Muon Collider Software

current status and future plans

N. Bartosik [a, b]

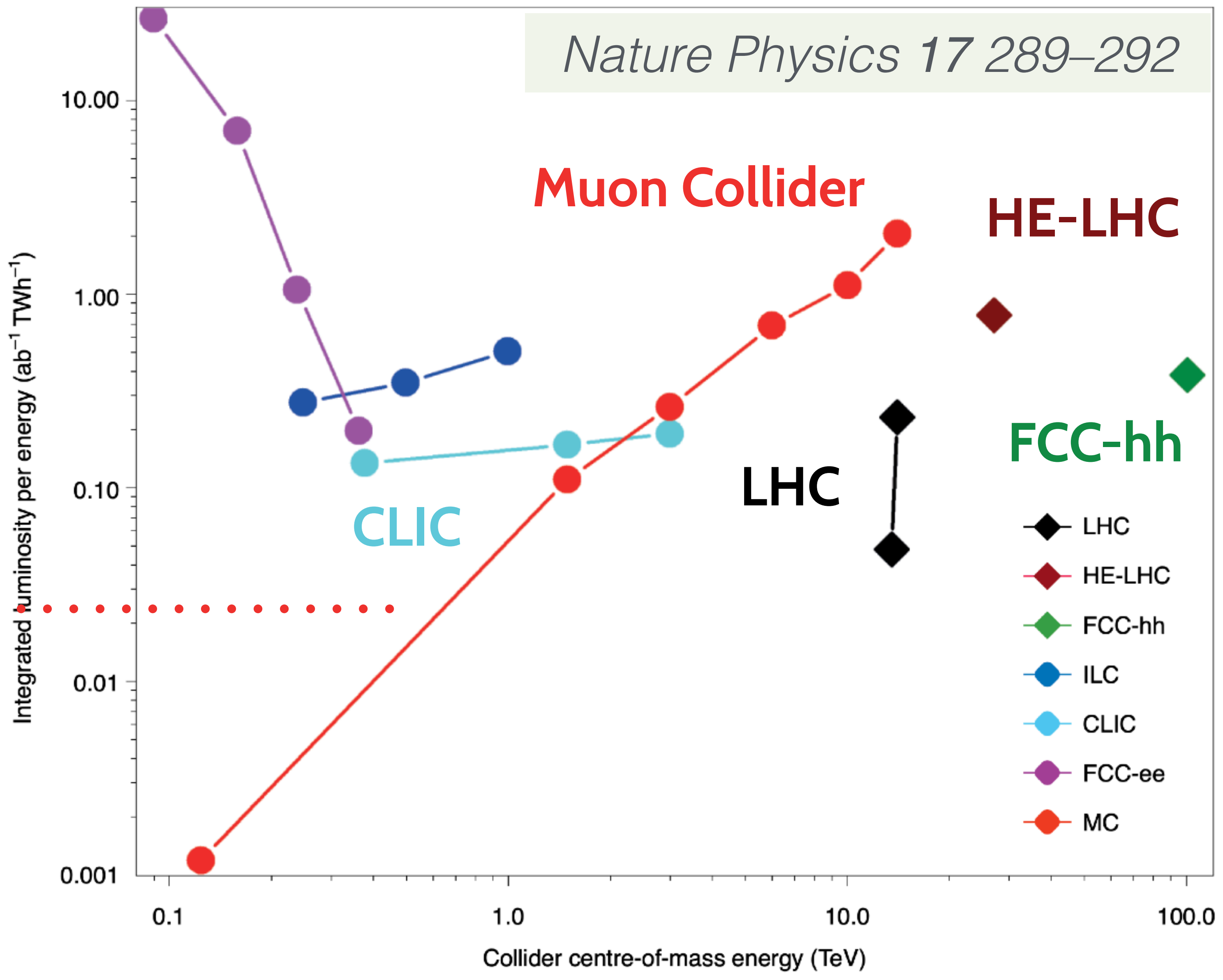
for the Muon Collider Physics and Detector Group

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

Muon Collider allows to combine in a single facility
high precision of e^+e^- colliders + high energy reach of *pp* colliders

- muons are elementary particles, like e^+/e^-, creating "*clean*" collisions
- $\times$ 200 higher mass → $\times$ 10^4 less synchrotron radiation losses

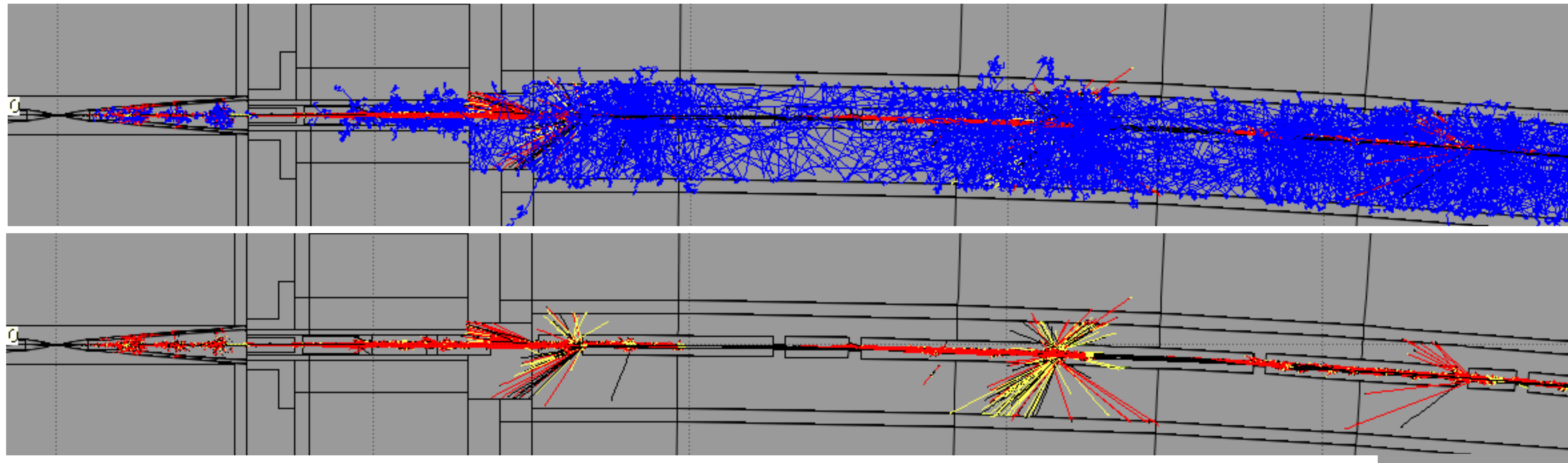
At $\sqrt{s} \geq 3$ TeV Muon Collider is the most energy efficient machine



Distinctive feature of the Muon Collider
↳ **Beam Induced Background (BIB)**
4.1×10^5 muon decays per meter of lattice at $\sqrt{s} = 1.5$ TeV

BIB is simulated in two steps:

- muon decays + interaction with accelerator lattice
  using FLUKA ▶
- interaction of particles with the detector
  using GEANT4

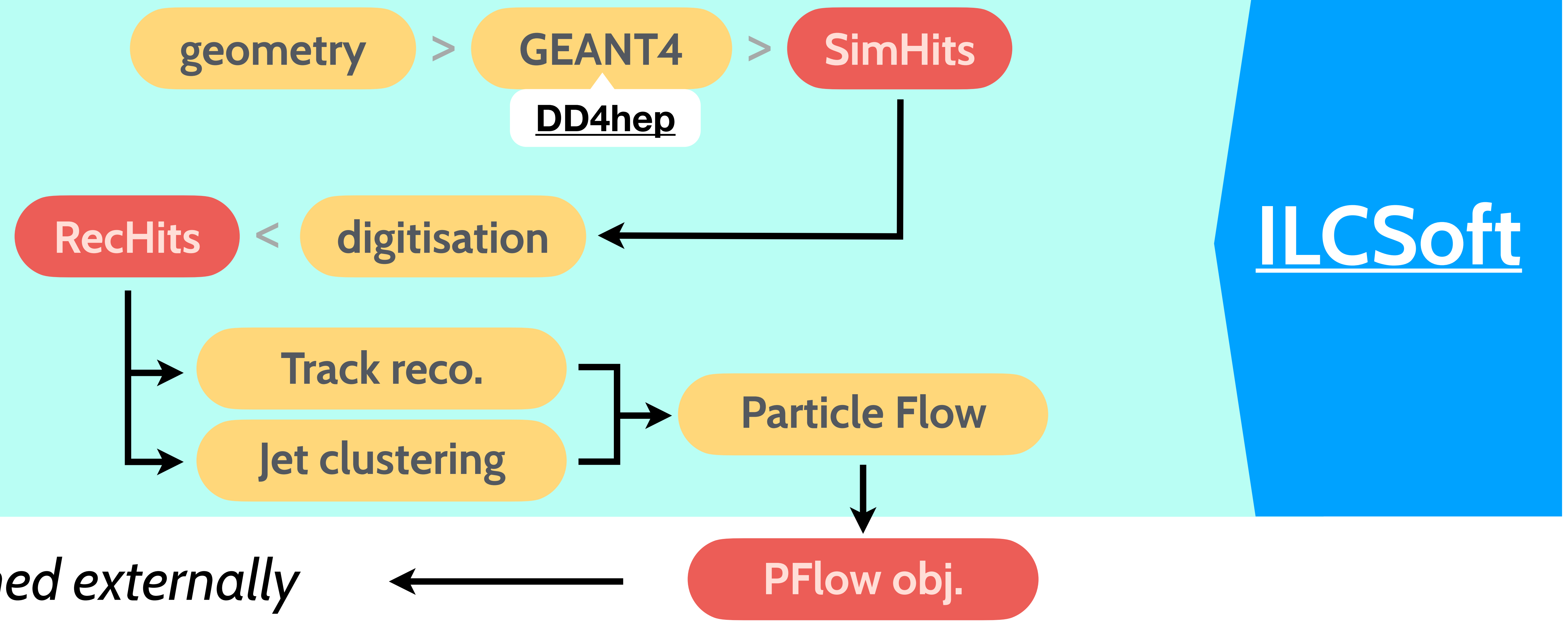


FLUKA simulation
neutrons photons electrons positrons

Main steps of a full-simulation study:

Matrix Element + Parton Shower — Beam Induced Background

1. **generation of stable input particles:** ME + PS **Pythia/Madgraph/Whizard** + BIB **FLUKA + FlukaLineBuilder**
2. **simulation of the detector response to the incoming particles**
3. **simulation of detector effects**
   efficiency, electronics noise + thresholds, ...
4. **reconstruction of higher-level objects**
   photons, tracks, jets, particle identification
5. **higher-level analysis** ← *can be performed externally* ←



Most of custom packages specific to the Muon Collider
maintained under the public Muon Collider Software repository

The whole simulation chain is fully working
with a dozen of physics-analyses and detector-design studies

Simulation workflow

Full simulated event obtained via three distinct stages:

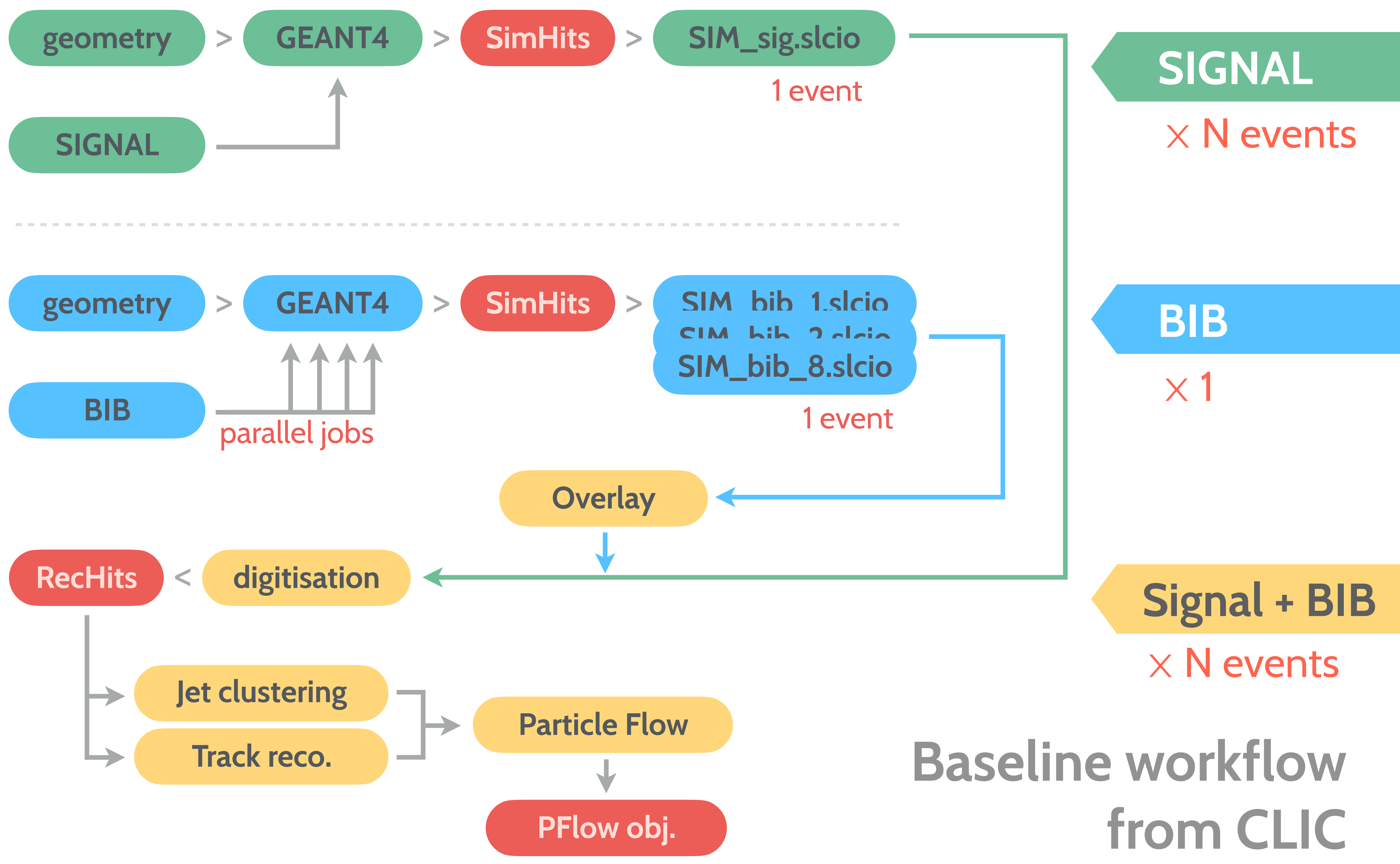
GEANT4 simulation of Signal: straightforward and fast

GEANT4 simulation of BIB: ~10^8 particles/BX
↳ **extremely slow** → need a pool of reusable events

Overlay of BIB: performed in each event before digitisation
↳ sensitive to the # of BIB SimHits and merging logics

Reconstruction speed of higher-level objects strongly depends on the amount of input RecHits from BIB

- especially relevant for track reconstruction *(combinatorics)*
- BIB contribution has to be suppressed as early as possible



Baseline workflow from CLIC

BIB contribution creates tremendous amount of data → every step requires careful treatment of computing resources

DISK STORAGE | DISK I/O | CPU TIME | RAM USAGE | DISTRIBUTION

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 to Key4hep started a few months ago

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

DONE →

Key4hep software stack:

- EDM4hep
- DD4hep
- Gaudi
- Spack

The latest release 2.8 can now be installed with two recipes: for ILCSoft and for Spack
↳ includes specific versions for the main dependencies: *ROOT, GEANT4, Marlin packages, etc.*

Spack installation also available under CVMFS: `/cvmfs/muoncollider.cern.ch/release/2.8-patch2/`

Using key4hep-spack as a downstream repository → only 11 mucoll-spack packages on top
↳ need to use specific commits from `spack` and `key4hep-spack` repositories to ensure stable builds

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

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

TO BE DONE → long term

Key4hep software stack:

- EDM4hep
- DD4hep
- Gaudi
- Spack

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

Release automation: using CERN resources

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

A more practical introduction to our software available in the training Wiki page

↳ created for a series of software tutorial sessions: the latest starts today at CERN

First steps towards adopting the EDM4hep data model have been made

↳ using EDM4hep for storing MCParticles from BIB and particle-gun samples

Next step in EDM4hep adoption: implement BIB overlay natively in Gaudi

↳ exact schema to use is not clear yet: support for intra-event parallelisation is important

Expecting more Key4hep-oriented developments in the near future
interface to ACTS tracking, Gaudi-native digitisers, etc.

↳ all must be thread-safe for intra-event parallelisation

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

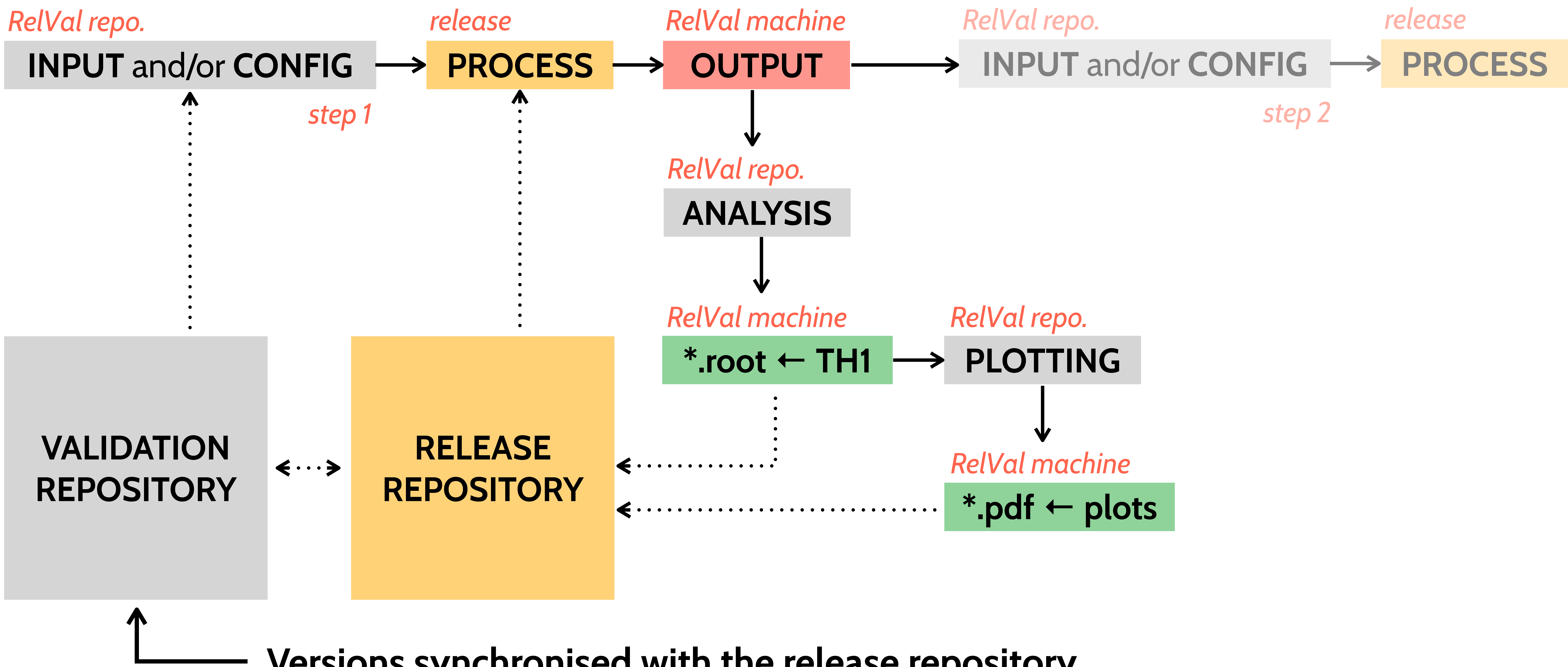
Started the 1st stage of migration to EDM4hep data model
generation → BIB overlay → digitisation → reconstruction

Several custom developments for Marlin to be migrated to Gaudi
↳ realistic digitisation of pixel detectors (timing, clustering); interface to ACTS;

Synergies with CEPC and other Key4hep contributors are highly appreciated
↳ reconstruction algorithms, analysis tools, etc. + documentation

BACKUP

The general workflow for Release Validation



Versions synchronised with the release repository

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%)

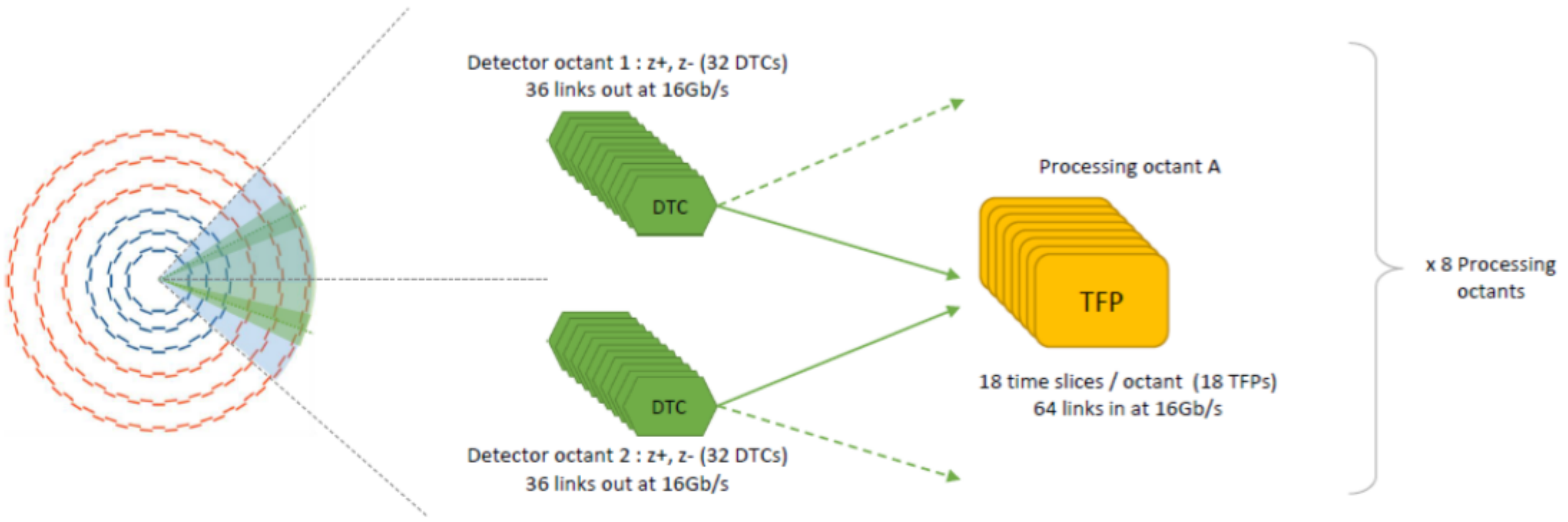
- `SimCalorimeterHit::position` → **we already know it from cellID**
- `CaloHitContribution::stepPosition` → **exact position within a cell is irrelevant for digitization**

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