JANA2 Update: Hierarchical vs Batched Events

Nathan Brei

Jefferson Lab

August 19, 2026

Latest JANA2 release: v2026.03.00

Dependencies

- Now builds with ROOT v6.40
- Adds optional bindings for **Perfetto**
- Swaps out optional dependency `libzmq` for `cppzmq`
- Building/installing examples is now optional

Bugfixes

- Parameter parsing conflict: JLogger::Level vs spdlog level
- Test cases run in the build directory without pulling in artifacts from the install directory
- Improved logging of swallowed exceptions

Refactoring

- Redesign of the TopologyBuilder interface
  - Now supports multi-level event sources
  - Topologies which include an Unfolder no longer also pull in a no-op Folder

Recap of hierarchical events

- **Event Level** := An enum tag attached to each JEvent container
  - Each JEvent holds data associated with a physics **Interval of Validity**
  - e.g. `Run`, `SlowControls`, `Timeframe`, `PhysicsEvent`, `Subevent`
- Establishes a "data family tree", e.g. `PhysicsEvent` is a child of `Timeframe`
- Enforces data isolation for algorithms: Parents accessible but **not** siblings
- Provides memory lifetime management and bounds

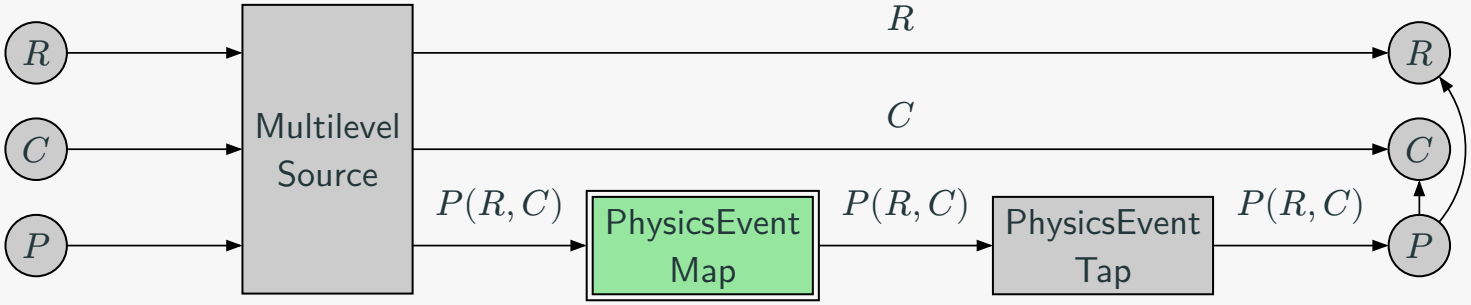
How this is actually modelled

```
1  JEvent ~= (event_nr, event_level,  {collection_name : JDatabundle},  {parent_level : JEvent} )
2  JDatabundle ~= (optional collection,  optional factory)
```

Latest updates

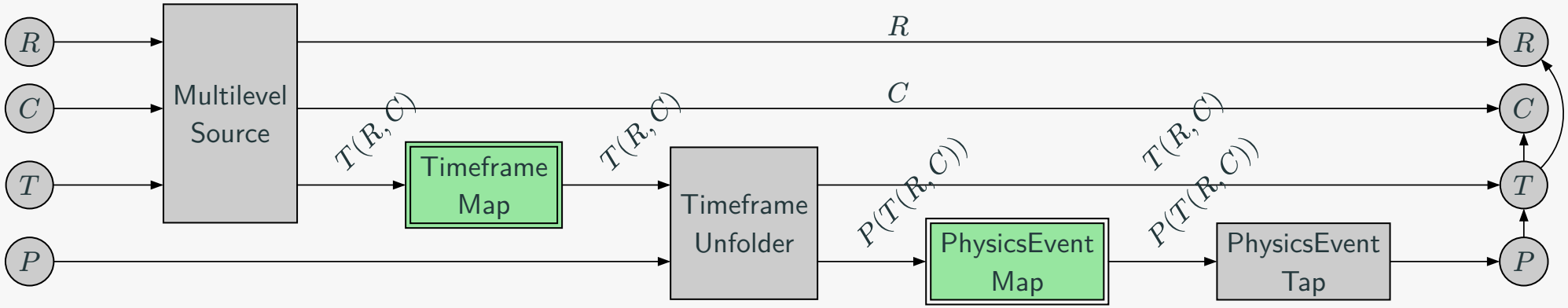
- `Timeframe` added to enum values, will show up in next release
- `JEventKey` := P:21(T:1,R:1234) encodes both the generalized event/run numbers AND the tree
- Multilevel sources are usable now

Multilevel sources are usable now!



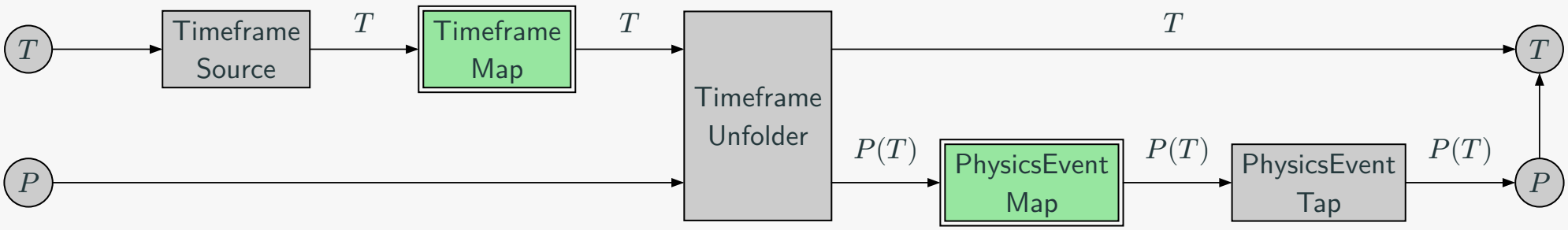
- Multilevel source emits PhysicsEvents with the most recent Run and Control events attached as parents
- The downstream components (JEventProcessors, J{Omni}Factories) don't require any modification
- R, C, P pools control number of Runs, SlowControls, and PhysicsEvents in flight.
- The PhysicsEvent pool forwards all parent events to their respective pools
- This topology also supports merging streams from separate input files/sockets

Multilevel sources with timeframe splitting



- Scenario: Begin-of-run and control events are interleaved with timeframes
- Composes with JEventUnfolder (e.g. TimeframeSplitter) exactly as you'd expect
- Downstreams (e.g. factories) transparently receive $P(T(R, C))$ instead of $P(T)$
- Downstream components can access R, C data by declaring optional inputs

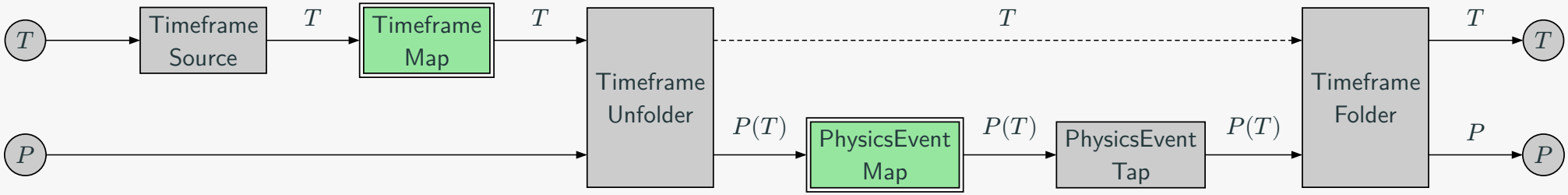
Recap of unfolder



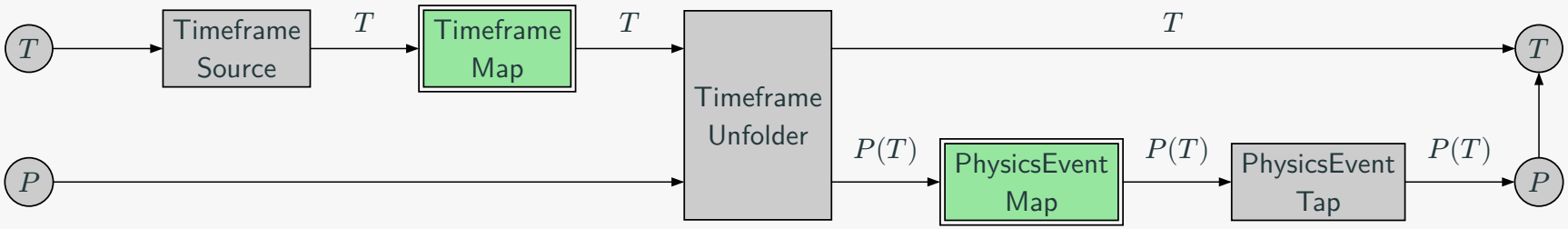

```C++
class JEventUnfolder {
    enum class Result { NextChildNextParent, NextChildKeepParent, KeepChildNextParent };
    virtual Result Unfold(uint64_t parent_nr, uint64_t child_nr, uint64_t item_nr) {
      // Read parent (Timeframe) collections via Input members
      // Write child (PhysicsEvent) collections via Output members
      // Update a cursor into the parent
      // Decide whether this cursor has reached the end of the parent
      return Result::NextChildKeepParent;
    };
```

Removing the no-op Folder

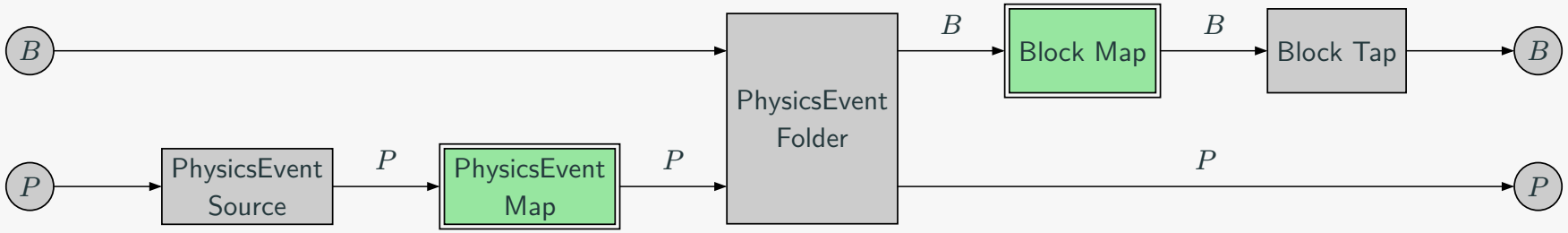
Before



After

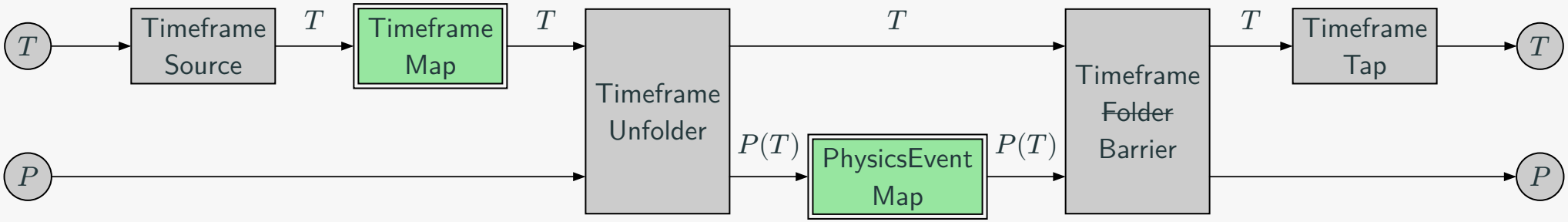


Recap of folder

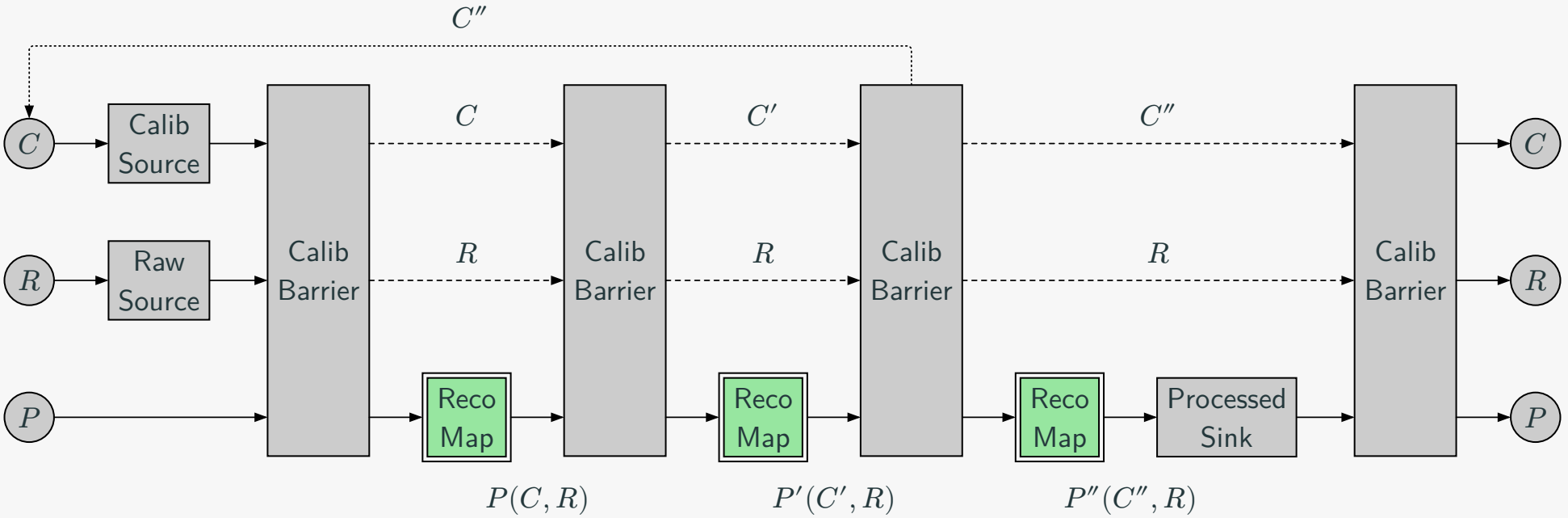


- Folding over **existing** parents is different from finding **new** parents
- What a folder does:
  1. Calculates collections on the parent using collections on all children (generally)
  2. Maintain cursor logic (new parents only)
  3. Detach parents from children, recycle the children (sometimes)
  4. Opens a context where the parent can be modified while the children wait (sometimes)
- **Folder is not quite ready to use yet**
  - Current implementation always detaches and recycles the child
  - Correctness requires enforcing that the child is not being modified while the parent is
  - Folder needs to either detach children, or be followed by a barrier

Folding over existing parents = Barrier



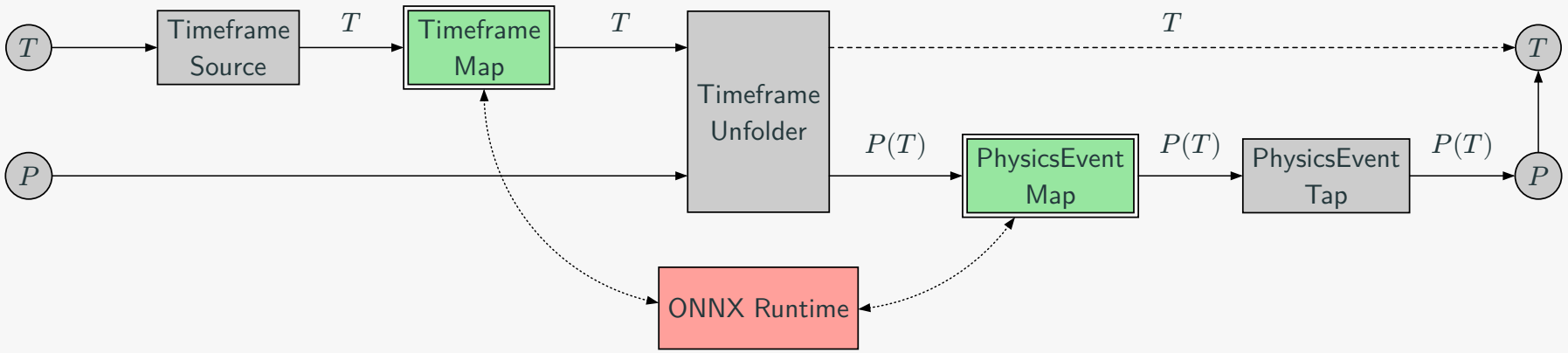
Sketch of possible streaming calibration topology



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

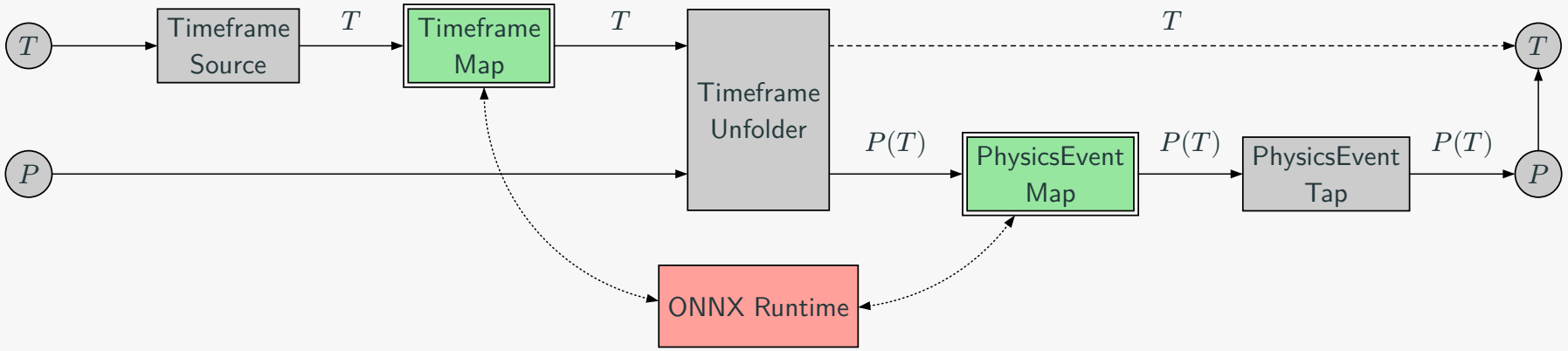
Current state of the art



Idea

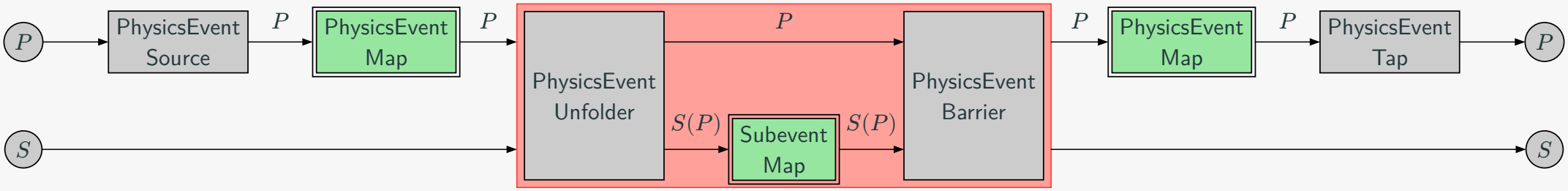
- Treat offloaded algorithms like any other and wrap them in a JFactory
- JANA2 will be run inside the parallel `Map` arrow(s)
- In `JFactory::Process()`, pack the data as needed, submit it to an inference server such as ONNX runtime, block until that task finishes, and finally unpack
- Oversubscribing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

Offloading from JFactories directly



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 ✗ ▪ ONNX multithreading becomes the users' problem

Offloading using subevents



Batching	▪ Batch size can be made arbitrary ✓ ▪ Batch can be smaller than a single PhysicsEvent or larger, but NOT both ✗
Multiple algorithms	▪ Fusion very possible using subevent-level factories ✓
Ergonomics	▪ Steep learning curve ✗ ▪ Layout-specific code scattered across three different components ✗ ▪ Confusion because "Subevent" means different things to different algorithms
Performance	▪ 3 additional components + subevent pool put pressure on JANA2 scheduler ▪ PODIO collection + frame + JEvent operations are **heavy**

Let's fix the ergonomics problems first

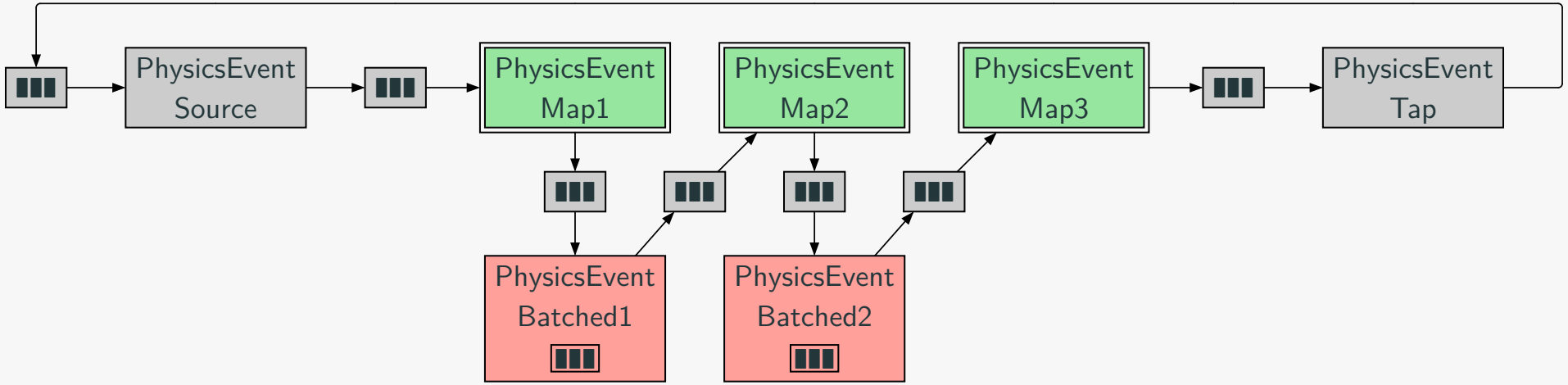
Tentative interface

C++

```
class JOffload {

    virtual std::pair<bool, size_t> Pack(size_t max_tasks) {
      // Read input collections from current physics event
      // Append up to max_tasks onto input tensor
      // Return (more_tasks_in_evt, tasks_placed)
    }
    virtual void Execute() {
      // input_tensor => output_tensor
    }
    virtual void Unpack(size_t cursor, size_t tasks_count) {
      // Retrieve tasks_count tasks from output tensor;
      // Populate output collections on current physics event
    }
};
```

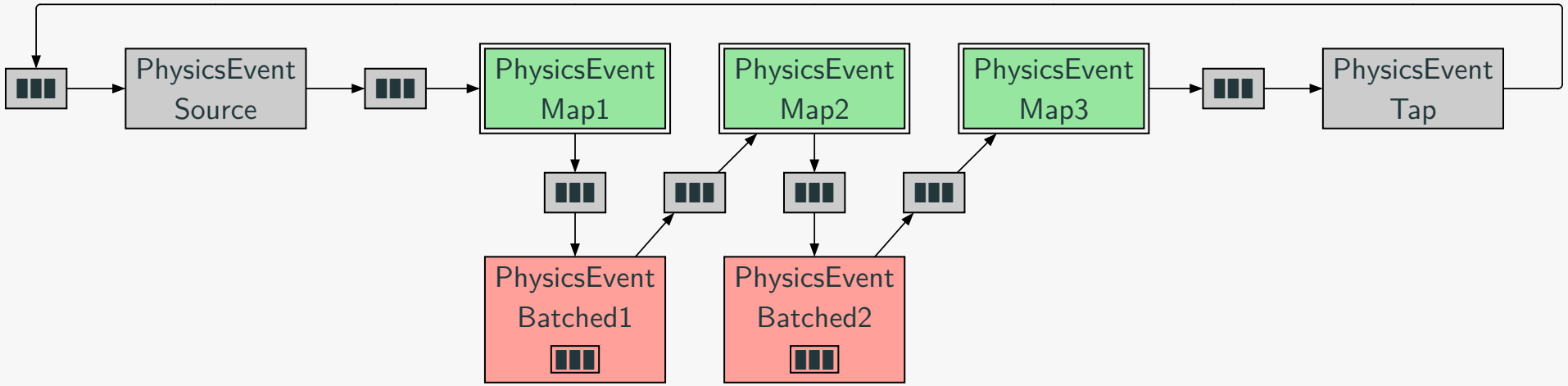
Offloading using batched events



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 ✓

Conclusion

- Hierarchical events are useful for representing data according to physics domain knowledge
- Batched events are useful for tuning an algorithm that runs on heterogeneous hardware, whose optimal batch size has nothing to do with the underlying physics domain
  - Relaxes data isolation requirement
  - Relaxes data closure requirement
- JOffload provides a simpler abstraction that lets users migrate a JFactory to heterogeneous hardware
- JBatchedArrows are a JANA-internal optimization over Unfolder/Map/Barrier arrows that provides better mechanical sympathy

Thank you!