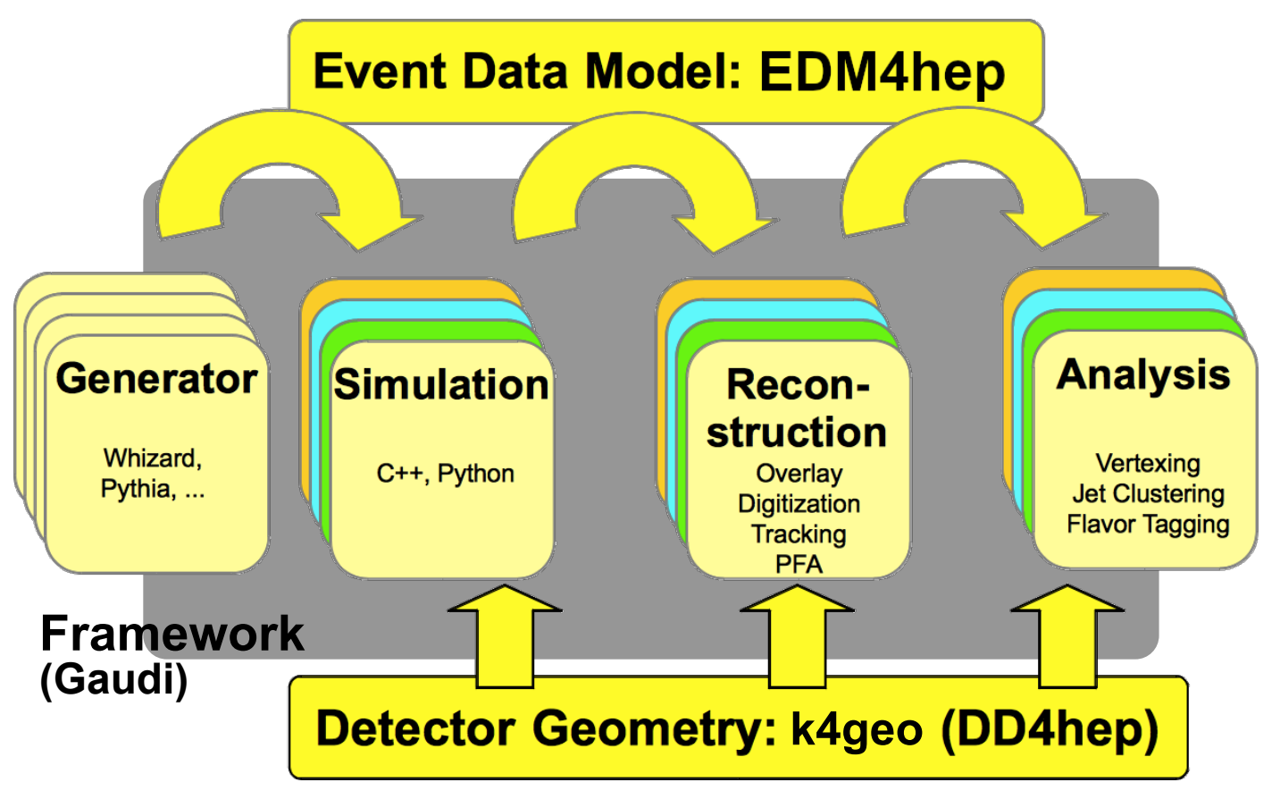
Key4hep: A Turnkey Software Framework for Future Collider Experiments With Practical Advice

Juan Miguel Carceller (CERN) on behalf of the Key4hep authors

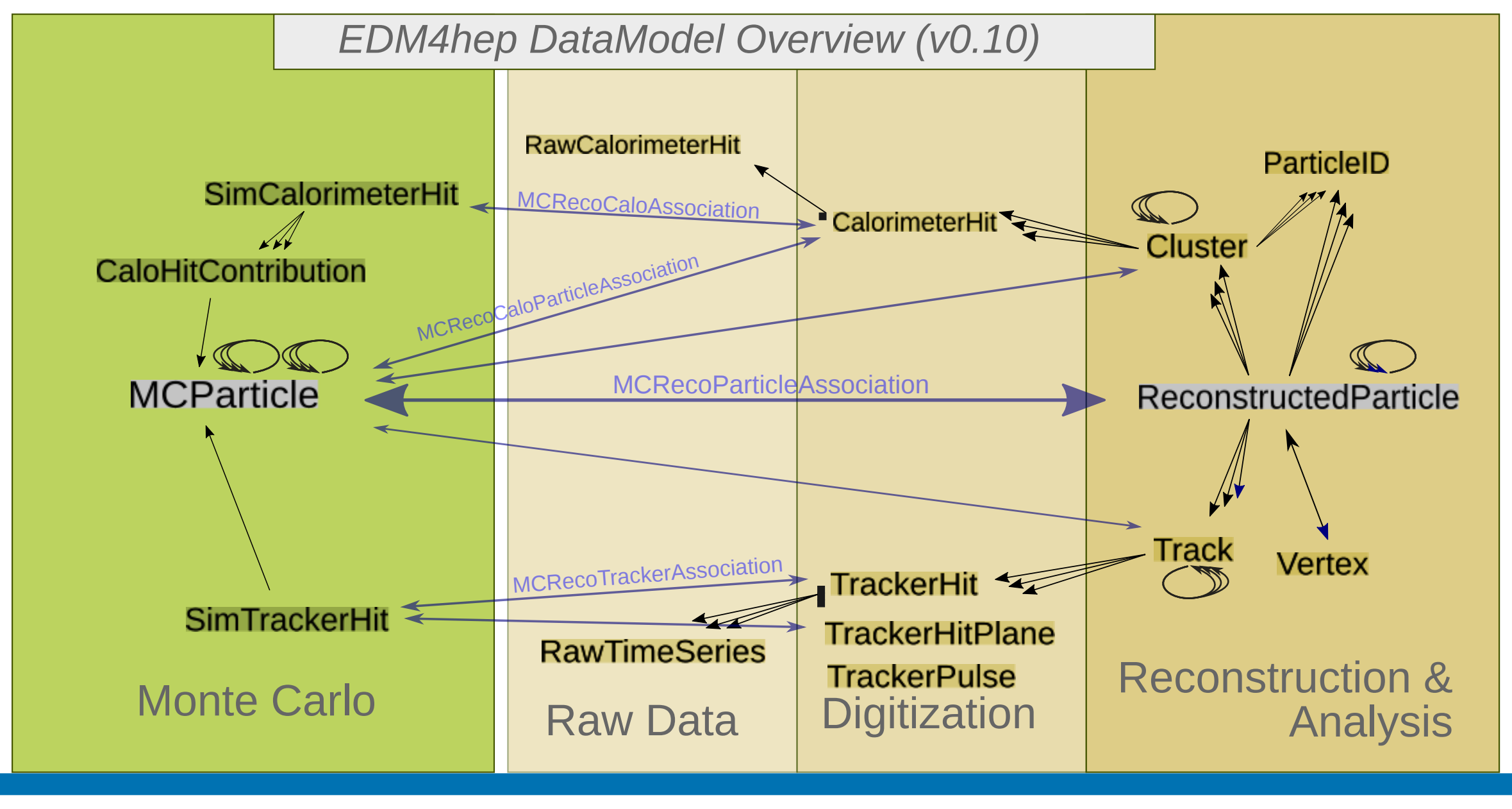
Introduction

- **Turnkey software framework**: Key4hep provides a complete data processing framework, from Monte Carlo generation to data analysis
- **Share components** across different experiments and communities and avoid duplication of effort
- **International community** with participants from CEPC, CLIC, EIC, FCC, ILC and the Muon Collider from CERN, DESY, IHEP, INFN and other institutes



EDM4hep

- EDM4hep is an **Event Data Model** and the **core component** of Key4hep
- **Common language** that all the components in Key4hep speak
- The goal is to be both **generic** and **address all the needs** of the experiments

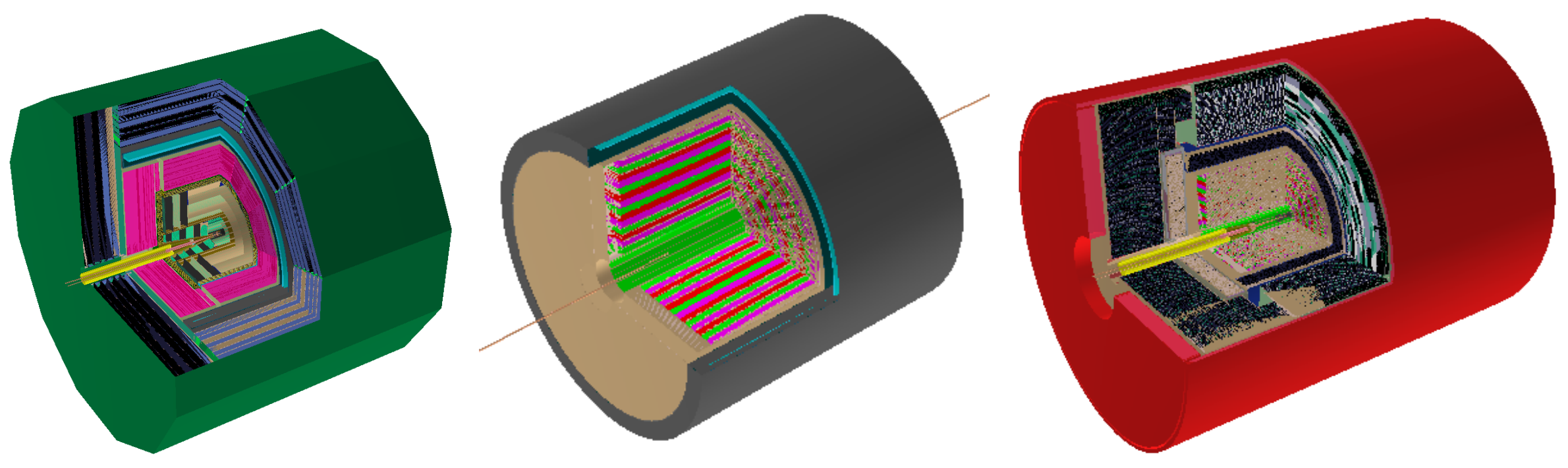


Gaudi in Key4hep

- Gaudi is an **event-processing framework**, used by ATLAS, LHCb and others
- Key4hep provides an interface to Gaudi, enabling the execution of **algorithms that read or write EDM4hep data**
- There are **more interfaces**: to Monte Carlo Generators, Geant4, Delphes and others
- **Ongoing work in other integrations or algorithms** like ACTS or Pandora
- Support for **multithreading** has been added recently

Detector studies with DD4hep

- Key4hep uses the **DD4hep** detector description framework **based on Geant4**
- The geometries of the detectors are stored in a **common repository** and deployed on cvmfs
- Users can easily test them and their different versions
- Steering files to run a full reconstruction chain are often provided
- **Validation pipeline** involving simulation and reconstruction to detect potential issues as the detector evolves



FCC-ee Detector Concepts: CLD, IDEA and ALLEGRO

The Key4hep stack

- **Complete software stack** of over 500 packages that are deployed on cvmfs
- Nightly build and stable releases
- Built with **spack**, a community-driven package manager
- **Supports multiple operating systems**: Alma 9, CentOS 7 and Ubuntu 22.04

How to develop a package

- How to make changes to a package and test it or run it when working with the Key4hep stack:

```
$ source /cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh
$ git clone https://github.com/user/package
$ cd package
$ k4_local_repo
$ # Make changes in the package
$ mkdir build; cd build
$ cmake ..
```

`k4_local_repo` will remove any paths in the current environment to package (if it exists in the stack) and add a set of predefined ones

Tip: Use `ccache` to speed up recompilations. It's included in the Key4hep stack so you only need to add to the cmake command: `-DCMAKE_CXX_LAUNCHER=ccache`

Working with EDM4hep: Python bindings

- Almost everything can be done from Python

Reading

```python
from podio.root_io import Reader
reader = Reader('myfile.root')
events = reader.get('events')
for frame in events:
    coll = frame.get('MCParticleCollection')
```

Writing

```python
import podio, edm4hep
writer = Writer('myfile.root')
coll = edm4hep.MCParticleCollection()
frame = podio.Frame()
frame.put(coll)
writer.write_frame(frame, 'events')
```

- Working in Python will be slower than in C++! It's good for exploration and prototyping but production should be done in C++ (or calling compiled code)

Tip: You can use Python interactively as documentation for EDM4hep classes. For example, how do I get the energy of a SimTrackerHit?

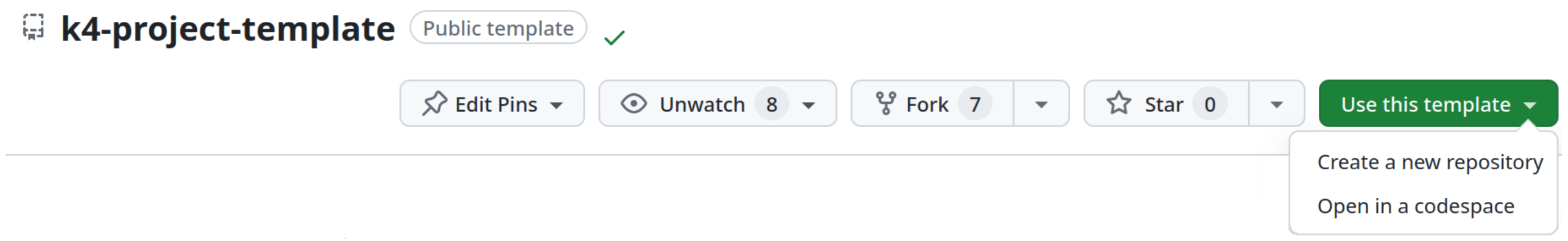
```python
import edm4hep
hit = edm4hep.SimTrackerHit()
hit.get<TAB>
hit.getCellID(         hit.getMCParticle(  hit.getObjectID(    hit.getPathLe
hit.getEDep(           hit.getMomentum(    hit.getParticle(    hit.getPositi
```

Press TAB to complete

- Works for every EDM4hep class

Starting a new Gaudi project

- Key4hep provides a template project to be used for projects that use Gaudi: https://github.com/key4hep/k4-project-template
- Click "Use this template" → "Create a new repository" and follow the instructions in the README



k4-project-template Public template ✓

Edit Pins ▾ | Unwatch 8 ▾ | Fork 7 ▾ | Star 0 ▾ | Use this template ▾

Create a new repository
Open in a codespace

Writing a Gaudi Algorithm

- Three types of Functional Gaudi Algorithms supported at the moment:
  - Consumer: Takes inputs, but doesn't have any outputs
  - Producer: Has outputs but doesn't take any inputs
  - Transformer: Has both inputs and outputs
- Few examples from the template
- Plenty of examples in the k4FWCore repository
- Example of an algorithm that takes as input MCParticles and does something with them

Name Algorithm type Output Input

```cpp
struct ExampleFunctionalConsumer final : k4FWCore::Consumer<void(const edm4hep::MCParticleCollection& input)> {
  ExampleFunctionalConsumer(const std::string& name, ISvcLocator* svcLoc)
      : Consumer(name, svcLoc, KeyValues("InputCollection", {"MCParticles"})) {}

  void operator()(const edm4hep::MCParticleCollection& input) const override {
    if (input.size() != 2) {
      fatal() << "Wrong size of MCParticle collection, expected 2 got " << input.size() << endmsg;
      throw std::runtime_error("Wrong size of MCParticle collection");
    }
  }
};
```

Input parameters and default values

For this example only the size of the input is checked

`operator()` has the same signature as the Consumer and this is the code that runs for every event

Tip: Algorithms based on `GaudiAlg` (they inherit from `GaudiAlg`) will not work in the future, inherit from `Gaudi::Algorithm` instead or (even better) use `Gaudi::Functional`