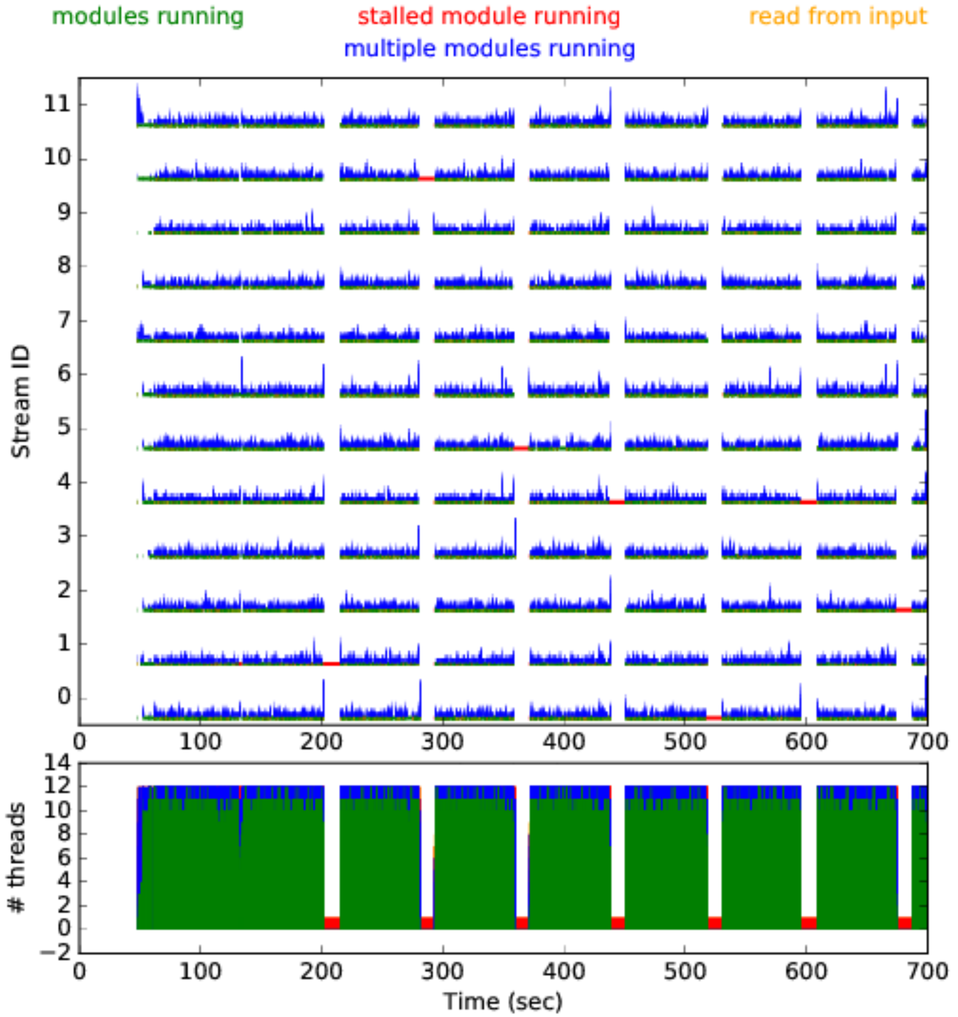
# CMS ROOT I/O experience and experiments

Dan Riley (Cornell)
ROOT I/O Workshop, 2017-06-12

# ROOT I/O limits CMS scaling

## CMS production jobs are multithreaded

- **Production jobs currently use 4 cores with 4 framework event streams**
- **Output is handled by "one" modules that can only be active on one thread at a time**
- **ROOT output is the dominant source of output stalls**
  - We lose efficiency with more than 4 cores, preventing us going to 8 cores
- **Compression is the principal bottleneck**
  - Especially for AOD and MINIAOD data compress with LZMA



# Using TMemFile for buffering

TMemFile is a TFile that uses memory as the storage medium

- **Interfaces are standard TFile**

Concept (from Philippe): TMemFiles as intermediate buffers, conceptually similar to TBufferMerger specialized for CMS

- **Events are written to an intermediate output module that writes to a TMemFile**
  - Framework maintains a pool of intermediate TMemFile output modules
  - Tradeoff: need enough parallelism to mitigate stalls, but too many wastes memory
- **Full TMemFiles are passed to a final output module to merge to the final output file**
- **High-level design choices involve where the compression happens**
  1. Write TMemFile uncompressed, compress in the merge
  2. Write TMemFile compressed, fast-clone in final output

# Prototype for scale testing

Prototype for scale testing writes the TMemFile compressed, fast-clone in final output

- **This design option was investigated first because it can be approximated without major changes to our framework**
  - Synthetic tests showed the need for realistic data
- **This is essentially fast-cloning a bunch of small files**
  - Unfortunately means more incompletely filled baskets and hence worse compression
  - Tradeoff with larger TMemFile flush threshold, more memory should give better compression
  - Basket sizes have to match for fast-cloning, so have to be careful about basket optimizations? This might be addressed by a “training” period to optimize basket sizes before creating the TMemFile pool

# Prototype Implementation

## Create a TMemFile producer with *n* instances

- **each instance handles one out of every *n* streams determined by a modulo filter**
- **TMemFile producer *consumes* all the products the output module wants**
  - Writes to a TMemFile using the same routines as for writing the final output file
  - When the size of the TMemFile passes threshold, do AutoSave("FlushBaskets") to finalize the event
  - Completed TMemFile is passed to the final output module
- **Final output module *consumes* the TMemFiles and fast-clones to the output file**

## Limitations

- **Framework is done with the Event when the TMemFile producer writes it**
  - But prototype still waits for the basket flushing and fast-cloning
  - So could do better by making those steps asynchronous?
- **Fixed assignment of streams to TMemFile modules**
  - Dynamic assignment from a pool would be better

# Test setup

Dataset:

- 5000 events (11 GB) of CMS TTBar simulated data, phase2 upgrade geometry and conditions
- Recent (CMSSW_9_2_1) CMS software release
- RECO job producing MINIAOD only, LZMA compression 9, 12 framework threads
  - Output file is only ~135 MB

Test machine

- 12 core Haswell E5-2620v3

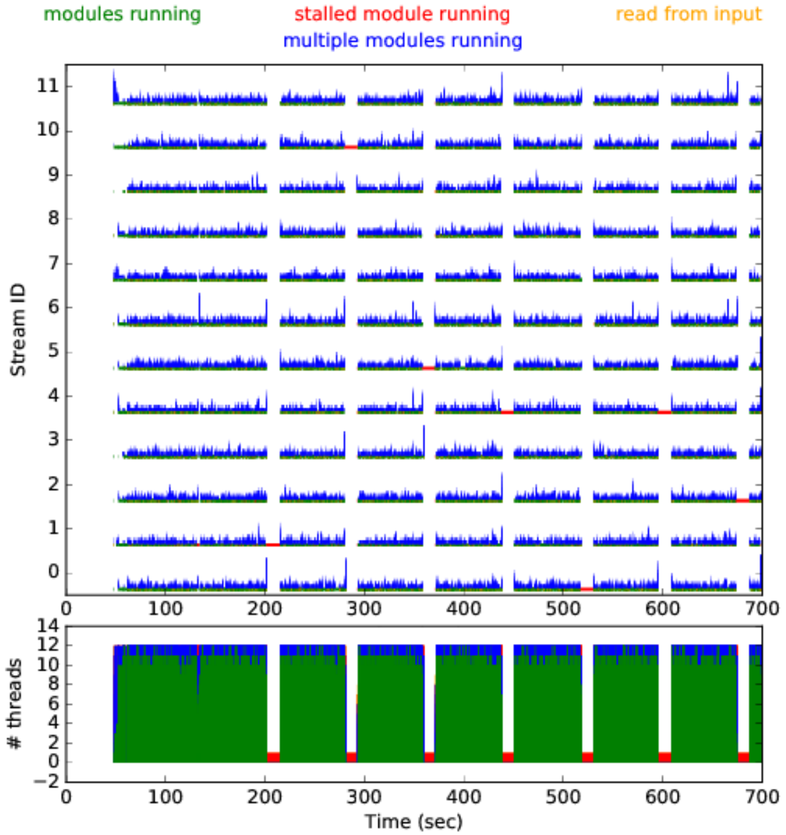
Variables (lots of knobs to turn):

- # of TMemFile intermediate output buffers
- TTree AutoFlush size, CMS default is 15 MB
  - Results show we should probably raise this substantially for multithreaded programs
  - Making this too small increases virtual memory and resident set size!
- Branch basket size, CMS default is 16 KB

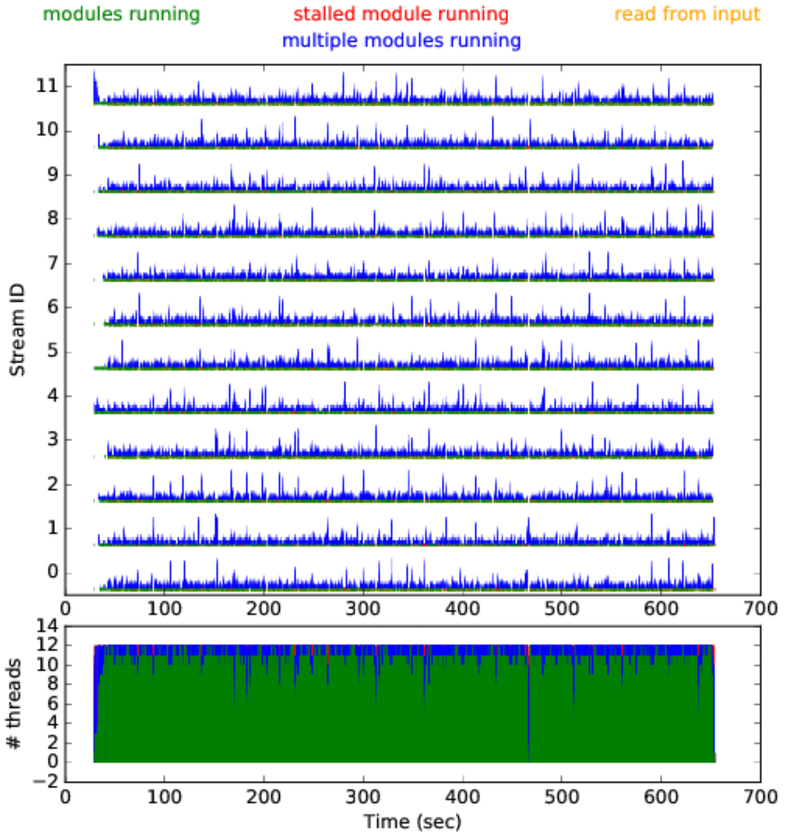
# A fair comparison is…hard to find?

Naively

- **Baseline is 15 MB autoflush size**
- **So if we have two TMemFile intermediates, try an 8 MB flush size**
  - 8 MB blows up VSIZE and RSS!
  - Output file 40% larger
  - Does give an appreciable efficiency gain




Baseline

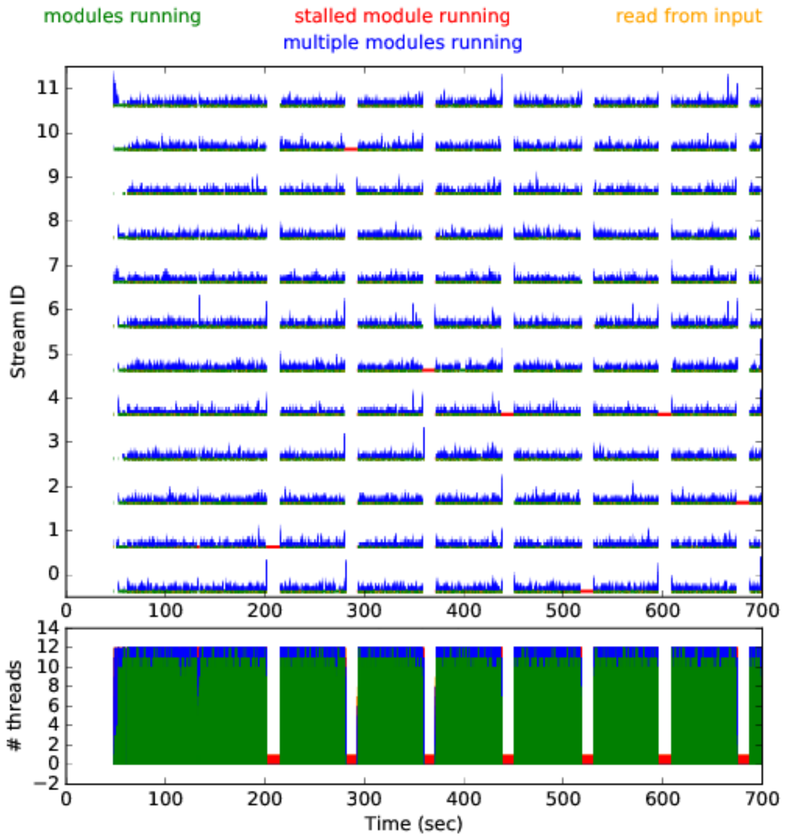



2xTMemFile 8MB

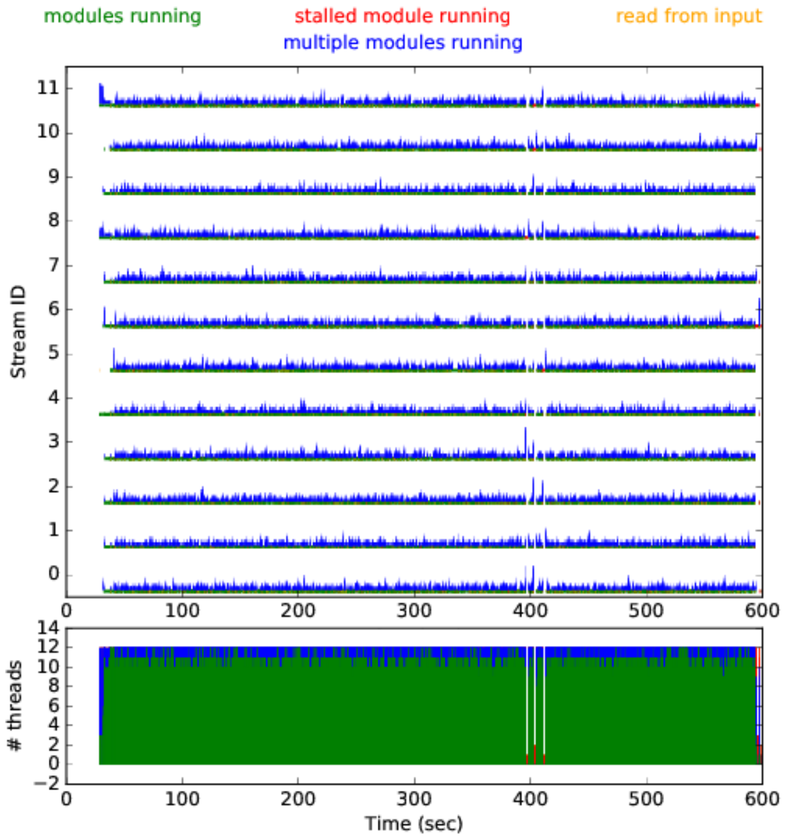
# A biased comparison…

4 TMemFile intermediates, 134 MB flush size each

- Note scale change for right plot
- File size is ~15% larger
- RSS 280 MB larger than baseline, less than expected?
- TMemFile case only flushes twice (145MB output)




Baseline




4xTMemFile 134MB

# Conclusions?

CMS will investigate our AutoFlush size choice for multithreaded jobs

- Buffer and basket sizes haven't been retuned for multithreaded jobs or current file contents
- Tuning ROOT parameters may address scaling issues to 8 or 12 cores
- Need to test with more realistic test cases, multiple output files
- RSS behavior seems counter-intuitive?

TMemFile intermediates show some promise for improving scaling at higher core counts, but…

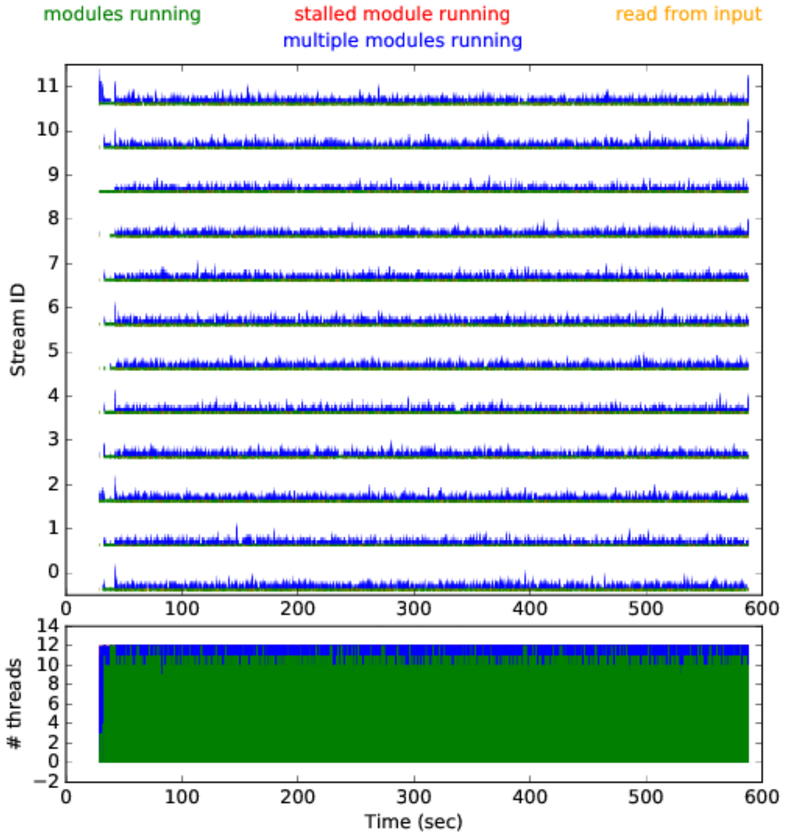
- Need to test at higher core counts (e.g., KNL)
- Technical limitations discussed on slide 5 may be hiding performance gains
- File size increase is an issue
  - But could be addressed if the intermediate buffers let us increase basket size without adding to stall time

# With autoflush larger than output file

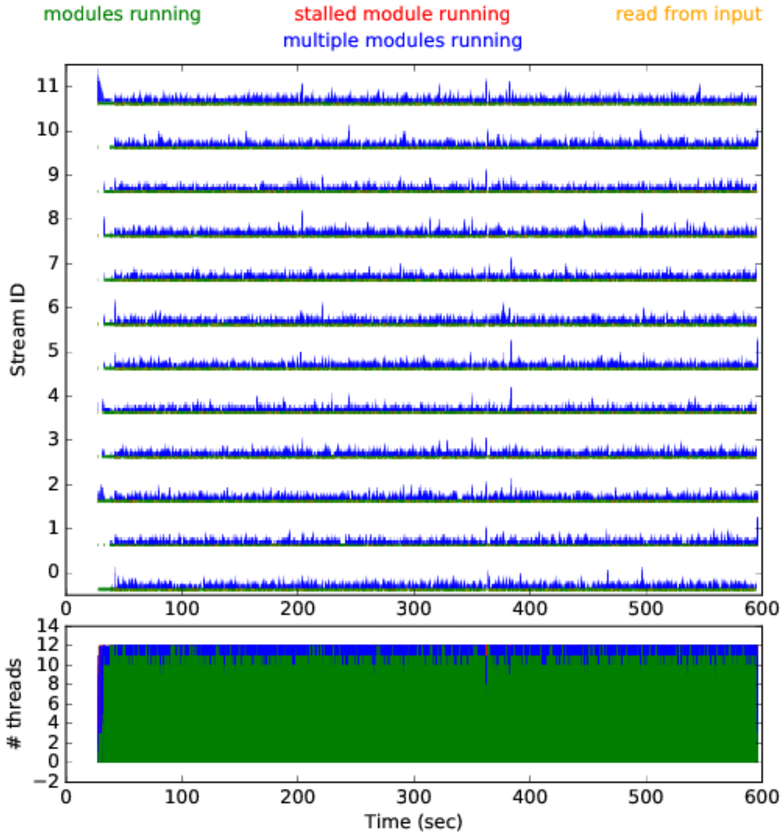
4 TMemFile intermediates, 500 MB flush size each vs. baseline with 500 MB autoflush

TMemFile basket size quadrupled to 64 KB

- File sizes same to ~1%




Baseline w/500MB




4xTMemFile 500MB