Contribution of the group of St. Petersburg University to the development of software for the BM@N experiment.

S.NEMNYUGIN, A.ALEXANDROV, A.IUSUPOVA, R.NIZAMOV

SAINT-PETERSBURG STATE UNIVERSITY

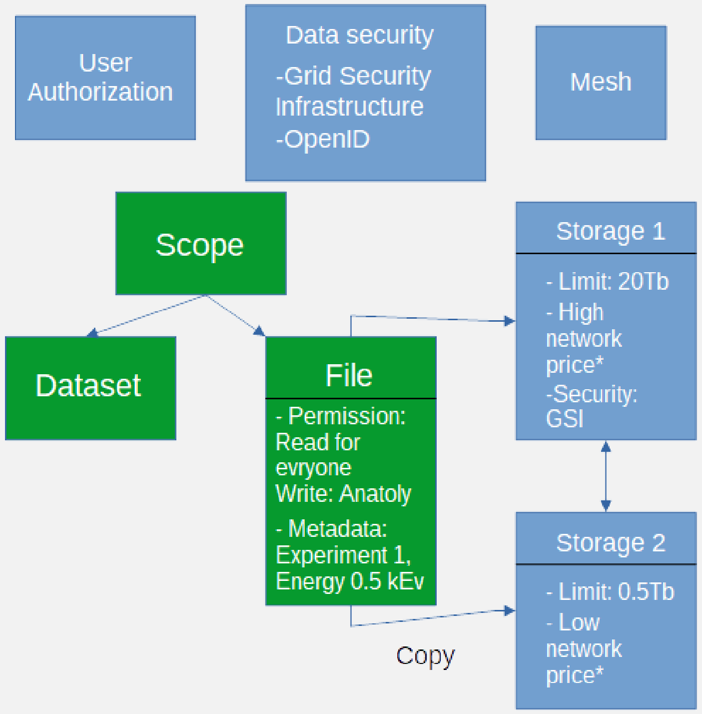
Outline

- Distributed big data storage management system.
- BmnRoot containerization.
- Virtual Reality and Augmented Reality for the BM@N.
- Performance optimization.

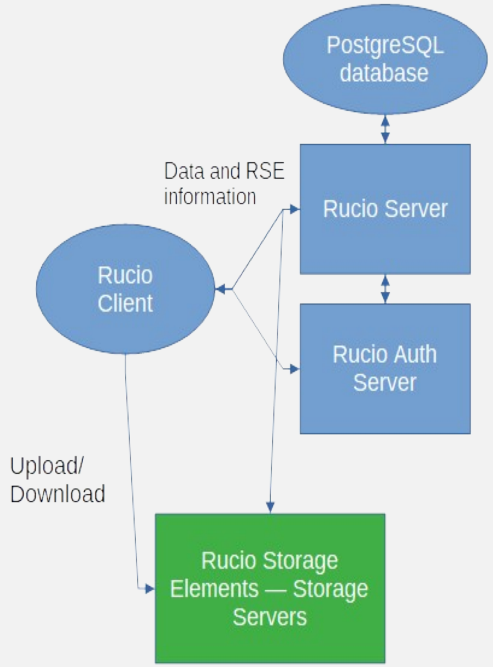
Integration of modern distributed big data storage management system for the BM@N experiment. Rucio.

Implementation of the Directory of files with simulated and experimental data of the BM@N experiment on the basis of the **Rucio** platform to implement high-intensity data processing. **Rucio** is a Distributed Data Management System. It efficiency was verified in LHC ATLAS.

Data Management System



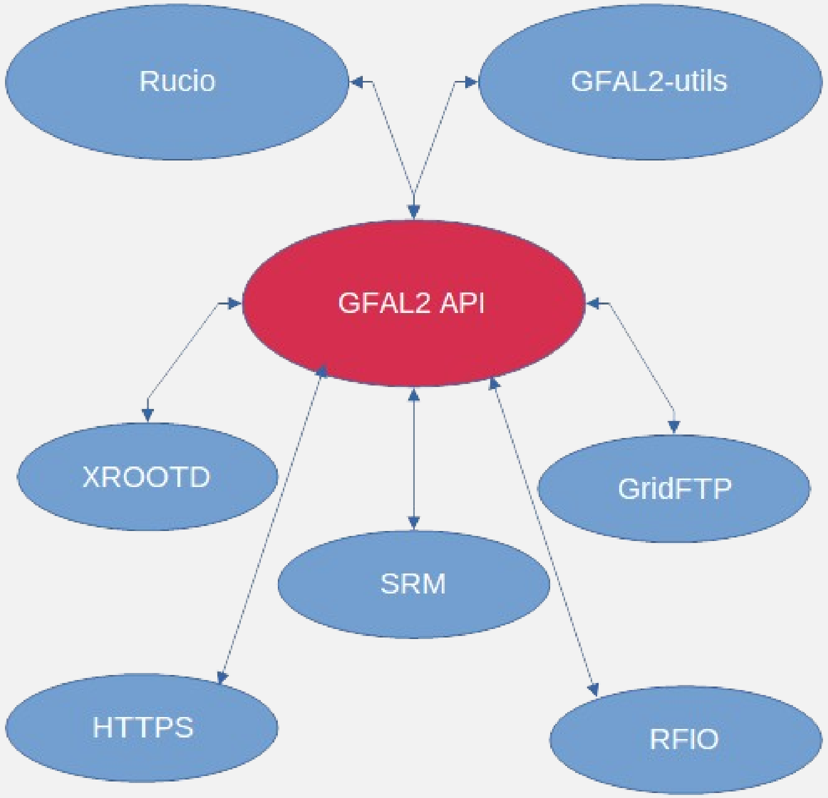
- Storage of detector data, simulator data, and user data.
- Unified interfacing of heterogenous network & storage infrastructure.
- Support for newer protocols in Storage & Network using plugins.
- Data Recovery.
- Adaptive Replication.
- Quota management.



✓ Server-client application.
✓ Support GFAL2 interface.
✓ Built-in user authorization system (X509, OpenID).

Software stack

- ✓ Rucio as data management system.
- ✓ GFAL2 compatible storage system: XROOTD, GridFTP, SRM, etc.
- ✓ PostgreSQL — RDBMS for Rucio database.



One interface combining data protocol interfaces used in the ATLAS project and other CERN projects

To deploy the Rucio server:

1. Create SSL certificate with Certification Authority certificate (Rucio not working without SSL).
2. Install and configure PostgreSQL (or another RDBMS).
3. Create Rucio database and initialize it, add admin username and password.
4. Install and configure Rucio Server.

BmnRoot containerization

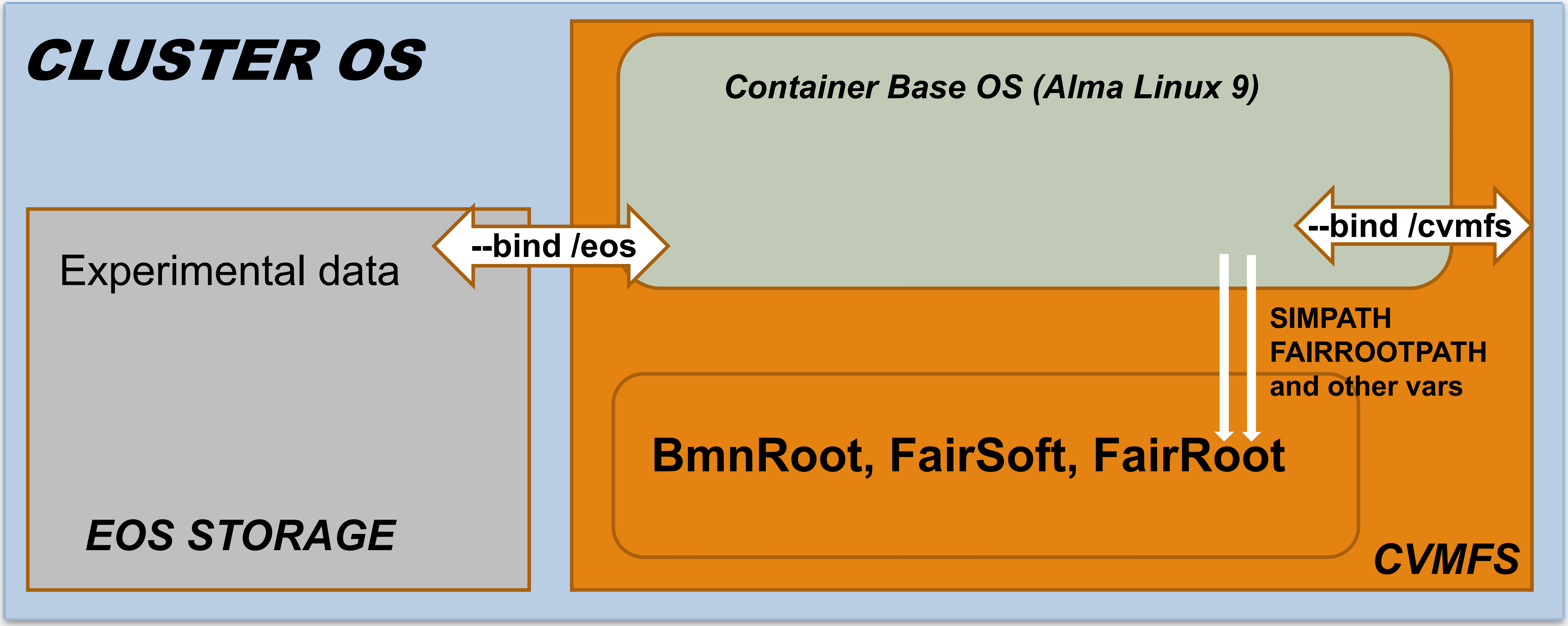
- Containers represented an alternative way to install and use BmnRoot.
- With containers, you don't have to worry about missing dependencies or libraries on the Host system.
- Images were made using **Docker**, but to run them, the **Apptainer** container system was used.
- **Apptainer**, the container system that specially created for high-performance computing (HPC) formerly known as **Singularity**.

Using containers with SLURM

The problem: to test the distributed macro runs under the container environment.

The benefit of this approach is the ability to use the prepared and assembled BmnRoot in CERN VM File System on any cluster (under any Host OS) in the container shell.

Scheme

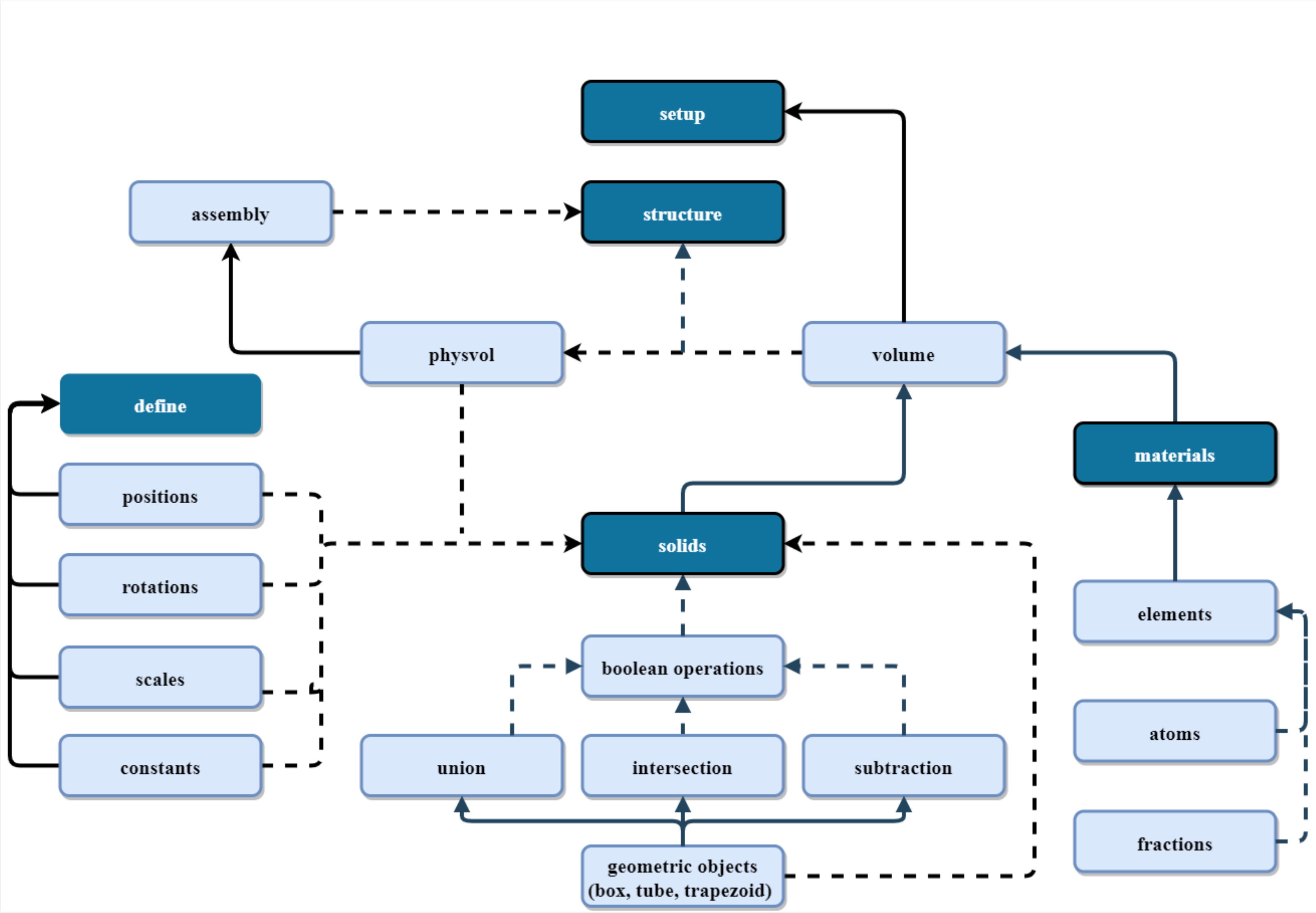

CLUSTER OS
Container Base OS (Alma Linux 9)
--bind /eos
--bind /cvmfs
Experimental data
SIMPATH
FAIRROOTPATH
and other vars
BmnRoot, FairSoft, FairRoot
EOS STORAGE
CVMFS

To test distributed macro runs two scripts were written:

- *cont_test.batch* – declares environment variables and launches the *apptainer_exec.sh* script under the container environment. Parallel start of reconstruction of several files implemented using SLURM Job arrays and $SLURM_ARRAY_TASK_ID environment variable

- *apptainer_exec.sh* – launches reconstruction macros with passed file names from EOS.

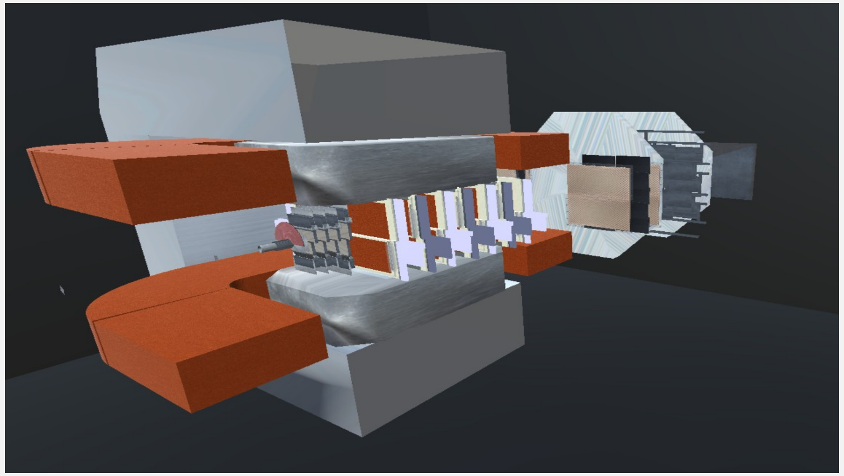
VR and AR visualization

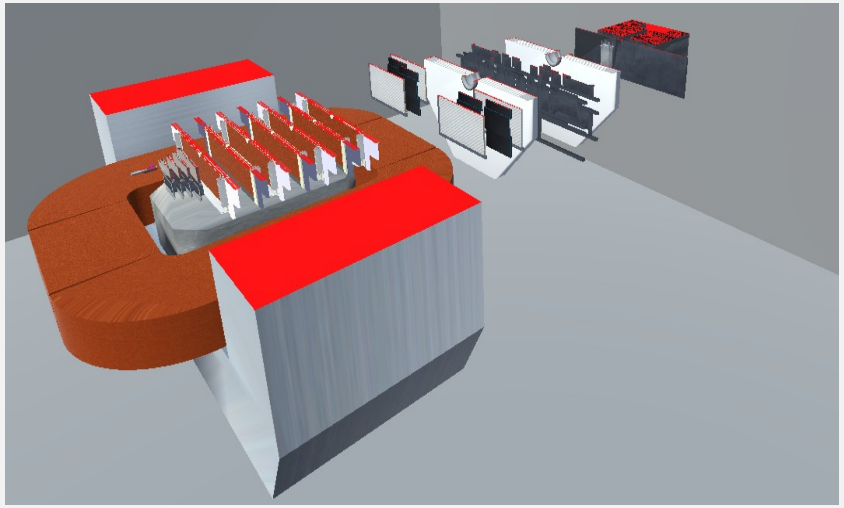


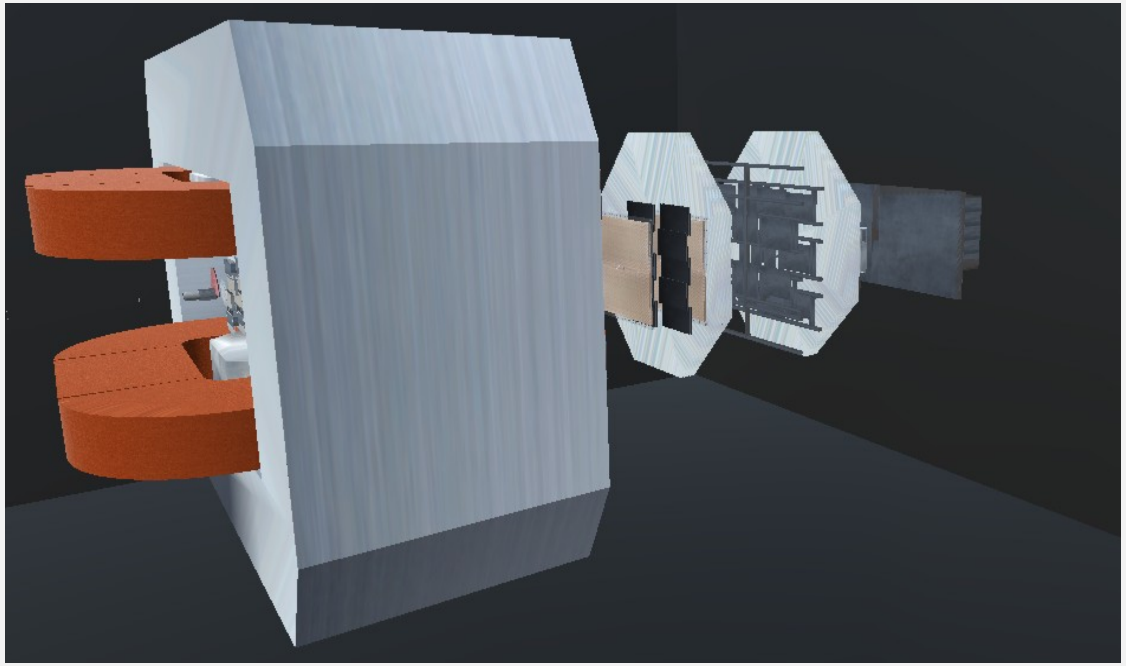
Detector geometry hierarchy structure

- Virtual Reality (VR) simulates the user's physical presence in a virtual environment.
- The detector geometry was exported to Unity using GDML macro for converting GDML into a C# script for import into Unity.

Imported geometry



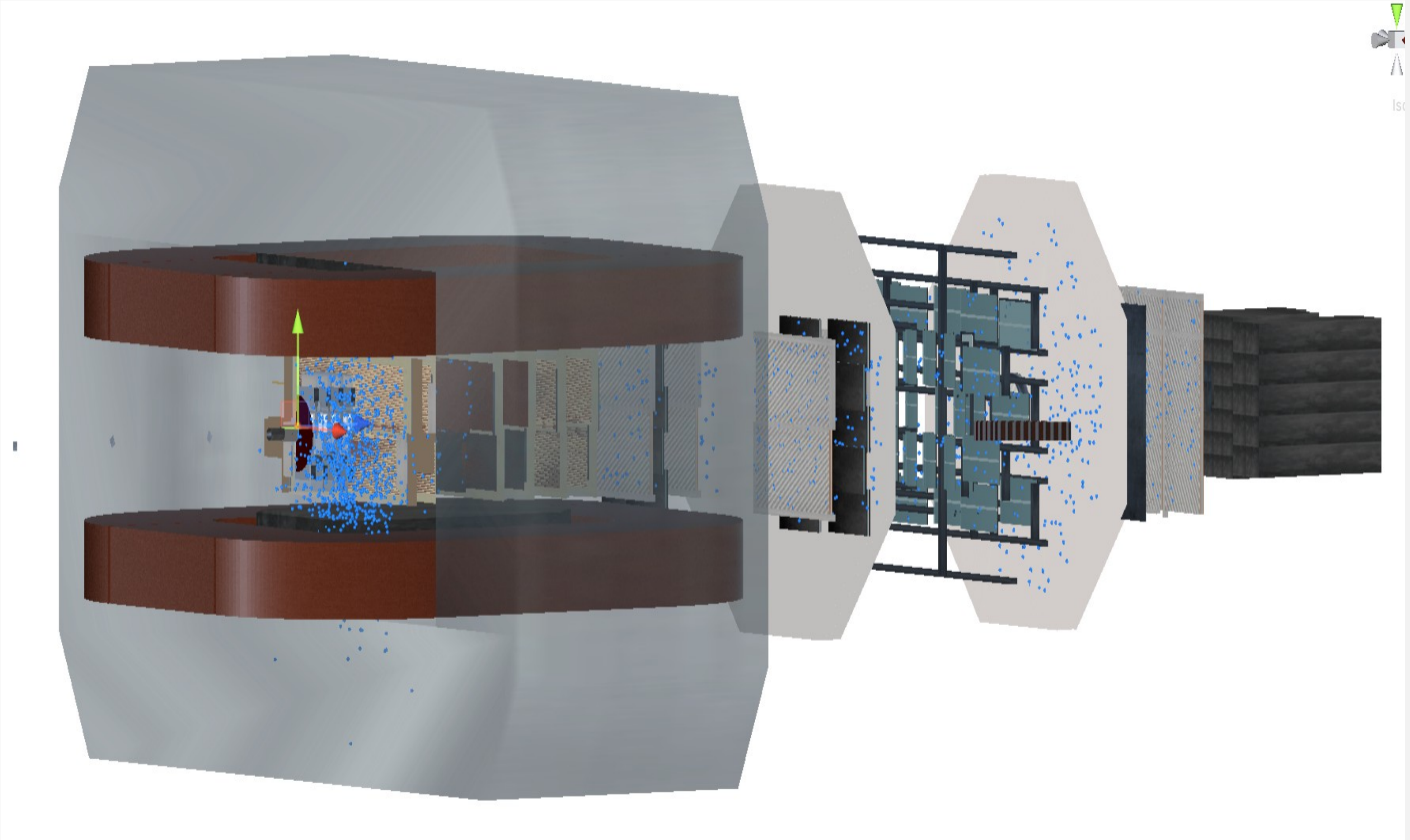




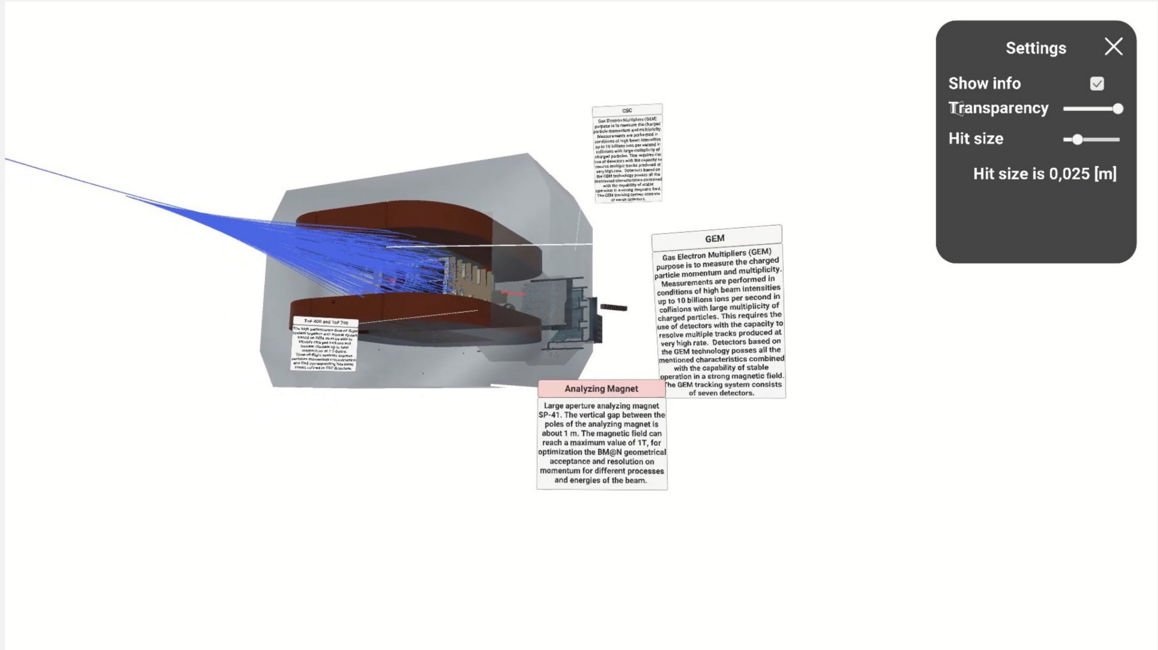
Transferred geometry of the BM@N experiment in Unty and its inner content (updated for geometry version 8)

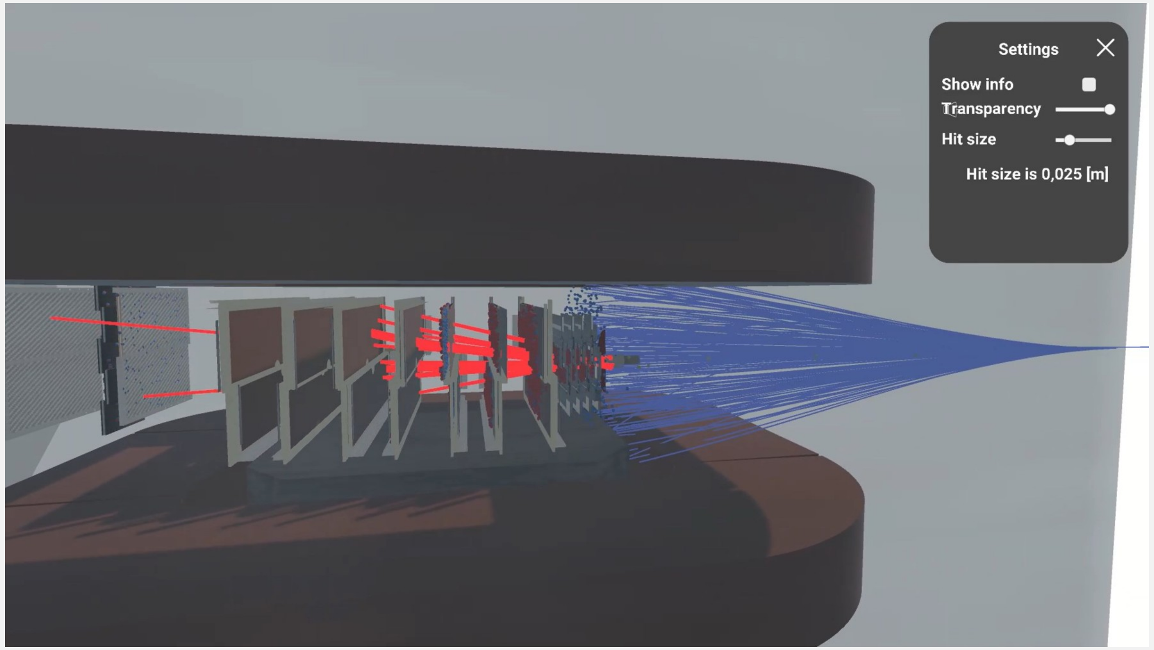
Export events, tracks, and points from simulation

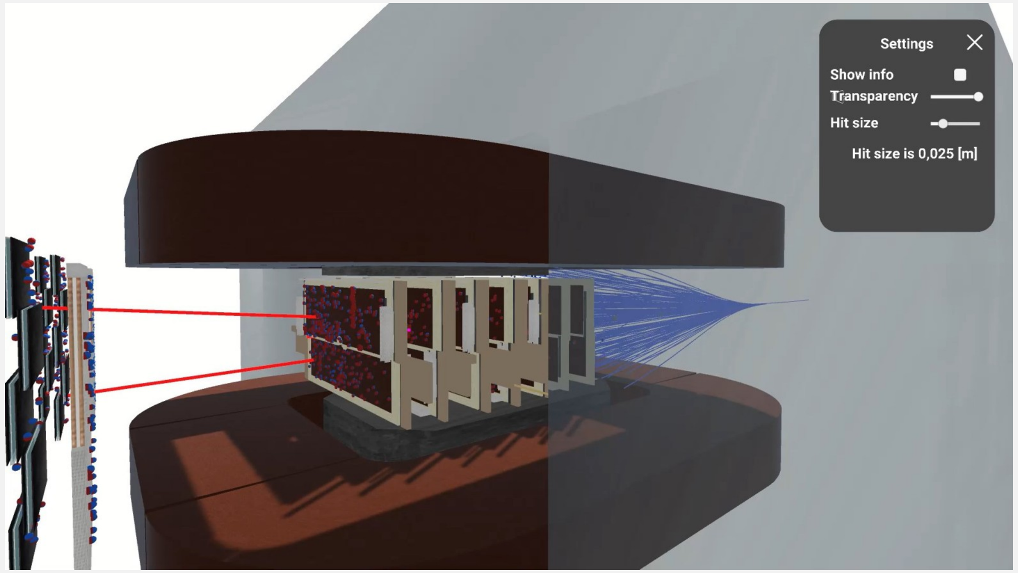
- Event includes:
    - event ID data,
    - vertex,
    - tracks
- Track includes:
    - track ID data,
    - mother ID,
    - vertex,
    - PDG (Particle Data Group) code,
    - Start time
    - points
- Point includes:
    - position,
    - out position,
    - length from start,
    - point ID,
    - detector type,
    - time of flight



Tracks and hits imported to Unity (1000 events)


Settings
Show info
Transparency
Hit size
Hit size is 0,025 [m]
GEM
Gas Electron Multipliers (GEM) purpose is to measure the charged particle momentum and multiplicity. Measurements are performed in conditions of high beam intensities up to 10 billions ions per second in collisions with large multiplicity of charged particles. This requires the use of detectors with the capacity to resolve multiple tracks produced at very high rate. Detectors based on the GEM technology posses all the mentioned characteristics combined with the capability of stable operation in a strong magnetic field. The GEM tracking system consists of seven detectors.
Analyzing Magnet
Large aperture analyzing magnet SP-41. The vertical gap between the poles of the analyzing magnet is about 1 m. The magnetic field can reach a maximum value of 1T, for optimization the BM@N geometrical acceptance and resolution on momentum for different processes and energies of the beam.


Settings
Show info
Transparency
Hit size
Hit size is 0,025 [m]


Settings
Show info
Transparency
Hit size
Hit size is 0,025 [m]

In progress

- GUI elements: object selection and manipulation.
- Displaying information about events, particles:
  - Particle mass, energy.
  - Time of flights.
  - Additional information.
- Process animation.
- Tracks hierarchy visualization.
- Integration with BmnRoot environment.

Current optimization status

Focus of optimization

- BmnFieldMap.cxx / BmnFieldMap.h
- BmnNewFieldMap.cxx / BmnNewFieldMap.h
- BmnKalmanFilter.cxx / BmnKalmanFilter.h

Test bench

ACER Nitro 5 AN515-52-75S2,CPU Intel Core i7 8750H (6 cores, 2x Hyperthreading, AVX2 vector extension), 32 Gb RAM.

Optimization methods under consideration

1. “Small” code improvements.
2. Vectorization of Kalman Filter and Field Map modules by vector intrinsics.
3. Evaluation of performance efficiency of computations offload on hybrid architectures.
4. Algorithmic optimizations.

«Small» code improvements of BmnNewFieldMap.cxx

```
#include "TMath.h"//Optimization. Explicit vectorization an O3 level
optimization
#pragma GCC optimize("O3")
…
Int_t   ixyz1 = ix * fNy * fNz + iy * fNz + iz;
Int_t   ixyz2 = ixyz1 + fNy * fNz;
Int_t   ixyz3 = ixyz1 + fNz;
Int_t   ixyz4 = ixyz2 + fNz;
fHa[0][0][0] = fcomp->At(ixyz1);    fHa[1][0][0] = fcomp->At(ixyz2);
fHa[0][1][0] = fcomp->At(ixyz3);    fHa[1][1][0] = fcomp->At(ixyz4);
fHa[0][0][1] = fcomp->At(ixyz1 + 1);    fHa[1][0][1] = fcomp->At(ixyz2 + 1);
fHa[0][1][1] = fcomp->At(ixyz3 + 1);    fHa[1][1][1] = fcomp->At(ixyz4 + 1);…
```

Estimated improvement in time ~10 % for reconstruction of simulated data, less for experimental (sample of 500 events).

Thank you for your attention