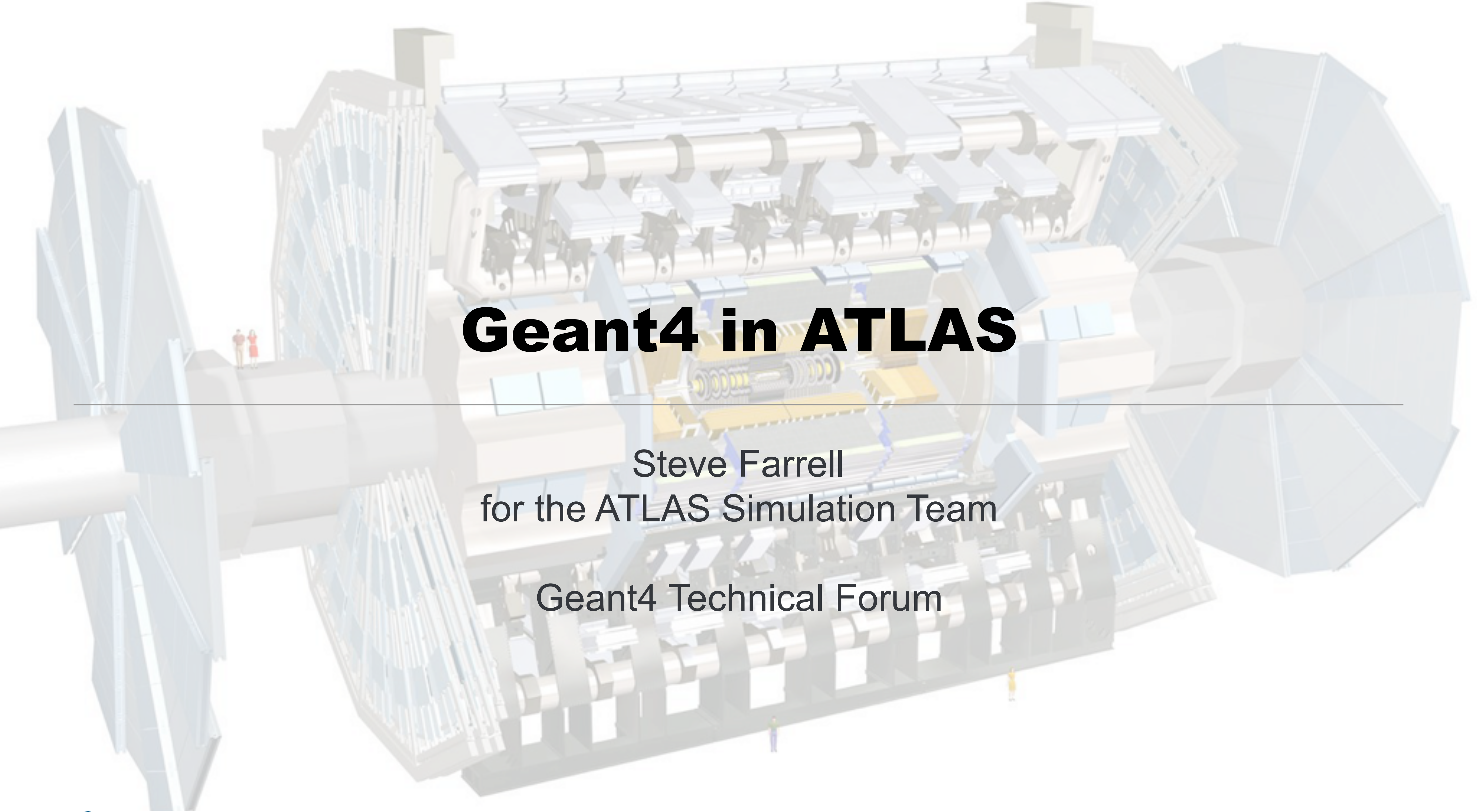


Geant4 in ATLAS

Steve Farrell
for the ATLAS Simulation Team

Geant4 Technical Forum

Dec 9, 2015

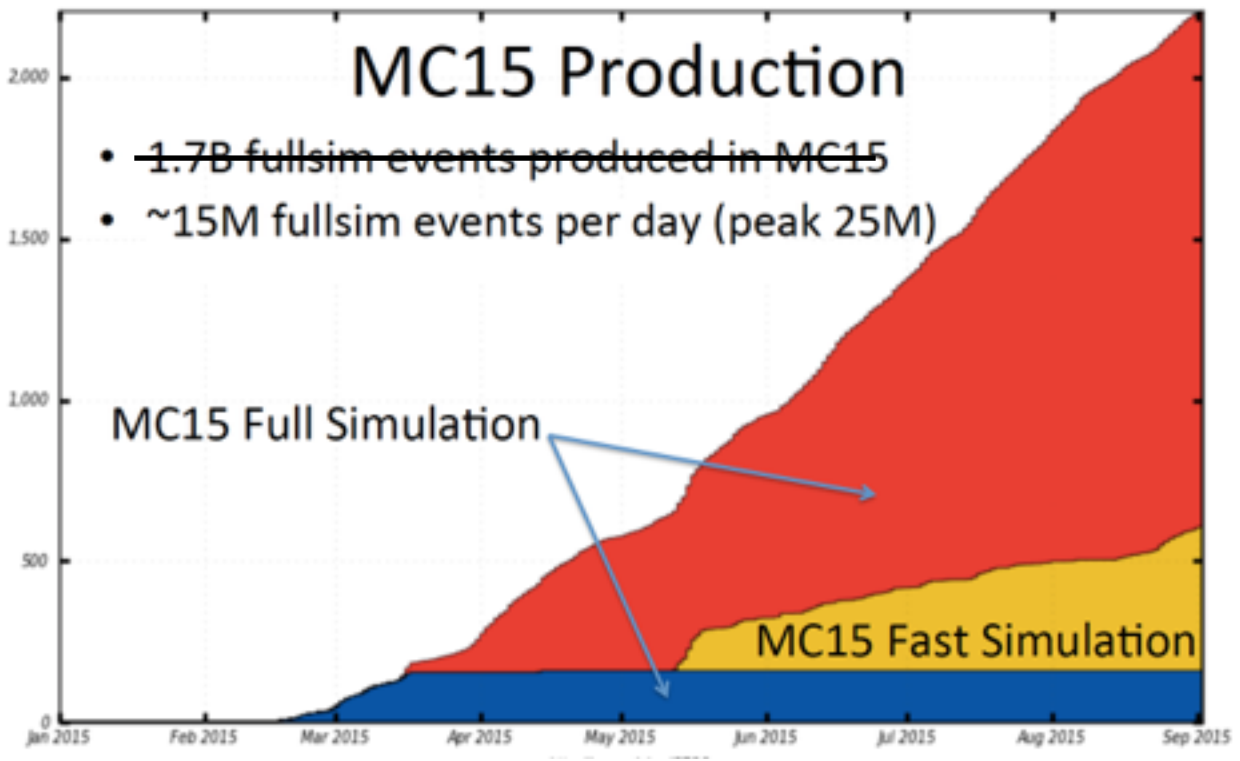
Monte Carlo productions

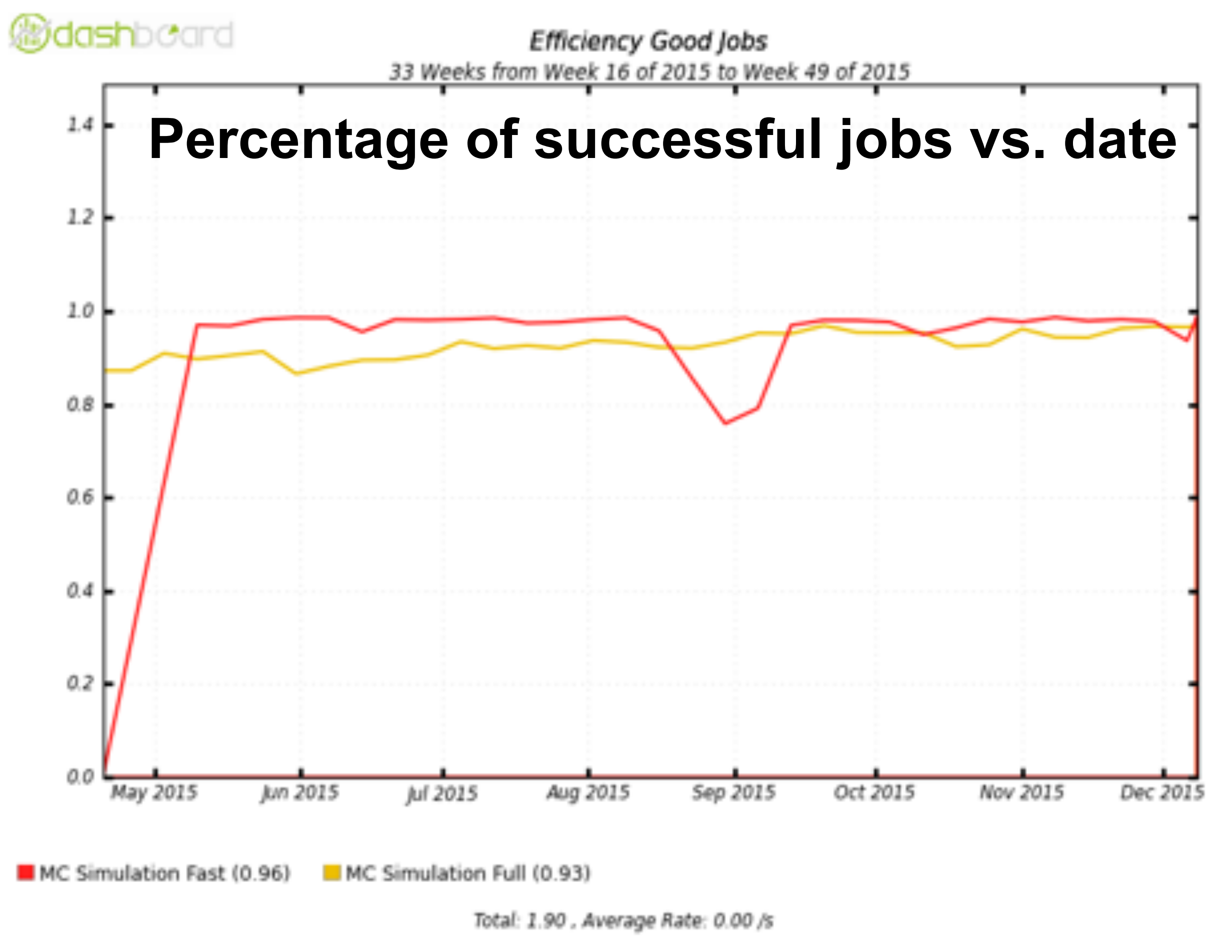
- **MC12: old Run-1 production tails**
  - Geant4 9.4 + patches
- **MC15: 13 TeV MC for the 2015 run**
  - Geant4 9.6 patch03 (validating patch04), CLHEP 2.1, 64-bit, gcc 4.7, SLC6, C++11
  - Current platform through mid-2016
  - Enormous number of ATLAS-specific updates
    - geometry and detector response
    - several speed-ups
  - Moved to ISF infrastructure by default
  - Running real production on HPCs with AthenaMP
    - ~few*10^7 events simulated
    - also validating clouds, BOINC, etc.
- **MC16: future production for 2016**
  - Geant4 10.1, CLHEP 2.2, 64-bit, gcc 4.9, SLC6, C++14
  - Expecting to start in April 2016; main platform through 2016-2017
  - Still testing ICC, Clang, Mac OSX builds (no production plans yet)

Production statistics

- MC15 production has been running pretty smoothly with high job efficiency
  - 96% of fast simulation jobs are “good”
  - 93% of full simulation jobs are “good”
- Events simulated
  - 3.2B full sim
  - 1.7B fast sim

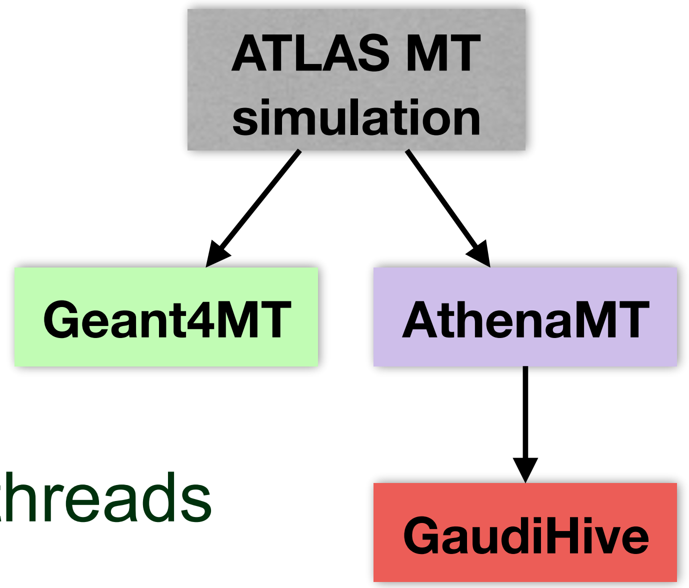
Inefficiencies include grid failures, not just Geant4

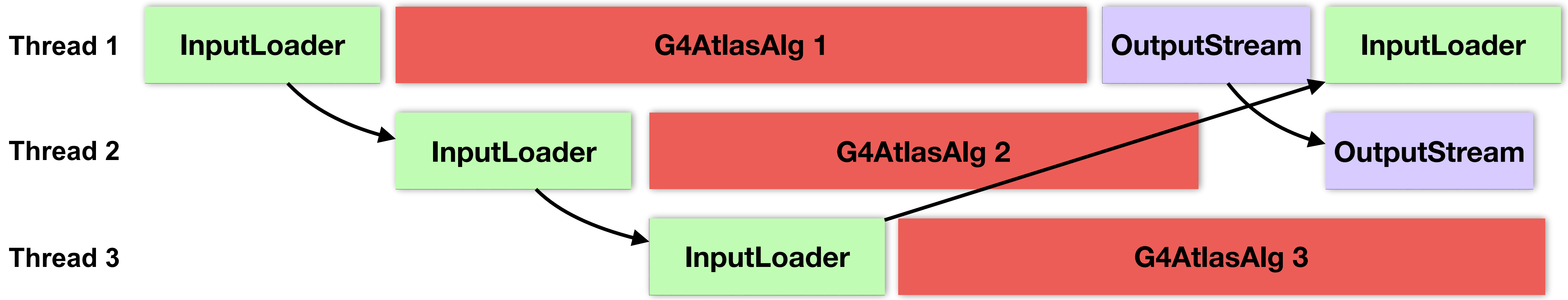




G4MT in ATLAS - introduction

- The ATLAS multi-threaded simulation framework: the marriage of Geant4MT with AthenaMT/GaudiHive
- Difficulties arise due to different choices of concurrency models and implementation details
  - Geant4: master-slave event parallelism via pthreads, with heavy use of TLS
  - GaudiHive: task-parallelism via TBB (including sub-event parallelism), no mapping of components to threads
- Parallelism
  - Event-level parallelism (so far)
  - The I/O algorithms are serialized, though, as shown below



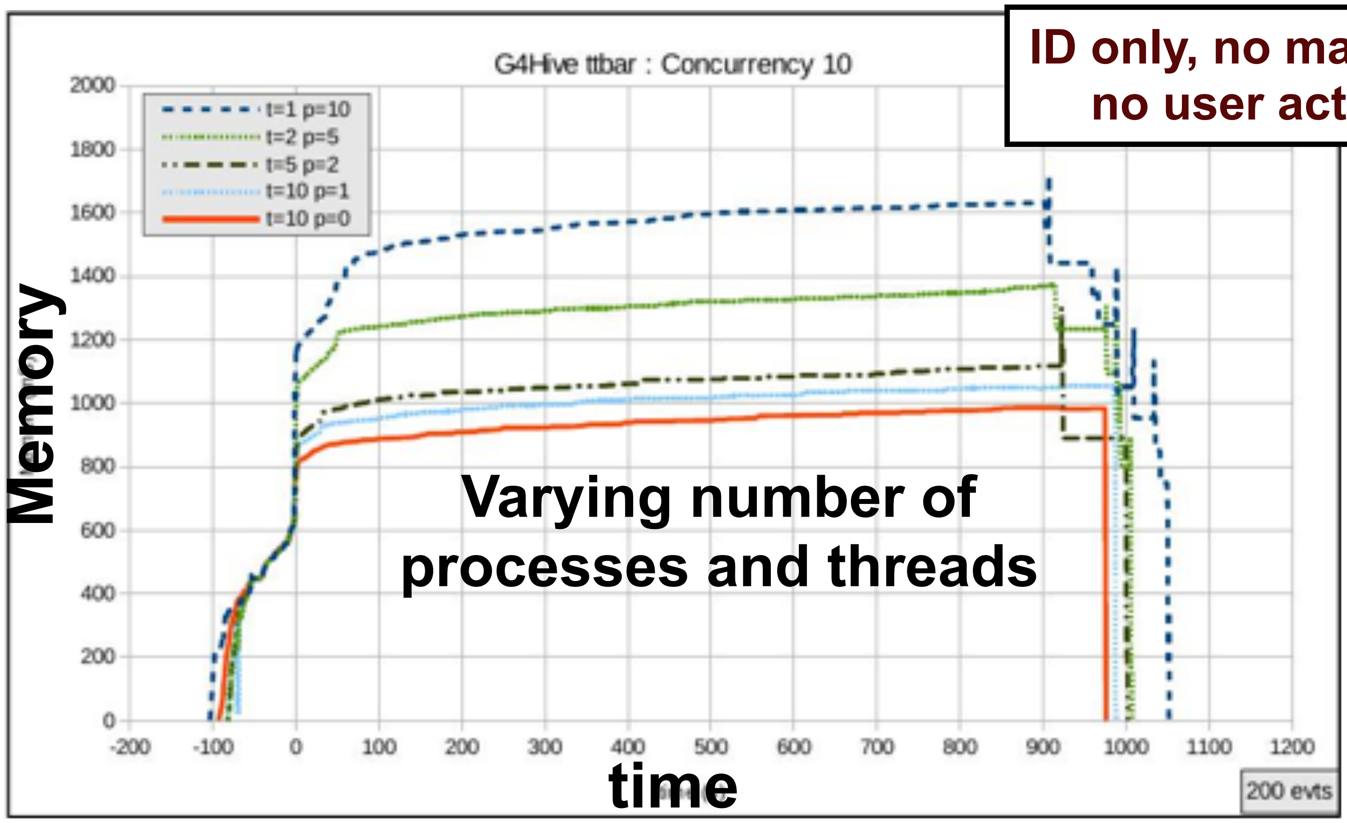


G4MT in ATLAS - implementation

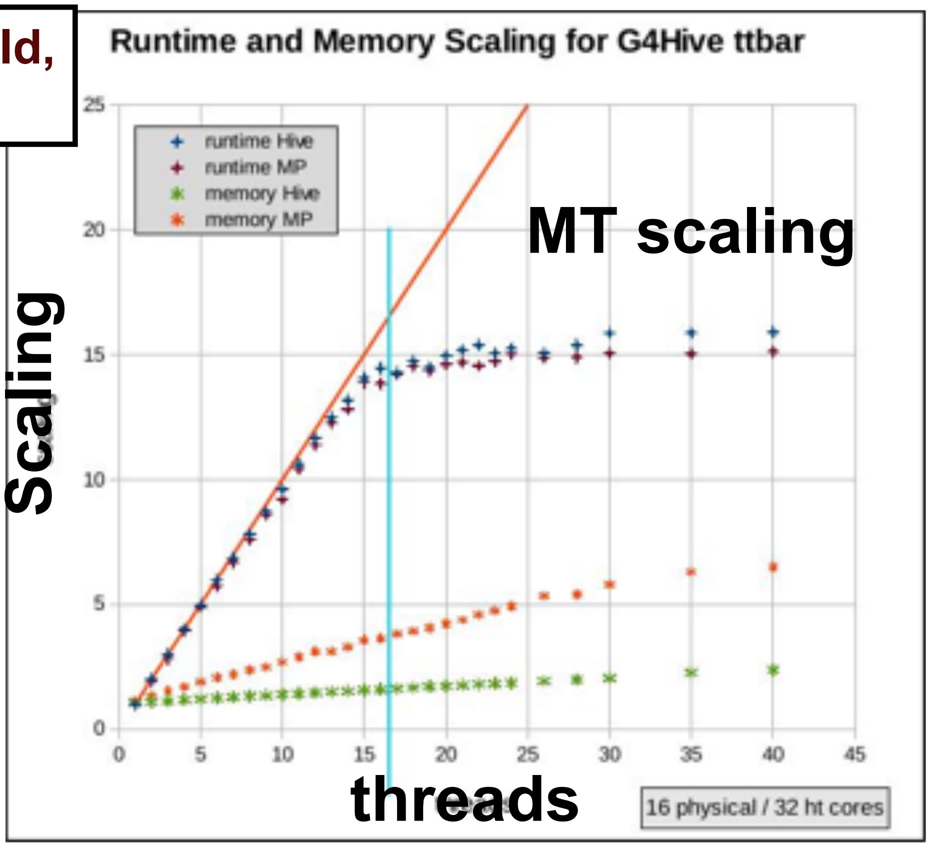
- Thread-local G4 objects are managed with framework components
  - Shared components (services) manage tools that are responsible for creating, configuring, and destroying the thread-local G4 components
  - This includes the sensitive detectors, user actions, etc.
- Thread-specific initialization required a new mechanism in the framework
  - Gaudi’s ThreadPoolSvc can have a thread-initializer tool which gets called concurrently on each thread with barrier synchronization
  - Used to mimic Geant4’s worker synchronization mechanism for setting up thread-local workspace and initialize worker run managers, etc.
- A lot of actual ATLAS-specific work is implemented via the user actions, but the design for MT is necessarily complicated
  - We need to be able to plug in multiple actions of each type
  - Many logical units need to be plugged in to multiple places (begin-event, post-tracking, end-run)

G4MT in ATLAS - status

- Migration to multi-threading is happening along with the simulation infrastructure migration from FADS to more Athena-friendly designs
- Sensitive detectors, geometry, physics lists, and algorithm code all working (so far) in MT
    - Numerous fixes for thread-safety and thread-friendly designs (e.g. custom EMEC geometry)
- User actions have a new MT-friendly design in place and are being migrated
    - The MCTruth-related action code is one of the more complicated beasts yet to be tamed
- Magnetic field is now thread-safe, though not yet fully tested (need to migrate field manager)
- We are able to run in a hybrid multi-threaded/multi-process framework mode
- Preliminary (yet old) performance measurements show promising results, good scalability



ID only, no mag field, no user actions



G4MT + ISF

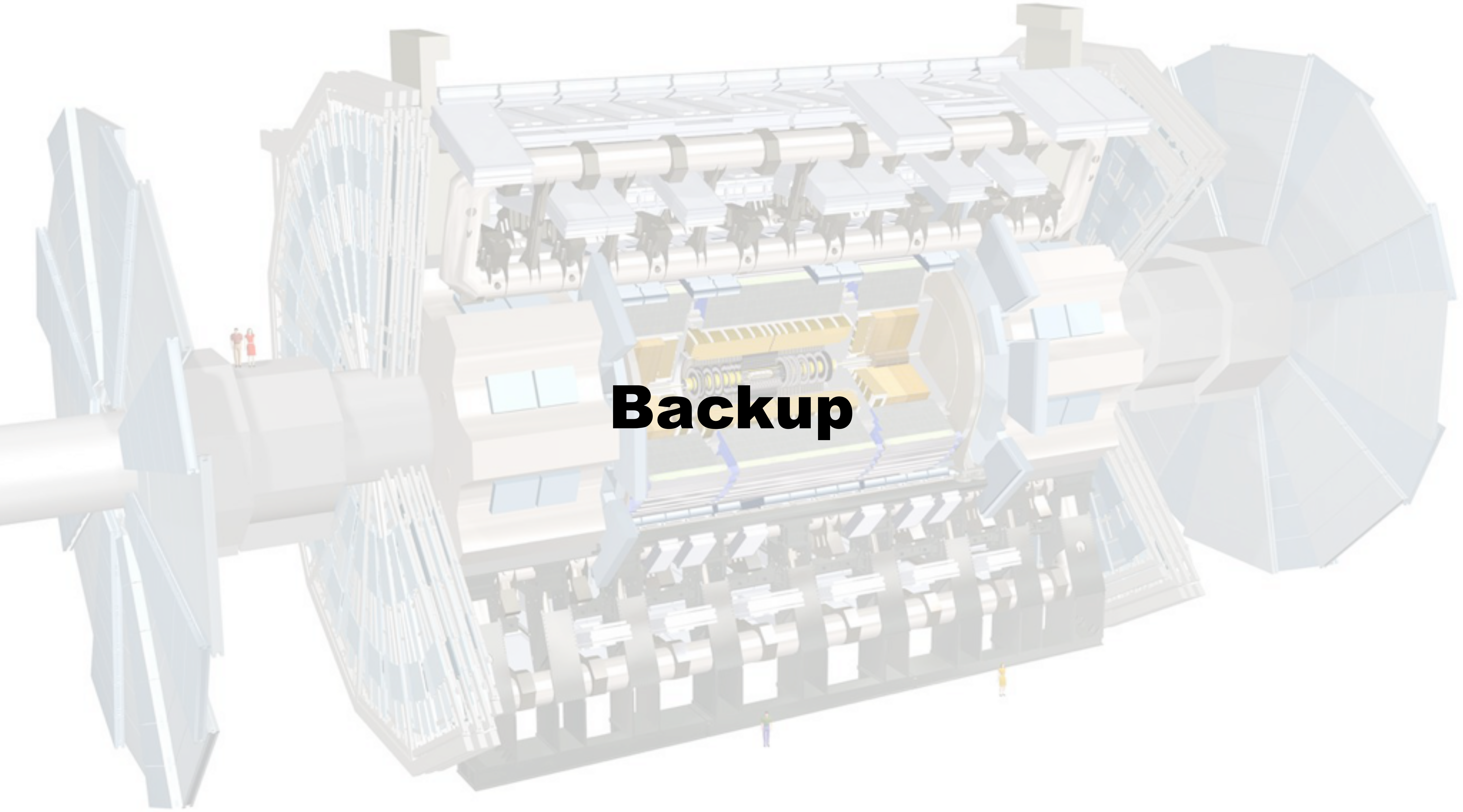
- For MT we have only been running the “old-style” simulation; *not* the ISF
    - Eventually, though, we’d like to just have one way of doing things (i.e., ISF)
- ISF does things a little differently which bring some new challenges for multi-threading
    - can do multiple G4 events per Athena event
    - uses customized brokering of particles to different simulation services (Geant4, FastCaloSim, FATRAS, etc.)
- Getting event-level parallelism working will take some thinking, but should be do-able
- Next question: can we tackle *sub-event-level parallelism* with the ISF?
    - opens yet another can of worms in terms of challenges, partially in dealing with particle containers and interacting with the framework
- Stay tuned for updates

Bugs and other issues

- The G4MultiLevelLocator issue
  - We think this is the origin of the “hyperspace bug”: particles entering a volume and then continuing for km. Originally thought to be due to G4PolyCone problems.
  - Also might be the origin of the tiny step problems: many steps that are order fm long. Still not understood. Does not appear to be ATLAS-specific, but also isn’t showing up in the simple Geant4 examples.
  - We might have this patched now; in the process of confirming the fix
- Slightly annoying field design issue discovered around the time of the last G4TF – any hope to resolve this?
  - Reminder: G4FieldManager and G4Stepper own copies of the field pointer, and even for steppers owned by specific managers these are not required to be in sync.
- Reproducibility with G4 10.1
  - We’re running tests with 1-5 processes, looking for differences in the events. No sign of differences seen yet, but saw some issues in production jobs that made us open this issue up again. Expect news by next G4 TF.

Bugs and other issues

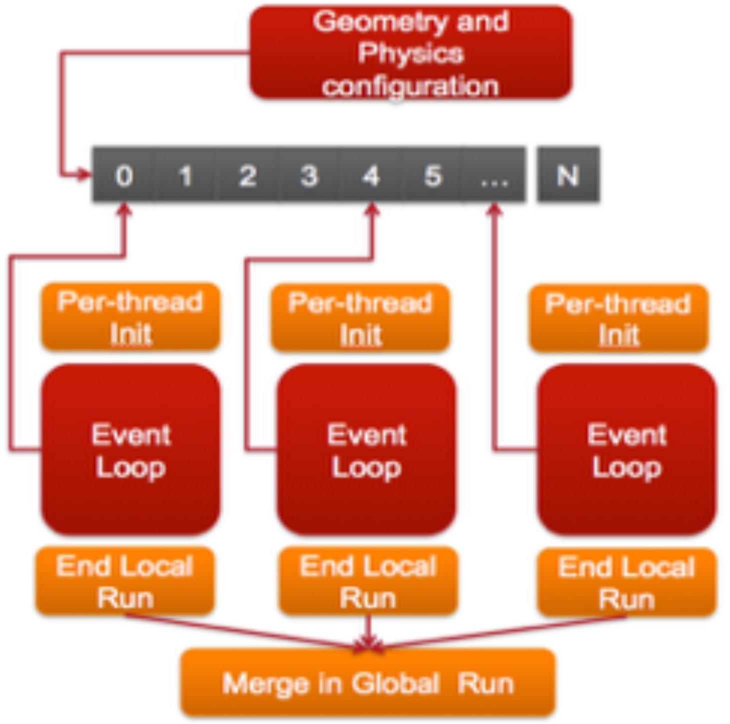
- We have many materials that are copies of each other with different names. Is there a way to reduce the memory consumption in this sort of situation?
  - Design choice, to be able to change the density of the materials in a region of the detector that we do not understand well; different materials allows this in principle, though we have not used the handle yet.
  - Starting to also look into non-uniform density materials, as our forward services have gotten rather complicated in run 2. Want to avoid simulating every cable.
- Can the hadronic cross section speed improvements (from ASCR analysis) be put into production soon? What about taking cross sections from a database instead of many small files?
- The multi-SD implementation went a long way in helping us reduce complexity in our SD code; could a similar thing be done for user actions?
- Any developments to make Geant4MT more friendly to task-based models like GaudiHive/TBB would be very welcome
  - Though we understand there are obstacles to making this work well



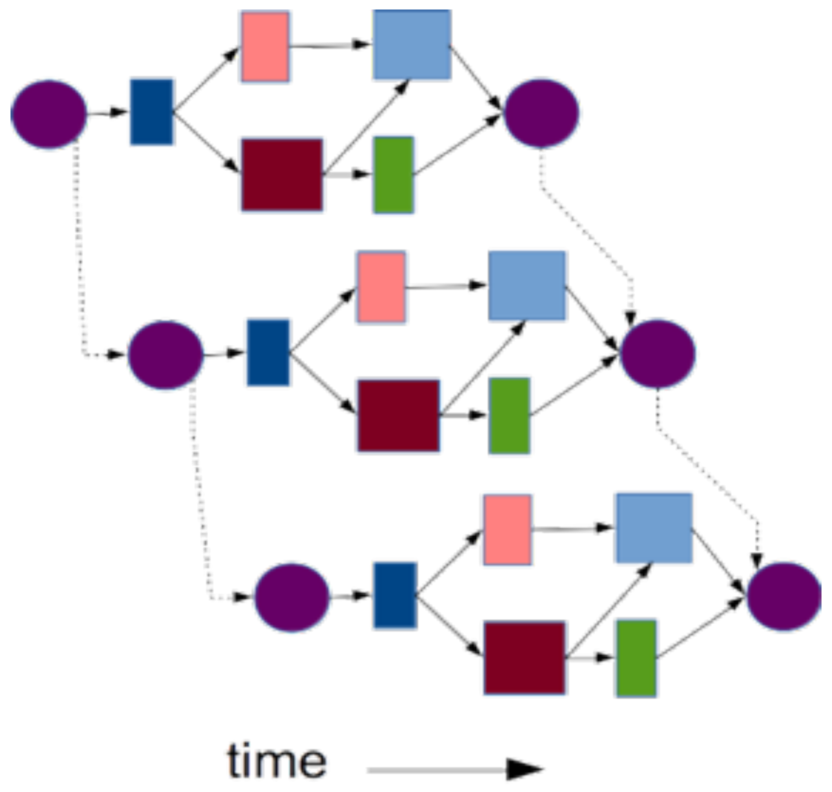
Backup

Geant4MT vs. GaudiHive

- Multi-threaded Geant4 released Dec 2013 (ver 10.0)
  - Minimal API changes
  - Lock-free event loop (good scaling)
- Master-slave concurrency model using pthreads behind the scenes
  - Event-level parallelism
  - Thread-local storage => thread safety + performant

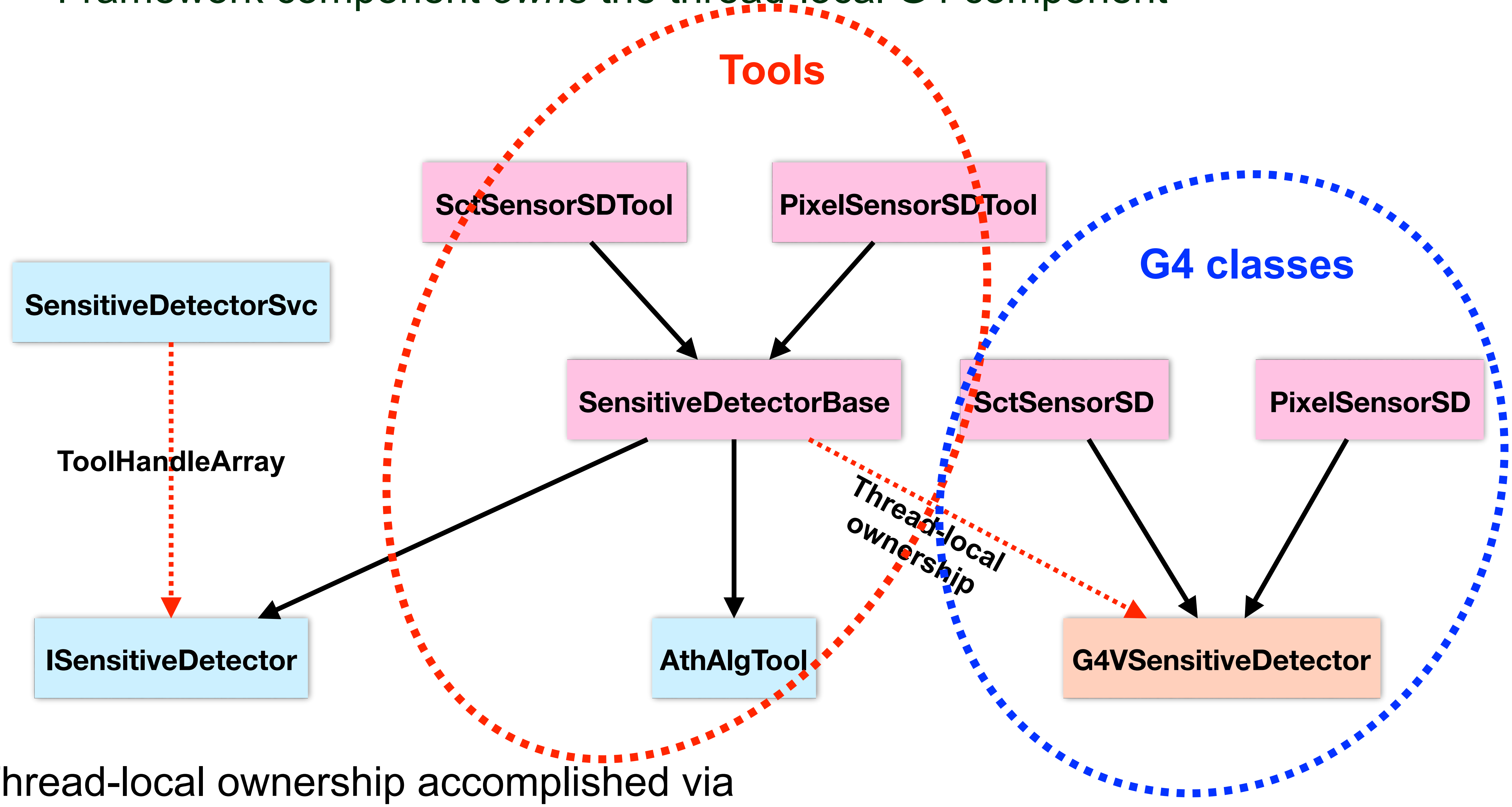


- GaudiHive summarized by Charles
- Task-based concurrency model built on top of Intel's TBB
  - Event-level parallelism
  - Algorithm-level parallelism
- Concurrency "knobs"
  - Number of threads
  - Number of events in flight
  - Cardinality of algorithms



Sensitive detectors

- Design for multi-threading
    - Framework component *owns* the thread-local G4 component



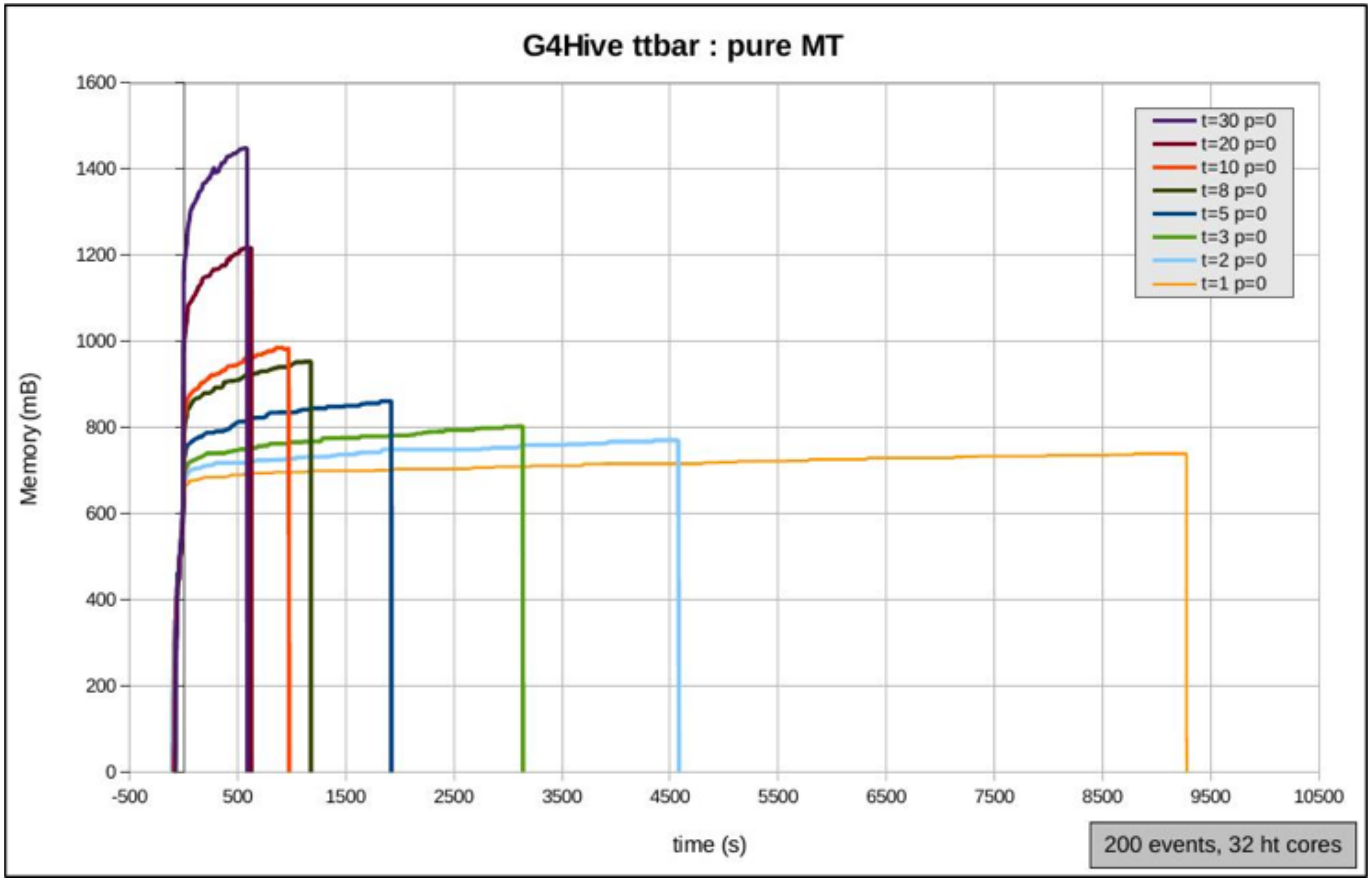
Thread-local ownership accomplished via tbb::concurrent_unordered_map of TID -> object

Use cases for user actions

- -) Truth (big one)
- -) Stacking only interesting particles
- -) Debugging (check mode, detailed timing, volume testing, verbose switching part-way through events)
- -) Printing information (memory, time, tracks)
- -) Saving perigees of Geant4 particles
- -) Interaction length and radiation length plots
- -) Testing (energy/momentum conservation, hyperspace tests, looper killer)
- -) Killing particles entering the calorimeter (for AF2)
- -) Recording particle fluxes / scoring
- -) Dumping Geant4 step information to load into event displays
- -) Highly-ionizing particle / stopped exotic particle killing
- -) Wrapping hits in time for cavern background
- -) Killing super-low energy photons

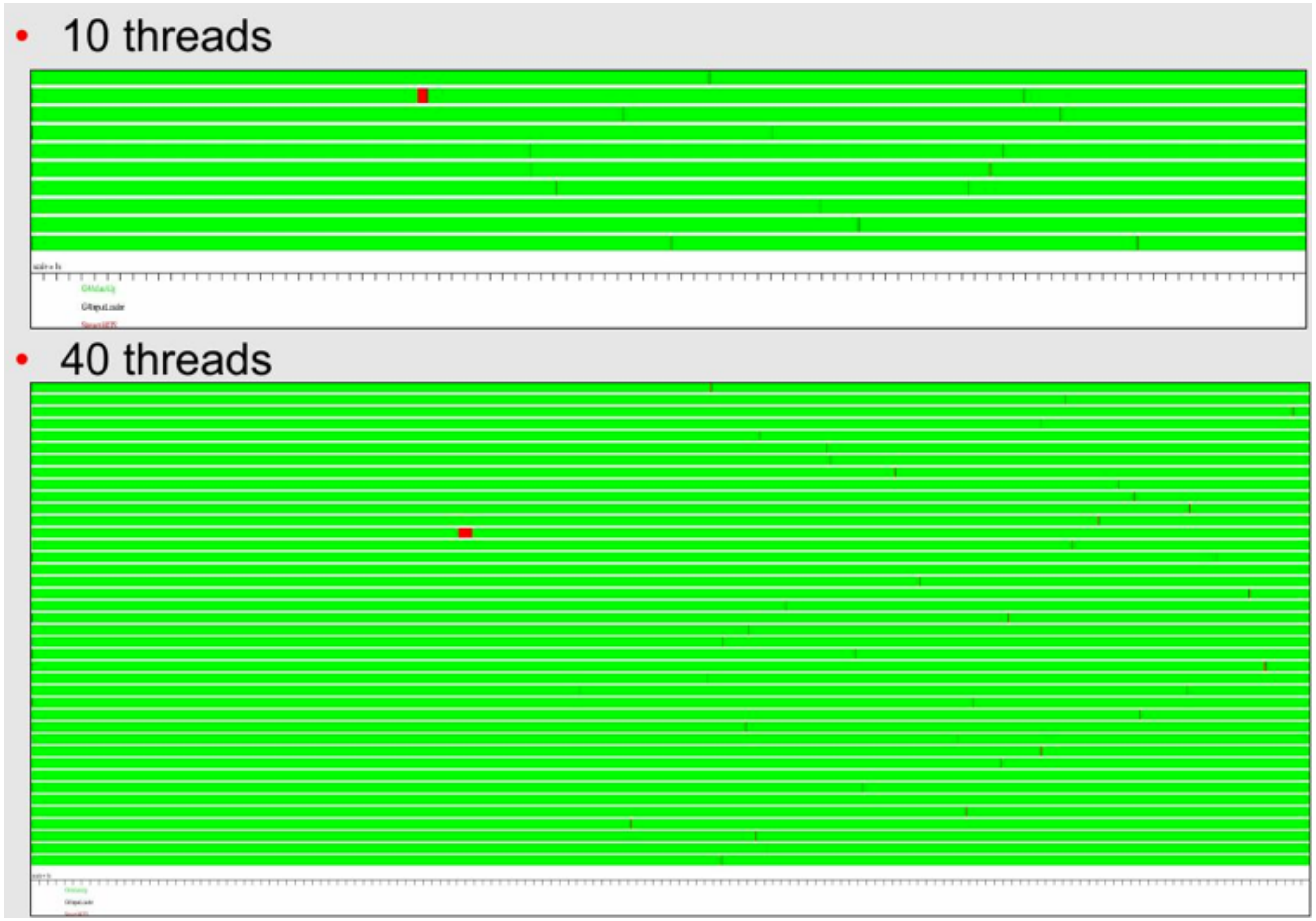
Performance of pure Hive

- Memory profile vs. runtime for pure Hive (no AthenaMP)
  - 200 ttbar events



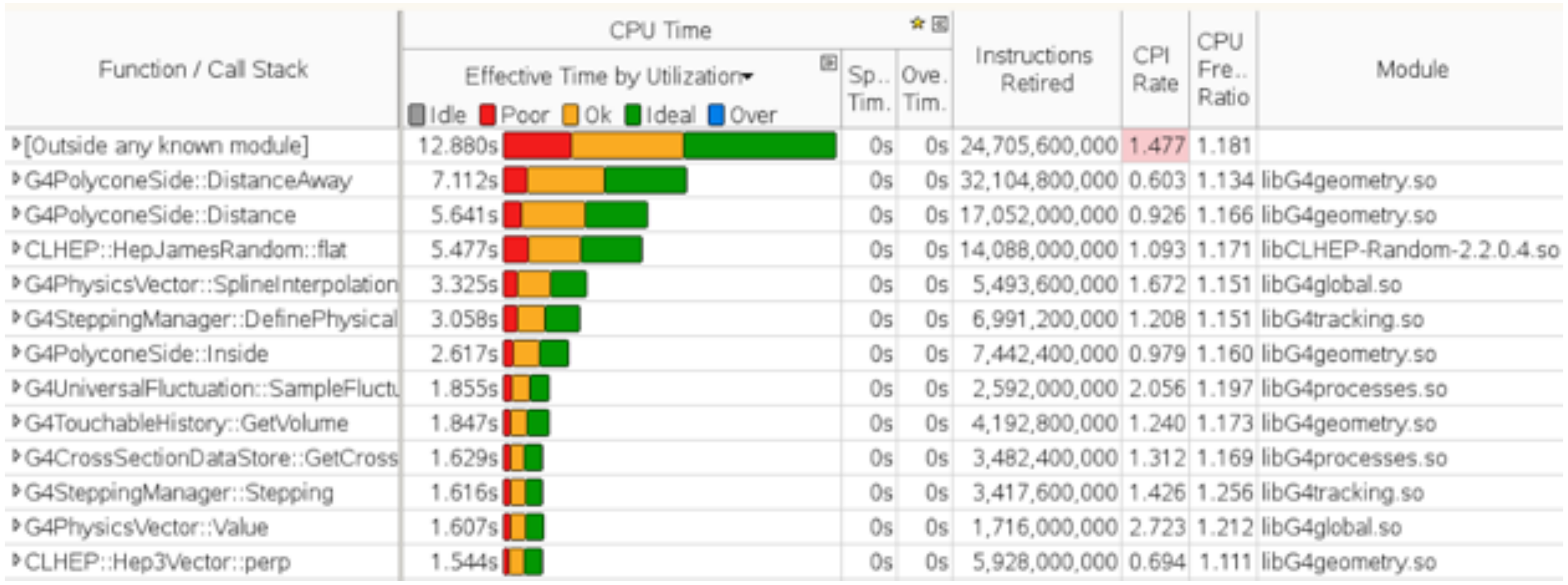
G4Hive thread occupancy

- Threads are beautifully full of simulation work
  - Heavy CPU use-case, that's why it's a good first candidate for study!



G4Hive performance studies

- NERSC summer student Mirella Da Silva studied performance of G4Hive on Edison in preparation for an Intel Dungeon session with VTune
- Hotspots in geometry and random number generation:



Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	CPU Time: Sp.. Tim.	CPU Time: Ove. Tim.	Instructions Retired	CPI Rate	CPU Fre.. Ratio	Module
[Outside any known module]	12.880s	0s	0s	24,705,600,000	1.477	1.181	
G4PolyconeSide::DistanceAway	7.112s	0s	0s	32,104,800,000	0.603	1.134	libG4geometry.so
G4PolyconeSide::Distance	5.641s	0s	0s	17,052,000,000	0.926	1.166	libG4geometry.so
CLHEP::HepJamesRandom::flat	5.477s	0s	0s	14,088,000,000	1.093	1.171	libCLHEP-Random-2.2.0.4.so
G4PhysicsVector::SplineInterpolation	3.325s	0s	0s	5,493,600,000	1.672	1.151	libG4global.so
G4SteppingManager::DefinePhysical	3.058s	0s	0s	6,991,200,000	1.208	1.151	libG4tracking.so
G4PolyconeSide::Inside	2.617s	0s	0s	7,442,400,000	0.979	1.160	libG4geometry.so
G4UniversalFluctuation::SampleFluctu	1.855s	0s	0s	2,592,000,000	2.056	1.197	libG4processes.so
G4TouchableHistory::GetVolume	1.847s	0s	0s	4,192,800,000	1.240	1.173	libG4geometry.so
G4CrossSectionDataStore::GetCross	1.629s	0s	0s	3,482,400,000	1.312	1.169	libG4processes.so
G4SteppingManager::Stepping	1.616s	0s	0s	3,417,600,000	1.426	1.256	libG4tracking.so
G4PhysicsVector::Value	1.607s	0s	0s	1,716,000,000	2.723	1.212	libG4global.so
CLHEP::Hep3Vector::perp	1.544s	0s	0s	5,928,000,000	0.694	1.111	libG4geometry.so

- Some particular results
  - High CPI rate of 1.588
  - Back-end bound unfilled pipeline slots: >60% of total running time.
  - Front-end latency: 16%. Large number of instruction cache misses.
  - Not much tradeoff between # processes and # threads
  - Not much improvement from hyper threading

We're still digesting the results