User Story: Integration of ROOT RNTuple to CMSSW's SoA data structures

Markus Holzer, Leonardo Beltrame,

Andrea Bocci, Felice Pantaleo, Simone Balducci

on behalf of the CMS collaboration

18 Nov 2025 - ROOT Users Workshop 2025

CERN | NexTGen Next Generation Triggers

Next-Generation Trigger Project (NGT)

5-year project, Collaboration between CMS, ATLAS, CERN Theory, IT

Goal: Get more physics information out of the HL-LHC data

Task 3.1.2:

- Development of data-oriented structures
- Better utilization of memory bandwidth and vectorization
- Seamless integration with machine learning models and remote offload through message passing
- Flexibility and maintainability
- Focus on heterogeneous computing

Challenges in CMSSW

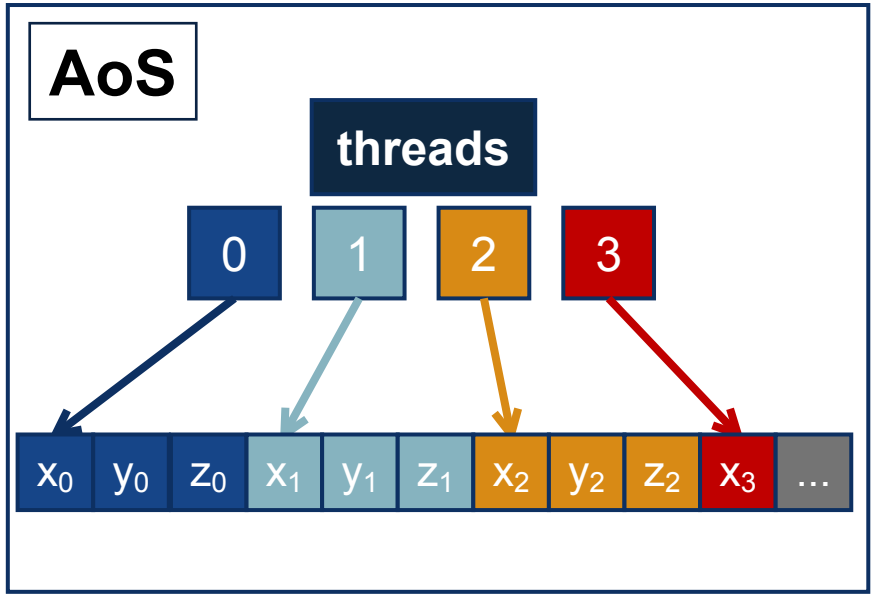
- Choice of Datastructures is essential to achieve good performance
- Multiple hand-optimized implementations in CMSSW
- Inconsistent handling of compile-time vs. runtime sizes
- Single- and Multi-buffer solutions
- Low level optimizations like cache hinting manually implemented
- Compatibility with C++20

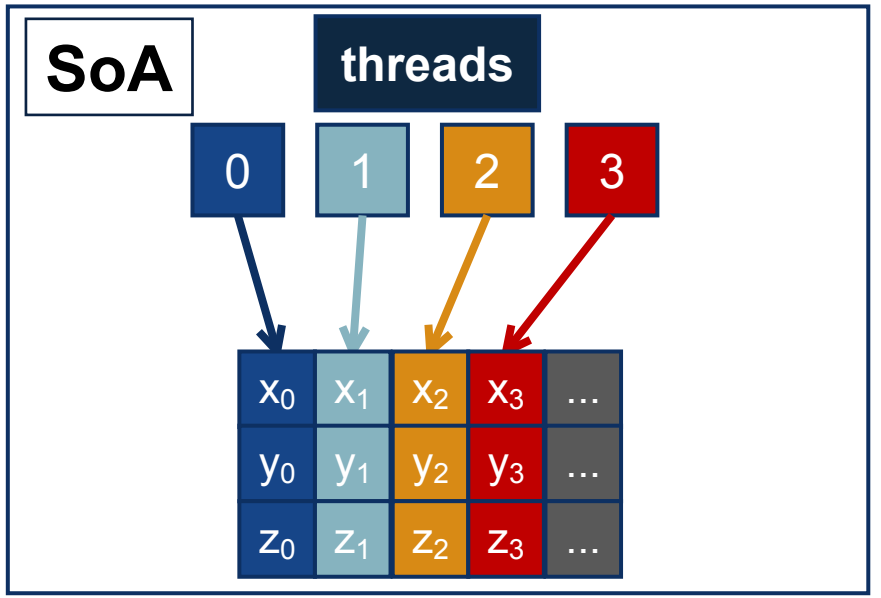
Unified solution for heterogeneous computing is essential

Basics

- Array of struct (AoS): data is stored as continuous array of structures
- Structure of Array (SoA): data is stored in continuous arrays per record

- SoA often more performant choice, especially on GPU hardware
- AoS more natural way to represent data in most languages





Basics

- AoS Intuitive and easy to use
- Aligns with OO programming
- Simple code and interfaces

- SoA less intuitive to implement
- Can lead to more complicated code

AoS

```
struct Point {
    float x;
    float y;
    float z;
};
struct Point points[N];
// access point n
float pn = points[n].x
```

SoA

```
struct Points {
    float x[N];
    float y[N];
    float z[N];
};
struct Points points;
// access point n
float pn = points.x[n]
```

Wish List

- **Unified implementation of SoA Datastructure with high flexibility**
- **Intuitive definition and usage (AoS-like access operators)**
- **Separation of concerns:**
  - Layout: persistent storage
  - View: data access
  - PortableCollection: buffer and memory management

Macro based implementation to generate the Layout and it's View

Technical implementation

- SoA Layout and View generated from Macros using BOOST::PP [1,2]
- Supported in C++20 and without relying on another language
- Each column has it's own datatype
- Eigen objects can be defined directly
- Scalar Values can be added

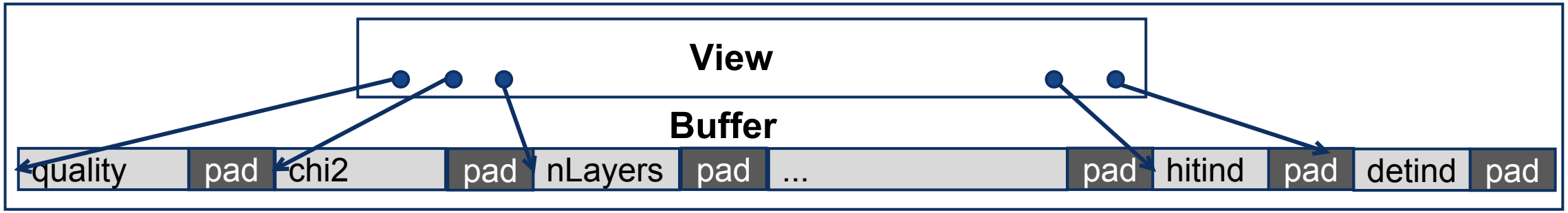
```
  GENERATE_SOA_LAYOUT(TrackSoALayout,
                      SOA_COLUMN(Quality, quality),
                      SOA_COLUMN(float, chi2),
                      SOA_COLUMN(int8_t, nLayers),
                      SOA_COLUMN(float, eta),
                      SOA_COLUMN(float, pt),
                      SOA_EIGEN_COLUMN(Vector5f, state),
                      SOA_EIGEN_COLUMN(Vector15f, covariance),
                      SOA_SCALAR(int, nTracks),
                      SOA_SCALAR(HitContainer, hitIndices),
                      SOA_SCALAR(HitContainer, detIndices))
};
```

https://github.com/cms-sw/cmssw/blob/master/CUDADataFormats/Track/interface/PixelTrackUtilities.h

[1]: The Boost Library Preprocessor Subset for C/C++, https://www.boost.org/doc/libs/1_67_0/libs/preprocessor/doc/index.html
[2]: Eric Cano and Andrea Bocci, Implementation of generic SoA data structures in the CMS software, ACAT 2022

Technical implementation

- Given the Layout, sizes and padding —> a continuous byte buffer can be defined
- View defines pointers into the buffer for memory access —> lightweight object that can be passed to kernels
- Each column adds padding for optimal usage across all platforms
- Single copy for whole buffer, no additional parameters



ROOT serialization

- SoA Layout is serialized by ROOT
- Macro generated function defines reading
- Each column is copied with it's individual size
- Default memory layout used by TTree is not ideal for GPUs

```
template <typename T>
void ROOTReadStreamer(T& onfile) {
    memcpy(quality_, onfile.quality_, sizeof(double) * onfile.elements_);
    memcpy(chi2_, onfile.chi2_, sizeof(float) * onfile.elements_);
    memcpy(nLayers_, onfile.nLayers_, sizeof(int8_t) * onfile.elements_);
    memcpy(eta_, onfile.eta_, sizeof(float) * onfile.elements_);
    memcpy(pt_, onfile.pt_, sizeof(float) * onfile.elements_);
    memcpy(state_, onfile.state_, sizeof(Vector5f::Scalar) *
       stateElementsWithPadding_);
    memcpy(covariance_, onfile.covariance_, sizeof(Vector15f::Scalar) *
       covarianceElementsWithPadding_);
    memcpy(nTracks_, onfile.nTracks_, sizeof(int));
    memcpy(hitIndices_, onfile.hitIndices_, sizeof(int8_t));
    memcpy(detIndices_, onfile.detIndices_, sizeof(int64_t));
}
```

Direct SoA reading and writing with RNTuple would be desirable

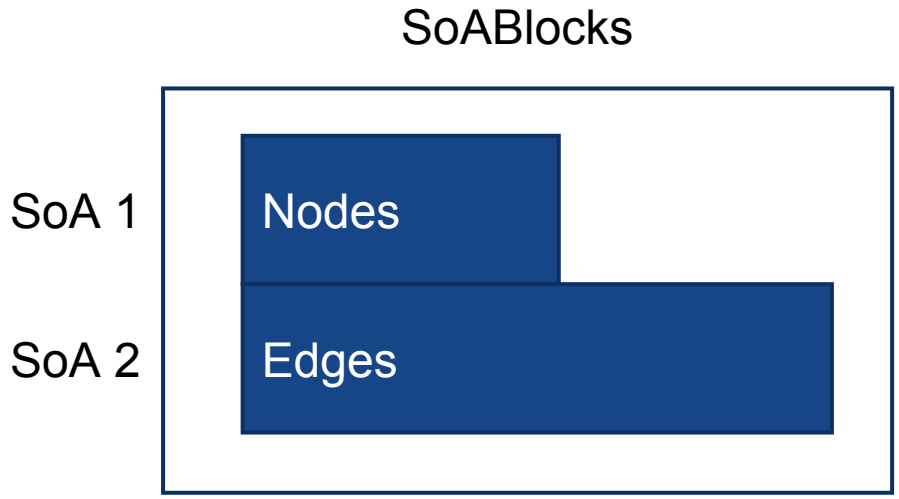
Combination with RNTuple

Goal: Enable CMSSW read and write to RNTuple

- **Columnar storage of AoS does not align well with CMSSW SoAs**
- **Columns with different sizes are not allowed**
- **No interface to read into the memory area of CMSSW**
- **Copying of memory should be avoided as much as possible**

SoA Blocks [1]

- SoAs with different sizes can be combined using SoA Blocks
- Allows to represent more complex structures like Graphs
- Columns are guaranteed to have the same size
- Scalar values are integrated in each block



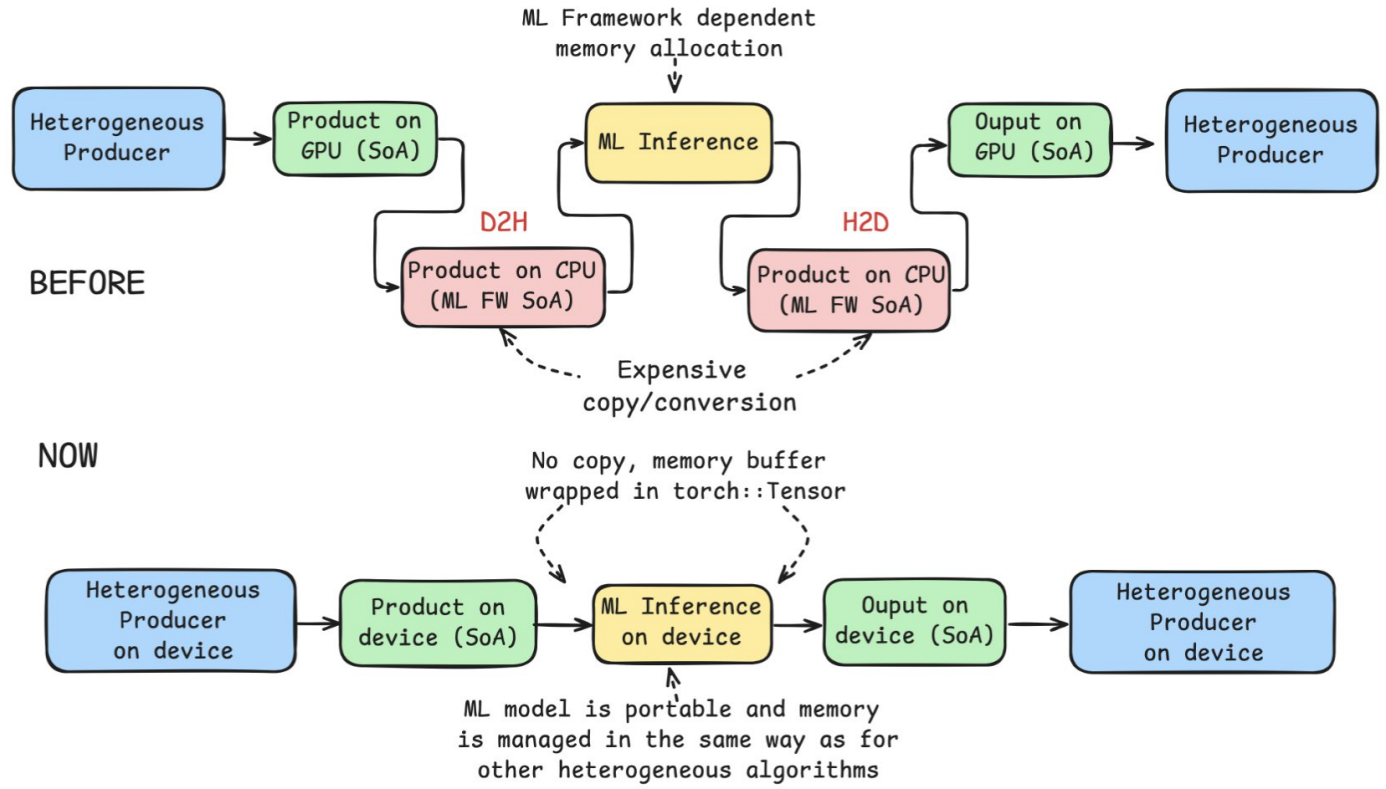

```
GENERATE_SOA_LAYOUT(SoAEdges, SOA_COLUMN(Edge, edges))
GENERATE_SOA_LAYOUT(SoANodes, SOA_COLUMN(Node, nodes))

GENERATE_SOA_BLOCKS(SoAGraphTemplate,
                    SOA_BLOCK(edges, SoAEdges),
                    SOA_BLOCK(nodes, SoANodes))
```

[1] L. Beltrame, Evolution of data structures for heterogeneous reconstruction in CMSSW. Next Generation Triggers, CERN, 2025.

Machine Learning in CMSSW [1]

- **Direct usage of SoA Data structures for ML**
- **Avoid copy between host and device**
- **Need for efficient storage to load data for ML models from memory to GPU buffer**



[1] C. Zeh, Efficient Data Movement for Machine Learning Inference in Heterogeneous CMS Software. Next Generation Triggers, CERN, 2025. doi: 10.17181/7272e-39998.

Conclusion

- CMSSW relies on the SoA Datastructure
- In Production already during run 3
- Direct SoA reading and writing would enable multiple benefits, like direct inference from RNTuple
- Having Columns with different sizes in the buffer gives a great flexibility that CMSSW relies on
- CMSSW SoAs can be represented as groups of columns with the same size

References & acknowledgments

- Next Generation Triggers, *“NGT Evaluation Report - 2024”*, doi: 10.17181/nkwjc-7pa06
- The Boost Library Preprocessor Subset for C/C++, url: https://www.boost.org/doc/libs/1_67_0/libs/preprocessor/doc/index.html
- Eric Cano and Andrea Bocci, Implementation of generic SoA data structures in the CMS software, ACAT 2022
- L. Beltrame, Evolution of data structures for heterogeneous reconstruction in CMSSW. Next Generation Triggers, CERN, 2025.
- C. Zeh, Efficient Data Movement for Machine Learning Inference in Heterogeneous CMS Software. Next Generation Triggers, CERN, 2025. doi: 10.17181/7272e-39998.

This work has been partially funded by the Eric & Wendy Schmidt Fund for Strategic Innovation through the CERN Next Generation Triggers project under grant agreement number SIF-2023-004.


CERN
NexTGen
Next Generation Triggers