The LHCb Alignment Framework

- LHCb detector description principles and elements
- Applying misalignments to detector components
- Some examples and uses in LHCb
- Conclusions

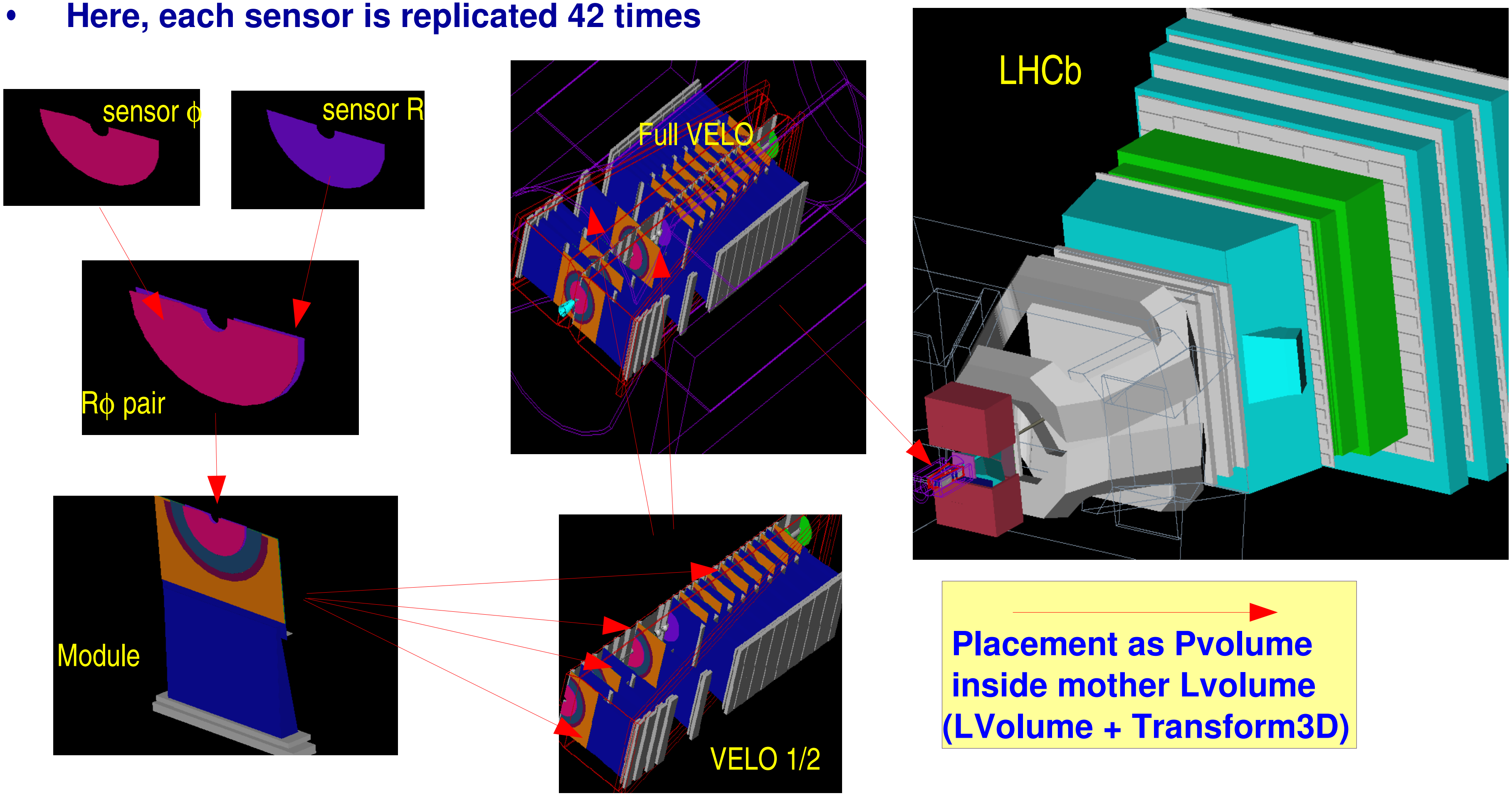
Juan P. Palacios
CERN

LHCb Detector Description

- **Two hierarchical structures**
  - Geometry
    - Re-usable blocks of geometry description
      - Volumes with shape, material...
      - Hierarchy from positioning of volumes within volumes
  - Detector structure
    - Coupled to physical structure of LHCb
    - Hierarchy of “interesting” detector elements
    - One-to-one correspondence to interesting components of real detector
    - Handle very sophisticated information

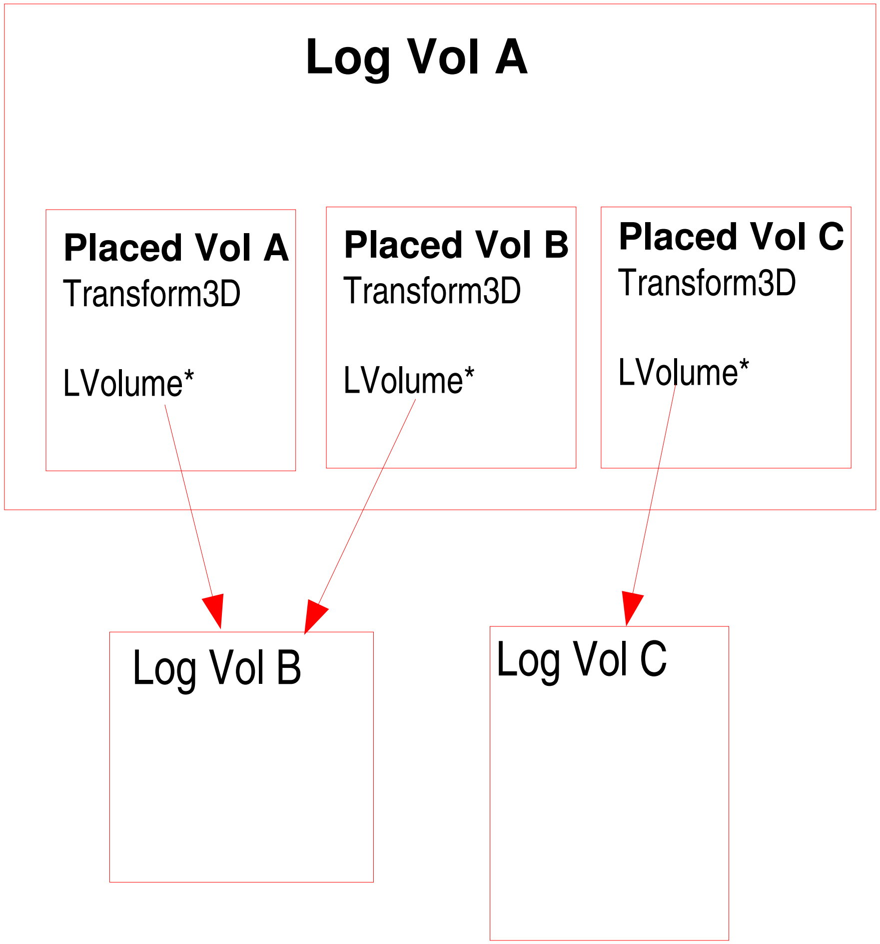
Geometrical description illustration

- Geometrical description elements (volumes) can be seen as replicable generic components that can be used many times
- Here, each sensor is replicated 42 times



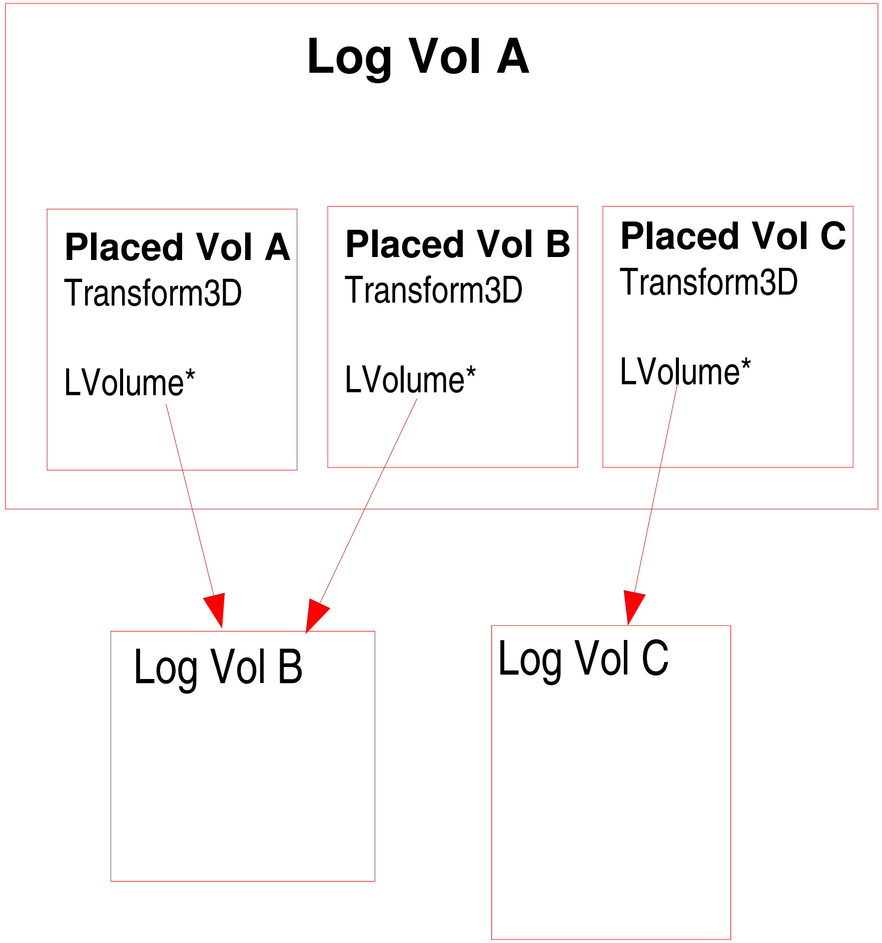
Geometrical description

- **Essential component: Logical Volume**
  - Fixes reference frame
  - Defines geometrical shape, material, optical properties of surfaces
  - Can be placed in other volumes:
    - Logical volume + placement via 3D transformation
    - One logical volume can be placed an arbitrary number of times



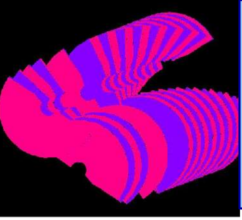
Geometrical description

- **Only link in hierarchy is from parents to children**
  - Parent fixes reference frame of children
  - For children, parents don't exist!
- **One logical volume can be placed many times within the same or different parent logical volumes**
  - Many placements coupling same logical volume to parent(s) via transformation matrices
  - Logical volume only occurs once in memory
- **All computations performed in the frame of reference of the parent**
  - For placed volumes, there is only one frame

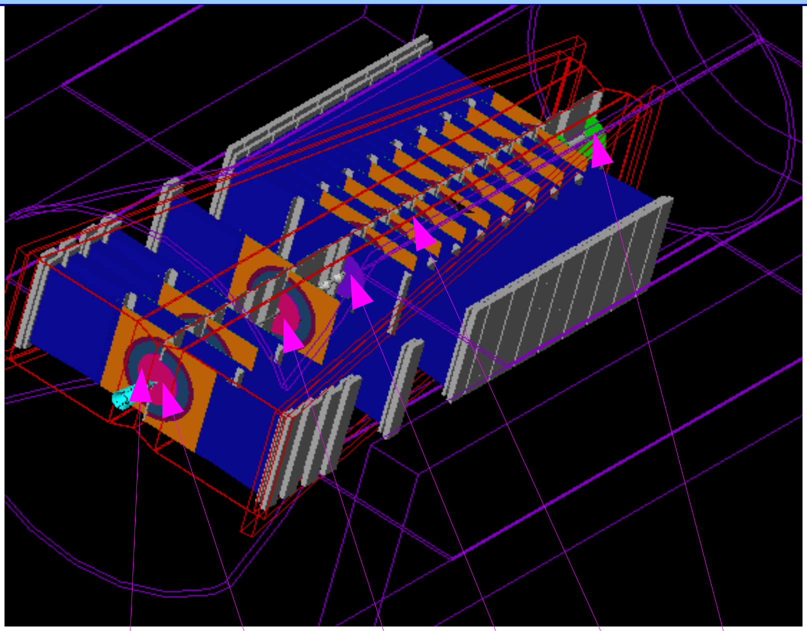


Detector structure: Detector Elements

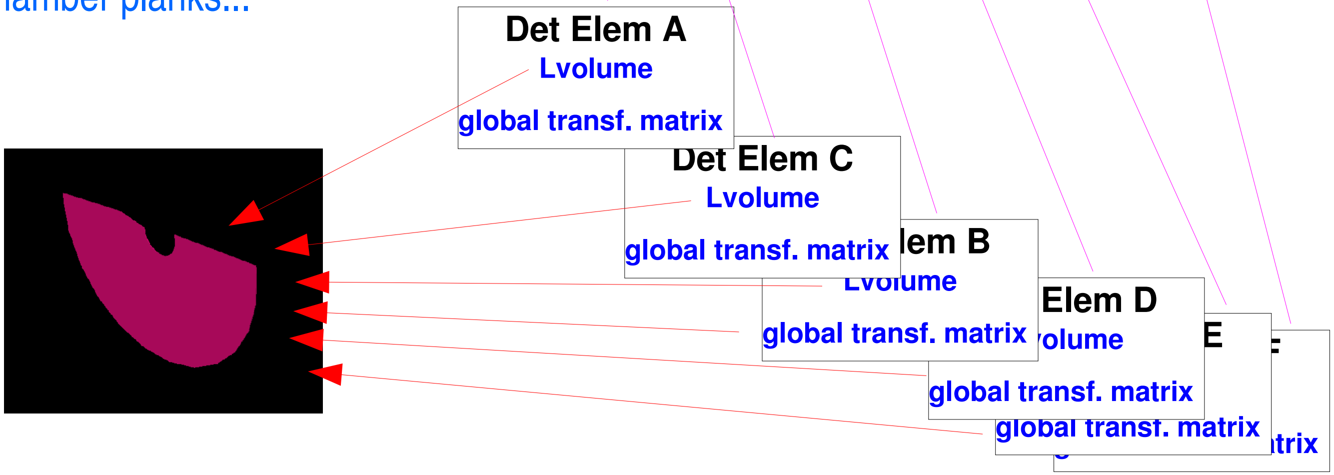
- **DetectorElement class:**
  - handles complex information to do with geometry, readout, calibrations, alignment, detector response...
- **detector structure is hierarchy of these objects**
  - Sub-detectors provide specialisations with arbitrary level of complexity and granularity
  - Coupled but not directly mapped to geometrical hierarchy
  - Allow navigation within hierarchy
- **This is where all users of LHCb detector description interact only with detector elements**



Detector structure hierarchy

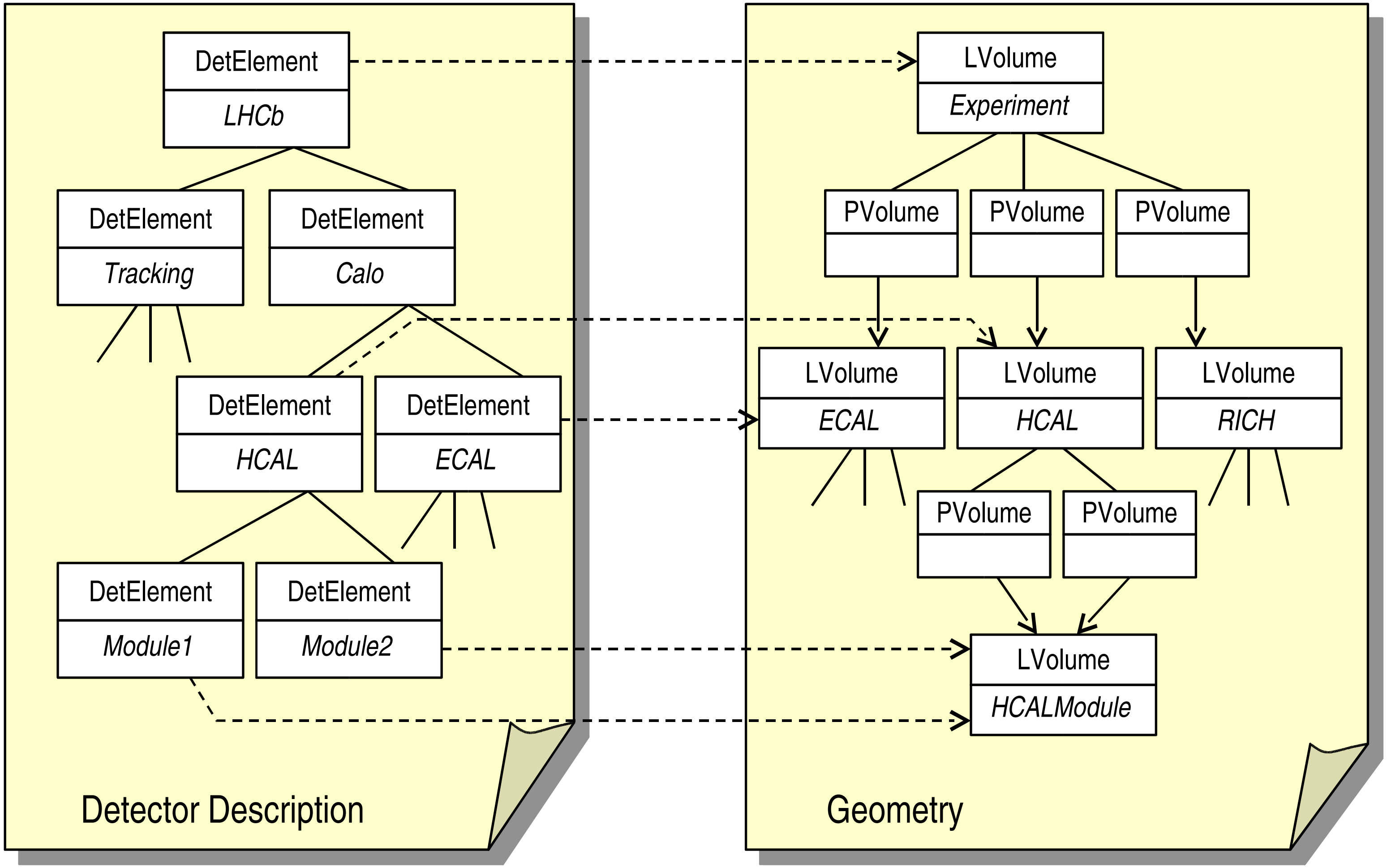


- **Detector elements have one-to-one mapping to interesting components of the real detector**
- **They can each contain *unique* attributes**
- **The granularity and hierarchy wrt. geometrical description is arbitrary**
  - Can decide what are interesting layers
    - sub-detectors, readout regions, Si, sensors, straw chamber planks...



Detector and geometry hierarchies

Detector hierarchy (detector elements) | **Geometry hierarchy (replicable, nestable elements)**

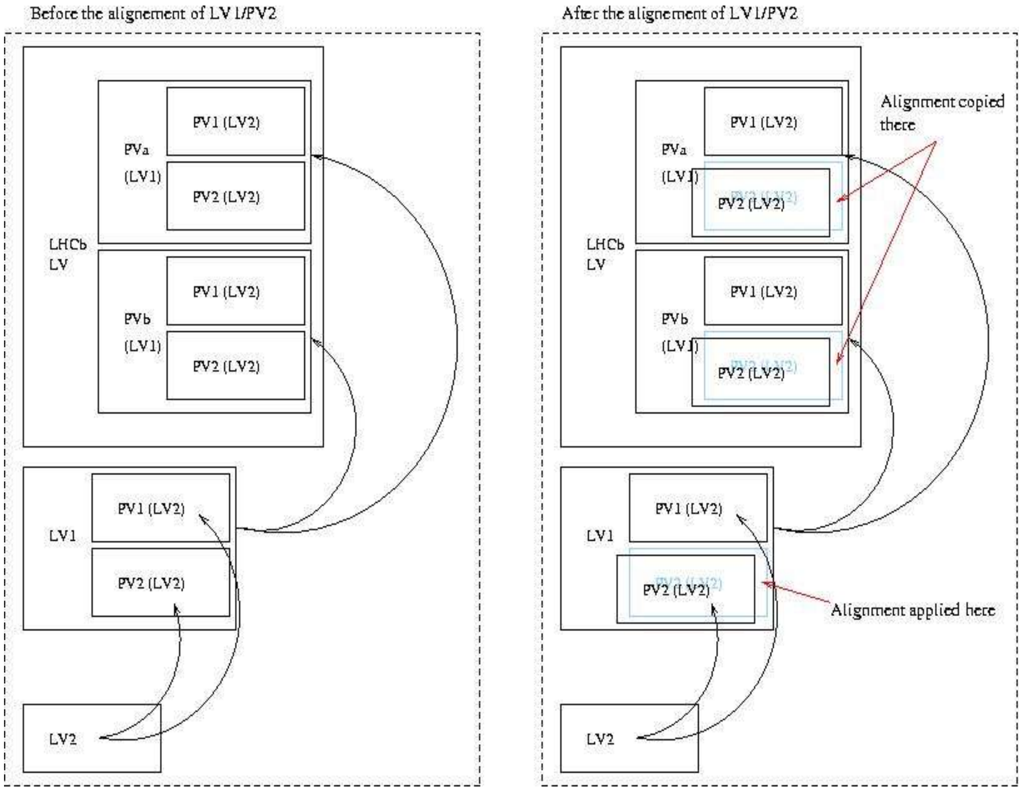


Detector structure

- **Detector element combines knowledge of definition and placements of logical volumes, plus UP and DOWN links, to obtain**
  - One-to-one mapping with LHCb components
  - Knowledge of position in space
    - position in global reference frame (transf. matrix)
    - position in parent volume (link to placement transformation)
  - Knowledge of place in hierarchy
    - UP and DOWN links to detector element parents and daughters
  - Knowledge of daughter placed volumes (not necessarily detector elements)
  - Many other things not relevant here
    - Gain calibrations, strip capacitances, S/N, etc.

Introducing misalignments into detector description

- **Applying misalignments within DetDesc framework and principles not trivial**
  - Replication of geometrical description elements:
    - Misalignments cannot be safely applied at that level (see diagram)
  - Propagation of misalignments through hierarchy
    - Both initialisation and run-time changes must be properly handled considering all inter-dependencies



- **Volume replicability and misalignments don't mix!**
- **Freely replicable volumes -> out of control propagation of misalignments**
- **MISALIGNMENTS CANNOT BE APPLIED TO GEOMETRY ELEMENTS**

Misalignments in detector structure

- **Characteristics of detector elements make them a good place to handle misalignments**
  - Combine local misalignment with local transf. matrix to obtain new local position
  - Use links to parents to establish global position after local misalignment
  - Use links to daughters to propagate misalignments to daughters' global position matrices
- **Misalignments can be handled here with minimal change to user code and maintaining the design principles of the LHCb detector description**

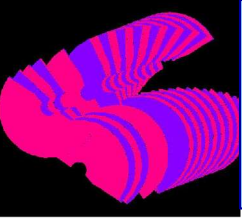
Misalignments into detector structure

- **Misalignments are applied through detector structure**
  - "Interesting" detector elements have access to misalignment matrix
  - Misalignment represents change from nominal alignment **in the reference frame of the detector element** ie relative to its parent detector element
  - Seems reasonable since this is the point of contact to DetDesc
- **Misalignment parameters stored in local files (debugging) or in conditions database (CondDB*)**
  - ***CondDB's update mechanism**** allows for propagation of misalignments after any type of change (time validity or simply manual change)
  - Misalignment is nine parameters: X,Y,Z of displacement, and pivot point, α, β, γ of rotation about cartesian axes

*See talk by M. Clemencic

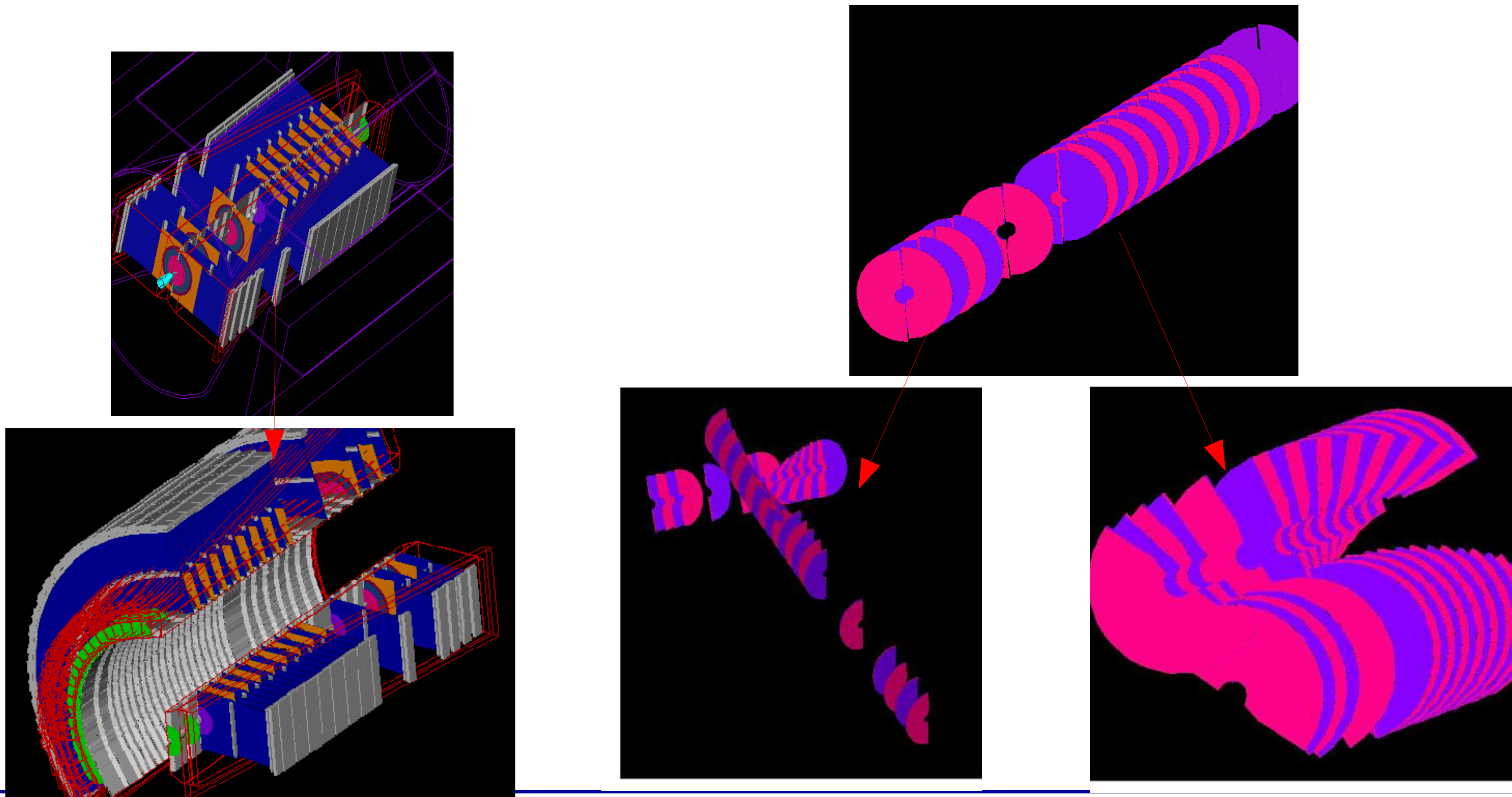
Access and use of misalignments

- **Detector description users see the off-nominal alignment automatically**
  - Local -> global and global -> local transformations now contain misalignments
  - Users must take care of binding all caching of geometrical information to validity of dependent parameters
    - In practice most of this already done in “core” components
  - Misalignments dependent on parent's misalignments
    - potentially many matrix calculations after each change
- **Users can modify misalignments at run-time**
  - changes are propagated and should be seen by all other users
  - This is a transient change in memory, not to be confused with updating database



Some (extreme) misalignment examples

- **Misalignment allows for rotation about arbitrary pivot point + translation**
- **Arbitrary misalignments applied at different levels of hierarchy during course of SW application**



Use of misalignment information in LHCb

Nominal + off-nominal alignment information accessible and usable everywhere

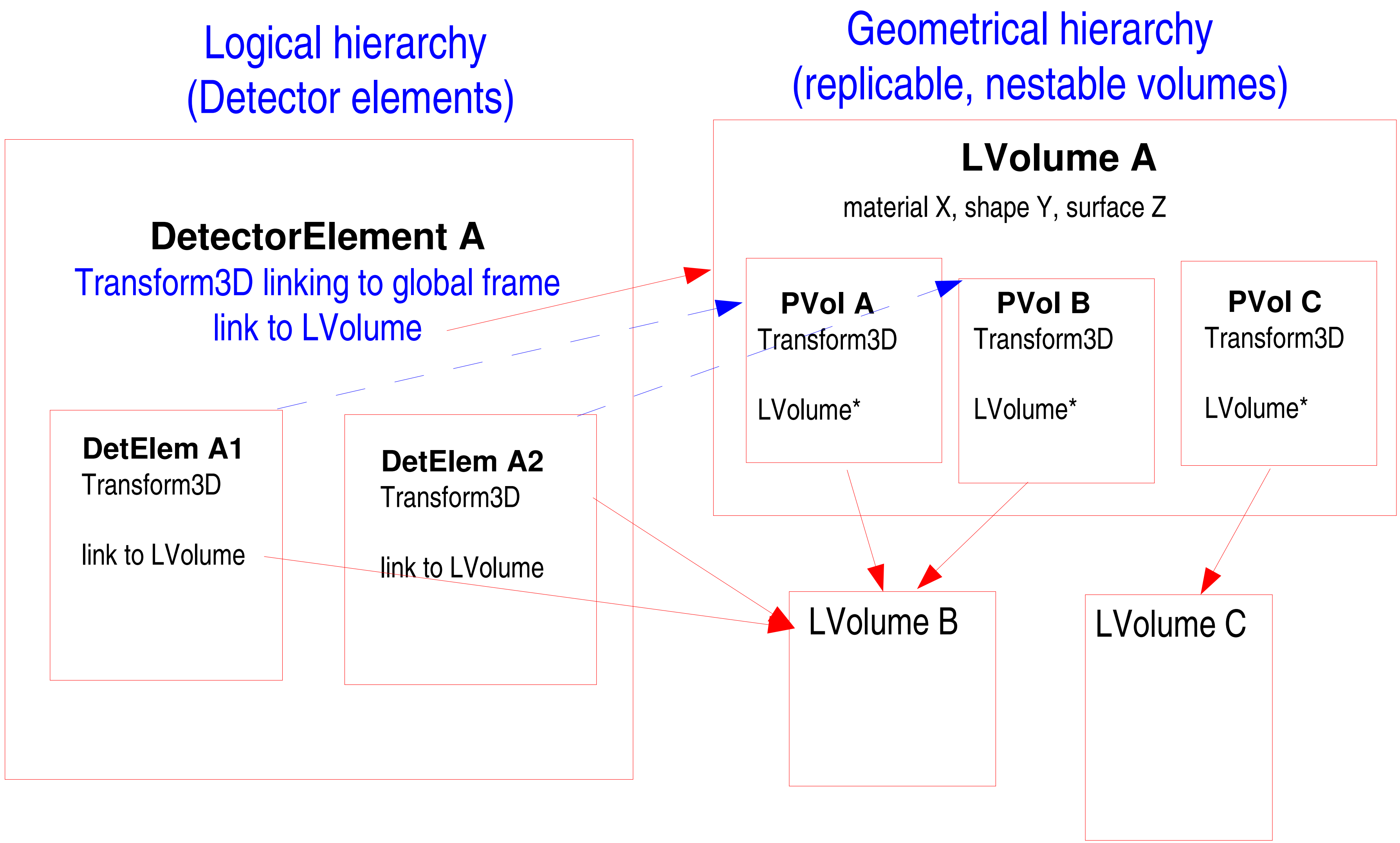
- **Tests of misalignment systematics**
  - Simulate misaligned detector
  - Digitise hits using correct “misalignments”
  - Reconstruct tracks, vertices, etc. with different alignment constants
  - See the effect on physics performance
- **Alignment algorithms**
- **Potential real running scenario: detector alignment**
  - Apply alignment procedure
  - New alignment constants to lightweight DB slice
  - Validation process
  - Tagged copy to master DB at CERN
  - Replicated in tier 1 centres

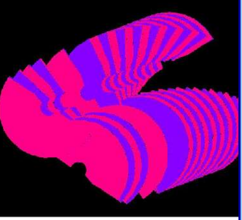
Summary and conclusions

- The LHCb detector description framework has been successfully extended to allow run-time misalignments to detector components
- Misalignments are tied in to the Conditions Database framework to allow both automatic run-time updating and propagation of changes, plus versioning and time dependence of alignment parameters
- The functionality has been tested within the LHCb reconstruction chain
- LHCb sub-detectors are using it to investigate detector alignment procedures and strategies, systematic effects, etc.
- The extension respects the design principles of the LHCb detector description suite and is therefore a non-intrusive enhancement of the framework

Backup slides

Logical and geometrical descriptions





Detector element approach
- Outline

- **One detector element per alignable object**
  - Example: whole VELO, VELO halves, r-ϕ pairs, individual sensors
- **One “delta” transformation matrix per alignable object**
  - one to one mapping between detector elements and delta transformations
- **No changes to LHCb detector geometry description philosophy**
  - Physical volumes are placed logical volumes
  - Physical volumes know only of position within mother logical volume

Detector element approach
- One detector element per alignable object

- **Up to sub detector SW people or alignment specialists to decide granularity**
- **Detector elements do not HAVE to be specialised**
  - Not necessary to write dedicated detector elements
  - Enough to have "**detelemref**" in XML structure
  - Need to have associated logical volume ***but not necessary to have associated solid***
- **Detector element has pointer to IGeometryInfo**
  - As its name indicates, this holds all the geometry information related to that detector element

Detector element approach
- Aside: IGeometryInfo

- **Pointer accessible through DetectorElement::geometry() methods**
- **Initialised from XMLDDDB <geometryinfo/>**
  - Logical volume name, physical volume support and path,…
- **Has local to global and global to local transformations and a host of other services**
  - See **http://lhcb-release-area.web.cern.ch/LHCb-release-area/LHCB/doc/ht**
- **Natural place to incorporate misalignments**
  - Each detector element will now have ideal and misaligned geometry information
  - User code should remain the same

Detector element misalignments
- Implementation

- **Conditions catalogue**
  - Need paths of conditions plus information to convert stored information into transformation matrices (and whatever else is needed)
- **Condition class: *AlignmentCondition***
  - Class to hold an alignment delta transformation and related info.
- **New *GeometryInfo*** implementation
  - Must use alignment deltas wisely
- **Example: the VELO**

Implementation
- Conditions catalogue

- **Added “condition” attribute to geometryinfo in structure.dtd**
  - Only necessary to add an

```
<geometryinfo condition=“somePath” />
```

  in each <detelem …. /> definition

  - Then store the condition somewhere:

```
<condition classID=“6” name=“somePath”>
     <paramVector name=“dPosXYZ” type=“double”> 0. 0. 0.</paramVector>
     <paramVector name=“dRotXYZ” type=“double”> 0. 0. 0.</paramVector>
</condition>
```

- **Information necessary to construct tranf. matrix**
- **VELO example Written in test XML files**
- **Can store as XML strings in CondDB: keep format the same!**

Implementation
- AlignmentCondition

- **Very simple class holding delta transformation related to an alignable object**
  - In object's mother's frame
  - Also contains inverse transformation
  - Is a ValidDataObject so knows about validity ranges
- **At the moment ONLY contains transformation + inverse, and a method to set a new transformation**
- **A basic building block, but obviously could evolve…**

Implementation
- AlignmentCondition creation

- **Constructed from condition path and dedicated XML “converter”**
- **classID = 6 maps to XmlAlignmentConditionCnv**
  - This is a little template which simply instantiates the right kind of condition
- **AlignmentCondition has access to the parameters and constructs the matrices**
  - ***Could add more information if requested!***

```
<condition classID=“6” name=“/dd/Conditions/LHCb/myDetector/Module67”>
     <paramVector name=“dPosXYZ” type=“double”> 0. 0. 0.</paramVector>
     <paramVector name=“dRotXYZ” type=“double”> 0. 0. 0.</paramVector>
</condition>
```

- **Constructed and accessed via data service:**

```
SmartDataPtr<AlignmentCondition> cond(datasvc(),
                    “/dd/Conditions/LHCb/myDetector/Module67”);
```

- **In practice, constricted in new GeometryInfo**

Implementation
- New GeometryInfo

- **Handles both ideal and delta transformations:**
  - DetectorElement public interface stays the same.
  - IGeometryInfo toGlobal and toLocal methods ***now deal with combined ideal + delta transformations***
  - Methods to get ideal geometry have been added
  - Possible for users to update delta matrix
  - Easy to “refresh” state of new GeometryInfo. All caching controlled from one method, aptly named ***cache()***
  - GeometryInfo accesses CondDB (or test XML file), generates all necessary matrices
- **VELO example implementation ready and tested with XML conditions catalogue**

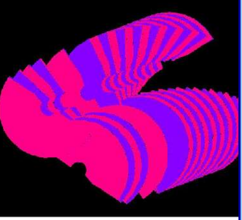
Implementation
- GeometryInfo details

- **Constructor uses path to AlignmentCondition**
  - Gets condition from data store (XML file or CondDB)
- **Scans down tree picking up parent transformations**
  - Iterative procedure finding GeometryInfos of support detector elements
  - Stores ideal and delta transformations for each level in vectors
    - Allow to re-calculate after updates
  - Combines ideal matrices to get ideal case local to global
  - Combines all matrices to get local to global with misalignments
- **"Local" misalignment matrix can be updated by user**
  - Re-calculates "global" matrix automatically
  - At the moment this is de-coupled from automatic update mechanism

Using misalignment information

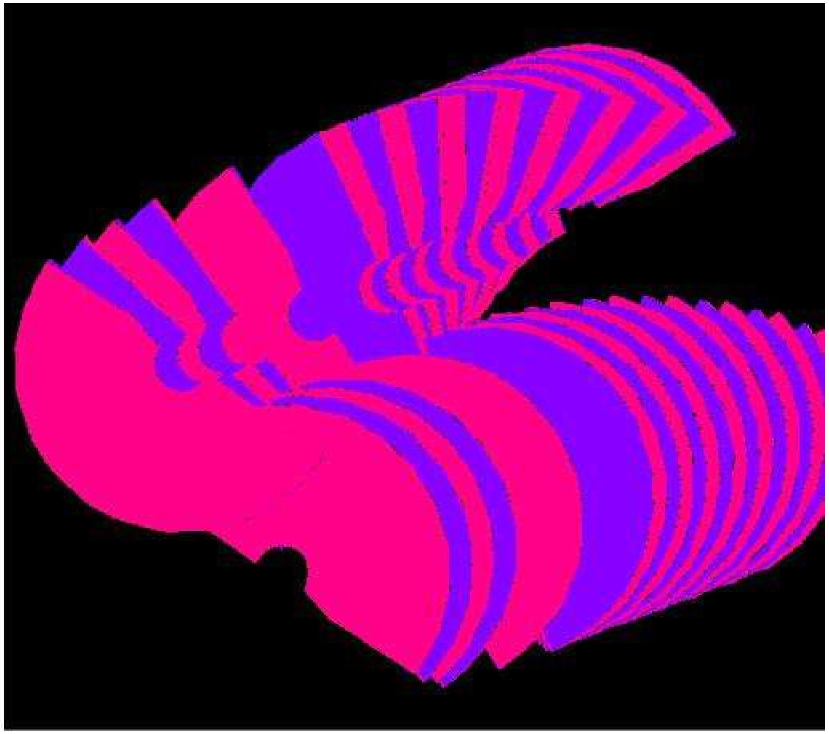
- **Keep currently used DetectorElement interface**
  - DetectorElement::geometry() points to new GeometryInfo
  - Standard transformation methods now deal with misaligned geometry
    - User code will automatically get misaligned geometry transformations if unchanged
  - New methods allow to perform ideal transformations
  - Delta matrix also available – and can be modified

•BEWARE: code using DetectorElement::geometry() will now get MISALIGNED GEOMETRY automatically!
•To get misaligned geometry, new code must use detector elements



Using misalignments in LHCb SW
- Panoramix case

- **Uses detector element to plot volume**
  - Automatic access to misalignment
- **BUT does not get misalignments for daughters**
  - Must specify ***each*** detector element



/dd/Structure/LHCb/Velo/VeloLeftModule13
/dd/Structure/LHCb/Velo/VeloLeftModule15
/dd/Structure/LHCb/Velo/VeloLeft/Module17

Open VELO with Z-dependent ϕ misalignemt from test XML conditions file

Plots with misalignments up to the module level but NOT sensor misalignments

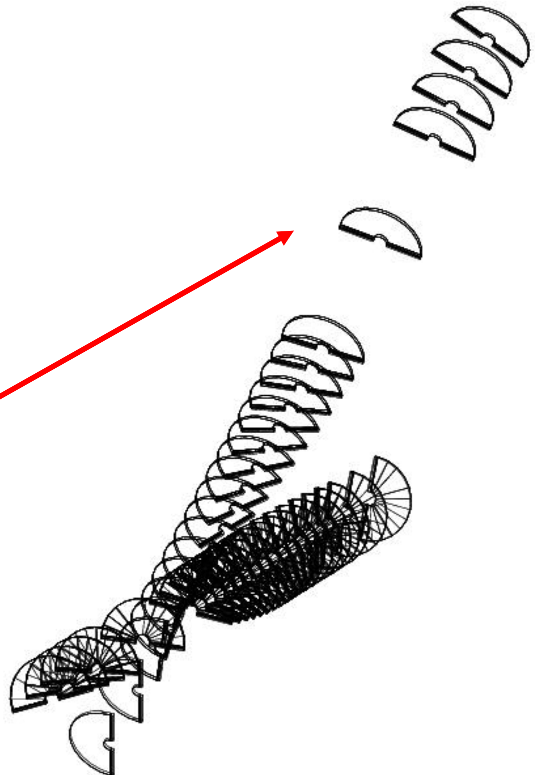
Using misalignments in LHCb SW
- Gauss case

- **In Gauss add detector geometry stream in options file:**

```
Geo.StreamItems +=
       {“/dd/Structure/LHCb/Velo”}
```

- **This would disregard any misalignments other than whole VELO**
- **To get all misalignments need to add leaves explicitly as in Panoramix**

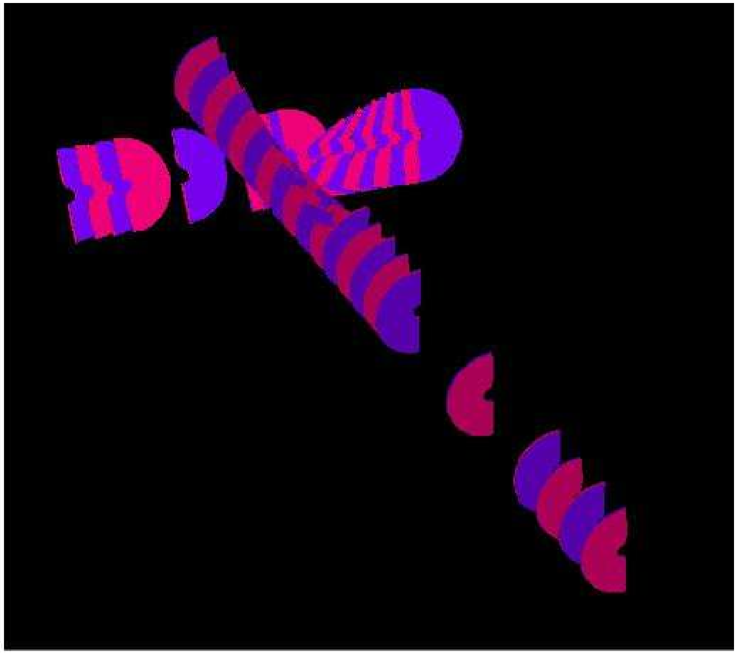
But what do do about PV without detector element? Or PV in wrond place in hierarchy? Will need to write some new GiGa code…



Geant4 Open VELO with Z-dependent ϕ misalignemt from test XML conditions file

Using misalignments in LHCb SW
- General remarks

- **All information is available, through the detector element, via the new GeometryInfo**
- **As long as daughter detector elements are accessed, and NOT daughter physical volumes, all should be automatic**



Open VELO with Z-dependent ϕ misalignemt and rotation about X axis. From test XML conditions file

Using misalignments in LHCb SW
- More general remarks

- **Must bear in mind the misalignments are *conditions* and could change in a job**
  - Caching of geometrical information should be done in a controllable manner
  - Update mechanisms will exist, but depend on users associating methods to conditions
  - Up to users to ensure that re-running a condition-dependent method really does what it needs to
- **Remember many volumes are in close proximity**
  - What about possible overlaps?
  - How to avoid them?

This framework gives total freedom to move things… up to users to move them intelligently!

Using CondDB and UpdateManager
- Accessing CondDB

- **Reading and writing to CondDB**
  - Would like to say “it is easy”
  - Can’t say yet
    - I haven’t had the time to do it (although have pretty good examples from **Marco Clemencic**… thanks!)
  - But it looks easy!
- **Important points:**
  - Data store interface: just requires appropriate paths to conditions in XML
  - Store XML strings: can keep the same format as have now in test XML file

Using CondDB and UpdateManager
- Updating information

- **How to update all necessary information when a condition changes**
  - Again, nice framework from Marco Cl. but I haven't had the time to use it yet
- **Basically, register function/condition pairs to UpdateManager**
  - It takes care of re-running function whenever condition changes
- **Important: authors of code must ensure changes get propagated wherever needed**
  - I take care of AlignmentCondition, GeometryInfo, default DetectorElement and any other "framework" code
  - For the rest, up to you!

Requirements from sub-detector SW

- **One detector element per alignable object**
  - Can be default (no code to write)
  - Must have path to an alignment condition
  - In current scheme this means a condition in the XML <geometryinfo /> definition of the detector element. This contains the correct path in the data store
- **A catalogue of conditions with one-to-one mapping to detector elements**
  - In absence of condition identity matrix is created so current XML descriptions still work for ideal
- **A geometry description where the alignable objects can be associated to an LVolume**
- **A controlled and clear caching of condition-dependent geometry information**
  - Control all caching from a minimal number of methods as they will have to be registered somewhere
  - Ensure that if these methods are run everything is updated in consistent manner

Requirements from sub-detector SW /2

- **You are required to tell me what you require!**
  - Need to know about current and future uses of geomatry information
- **AlignmentCondition**
  - Currently a matrix wrapper with time validity
  - Could make it more interesting: errors on alignment parameters?
  - More ways to construct a transformation?
    - At the moment, just six numbers…
- **GeometryInfo**
  - Any other information/functionality required?
- **DetectorElement**
  - Anything more clever needed (besides returning pointer to GeometryInfo)?
- **Likely uses**
  - Simulation
  - Tracking
  - Alignment
  - Others?

Not all require full det. Description…

Next steps

1. **Test algorithm to populate CondDB with snail matrices**
2. **Retrieving snail matrices from CondDB**
    1. Simple test algorithm
    2. DetDesc (via appropriate paths in XML)
3. **Test algorithm to manually change matrices in DetDesc**
4. **Incorporate UpdateManager into GeometryInfo, DetectorElement**
5. **Write dedicated GiGaStream to use misalignments in simulation**

Recap: (some) open questions

- More information in AlignmentCondition?
- More ways of defining transformations?
- IGeometryInfo interface?
- DetectorElement interface?
- Simulation: overlaps?
- Transport service: overlaps?
- Questions?

Two Hierarchies