Progress on CMS' Threaded Framework

Christopher Jones *FNAL*

Outline

Parallelization

Transitions and Modules

Context

Parallelization

Multiple Levels

Run multiple events simultaneously

Allows memory sharing of long IOV items

Introduces new concepts: *Global* and *Stream*

Within one event run multiple modules simultaneously

Have to take into account module dependencies

Want to minimize any required changes to module code

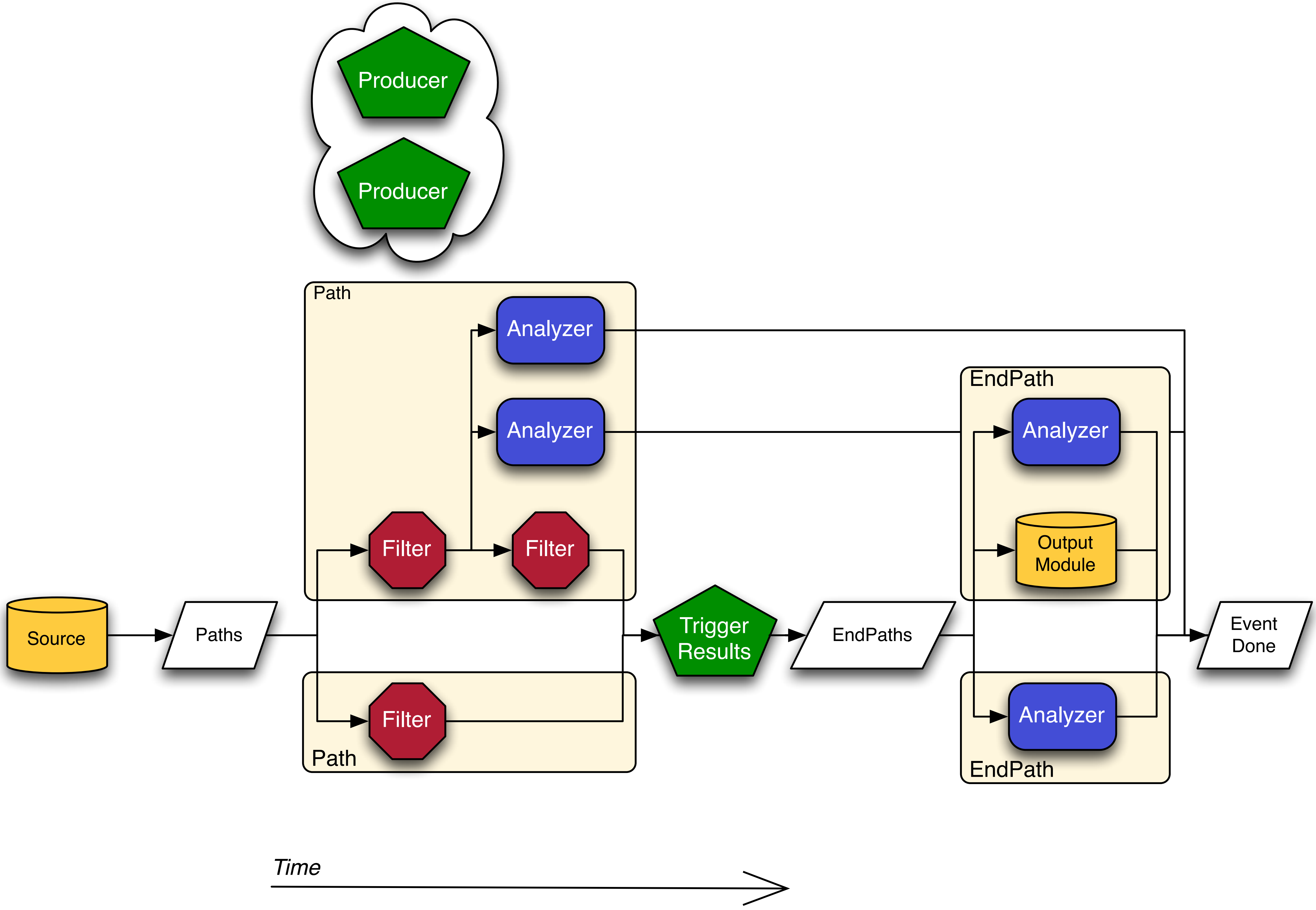
Within one module be able to run multiple tasks simultaneously

Intel's Threaded Building Blocks used for all of the above

Break down work into 'tasks' and TBB can run the tasks in parallel

http://threadingbuildingblocks.org

Sub-Event Parallelization



Sub-Event Parallelization

Paths can run in parallel

- Filters on one path must run in series
- Analyzers can run once all Filters in front of them have finished

EndPaths can run in parallel once all Paths have finished

- That is when the TriggerResults is created
- Modules on the same EndPath can run simultaneously

Producers can run in parallel

- They run the first time their data is requested for that Event, Lumi or Run

Sub-module Parallelization

Can use TBB directly inside a module

TBB will handle scheduling tasks for both modules and sub-modules

TBB has some convenience functions

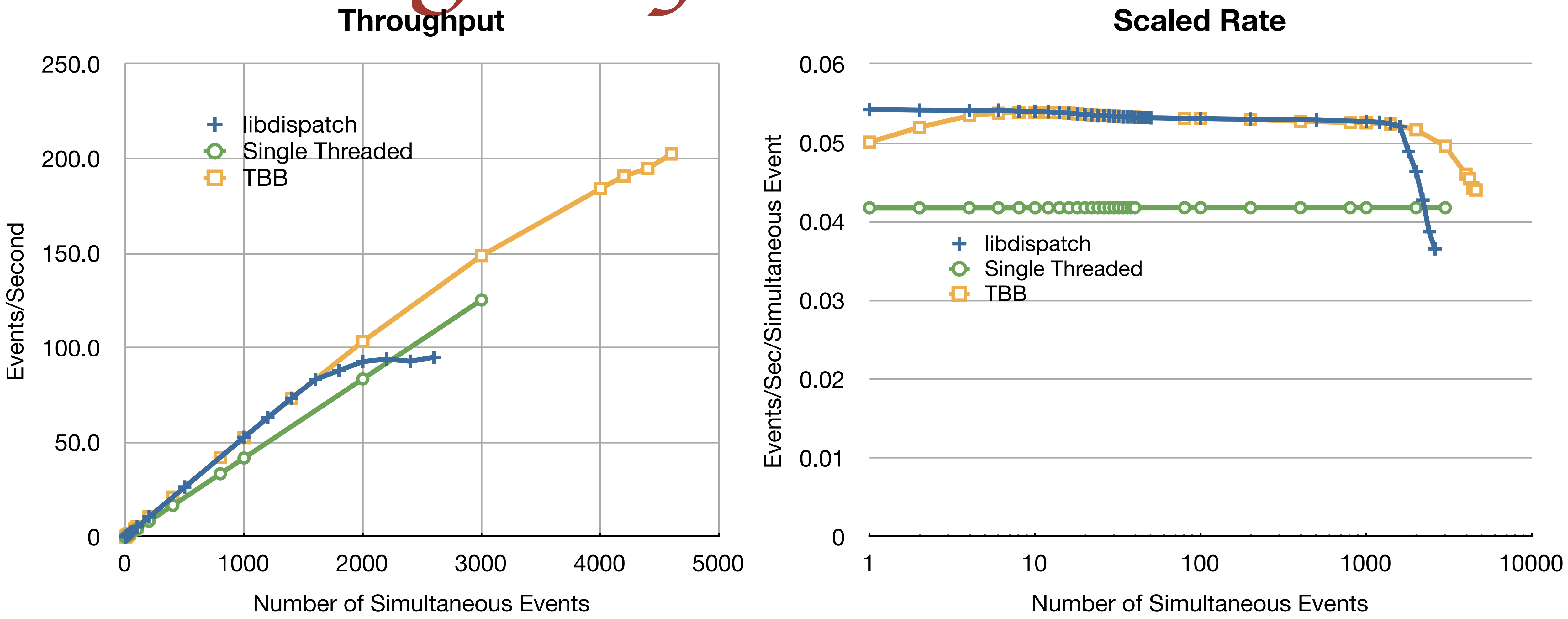
```
std::vector<Results> results(input.size(),Results());
tbb::parallel_for(0U,input.size(),
                  [&](unsigned int i) {
   results[i]=doWork(inputs[i]);
   });
```

Can create own tasks for complex algorithms

```
   class MyTask : public tbb::task { ... };
   ...
   MyTask* mt = new (tbb::task::allocate_root()) MyTask;
   tbb::task::spawn_root_and_wait( mt );
```

Framework will provide some helper classes

Scaling: Infinite Cores



All Producers are calling usleep

TBB stops scaling around 2000 simultaneous events (se)

Is using 1.3 threads/simultaneous event

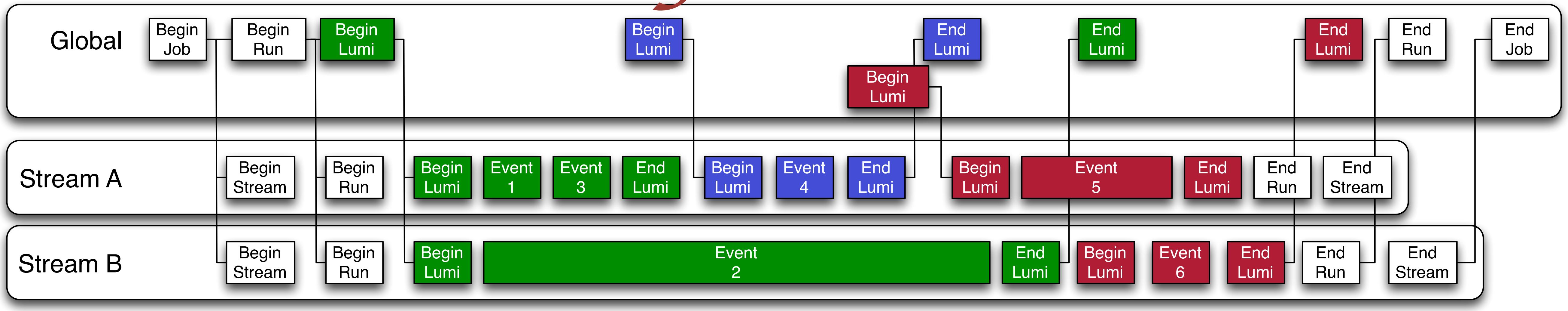
Lowering threads/simultaneous events improves scaling limit slightly

libdispatch hits scaling limit around 1600 se

Single threaded framework hits memory limit at 3000 se

Transitions and Modules

New Concepts



Global

Sees transitions on a 'global' scale

- see begin of Run and begin of Lumi when source first reads them
- sees end of Run and end of Lumi once all processing has finished for them

Multiple transitions can be running concurrently

Events are not seen 'globally'

Stream

Processes transitions serially

- begin run, begin lumi, events, end lumi, end run

Multiple streams can be running concurrently each with own events

- One stream only sees a subset of the events in a job

Present cmsRun is equivalent to running with only one stream

Paths and EndPaths are a per Stream construct

- The same module can be shared across Streams
- The Stream knows if a module was run for a particular event

Module Interfaces

Present modules will work in threaded framework

- Will only be given one event at a time
- This will cause performance bottlenecks

Have design for new thread-safe module interfaces

- Stream modules
- Global modules

Documentation available at

https://twiki.cern.ch/twiki/bin/view/CMSPublic/FWMultithreadedFrameworkModuleTypes

Stream Module

Replicate an instance of a module configuration for each Stream

e.g. if have 8 Streams in a job will have 8 copies of a module

Each copy has its own instance of the class member data

NOTE: There are ways to allow sharing data between copies in a thread safe manner

A Stream only processes one Event at a time

A module copy will only be called at most once per event

Member data does not have to be thread safe

One Stream only sees a fraction of the Events in the job

Therefore a module copy only sees a fraction of the events

Not a problem for most EDProducers and EDFilters

Easy to convert from 'Classic' to Stream interface

```
class TrackClusterRemover : public edm::stream::EDProducer<> {
…};
```

Most Filters and Producers should be easy to convert

Global Module

One instance of a module shared by all Streams

- One module sees all Runs, LuminosityBlocks and Events

All member functions and member data must be thread-safe

- The interface provides ways to help you with thread-safety

Only use if

- Need to share as much memory across Streams as possible or
- Algorithm must see all Runs, LuminosityBlocks or Events

OutputModules most likely to be global modules

Context

Need to Follow Context

Services

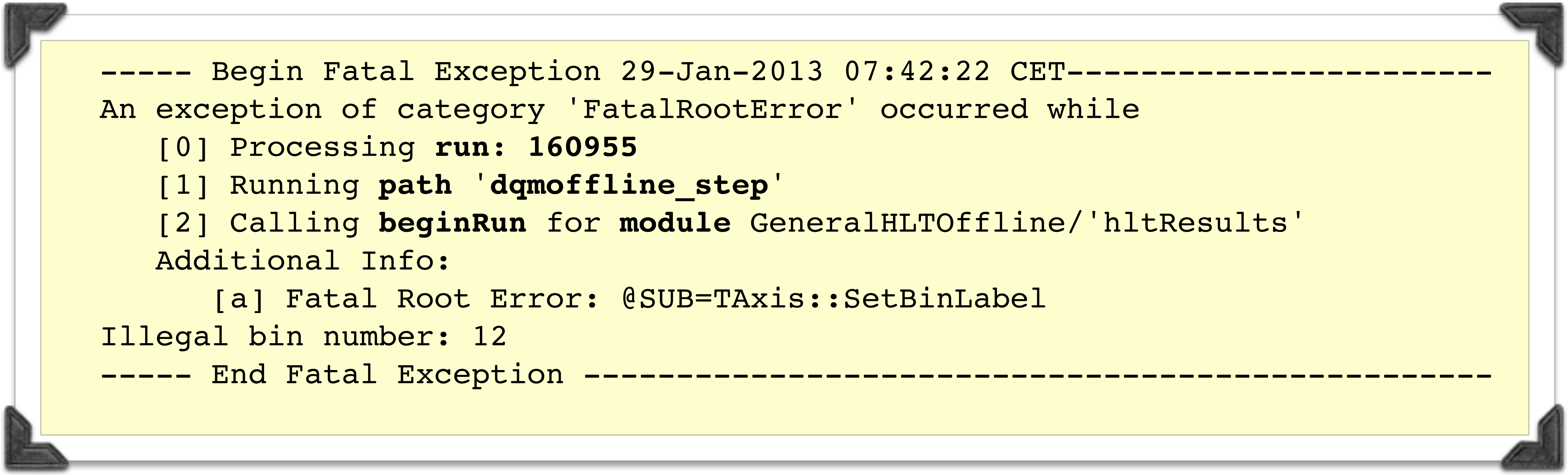
Provide global access to 'non-physics' functionality

Monitor present state of the framework

- MessageLogger prints which module issued a log message
- Tracer prints out all internal transitions for debugging

Exceptions

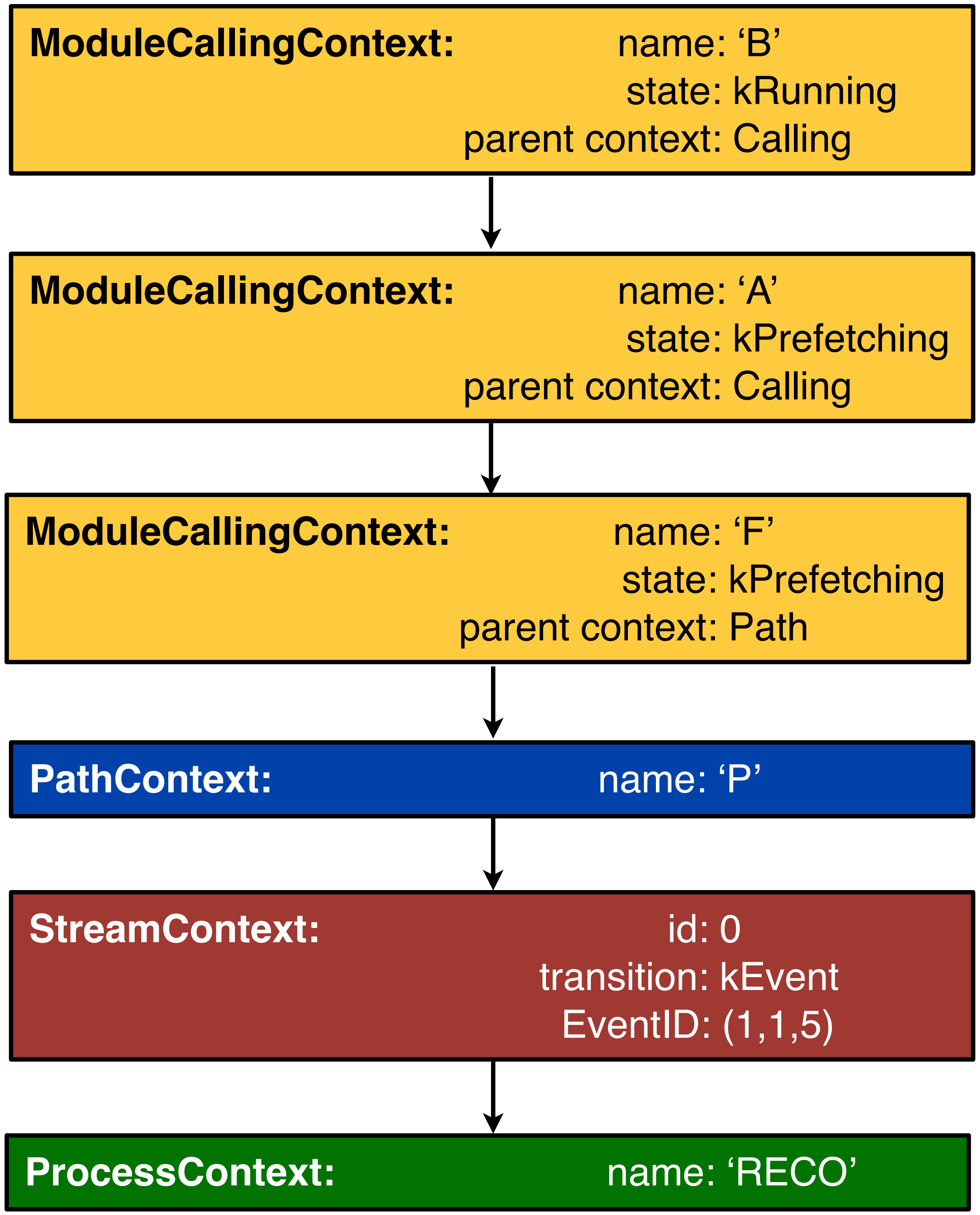
Messages should include the framework state



```
----- Begin Fatal Exception 29-Jan-2013 07:42:22 CET-----------------------
An exception of category 'FatalRootError' occurred while
   [0] Processing run: 160955
   [1] Running path 'dqmoffline_step'
   [2] Calling beginRun for module GeneralHLTOffline/'hltResults'
   Additional Info:
      [a] Fatal Root Error: @SUB=TAxis::SetBinLabel
Illegal bin number: 12
----- End Fatal Exception -------------------------------------------------
```

Need to understand context of the call site

Context Linked List



ModuleCallingContext: name: 'B'
state: kRunning
parent context: Calling

ModuleCallingContext: name: 'A'
state: kPrefetching
parent context: Calling

ModuleCallingContext: name: 'F'
state: kPrefetching
parent context: Path

PathContext: name: 'P'

StreamContext: id: 0
transition: kEvent
EventID: (1,1,5)

ProcessContext: name: 'RECO'

Context Thread Safety

All *Context objects can be preallocated

Run,Lumi,Event #s have to be updated once per transition

ModuleCallingContext only needs pointer to parent set

This happens right before calling the module and in the same thread

All other values are immutable