Fine Grained I/O and Storage Infrastructure

Barnali Chowdhury and Peter Van Gemmeren
Argonne National Laboratory

The Second Wire-Cell Reconstruction Summit

April 11th, 2024

An Effort to Optimize Memory

- Memory Profiling - analyzing memory usage and finding memory problems
  - for ProtoDUNE-SP simulation and reconstruction (Production IV)
  - for DUNE-FD2-VD simulation (1x8814_3view_30deg production)
  - various memory profilers were used, for example, PRocess MONitor, MAP, and massif

- Overall memory usage remain stable
  - at different stages of simulation and reconstruction chain

- Memory allocation is
  - approximately 7GB for simulation
  - approximately 4GB for reconstruction

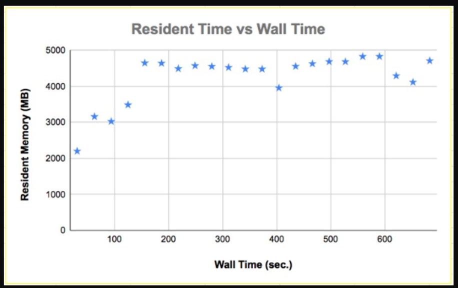
- No significant memory leak was found
  - Roughly 90% of total memory allocation comes from higher level system calls

- The memory estimation is based on ProtoDUNE….DUNE is challenging

Memory Profiling for ProtoDUNE SP

ProtoDUNE G4 Stage

- **/dune/app/users/barnali/prmon/prmon_3.0.2_x86_64-static-gnu94-opt/bin/prmon --interval 30 --disable netmon -- lar -c protoDUNE_refactored_g4.fcl -o g4.root gen.root**
- Processing 10 events takes 11.38 minutes
- This includes
  - G4 stage 1: Geant 4 simulation
  - G4 stage 2: Light simulation without space charge effect

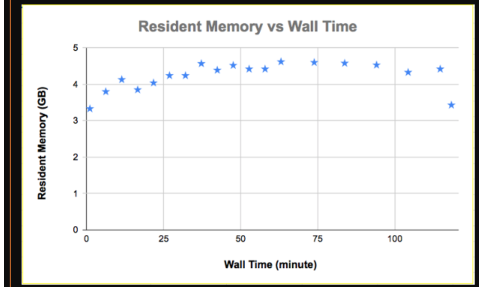


Wall Time (sec.)	Resident Memory (MB)
31	2199
63	3160
94	3026
125	3486
156	4648
187	4642
218	4491
249	4574
559	4829
590	4829
621	4293
652	4115
683	4709

4

ProtoDUNE Detsim Stage

- **/dune/app/users/barnali/prmon/prmon_3.0.2_x86_64-static-gnu94-opt/bin/prmon --interval 30 --disable netmon -- lar -c protoDUNE_refactored_detsim.fcl -o detsim.root g4.root**
- Processing 10 events take 118.33 minute
- This includes
  - Detsim Stage 1
  - Detsim Stage 2

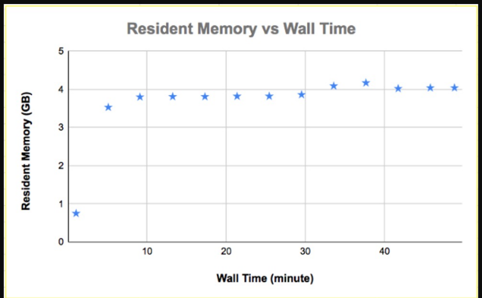


Wall Time (minute)	Resident Memory (GB)
1.05	3.33
6.21	3.8
11.38	4.13
16.55	3.85
21.7	4.04
26.88	4.24
32.05	4.24
37.21	4.57
42.38	4.39
47.55	4.52
52.71	4.42
57.88	4.42
63.05	4.62
73.83	4.6
83.71	4.58
94.05	4.53
104.3	4.33
114.71	4.42
118.33	3.43

5

ProtoDUNE Reco Stage 1

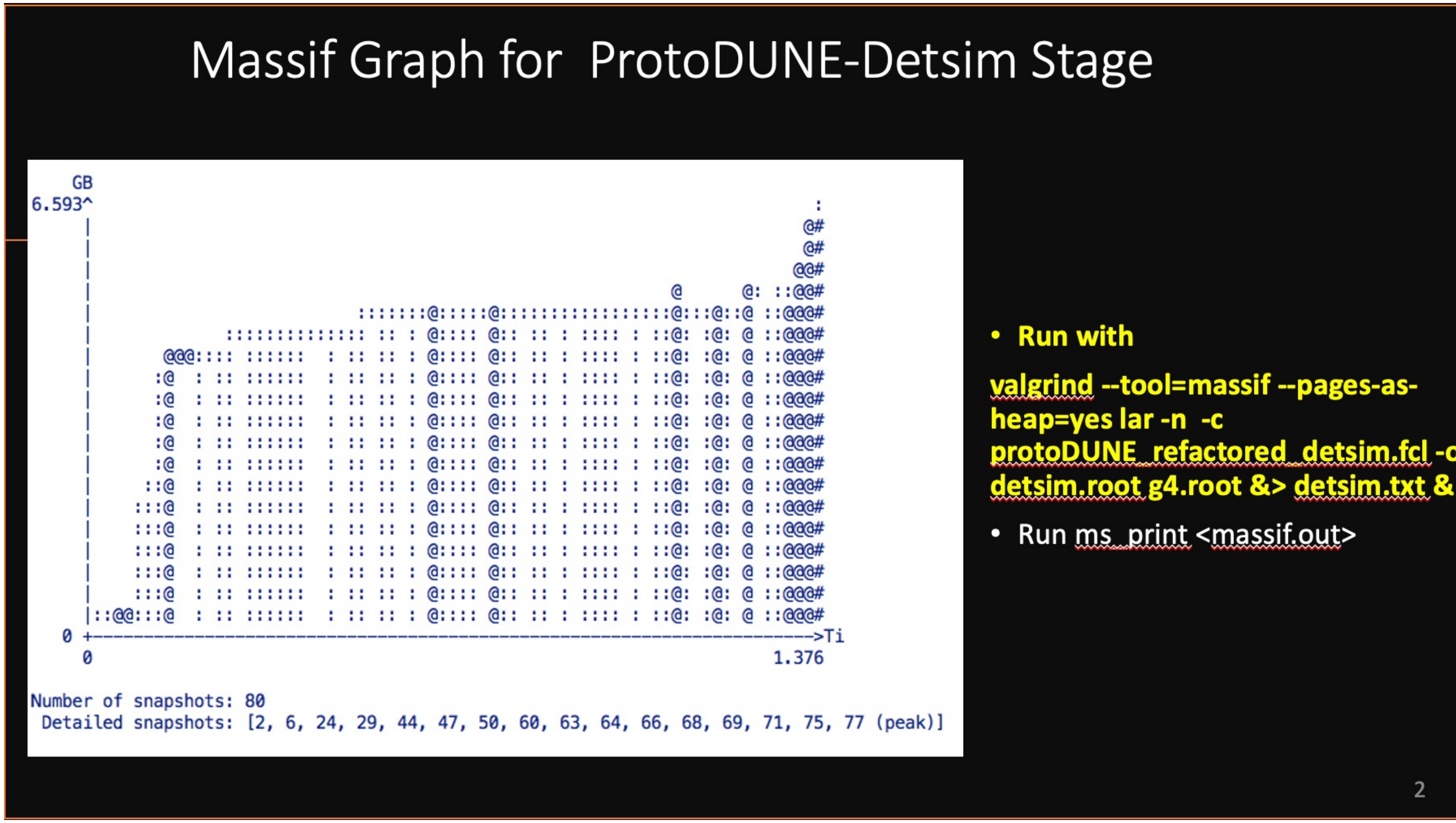
- **/dune/app/users/barnali/prmon/prmon_3.0.2_x86_64-static-gnu94-opt/bin/prmon --interval 60 --disable netmon -- lar -c protoDUNE_refactored_reco_stage1.fcl -o reco.root detsim.root**
- Processing 10 events take 48.81 minute



Wall Time (minute)	Resident Memory (GB)
1.03	0.75
5.1	3.53
9.1	3.8
13.23	3.81
17.3	3.81
21.36	3.82
25.42	3.82
29.5	3.86
33.56	4.09
37.63	4.17
41.7	4.02
45.76	4.04
48.81	4.04

6

Memory Profiling for ProtoDUNE SP



- Simulating ProtoDUNE-SP Detsim stage with 6 APAs and for 10 events consumes ~6.5 GB.
- Simulating FD with 150 APAs will make large impact on memory consumption

Need for Careful Memory Handling

- DUNE reco/sim jobs are 4-6 GB/process (this is for ProtoDUNE-SP)
  - CPU estimates include a factor for memory use
  - At some sites, a high memory job, such as 6 GB/process, must reserve multiple cores
  - 1 MWC = an 11 HS06 core with 2 GB of memory
  - With 1MWC DUNE is unable to utilize all its cores in a machine
  - Leaving cores idle is waste of CPU resources. In other words, wasting money.
  - Framework improvements to improve memory management would have a large impact on resources!

- Raw data comes in 10-30 MB chunks – ~ 1APA/CRP
  - With 150 APAs, raw data for FD is ~25 MB x 150 = 3750 MB (large memory consumption)
  - Raw data then follows signal processing
  - Art Framework not designed to handle objects of this size

- Simulation is even more challenging as we go to larger detectors (FD currently simulates over 12-24 APA)

- Need a major framework redesign to handle smaller chunk of data

DUNE Existing Framework Challenges

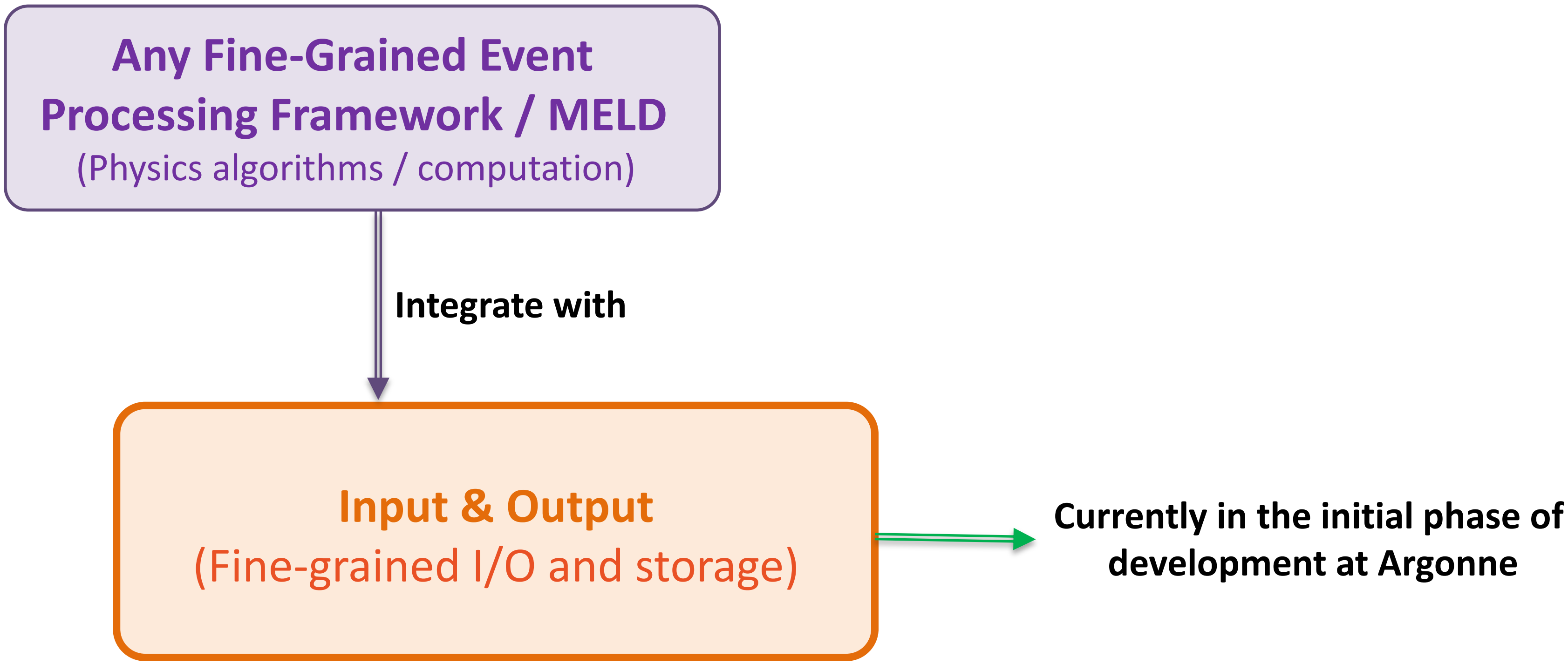
- Three challenges for DUNE computing
  - On disk and in memory data organization for neutrino-oscillation physics
  - Efficient data-handling for Supernovae reconstruction and analysis
  - Timely end-analysis of large-scale parameter estimation

- Existing computing frameworks are based on collider-physics concepts
  - Data products are based on run, subrun, and event
  - An ATLAS/CMS event is significantly smaller, less than a few MB
  - But a DUNE event is more than a factor of 1000 times larger in size
  - DUNE requires to process some of data at the APA level to keep memory in check

- Suggest we need a new Fine Grained Event Processing Framework to overcome current framework limitations

- MELD, Kyle's LDRD project, should be able to break apart events into smaller chunks for more granular event processing

Overview of Framework



Fine-grained I/O & Storage Framework

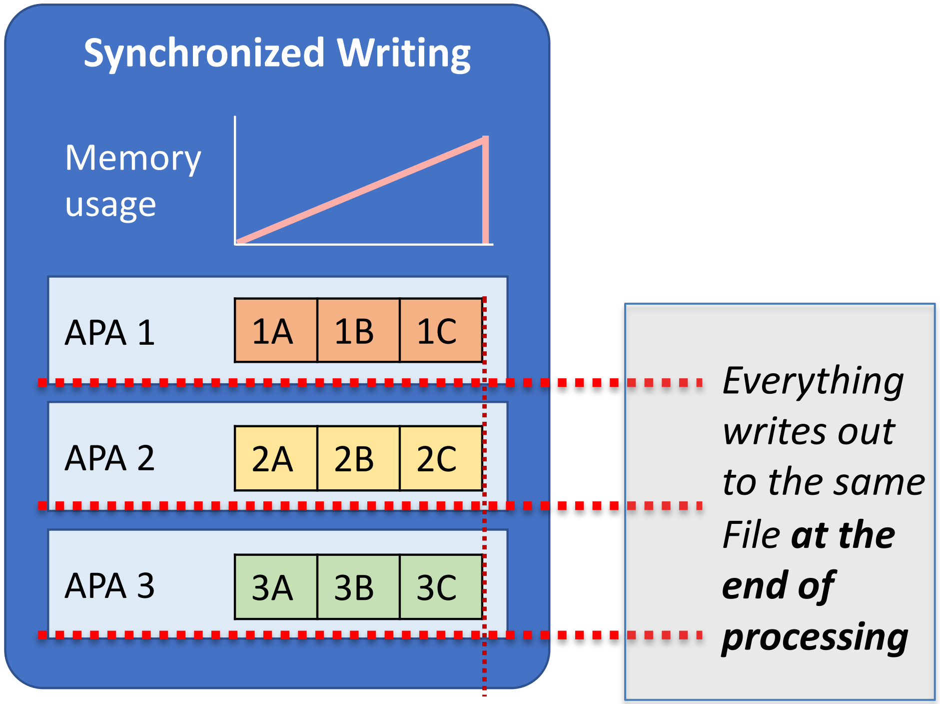
Output from MELD/any Fine-Grained Event Processing Framework

(Physics algorithms / computation)

Fine-grained I/O and storage

- MELD breaks down 1TR in to sub events (preferably in APA) and process the sub events with physics algorithms
- But MELD alone is not going to solve memory issue unless we can write/read 1 APA to/from a file
- Or we can store data in finer container and read back in segments
- **By breaking down persistence storage in smaller segments, the framework I/0 will meet all the challenges above**

Writing out in Art: Memory Problem



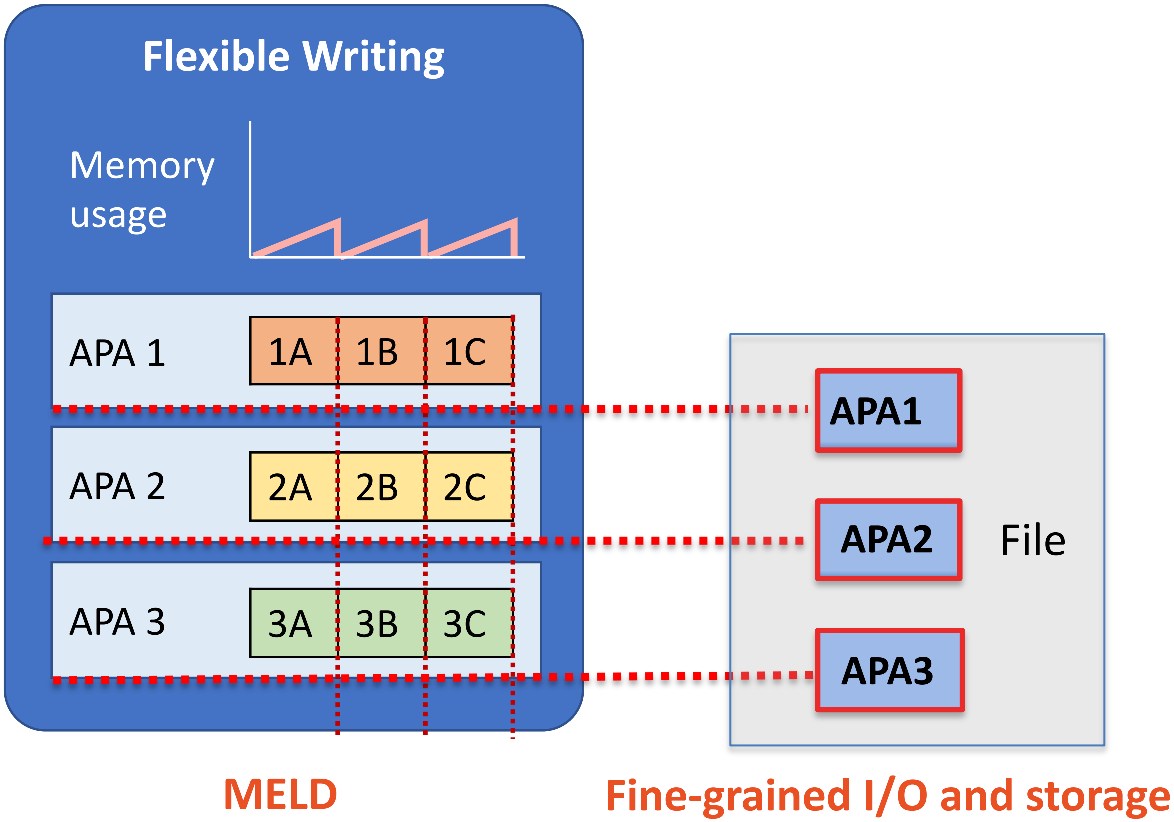
- Art writes out data to a file at the end of processing a Trigger Record.
- Data is stored in a single TTree with 1 entry/event (TR)
- Branches are read "on-demand"

An example of Art writing out Large Data Object

- Event data objects like “simb::MCParticles” are large
- The container has a size of 1.7GB/10 = 170 MB (found in TTree)
- Writing data objects this large consumes significant memory
- The large vector was created by simulating ProtoDUNE SP 6 APA

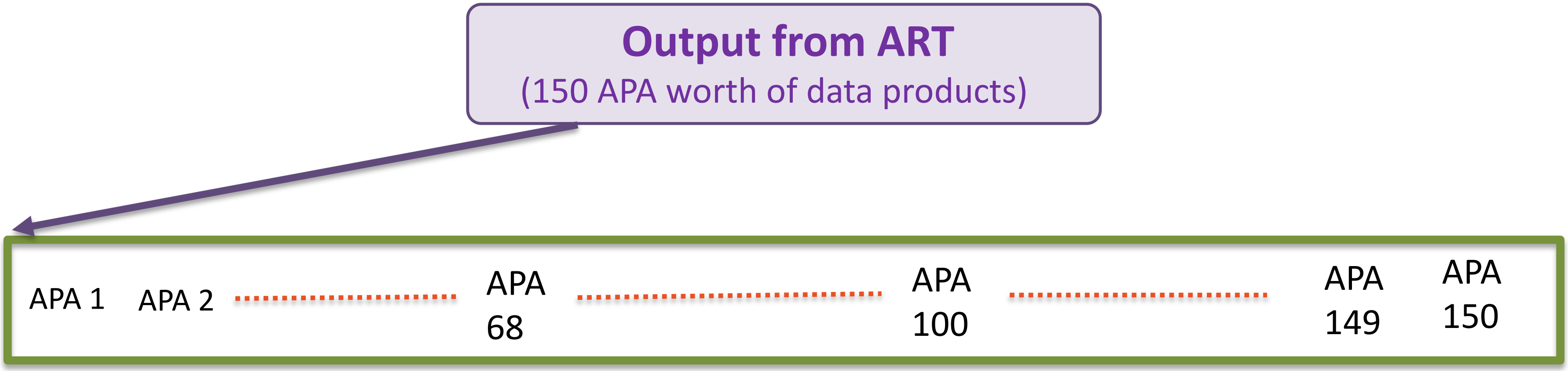
```
*............................................................................*
*Br  279 :simb::MCParticles_largeant__G4.obj.ftrajectory.ftrajectory :       *
*         | vector<pair<TLorentzVector,TLorentzVector> > ftrajectory[simb:   *
*         | :MCParticles_largeant__G4.obj_]                                  *
*Entries :       10 : Total  Size= 1772826741 bytes  File Size  =  669242335 *
*Baskets :       10 : Basket Size=      16384 bytes  Compression=   2.65     *
*............................................................................*
```

MELD & Fine Grained I/0 : Fixing Memory Problem



- MELD / any FGPF with fine grained I/O is flexible to write data in segment
- Multiple smaller entries/tree
- Stores data in finer container
- Frequent I/O calls
- Reduces memory consumption of the program

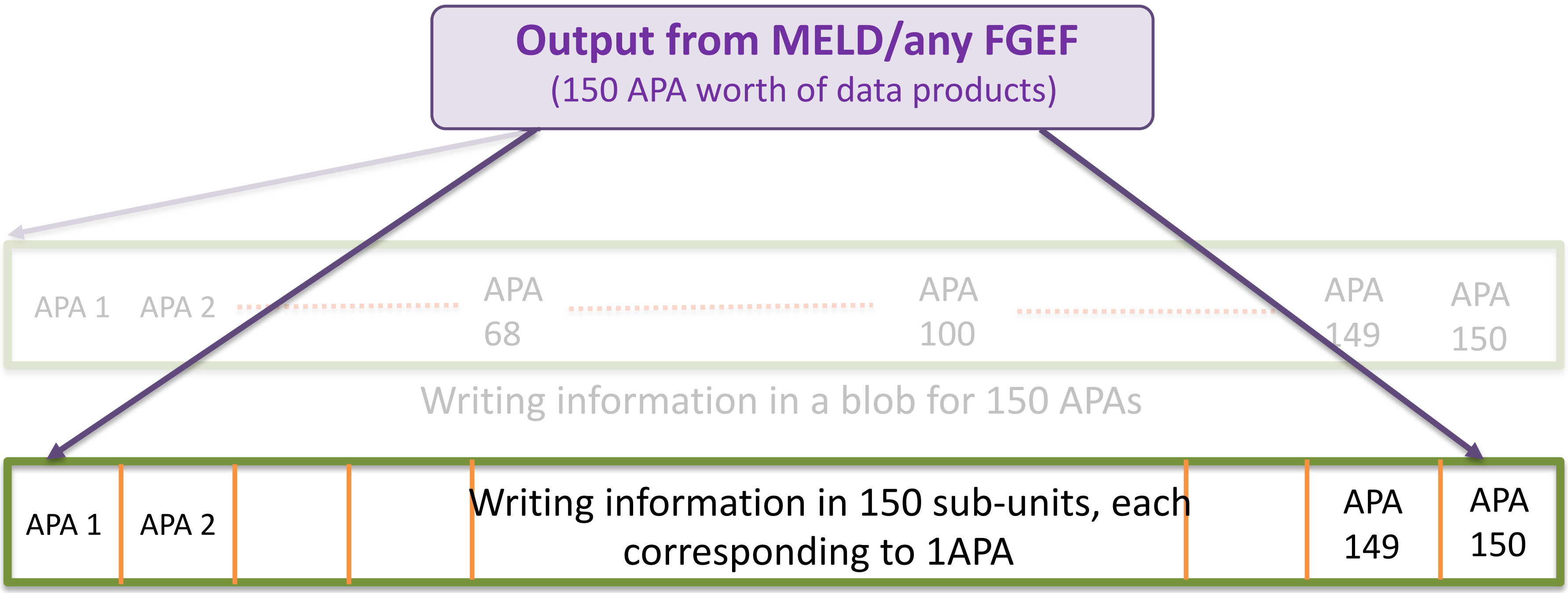
Event Storage Container for Art



Writing information in a large container for 150 APAs

- Traditionally DUNE writes large data products in a TTree with 1 entry (information from all 150 APAs) /event (TR)
- Makes it difficult to handle data
- Memory consumption for writing and reading is large

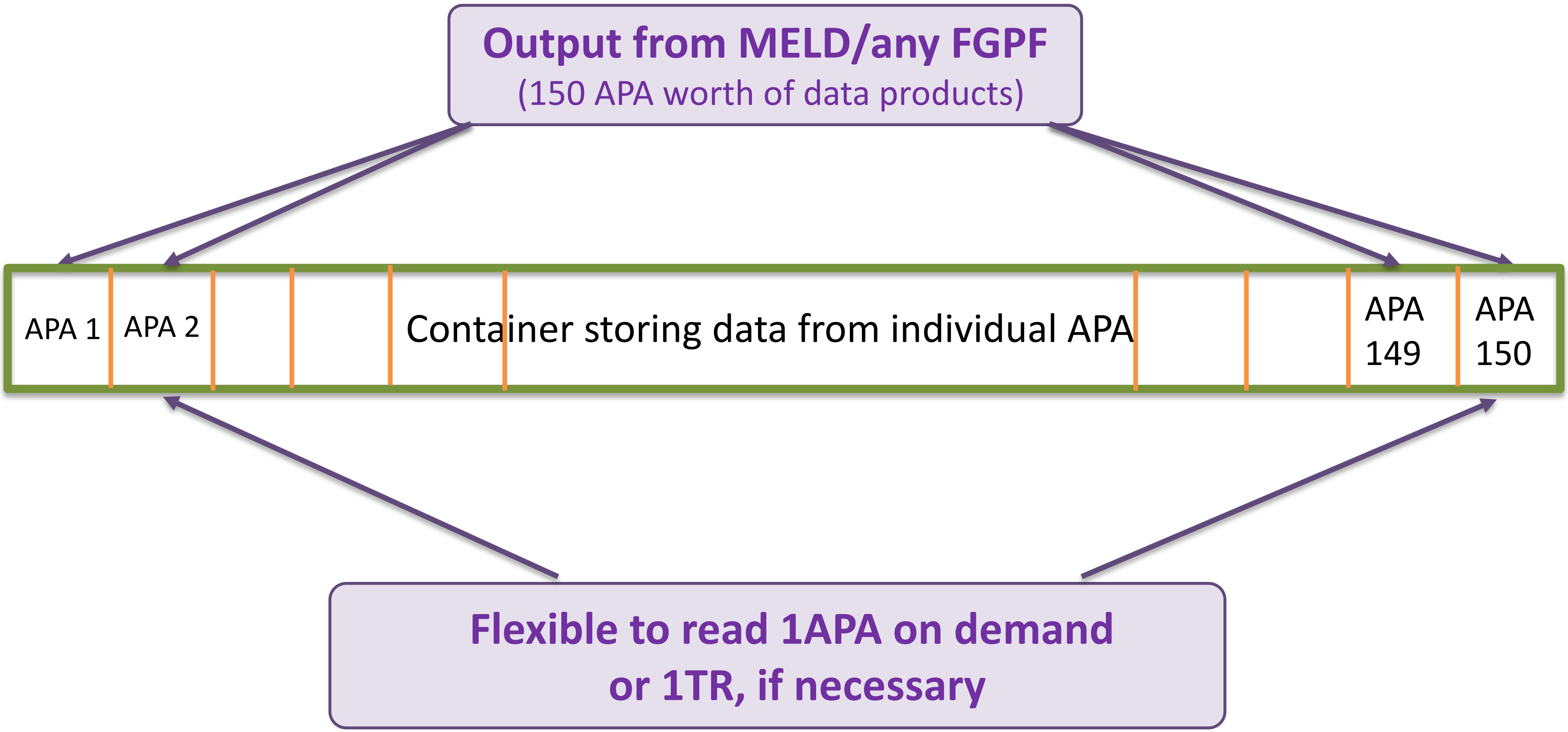
Fine-grained I/O Storage Container (Writing)



Flexible to write in multiple smaller entries or 1 chunk of data on demand

- The I/O infrastructure can write multiple entries (preferably from an individual segment or 1 APA) / event (TR)
- Writing data as 1APA/entry in a tree leads to 10 times less memory consumption compared to art

Fine-grained I/O & Storage Container (Reading)



- Read and access an individual APA
- Or store metadata (location information of APA1, APA2, etc.) to retrieve information about APAs
- Manageable memory allocation to read data

Fine Grained I/O “Toy” Framework Design (Standalone Code in ROOT)

```
void generate(std::vector<float>& vrand)
{
```

```
/** I/O Framework code toy */
template <typename T> void fill_branch(TFile* file, std::string treename, std::string name, std\
::vector<T>& v, Int_t nseg = -1) {
  static std::map<std::string, TTree*> open_trees; // In real this would be member of a class, \
not static
  if (open_trees.find(treename) == open_trees.end()) {
    TTree* tree;
    file->GetObject(treename.c_str(), tree);
    if (tree == nullptr) { return; }
    open_trees.insert({treename, tree});
  }
  TTree *tree = open_trees.at(treename);

  static std::map<std::string, std::vector<std::string>* > references; // In real this would be\
 member of a class, not static
  if (references.find(name) == references.end()) {
    std::vector<std::string>* tokens = new std::vector<std::string>();
    references.insert({name, tokens});
  }
  std::vector<std::string>* tokens = references.at(name);
```

Fine Grained I/O "Toy" Framework Design (Standalone Code in ROOT)

```cpp
template <typename T> void read_branch(TFile* file, std::string treename, std::string name, std\
::vector<T>** v, Int_t nevent, Int_t nseg = -1) {
  static std::map<std::string, TTree*> open_trees; // In real this would be member of a class, \
not static
  if (open_trees.find(treename) == open_trees.end()) {
    TTree* tree;
    file->GetObject(treename.c_str(), tree);
    if (tree == nullptr) { return; }
    open_trees.insert({treename, tree});
  }
  TTree *tree = open_trees.at(treename);

  TBranch *branch = 0;
  if (tree->GetBranch(name.c_str()) != nullptr && nseg < 0) { // read content
    if (nseg < 0) {
      tree->SetBranchAddress(name.c_str(), v, &branch);
      Long64_t tentry = tree->LoadTree(nevent);
      branch->GetEntry(tentry);
    } else { // fine grained read on large data
      *v = new std::vector<T>();
    }
  } else { // try navigation
    name += "_tok";
    std::vector<std::string>* tokens = 0;
    tree->SetBranchAddress(name.c_str(), &tokens, &branch);
    Long64_t tentry = tree->LoadTree(nevent);
    branch->GetEntry(tentry);
    *v = new std::vector<T>();
    for (UInt_t j = 0; j < tokens->size(); ++j) {
      std::string source = tokens->at(j);
      std::string token_treename, token_branchname;
      unsigned int token_offset, token_nseg;
```

Fine Grained I/O "Toy" Framework with ROOT (Writing)

Traditional TTree with writing 1 entry/event (TR)

Art Style

```
// Create the default/large TTree
TTree *tree_def = new TTree("tvec","Tree with large vectors");

if (DO_FULL_WRITE) {
  // event writing, current framework, will also write references
  fill_branch(file, "tvec", "rand", vrand);
  memc();
```

The 2nd TTree with writing 150 entries/event (TR)

Fine-Grained I/O Style

```
// Create the segmented TTree
TTree *tree_seg = new TTree("tvec_seg","Tree with segmented vectors");

 if (DO_SEGM_WRITE) {
   // sub-event writing /** called by Framework extension e.g. MELD */
   fill_branch(file, "tvec_seg", "rand", vrand_seg, nseg);
   memc();
 }
```

Memory Consumption after writing in segments

```
-bash-4.2$ ./toy_frameworkIO_P.exe
Start Write!
 MEMORY CONSUMPTION AFTER INITIALIZING TREES
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384  0.0  1.4 558744 172636 pts/0   S+   15:41   0:00 ./toy_frameworkIO_P.exe
 Write Event No. 0
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384  0.0  1.4 558744 172636 pts/0   S+   15:41   0:00 ./toy_frameworkIO_P.exe
Write Event segments 0
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384  100  1.7 604716 213940 pts/0   S+   15:41   0:01 ./toy_frameworkIO_P.exe
Write Event segments 1
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384  133  1.7 597112 211044 pts/0   S+   15:41   0:01 ./toy_frameworkIO_P.exe
Write Event segments 2
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384 81.0  1.7 597112 211044 pts/0   S+   15:41   0:01 ./toy_frameworkIO_P.exe
Write Event segments 3
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384 85.5  1.7 597112 211044 pts/0   S+   15:41   0:01 ./toy_frameworkIO_P.exe
Write Event segments 4
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384  100  1.6 586664 200596 pts/0   S+   15:41   0:02 ./toy_frameworkIO_P.exe
Write Event segments 5
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384  105  1.6 586664 200596 pts/0   S+   15:41   0:02 ./toy_frameworkIO_P.exe
Write Event segments 6
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384 85.0  1.8 610448 218512 pts/0   S+   15:41   0:02 ./toy_frameworkIO_P.exe
Write Event segments 7
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  23384  102  1.8 609824 218908 pts/0   S+   15:41   0:03 ./toy_frameworkIO_P.exe
```

Resident memory for writing each entry (or an individual APA) in tree2 is ~213MB

```
Event = 0: check = 1.67772e+07
USER       PID %CPU %MEM     VSZ   RSS TTY      STAT START   TIME COMMAND
barnali  24233 99.6  7.9 1536360 968124 pts/0   S+   15:44   0:07 ./toy_frameworkIO_P.exe
```

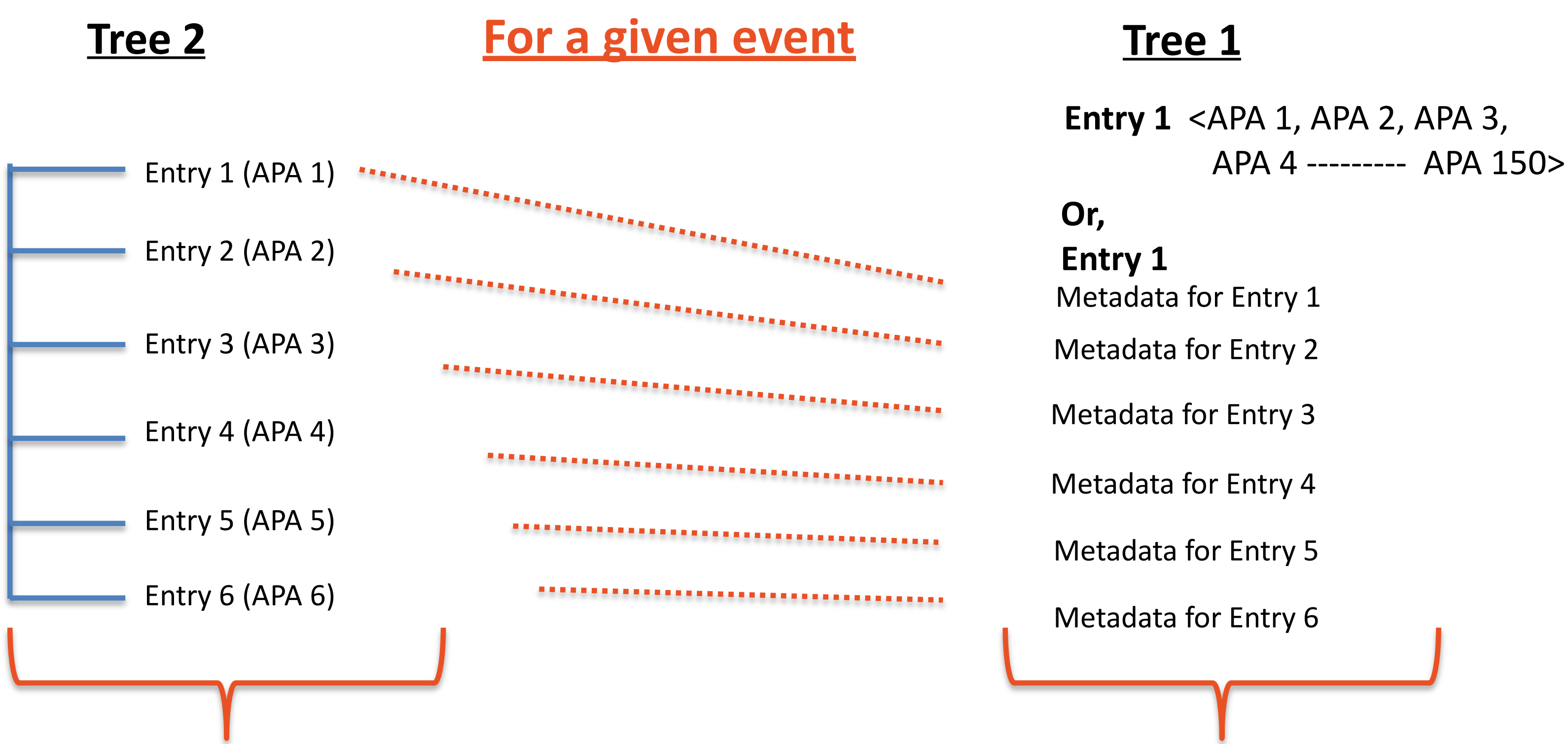
Resident memory for writing a single entry (accumulated APAs) in tree1 is ~968 MB

Argonne NATIONAL LABORATORY DUNE

Fine Grained I/O "Toy" Framework with ROOT (Reading)

- Traditionally DUNE jobs read data using art handle
  - Reads an event/TR of data at a time and keeps in memory

- Currently, by writing data in smaller chunks we can read data in segments



You could read data in 150 smaller data segments but DON'T

Allows "Fine"/ "On Demand" reading

Memory Consumption after reading in segments

```
-bash-4.2$ ./toy_frameworkIO_P.exe
Start Write!
Done Write!
Start Read!
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937  0.0  1.4 560812 175960 pts/6   S+   07:25   0:00 ./toy_frameworkIO_P.exe
Read event 0
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937  0.0  1.4 560812 175960 pts/6   S+   07:25   0:00 ./toy_frameworkIO_P.exe
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937  0.0  1.4 560812 175960 pts/6   S+   07:25   0:00 ./toy_frameworkIO_P.exe
Read event segments 0
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937 78.0  2.0 657844 248588 pts/6   S+   07:25   0:00 ./toy_frameworkIO_P.exe
Read event segments 1
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937 86.0  2.0 657844 248832 pts/6   S+   07:25   0:00 ./toy_frameworkIO_P.exe
Read event segments 2
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937 93.0  2.0 657844 248848 pts/6   S+   07:25   0:00 ./toy_frameworkIO_P.exe
Read event segments 3
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937 95.0  2.0 657844 248848 pts/6   S+   07:25   0:00 ./toy_frameworkIO_P.exe
Read event segments 4
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937  101  2.0 657844 248848 pts/6   S+   07:25   0:01 ./toy_frameworkIO_P.exe
Read event segments 5
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937  104  2.0 657844 248848 pts/6   S+   07:25   0:01 ./toy_frameworkIO_P.exe
Read event segments 6
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937  112  2.0 657844 248848 pts/6   S+   07:25   0:01 ./toy_frameworkIO_P.exe
Read event segments 7
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    7937 61.0  2.0 657844 248848 pts/6   S+   07:25   0:01 ./toy_frameworkIO_P.exe
```

Resident memory for reading each entry in tree2 is ~248MB

```
READ FULL EVENT
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
barnali    4503 99.7 20.5 2900512 2491780 pts/10 S+  11:10   0:06 ./toy_frameworkIO_P.exe
```

Resident memory for reading a single entry in tree1 is ~2.4 GB

Summary and Outlook

- **Summary**
  - Fine-Grained event processing framework requires “Fine-Grained I/O” infrastructure for handling memory efficiently

- **Outlook**
  - What kind of file format and data format I/O infrastructure is going to support
  - I/O infrastructure and Supernova
  - I/O infrastructure must be able to run in HPC
  - How are we going to plug in our I/O framework with event processing framework?

Thank You !

Milestones

- A framework is required that can *automatically*:
  - Decompose data into user-requested data groupings
  - Adapt its processing according to the requested data grouping
  - Regroup the data according to further processing needs

- The following milestones assume that our efforts will meet computing challenges of DUNE

2027	Framework solution(s) in place
2028	I/O format finalized
2029	(Event-)data model finalized
2031	Physics analysis begins at large scale