The ICEDUST Framework

Ajit Kurup

Workshop on Silicon Detectors for
g-2/EDM/COMET Experiments
21st February 2014

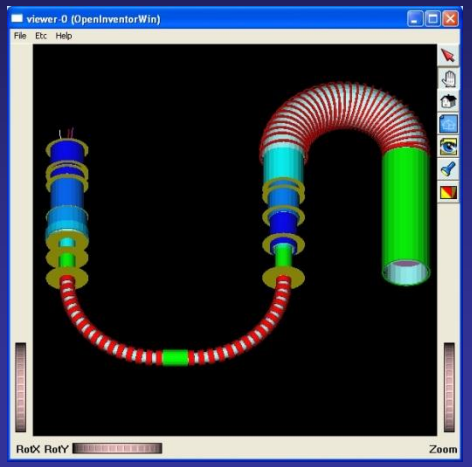
Imperial College
London

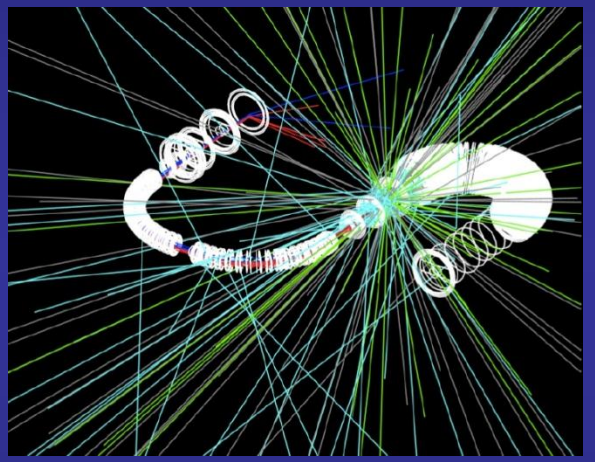
Introduction

- Brief history
  - comet_g4 and G4beamline.
  - ND280 framework.
- Ben Krikler's presentation at the IPNS technical review meeting.
  - Changes from ND280 framework.
  - ICEDUST details.
  - Software group organisation.
  - ICEDUST Development.
- Points to consider for g-2/EDM.

Brief History

- G4beamline simulation.





- comet_g4
  - Facility for complex detector simulation.
  - Includes G4beamline field models.
  - Specific physics model, $\mu \rightarrow e$, DIO, etc.
- Need for offline framework for simulation, reconstruction and analysis.
  - Definition of requirements document and evaluation of possible existing frameworks.
- Adoption of ND280 framework as basis for COMET.

Challenges for the Software

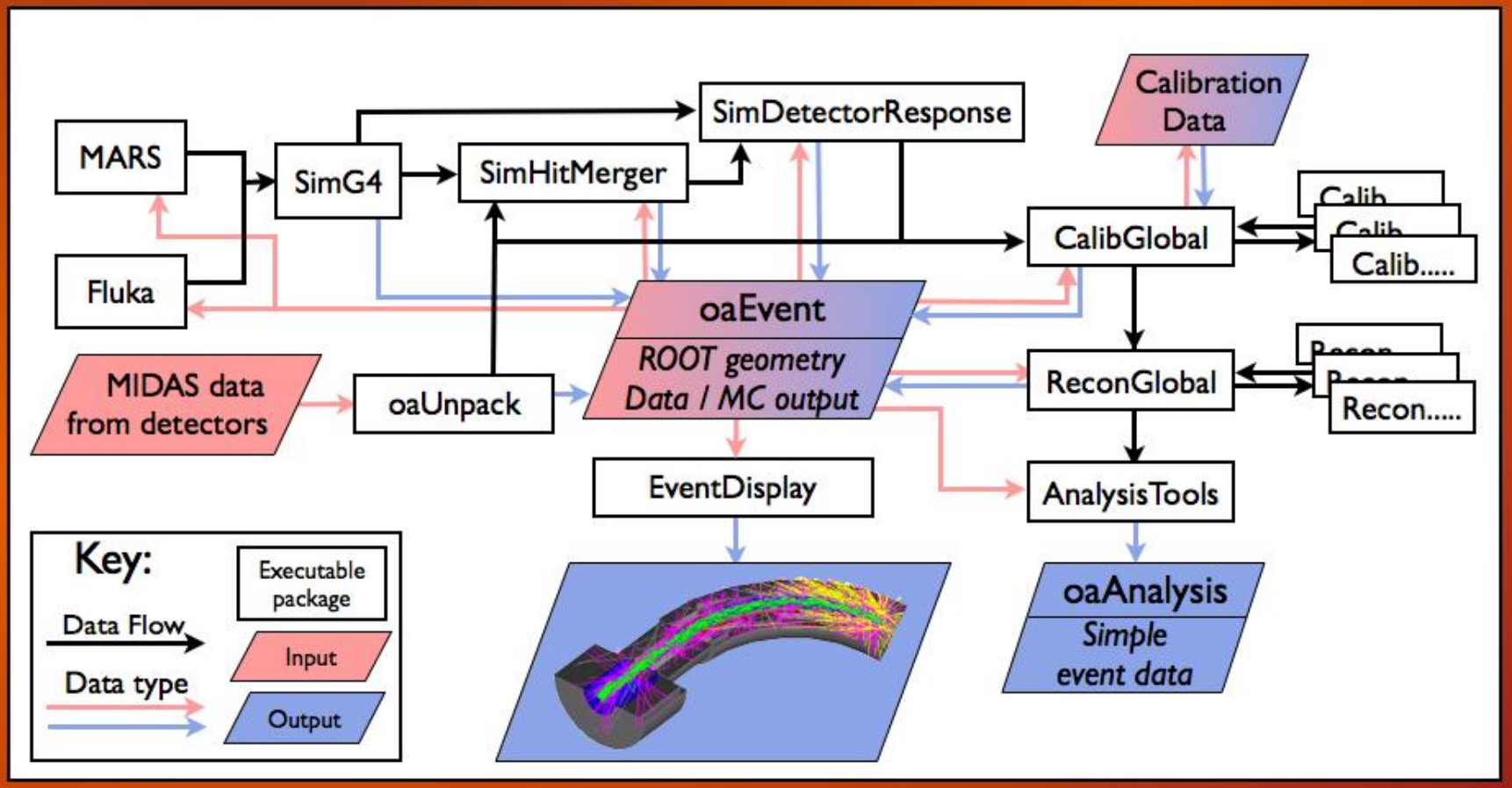
- Large data samples
  - Mostly background, small signal.
- High multiplicity events.
  - Challenges for track finding and fitting.
- Understand rates of background processes.
  - Decay in orbit.
  - Radiative capture.
  - Neutrons.
  - Cosmic rays.

Outline

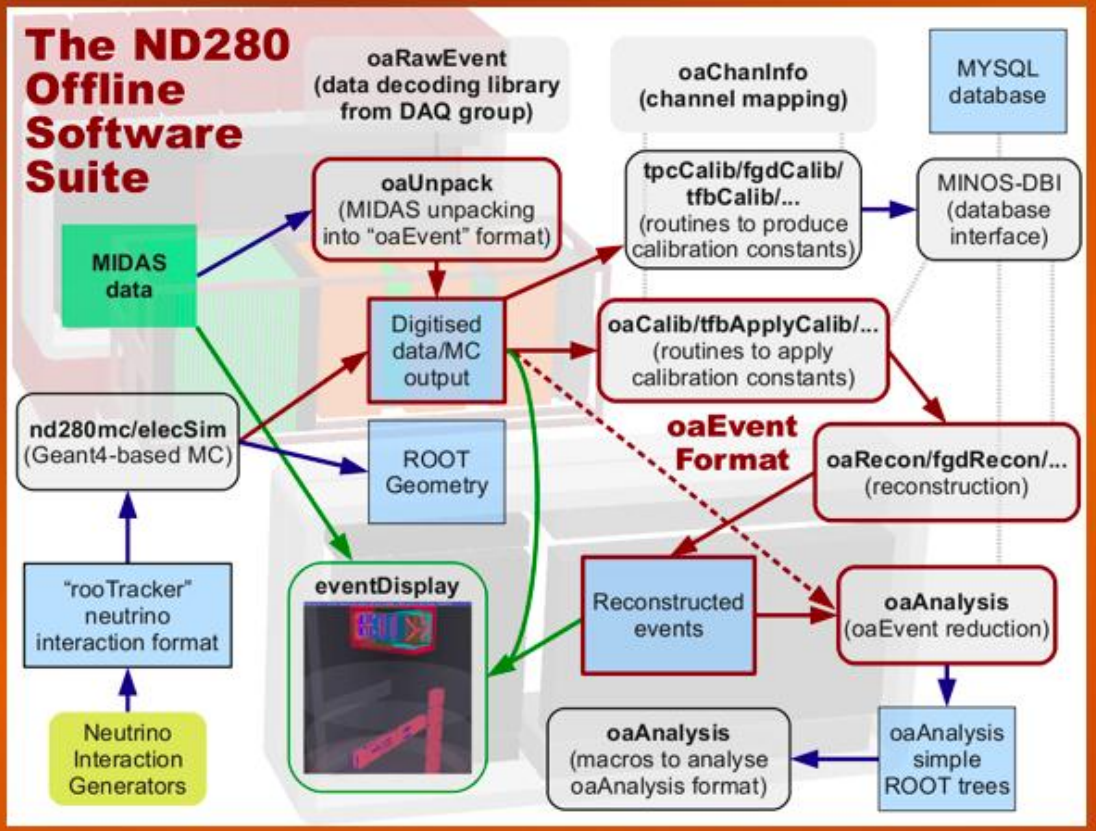
- Overview of framework
- Handling data
- Simulation
  - Geometry model
  - Hadron production models
- Reconstruction
- Analysis
- Specific COMET challenges
- Schedule
- Organisation
- Summary

What is ICEDUST?

- Integrated Comet Experiment Data User Software Toolkit
- Structure based on ND280 software framework



ND280 framework



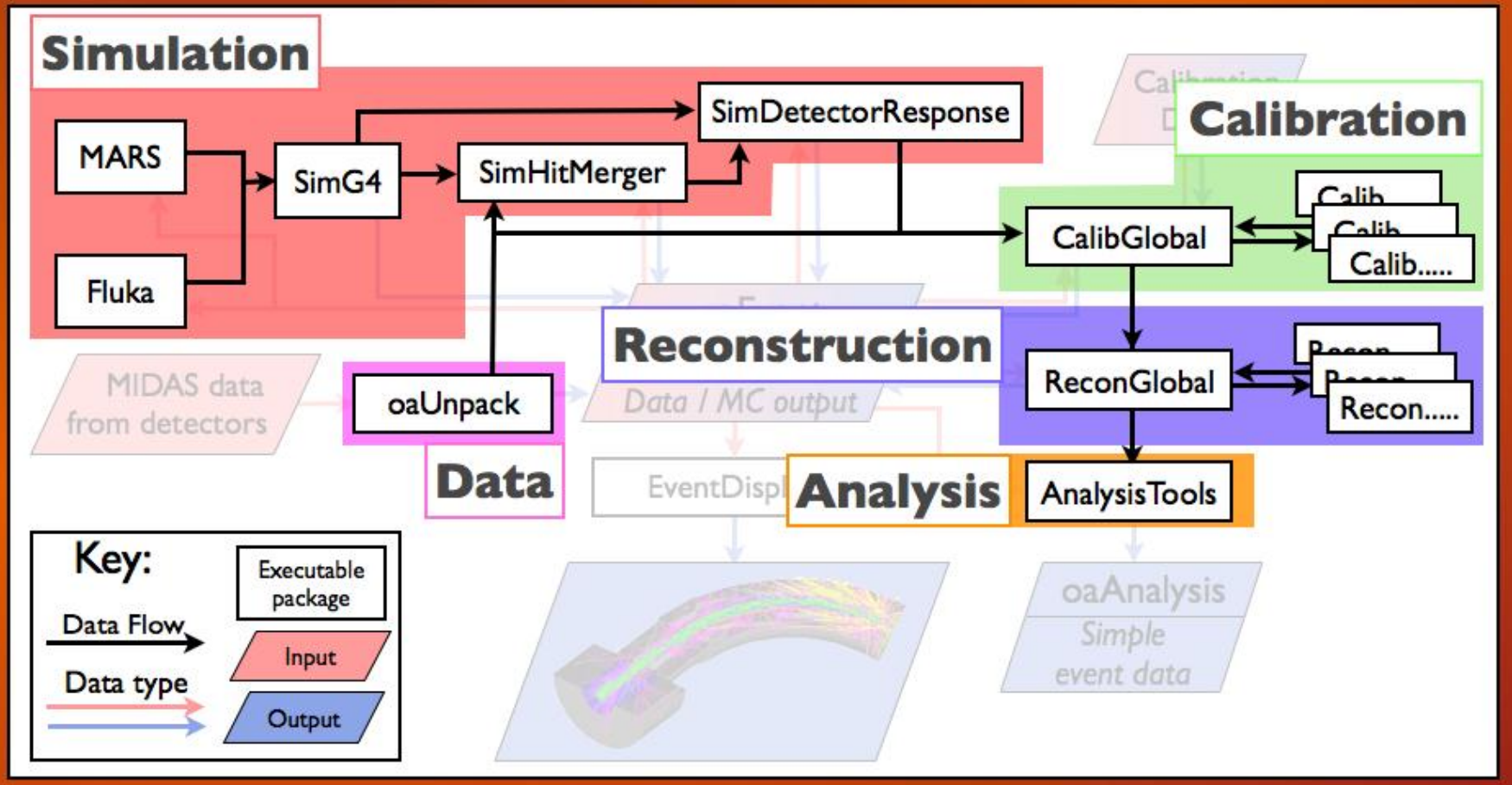
Superficial changes:

- Build systems
- Version control
- Coding conventions
- Package naming
- Supported systems

Contents changes:

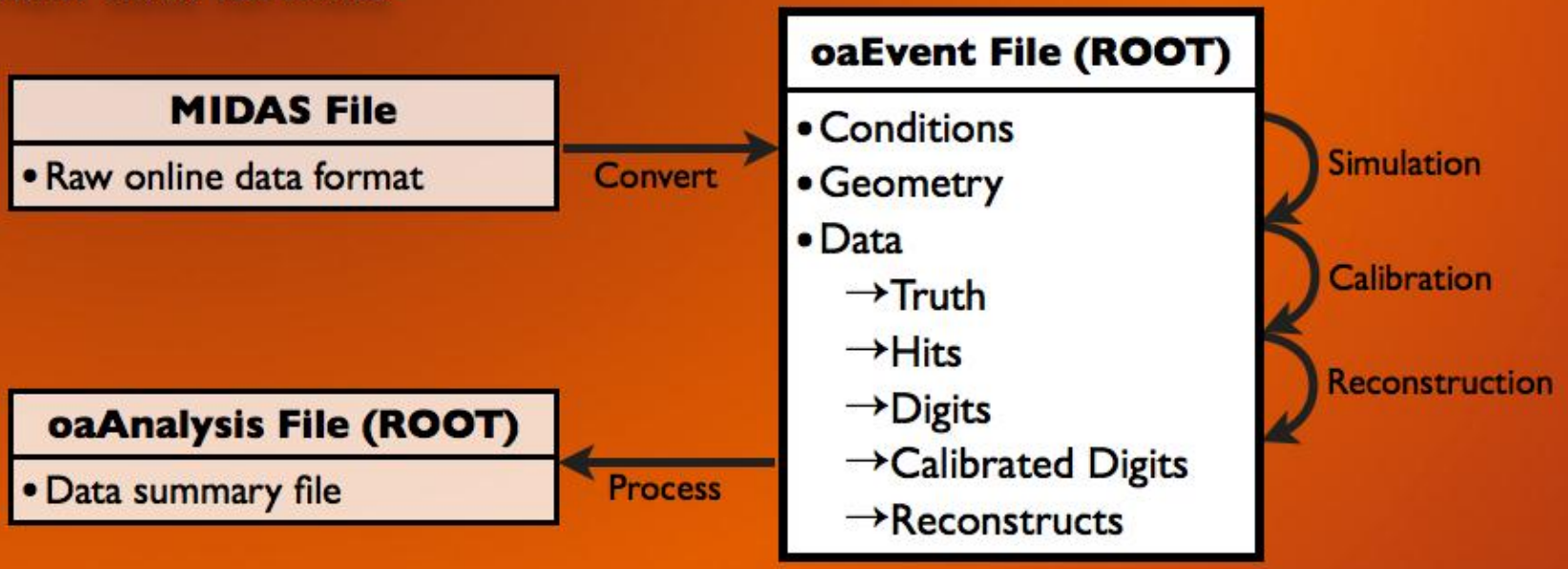
- Event generators
- Physics models
- SimHitMerger
- Reconstruction packages
- Geometry description
- Magnetic field description

What is ICEDUST?



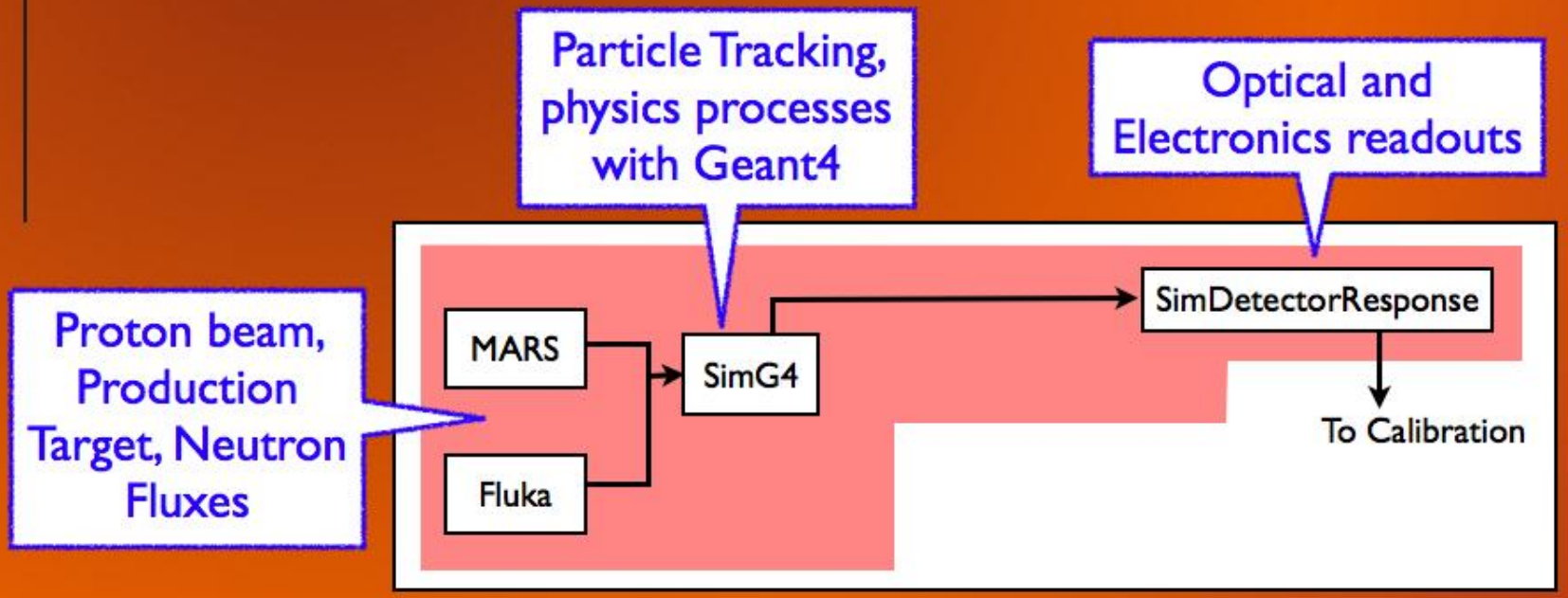
Data formats

3 main data formats

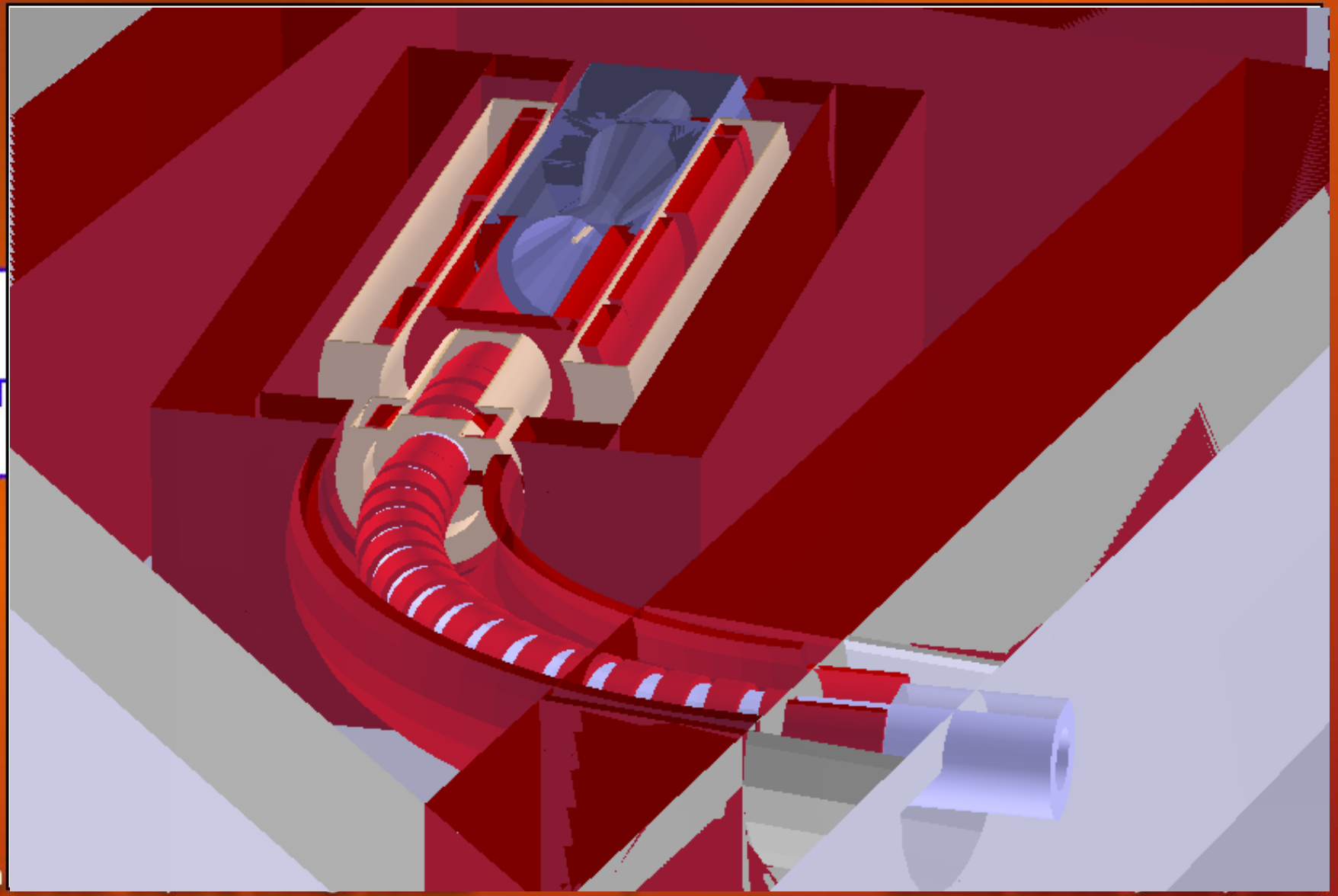


- oaEvent used by all packages within the framework
  - Physically meaningful representation of data / objects
- Data and MC indistinguishable from an early point
- MIDAS output data conversion maintained by DAQ group
  - Very elegant interface to offline software

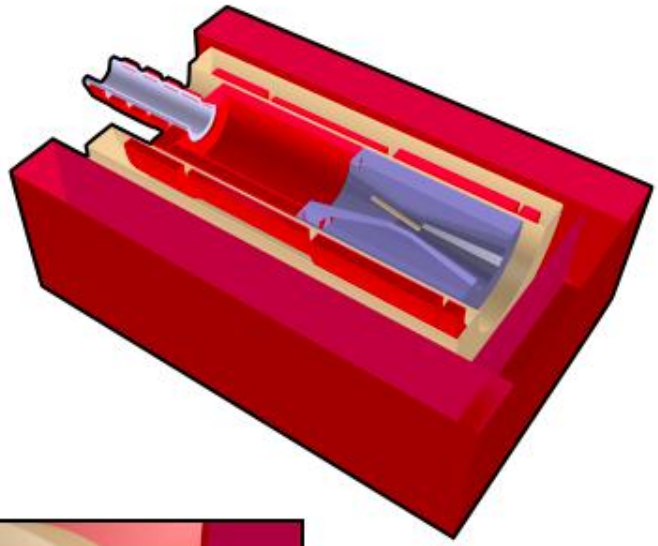
The Simulation

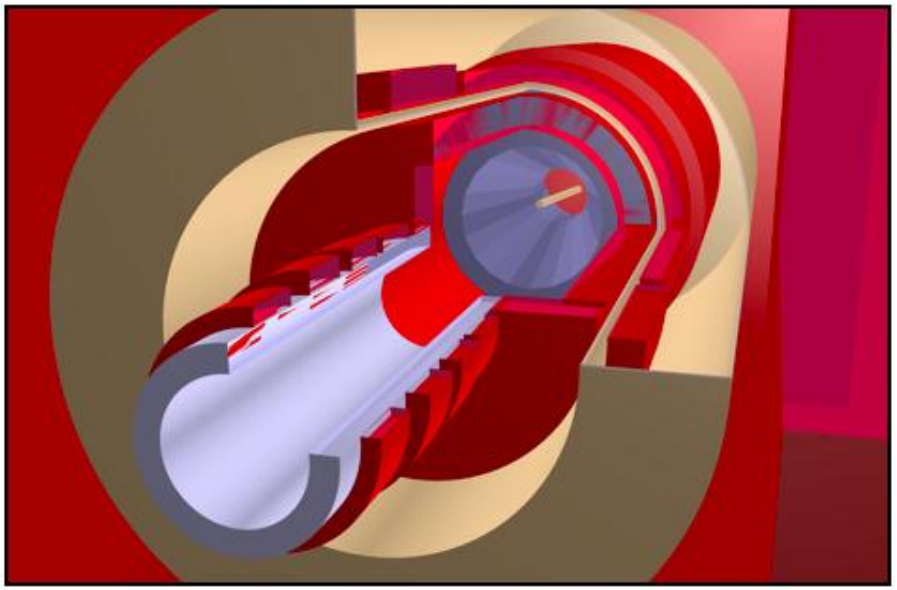


The Simulation



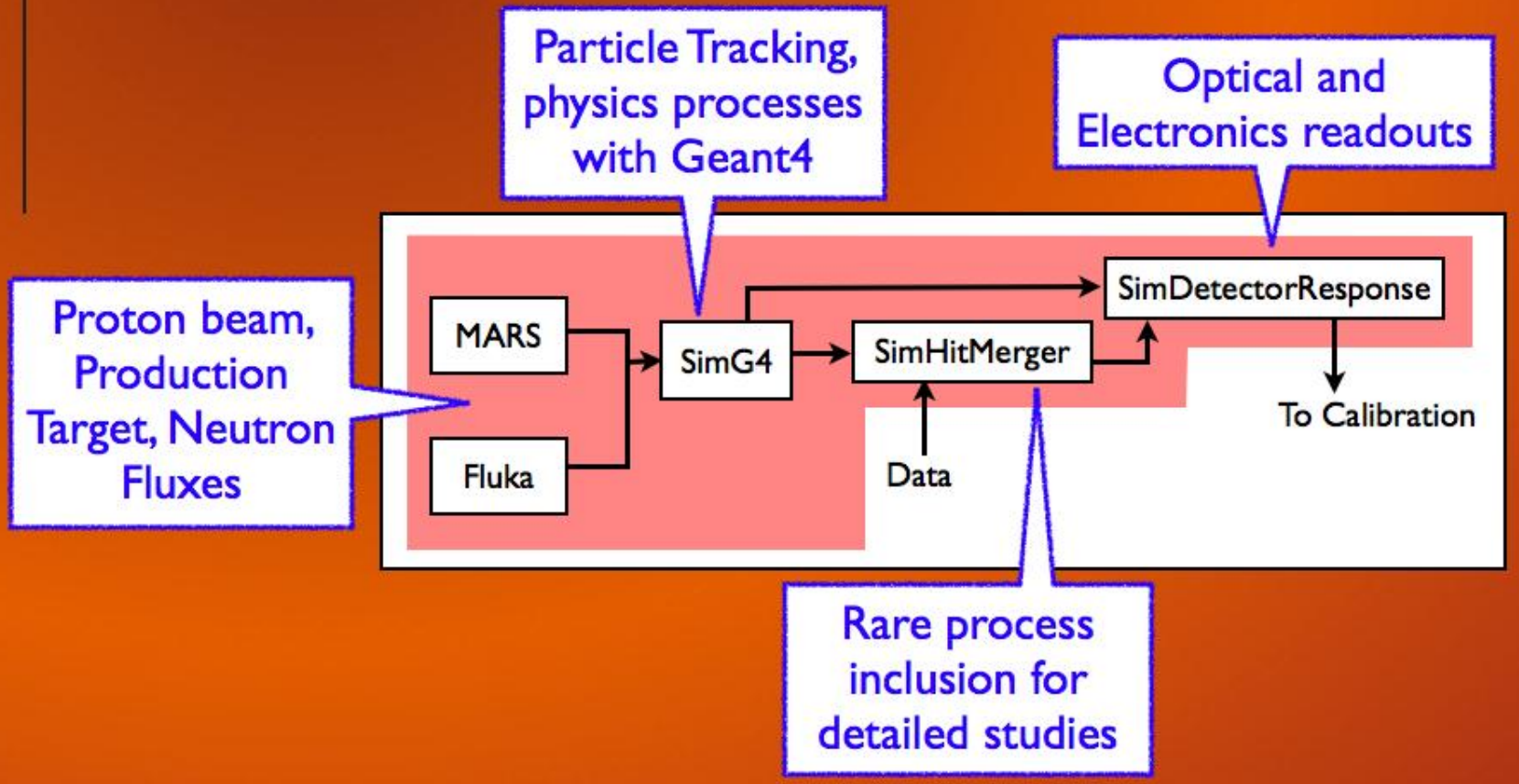
The Simulation





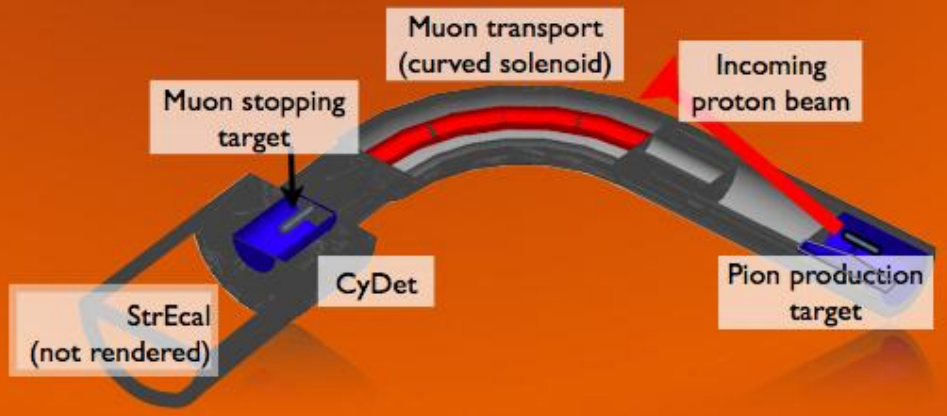
Ben

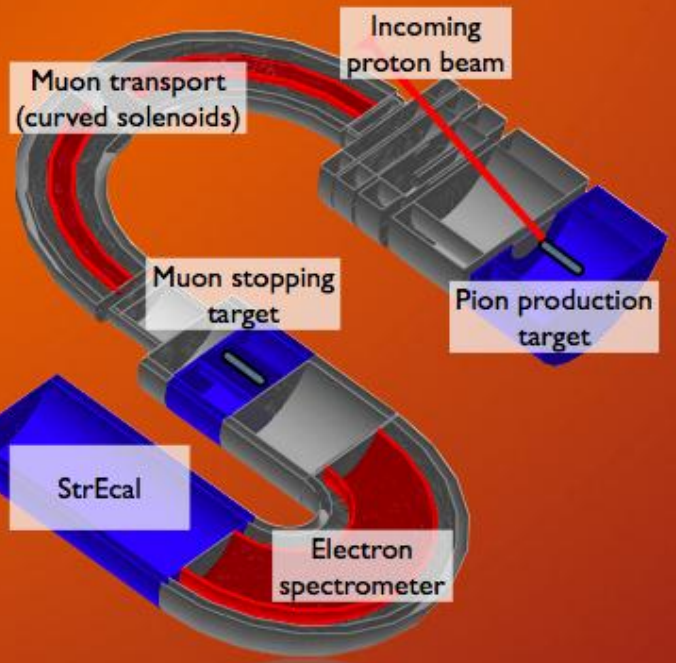
The Simulation



Geometry Handling

- Need high level of detail
  - Backgrounds from any material, eg. support structure, yokes, shielding
- Geometry changes from Phase-I to Phase-II

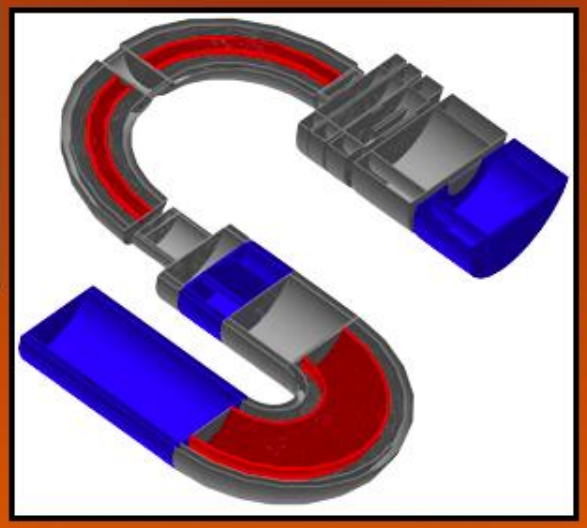




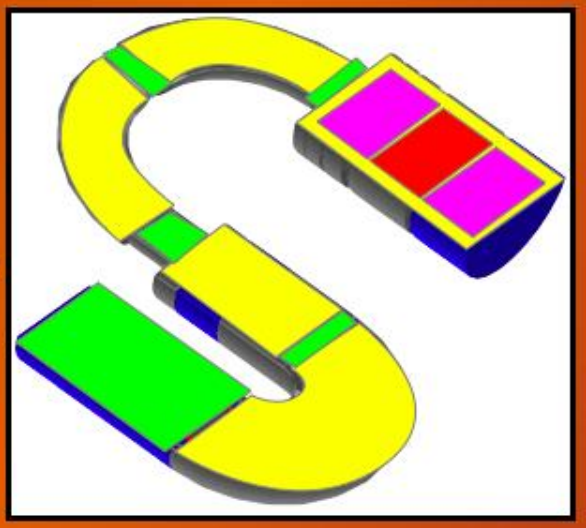
Also have:

- Cosmic ray veto
- Concrete Shielding
- X-ray Detector
- Beamline monitor
- Active Si-Target

Geometry Handling



Divide experiment into components



For each component write 3 classes:

Describes component layout — **Geometry**

Manages all parameters for a component — **Controller**

- Calculation of lengths etc
- Talk to other components

Handles run-time interaction with component — **Messenger**

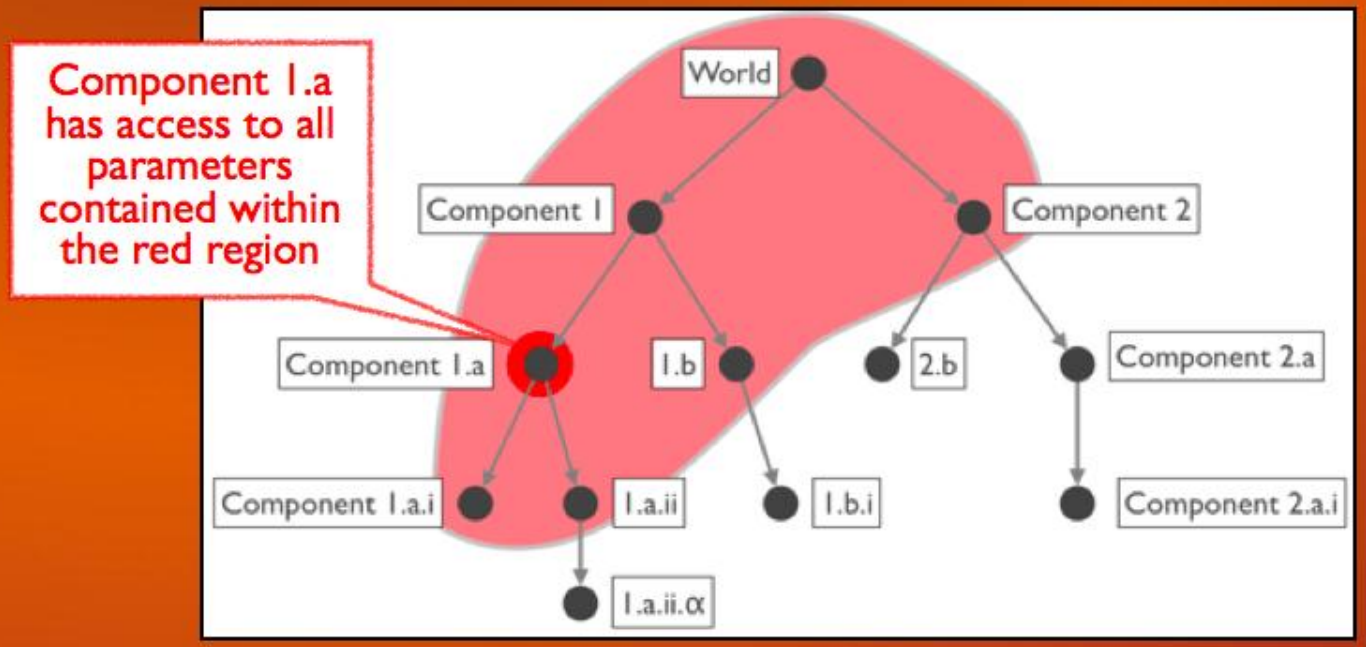
Component Communication

- Communication between components
  - All handled by controllers
- Restrict flow based on the component position in the geometry heirarchy

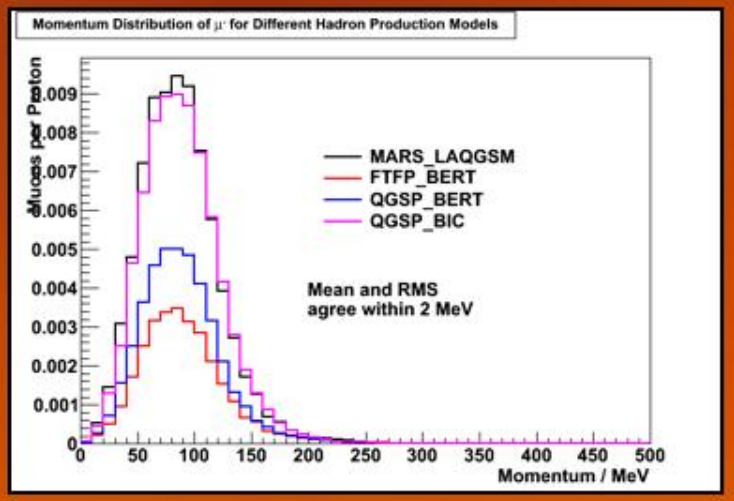
Describes component layout — **Geometry**

Manages all parameters for a component — **Controller**

Handles run-time interaction with component — **Messenger**

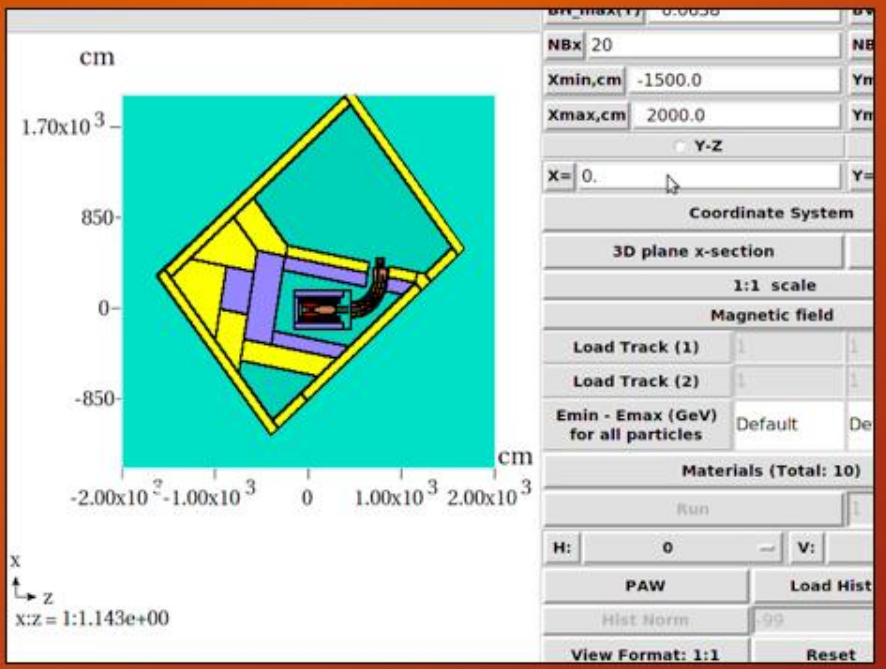


Hadron Production Models

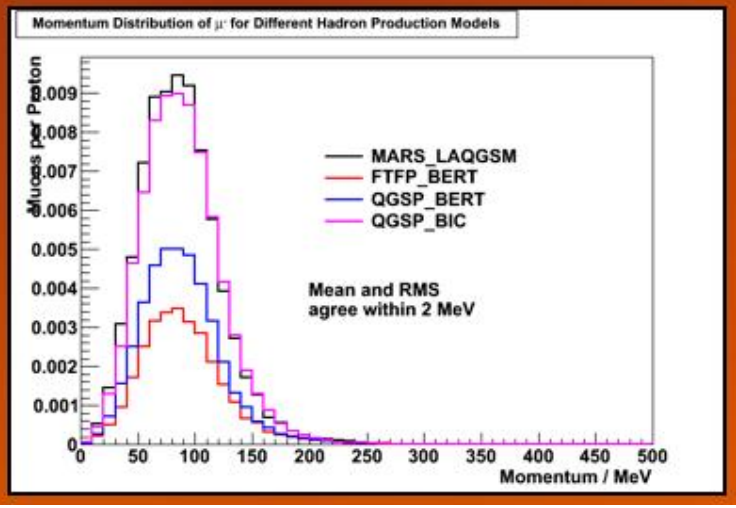


- Disagreement between physics models
- Must include and compare to external measurements

- MARS, FLUKA and Geant4 integrated into the software
- Use the same geometry as all other packages
- Flexibility within the software for other models



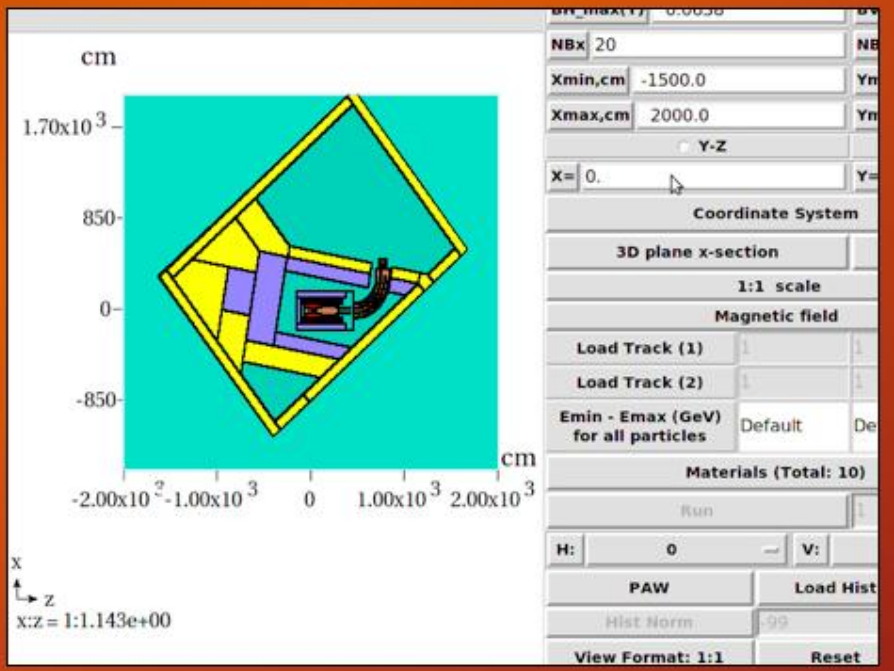
Hadron Production Models



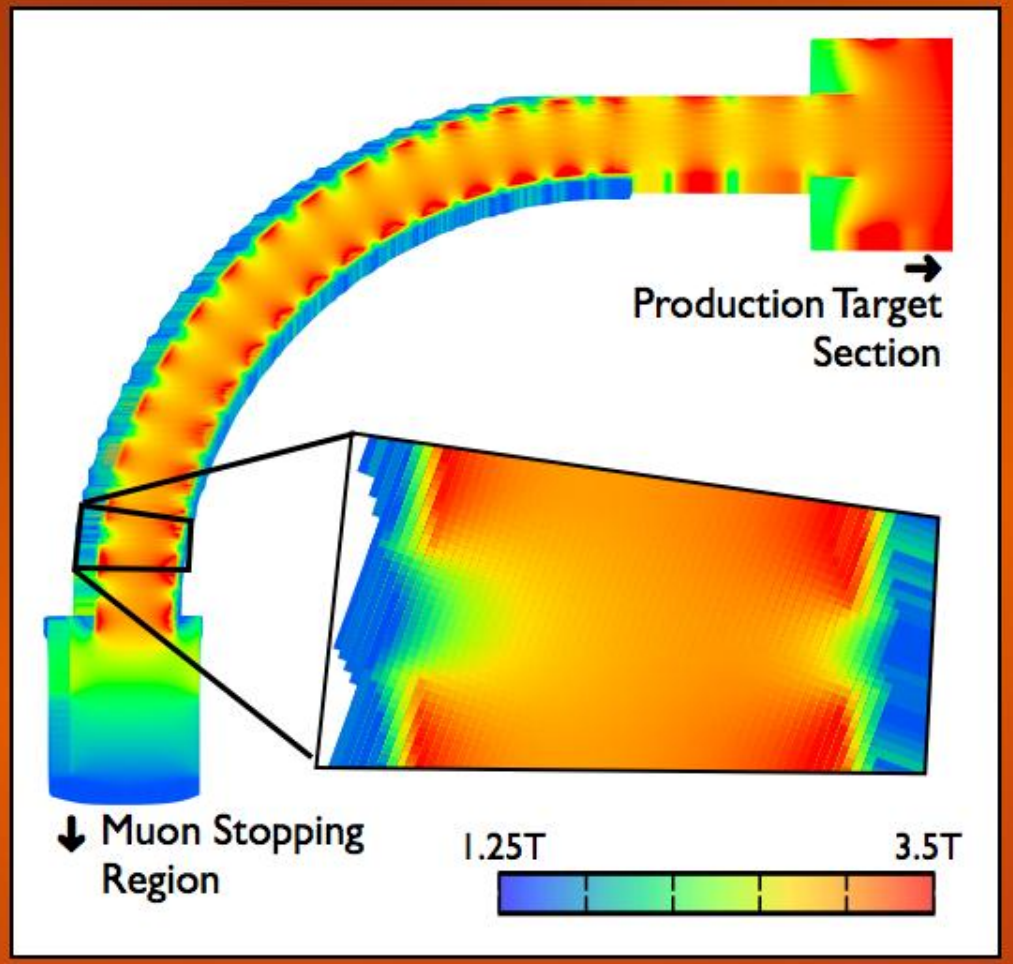
- Disagreement between physics models
- Must include and compare to external measurements

- Looking at integrating PHITS into ICEDUST.
- Yang Ye from Kyushu University looking a this.

- Flexibility within the software for other models



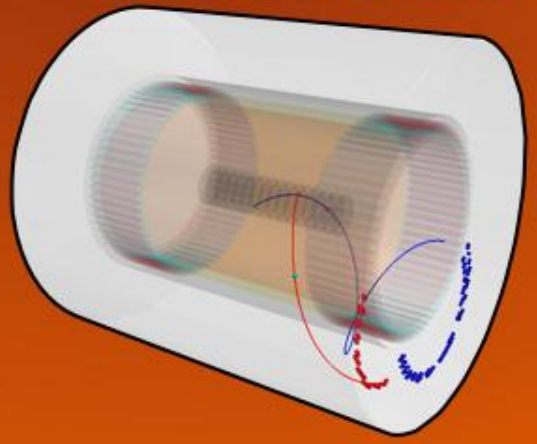
Magnetic Field



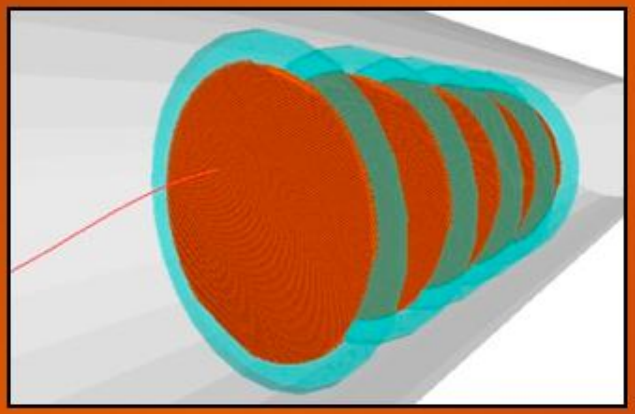
- Important for studying trapped particles
- Full fringe field simulation provided by Toshiba, including Iron yokes
- Can perform field calculations within SimG4
- Can overlap multiple fields
- Use a root format to share fieldmaps between packages

Reconstruction

Cylindrical Detector

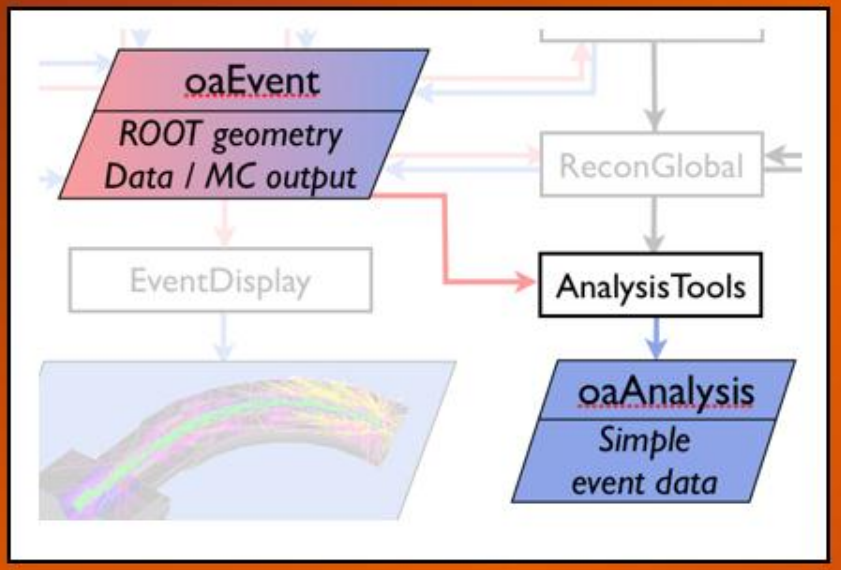


Straw Tracker



- Pileup and helical trajectories
  - Possibly want multi-turn fitting
- Use GENFIT with Runge-Kutta
- Flexibility for other packages
  - Will use several different routines / packages for each sub-detector
  - Persist physical quantities: implementation independent
- Track and cluster finding must be implemented

Analysis



- Beginning discussions
- Still defining desired approach
- Already have flexibility for many different techniques
  - Multiple independent methods ➜ avoids bias
  - Blind analysis looks favoured

Organisation

Ajit Kurup ***(Imperial College London, UK):*** Sub-group coordinator

Andy Edmonds
(University College London, UK)
MARS, SimG4

Yoshi Uchida
(Imperial College London, UK)
SimHitMerger, ND280 support

Chen Wu
(IHEP, China and Nanjing University, China)
Build system, repository, CyDet

Wilfrid da Silva, Frederic Kapusta
(Laboratory of Nuclear and High Energy Physics, France)
GENFIT, Active Target

Fedor Ignatov
(Budker Institute of Nuclear Physics, Russia)
Reconstruction, GENFIT

Vladimir Kalinnikov, Elena Velicheva ***(Joint Institute for Nuclear Research, Russia)***
ECAL

Sam Tygier
(University of Manchester, UK)
Fluka

Ben Krikler
(Imperial College London, UK)
SimG4, overall framework

Per Johnsson
(Imperial College London, UK)
Unit tests, ND280 support

Phill Litchfield
(University College London, UK)
Offline databases, ND280 support

Kazuki Ueno
(KEK, Japan)
Straw tracker

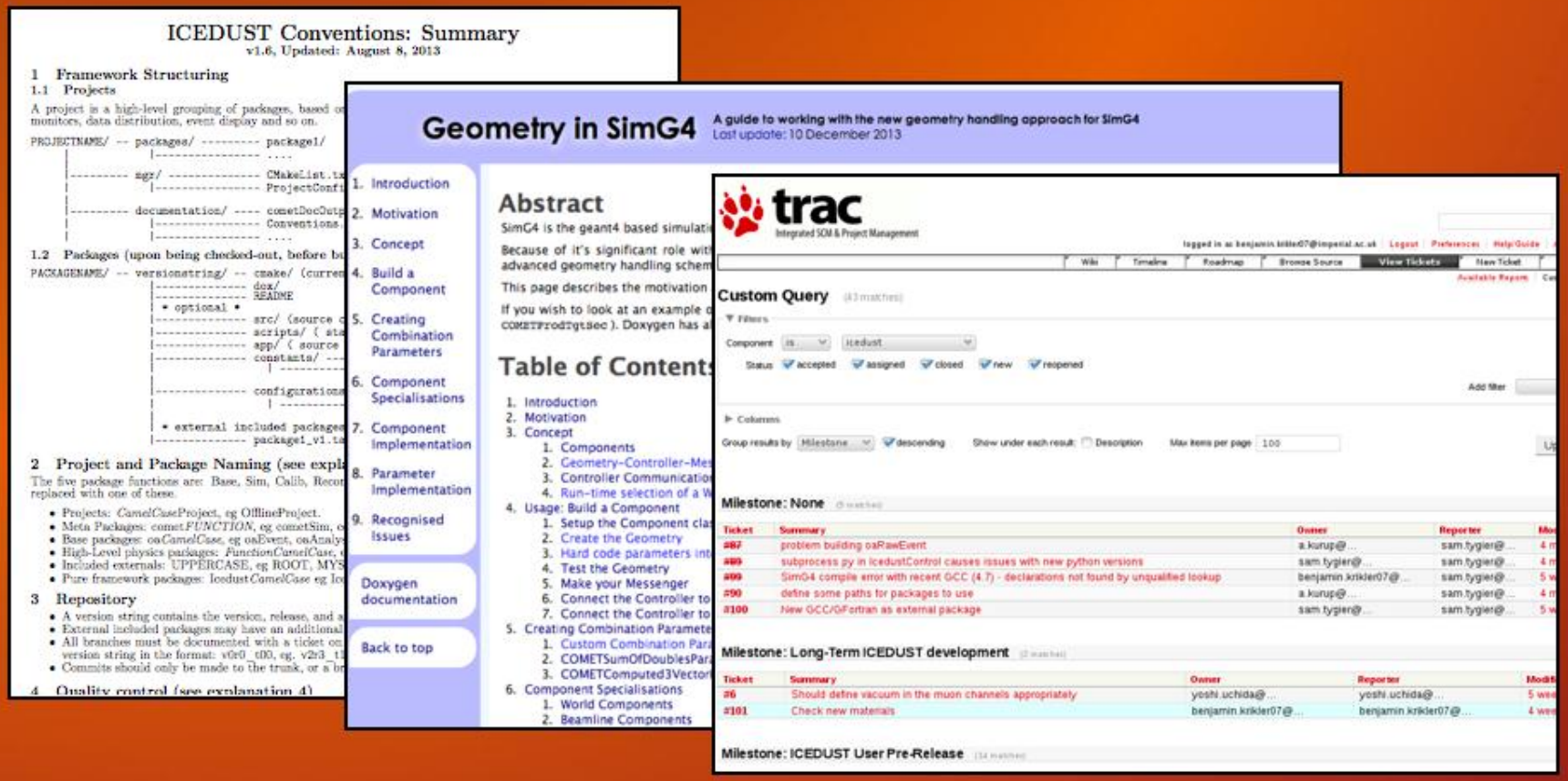
Schedule

- Four key releases planned:

Release 0 April '14	• Core framework development • Basic flow of data complete
Release 1 April '15	• Detailed Geometry in place • Online software interface review
Release 2 January '16	• Calibration and conditions prepared • Reconstruction optimised • Analysis code implemented
Release 3 October '16	• All round refinement and optimisation against external results • Interface to Accelerator DAQ • Data taking

Documentation

- Key documents
  - Supported systems
  - Conventions document
- User documentation
  - SimG4 documentation
  - Doxygen
- Trac Site



New Conventions

COMET-doc-36
Tickets #59, #84

ICEDUST Conventions: Summary

v1.6, Updated: August 8, 2013

1 Framework Structuring

1.1 Projects

A project is a high-level grouping of packages, based on the packages' purpose, such as offline software, online monitors, data distribution, event display and so on.

```
PROJECTNAME/ -- packages/ ---------- package1/
                  |----------------- ....
                |--------- mgr/ ---------- CMakeList.txt
                  |----------------- ProjectConfig.cfg
                |
                |--------- documentation/ ---- cometDocOutput/
                  |----------------- Conventions.pdf
                  |----------------- ....
```

1.2 Packages (upon being checked-out, before building)

```
PACKAGENAME/ -- versionstring/ -- cmake/ (currently cmt)
                  |-------------- doc/
                  |-------------- README
                  | * optional *
                  |-------------- src/ (source code, header & implementation files )
                  |-------------- scripts/ ( stand alone utility scripts )
                  |-------------- app/ ( source for executables )
                  |-------------- constants/ ----- calibration_tables/
                  |                  | ------------ particle_distributions/
                  |
                  |-------------- configurations/ -- parameters.txt
                  |                  | -------------- run.cfg
                  |
                  | * external included packages contain a source tar file *
                  |-------------- package1_v1.tar.gz
```

2 Project and Package Naming (see explanation 2)

The five package functions are: Base, Sim, Calib, Recon, Analysis. *FUNCTION* in the names below should be replaced with one of these.

- Projects: *CamelCase*Project, eg OfflineProject.
- Meta Packages: comet*FUNCTION*, eg cometSim, cometBase, cometAnalysis
- Base packages: oa*CamelCase*, eg oaEvent, oaAnalysis
- High-Level physics packages: *FunctionCamelCase*, eg AnalysisTools, SimG4, CalibGlobal
- Included externals: UPPERCASE, eg ROOT, MYSQL
- Pure framework packages: Icedust*CamelCase* eg IcedustPolicy, IcedustDoc

3 Repository

- A version string contains the version, release, and an optional patch number, eg v2r1 v2r1p2.
- External included packages may have an additional number to signify local patches, eg v5r34p01n02.
- All branches must be documented with a ticket on the Trac site, whose number should be appended to the version string in the format: v0r0_t00, eg. v2r3_t127.
- Commits should only be made to the trunk, or a branch of a package or project, and not a tag.

4 Quality control (see explanation 4)

- New packages and projects must be accepted by the software group before being added to the repository.
- New package names must be based on these guidelines.

5 Geometry Names and Numbers

- When code refers to a component of the geometry, it must use the relevant naming convention.
- Numbering of repeated components must agree with the numbering used in the real, physical system (eg. ECal crystals, Drift Chamber wires).

6 Filenames and Extensions (see explanation 6)

- Filenames should match contained classes (or the main, public class)
- Implementation files' names should only differ from the header file by their extension.
- File extensions :

	C	C++
Header	*filename*.h	*filename*.hxx
Implementation	*filename*.c	*filename*.cxx

	Python	Perl
Module	*filename*.py	*filename*.pl
Script	*filename*	*filename*
	filename.py	*filename*.pl

7 Scripting Conventions (see explanation 7)

For all scripts who are directly executable:

python	#!	/usr/bin/env python2
shell	#!	/bin/bash
perl	#!	/usr/bin/env perl

8 Executable Naming

- For executables providing standard functionality of a package, the name should be of the form *FunctionPackage* eg. RunSimG4, RunCalibApply, TestOAEvent, TestReconGlobal.
- The current list of these standard executable functionalities is: Run, Test, Validate.
- All other executables should use the package name followed by an underscore as a prefix: oaEvent_dumpevent.

9 C++ Coding Conventions (see explanation 9)

9.1 Class Conventions

	Prefix	Example	Comments
Standard classes	I	IExampleClass	
Abstract classes	IV	IVExampleBaseClass	
Exception classes	E	EExampleException	Must inherit (indirectly) from std::exception
Class data members	f	fMax, fDataPoint	Avoid public access, use private or protected
Class Methods	Capital	GetMax(), TwoFlushDump()	
Destructors			Always set as virtual
Sub-Class names		IMainClass::SubClass	If behaviour mimics an STL type, then name similarly eg. IMainClass::iterator

9.2 Namespaces

- Put all classes in the COMET:: namespace, although sub-namespaces can be used, eg: COMET::ECAL::.
- 'using namespace' directives must not be included in header files

9.3 Free Functions, Variables and Arguments

All functions, variables and arguments that are not part of a class (ie. free) must NOT use the class conventions, and should instead follow these:

	Prefix	Example	Comments
Functions	Lower case	come_fly_with_me()	underscores for long names
Variables & Parameters	Lower case	myWay	CamelCase for long names
Global Parameters	g		Avoid at all costs!!
enums	k	kSomeEnum	CamelCase for long names

9.4 Include Guards (see explanation 9.4)

- All header files must contain an include guard, to prevent multiple inclusion of the same file.
- All include guards should take the form: `[PACKAGE]_[FILENAME]`.
- Any non-alphanumeric characters in the filename should be replaced with underscores.
- For example, for IMCHit.hxx in oaEvent, use: `OAEVENT_IMCHIT_HXX`

9.5 Output and Error Messages

- Functions from ICOMETLog.hxx should be used instead of C++ STL output (std::cout) and error (std::cerr).
- This provides standard ways for dealing with verbosity and format.

9.6 Pointer/Memory Handling (see explanation 9.6)

- Bare pointers should almost never be returned.
- Use a IHandle instead (see below).

10 Documentation

- Each packages should contain a basic README file in it's top layer, describing it's purpose, the contents of its folders and where to find documentation.
- Package histories should be documented in doxygen markup in the `dox` directory of that package.
- All header files should be fully commented in doxygen markup.
- The package IcedustDoc is used to produce documentation from the source code. Its output can either be stored in the top-level documentation directory, or alongside the individual packages.
- All stand-alone documentation must have a version number and the date last updated just below the title.

11 Units and Constants

- HEPUnits.hxx defines the standard set of units and all derived ones and should be used for all output data.
- HEPConstants.hxx defines a set of useful constants, such as pi, the speed of light, standard temperature etc.
- Both these files are in the oaEvent package and all units are contained in the 'unit' namespace.
- The standard set of units is:

millimeter (mm)	positron charge (eplus)	luminous intensity (candela)
nanosecond (ns)	degree Kelvin (kelvin)	radian (rad)
Mega electron Volt (MeV)	the amount of substance (mole)	steradian (steradian)

New Conventions

COMET-doc-36
Tickets #59, #84

1 Framework Structuring

1.1 Projects

A project is a high-level grouping of packages, based on the packages' purpose, such as offline software, online monitors, data distribution, event display and so on.

```
PROJECTNAME/ -- packages/ --------- package1/
     |             |---------------- ....
     |
     |--------- mgr/ -------------- CMakeList.txt
     |             |---------------- ProjectConfig.cfg
     |
     |--------- documentation/ ---- cometDocOutput/
     |             |---------------- Conventions.pdf
     |             |---------------- ....
```

Projects are new to ICEDUST (not in nd280)

1.2 Packages (upon being checked-out, before building)

```
PACKAGENAME/ -- versionstring/ -- cmake/ (currently cmt)
                  |-------------- dox/
                  |-------------- README
                  | * optional *
                  |-------------- src/ (source code, header & implementation files )
                  |-------------- scripts/ ( stand alone utility scripts )
                  |-------------- app/ ( source for executables )
                  |-------------- constants/ ----- calibration_tables/
                  |                  | ------------- particle_distributions/
                  |
                  |-------------- configurations/ -- parameters.txt
                  |                  | ------------- run.cfg
                  |
                  | * external included packages contain a source tar file *
                  |-------------- package1_v1.tar.gz
```

Standardise package contents

Supported Systems

Ticket #61

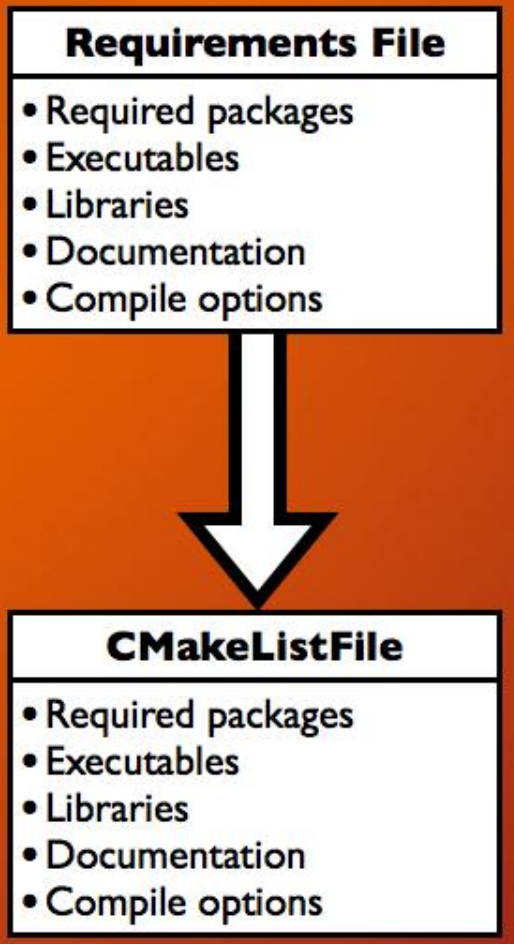
- Defines the framework's dependency on external libraries.
- Stated aims
  - To run on a range of operating systems
  - To provide external packages
  - To keep reasonably up to date with external dependencies
  - To use free and open source external packages as much as possible
- We target primarily Linux, but avoid decisions that rule out other systems.
- Desired minimum system requirements (based on RedHat 6)
  - BASH 4.2
  - GCC 4.4
  - Python 2.6
  - Perl 5.10
  - CMake 2.6
- Aware of other potential externals:
  - xerces-c (for xml)
  - Qt (graphics toolkit)
  - Coin3D (3D toolkit)
  - Dawn and dawn_cut (visualisation)
  - GenFit ?
- Some 'external' packages already in framework:
  - CLHEP
  - ROOT
  - GEANT4
  - MYSQL
  - RECPACK

Building the Framework

Tickets #57

- Currently uses CMT
- Requirements file to specify package dependencies
- Can build entire project + package dependency tree or just one package

- Switch to CMake
  - Faster
  - Better documentation
  - Wider support
- Requirements files need translating
- Help from Gaudi who are also switching



ICEDUST Development

- Other important items:
  - Clean-up of ND280 and other legacy code.
  - Look at updating to latest CLHEP, Geant4 and ROOT versions.
  - Implement GENFIT.
  - Implement SimHitMerger.
  - Storage of field maps in ROOT file.
  - Testing ICEDUST on the Grid.
  - Look at analysis methods including unbiased methods, e.g. blinding.
    - Flexibility exists but need to consider specific methods.
  - Audit trails for simulation, reconstruction and analysis.

Points to Consider for g-2/EDM

- High multiplicity track finding and fitting.
- Detector optimisation.
  - Requires consistent geometry in simulation and reconstruction.
- Time-varying EM-fields, G4beamline functionality available.
  - Field non-uniformity, phasing studies.
- Simulation of Si response and read-out electronics.
- Need for MARS and FLUKA?
  - Neutrons from beam line, other background processes.
- Spin tracking in Geant4.
- Interface to online/DAQ system.

Some Questions and Areas for Collaboration

- What is the overall timescale?
- What technical effort/resources are available (in particular to address points below)?
- Areas for collaboration:
  - Implementation of GENFIT.
  - Grid support, MC generation, either in France or Japan.
  - Longer term items
    - Database management.
    - Interface to accelerator

Issues

- Need to remove ND280 specific code.
- COMET has limited effort, cannot guarantee significant support.
- Need to agree how to distribute and maintain code.
- Need agreement at the collaboration level.

Summary

- ICEDUST is flexible and can easily be used for other experiments.
  - Due to structure inherited from ND280.
- Clean-up needs to be done before code can be distributed.
- Need to consider areas for collaborative development of ICEDUST, timescales and effort available.