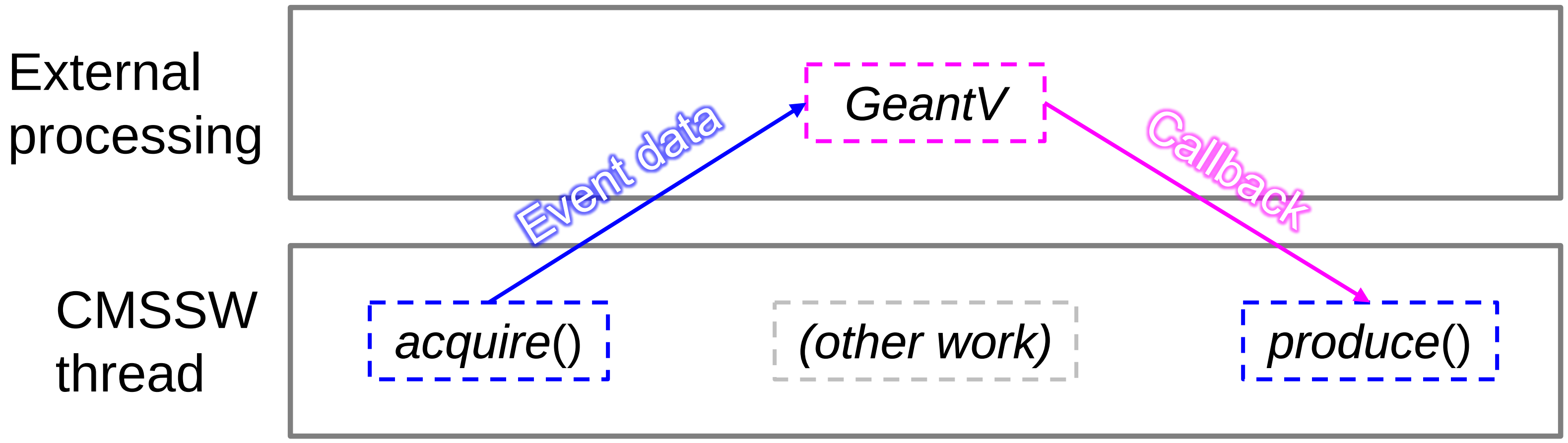
Status of GeantV Integration in CMSSW

Kevin Pedro, Sunanda Banerjee
(FNAL)

September 13, 2019

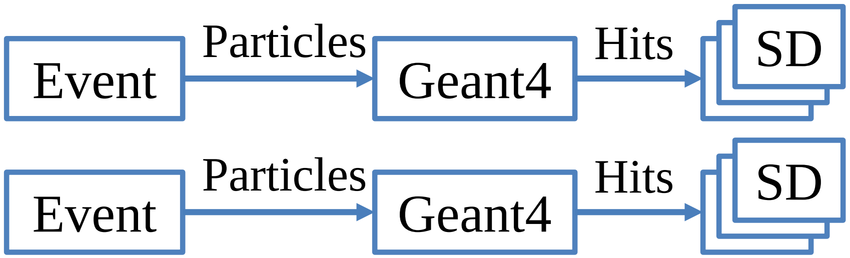
GeantV Integration in CMSSW

- Repositories: install-geant, SimGVCore
- ✓ Generate events in CMSSW framework, convert HepMC to GeantV format
- ✓ Build CMSSW geometry natively and pass to GeantV engine (using TGeo)
- Using constant magnetic field, limited EM-only physics list
- ✓ Calorimeter scoring adapted
- ✓ Run GeantV using CMSSW ExternalWork feature:
  - Asynchronous, non-blocking, task-based processing

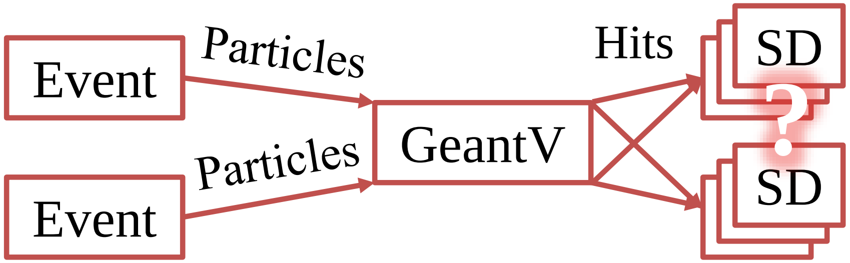


Geant4 vs. GeantV Scoring

- Sensitive detectors (SD) and scoring trickiest to adapt
  - Necessary to test “full chain” (simulation → digitization → reconstruction)
  - Significantly more complicated than Geant4 MT



Geant4 shares memory, but each event processed in separate thread

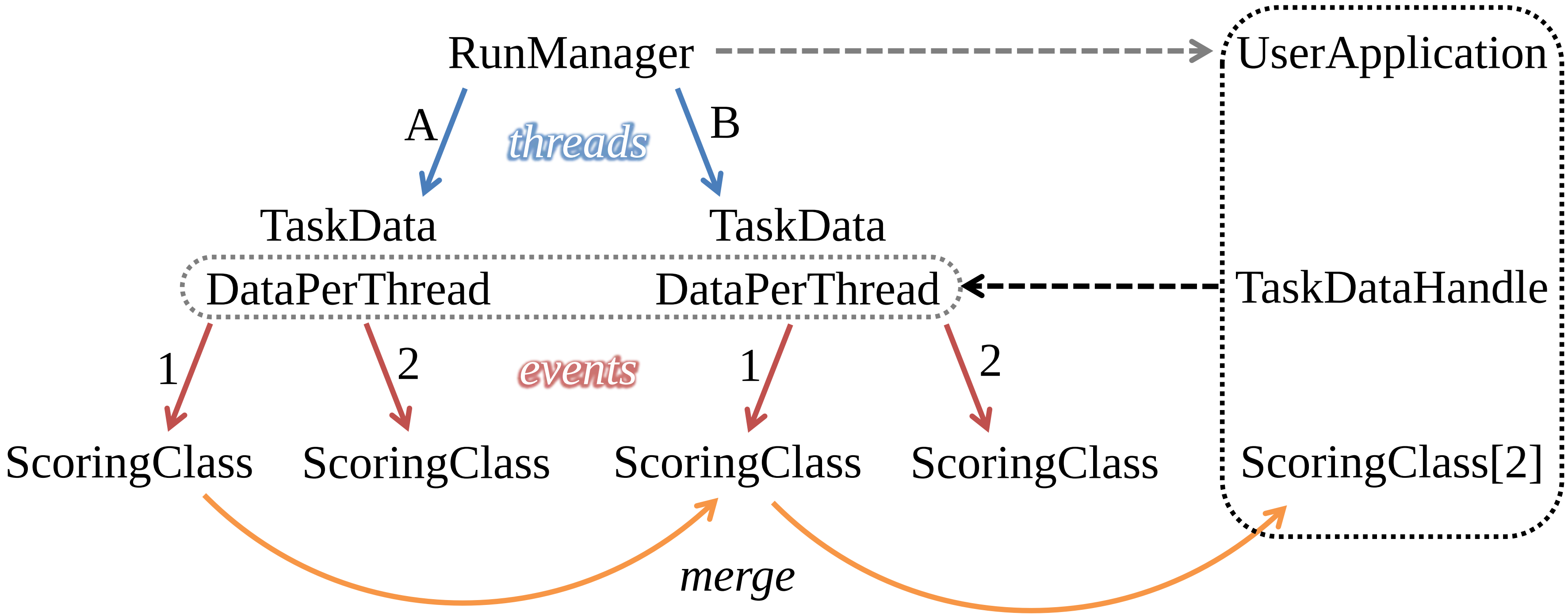


Each event processed in multiple threads, mixed in with other events

- Duplicate SD objects per event per thread, then aggregate → 4 streams, 4 threads = 16 SD objects
  - GeantV TaskData supports this approach

➢ Use template wrappers to unify interfaces and operations
  - Avoid upsetting delicate and complicated SD code, minimize overhead
  - See backup for more details

GeantV Data Aggregation



- Each ScoringClass object has instance of CaloSteppingAction
  - Some additional memory overhead from duplicated class members
- GeantV assigns slot number to each event
  - May not match stream number in CMSSW, keep track w/ StreamCache
- Merged ScoringClass object in UserApp puts output products into event

Testing GeantV in CMSSW

- Need to validate physics and measure CPU and memory performance
- Previously saw discrepancy in # hits (more in GV than G4)
- Investigated and understood:
  - All CMS-specific G4 optimizations disabled
  - Same production cuts (default 1mm)
  - Confirmed intra-simulation reproducibility in single-thread mode (run GV twice on same input, get same output)
  - Found slightly better agreement with magnetic field *disabled* → in single thread mode, but not multithread mode?
  - ➢ Fixed main culprit: data race in CMS Geant4 application (affected Watchers used for scoring demo , not sensitive detectors used in prod)
- Latest validation results and initial performance results follow

Physics Validation

- Generate 1000 events of single electrons at 100 GeV with a fixed direction ($\eta = 1.0$, $\varphi = 1.1$)

1. Run Geant4 and GeantV setup on single thread with same input file, B = 0 and compare GeantV against Geant4

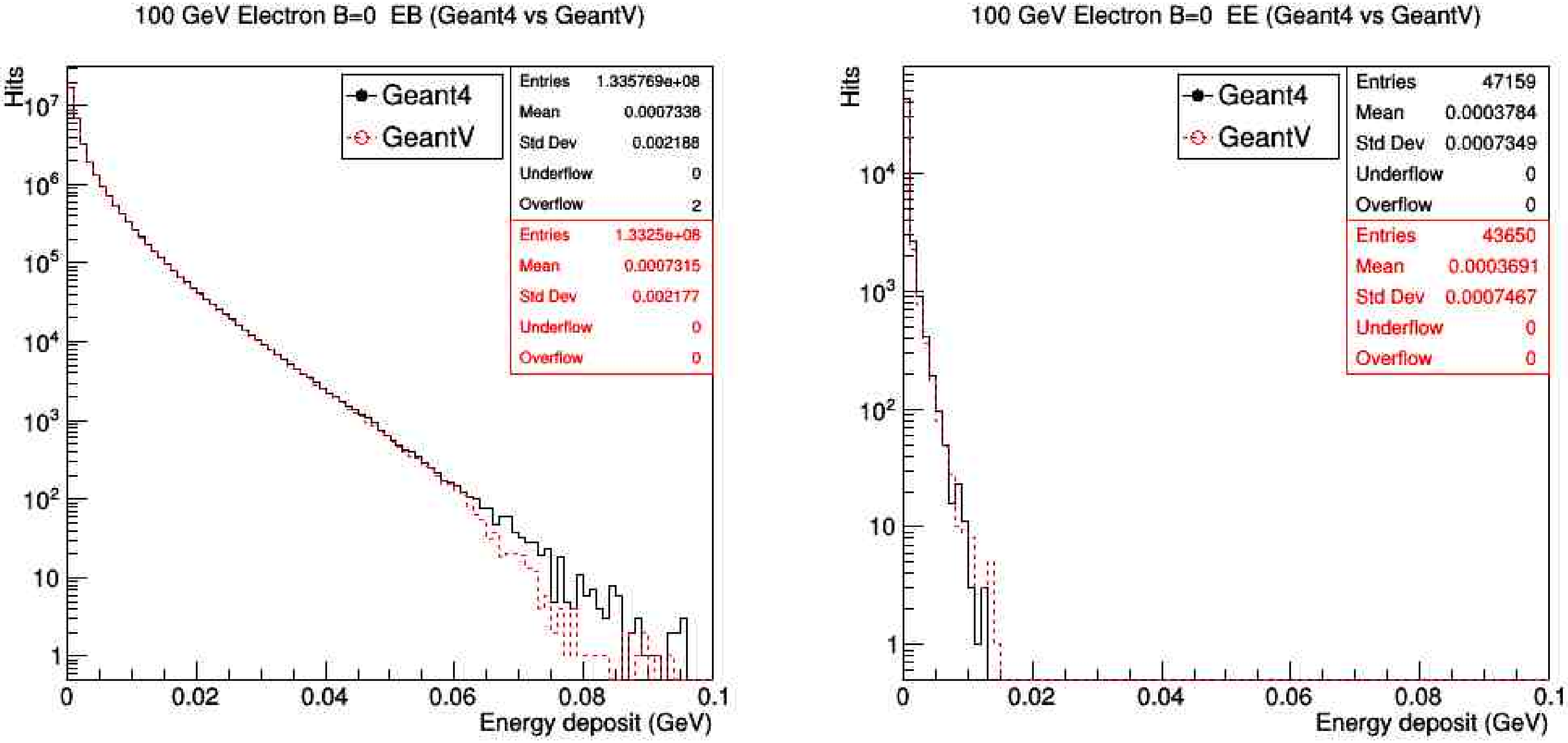
2. Compare GeantV against Geant4 for 100 GeV electrons with B = 3.8 Tesla

3. Generate 1000 events of single electrons at 2, 10 and 50 GeV at a fixed direction and compare GeantV against Geant4 with magnetic field off and on at 3.8 Tesla

4. Generate 100 events of 50 GeV double electrons at 50 GeV with $-3 < \eta < 3$ and $0 < \varphi < 2\pi$, run in multi-threaded mode (4 threads), B = 0 Tesla

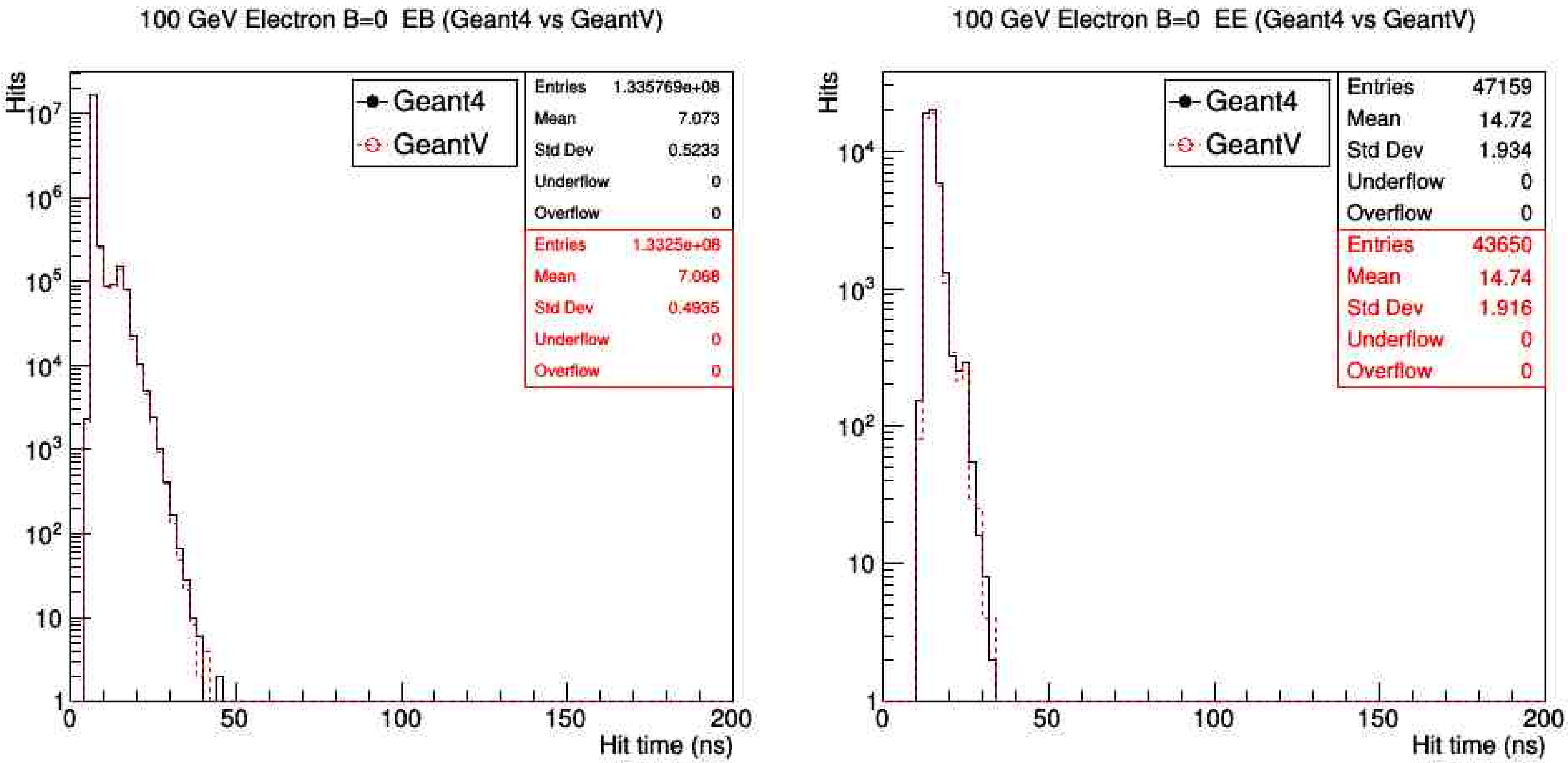
5. Repeat multi-threaded test with B = 3.8 Tesla

1. Energy Deposits for 100 GeV e- (B=0)



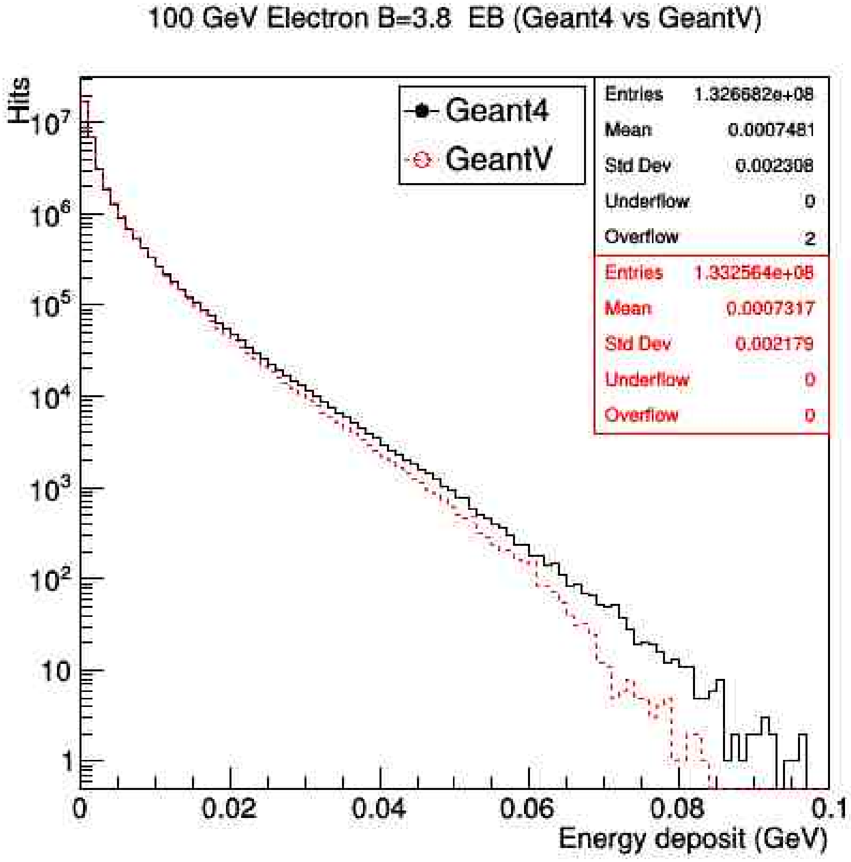
- The number of entries differ by 0.3% (7.4%) in EB (EE) with the electrons going in the barrel
- The means differ by 0.2% for EB and 2.5% for EE

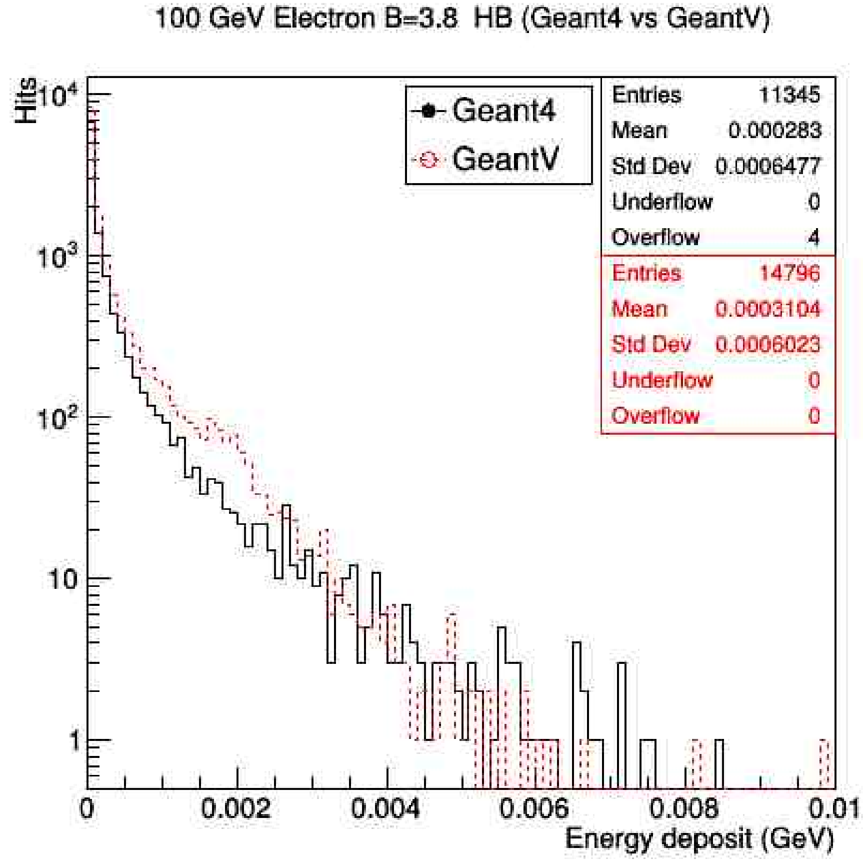
1. Hit Time for 100 GeV e- (B=0)



- Means differ by 0.07% for EB and 0.13% for EE with the electrons going in the barrel
- GeantV and Geant4 applications provide roughly the same distributions

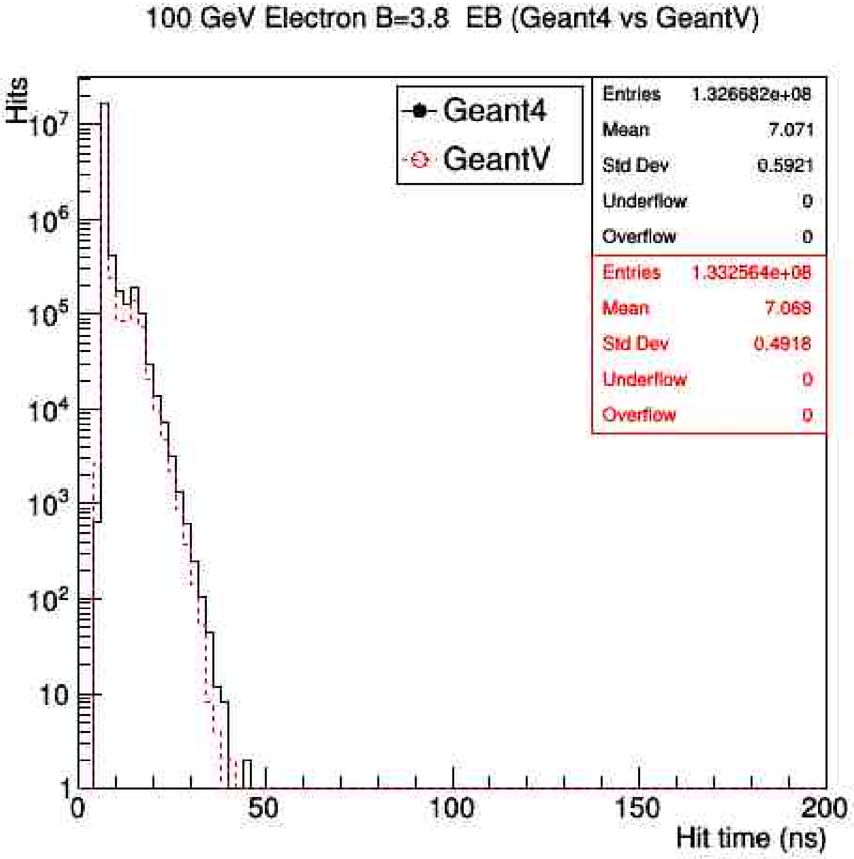
2. Energy Deposits for 100 GeV e- (B=3.8)

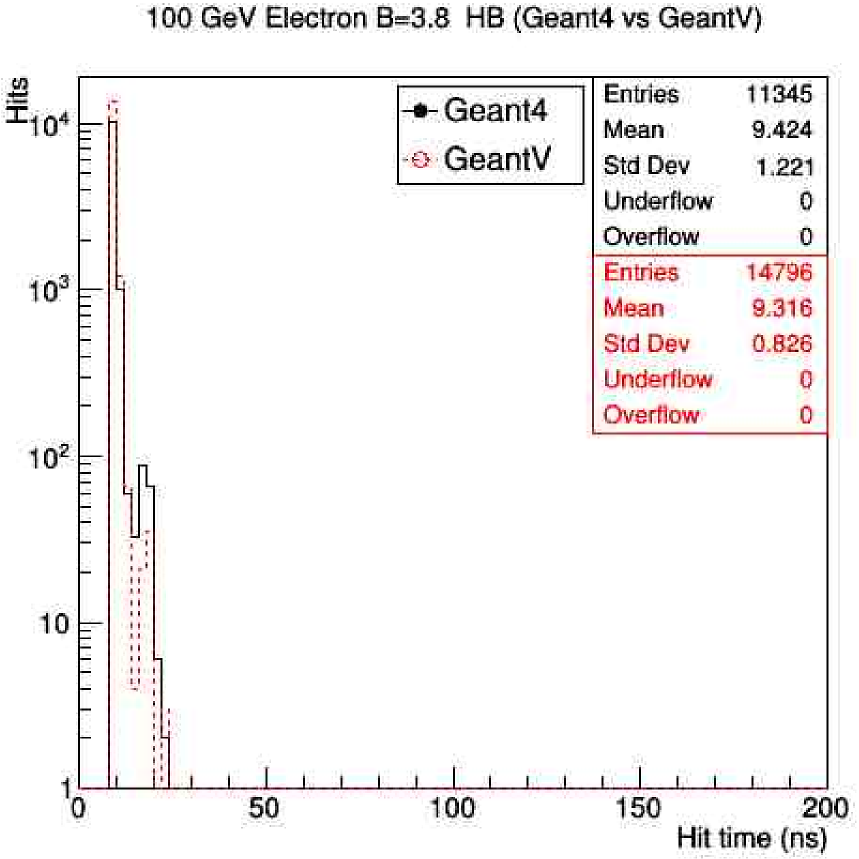




- The number of entries differ by 0.4% (23.3%) in EB (HB) with the electrons going in the barrel
- The means differ by 2.2% for EB and 8.8% for HB

2. Hit Time for 100 GeV e- (B=3.8)

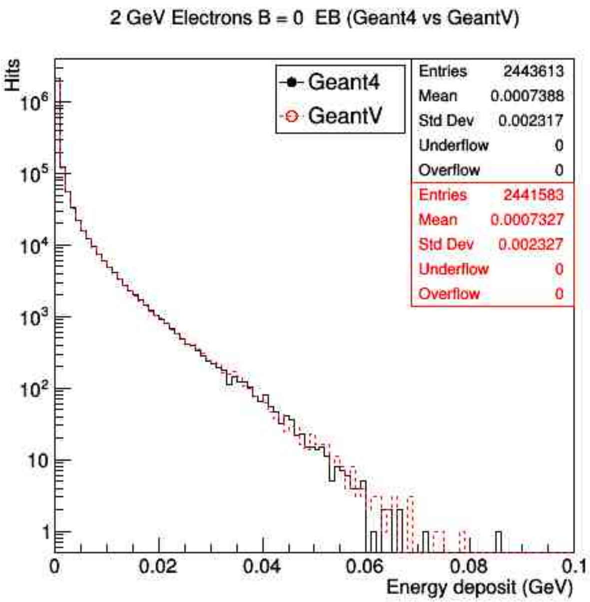




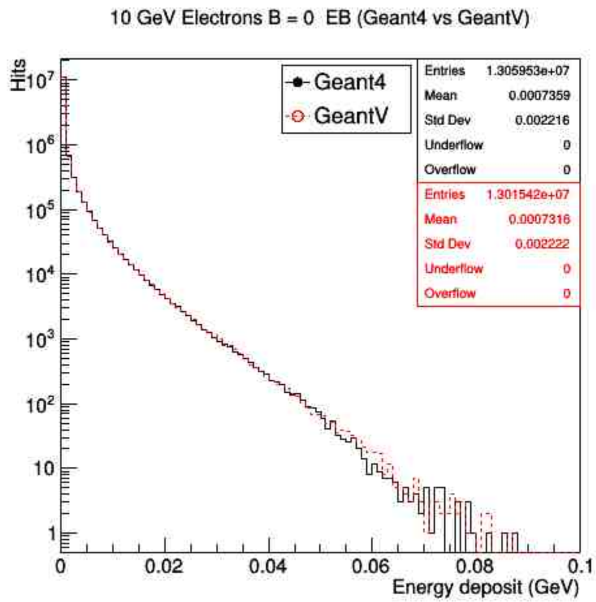
- The means differ by 0.03% for EB and 1.15% for EE with the electrons going in the barrel
- There is a small difference in the physics results of GeantV and Geant4 applications in the presence of B-field

3. Energy Deposit with B = 0

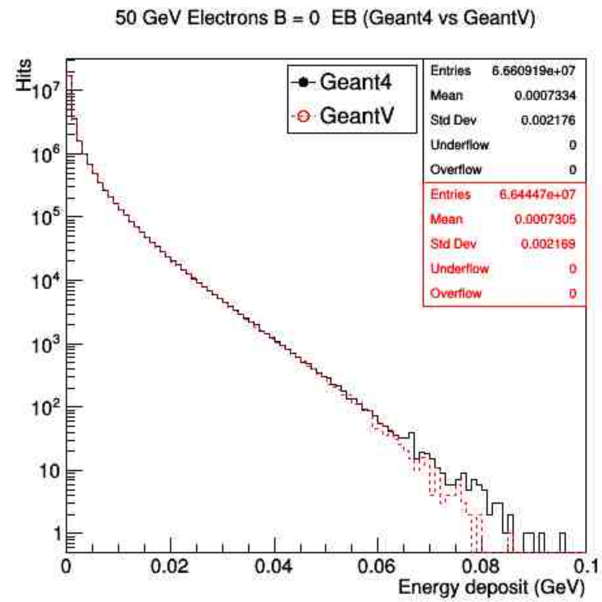
2 GeV Electrons



10 GeV Electrons



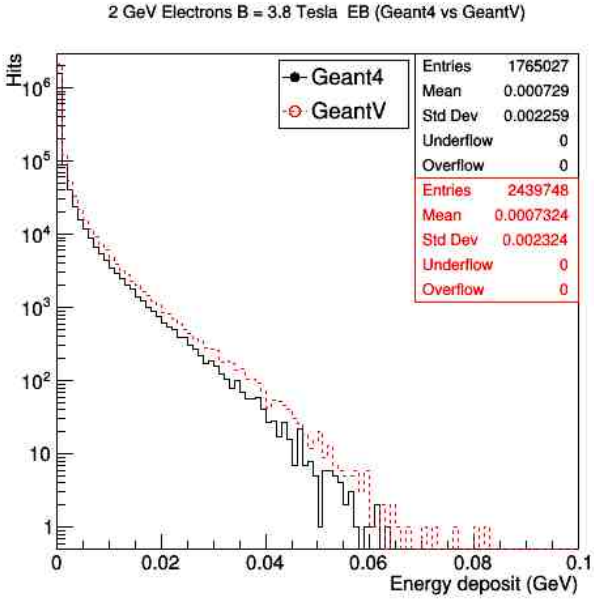
50 GeV Electrons



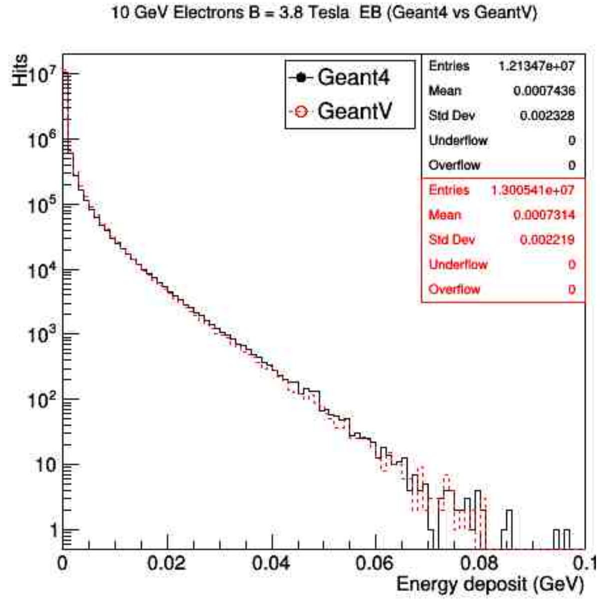
- Number of hits is the same for all 3 energies. The differences are at the level of 0.1/0.3/0.2% for 2, 10 and 50 GeV
- The means differ by 0.8/0.6/0.4% at the three energies

3. Energy Deposit with B = 3.8

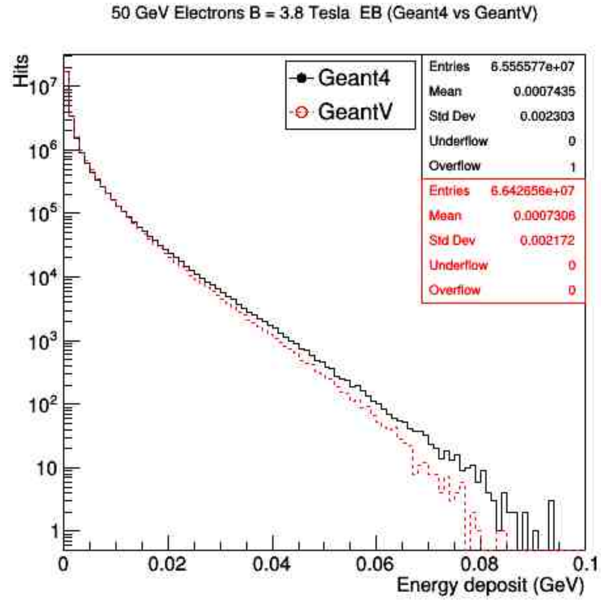
2 GeV Electrons



10 GeV Electrons

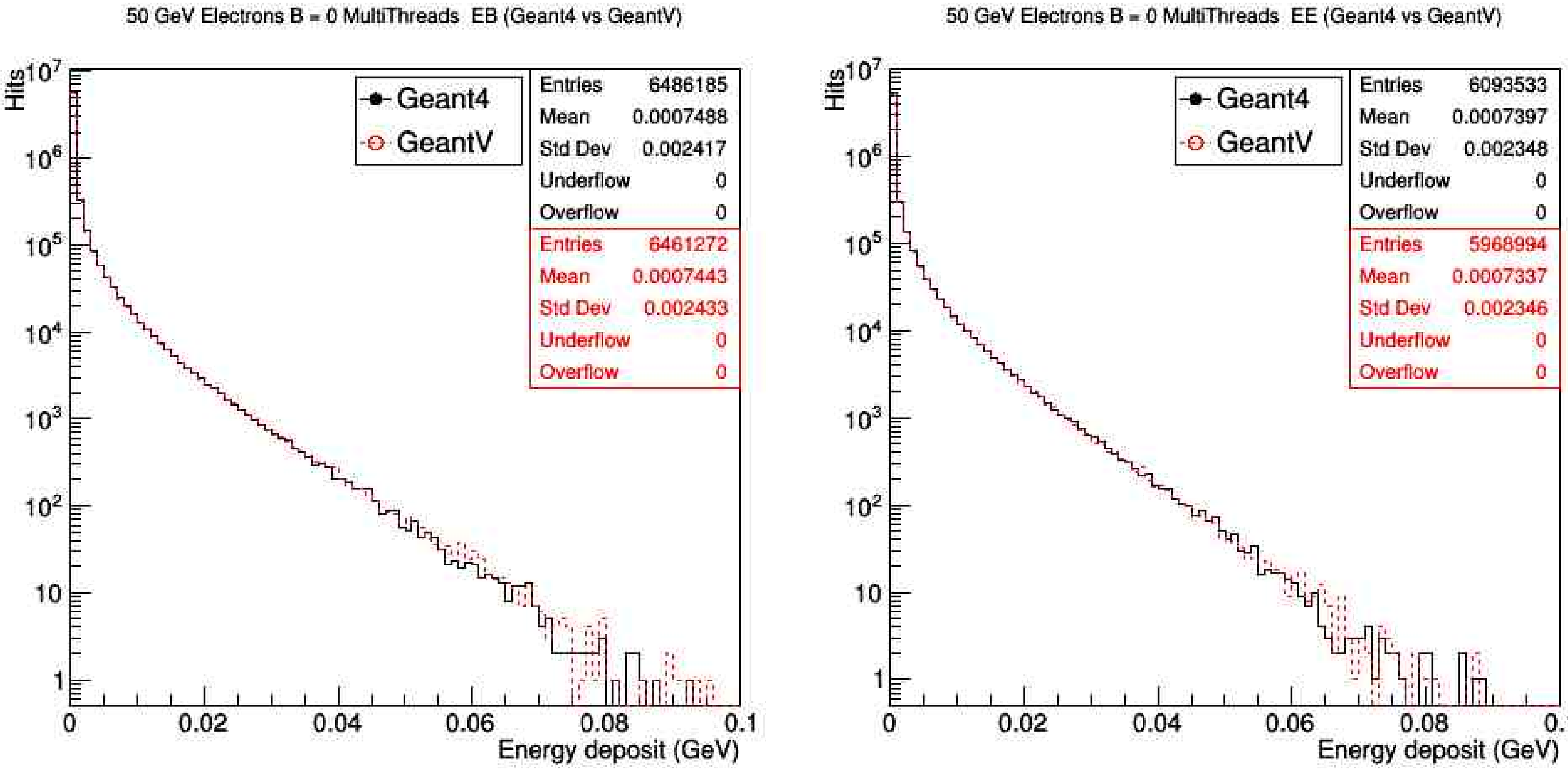


50 GeV Electrons



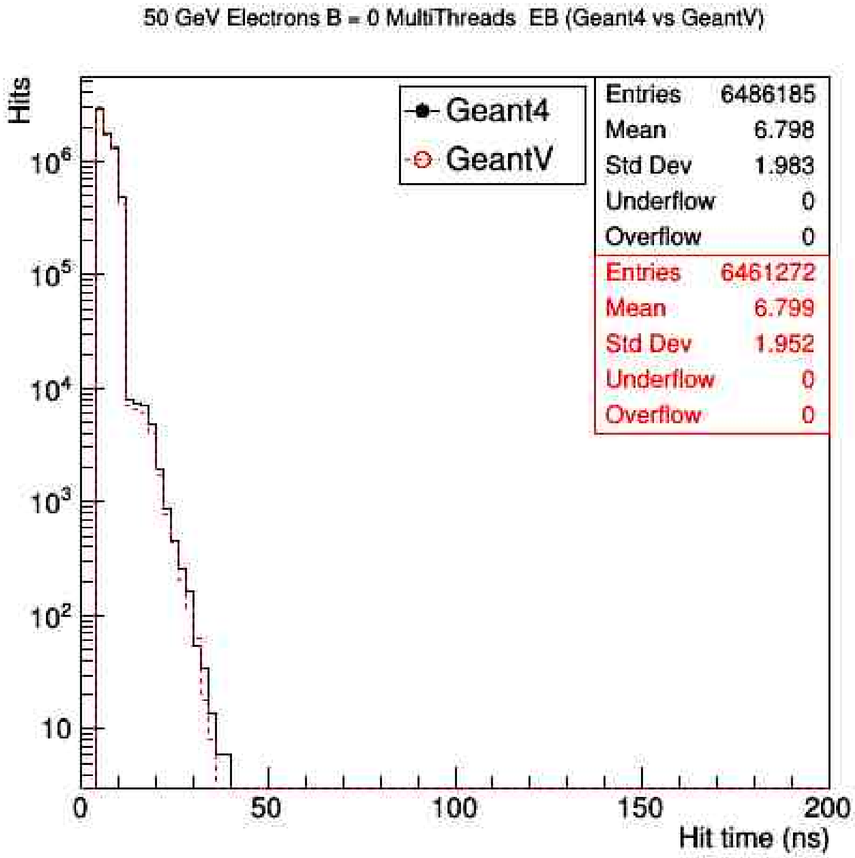
- Number of hits is the same for all 3 energies. The differences are at the level of 27.7/6.7/1.3% for 2, 10 and 50 GeV
- The means differ by 0.5/1.6/1.7% at the three energies

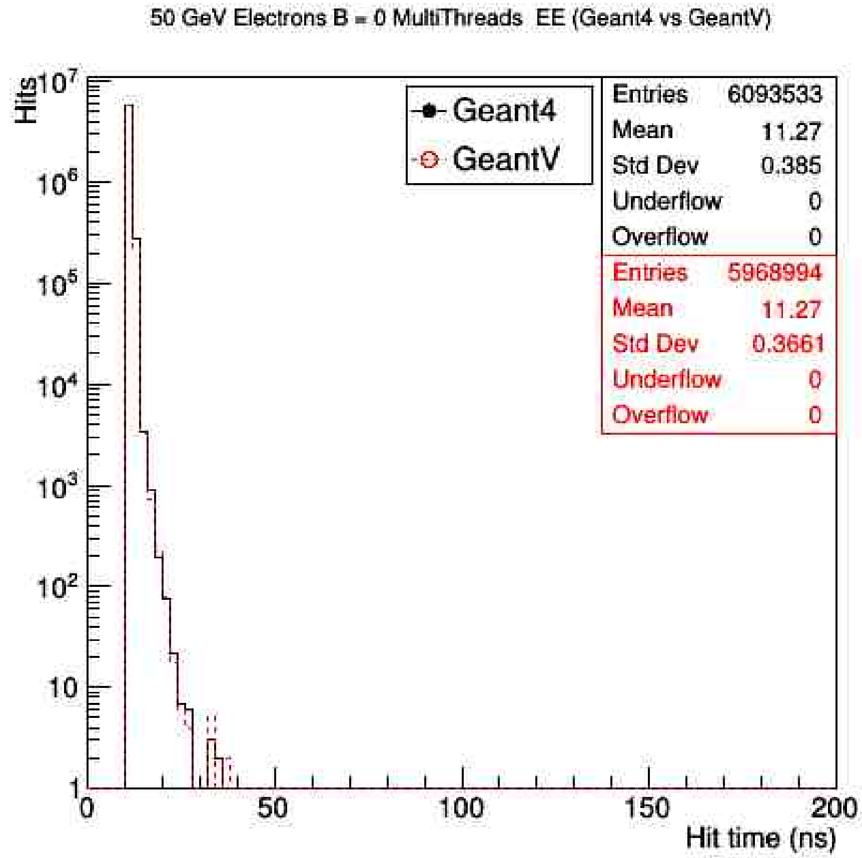
4. Energy Deposit with B = 0, MT



- Events are generated with 50 GeV electrons having random direction within a limited range of η and φ
- The agreement is pretty good in the B=0 option for both # of hits as well as in the shape of the distributions for EB and EE

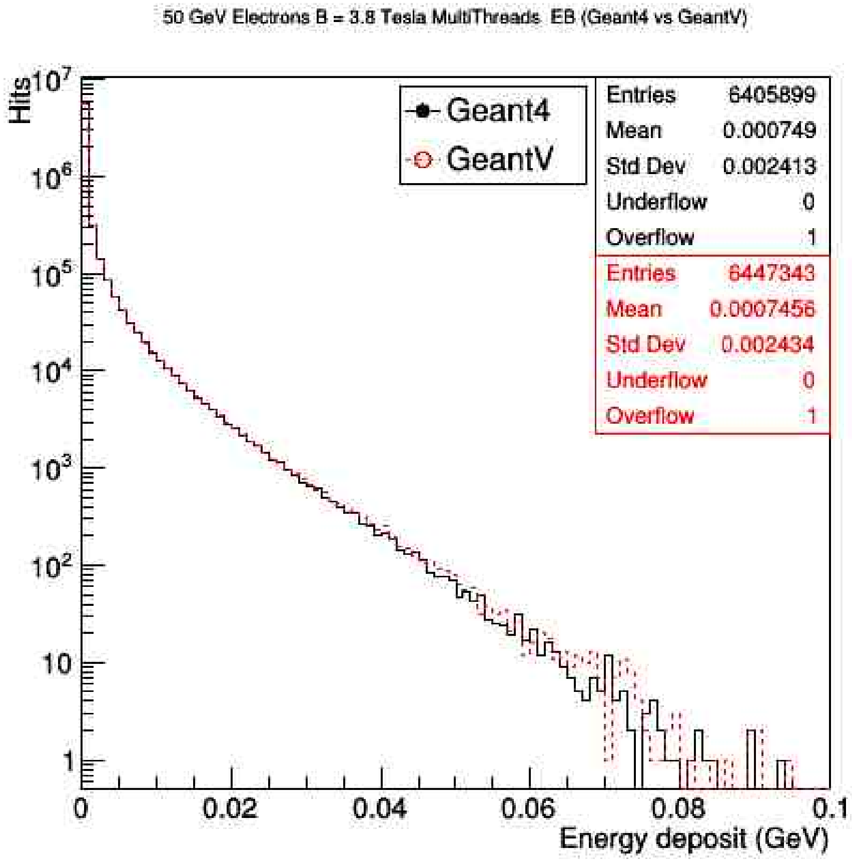
4. Hit Times with B = 0, MT

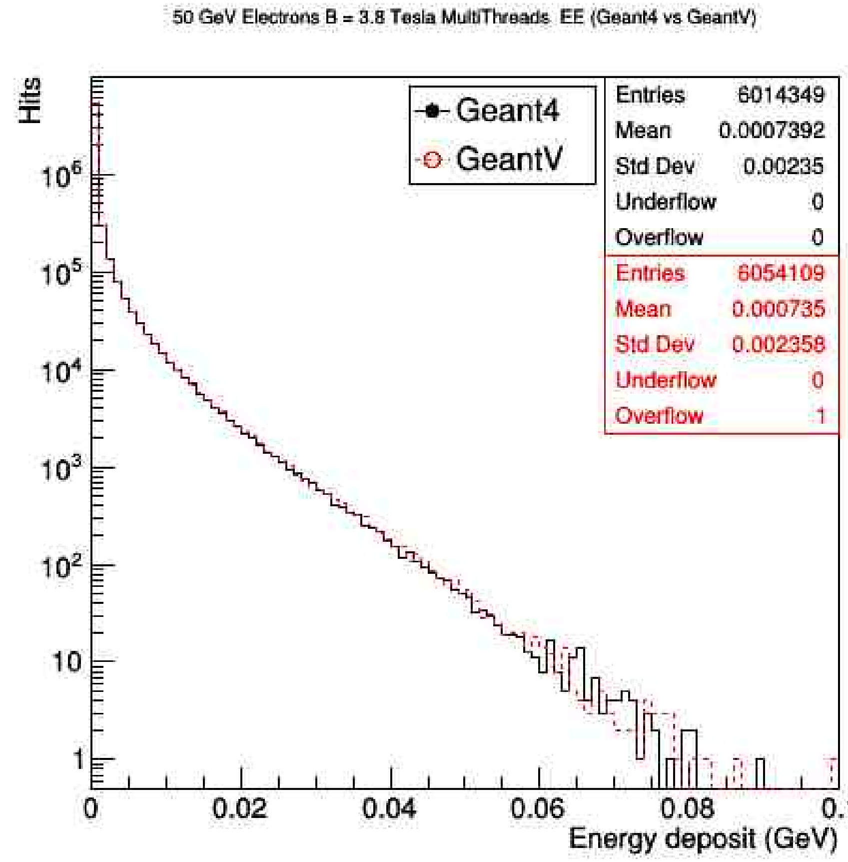




- Hit time distributions are also in good agreement for the B=0 option in EB as well as in EE

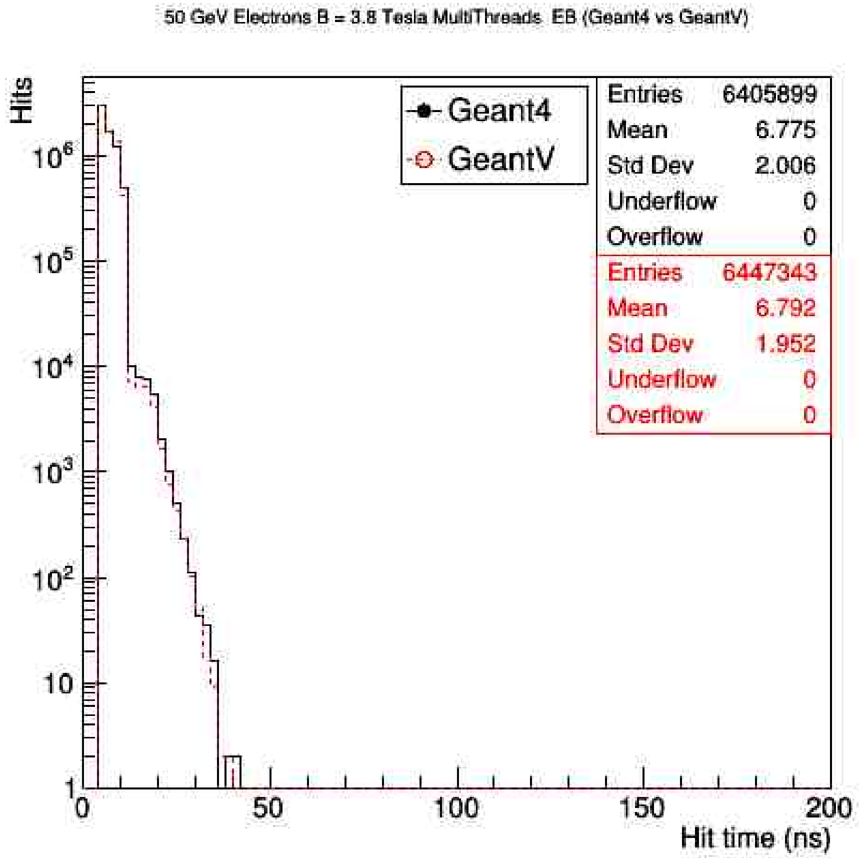
5. Energy Deposit with B = 3.8, MT

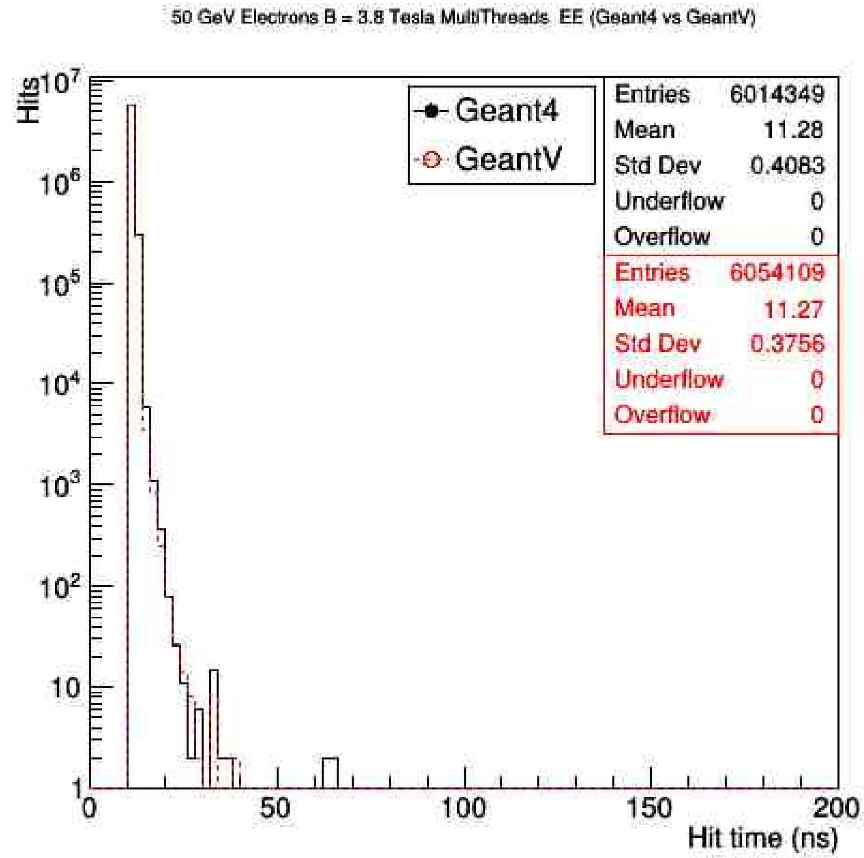




- Same events (50 GeV electrons, random direction within a limited range of η and φ) are simulated in a uniform B-field option of 3.8 Tesla
- The agreement is still good for both # of hits as well as in the shape of the distributions for EB and EE

5. Hit Times with B = 3.8, MT





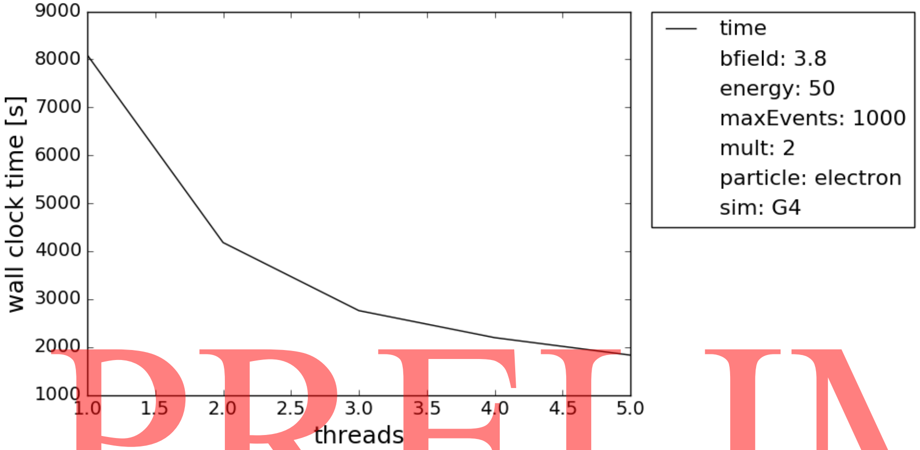
- Hit time distributions are also in reasonable agreement for the B = 3.8 Tesla option in EB as well as in EE

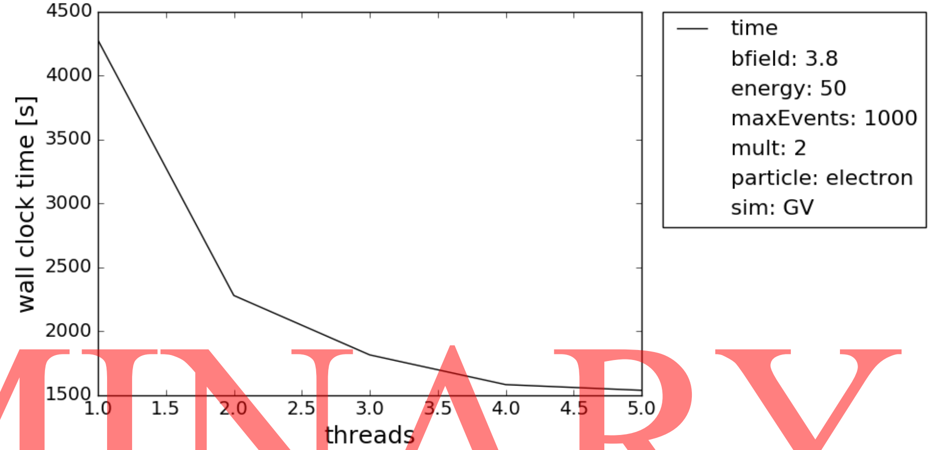
Performance Tests

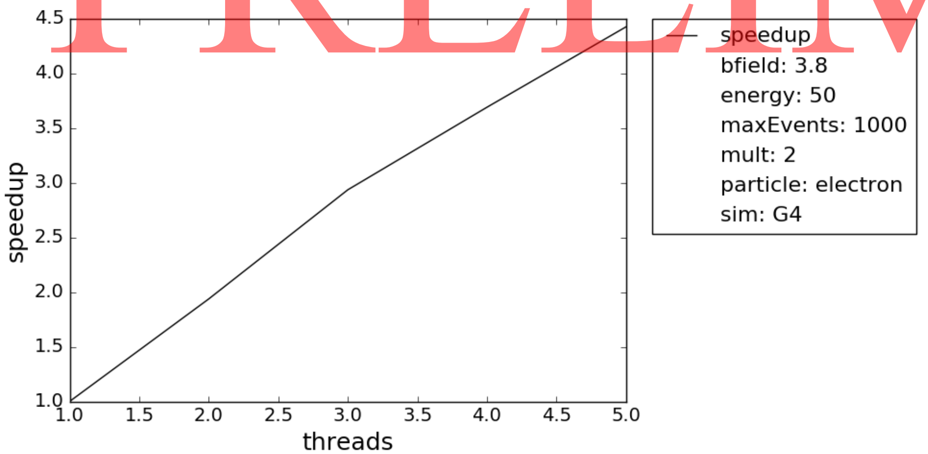
- Compare GeantV and Geant4 CPU usage simulating exact same generated 1000 events (2 electrons w/ E = 50 GeV, random directions)
- Running on FermiCloud VM with:
  - Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz
  - sse4.2 instructions
- Keep other threads busy when running MT tests
- Track memory with CMSSW TimeMemoryInfo tool
  - Measures VSIZE, RSS per event
  - Also measures wall clock time → calculate speedup
- Track CPU usage with igprof (measures all threads together):
  - total = other + geant + output
  - other = initialization, overhead, etc.
  - geant = event loop in Geant4 or GeantV code
    - scoring = subset of event loop in user code
  - output = writing hits to CMSSW EDM ROOT file

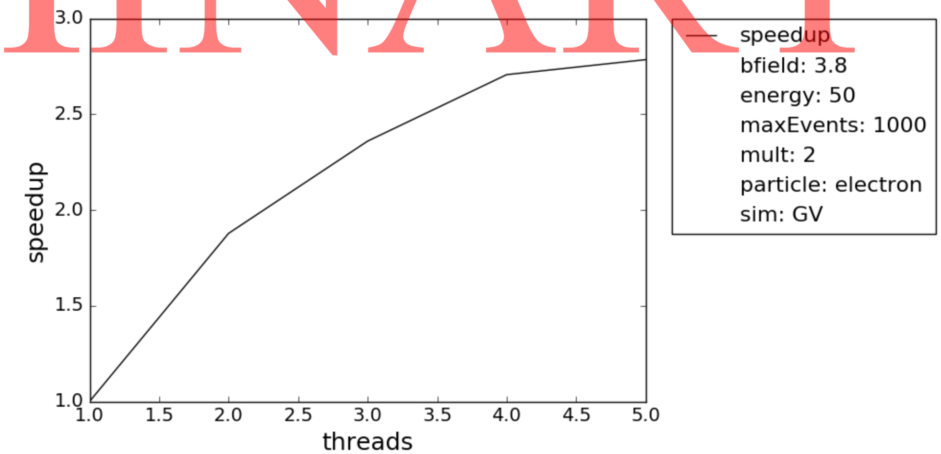
Time Performance

PRELIMINARY



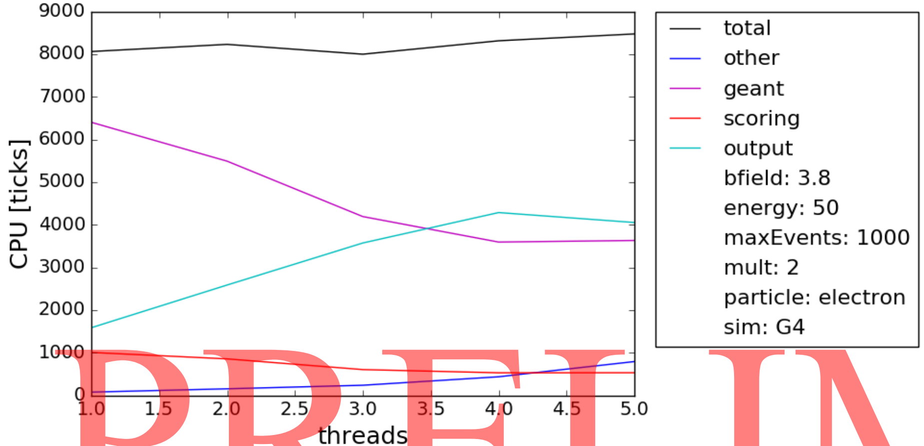


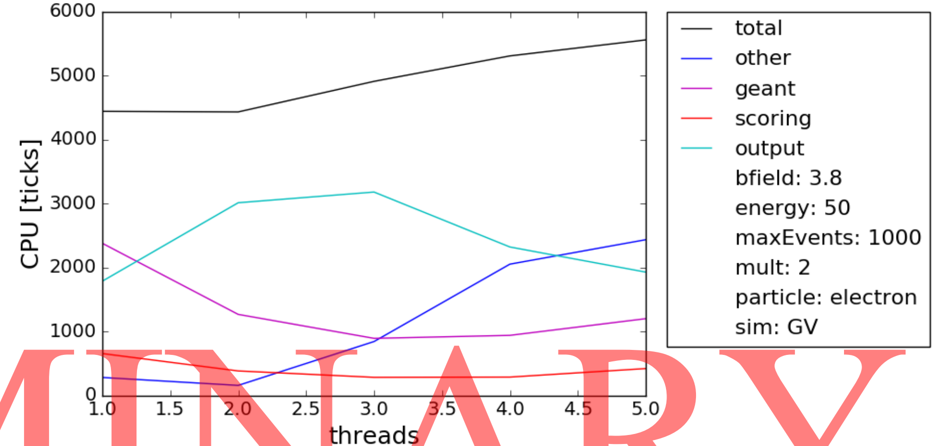


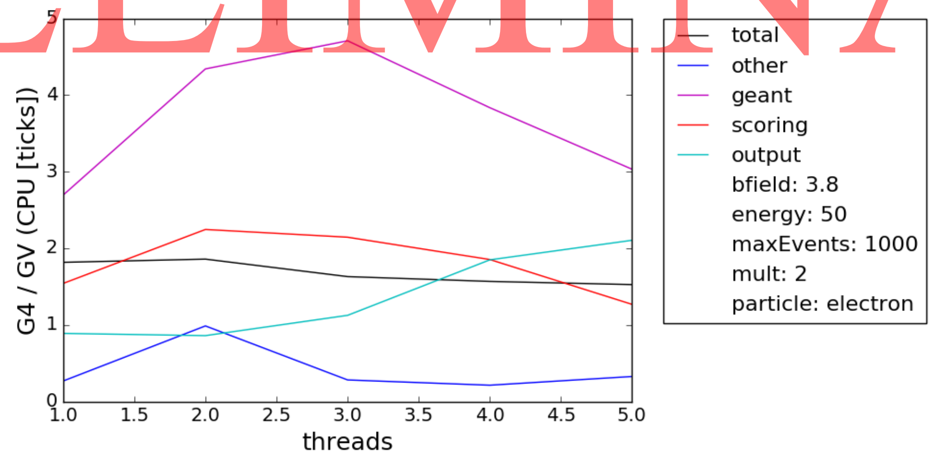


- G4 has better scaling w/ # threads than GV (expected?)

CPU Performance

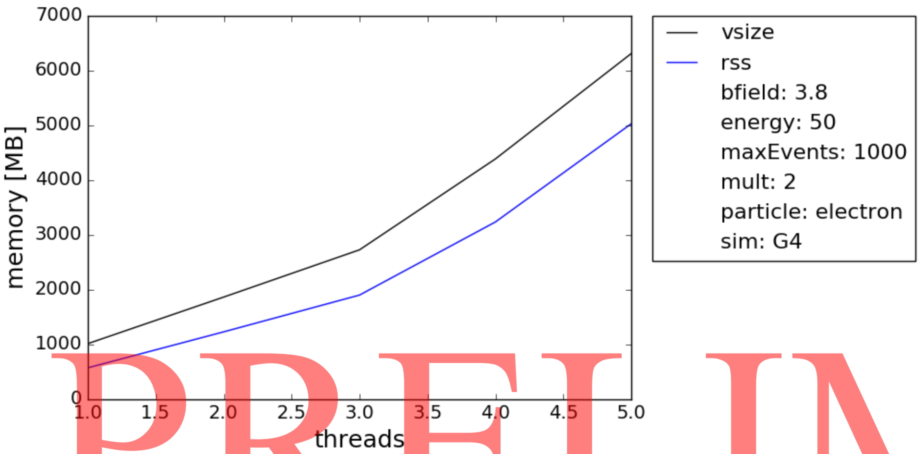


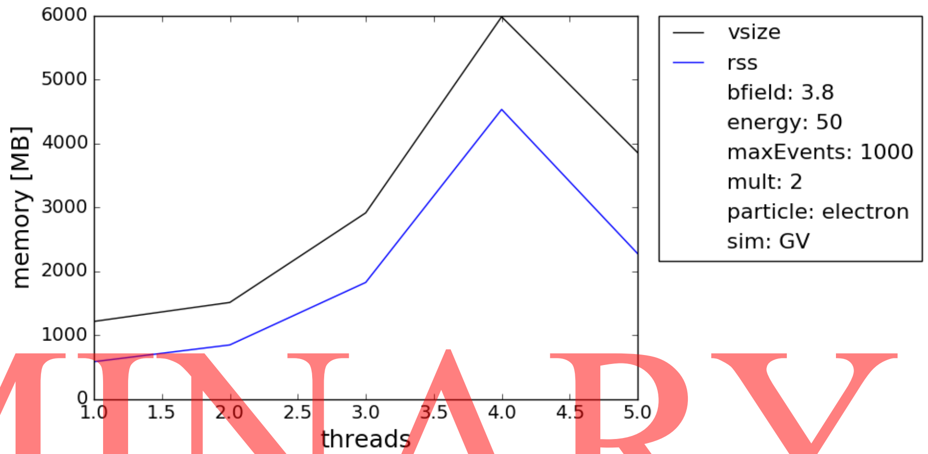


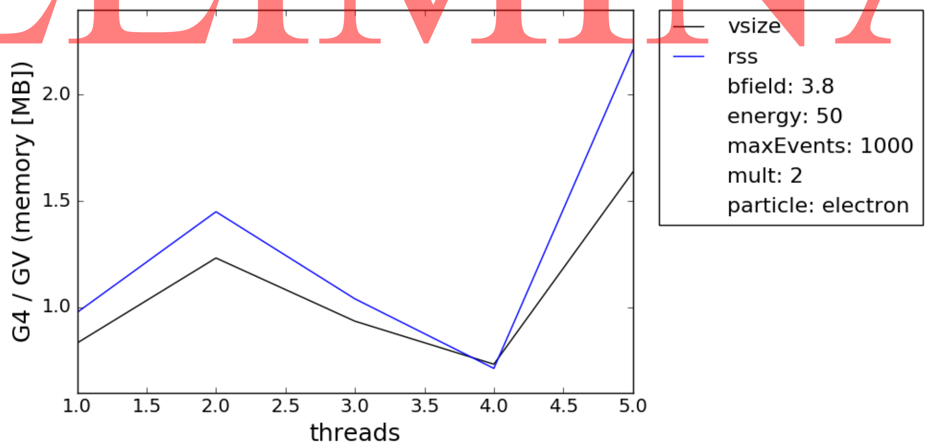


- GV close to factor of 2 better than G4 in total CPU usage
  - ~3× in event loop, ~2× in scoring, similar in output, worse in initialization

Memory Performance







- Expected GV to use more memory than G4
- True for 1 thread, but not for MT → dominated by output?
- Some fluctuations observed in GV, to be investigated
- Memory overhead from duplicated ScoringClass instances can be optimized

Outlook

- ➢ Demonstrator of first “full” GeantV-CMSSW integration is ready
  - o Major remaining item: magnetic field map
- “Rosetta stone” mostly contained in StepWrapper and VolumeWrapper:

Geant4	GeantV
StepWrapper VolumeWrapper	StepWrapper VolumeWrapper

- Physics validation nearly complete
  - o Gaining confidence that G4 and GV are simulating the same things
- Now starting to test computing performance
- ➢ Promising early results!

Backup

Template Wrappers

- **Goal**: use *exact same* SD code for Geant4 and GeantV
- Problem: totally incompatible APIs
  - Example: `G4Step::GetTotalEnergyDeposit()` vs. `geant::Track::Edep()`
- Solution: template wrapper with unified interface
  e.g. `StepWrapper<T>::getEnergyDeposit()`
  - SD code *only calls* the wrapper
  - Wrapper stores pointer to T (minimize overhead)
- Current wrappers:
  - BeginRun
  - BeginEvent
  - Step
  - Volume
  - EndEvent
  - EndRun

Traits

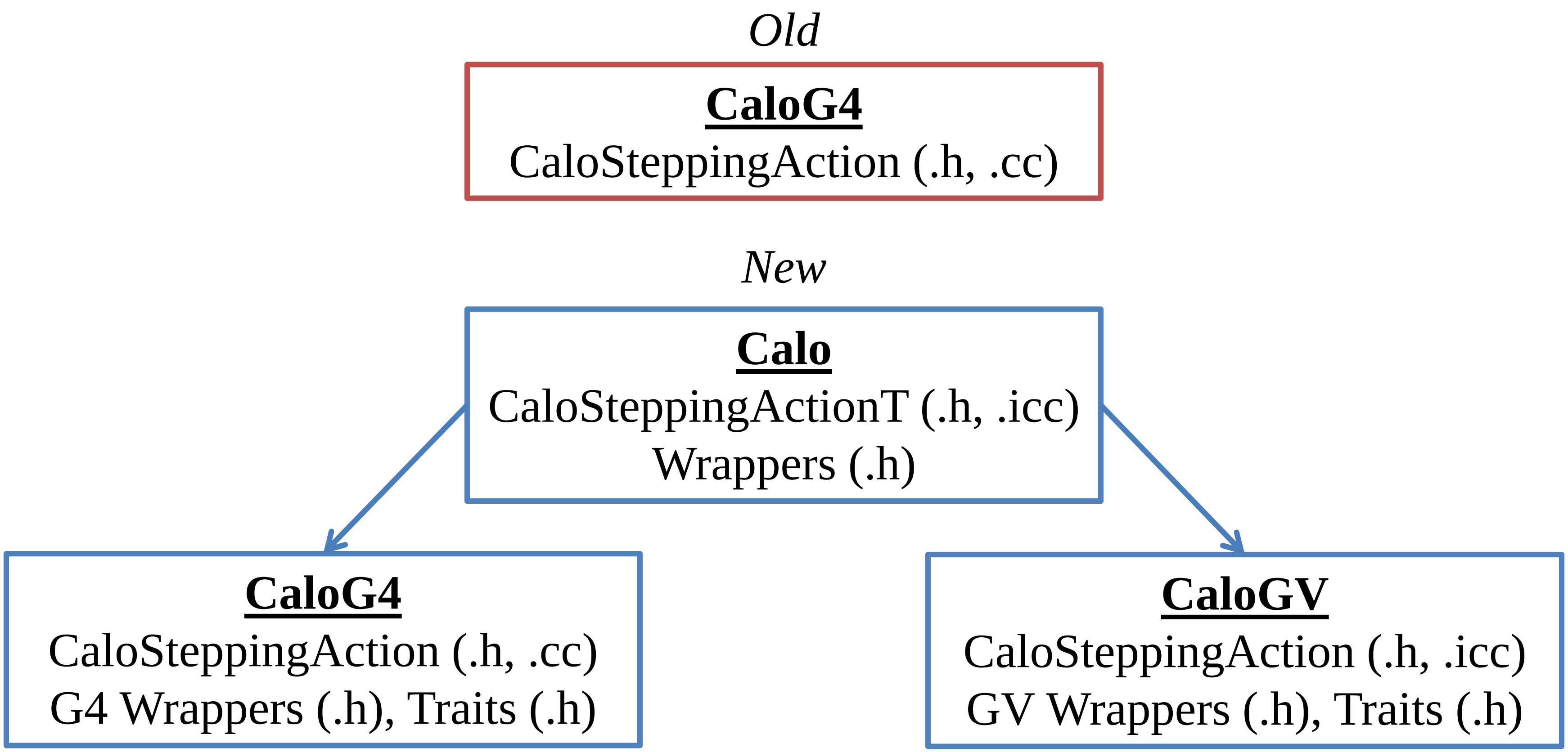
- Collect Geant4/GeantV-specific types and wrappers into unified **Traits** class:

```
struct G4Traits {
  typedef G4Step Step;
  typedef sim::StepWrapper<Step> StepWrapper;
};
struct GVTraits {
  typedef geant::Track Step;
  typedef sim::StepWrapper<Step> StepWrapper;
};
```

- Provides standardized typenames to be used by SD class:

```
template <class Traits> class CaloSteppingActionT : …,
  public Observer<const typename Traits::Step *>
{
  public:
    void update(const Step * step) override {
update(StepWrapper(step)); }
  private:
    // subordinate functions with unified interfaces
    void update(const StepWrapper& step);
};
```

Organization



- SD interface & implementation in **Calo** (.icc file), w/ unimplemented wrapper interfaces
- G4/GV wrapper specializations in **CaloG4/GV**, w/ specific instances of templated SD class → isolate dependencies

Scoring Approaches

- Two approaches to scoring in CMSSW:

1. Inherit from **G4VSensitiveDetector** (Geant4 class)
   → automatically initialized for geometry volumes marked as sensitive
2. Inherit from **SimWatcher** (CMSSW standalone class)
   → need to specify names of watched geometry volumes

- CaloSteppingAction is a demonstrator class w/ approach 2
  - Simplified version of ECAL and HCAL scoring
  - Less dependent on Geant4 interfaces
- “Real” SD code uses approach 1

➢ More work to extract Geant4 dependencies will be necessary
  - Some SD class methods directly from Geant4 (via inheritance)
  - Need to mock up Geant4-esque interfaces w/ dummy classes for GeantV