Software performance in CMS

David Lange

May 4, 2016

Motivation: Technical improvements can pay off

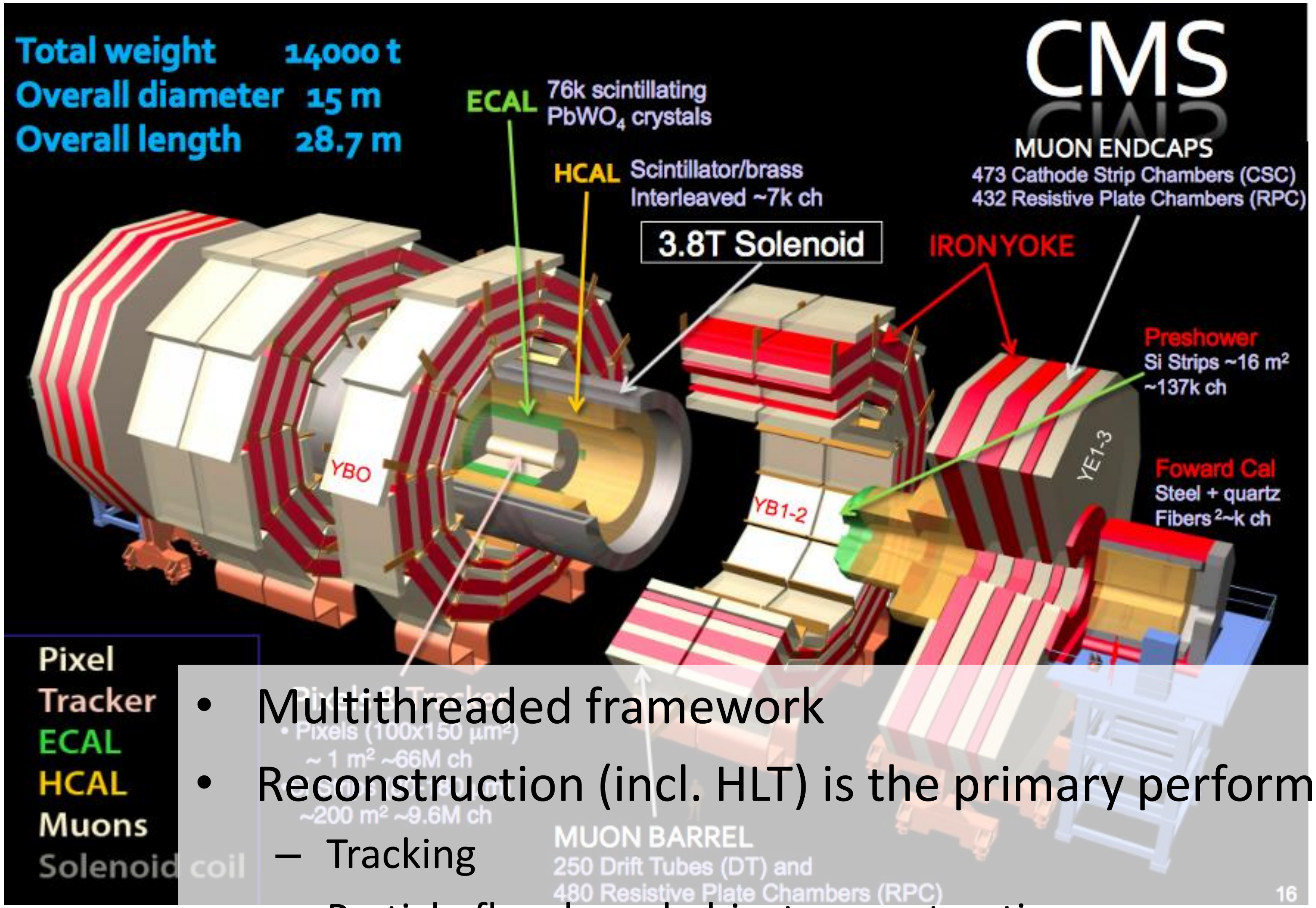
- CPU time/event and event size determine computing requirement per trigger

CPU needed = N Events × CPU/Event

Faster reconstruction =?= More Physics

Or: less cost to do the same physics

Basic components in CMS (software)

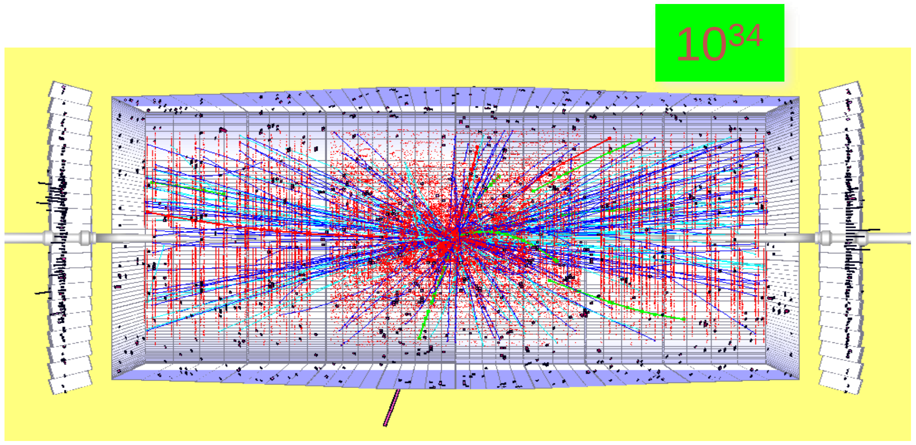


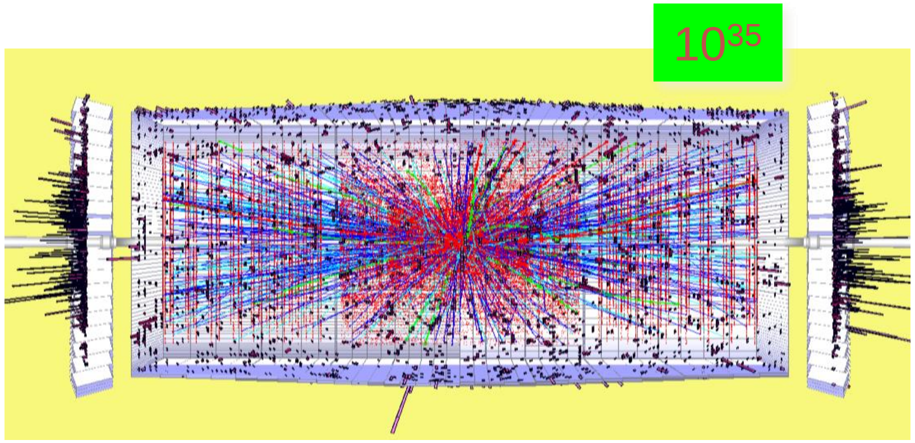
- Multithreaded framework
- Reconstruction (incl. HLT) is the primary performance target
  - Tracking
  - Particle flow based object reconstruction
  - For the future: high granularity endcap calorimetry

Challenge for LHC: Event complexity and data rates continue to grow

CMS at the start of Phase 2 (HL-LHC):

- 5x higher output rate of trigger compared to 2016
- Significantly more complex events Events (140+ interactions/crossing)

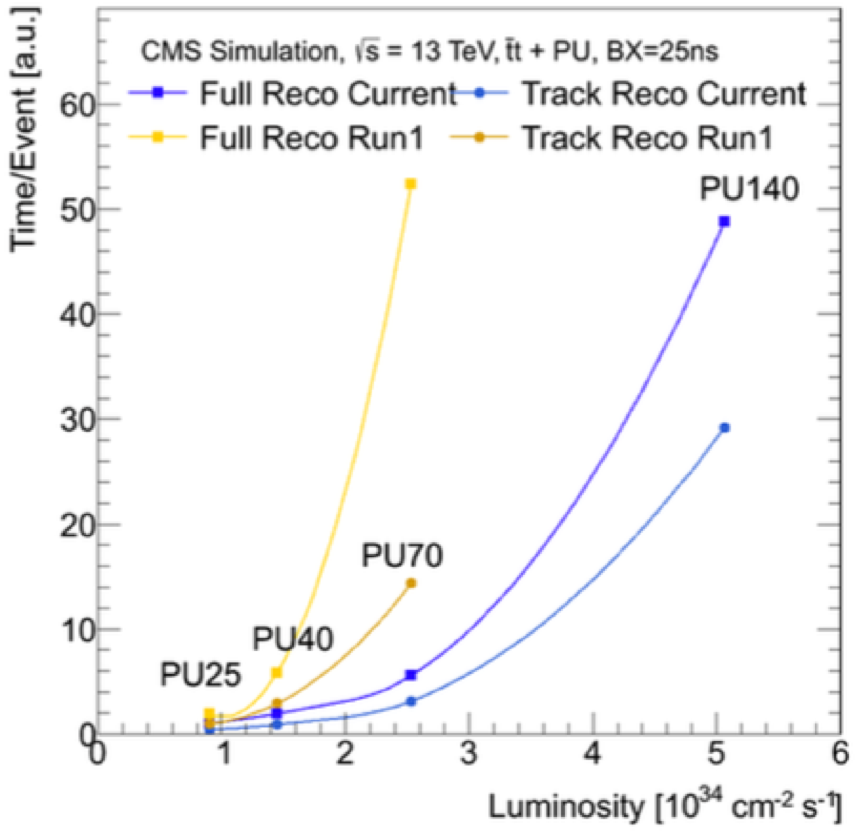




HL-LHC presents increased challenges for Triggering, Tracking and Calorimetry, in particular for low to medium P_T objects

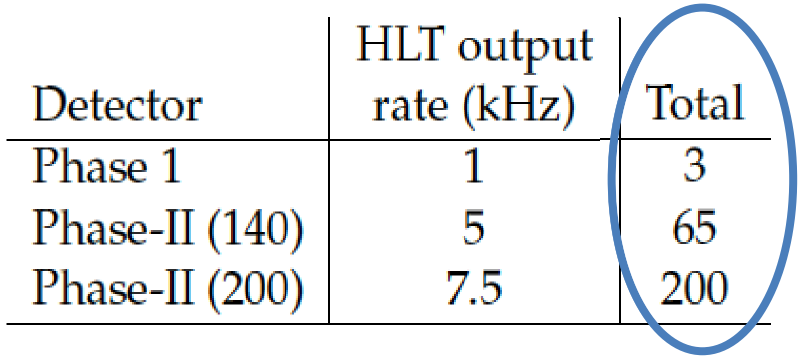
Processing time scales drastically with event complexity

- Resources needed by combinatorically driven reconstruction algorithms increase non-linearly with interactions/crossing
  - Dedicated analyses of performance at high pileup resulted in scaling improved in recent years



HL-LHC: Size of the processing problem if we continue as we continue with our current model

- Factoring in the trigger rate and taking a weighed average of the data and simulation tasks we see the computing challenge is 65-200 times worse than Run2 (2016-like conditions)



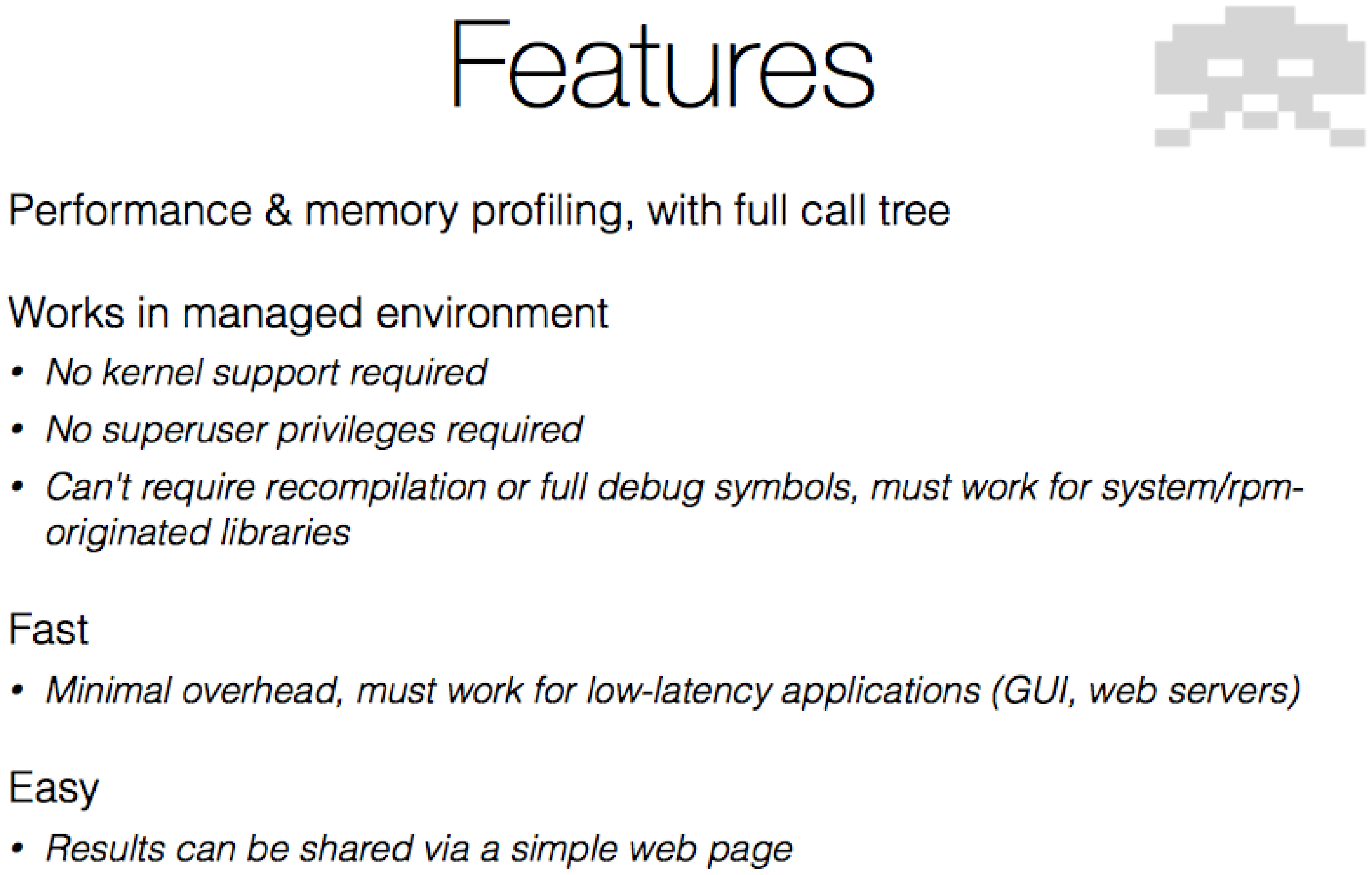
Detector	HLT output rate (kHz)	Total
Phase 1	1	3
Phase-II (140)	5	65
Phase-II (200)	7.5	200

Scale of computing resource needs relative to Run 2 including the increase in projected HLT output rate

(More detailed estimates in our 2015 Technical Proposal) https://cds.cern.ch/record/2020886?ln=en

Software improvements are an important piece to close this gap

Most relied on optimization tools in CMS: igprof



Features

Performance & memory profiling, with full call tree

Works in managed environment

- *No kernel support required*
- *No superuser privileges required*
- *Can't require recompilation or full debug symbols, must work for system/rpm-originated libraries*

Fast

- *Minimal overhead, must work for low-latency applications (GUI, web servers)*

Easy

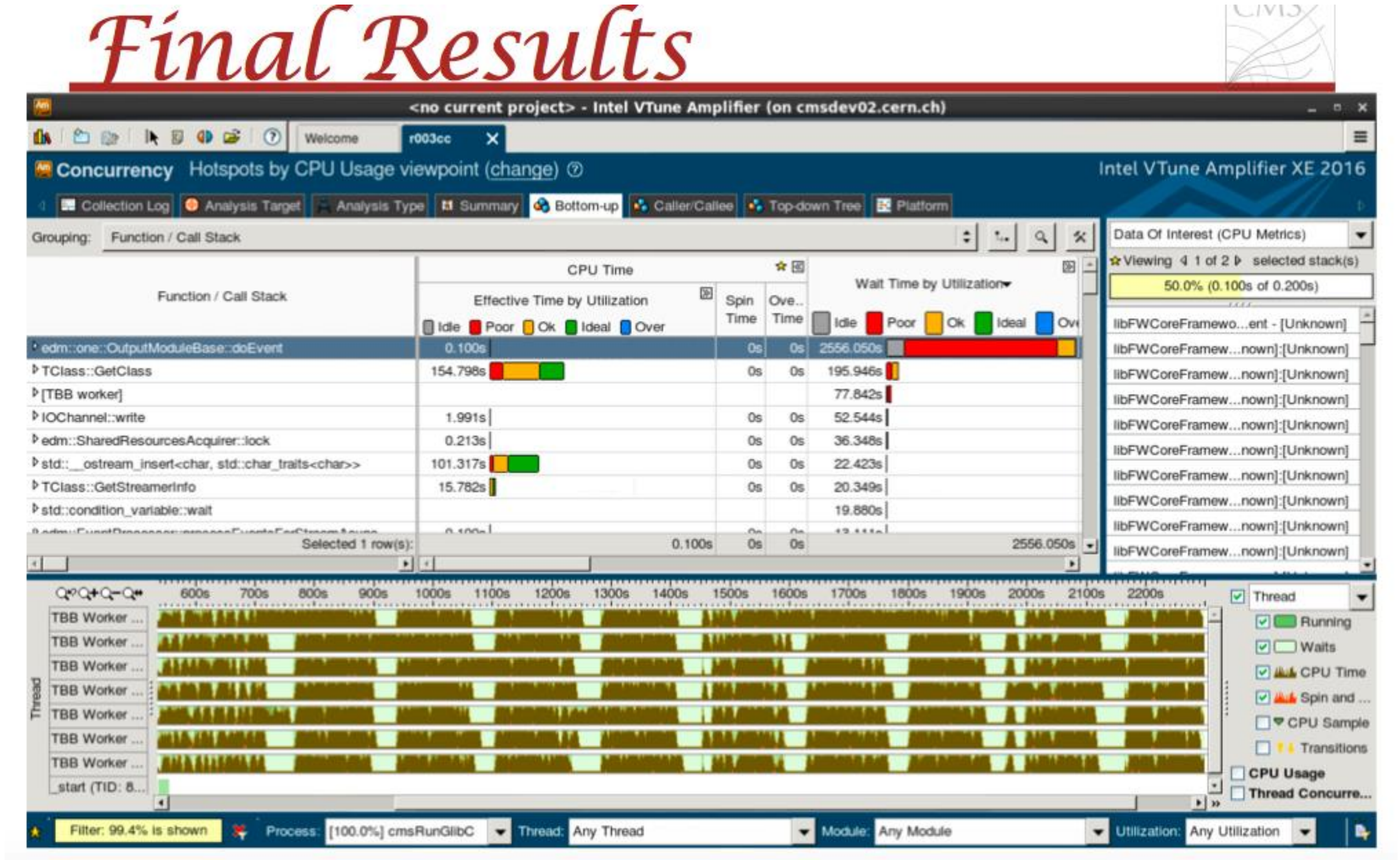
- *Results can be shared via a simple web page*

Recent targets

Large individual targets are gone. Tool like igprof particularly important to identify hot spots for investigation at this stage

- Redundant calculations: Remove
- Numerical precision: Use whats needed
- Math functions: VDT where possible
- Push persistency overhead away: Use original objects in algorithms
- Take advantage of C++11 constructs
- Vectorization: Ensure loops are vectorized by compiler (but we miss automatic methods)

Threading optimization: We use VTune very successfully



<no current project> - Intel VTune Amplifier (on cmsdev02.cern.ch)

Welcome | r003cc

Concurrency Hotspots by CPU Usage viewpoint (change)

Intel VTune Amplifier XE 2016

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time	Wait Time by Utilization (Idle, Poor, Ok, Ideal, Over)
edm::one::OutputModuleBase::doEvent	0.100s	0s	0s	2556.050s
TClass::GetClass	154.798s	0s	0s	195.946s
[TBB worker]				77.842s
IOChannel::write	1.991s	0s	0s	52.544s
edm::SharedResourcesAcquirer::lock	0.213s	0s	0s	36.348s
std::__ostream_insert<char, std::char_traits<char>>	101.317s	0s	0s	22.423s
TClass::GetStreamerInfo	15.782s	0s	0s	20.349s
std::condition_variable::wait				19.880s
Selected 1 row(s):	0.100s	0s	0s	2556.050s

Data Of Interest (CPU Metrics)

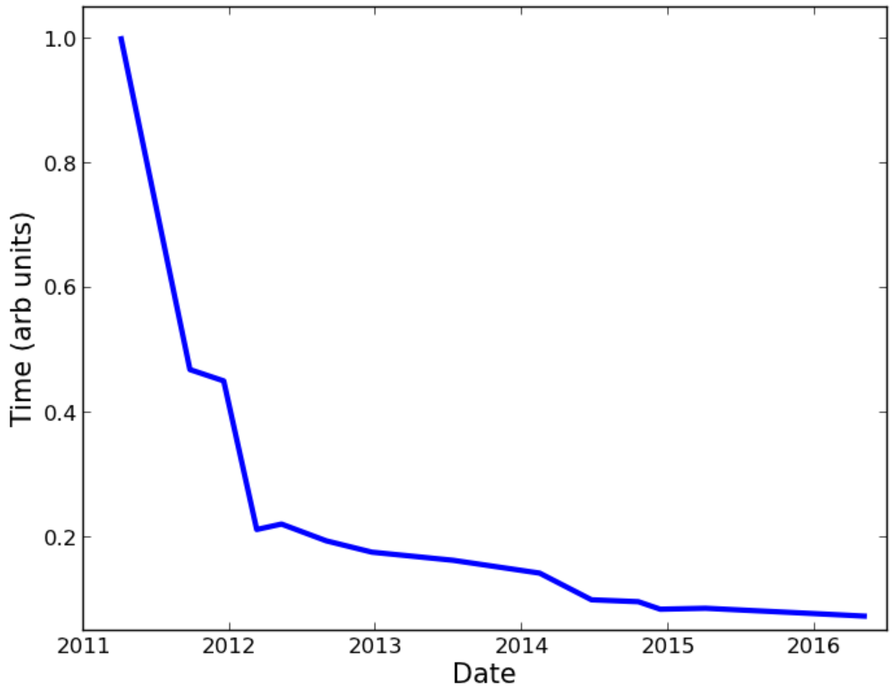
Viewing 1 of 2 selected stack(s)

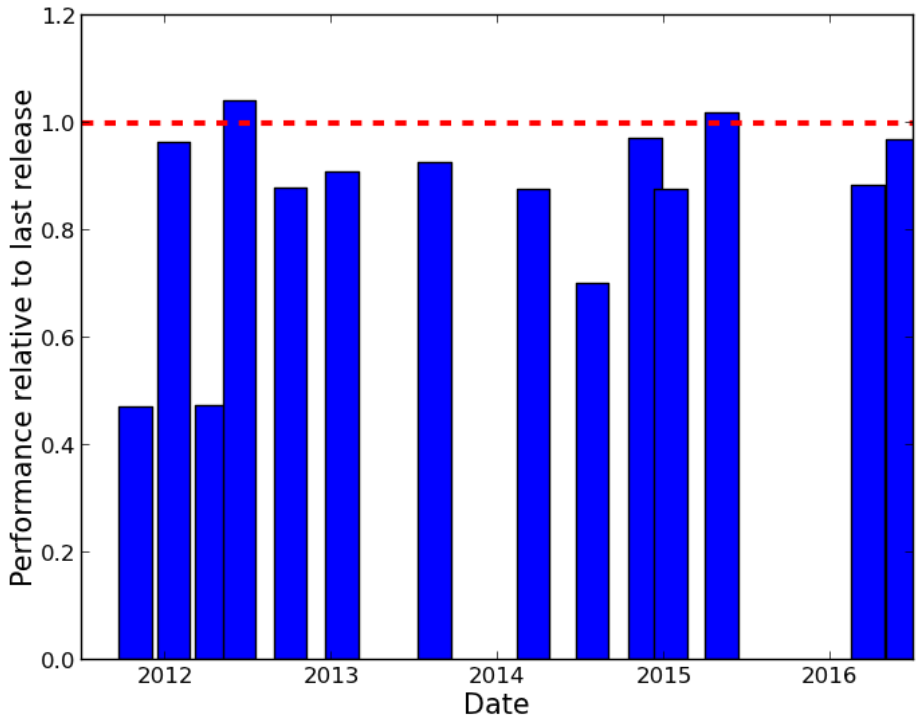
50.0% (0.100s of 0.200s)

libFWCoreFramewo...ent - [Unknown]
libFWCoreFramew...nown]:[Unknown]
libFWCoreFramew...nown]:[Unknown]
libFWCoreFramew...nown]:[Unknown]
libFWCoreFramew...nown]:[Unknown]
libFWCoreFramew...nown]:[Unknown]
libFWCoreFramew...nown]:[Unknown]
libFWCoreFramew...nown]:[Unknown]
libFWCoreFramew...nown]:[Unknown]
libFWCoreFramew...nown]:[Unknown]

Filter: 99.4% is shown | Process: [100.0%] cmsRunGlibC | Thread: Any Thread | Module: Any Module | Utilization: Any Utilization

Reconstruction performance against 2015 data





Incremental progress continues through Run 2 while maintaining the physics

Conclusion

- CMS is successful in achieving continued technical performance gains sing a few tools as a guide for investigation
  - How far can we push purely technical changes before we need to consider impacting physics?

- Future hopes
  - – Detailed CPU/memory profiling with threading – be ready for a many-core environment
  - – Add CMS analysis to our list of applications