Jefferson Lab

JANA2 Updates for Streaming Computing

Nathan Brei
nbrei@jlab.org

ePIC Collaboration Meeting
July 14, 2025

Motivation

1. Preparing for ePIC calibrations
   - This is a golden opportunity to revisit and improve the underlying design
2. Challenges relating to intervals of validity
   - Propagating file metadata is nontrivial because not all events in the stream necessarily came from the same file
   - Loading heavy resources (e.g. magnetic field maps, ML model weights) exactly once, associating physics events with the correct version, and evicting them from cache
3. Re-establishing closure principle
   - Any data stream JANA2 can write, JANA2 should also be able to read
   - JEventUnfolder produces hierarchically structured event streams, which (until now) couldn't be reread by a JANA2 event source without flattening
   - Important for distributing the computation across nodes

Revisiting calibrations

- Inputs, Outputs, and Services are declarative and injected; calibrations are not…
- ChangeRun() is called when the factory encounters an event with a different run number
- Inside ChangeRun(), the user code may manually retrieve calibration values from one or more Services
- These values are stored directly as members of the factory
- Service is responsible for caching

```c++
#include <JANA/Components/JOmniFactory.h>
#include <JANA/Calibrations/JCalibrationManager.h>
#include <datamodel/CalorimeterClusterCollection.h>

class Cluster_fac_filtered: public JOmniFactory<Cluster_fac_filtered>{
private:
    PodioInput<CalorimeterCluster> m_clusters_in {this};
    PodioOutput<CalorimeterCluster> m_clusters_out {this};
    Service<JCalibrationManager> m_calib_manager {this};
    double m_threshold = 100.0;
public:
    void Configure() {}

    void ChangeRun(int32_t run_nr) {
        m_calib_manager->GetJCalibration(run_nr)
                       ->Get("MyCAL/cluster_threshold", m_threshold);
    }

    void Execute(int32_t /*run_nr*/, uint64_t /*evt_nr*/) {
        m_clusters_out()->setSubsetCollection(true);
        for (auto cluster : *m_clusters_in()) {
            if (cluster.getEnergy() >= m_threshold) {
                m_clusters_out()->push_back(cluster);
            }
        }
    }
};
```

Problems with this approach

- JANA2 discourages statefulness but also mandates it in specific places
- Only supports Run-level data. Otherwise have to query CalibrationService on every PhysicsEvent, or use barrier events
- Parallel processing mixes up the event ordering, potentially leading to calibration thrashing
- Each Service needs to have its own strategy for evicting resources after their interval of validity has expired

```c++
#include <JANA/Components/JOmniFactory.h>
#include <JANA/Calibrations/JCalibrationManager.h>
#include <datamodel/CalorimeterClusterCollection.h>

class Cluster_fac_filtered: public JOmniFactory<Cluster_fac_filtered>{
private:
    PodioInput<CalorimeterCluster> m_clusters_in {this};
    PodioOutput<CalorimeterCluster> m_clusters_out {this};
    Service<JCalibrationManager> m_calib_manager {this};
    double m_threshold = 100.0;
public:
    void Configure() {}

    void ChangeRun(int32_t run_nr) {
        m_calib_manager->GetJCalibration(run_nr)
                       ->Get("MyCAL/cluster_threshold", m_threshold);
    }

    void Execute(int32_t /*run_nr*/, uint64_t /*evt_nr*/) {
        m_clusters_out()->setSubsetCollection(true);
        for (auto cluster : *m_clusters_in()) {
            if (cluster.getEnergy() >= m_threshold) {
                m_clusters_out()->push_back(cluster);
            }
        }
    }
};
```

Barrier events

- The event source can declare an event to be a “barrier event” before it is emitted by calling event.SetSequential()
- This performs a pipeline flush: The barrier event is held back until all preceding events are finished, and no subsequent events may be emitted until the barrier event finishes
- This allows for safe arbitrary global state updates, and is used by GlueX for handling SlowControls data.

OmniFactory Resources

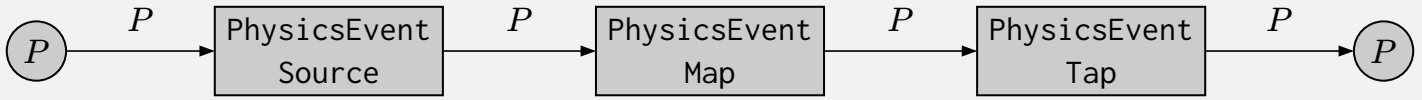
- OmniFactories can declare Resource<T> analogous to PodioInput<T>.
- This is pure syntactic sugar (for now!)
- Addresses the social problems with mandatory statefulness, but doesn’t address the underlying technical issues
- Question: What is the real difference between a Resource and a Run-level Input?

Requirements

- Directly model **intervals of validity** for all data
- Intervals can be defined at arbitrary levels in the JANA2 event hierarchy, e.g. Run, Subrun, SlowControls, Block, Timeframe, PhysicsEvent, Subevent.
- Levels are partially ordered: e.g. PhysicsEvent is a child of Timeslice and SlowControls, but Timeslice and SlowControls are incomparable. The ordering is experiment-dependent.
- The number of in-flight ~~events~~ intervals can be controlled for each level individually
- Automatic cache eviction, destruction, and/or recycling
- Motivating example: Magnetic field map changes every 15 minutes, and the framework enforces that exactly 1 (or n) magnetic field maps exists in memory at any given time

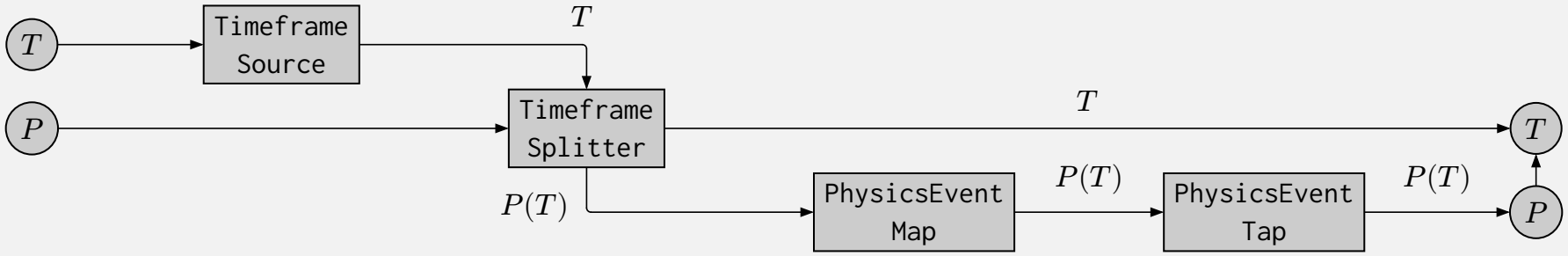
Good news! We've already built a lot of this!

Recap: EICrecon PhysicsEvent processing topology



- All events in the topology are PhysicsEvents
- Source is responsible for sequentially reading the input file
- Map is responsible for calculating all reconstruction data in parallel
- Tap is responsible for sequentially writing the output file
- PhysicsEvents are taken from and returned to a pool in order to avoid allocations and limit the number in memory

Recap: EICrecon timeframe splitting



- Timeframe source sequentially reads and emits timeframes
- Timeframe splitter produces a stream $P(T)$ of PhysicsEvents P with a reference back to their parent Timeframe T. The Timeframe's lifetime is at least as long as that of each child PhysicsEvent.
- The PhysicsEvent pool forwards each attached parent Timeframe to the Timeframe pool, respecting its child reference count.

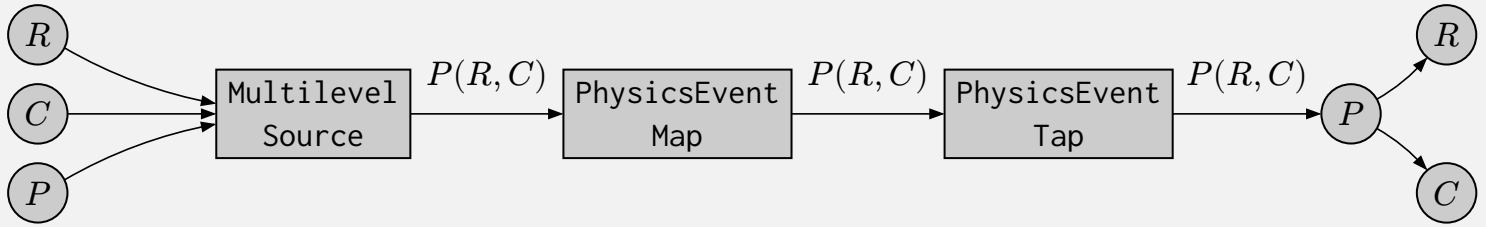
Design principle: Symmetry

- We've already added first-class support for different event levels that capture the memory semantics of different intervals of validity!
- JEvents **are** intervals of validity
- The "run number" is the same as the number of the Run-level parent JEvent.
- Any event at any level has a concise stamp describing its full lineage, e.g. "P:22(R:1,C:3,T:1)", which is sufficient to retrieve all data dependencies
- If a parent event isn't attached, the child JEvent can still declare a "virtual parent" at any level which simply tracks the parent number, analogous to how Run number was formerly handled
- Input and Resource are equivalent, except Input always retrieves data from a parent JEvent in the data stream, whereas Resource side-loads it from a Service keyed off of (level, number).
- In the future, Resource could cache its results on the parent JEvent, giving us exactly the memory semantics we want, and making factories agnostic about where the data comes from

Design principle: Closure

- Any data stream JANA2 can write, JANA2 should also be able to read
- Although JEventUnfolder produces hierarchically structured event streams which a JEventProcessor can write, a JEventSource wasn't (until recently) capable of emitting such an event stream
- Important for distributing the computation across nodes, particularly for online calibrations
- Interleaved streams are a first-class concept for streaming hierarchical data, should be understood, not avoided
- Interleaved streams are tricky to work with
  - Require preserving the stream ordering
  - Parent events need to be broadcasted, not scattered
  - Incredibly effective for serializing and for APIs…

Introducing multilevel sources



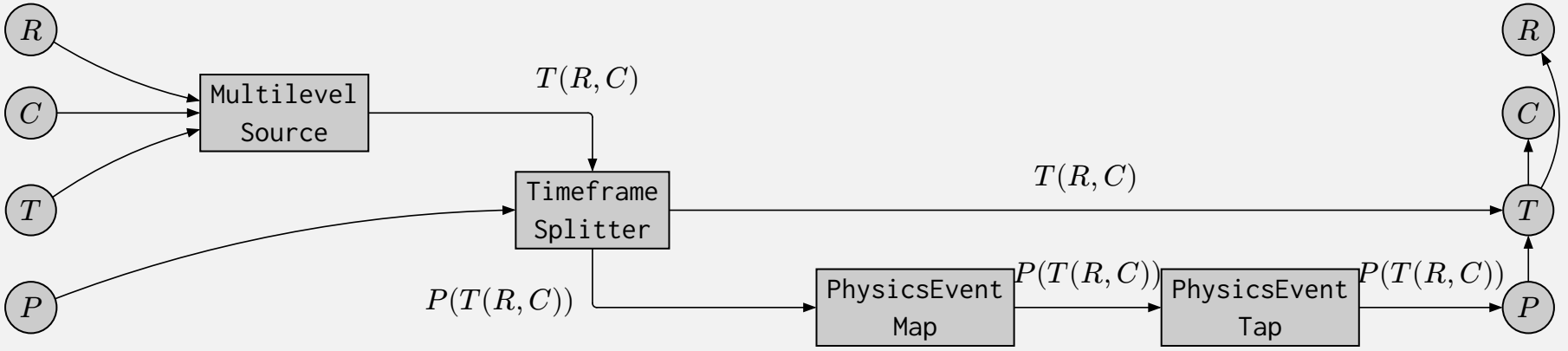
- Multilevel source emits PhysicsEvents with the most recent Run and Control events attached as parents
- The downstream components (JEventProcessors, J{Omni}Factories) don't require any modification
- R, C, P pools control number of Runs, SlowControls, and PhysicsEvents in flight.
- The PhysicsEvent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

JEventSource interface

- User can now set any number of JEventLevels
- Each {Podio}Output<T> can be assigned an event level, and will only be used if it matches the level of the provided JEvent container
- User can request a JEvent with a specific level using SetNextEventLevel()
- User can return FailureLevelChange in case the wrong level is provided

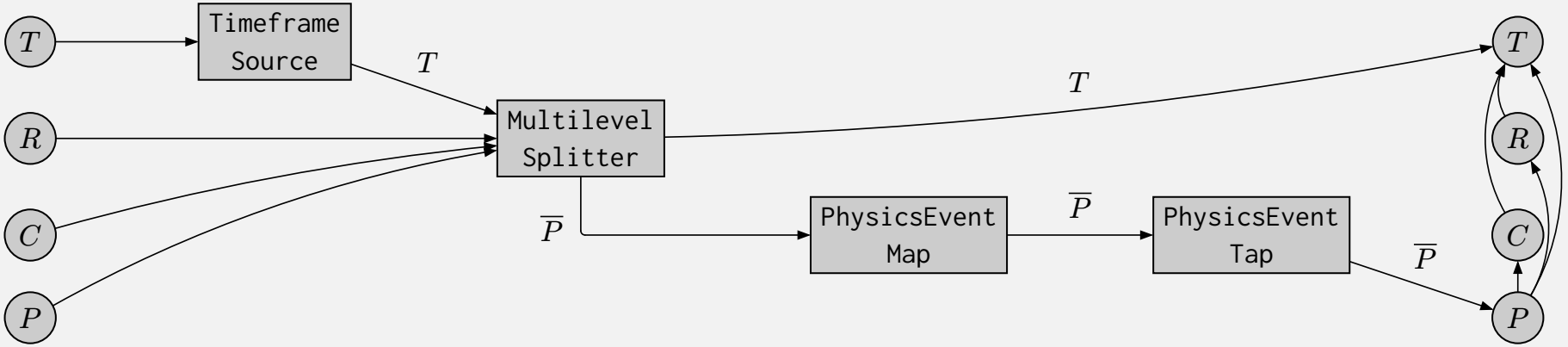
```cpp
class MyMultilevelSource : public JEventSource {

    PodioOutput<ExampleCalib> m_calibs_out {this};
    PodioOutput<ExampleControl> m_controls_out {this};
    PodioOutput<ExampleHit> m_hits_out {this};

    MyMultilevelSource() {
        SetCallbackStyle(CallbackStyle::ExpertMode);
        SetEventLevels({JEventLevel::Run, JEventLevel::SlowControls,
                        JEventLevel::PhysicsEvent});

        m_calibs_out.SetLevel(JEventLevel::Run);
        m_controls_out.SetLevel(JEventLevel::SlowControls);
        m_hits_out.SetLevel(JEventLevel::PhysicsEvent);
    }

    Result Emit(JEvent& event) override {
        switch (event.GetLevel()) {
          case JEventLevel::Run:          m_calibs_out() = ReadCalibs(); break;
          case JEventLevel::SlowControls: m_controls_out() = ReadControls(); break;
          case JEventLevel::PhysicsEvent: m_hits_out() = ReadHits(); break;
          default: break;
        }
        SetNextEventLevel(ReadNextLevel());
        return Result::Success;
    }
};
```

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

Timeframe sources with multilevel splitting



- Scenario: Begin-of-run and control events are bundled inside of timeframes
- Requires a multilevel unfolder analogous to the multilevel source
- Splitter produces either $\overline{P} := P(T, R, C)$ or $\overline{P} := P(T, R(T), C(T))$ as needed

Status of work

- [x] Extend JEventSource interface to emit multilevel events
- [x] Extend JEventPool to ingest multilevel events
- [x] Implement JMultilevelSourceArrow
- [x] Manually wire JEventSourceArrow topology
- [ ] Automatically wire JMultilevelSourceArrow topology
- [ ] Implement JMultilevelUnfoldArrow
- [ ] Automatically wire JMultilevelSourceArrow topology
- [ ] Improve topology visibility/debuggability
- [x] Implement generalized run numbers ("virtual parents")
- [ ] Extend JOmniFactory::Resource<T> to use virtual parents
- [ ] Extend JOmniFactory::Resource<T> to read/write to physical parents

Thank you!