Enabling Heterogeneous Architectures in JANA2

Nathan Brei

Jefferson Lab

May 13, 2026

Model training on GPU (Priority: Low)

Current status

- Models are trained outside of JANA2
- Model training on a GPU is fundamentally a critical section, which limits the performance benefits of doing this from inside JANA2
- Podio data is laid out in rows, not columns, which makes this harder
- Offloading a JFactory's computation and subsequently running other JFactories on the results constitutes a cycle in the underlying arrow topology, which makes this harder

Next steps

- I can't solve the abstract problem until I've solved at least one concrete instance
- A performance analysis of training from Podio vs RDataFrames, inside a JEventProcessor vs outside JANA

AI clustering for the NPS detector

- Courtesy of Chi Kin Tam and Dmitry Romanov
- Self-contained, simple example to improve upon
- Motivates some JANA changes: https://github.com/JeffersonLab/JANA2/pull/495
- **Generalize later**
- **Work-in-progress**

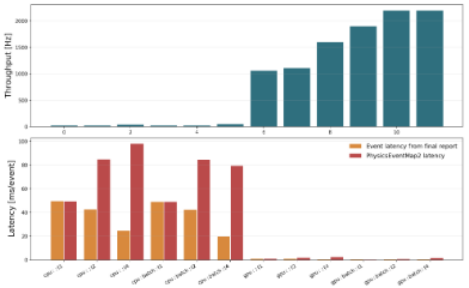
Latency Test

Configuration

- Multi-thread management solely from `JANA2`, always configure `ONNX` to use single thread.
- **random** event source with only ML reconstruction factory.
- NVIDIA A800 node

Test Cases

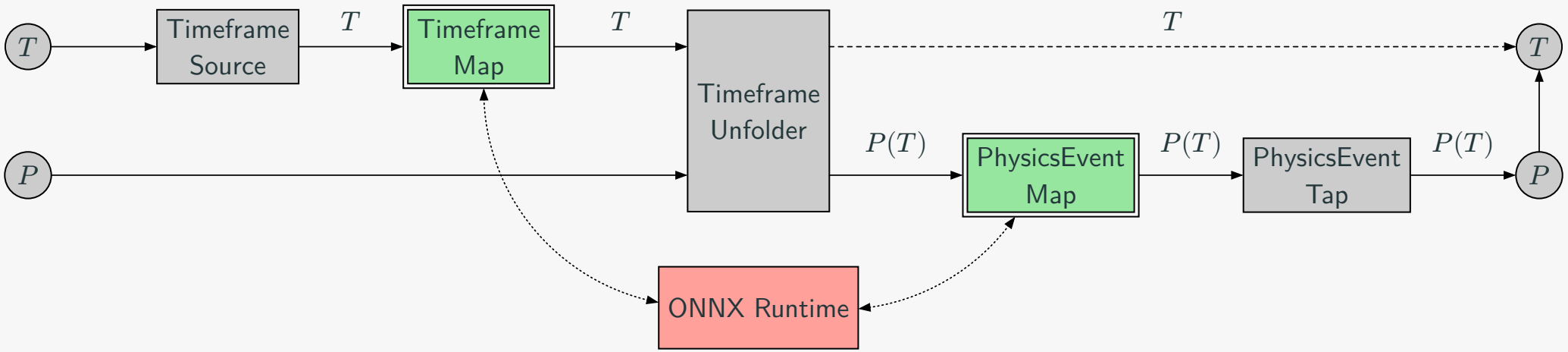
- compare CPU, GPU runtime
- `nthreads=1,2,4` → surprisingly low impact on latency
- batching : accumulate events in factory level
- GPU with batching significantly increase in throughput (~ 2 kHz)
  - total latency does not include traditional reconstruction time ← bottleneck.



Requirements

1. Batching
   - Batch sizes have a large effect on an offloaded algorithm's performance
   - JANA2 JFactories operate on exactly one JEvent (e.g. PhysicsEvent, Timeframe, etc) at a time
   - Physics events often contain small amounts of data
   - Batch sizes should be driven by the algorithm, not by the physics
2. Data layout
   - The data layout preferred by the offloaded algorithm almost certainly doesn't match what the data model provides
   - The data layout is tightly coupled to the specific algorithm
3. Multiple algorithms in a chain might be offloaded
   - If they are connected, it would be desirable to fuse them to avoid extra data movement
   - Offloaded algorithms should be substitutable with their CPU-based versions
4. Framework ergonomics
   - Performance analysis should be straightforward
   - Users should not need their own locks

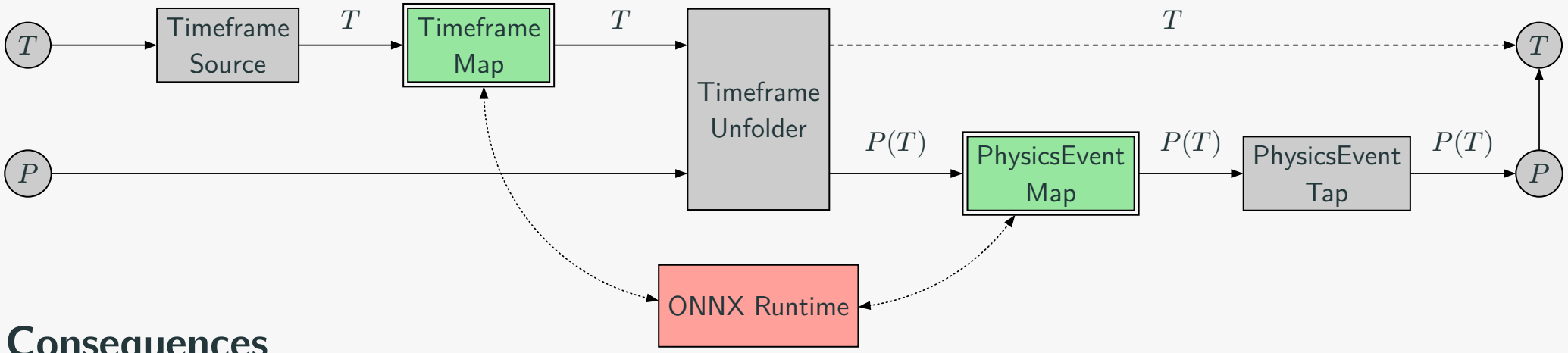
Basic JFactories



Idea

- Treat offloaded algorithms like any other and wrap them in a JFactory
- JANA2 will be run inside the parallel `Map` arrow(s)
- In `JFactory::Process()`, pack the data as needed, submit it to an inference server such as ONNX runtime, block until that task finishes, and finally unpack
- Oversubscribing threads may reduce the performance hit of waiting on the external scheduler

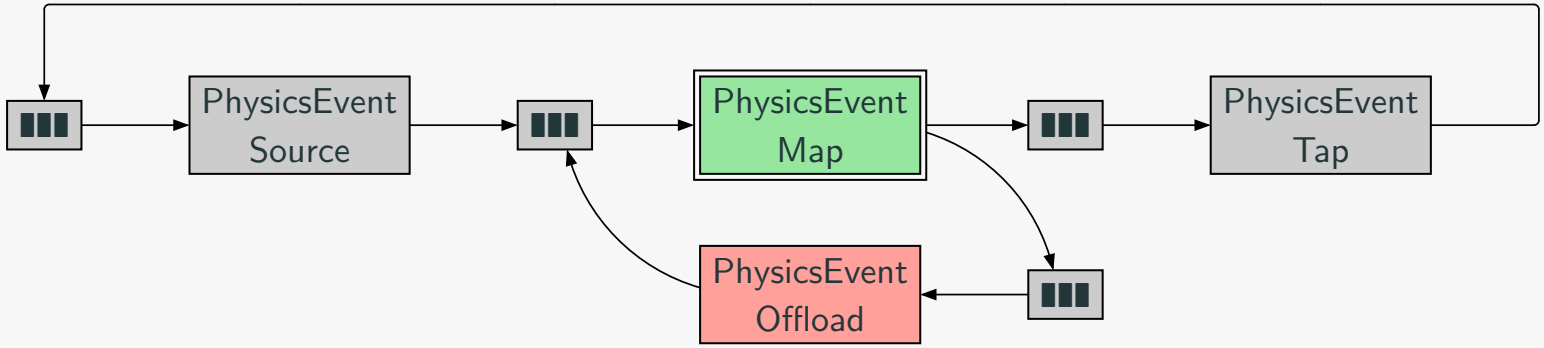
Basic JFactories



Consequences

Batching	▪ JFactories can only process individual `PhysicsEvents`/`Timeframes`/etc ✗
Data layout	▪ The data can be packed and unpacked within each call to `Process()` ✓
Multiple algorithms	▪ No fusion possible between adjacent offloaded factories ✗ ▪ Easy to substitute factories with CPU version ✓
Ergonomics	▪ Blocking `Process()` while delegating to an external scheduler, such as ONNX runtime, obfuscates performance analysis ✗

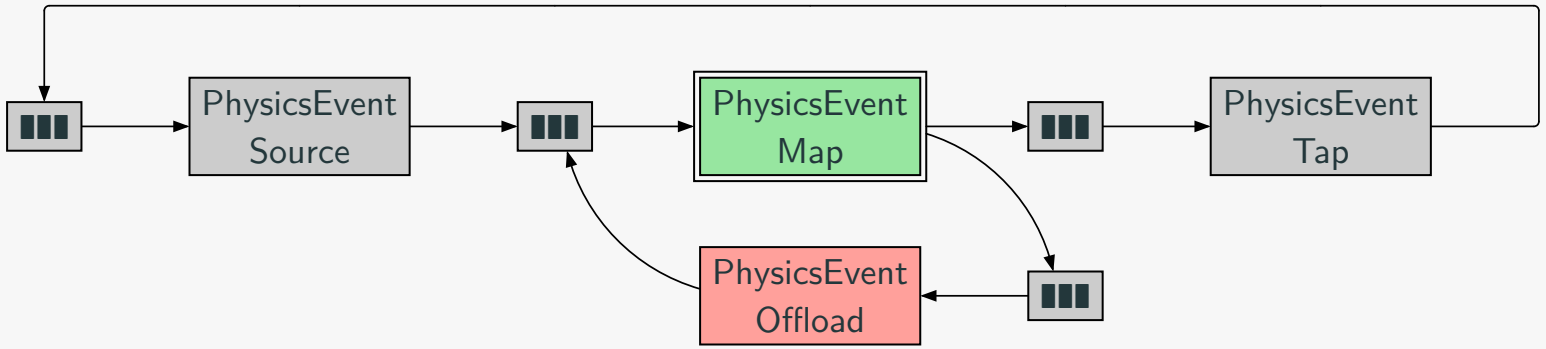
Idea: Map/Offload loop



Idea

- Represent the handover of JEvents from the JANA2 scheduler to the ONNX runtime scheduler explicitly in the JANA2 processing topology
- Each JEvent is passed back and forth between `Map` (which runs CPU factories), and `Offload`, which runs ONNX inferences, until algorithm chain is complete
- "Continuation" object tracks the algorithm chain's progress so that `Map` and `Offload` each know what factories must execute next
- Limitations: Non-lazy. Either the algorithm chain has to be specified ahead of time, or computed by traversing the graph of declared `Inputs`

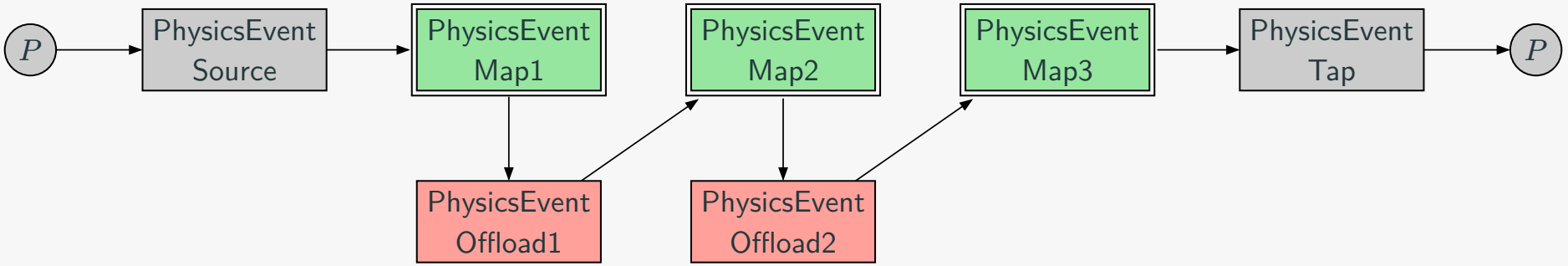
Idea: Map/Offload loop



Consequences

Batching	▪ JFactories can only process individual `PhysicsEvents`/`Timeframes`/etc ✗
Data layout	▪ Does data packing/unpacking happen on MapArrow or OffloadArrow?
Multiple algorithms	▪ No fusion possible between adjacent offloaded factories ✗ ▪ Easy to substitute factories with CPU version ✓
Ergonomics	▪ JANA2 scheduler no longer fights with ONNX scheduler ✓ ▪ Performance analysis is still muddled when there are multiple ONNX factories in the chain ✗

Idea: Unrolled Map/Offload Loop



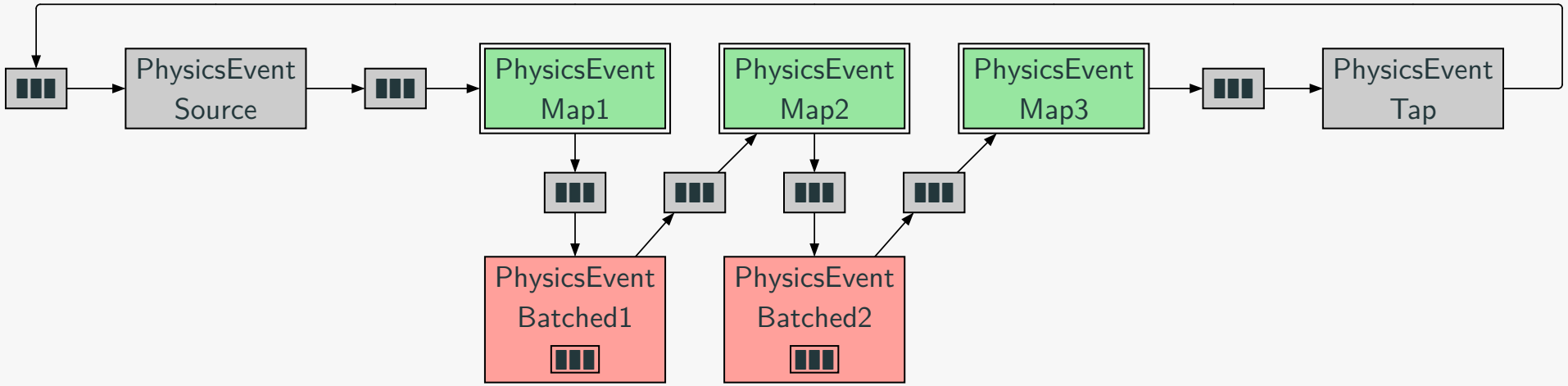
Idea

- The cycle between MapArrow and OffloadArrow is unrolled
- This considerably simplifies the machinery, e.g. no Continuation object needed

Consequences

- Ergonomics: Crystal-clear performance analysis
- Data layout: Who does the packing/unpacking?
- Multiple algorithms: Fusion is possible (although not with JFactories per se)

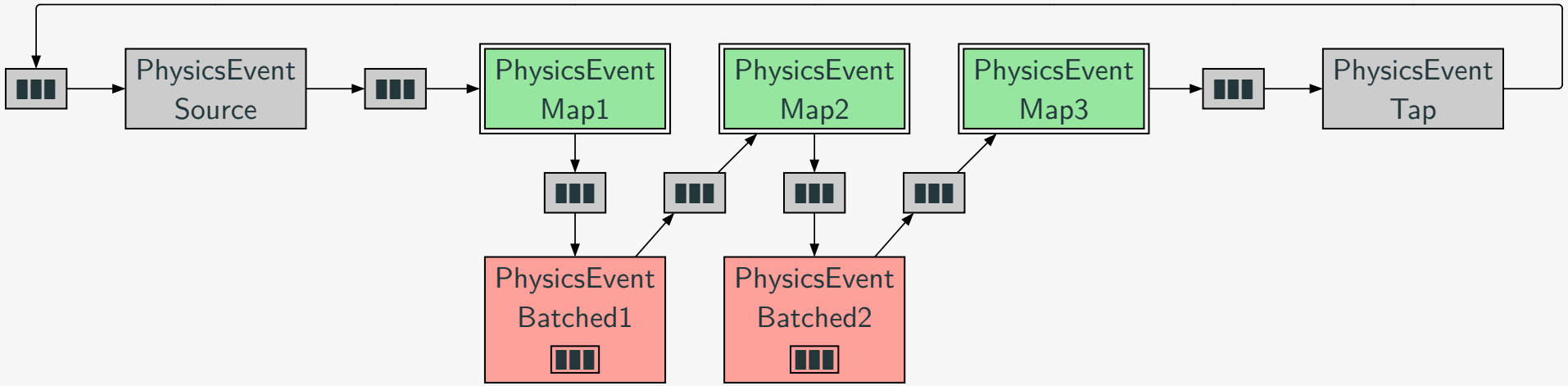
Idea: Batched arrows



Idea

- MapArrow, OffloadArrow, etc all pop one JEvent off their input queue, process it, and immediately push it to an output queue. BatchedArrow holds on to as many JEvents as it needs in order to fill the batch.
- This requires a different abstraction from JFactory. Packing and unpacking the batches become part of the interface, with dedicated callbacks
- This permits fusion of adjacent offloaded algorithms as long as they use the same batch size

Idea: Batched arrows



Consequences

Batching	▪ Algorithms can process arbitrary batches ✓
Data layout	▪ Data layout is encapsulated ✓
Multiple algorithms	▪ Fusion possible, although requires using the same batch size ✓ ▪ Substituting factories with their CPU versions is a bit weirder ×
Ergonomics	▪ Clear interface, relationship between schedulers, and performance analysis ✓

Idea: Custom JArrows

Goal:

Make it easier to use custom JArrows in user code. Imagine if all of the topologies described in this talk could be implemented in EICrecon without requiring any changes to JANA2 at all!

Next steps:

- Configure the JANA2 processing topology via external wiring file
- Create JArrowGenerator?

```C++
void configure_batched_topology(JTopologyBuilder& builder,
                                JComponentManager& component_manager) {

    auto* src_arrow = new JSourceArrow("PhysicsEventSource",
                                       JEventLevel::PhysicsEvent,
                                       component_manager.get_evt_srces());

    auto* batched_arrow = new BatchedArrow("PhysicsEventBatch",
                                           JEventLevel::PhysicsEvent, 8);

    auto* tap_arrow = new JTapArrow("PhysicsEventTap",
                                    JEventLevel::PhysicsEvent,
                                    component_manager.get_evt_procs());

    builder.AddArrow(src_arrow);
    builder.AddArrow(batched_arrow);
    builder.AddArrow(tap_arrow);
    builder.ConnectPool("PhysicsEventSource", "in",
    JEventLevel::PhysicsEvent);
    builder.ConnectQueue("PhysicsEventSource", "out", "BatchedArrow", "in");
    builder.ConnectQueue("BatchedArrow", "out", "PhysicsEventTap", "in");
    builder.ConnectPool("PhysicsEventTap", "out", JEventLevel::PhysicsEvent);
}
```

Thank you!