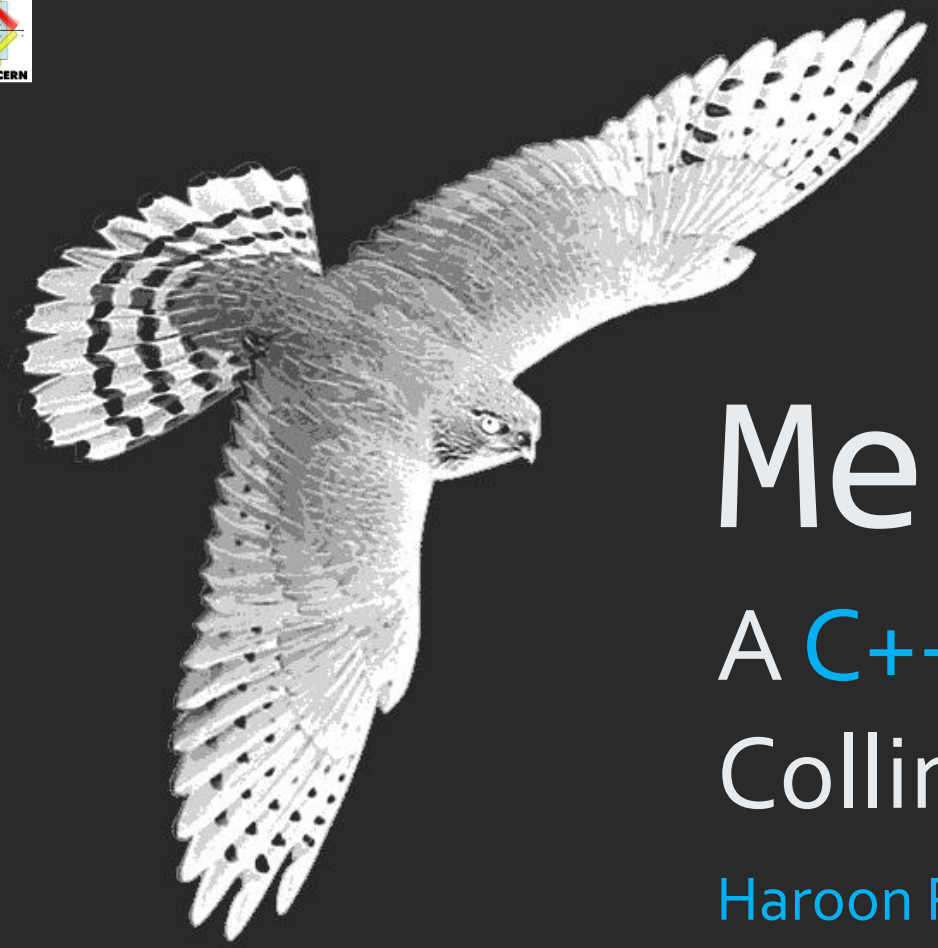


Merlin

A C++ Class Library for Collimation Studies

Haroon Rafique, R. Appleby, R. Barlow

J. Molson, M. Serluca, A. Toader

Contents

- Short & Long Term Plans
- Merlin
- Example 1: No input files
- Example 2: tfs, collimator and aperture input files
- Materials
- ScatteringModel
- ScatteringProcess
- Collimation Process
- User Defined PhysicsProcess

Introduction

General code

Developed for Accelerator Physics

Structure being prepared for HiLumi era

(Materials, PhysicsProcesses etc)

Collimation code

Developed for LHC loss maps

Currently used for loss maps

Implementing new scattering

Release 5-01

New scattering Comparable with Sixtrack

Merge Optimise

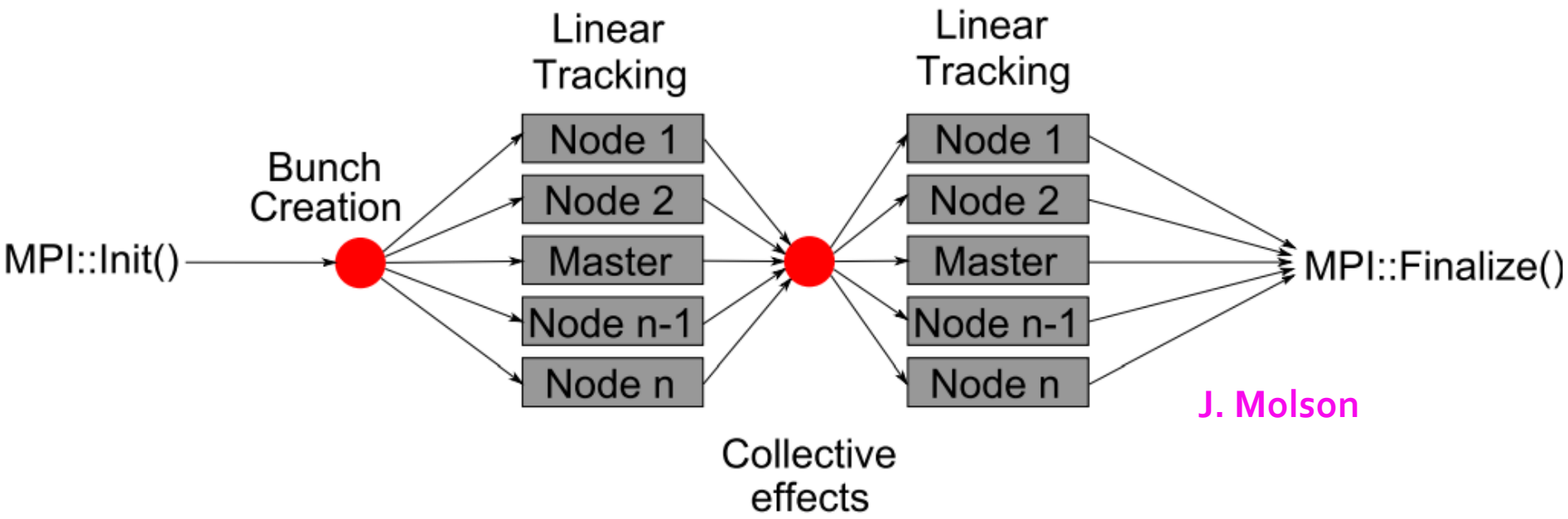
Release 5-02

Merlin

- C++ Accelerator physics library
- Originally created by Nick Walker (ILC) @ DESY
- Storage ring functionality added by Andy Wolski
- **Modular design**
  - User writes own code
  - 3 main sections:
    - Accelerator lattice creation
    - Tracker
    - Physics processes
  - Can take MAD tfs table or XTFF for input
  - Created AcceleratorModel can be manipulated
  - All AcceleratorComponents have
    - EMField
    - AcceleratorGeometry
    - Aperture
- Additional PhysicsProcesses may be applied over tracking, at user defined locations in lattice (e.g. at specific elements)
- PhysicsProcesses such as Collimation exist
- User may define own PhysicsProcess easily
- Uses simple MPI: code split onto N nodes
- Structure appropriate for HiLumi interests:
  - Novel materials (mixtures)
  - Novel collimation; e-lens, crystal etc

Merlin

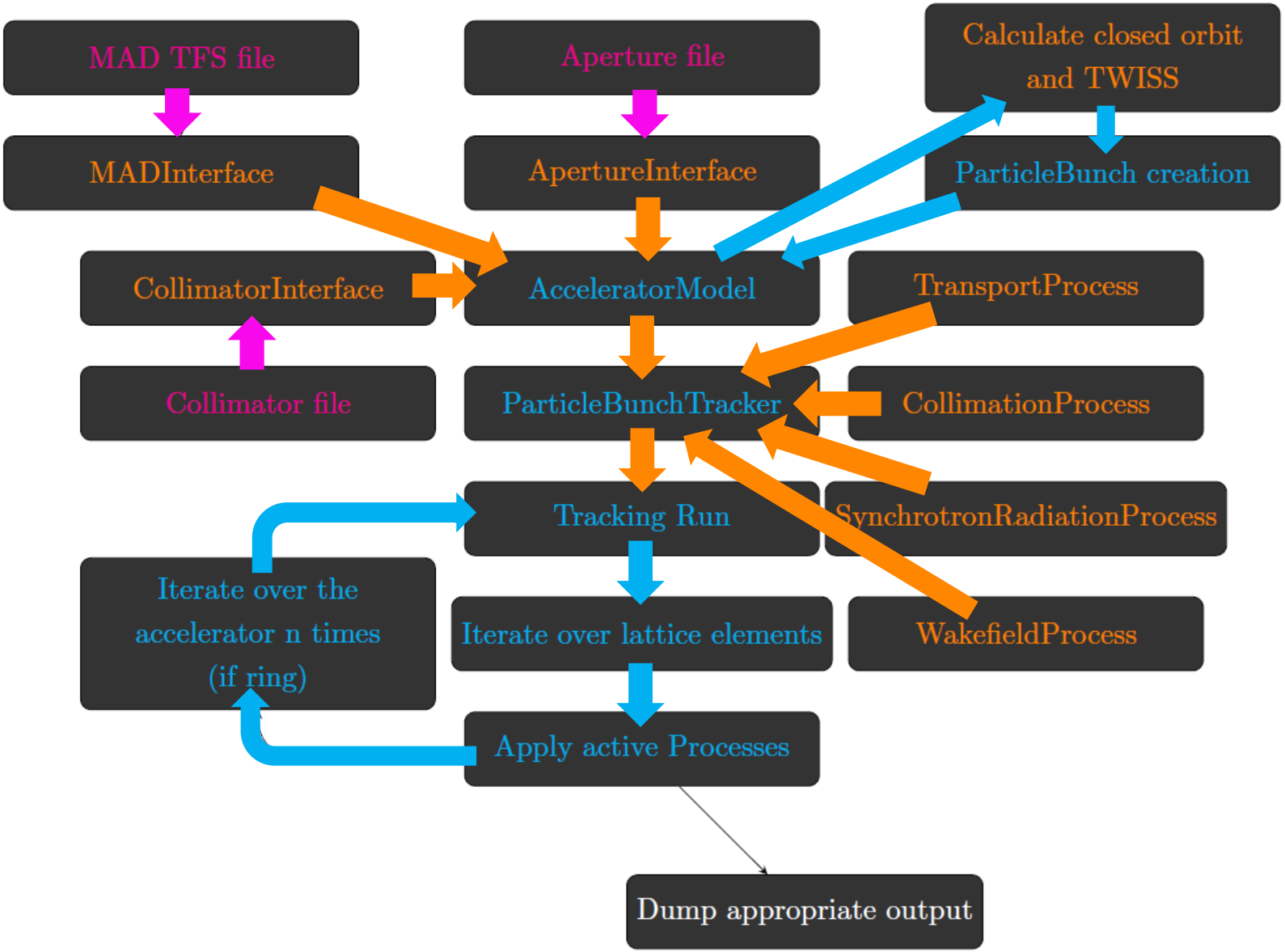
- C++ Accelerator physics library
- Originally created by Nick Walker (ILC) @ DESY
- Additional PhysicsProcesses may be applied over tracking, at user defined locations in lattice (e.g. at specific elements)



J. Molson

- All AcceleratorComponents have
  - EMField
  - AcceleratorGeometry
  - Aperture
- Novel collimation; e-lens, crystal etc

Merlin 5-01


MAD TFS file
Aperture file
Calculate closed orbit and TWISS
MADInterface
ApertureInterface
ParticleBunch creation
CollimatorInterface
AcceleratorModel
TransportProcess
Collimator file
ParticleBunchTracker
CollimationProcess
Tracking Run
SynchrotronRadiationProcess
Iterate over the accelerator n times (if ring)
Iterate over lattice elements
WakefieldProcess
Apply active Processes
Dump appropriate output

Example 1: No input files

1. Construct AcceleratorModel using individual AcceleratorComponents, and AcceleratorModelConstructor
2. Construct ProtonBunch using ParticleBunchConstructor
3. Construct ParticleTracker using AcceleratorModel::Beamline or ::RingIterator
4. Add PhysicsProcesses to ParticleTracker
5. Track
6. Output

Class

User-defined object

Function

Other code

Pseudo code

Example 1: AcceleratorModel

```
AcceleratorModelConstructor* myAccCtor = new AcceleratorModelConstructor();
        myAccCtor->NewModel();

Quadrupole* quad = new Quadrupole( Name , Length, K1 );
        myAccCtor->AppendComponent( *quad );

ScatteringModel* myScatter = new ScatteringModel;
        myScatter -> AddProcess( new Process() );
        myScatter -> AddProcess( new Inelastic() );

Collimator* coll = new Collimator
         (Name, Length, Material, ScatteringModel, Beam Momentum);
        Aperture* ap = new CircularAperture( .0002 );
        coll -> SetAperture( ap );
        myAccCtor -> AppendComponent( *coll );

AcceleratorModel* mymodel = myAccCtor -> GetModel();
```

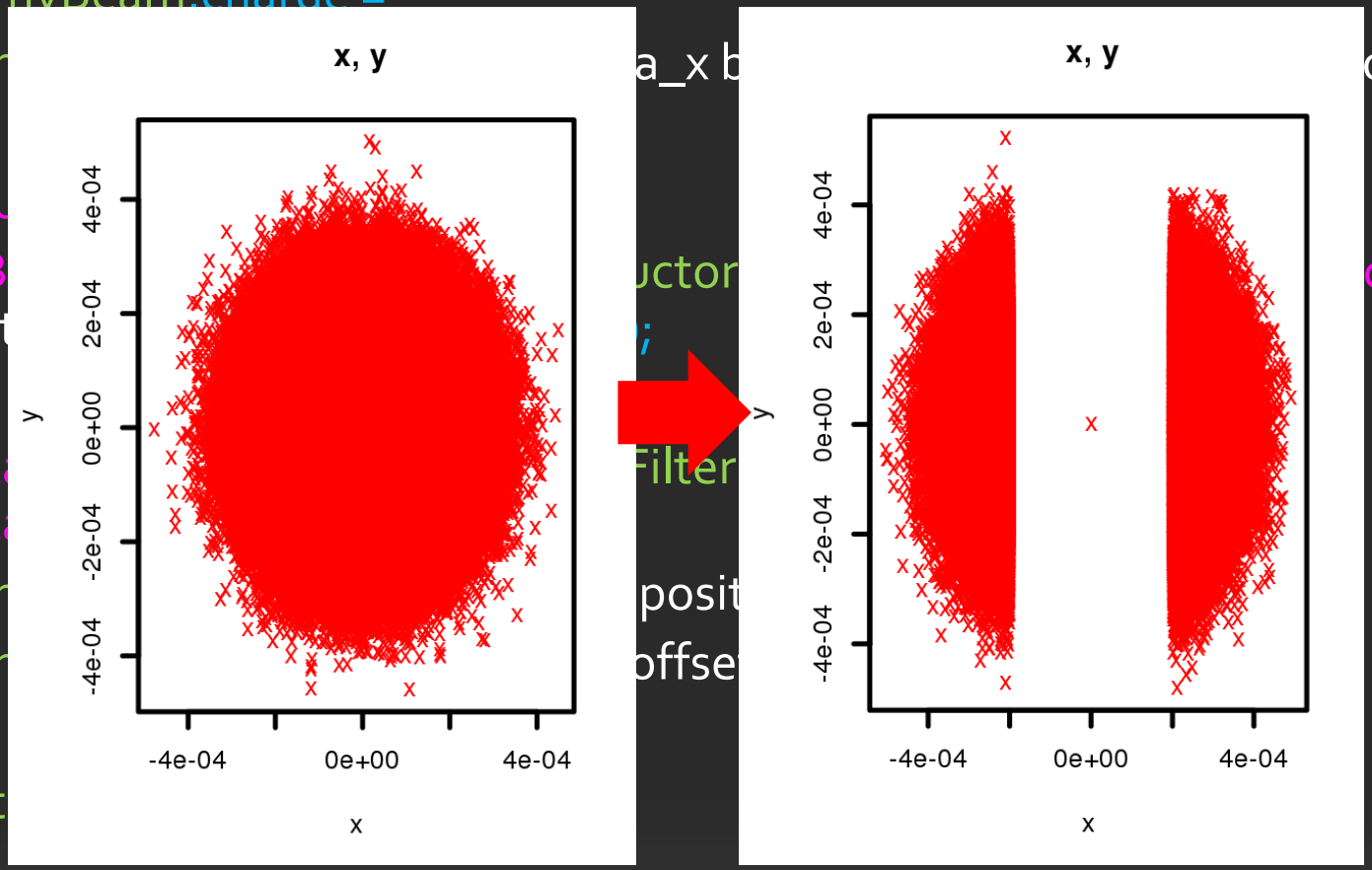
Class
User-defined object
Function
Other code
Pseudo code

Example 1: Bunch

```
BeamData myBeam;
        myBeam.charge =
        myBeam.alpha_x alpha_y beta_x beta_y emit_x emit_y sig_z, xo, etc

ProtonBunch* myBunch;
ParticleBunchConstructor* myConstructor = new ParticleBunchConstructor(
BeamData, No_particles, Distribution);

HorizontalHaloParticleBunchFilter* hFilter = new
HorizontalHaloParticleBunchFilter();
        hFilter -> SetHorizontalLimit (position);
        hFilter-> SetHoriztonalOrbit (offset);

myConstructor -> SetFilter( hFilter );

myBunch = myConstructor->ConstructParticleBunch<ProtonBunch>();
```

Example 1: Bunch

```
BeamData myBeam;
        myBeam.charge =
        r                                   a_x b                                   o, etc

ProtonBu
ParticleB                                   uctor                                   ctor(
BeamDat                                     ;

Horizonta                                   Filter
Horizonta
        h                                   posit
        h                                   offse

myConst
```




```
myBunch = myConstructor->ConstructParticleBunch<ProtonBunch>();
```

Example 1: Tracker

```
//Ringiterator for circular/repeating lattice
AcceleratorModel::RingIterator ringbeamline = model -> GetRing();
ParticleTracker* myRingTracker = new ParticleTracker( ringbeamline, myBunch );

//Beamline for one-pass lattice
AcceleratorModel::Beamline beamline = model -> GetBeamline ( start element name, end element name );
ParticleTracker* myLineTracker = new ParticleTracker( beamline, myBunch );
```

Example 1: PhysicsProcess

```
CollimateProtonProcess* myCollimationProcess = new CollimateProtonProcess(
mode, priority, ofstream);
        myCollimationProcess -> ScatterAtCollimator( true );
        myCollimationProcess -> SetLossThreshold ( percent );

//Ring Tracker
myRingTracker -> AddProcess( myCollimationProcess );

//Beamline Tracker
myLineTracker -> AddProcess( myCollimationProcess );
```

Example 1: Track

```
//single pass
myLineTracker -> Track (myBunch)

//many turns
double nturns = 100;

for( int i = 0; i <= nturns; i++ ){
        myRingTracker -> Track (myBunch);
        if( myBunch.size() <= 1){
                cout << "Exiting in turn " << i << endl;
                break;
        }
}
```

Example 1: Output

Automatic loss file output done via CollimateProtonProcess

Additional output available

```
// User defined 'special' output
myCollimationProcess -> SpecialOutput (ofstream);

// Collimator losses only
myCollimationProcess -> OutputCollimatorLosses (ofstream);

// Final bunch
myBunch -> Output (ofstream);
```

Example 2: .tfs + Collimator + Aperture input files

1. Construct AcceleratorModel using MAD .tfs input
2. Modify AcceleratorModel using Collimator and Aperture input files
3. Construct ProtonBunch using ParticleBunchConstructor
4. Construct ParticleTracker using AcceleratorModel::Beamline or ::RingIterator
5. Add PhysicsProcesses to ParticleTracker
6. Track
7. Output

Class

User-defined object

Function

Other code

Pseudo code

Example 2: .tfs -> MADInterface

```
MADInterface* myMADinterface = new MADInterface("twiss.7.0tev.b1.tfs", energy);
        myMADinterface->TreatTypeAsDrift("RFCAVITY");
        myMADinterface->ConstructApertures(false);

AcceleratorModel* myAccModel = myMADinterface->ConstructModel();
```

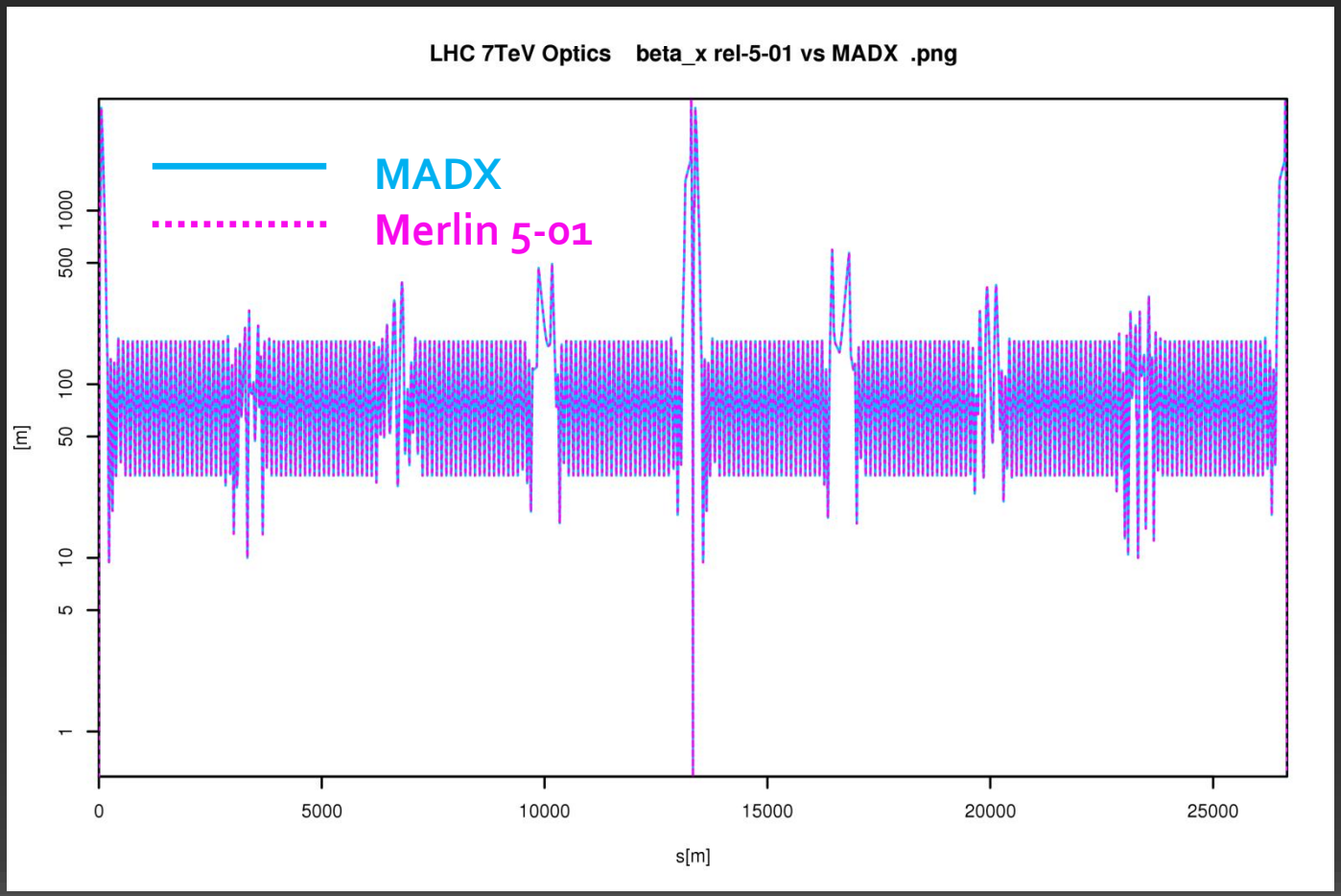
Example 2: CollimatorInterface

```
ScatteringModel* myScatter = new ScatteringModel;
        myScatter -> AddProcess ( new Rutherford() );
        myScatter -> AddProcess ( new NucleonElastic() );
        myScatter -> AddProcess ( new Inelastic() );
        myScatter -> AddProcess ( new NucleusElastic() );
        myScatter -> AddProcess ( new NucleonDiffractive() );

CollimatorInterface* myCollinterface = new
CollimatorInterface("collimator.7.0.sigma", bool use_sigma, ScatteringModel, P0);
        myCollinterface->SelectImpactFactor("TCP.C6L7.B1", 1.0E-6);

double impact = myCollinterface -> ConfigureCollimators ( AcceleratorModel,
emittance_x, emittance_y, LatticeFunctionTable);
```

Example 2: ApertureInterface

```
ApertureInterface* myAPinterface = new ApertureInterface("aperture_7TeV.tfs");
myAPinterface->ConfigureElementApertures(myAccModel);
```

Tracker, PhysicsProcesses, Track, Output as in previous example (slides 8-12)



Materials

- Collimator interactions depend on material
- Collimators can have materials from the StandardMaterials database or user defined materials

```
MaterialProperties* Uo =  new MaterialProperties (Atomic Mass, Atomic Number,
Sigma_E, Sigma_I, Sigma_R, dEdx, Radiation Length, Density, Conductivity,
MeanExcitationEnergy );

MaterialProperties* Be =  new MaterialProperties (9.012182, 4,  0.069, 0.199,
                    0.000035, 1.594, 65.19, 1.848, 3.08E7, (63.7*eV));
Be.PrintTable();

Materials mix1;
        mix1.StandardMaterials();
//MakeMixture( name, constituent elements, proportions, density, conductivity )
mix1.MakeMixture ("Mix1", "Al Cu W", 1, 2, 3, 77., 56.);
mix1.PrintTable();
```

ScatteringProcess

Contains all scattering processes, more may be added easily

- Rutherford
- NucleonElastic
- NucleusElastic
- NucleonDiffractive
- Inelastic

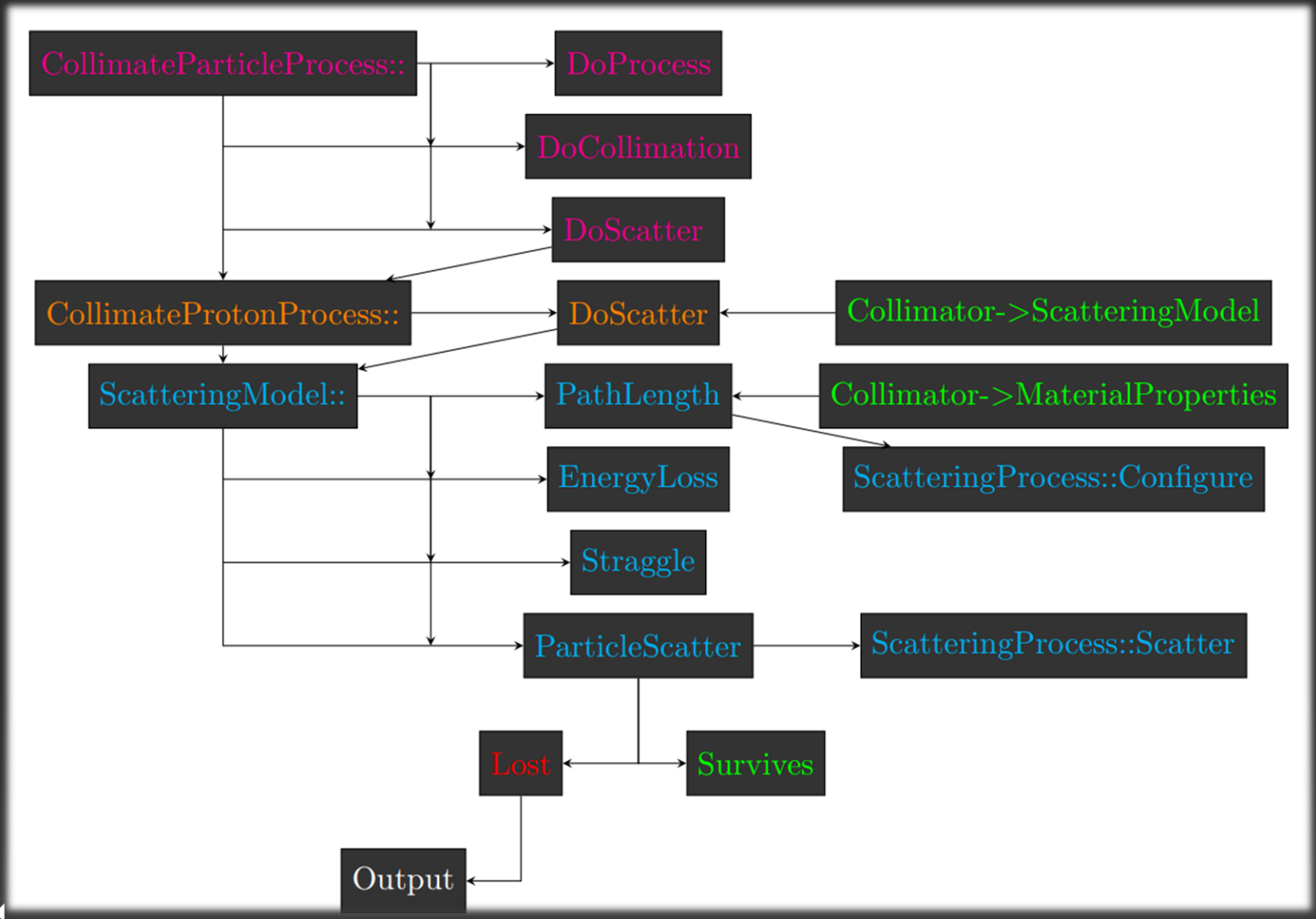
General code (5-01)uses simple scattering at the moment, ScatteringProcesses to be merged with loss map version of Merlin when new scattering is complete

ScatteringModel

Contains functions required for performing particle scattering

- PathLength – using mean free path (cross sections from user included ScatteringProcesses)
- EnergyLoss – energy loss via ionisation and atomic excitation
- Straggle – multiple coulomb scattering in position and angle
- ParticleScatter – randomly selects which ScatteringProcess to call
- DeathReport – called when particle lost, can be used for output etc

Collimation Process



User defined PhysicsProcess

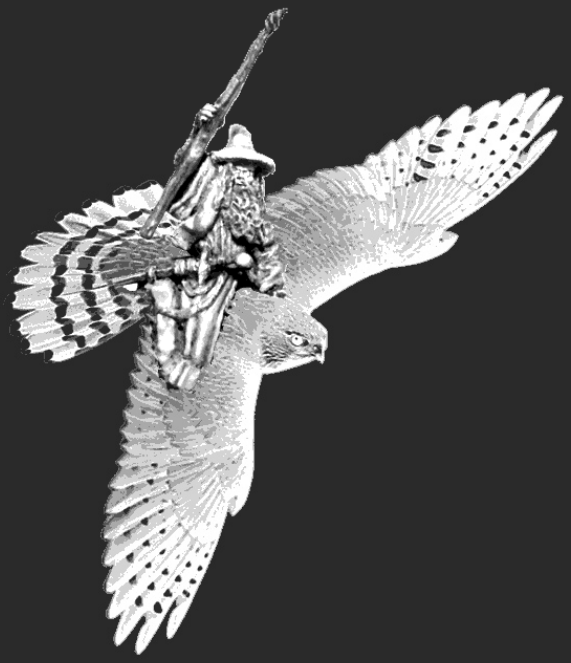
```
#include "BeamDynamics/ ParticleTracking / ParticleBunchProcess .h"
 namespace ParticleTracking {
class NewParticleProcess : public ParticleBunchProcess{
public :      //Constructor
              NewParticleProcess ( constructor arguments ) ;
               // Initialise process
              virtual void InitialiseProcess ( Bunch& bunch ) ;
              // Sets the current accelerator component
               virtual void SetCurrentComponent ( AcceleratorComponent& component ) ;
               // Returns the current maximum step length for this process
              virtual double GetMaxAllowedStepSize ( ) const ;
              // Perform the process for the specified step ds
              virtual void DoProcess ( double ds ) ;
private :
              // Data members for class attributes
              int alpha ;
              bool beta ;
              double gamma ;
};
```

Summary

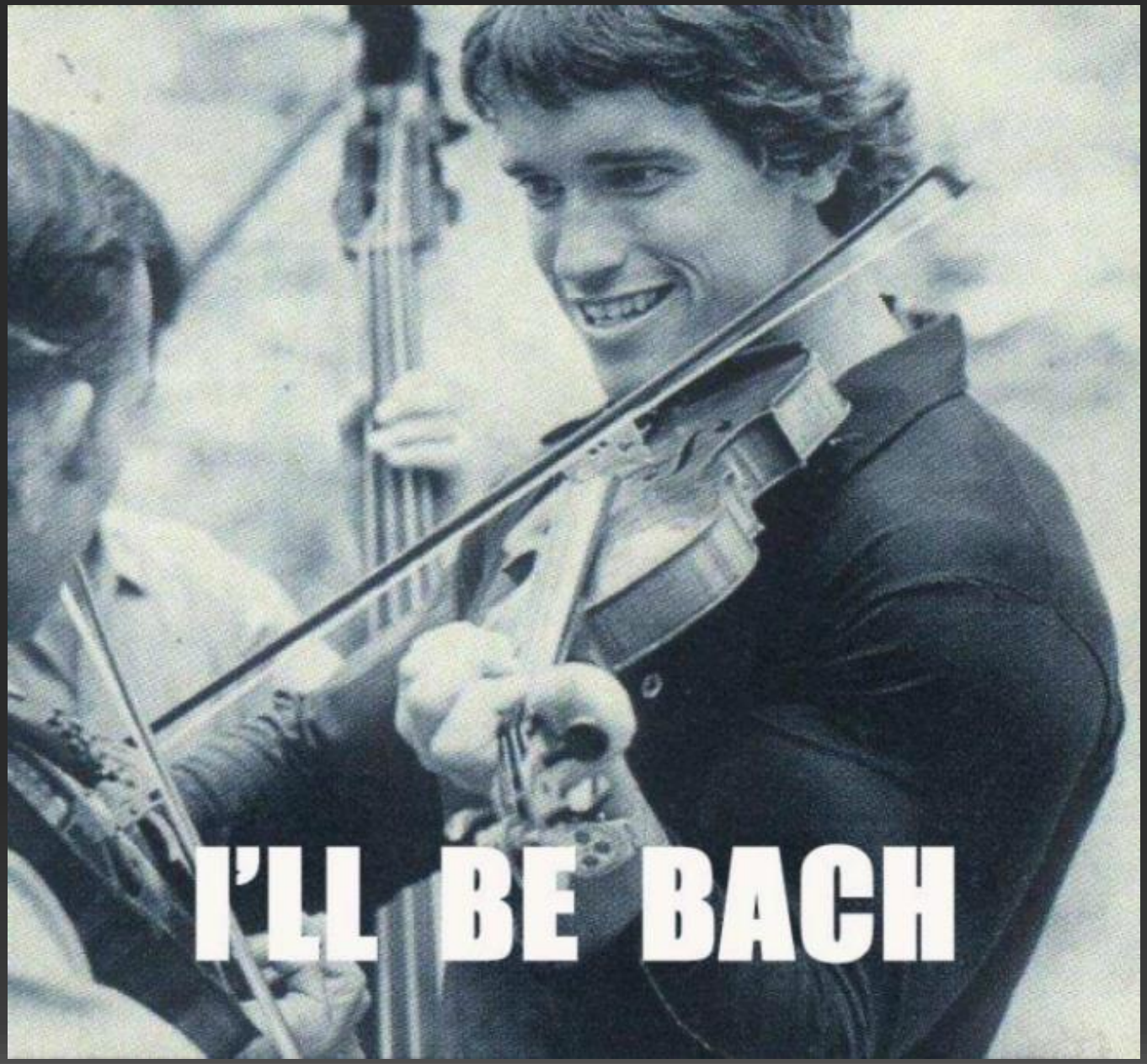
- Two strands of Merlin currently being developed (general & loss map versions)
- Short term: Release 5-01 (general), new scattering (loss map)
- Mid term: Merge both into release 5-02
- Long term: Investigate novel materials and collimation schemes etc

Merlin:

- C++ library
- Modular
- Easy to use
- Fast (using simple scattering models 6.4million particles for 200 turns in LHC collimation run takes ~ 1day on a 4core i5 3.2GHz desktop, 4GB RAM)



Backup Slides



Materials

- Homogeneous mixture
- Layers not present in code but easy to add
- For scattering: a weighted random nucleus is selected from the mixture
- More complex methods of material usage, e.g layers, may be added via:
  - new PhysicsProcess OR
  - a new class that inherits from MaterialProperties/Materials OR
  - a modification to the existing CollimateParticleProcess

- To maintain modular structure a new LayeredMaterial class (inherited from MaterialProperties/Materials) together with a new or modified CollimateParticleProcess would be preferred

Why 2 codes?

Original problem - take original Merlin and patch in specifics for LHC collimation

LHC collimation works – now optimise and maintain modular structure

Why 2 codes?

Original problem - take original Merlin and patch in specifics for LHC collimation

LHC collimation works – now optimise and maintain modular structure

Collimation (Loss Map) Merlin

Why 2 codes?

Original problem - take original Merlin and patch in specifics for LHC collimation

LHC collimation works – now optimise and maintain modular structure

Collimation (Loss Map) Merlin

Why 2 codes?

Original problem - take original Merlin and patch in specifics for LHC collimation

LHC collimation works – now optimise and maintain modular structure

C

Release 5-02