Software Performance: Some Observations from ATLAS

Graeme Stewart

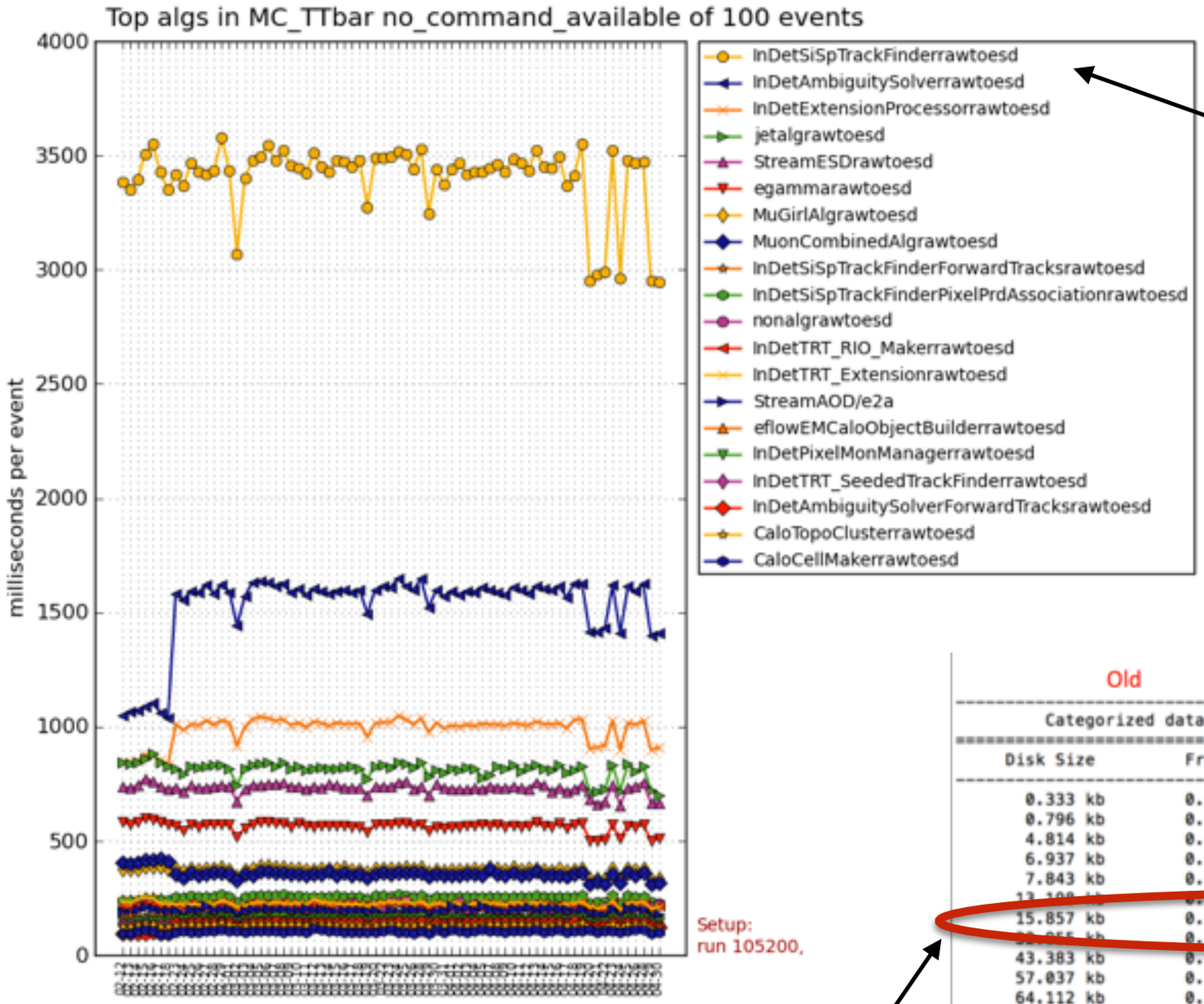
With particular thanks to Antonio Limosani, Andreas Salzburger, Scott Snyder, Shaun Roe, Zach Marshall

University of Glasgow | School of Physics & Astronomy

2016-05-04

You can’t improve what you can’t measure…

- ATLAS Performance Management Board runs standard reconstruction and simulation jobs every day
  - Instrumented to provide information on CPU and memory consumption per algorithm and tool
  - PMB makes regular reports to software meetings
- Unexpected increases have to be understood
  - Most often these are bugs that creep in and blow up cpu times or leak memory
  - Sometimes they are new algorithms that produce better quality physics, e.g., MVA tools
    - Even so, increases need to be justified and, usually, compensated by other reductions



Top consumer of CPU is Tracking/Inner Detector, but everything else matters too!

Old

Categorized data		
Disk Size	Fraction	Category Name
0.333 kb	0.001	*Unknown*
0.796 kb	0.002	MetaData
4.814 kb	0.009	Muon
6.937 kb	0.013	EvtId
7.843 kb	0.015	MET
[illegible]	[illegible]	Calo
15.857 kb	0.031	tau
[illegible]	0.063	egamma
43.383 kb	0.083	BTag
57.037 kb	0.110	PFO
64.112 kb	0.123	Jet
68.881 kb	0.133	CaloTopo
95.344 kb	0.184	Truth
108.080 kb	0.208	InDet
519.570 kb	1.000	Total

New

Categorized data		
Disk Size	Fraction	Category Name
0.333 kb	0.001	*Unknown*
0.803 kb	0.002	MetaData
4.814 kb	0.009	Muon
6.937 kb	0.013	EvtId
7.842 kb	0.015	MET
13.198 kb	0.025	Calo
22.102 kb	0.042	tau
32.955 kb	0.063	egamma
43.383 kb	0.083	BTag
57.037 kb	0.108	PFO
64.112 kb	0.122	Jet
68.881 kb	0.131	CaloTopo
95.344 kb	0.181	Truth
108.080 kb	0.206	InDet
525.820 kb	1.000	Total

Size matters too — even small size increases can become significant on the grid

Standard Candles

- ATLAS uses a few standard jobs that are easy to rerun
  - `Reco_tf.py --AMI q431`
- Making simple things matter: `RunTier0Tests.py`
  - Run standard jobs with and without your changes
    - Automatic comparison made for
      - CPU increases
      - Memory leaks/increases
      - Size of outputs
    - When it's easy developers are far more likely to make the checks!
      - Plus, we'll now reject tags for Tier-0 unless the test output is shown ;-)
- Evidently for benchmark tests the usual considerations apply — well controlled machine, running controlled tests without other loads, etc.

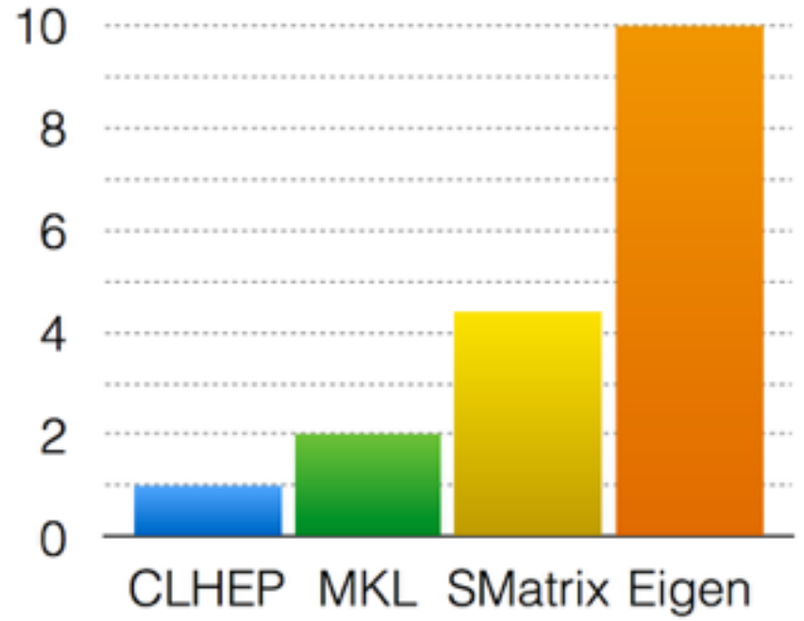
Improving ATLAS Performance — LS1 Experience

- Run1 was very successful for ATLAS, but we reached some real scaling limits on some parts of the software and workflows
- Facing Run2 then, we had experience, some time and new challenges on the horizon
- Then we needed to
  - Identify the problem
  - Evaluate solutions
  - Implement
  - Test

In theory all factorised, but that was hard in LS1 because of major migrations that needed to happen in parallel

Linear Algebra

- We knew that CLHEP was slow
- Tested various implementations
  - Specifically on our use case, 5x5 matrixes for tracking
- Eigen was the winner
  - Also has SIMD implementations for particular problems, c.f. doing this by hand
- Actual migration from CLHEP to Eigen was extremely painful
  - Direct use of CLHEP interfaces
  - Inconsistent interfaces (indexed from 0 or 1)
- But this time we wrapped the implementation to ease any future migrations



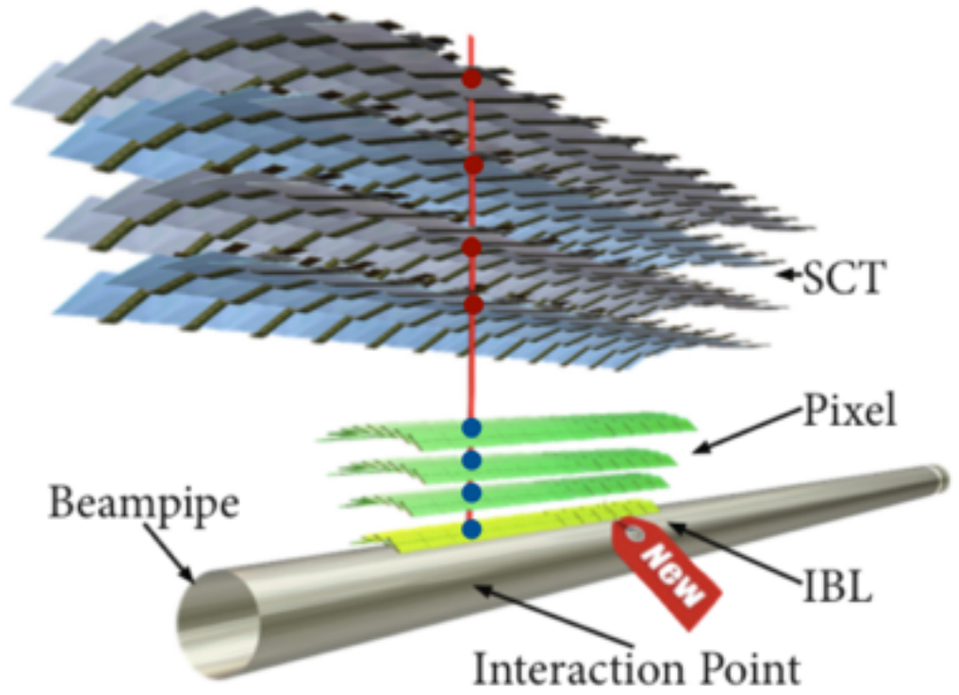
Achieved speed-up w.r.t. CLHEP in 5x5 matrix multiplication testbed

```
__m256d C_012  = _mm256_set1_pd(0.33333333);
__m256d A_012  = _mm256_load_pd(A_V);
__m256d S3_012 = _mm256_load_pd(S3_V);
__m256d V0_012 = _mm256_load_pd(V0);
__m256d V3_012 = _mm256_load_pd(V3);
__m256d V4_012 = _mm256_load_pd(V4);
__m256d V6_012 = _mm256_load_pd(V6);

__m256d H0_012 = _mm256_load_pd(H0);
__m256d H0_201 = CROSS_SHUFFLE_201(H0_012);
__m256d H0_120 = CROSS_SHUFFLE_120(H0_012);
```

Optimised Tracking

- Run1 tracking was quite conservative in its setup
  - Don't miss anything, be careful while we understand the detector
- Re-evaluating how we create triplet seeds showed that seeding from strip detector produced a much purer candidates
- Purer seeds gives much better suppression of combinatorics with pile-up
- Plus… make major simplifications to tracking interfaces
  - Reduce lines of code by 97% (!)



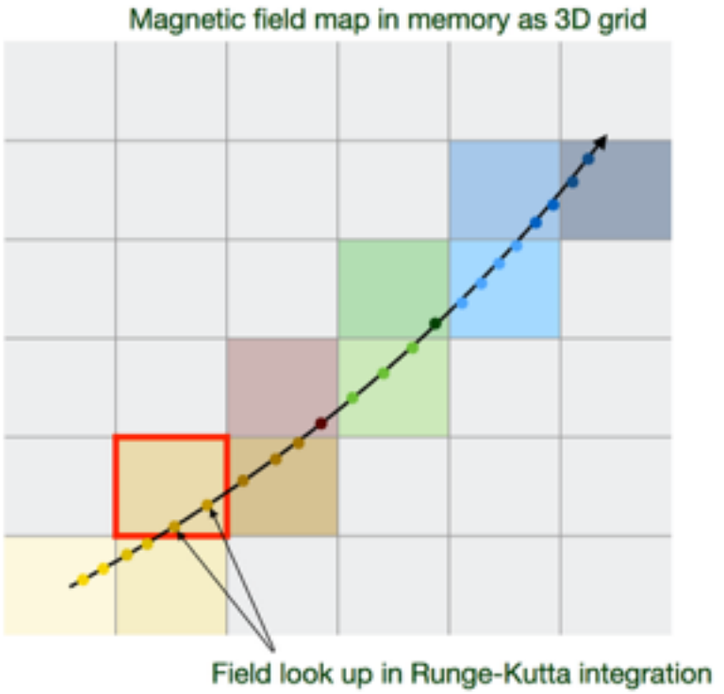
triple seeds can we built as:
- pixel space points only (PPP | ●●●)
- strip space points only (SSS | ●●●)
- a combination of both, e. g. (PSS | ●●●)

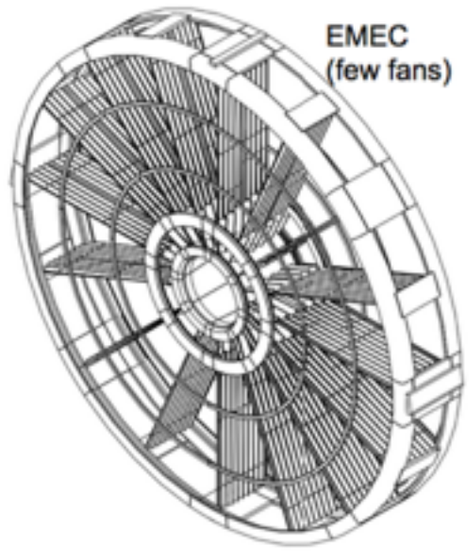
<μ>	PPP+I	PPS+I	PSS+I	SSS+I
0	79%	53%	52%	86%
40	39%	8%	16%	70%

Percent of seeds finally associated with good tracks

Magnetic Field and Simulation

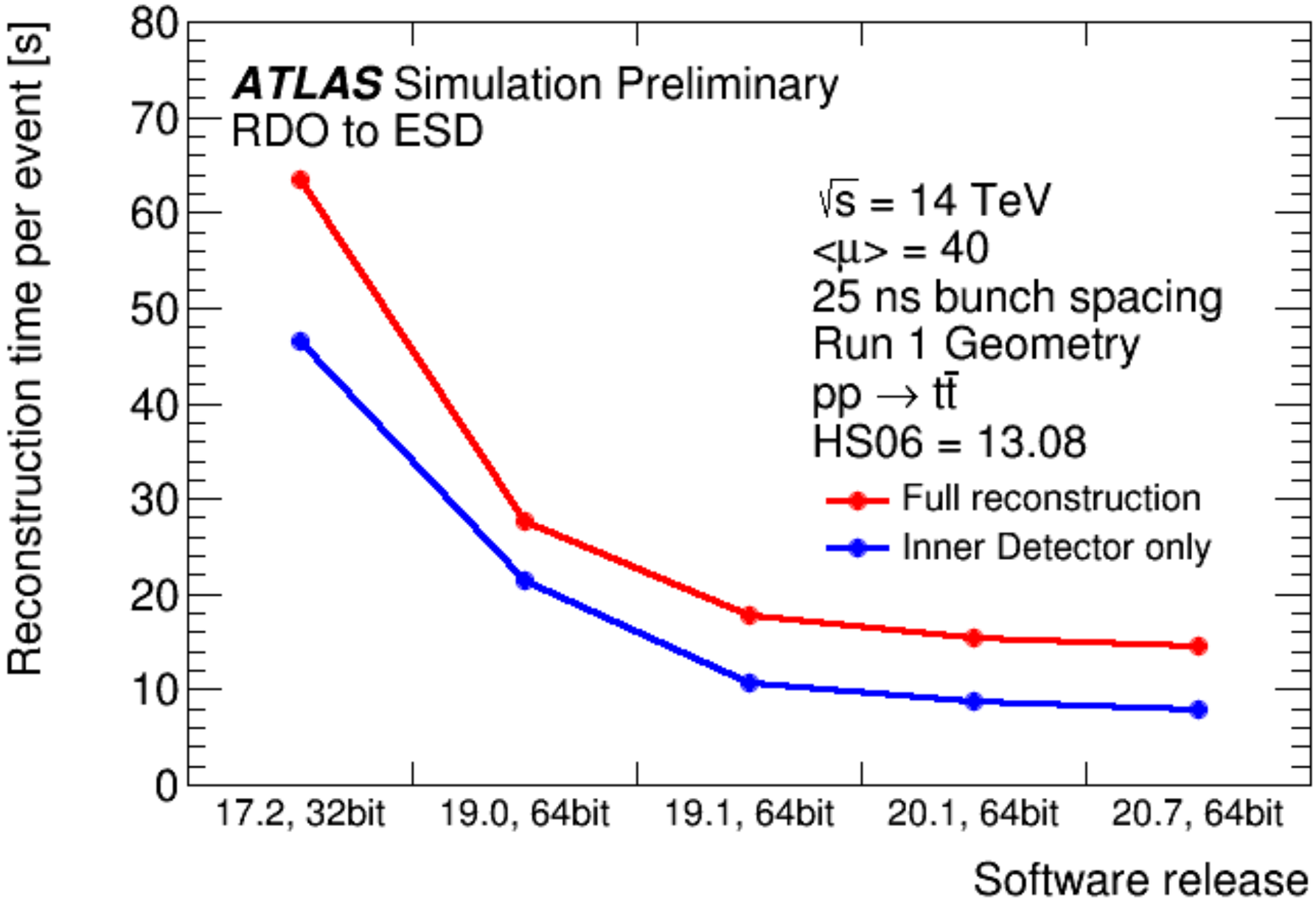
- Magnetic field access is a big CPU consumer in ATLAS
  - numerical (Runge-Kutta) field integration used heavily — especially in simulation
  - ATLAS adaptive Runge-Kutta propagator has been highly optimised
- Run1 field access was not yet optimised
  - deep caller chain, field data needed conversion, was written in FORTRAN90
- New field service implemented
  - simplified caller chain, use native units, use cell caching to store value of field
- Minimised cache misses
  - speed-up of 20% in simulation, few % in reconstruction



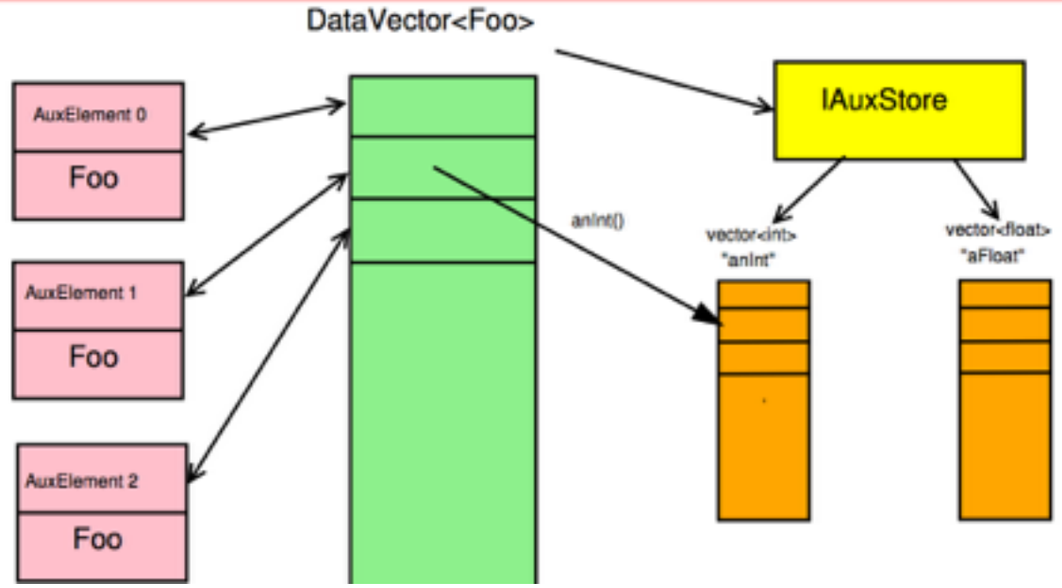


EM endcap simulation vastly improved — used to be 25% of simulation

All Together…



xAOD EDM



- A major problem in Run1 was the inability of physics groups to work with the reconstruction EDM for analysis
  - Slow jobs ran over all AOD output data to write out flat NTUPs for each analysis
- We needed a new EDM that would be
  - Faster, Accessible directly from ROOT, Allow data to be slimmed, thinned and augmented
- xAOD developed to do this
  - Small static core, with most data stored in auxiliary containers that can be modified when data is processed
  - Kept internal storage simple, easy to read back

Performance and Quality

- In general poor quality software does not perform well
  - Code that you can't understand you won't be able to improve
  - This only gets worse over time as authors leave and other people are asked to patch up problems
- So we need to pay attention to quality as well
- It is beneficial to expose the code to different compilers; e.g. clang builds give additional info:
- Mismatched struct/class; Unused private class members; Mismatched header guard.
- Developing gcc plugin already detects naming violations and many threading gotchas

Code howlers:

```
std::abs(eta < 2.5)
if ((bitfield && 0x0011) != 0) ...
!val==10
```

Static Analysis

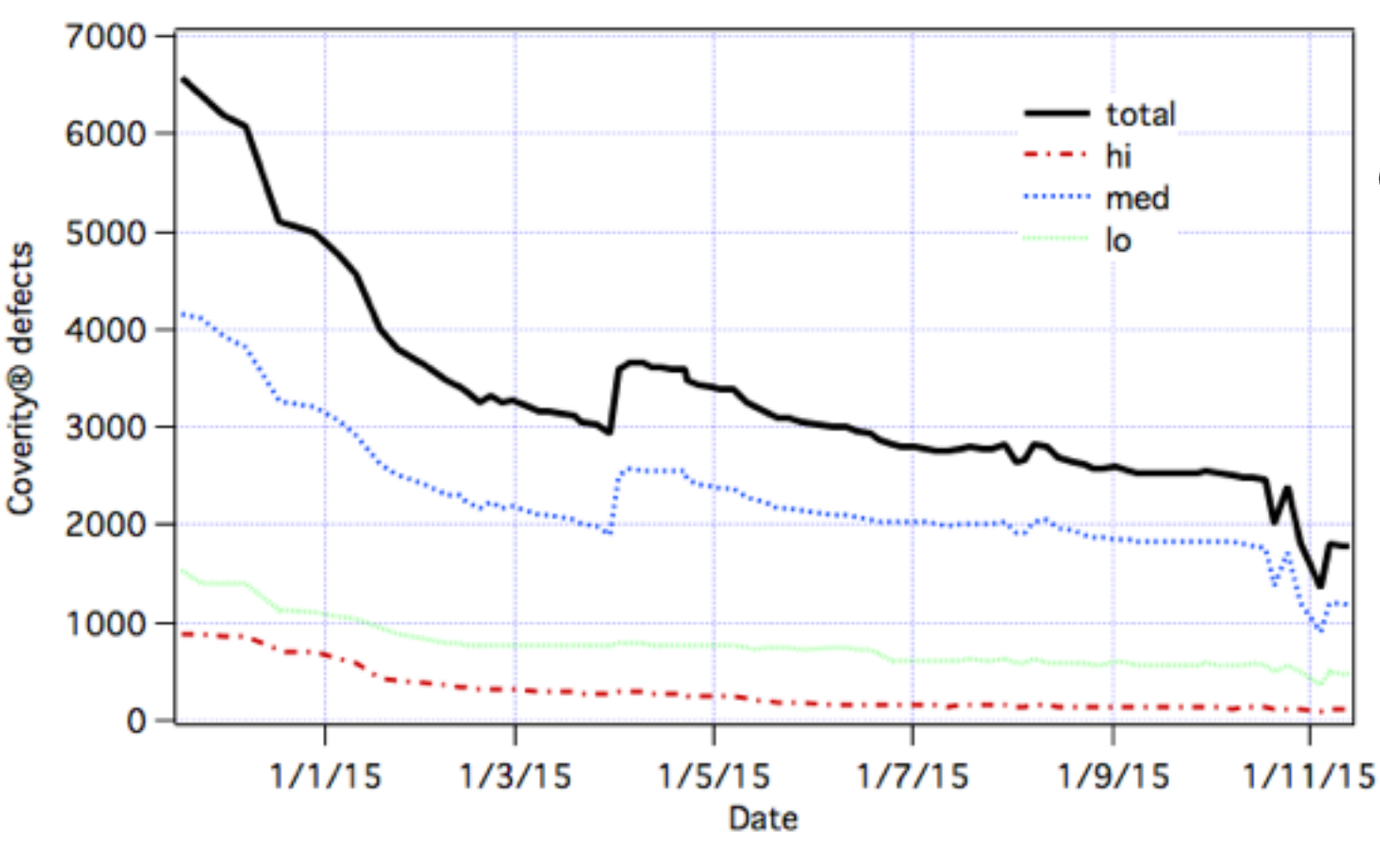
- Attempts to follow your program and points out possible errors; some examples:
  - ‘new’ without delete
  - scalar ‘delete’ on array object
  - array indices out-of-bounds
  - possible cut-and-paste errors
  - suspect indentation

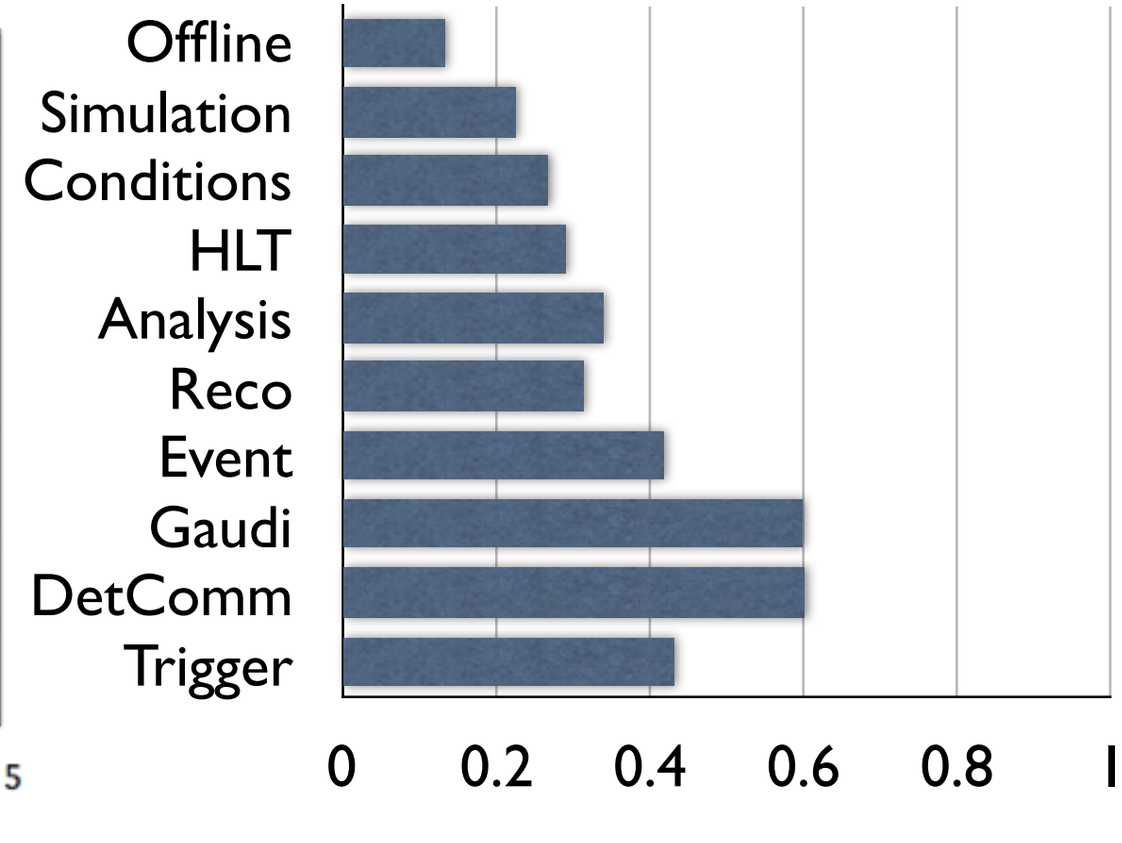
```
if (m_minNUsedHitsdEdx > 0) {
  original: this->m_minNUsedHitsdEdx looks like the original copy.
  ATH_MSG_INFO( "  Minimum used hits for dEdx: " << m_minNUsedHitsdEdx );
  auto dEdx = std::make_shared<MinUsedHitsdEdxCut>(this, m_minNUsedHitsdEdx);
  ATH_CHECK( dEdx->initialize() );
  m_trackCuts["dEdxHits"].push_back(dEdx);
}
if (m_minNOverflowHitsdEdx > 0) {
  CID 28963 (#1 of 1): Copy-paste error (COPY_PASTE_ERROR)
  copy_paste_error: m_minNUsedHitsdEdx in this->m_minNUsedHitsdEdx looks like a copy-paste error.
  Should it say m_minNOverflowHitsdEdx instead?
  ATH_MSG_INFO( "  Minimum IBL overflow hits for dEdx: " << m_minNUsedHitsdEdx );
```

- Examples are drawn from Coverity®, one of the most widely used static analysers

```
   parent: This 'if' statement is the parent, indented to column 3.
64 if (m_fullMaterial)
   nephew: This statement is nested within its parent, indented to column 8.
65    sl << "    - fullMaterial          : " << *m_fullMaterial << std::endl;
   CID 11566 (#1 of 1): Nesting level does not match indentation (NESTING_INDENT_MISMATCH)
   uncle: This statement is indented to column 8, as if it were nested within the preceding parent statement, but it is not.
66    sl << "    - split factor          : " << m_splitFactor << std::endl;
67 return sl;
```

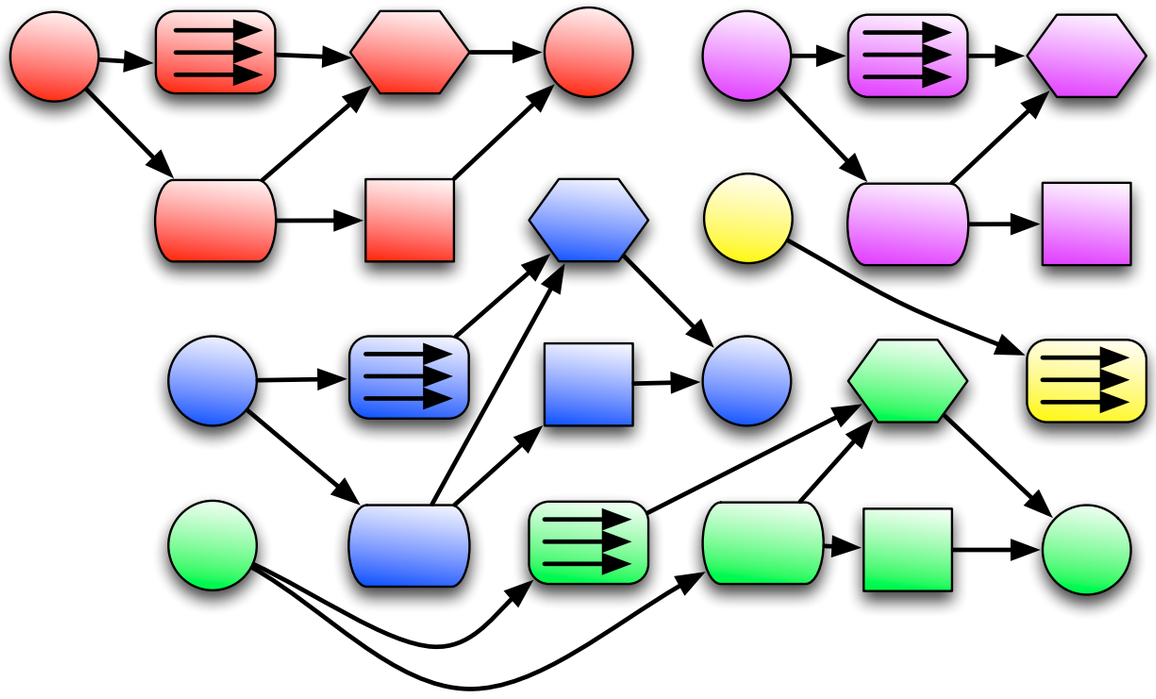
Coverity® progress





- “Coverity’s analysis found an average defect density of .69 for open source software projects that leverage the Coverity Scan service, and an average defect density of .68 for proprietary code developed by Coverity enterprise customers. Both have better quality as compared to the accepted industry standard defect density for good quality software of 1.0 [defects/kloc].”

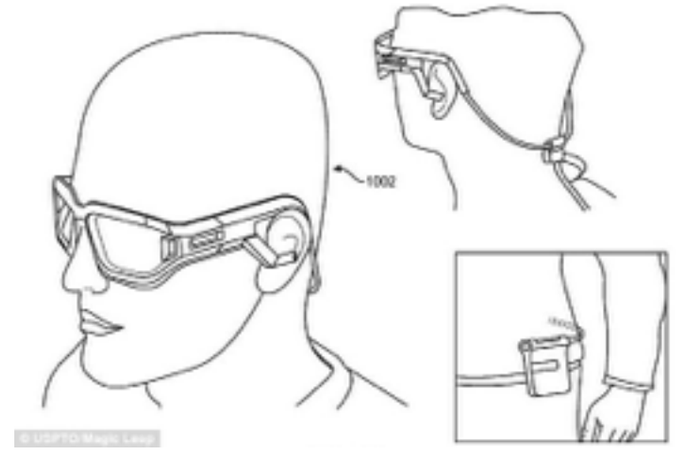
Run3: Framework Evolution



Run3 multi-threaded reconstruction cartoon: Colours represent different events, shapes different algorithms; all one process running multiple threads

- Many ideas and concepts stay the same
  - Mature model of event processing already
  - Evolve towards concurrency
  - Support HLT usecases
- Evolution of Gaudi framework
  - Collaboration with LHCb, SFT and FCC
  - AthenaMT

Offline Code Reviews



- To try and understand where we are with the algorithmic code we are undertaking a software review
  - A high level review of subsystem code
    - What's the design…?
    - Obstacles to threading?
    - Opportunities for parallelism?
  - Much benefit in asking sub-systems to prepare this material — oblige people to put on their 'design goggles'
    - Make them aware of challenges of the new framework
  - Opportunity for reviewers to learn from a different area of the software
- We are now about 25% done, but very happy with the results

Summary

- Measure what you want to improve
- Make it really easy for developers to see the impact of their changes
  - Little tools can make a big difference, e.g., MemoryMonitor
- Strong teams for core software and subdomains can determine the right strategic goals
  - Though major changes can be a major pain
  - Especially when paying back a software debt
- On the road to Run3 and AthenaMT we, again, have major challenges to face
  - Retooling is underway
  - Software review helps us understand challenges
  - Testing is an area we have to invest in (not always trivial in a complex framework — something we're working on)