February 14th, 2023

Detector & MDI meeting

Key4HEP migration plan

for the Muon Collider software

from ILCSoft framework

N. Bartosik [a, b]

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

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

Muon Colider software: current stack

The main components of our current software stack:

1. **LCIO** → **data format** [`LCIO::SimCalorimeterHit`, `LCIO::MCParticle`, ... stored in *.slcio files]

Muon Colider software: current stack

The main components of our current software stack:

1. **LCIO** → **data format** [`LCIO::SimCalorimeterHit`, `LCIO::MCParticle`, ... stored in *.slcio files]
2. **DD4hep** → **flexible geometry-description language** + interface with Geant4

Muon Colider software: current stack

The main components of our current software stack:

1. **LCIO** → **data format** [`LCIO::SimCalorimeterHit`, `LCIO::MCParticle`, ... stored in *.slcio files]
2. **DD4hep** → **flexible geometry-description language** + interface with Geant4
3. **Marlin** → **framework for writing simulation code** + chaining them together via *.xml files

Muon Colider software: current stack

The main components of our current software stack:

1. **LCIO** → **data format** [`LCIO::SimCalorimeterHit`, `LCIO::MCParticle`, ... stored in *.slcio files]
2. **DD4hep** → **flexible geometry-description language** + interface with Geant4
3. **Marlin** → **framework for writing simulation code** + chaining them together via *.xml files
4. **ILCSoft** → **framework for putting together all the necessary software on a user's machine**
   ↳ collection of Python scripts and configuration files *(package URLs, versions, etc.)* to install dependencies, compile Marlin packages, etc.

Muon Colider software: current stack

The main components of our current software stack:

1. **LCIO** → **data format** [`LCIO::SimCalorimeterHit`, `LCIO::MCParticle`, ... stored in *.slcio files]
2. **DD4hep** → **flexible geometry-description language** + interface with Geant4
3. **Marlin** → **framework for writing simulation code** + chaining them together via *.xml files
4. **ILCSoft** → **framework for putting together all the necessary software on a user's machine**
   ↳ collection of Python scripts and configuration files *(package URLs, versions, etc.)* to install dependencies, compile Marlin packages, etc.

In the meantime a new software stack has emerged: [Key4hep](#)
that is used in several experiments: CLIC, FCC, CEPC, ILC → clearly more future-proof

Using tools with a larger user base we can profit from developments by other experiments
↳ evolving HEP tools will be more compatible with Key4hep that with ILCSoft

Particle Flow: PandoraPFA → Pandora SDK → k4Pandora; **Clustering:** CLUE → k4Clue;

Transition step: ILCSoft

Our software stack:		**Key4hep software stack:**
1. LCIO		EDM4hep
2. DD4hep		DD4hep
3. Marlin		Gaudi
4. ILCSoft	→ **EASY** →	Spack
	custom set of installation scripts used only by us	advanced package manager used by industry

All our current software stack can be set up using <u>Spack</u> instead of <u>ILCSoft</u> install scripts

↳ **more elegant solution** → no need to copy&paste installation commands in the terminal

Only initial effort required for configuring the present environment in Spack fashion

↳ all further maintenance should be more straightforward than with ILCSoft

Nothing would change for users of Docker → installation process already baked into the image

Transition step: Marlin

Our software stack:

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

Key4hep software stack:

- EDM4hep
- DD4hep
- Gaudi
- Spack

EASY: Marlin → Gaudi

older framework with plenty of existing processors: *CLIC, MuC*

- jobs configured with XML
- NO parallelisation mechanism

newer framework with less existing code, but much better usability

- jobs configured with Python
- parallelisation mechanism provided

Gaudi has MarlinProcessorWrapper → we can easily run all our workflow in Gaudi
↳ **no code changes required** → only Marlin configuration files need to be rewritten in Python

Python configuration is more intuitive and programmable → perfect for systematic variations

XML config + Bash script → Python config

Transition step: LCIO

Our software stack:

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

DIFFICULT →

Key4hep software stack:

- EDM4hep
- DD4hep
- Gaudi
- Spack

LCIO: custom data format used only by Muon Collider now

- limited support of parallelization
- will require maintenance for compatibility with future tools

EDM4hep: unified data format built with podio and used by several future experiments

- designed with multithreading in mind
- interfaces with other tools are better maintained by the community
  e.g. TPC hits, Dual Readout calo. hits

All EDM4hep data classes conveniently defined in a single YAML file: edm4hep.yaml

↳ **all the actual C++ code for compilation is generated with a Python script → clean schema evolution**

Switching from LCIO to EDM4hep would change input for all our Marlin processors

↳ **each processor would have to be adapted to the new data format** → quite a lot of work in some cases

Transition in a single step would be too difficult → need a staged approach

BIB overlayed to a single event ▶
simulated in GEANT4 → **120M SimHits**

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

On-the-fly LCIO → EDM4hep conversion possible
using EDM4hep2LCIO processor developed for CLIC

We can't afford in-memory conversion of all SimHits

Doing it for filtered digitized hits might be feasible

↳ little extra RAM needed if we delete original collections from memory after the conversion

	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**

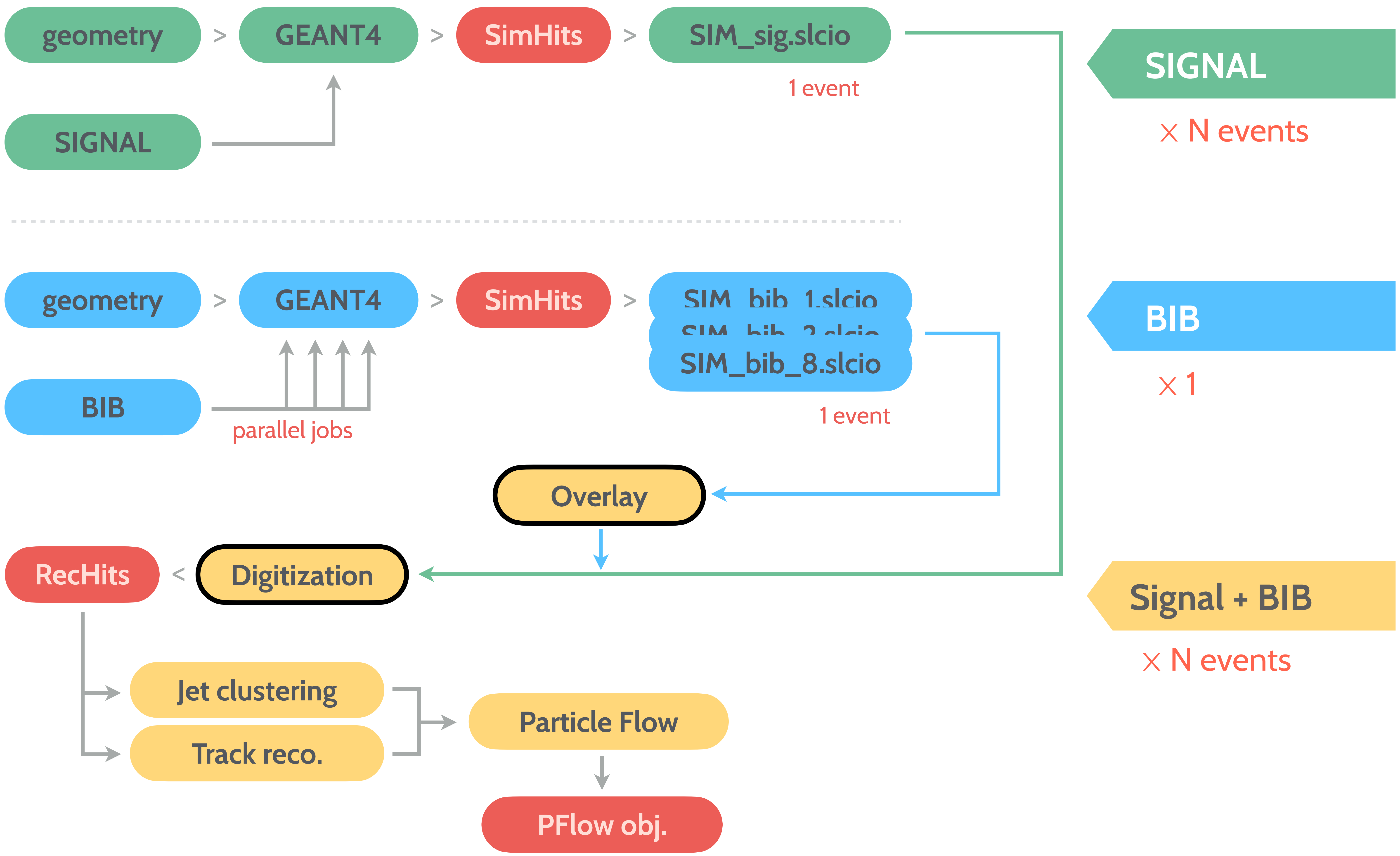
Transition step: LCIO

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

3. **Track reconstruction**
   parallelised execution of multiple $\Delta\phi$ slices



We can start with Overlay 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 shower particle that caused this contribution
    - 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
```

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 shower particle that caused this contribution
    - 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 process
↳ **on average 10 contributions / SimCalorimeterHit** → 354 B/hit

We can 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**

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 shower particle that caused this contribution
    - 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 process
↳ **on average 10 contributions / SimCalorimeterHit** → 354 B/hit

We can 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**

Positions are handy for drawing. BUT we never draw directly from LCIO files → can be added in LCTuple

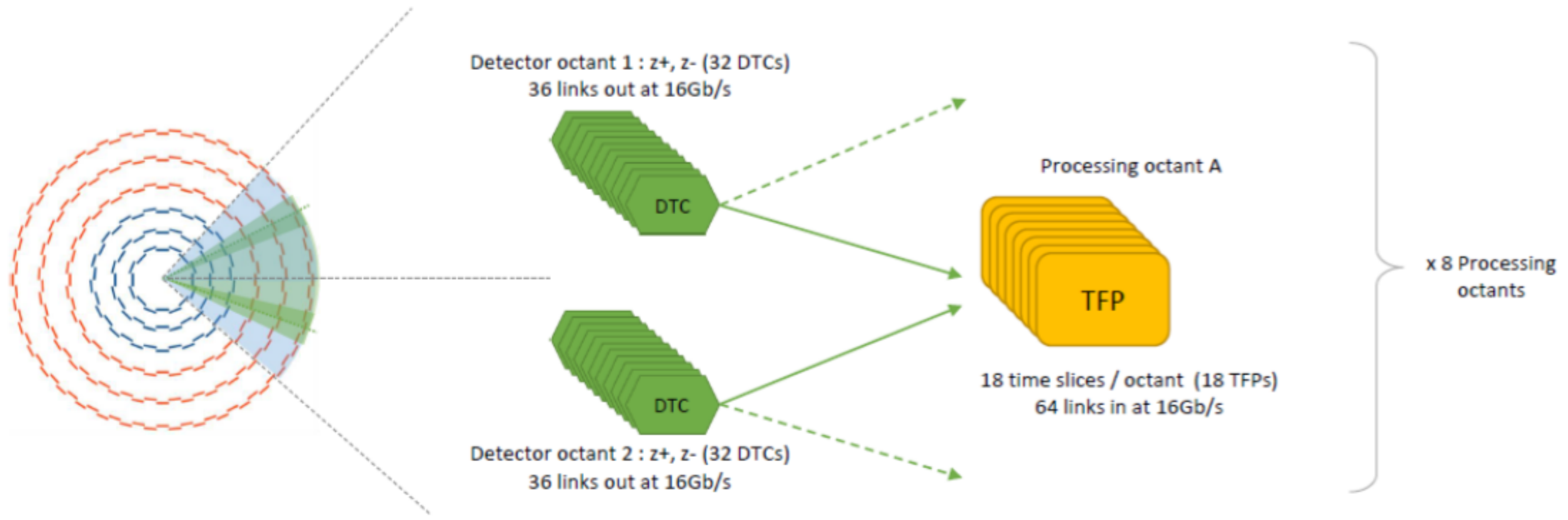
Tracking optimisation: ϕ slicing

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

Project proposal submitted for the CERN summer school in 2023

on behalf of the CERN software department *(agreed with Andre Sailer)*

Integration of Muon Collider simulation code into Gaudi framework

Project description

Muon Collider is a promising candidate for a flagship post-LHC energy-frontier machine, which for the first time in history would collide high-energy beams of unstable muons. Its design study requires very high computational efficiency in order to accurately simulate effects from background radiation of unprecedented intensity.

This project will focus on implementation of the "background overlay" package that mixes into a single event detector signals from the primary collision and signals from background particles. The existing algorithm implemented in Marlin and struggles with ~10^8 particles/event present at Muon Collider. Therefore it has to be rewritten for an improved use of computing resources.

This project is part of the larger effort towards gradual transition of the present simulation code to Gaudi framework, adopting Key4hep software stack. In practice this work will include:

- adapting code to Gaudi-native EDM4hep format of input data;
- adopting Gaudi multithreading interface for intra-event parallelization;
- implementing user-configurable filters of input collections to reduce RAM usage;
- validation and profiling of code performance as part of the simulation chain.

The selected candidate will work closely with members of the *Muon Collider Detector and Physics* group, interacting regularly with Key4hep developers from the EP-SFT group. Once finished, this code will become part of the official Muon Collider software release and will be used in all future simulation studies performed by the collaboration.

Supervisors:

- **Nazar Bartosik** (Muon Collider)
- **Juan Miguel Carceller Lopez** (Key4hep)

The outcome will be known closer to June

Summary

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

Most of the software stack can be applied directly without any changes in our code
Spack package management + Gaudi processing framework

Change of the data model is a longer-term issue to done in steps
keep using LCIO for the most part

I would try writing the new Overlay processor based on EDM4hep data model
1st step towards multithreading of our simulation process

Then we gradually migrate subsequent steps to EDM4hep
hit filtering → digitization → reconstruction