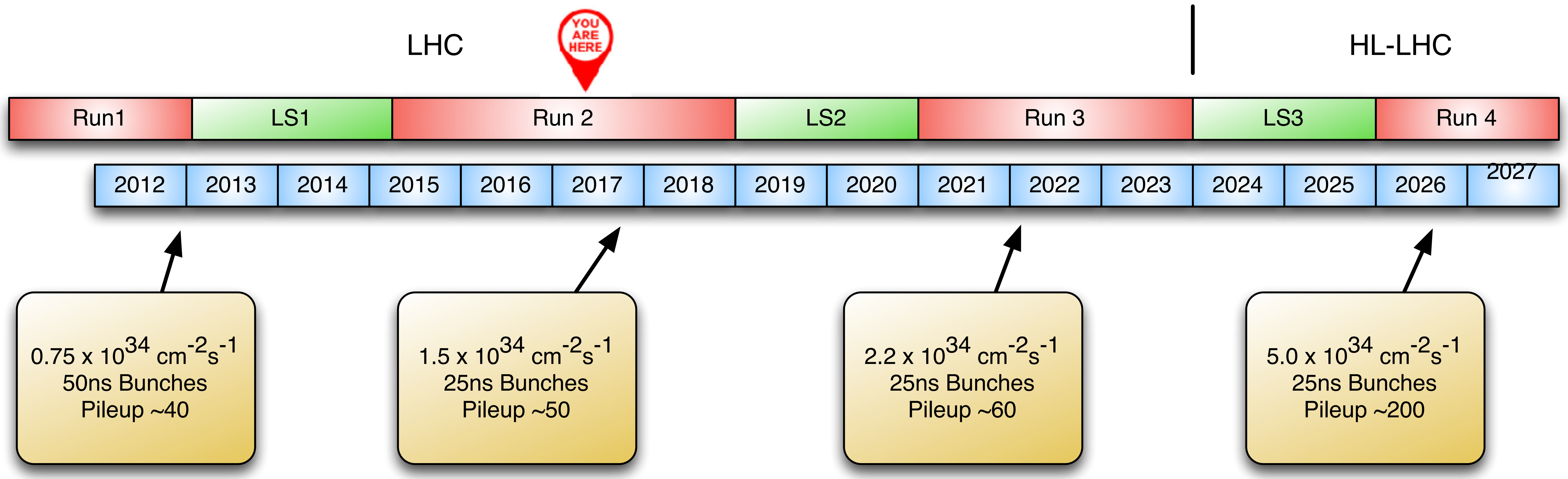
Software Evolution: a view from ATLAS

Graeme Stewart and Walter Lampl

University of Glasgow | School of Physics & Astronomy

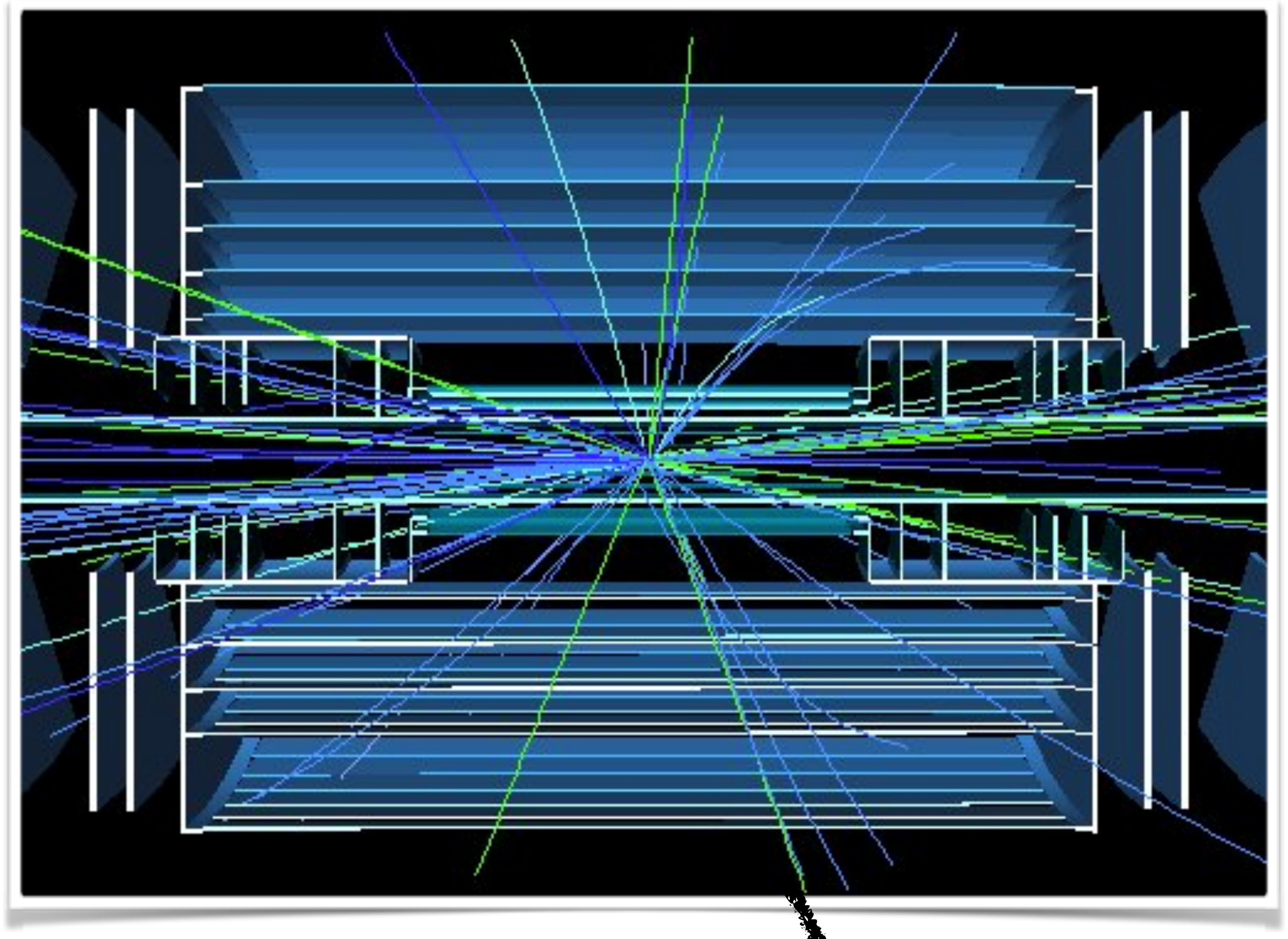
2017-03-23

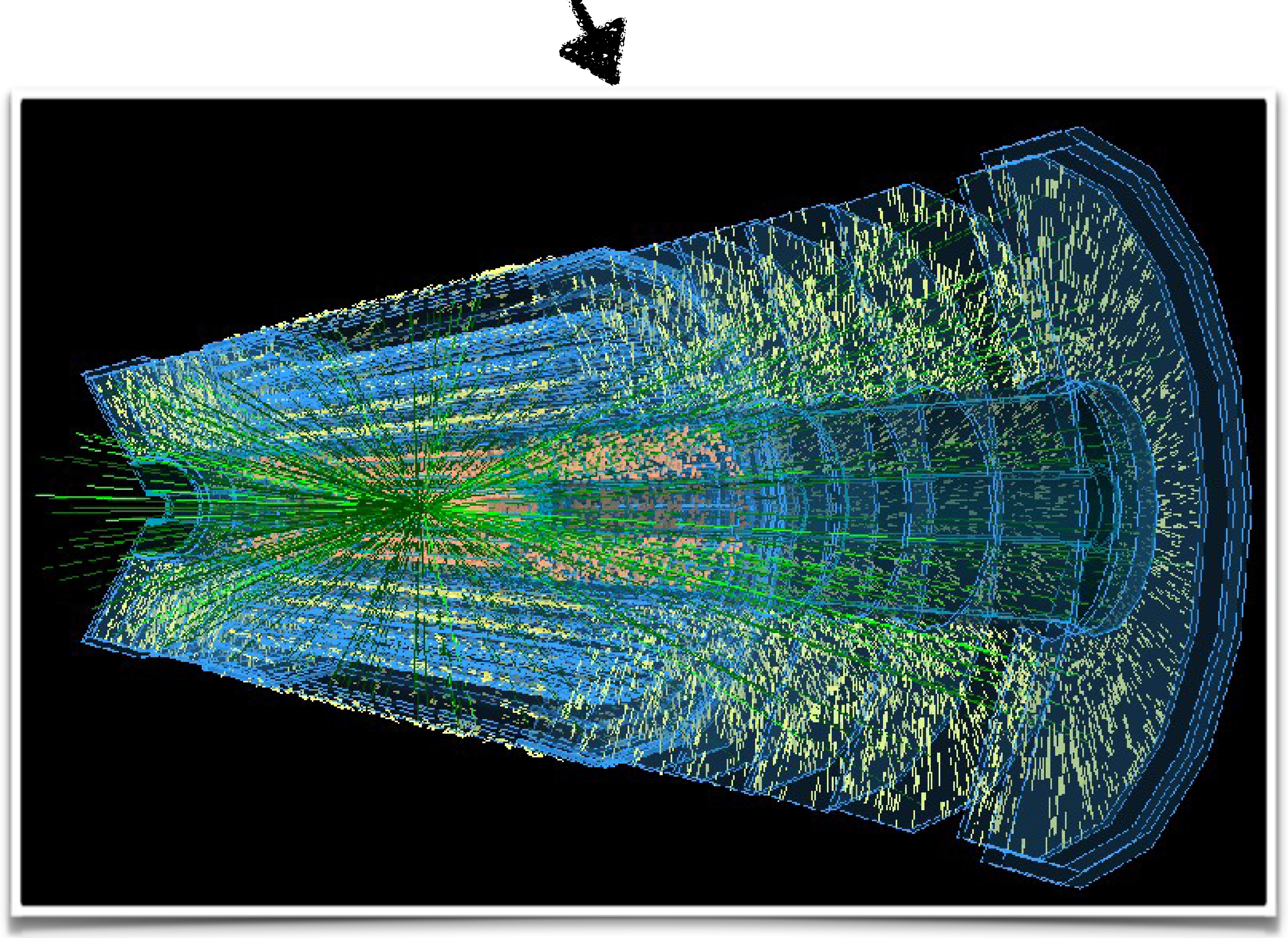
High Luminosity LHC



- High luminosity LHC will deliver about x10 increase in luminosity over what we have today to ATLAS and CMS
  - Needed for precision physics program and to increase the discovery reach of ATLAS

The Challenge

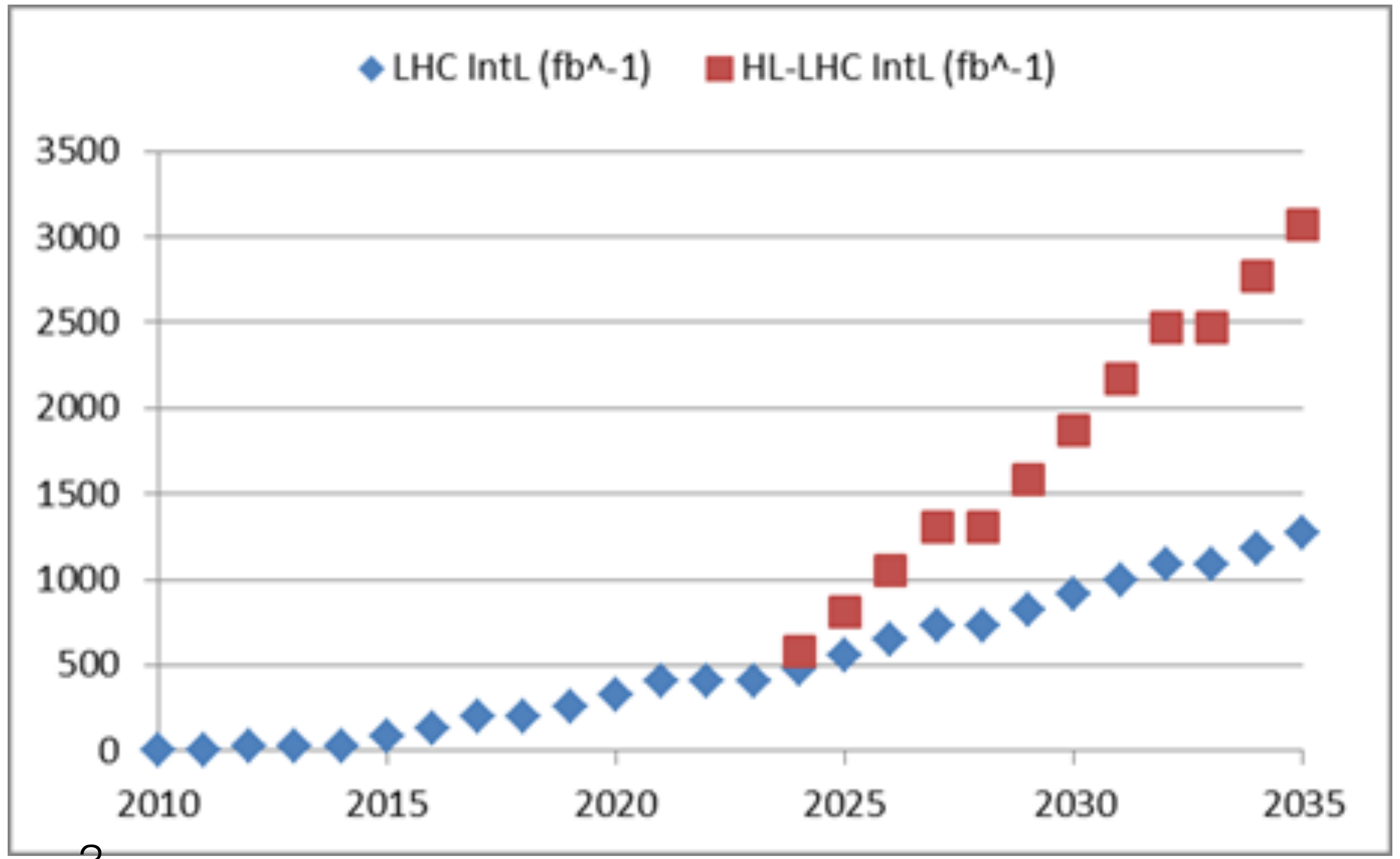




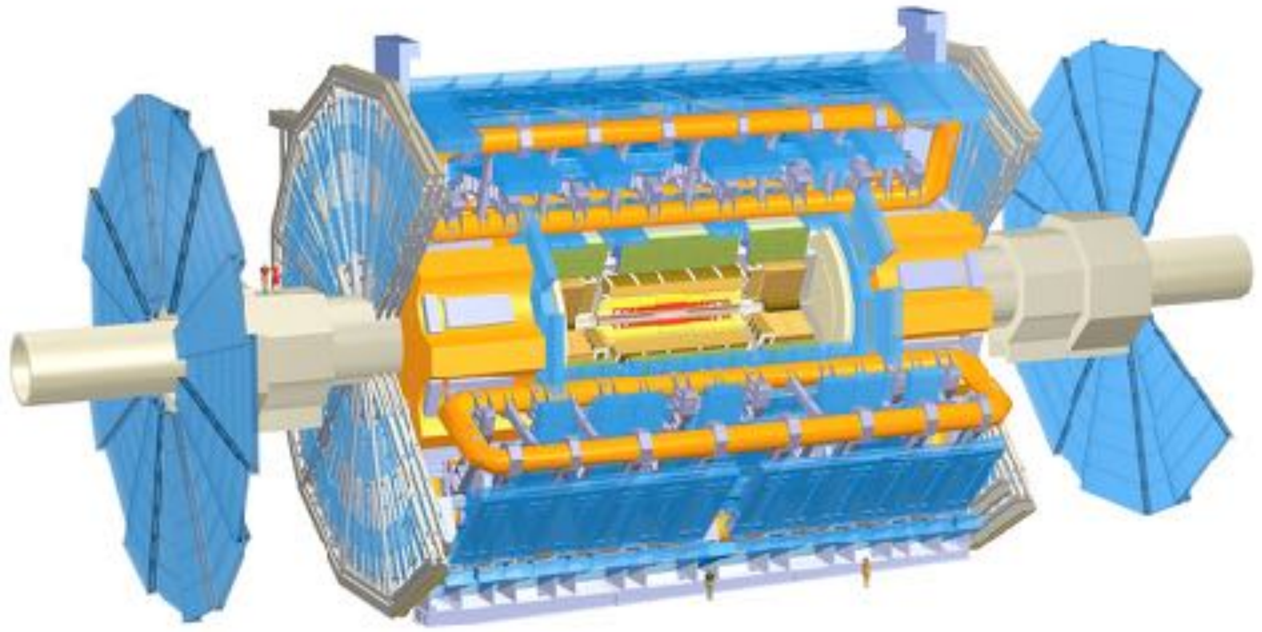
X

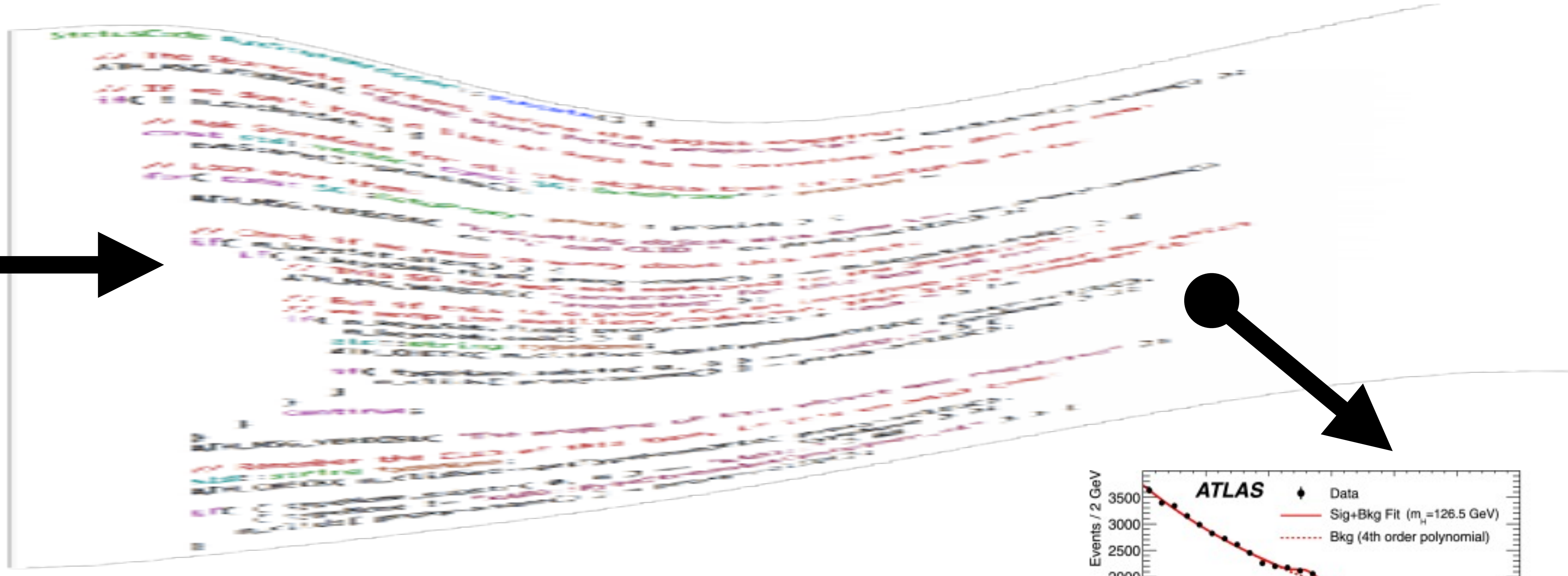
Event Complexity x Rate = Computing Challenge

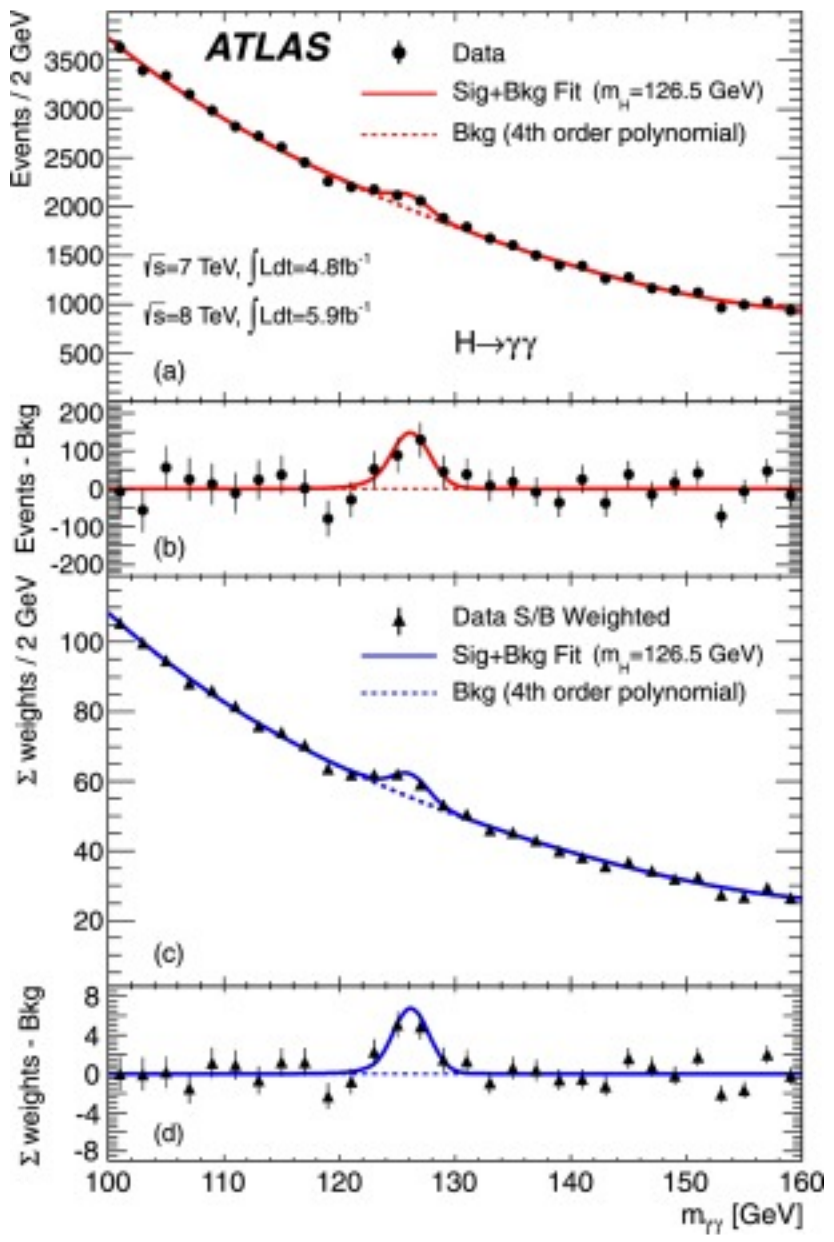
- Reconstruction event complexity is highly non-linear with the number of interacting protons (we call this pileup)
- Rate increases
  - 40MHz LHC interaction rate
  - ATLAS trigger will reduce that rate to ~1MHz in hardware then to 10kHz written to offline
  - This is x10 more than we have today



ATLAS Software







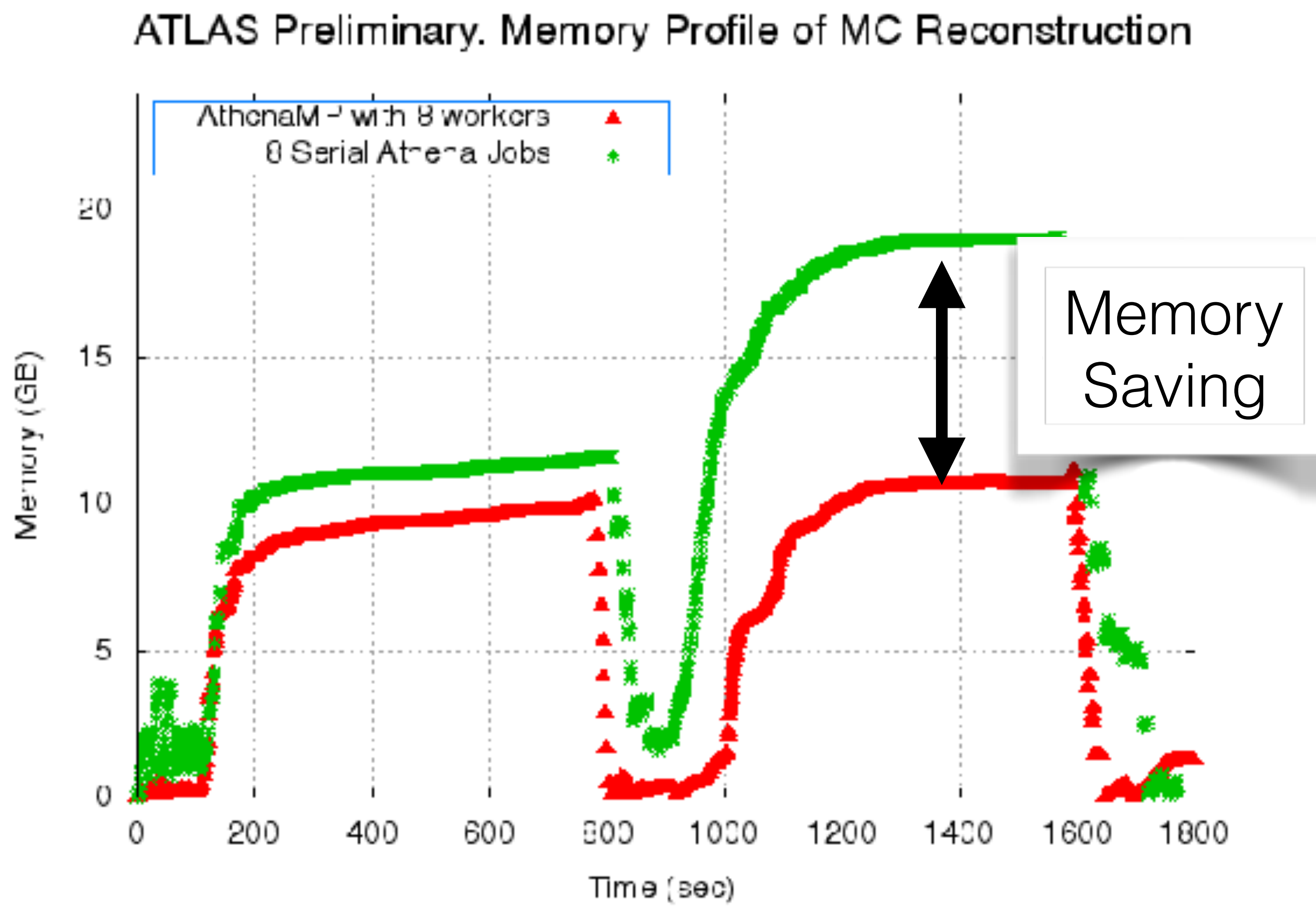
- Software plays a critical role in ATLAS physics production
- Our main Athena code base is ~4M lines of C++ and ~1.5M lines of python
  - This does event generation, simulation, digitisation, reconstruction
  - This excludes a lot of the ‘end of chain’ analysis code (see Axel’s talk later)

Human Challenge

- We have a very diverse developer community
  - A few super-experts who are genuine hard core C++ gurus
  - A modest pool of physicist programmers who specialise in software and do write (very) good code
  - A long tail of 100s people with declining levels of experience in C++, right down to starting graduate students who barely know how to get started
    - But who will accumulate experience over time — some will become experts
- We have to have programming models and patterns that allow non-experts to contribute effectively
  - We have invested a lot recently in re-tooling in ATLAS to move to a more standard open source development model: git, GitLab, CMake
  - Code review and continuous integration are now critical parts of our workflow
  - Core and framework code must do the heavy lifting in areas like concurrency and vectorisation

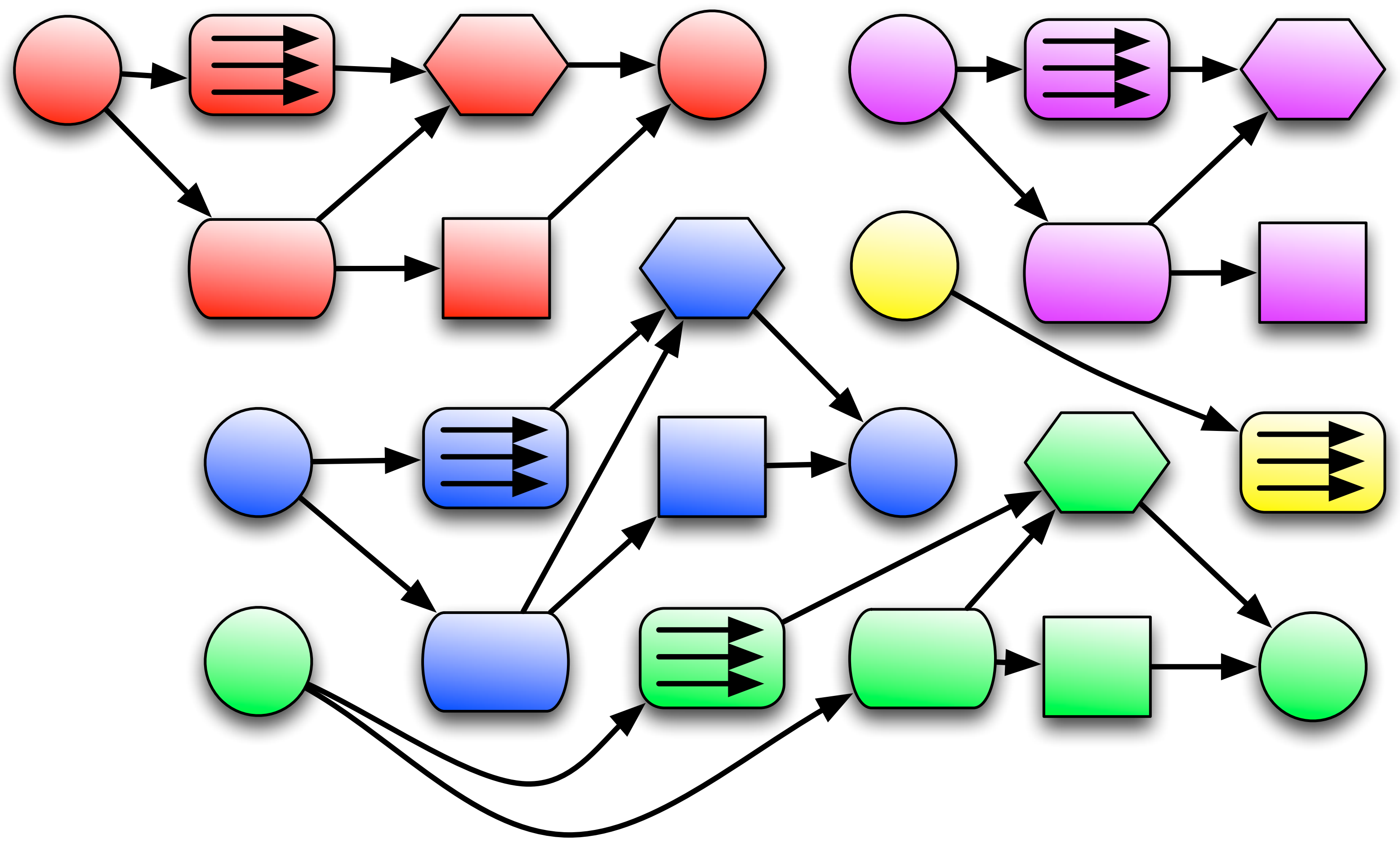
Memory Crisis

- Our highest ‘wall’ right now is the memory wall
  - We have ~100M detector channels, a complex geometry and complex magnetic field and we are supporting a precision physics program
  - All this is memory hungry work and we already have trouble squeezing into the memory/core limits on many of our grid sites (generally 2GB/*physical* core)
    - We’re throwing away 15-20% by not being able to use hyper threaded cores
  - We are surviving today for LHC Run 2 by using multi-processing, AthenaMP
    - Initialise large static memory structures and then fork multiple event workers
      - Takes advantage of the kernel’s copy on write to share a lot of memory
  - However, this technique is already sub-optimal
    - It practically fails already for some workflows, e.g., Heavy Ion reprocessing
    - We are not really able to use machines with memory/core < 2GB
      - Many core machines or weak core systems

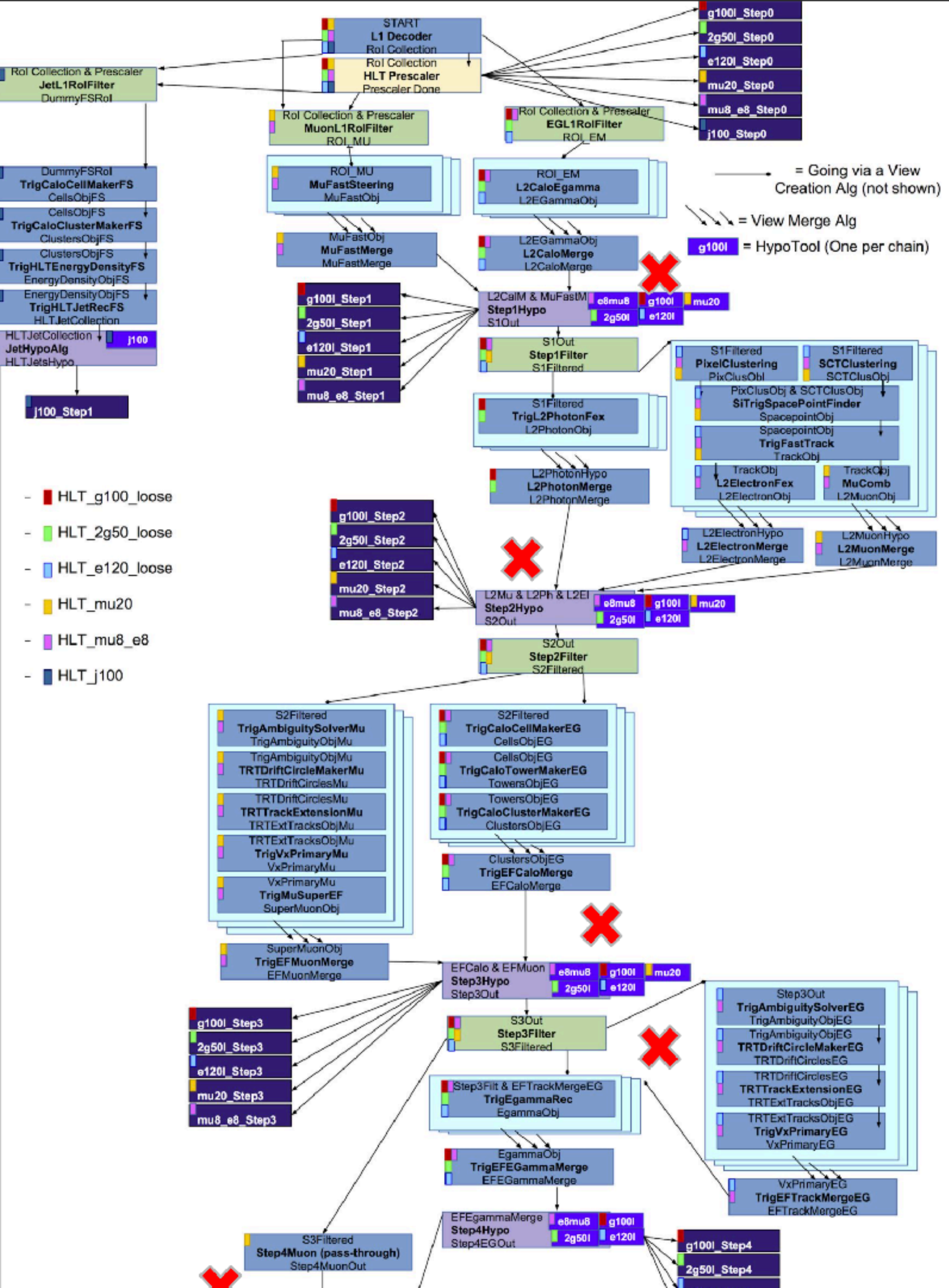


Framework Upgrade

- We have a major project now to upgrade to a multithreaded version of our framework
  - This is called AthenaMT and is based on an evolution of the Gaudi framework that we share with LHCb
- The intention here is to have a framework which is primarily data driven
  - We exploit the fact that our data processing can be broken down
    - Into events that are independent
    - With parallelism between reconstruction algorithms possible
    - We allow for the possibility of exploiting some parallelism within expensive algorithms
- Although we call this our multithreaded upgrade, in fact we express the workflow as a set of tasks and use a task based scheduler that manages the thread pool
  - Currently this is Intel's Threaded Building Blocks

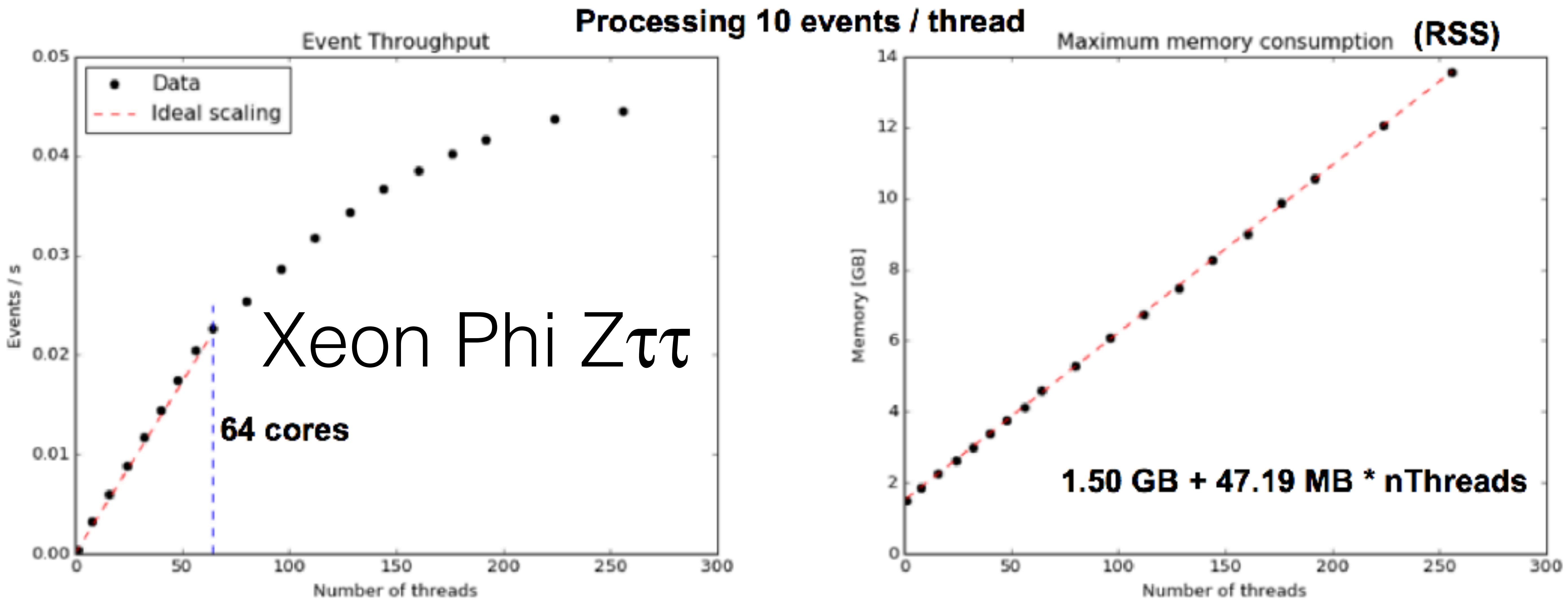


Roughly, view each row as a thread, each colour as an event, each box as an event processing step



AthenaMT: Complexity and Early Results

- Note that our scheduling problem is highly non-trivial
  - Many control flow dependencies to support early rejection of events online
- Results from running simulation in the new framework very much vindicate our approach



Software Tools

- Much of our code was written with deep assumptions about serial processing embedded in it
- A lot will have to be re-written to become compatible with the new framework
    - This is a large undertaking from a community already supporting ongoing data taking and physics analysis
- Good software tools already help (but always room for improvements):
    - Compilers (gcc, clang)
    - Static code checkers (Covertity, cppcheck, ASAN, UBSAN, …)
    - Performant libraries (MKL, Eigen)
- Areas where things feel weak:
    - Performance analysis — partly hampered by the size and complexity of our code base, these require a lot of investment and can be hard to map to code improvements
        - We really struggle to understand how to improve data flow in and out of memory to best use CPU caches
    - Refactoring tools — would be very useful for non-trivial API changes
    - Vectorisation — we have not found a way to vectorise our code in a way that's generally accessible to non-experts and portable across the code base
        - Our Event Data Model probably does us no favours here, but this is one of things that non-experts are very exposed to

Technology Outlook

- Slow death of Moore's Law
    - We will need to invest more in making the best use of the hardware that is on the market
    - We are absolutely COTS and we do not drive the market in any way
        - Does the hardware that will be available map well to our tasks?
            - Low power, many core systems
        - If not, how do we adapt to use what we can?
            - Except for a few specialist areas we are *very* far away from being able to use GPGPUs for most ATLAS data processing
    - We need good software tools to help here
        - We will never reach 'peak' efficiency, but which gaps are easier to bridge?
- Storage requirements are steeply rising with HL-LHC
    - Disk capacities increasing, but not i/o rates, which is a very serious issue
    - Tape market looks shaky and hard to see what we could replace it with cost effectively
- And we will not have any large budget increases to support our computing

Summary

- Understanding our social coding environment is critical
  - We need to make best use of our developer community, understanding its limitations
- Improvements should come in a semi-automatic way
  - Improved optimisations
  - Redesign data layouts (engaging with the experts but not bringing hurdles for others)
- On concurrency, we have a plan that we are confident in
  - It already commits our developers to a significant amount of work in the next years
- Next software challenges
  - Vectorisation
  - Data flow optimisation
- The supporting technology for distributed computing is also critical: storage and networking