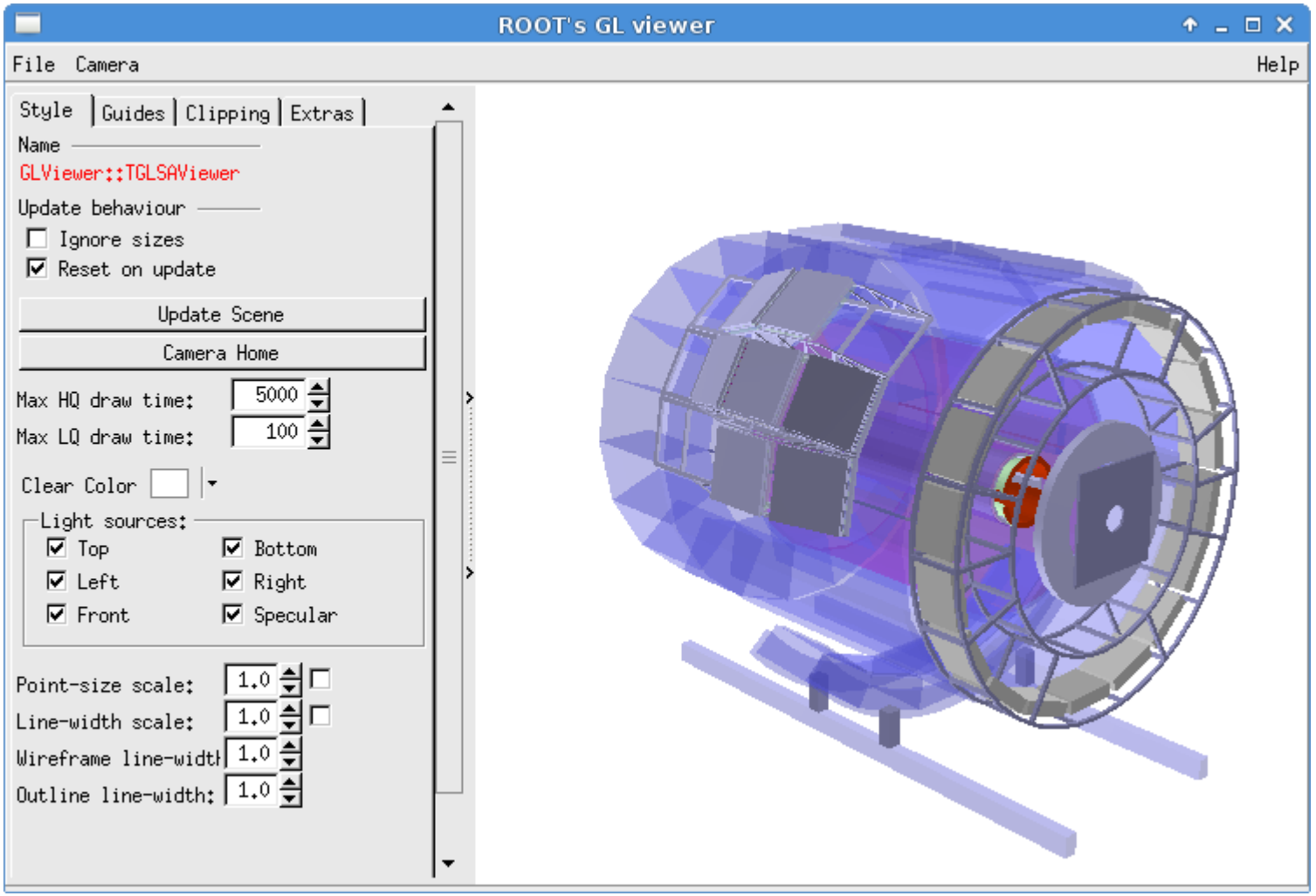
Navigation in and Alignment of (Panda)ROOT Geometries

Prometeusz Jasinski
08.06.2015
Panda Collaboration Meeting

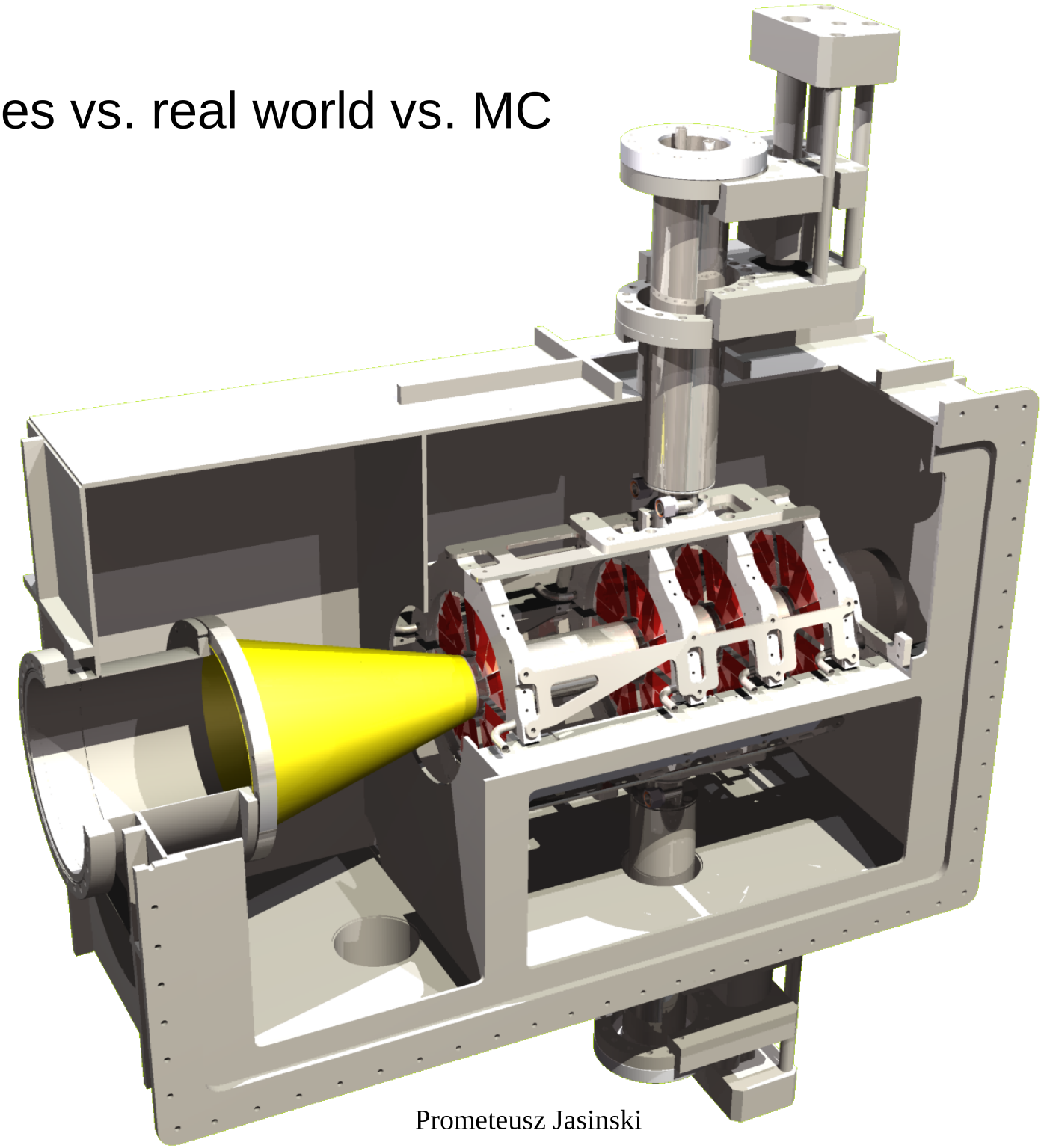
Navigation in (Panda)ROOT

$ROOTSYS/tutorials/geom/geomAlice.C

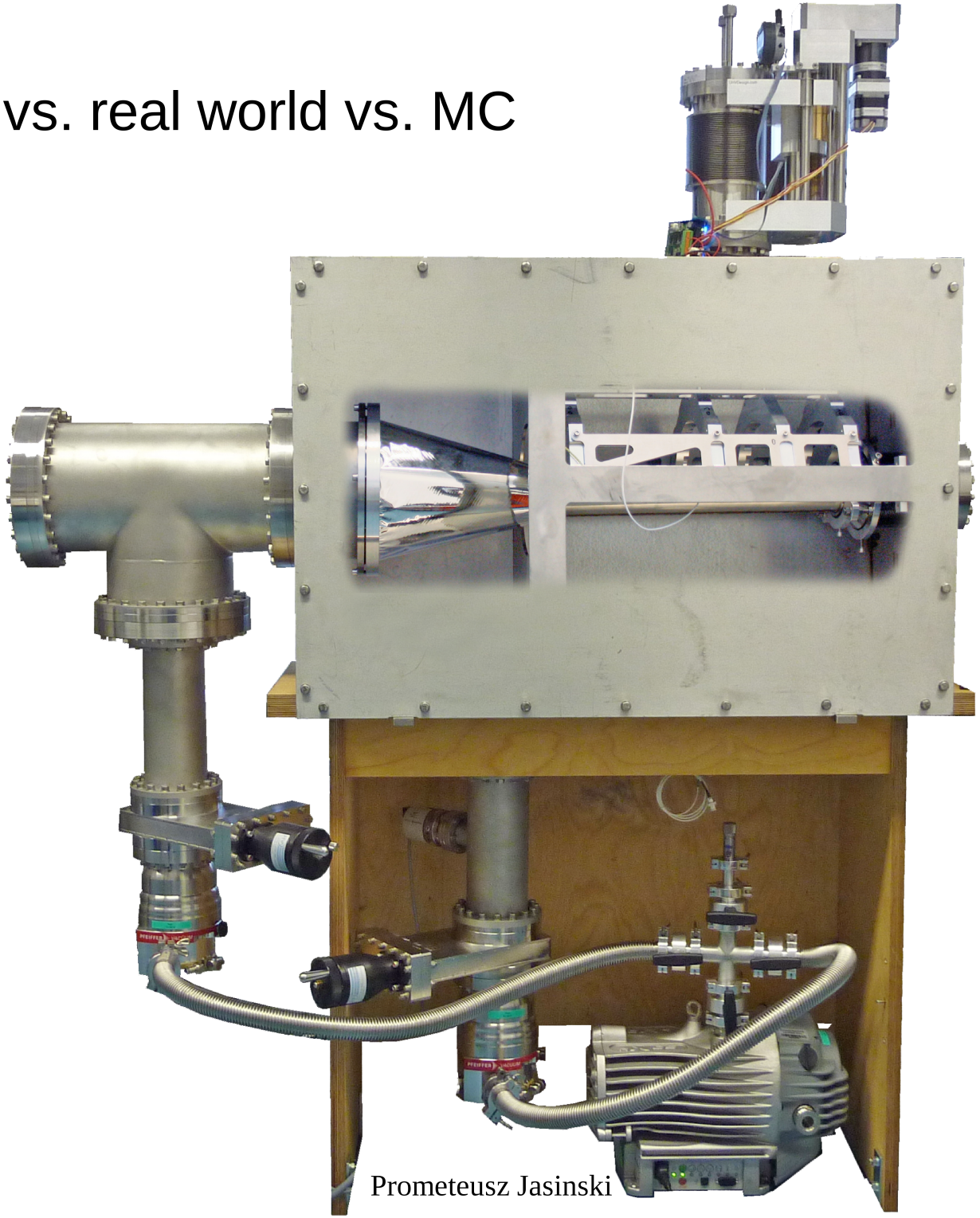


Reminder: All TGeo* and VMC stuff in root was developed for/by ALICE
http://iopscience.iop.org/1742-6596/331/3/032016

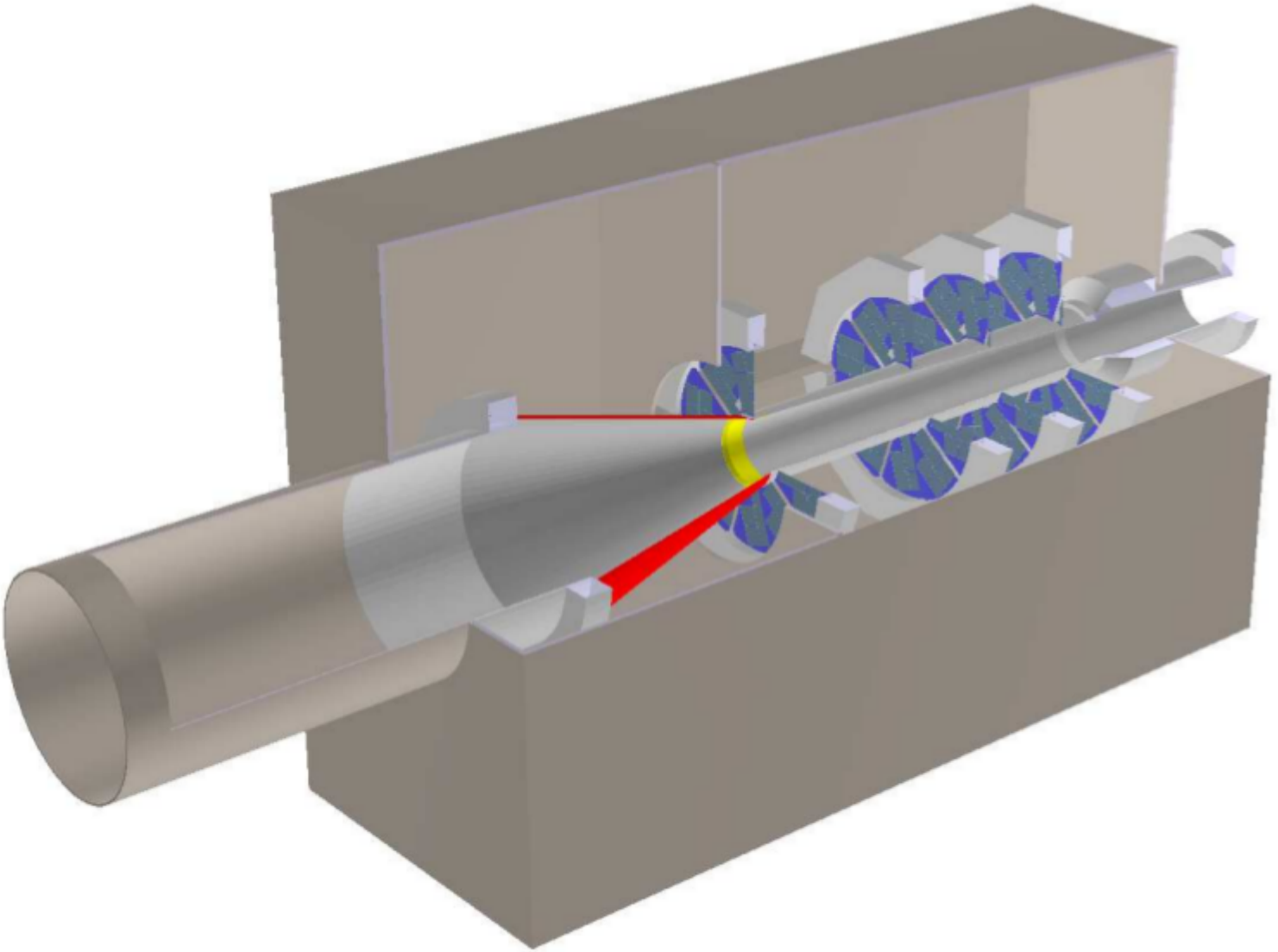
Sketches vs. real world vs. MC



Sketches vs. real world vs. MC

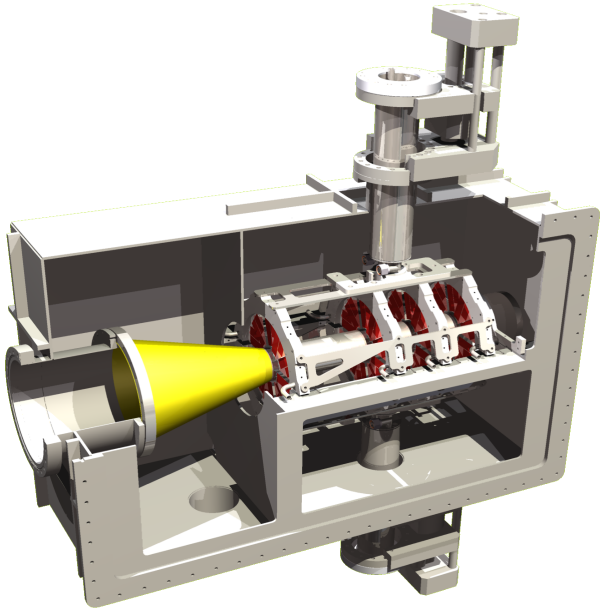


Sketches vs. real world vs. MC

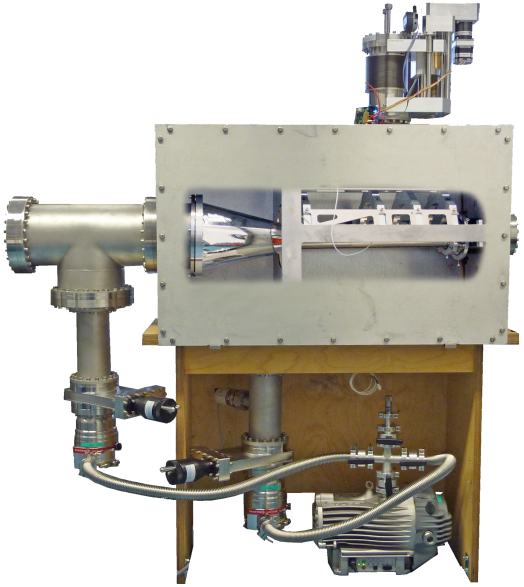


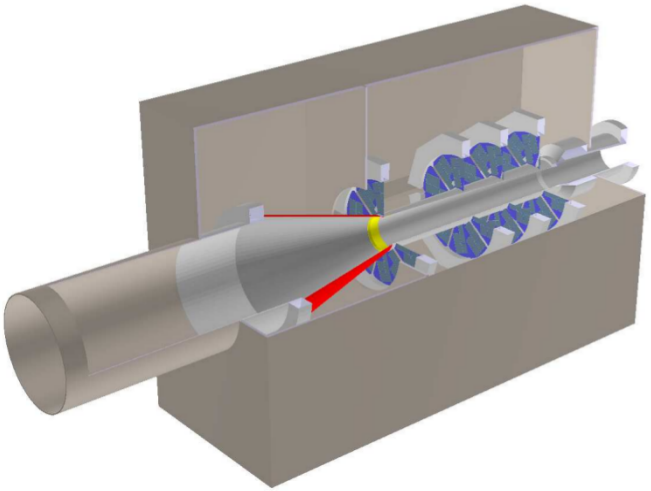
Sketches vs. real world vs. MC

Straight forward:
give sketches
to the workshop



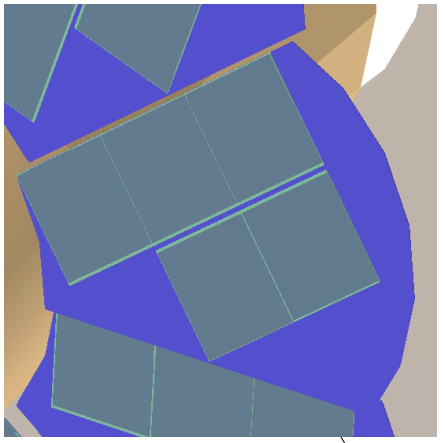
Not so easy:
Use MVD approach or
create it from scratch in ROOT



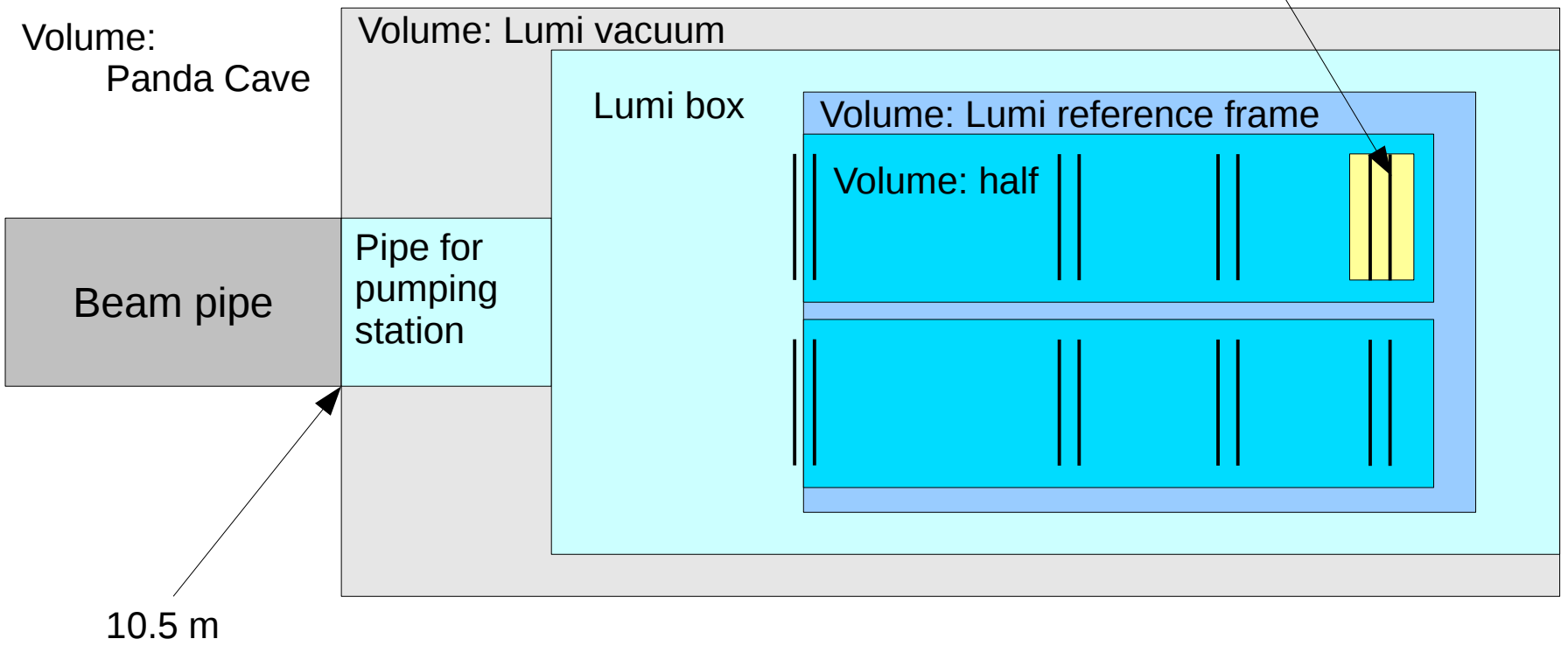


Mother-Daughter Volume Relations

Volume: plane
-> Volume: module
-> Volume: side (back is rotated!)
-> Volume: die (currently not misaligned)
-> Volume: sensor (not misaligned)



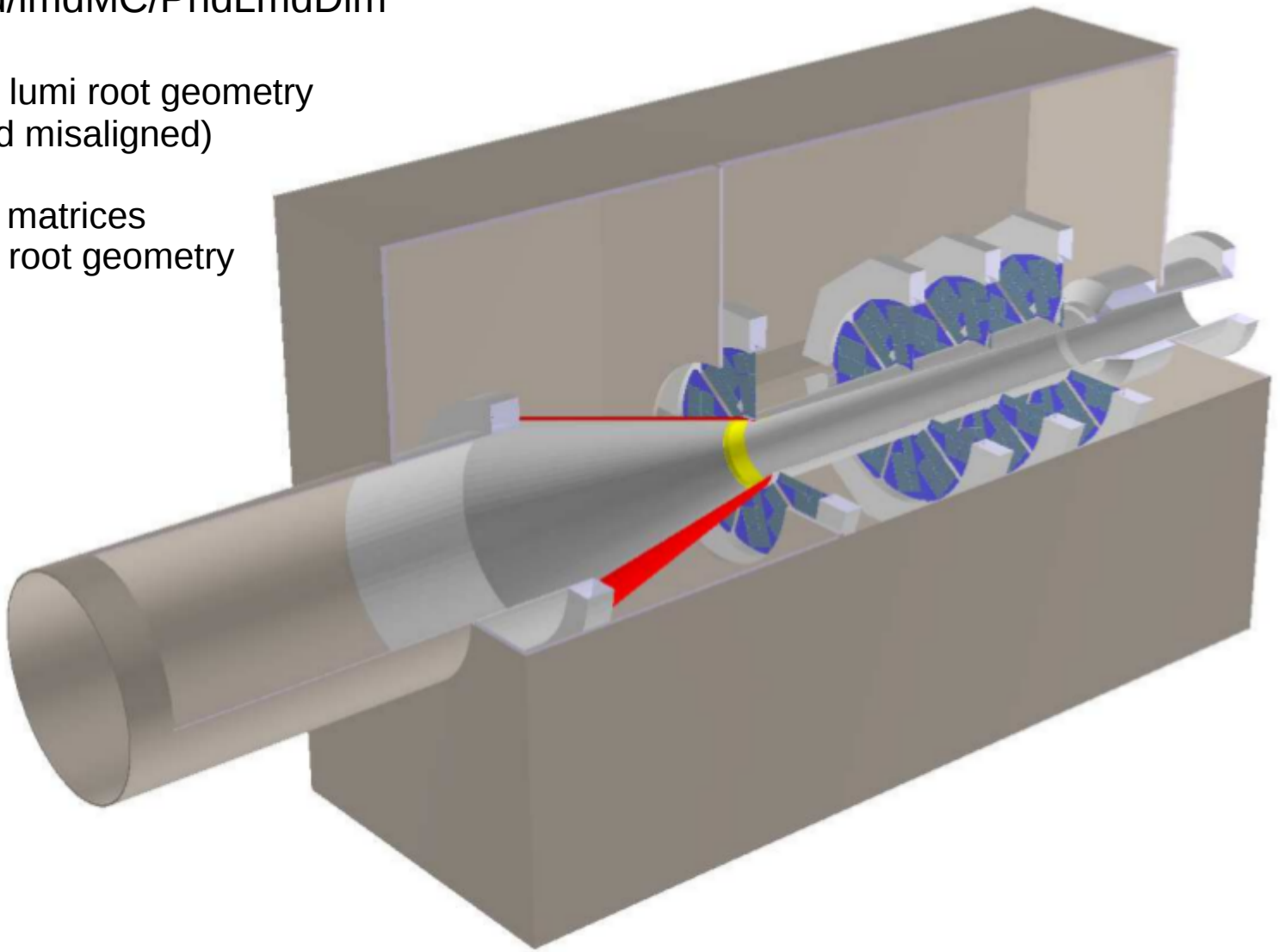
Notice: Sensors contain active AND passive Material



Creation of LMD ROOT Geometries

$PANDAROOT/lmd/lmdMC/PndLmdDim*

- Construction of lumi root geometry
  (aligned and misaligned)
- Lumi constants
- Transformation matrices
- Interplay with a root geometry

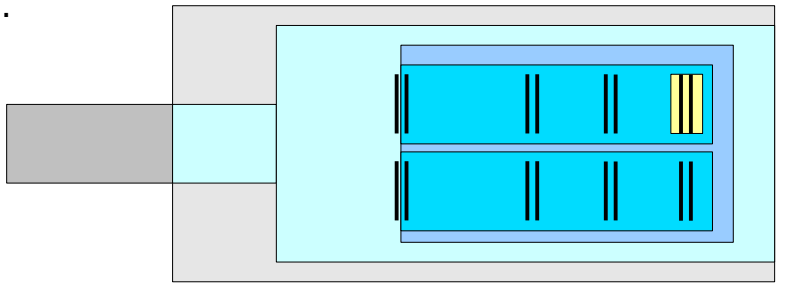


How to access/modify transformation matrices in nodes?

ROOT TGeoManager Navigation

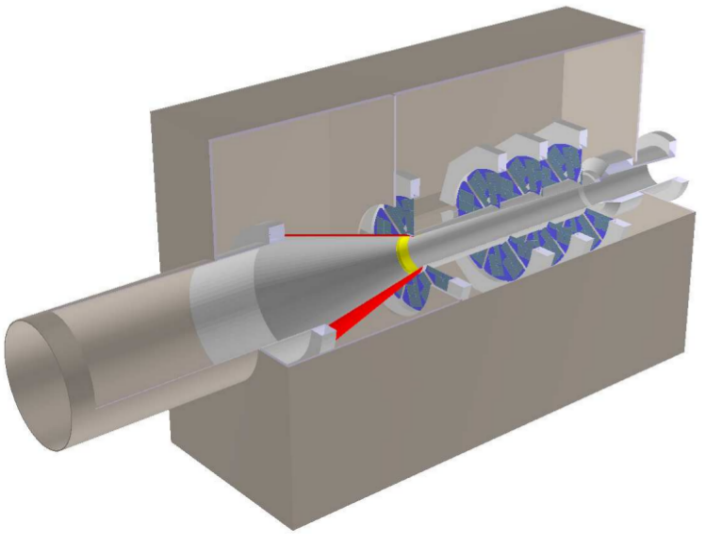
Quoting Chapter 19 of the root documentation:

„Physical nodes are the actual “touchable” objects in the geometry, representing actually a path of positioned volumes starting witrh the top node: path=/TOP/A_1/B_4/C_3 , where A, B, C represent names of volumes.
[...] The knowledge of the path to the objects that need to be misaligned is essential since there is no other way of identifying them. One can however create “symbolic links”...“



```
gGeoManager->cd("/lum_1/lmd_vol_vac_3/lmd_vol_ref_sys_0/"
        "lmd_vol_half_1/lmd_vol_plane_3/lmd_vol_module_4/"
        "lmd_vol_side_1/lmd_vol_die_1/LumActivePixelRect_398");
gGeoManager->GetCurrentNode()->GetMatrix();
```

Tested myself for our geometry:
Random access performance is pretty much the same as a std::map (binary search tree) with a dynamically constructed string as a key



BUT the path (Sensor ID) can change, when you hand over your geometry to FAIRROOT!

The Complication with the Detector ID and Sensor ID

Path when generating the geometry:

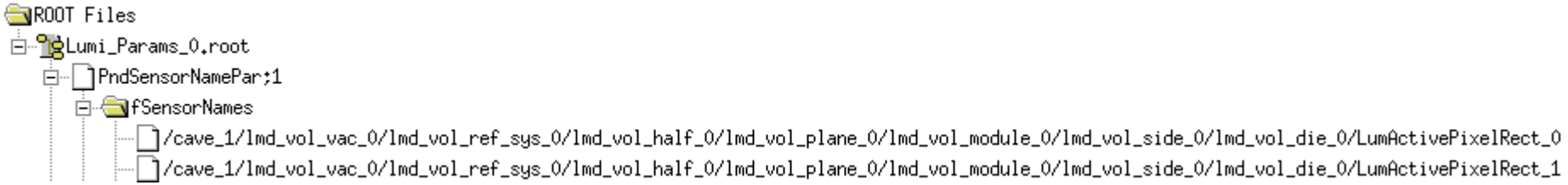
The copy number
is actually the Sensor ID

```
gGeoManager->cd("/lum_1/lmd_vol_vac_3/lmd_vol_ref_sys_0/"
        "lmd_vol_half_1/lmd_vol_plane_3/lmd_vol_module_4/"
        "lmd_vol_side_1/lmd_vol_die_1/LumActivePixelRect_398");
gGeoManager->GetCurrentNode()->GetMatrix();
```

Handing over to FairROOT:

```
//  -------------------------  MVD  ----------------
  PndMvdDetector *Mvd = new PndMvdDetector("MVD", kTRUE);
  Mvd->SetGeometryFileName("Mvd-2.1_FullVersion.root");
  fRun->AddModule(Mvd);
//  -------------------------  LMD  ----------------
  PndLmdDetector *Lum = new PndLmdDetector("LUM", kTRUE);
  Lum->SetExclusiveSensorType("LumActive");  //ignore MVD
  Lum->SetGeometryFileName("Luminosity-Detector.root");
  fRun->AddModule(Lum);
```

Result in the pseudo DB:



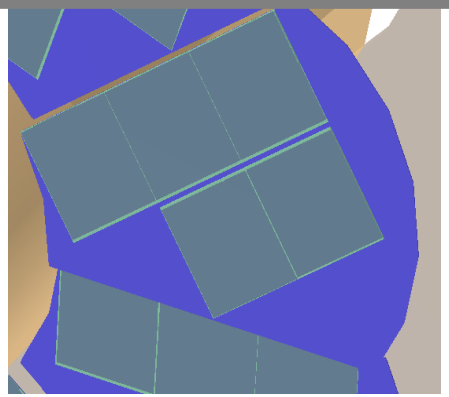
mother volume changed

in the past: copy number changed
(seems to be fixed now and DetectorID is used instead)

In Principle PndSensorNamePar gives you the path, but only for active volumes!
Actually why is it not a FAIRROOT feature?

Alignment of the LMD

1. Align Sensors with respect to each other on a module with hits



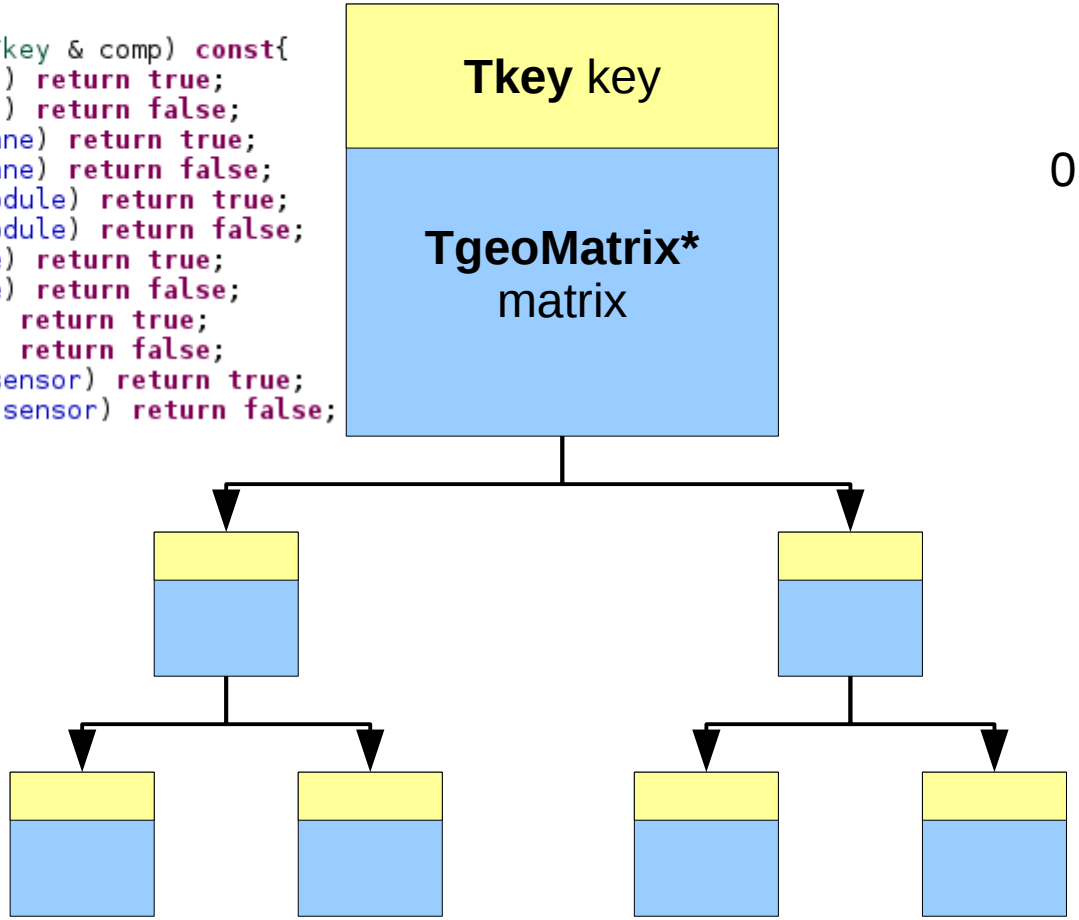
2. Align Modules in respect to each other via Millipede with tracks
3. Align the LMD with respect to the IP and magnetic fields with propagated tracks

-> Touching only mother volumes in the geometry tree

If we touch only active material we get clashing volumes!

Matrices in the LMDDim class

```
class Tkey {
public:
    signed char half;
    signed char plane;
    signed char module;
    signed char side;
    signed char die;
    signed char sensor;
    bool operator < (const Tkey & comp) const{
        if (half < comp.half) return true;
        if (half > comp.half) return false;
        if (plane < comp.plane) return true;
        if (plane > comp.plane) return false;
        if (module < comp.module) return true;
        if (module > comp.module) return false;
        if (side < comp.side) return true;
        if (side > comp.side) return false;
        if  (die < comp.die) return true;
        if  (die > comp.die) return false;
        if  (sensor < comp.sensor) return true;
        if  (sensor >= comp.sensor) return false;
        return false;
    }
```



depth

0 = lmd ref frame

-1 = half

-2 = plane

-3 = plane

-4 = module

...

Gain in speed = x 3.5 for random access
+ access to nearly all matrices of the geometry
+ geometry does not have to be loaded

Why is it NOT the wholy grale?

Quoting Chapter 19 of the root documentation:

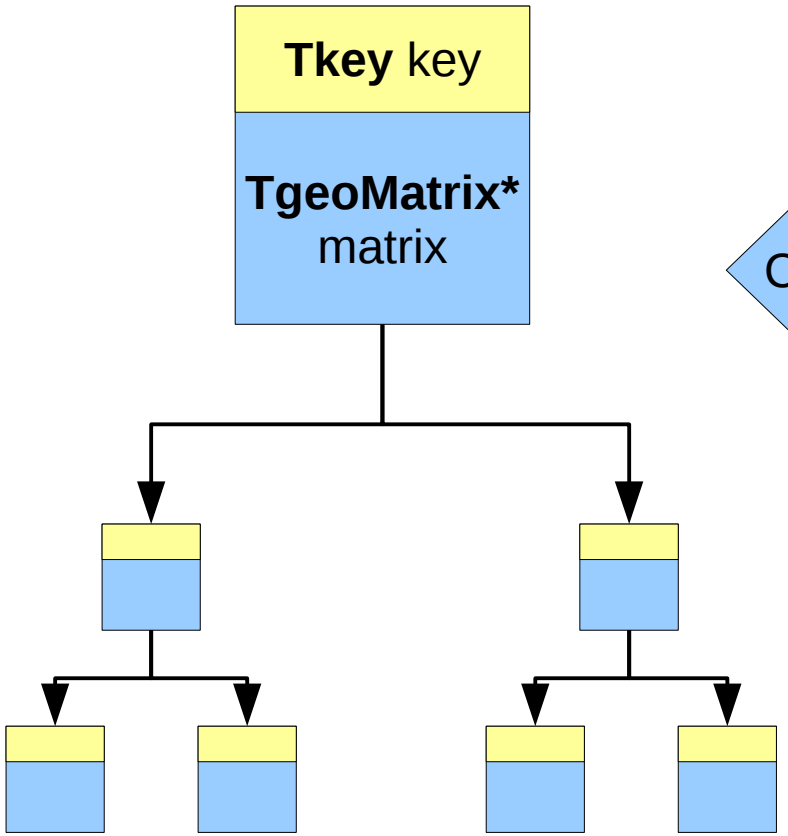
[...] it is impossible to create all physical nodes as objects in memory. [...]

Question: does it still apply to todays computer farms?

AND ...

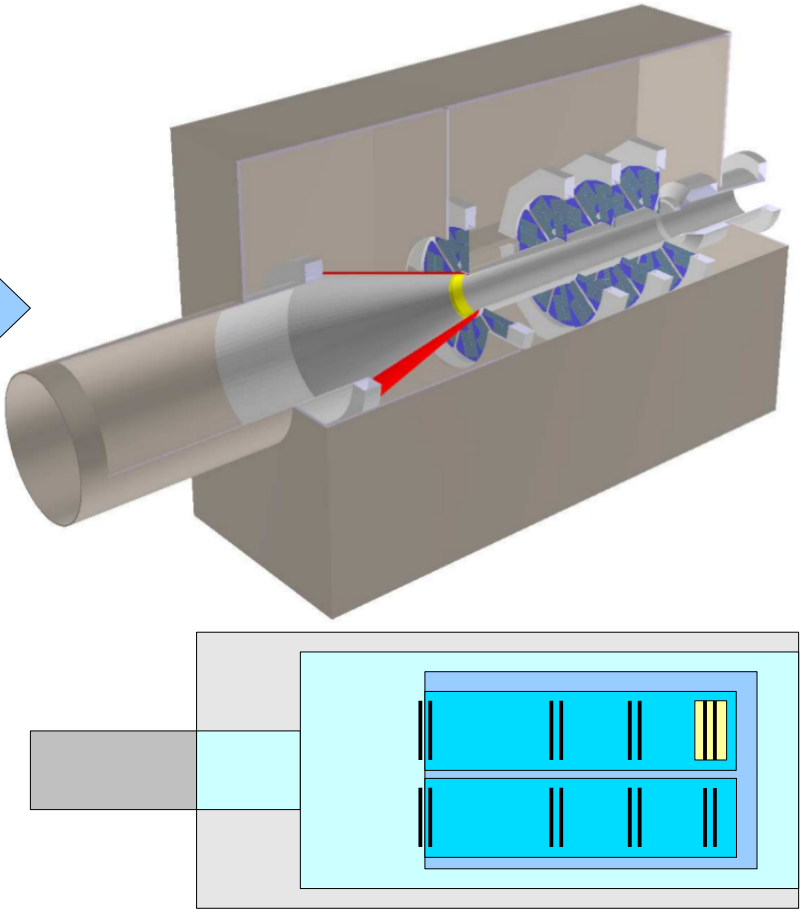
Two parallel worlds

PndLmdDim tree



Consistency

Root Geometry



used at some special LMD calculation places
for example: **(Mis)alignment**

used nearly everywhere across PandaROOT

Synchronizing PndLmdDim with a root geometry

```
// get a list of sensor paths in the geometry navigation model
// returns the path to the lmd top volume
// It is a recursive search, call it once with the default found_lmd variable
// in case first_call then gGeoManager->CdTop(); is executed to get to the top node
// of a geometry
// The geometry must be loaded
string Get_List_of_Sensors(vector <string>& list_of_sensors, bool found_lmd = false, bool first_call = true);

// check a list of sensor paths for validity
// result is true if tests were sucessful
// offset is the offset in the sensor number which may be
// not 0 if the geometry was not created in the first place
bool Test_List_of_Sensors(vector <string> list_of_sensors, int& offset);

// Get an offset for a volume, if not existent and random
// a random offset is generated and stored
// Is used during generation of geometries when calling
void Get_offset(int ihalf, int iplane, int imodule, int iside, int idie, int isensor,
        double& x, double& y, double& z,
        double& rotphi, double& rottheta, double& rotpsi, bool random = false);
```

Synchronizing PndLmdDim with a root geometry

```
// get the transformation matrix for an path in an existing root geometry
// no checks are performed in advance
// result may be 0!
// if (aligned) the matrix after a possible alignment is returned
// in that case details to the matrix must be provided in form
// of ihalf ... isensor
// to do: get rid of path and do it only on the basis of ihalf ... isensor
// if (!aligned) the original matrix is returned
TGeoHMatrix* Get_matrix(string path, bool aligned = true,
        int ihalf = -1, int iplane= -1, int imodule = -1, int iside = -1, int idie = -1, int isensor = -1);

// set the transformation matrix for an path in an existing root geometry
// A matrix can be only aligned,
// therefore by default original matrices are not touched!
// since a key must be created for a node
// details to it must be provided in form of
// ihalf ... isensor
// to do: get rid of path and do it only on the basis of ihalf ... isensor
bool Set_matrix(string path, TGeoHMatrix* matrix,
        int ihalf = -1, int iplane= -1, int imodule = -1, int iside = -1, int idie = -1, int isensor = -1);/
```

Synchronizing PndLmdDim with a root geometry

```
// read transformation matrices from a loaded geometry
// aligned and not aligned are two separate maps
// containing the description of the detector positions
// the geometry must be loaded otherwise matrices cannot be read
// version number will be set according to the geometry version number
// To Do: multiply also matrices on the way to the key matrices
//        in case those are not unity matrices
bool Read_transformation_matrices_from_geometry(bool aligned = true);

// apply transformation matrices to a loaded geometry
// aligned and not aligned are two separate maps
// containing the description of the detector positions
// the geometry must be loaded otherwise matrices cannot be read
// version number will be set according to the geometry version number
// IMPORTANT: you may choose which PndLmdDim matrices you want to use
// but a ROOT Geometry can always be only aligned. The original
// matrix stays untouched!
// To Do: multiply also matrices on the way to the key matrices
//        in case those are not unity matrices
// To Do: Find out how to store the aligned geometry as a default
//        one to pandaroot parameter files
bool Write_transformation_matrices_to_geometry(bool aligned = true);
```

Warning: those changes are *not persistent* and must be applied for *each task*!

And quoting the documentation once more:

„The Align() [I'm using it for each matrix to be set] method will actually duplicate the corresponding branch within the logical hierarchy, creating new volumes and nodes. This is mandatory in order to avoid problems due to replicated volumes and can *create exhaustive memory* consumption if used abusively. „

-> It should be checked if performance is ok,
when „(mis)aligning“ the whole Panda Geometry,
or do we have to (mis)align and to store as a default geometry?

Conclusion

We are able to align and navigate through the geometry.

We can perform transformations with our own matrix handler
but retaining consistency is a complication.

We need common design rules and a common basis before proceeding
with DB entries and so on.

Would be nice to know the experience from Alice with their own framework.
What are the pitfalls?

Thank you

Remark: I'm leaving Panda on 01.07.15 for a permanent position in the industry.
Thank you for the great time here!

Backup

Code examples

misalignment:

```
lmddim = PndLmdDim::Instance();
if (readAlign){
    string dir = getenv("VMCWORKDIR");
    lmddim->Read_transformation_matrices(dir+"/geometry/trafo_matrices_lmd_misaligned.dat", true);
    lmddim->Write_transformation_matrices_to_geometry(true);
}
```

or some tests:

```
if (1){ // consistency checks
    cout << " reading matrices from file into the aligned map " << endl;
    lmddim.Read_transformation_matrices(dir+"/geometry/trafo_matrices_lmd_misaligned.dat", true);
    cout << " loading transformation matrices from geometry into not aligned map " << endl;
    lmddim.Read_transformation_matrices_from_geometry(false);
    cout << " writing matrices from aligned map to geometry" << endl;
    lmddim.Write_transformation_matrices_to_geometry(true);
    cout << " reading transformation matrices from geometry once again " << endl;
    lmddim.Read_transformation_matrices_from_geometry(true);
    //cout << " testing matrices " << endl;
    //lmddim.Calc_matrix_offsets();
}

if (0){ // test setting individual matrices
    TGeoHMatrix* matrix = lmddim.Get_matrix("/lum_1/lmd_vol_vac_3/lmd_vol_ref_sys_0/lmd_vol_half_1", false, 1,-1,-1,-1,-1,-1);
    matrix->Print();
    matrix->RotateX(90.);
    matrix->Print();
    lmddim.Set_matrix("/lum_1/lmd_vol_vac_3/lmd_vol_ref_sys_0/lmd_vol_half_1", matrix, 1,-1,-1,-1,-1,-1);

    cout << " the original matrix is " << endl;
    matrix = lmddim.Get_matrix("/lum_1/lmd_vol_vac_3/lmd_vol_ref_sys_0/lmd_vol_half_1", false, 1,-1,-1,-1,-1,-1);
    matrix->Print();

    cout << " the aligned matrix is " << endl;
    matrix = lmddim.Get_matrix("/lum_1/lmd_vol_vac_3/lmd_vol_ref_sys_0/lmd_vol_half_1", true, 1,-1,-1,-1,-1,-1);
    matrix->Print();
}
cout << " testing matrices " << endl;
lmddim.Calc_matrix_offsets();

gGeoMan->RefreshPhysicalNodes(); // should be called but is not a must
//gGeoMan->CloseGeometry();
top->Draw("ogl"); // an already drawn geometry will be not updated according to changes in the matrices
```

Search tree was: optimized string based key

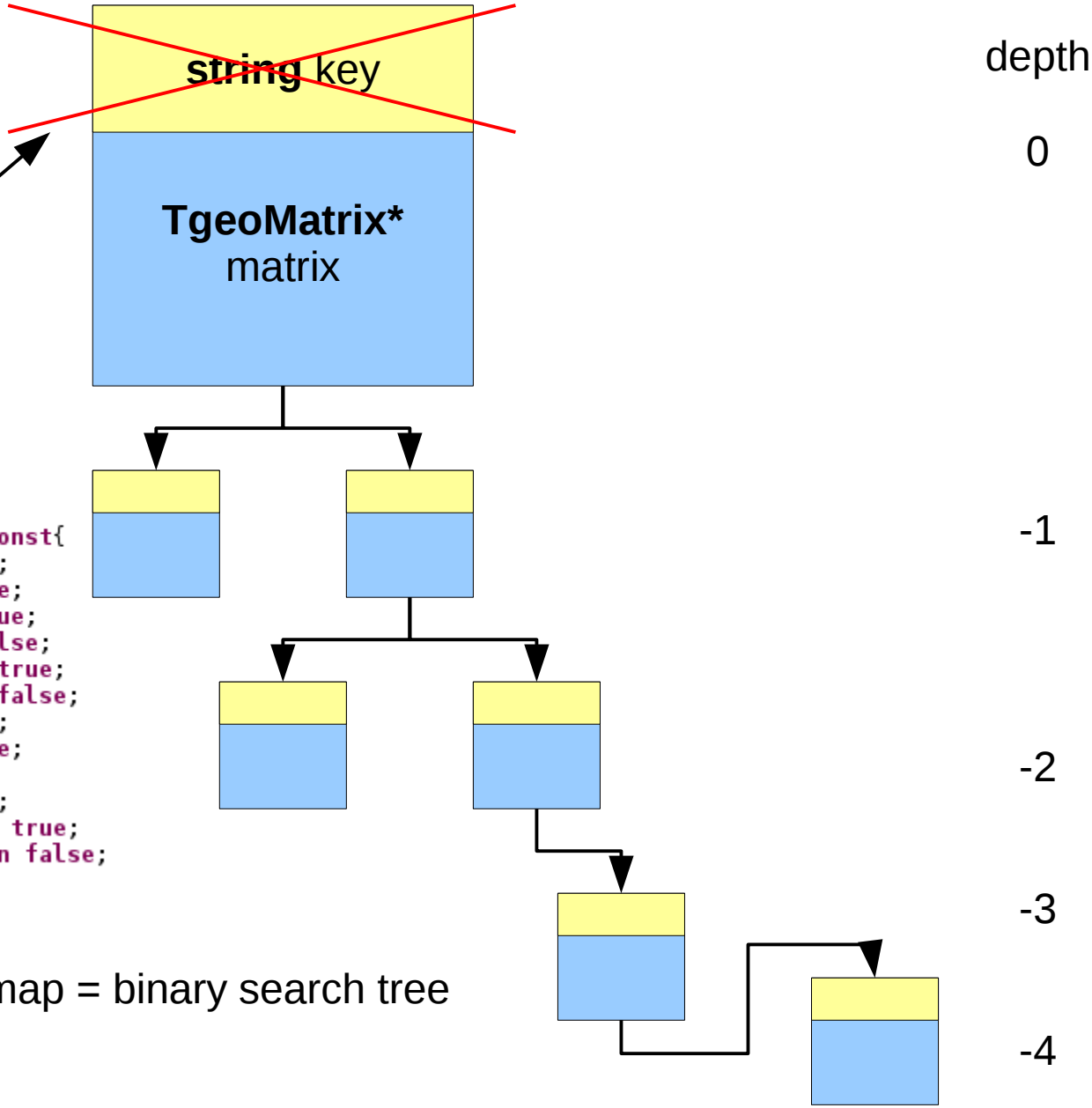
```
/**
     * C++ version 0.4 char* style "itoa":
     * Written by Lukás Chmela
     * Released under GPLv3.
     */
char* itoa(int value, char* result, int base) {
    // check that the base if valid
    char* last_char;
    if (base < 2 || base > 36) { *result = '\0'; return result; }
    char* ptr = result, *ptr1 = result, tmp_char;
    int tmp_value;
    do {
        tmp_value = value;
        value /= base;
        *ptr++ = "zyxwvutsrqponmlkjihgfedcba9876543210123456789abcdefghijklmnopqrstuvwxyz" [35 + (tmp_value - value * base)];
    } while ( value );
    // Apply negative sign
    if (tmp_value < 0) *ptr++ = '-';
    last_char = ptr;
    *ptr-- = '\0';
    //cout << last_char << endl;
    while(ptr1 < ptr) {
        tmp_char = *ptr;
        *ptr--= *ptr1;
        *ptr1++ = tmp_char;
    }
    return last_char;
}

string Generate_key(int ihalf, int iplane, int imodule, int iside, int idie, int isensor){
    char key[100];
    char* ptr;
    ptr = itoa(ihalf, key, 10);
    ptr = itoa(iplane, ptr, 10);
    ptr = itoa(imodule, ptr, 10);
    ptr = itoa(iside, ptr, 10);
    ptr = itoa(idie, ptr, 10);
    ptr = itoa(isensor, ptr, 10);
    string result(key);
    //stringstream keystream;
    //keystream << ihalf << iplane << imodule << iside << idie << isensor;
    return result;
}
```

A typical key was:
„0 2 1 0 -1 -1“
with the standard „<“ operator therefore
not mapping the mother-daughter relations

Reducing the depth of the PndLmdDim tree

```
class Tkey {
public:
    signed char half;
    signed char plane;
    signed char module;
    signed char side;
    signed char die;
    signed char sensor;
    bool operator < (const Tkey & comp) const{
        if (half < comp.half) return true;
        if (half > comp.half) return false;
        if (plane < comp.plane) return true;
        if (plane > comp.plane) return false;
        if (module < comp.module) return true;
        if (module > comp.module) return false;
        if (side < comp.side) return true;
        if (side > comp.side) return false;
        if  (die < comp.die) return true;
        if  (die > comp.die) return false;
        if  (sensor < comp.sensor) return true;
        if  (sensor >= comp.sensor) return false;
        return false;
    }
```



map = binary search tree