Non-Event Data in CMS and Concurrency

David Dagenhart

21 October 2020

In partnership with:

Context

CMS uses a multi-threaded framework

- Used in production since 2016
- Built using Intel's Thread Building Block (TBB) task library
- Many people have contributed significantly to this effort
- Chris Jones has been the prime mover

Initially only supported

- Concurrent processing of events and
- Concurrent processing of modules within an event

Input File Concurrency

CMS processes one primary input file at a time

- The primary input file is closed before the next is opened
- There can be other input files open for mixing or secondary input
- Most of what occurs when files are opened and closed is serial and not concurrent
- Generally the module reading from the primary input file is only doing one thing at a time
- There is no plan to change this.

Non Event Metadata Supporting Concurrency in 2016

1. Configuration data saved from all processes in the history of a file.
2. Tables supporting links between data products (for example a link from a track to a particular hit)
3. Tables supporting links when collections are copied with entries removed.
4. File version and a unique file identifier
5. Tables storing identifiers of data which was used when some other data was created
6. Indices into a file that point to Events, Runs, and Lumis and are used to navigate when iterating or processing a single Event.
7. A table documenting the processing history of a file
8. A registry of each data product branch stored in Events/Runs/Lumi TTrees.

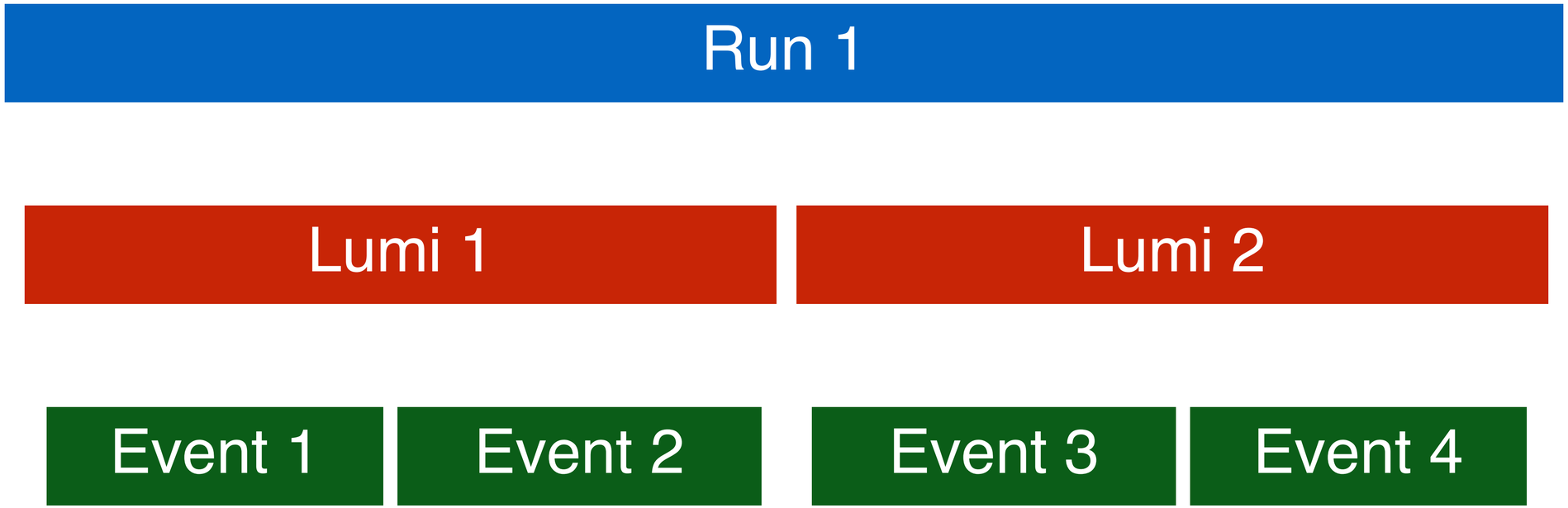
Non Event Metadata Supporting Concurrency in 2016

The items listed on the previous slide were manually upgraded to support concurrency in the years prior to 2016. Several strategies were used:

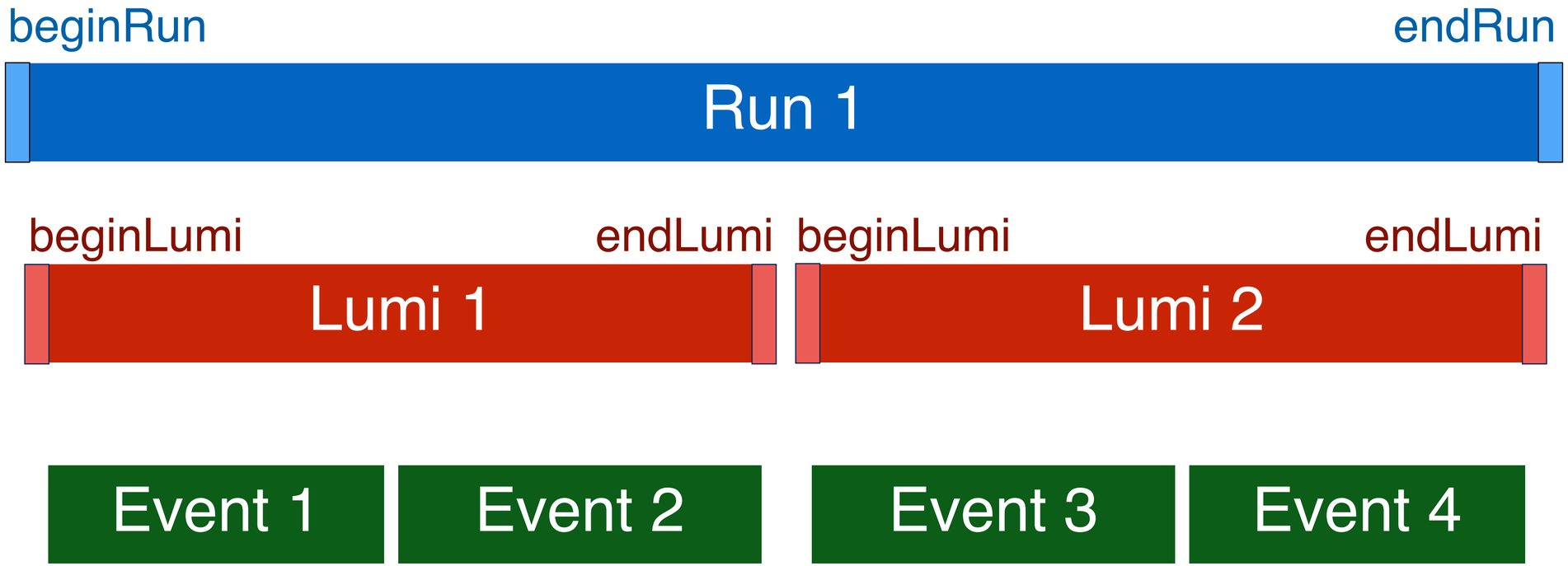
1. As often as possible we restricted the data to only change at initialization and during input/output file opening and closing transitions. The rest of the time only constant operations are allowed.
2. Where necessary concurrent containers are used. Functions that need to run concurrently often use atomics to maintain thread safety.
3. In some cases, there is a copy of the data for each Event/Run/Lumi that is allowed to be processed concurrently.

In short, the code supporting this metadata was redesigned to support concurrent operations. It's custom and different in each case what has been done. We try hard to avoid mutex locks.

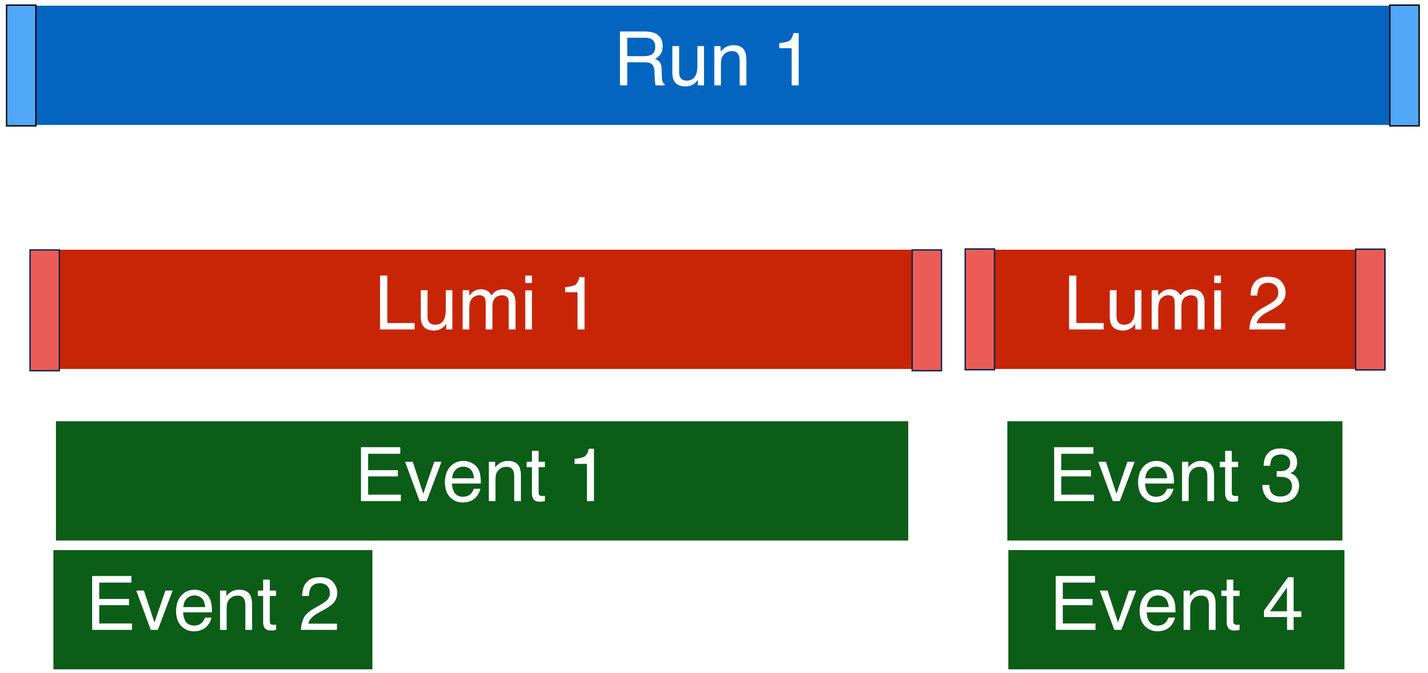
CMS Data Hierarchy



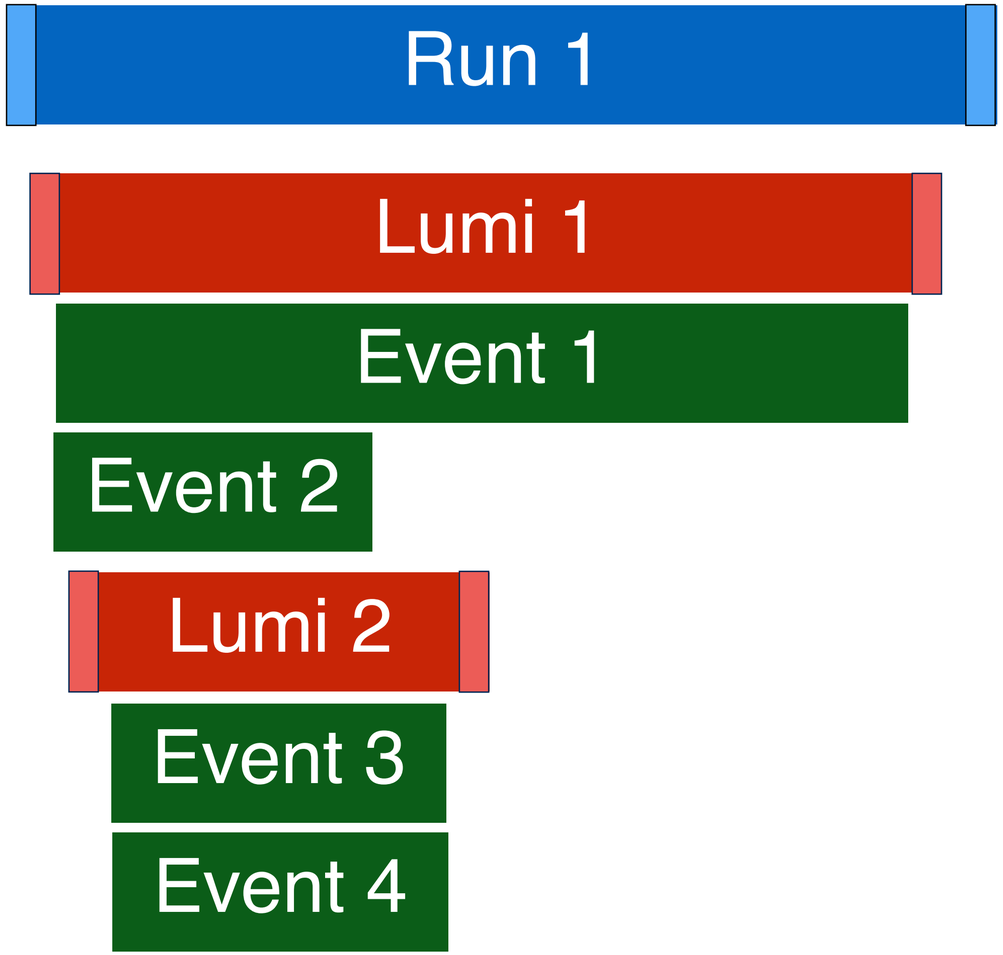
CMS Data Processing Transitions



Original Concurrent Transitions



Fully Concurrent Transitions



Constraining Memory

CMS' driving force for multi-threading is to reduce memory usage

- Allows average memory per core to be decreased

Configuration used to set limits

- Independently control number of allowed concurrent events and lumis

Shared Resources and Task Queues

Most work in the framework is done via TBB tasks

Tasks needing the same resource are placed in a queue

- Each unique resource gets its own queue
  - E.g. writing to a particular TFile
  - E.g. processing Lumis

When a resource is available, the task queue starts a waiting task

- E.g. when a task using a resources finishes, the queue starts the next task

Lumi Limited Task Queue

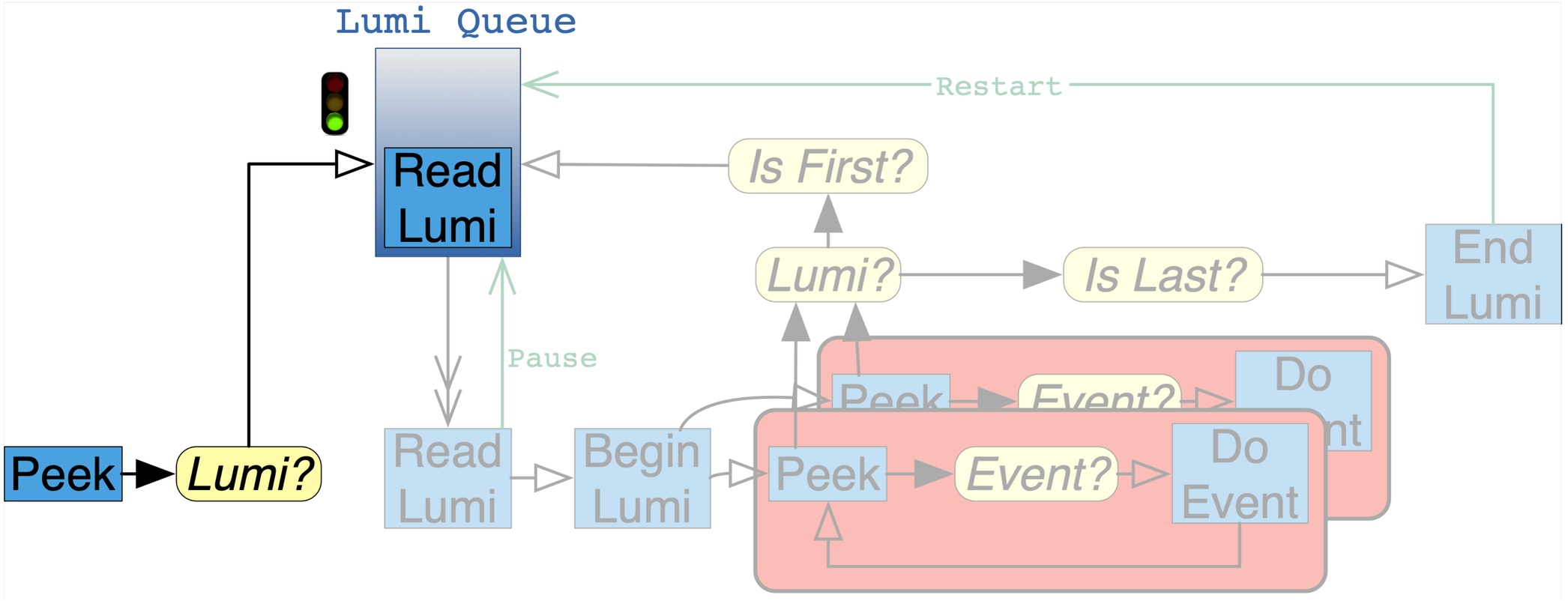
Limited Task Queue

- Has multiple independent ***lanes*** where each lane runs its own task
- All lanes pull tasks from the same waiting task list
- Each lane can be paused/restarted independently
- If all lanes are paused, no new tasks will be started from the queue

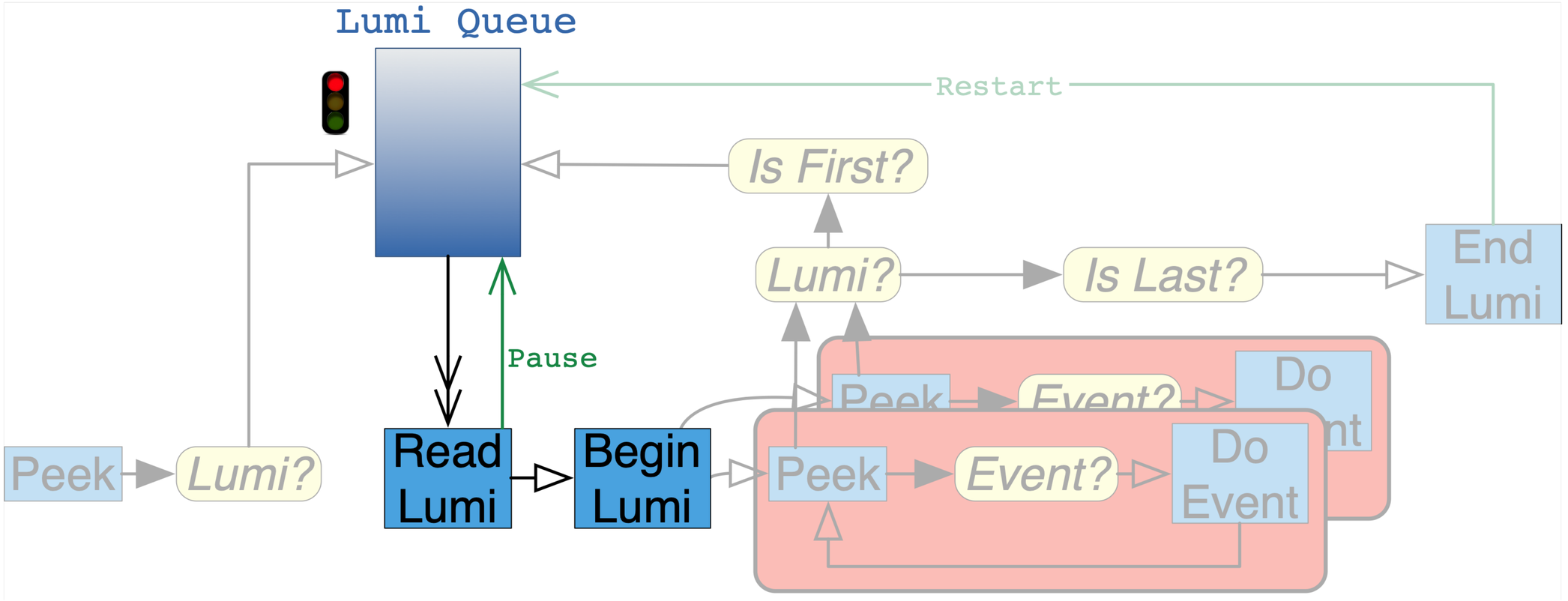
Number of concurrent Lumis controlled via a limited queue

- How many concurrent Lumis is set in the configuration to constrain memory use

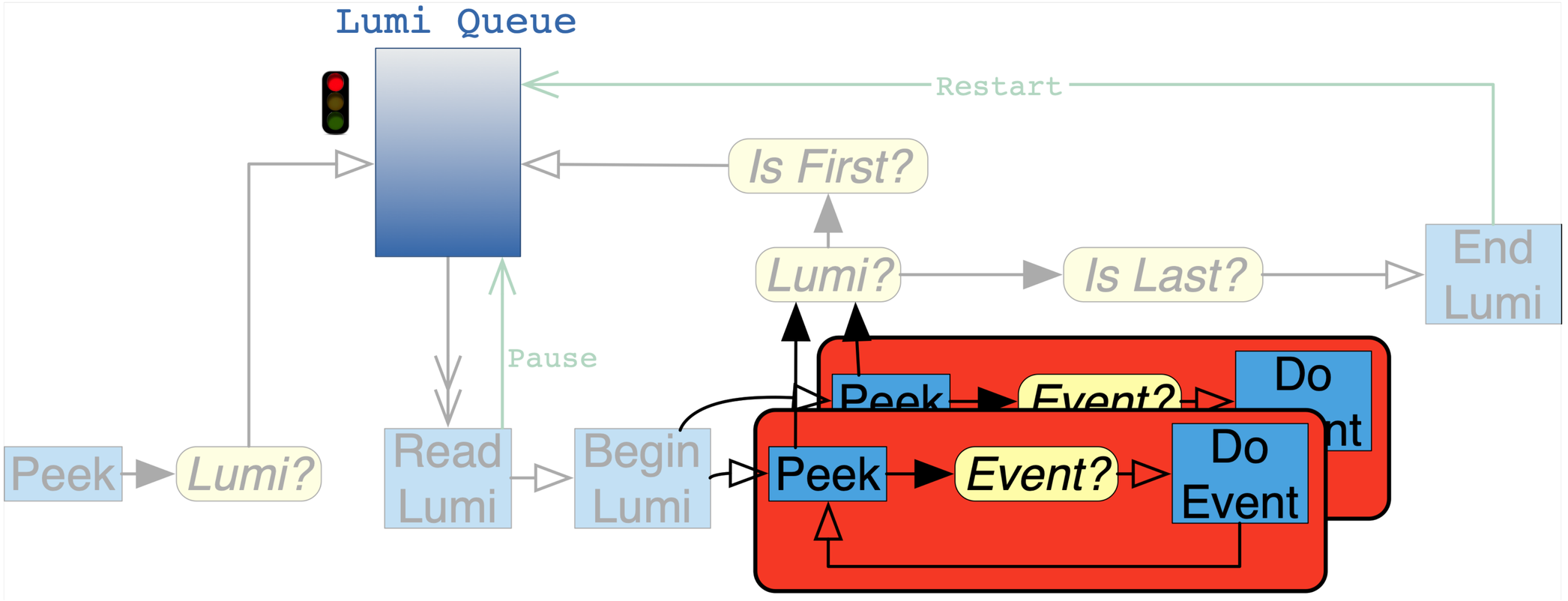
Lumi Processing with Queue



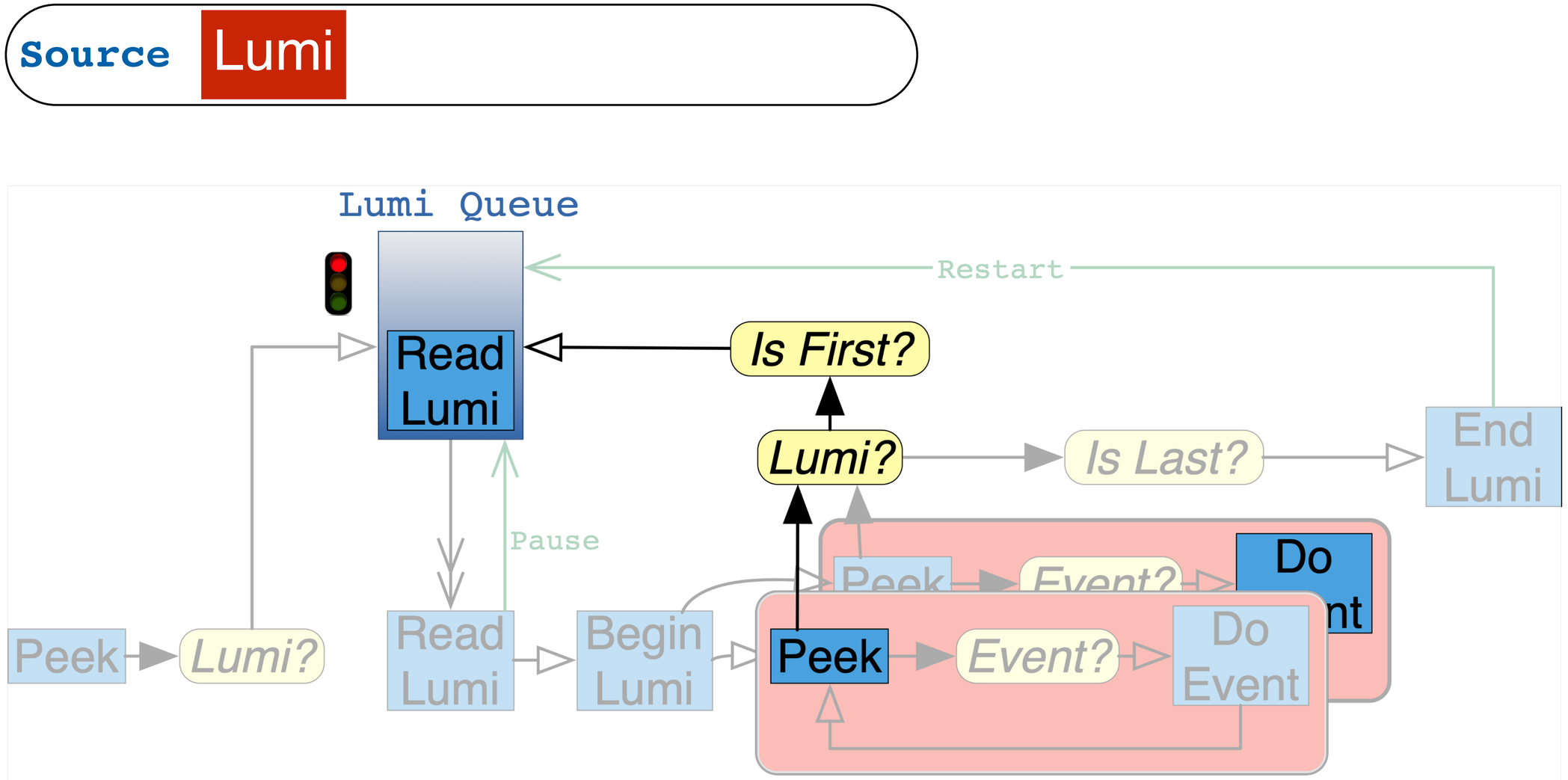
Lumi Processing with Queue



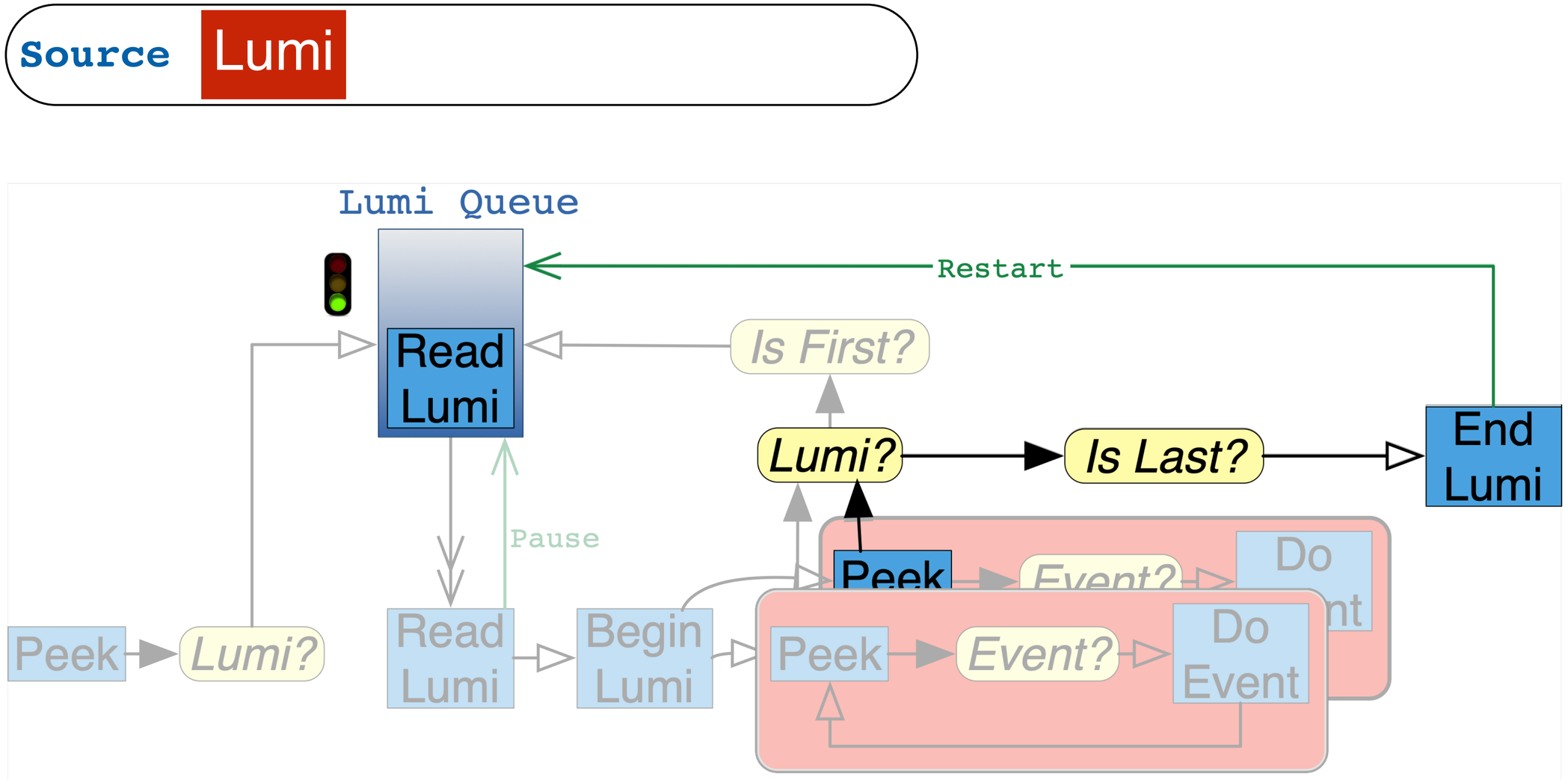
Lumi Processing with Queue



Lumi Processing with Queue



Lumi Processing with Queue



Measurements

Input file

- 1 Run
- 8 Lumis
- 200 events per Lumi

Standard CMS reconstruction job

KNL Hardware

- Use 64 threads

Measurement variations

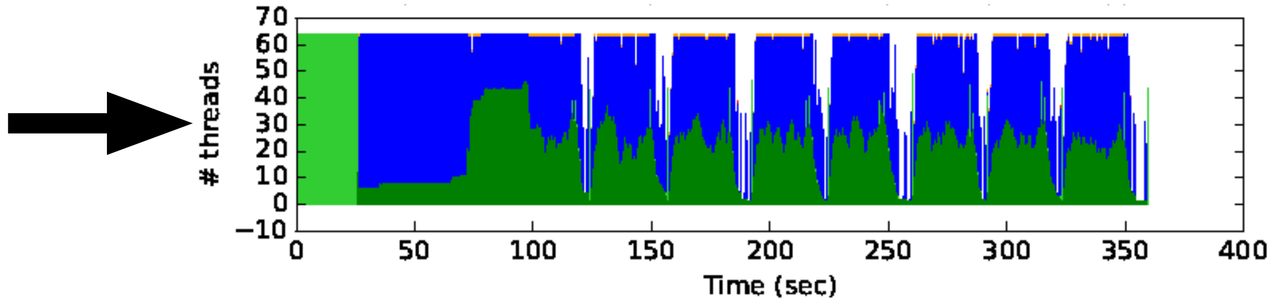
- Only one Lumi at a time
- 8 concurrent Lumis

Reading Concurrency Plots

Total number of concurrent modules

Perfect efficiency when

number of modules == number of threads

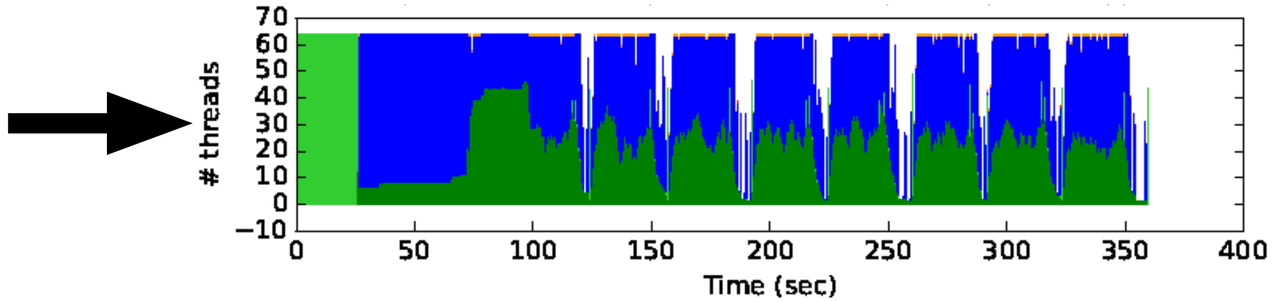


Reading Concurrency Plots

Total number of concurrent modules

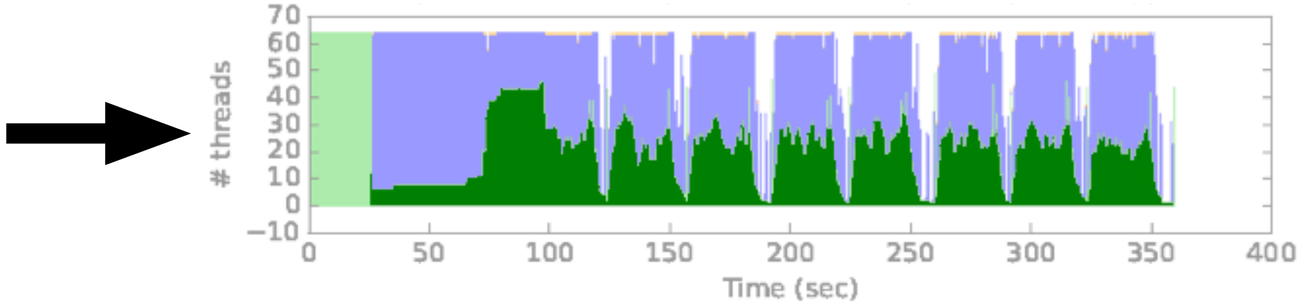
Perfect efficiency when

number of modules == number of threads



Dark Green

Number of concurrent events with modules actually running

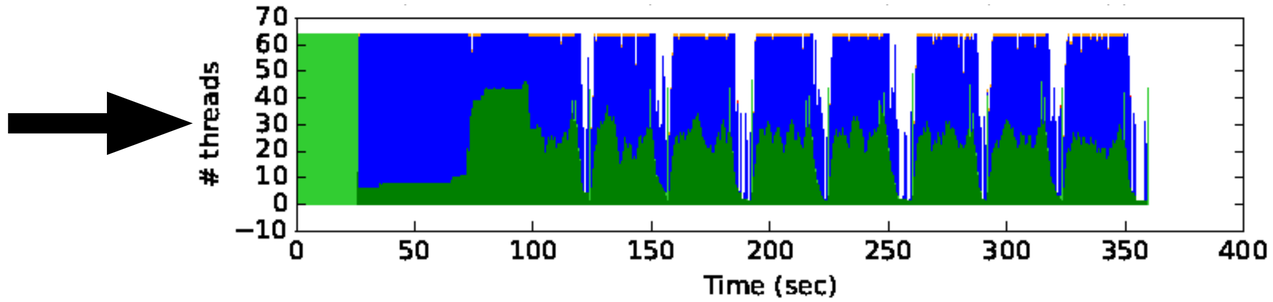


Reading Concurrency Plots

Total number of concurrent modules

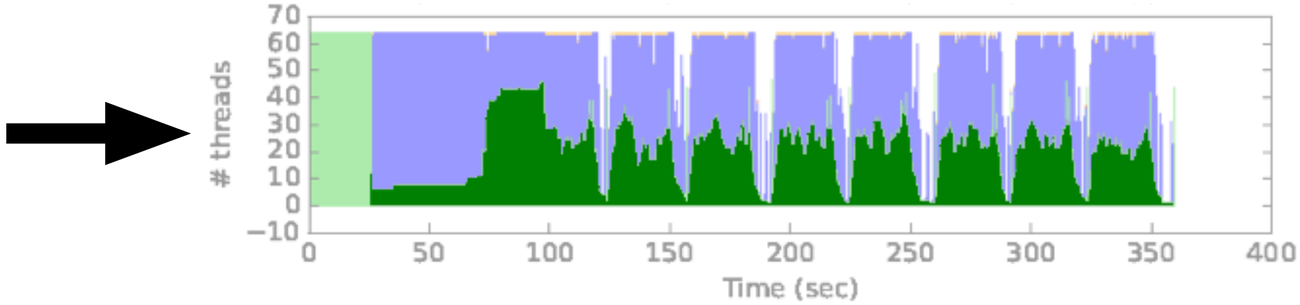
Perfect efficiency when

number of modules == number of threads



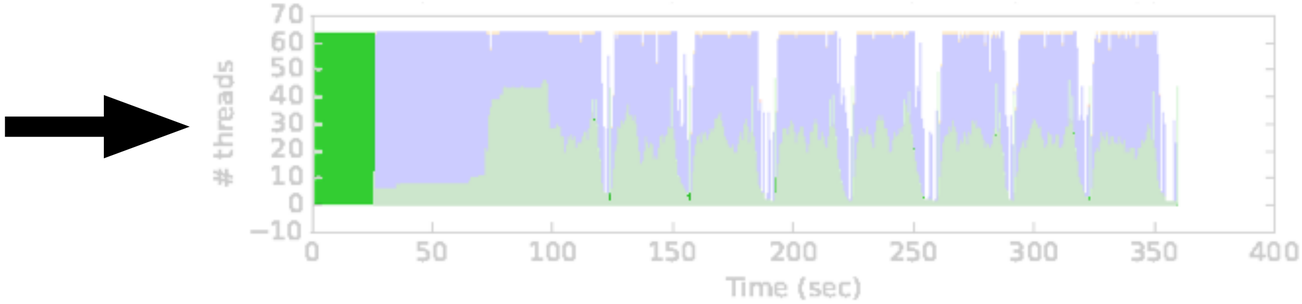
Dark Green

Number of concurrent events with modules actually running



Light Green

Number of concurrent modules processing Lumis or Runs

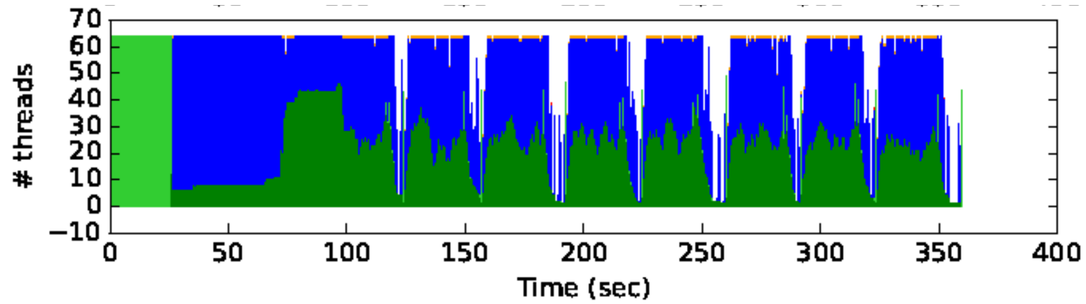


Measurement Results

Single Lumi

Synchronizing on Lumi Boundaries

Thread utilization is poor



Results

Single Lumi

Synchronizing on Lumi Boundaries

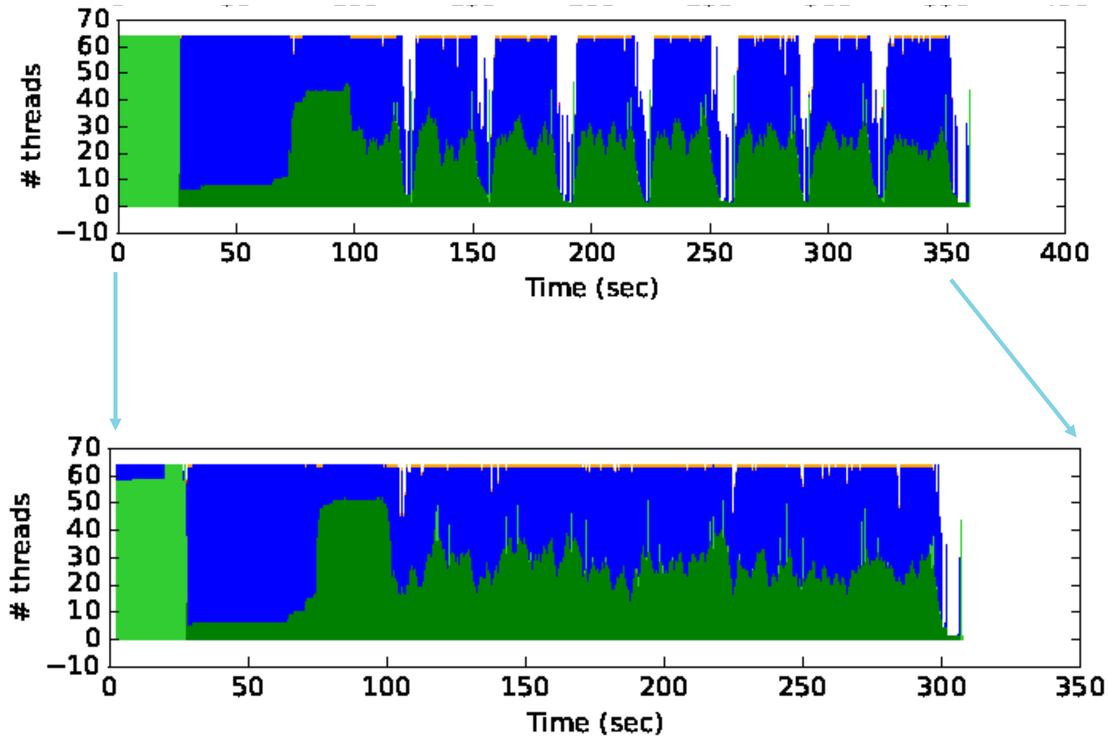
Thread utilization is poor

8 Concurrent Lumis

Synchronizations are gone

Excellent thread utilization

Job finishes faster (~15%)



Complication

CMS supports modules which can only handle one thread at a time

- The framework serializes access to those modules

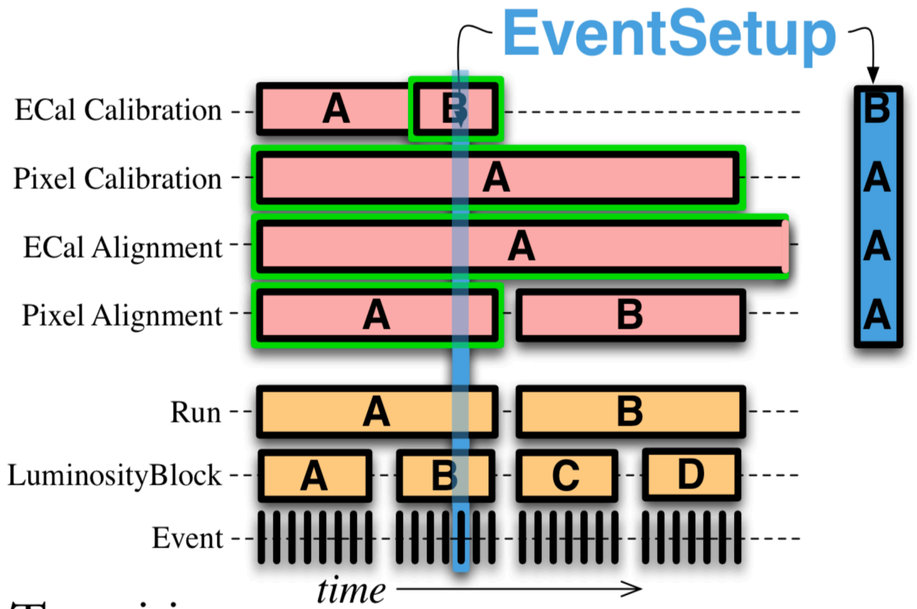
Serial module can *opt in* to see Lumi and/or Run transitions

- Module will not see next Lumis beginLumi until it has seen last Lumis endLumi

Concurrent Runs

- We plan to add support for concurrent Runs in the future.
- For the present there are other higher priorities
- The performance gain related to this is less significant than other things we are doing now.
- The design will be analogous to the design for concurrent lumis.

EventSetup System Manages Data Associated with IOVs



Data Taking Transitions

Runs contain LuminosityBlocks which contains Events
Data products can be associated with each transition

Conditions

Data or algorithms which are have an *interval of validity* (IOV)
An IOV has a begin and end *time*
usually specified as (Run, LuminosityBlock)
Data/algorithms with same IOV are grouped into a *Record*
All Records valid for a particular instance in time are in the *EventSetup*

Concurrent IOVs (Interval of Validity)

- We've added a Limited Task Queue for each record type. This is the same mechanism we use to control concurrent lumis.
- We can independently configure the number of concurrent IOVs for each type of record
- The task to start a lumi will not be placed into its limited queue until all the IOVs it requires have started

Concurrent IOVs (Interval of Validity)

- We added support for concurrent IOVs in 2019.
- Framework support exists now and is deployed although CMS still has work to do to get this validated and in use in production
- We expect to need this in the future more than now

Modules Providing IOV Data

- This year (2020) we've added support for running modules that provide IOV related data concurrently
- It is implemented in a manner similar to the way Event data is handled
- Modules must declare the data they produce and consume.
- Modules do not run until data they consume is ready
- CMS is in the middle of a migration for all modules that consume this type of data to declare what they consume
- Other than the consumes declarations, the feature is fully implemented and working in the CMS Framework
- The migration can be implemented incrementally.

ProcessBlock Data

- This is a new feature under development this year.
- It allows transitions and production of data at the beginning and end of each process.
- It is similar to beginJob and endJob transitions we previously supported, but allows production of data, concurrent execution of modules, and ordering execution of modules at the beginning and end of processes.
- A transient version of this is already implemented and in use by CMS' data quality monitoring code.
- We are currently implementing persistence for this type of data.

Conclusion

CMS implemented concurrency for Non-Event data needed to support concurrent events and modules prior to 2016.

Concurrent Lumis were introduced in 2018.

Concurrent IOVs were introduced in 2019 and concurrent execution of modules producing data managed with IOVs was introduced in 2020.

ProcessBlock data and transitions are coming in 2020.

Concurrent Runs are planned for the future.