# EDM4hep <-> LCIO conversion

Today: New tales from {Particle,Object}IDs

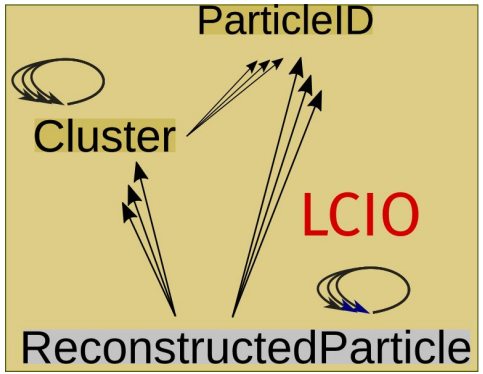
May 22, 2025

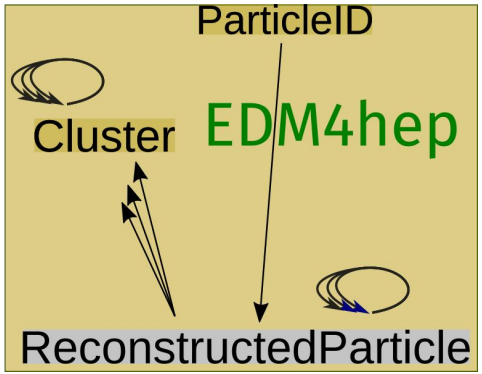
Thomas Madlener

# LCIO vs EDM4hep (re: ParticleID)

- Direction of relation different
  - Multithreading & mutability
  - “Don’t change what isn’t yours”
- ParticleID stored as part of ReconstructedParticle (or Cluster) in LCIO
  - Separate collections in EDM4hep

```
edm4hep::ParticleID:
  Description:  "ParticleID"
  Author: "EDM4hep authors"
  Members:
    - int32_t   type           // userdefined type
    - int32_t   PDG            // PDG code of this id - ( 999999 ) if unknown
    - int32_t   algorithmType  // type of the algorithm/module that created this hypothesis
    - float likelihood     // likelihood of this hypothesis - in a user defined normalization
  VectorMembers:
    - float parameters     // parameters associated with this hypothesis
  OneToOneRelations:
    - edm4hep::ReconstructedParticle particle // the particle from which this PID has been computed
```



# Interlude 1: Basics of EDM conversion

- Two step process
  - Convert all data, record mapping between converted EDM4hep and LCIO objects (potentially bi-directional)
  - Re-establish relations between objects
- Bonus for MarlinWrapper
  - Keep a “global” (lookup) mapping of EDM4hep <-> LCIO objects to keep event contents consistent

```
/**
 * Convert an MCParticle collection and return the resulting collection.
 * Simultaneously populates the mapping from LCIO to EDM4hep objects.
 */
template <typename MCParticleMapT>
std::unique_ptr<edm4hep::MCParticleCollection>
convertMCParticles(const std::string& name, EVENT::LCCollection* LCCollection, MCParticleMapT& mcparticlesMap);
```

```
/**
 * Resolve the relations for the MCParticles.
 */
template <typename MCParticleMapT, typename MCParticleLookupMapT>
void resolveRelationsMCParticles(MCParticleMapT& mcparticlesMap, const MCParticleLookupMapT& lookupMap);
```

```
/**
 * The LCIO <-> EDM4hep object mapping that holds the relations between all
 * converted objects from all converters that are running.
 */
struct GlobalConvertedObjectsMap {
  template <typename K, typename V>
  using ObjectMapT = k4EDM4hep2LcioConv::VecMapT<K, V>;

  ObjectMapT<EVENT::Track*, edm4hep::Track> tracks{};
  ObjectMapT<EVENT::TrackerHit*, edm4hep::TrackerHit3D> trackerHits{};
  ObjectMapT<EVENT::SimTrackerHit*, edm4hep::SimTrackerHit> simTrackerHits{};
  ObjectMapT<EVENT::CalorimeterHit*, edm4hep::CalorimeterHit> caloHits{};
  ObjectMapT<EVENT::RawCalorimeterHit*, edm4hep::RawCalorimeterHit> rawCaloHits{};
  ObjectMapT<EVENT::SimCalorimeterHit*, edm4hep::SimCalorimeterHit> simCaloHits{};
  ObjectMapT<EVENT::TPCHit*, edm4hep::RawTimeSeries> tpcHits{};
  ObjectMapT<EVENT::Cluster*, edm4hep::Cluster> clusters{};
  ObjectMapT<EVENT::Vertex*, edm4hep::Vertex> vertices{};
  ObjectMapT<EVENT::ReconstructedParticle*, edm4hep::ReconstructedParticle> recoParticles{};
  ObjectMapT<EVENT::MCParticle*, edm4hep::MCParticle> mcParticles{};
  ObjectMapT<EVENT::TrackerHitPlane*, edm4hep::TrackerHitPlane> trackerHitPlanes{};
  ObjectMapT<EVENT::ParticleID*, edm4hep::ParticleID> particleIDs{};
```

# ParticleID conversion basics

- EDM4hep -> LCIO
  - Convert ReconstructedParticles
  - Convert ParticleIDs (store in mapping only)
  - Resolve relations
- LCIO -> EDM4hep
  - Convert ReconstructedParticles
  - Create ParticleID collections in parallel (one collection per PID algorithm)
  - (all relations immediately resolved during data conversion)

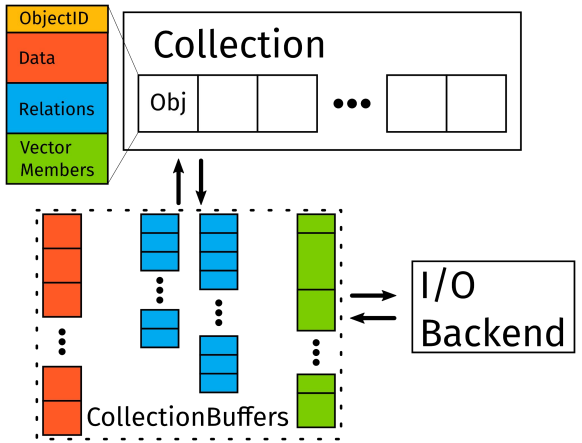
```
template <typename PidMapT, typename RecoParticleMapT>
void resolveRelationsParticleIDs(PidMapT& pidMap, const RecoParticleMapT& recoMap) {
  for (auto& [lcioPid, edmPid] : pidMap) {
    const auto edmReco = edmPid.getParticle();
    const auto lcioReco = k4EDM4hep2LcioConv::detail::mapLookupFrom(edmReco, recoMap);
    if (lcioReco) {
      lcioReco.value()->addParticleID(lcioPid);
    } else {
      std::cerr << "Cannot find a reconstructed particle to attach a ParticleID to" << std::endl;
    }
  }
}
```

```
// Set up a PIDHandler to split off the ParticlID objects stored in the
// reconstructed particles into separate collections. Each algorithm id /
// name get's a separate collection
auto pidHandler = UTIL::PIDHandler(LCCollection);
// TODO: parameter names
std::map<int, std::unique_ptr<edm4hep::ParticleIDCollection>> particleIDs;
for (const auto id : pidHandler.getAlgorithmIDs()) {
  particleIDs.emplace(id, std::make_unique<edm4hep::ParticleIDCollection>());
}
```

# Interlude 2: ObjectIDs

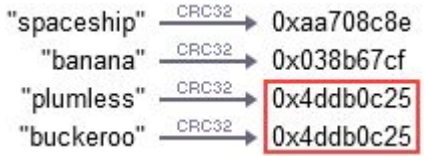
- Uniquely identifies an object **for I/O** purposes
- CollectionID computed as a 32 bit hash of collection name
  - **Assigned when collection is added to a Frame**
- Not really considered to have any guarantees (in podio)
- But technically a convenient way to get from an object back to the collection it belongs to

```
class ObjectID {

public:
  /// not part of a collection
  static const int untracked = -1;
  /// invalid or non-available object
  static const int invalid = -2;

  /// index of object in collection
  int index{untracked};
  /// ID of the collection
  uint32_t collectionID{static_cast<uint32_t>(untracked)};
```



# Interlude 2.1: hash collision probabilities vs hash size

- Hash collision: two distinct values are mapped to the same value




- More likely than you think
  - Somewhat risky for podio(!)
  - 32 bits puts us somewhere between 1:10000 and 1:100000

| Number of **32-bit** hash values | Number of **64-bit** hash values | Number of **160-bit** hash values | **Odds of a hash collision** |
|---|---|---|---|
| 77163 | 5.06 billion | $1.42 \times 10^{24}$ | 1 in 2 |
| 30084 | 1.97 billion | $5.55 \times 10^{23}$ | 1 in 10 |
| 9292 | 609 million | $1.71 \times 10^{23}$ | 1 in 100 |
| 2932 | 192 million | $5.41 \times 10^{22}$ | 1 in 1000 |
| 927 | 60.7 million | $1.71 \times 10^{22}$ | 1 in 10000 |
| 294 | 19.2 million | $5.41 \times 10^{21}$ | 1 in 100000 |
| 93 | 6.07 million | $1.71 \times 10^{21}$ | 1 in a million |
| 30 | 1.92 million | $5.41 \times 10^{20}$ | 1 in 10 million |
| 10 | 607401 | $1.71 \times 10^{20}$ | 1 in 100 million |
| | 192077 | $5.41 \times 10^{19}$ | 1 in a billion |
| | 60740 | $1.71 \times 10^{19}$ | 1 in 10 billion |
| | 19208 | $5.41 \times 10^{18}$ | 1 in 100 billion |
| | 6074 | $1.71 \times 10^{18}$ | 1 in a trillion |
| | 1921 | $5.41 \times 10^{17}$ | 1 in 10 trillion |
| | 608 | $1.71 \times 10^{17}$ | 1 in 100 trillion |
| | 193 | $5.41 \times 10^{16}$ | 1 in $10^{15}$ |
| | 61 | $1.71 \times 10^{16}$ | 1 in $10^{16}$ |
| | 20 | $5.41 \times 10^{15}$ | 1 in $10^{17}$ |
| | 7 | $1.71 \times 10^{15}$ | 1 in $10^{18}$ |

Odds of a full house in poker
1 in 693

Odds of four-of-a-kind in poker
1 in 4164

Odds of being struck by lightning
1 in 576000

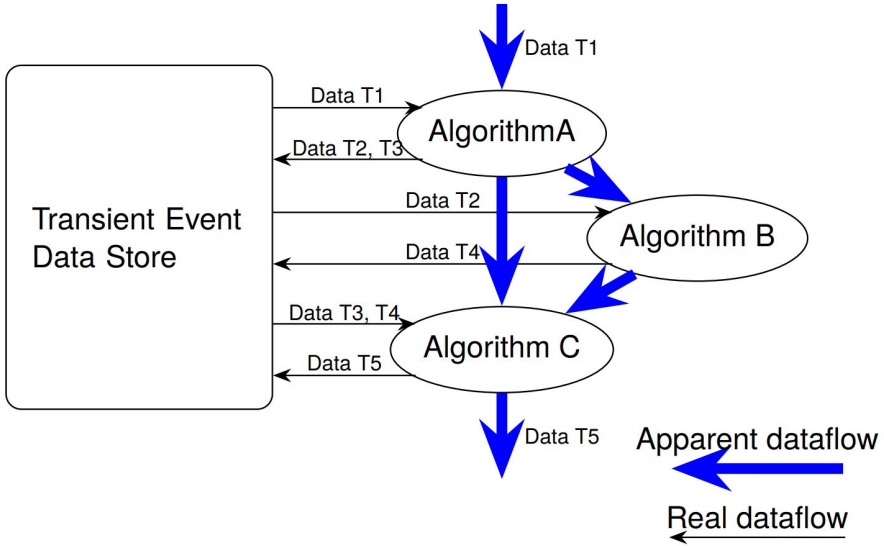
Odds of winning a 6/49 lottery
1 in 13.9 million

Odds of dying in a shark attack
1 in 300 million

Odds of a meteor landing on your house
1 in 182 trillion

# Interlude 3: The transient event store in Gaudi

- Used for passing data between individual algorithms
- A-priori independent of podio & EDM4hep



# Interlude 3 (ctd): Data services in k4FWCore

- k4FWCore implements I/O functionality for EDM4hep on top of Gaudi
  - Legacy *PodioDataSvc*
  - New *IOSvc*
  - Both offer functionality for necessary type casting, etc.
- PodioDataSvc uses a Frame internally
  - Custom wrapper around EventDataSvc
  - Effectively bypasses Gaudi TES for EDM4hep collections and puts / gets collections to / from its internal Frame
  - **Collections get a collectionID when they are handed to the TES** (aka the Frame)
- IOSvc only deals with I/O
  - Uses standard Gaudi EventDataSvc
  - Reads Frame and puts collections into the TES
  - Creates / Reuses Frame for writing
  - **Collections get a collectionID when they are written**

# ParticleID conversion of metadata

- ParticleIDs can have PID metadata attached
  - Algorithm name
  - Parameter names
- Stored as metadata (parameters) of collection
  - Attached to ReconstructedParticle collection in LCIO
  - Stored as collection parameters for ParticleID collection in EDM4hep
- Valid for all collections
  - Still attached to each collection in LCIO
- Challenge: Need to figure out the collection name of the **ReconstructedParticle** collection in LCIO

# ParticleID metadata attaching (EDM4hep -> LCIO)

- What we have
  - Fully converted ParticleID and ReconstructedParticle collections
  - (Maybe) PID meta information
  - Mapping of collection IDs to collection names
- What we need
  - Name of the ReconstructedParticle collection (LCIO) to which we should attach the PID meta information
- Problem: Collections that have not yet seen a Frame have no collection ID
  - Will get a “random” name

```
const auto id = (*pidCollMeta.coll)[0].getParticle().id().collectionID;
if (auto it = m_idToName.find(id); it != m_idToName.end()) {
  auto name = it->second;
```

Current implementation

This will be **0xffffffff** (i.e. (uint32_t) -1) for collections that were created during processing

# A possible solution

- Need to assign collection IDs when collection is handed to the TES
- **Need to avoid giving any (accidental) guarantees about collection IDs**
  - They should remain an “implementation detail”
- Introduce another layer of abstraction
  - Make it possible to get to a collection from an object
  - **iff: collection is known to the TES (or the Frame)**
  - Can use the collection ID internally

podio#782

```
template <CollectionType CollT>
const CollT& Frame::get(const typename CollT::value_type& object) const {
  const auto name = m_self->getIDTable().name(object.id().collectionID);
  return get<CollT>(name.value_or(""));
}
```

Already handles “unknown” names

Only gives a valid name if known

```
template <podio::ObjectType O>
const typename O::collection_type* getCollectionFor(const O& object) const {
  return dynamic_cast<const typename O::collection_type*>(getCollectionFor(object.id()));
}

// TODO: return string_view? Some form of DataHandle?
template <podio::ObjectType O>
const std::optional<std::string> getCollectionNameFor(const O& object) const {
  return getCollectionNameFor(object.id());
}
```

k4FWCore#312

```
StatusCode initialize() final {
  m_collFromObjSvc = service("CollectionFromObjSvc", false);
  if (!m_collFromObjSvc) {
    return StatusCode::FAILURE;
  }
  return StatusCode::SUCCESS;
}

void operator()(const edm4hep::MCParticleCollection& inputColl) const final {
  const auto mc = inputColl[0];
  debug() << "Retrieving collection for object with id " << mc.id() << endmsg;
  const auto* collFromObj = m_collFromObjSvc->getCollectionFor(mc);
  const auto checkMC = (*collFromObj)[0];
  if (mc != checkMC) {
    throw std::runtime_error("Could not get the expected collection from the object");
  }
```

# Summary

- ParticleID relations have different directions between EDM4hep and LCIO
- Requires a “reverse” lookup (from object to collection)
- ObjectID is an implementation detail for I/O purposes in podio / EDM4hep
  - Need to avoid overloading it with “guarantees” (cf. Hyrums Law)
- There is no problem that another layer of abstraction doesn’t solve
- Proposed functionality in podio to get collection from object
- Proposed new *CollectionFromObjectSvc* to k4FWCore to do the same
- Both still use collectionID internally
  - At this point that is an optimization detail and it could also be done differently transparently for the user