June 20th, 2023

IMCC Annual Meeting

Key4HEP migration

of the Muon Collider software

current status and future plans

N. Bartosik [a, b]

for the **Muon Collider Physics and Detector Group**

[a] INFN Torino *(Italy)* [b] CERN *(Switzerland)*

Introduction: need for change

Current software framework served us well to kick-start our full-simulation studies

↳ most components reused from the CLIC experiment + several developments on top

In the meantime a new software stack has emerged → that is now used by several experiments
(ILC, CLIC, FCC, CEPC)

Key4hep
turnkey software for future colliders

- more modern and future-proof tools
- larger pool of users and developers

By adopting Key4hep we can benefit from developments by other experiments
with less maintenance on our side required to keep our software up to date

↳ **existing HEP tools are evolving + new ones are appearing**

- **Particle Flow:** PandoraPFA → Pandora SDK → k4Pandora;
- **Clustering:** CLUE → k4Clue;
- **ROOT DataFrames:** support being added to the Key4hep data model;

Muon Colider software: current stack

The main components of our current software stack:

1. **LCIO** → **event-data model** [`LCIO::SimCalorimeterHit`, ... stored in *.slcio files]
2. **DD4hep** → **flexible geometry-description language** + interface with Geant4
3. **Marlin** → **framework for simulation components** + chaining them together via *.xml files
4. **ILCSoft** → **collection of scripts for putting all the software together** + all the dependencies

The two main methods for distributing our software:

1. **Local install** → **a set of instructions to install the software on a specific machine**
   with full control over each component's code → **best for development**
2. **Container** → **download and run on any machine via Docker/Singularity/Apptainer**
   with limited possibility to modify the code → **best for analysis**

Transition step: DD4hep

ILCSoft software stack:

1. LCIO
2. DD4hep
3. Marlin
4. ILCSoft

Key4hep software stack:

- EDM4hep
- DD4hep
- Gaudi
- Spack

DD4hep → DD4hep: DONE

We both use DD4hep for detector-geometry description

↳ no changes needed on our side

Transition step: Spack

ILCSoft software stack:

1. LCIO
2. DD4hep
3. Marlin
4. ILCSoft

Key4hep software stack:

- EDM4hep
- DD4hep
- Gaudi
- Spack

ILCSoft → DONE → Spack

hand-made set of installation scripts used only by us (inherited from CLIC)

advanced package manager used in research and industry

The latest release 2.8 can now be installed with two recipes: for ILCSoft and for Spack

Corresponding Docker images are also available for both variants

↳ mechanism for code modification under Docker is different in each case

Transition step: Gaudi

ILCSoft software stack:

1. LCIO
2. DD4hep
3. Marlin
4. ILCSoft

IN PROGRESS → short term

Key4hep software stack:

- EDM4hep
- DD4hep
- Gaudi
- Spack

ILCSoft	Key4hep
configured via XML	configured via Python
NO multithreading support	built with multithreading in mind

Gaudi has a Marlin-wrapper package → only configuration files have to be adapted (no code changes)

XML config + Bash script → Python config

Transition step: EDM4hep

ILCSoft software stack:

1. LCIO
2. DD4hep
3. Marlin
4. ILCSoft

Key4hep software stack:

- EDM4hep
- DD4hep
- Gaudi
- Spack

LCIO → EDM4hep: **TO BE DONE** → long term

used only by us → no other maintainers

NO multithreading support

used and maintained by other experiments

built with multithreading in mind

All EDM4hep data classes defined in a single YAML file: [edm4hep.yaml] → generates actual C++ code

Switching from LCIO → EDM4hep will change input for all our simulation code

↳ each processor has to be adapted to the new data format → **substantial amount of work**

Event data model: transition plan

On-the-fly EDM4hep ↔ LCIO conversion is available
using EDM4hep2LCIO module developed for CLIC

Beam Induced Background in a single event ▶
simulated in GEANT4 → **120M SimHits**

↳ **enormous amount of data to be processed**
~25 GB (SimHits) **+ ~10 GB** (RecHits) **of RAM**

We can't afford in-memory conversion of all SimHits
but can be feasible for filtered digitized RecHits

↳ **transition to EDM4hep must happen in one step**
for all the code taking SimHits as input: BIB overlay + digitisers

	Collection name	# of elements
SimCalorimeterHit	ECalBarrelCollection	52.219.721
	ECalEndcapCollection	11.489.880
	HCalBarrelCollection	20.657.110
	HCalEndcapCollection	15.296.598
	HCalRingCollection	1.858.377
SimTrackerHit	InnerTrackerBarrelCollection	2.839.607
	InnerTrackerEndcapCollection	2.553.195
	OuterTrackerBarrelCollection	5.111.755
	OuterTrackerEndcapCollection	3.386.256
	VertexBarrelCollection	2.816.752
	VertexEndcapCollection	2.135.425
	YokeBarrelCollection	273
	YokeEndcapCollection	35.267
	TOTAL	**120.400.216**

Release distribution: CVMFS

A 3rd distribution method established for our software stack: **CMVFS** (CERN Virtual Machine File System)
↳ all the software deployed under a dedicated repository: `/cvmfs/`**muoncollider.cern.ch/**

Makes software readily available on any machine with CVMFS configured (e.g. **lxplus9**.cern.ch)
and compatible OS: Alma Linux 9 (long-term support by CERN till 2035)
↳ **to activate release** `2.8` → `source /cvmfs/muoncollider.cern.ch/release/2.8/setup.sh`

Modifying any part of the release code + adding a new package is fairly straighforward
using dedicated Spack functionality: `spack develop <package>; spack install`

Adopting now the release strategy of Key4hep community

- **common packages**
  as upstream installations
- **nightly releases**
  built every night from the latest code
- **stable releases**
  built every few months

Expecting CVMFS installations to be the primary method for using our software
keeping support for local installations + containers

Several computing resources at CERN have been recently established for Muon Collider to automate our software-related tasks

1. **CVMFS repository:** `/cvmfs/muoncollider.cern.ch/`
   - to store our software for use by the whole collaboration

2. **GitLab group:** https://gitlab.cern.ch/muon-collider
   - Docker image registry with web GUI
   - repository with deployment pipelines: mucoll-deploy
     ↳ running on the dedicated GitLab Runner machines →

3. **OpenStack project:** Muon Collider Software
   - dedicated Virtual Machines to run the lengthy automation tasks set up as GitLab Runners
     - deployment of releases to CVMFS (stable + nightly builds)
     - building of Docker images + conversion to Singularity/Apptainer images
     - running release validation workflows

Adopting more frequent release-deployment cycle requires a reliable validation workflow
to minimise probability of unintented changes

All relevant code organised under a single repository: mucoll-benchmarks →

Each stage from generation to plotting has baseline configuration files and scripts

↳ **referenced and overriden by workflow-specific scripts** → chained in mucoll-deploy pipelines

List of workflows will expand over time
adding generation of signal and BIB samples

Will serve as a practial example of using our software

```
|-- generation/
|     |-- bib/
|     |     |-- fluka_to_slcio.py
|     |     |-- mars_to_slcio.py
|     |-- pgun/
|     |     |-- pgun_to_lcio.py
|     `-- signal/
|           `-- mumu_H_bb_3TeV.sin
|-- simulation/
|     `-- steer_sim.py
|-- reconstruction/
|     |-- steer_reco.xml
|     `-- subconfigs/
|           |-- overlay.xml
|           |-- digi_trk.xml
|           |-- digi_cal.xml
|           |-- reco_trk.xml
|-- analysis/
|     |-- lctuple_drawer.py
|     |-- mcp/
|     |     `-- lctuple.xml
|     |-- sim/
|     |     |-- lctuple.xml
|     |     |-- trk_hit_mcp.py
|     |     `-- cal_hit_mcp.py
|-- plotting/
|     `-- histo_drawer.py
`-- workflows/
      |-- relval/
      |     |-- pgun_reco.sh
      |     |-- Hbb_reco.sh
      |     `-- pgun_bib_reco.sh
      `-- bib_production/
            |-- fluka_3TeV.sh
            `-- fluka_10TeV.sh
```

reference configurations for individual stages

release validation workflow

BIB production workflow

Ongoing activities

You can get an even more practical introduction to our software at the upcoming
MuCol: training on detector design and physics performance tools at CERN (July 5-6, 2023)
↳ will also include a hands-on session of using the latest release

Everyone is welcome to register: https://indico.cern.ch/event/1277924/

First steps towards adopting the EDM4hep data model have started

A summer student at CERN adapting some generation-stage scripts to EDM4hep output format
↳ boosted progress on implementation of Python interfaces to EDM4hep

Next step
↳ **implement BIB-overlay process as a native Gaudi module with EDM4hep input/output**

Expecting more Key4hep-oriented developments soon: interface to ACTS, Gaudi-based digitisers, etc.

Summary

Key4hep has a number of advantages for out simulation workflow
better performance and usability, larger developer community, more future proof

The easy part of Key4hep migration is done
↳ Spack package management

We use CERN computing infrastructure to improve usability and stability of our software
building and validating on CERN machines + deployment to CVMFS

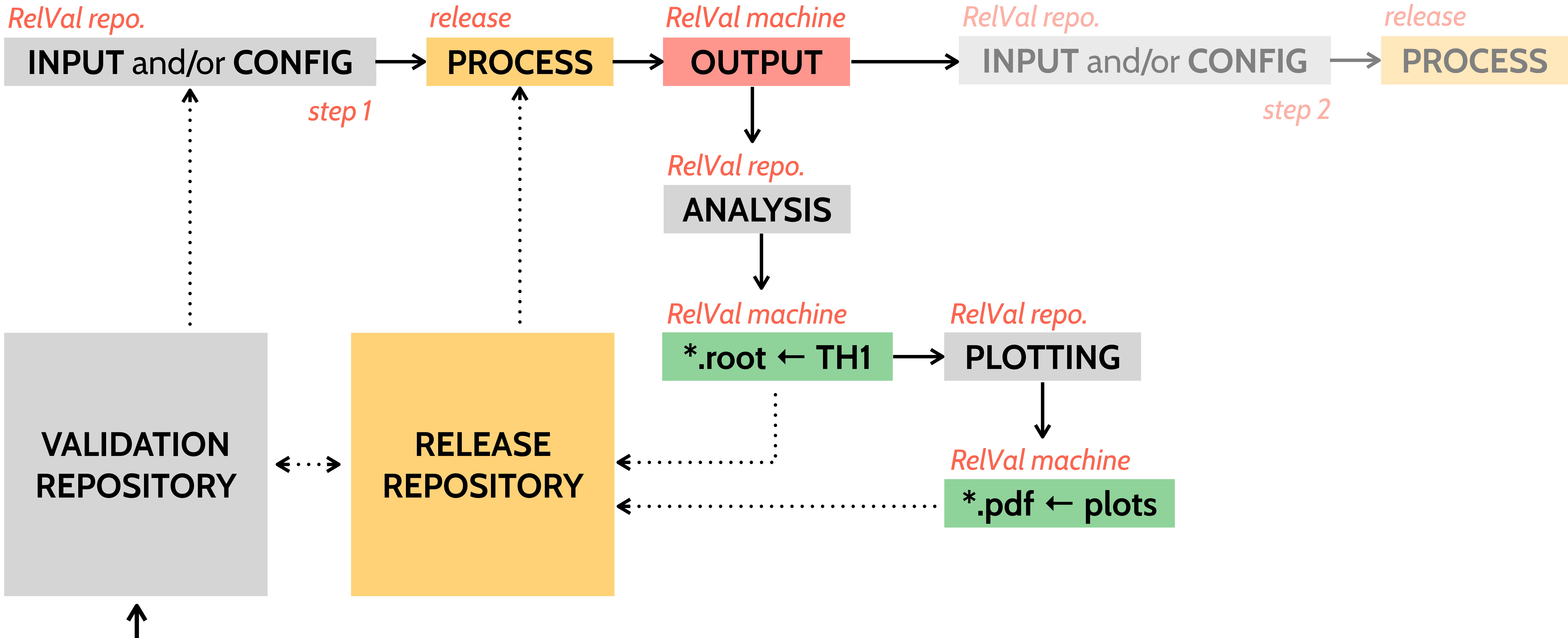
Started the 1st stage of migration to EDM4hep data model
generation → BIB overlay → digitisation → reconstruction

Key4hep community equally interested in us joining the club
↳ invited to share our experience and plans at CEPC workshop
↳ adopted Key4hep a while ago

BACKUP

Event data model: transition plan

The general workflow for Release Validation



Versions synchronised with the release repository

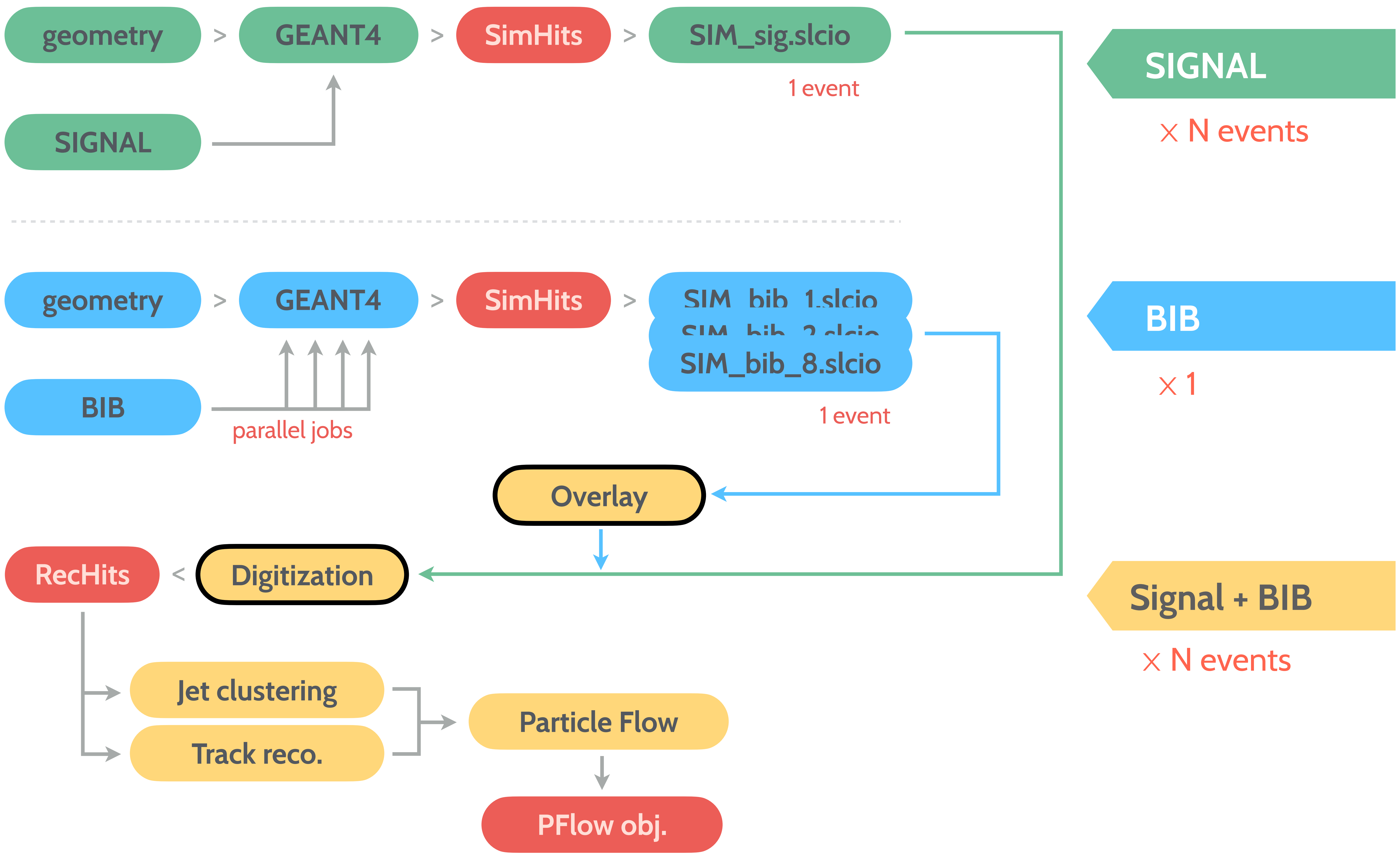
Event data model: transition plan

We need to modify several components of our simulation chain → good candidates for the 1st transition

1. **Overlay**
   dynamic mixing of small batches from FLUKA BIB simulation

2. **Digitisation**
   TRK: realistic treatment of timing
   CAL: more efficient class structure
   + new detectors: *CRILIN, MPGD*

3. **Track reconstruction**
   parallel processing of multiple slices in ϕ



We can start with <u>Overlay</u> processor
working only with EDM4hep SimHits

↳ making it with optimised I/O and multithreaded

Data-model optimisation: CAL hits

SimCalorimeterHit in EDM4hep
identical to LCIO implemenation

- **SimHit:** 32 bytes
- **Contribution:** 32 bytes

```
#------------- CaloHitContribution
edm4hep::CaloHitContribution:
  Members:
    - int32_t           PDG            // PDG code of the particle contributing to the shower
    - float             energy         // energy in [GeV] of the this contribution
    - float             time           // time in [ns] of this contribution
    - edm4hep::Vector3f stepPosition   // position of this energy deposition (step) [mm]
  OneToOneRelations:
    - edm4hep::MCParticle particle     // primary MCParticle that caused the shower

#------------- SimCalorimeterHit
edm4hep::SimCalorimeterHit:
  Members:
    - uint64_t          cellID    // ID of the sensor that created this hit
    - float             energy    // energy of the hit in [GeV]
    - edm4hep::Vector3f position  // position of the hit in world coordinates in [mm]
  OneToManyRelations:
    - edm4hep::CaloHitContribution contributions  // MC step contribution - parallel to particle
```

100M objects stored on disk + read into RAM + processed by CPU in every event during Overlay
↳ **on average 10 contributions / SimCalorimeterHit** → 354 B/hit

We can save a lot of memory by removing redundant and non-critical information: 88 B/hit (25%)

- **`SimCalorimeterHit::position`** → **we already know it from cellID**
- **`CaloHitContribution::stepPosition`** → **exact position within a cell is irrelevant for digitization**

The power of splitting Tracker hits in smaller subsets has been demonstrated by Massimo long ago

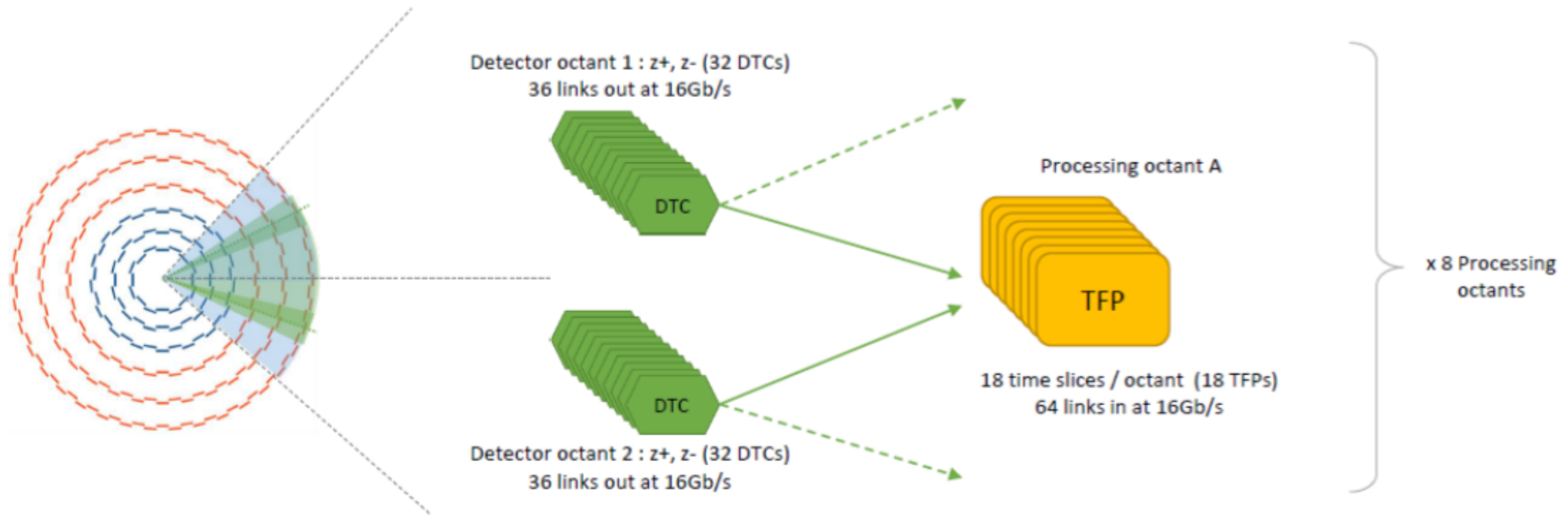
↳ less input hits in a single subset → much less combinatoriscs for track reconstruction

Splitting in polar angle might not be optimal

BIB density is not uniform in Θ

CMS Phase-II Tracker will be split into 8 octants

for fast tigger-level track reconstruction



We should integrate this approach in our workflow
making it a default taking advantage of parallelization in Gaudi

- **Overlay:** adding BIB hits to every Tracker hit collection as we do now
- **Splitting:** split each Tracker hit collection in ϕ sectors
- **Digitization:** run digitization of each ϕ sector in parallel [lin. speed-up]
- **Filtering:** stub matching in each ϕ sector in parallel [lin. speed-up]
- **Track reconstruction:** run ACTS tracking in each sector independently [exp. speed-up]
  + maybe apply splitting in Θ internally at the level of a processor