

# Materials and Geometry

Davide Chiappara  
University of Padova (UNIPD)  
Istituto Nazionale di Fisica Nucleare (INFN)

# User classes (starting...)

## At initialization

**G4VUserDetectorConstruction**

G4VUserPhysicsList

G4VUserActionInitialization

**Global:** only one instance exists in memory, shared by all threads.

## At execution

G4VUserPrimaryGeneratorAction

*G4UserRunAction*

*G4UserEventAction*

*G4UserStackingAction*

*G4UserTrackingAction*

*G4UserSteppingAction*

**Thread-local:** an instance of each action class exists for each thread.

# Note: Geant4 basic types

---

- Aliases for the primitive data types to provide cross-platform compatibility:
  - **G4double**, **G4float**, **G4int**, **G4bool**, **G4long**
- Enhanced version of string called **G4String**
  - inherits from **std::string**  $\Rightarrow$  all methods and operators
  - several **additional methods**
- **G4ThreeVector** is a three-component class corresponding to a real physics vector (example later)

**G4ThreeVector** dimensions {1.0, 2.0, 3.0};

Please, use these types for best compatibility (e.g. **G4int** instead of **int**, etc., **G4ThreeVector** when it makes sense etc.)



# Part I: Units and Materials

- *System of units & constants*
  - *Definition of elements*
  - *Materials and mixtures*
    - *NIST database*

# Units in Geant4

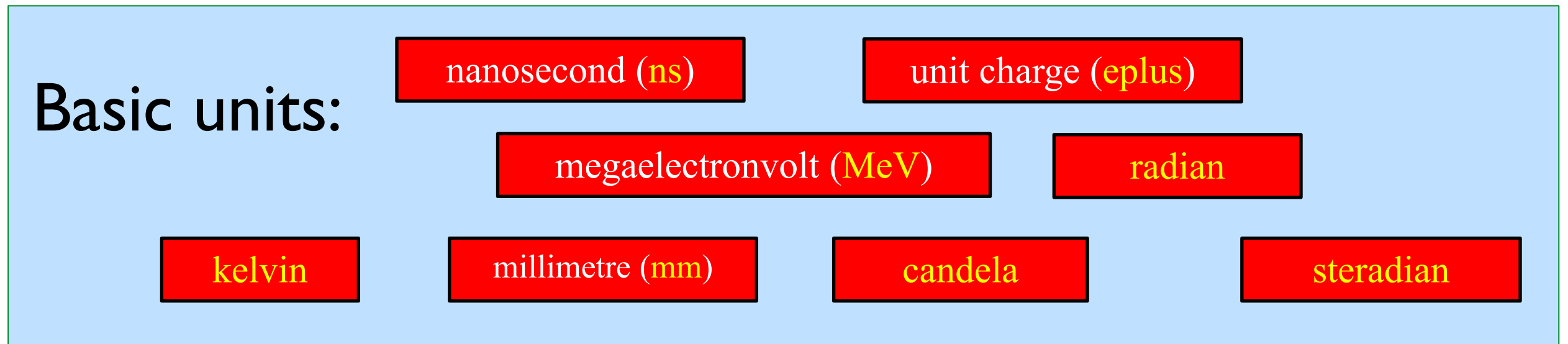
- Don't use default units!
- When specifying dimensions, always multiply by an appropriate unit:

```
G4double width = 12.5 * m;  
G4double density = 2.7 * g/cm3;
```

- Most common units are defined in CLHEP library (included in Geant4):
  - ▶ G4SystemOfUnits.hh
  - ▶ CLHEP/SystemOfUnits.hh
- You can define new units (not shown here)
- Output data in terms of a specific unit:
  - divide a value by the unit

```
G4cout << dE / MeV << " (MeV)" << G4endl;
```

# System of units



- All other units derived from the basic ones.
- **Useful feature:** Geant4 can select the most appropriate unit to use
  - specify the *category* for the data ([Length](#), [Time](#), [Energy](#), etc...):

```
G4cout << G4BestUnit(StepSize, "Length");
```

- **StepSize** will be printed in **km**, **m**, **mm** or ... **fermi**, depending on its actual value

# Materials

---

- Different levels of material description:
  - isotopes → **G4Isotope**
  - elements → **G4Element**
  - molecules, compounds and mixtures → **G4Material**
- Attributes associated:
  - temperature, pressure, state, **density**
- **G4Isotope** and **G4Element** describe properties of the atoms:
  - Atomic number, number of nucleons, mass of a mole, shell energies, cross-sections per atoms, etc...
- **G4Material** describes the **macroscopic** properties of the matter:
  - Temperature, pressure, state, density
  - Radiation length, absorption length, etc...
- **G4Material** is used by tracking, geometry and physics

# Making elements

- **Isotopes** can be assembled into elements

```
G4Isotope (const G4String& name,  
          G4int      z,      // atomic number  
          G4int      n,      // number of nucleons  
          G4double   a );   // mass of mole
```

Not number of neutrons!

Do not forget unit  
(g/mole)

- ... **building** elements as follows:

```
G4Element (const G4String& name,  
          const G4String& symbol, // element symbol  
          G4int      nIso ); // n. of isotopes  
G4Element::AddIsotope (G4Isotope* iso, // isotope  
                      G4double relAbund); // fraction of atom per volume
```

- Otherwise, create **G4Element** with **natural isotopic abundance**:

```
G4Element (const G4String& name,  
          const G4String& symbol,  
          G4int      z,      // atomic number  
          G4double   a );   // mass of mole
```

Do not forget unit  
(g/mole)



# Elements and compounds

- **Single-element material**

```
G4double z, a, density;  
density = 1.390*g/cm3;  
a = 39.95*g/mole;  
G4Material* lAr = new G4Material("liquidAr", z=18, a, density);
```

- **Molecule material** (composition by number of atoms):

```
a = 1.01*g/mole;  
G4Element* elH = new G4Element("Hydrogen", symbol="H", z=1., a);  
  
a = 16.00*g/mole;  
G4Element* elO = new G4Element("Oxygen", symbol="O", z=8., a);  
  
density = 1.000*g/cm3;  
G4Material* H2O = new G4Material("Water", density, ncomponents=2);  
H2O->AddElement(elH, natoms=2);  
H2O->AddElement(elO, natoms=1);
```

# Mixtures

- Composition by fraction of mass

```
a = 14.01*g/mole;  
G4Element* e1N = new G4Element(name="Nitrogen",symbol="N", z= 7., a);  
a = 16.00*g/mole;  
G4Element* e1O = new G4Element(name="Oxygen",symbol="O", z= 8., a);  
density = 1.290*mg/cm3;  
G4Material* Air = new G4Material(name="Air", density, ncomponents=2);  
Air->AddElement(e1N, 70.0*perCent);  
Air->AddElement(e1O, 30.0*perCent);
```

- Composition of mixtures

```
G4Element* e1C = ...; // define "carbon" element  
G4Material* SiO2 = ...; // define "quartz" material  
G4Material* H2O = ...; // define "water" material  
density = 0.200*g/cm3;  
  
G4Material* aerogel = new G4Material("Aerogel",  
                                   density, ncomponents=3);  
aerogel->AddMaterial(SiO2, fractionmass=62.5*perCent);  
aerogel->AddMaterial(H2O, fractionmass=37.4*perCent);  
aerogel->AddElement(e1C, fractionmass= 0.1*perCent);
```

# An example: a gas

- Necessary to specify **temperature** and **pressure**
- affect **dE/dx** calculations, thermal scattering

```
G4double density = 27. * mg/cm3;  
G4double temperature = 325. * kelvin;  
G4double pressure = 50. * atmosphere;  
  
G4Material* CO2 = new G4Material("CO2Gas", density,  
    ncomponents=2, kStateGas, temperature, pressure);  
CO2->AddElement(C, natoms = 1);  
CO2->AddElement(O, natoms = 2);
```

- **Absolute vacuum** does not exist: gas at very low  $\rho$  !
- **Cannot define materials with  $\rho=0$**

```
G4double rho = 1.e-25*g/cm3;  
G4double pr = 3.e-18*pascal;  
G4Material* Vacuum = new G4Material("interGalactic", Z, A, rho,  
    kStateGas, temperature, pr);
```

# NIST material database

- No need to predefine elements and materials
- Retrieve elements and materials from **NIST manager**:

```
G4NistManager* manager = G4NistManager::Instance();  
G4Material* H2O = manager->FindOrBuildMaterial("G4_WATER");  
G4Material* air = manager->FindOrBuildMaterial("G4_AIR");  
G4Material* vacuum = manager->FindOrBuildMaterial("G4_Galactic");  
G4Element* Si = manager->FindOrBuildElement("Si");
```

- UI commands:

```
/material/nist/printElement
```

← print defined elements

```
/material/nist/listMaterials
```

← print defined materials

```
▶ G4NistManager.hh
```

# NIST material database

- [NIST database](#) for materials is imported inside Geant4
- <http://physics.nist.gov/PhysRefData>
- [UI commands](#) specific for handling materials
- The best accuracy for the most relevant parameters guaranteed:
  - Density
  - Mean excitation potential
  - Chemical bounds
  - Element composition
  - Isotope composition
  - Various corrections

Z	A	m	error	(%)	A <sub>eff</sub>
14	Si	22	22.03453	(22)	28.0855(3)
		23	23.02552	(21)	
		24	24.011546	(21)	
		25	25.004107	(11)	
		26	25.992330	(3)	
		27	26.98670476	(17)	
		28	27.9769265327	(20)	92.2297 (7)
		29	28.97649472	(3)	4.6832 (5)
		30	29.97377022	(5)	3.0872 (5)
		31	30.97536327	(7)	
		32	31.9741481	(23)	
		33	32.978001	(17)	
		34	33.978576	(15)	
		35	34.984580	(40)	
		36	35.98669	(11)	
		37	36.99300	(13)	
		38	37.99598	(29)	
		39	39.00230	(43)	
		40	40.00580	(54)	
		41	41.01270	(64)	
		42	42.01610	(75)	

- **Natural isotope compositions**
- **More than 3000 isotope masses**

<http://geant4.cern.ch/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/apas08.html>

# NIST materials

- NIST elementary materials:
  - $H \rightarrow Cf$  ( $Z = 1 \rightarrow 98$ )
- NIST compounds:
  - e.g. “G4\_ADIPOSE\_TISSUE\_IRCP”
- HEP and Nuclear materials:
  - e.g. Liquid Ar, PbWO
- Possible to build mixtures of NIST and user-defined materials

```
#####  
### Elementary Materials from the NIST Data  
#####  
Z Name ChFormula density(g/cm^3) I(eV)  
#####  
1 G4_H H_2 8.3748e-05 19.2  
2 G4_He 0.000166322 41.8  
3 G4_Li 0.534 40  
4 G4_Be 1.848 63.7  
5 G4_B 2.37 76  
6 G4_C 2 81  
7 G4_N N_2 0.0011652 82  
8 G4_O O_2 0.00133151 95  
9 G4_F 0.00158029  
10 G4_Ne 0.000838505  
11 G4_Na 0.971
```

```
#####  
### Compound Materials from the NIST Data Base  
#####  
N Name ChFormula density(g/cm^3) I(eV)  
#####  
13 G4_Adipose_Tissue 0.92 63.2  
1 0.119477  
6 0.63724  
7 0.00797  
8 0.232333  
11 0.0005  
12 2e-05  
15 0.00016  
16 0.00073  
17 0.00119  
19 0.00032  
20 2e-05  
26 2e-05  
30 2e-05  
4 G4_Air 0.00120479 85.7  
6 0.000124  
7 0.755268  
8 0.231781  
18 0.012827  
2 G4_Csl 4.51 553.1  
53 0.47692  
55 0.52308
```



# Part II: Geometry

- *Basic concepts*
- *Implementation*
- *Tools for geometry checking*

# Describe your Detector

---

- A detector geometry is made of a number of volumes
- The largest volume is called **World** volume
  - It must contain **all other volumes**
- Derive your own concrete class from **G4VUserDetectorConstruction** abstract base class
- Implementing the pure virtual method **Construct ()** :
  - Define **shapes/solids** required to describe the geometry
  - Construct all necessary **materials**
  - **Construct** and **place** volumes of your detector geometry
  - (Define "**sensitivity**" properties associated to volumes)
  - (Associate **magnetic field** to detector regions)
  - (Define **visualization** attributes for the detector elements)



# Geometry - implementation basics

- Implement a class inheriting from the abstract base class **G4VUserDetectorConstruction**:

```
class MyDetector : public G4VUserDetectorConstruction {
public:
    virtual G4VPhysicalVolume* Construct();           // required

    virtual void ConstructSDAndField();              // optional
    // ...
};
```


- Create an instance in the main program:

```
MyDetector* detector = new MyDetector();
runManager->SetUserInitialization(detector);
```

- **Note**: Split the implementation into more classes and methods! (good programming practice)
  - especially for complex geometries!
- **Note2**: you should not delete the **MyDetector** instance! Run manager does that automatically.

# G4VUserDetectorConstruction

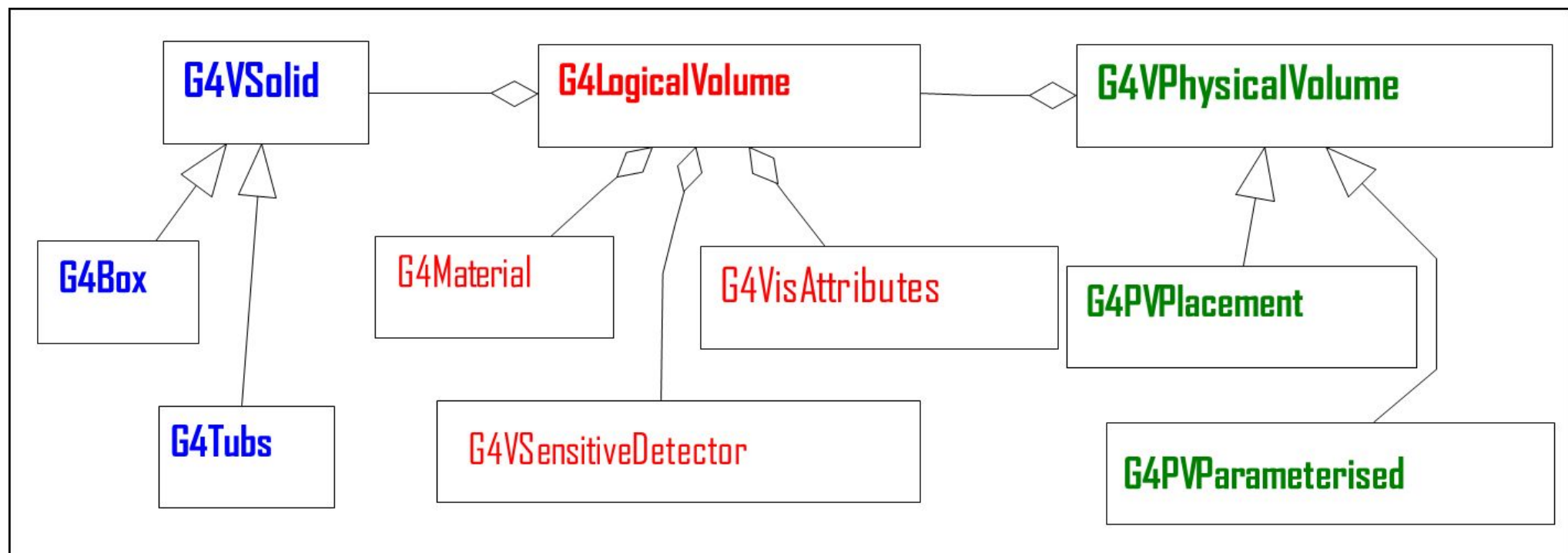
---

- Method **Construct()**
  - Define **materials**
  - Define **solids** and **volumes** of the geometry
  - Build the **tree hierarchy** of volumes
  - Define **visualization** attributes
  - Return the **world physical volume!** 
- Method **ConstructSDAndField()**
  - Assign **magnetic field** to volumes / regions
  - Define **sensitive detectors** and assign them to volumes

► `G4VUserDetectorConstruction.hh`

# Three Conceptual Layers

- **G4VSolid**
  - *Shape, size*
- **G4LogicalVolume**
  - *Hierarchy of volumes, material, sensitivity, magnetic field*
- **G4VPhysicalVolume**
  - *Position, rotation. The same logical volume can be placed many times (repeated modules)*

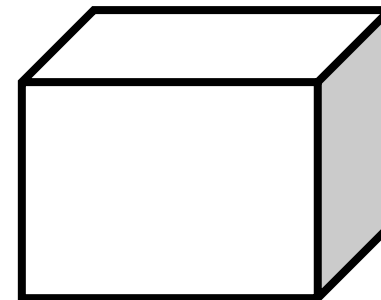


# Define Detector Geometry

- Basic strategy

```
G4VSolid* pBoxSolid =  
    new G4Box("aBoxSolid",  
             1.*m, 2.*m, 3.*m);
```

Solid: shape and size.



**Step 1**  
Create the  
geom. object:  
box

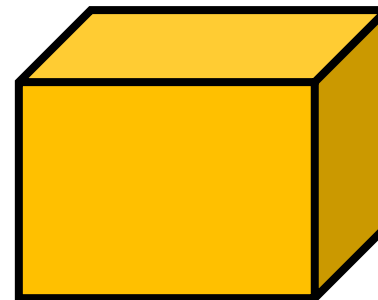
# Define Detector Geometry

- Basic strategy

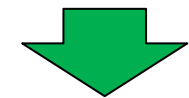
Logical volume : + material, sensitivity, etc.

```
G4VSolid* pBoxSolid =  
    new G4Box("aBoxSolid",  
             1.*m, 2.*m, 3.*m);
```

```
G4LogicalVolume* pBoxLog =  
    new G4LogicalVolume( pBoxSolid,  
                         pBoxMaterial, "aBoxLog", 0, 0, 0);
```



**Step 1**  
Create the  
geom. object:  
box

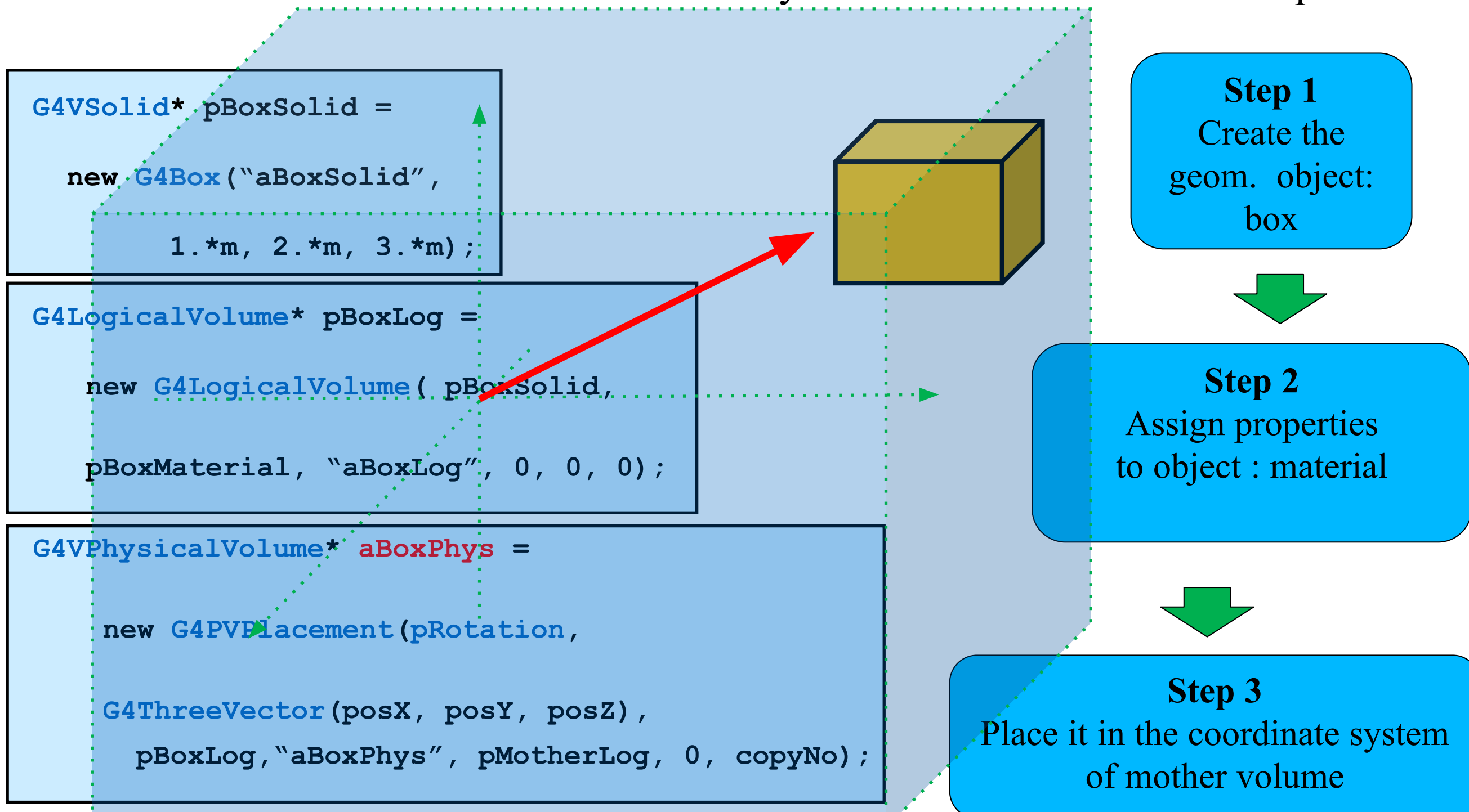


**Step 2**  
Assign properties  
to object : material

# Define Detector Geometry

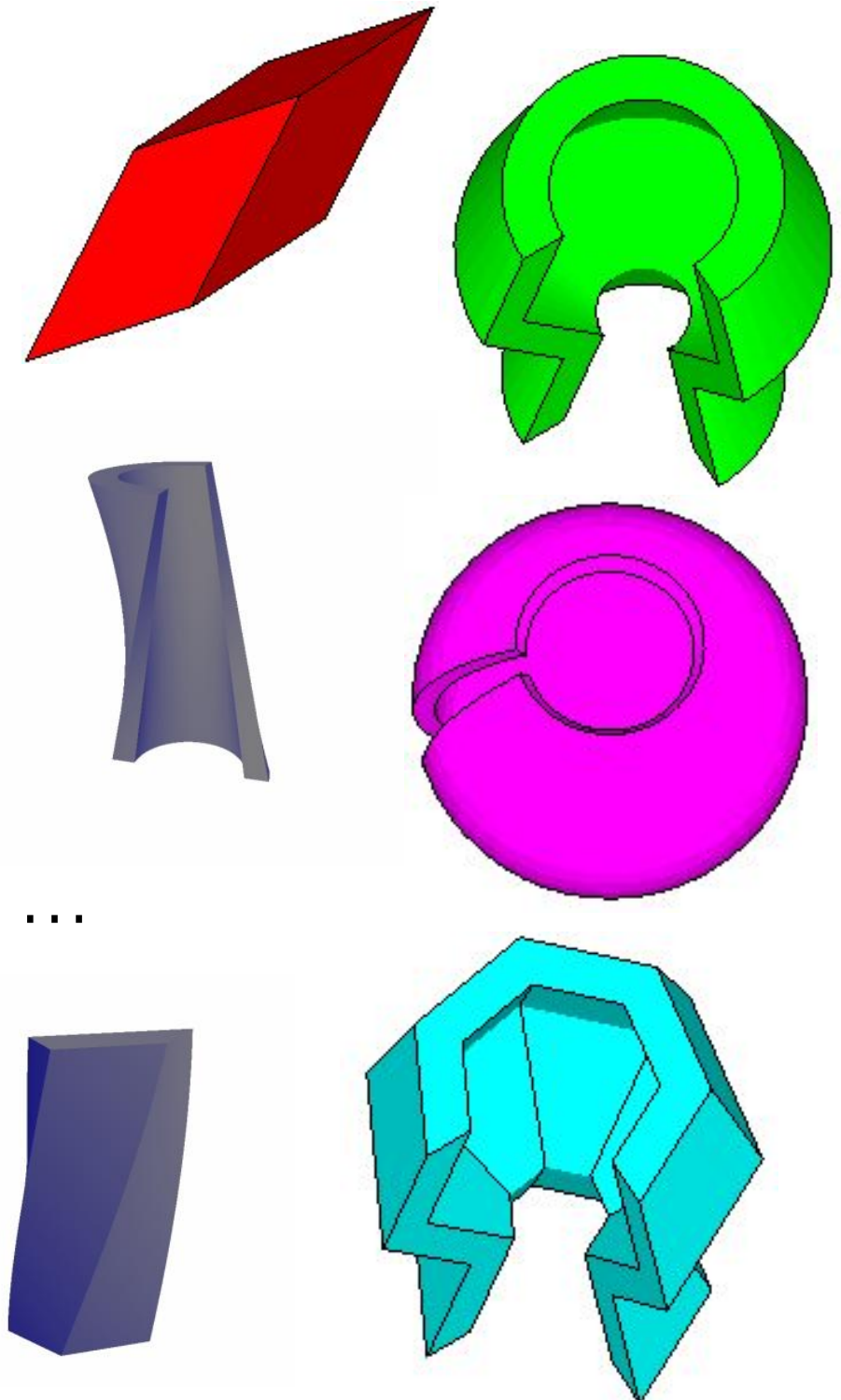
- Basic strategy

Physical volume : + rotation and position



# Solids

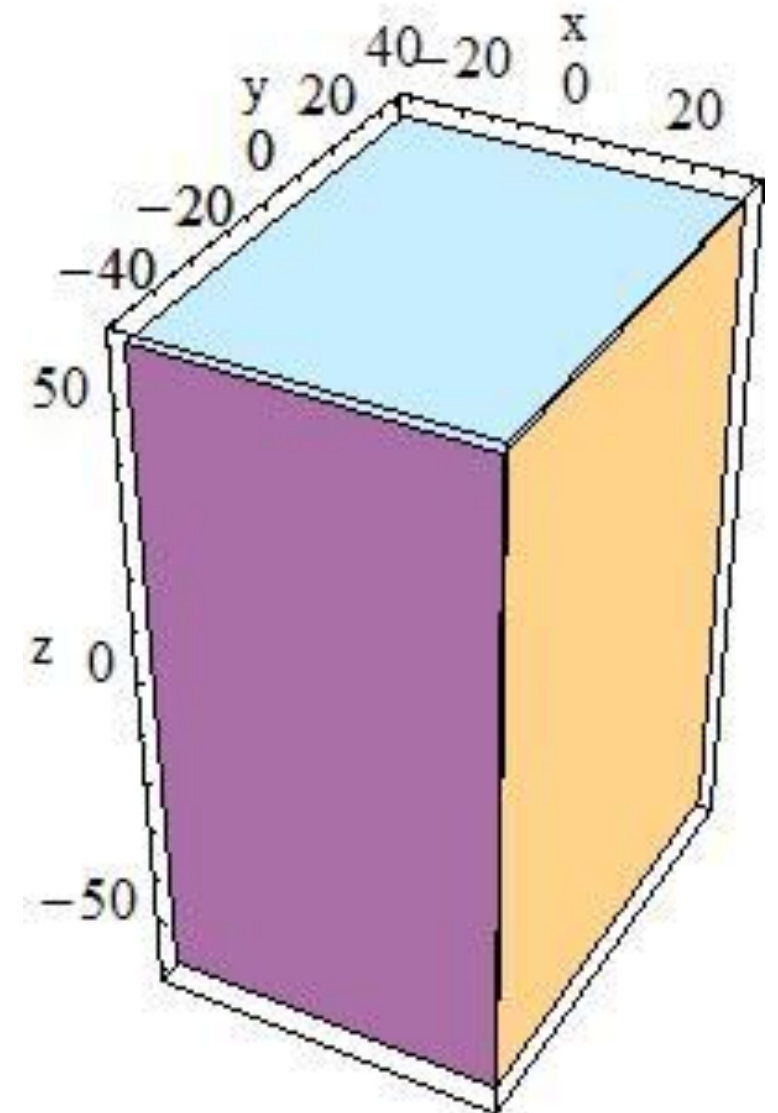
- **CSG** (Constructed Solid Geometry) solids
  - G4Box, G4Tubs, G4Cons, G4Trd, ...
  - Analogous to simple GEANT3 CSG solids
- **Specific** solids (CSG like)
  - G4Polycone, G4Polyhedra, G4Hype, ...
  - G4TwistedTubs, G4TwistedTrap, ...
- **BREP** (Boundary REPresented) solids
  - G4BREPSolidPolycone, G4BSplineSurface, ...
  - Any order surface
- **Boolean** solids
  - G4UnionSolid, G4SubtractionSolid, ...



# CSG: G4Box

```
G4Box (const G4String& pname, // name
       G4double pX, // half-length in X
       G4double pY, // half-length in Y
       G4double pZ, // half-length in Z) ;
```

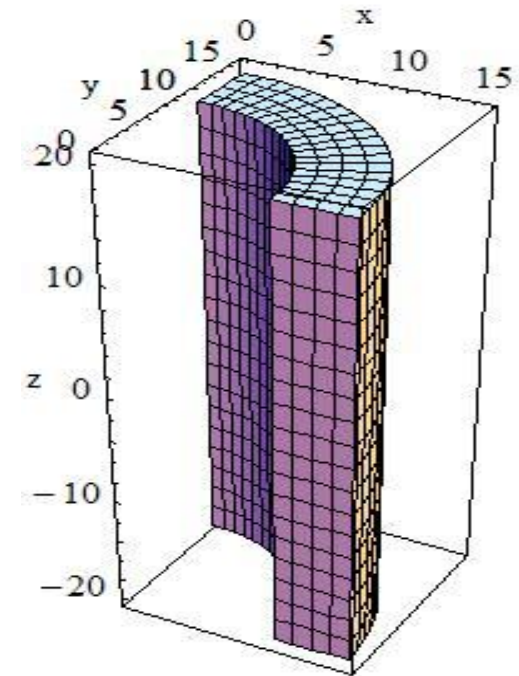
Note the half-length!



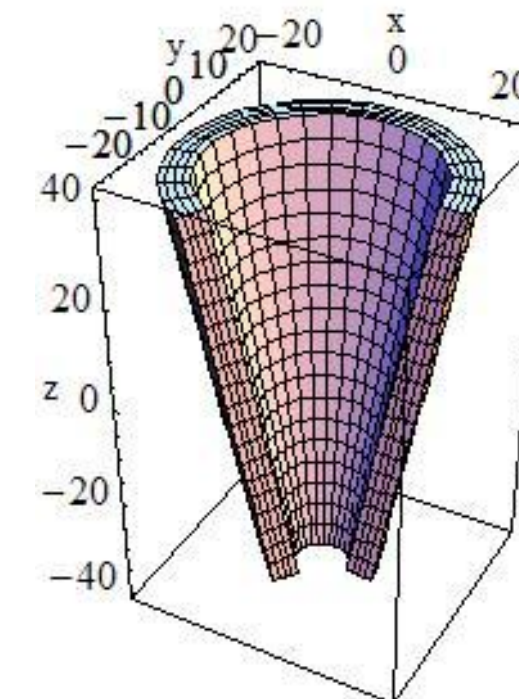


# CSG: G4Tubs & G4Cons

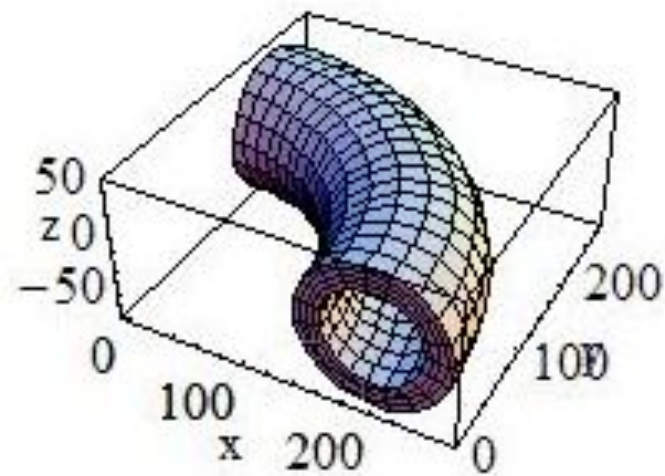
```
G4Tubs (const G4String& pname, // name
        G4double pRmin, // inner radius (0)
        G4double pRmax, // outer radius
        G4double pDz, // Z half! length
        G4double pSphi, // starting Phi (0)
        G4double pDphi); // segment angle (twopi)
```



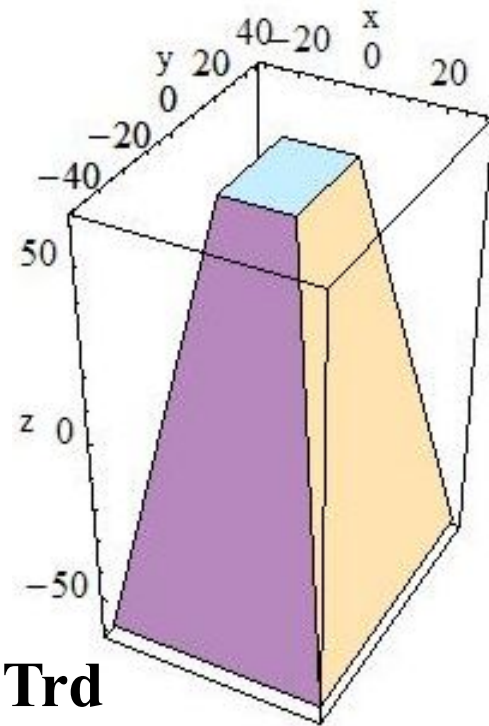
```
G4Cons (const G4String& pname, // name
        G4double pRmin1, // inner radius -pDz
        G4double pRmax1, // outer radius -pDz
        G4double pRmin2, // inner radius +pDz
        G4double pRmax2, // outer radius +pDz
        G4double pDz, // Z half length
        G4double pSphi, // starting Phi
        G4double pDphi); // segment angle
```



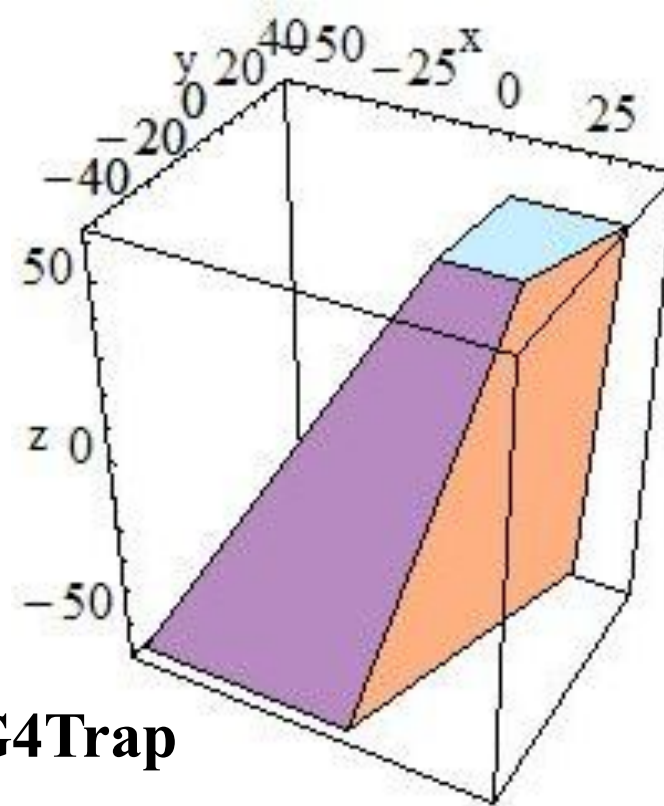
# Other CSG Solids



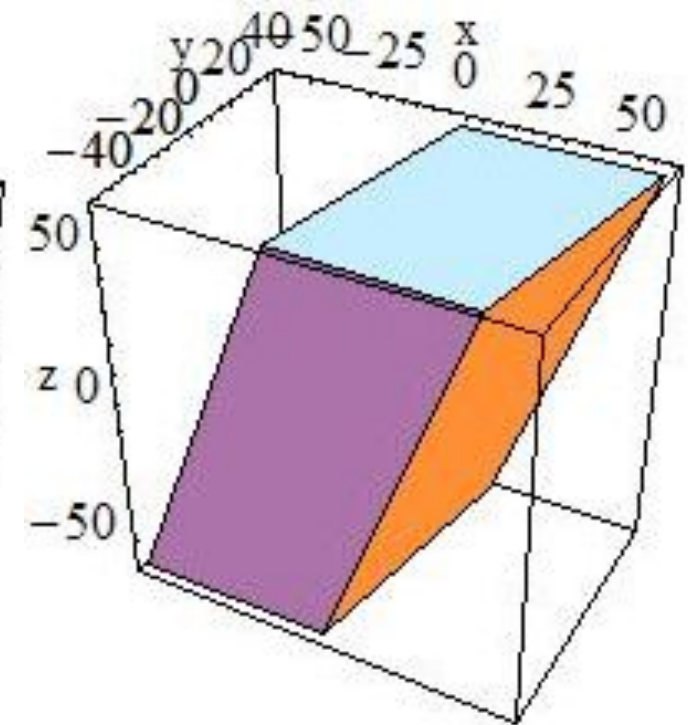
**G4Torus**



**G4Trd**

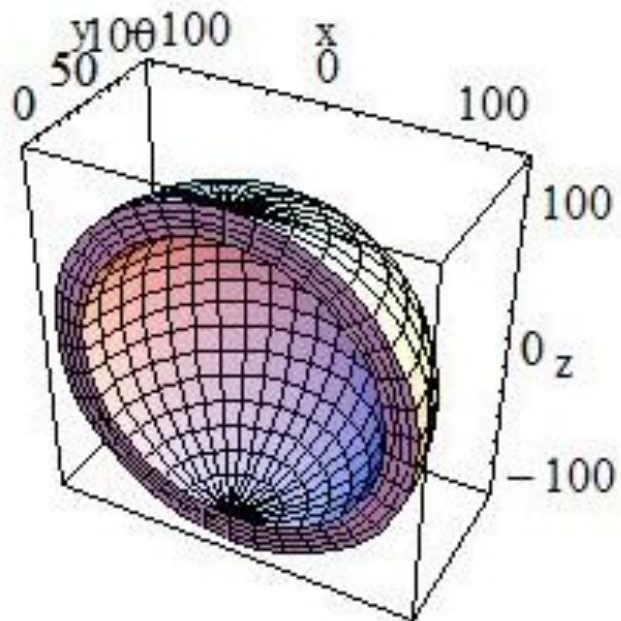


**G4Trap**

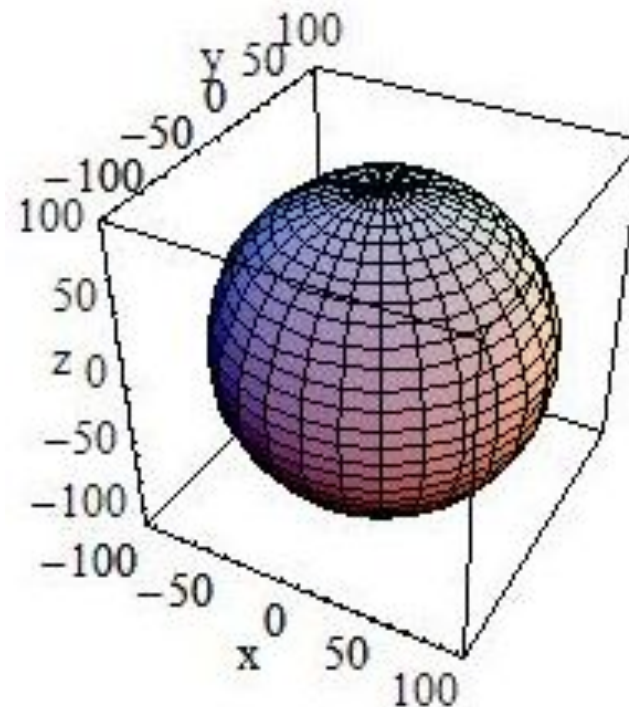


**G4Para**  
(parallelepiped)

**G4Sphere**



**G4Orb**  
(full solid  
sphere)

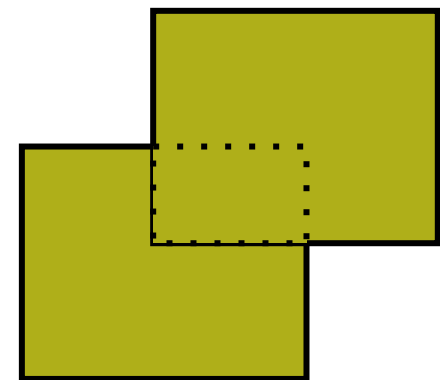


Check [Section 4.1.2](#) of Geant4 Application Developers Guide for all available shapes

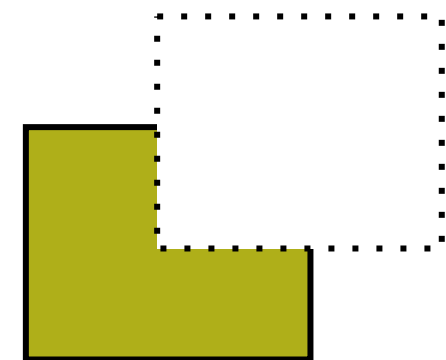
# Boolean Solids

- Solids can be combined using **boolean operations**:
  - **G4UnionSolid**, **G4SubtractionSolid**, **G4IntersectionSolid**
  - Requires: 2 solids, 1 boolean operation, and an (optional) transformation for the 2<sup>nd</sup> solid
  - 2<sup>nd</sup> solid is positioned relative to the coordinate system of the 1<sup>st</sup> solid
  - Result of boolean operation becomes a solid → re-usable in a boolean operation
- Solids to be combined can be either CSG or other Boolean solids
- **Note**: tracking cost for the navigation in a complex Boolean solid is proportional to the number of constituent CSG solids

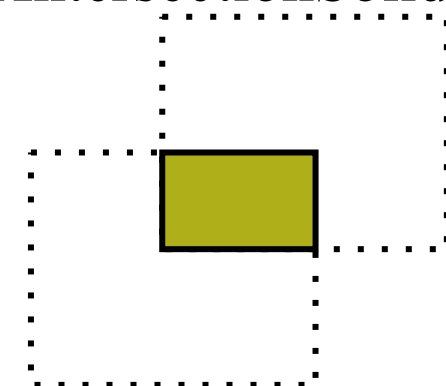
G4UnionSolid



G4SubtractionSolid



G4IntersectionSolid



# Boolean Solids - an Example

```
G4VSolid* box = new G4Box("Box", 50*cm, 60*cm, 40*cm);
G4VSolid* cylinder =
    new G4Tubs("Cylinder", 0., 50.*cm, 50.*cm, 0., twopi);

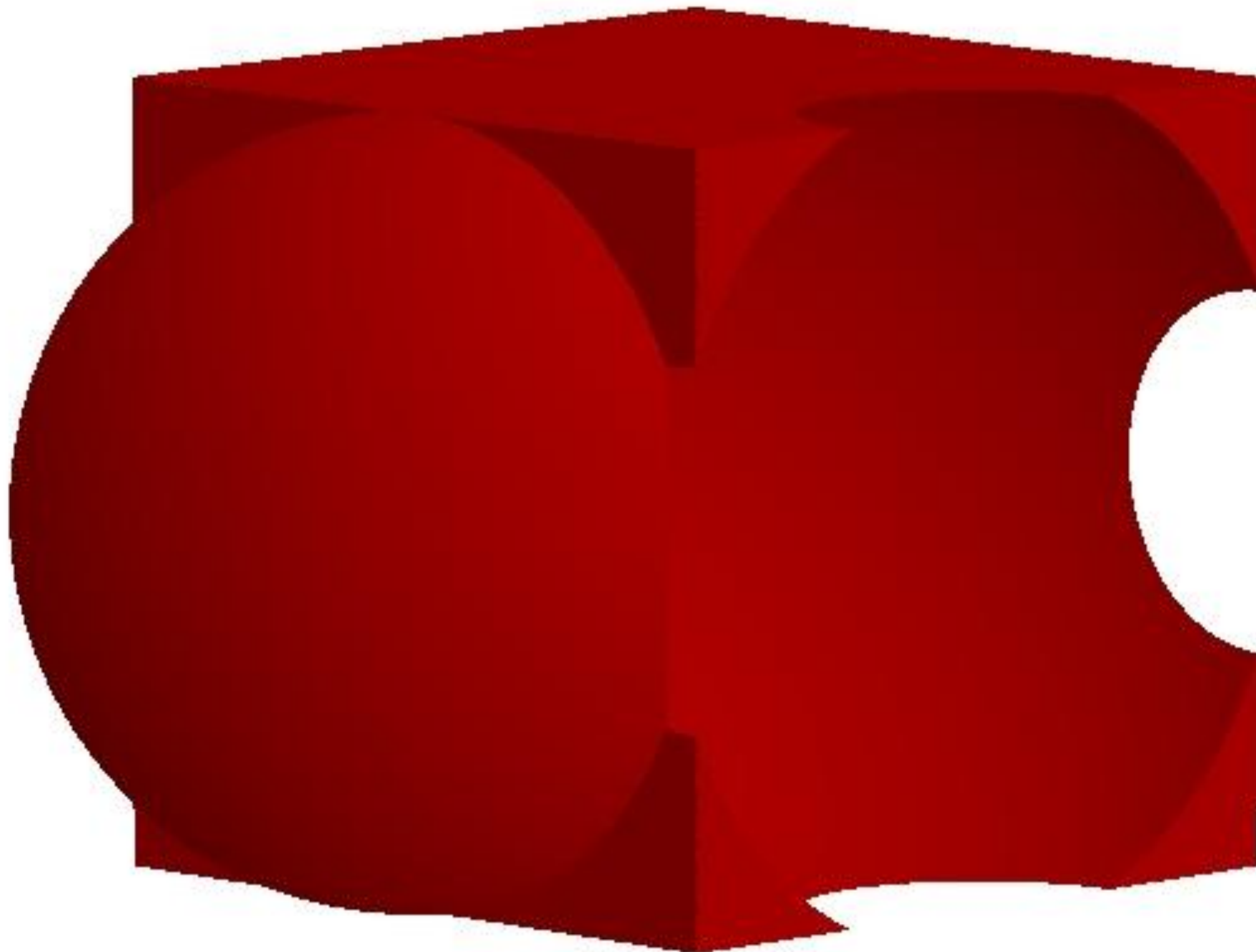
G4VSolid* union =
    new G4UnionSolid("Box+Cylinder", box, cylinder);

G4VSolid* subtract =
    new G4SubtractionSolid("Box-Cylinder", box, cylinder,
        0, G4ThreeVector(30.*cm, 0., 0.));

G4RotationMatrix* rm = new G4RotationMatrix();
rm->RotateX(30.*deg);
G4VSolid* intersect =
    new G4IntersectionSolid("Box&&Cylinder",
        box, cylinder, rm, G4ThreeVector(0., 0., 0.));
```

# Boolean Solid - Example

---



# Logical Volumes

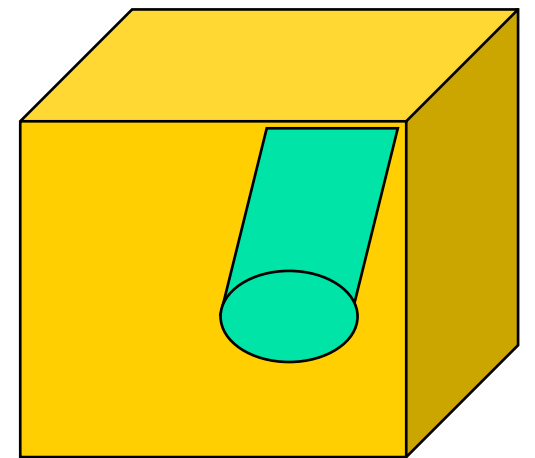
- Contains **all information** of volume **except position**:
  - **Shape** and **dimension** (**G4VSolid**)
  - **Material**, sensitivity, visualization attributes
  - Position of **daughter** volumes
  - **Magnetic** field, User limits
- **Physical volumes** of same type **can share** a logical volume.
- The pointers to solid and material **must be not nullptr**

```
G4LogicalVolume (G4VSolid* pSolid,  
                 G4Material* pMaterial,  
                 const G4String& name,  
                 G4FieldManager* pFieldMgr=0,  
                 G4VSensitiveDetector* pSDetector=0,  
                 G4UserLimits* pULimits=0,  
                 G4bool optimise=true);
```

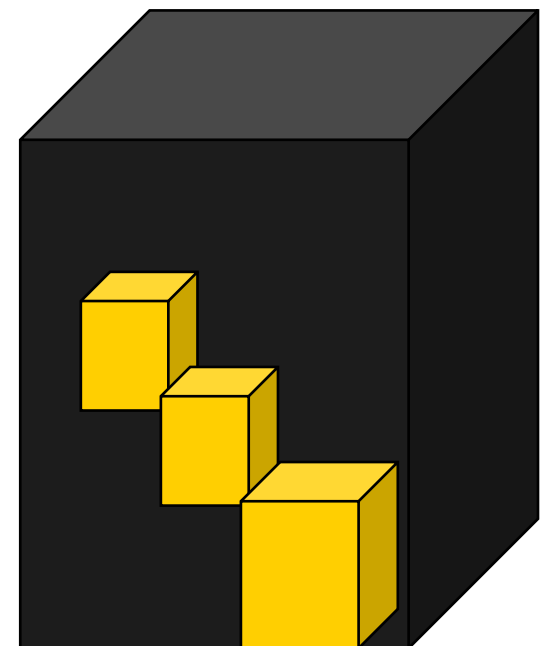
} optional

# Physical Volumes

- A **physical volume** is a positioned instance of a **logical volume** inside **another logical volume** (the mother volume)
- Placement (**G4PVPPlacement**)
  - it is one positioned volume
- Repeated: a volume placed many times
  - can represent any number of volumes
  - reduces use of **memory**
  - **G4PVReplica** (= simple repetition)
  - **G4PVParameterised** (= more complex pattern)
    - **G4PVDivision**



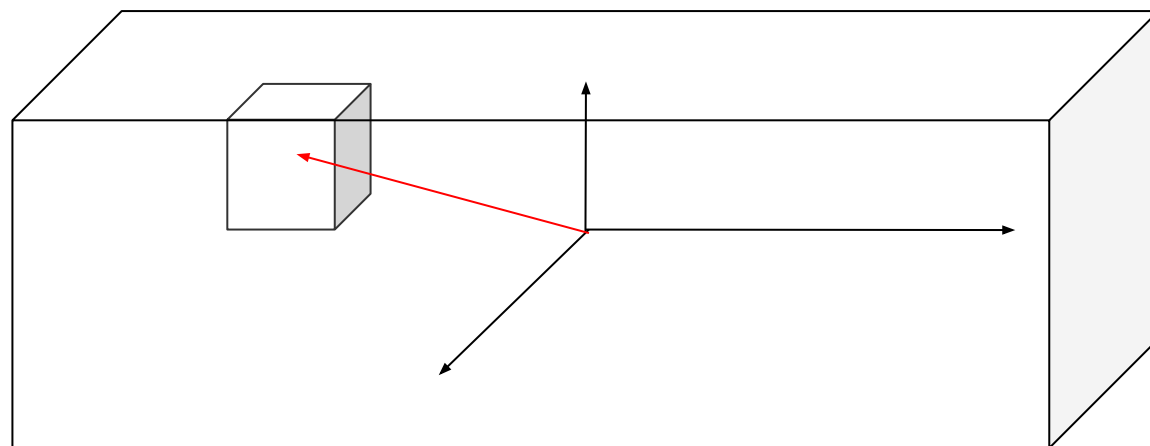
*placement*



*repeated*

# Geometry Hierarchy

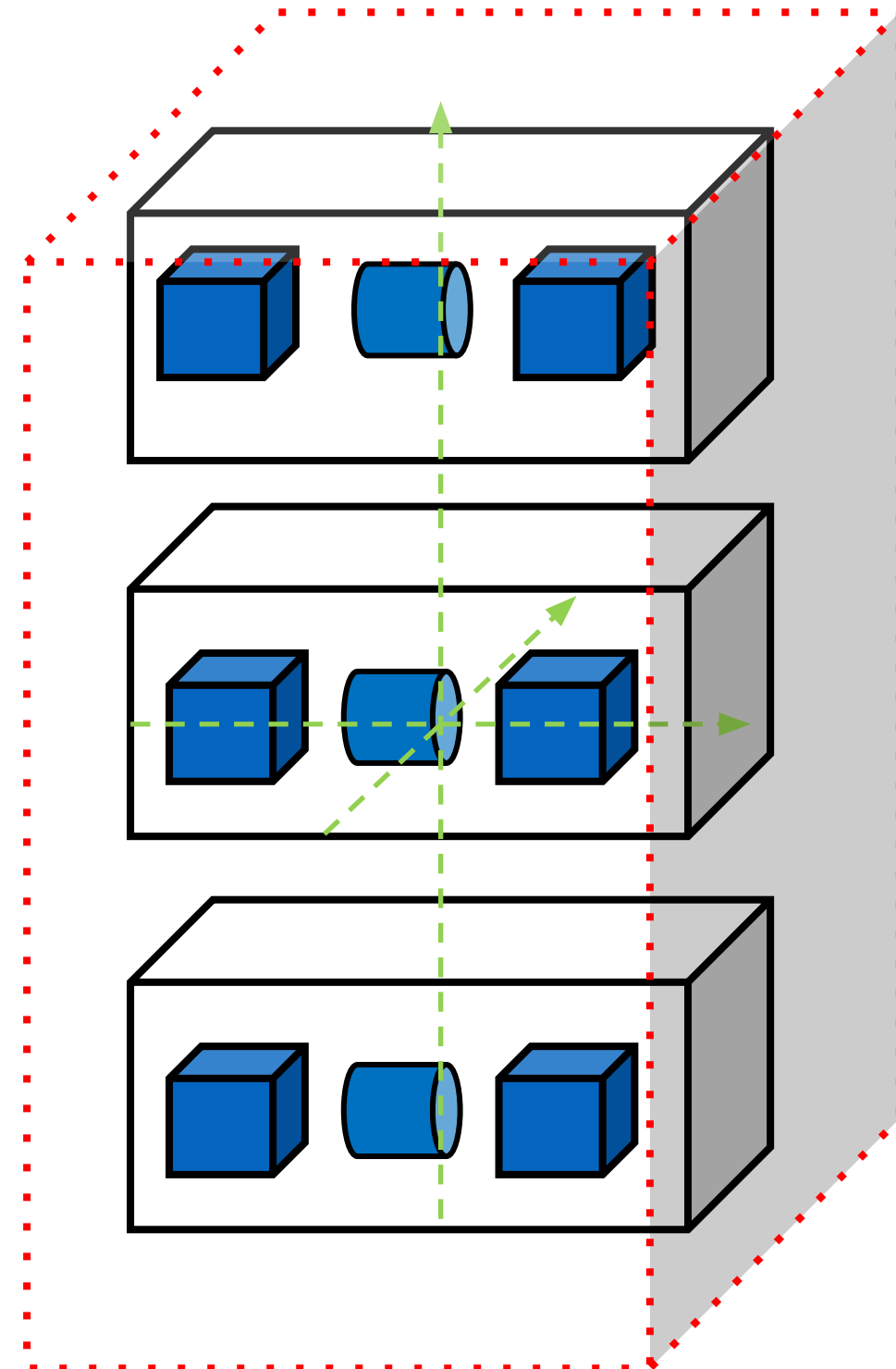
- A volume is **placed** in its **mother volume**
- **Position and rotation** of the daughter volume is described with respect to the **local coordinate system** of the mother volume
- The **origin** of the mother's local coordinate system is at the **center** of the mother volume
- Daughter volumes **cannot protrude** from the mother volume
- Daughter volumes **cannot overlap**
- The **logical volume** of mother **knows** the daughter volumes it contains
- It is **uniquely** defined to be their mother volume





# Geometry Hierarchy

- One logical volume can be placed **more than once**. One or more volumes can be placed in a mother volume
- The **mother-daughter relationship** is an information of **G4LogicalVolume**
- If the **mother volume is placed more than once**, **all daughters** by definition appear in each placed physical volume
- The **world volume** must be a unique physical volume which **fully contains** all other volumes (root volume of the hierarchy)
- The world volume defines the **global coordinate system**. The **origin of the global coordinate system** is at the center of the world volume
- **Position of a track** is given with respect to the **global coordinate system**



# G4PVPlacement

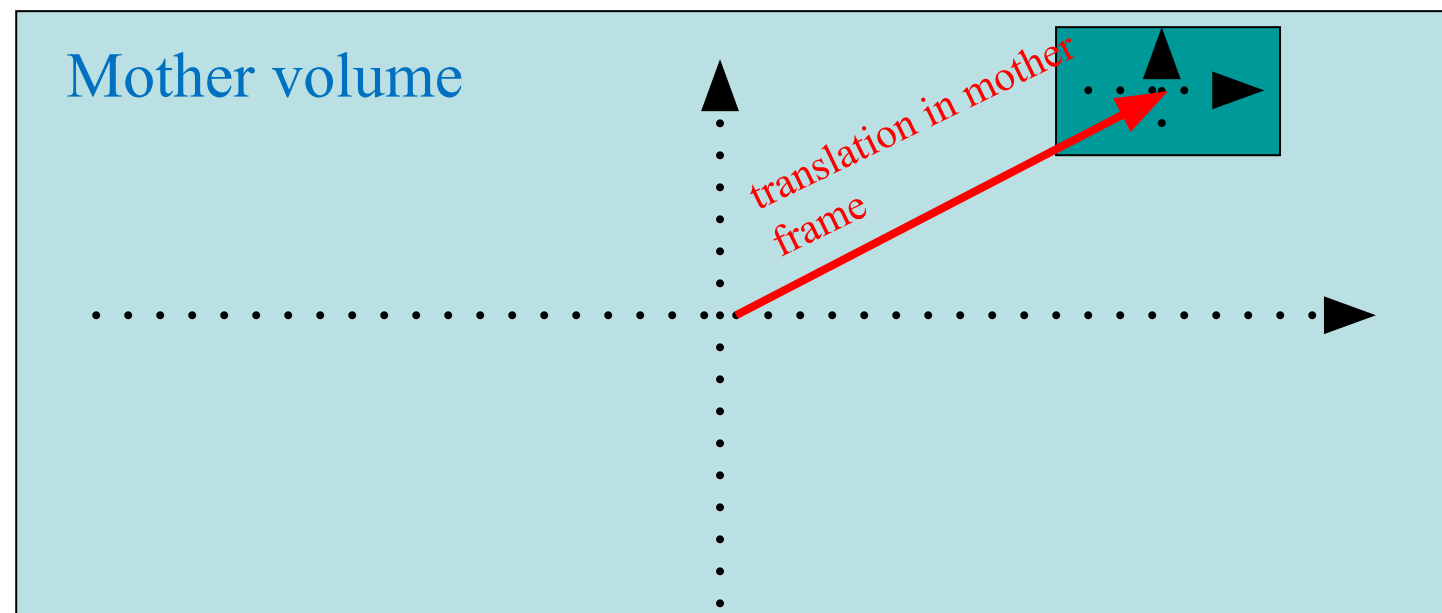
---

- **Single volume** positioned **relatively** to the **mother volume**
- In a frame **rotated** and **translated** relative to the coordinate system of the **mother volume**
- A **few variants** available:
  - Using **G4Transform3D** to represent the direct rotation and translation of the solid instead of the frame (*alternative constructor*)
  - specifying the mother volume as a **pointer** to its **physical volume** instead of its **logical volume**
- **Four constructors** available
  - logical OR physical volume as mother
  - active OR passive transformation of the coordinate system

# G4PVPlacement: Rotation **of** mother frame ...

- Single volume positioned relatively to the mother volume (**passive transformation**)

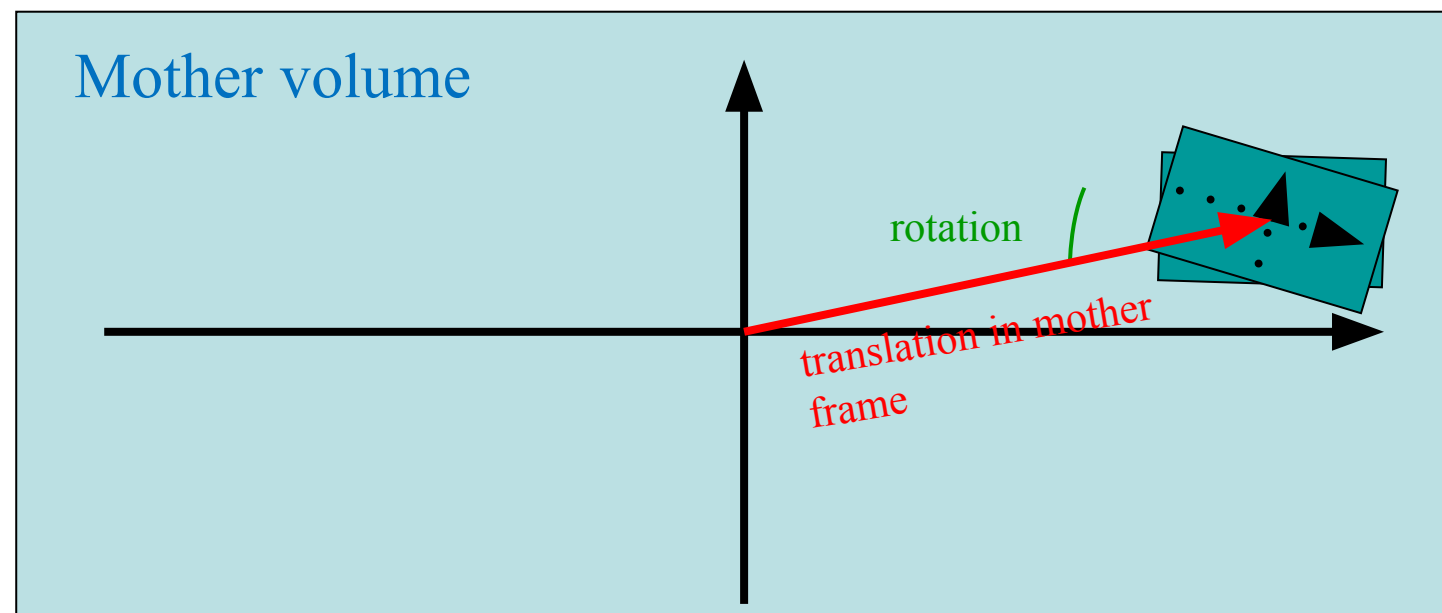
```
G4PVPlacement(G4RotationMatrix* pRot,          // rotation of mother frame
              const G4ThreeVector& tlate,     // position in mother frame
              G4LogicalVolume* pCurrentLogical,
              const G4String& pName,
              G4LogicalVolume* pMotherLogical,
              G4bool pMany,                   // not used. Set it to false..
              G4int pCopyNo,                 // unique arbitrary index
              G4bool pSurfChk=false ); // optional overlap check
```



# G4PVPlacement: Rotation **in** mother frame ...

- Single volume positioned relatively to the mother volume (**active transformation**)

```
G4PVPlacement (G4Transform3D (
    G4RotationMatrix &pRot,          // rotation in daughter frame
    const G4ThreeVector &tlate),    // position in mother frame
    G4LogicalVolume *pDaughterLogical,
    const G4String &pName,
    G4LogicalVolume *pMotherLogical,
    G4bool pMany,                    // not used, set it to false...
    G4int pCopyNo,                  // unique arbitrary integer
    G4bool pSurfChk=false );       // optional overlap check
```



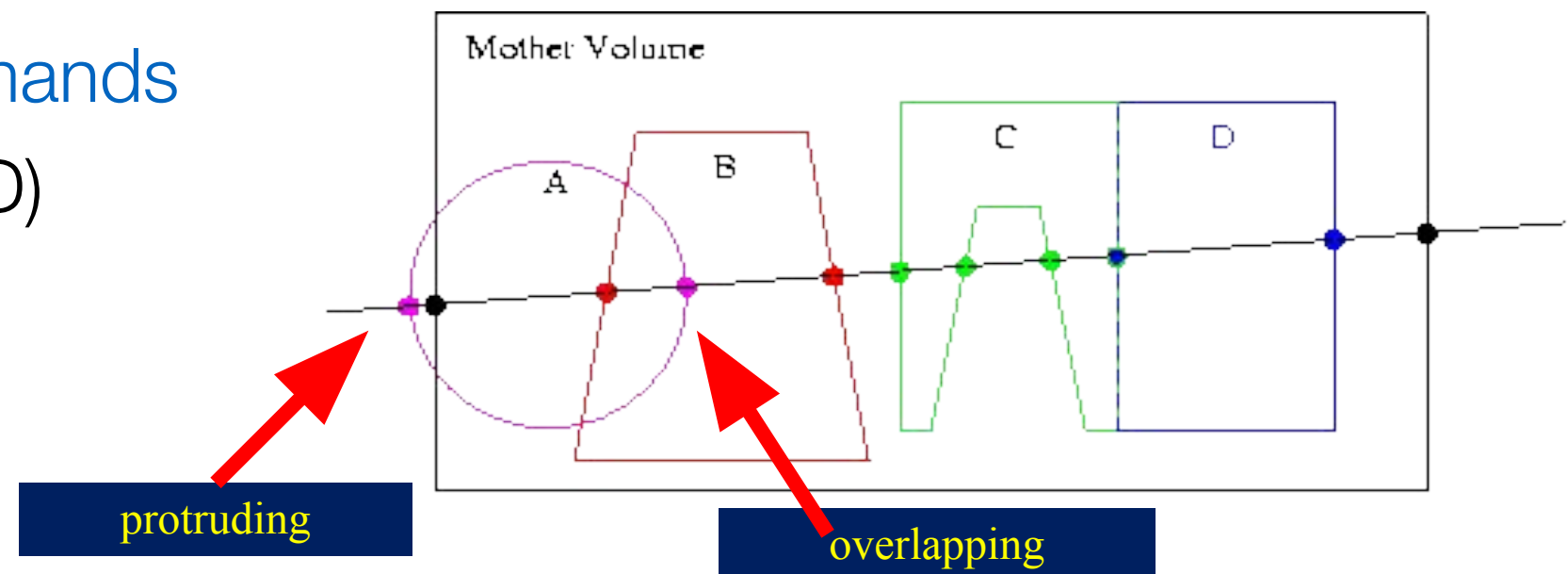
# Bird's eye on repeated volumes

---

- Placement volume (**G4PVPPlacement**): one positioned volume
  - One physical volume represents one "real" volume
- Repeated volume: a volume placed many times
  - One physical volume represents any number of "real" volumes
    - Reduced *use of memory*
    - Very convenient for *large voxelized* geometries
  - Parametrized (repetitions w.r.t. copy number)
    - **G4VPVParameterisation**
  - Replicas and Divisions (**G4PVReplica**, **G4PVDivision**)
- **Note**: a repeated volume is not equivalent to a loop of placements
  - All placements of the loop exists *individually* in the *memory*

# Geometry Problems

- Geant4 **does not allow** for malformed geometries, neither **protruding** (daughter/mother) not **overlapping** (sisters)
- The behavior of navigation is **unpredictable** for such cases
- The problem of **detecting overlaps** between volumes is bounded by the complexity of the solid models description
- **Utilities** are provided for detecting wrong positioning
- **Optional checks** at construction
- Kernel **run-time commands**
- **Graphical** tools (DAVID)



# Tools for Geometry Check

- Constructors of **G4PVPlacement** and **G4PVParameterised** have an optional argument “**pSurfChk**”

```
G4PVPlacement (G4RotationMatrix* pRot,  
               const G4ThreeVector &tlate,  
               G4LogicalVolume *pDaughterLogical,  
               const G4String &pName,  
               G4LogicalVolume *pMotherLogical,  
               G4bool pMany, G4int pCopyNo,  
               G4bool pSurfChk=false);
```

- If this flag is **true**, **overlap check** is done at the construction
- Some number of points are **randomly sampled** on the surface of creating volume
- This check **requires lots of CPU time**, but it is worth to **try** at least **once**
- **Built-in run-time commands** to activate verification tests for the user geometry:
  - **/geometry/test/run** or **/geometry/test/grid\_test**
  - start verification of geometry for overlapping regions based on a **standard grid setup**, limited to the first depth level
  - **/geometry/test/recursive\_test** for all depth levels (CPU intensive!)

# Tools for Geometry Check

Source

```
void MyDetectorConstruction::CheckOverlaps()
{
    G4PhysicalVolumeStore* thePVStore = G4PhysicalVolumeStore::GetInstance();
    G4cout << thePVStore->size() << " physical volumes are defined" << G4endl;
    G4bool overlapFlag = false;
    G4int res = 1000;
    G4double tol = 0.; // tolerance
    for (size_t i=0; i<thePVStore->size(); i++)
    {
        overlapFlag = (*thePVStore)[i]->CheckOverlaps(res,tol,fCheckOverlapsVerbosity)
            | overlapFlag;
    }
    if (overlapFlag)
        G4cout << "Check: there are overlapping volumes" << G4endl;
}
```

Output

```
Checking overlaps for volume BeamLineSupport ... OK!
Checking overlaps for volume BeamLineCover ... OK!
Checking overlaps for volume BeamLineCover2 ... OK!
Checking overlaps for volume VacuumZone ... OK!
Checking overlaps for volume FirstScatteringFoil ... OK!
.....
----- WWWWW ----- G4Exception-START ----- WWWWW -
*** G4Exception : GeomVol1002 issued by : G4PVPlacement::CheckOverlaps()
Overlap with volume already placed !
    Overlap is detected for volume BrassTube2
    with HoleNozzleSupport volume's
    local point (12.6381,12.8171,-25.1867), overlapping by at least: 3.5 mm
*** This is just a warning message ***
----- WWWWW -----
```

This method can be called at any point after run-



# Regions

---

- A region is a **sub-set** of the geometry
- It may have its **specific**
  - Production **thresholds** (cuts)
  - User **limits**
    - Artificial limits affecting to the tracking, e.g. max step length, max number of steps, min kinetic energy left, etc.
  - **Field** manager
- **World** logical volume is recognized as **the default region**. User is not allowed to define a region to the world logical volume

# ====> Task1

Link to the Tasks:

<http://geant4.lns.infn.it/alghero2022/task1/task1a.html>

## Task 1 - Geometry

- Defining and using materials
- Constructing a volume using solids, logical and physical volumes
- Magnetic fields