FLUKA

Advanced geometry

Advanced settings, transformations and geometry replication (LATTICE)

Overview

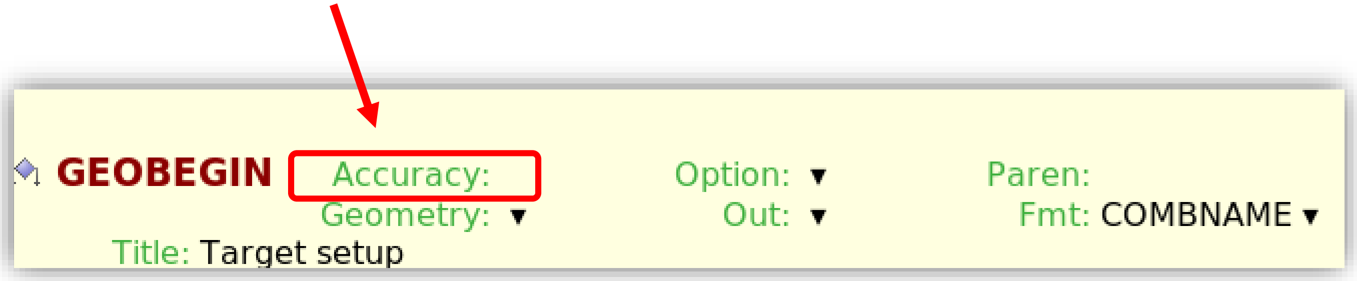
- Some less known details
  - The `GEOBEGIN` card, parentheses, adjacent zones
- Refresher: transformations and the `ROT-DEFI` card
- Geometry replication: `LATTICE`
  - Using replicas
  - Scoring with lattices
  - The `lattic.f` user routine and auxiliary routines

Geometry: some less known details

The GEOBEGIN card | Parentheses | Regions: adjacent zones

GEOBEGIN card: tracking accuracy

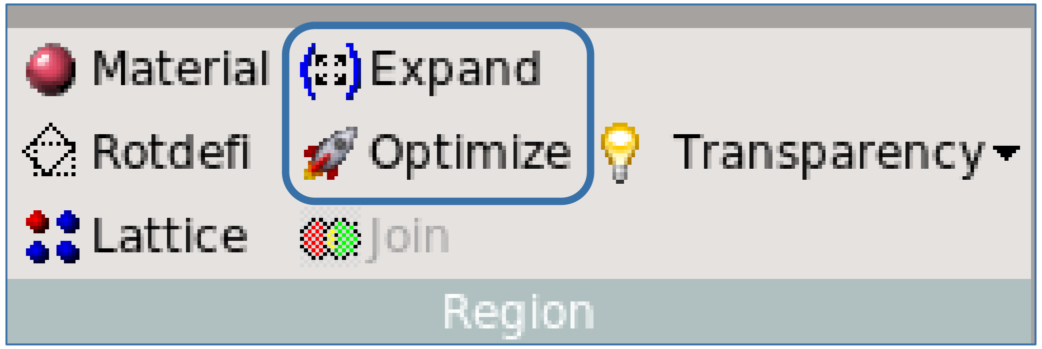
- The tracking accuracy parameter can be adjusted in the **GEOBEGIN** card
  - Defines absolute accuracy used for tracking and boundary identification (**in units of 10^{-6} cm**)
  - Should be larger than A_r*L , where:
    - L is the largest coordinate value in the geometry (excluding the BLCKHOLE confinement)
    - A_r is the relative accuracy achievable in double precision (~10^{-14}-10^{-15}).
  - By default, FLUKA tries to guess the parameter based on the second largest body in your geometry (assuming the largest one must be the BLCKHOLE confinement)
    - Can be overwritten by explicitly setting the parameter in **GEOBEGIN**



GEOBEGIN Accuracy: Option: ▾ Paren:
Geometry: ▾ Out: ▾ Fmt: COMBNAME ▾
Title: Target setup

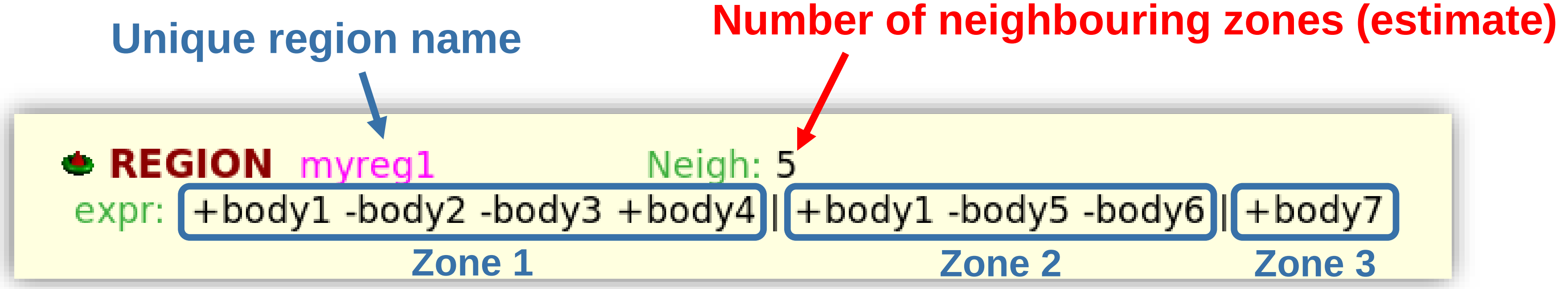
Parentheses in region definitions

- Parentheses can be used to perform complex Boolean operations in region descriptions
- The highest operator precedence is given to parentheses, followed by +, - and the | operator when evaluating the expressions
- The extensive use of parenthesis is discouraged and can create inefficient code
  - It is perfectly possible to implement (complex) geometries without using parentheses
- Flair can expand expressions with parentheses, as well as optimise region definitions with multiple zones with the options available in the Geometry tab:



The number of adjacent zones (REGION card)

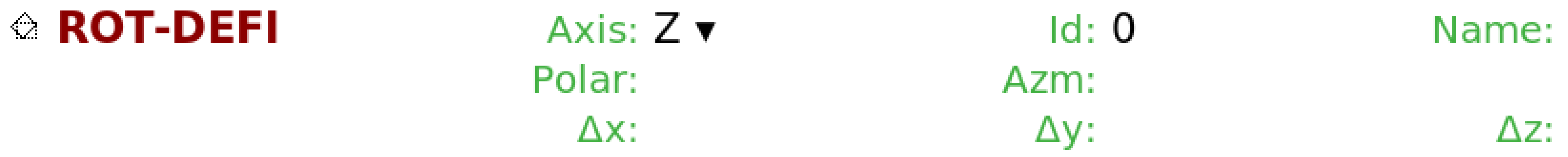
- **"Neigh"** parameter in the **REGION** card:
  - It is a rough estimate of the number of zones a particle can enter when leaving the current region zones (**5 by default**). What actually matters is the **sum over all regions**, defining the size of the contiguity list. **Note: for many simulation cases, the default value is fine.**
  - While tracking, the program searches in the contiguity list for the neighbouring zones of each zone. If the zone is not yet in the list, the whole geometry is scanned and it is added to the list with its neighbouring zones.
  - When the limit is reached (i.e. the list is full) the code prints a warning: **GEOMETRY SEARCH ARRAY FULL**. **This is not lethal: the calculation continues but with a reduced efficiency**.
  - If you have more than 1000 regions, you must issue a **GLOBAL** card putting in WHAT(1) a higher limit (not beyond 10000).



```
REGION myreg1                    Neigh: 5
expr: +body1 -body2 -body3 +body4 | +body1 -body5 -body6 | +body7
```

Refresher:
transformations and the `ROT-DEFI` card

The ROT-DEFIni card

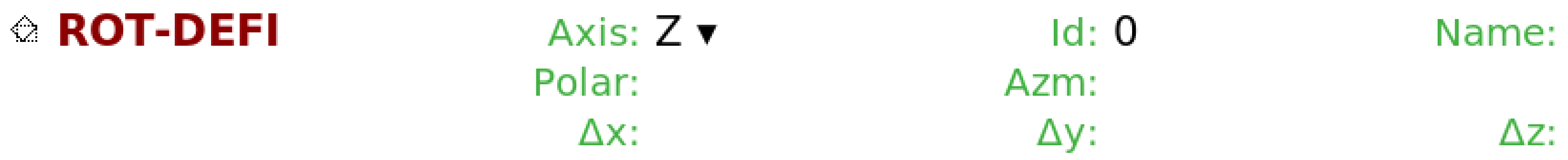


ROT-DEFI	Axis: Z ▾	Id: 0	Name:
	Polar:	Azm:	
	Δx:	Δy:	Δz:

The ROT-DEFIni card defines roto-translations that can be applied to bodies, USRBIN and EVENTBIN cards (via ROTPRBIN card), and LATTICE objects

Axis:	rotation with respect to axis
Id:	transformation index If set to 0, then Id is automatically assigned
Name:	transformation name Optional but recommended for easy referencing
Polar:	polar angle of the rotation $\mathbf{R}_{\mathbf{pol}}$ ($0 \leq \vartheta \leq 180$ degrees)
Azm:	azimuthal angle of the rotation $\mathbf{R}_{\mathbf{azm}}$ ($-180 \leq \varphi \leq 180$ degrees) [clockwise]
Δx, Δy, Δz:	offset for the translation **T**

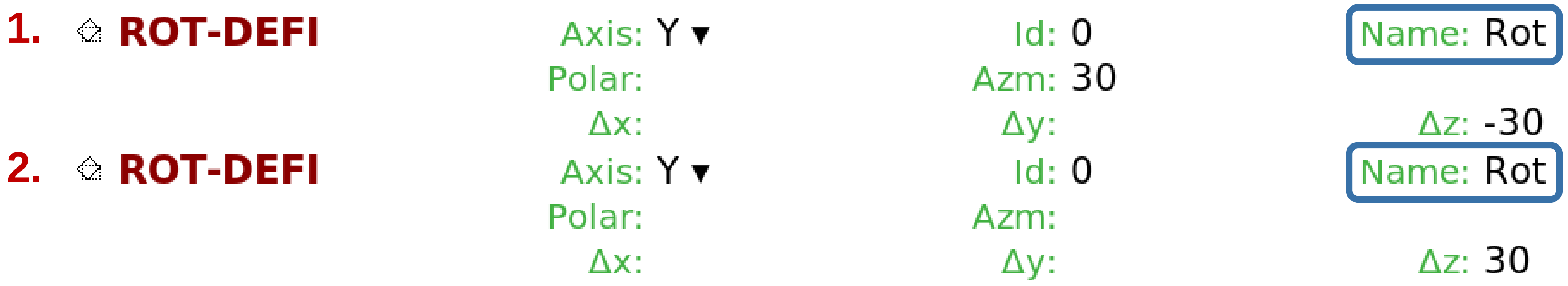
The ROT-DEFIni card



ROT-DEFI	Axis: Z ▾	Id: 0	Name:
	Polar:	Azm:	
	Δx:	Δy:	Δz:

- In a ROT-DEFI, the transformation is defined as $X_{new} = \overset{3.}{\mathbf{R}_{\mathbf{pol}}(\vartheta)} \times \overset{2.}{\mathbf{R}_{\mathbf{azm}}(\varphi)} \times \overset{1.}{(X_{old} + \mathbf{T})}$
- The order of translation / rotation is relevant. They are not commutative!
- The rotations are always performed around the origin of the coordinate system!
- It is preferable to define rotations through the azimuthal angle
- The convention used in the rotation matrices is available in the manual
  See: Section 7 – ROT-DEFIni – Note 4

The ROT-DEFIni card – “chaining”

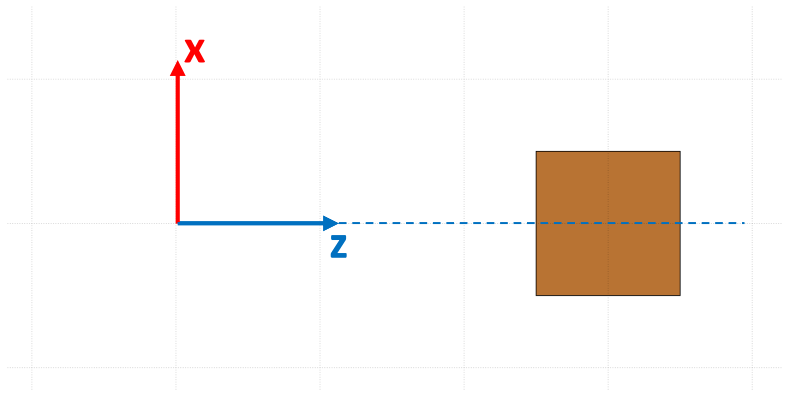


1. ROT-DEFI	Axis: Y ▾	Id: 0	Name: Rot
	Polar:	Azm: 30	
	Δx:	Δy:	Δz: -30
2. ROT-DEFI	Axis: Y ▾	Id: 0	Name: Rot
	Polar:	Azm:	
	Δx:	Δy:	Δz: 30

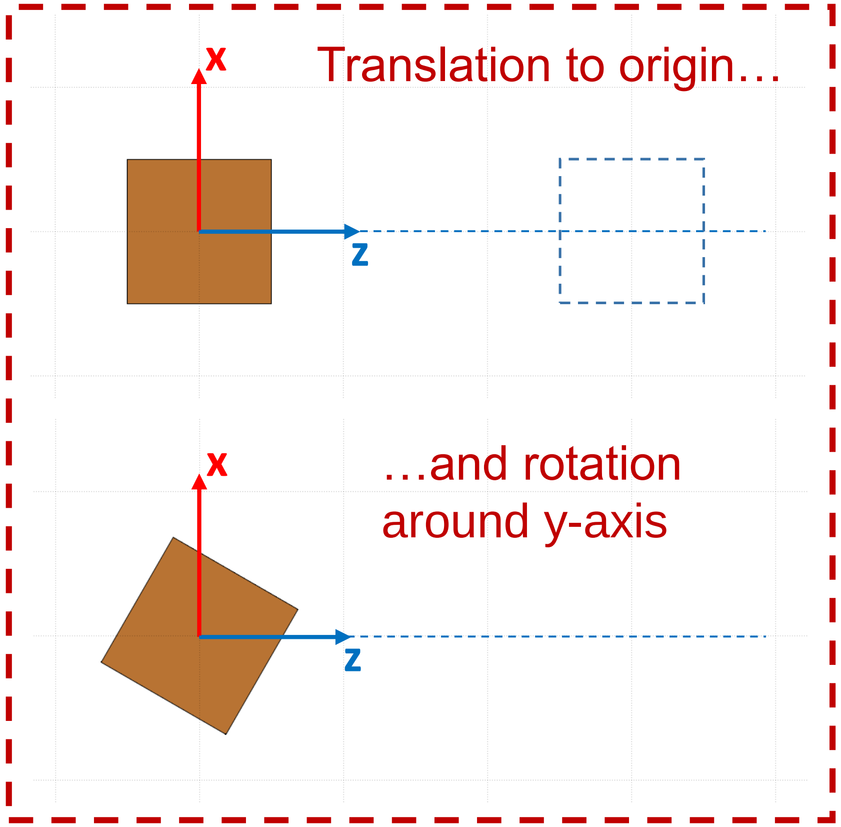
- It is possible to “chain” multiple `ROT-DEFIni` cards as a single transformation
    - The Name (or Id) on the “chained” `ROT-DEFIni` cards has to be the same
    - The `ROT-DEFIni` cards are applied from top to bottom
- The inverse transformation is also accessible with a minus sign (“–”) before the name or Id number

The `ROT-DEFIni` card – “chaining”

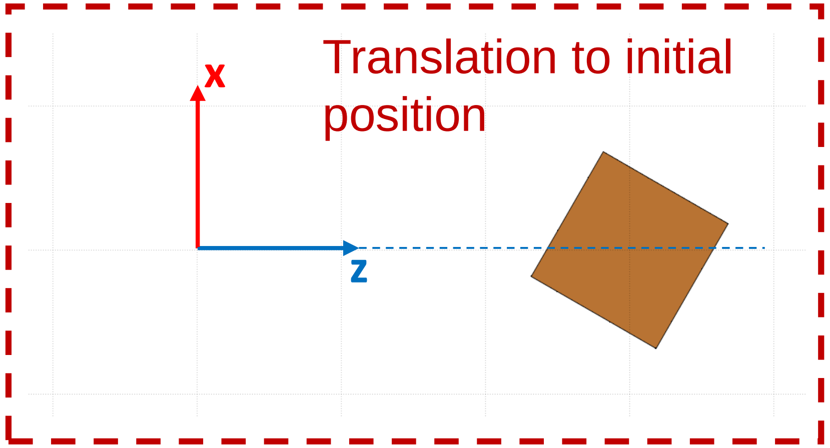
1. ROT-DEFI	Axis: Y ▾	Id: 0	Name: Rot
	Polar:	Azm: 30	
	Δx:	Δy:	Δz: -30
2. ROT-DEFI	Axis: Y ▾	Id: 0	Name: Rot
	Polar:	Azm:	
	Δx:	Δy:	Δz: 30



1.



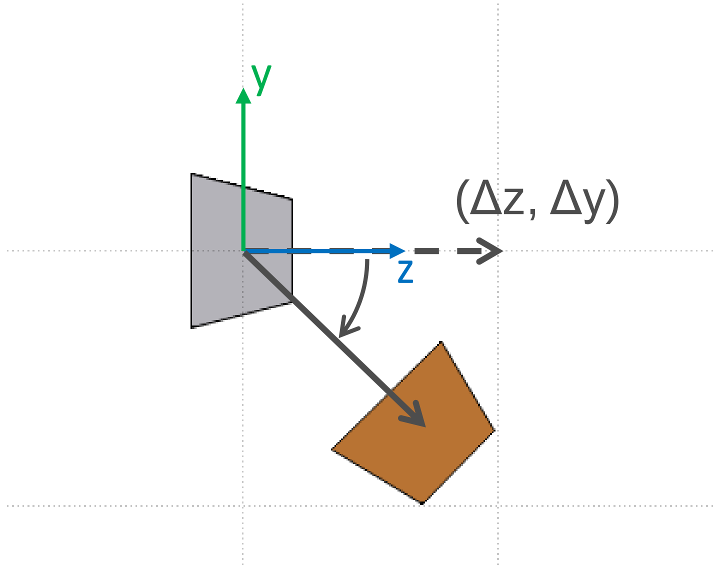
2.

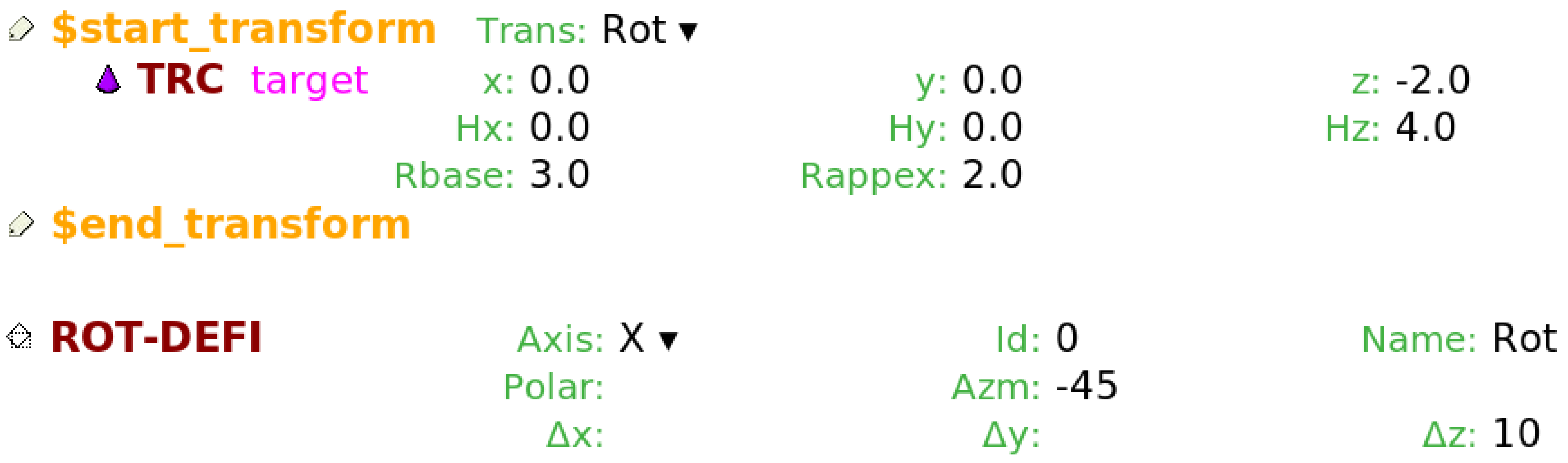


Geometry directive: transform

```
$start_transform
...
$end_transform
```

applies a roto-translation (pre-defined via **ROT-DEFI**) to all bodies embedded within the directive



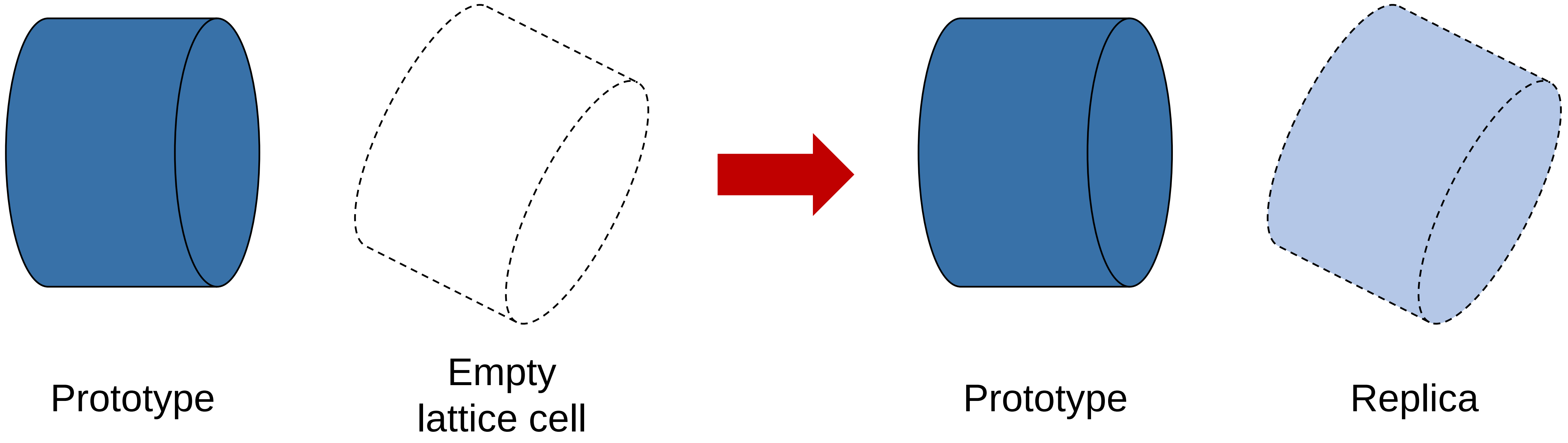


```
$start_transform   Trans: Rot ▾
  TRC  target        x: 0.0            y: 0.0          z: -2.0
                    Hx: 0.0           Hy: 0.0         Hz: 4.0
                 Rbase: 3.0       Rappex: 2.0
$end_transform

ROT-DEFI              Axis: X ▾          Id: 0        Name: Rot
                     Polar:             Azm: -45
                        Δx:              Δy:             Δz: 10
```

Geometry replication: LATTICE

Lattice: the concept

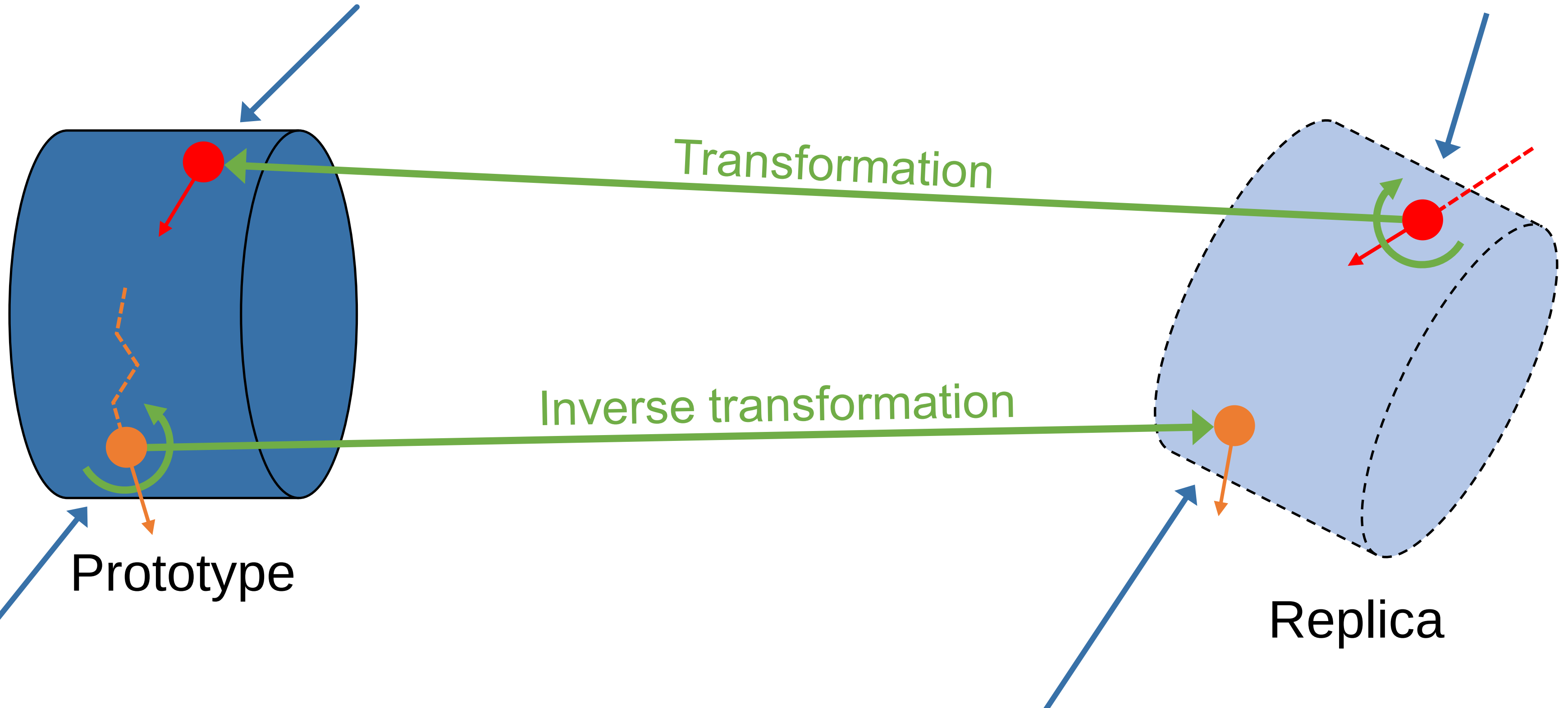
- FLUKA offers replication capabilities via the `LATTICE` card, which creates a replica of a model within an empty cell defined by a closed body identical to the container body of the model



Lattice: the concept

2. …its coordinates are transformed to the prototype, inside which FLUKA performs the tracking

1. For every particle entering the replica…

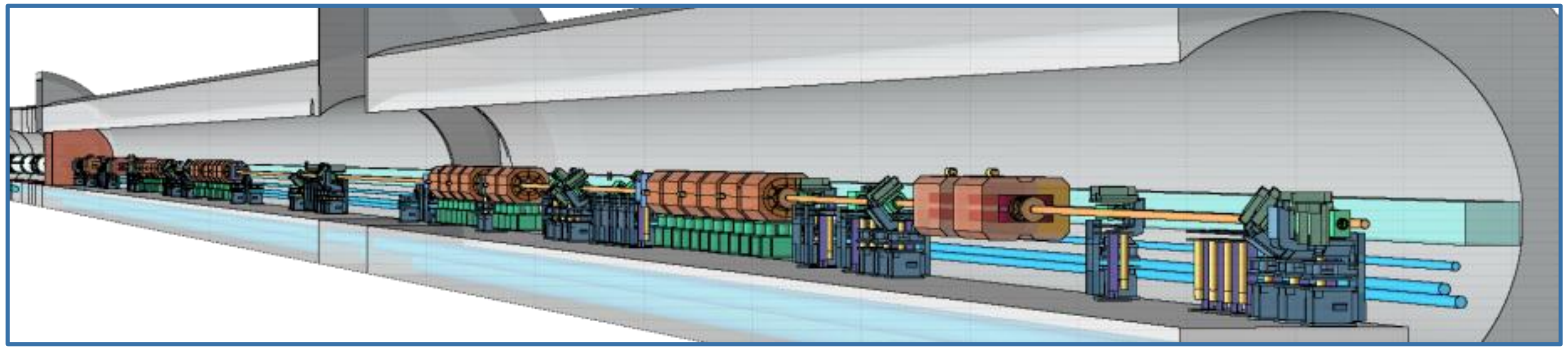


3. A resulting particle exiting the prototype…

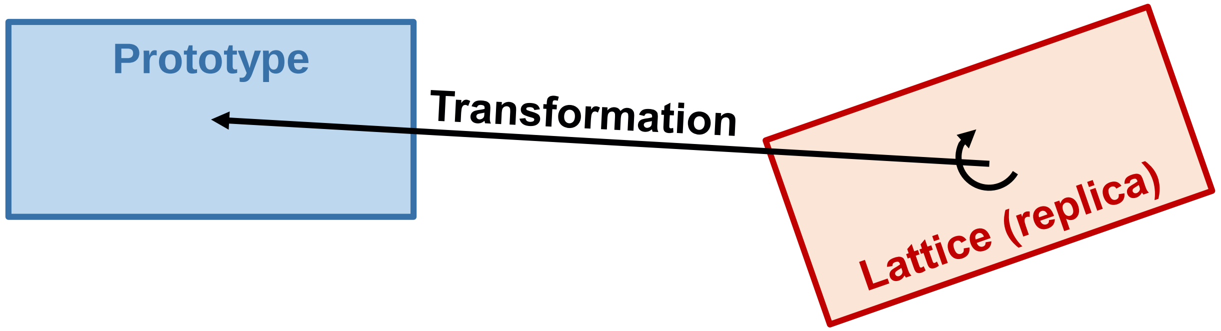
4. …is transformed back to the replica boundary to be tracked in the surrounding geometry

Lattice: basic usage

- Very useful for geometries where models are used multiple times (e.g. beamlines, detector arrays etc.)

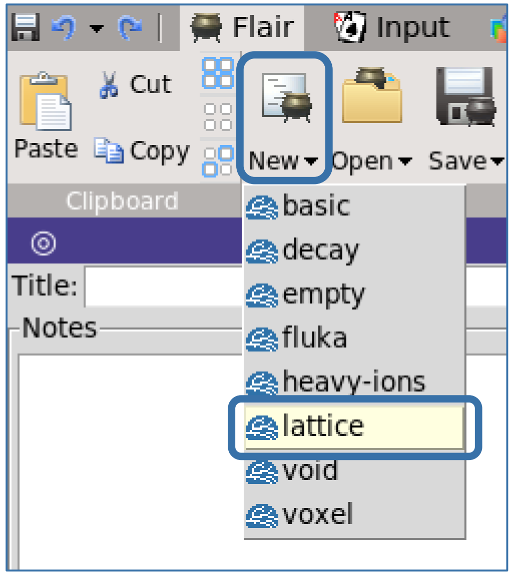


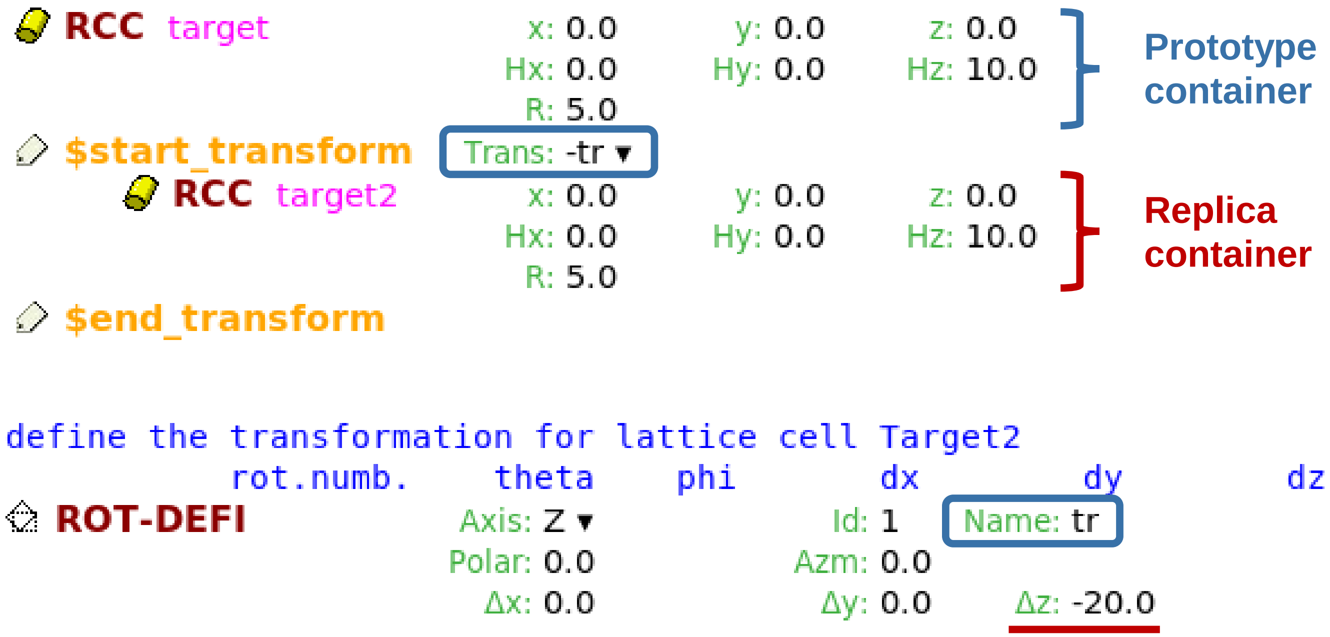
- The prototype is defined in detail with all the necessary information (geometry, materials etc.) inside a closed container body (**`RCC`**, **`RPP`** etc.). Materials and other properties are assigned only to the regions constituting the prototype.
- The lattices (replicas) are defined as “empty” regions in their correct location and declared as such with the **`LATTICE`** card at the end of the geometry input (but before the **`GEOEND`** card)
- The transformations exactly mapping the replicas onto the prototype are defined using **`ROT-DEFI`** cards



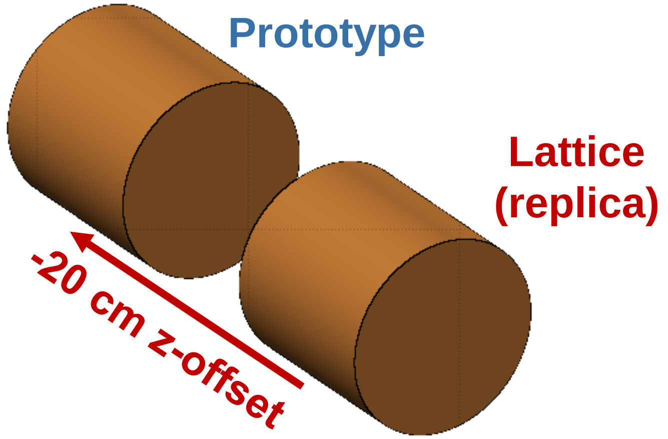
Getting started: a simple example

- A simple working example can be obtained by starting a new project in Flair and loading the lattice template →
- The prototype and the replica are contained in identical container bodies
- **Note:** The transformation maps the replica onto the prototype
- The inverse transformation is applied to the replica container to bring it to the replica location





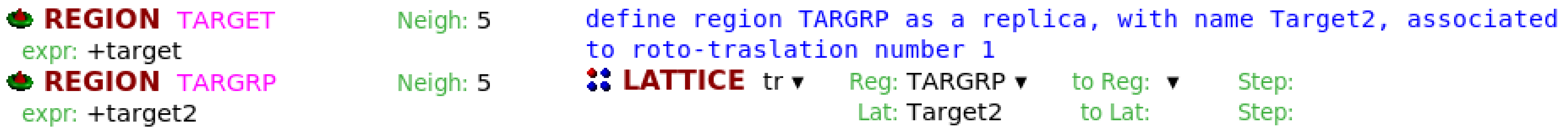
```
RCC  target                x: 0.0       y: 0.0       z: 0.0
                          Hx: 0.0      Hy: 0.0      Hz: 10.0     Prototype container
                           R: 5.0
$start_transform    Trans: -tr ▾
     RCC  target2          x: 0.0       y: 0.0       z: 0.0
                          Hx: 0.0      Hy: 0.0      Hz: 10.0     Replica container
                           R: 5.0
$end_transform

define the transformation for lattice cell Target2
         rot.numb.    theta    phi      dx        dy        dz
ROT-DEFI             Axis: Z ▾          Id: 1   Name: tr
                    Polar: 0.0        Azm: 0.0
                       Δx: 0.0         Δy: 0.0     Δz: -20.0
```



Good practice: To define the replica container body, clone the container body of the prototype, rename it and apply to it the inverse of the desired transformation via a $start_transform directive

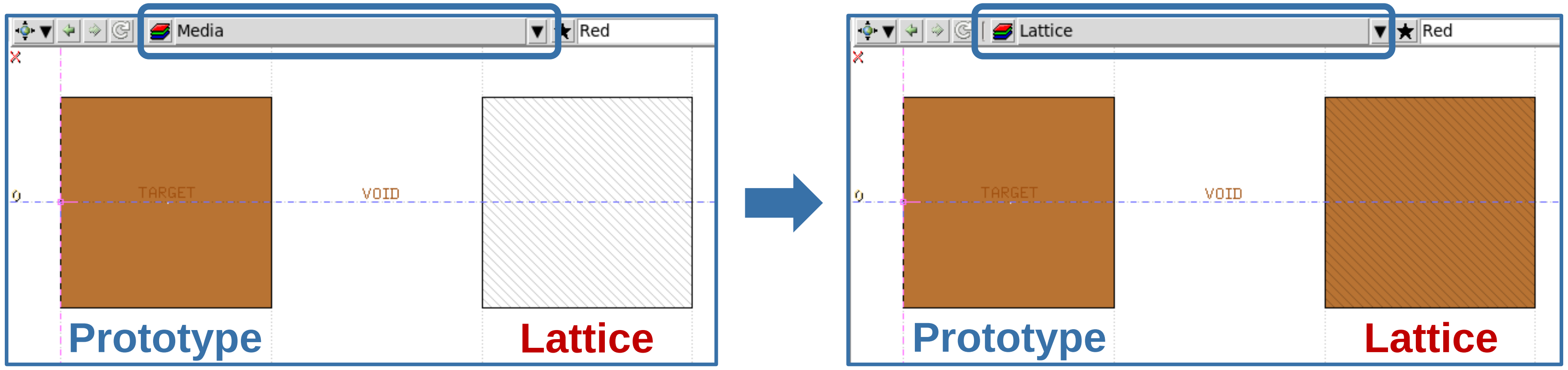
Getting started: a simple example

- A second region is created (TARGRP) inside the replica container body and declared as a lattice region (named Target2) associated to the transformation



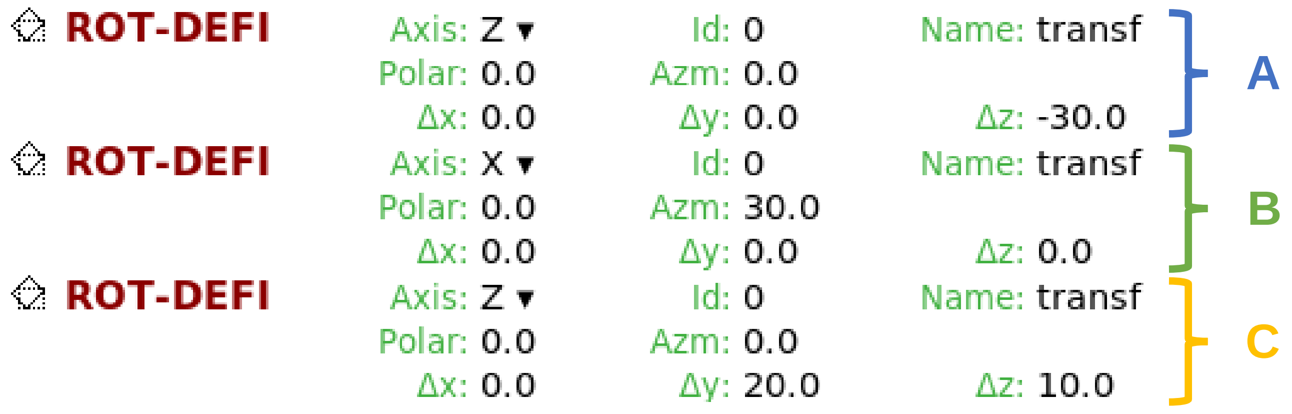
```
REGION  TARGET            Neigh: 5      define region TARGRP as a replica, with name Target2, associated
  expr: +target                         to roto-traslation number 1
REGION  TARGRP            Neigh: 5      LATTICE  tr ▾    Reg: TARGRP ▾    to Reg: ▾     Step:
  expr: +target2                                         Lat: Target2      to Lat:       Step:
```

- In Flair, the lattice region is displayed as a container filled with a diagonal pattern
- To display the geometry of the lattice, change the view from Media to Lattice in the viewport

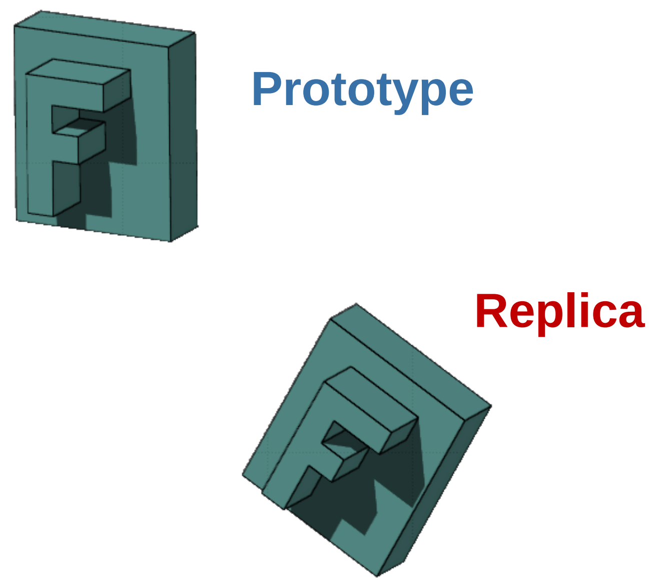


- **Tip:** You can click on a lattice region and type L to be taken to the prototype

A more extended example

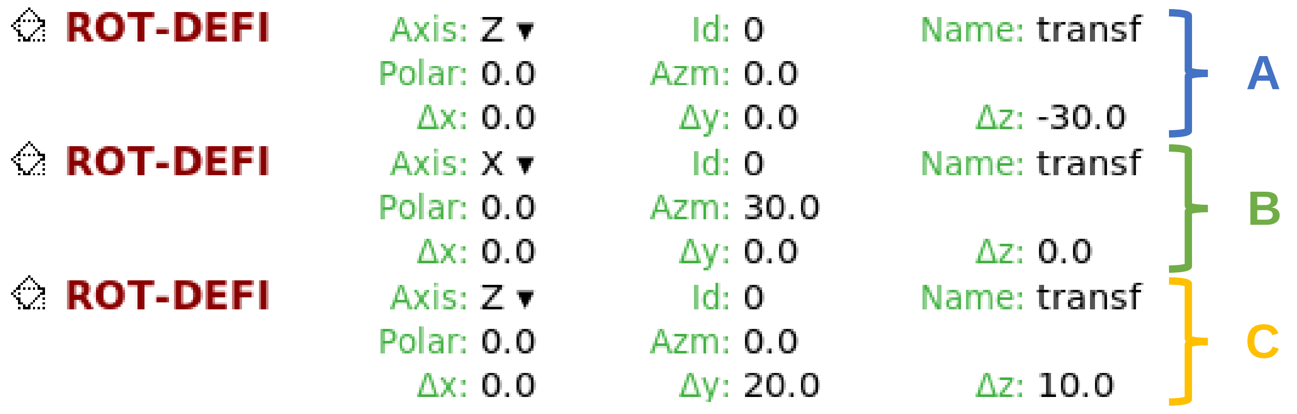


```
ROT-DEFI     Axis: Z ▾        Id: 0         Name: transf  ⎫
             Polar: 0.0       Azm: 0.0                    ⎬ A
             Δx: 0.0          Δy: 0.0       Δz: -30.0     ⎭
ROT-DEFI     Axis: X ▾        Id: 0         Name: transf  ⎫
             Polar: 0.0       Azm: 30.0                   ⎬ B
             Δx: 0.0          Δy: 0.0       Δz: 0.0       ⎭
ROT-DEFI     Axis: Z ▾        Id: 0         Name: transf  ⎫
             Polar: 0.0       Azm: 0.0                    ⎬ C
             Δx: 0.0          Δy: 20.0      Δz: 10.0      ⎭
```



- This transformation is defined with 3 **`ROT-DEFI`** cards
- Applying the 3 roto-translations *A*, *B* and *C* in the sequence in which they appear should map the replica onto the prototype

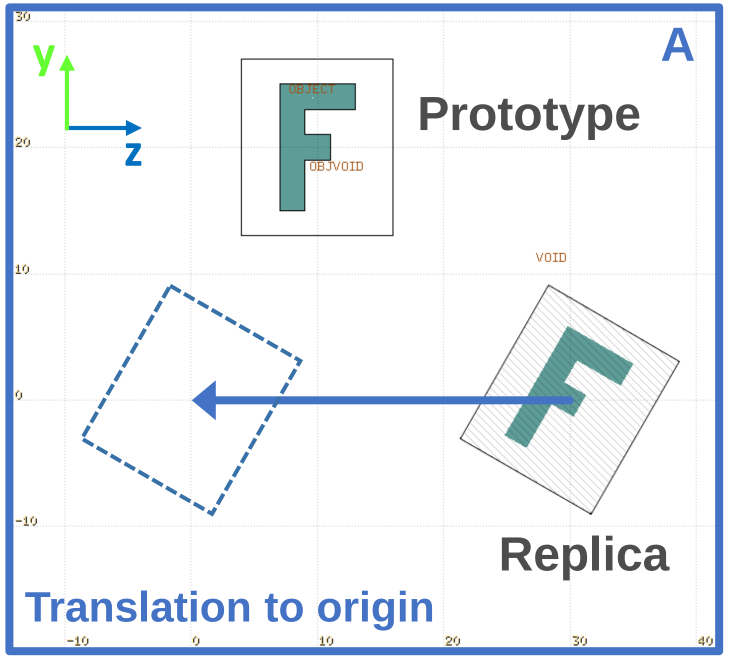
A more extended example

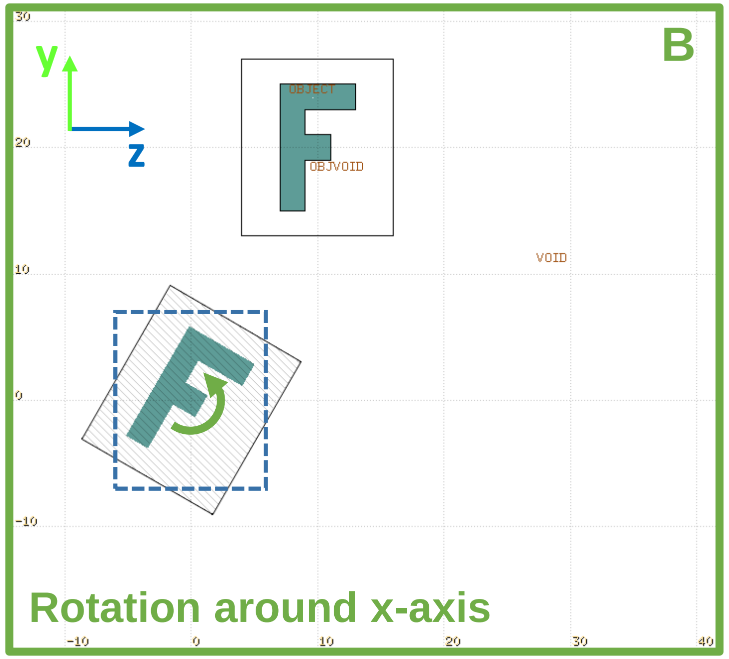


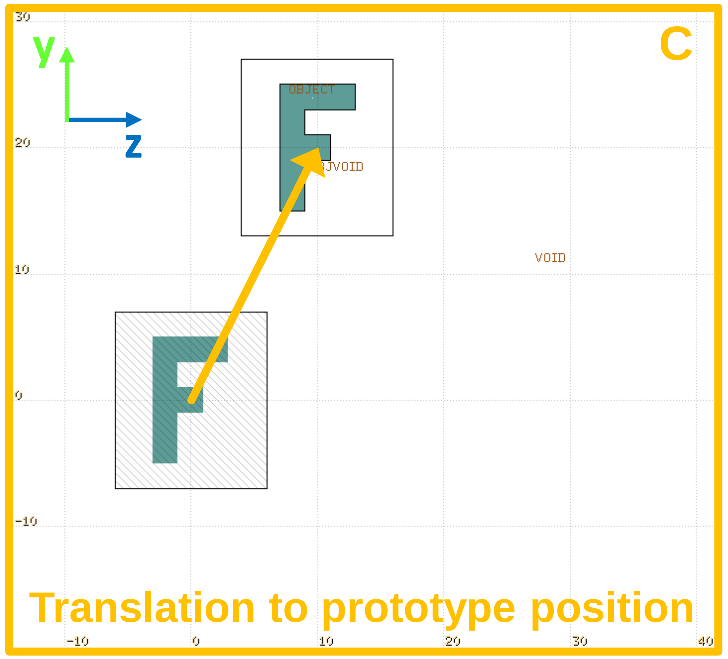
ROT-DEFI	Axis: Z ▾	Id: 0	Name: transf	A
	Polar: 0.0	Azm: 0.0		
	Δx: 0.0	Δy: 0.0	Δz: -30.0	
ROT-DEFI	Axis: X ▾	Id: 0	Name: transf	B
	Polar: 0.0	Azm: 30.0		
	Δx: 0.0	Δy: 0.0	Δz: 0.0	
ROT-DEFI	Axis: Z ▾	Id: 0	Name: transf	C
	Polar: 0.0	Azm: 0.0		
	Δx: 0.0	Δy: 20.0	Δz: 10.0	

Note: if a transformation *R* consists of 3 roto-translations *A*, *B* and *C* applied in this order, then $R = CBA$ and the inverse transformation is $R^{-1} = A^{-1}B^{-1}C^{-1}$

Here, *R* maps the replica onto the prototype, whereas R^{-1} is used to transform the replica container from the prototype to the replica







Numerical precision issues

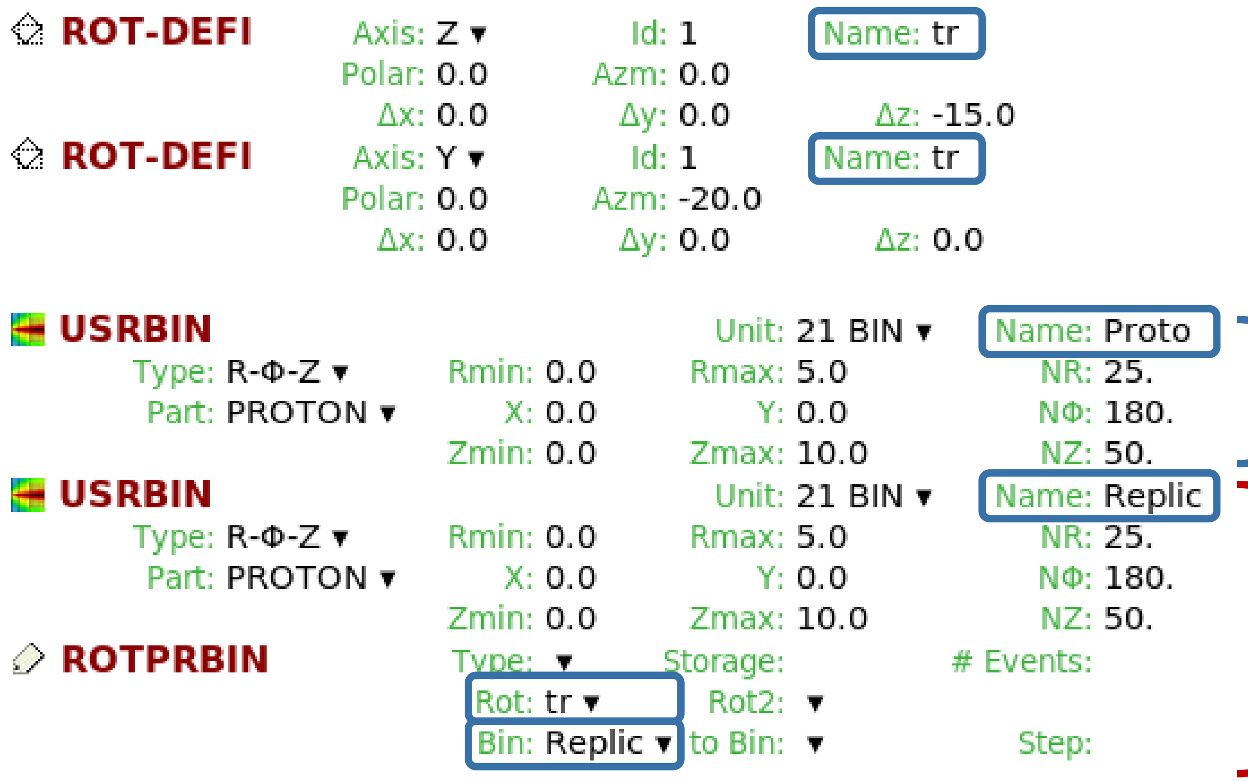
- Where transformations are involved, it is possible that the user may encounter problems stemming from numerical precision
- Geometry errors can be generated e.g. when:
  - There is an imperfect mapping of the replica onto the prototype, even when the transformation seems broadly correct at first glance
  - Replicas are touching or are separated by small layers of “real” geometry
- To minimise such issues, use as many digits as possible to describe the transformation and the container bodies of the prototype and replicas
  - E.g., the **`ROT-DEFI`** cards can be enclosed between a **`FREE`** and a **`FIXED`** card to allow input of more than 9 digits (see Section 7.22 of the manual)

Scoring with lattices: the `ROTPRBIN` card

- Cartesian and cylindrical meshes (e.g. `USRBIN`) are geometry-independent
- If superimposed on a replica, they will score the desired quantity at that location (i.e. not in the prototype or another replica)
- A mesh covering the prototype can be cloned and applied to a replica via the `ROTPRBIN` card, which allows to apply a transformation to `USRBIN` or `EVENTBIN` scorings
  - As with lattices, the inverse transformation must be used in the `ROTPRBIN` card, i.e. it is not the scoring mesh that is transformed, but the transformation is applied to the scoring location, mapping it onto the initial mesh

Scoring with lattices: the **ROTPRBIN** card

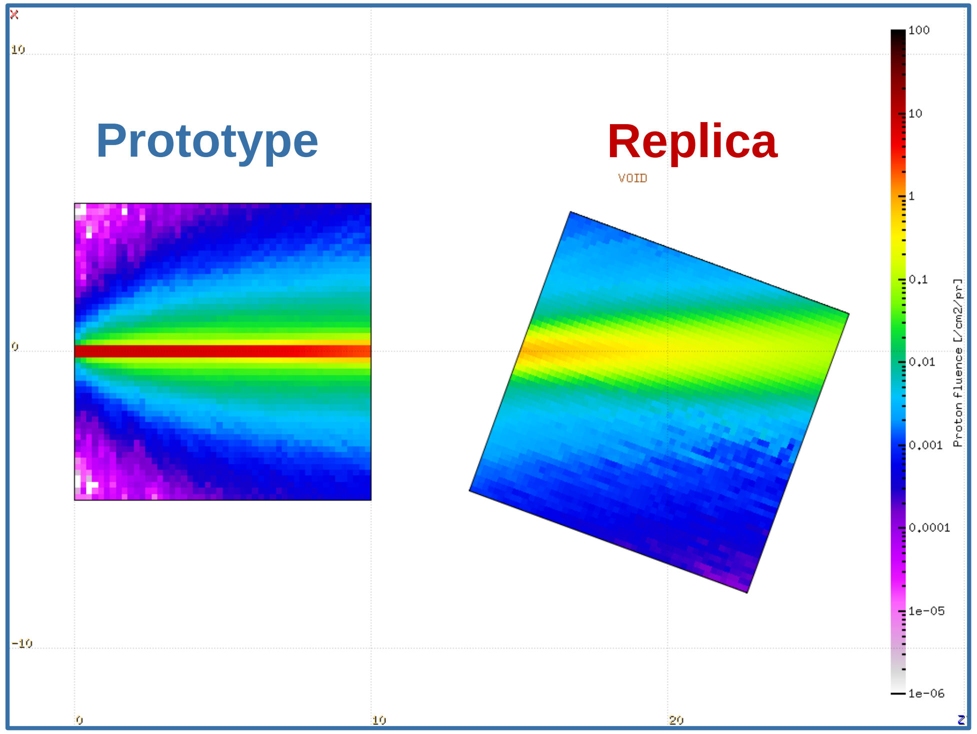
- Example (slight variation of the lattice template):
  - 1 GeV proton beam impinging on Cu cylinder; a replica of the cylinder is downstream and tilted
  - We wish to apply an R-Φ-Z proton fluence USRBIN to each object
  - We define the mesh for the prototype, then clone and rename it, and add a **ROTPRBIN** card associating it with the transformation used for the replica



```
ROT-DEFI      Axis: Z ▾         Id: 1          Name: tr
             Polar: 0.0        Azm: 0.0
                Δx: 0.0         Δy: 0.0           Δz: -15.0
ROT-DEFI      Axis: Y ▾         Id: 1          Name: tr
             Polar: 0.0        Azm: -20.0
                Δx: 0.0         Δy: 0.0           Δz: 0.0

USRBIN                          Unit: 21 BIN ▾   Name: Proto
      Type: R-Φ-Z ▾   Rmin: 0.0   Rmax: 5.0        NR: 25.
      Part: PROTON ▾     X: 0.0      Y: 0.0        NΦ: 180.
                      Zmin: 0.0   Zmax: 10.0       NZ: 50.
USRBIN                          Unit: 21 BIN ▾   Name: Replic
      Type: R-Φ-Z ▾   Rmin: 0.0   Rmax: 5.0        NR: 25.
      Part: PROTON ▾     X: 0.0      Y: 0.0        NΦ: 180.
                      Zmin: 0.0   Zmax: 10.0       NZ: 50.
ROTPRBIN              Type: ▾      Storage:        # Events:
                      Rot: tr ▾     Rot2: ▾
                      Bin: Replic ▾ to Bin: ▾          Step:
```

Mesh on prototype

Mesh on replica



Keep in mind…

- Lattice nesting is not allowed!
- Multiple prototypes and multiple replicas can be defined and coexist in the same geometry
  - A LATTICE card needs to be included for each one
- The name or number assigned to a lattice in the `LATTICE` card is used to identify the replica in all user routines and scoring
  - Prototypes are assigned the lattice number 0
- **ATTENTION:** Scoring a quantity by region when lattices are involved will give the sum of the quantity over the prototype and all replicas; this is useless in most cases
  - Use the special per region per lattice scoring (see WHAT(1) of USRBIN card)
- The `fluscw.f` user routine can be applied to various scorings (USRBIN, USRTRACK, …) to filter scoring based on the lattice number

The `lattic` routine

- The **`lattic.f`** user routine can alternatively be used to define the transformations used to map the lattice replicas onto the corresponding prototypes
  - The SDUM of the (still necessary) **`LATTICE`** card should be left blank in this case
- Its use is only mandatory to describe a mirrored geometry, which generally cannot be achieved via roto-translations alone, unless the prototype already has some symmetry
- It can also be helpful when placing a large number of replicas according to a simple rule (e.g. a detector array)

```
SUBROUTINE LATTIC ( XB, WB, DIST, SB, UB, IR, IRLTGG, IRLT, IFLAG )
```

XB, WB: vectors with the current particle position and direction

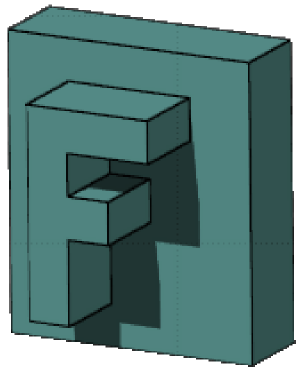
IR: current region number

IRLTGG: current lattice number (it may optionally be defined in the **`LATTICE`** card)

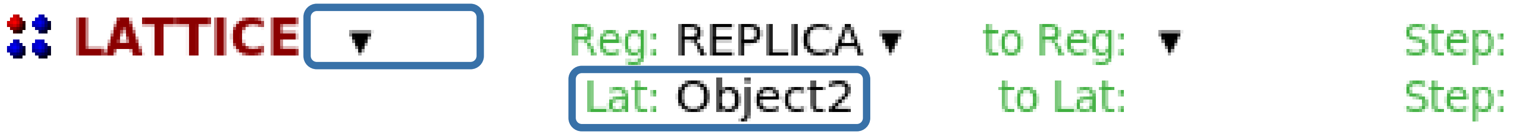
The routine must return the SB and UB vectors containing the position and direction of the particle transported to the prototype

The `lattic` routine: mirroring example

- Consider an object that cannot be mirrored using only roto-translations:



- We can use a simple `lattic` routine to achieve the desired effect
- The routine must transform coordinates and direction cosines from the replica to the prototype
- A **`LATTICE`** card is still needed, but the SDUM (transformation) is left empty:



```
LATTICE  ▾        Reg: REPLICA ▾     to Reg: ▾        Step:
                  Lat: Object2        to Lat:          Step:
```

The `lattic` routine: mirroring example

- The routine must return transformed coordinate and direction cosine vectors SB and UB:

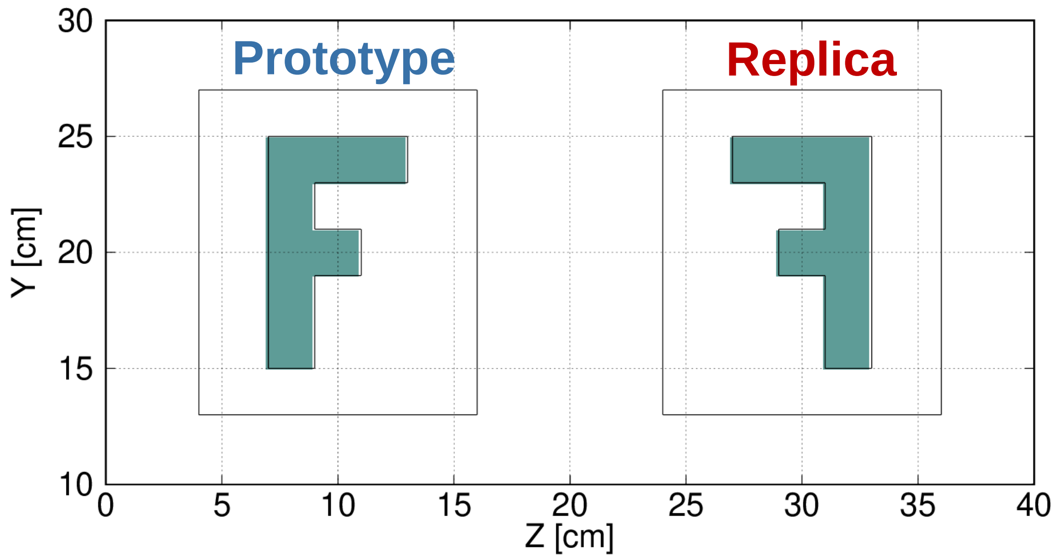
```
*  |  First time initialization:
      IF ( LFIRST ) THEN
*        Get lattice number
         CALL GEON2L ( 'Object2 ', IREP, IRTLAT, IERR )
         LFIRST = .FALSE.
      END IF
[...]
         IF ( IRLTGG .EQ. IREP ) THEN
            SB (1) =  XB (1)
            SB (2) =  XB (2)
            SB (3) = -XB (3) + 40.0D0
            UB (1) =  WB (1)
            UB (2) =  WB (2)
            UB (3) = -WB (3)
         END IF
```

Get the lattice number

- The LATNOR entry transforms the direction cosines UN of the vector normal to the surface in the prototype cell to the lattice cell number IRLTNO

```
ENTRY LATNOR ( UN, IRLTNO, IRLT )
[...]
UN (1) = UN (1)
UN (2) = UN (2)
UN (3) = -UN (3)
```

Finally



Geometry-related auxiliary routines

- These can be called inside user routines, preferably only the first time (usually under the `IF (LFIRST)` condition) and saving the required information

- Convert the region number to region name and vice-versa

```
CHARACTER*8 REGNAM
CALL GEON2R(REGNAM, NREG, IERR) Region name to region number
CALL GEOR2N(NREG, REGNAM, IERR) Region number to region name
```

  - IERR=0 in case of success

- Convert the lattice number to region name and vice-versa

```
CHARACTER*8 LATNAM
CALL GEON2L(LATNAM, NLATT, IRTLAT, IERR)       Lattice name to lattice number
CALL GEOL2N(NLATT, LATNAM, IRTLAT, IERR)       Lattice number to lattice name
```

  - IERR=0 in case of success
  - The index of the associated roto-translation (if present) is returned in IRTLAT

Geometry-related auxiliary routines

- Roto-translation routines

```
DIMENSION XPOINT (NPOINT), YPOINT (NPOINT), ZPOINT (NPOINT)
```

```
SUBROUTINE DOTRSF ( NPOINT, XPOINT, YPOINT, ZPOINT, KROTAT )
```

Applies, with a possible translation included, the KROTAT transformation (defined by ROT-DEFI) to NPOINT points, defined in the X/Y/ZPOINT arrays

```
SUBROUTINE UNDOTR ( NPOINT, XPOINT, YPOINT, ZPOINT, KROTAT)
```

Applies the inverse transformation, with a possible translation included

```
SUBROUTINE DORTNO ( NPOINT, XPOINT, YPOINT, ZPOINT, KROTAT)
```

Applies the KROTAT transformation without the possible translation

```
SUBROUTINE UNDRTO ( NPOINT, XPOINT, YPOINT, ZPOINT, KROTAT)
```

Performs the inverse transformation, without the possible translation

FLUKA