

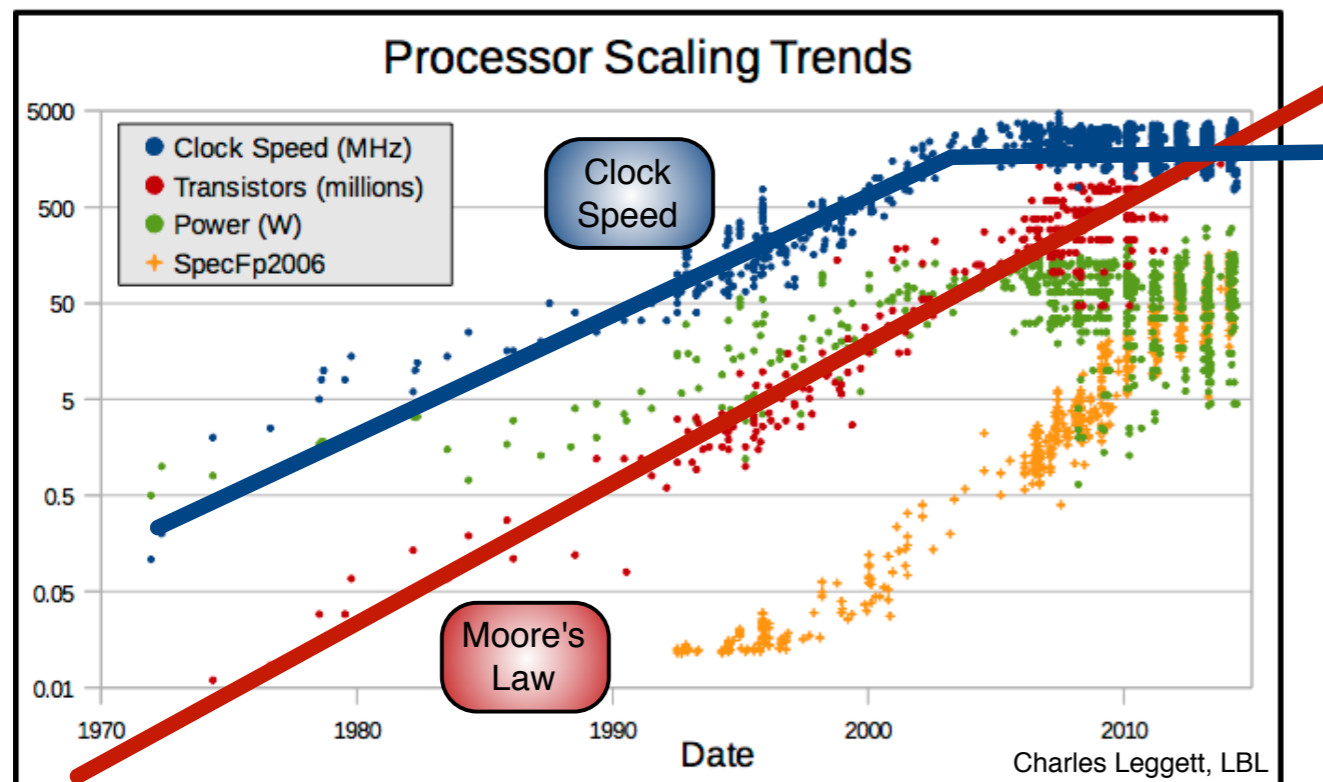


Multi-threaded Software Framework Development for the ATLAS Experiment

Graeme Stewart, Tomaz Bold, John Baines, Paolo Calafiura, Andrea Dotti, Steve Farrell, Charles Leggett, David Malon, Elmar Ritsch, Scott Snyder, Vakho Tsulaia, Peter Van Gemmeren, Ben Wynne *for the ATLAS Experiment*

Computing Challenges

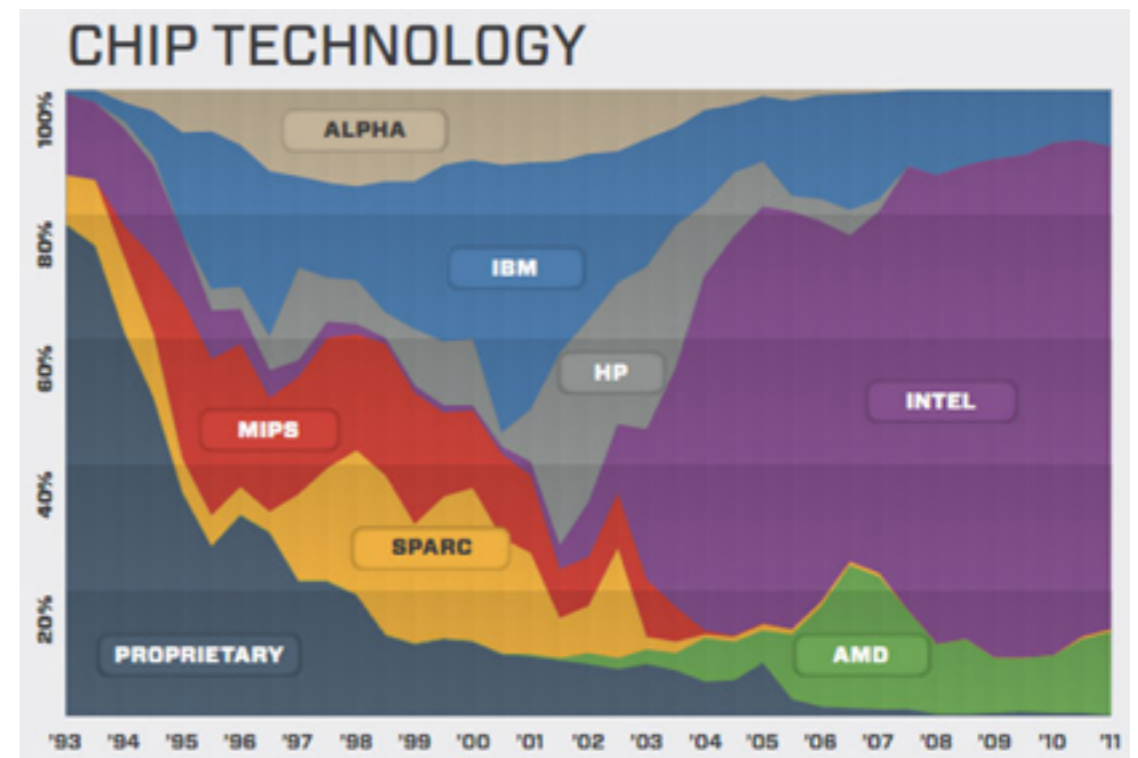
- Great challenge of computing in the next decade will be one of power
 - nJ per instruction
 - Note it is likely that the power costs of memory access would be greater than CPU power in an exascale machine
 - This is driving evolution of larger numbers of cores on dies
 - More transistors but no more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



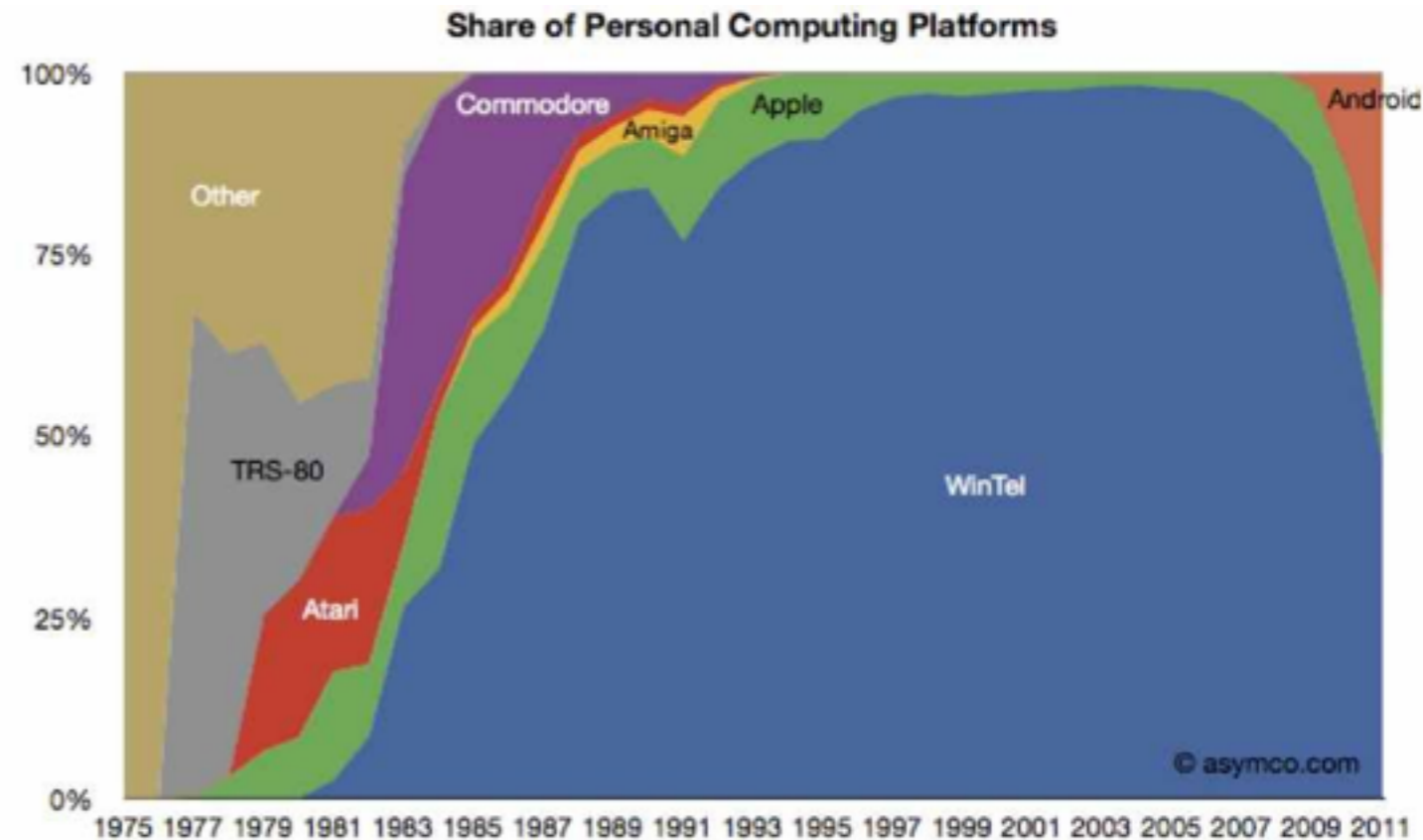
New NERSC CRT building housing Cori machine room at LBNL

End of Homogeneity

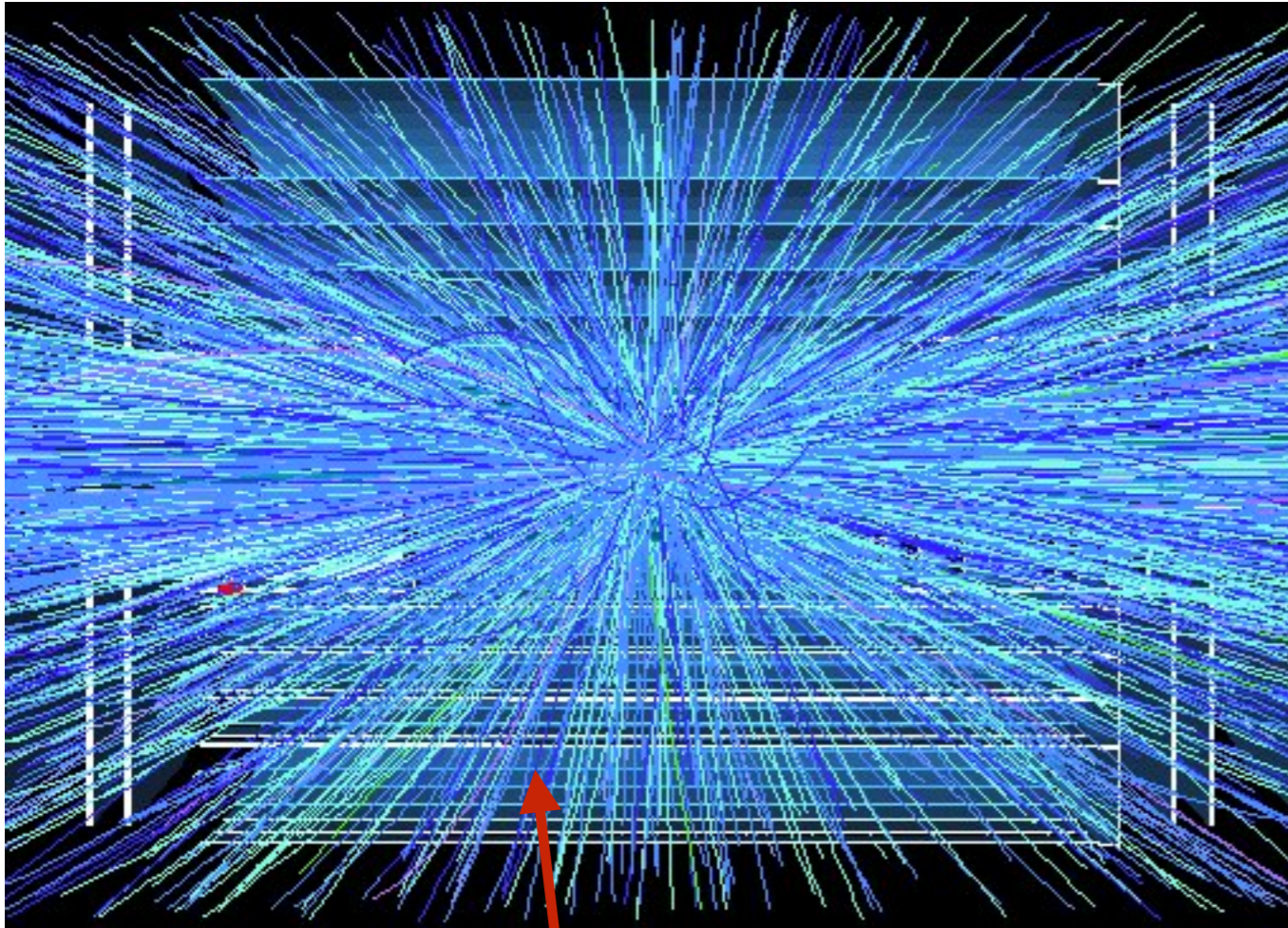
- Computing for LHC experiments was developed at a time when the market became extremely homogeneous
 - x86 architecture was dominant, mainly Intel CPUs
- Physical limitations of high performance and high power efficiency CPU computing are forcing a challenge to this homogeneity
 - Different CPU architectures: Aarch64, PowerPC
 - 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



HPC Installed CPU Shares

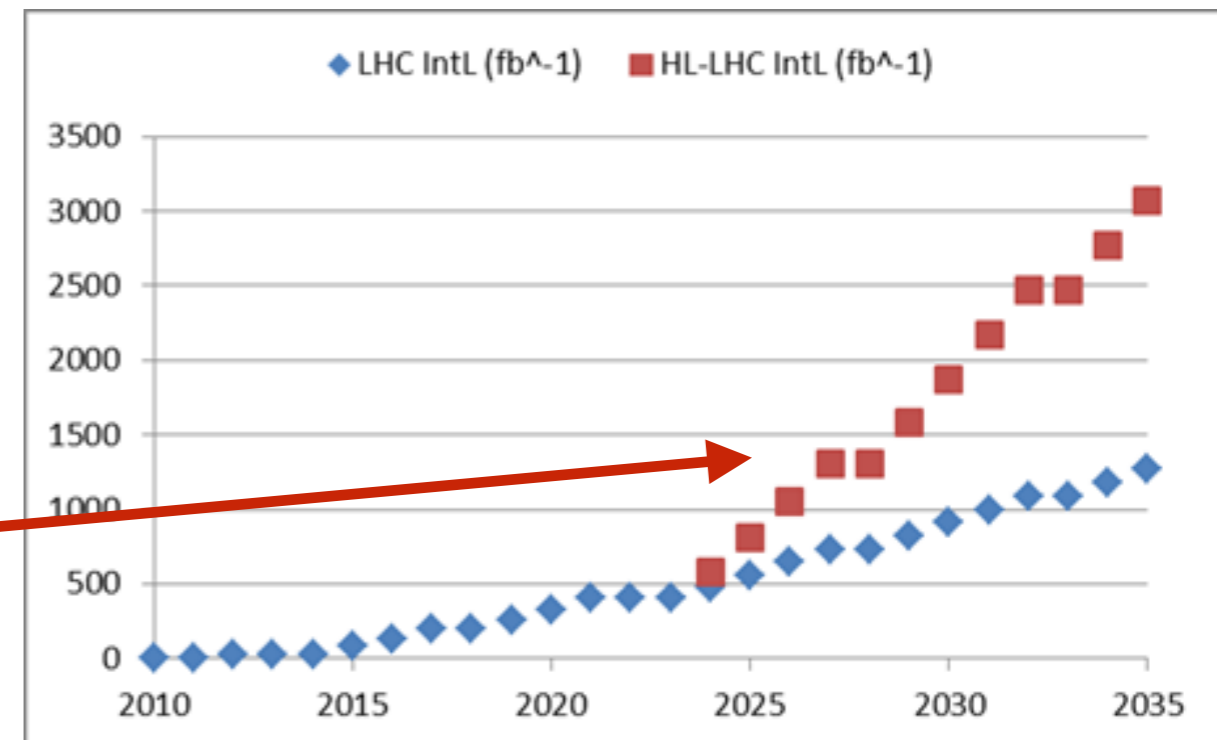


The Road to High Luminosity LHC



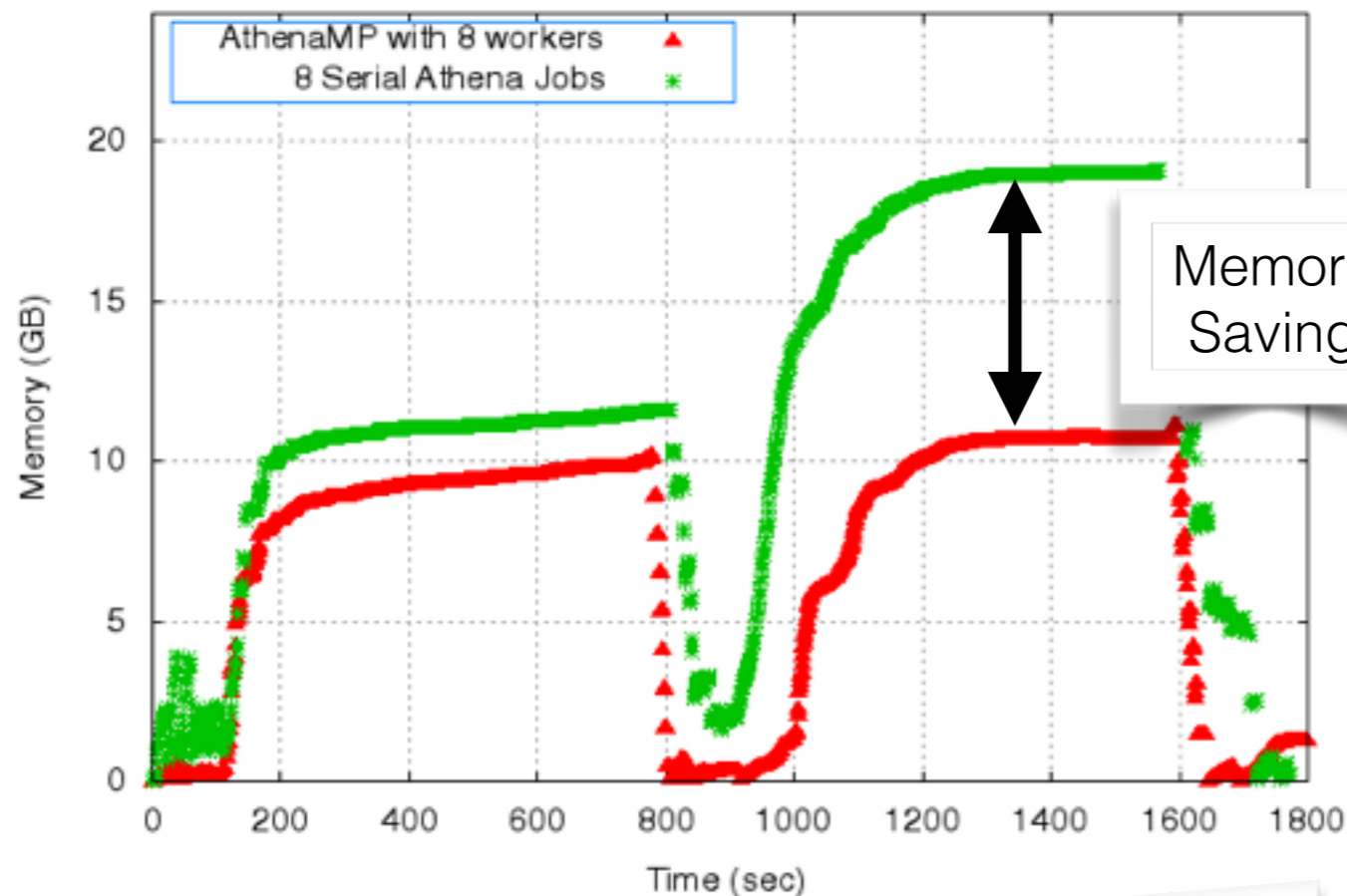
Event Complexity x Rate = Challenge

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



AthenaMP and beyond

ATLAS Preliminary. Memory Profile of MC Reconstruction



- First big wall hit by ATLAS is the *memory wall*
- Multi-processing with *copy on write* (Athena MultiProcess, or AthenaMP) is serving ATLAS well in Run2, but we don't expect this to scale for Run3
- Need a multi-threading solution — genuine memory sharing, with all its known advantages and problems

Current ATLAS software framework is *Athena*, built on top of a generic event processing framework, *Gaudi* (originally developed by LHCb, now a shared project)

FFReq



ATLAS NOTE
ATLAS-SOFT-COM-2014-048
2014-03-13

Draft version 1.1



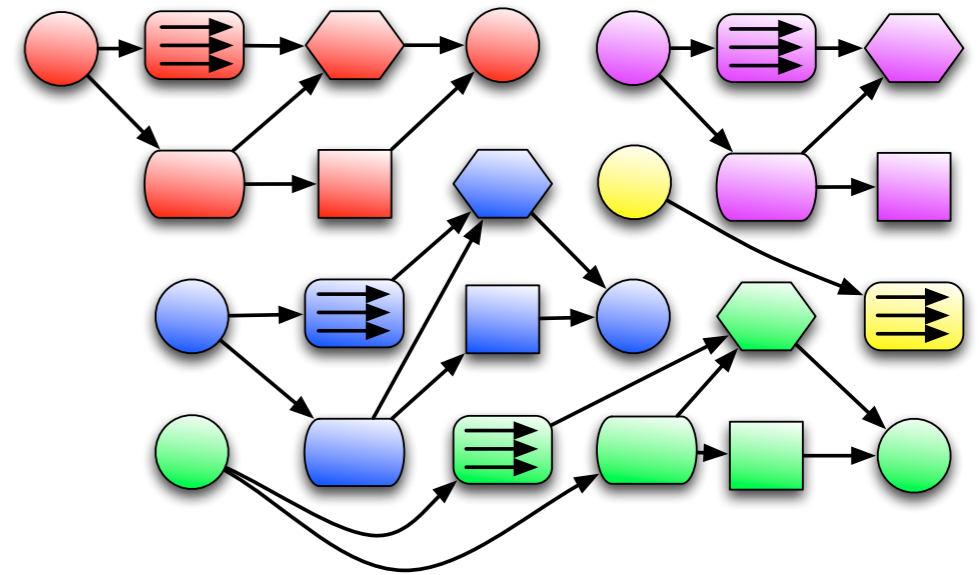
ATLAS Future Framework Requirements Group Report

John Baines, Tomasz Bold, Paolo Calafiura, Sami Karna, Charles Leggett, David Malon,
Graeme A Stewart, Benjamin M Wynne

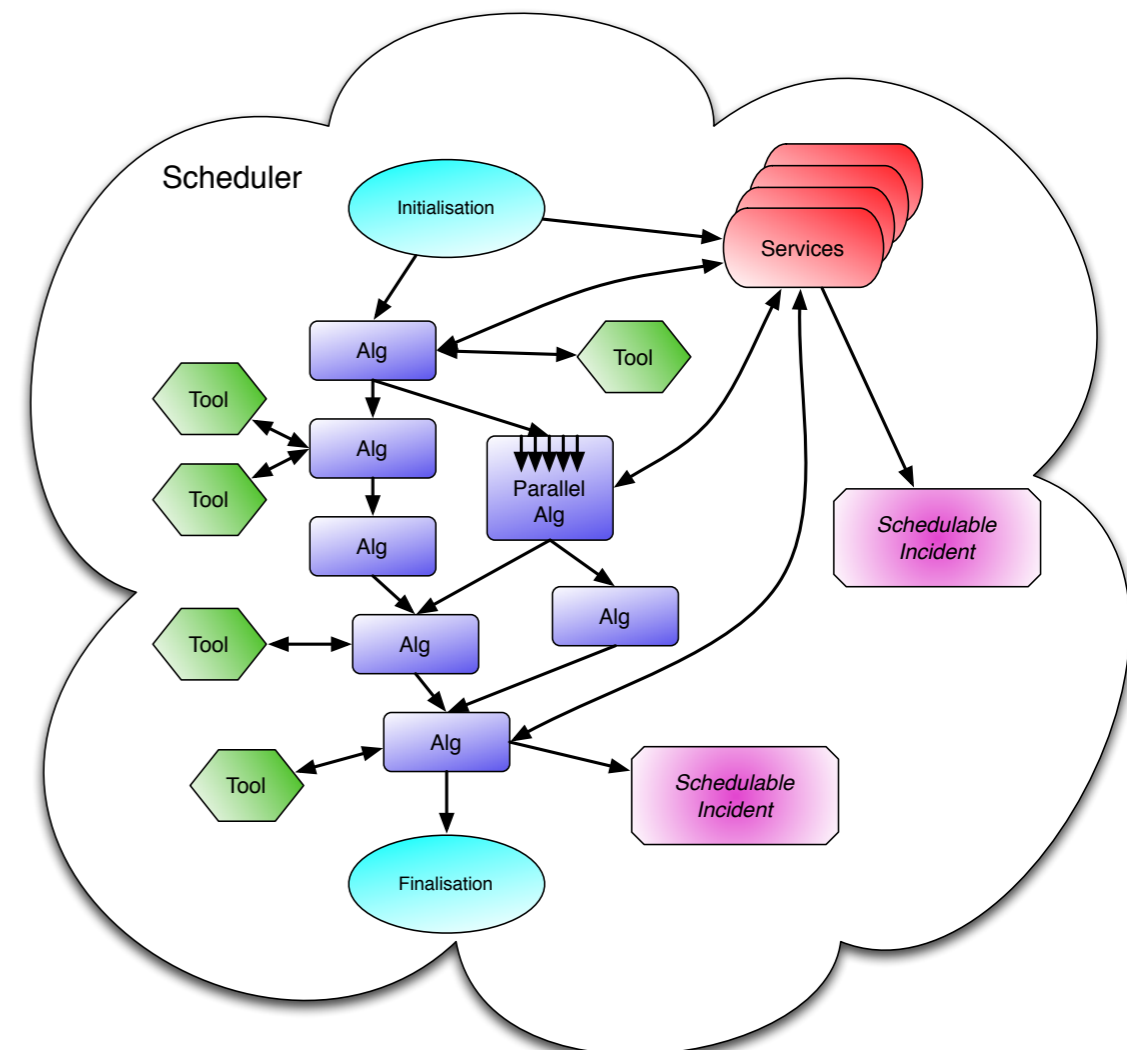
- Step 1 — what do we need?
 - *Future Framework Requirements Taskforce*, aka FFReq
- Design study reported end of 2014 on the requirements for a Run3 framework
 - Large additional motivation was better integration with the ATLAS trigger
 - In particular support for partial event processing in regions of interest
 - At the moment this is served by running a single HLT master algorithm on each event
 - Effectively a sub-event scheduler built into a Gaudi algorithm
- N.B. Easier use of offline algorithms directly in the trigger is one of the things needed for Run3 — maintain trigger's rejection/selection power at higher pile up and higher L1 rates

FFReq: Key Points

- Support high level trigger use cases natively
- 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)



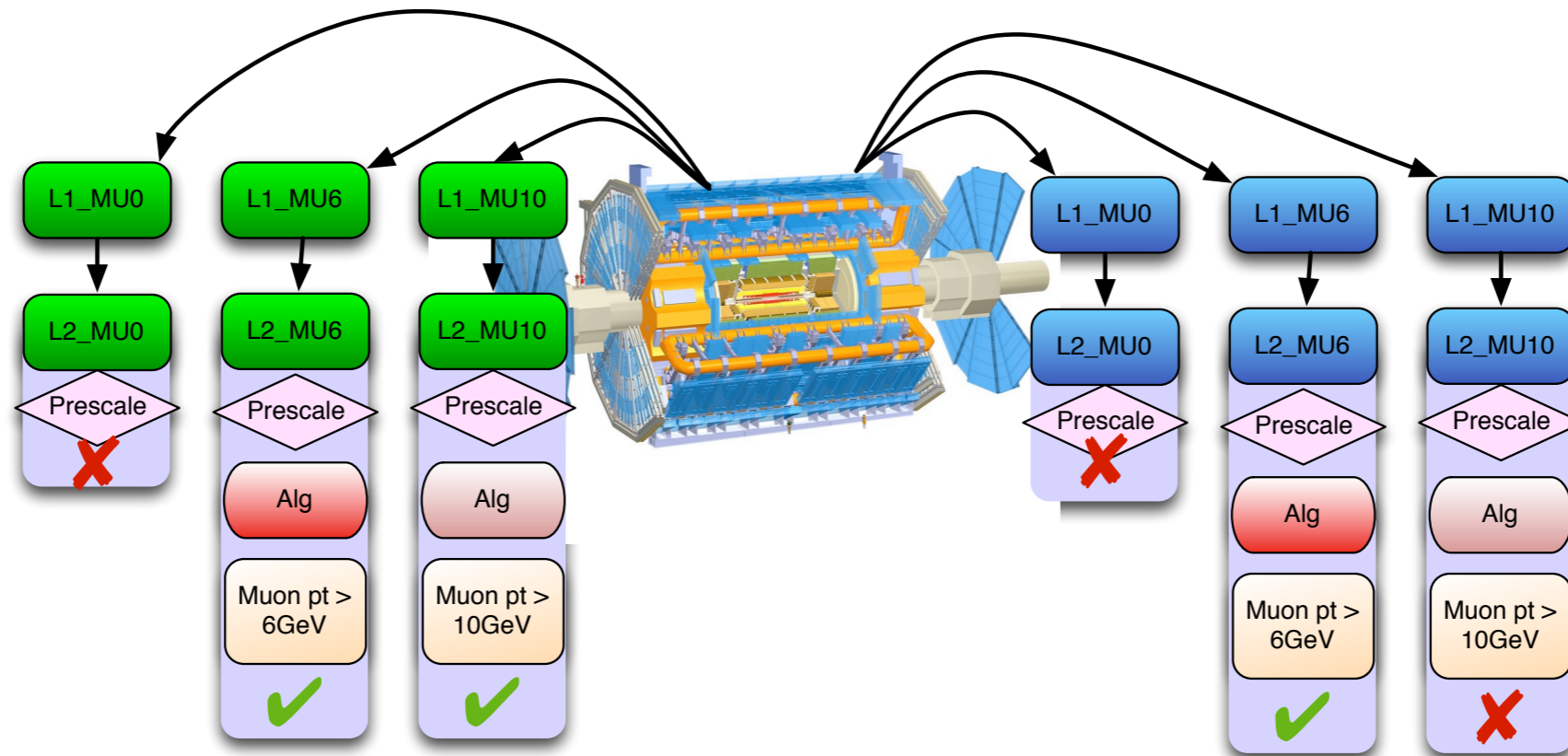
Framework element interaction within a single event

Gaudi & AthenaMT



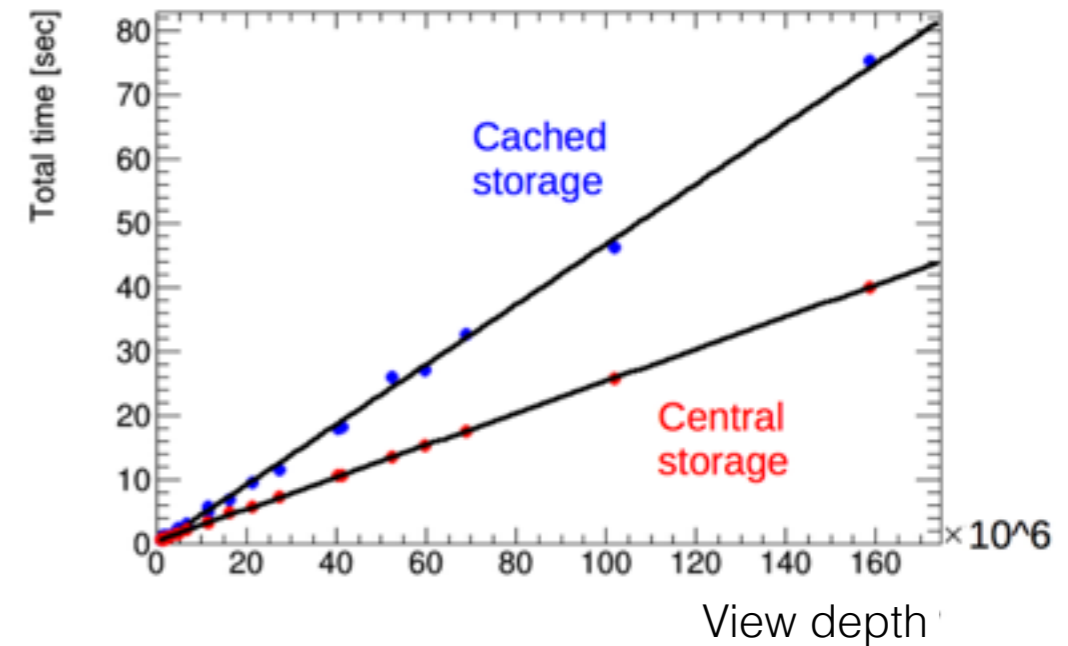
- After due consideration we decided that an updated version of Gaudi fitted ATLAS's needs
- Collaboration with LHCb and SFT was deemed highly desirable
- GaudiHive demonstrator had shown promising results
 - Underlying Intel Threaded Building Blocks had been shown to perform well
- Similar ATLAS CaloHive 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**

Event Views



- Multiple seeds from L1
- In order to minimise investment in *rejected* events (99%) only consider restricted data in each trigger chain
 - Do this by creating a *view* for each *region of interest*
 - Algorithms will run on each RoI that interests them (generally, >1)

Implementation Progress

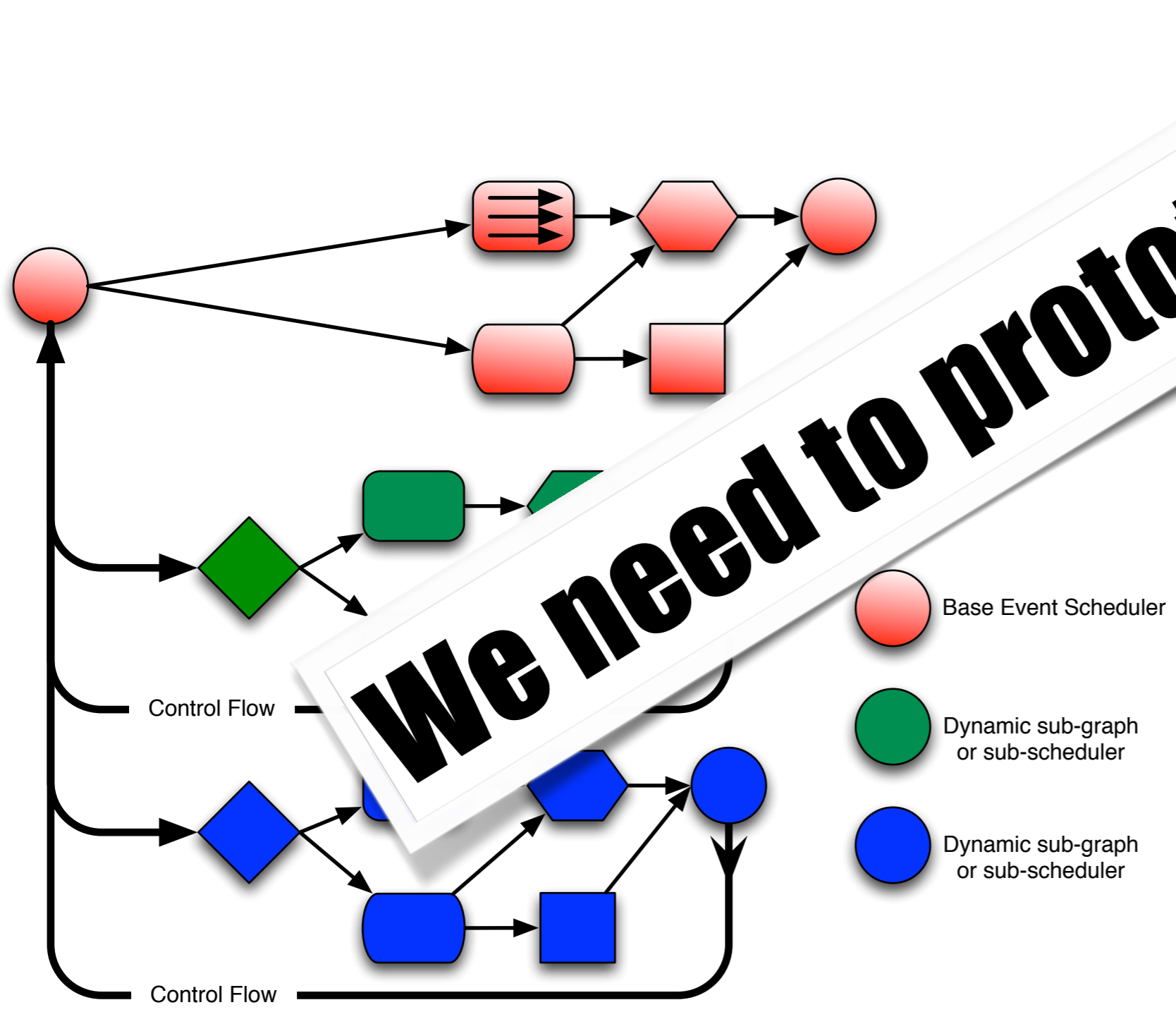





- Measuring synthetic performance indicates that probably the best place for a view to live is within the main event's whiteboard (**central** vs. **cached**)
- We recently extended IProxyDict interface to allow event views to act on a ProxyDict using *data handles*
 - Data proxies used to allow on-demand reading (so it's a bit cleverer than a pointer to memory)
- Then algorithms can read from our offline event store or from an EventView all transparently
 - Both implement IProxyDict interface
 - Transparent use of offline algorithms in trigger ✓
 - No longer a need for HLT algorithms and offline algorithms as distinct entities

Scheduling and Views

- As the number of views is not known until runtime, how do we schedule them?
- Don't want a separate HLT specific scheduler — little better than the solution now
 - Want to take advantage of generic improvements and wider community
- Treat views as separate (sub)events?
 - Problematic for merging — do we really use the same graph for views as the whole event?
- Or use a *multi-scheduler* approach
 - Generation of a view spawns a scheduler to handle that view and run all the needed algorithms on it
 - Parent scheduler will wait for children to finish (child will probably set a control flow state)
- Or dynamic scheduler extensions
 - Will require deeper changes into the scheduler code

View Schedulers



-  Base Event Scheduler
-  Dynamic sub-graph or sub-scheduler
-  Dynamic sub-graph or sub-scheduler

- can just deep copy scheduler (with state!)
- ... just create a view — ... more
- May need to support cases where the child scheduler has a different flow graph
- Would probably be also useful for accelerator plugins
- or*
- At certain points in the graph, allow the dynamic extension with a known sub-graph connected to a view
- Allows for a single scheduler
- Optimises throughput through consumption analysis

Data Handles

- ATLAS's current event store (StoreGateSvc) has interfaces used directly by most tools and algorithms
 - Too tied to a particular usage pattern
 - Not visible to the scheduler
 - First multi-threading prototypes implemented a cumbersome manual declaration of dependencies
- Thus we migrate to *data handles* in order to:
 - Automatically declare data dependencies to the scheduler
 - Note this percolates from algorithm to tool to tool, etc. (very common design pattern in ATLAS to delegate work to a chain of tools)
 - Abstract away from specifics of the event store and treat data (handle) with OO semantics
 - May use different event store implementation for analysis use cases

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 for Run3

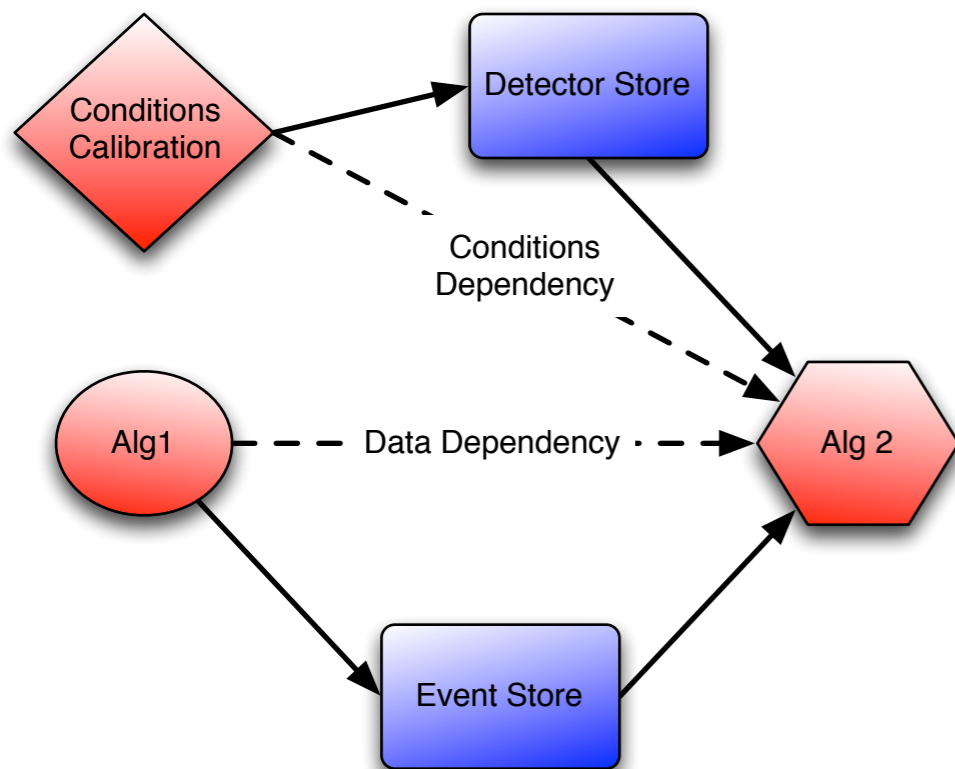
- In parallel to any framework changes we are planning a major upgrade to our conditions data infrastructure for Run3
- Guiding idea is to greatly simplify the database internals (schemas and tables), embed Frontier (caching http layer) into the core design, simplify the client side to REST interactions and unify the serialisation method (e.g., JSON)
 - This borrows heavily from CMS's work during LS2 with whom we're cooperating closely
- Conditions will be stored on lumi-block boundaries as open intervals
- Payload is always just a BLOB
 - Client must know how to interpret it, but the supporting infrastructure is simple
- Global tag (used a lot in ATLAS) is a simple list of child tags

Current Conditions Access

- Conditions access is based on the concept of intervals of validity
 - Basically a value, plus a time or event range for which this value is valid
 - Different conditions have different cadence - from single events, to luminosity blocks (60s) to whole LHC runs
- Current ATLAS model for retrieving conditions is not very suited to multi-event processing
 - Conditions related tools register a callback when a validity boundary is crossed for their folders
 - Callback generally performs a calibration calculation on their conditions data, results stored in private cache
 - Users of this data call the tool to retrieve current calibrated value
 - Conditions database component of Athena is in charge of checking at the beginning of each event if any conditions have gone out of validity
 - If they have then the new data is retrieved from the database; callback is then fired to notify all clients

Conditions as Data

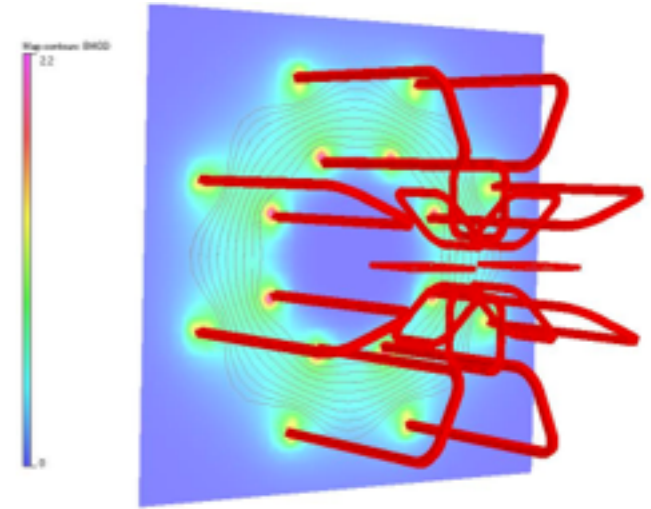
- Instead of conditions being retrieved 'on demand', inform the scheduler they are a data input for an algorithm
 - Prevents stalling while conditions are retrieved
- Instead of callbacks and private caching, write conditions algorithms that perform the calibration and use the event store for calibrated values
 - Use detector store because conditions cadence is very different from event cadence
 - Internally a container by IOV
- 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
 - Bulk requests and more efficient interactions
- Time varying data, with a different cadence to events is a common problem
 - Scope for a common project with Gaudi partners



I/O and Metadata

- We have started to look at our i/o infrastructure
 - Audit for thread safety issues
- Near term goals:
 - Simultaneous reading and writing from i/o layer — more or less in place
 - Supporting Athena G4 multi-threading prototype
 - Simultaneous writing to multiple streams — should come early next year
- Longer term we want to
 - Revisit transient/persistent layers and look again at direct ROOT conversion services
 - Try and simplify things
 - Ensure we make less assumptions about how we read data
 - Particularly towards more event streaming over the network
- In-file Metadata sub-system requires a real re-design
 - Currently incident driven — even with AthenaMP we have issues and file merging is a real pain

Service Migration: Magnetic Field

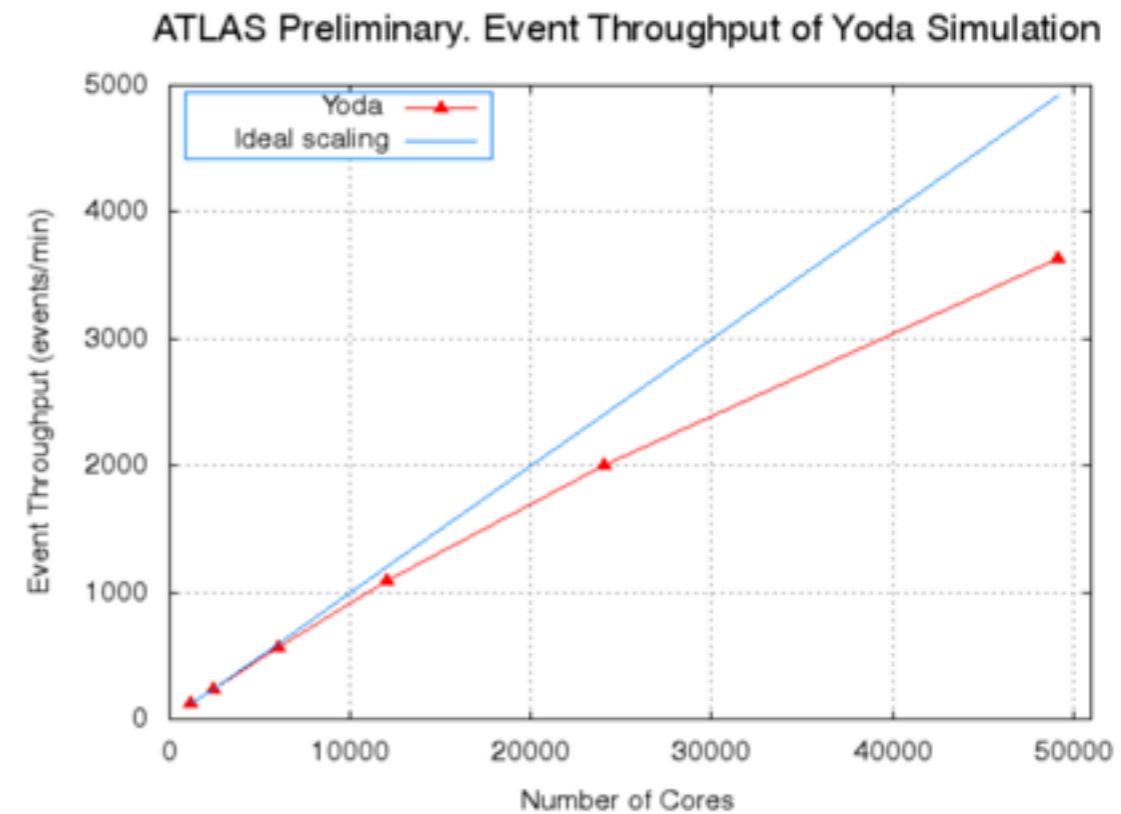


- Good example of efficient, but serial, code is the ATLAS magnetic field service
 - Field is computed via a Biot-Savart component from magnet currents in the detector
 - Plus an interpolated correction, based on position: $B_{cor}(x,y,z)$
 - Looked up in a 300MB field map, which is expensive (and used 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

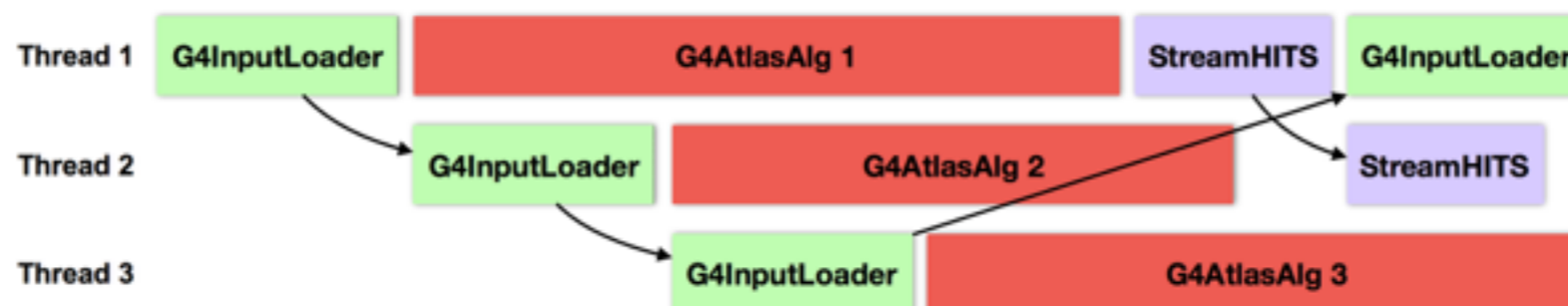
Multi-Process-Thread Flexibility



- Note that our multi-process architecture in AthenaMP will not disappear
 - We already make good use of it when running on big machines, such as supercomputers
 - Cannot scale to those machines with threads alone
- Knight's Landing boards will come with NUMA domains
- AthenaMP and AthenaMT together
 - Run processes per NUMA domain, with threads in each domain to constrain memory
- We also use MPI to straddle across nodes for cases where the scheduling unit of a machine is multiple nodes (supercomputers)
 - Have run millions of CPU hours on NERSC's Edison machine in this way
- Goal remains to maximise the range of resources we can take advantage of

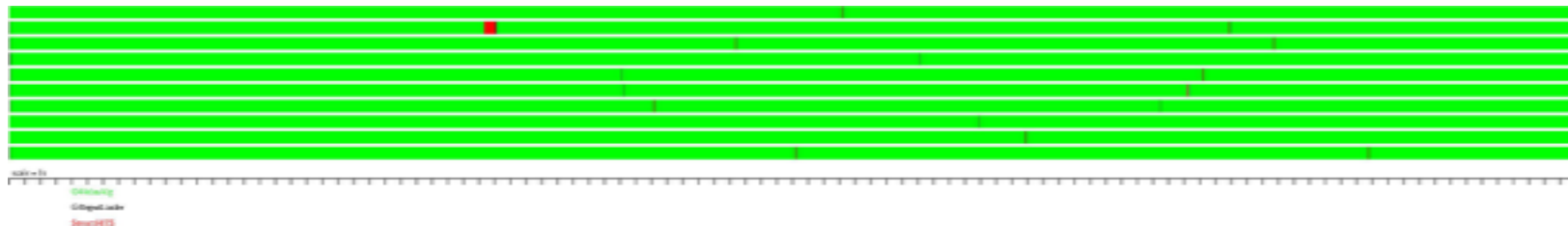
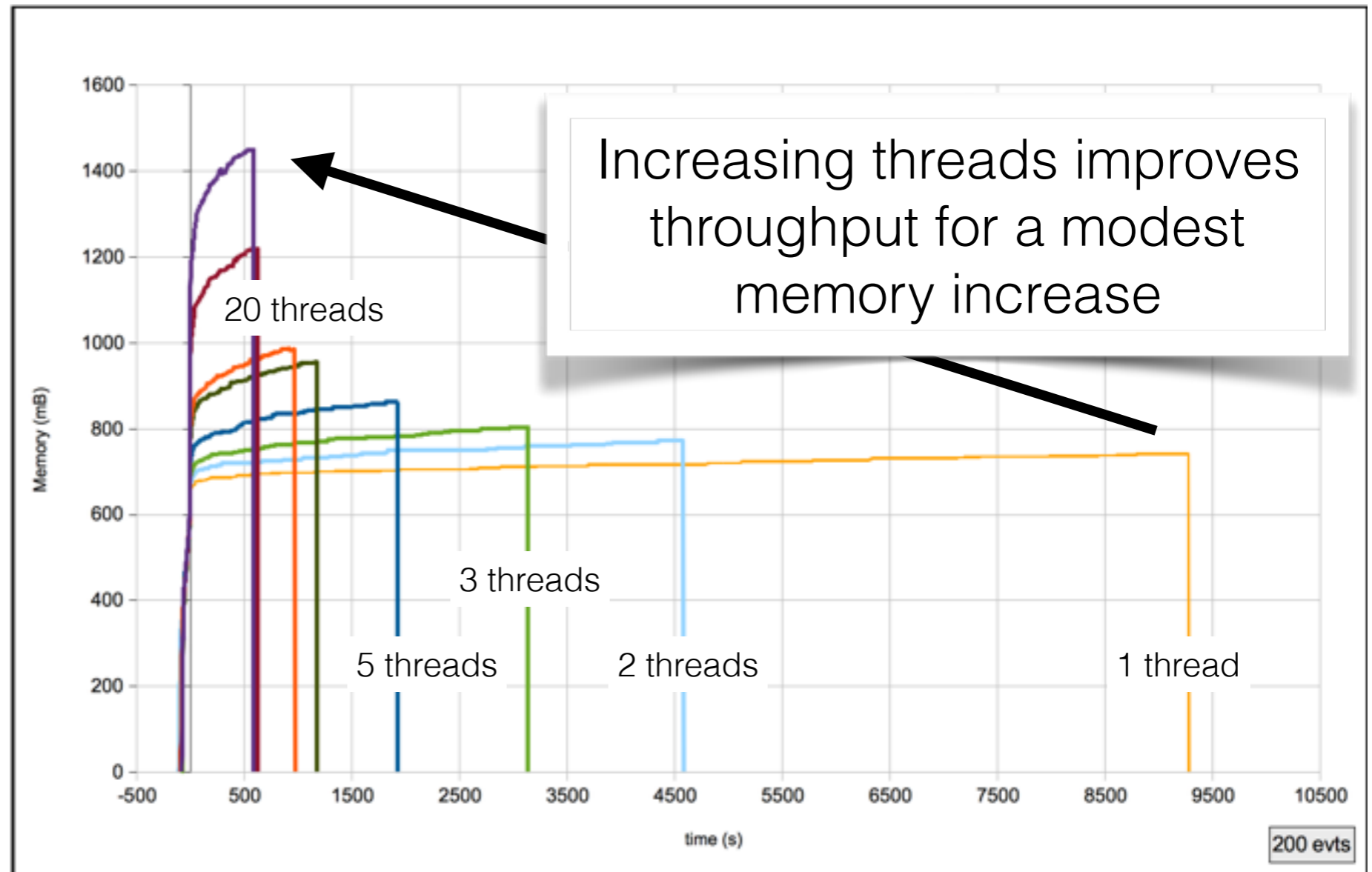
~~G4Hive~~ AthenaG4MT

- 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 serial processing built in (see previous slide on i/o)
- Teaching us about the balancing act between hacked solutions and over elaborate designs — focus on the actual problem!



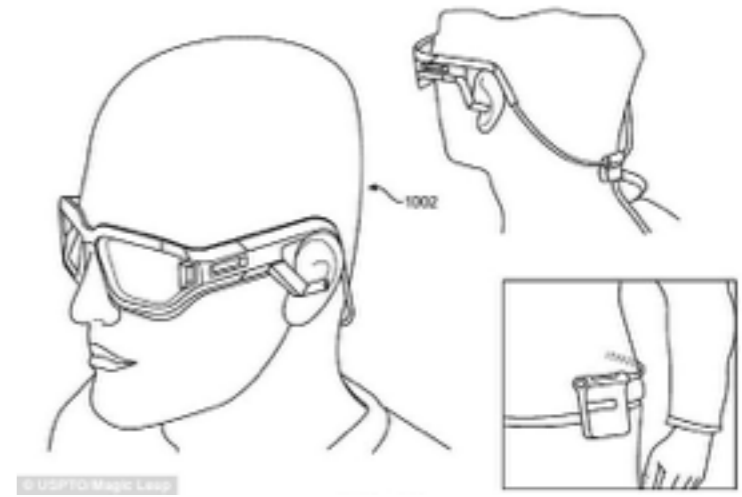
AthenaG4MT Results

- 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

Code Review



- To try and understand where we are with the algorithmic code we will undertake a software review this year
 - A high level review of subsystem code
 - What's the design...? (Is there a 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
- Outcome may well be just start over — e.g., some ATLAS muons algorithms and Simulation base infrastructure rewrite

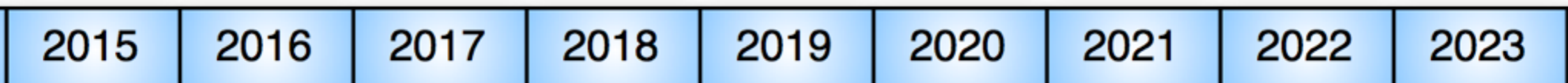
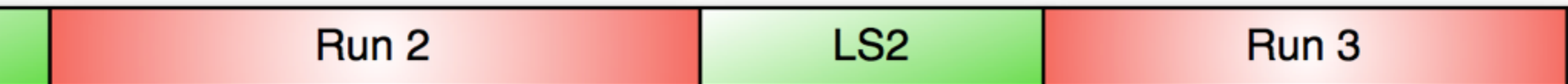
Migration Tools and Strategy



- 3.5 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
 - May then be able to 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

- Phase I 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
- Very healthy revival of Gaudi as a community effort
 - Particularly helpful discussions with LHCb
- Have started to seriously think about what the algorithmic code should look like for Run3
 - There will be a lot of code we need to rewrite
 - Important to start discussions with reco, sim and analysis groups to shape the new framework and the new interfaces properly
 - Code review will help us to understand and evolve today's code
 - And provide good examples for the community

“Engineering is really a social undertaking”

—Gene Amdahl (1922-2015)

Gene Amdahl — pioneer of understanding parallel computing