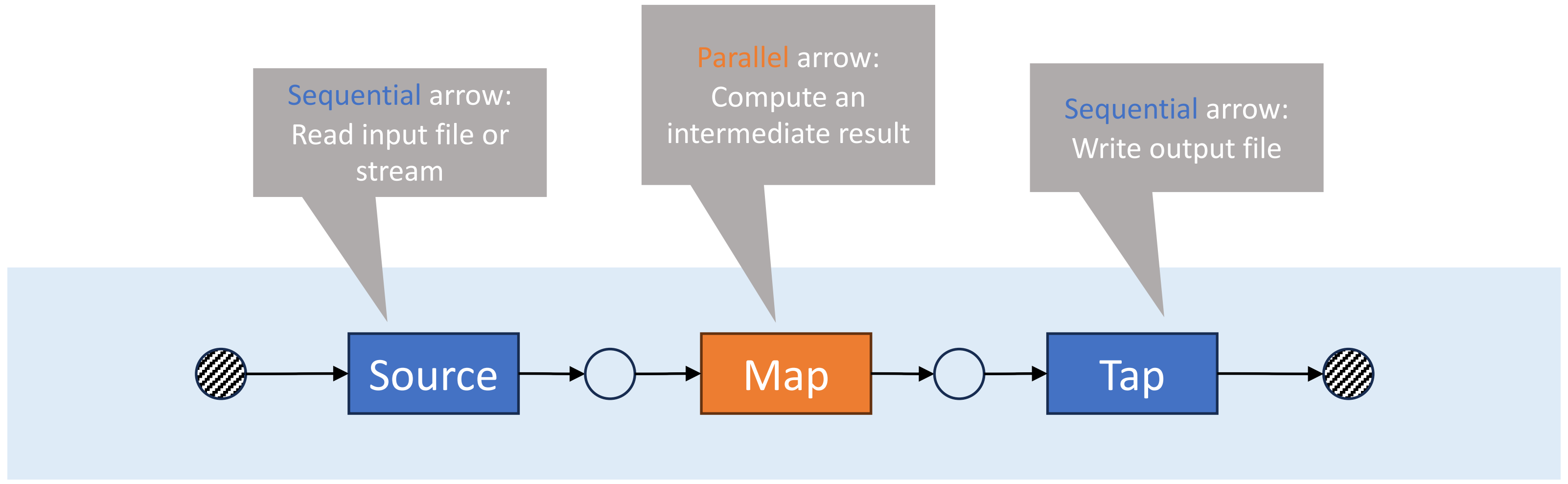
Timeslices in JANA2

Nathan Brei

Jefferson Lab

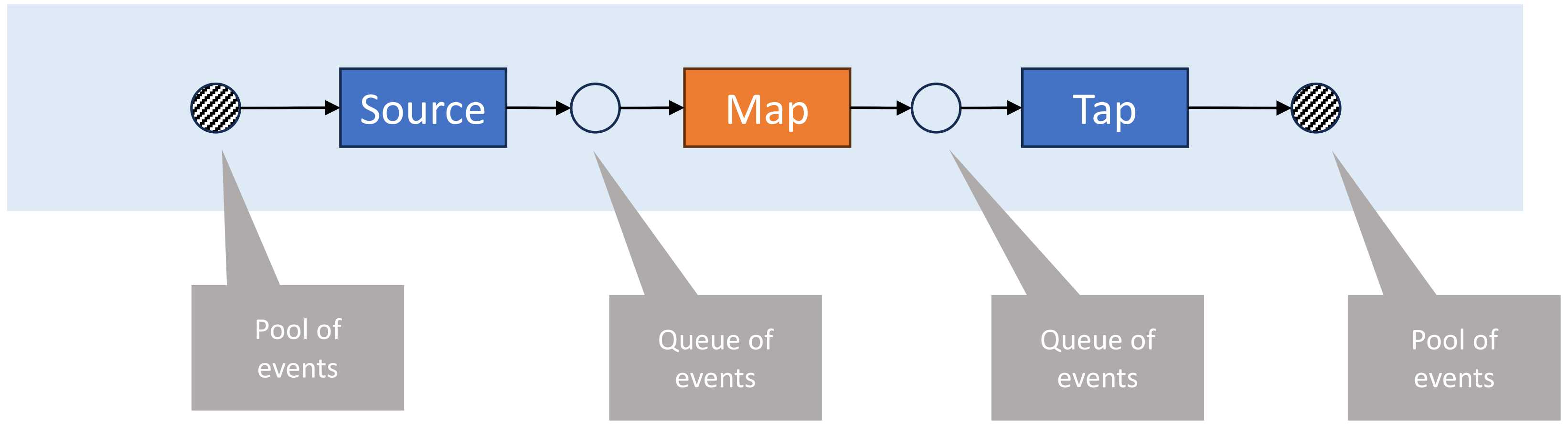
22 April 2024

How JANA2 works internally – Formalism



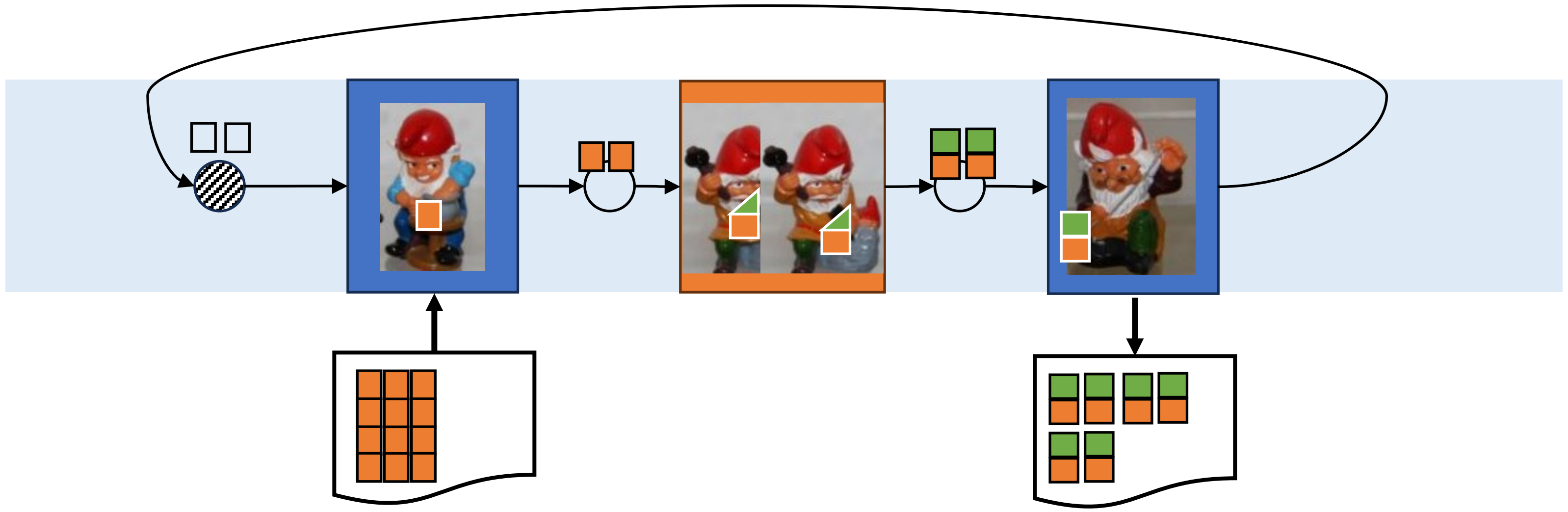
- Dataflow-parallel **processing topology** consisting of **arrows, queues,** and **pools**
- Arrows represent fixed tasks which may be sequential or parallel
- Arrows may have multiple queues and pools for their inputs and outputs
- Queues allow asynchronous processing so that no thread is directly waiting for a computation to finish

How JANA2 works internally – Formalism



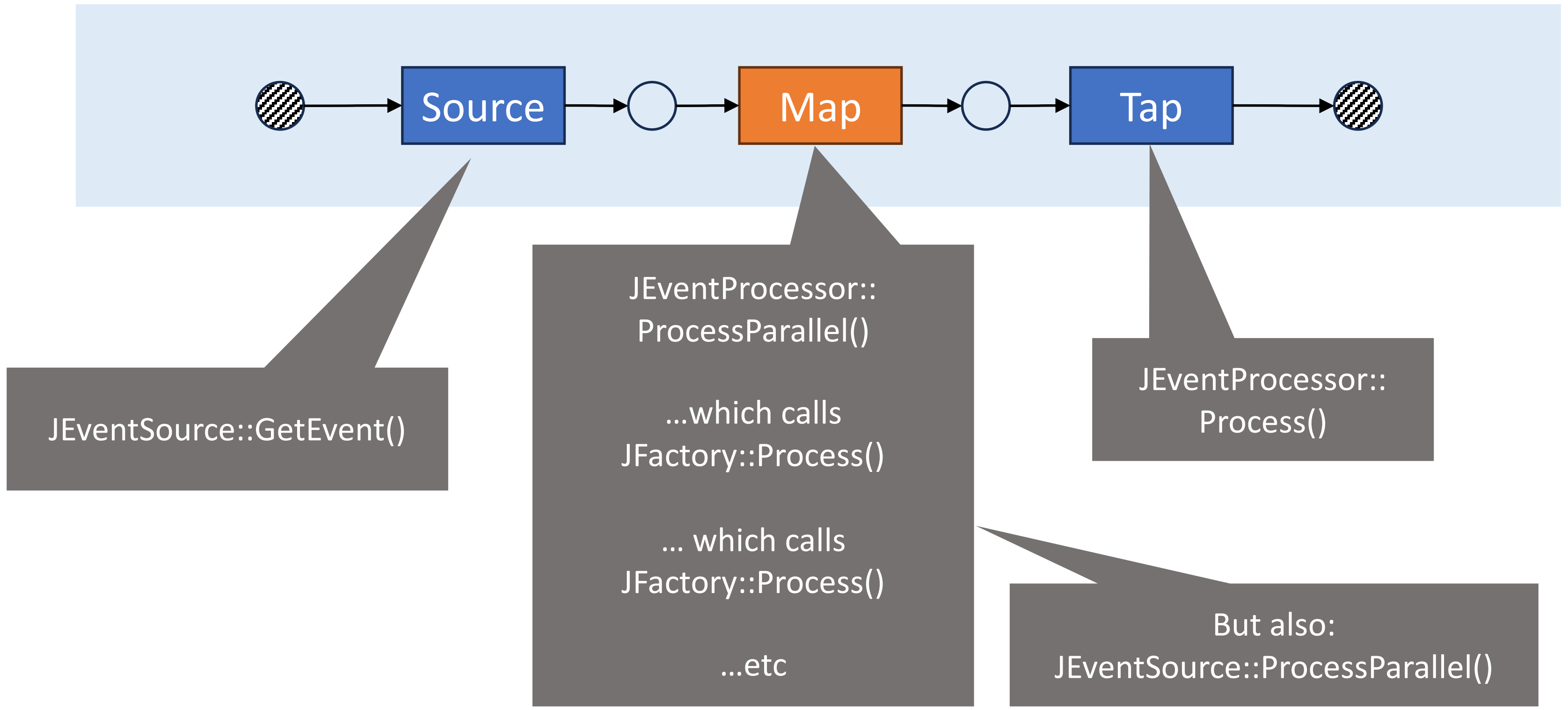
- Dataflow-parallel **processing topology** consisting of **arrows, queues,** and **pools**
- Arrows represent fixed tasks which may be sequential or parallel
- Arrows may have multiple queues and pools for their inputs and outputs
- Queues allow asynchronous processing so that no thread is directly waiting for a computation to finish

How JANA2 works internally – Cartoon



How JANA2 Components map to Arrows

- The user doesn't interact with topologies or arrows directly
- Instead, the user provides JANA with components such as JEventSources, JEventProcessors, JFactories
- Components are **decoupled** from each other. "**Only communicate through the data model**"
- JANA2 assigns the components' callbacks to arrows in the processing topology

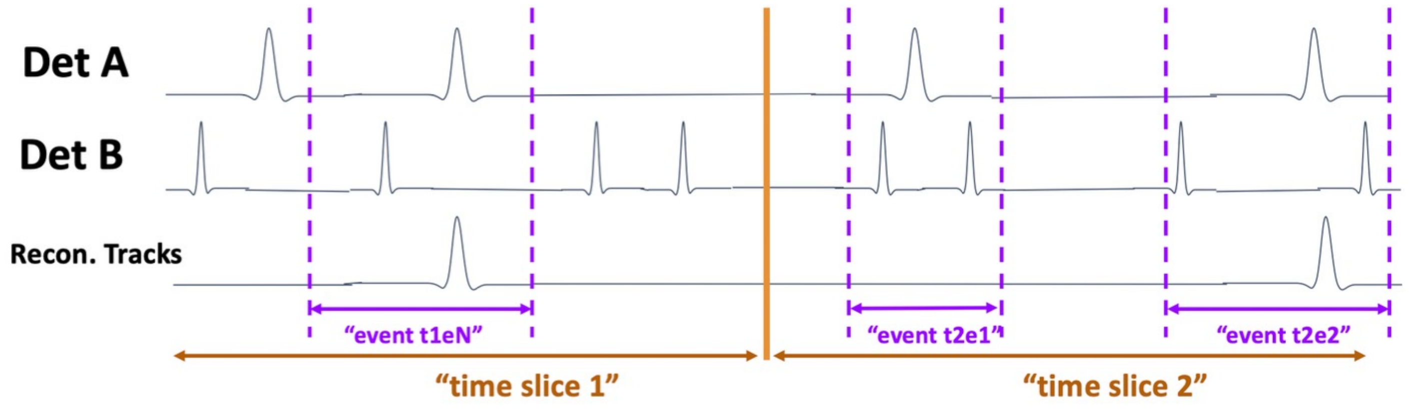


Event levels

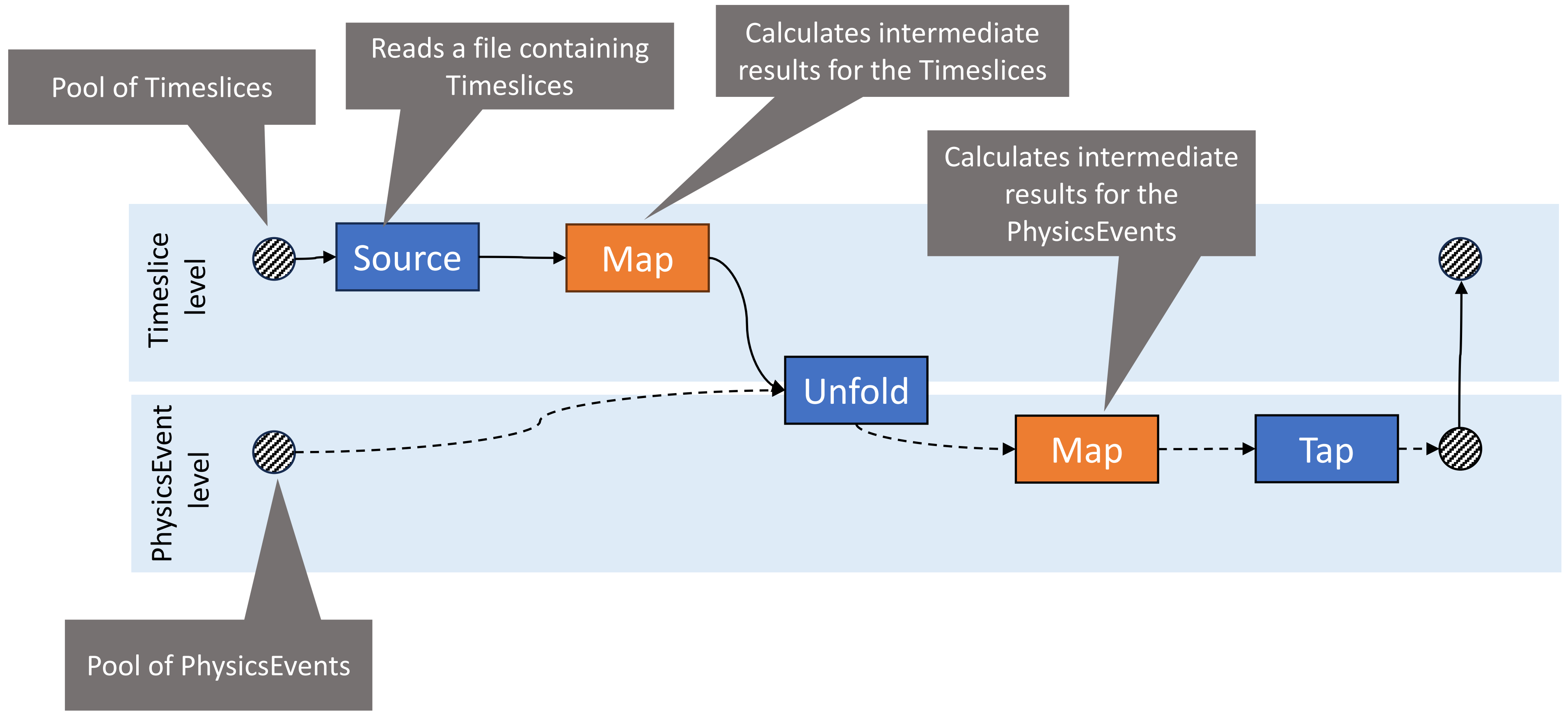
- JANA2 has a JEvent abstraction which previously meant both
  1. A container of intermediate data that is used as JANA's unit of parallelism
  2. A physics event
- Now, JEvent strictly means (1).

- Each JEvent is *tagged* (not typed!) as belonging to some JEventLevel.
- For now, JEventLevel is an enum, although user-definable event levels may be supported in the future.
- JANA2 doesn't assume that all event levels are hierarchical, e.g. that one physics event fits inside exactly one block, or even fully ordered. Instead, users establish that relationship explicitly.

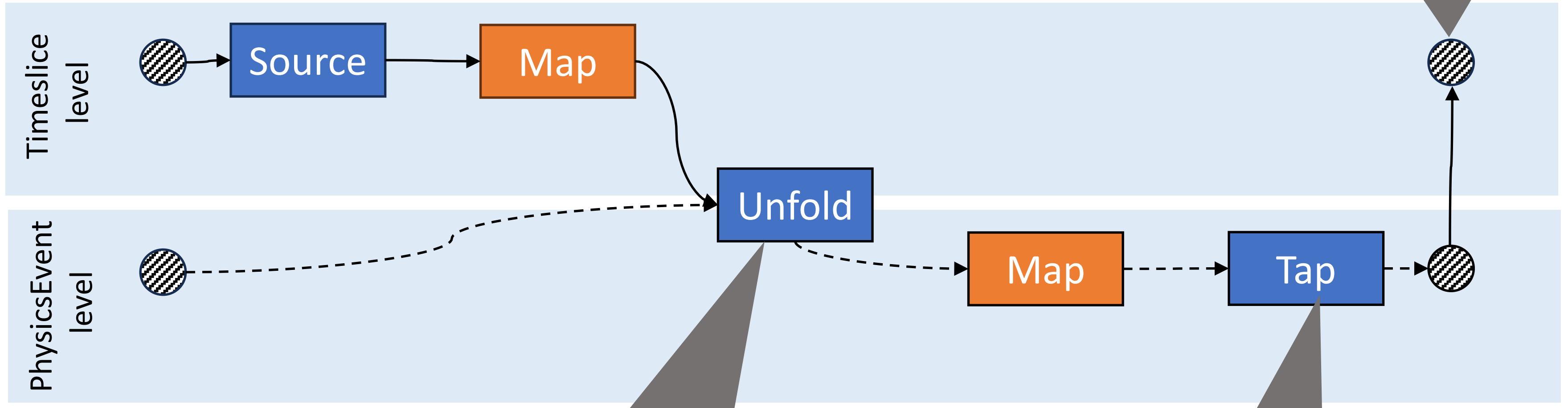
```
enum class JEventLevel {
    Run,
    Subrun,
    Timeslice,
    Block,
    SlowControls,
    PhysicsEvent,
    Subevent,
    Task,
    None
};
```



Generalizing to two event levels


Pool of Timeslices
Reads a file containing Timeslices
Calculates intermediate results for the Timeslices
Calculates intermediate results for the PhysicsEvents
Timeslice level
PhysicsEvent level
Source
Map
Unfold
Map
Tap
Pool of PhysicsEvents

Generalizing to two event levels



PhysicsEvent pool knows how to recycle Timeslices once their reference count reaches zero

Splits Timeslices into PhysicsEvents. Keeps the Timeslice around and lets the PhysicsEvent reference it.

This uses a new type of component!

Writes PhysicsEvents (and possibly Timeslices) to file

Introducing JEventUnfolder component

```
Result Unfold(
    const JEvent& parent,
    JEvent& child,
    int child_index) override;
```

```
enum class Result {
    NextChildNextParent,
    NextChildKeepParent,
    KeepChildNextParent
};
```

- JEventUnfolder looks and feels very similar to a JOmniFactory
- Users may declare Parameters, Services, Resources, Inputs, Outputs, or access everything through JApplication/JEvent
- No Generator needed as there will only be one instance active for any given level, same as JEventProcessors

- Provides an **Unfold** callback
  - Name comes from functional programming and stream processing
  - Unfold handles both “splitting” and “merging” streams
  - Returns a Result code indicating whether the parent and child belong together
  - We never need to have all PhysicsEvents corresponding to one Timeslice in memory at once

- Inputs come from the parent event (e.g. Timeslice)
- Outputs are inserted into the child event (e.g. PhysicsEvent)
- The child event keeps a pointer to the parent event around, so that any factory can access Timeslice-level data

What does this mean for our Factories?

- OmniFactories look almost exactly the same as before
- OmniFactories each belong to a particular event level. All of their outputs belong to that level.
- OmniFactory::Input helper now takes event level as an optional parameter
- Event level information can be applied **entirely** at the JOmniFactoryGenerator level
- The same algorithm and factory can be wired and reconfigured for different event levels

```
struct MyProtoclusterFactory
  : public JOmniFactory<MyProtoclusterFactory> {

PodioInput<ExampleHit> hits_in {this};
PodioOutput<ExampleCluster> clusters_out {this};

void Configure() {
}

void ChangeRun(int32_t run_nr) {
}

void Execute(int32_t run_nr, uint64_t evt_nr) {
    ...
}
```

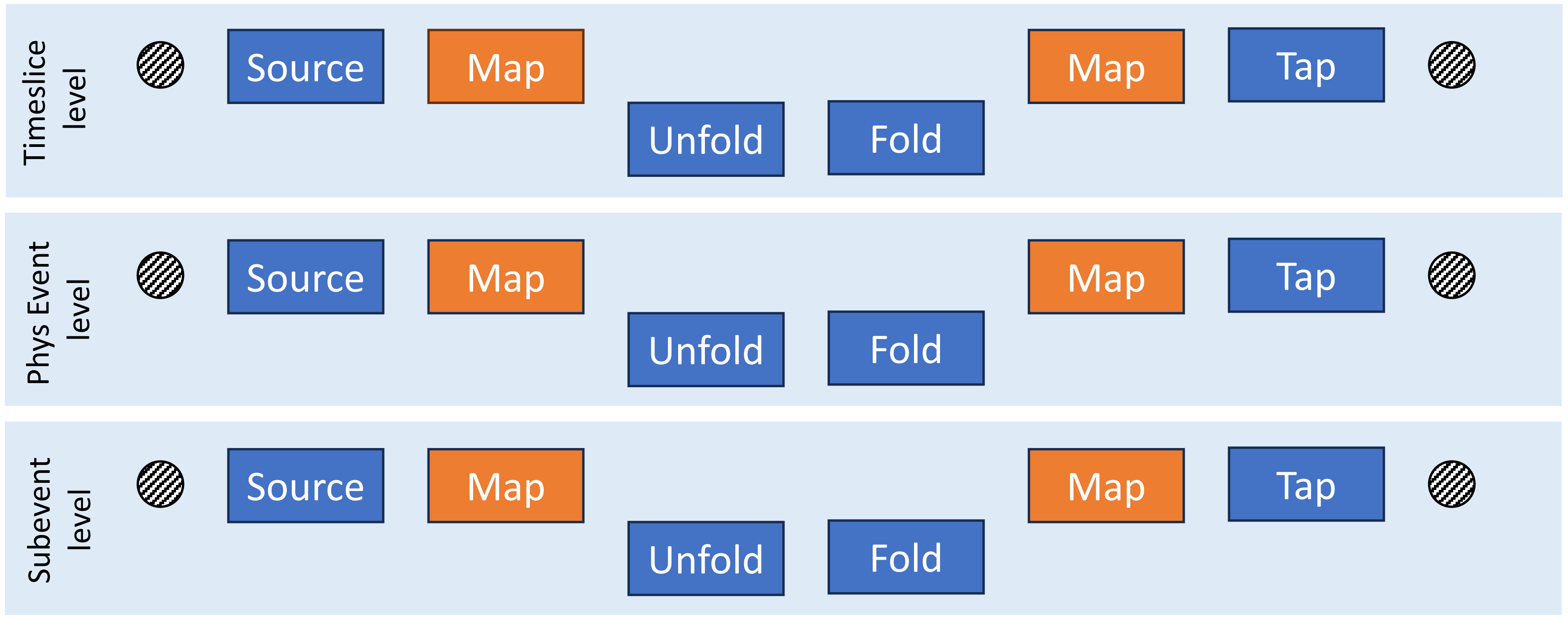
```
// Factory that produces timeslice-level protoclusters
// from timeslice-level hits
app->Add(new JOmniFactoryGeneratorT<MyProtoclusterFactory>(
    { .tag = "timeslice_protoclusterizer",
      .level = JEventLevel::Timeslice,
      .input_names = {"hits"},
      .output_names = {"ts_protoclusters"}
    }));

// Factory that produces event-level protoclusters
// from event-level hits
app->Add(new JOmniFactoryGeneratorT<MyProtoclusterFactory>(
    { .tag = "event_protoclusterizer",
      .input_names = {"hits"},
      .output_names = {"evt_protoclusters"}
    }));
```

What does this mean for JEventSources?

```cpp
#include <JANA/JEventSourceGenerator.h>
#include "MyFileReader.h"

class MyFileReaderGenerator : public JEventSourceGenerator {

    double CheckOpenable(std::string resource_name) override {
        if (resource_name.find(".root") != std::string::npos) {
            return 0.01;
        }
        return 0;
    }

    JEventSource* MakeJEventSource(std::string resource_name) override {

        auto source = new MyFileReader;

        if (resource_name.find("timeslices") != std::string::npos) {
            source->SetLevel(JEventLevel::Timeslice);
        }
        else {
            source->SetLevel(JEventLevel::PhysicsEvent);
        }
        return source;
    }
};
```

- JANA2 can figure out that the input file contains timeslices from inside the JEventSourceGenerator
- This means that this critical information is already known before the time of topology construction
- The topology builder is able to decide what topology to build based off what components were provided.
- The same PODIO event source class can be reused for files containing timeslices vs physics events with minimal modification

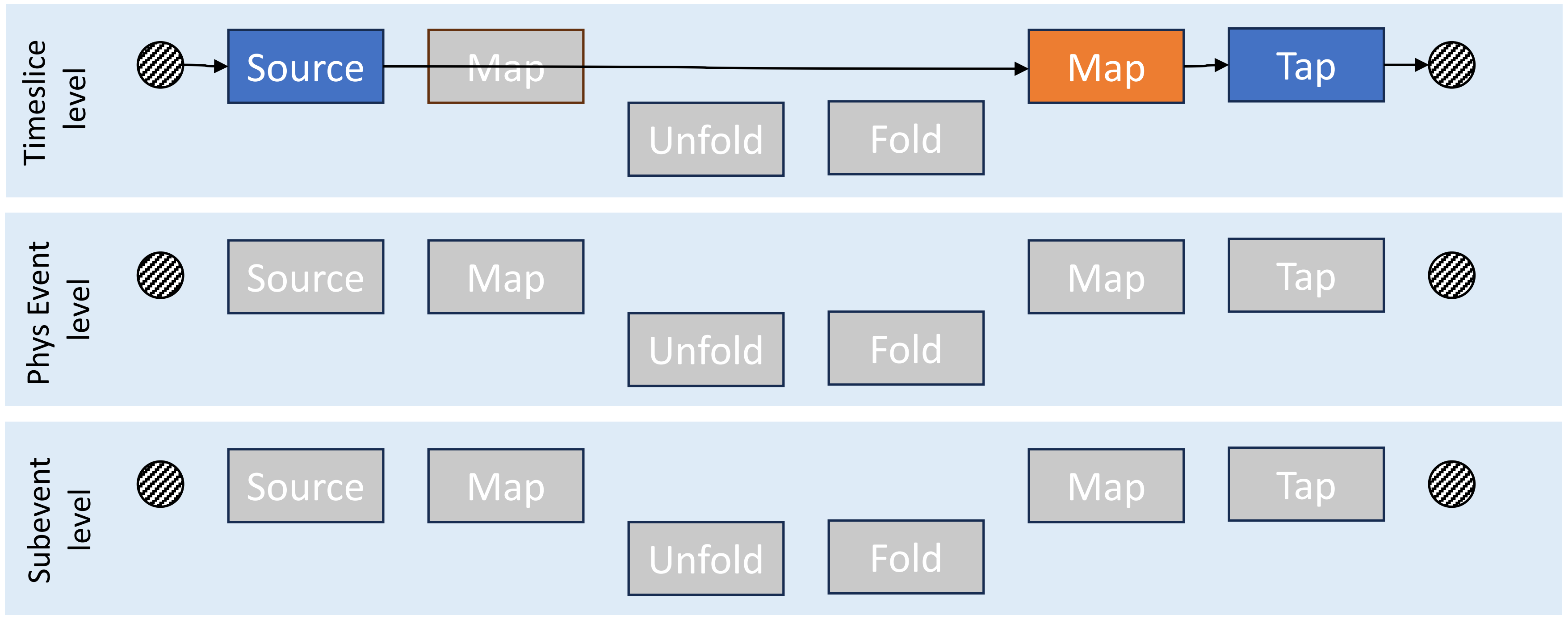
Generalizing further



- Source calls
  - JEventSource::GetEvent()
- Map calls
  - JOmniFactory::Process()
  - JEventProcessor::ProcessParallel()
  - JEventSource:: ProcessParallel()
  - JEventUnfolder:: ProcessParallel()
  - JEventFolder:: ProcessParallel()
- Tap calls
  - JEventProcessor::Process()
- Unfold calls
  - JEventUnfolder::Unfold()
- Fold calls
  - JEventFolder::Fold()

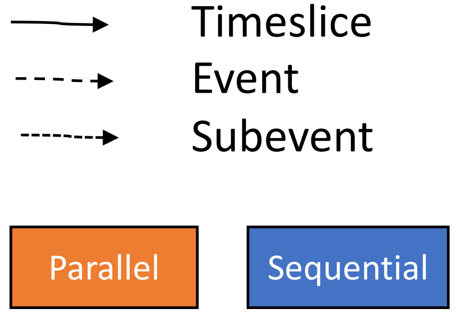
- The arrows in the further generalized topology (abstractly) form a grid:
  `{Source, Map1, Unfold, Fold, Map2, Tap} x {Timeslice, PhysicsEvent, Subevent,…}`
- Depending on which components the user provides, JANA2 can activate and wire the arrows automatically
- This wiring could also be specified manually

Basic topology

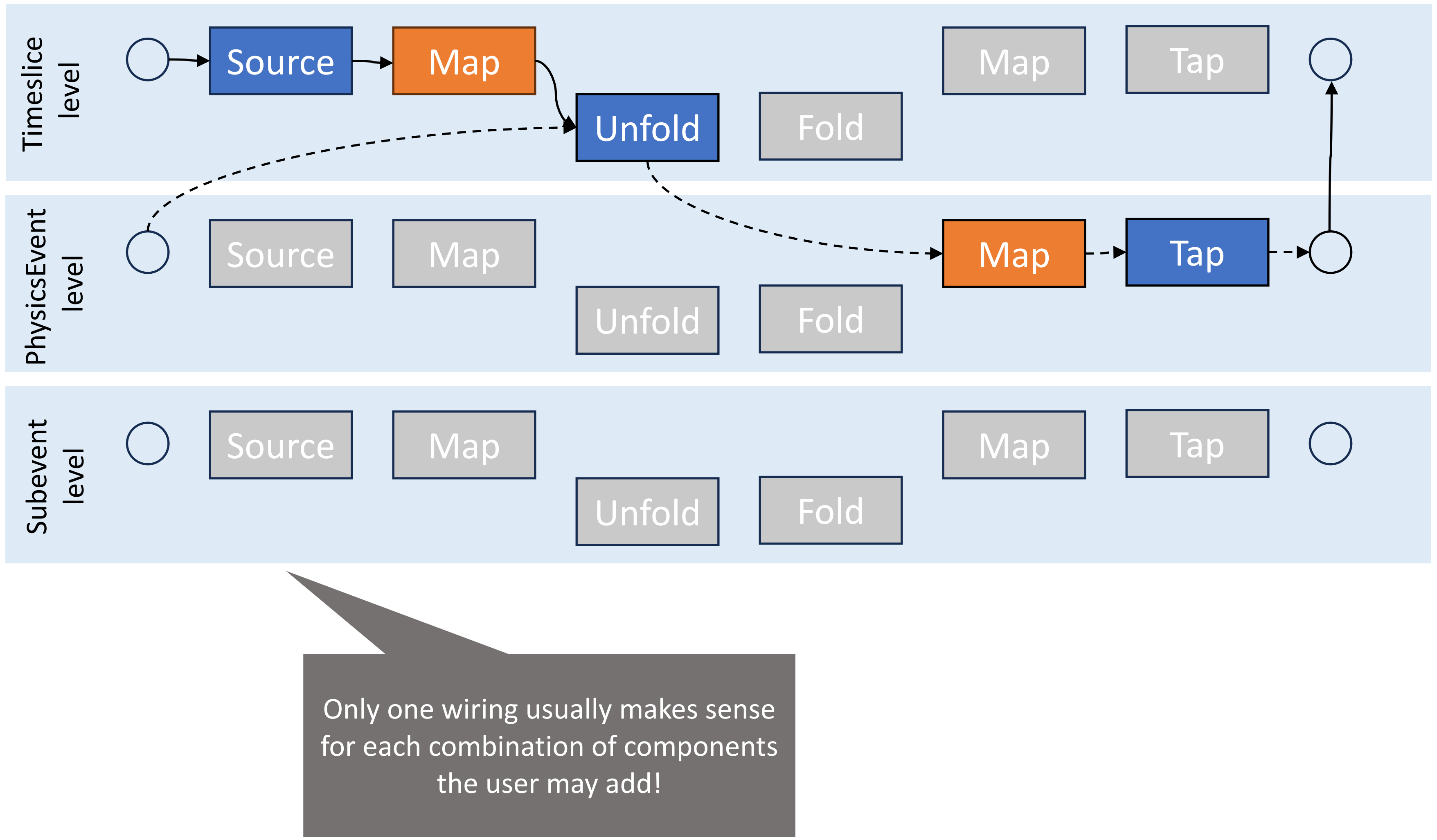


User provides:

- JEventSource [Timeslice]
- JEventProcessor [Timeslice]
- JFactory [Timeslice]

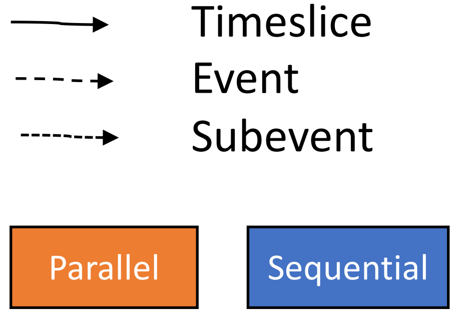


Timeslice splitting topology

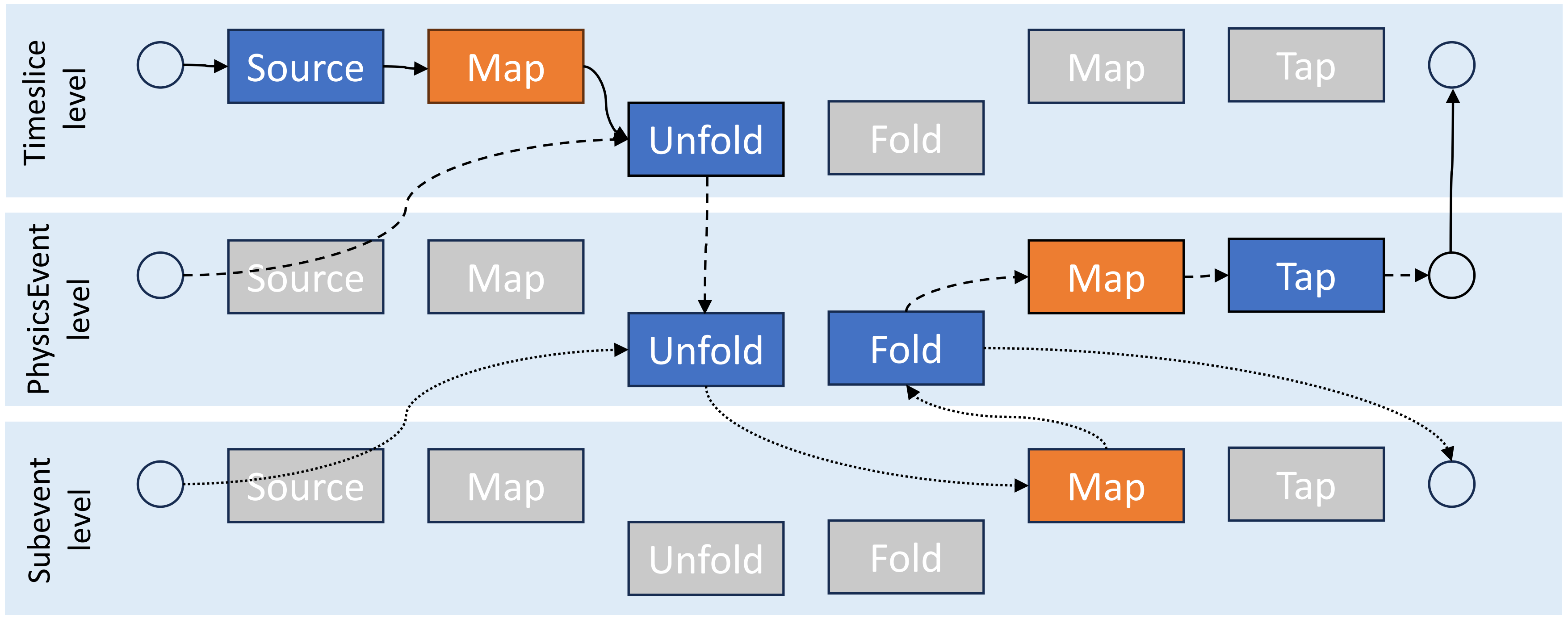


User provides:

- JEventSource [T]
- JFactory [T]
- JEventUnfolder [T -> P]
- JEventProcessor [P]
- JFactory [P]



Timeslices + subevents topology



User provides:

- JEventSource [T]
- JEventProcessor [P]

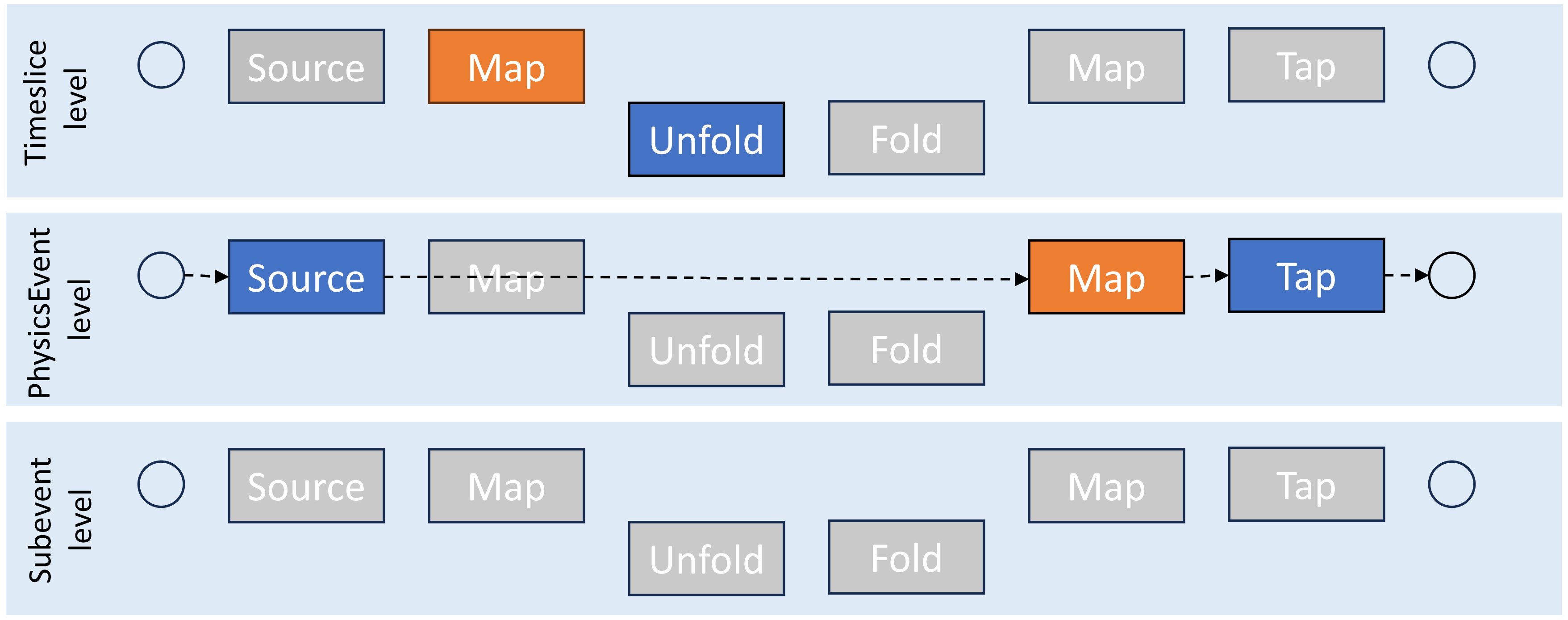
- JEventUnfolder [T -> P]
- JEventUnfolder [P -> S]
- JEventFolder[S -> P]

- JFactory [T]
- JFactory [P]
- JFactory [S]

⟶	Timeslice
- - -▸	Event
·····▸	Subevent

🟧 Parallel	🟦 Sequential

What happen if the user provides “extra” components?



User provides:

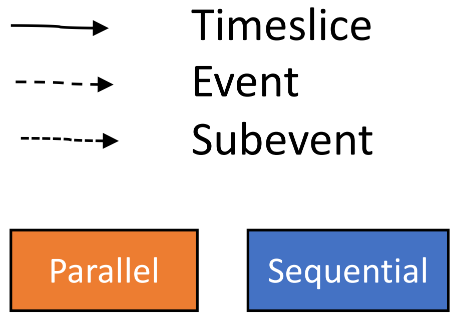
- JEventSource [P]
- JEventProcessor [P]
- JEventUnfolder [T -> P]

IGNORED!

- JFactory [T]

IGNORED!

- JFactory [P]



What does this mean for EICrecon?

- We can define our factories and algorithms once
- We can add generators that wire them differently for the timeslice input files and for physics input files
- These wirings can live side-by-side without interfering with each other
- We can define our PODIO event source and processor once
- We can add a generator that configures the source's event level
- The topology builder choose which topology to build based off of which components (most notably, sources) are present
- **No additional configuration necessary! Eases the transition from events to timeslices**

Memory management – Concept

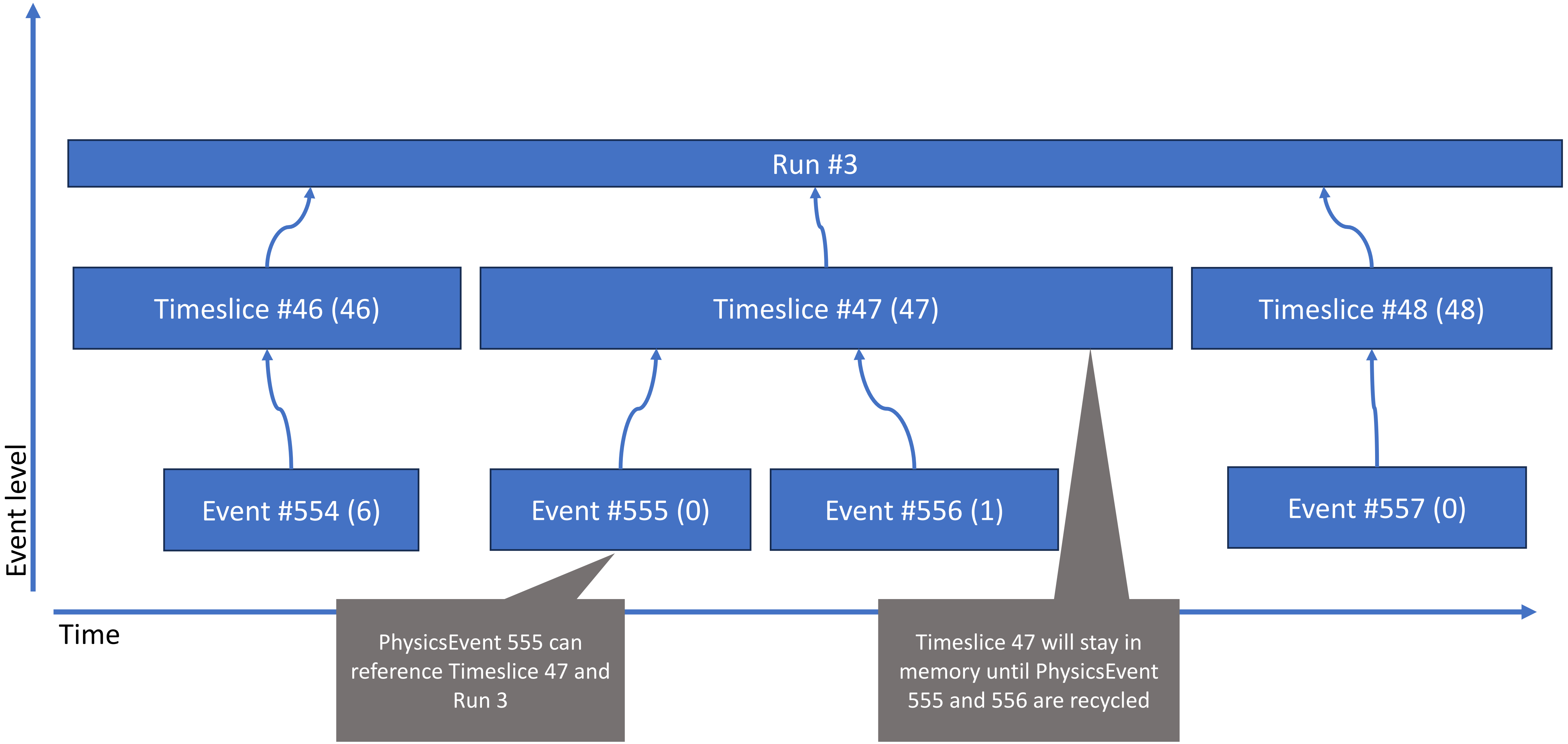
As of right now:

- Parents have shared-ptr-like semantics (except they are recycled to a pool)
- Parents always outlive their children
- Events can have multiple parents
- Parents are uniquely identified by their event level: “Diamond inheritance” not permitted
- To get data from a parent, you have to ask for the parent explicitly (no searching or “importing into the global namespace”)

Future improvements:

- Event sources will eventually be able to emit events that already have parents
- Data in adjacent timeslices will be accessible via a ‘sibling’ reference, analogous to parents except weak-ptr-like

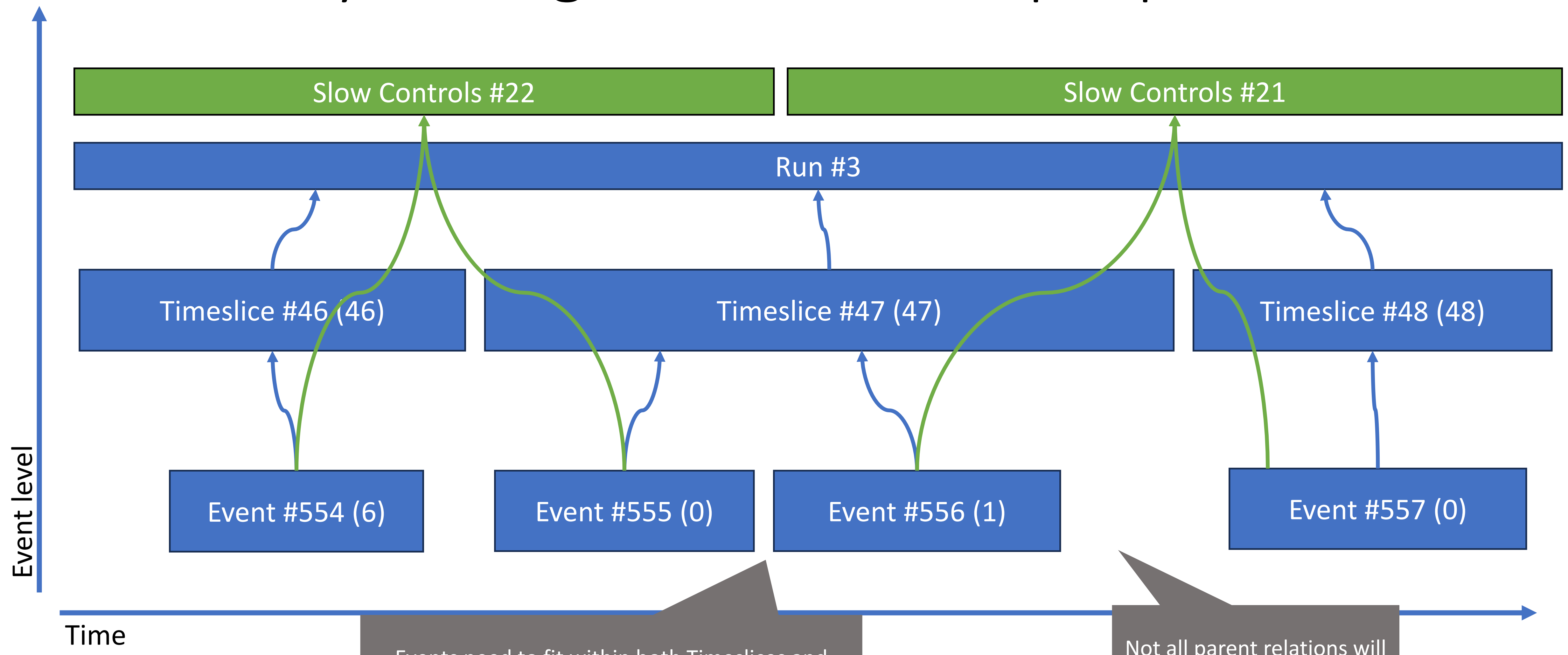
Memory management – Parent relation



PhysicsEvent 555 can reference Timeslice 47 and Run 3

Timeslice 47 will stay in memory until PhysicsEvent 555 and 556 are recycled

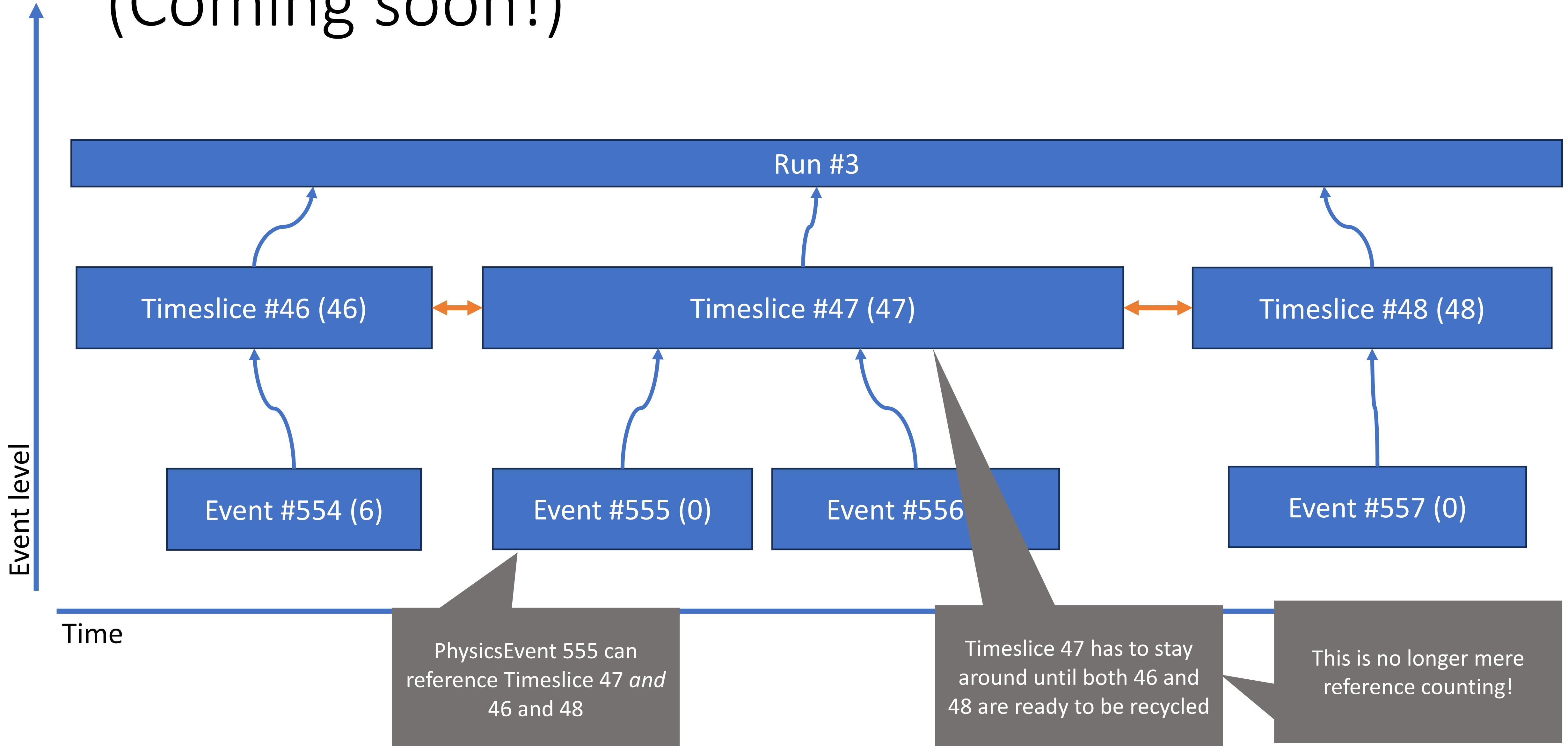
Memory management – Multiple parents



Events need to fit within both Timeslices and SlowControls, but SlowControls and Timeslices can overlap!

Not all parent relations will necessarily come from the Unfolder!

Memory management – Sibling relations (Coming soon!)



PhysicsEvent 555 can reference Timeslice 47 *and* 46 and 48

Timeslice 47 has to stay around until both 46 and 48 are ready to be recycled

This is no longer mere reference counting!

Event key

- Generalizes the concept of event and run number to streaming scenarios
- Will eventually replace the awkward arguments to JOmniFactory::Execute

- Event number: For each level **inside** our unfold/fold hierarchy, we have:
  - Absolute number: Starts at 0, increments by 1
  - Relative number: Starts at 0 for each parent, increments by 1
  - User key: Could be anything, bunch crossing number in practice

- Run number: Separate numbers for each parent level **outside** of the unfold/fold hierarchy
  - Goal: Take advantage of the symmetry between “side-loading data from a database” and “retrieving data from events that live at a different level but were intermingled in the event stream”, e.g. BOR, slow controls
  - Might all end up being intervals of bunch crossing numbers in practice
  - Challenge: Getting JEventSource to emit events that already have parents

Summary

- JEvents and components can all be tagged with an **event level** := {…, Timeslice, PhysicsEvent, Subevent, …}
- We introduce a `JEventUnfolder` which lets us **split** a timeslice into events, and also **merge** two independent streams
- Components at any level (e.g. PhysicsEvent) are able to safely and easily reference the data at higher levels (e.g. Timeslice)
- We extend the **OmniFactory** interface patterns to JEventUnfolder
- JANA2 is now able to automatically build a complex topology from different components at different event levels.
- EICrecon will be able to tell just from the input file what topology needs to be built and how to build it => Smoother transition

Next steps

- A working prototype is already in master
  - src/examples/TimesliceExample
  - https://github.com/JeffersonLab/JANA2/

- Create timeslice data file
- Implement logic for splitting timeslice into physics events
- Ironing out small details
  - Recycling parents via an EventFolder vs directly to event pool
  - Improving the JEventKey to better generalize event and run numbers