ATLAS Offline Software — Plans for 2016 and beyond

Graeme Stewart and Walter Lampl

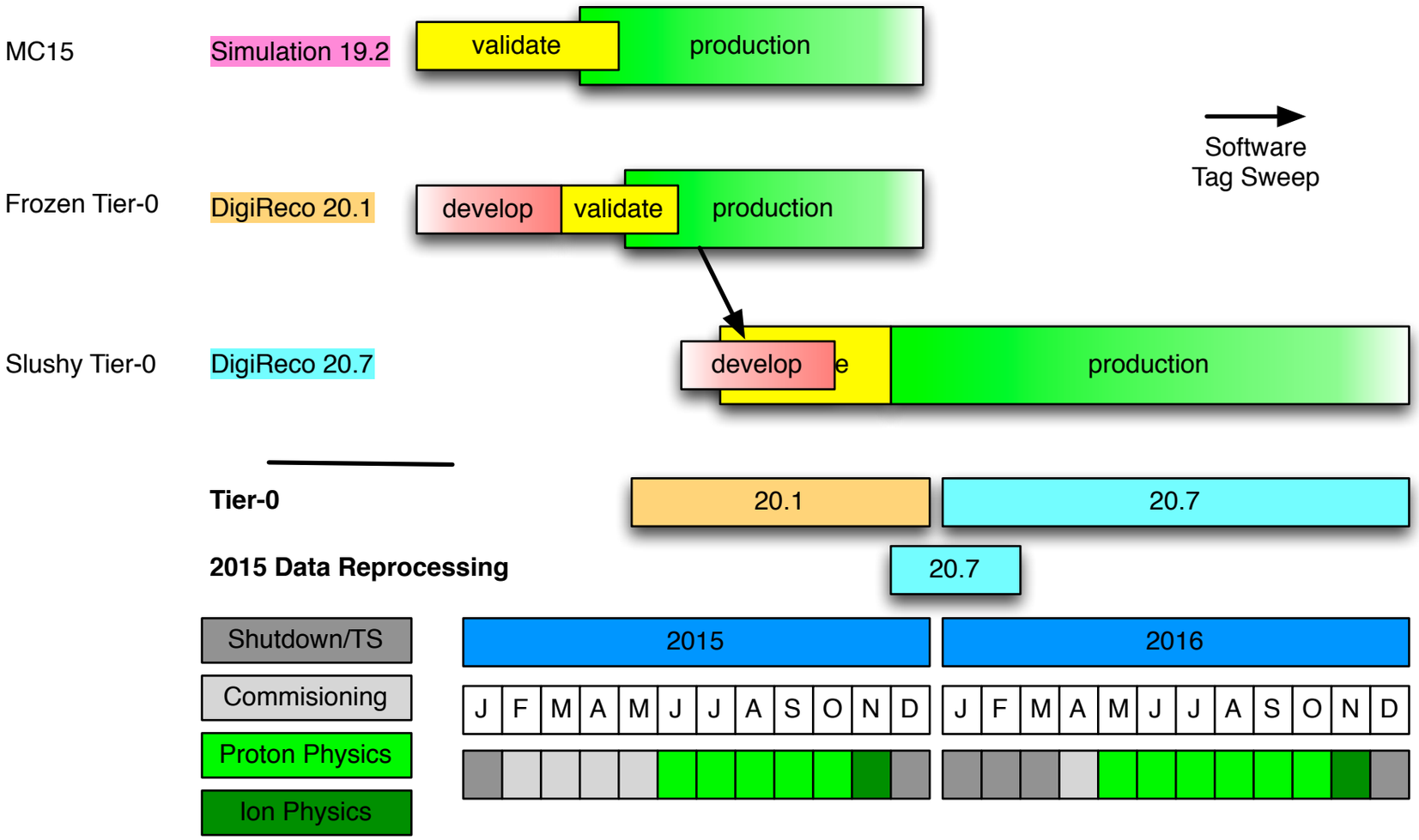
University of Glasgow | School of Physics & Astronomy

2016-01-27

Outline

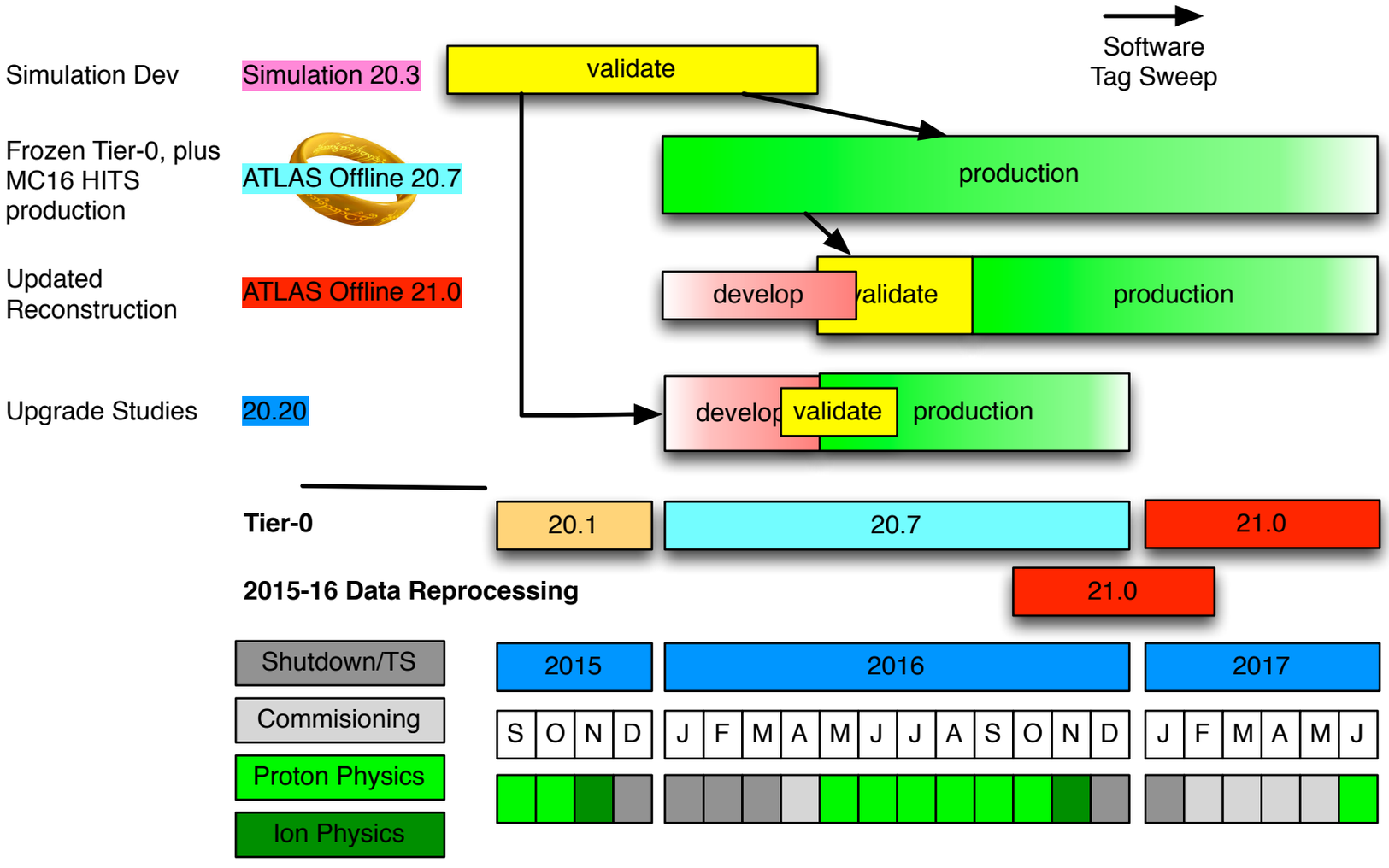
- Release plans for 2016
  - Tier-0, MC16 and Release 21
  - Infrastructure Issues
  - CentOS7
- Porting to other architectures
  - Aarch64
- Memory and resources
  - AthenaMP
  - RAW to ALL
- The future
  - AthenaMT

Release Review — 2015



- Simulation used release 19.2 with Geant4 9.6
- Reconstruction (and digitisation) ran in release 20.1
  - Many LS1 improvements
- Release 20.7, with improved reconstruction, was prepared in the autumn
  - Used for the reprocessing of the 2015 data over the Christmas break
  - Technical migration to ROOT6

Release Plans — 2016



- Simulation developments now sweep into 20.7
  - Should will use Geant4 10.1
  - Basis for our new MC16 campaign
- 20.7 becomes ‘one release to rule them all’
  - Simplified release management
- 21.0 updates reconstruction
  - Also contains MC16 simulation tags
- 20.20 used for special upgrade studies

Analysis and Derivation Releases

- Releases to make derivations (DAODs) go more or less in parallel with the primary reconstruction release
  - Same release numbering, but *AtlasDerivation* release instead of AtlasProduction
- We make lightweight athena analysis releases for each reconstruction release as well
- These live for as long as analyses using reconstructed data are alive for, which is quite a bit longer than taking/simulating data with that release

Reconstruction Series	AthAnalysisBase Series
20.1	2.3
20.7	2.4
21.0	3.0

Software Infrastructure

- ATLAS is modernising its software infrastructure
  - Currently migrating from home grown CMT build system to standard CMake
    - Impressive improvements in build speed
      - Will be used for Release 21
  - Investigation of git and Jenkins are on going
    - git for source control, plus gitlab/github for collaboration
    - Jenkins for continuous build and integration system
- None of these migrations are completely straightforward (4M lines of C++, 1.4M python, 2500 packages and 15 years of history), but they are a worthwhile investment

Software Quality

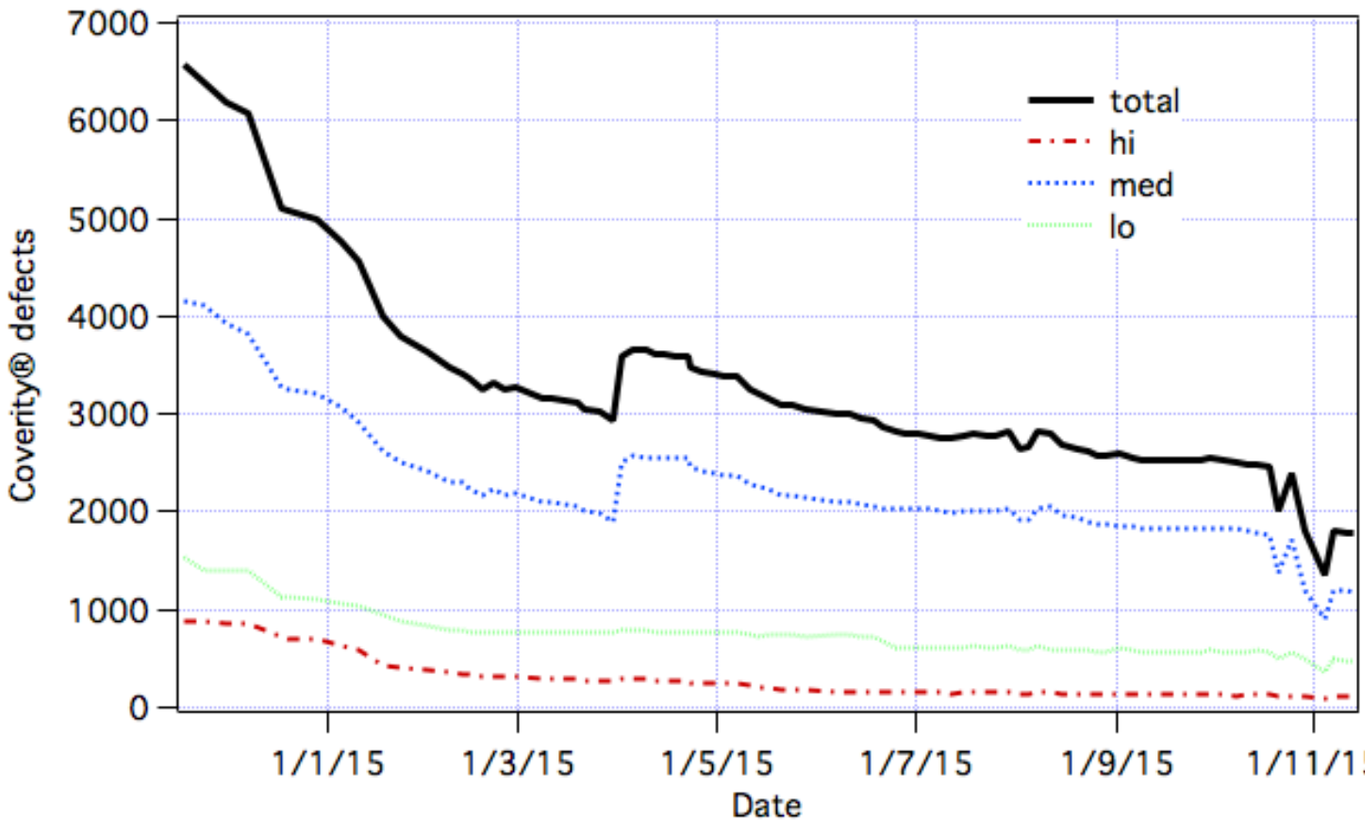
- More that a year ago we started to seriously use new software quality tools
  - Even using different compilers helps, e.g., clang finds
    - `std::abs(eta < 2.5)`
    - `if ((bitfield && 0x0011) != 0) ...`
    - `!val==10`

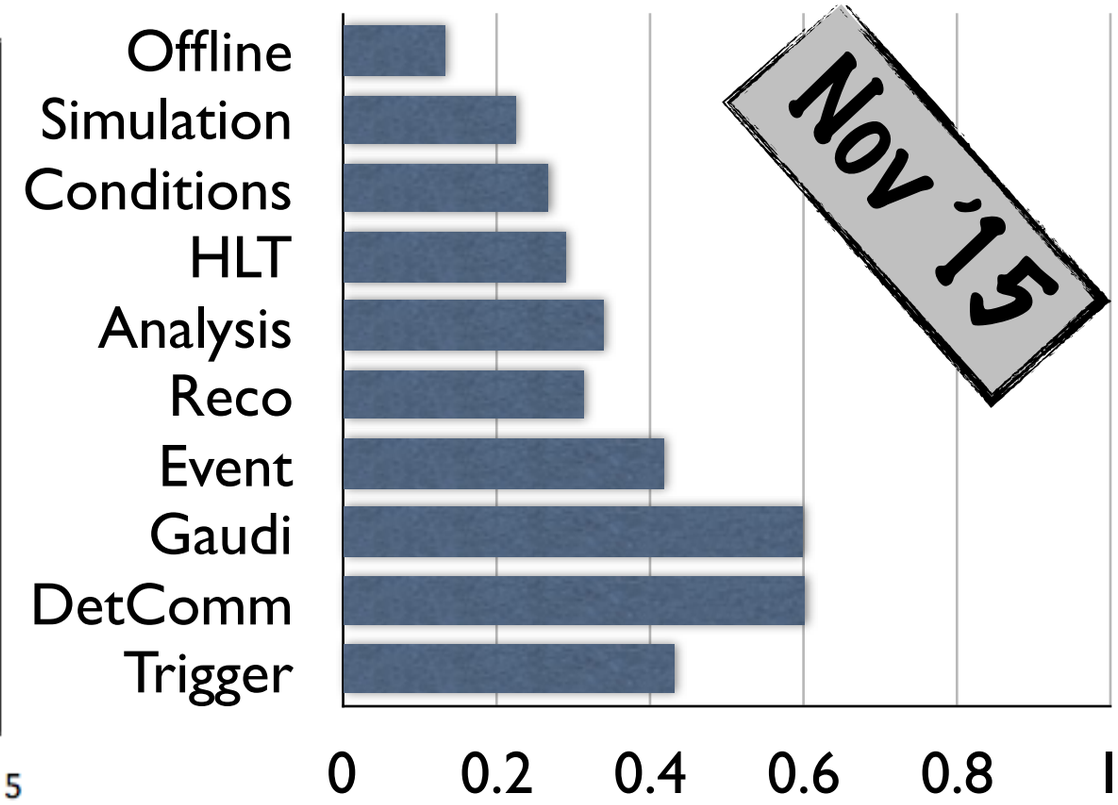
```
if (m_minNUsedHitsdEdx > 0) {
  original: this->m_minNUsedHitsdEdx looks like the original copy.
  ATH_MSG_INFO( "  Minimum used hits for dEdx: " << m_minNUsedHitsdEdx );
  auto dEdx = std::make_shared<MinUsedHitsdEdxCut>(this, m_minNUsedHitsdEdx);
  ATH_CHECK( dEdx->initialize() );
  m_trackCuts["dEdxHits"].push_back(dEdx);
}
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 );
```

- A lot of progress using Coverity, commercial C++ static analysis tool which can find many problems, including
  - ‘new’ without delete
  - scalar ‘delete’ on array object
  - array indices out-of-bounds
  - possible cut-and-paste errors
  - suspect indentation

```
   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]."
- ATLAS offline now ~0.4 defects/kloc
  - 20.7 reprocessing had no software crashes running over physics_main stream

SLC6 and CentOS7

- SLC6 is rather old now
- Our initial CentOS7 investigations were quite positive
  - Not very hard to get SLC6 releases running (usual HEP_OSlibs metapackage)
  - CentOS7 native build does not look like it presents any real problems
- Currently on hold pending migration of development releases to CMake
  - Should have working release 21 builds *native* with CentOS7 this summer
  - Will need to be validated
- N.B. Migration of analysis to CentOS7 is slightly more involved
  - Analysis code usually requires to be built
  - Need to validate the build environment
    - At least early tests of this were also fairly successful, but be aware this is an extra step…

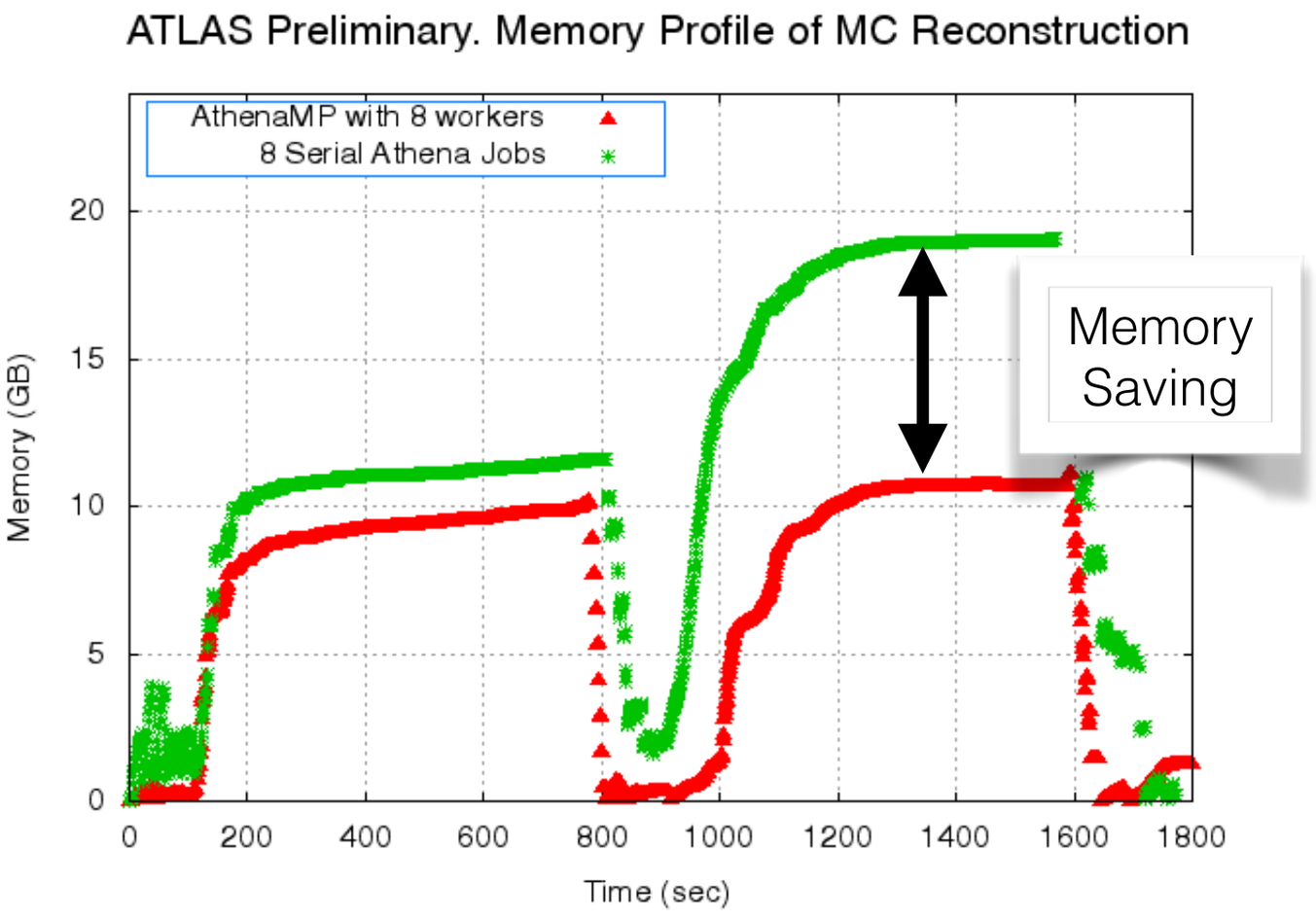
Other CPU Platforms

- We are investigating ports of Athena to non-x86_64 platforms
- Most progress on Aarch64 (a.k.a. ARM64)
  - Managed to get some basic athena jobs running last year
  - Now concentrating on stand alone ATLAS simulation release
    - Easier to port and validate
  - We hope to have this validated this year
- Also interested in PowerPC (main motivation Argonne Mira and Oak Ridge Summit machines)
- Investment here subject to significant effort constraints on our side — will be proportional to expected gains

Accelerated Platforms

- ATLAS Simulation also being prepared for Xeon Phi
  - Intel many-core architecture
  - Based on x86_64 low power atom cores
    - Thus in principle trivial port
    - However, memory per core is challenging
      - Unlikely that AthenaMP will scale here
    - Memory access patterns are extremely unlikely to be good for ATLAS software (especially reconstruction)
    - We essentially don’t use vector registers at all
- GPGPUs are currently just too difficult to use for any general ATLAS workload

Run2 Resources: AthenaMP



- Serial Athena processing for digitisation and reconstruction in 64bit a long way over 2GB/core
- Our strategy for Run2 is to use multi-processing with *copy on write* (Athena MultiProcess, or AthenaMP)
  - Child processes do not need additional copies of shared memory pages
- Memory savings are significant

Reminder: Memory Measurements

- Traditional measures of memory don't work well
  - RSS does not properly account for sharing of pages between processes
  - VSS (mapped memory) bears little resemblance to memory that's actually written to
- Please use PSS (proportional set size) if you need to keep an eye on how much physical memory a multi-core ATLAS job is using
- cgroups works really well for preventing problems
  - See talk from Gang Qin and Dave Britton at last week's ACAT meeting

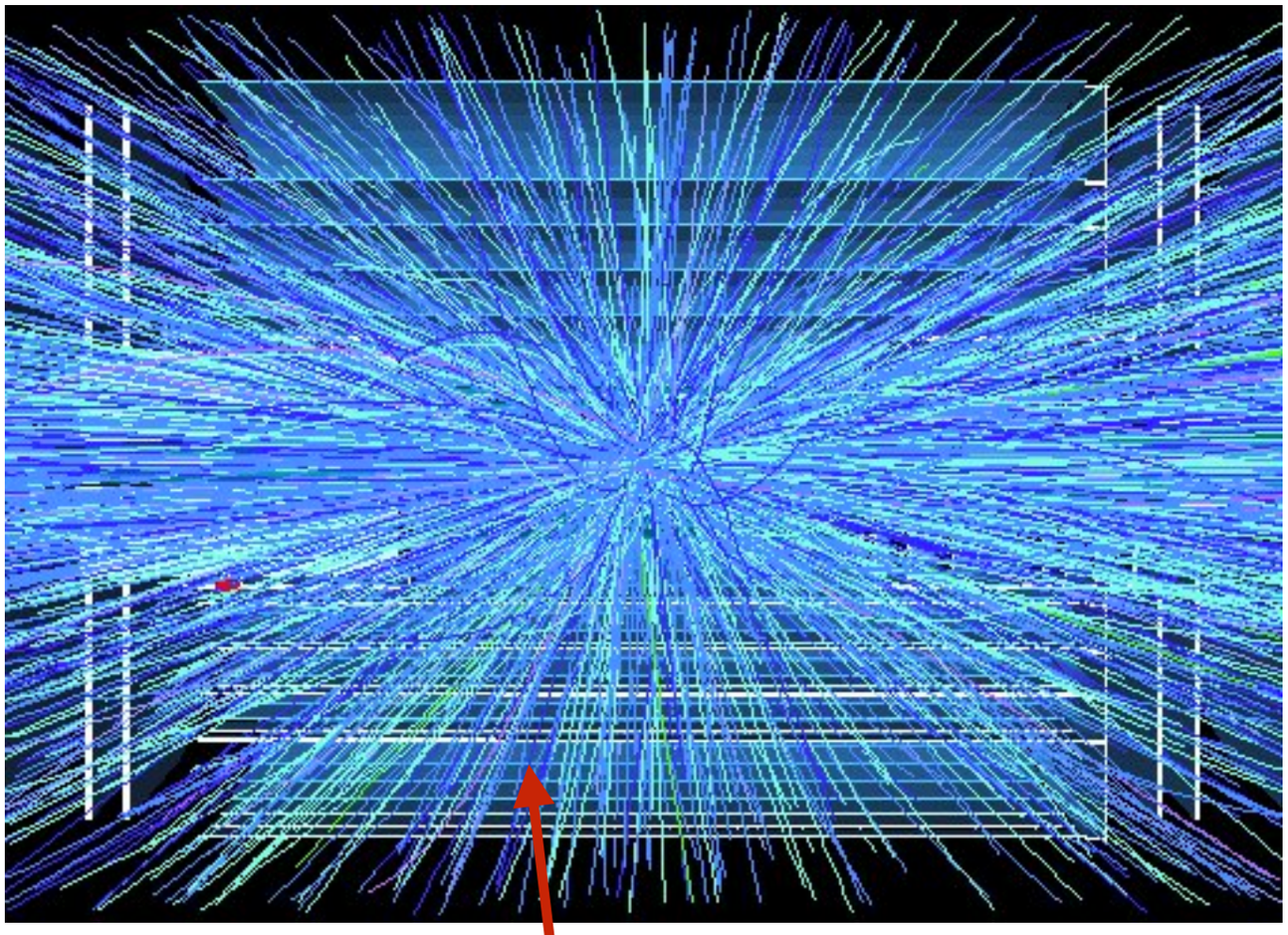
Serial Overheads

- One pressing issue with MP workflows is serial overheads
  - Merging outputs from multiple MP workers down to a single file affects overall job efficiency
  - Running multiple athena substeps duplicates initialisation and finalisation and file based communication increases i/o load
- We are developing/validating a new reconstruction workflow that
  - Produces AOD directly from RAW/RDO
  - Additionally can write DRAW and DESDs directly in this Athena step
- Reduces i/o load considerably — no need to serialise/deserialise ESD
- Size jobs to avoid multi-core merging if possible
- Will *increase* memory — we are tuning and optimising to keep this part under control
  - Use of cgroups for proper job resource control will be even more important!

Step	Setup	RAWtoESD*	RAWtoESD validation	ESDtoAOD setup	ESDtoAOD*	ESDtoAOD validation	DQHistogram Merge setup	DQHistogram Merge	DQHistogram Merge valid.
Wall time (*MP)	6m 26s	2h 47m 26s	6m 29s	7m 56s	1h 0m 54s	4m 13s	19s	2m 37s	1s
CPU time, efficiency	N/A	10h 20m 56s 92.7%	N/A	N/A	2h 34m 16s 63.3%	N/A	N/A	29s 18.5%	N/A

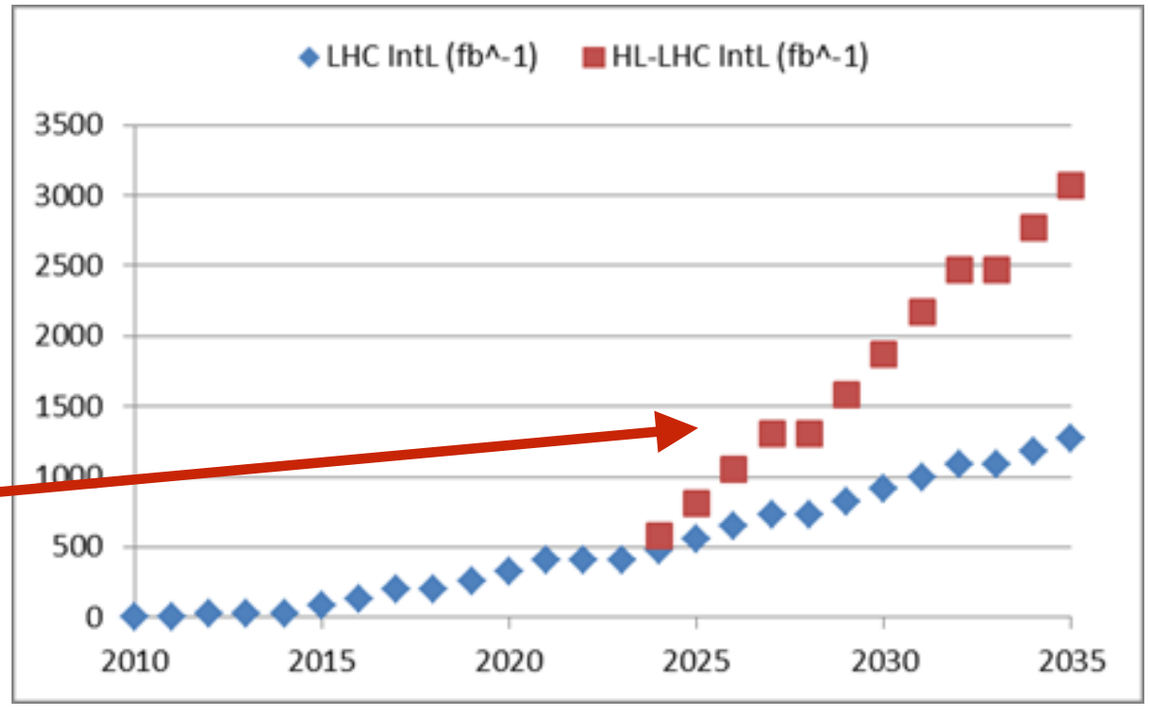
Tier-0 job efficiency in 2015

The Road to High Luminosity LHC



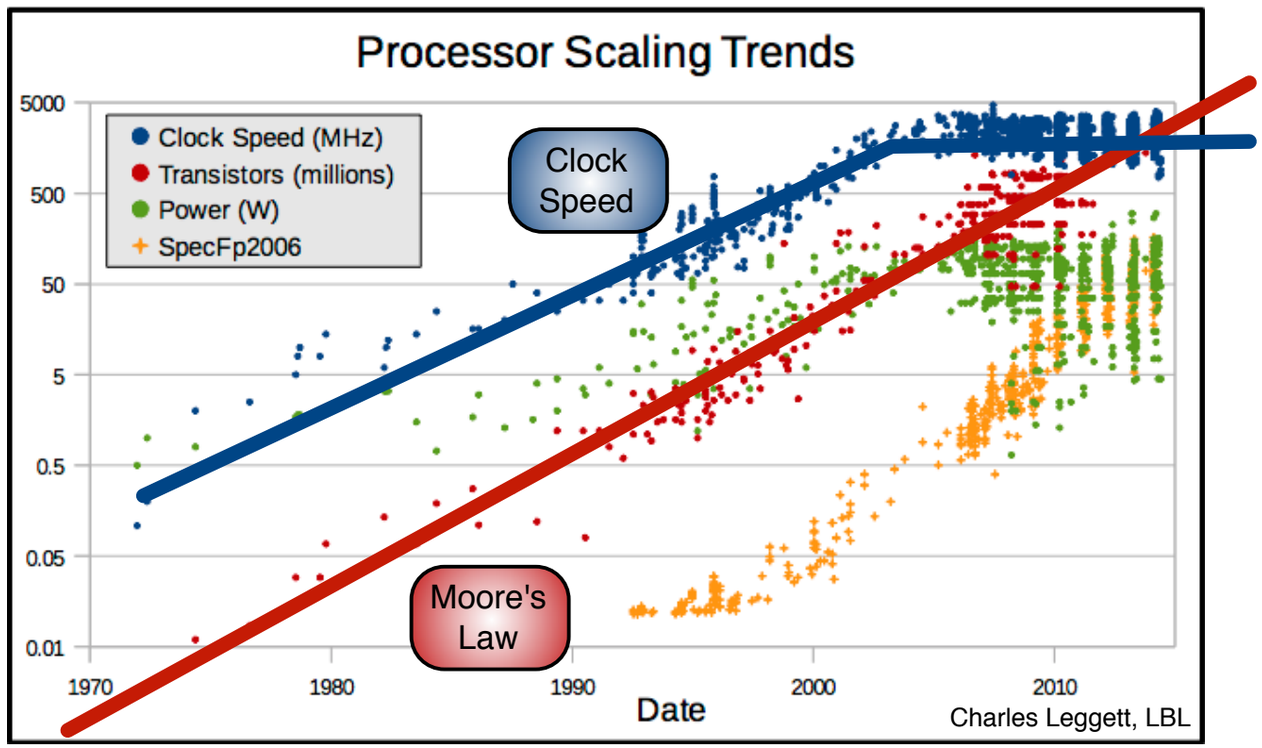
	Integrated Luminosity	ATLAS Pileup
Run1 (2009-2012)	25	35 (@50ns)
Run2 (2015-2018)	125	40
Run3 (2021-2023)	300	60-80
HL-LHC (2026+)	300/yr	140-200

Event Complexity x Rate
= Challenge



Computing Challenges

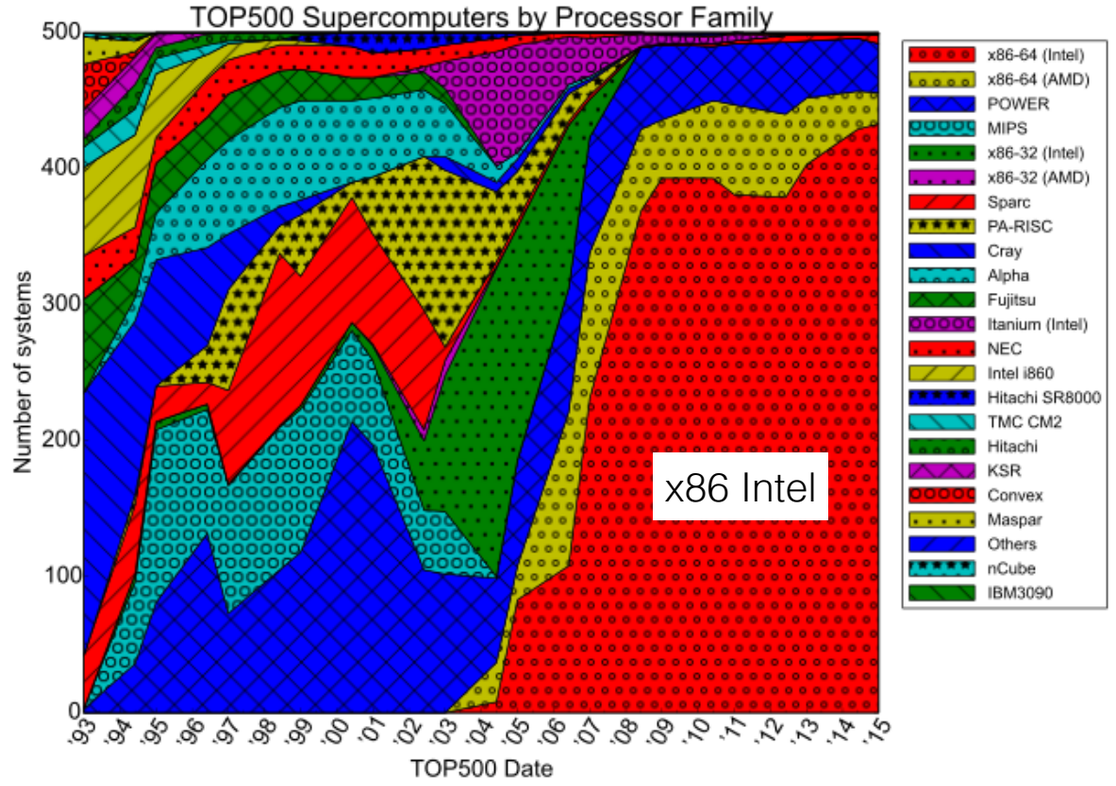
- Great challenge of computing in the next decade will be one of power
  - nJ per instruction
  - Note it is likely that the power costs of memory access would be greater than CPU power in an exascale machine
  - This is driving evolution of larger numbers of cores on dies
    - More transistors but no more clock speed
  - And lower amounts of memory per core



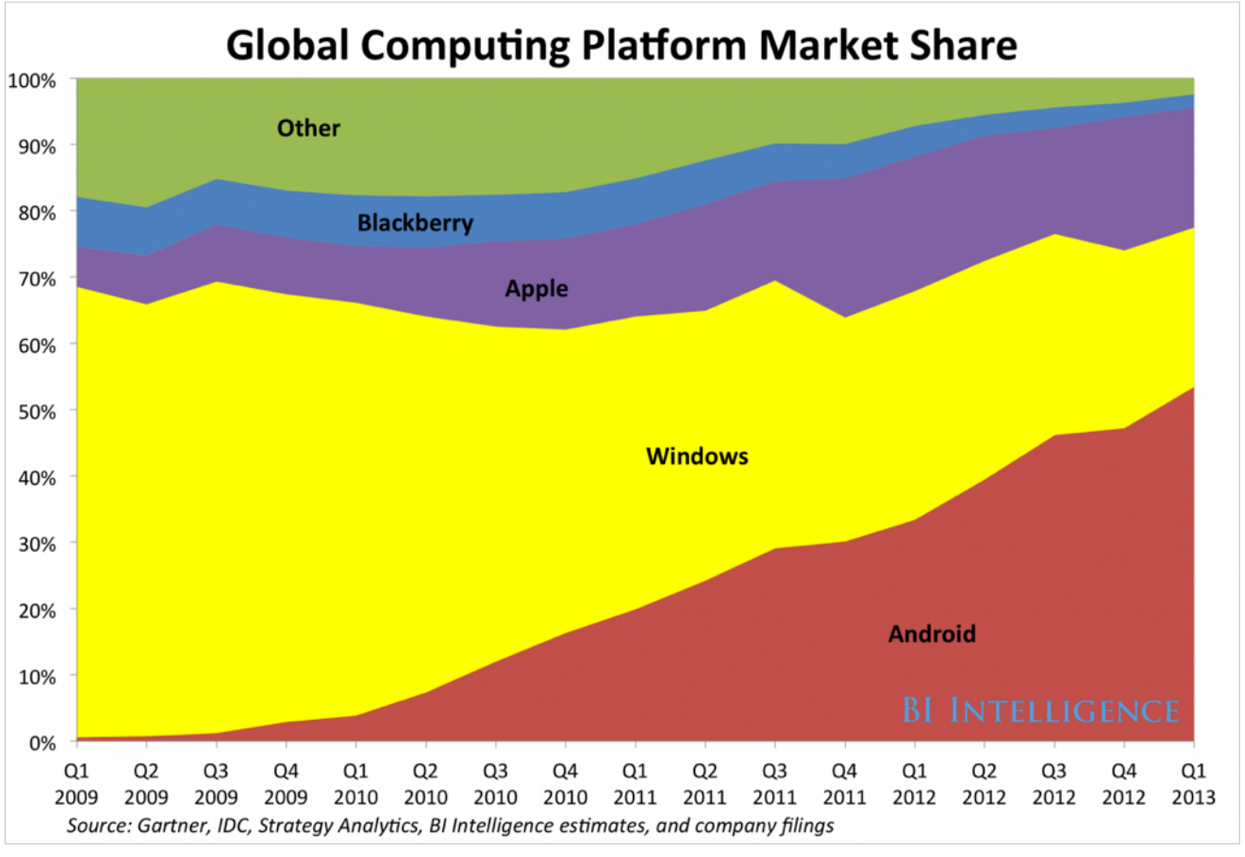
New NERSC CRT building housing Cori machine room at LBNL

End of Homogeneity

- Computing for LHC experiments was developed at a time when the market became extremely homogeneous
    - x86 architecture was dominant, mainly Intel CPUs
- Physical limitations of high performance and high power efficiency CPU computing are forcing a challenge to this homogeneity
    - Different low power CPU architectures: Aarch64
    - Different architectures: GPGPUs, Hybrid CPU + FPGAs
- More and more ‘features’ increasing theoretical performance
    - But not easy to use — especially for legacy HEP code
- Dark silicon might start to dominate in the future — specialist computing units lit up only when needed

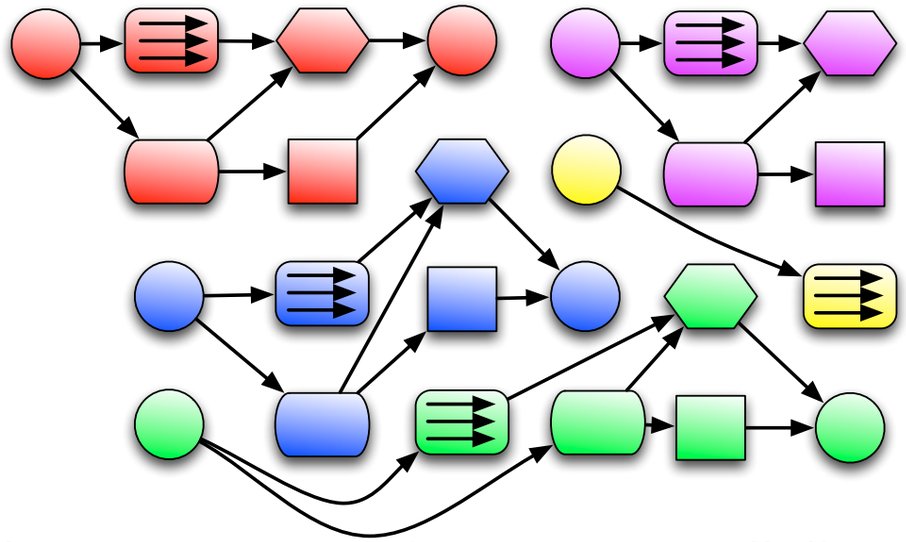


Processor families in TOP500 supercomputers (Moxfyre)

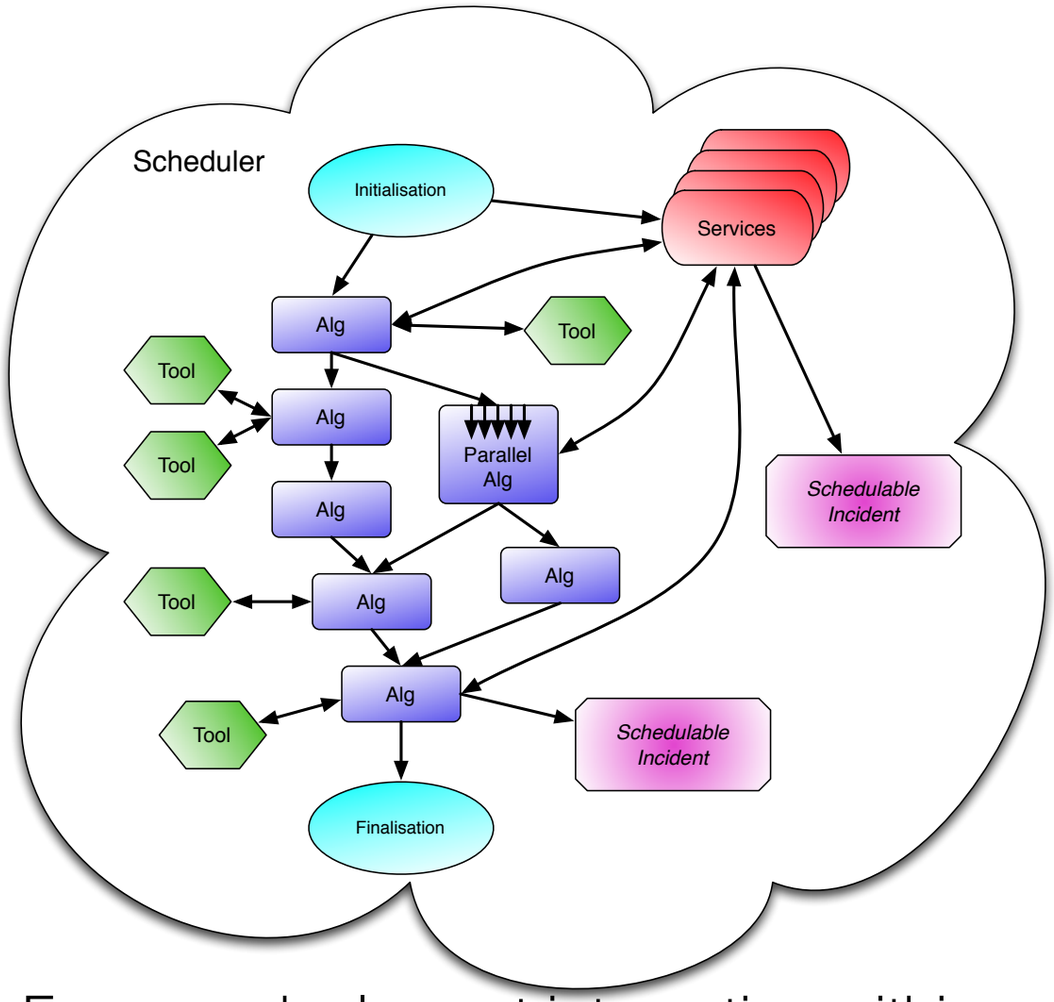


Future Framework Requirements: Key Points

- Support high level trigger use cases natively
  - Event Views to transparently support partial event processing
- Clearly separate code that sees one event at a time
  - Algorithm and Tool
- From code that sees all events at once
  - Services
- All inter-algorithm communication goes via the event store
- Try to limit the use of asynchronous incidents
- Exploit:
  - Multiple event parallelism
  - Inter-event parallelism at the algorithm level
  - Allow for in-algorithm parallelism — very likely necessary for high pileup tracking
- Use an underlying generic threading toolkit, layering on only what we need



Inter-event and in-event parallelism
(colours are events, shapes algorithms)



Framework element interaction within a single event

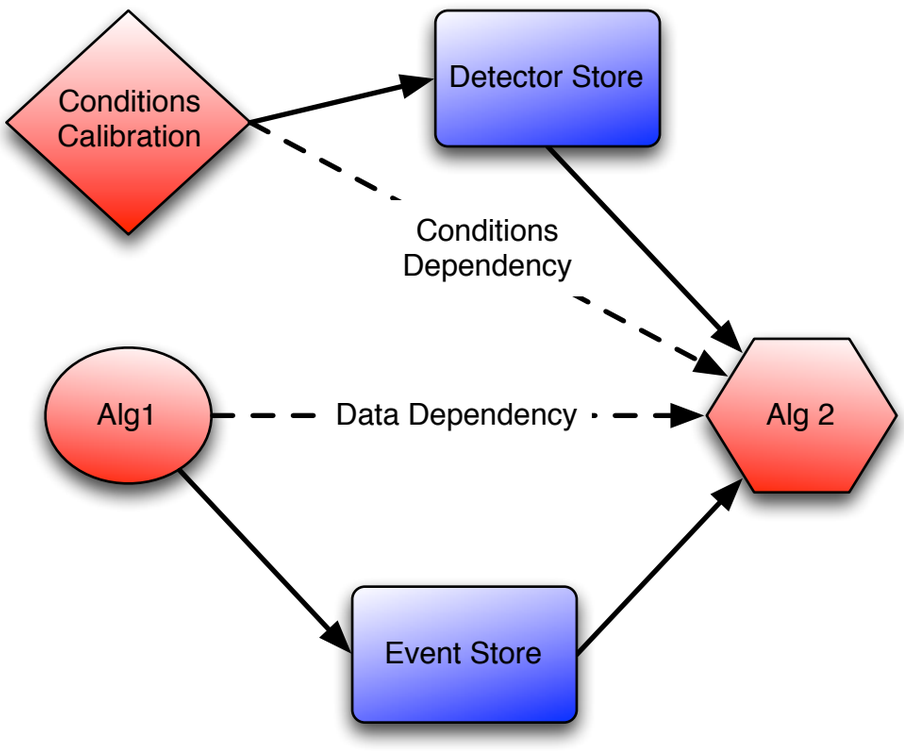
Gaudi & AthenaMT

- After due consideration we decided that an updated version of Gaudi fitted ATLAS's needs
- Collaboration with LHCb and SFT was deemed highly desirable
- GaudiHive demonstrator had shown promising results
  - Underlying Intel Threaded Building Blocks had been shown to perform well
- Similar ATLAS CaloHive example had demonstrated memory savings could be achieved in practice
  - And also given us insight into many of the pitfalls that would be faced along the way
- Reinvigoration of the Gaudi project has been extremely welcome!
- ATLAS specifics incorporated into Athena MultiThreaded, **AthenaMT**

Re-entrant Algorithms and Handles

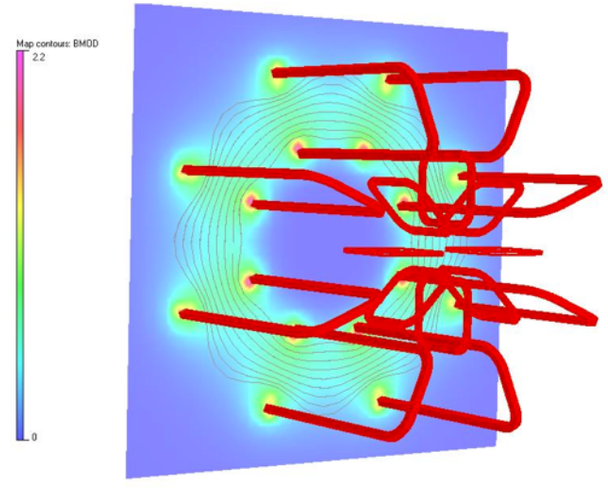
- Early versions of GaudiHive had two algorithm classes
  - Cloneable and non-cloneable
- Cloneable is better than non-cloneable, but consumes additional memory
- Best case is an algorithm with re-entrant execute method
  - **`const execute()`** able to run multiple events simultaneously, without cloning
- However, first implementation of handles did not play nicely with this
  - Private data member used ‘magic’ to cache the event specific handle information
  - Would need to vary event by event → not const anymore :-(
- New design requires an extra call at the top of execute
  - Resolve the handle property to the event specific proxy (stored on the stack)
  - Exact implementation is being discussed, but will be prototyped soon

Conditions as Data



- Instead of conditions being retrieved ‘on demand’ (as now), inform the scheduler they are a data input for an algorithm
  - Prevents stalling while conditions are retrieved
- Instead of callbacks and private caching, write conditions algorithms that perform the calibration and use the event store for calibrated values
  - Use detector store because conditions cadence is very different from event cadence
- Calibration algorithm (plus tools) first check if current values are still valid
  - If not, new retrieval is triggered
- Underlying Athena service is in charge of actual DB interactions to bulk requests for more efficient interactions
- Time varying data, with a different cadence to events is a common problem
  - Scope for a common project with Gaudi partners

Service Migration: Magnetic Field



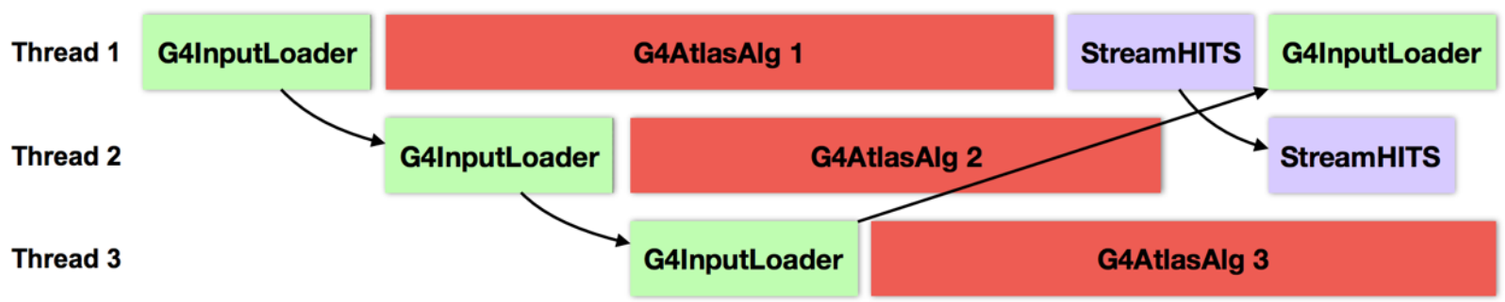
- Good example of efficient, but serial, code is the ATLAS magnetic field service
  - Field is computed via a Biot-Savart component from magnet currents in the detector
  - Plus an interpolated correction, based on position: $B_{cor}(x,y,z)$
    - Looked up in a 300MB field map, which is expensive (and used extensively in simulation)
  - Cache field values used in an interpolated volume
    - Very good chance of a hit when following a G4 particle
- However, when multiple particles are in flight lookup order becomes randomised
  - Lose all benefits of the cache

Thread Safe State

- New implementation of the field service uses *thread local storage* to manage the cache
  - First call on a thread allocates store
  - As Intel TBB keeps each task element on its own thread, following a particle per thread keeps the cache benefits
  - No client side changes
- However, thread local storage is not perfect
  - We are also planning to introduce client side caching
    - Client can pass in a non-const cache object using a different interface
  - Best performance and flexibility when in performance critical parts of the code

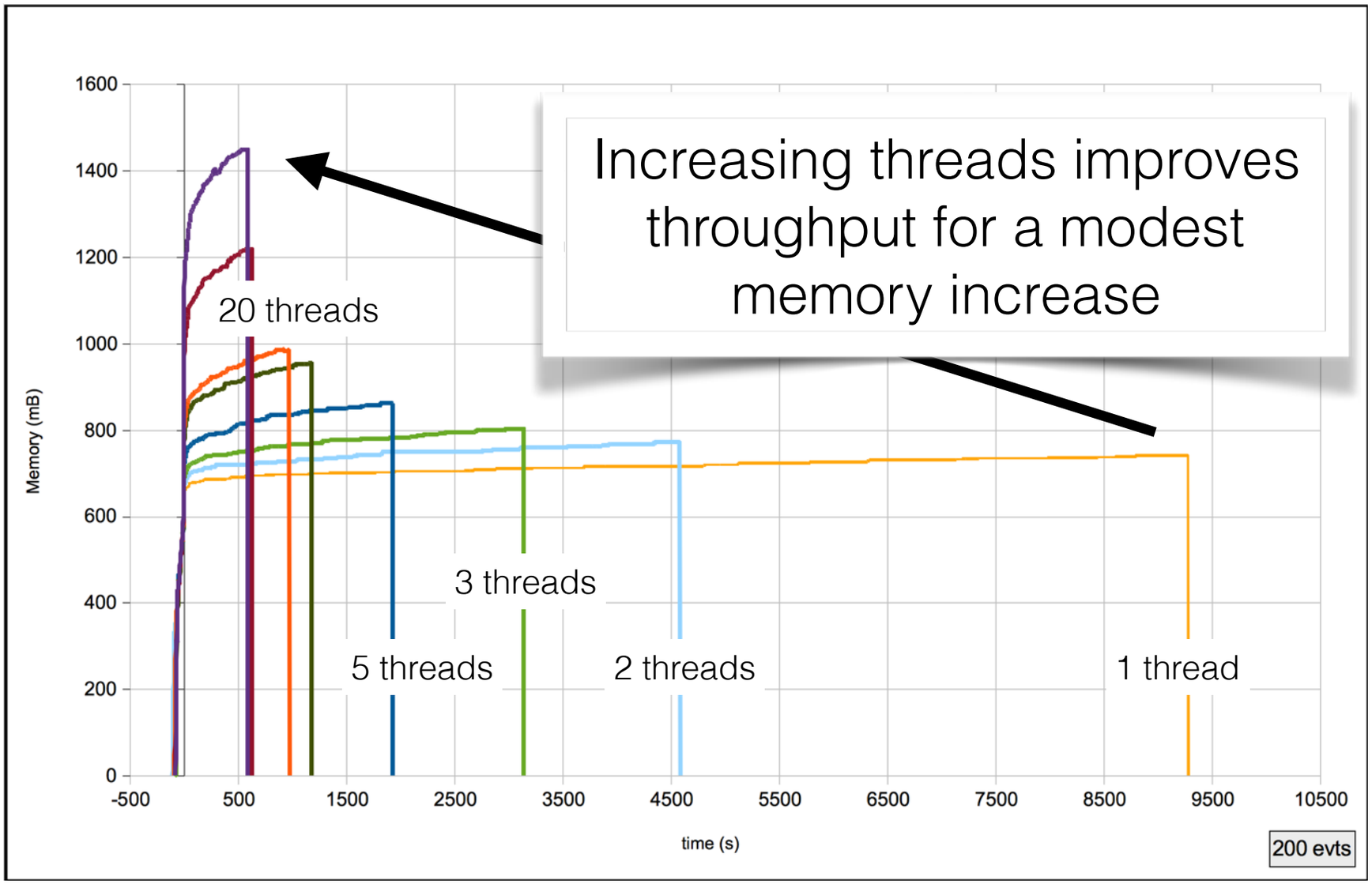
~~G4Hive~~ AthenaG4MT

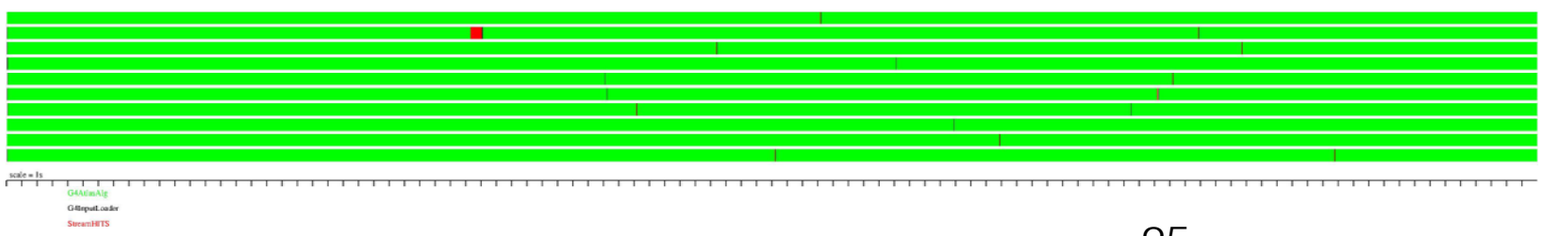
- Attempt to get multiple G4 events running on different threads, controlled by Gaudi scheduler
  - Strong motivation is Phase II Cori machine at NERSC
  - 9300 Knights Landing machines (670 000 cores, >~1GB/core)
- This has been a very instructive exercise
  - Sensitive detector classes needed a new implementation to support on demand creation per thread
  - User actions required considerable refactoring and lots of tedious recoding
  - I/O system turned out to have many assumptions about a serial processing model
- Teaching us about the balancing act between hacked solutions and over elaborate designs — focus on the actual problem!



AthenaG4MT Results

- Clear demonstration of good multi-threading scaling
- Threads all kept ~100% busy
- N.B. No magnetic field in this example





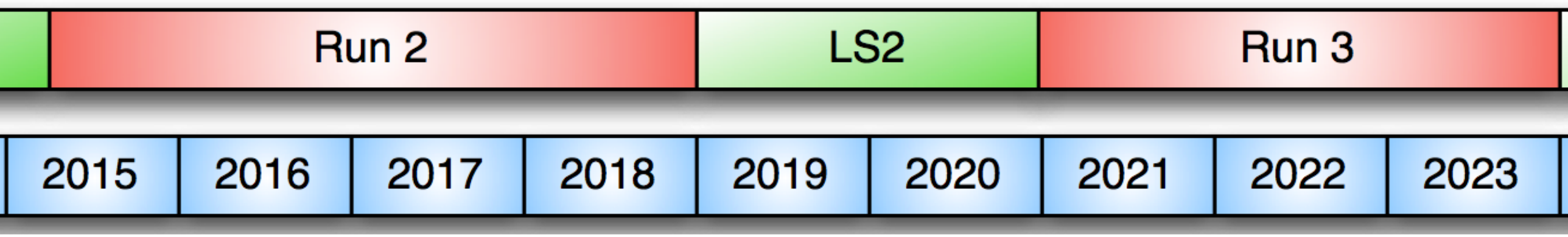
Thread CPU usage measured by Intel VTune

Migration Tools and Strategy

- 4 million lines of C++ is a serious headache to migrate
- Strategy 1: Use machines
  - Making more use of sanitisation checkers
    - ASAN — check for memory errors; UBSAN — check for undefined behaviour; TSAN — thread sanitiser
  - gcc static checker plugin
    - Redundant division checker; Naming guidelines; Thread safety checks
- Strategy 2: Invest in the test
  - We need to invest more in testing our code — looking at gmock as a way to help test framework components
  - Take advantage of code refactoring tools
- Strategy 3:
  - Developer education is vital
    - Good advice
    - Real examples
    - Always keep it simple
  - Make things as simple as possible for the ‘average’ developer — writing a basic Athena algorithm or tool should have few gotchas and most of those should be machine trapped (**const** execute!)

Timeline and Goals

Dates	Framework	Algorithmic Code
2015	Baseline Functionality	Very few algorithms, concentrate on high inherent parallelism; general clean-up
2016	Most functionality available (including views)	Wider set, including CPU expensive algorithms with internal parallelism; continue clean-up/prep; first trigger chains
2017	Performance improvements and final features	Migration starts with select groups
2018	Performance improvements	Start bulk migration
2019	Bug fixes	Finish bulk migration
2020	Bug fixes	Integration



Summary

- Software plans for 2016 are now well known
  - Many infrastructure and quality improvements
  - AthenaMP is the main plank in which production software will run for next years
    - We are optimising to reduce i/o load and serial overheads
- CentOS7 is in the pipeline for this year
- We expect that ARM64 and Intel Phi should be ok for simulation by end of the year (more tentative!)
- Phase I ATLAS Offline Software Upgrade is underway
  - We know what we want to achieve
- Already substantial progress in many areas
  - Effort to work on core framework is identified already
  - Investment in tools and tests will pay off handsomely
  - And we also need to train the development community
  - There will be a lot of code we need to review and rewrite