Task 3.1.2: Efficient data structures for heterogeneous event reconstruction

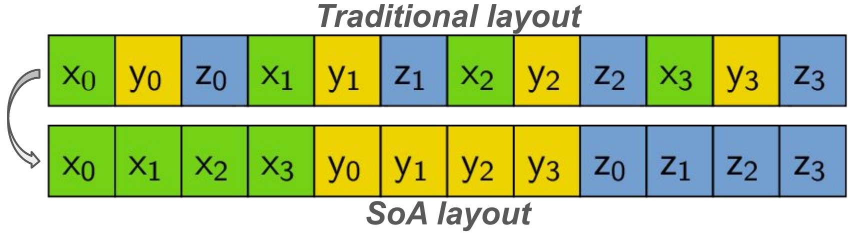
Leonardo Beltrame, Andrea Bocci, Eric Cano,
Felice Pantaleo, Davide Valsecchi for WP3

felice@cern.ch

CERN | NexTGen Next Generation Triggers | CMS

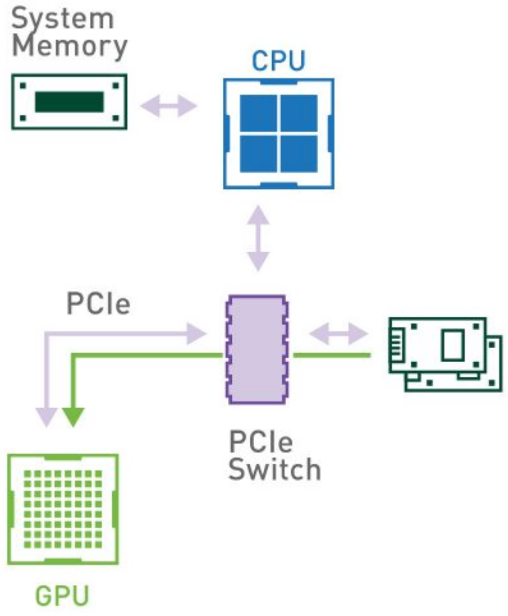
Motivation for Efficient Data Structures

Data structures in heterogeneous environments might become the bottleneck when the cost of sequential copies and conversions are not negligible wrt to parallel algorithms running on GPU



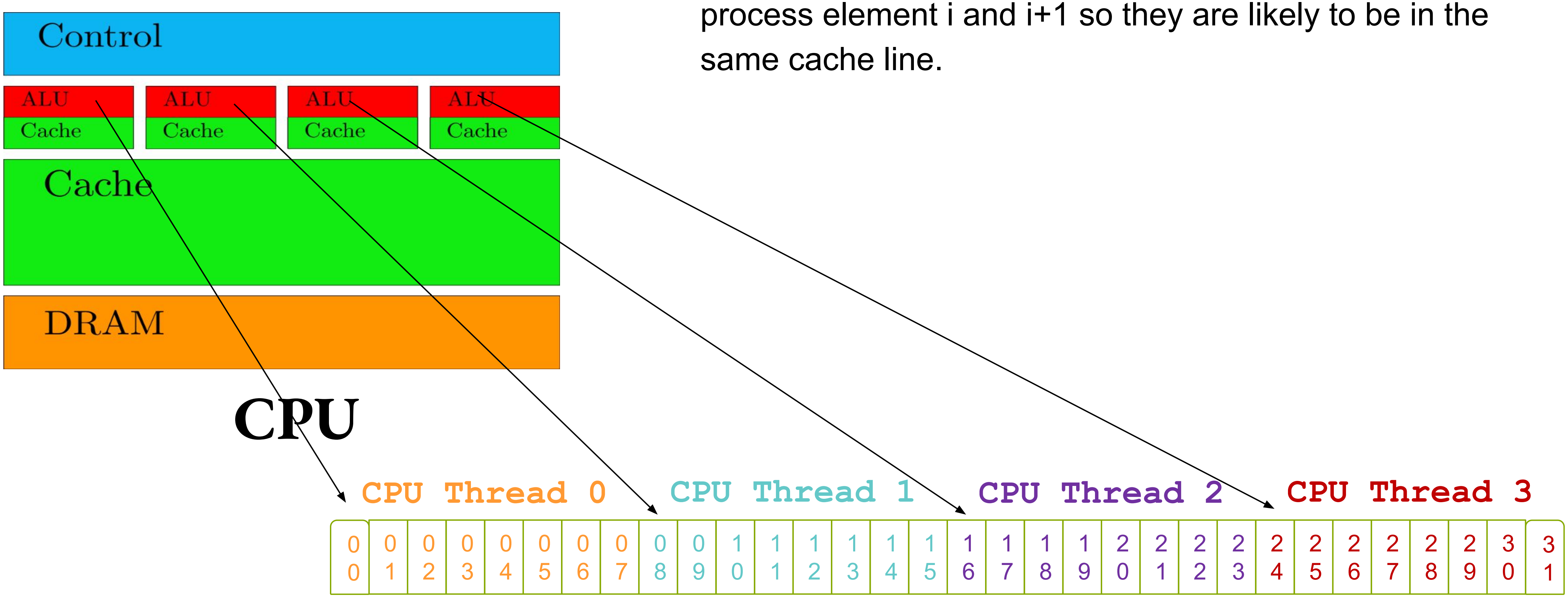
Goals:

- Efficient memory access patterns
- Seamless integration with machine learning models and remote offload through message passing
- Flexibility and maintainability



Memory access patterns: cached

For optimal CPU cache utilization, a thread a should process element i and i+1 so they are likely to be in the same cache line.



Memory access patterns: coalesced

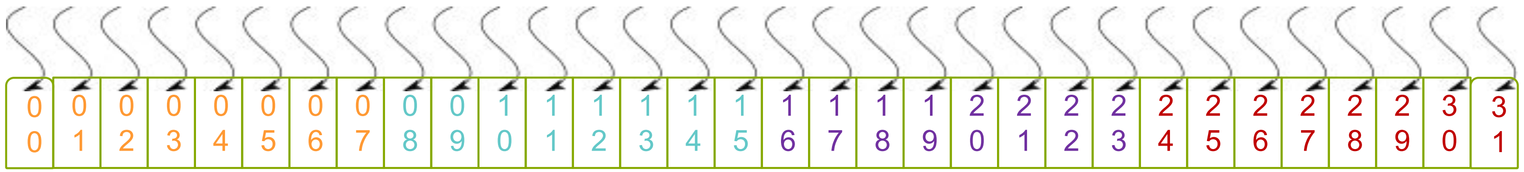
- L1 data cache is shared among Arithmetic Logic Units (ALUs)
- ALUs work in groups of 16, 32, 64 threads called **warps**
- Warps execute instructions in a SIMD (Single Instruction, Multiple Data) fashion

Memory Access Patterns:

- Optimal when threads in a warp access consecutive memory addresses
- **Coalesced Memory Access:**
  - Thread `a` accesses element `i`
  - Thread `a+1` accesses element `i+1`

Performance Implications:

- Coalesced access maximizes memory throughput
- Non-coalesced or cached access patterns can result in significant performance loss (up to an order of magnitude)
  - If a load is issued by each thread, they have to wait for all the loads in the same warp to complete before the next instruction can execute



Challenges in CMSSW SoA Implementations

Need for Standardization:

- Heterogeneous code is becoming ubiquitous in CMSSW
- Standardizing data structures is crucial for maintainability

Previous Issues:

- Multiple ad-hoc SoA implementations
- Inconsistent handling of compile-time vs. runtime sizes
- Multiple memory allocations per SoA
- Inefficient data transfers between host and device

Any approach must be compatible with C++20, the current standard in CMSSW

Generic SoA with Boost::PP

- Automate SoA definition with macros
- Use Boost Preprocessor (Boost::PP) library
- Support for runtime-sized SoAs
- Generate layouts and views automatically
- Simplify user experience and code maintenance
- Ensure compatibility with C++20 standard

SoA Layout Declaration and Access

Example SoA layout declaration:

```
GENERATE_SOA_LAYOUT(SoAHostDeviceLayoutTemplate,
    SOA_COLUMN(double, x),
    SOA_COLUMN(double, y),
    SOA_COLUMN(double, z),
    SOA_EIGEN_COLUMN(Eigen::Vector3d, a),
    SOA_EIGEN_COLUMN(Eigen::Vector3d, b),
    SOA_EIGEN_COLUMN(Eigen::Vector3d, r),
    SOA_SCALAR(const char*, description),
    SOA_SCALAR(uint32_t, someNumber))
```

AoS style preserved with SoA efficiency

```
SoAHostDeviceLayout::View h_soav{h_soa};
h_soav.x()[i] = /* value */;
h_soav[i].a() = /* Eigen vector */;
```

Flexible SoA Composition

Dynamic Data Requirements:

- Different algorithms may require different subsets of data
- Avoid carrying unnecessary data through the processing pipeline

Performance Optimization:

- Minimize data copying to reduce overhead
- Enable efficient use of memory and bandwidth

Customization:

- Create tailored SoAs combining relevant data from existing SoAs
- Customize data structures for specific algorithms without incurring overhead

Flexible SoA Composition demonstrator

- **PhysicsObject SoA:** Contains `x`, `y`, `z`, `detectorType`
- **PhysicsObjectExtra SoA:** Contains PCA decomposition eigenvalues, eigenvectors, and `direction`

Create a new SoA containing only `x`, `y`, `z`, and `direction` without copying data

Treat columns as independent entities

- Columns own information about data type and size
- Similar to `alpaka::View` in functionality
- Efficient data copies and transfers
- Greater flexibility in data management

```
// Assuming existing SoAs
PhysicsObjectSoA physicsObjSoA(numElements);
PhysicsObjectExtraSoA physicsObjExtraSoA(numElements);

// ...

GENERATE_SOA_LAYOUT(CombinedPhysicsObjectSoA,
    SOA_COLUMN(double, x),
    SOA_COLUMN(double, y),
    SOA_COLUMN(double, z),
    SOA_EIGEN_COLUMN(Eigen::Vector3d, direction));

// Create the combined SoA
const auto phObj = physicsObjSoA.records();
const auto phObjExtra = physicsObjExtraSoA.records();

CombinedPhysicsObjectSoA combinedSoA(
    phObj.x(),
    phObj.y(),
    phObj.z(),
    phObjExtra.direction());
```

Advantages of the Demonstrator

- **Flexibility:**
  - Select and combine only the necessary data
  - Tailor data structures for specific algorithms
- **Efficiency:**
  - Avoid unnecessary data copies
  - Optimize memory usage and data transfers
- **Integration with Framework:**
  - Potential to optimize data transfers by migrating only required columns
  - Under discussion with CMSSW core team for integration as persisting these Composed SoAs or moving them across device would require a redesign of the edm::Ref to become heterogeneous

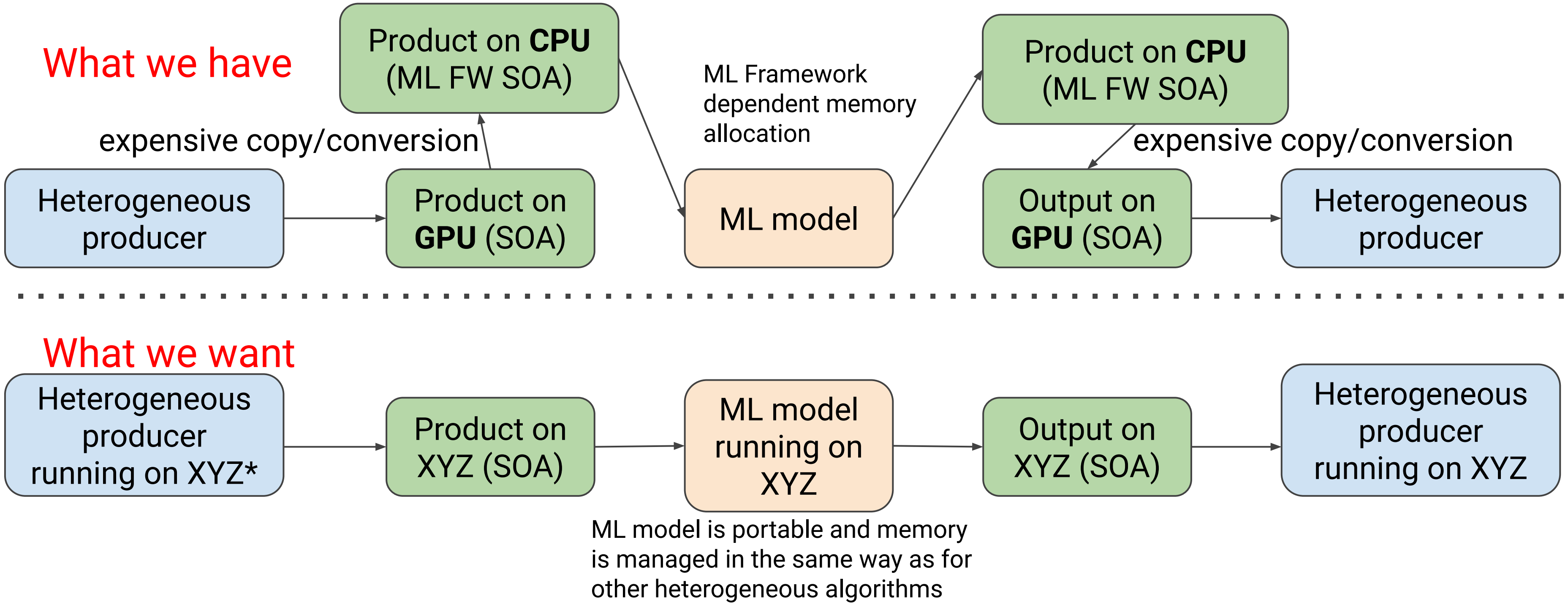
SoA and Machine Learning Integration

Direct Interface to Heterogeneous ML Models

- Reduced overhead in data preparation
- Improved inference performance
- Efficient data feeding into models

Collaboration with Task 1.7, EP-SFT and CMS Machine Learning Group

- Ensuring compatibility with ML workflows
- Optimizing data structures for inference tasks



Integration with Machine Learning Models

Developing a demonstrator interfacing Composed SoA with ML models

- Machine learning models often expect data as contiguous blobs
- Need to pass SoA data efficiently to models
- **Solutions:**
    - **Aggregate Method:**
        - Create a `aggregate()` method for the SoA
        - Allocate a single contiguous buffer
        - Copy selected columns into the buffer
        - Pass the buffer as a single blob to the ML model
    - **Modify Model Input Layer:**
        - Adjust the ML model to accept multiple tensors (one per column)
        - Avoids the need to copy data into a single buffer

Conclusion

- Efficient data structures like our generic SoA are essential for optimizing performance in heterogeneous computing environments. Our approach addresses the challenges faced in previous implementations, providing flexibility, maintainability, and efficiency.
- The ability to flexibly compose SoAs allows us to create tailored data structures for specific algorithms without unnecessary data copying.
- We're continuing to work on integrating this approach with the CMSSW framework to further enhance its utility, while working in strong collaboration with Task 1.7, CMSSW Framework core team

CERN

NexTGen

Next Generation Triggers