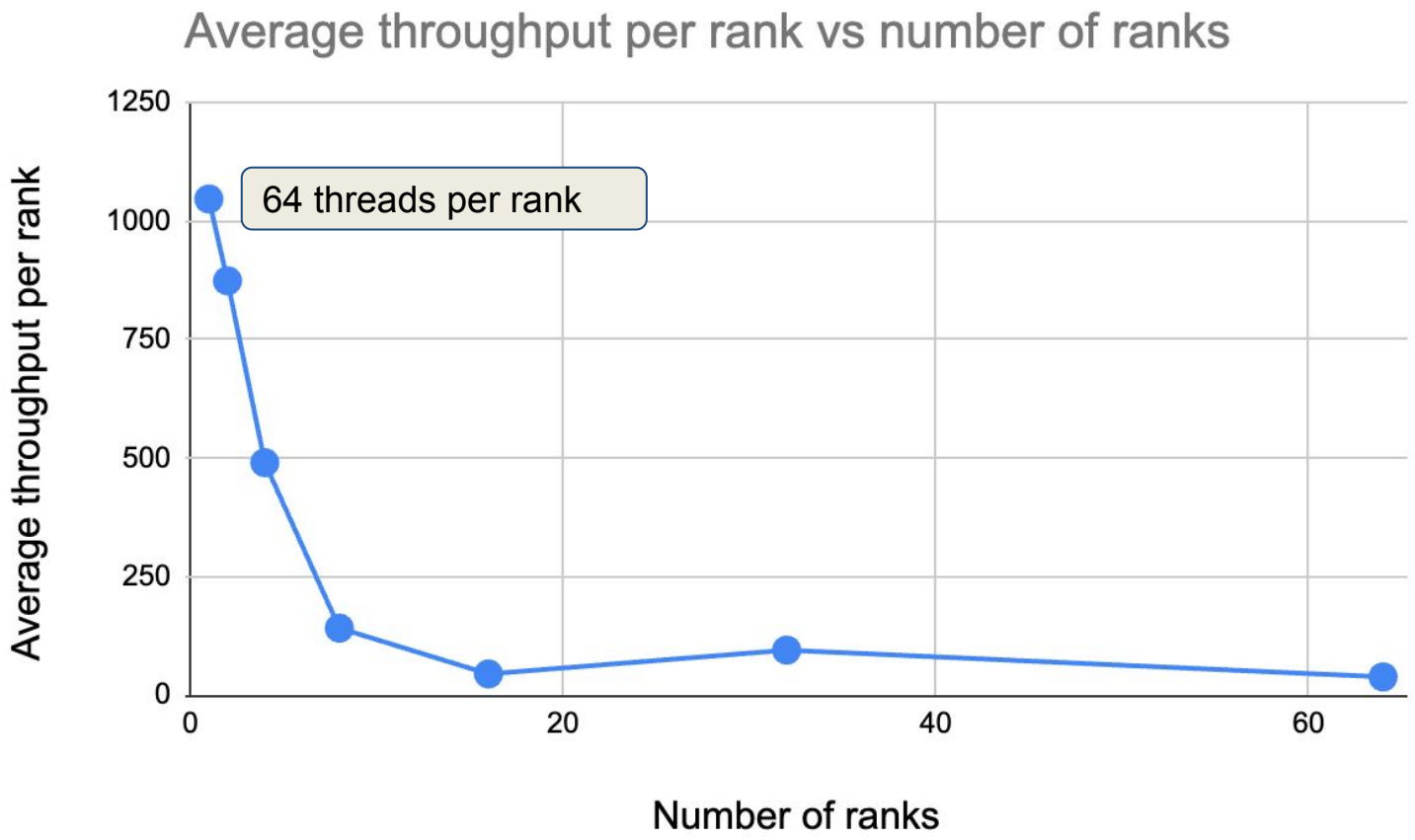
Conclusion

- **Developed test framework** which can perform I/O operation on HEP data using HDF5+MPI libraries
  - Snowmass white paper based on this work presented in the computational frontier
    - [https://arxiv.org/abs/2203.07885]
- **Identify limitations and possible solutions** to enable/optimize parallel I/O for the HEP data
    - Requires to adopt HPC friendly storage like HDF5 → ROOT as a tool to write data into HDF5
    - Data mapping and event batching, tuning of HDF5 parameters
- **Developed mechanism to assist existing HEP experiments** to write existing data models in HPC friendly format
    - ATLAS has developed a proto-type that supports writing data into HDF5 using ROOT serialization
    - Studies with parallel I/O could help to evaluate DUNE use case of DAQ raw data
- **Developed mechanism to adopt new storage backends** will help in future migration (even to ROOT::RNtuple)

ACKNOWLEDGEMENT

This work was supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, High Energy Physics Center for Computational Excellence (HEP-CCE)

BACKUP



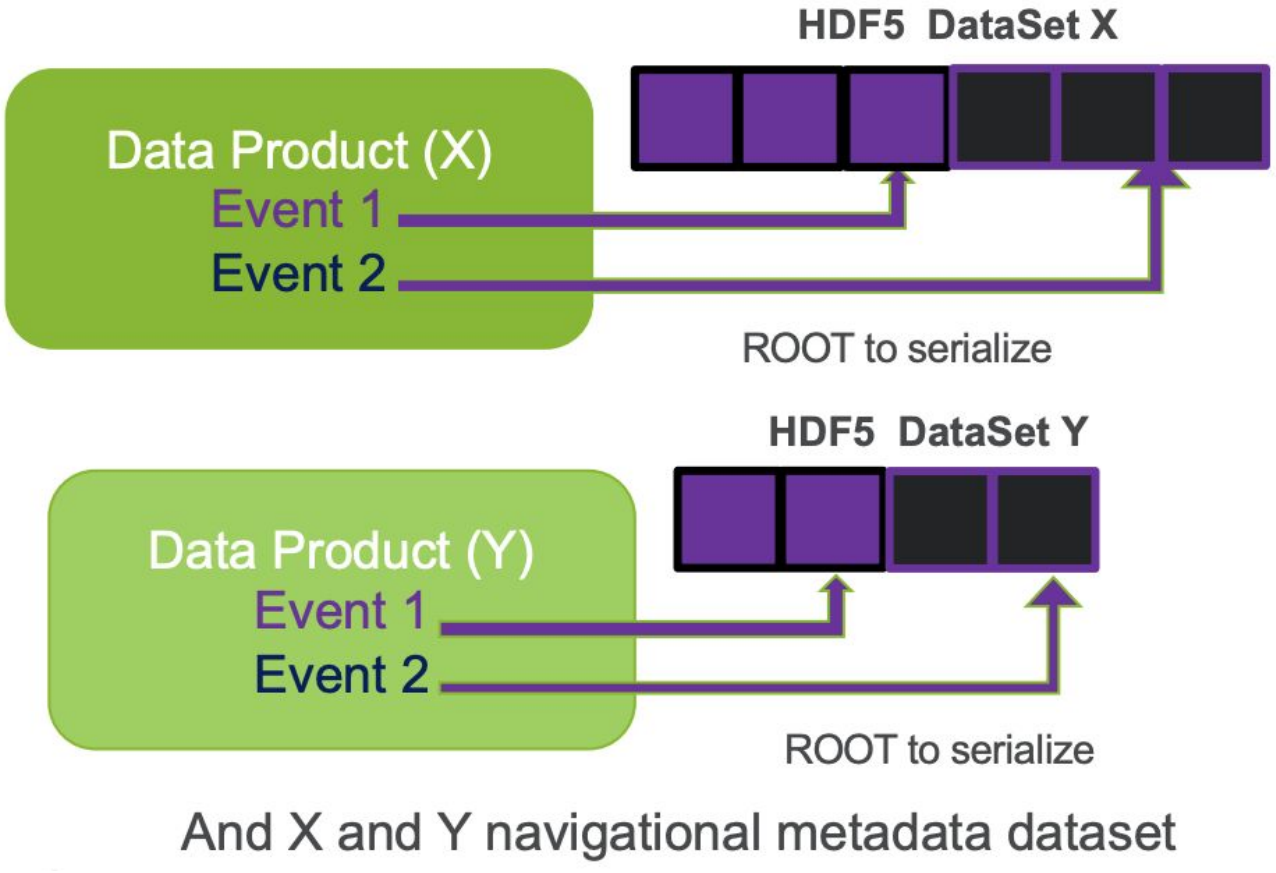
Batch size 100
64 threads per node
Test done in 1 node.

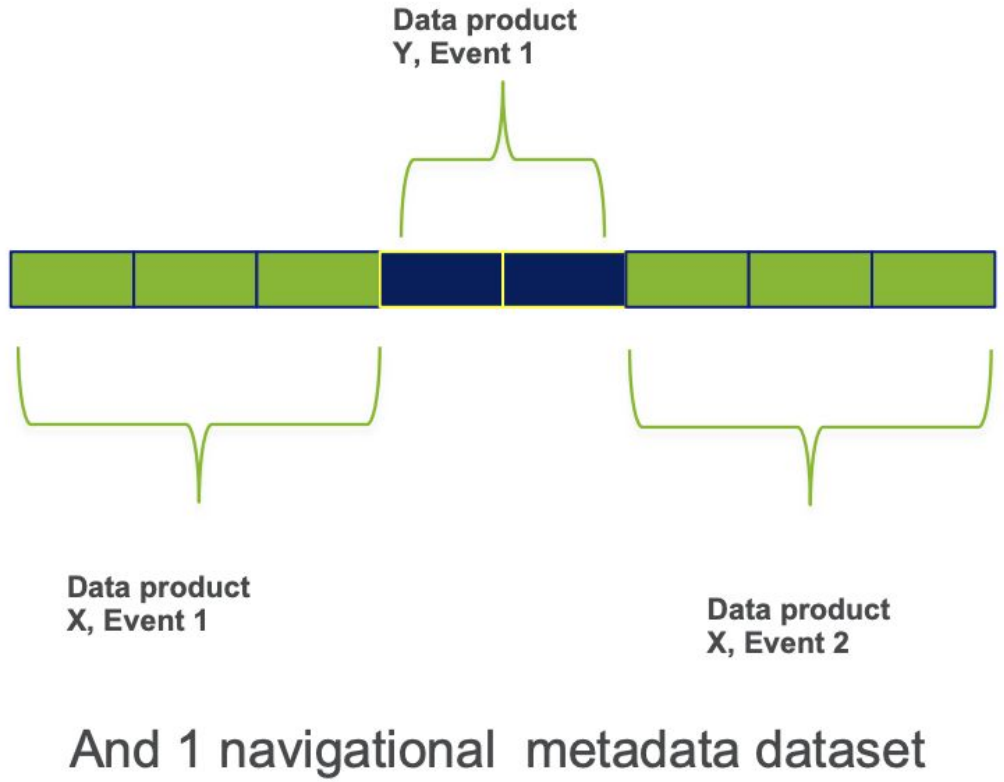
5000 events per thread

Average throughput in each rank to process total events (320000) as a function of total MPI ranks.

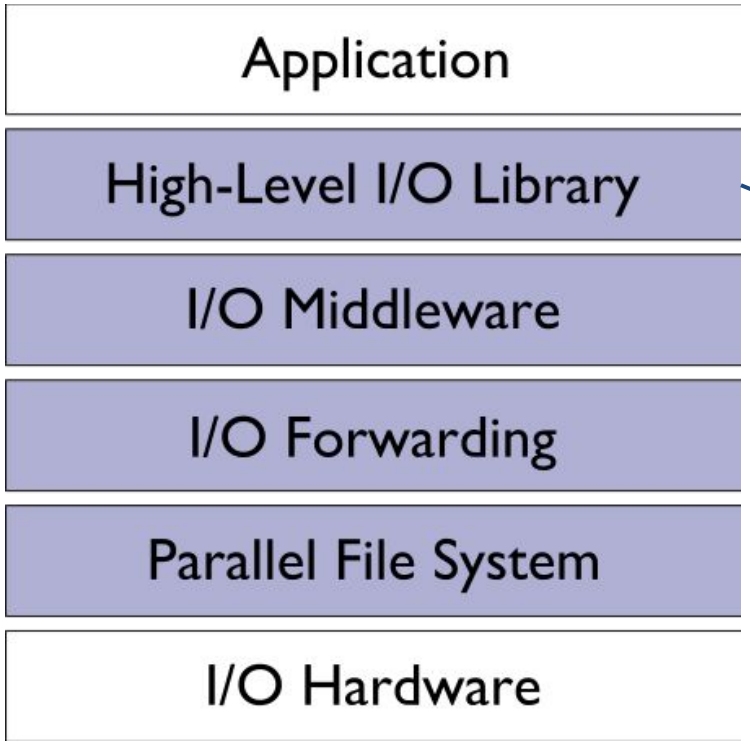
Mapping HEP data to HDF5

Two methods of data mapping methods currently being explored





HEP Requirements for the HPC Storage Systems



HPCs use **parallel file systems** for data-storage and access.

- (Hierarchical Data Format v5) **HDF5**, **PnetCDF** etc provide high level I/O libraries
  - **Interface** between user and low level I/O
  - High Level Libraries to deal with usually complex **mid-level parallel I/O** (like MPI I/O).
  - **Parallel I/O**: Multiple processes performing I/O on a single file in parallel
  - **Process**: An instance of a framework which may support multi-threaded application