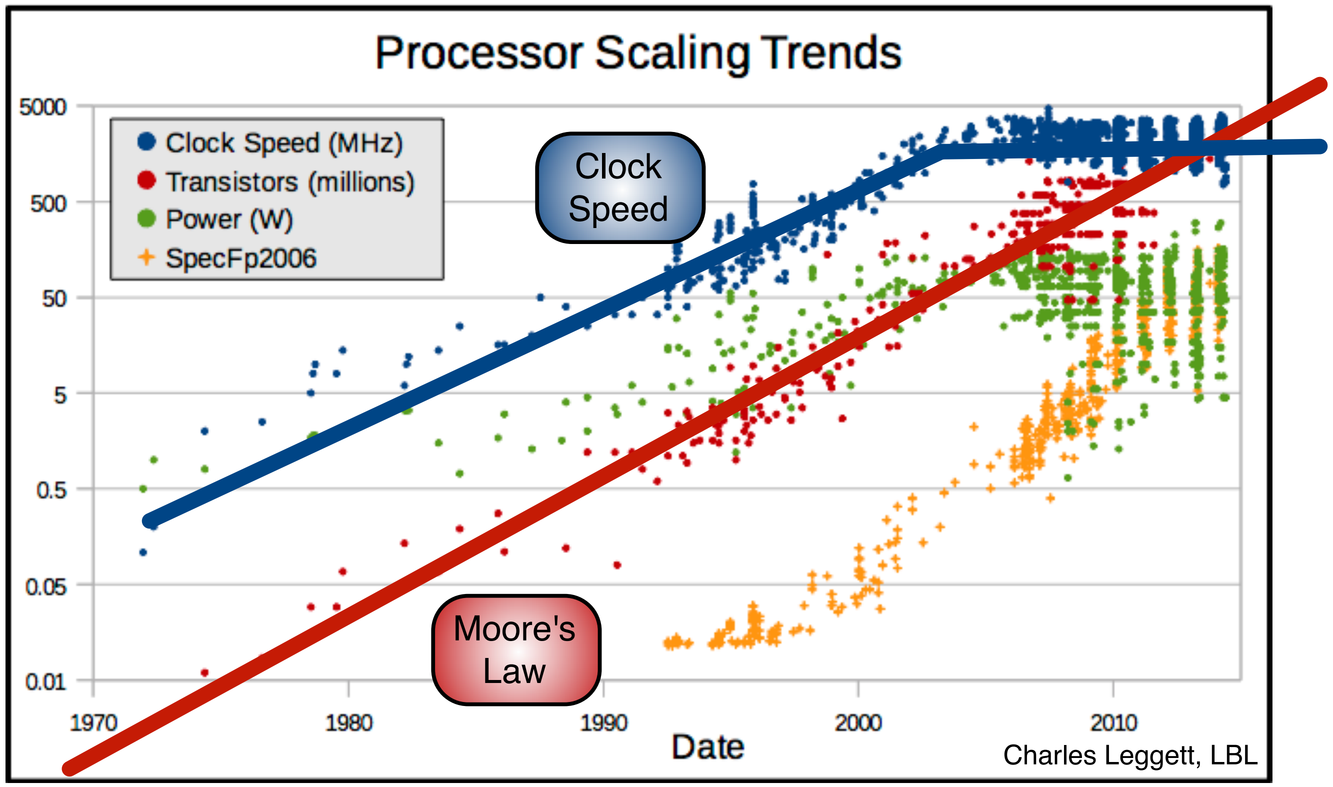
Overview and Status for ATLAS Phase I Software Upgrades

Graeme Stewart

University of Glasgow | School of Physics & Astronomy

2015-12-01

Processor Evolution



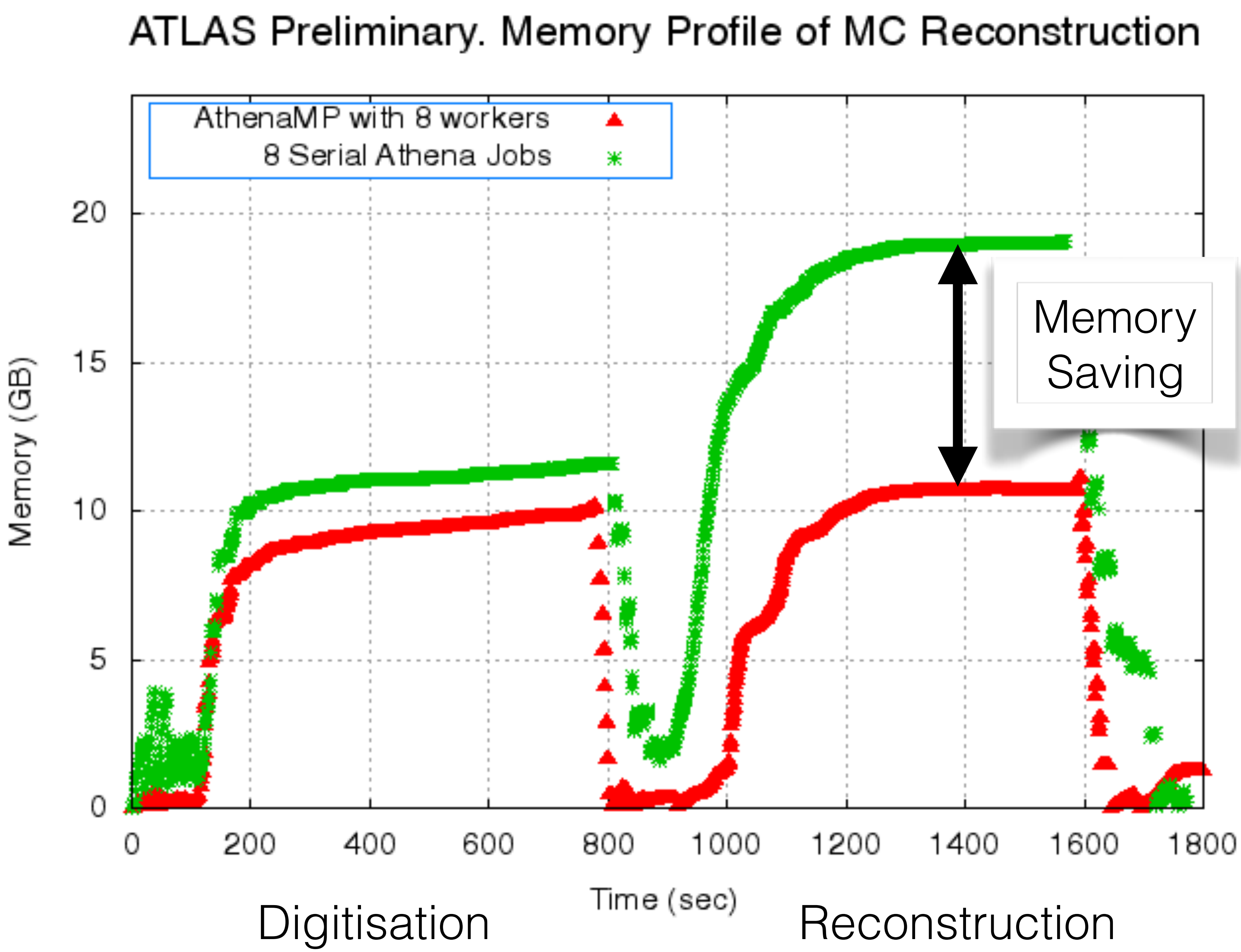
New NERSC machine room at LBNL

- Moore's Law continues
  - Doubling transistor density every ~24 months
  - Exact doubling time has a significant effect when integrated (especially for Phase II)
- Clock speed stalled ~2005
  - Single core performance is essentially also stalled
- Driven now by energy performance
  - Figure of merit is nJ per instruction
  - Mobile devices and data centres are the key volume markets

HEP and Modern CPUs

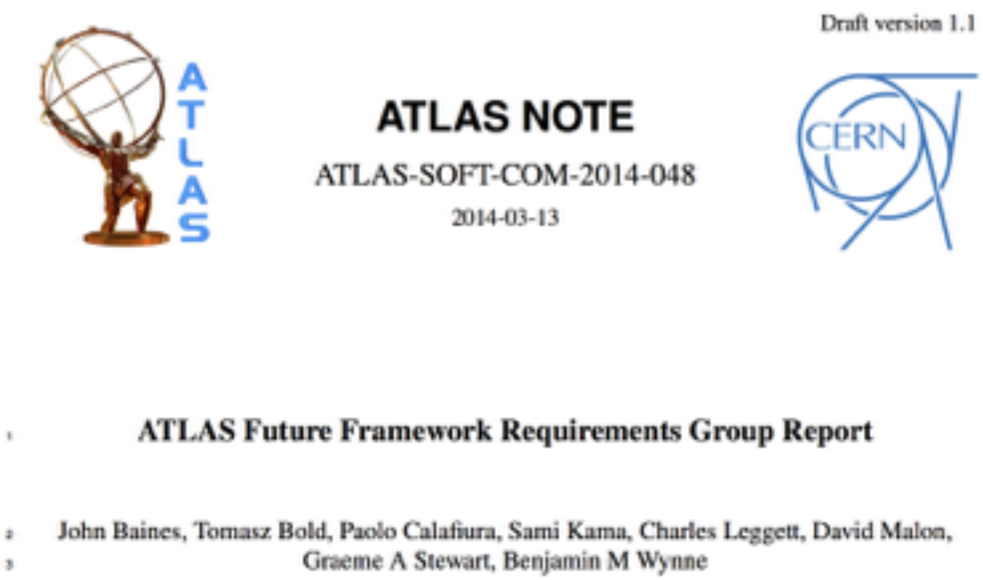
- Away from the detector itself we are firmly *Commodity Off The Shelf* (COTS)
- Doubling transistor density does not double our computing throughput
  - On the die we have more and more cores
  - Lower memory per core
  - Larger caches, but with decreasing payoffs
  - Wide vector registers
  - Built in 'specialist' features, e.g., integrated GPUs in Intel Skylake
  - Integrated network controllers — more *System on a Chip* (SoC)
- None of these features are trivial to take advantage of in our code
  - Our frameworks and algorithms written for an earlier era of hardware and are hard to adapt
- We also need to factor in the real cost of a server — less improvement than a CPU
  - Plus disk, tape, and network evolution (though I shall not cover these here)

Run2: AthenaMP



- Multi-processing with *copy on write* (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

Future Framework Requirements


Draft version 1.1

ATLAS NOTE

ATLAS-SOFT-COM-2014-048

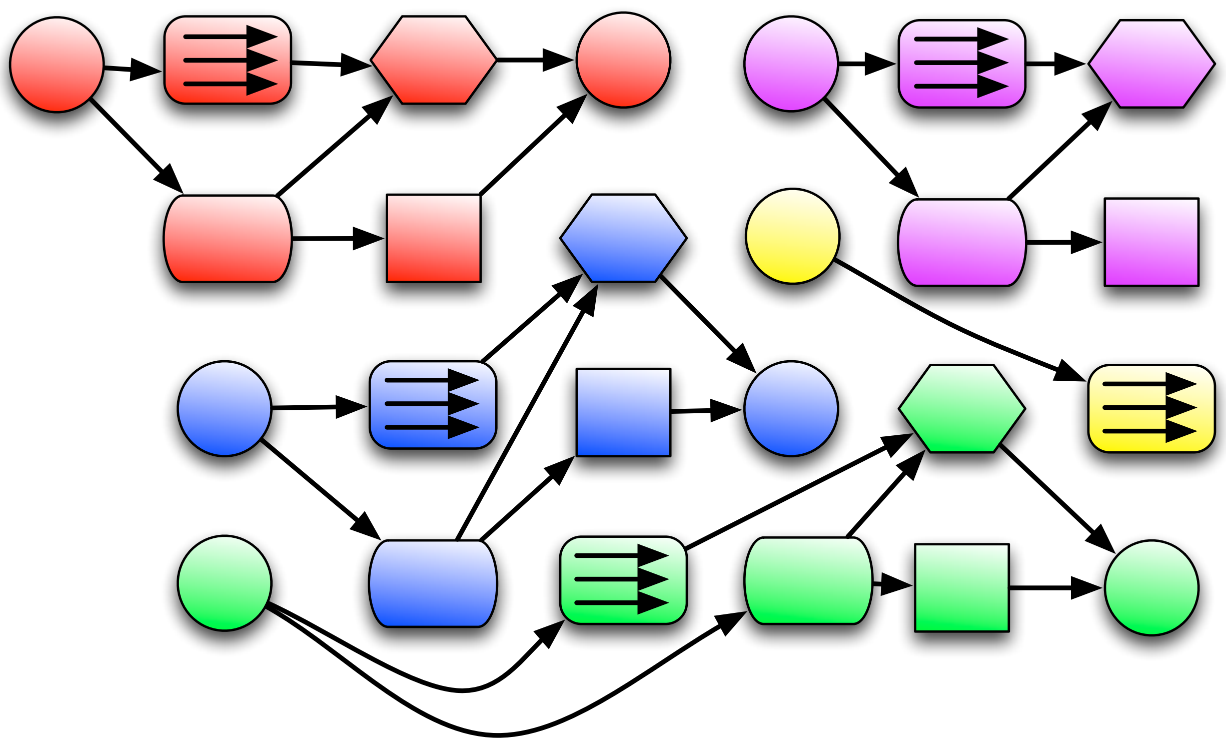
2014-03-13

ATLAS Future Framework Requirements Group Report

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

- Design study reported end of 2014 on the requirements for a Run3 framework
    - Multi-threading a key requirement
    - Additional motivation was better integration with the ATLAS trigger
        - In particular support for partial event processing in regions of interest
            - Current solution is not ideal and prevents easy utilisation of offline algorithms
    - Easier use of offline algorithms directly in the trigger one of the things we will need for Run3 — maintain trigger's rejection/selection power at higher pile up and L1 rates

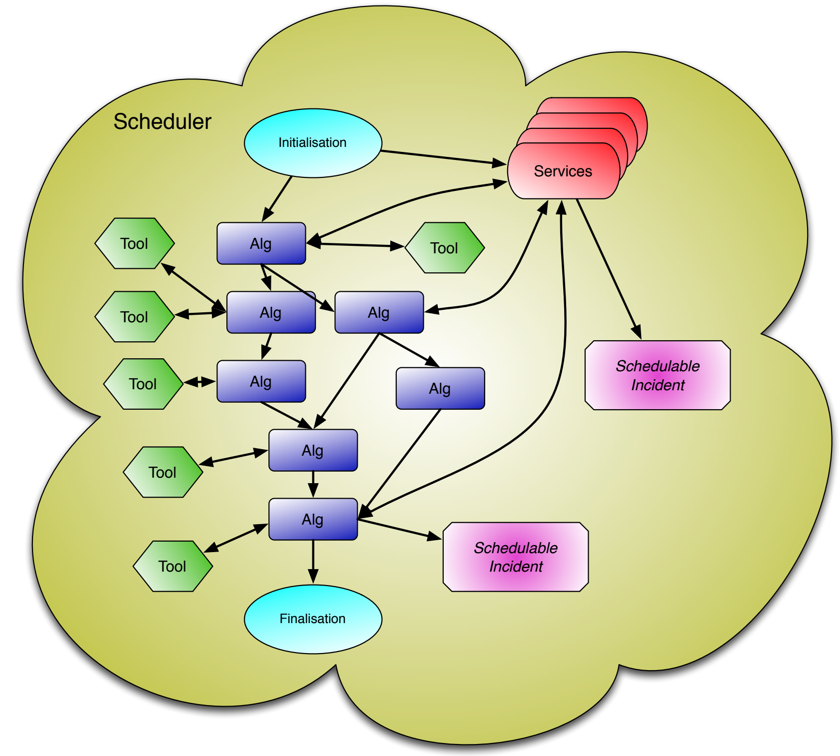
Framework Evolution



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

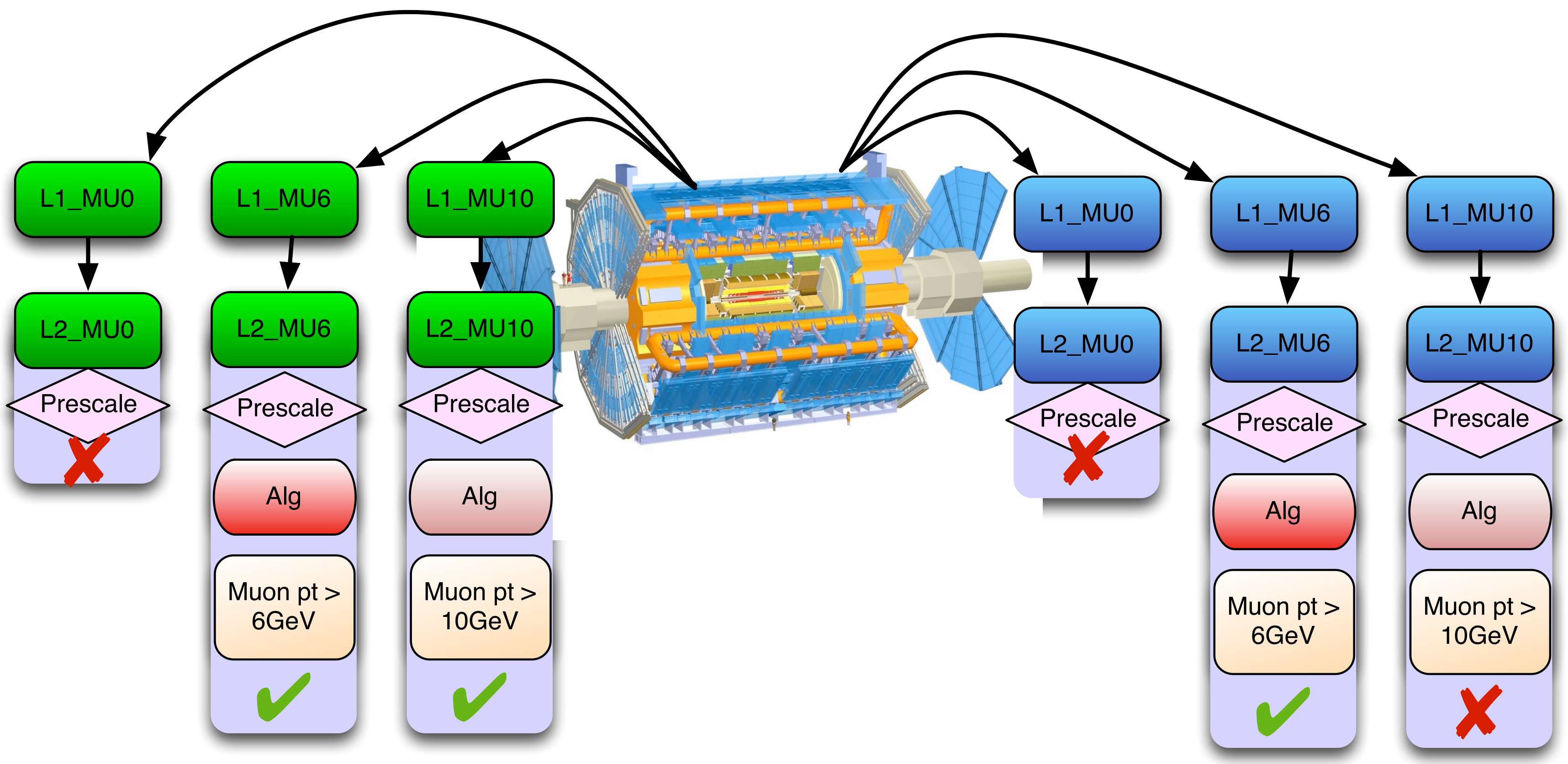
- Many ideas and concepts stay the same
    - Mature model of event processing already
    - Evolve towards concurrency
    - Support HLT usecases
- Best fit to the requirements was to evolve the Gaudi framework
    - Beneficial collaboration with LHCb, SFT and FCC
    - New ATLAS framework will be *AthenaMT*

Key Concept Changes



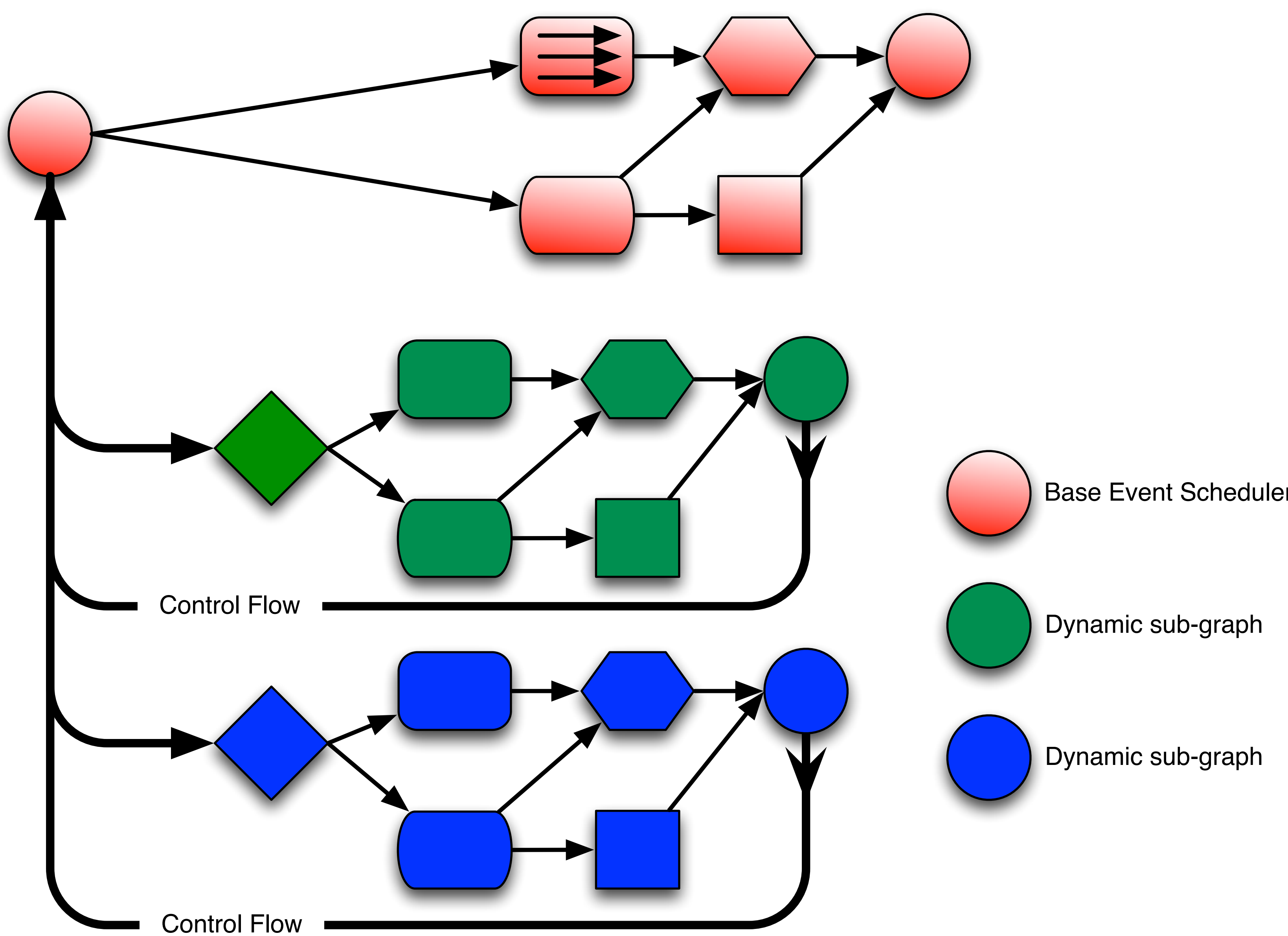
- Data dependencies are explicit and visible
  - Happen via the whiteboard
- Conditions data is *just data*, retrieved in advance of running an algorithm
- Scheduler will parallelise algorithms and events when possible (subject to constraints)
- Scheduler handles non-event work
  - e.g., Incidents, if still necessary, become ‘tasks’
- Algorithms and tools are event specific
  - Tools are always private
    - Use **only the whiteboard** for inter-algorithm communication
    - Use *sequences* for algorithms that create, modify, modify (+done) a data object
- Services are global - must be aware of context when called from algs and tools
  - Also need to be thread safe — trickiest elements to program

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)

Dynamic Scheduler Extension for Views



- 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
- Clearly more prototyping needed

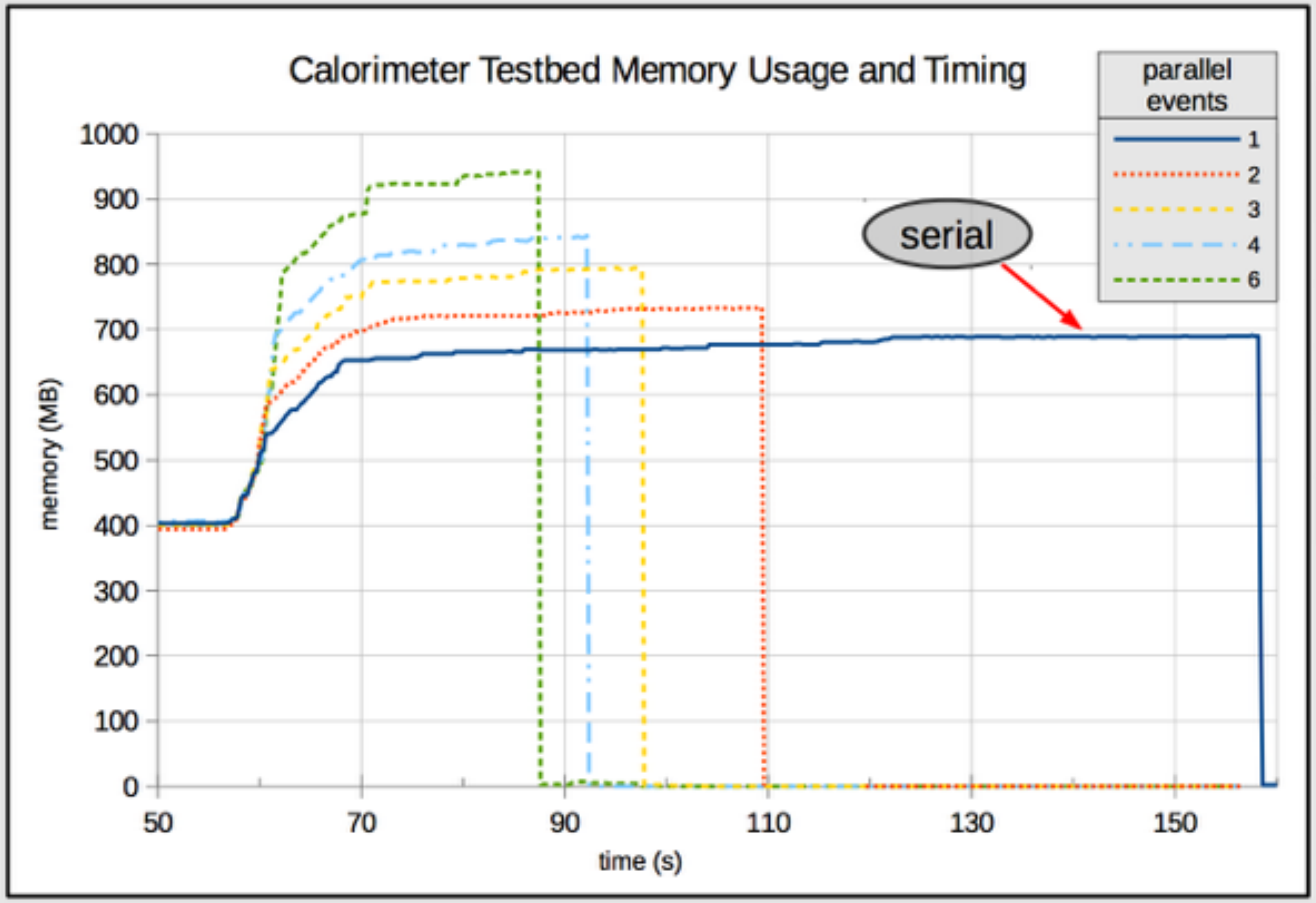
Data Interactions

- Our code, by and large, is all about data interactions
  - Complexity of the workflow is defined by which pieces of data interact via algorithms
- Current serial implementation allows may avenues for data to interact
  - White board, public tools, cached variables, etc.
- Makes scheduling hard and gives rise to data races
- *Data handles* are an abstract way for algorithms and tools to interact with the event store
  - Handles allow for automatic declaration of 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
- New implementation will allow for **`const`** `execute()` algorithms
  - This pattern is the most beneficial for the new framework
    - Allows execution on multiple events with minimum memory consumption
    - `const` declaration allows for smart compile time checks

Design and Implementation

- Design and implementation methodology is *agile*
  - Maximise flexibility
  - Rapid feedback between design and implementation
  - Resist over-designing for imagined use cases
- Thus we have some early implementations to
  - Prove tangibly the approach is correct
  - Uncover issues early that require re-design
  - Test different prototypes when alternatives exist

Calorimeter Testbed Timing Results

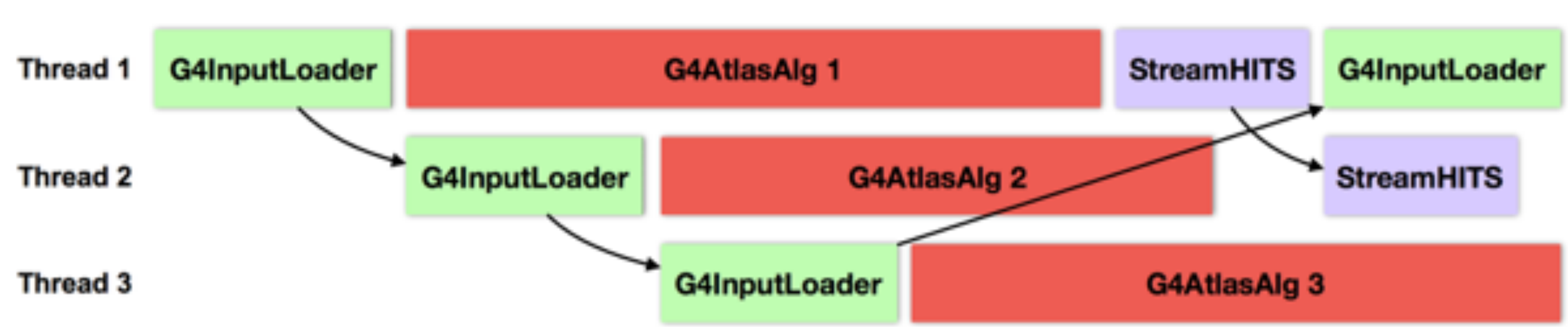


parallel events	speedup wrt serial	speedup wrt n*Serial	memory ratio to serial	memory ratio to n*Serial
1	1.00	1.00	1.00	1.00
2	2.07	1.04	1.06	0.53
3	2.80	0.93	1.15	0.38
4	3.33	0.83	1.22	0.31
6	4.01	0.67	1.36	0.23

- Concurrency limited by small number of Algorithms in configuration, some of which could not be run concurrently for thread safety issues
- Best performance is with **6** concurrent events
  - **401%** event throughput, (ignoring startup to 1st event), and **36%** increase in memory consumption *vs* one serial job
  - **67**% event throughput, and **23%** of memory utilization of 6 serial jobs running concurrently

G4Hive

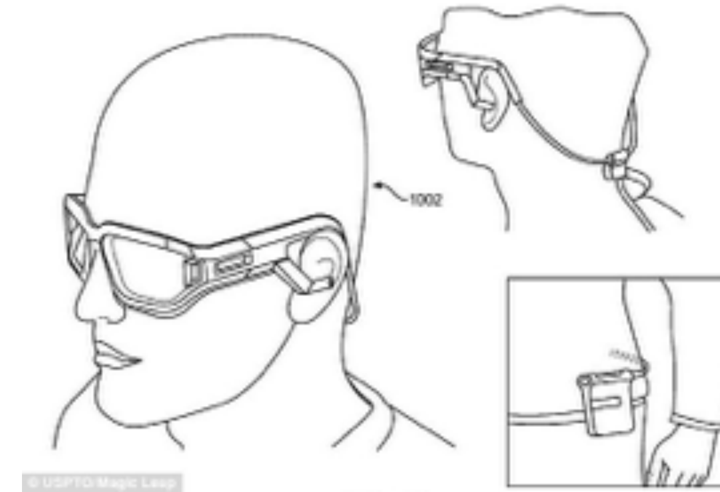
- Attempt to get multiple G4 events running on different threads, controlled by Gaudi scheduler
  - Strong motivation is Phase II Cori machine at NERSC and other HPCs
  - 9300 Knights Landing machines (670 000 cores)
- This has been a very instructive exercise
  - Sensitive detector classes needed a 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!



Parallel Tracking?

- One way to overcome an expensive set of algorithms is just to throw more events into flight
  - But at some point we are going to run out of memory again
- So we really want to open up parallelism within algorithms
- But actually, tracking is one of the most serial pieces of code we have
  - Clever serial design to battle n! combinatorics
  - Ambiguity solver (crudely) picks good tracks one at a time and removes hits
- We can foresee some improvements to our current model
  - Try parallel track seeding and fitting
  - We have an idea for a pattern where a serial tool is run in parallel over a container by the framework
- But maybe we just need to do something entirely new (e.g., Data Science machine learning workshop last month)
  - For Run4?

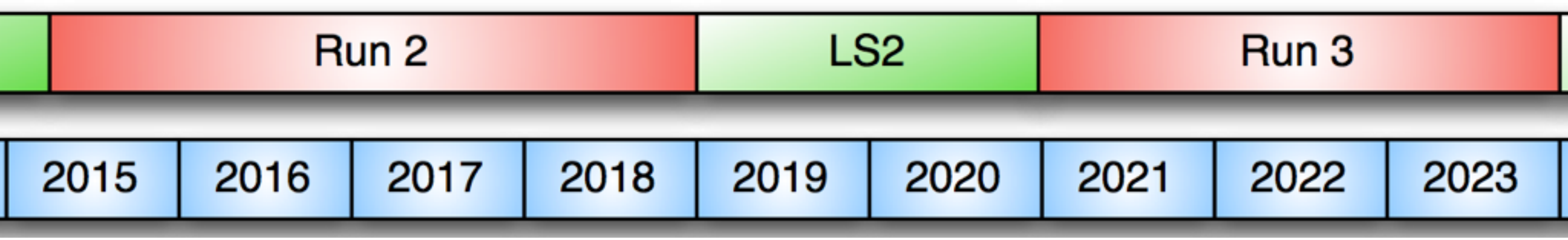
Code Review



- To try and understand where we are with the algorithmic code we will undertake a software review next 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., MuGirl algorithms and Simulation infrastructure rewrite

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

“Engineering is really a social undertaking”

—Gene Amdahl (1922-2015)

Gene Amdahl — pioneer of understanding parallel computing

- 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