Key4hep Status and Next Steps

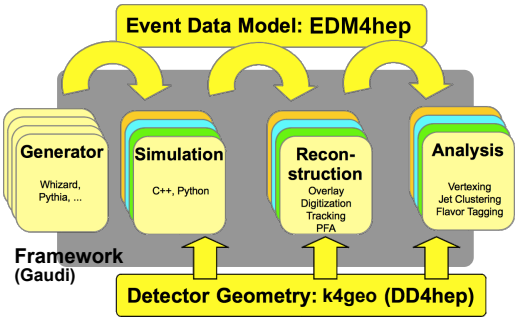
Juan Miguel Carceller j.m.carcell@cern.ch

CERN, EP-SFT

January 15, 2025

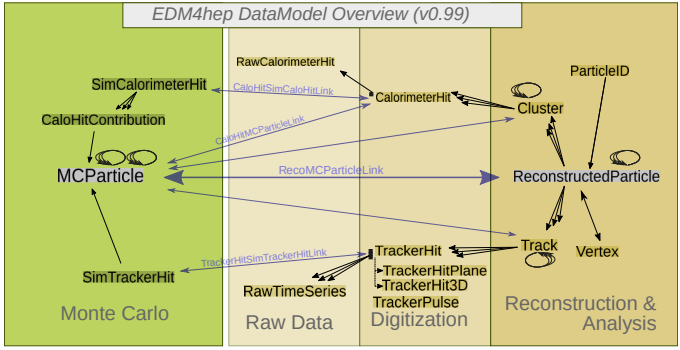
Key4hep

- **Turnkey software for future colliders**
- **Share components** to reduce maintenance and development cost and allow everyone to benefit from its improvements
- **Complete data processing framework** from generation to data analysis
- Community with people from **many future experiments**: FCC, ILC, CLIC, CEPC, EIC, Muon Collider, etc.
- Open biweekly meetings



The Event Data Model: EDM4hep

- **Common language** that all components speak in Key4hep speak
- Classes for physics objects, like `MCParticle`, with **relations** to other objects
- **Links** between objects
- Objects are grouped in **collections**, like `MCParticleCollection`
- Open biweekly meetings

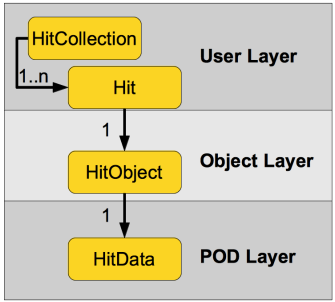


Podio and EDM4hep

- Podio (Plain Old Data IO) is the tool used to generate code for EDM4hep
- Podio favors **composition over inheritance** with its layered design
- The specification for EDM4hep is done in a yaml file, podio converts it to C++ code

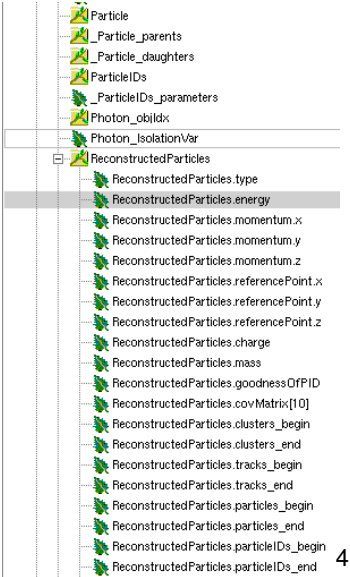
```yaml
edm4hep::MCParticle:
  Description: "Monte Carlo particle..."
  Author: "EDM4hep authors"
  Members:
  - int32_t PDG
  - float charge
  - float time [ns]
  - double mass [GeV]
...
  OneToManyRelations:
  - edm4hep::MCParticle parents
  - edm4hep::MCParticle daughters
```

Jinja templates



Podio and EDM4hep

- Podio provides a ROOT TTree and RNTuple backend
  - Standalone `podio-dump` to show the contents
  - Almost flat TTrees and RNTuples
- Files can be used without having EDM4hep
  - Relations have to be solved manually, convoluted and error-prone
  - **New `podio::DataSource`** to help with RDataFrame analyses
- **New generic `Reader` and `Writer`**
  - `Reader` will detect the format of the input file



Reading

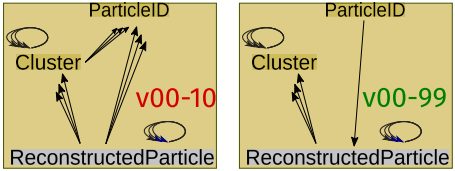
```cpp
auto reader = podio::makeReader("example.root");
auto frame = reader.readNextFrame(podio::Category::Event);
auto coll = frame.get("MCParticles");
```

Writing

```cpp
// Assume we have a frame called frame
auto writer = podio::makeWriter("example.root");
frameWriter.writeFrame(frame, podio::Category::Event);
```

EDM4hep: Recent changes

- Issue: After creating a reconstructed particle, it is not possible to modify it
  - Additional PID algorithms would have to copy it and create a new one
  - Doesn't work well with multithreading
- **Solution: Rework relations** in several places in EDM4hep
- Utility to simplify navigation was added



- Another issue: multiple tracker hit types, must use one
  - **Interface types** have been added
  - `TrackerHit` is the interface for our tracker hits

```
auto track = edm4hep::Track{};
track.addHit(edm4hep::TrackerHit3D{});
track.addHit(edm4hep::TrackerHitPlane{});
```

EDM4hep: And more

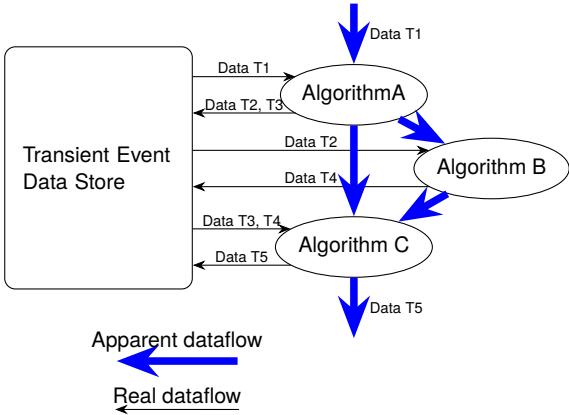
- More recently:
  - Templated links
  - Members changed or removed, `colorFlow` from MCParticles, several members in Tracks, etc.
  - Change the run number from 32 bits to 64 bits

See v0.99 Release notes

- Future (?)
  - Hit for drift chambers and dual readout
  - Tensor type for ML
- **What's the current status?**
  - Version 0.99 released with several major and breaking changes from version 0.10
  - Some changes will go in before 1.0
  - Version 1.0 not defined exactly, release date unknown but soon

The Key4hep Framework

- Gaudi based core framework:
  - k4FWCore provides the interface between EDM4hep and Gaudi
  - k4Gen for integration with generators
  - k4SimDelphes for integration with Delphes
  - k4MarlinWrapper to call Marlin processors
  - Algorithms for trackers (k4RecTracker), calorimeters (k4RecCalorimeter)
  - Algorithms ported from the linear collider community
  - …



Framework Core: Current status

- `k4FWCore` provides the **interface between EDM4hep and Gaudi**
- To create an algorithm a C++ class has to be created. Two choices are possible
  - Use `Gaudi::Algorithm` as a base class (legacy)
  - Use **functional algorithms**, they do not have internal state and are suited for **multithreading** (preferred)

Algorithm (C++)

```
struct ExampleFunctionalProducer final :
  k4FWCore::Producer<edm4hep::MCParticleCollection()> {

  edm4hep::MCParticleCollection operator()() const override {
    auto coll = edm4hep::MCParticleCollection();
    return coll;
  }
};
```

Steering file (Python)

```
producer = ExampleFunctionalProducer(
                  "Producer",
                  OutputCollection=["MCParticles"])
ApplicationMgr(
    TopAlg=[producer],
    EvtSel="NONE",
    EvtMax=10,
    ExtSvc=[EventDataSvc("EventDataSvc")],
    OutputLevel=INFO,
)
```

- Fork k4-project-template to start from a working example!

Functional algorithms in Key4hep

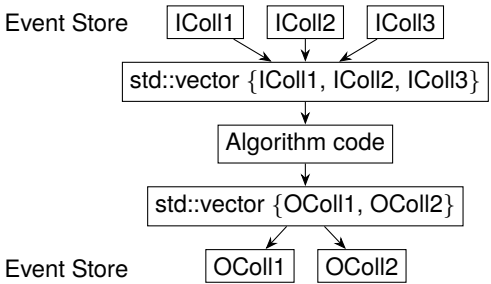
- New service, `IOSvc`, **supports multithreading**, reading and writing ROOT TTrees and ROOT RNTuples
  - Reading detects automatically if it's a TTree or RNTuple
- Two input/output algorithms: `Reader` and `Writer`
  - `Reader` will ask `IOSvc` to read and then will push the collections to the store
  - `Writer` will write the collections to a file
- Easily change to multithreading by using Gaudi's `HiveWhiteBoard`
- Metadata utilities compatible with `IOSvc`

```
svc = IOSvc("IOSvc")
svc.Input = "input.root"
svc.Output = "output.root"
svc.OutputType = "RNTuple"
```

Functional algorithms in Key4hep: Features

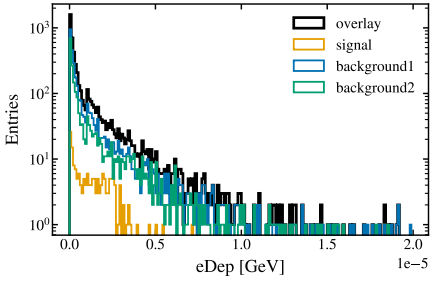
- Support for having as **input or output an arbitrary number of collections** with `std::vector`
- Algorithms should work for all detectors (with different number of subdetectors)

- Algorithms can now:
  - Pick up multiple collections and store them in a `std::vector` when reading
  - Iterate over the collections and push them individually when pushing a `std::vector`



Example Algorithm: Overlay

- Ported from `Overlay Timing` from iLCSoft
- Reads collections from background files and overlays them on top of the signal
  - `edm4hep::MCParticle`: all particles with a time offset for background
  - `edm4hep::SimTrackerHit`: only hits within a configurable time-window
  - `edm4hep::SimCalorimeterHit`: only if they have any `edm4hep::CaloHitContribution` within a specific time window




```
overlay = OverlayTiming()
overlay.MCParticles = ["MCParticles"]
overlay.BackgroundMCParticleCollectionName = "MCParticle"
overlay.SimTrackerHits = ["VertexBarrelCollection", "VertexEndcapCollection"]
overlay.OutputSimTrackerHits = ["NewVertexBarrelCollection", "NewVertexEndcapCollection"]
...
```

LCIO Converter

- LCIO is the **EDM** in the **linear collider community**
- Marlin processors (equivalent to Gaudi algorithms) take LCIO input and have LCIO output
  - Can be used with EDM4hep in Gaudi using the `MarlinProcessorWrapper`
  - Duplicate events in memory and overhead of the conversion
- Standalone converter `lcio2edm4hep` to convert files

LCIO Converter

- LCIO is the **EDM** in the **linear collider community**
- Marlin processors (equivalent to Gaudi algorithms) take LCIO input and have LCIO output
  - Can be used with EDM4hep in Gaudi using the `MarlinProcessorWrapper`
  - Duplicate events in memory and overhead of the conversion
- St

  - Issue: the wrapper is not compatible with `IOSvc`
    - PRs in `k4EDM4hep2LcioConv` and `k4MarlinWrapper` to fix this
    - CLIC reconstruction with EDM4hep input and output with `IOSvc` works
    - Same output as when not using `IOSvc`

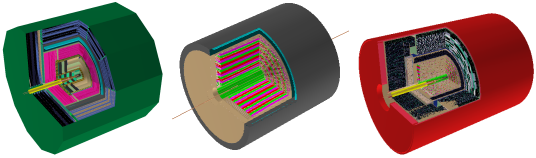
EDM4hep2LCIO converter

LCIO2EDM4hep converter

DD4hep

- DD4hep provides a way to build a detector description by using XML compact files
- **Interfaces Geant4** and can run physics simulations with `ddsim`
- Plugin-based system
- Detector models and geometries are stored in k4geo
- `GeoSvc` to use DD4hep in algorithms
- PR to be able to **read EDM4hep files as input**

```
ddsim --compactFile /path/to/file.xml
      -G --numberOfEvents 3
      --outputFile sim.edm4hep.root
      --gun.energy "10*GeV"
      --gun.particle "mu-"
      --gun.multiplicity 1
      --gun.distribution uniform
      --gun.etaMin -3
      --gun.etaMax 3
```



FCC-ee Detector Benchmarks:
CLD, IDEA and ALLEGRO

The Key4hep Stack: Summary

- Builds are **deployed to CVMFS**:
  - `/cvmfs/sw.hsf.org` for **releases** (every N months, on demand)
  - `/cvmfs/sw-nightlies.hsf.org` for **nightly** builds, every day
- Around **600 packages** including dependencies
- Compilers are taken either from the system or installed on CVMFS
- **Several flavours of builds**: now AlmaLinux 9 and Ubuntu 22.04 (previously also CentOS 7) with an optimized and debug version of each build
- key4hep-spack where many of the recipes for packages in Key4hep live but also where issues are discussed

The Key4hep Stack: User side

- We provide a script that checks the OS and sources the appropiate environment script
- With `source /cvmfs/sw.hsf.org/key4hep/setup.sh` users access the stack

```
AlmaLinux/RockyLinux/RHEL 9 detected
Setting up the Key4hep software stack release latest-opt from CVMFS
Use the following command to reproduce the current environment:

        source /cvmfs/sw.hsf.org/key4hep/setup.sh -r 2024-10-03

If you have any issues, comments or requests, open an issue at https://github.com/key4hep/key4hep-spack/issues
Tip: A new -d flag can be used to access debug builds, otherwise the default is the optimized build
```

- For the nightlies: `source /cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh`
- The setup script has several options, passing `--help` or `-h` displays the usage
- Example: how to see which version of EDM4hep will be picked up?

```
$ source /cvmfs/sw.hsf.org/key4hep/setup.sh --list-packages | grep edm4hep
edm4hep 0.99.1
```

The Key4hep Stack: User side

- After sourcing, users are mostly on their own
- A function to setup packages locally was developed and is provided in the `setup.sh` script
- `k4_local_repo` will remove the paths from the stack for the current package and add the local ones

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
git clone https://github.com/key4hep/edm4hep
cd edm4hep
k4_local_repo
mkdir build; cd build
cmake ..
cmake --build .
```

The Key4hep Stack: User side

- After sourcing, users are mostly on their own
- A function to setup packages locally was developed and is provided in the `setup.sh` script
- `k4_local_repo` will remove the paths from the stack for the current package and add the local ones

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
git clone https://github.com/key4hep/edm4hep
cd edm4hep
k4_local_repo
mkdir build; cd build
cmake ..
cmake --build .
```

- Outlook: moving to `LCGCmake`?
  - Build system used by the LCG stacks, used by ATLAS and LHCb
  - Would support Ubuntu 24.04 and (maybe) MacOS
  - Key4hep stack already built on Linux with GCC and Clang
  - Few packages missing for MacOS

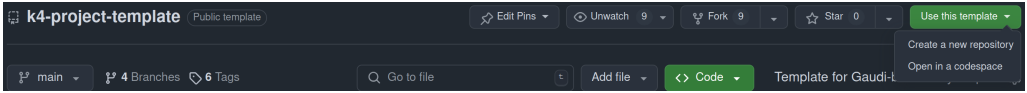
Summary

- Key4hep continues to be a **software framework for future colliders studies**
  - **Fully adopted by the FCC** and the linear collider community
  - Involvement of other experiments varies (CEPC, EIC, Muon Collider)
- 2024 was a year of improvements and new features
- Roadmap for the next months and year
- Many new features and improvements planned for 2025

Backup

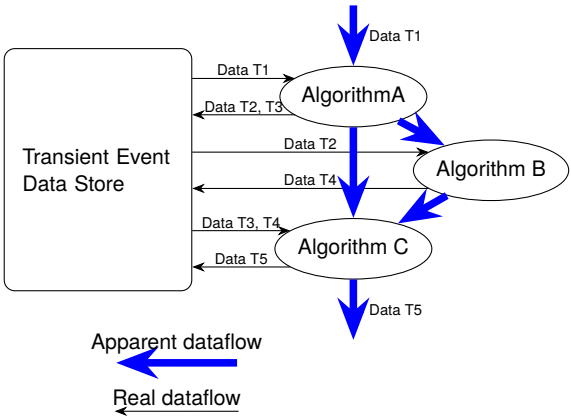
How to develop an algorithm

- This is all very cool but I don't know how to get started
- **Template repository** provided
- k4-project-template
- With a few examples of simple algorithms
- Ready to be compiled and be used as starting point



Gaudi

- Event processing framework
- Used by LHCb, ATLAS and others
- Algorithms are written in C++ and are configured with steering files in Python
- Data is passed between algorithms using a Transient Event Data Store
- Lots of services: histogramming, logging, etc.



key4hep-stack

- Depends on most of the Key4hep packages
- Pulls lots of package as dependencies
- Creates a `setup.sh` script that sets up the paths
- Users source this script (indirectly) to get the environment

```
depends_on("acts")
depends_on("babayaga")
depends_on("bdsim")
depends_on("bhlumi")
depends_on("cldconfig")
depends_on("dd4hep")
depends_on("delphes")
depends_on("edm4hep")
...
```

```
$ cat /cvmfs/sw.hsf.org/key4hep/releases/2024-10-03/x86_64-almalinux9-gcc14.2.0-opt/key4hep-stack/2024-10-08-k6xtr3/setup.sh
export ACLOCAL_PATH=/cvmfs/sw.hsf.org/key4hep/releases/2024-10-03...
export CC=/cvmfs/sw.hsf.org/contrib/x86_64-almalinux9-gcc11.4.1-o...
export CLDCONFIG=/cvmfs/sw.hsf.org/key4hep/releases/2024-10-03/x8...
export CMAKE_PREFIX_PATH=/cvmfs/sw.hsf.org/key4hep/releases/2024-...
...
```

Functional algorithms in Gaudi

- `Gaudi::Functional` algorithms
  - Multithreading friendly, no internal state
  - Leave details of the framework to the framework

```
class MySum : public TransformAlgorithm<OutputData(const Input1&, const Input2&)> {
     MySum(const std::string& name, ISvcLocator* pSvc)
      : TransformAlgorithm(name, pSvc, {
              KeyValue("Input1Loc", "Data1"),
              KeyValue("Input2Loc", "Data2") },
            KeyValue("OutputLoc", "Output/Data")) { }

    OutputData operator()(const Input1& in1, const Input2& in2) const override {
        return in1 + in2;
    }
```

- Adapted to work in Key4hep with EDM4hep

Functional algorithms

- Example: producer of an arbitrary number of collections

```
struct ExampleFunctionalProducerRuntimeCollections final
    : k4FWCore::Producer<std::vector<edm4hep::MCParticleCollection>()> {
  ExampleFunctionalProducerRuntimeCollections(const std::string& name, ISvcLocator* svcLoc)
      : Producer(name, svcLoc, {}, {KeyValues("OutputCollections", {"MCParticles"})}) {}

  std::vector<edm4hep::MCParticleCollection> operator()() const override {
    const auto locs = outputLocations();
    std::vector<edm4hep::MCParticleCollection> outputCollections;
    for (size_t i = 0; i < locs.size(); ++i) {
      info() << "Creating collection " << i << endmsg;
      auto coll = edm4hep::MCParticleCollection();
      coll.create(1, 2, 3, 4.f, 5.f, 6.f);
      coll.create(2, 3, 4, 5.f, 6.f, 7.f);
      outputCollections.emplace_back(std::move(coll));
    }
    return outputCollections;
  }
};
```

Functional algorithms in Key4hep: IOSvc

- Example of a steering file

```
from Gaudi.Configuration import INFO
from Configurables import ExampleFunctionalTransformer
from Configurables import EventDataSvc
from k4FWCore import ApplicationMgr, IOSvc

svc = IOSvc("IOSvc")
svc.Input = "input.root"
svc.Output = "output.root"

transformer = ExampleFunctionalTransformer(
    "Transformer", InputCollection=["MCParticles"], OutputCollection=["NewMCParticles"]
)

mgr = ApplicationMgr(
    TopAlg=[transformer],
    EvtSel="NONE",
    EvtMax=-1,
    ExtSvc=[EventDataSvc("EventDataSvc")],
    OutputLevel=INFO,
)
```

DD4hep in Key4hep

- For using DD4hep inside algorithms a `GeoSvc` is provided
- Creating an instance in the steering file will build the geometry and make it available in the algorithm

```
geoservice = GeoSvc("GeoSvc")
geoservice.detectors = [os.environ["K4GEO"]+"/FCCee/CLD/compact/CLD_o2_v07/CLD_o2_v07.xml"]
```

```
  m_geoSvc = serviceLocator()->service(m_geoSvcName);
...
  const auto detector = m_geoSvc->getDetector();

  const auto         surfMan = detector->extension<dd4hep::rec::SurfaceManager>();
  dd4hep::DetElement det     = detector->detector(m_subDetName.value());
  surfaceMap                 = surfMan->map(m_subDetName.value());
```