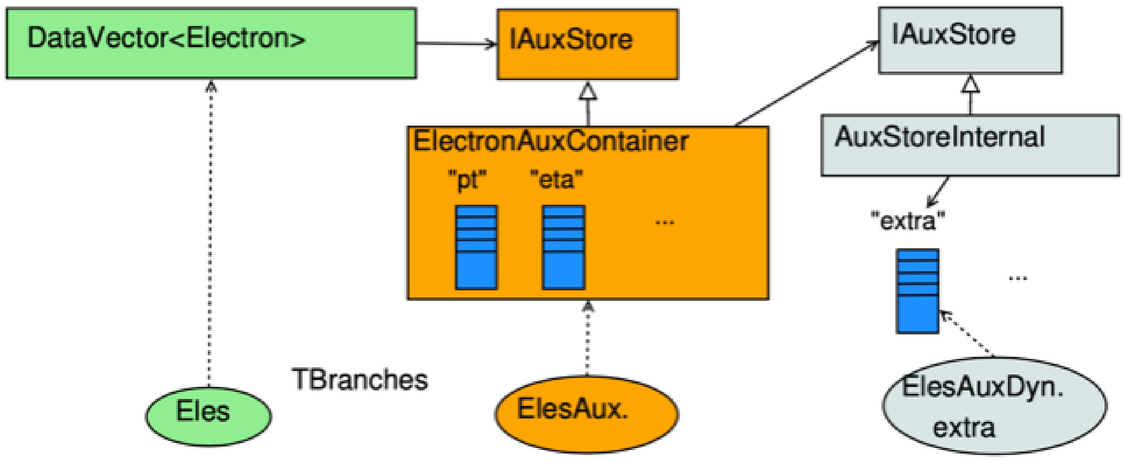
Survey:

ATLAS

Based on Talk given by Scott Snyder (Jan 10, 2023)

Structure of the current DATA:

- Data model is xAOD.
- Use of object pointers to access the variables related to the data object.
- Object pointers are stored in std::vector-like<T*> containers and the variables are stored in contiguous arrays.



Challenges or Issues to make the Data GPU Friendly

- Structure of the data
  - Most of the data represented as contiguous arrays
    - Can be copied to the GPUs explicitly
- Memory allocation
  - cannot use **std::vector**
  - **std::polymorphic_allocator**
    - To allocate memory for the dynamic variables, store a memory resource using the "**std::pmg::allocator**". This resource can be shared with other dynamic variables etc.
    - Seems more useful for multi-threading issues though (Not sure where it fits in the GPU friendly data model.)
- Nested Vectors (and 2D arrays)
  - Use of flat arrays (not yet implemented)
  - Aux variables are already represented as contiguous arrays of fixed sized objects.
    - Could use array (begin,end) to access variables from the contiguous array. (I don't understand the example in slide 12 use case in the context of the GPUs.).

Changes to move to the GPUs:

Contiguous arrays
Contiguous arrays can be copied over to the GPUs in their original form.

Memory allocation:
Currently managed by std::vector → Use std::polymorphic_allocator.
→ std::polymorphic_allocator can pass memory allocations to Container constructors that store dynamic variables.
→ Need some edit in TStreamerInfo to address ATLAS-specific forward compatibility issues.

Nested Vector:
Not implemented but possible to use flat arrays such that aux variables are represented as contiguous arrays of fixed size objects. Keep track of size in the flattened 1D space with begin and end indexes.

Accessing variables from the Container:
In the CPU done with a pointer to the owning container and the index of the item within the container.

Possible modifications to access variables in the GPU:

1. Avoid pointers at all and retrieve variables directly from the GPU. Challenges:
    a. Not supported by the xAOD class definitions.
    b. Could still have vectorization issues

Direct access of the variables:

Avoid pointers at all and retrieve variables directly from the GPUs.

Challenges

- Not supported by the current xAOD class.
- Could still have vectorization issues depending upon
- Compilers may not support vectorization
    - Probably issue on the host side only?

xAODElement as temporary container

xAODElement pointer to container → Temporary container of object.

Challenges

- Issues related to **const type** elements.
- ATLAS Forward compatibility related issues.

Current	GPU Friendly

class Foo : public xAOD::AuxElement {
  float& pt() {
 	
   static const Accessor<float> acc ("pt");
   return acc(*this);
  }
 	
const xAOD::FooContainer& foocont = xxx;
for (const Foo* foo : foocont) {
  doSomething (foo->eta(), foo->pt());
	Direct Access of the Variable
	
	static const Accessor<float> pt ("pt");
	static const Accessor<float> eta ("eta");
	for (size_t i = 0; i < foocont.size(); i++) {
	  doSomething (eta (foocont, i), pt (foocont, i));

	Temporary Container
	
	template <class T>
	class EltVector
	{
	  EltVector (DataVector<T>& v) : m_v (v) {}
	  T operator[] (size_t n) { return T (&m_v, n);
	}
	  DataVector<T>& m_v;
	};
	
	float ptsum (const DataVector<Foo>& v) {
	  size_t sz = v.size();
	  EltVector<Foo> ev (v);
	  float sum = 0;
	  for (size_t i = 0; i < sz; i++) {
	    sum += ev[i].pt();
	}
	return sum; }

Table I: Possible transformation of xAODElement to make xAOD data object GPU friendly.

Experiment: ATLAS
Target System: HPC
Language: CUDA