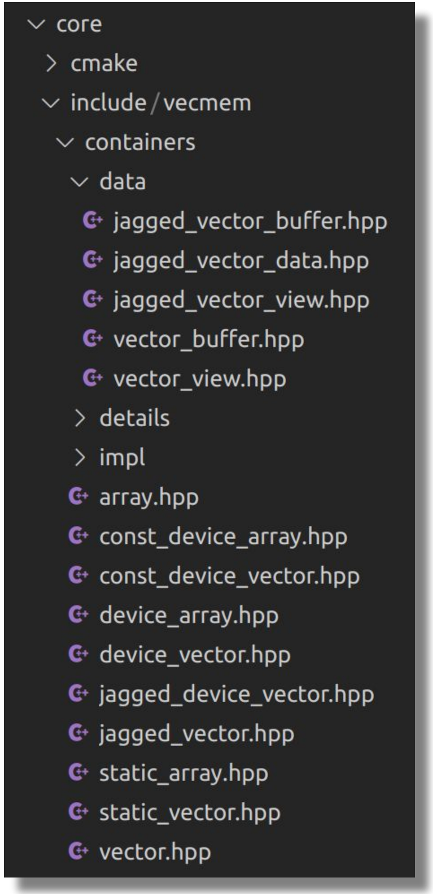
Acts/traccc Event Data Model

Some thoughts…

Attila Krasznahorkay

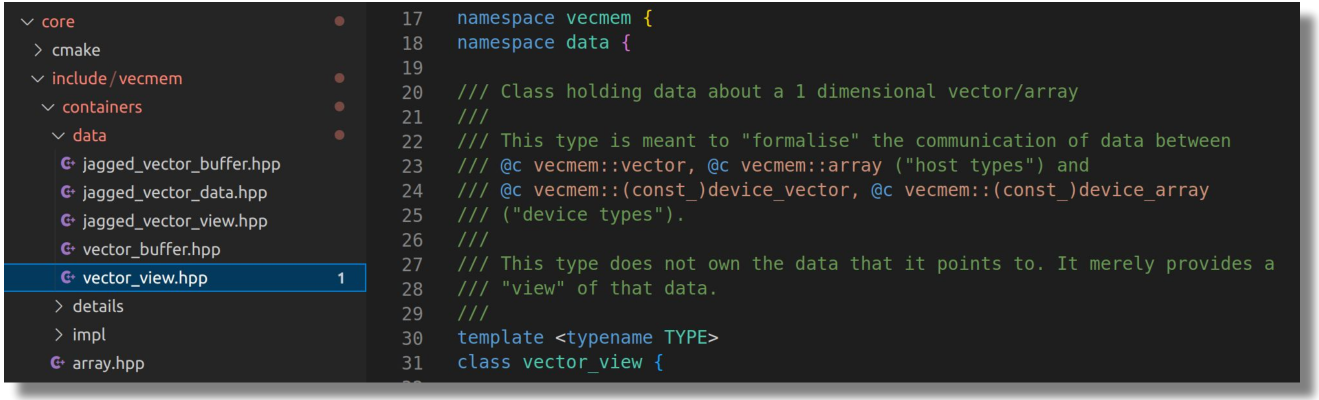
VecMem

- The R&D project practically only uses the `vecmem::(device_)(jagged_)vector` types from VecMem
  - We “invented” some other data types (static vector/array) as well, but they did not find a use yet
- VecMem provides a convenient way to manage these 1D/2D vectors in host and device code
  - Including STL-like access to them in device code, and efficient ways of copying them from/to a device



```
∨ core
  > cmake
  ∨ include / vecmem
    ∨ containers
      ∨ data
          jagged_vector_buffer.hpp
          jagged_vector_data.hpp
          jagged_vector_view.hpp
          vector_buffer.hpp
          vector_view.hpp
      > details
      > impl
        array.hpp
        const_device_array.hpp
        const_device_vector.hpp
        device_array.hpp
        device_vector.hpp
        jagged_device_vector.hpp
        jagged_vector.hpp
        static_array.hpp
        static_vector.hpp
        vector.hpp
```

Views

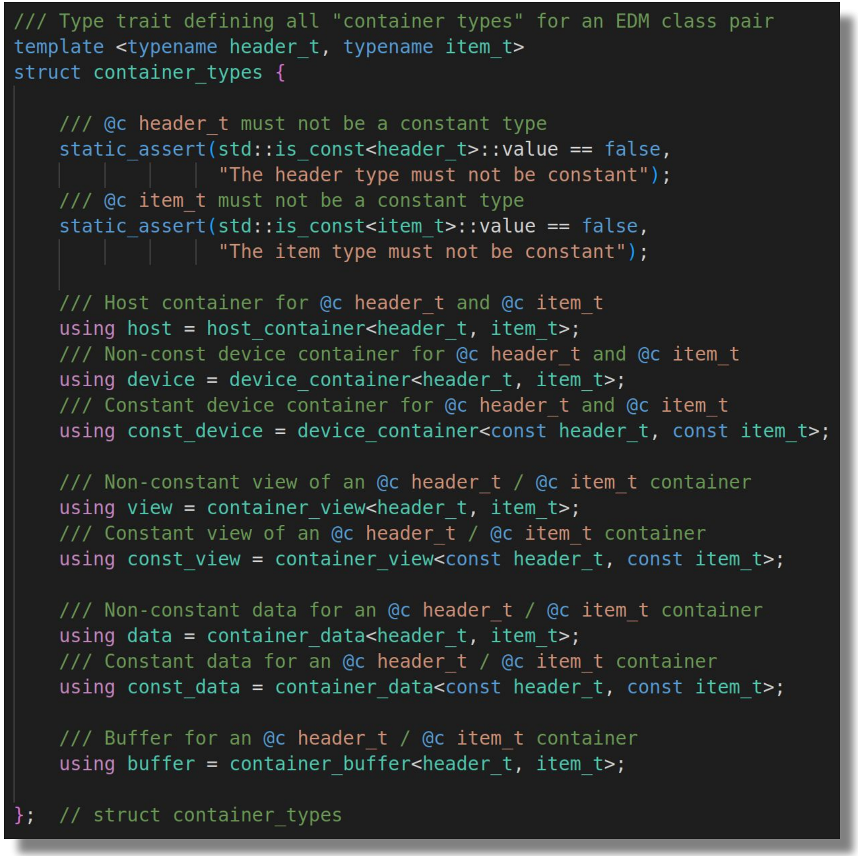


```
namespace vecmem {
namespace data {

/// Class holding data about a 1 dimensional vector/array
///
/// This type is meant to "formalise" the communication of data between
/// @c vecmem::vector, @c vecmem::array ("host types") and
/// @c vecmem::(const_)device_vector, @c vecmem::(const_)device_array
/// ("device types").
///
/// This type does not own the data that it points to. It merely provides a
/// "view" of that data.
///
template <typename TYPE>
class vector_view {
```

- We pass containers from host- to device code using “view types”
  - These are similar to std::span with “some amount of” resizability
- Algorithms in traccc (are meant to) receive input data through views
  - They don’t need to know how the 1D/2D vectors were created and managed in memory, they just need to know where they are in memory “right now”
  - Host code can work like this happily as well

Collections / Containers



```cpp
/// Type trait defining all "container types" for an EDM class pair
template <typename header_t, typename item_t>
struct container_types {

    /// @c header_t must not be a constant type
    static_assert(std::is_const<header_t>::value == false,
                  "The header type must not be constant");
    /// @c item_t must not be a constant type
    static_assert(std::is_const<item_t>::value == false,
                  "The item type must not be constant");

    /// Host container for @c header_t and @c item_t
    using host = host_container<header_t, item_t>;
    /// Non-const device container for @c header_t and @c item_t
    using device = device_container<header_t, item_t>;
    /// Constant device container for @c header_t and @c item_t
    using const_device = device_container<const header_t, const item_t>;

    /// Non-constant view of an @c header_t / @c item_t container
    using view = container_view<header_t, item_t>;
    /// Constant view of an @c header_t / @c item_t container
    using const_view = container_view<const header_t, const item_t>;

    /// Non-constant data for an @c header_t / @c item_t container
    using data = container_data<header_t, item_t>;
    /// Constant data for an @c header_t / @c item_t container
    using const_data = container_data<const header_t, const item_t>;

    /// Buffer for an @c header_t / @c item_t container
    using buffer = container_buffer<header_t, item_t>;

};  // struct container_types
```

- In traccc all data is stored in either simple 1D vectors (collections) or in a 1D+2D vector combination (container)
    - Much of our data can be described using N “elements” that each have M_N “items”
- Not clear to me yet how we would map this into Acts data structures eventually 🤔

AoS / SoA

- At the moment traccc uses an AoS data model
  - All our structs are small. They map (reasonably) well onto GPU memory load operations.
  - Larger structs become less efficient like this
- As Paul showed, Acts (and ATLAS offline) use a SoA memory layout instead
  - It is necessary for vectorisation on CPUs, and generally provides a more flexible EDM

```
struct spacepoint {
    float x, y, z;
};

using spacepoint_collection =
    vecmem::vector<spacepoint>;
```

```
struct spacepoint_collection {
    vecmem::vector<float> x, y, z;
};
```

(My) Questions / Discussion Points

- How to declare the types in Acts?
  - Even if EDM classes have a templated user-facing API, they **must** have a non-templated base (device code offload must not be exposed to the user)
  - How to integrate the VecMem based memory management with the storage backend developed for the track states (and with ATLAS offline's own memory management)?
- We can definitely require clients to use std::pmr::memory_resource to interact with Acts
  - I'm less sure about publicly exposing VecMem container types in the Acts API… 🤔

http://home.cern