Processing multiple events, lumis, runs

...

Benedikt Hegner
(CERN EP-SFT)

3.8.2016

A little warning

It may be that my proposal solves a problem that de-facto does not exist for ATLAS and LHCb

Run/lumi-dependent data

One of the dreams is to have “stateless” (well, re-entrant) algorithms. This is destroyed by *frame*-dependent (*) data like:

1. Statistics counters (#events seen, efficiencies, ...)
2. Monitoring histograms
3. ...

⇒ we should factor them out and **separate algorithms and data**
⇒ frame-dependent data are hidden behind special handles and live outside the algorithm

(*) *frame* is a more generic term for run, lumi, ...

Using frame slots

Generalize and replace the *eventSlot* with a **frameSlot:**

- ➔ the number of frameSlots defines how many lumis/runs/conditions can be active at a time
- ➔ access to run/lumi-specific data is forwarded to the respective slot
- ➔ both event and frame slots are managed by the *EventLoopMgr*

Could re-use interface of data handles:

1. *Frame*Specific<T> for frame specific data
2. *Frame*Read<T>, *Frame*Write<T> for data stored in run/lumi blocks

From user perspective:

```
// event specific
ReadDataHandle<ElectronCollection> m_electrons{...};
// run data
RunSpecific<Counter> m_events;
```

Notification about begin/end of frames

The incidents as implemented now are not context-aware

⇒ need to add a context to such messages

Once a new frame enters the event loop, the EventLoopManager picks a frameSlot, and notifies clients via ::begin*Frame*(frameSlot)

Once all events of a frame left the event loop, the EventLoopManager notifies all clients via ::end*Frame*(slot) and is then free to re-use the slot.

It is not entirely trivial, as a certain order may be needed, e.g. run data has to written into the run record ***before*** *the data are written to disk.*

⇒ no new requirement w.r.t. current framework though

A full example

Reading new run and a few events from input file triggers the following behaviour:

The EventLoopMgr

1. Opens a new frame slot for that run
2. Triggers `beginRun(frameSlot)` for all relevant components
    a. Algorithms initialize run-specific data
1. Opens eventSlots for all events and pushes them to the scheduler
2. Upon completion of all events in the run trigger `endRun(frameSlot)`
    a. Algorithms flush run-specific data
3. Frees the frame slot

Comparison to CMS' solution

This proposal: separate algorithms and data

⇒ algorithms are frame-independent

⇒ data are multiplied for multiple frames

⇒ access to data is dispatched at runtime

⇒ #concurrent frames limited by #slots

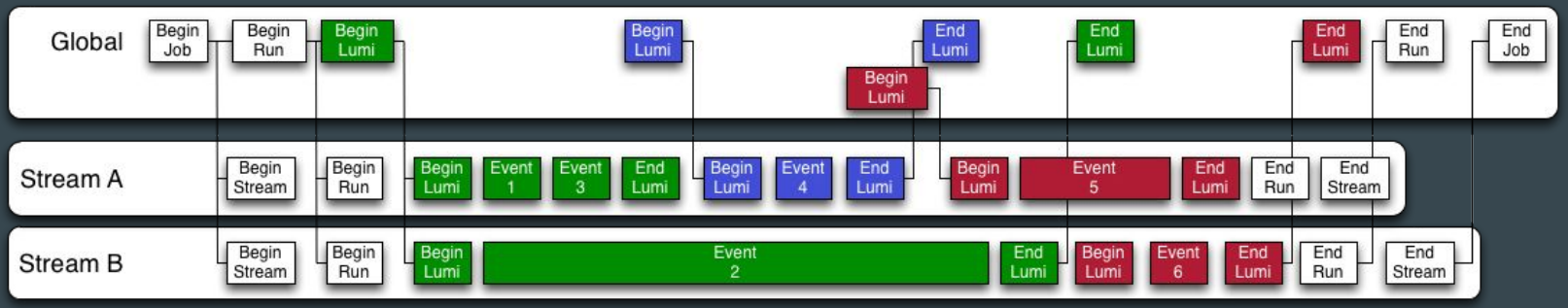
CMS: split application into "streams" that have a guaranteed run ordering

⇒ algorithms with run-specific states are tied to a given stream

(required deriving from different classes to pick proper behaviour)

⇒ algorithms and states therein are multiplied for multiple streams

⇒ #concurrent runs limited by #streams



Status and Summary

Proposed a solution for handling runs and lumi-sections in general

1. Even better separation of algorithms and data
2. Context-aware pointers

Made a (dirty!) proof-of-principle in a branch of mine: https://gitlab.cern.ch/hegner/Gaudi/tree/frame