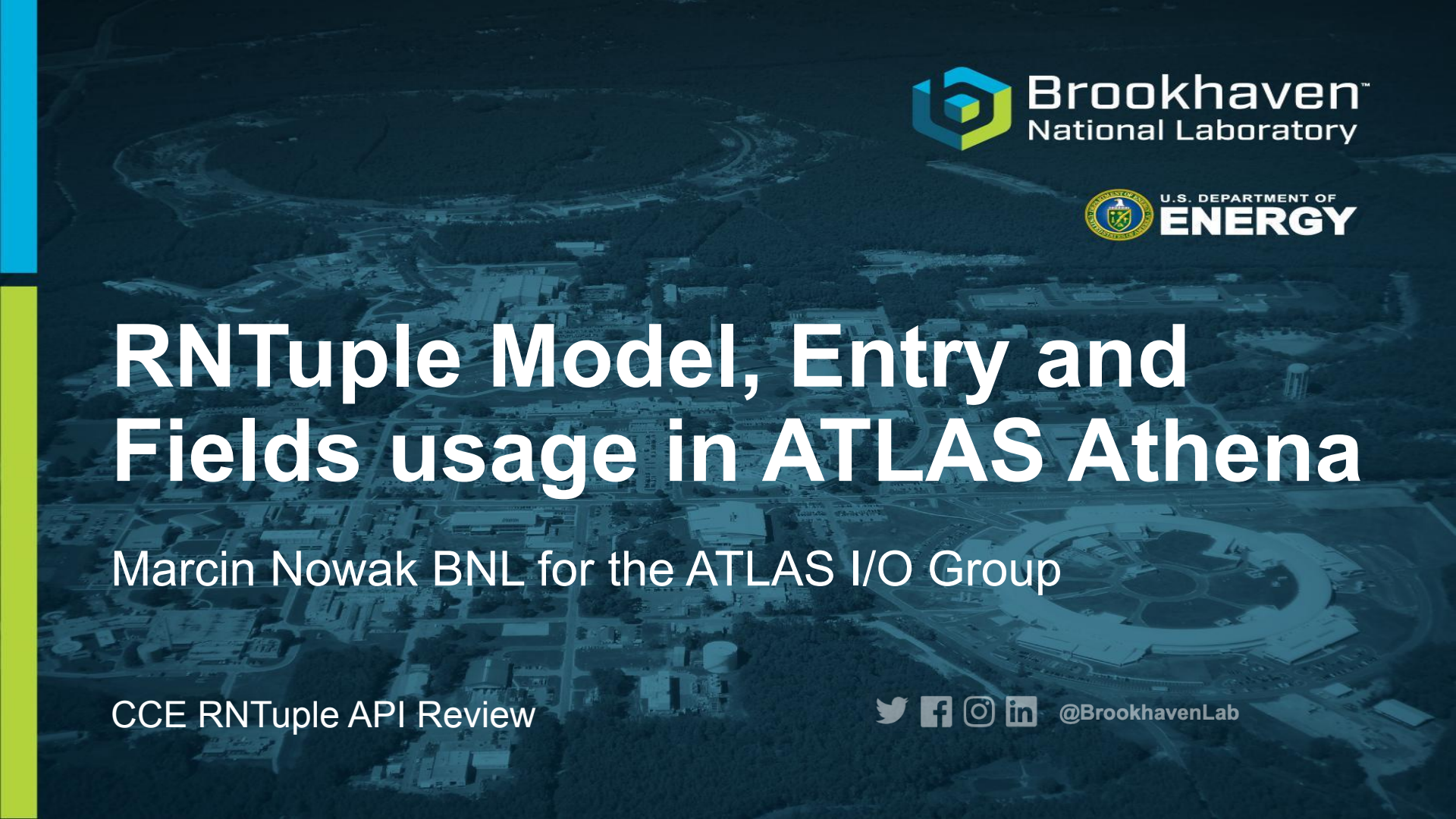

RNTuple Model, Entry and Fields usage in ATLAS Athena

Marcin Nowak BNL for the ATLAS I/O Group

CCE RNTuple API Review

@BrookhavenLab

Writing

- Athena writing characteristics:
  - Writing “Event at a time” - Event being a collection of independent objects
  - The first Event defines the schema and every subsequent Event must have the same set of objects
    - (technical requirement coming from the choice of storage tech: TTree/RNTuple)
    - New dynamic xAOD attributes may appear at any time
  - Objects belong to the framework and are created fresh for every Event
  - Objects are passed to ROOT as a whole
    - Except the dynamic xAOD attribute - which are treated as objects on their own

Write API

Start:

```
auto model =RNTupleModel::Create()
```

Preparation:

```
Foreach Object:
   auto field = RFieldBase::Create(fieldName, fieldTypeName).Unwrap();
   if( model )  model->AddField( std::move(field) );
   else addDynamicAttribute( field )
```

Event writing:

```
/* if first write, it consumes the model */
if ( model )  createRNTuple(model)
auto entry = ntupleWriter->GetModel().CreateBareEntry();
Foreach Object:
   if( data_ptr )  entry->BindRawPtr(fieldName, data_ptr);
   else               entry->EmplaceNewValue(fieldName);
ntupleWriter->Fill( *entry );
```

Model update:

```
auto updater =
ntupleWriter->CreateModelUpdater();
updater->BeginUpdate();
updater->AddField( std::move(field) );
updater->CommitUpdate();
```

Reading

- Athena reads objects on demand
  - One entire object at a time, from the Event object collection (one row)
- 2 modes of reading:
  - ‘into’ an existing object
  - with an allocation of a new object
- Regardless of how the object was read, Athena will delete it eventually
- Some objects require ‘on-read’ actions
- User-defined collections need special treatments

Read API

Preparation:

```
Foreach Field:
   auto view = ntupleReader->GetView<void>( fieldName, nullptr )
   view_uptr = std::make_unique<RNTupleView<void,true>>( view );
```

Single object read:

```
if( !object_ptr )
   object_ptr = view_uptr->GetField().CreateObject<void>().release();
view_uptr->BindRawPtr( object_ptr );
// read into the object
(*view_uptr)(row);
view_uptr->BindRawPtr( nullptr );
```

Bottom Line

- RNTuple read/write API has all the features Athena has been using up to now with TTrees
  - (because we asked for it…)
  - A bit weird to need a pointer to keep a View. Should we discuss it?
  - Up to now, (in terms of API) we mainly focused on getting the functionality, as opposed to performance
- To keep in mind
  - There are things that are equally (if not more) important for us, that do not fall into “API” category”:
    STL collections support, user-defined collections support, read callbacks