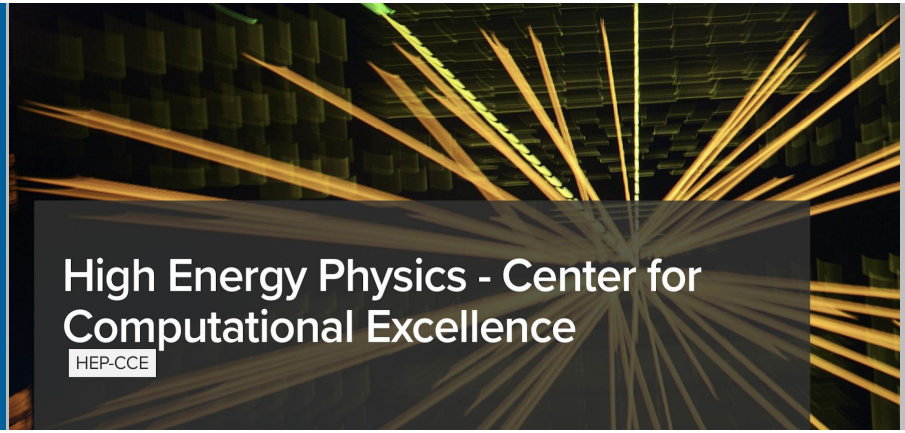
SoA in RNTuple



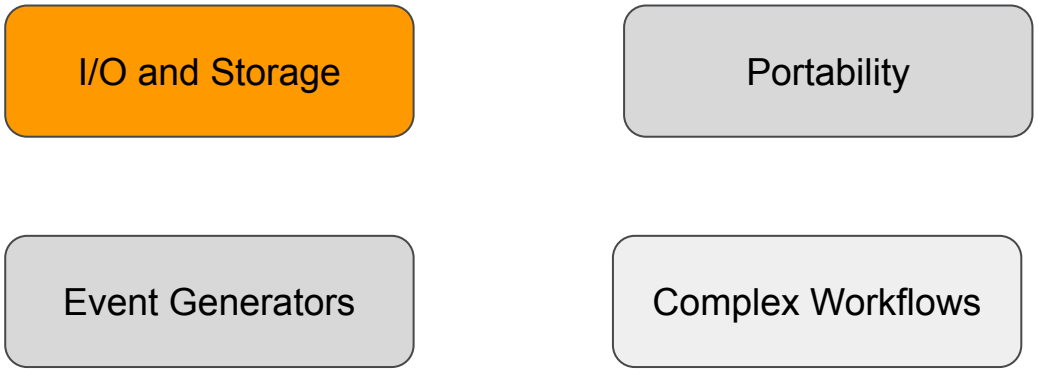
AMIT BASHYAL for HEP-CCE

ARGONNE NATIONAL LABORATORY

November 7, 2023

HEP- Center for Computational Excellence

- Initiated in FY20
  - Argonne, FNAL, LBNL and BNL
- Two main Motivations
  - Major increase in expected HEP experimental data volumes
  - Potential shortage of conventional HEP computing and storage resources to meet demand
  - ASCR computing facilities as a potential solution
    - HEP software stacks needs to be refactored accordingly



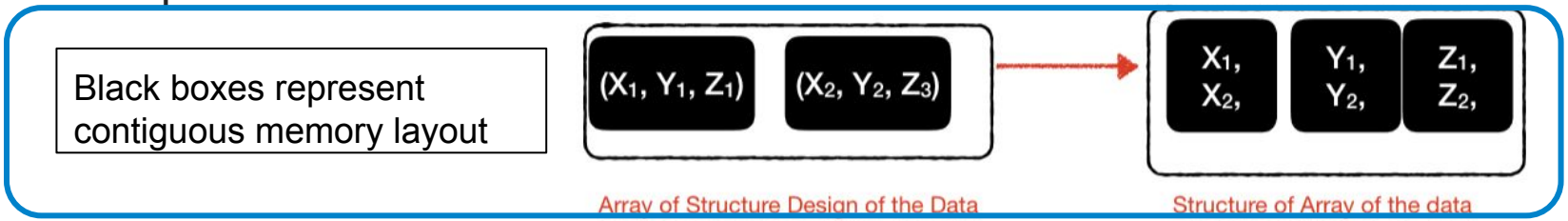
CCE and GPU Friendly Data Model

- CCE - I
  - Storing HEP data in HPC friendly format (HDF5)
  - Survey among experiments to understand efforts being made by experiment to make their data GPU friendly
  - Initiation of GPU Friendly Data Model Design based on survey findings

- CCE -2
  - **Continuation of GPU Friendly Data Model Design**
  - Storage in HDF5
  - Storage in RNTuple (?)

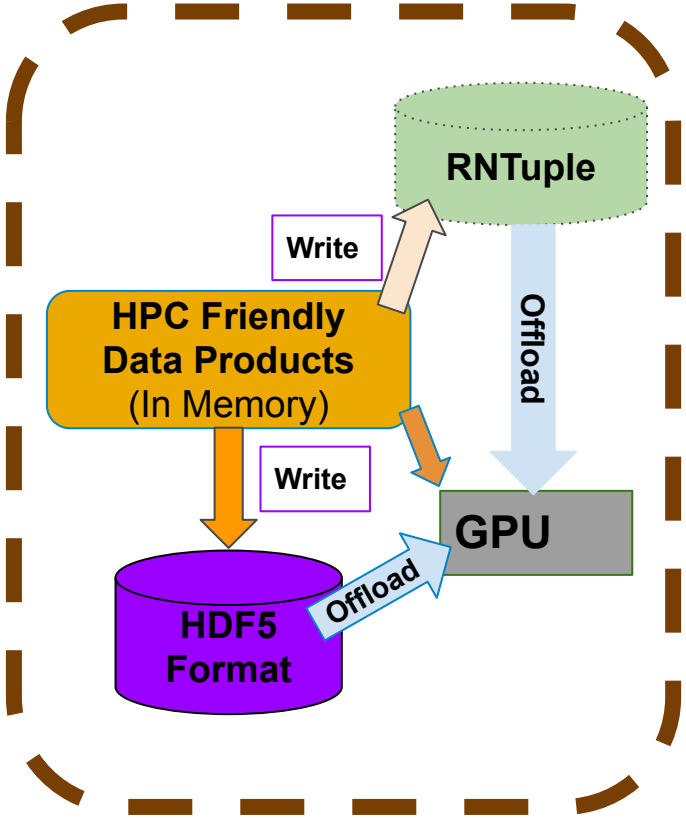
Survey on GPU Friendly Data Model

- HEP-CCE conducted a survey among experiments to understand the efforts made by experiments to make their data HPC friendly
- **Common Challenges**
  - HEP data models are object oriented with complex data models optimized for traditional computing workflow
  - Design based on experimental needs and computational technology at that time
- **Common Solutions**
  - Utilization of Arrays, nested arrays, (Ao)SoA
  - Experiments apply these common solutions according to their use case and experimental needs



CCE and GPU Friendly Data Model

- CCE-1
  - Developed techniques to store HEP data in HPC friendly format like HDF5
    - Object oriented data stored as ROOT serialized buffer
    - HPC friendly storage but not HPC friendly data format
- CCE-2
  - Data model that can be stored in HPC friendly format + Offloaded directly into HPCs (GPUs)
  - Simpler data models that can be stored in HDF5 without serialization
- CCE-2 AND RNTuple
  - RNTuple will also have limited support for data models
  - RNTuple will replace TTree as primary storage
  - Support for SoA would make RNTuple as the GPU friendly storage format for HEP data



RNTuple in Run IV and Beyond

- RNTuple will replace TTree as the primary storage mode
- HPCs (GPUs) will have larger role in HEP computing
- Need for HPC friendly storage format
  - See Snowmass White paper for more info (Link)
- Support for GPU friendly data model in RNTuple in the era of HPCs

- RNTuple is better suited for SoA data model design
  - Supports simpler data types
  - Columnar Data storage
  - Column-wise storage layout → Selective reading of event attributes

SoA in CMS (Matti's Talk)

SoA example

```
namespace reco {

  using PFRecHitsNeighbours = Eigen::Matrix<int32_t, 8, 1>;
  GENERATE_SOA_LAYOUT(PFRecHitSoALayout,
                      SOA_COLUMN(uint32_t, detId),
                      SOA_COLUMN(float, energy),
                      SOA_COLUMN(float, time),
                      SOA_COLUMN(int, depth),
                      SOA_COLUMN(PFLayer::Layer, layer),
                      SOA_EIGEN_COLUMN(PFRecHitsNeighbours,
                                       neighbours),  // Neig
                      SOA_COLUMN(float, x),
                      SOA_COLUMN(float, y),
                      SOA_COLUMN(float, z),
                      SOA_SCALAR(uint32_t, size)  // Number
  )

  using PFRecHitSoA = PFRecHitSoALayout<>;

}  // namespace reco
```

- *Layout* specifies how the memory block is interpreted
  - Can contain scalars, columns, and Eigen vector/matrix
  - Padding at the end of each column to match alignment
- Memory ownership is handled separately
- Want the columns to be visible as columns in TTree/RNTuple

Use Case - (CAF)

- Neutrino Experiments (DUNE, NOvA, MicroBooNE etc)
  - Common Analysis Format (CAF)
  - Columnar Data Model with multiple level of hierarchies and segmentations
  - Data written in HDF5 to utilize MPI based parallel I/O
- Can RNTuple be used to Store CAF data?
  - Ao(SoA) to support (?)

Table 1
NOvA data table organization with one entry per slice.

run	subrun	event	sub-event	distallpngtop	... 35 more ...
433	61	6124	35	nan	
433	61	6124	36	-0.7401	
433	61	6124	37	nan	
433	61	6125	1	nan	
433	61	6125	2	423.633	
433	61	6125	3	-2.8498	

Table 2
NOvA data table organization with one entry per vertex.

run	subrun	event	sub-event	vtxid	npng3d	... 6 more ...
433	61	6124	35	0	0	
433	61	6124	36	0	1	
433	61	6124	36	1	1	
433	61	6124	36	2	5	
433	61	6125	1	0	1	
433	61	6125	3	0	0	

Figure: Tabular representation of the data in NOvA. Each complex variable (SRvertex object in this example (Table 2)) is written in separate tables with the relevant attributes and metadata. Since the indexing or the metadata (the first 4 columns in the tables above)are part of each separate table, it is stored redundantly in the memory to allow faster I/O and is highly compressed in the disk.

GPU Friendly Data Models and SoA

- SoA design by itself does not make data GPU friendly
- I/O requirements for efficient utilization of the GPU resources
  - **Contiguous memory layout** (RNTuple supports this design)
  - **Minimum data transfer** between host and device for **maximum computation** to reduce I/O bottleneck
  - **Parallel (Batch) Processing** of data
    - MPI support? (Showed MPI support for TTree)
  - **Data Alignment**
    - Padded type (See CMS design)
    - Compact type (for sparse data?)
  - **Chunking** (HDF5)?
    - Selective I/O in chunks for large data-sets (DUNE?)
  - Other things…..

SoA Studies in RNTuple

- My **Preliminary** efforts to explore RNTuple as a SoA storage format
  - Part of CCE-2 Efforts
  - **DUNE Trigger level data** as a **base Data model**
  - Design SoA **roughly based on** CMS Patatrack
  - Store Data in HDF5/RNTuple
  - Offload to GPU/CPU
  - Use Darshan to profile I/O (see backup slides)

ProtoDUNE Trigger Data Model Description

```
"file_layout_parameters":{
  "trigger_record_name_prefix": "TriggerRecord",
  "digits_for_trigger_number": 6,
  "digits_for_sequence_number": 0,
  "trigger_record_header_dataset_name": "TriggerRecordHeader",

  "path_param_list":[ {"detector_group_type":"TPC",
                       "detector_group_name":"TPC",
                       "region_name_prefix":"APA",
                       "digits_for_region_number":3,
                       "element_name_prefix":"Link",
                       "digits_for_element_number":2},
                      {"detector_group_type":"PDS",
                       "detector_group_name":"PDS",
                       "region_name_prefix":"Region",
                       "digits_for_region_number":3,
                       "element_name_prefix":"Element",
                       "digits_for_element_number":2} ]
}
```

- Proto DUNE writes data in HDF5
- Data Structures
  - File Attributes as Metadata
  - Data: Trigger Record (Group)
  - Groups also have attributes
  - Raw Data as WIB
    - Stored as HDF5 Data Sets
  - Other Hardware related information

See Kirby's Talk for more detail

DUNE's Trigger Level Data is persisted in HDF5.

Constructing SoA

```
Generate_Arrays(soa_trigger,
        //metadata… (not yet implemented)
        AddArray(uint32_t, wib0,arr_size);
        AddArray(uint32_t, wib1,arr_size);
        AddArray(uint32_t, wib2,arr_size);
        ….
        AddScalar(uint32_t, trig_cand);
        …..

);
```

```
struct soa_simple{
uint32_t wib0[arr_size];
uint32_t wib1[arr_size];
...
..
uint32_t trig_cand;
...
};
```

```
<lcgdict>
 <class name="soa::DUNETriggerData" />
 <class name="soa::DUNETriggerData::soa_trigger" />
<class name="soa::DUNETriggerData::soa_trigvec" >
</class>
</lcgdict>
```

To Generate Dictionary

```
Generate_Arrays(soa_trigvec,
        Addvector(uint32_t, wib0);
        Addvector(uint32_t, wib1);

        Addvector(uint32_t, wib2);
        Addvector(uint32_t,wib3);
        //and other stuff
);
```

Or Write into vectors

Persisting SoA in RNTuple

- Persisting Simple type arrays
  - RNTuple (Only supports ROOT VERSION >6.26)
  - This work done with 6.26
  - Written with 4 threads (easier implementation)

Simple Type arrays stored as (std::array<T,size>)

```
*************************************** NTUPLE ***************************************
* N-Tuple : NTuple                                                                    *
* Entries : 4                                                                         *
*****************************************************************************************
* Field 1          : Trig0 (soa::DUNETriggerData::soa_trigger)                        *
*   Field 1.1      : wib0 (std::array<std::uint32_t,50000>)                           *
*     Field 1.1.1  : std::uint32_t (std::uint32_t)                                    *
*   Field 1.2      : wib1 (std::array<std::uint32_t,50000>)                           *
*     Field 1.2.1  : std::uint32_t (std::uint32_t)                                    *
*   Field 1.3      : wib2 (std::array<std::uint32_t,50000>)                           *
*     Field 1.3.1  : std::uint32_t (std::uint32_t)                                    *
*   Field 1.4      : wib3 (std::array<std::uint32_t,50000>)                           *
      *     Field 1.4.1    : std::uint32_t (std::uint32_t)
```

Output from RNTupleReader::PrintInfo()

SoA array types with vector types

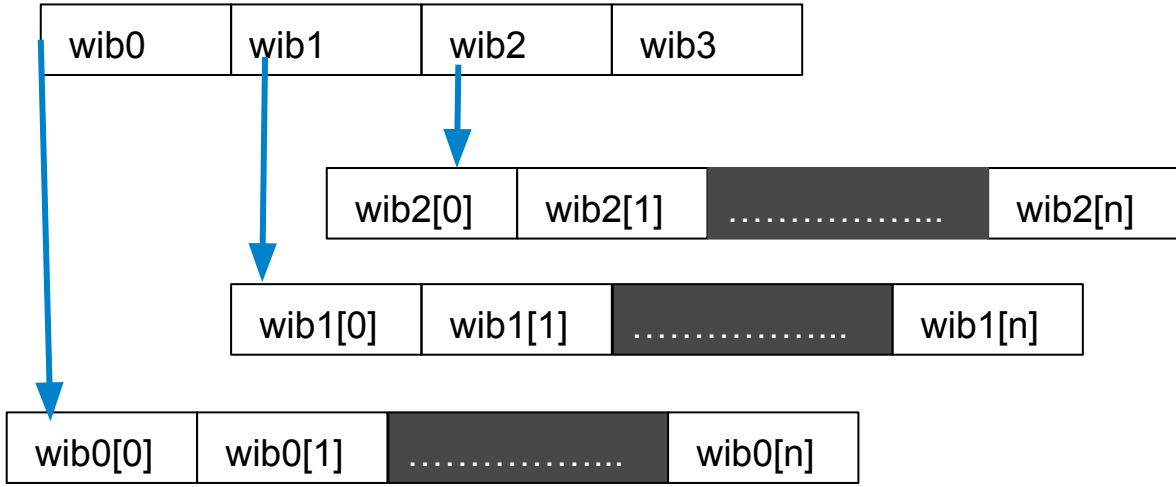
```
auto _val =
entries[i]->Get<DUNETriggerData::soa_trigvec>("Trig0"
);
for(int j=0;j<arr_size;j++){
Double_t arr[10];
prng->RndmArray(10,arr);
uint32_t v0 = uint32_t(arr[0]*10);
uint32_t v1 = uint32_t(arr[1]*10);
_val->wib0.emplace_back(v0);
_val->wib1.emplace_back(v1);
```

Output from RNTupleReader::PrintInfo()

```
********************************** NTUPLE ***********************************
* N-Tuple : NTuple                                                          *
* Entries : 4                                                               *
*****************************************************************************
* Field 1         : Trig0 (soa::DUNETriggerData::soa_trigvec)               *
*   Field 1.1     : wib0 (std::vector<std::uint32_t>)                       *
*     Field 1.1.1 : _0 (std::uint32_t)                                      *
*   Field 1.2     : wib1 (std::vector<std::uint32_t>)                       *
*     Field 1.2.1 : _0 (std::uint32_t)                                      *
*   Field 1.3     : wib2 (std::vector<std::uint32_t>)                       *
*     Field 1.3.1 : _0 (std::uint32_t)                                      *
*   Field 1.4     : wib3 (std::vector<std::uint32_t>)                       *
*     Field 1.4.1 : _0 (std::uint32_t)
```

Memory Layout in RNTuple

```
struct soa{

std::vector<uint32_t>wib0;
std::vector<uint32_t>wib1;
std::vector<uint32_t>wib2;
std::vector<uint32_t>wib3;
//More stuff....
//Meta data related stuff...

};
```



Variables With Padding

Create a custom class for a 128 byte alignment.

```
template<typename T, int bytealignment=128, int offset = bytealignment-sizeof(T)%bytealignment>
class myval{
    public:
      myval():_val(NULL){}
       myval(const T& init_val):_val(init_val){
       };
       ~myval(){};
   // OTHER STUFF
    private:
       char pad[offset];
       T _val;
};
```

Fix the alignment at 128 bytes by default

Persisting SoA with Padded variable is having issues…
Complains does not know “myval” type

myval<T>&

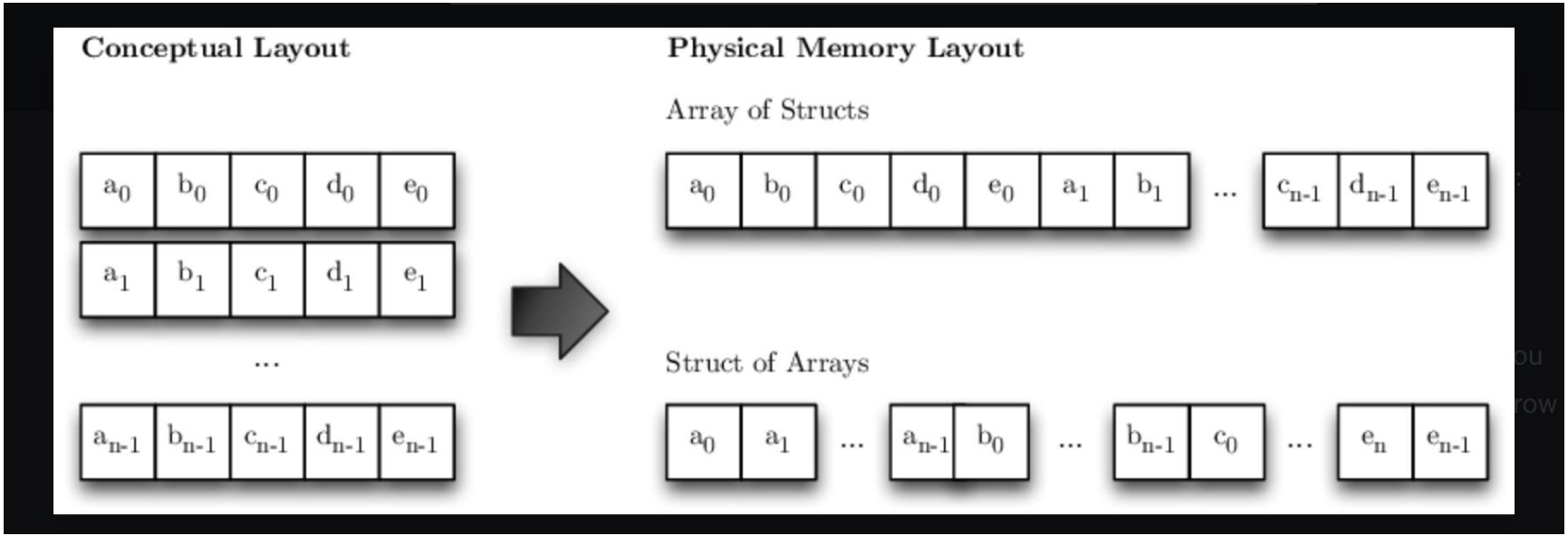
assign values.

Outlook

- GPU Friendly data model design : One of the work area of CCE-2
- SoA as one of the data model candidates (based on CMS work)
  - Based on Survey result
- SoA in RNTuple and HDF5
  - RNTuple as a storage system for HEP data in the HPC era
  - Darshan to profile I/O
  - Data model should be offloaded in GPU directly
  - Common enough to be followed by all HEP experiments
  - Design motivated by the survey done in CCE-1

BACKUP

Structure of Arrays



SoA array types with Vectors

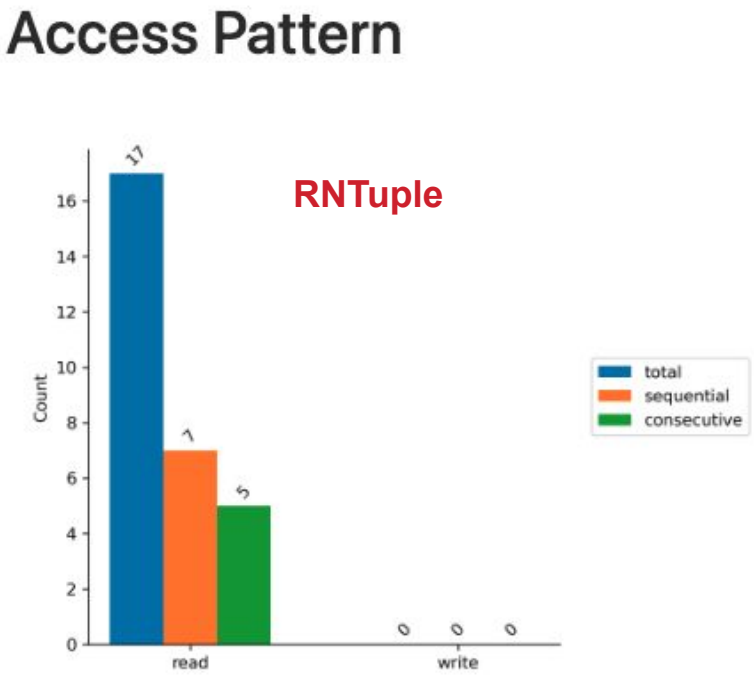
```
for(int i=0;i<NWriterThreads;i++){
threads.emplace_back([i,&entries,&ntuple](){
static std::mutex gLock;
auto prng = std::make_unique<TRandom3>();
prng->SetSeed();
//DUNETrigger::soa_trigvec d_trigvec;
auto _val = entries[i]->Get<DUNETriggerData::soa_trigvec>("Trig0");
for(int j=0;j<arr_size;j++){
Double_t arr[10];
prng->RndmArray(10,arr);
uint32_t v0 = uint32_t(arr[0]*10);
uint32_t v1 = uint32_t(arr[1]*10);
_val->wib0.emplace_back(v0);
_val->wib1.emplace_back(v1);
```

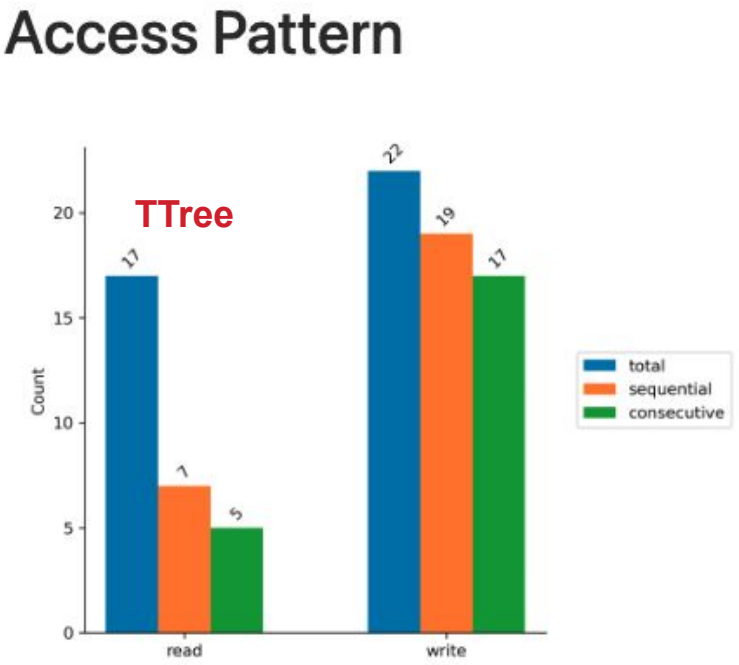
```
struct soa_trigvec{
  std::vector<uint32_t>wib0;
  std::vector<uint32_t>wib1;
  std::vector<uint32_t>wib2;
 /*
  All the way upto wib9;
  And 3 more information related to
metadata and hardware inference
*/

  uint32_t trig_scalar;

};
```

```
_val->wib2.emplace_back(uint32_t(arr[2])*10);
_val->wib3.emplace_back(uint32_t(arr[3])*10);
```

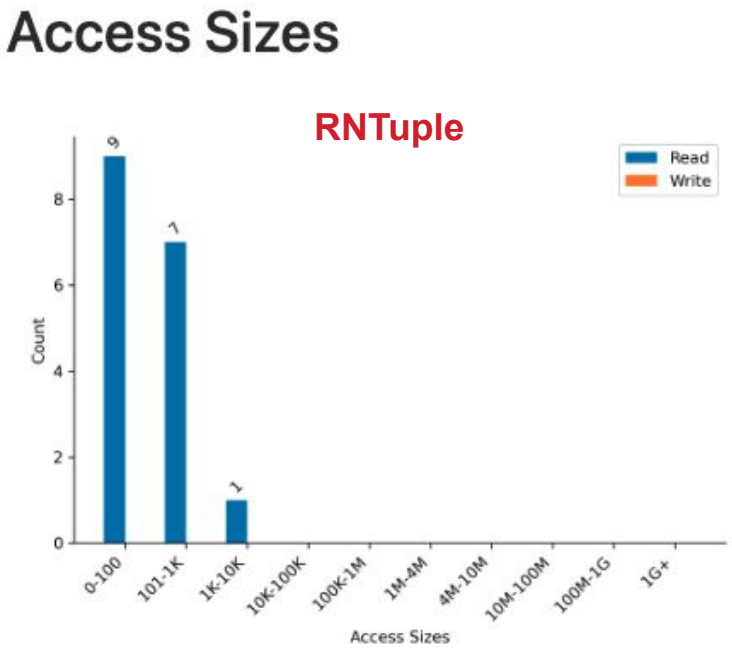
This wont work btw…

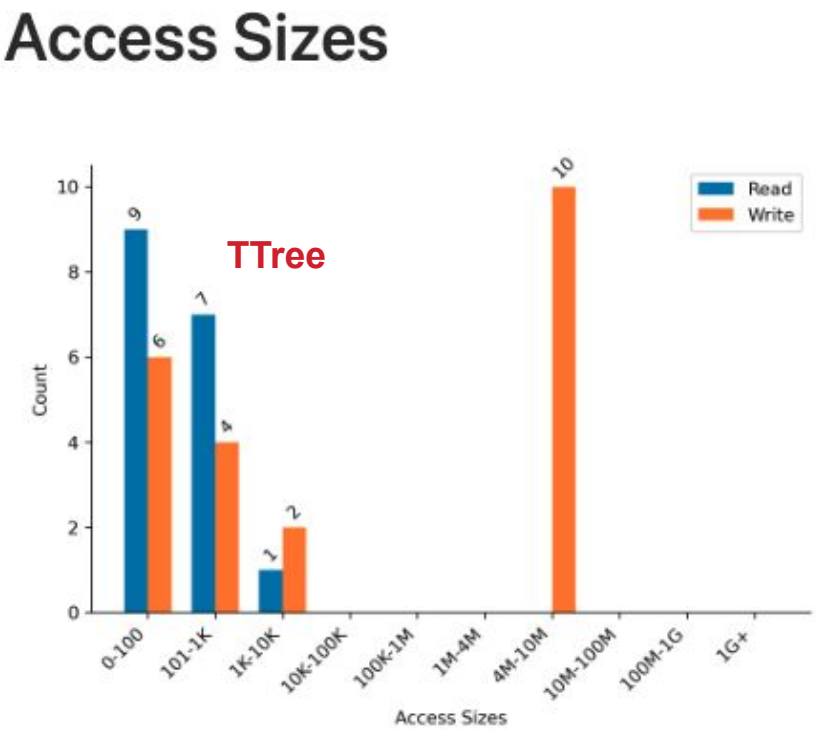
POSIX


Access Pattern
RNTuple
Count
0
2
4
6
8
10
12
14
16
17
7
5
0
0
0
read
write
total
sequential
consecutive

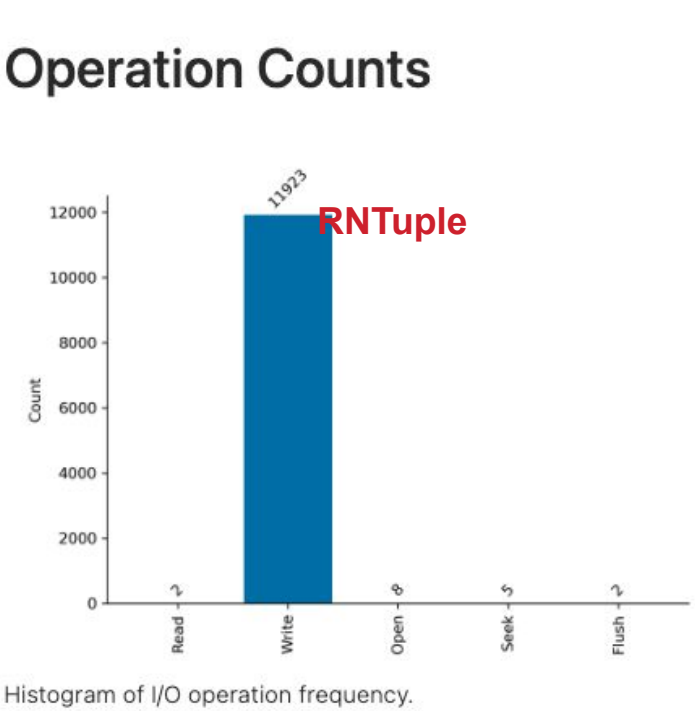

Access Pattern
TTree
Count
0
5
10
15
20
17
7
5
22
19
17
read
write
total
sequential
consecutive

POSIX

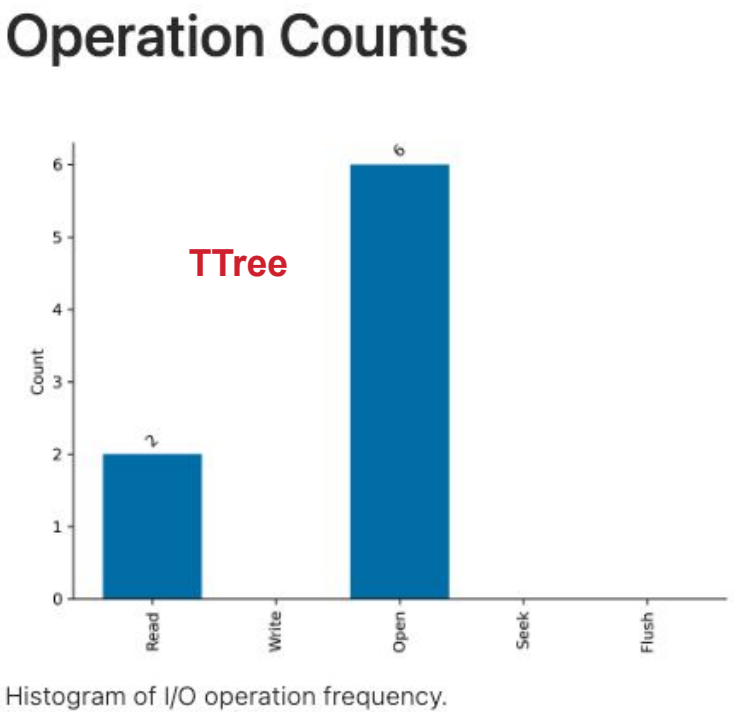

Access Sizes
RNTuple
Read
Write
Count
0
2
4
6
8
9
7
1
0-100
101-1K
1K-10K
10K-100K
100K-1M
1M-4M
4M-10M
10M-100M
100M-1G
1G+
Access Sizes


Access Sizes
TTree
Read
Write
Count
0
2
4
6
8
10
9
6
7
4
1
2
10
0-100
101-1K
1K-10K
10K-100K
100K-1M
1M-4M
4M-10M
10M-100M
100M-1G
1G+
Access Sizes

STDIO : Note Very different Scales



Histogram of I/O operation frequency.



Histogram of I/O operation frequency.

POSIX (TOP) and STDIO OVER-VIEW (BOTTOM)

files accessed	8
bytes read	3.19 KiB
bytes written	0 Bytes
I/O performance estimate	7.72 MiB/s (average)

RNTuple

Overview

files accessed	8
bytes read	8 Bytes
bytes written	50.66 MiB
I/O performance estimate	77.29 MiB/s (average)

Overview

files accessed	9
bytes read	3.19 KiB
bytes written	50.46 MiB
I/O performance estimate	1437.81 MiB/s (average)

TTree

Overview

files accessed	6
bytes read	8 Bytes
bytes written	0 Bytes
I/O performance estimate	0.04 MiB/s (average)