Workshop on Muon at the LHC and Tevatron

FNAL, thursday, April 15, 2004

Muon Reconstruction with ORCA

ORCA tutorial

Stefano Lacaprara

Stefano.Lacaprara@pd.infn.it

INFN and Padova University

Outline

- General introduction,
  - Goal of this tutorial,
  - Muon related packages,
  - Information: who and where,
  - Muon BuildFile,
  - `.orcarc` and co,
  - Basics on CommonDet,
- Exercises: access to Geometry, SimHits, Digis and RecHits
  - access DT geometry information,
  - access RPC SimHits,
  - access CSC Digis,
  - access RecHits for all 3 systems,
  - homework,

Introduction: Goals

- Goals of this tutorial
  - general description of Muon software
  - description of Muon detector modeling
  - how to access geometry of DT, CSC and RPC
  - how to access SimHits, Digis and Rechits (local reconstruction) for the 3 systems
- This tutorial will not cover:
  - How to access and use muon reconstructed tracks, DST (Norbert's tutorial)
  - Algorithmic details about how the reconstruction is performed
  - L1 simulation and HLT reconstruction

Muon packages

There are 3 packages related to Muon reconstruction in ORCA:

- Muon
  - Responsible for detector description of DT, CSC and RPC (reading from DDD), OO modeling of detectors
  - access to SimHits, simulation and access to Digis
  - local reconstruction (hits and segments)
  - stand alone reconstruction, using only muon detectors
  - (mis)alignment

  The Muon package will be soon (next release?) split in two

  Muon geometry, SimHits, Digis

  MuonLocalReco RecHits (reconstruction inside detectors) and stand alone reconstruction

Muon packages (II)

- MuonReco
  - responsible for global reconstruction (with tracker)
  - HLT reconstruction (L2, L3), using L1 trigger input
  - muon isolation (using calo's, pixel and tracker)
  - Off-line muon reconstruction and isolation
  - Muon identification
- MuonAnalysis
  - Framework for analysis for the PRS-μ group
  - Used to produce root trees for the DAQ TDR
  - code now obsoleted by DST

Documentation

Where to find it:

- A good reference book on **C++** (like Stroustrup's)
- For **STL**: `http://www.sgi.com/stl/`
- **ORCA** `http://cmsdoc.cern.ch/orca/`
- ORCA UserGuide: Muon section
  - large effort to write and update the DT part
  - for CSC CMS Note 2001/013
- ORCA Reference Manual: for class interface and documentation about class methods
- for local reconstruction CMS Note 2002/043 (DT part obsolete since december...)
- for track reconstruction DAQ TDR vol II (mostly HLT, but useful)

People

Who can answer to your questions

- Stefano Lacaprara all Muon, DT, reconstruction;
- Nicola Amapane DT, isolation;
- Tim Cox, Rick Wilkinson CSC;
- Artur Kalinosky, Giacomo Bruno RPC;
- Norbert Neumeister global muon reconstruction;
- Celso-Martinez Rivero (mis)alignment;

Don’t forget the general ORCA mailing list:
`cms-orca-feedback@cern.ch`

For bug report and feature request use **savannah**:
`https://savannah.cern.ch/projects/orca/`

BuildFile

What to put into your BuildFile

- The content of your BuildFile determines what you can actually do in your executable.
- Documented from ORCA page (follow BuildFile link in the upper part)
- General rules:
  - keep it as simple as possible (but not simpler...)
  - use only **groups**, not directly library (except your own): if they do not work, complain with author (eg via `feedback` mailing list)
  - find a working example and copy it!

BuildFile (II)

Simple example: private analysis library and executable which uses just muon from stand alone reconstruction starting from Digi

```
<environment>
  <lib name=MyGreatAnalysisLibrary></lib>

  <group name=MuonInternalReco>
  <use Muon>

  <group name=RecReader>
  <External ref=COBRA Use=CARF>

  <bin file=MyAnalysis.cpp name=MyAnalysis></bi
</environment>
```

BuildFile: Muon groups (I)

with `<Use Muon>`

MuonRecHitReader : For RecHit reconstruction on DT, CSC and RPC;

MuonDigiReader : For reading on the 3 sub-detectors: no RecHit reconstruction is guaranteed

MuonSimHitReader : For SimHit reading on the 3 sub-detectors: no RecHit reconstruction and/or Digi simulation is guaranteed

MuonLayout : To access DetLayers (see after) for the Muon system [to be used with `<use CommonReco>` :( ]

⋆ Identical groups exist separately for each sub-system, replacing `Muon` with `MB, ME, MRpc`, respectively for DT (barrel), CSC (endcap) and RPC systems;

BuildFile: Muon groups (II)

MuonDSTReading : For reading (and writing) muon track reconstruction with standalone detector from DST;

MuonInternalReco : For muon track reconstruction with standalone detector;

with `<Use MuonReco>`

MuonReconstruction : For global reconstruction (with tracker);

MuonIsolation : For isolation;

MuonRecoReader : to read reconstructed muon (even with isolation info) from DST;

⋆ **Warning** ⋆ : to perform off-line reconstruction (`GlobalMuonReconstructor`), also MuonInternalReco needed.
No problem to **read** muons from DST!

.orcarc and co

What to put into your .orcarc

- The .orcarc files contains parameter and setting for ORCA software
- Can have different name. Usage MyExacutable -c <MyOrcarcFile>
- Documented from ORCA page (follow **orcarc** link in the upper part)
- keep it as simple as possible: the default is (almost) always the best choice, unless you know what you are doing
- Mandatory:
  - InputFileCatalogURL = @{<your PoolFileCatalog>}@
  - InputCollections = /System/<Owner>/<Dataset>
  - MaxEvents = ... Default (-1) == all events

.orcarc and co (II)

- To switch off completely access to all Digis, SimHits but the Muon ones

```
FrontEnd:DefaultRequest = Nop
MuonHits*:Request = Auto
MuonDigi*:Request = Auto
```

- Similars card for all other detector

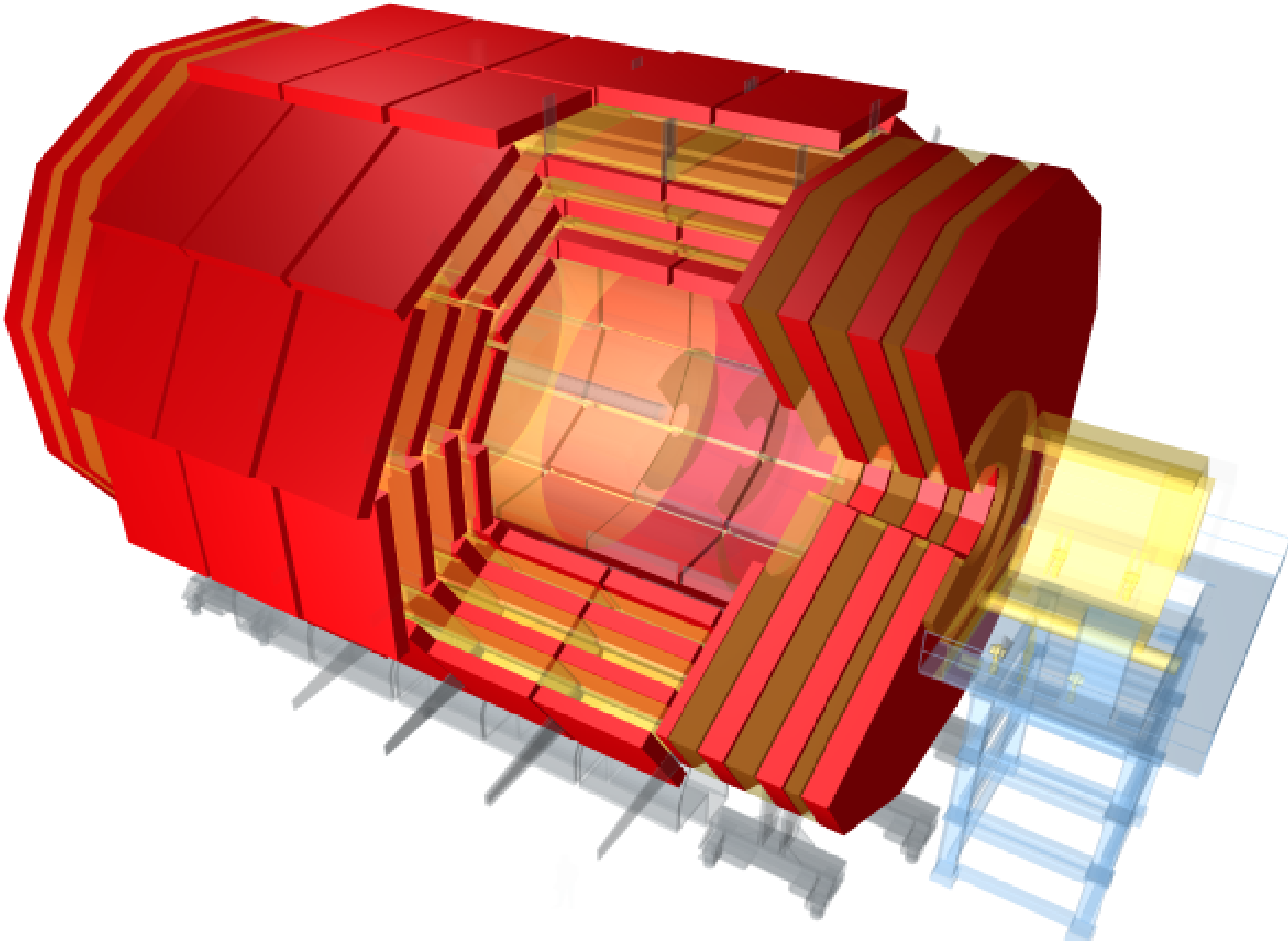
IMPORTANT!! To perform Muon track reconstruction we still use GEANE for extrapolation: need initialization, via environment variable

```
setenv GEANEUSED TRUE (csh)
export GEANEUSED=TRUE (sh)
```
ALL CAPITAL!!

If not, no reconstructed muons!

Warning: If you just read from DST reconstructed muon track you DON'T need it

Geometry in ORCA



CommonDet

- CommonDet is a base ORCA package to describe geometry and detectors
- It widely used by Muon and Tracker sub-systems
- Defines common interfaces and implement functionalities which are not system dependent
- Muon and Tracker have specialization of base CommonDet classes, to deals with details (such as geometry structure, hierarchy, reconstruction algorithms) which are different
- The common interface allow for a transparent use of Muon and Tracker system by user
- Great use in muon reconstruction but not only
- Base object in CommonDet are `Det`, `DetUnit`, `RecHit` and related classes

Det

- a `Det` model a “Detector” in ORCA software
- What is a “Detector”?
  - For Tracker: a silicon or pixel model (not a single strip or pixel)
  - For Muon: a Chamber (DT, CSC, RPC)
  - DT and CSC are multi layers: also the individual layers are Det (also the SuperLayer for DT)
  - This leads naturally for a CompositeDet, with a hierarchial structure (see after)
  - In general: a Det models a well defined physical object, with reconstruction capabilities, but not the smallest possible one (eg silicon strip or DT cell)

Responsibility of a `Det`

The responsibility defines also the interface

- Det deals with reconstruction
- Has knowledge of it's position within CMS reference frame, its size, defines a local reference frame and is able to transform back and forth from local to global ref. frame
- Reconstruct the detector response in term of position (and eventually direction), in case using external input (such as track angle) and return the results as RecHits
- Det is not responsible to access the Digis and/or SimHits (DetUnit see after)
- Part of Det responsibility is delegated to specialized classes: eg position to `Surface` and bounds to `BoundSurface`, which are hidden behind Det interface
- Specialization of Det of Muon system are: `MuBarChamber, SL, Layer, MuEndChamber, Layer, MRpcDetUnit`

Det interface

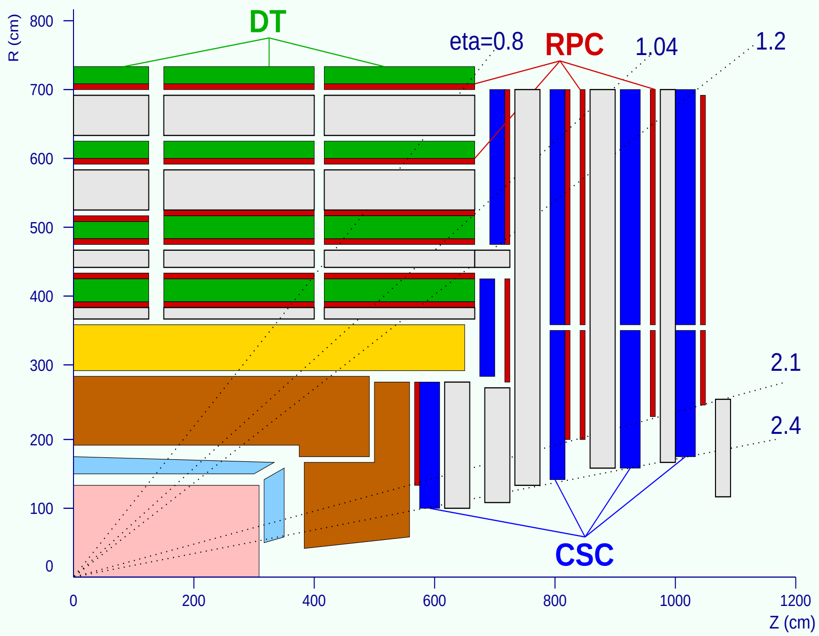
- `position(), rotation()` of Det surface in CMS reference frame
- `toLocal(...), toGlobal(...)` transforms a point, vector, error from global (CMS) to local (Det) ref. frame. `toLocal` wants a Global<Object> and returns a Local<Object>, and vice-versa.
- `recHits(), recHits(TSOS)` return all RecHits reconstructed by the Det, eventually using the input TSOS.
- `measurements(...)` return all the measurements compatible with an input state according to a user definable criteria (χ^2, distance, etc).
- `DetUnits()` return all DetUnits (see after) the Det is composed of, or itself if not composed
- Other methods to implement the ones above in a detector specific way

CompositeDet

- A Det can be composite, ie made of other Dets
- Good examples are DT chambers (each made of 3 SuperLayers, each made of 4 Layers) or CSC (each made of 6 Layers)
- Very useful to define logical collection of physical detectors with common features, which is useful to handle together (DetLayer)
- `CompositeDet` is a Det made of Dets
- `CompositeDet::dets()` returns the components as `vector<Det*>`.
- all other Det methods are available, including recHits(), measurements(),...

DetLayer

- Very important CompositeDet is a `DetLayer`
- DetLayer is a logical CompositeDet, made of all detector of a given type which lays “more or less” on the same surface
- The surface can be a cylinder (barrel) or a disk (endcap)



- Muon barrel: 4 DetLayers corresponding to the 4 DT stations, plus 6 for RPCs, all cylinders
- Muon endcap: 5 disks ($\times 2$) for CSC and 5 for RPC (neglecting staging ...)
- The physical dets does not lie necessarily exactly on the surface, but at least very close to

DetLayer (II)

- Responsibility of DetLayer is navigation
- Given this state, which is the next DetLayer where I should look to find other RecHits?
- Other responsability is to find efficiently which are all the components Det which are truly compatible with the propagated state
- The set of DetLayers give a simplified description of CMS often called the **reconstruction geometry**
- Important by-product of DetLayer is the ability to access to the Det geometry in a uniform way for all the sub-systems, via `DetLayer::dets()`
  - Caveat: for Muon need group MuonLayout in BuildFile
  - Warning: the Dets are accessed as `Det`, not as specific object (eg `MuBarChamber`, ...). Not full interface is available (can `dynamic_cast`)

DetUnit

- Specialization of Det
- Has a pointer to `SimDet` where SimHit (hits simulated by OSCAR/cmsim) are stored: can be void (eg after 2007...)
- Has pointer to `Readout`, which acts as a Digi container, thus gives access to Digis
- The digis are detector specific, so it's not possible to access to a generic `vector<Digis>` from a DetUnit: must use the specific interface of concrete implementation of DetUnit for various systems (DT: MuBarLayer, CSC: MuEndLayer, RPC:MRpcDetUnit)

Geometry building

- Who builds the Dets? NOT YOU!!
- There must be only one instance of the Geometry in memory, and the user just access it
- The geometry is built when the proper library is loaded
- The ORCA geometry is focused on reconstruction, and is designed to be as simple as possible
- It is different from the detector description used in OSCAR (simulation), which describes as many detail as possible
- eg. in ORCA only sensitive detectors are described, not passive material
- Different applications have different needs, thus “different” geometries
- Both geometries come from the same database!!
- Database is a set of `xml` files, almost human readable, and ORCA is interfaced to xml via **D**etector **D**escription **D**atabase package

Local reconstruction: `RecHit`

- A `Det` is a reconstructing detector
- the local reconstructed object is always returned as a `RecHit`
- a `RecHit` can be different things:

Hit 1D only one coordinate is measured (in the local Det ref. frame). ex. Drift Tube alone (only distance from wire)

Hit 2D two coordinates are measured. ex. pixel detectors or CSC Layers if wires and strips information are matched

a Hit pair ex. Drift Tube without Left/Right ambiguity solved

Segment 2D position and direction in one coordinate. ex a DT SuperLayer

Segment 4D position and direction in both coordinates. ex a DT chamber segment (matching ϕ and θ SL segments), a CSC chamber segment.

Local reconstruction: `RecHit` (II)

- The RecHit is always defined at the Det surface (so local **z** is always 0)
- RecHit interface is wide enough to return information for the most complex object.
- Not all methods are meaningful for all kind of RecHits: eg no direction for a 1D hit!
- A RecHit is a composed object: can be made of other RecHits (eg a segments or a pair)
- When needed (eg for Kalman filter), all available information about a RecHit are used properly, with a common interface
- RecHit always accessed **by value**! Always do a local copy of `vector<RecHit>` before iterating

RecHit and tracks

- RecHits are used to build Tracks (see also Norbert's tutorial)
- A track has a collection of `TrajectoryMeasurement`
- Each TM has:

predictedState() according to "last" measurement and propagated to the actual Det surface,

recHit() the RecHit found and used to update the trajectory: **can be invalid if no compatible RecHit has been found**. Always check if `RecHit::isValid()`

updatedState() the starting point for next iteration.

RecHit and tracks (II)

The pattern recognition in CommonDet framework

(bird's eye view: reality can be much more complex)

- Define a starting state (seed) on a Det: eg. using the Det RecHits (as in StandAlone muon reco, starts from DT/CSC segments) or external input (L1 trigger)
- Ask to the DetLayer (to who the actual Det belong) which is next DetLayer compatible with actual state
- propagate state to DL surface, "long" propagation, in Muon system likely across iron: CPU expensive!
- Find inside the next DetLayer the Dets compatible with the state (reminder: a DetLayer can be a cylinder $\phi \in [0, 2\pi]$ many possible Det, mostly wrong ones!)
- Propagate from DL surface to compatible Det surfaces (if not the same), short propagation
- find best RecHit in the Dets, check if good enough, and update the state with RecHit information
- restart the procedure with the updated state

ORCA Exercises

Exercise

To run the exercise (ie the solution) just do:

```
wget http://www.pd.infn.it/~lacaprar/Tutorial/go
chmod +x goMuon
./goMuon
```

or `goMuon --loc CERN` at CERN or `goMuon --loc PD` in Padova.
For code with just comments and hints (to be completed by you), go to `cd ORCA_8_0_1/src/MuonAnalysis/MuonTutorial/src/` and complete `MuonTutorial.cc_ex`. Then copy it to `.cc`, compile the library `scram b`, compile the executable `scram b` (from test) and run it.

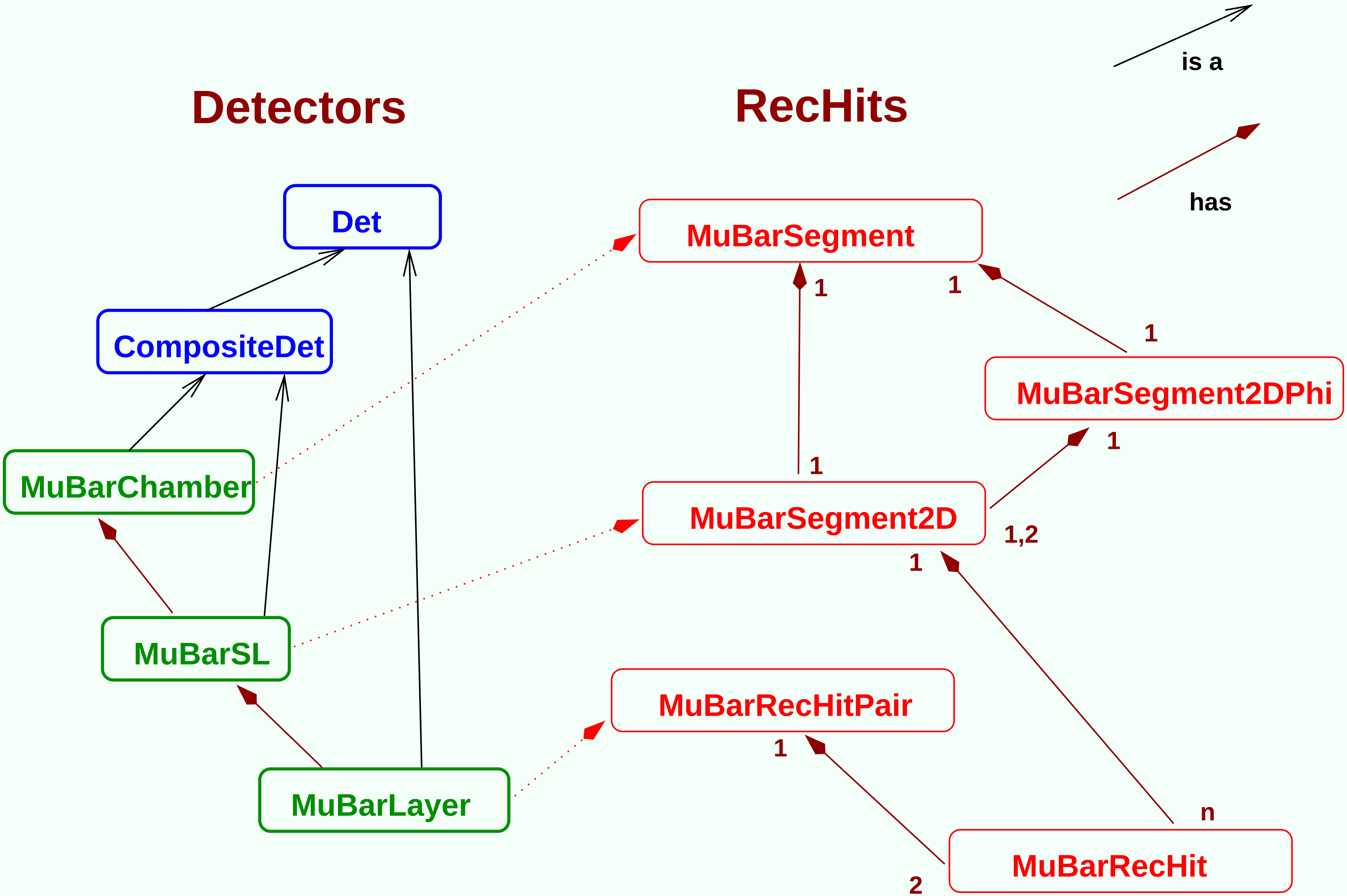
Ex 1: DT geometry

Goal Get number of wires for each DT chamber in central wheel

Quick description of DT geometry:

- CMSMuonBarrel: the whole system
- has: 4 `DetLayers`, cylinders, aka stations, used for navigation during track finding
- MuBarChamber: access to 4D segments (RecHits)
- each chamber has 3 (2) MuBarSL: access to 2D segments (RecHits)
- each SL has 4 MuBarLayer: access to SimHits, Digis, Hits (RecHit)
- each Layer has $\mathcal{O}(100)$ MuBarWire
- Possible to navigate from top to bottom in different ways

DT geometry (II)



DT geometry (III)

template in src/MuonTutorial.cc MuonTutorial::printDTGeom()

- Access to Dt Geometry via `CmsMuon` (need **MuonLayout** group):

```
CmsMuon muonSystem;
MuBarrelSetup* mbSetup = muonSystem.DTSetup(
const MuBarDetectorMap& map = mbSetup->map()
```

- Access to Dt Geometry via `Singleton`:

```
MuBarrelSetup* mbSetup =
  Singleton<MuBarrelSetup>::instance();
const MuBarDetectorMap& map =
  mbSetup->map();
```

DT geometry (IV)

- From setup you can access to `CMSMuonBarrel` and then to `DetLayer` reconstruction geometry `MuBarrelSetup::CMSMBarrel()` If no reconstruction geometry is available (eg if MBLayout not loaded), throws exception
- Get dets() again and again until you reach the Layer level (3 times) *you'll need to dynamic_cast Det to CompositeDet to use dets()*
- You have a Layer (as a Det), must again cast to use MuBarLayer specific interface
- Get number of wires per Layer (not easy nor efficient!)
- Or remember that for DT, the Layers are DetUnit
- Get directly from DetLayers all detUnits()
- Cast the DetUnit to MuBarLayer
- Easier, but not efficient (very slow)

DT geometry (V)

Best solution: we need MuBarLayer specific interface, so it's better to get MuBarLayer as such and not as Det

- Get a reference to `MuBarDetectorMap`, which allows to access the whole DT geometry `MuBarrelSetup::map()`
- get all `MuBarChamber` via MuBarDetectorMap::chambers()
- for each chamber, loop through all SL, then all Layer and count wires
- each chamber (as well as SL and Layer) have a "name", `MuBarChamberId`, via id() method. `MuBarChamberId::wheel()` gives the wheel of chamber ($[-2, +2]$)
- print nWires only if wheel()==0

Ex 2: Get RPC SimHits

template in MuonTutorial::printRPCSimHits()

- For all system, the SimHits are accessible via SimDet
- A SimDet is accessible from a DetUnit
- There are different ways to get all DetUnits
- get them via the DetLayers::detUnits()
- To get DetLayers:
  - Get Rpc setup
  - Get simplyfied geometry from setup
  - Get DetLayers
- As for DT, either via `CmsMuon` or via Singleton

```
MRpcSetUp* mrpcSetup = muonSystem.RPCSetup();
(   ''    = Singleton<MRpcSetUp>::instance();
CMSMuonRpc* cmsrpc=mrpcSetup->CMSMRpc();
vector<DetLayer*> dls=cmsrpc->allLayers();
```

Ex 2: Get RPC SimHits (II)

- Other methods can be
  - Get directly all DetUnit via MRpcMap, in turn accessed as a singleton
  - Navigate through the Rpc geometry in an other (Rpc specific) way:
    - get all MRpcDetectors from setup
    - get all MRpcChamber from MRpcDetectors (MRpcDetectors is composed of MRpcChambers, use getComponent(i) method)
    - get all DetUnit from MRpcChamber (again as components): there are 1 to 3 DetUnit for each chamber, as in reality “chambers”are made by 1 to 3 “detectors”
- Once we have the `DetUnit`, we should get the `SimDet`
- Check if the `SimDet` is really present (in real world there are not such a thing as a `SimWhatever`...)
- Finally, the `SimDet` can give us the `SimHits`

Ex 2.3: Get CSC Digis

template in MuonTutorial::printCSCDigis()

CSC geometry

- `CmsMuonEndcap`: collection of DetLayers (disk)
- `MuEndcapSystem` (all), `MuEndcap` (2), `MuEndStation` (the disks 4+4), `MuEndRing` (2 or 4)
- `MuEndChamber`: provides 4D segments (`RecHit`)
- `MuEndLayer` (6 for each chamber), provides `SimHits`, Digis (separately for Wires and Strips) and hits (aka cluster) (`RecHit`)
- To get the digis we must get all the `MuEndLayers`
- Completely different approach (better?)
- Use `MuEndLayerIterator` (almost STL like, Tim is improving it)

Ex 2.3: Get CSC Digis (2)

- As before, get the `MuEndSetUp` via `CmsMuon` or via Singleton
- Get MuEndSystem from the setup

```
MuEndcapSetUp * setup = muonSystem.CSCSetup(
(    ''     =
  Singleton<MuEndcapSetUp>::instance(); )
MuEndcapSystem * endcapSystem =
  setup->MEndcap();
```

- Instantiate a `MuEndLayerIterator` with the system (the constructor get the system as argument)
- Iterate over all the layer

```
while((MuBarLayer* layer = layerItr.next()))
  {...}
```

- Get the Digis
- That's it!

Ex 4: Print all RecHits

template in MuonTutorial::printAllRecHits()

C'mon, it's the last exercise!

- The goal is to print all the RecHits of the 3 systems
- For RPC, the MRpcDetUnit provide the only RecHits
- For the CSC and DT, there is a hierarchy of RecHits, as there is a hierarchy of Dets
- CSC: Chamber provides segments, Layers provide Hits
- DT: Chamber provide 4D segments, SL 2D segments, layer hits
- Moreover, the segments are composed RecHits, made of RecHits
- Let's concentrate on highest level RecHits, ie segments for DT and CSC

Ex 4: Print all RecHits (II)

- We already know how to access DT chamber, CSC chamber and Rpc DetUnits
- all these object are `Det`, so have a common interface
- To get RecHits from a Det, just use Det::recHits()
- Q: wait! I know that these 2 types of RecHits are different (segment vs hit). How can it be that they are all the same object (RecHit)??
- A: good question! They have the same interface, but have different information. Eg. only the segments have direction. When used in the Kalman fit, every RecHit provide information according to what it really is.
- Print position and direction of DT and CSC, and check that direction is not defined for RPC

Homework

- Repeat ex 1,2,3 for the other 2 sub-systems
- Print also the low level RecHits of CSC and DT
  - Access them via appropriate Det (MuEndLayer, and MuBarSL/Layer, respectively)
  - Do the same also via the composite RecHit interface (namely via RecHit::recHits()). Do you get the same number of RH in both cases? Why?
- For a given event, plot the global position of RecHits and SimHits in $R-Z$ and $x-y$ planes, so to have a simple event display
- Compare SimHits and RecHits to get residuals for all sub-system
- Get a reconstructed muon (see Norbert tutorial), get all the Dets which contribute with a RecHit (valid or not), and look if there are other RH in that Det, and how far

Conclusion

- This was an overview of Muon reconstruction software

Conclusion

- This was an overview of Muon reconstruction software
- Lot of details have been left out

Conclusion

- This was an overview of Muon reconstruction software
- Lot of details have been left out
- Do you think you could do better?

Conclusion

- This was an overview of Muon reconstruction software
- Lot of details have been left out
- Do you think you could do better?
- You have idea about major improving?

Conclusion

- This was an overview of Muon reconstruction software
- Lot of details have been left out
- Do you think you could do better?
- You have idea about major improving?
- There is a lot of room for improvement and for people willing to contribute!

Conclusion

- This was an overview of Muon reconstruction software
- Lot of details have been left out
- Do you think you could do better?
- You have idea about major improving?
- There is a lot of room for improvement and for people willing to contribute!
- Don’t stay just a *user*, become a developer!

Conclusion

- This was an overview of Muon reconstruction software
- Lot of details have been left out
- Do you think you could do better?
- You have idea about major improving?
- There is a lot of room for improvement and for people willing to contribute!
- Don't stay just a *user*, become a developer!

- from CERN *Picked up for you*:

Conclusion

- This was an overview of Muon reconstruction software
- Lot of details have been left out
- Do you think you could do better?
- You have idea about major improving?
- There is a lot of room for improvement and for people willing to contribute!
- Don’t stay just a *user*, become a developer!

- from CERN *Picked up for you*:

“You must be the change you want to see in the world”,
M.K.Ghandi

Conclusion

- This was an overview of Muon reconstruction software
- Lot of details have been left out
- Do you think you could do better?
- You have idea about major improving?
- There is a lot of room for improvement and for people willing to contribute!
- Don't stay just a *user*, become a developer!

- from CERN *Picked up for you*:

"You must be the change you want to see in the CMS"

Thanks

- Many thanks to the organizing committee for this workshop
- In particular Darin for inviting me
- Hans and Natalia for technical support at FNAL