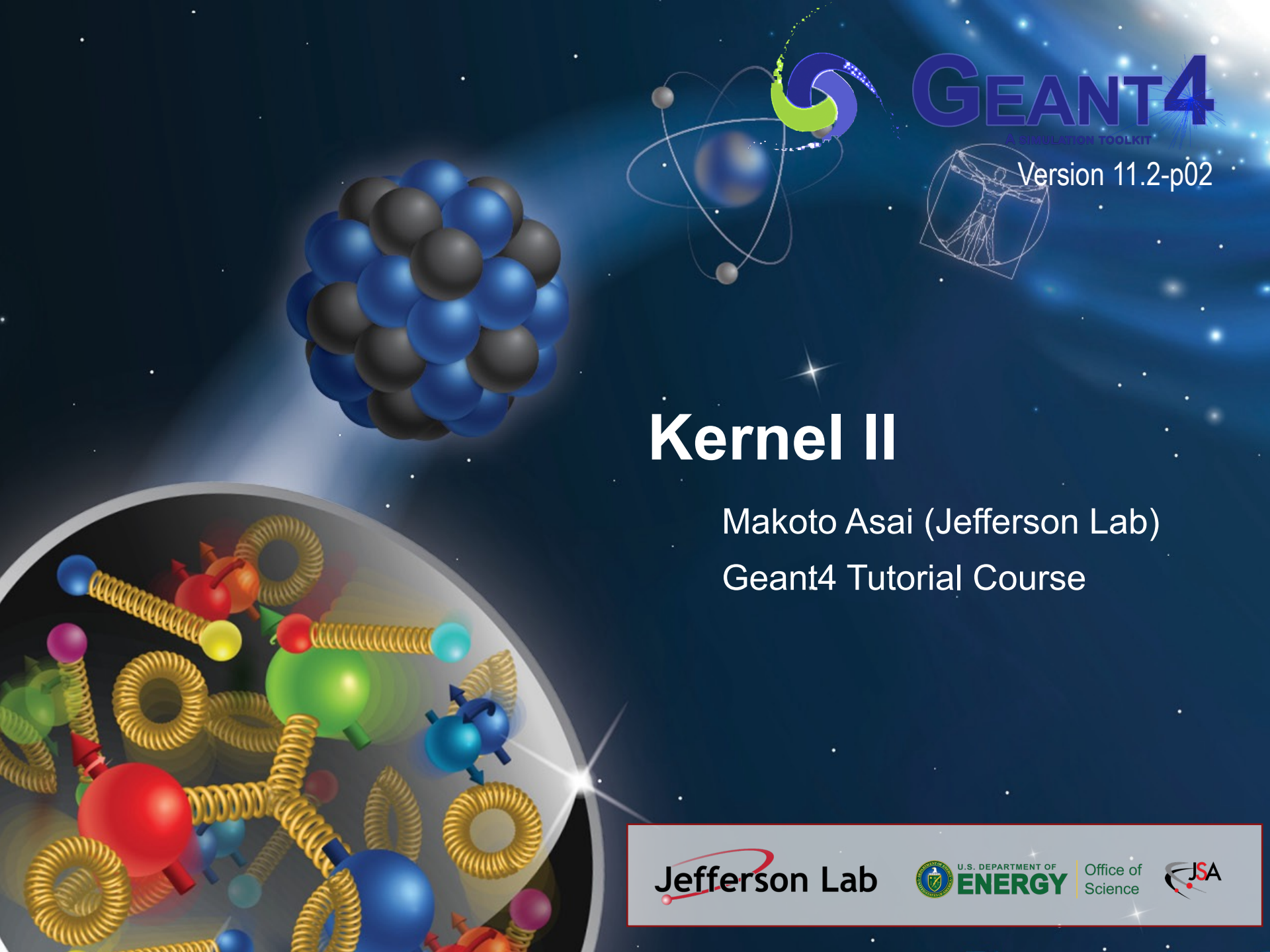


Kernel II

Makoto Asai (Jefferson Lab)

Geant4 Tutorial Course

Jefferson Lab
U.S. DEPARTMENT OF ENERGY | Office of Science
JSA

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- User limits
- Attaching user information to G4 classes
- Accumulating event data for a run

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science
JSA

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- User limits
- Attaching user information to G4 classes
- Accumulating event data for a run

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science
JSA

G4UserLimits

- User limits are artificial limits affecting to the tracking.

```
G4UserLimits(G4double ustepMax = DBL_MAX,
             G4double utrakMax = DBL_MAX,
             G4double utimeMax = DBL_MAX,
             G4double uekinMin = 0.,
             G4double urangMin = 0. );
```

  - ustepMax; // max allowed step size in this volume
  - utrakMax; // max total track length
  - utimeMax; // max global time
  - uekinMin; // min kinetic energy remaining (only for charged particles)
  - urangMin; // min remaining range (only for charged particles)

  Blue : affecting to step

  Red : affecting to track

- You can set user limits to logical volume and/or to a region.
  - User limits assigned to logical volume do not propagate to daughter volumes.
  - User limits assigned to region propagate to daughter volumes unless daughters belong to another region.
  - If both logical volume and associated region have user limits, those of logical volume win.

Processes co-working with G4UserLimits

- In addition to instantiating G4UserLimits and setting it to logical volume or region, you have to assign the following process(es) to particle types you want to affect.
- Limit to step

  ustepMax : max allowed Step size in this volume

  – G4StepLimiter process must be defined to affected particle types.

  – This process limits a step, but it does not kill a track.
- Limits to track

  utrakMax : max total track length

  utimeMax : max global time

  uekinMin : min kinetic energy (only for charged particles)

  urangMin : min remaining range (only for charged particles)

  – G4UserSpecialCuts process must be defined to affected particle types.

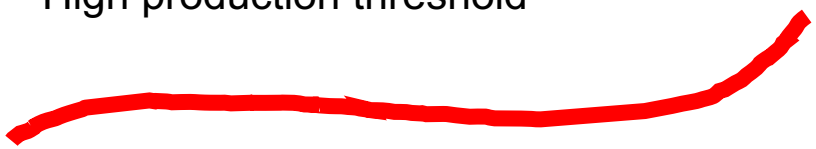
  – This process limits a step and kills the track when the track comes to one of these limits. Step limitation occurs only for the final step.

Production thresholds (a.k.a. cuts)

- Geant4 does not have any tracking cut. It always tracks a particle down to zero kinetic energy.
  - Unless the particle interacts or goes away.
  - A particle may decay or be captured even after it stops.
- Of course each physics model has its limited applicability.
  - For example Standard EM has lower limit of 990 eV.
  - It means we can calculate the range of a particle at E_k = 990 eV.
  - So, when a particle comes down to E_k = 990 eV, instead of killing it at that point, Geant4 makes one more step to push the particle to its final stopping point.
- Geant4 does have production thresholds applied to secondary particle production.
  - To address the infrared divergence of some physics processes
    - e.g. ultra-soft forward gamma production in Bremsstrahlung
  - To also address the limited computing resources
    - e.g. too many soft delta-rays
- Geant4 requires production thresholds to be specified in length.
  - Secondaries that won't travel more than the threshold won't be generated but their kinetic energies are deposited along the trajectory of their parent particle.

Production thresholds (a.k.a. cuts)

- High production threshold

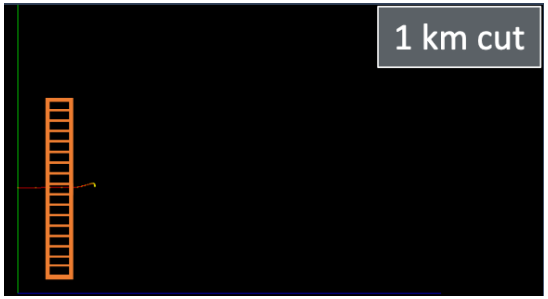


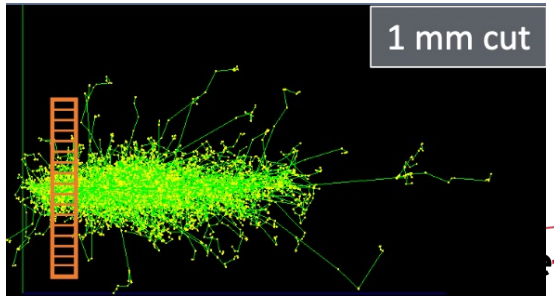
  - Few secondary produced by ionization or Bremsstrahlung
  - Most energy are deposited along the parent track.
    - Continuous energy loss
  - You will see higher total energy deposition along a step of the parent track.

- Low production threshold

  - Lots of secondaries produced by ionization or Bremsstrahlung
  - Energy deposition is shared by local deposition along the parent and secondaries.
  - You will see lower total energy deposition along a step of the parent track.

- If energy deposition is scored in a large bulk volume, you won't see difference in high and low thresholds.
  - Except for the difference in computing time
- If energy deposition is scored in granular scoring cells, you will see large difference.
- We recommend the production thresholds being comparable to volume dimensions.
  - Default length : 0.7 mm





Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- User limits
- Attaching user information to G4 classes
- Accumulating event data for a run

Attaching user information

- Class extension
  - You can create your own class derived from provided base class
  - G4Run, G4VHit, G4VDigit, G4VTrajectory, G4VTrajectoryPoint
- Aggregation
  - You can attach a user information class object
    - G4Event - G4VUserEventInformation
    - G4Track - G4VUserTrackInformation
    - G4PrimaryVertex - G4VUserPrimaryVertexInformation
    - G4PrimaryParticle - G4VUserPrimaryParticleInformation
    - G4Region - G4VUserRegionInformation
  - User information class object is deleted when associated Geant4 class object is deleted.

Trajectory and trajectory point

- Trajectory and trajectory point class objects persist until the end of an event.
- G4VTrajectory is the abstract base class to represent a trajectory, and G4VTrajectoryPoint is the abstract base class to represent a point which makes up the trajectory.
  - In general, trajectory class is expected to have a vector of trajectory points.
- Geant4 provides G4Trajectoy and G4TrajectoryPoint concrete classes as defaults. These classes keep only the most commonly-used quantities.
  - G4RichTrajectory and G4SmoothTrajectory are also available.
  - If the you want to keep some additional information, you are encouraged to implement your own concrete trajectory and trajectory point classes deriving from G4VTrajectory and G4VTrajectoryPoint base classes.
  - Do not use G4Trajectory nor G4TrajectoryPoint concrete class as base classes unless you are sure not to add any additional data member.
    - Source of memory leak

Creation of trajectories

- Naïve creation of trajectories occasionally causes a memory consumption concern, especially for high energy EM showers.
- In UserTrackingAction, you can switch on/off the creation of a trajectory for the particular track.

```
void MyTrackingAction
          ::PreUserTrackingAction(const G4Track* aTrack)
{
  if(...)
  { fpTrackingManager->SetStoreTrajectory(true); }
  else
  { fpTrackingManager->SetStoreTrajectory(false); }
}
```

- If you want to use user-defined trajectory, object should be instantiated in this method and set to G4TrackingManager by SetTrajectory() method.

```
fpTrackingManager->SetTrajectory(new MyTrajectory(…));
```

Bookkeeping issues

- Connection from G4PrimaryParticle to G4Track

  G4int G4PrimaryParticle::GetTrackID()

  – Returns the track ID if this primary particle had been converted into G4Track, otherwise -1.

    - Both for primaries and pre-assigned decay products

- Connection from G4Track to G4PrimaryParticle

  G4PrimaryParticle* G4DynamicParticle::GetPrimaryParticle()

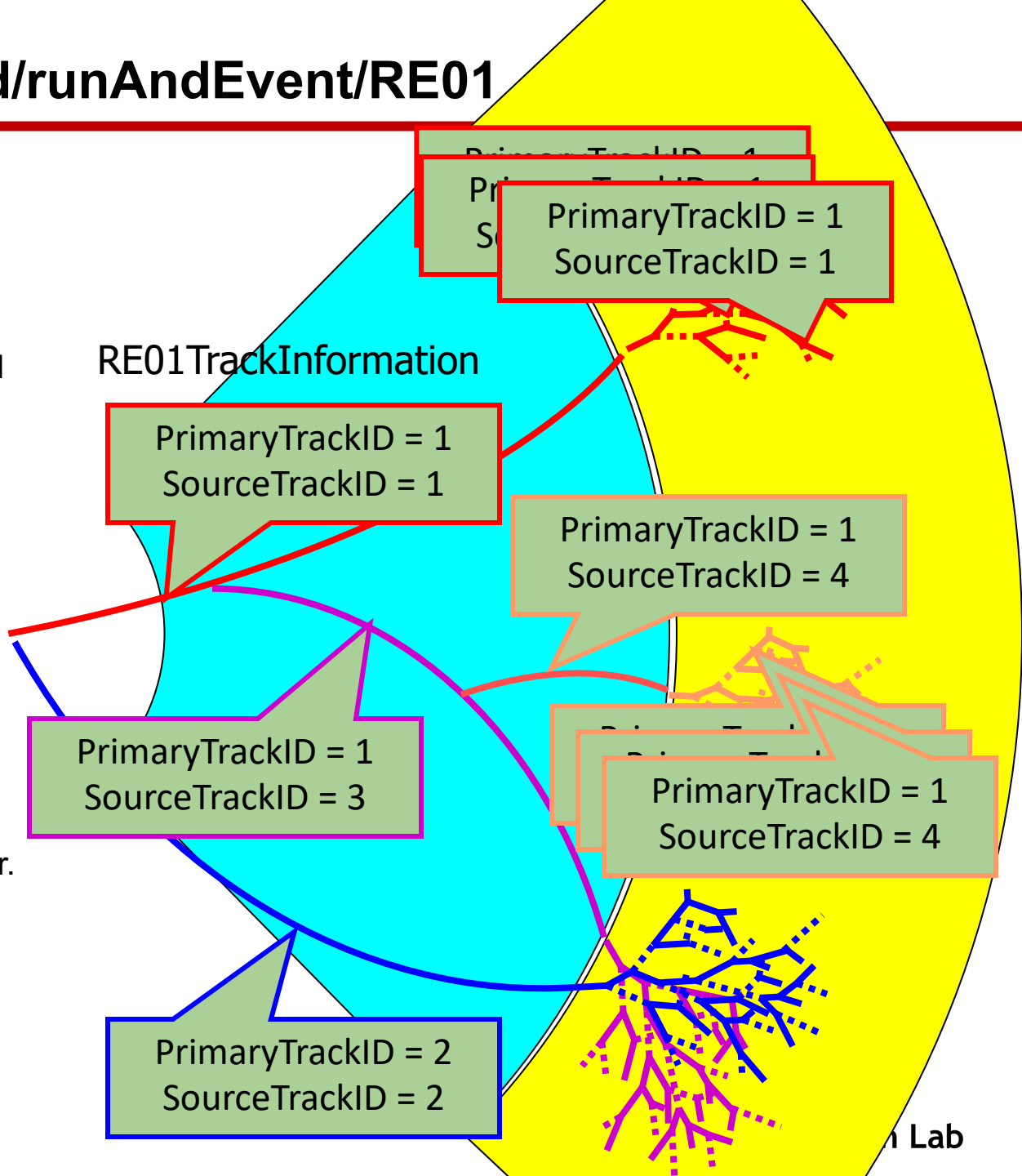
  – Returns the pointer of G4PrimaryParticle object if this track was defined as a primary or a pre-assigned decay product, otherwise null.

- G4VUserPrimaryVertexInformation, G4VUserPrimaryParticleInformation and G4VUserTrackInformation may be used for storing additional information.

  – Information in UserTrackInformation should be then copied to user-defined trajectory class, so that such information is kept until the end of the event.

Examples/extended/runAndEvent/RE01

- An example for connecting G4PrimaryParticle, G4Track, hits and trajectories, by utilizing G4VUserTrackInformation and G4VUserRegionInformation.
- PrimaryTrackID is copied to UserTrackInformation of daughter tracks.
- SourceTrackID means the ID of a track which gets into calorimeter.
- SourceTrackID is updated for secondaries born in tracker, while just copied in calorimeter.



Examples/extended/runAndEvent/RE01

```
Primary particles ------------------------------------------------------------

Primary vertex (0,0,0)    at t = 0 [ns]
  ==PDGcode 25 is not defined in G4 (19.53824,24.846369,-6.0465937) [GeV] >>> G4Track ID 1
    ==PDGcode 23 is not defined in G4 (1.1302123,-23.156443,114.16953) [GeV] >>> G4Track ID 6780
      ==PDGcode 13 (mu-) (-22.464989,-38.451706,20.864853) [GeV] >>> G4Track ID 6782
      ==PDGcode -13 (mu+) (23.595201,15.29526,93.304688) [GeV] >>> G4Track ID 6781
    TrackID =6782 : ParentID=6780 : TrackStatus=1
    Particle name : mu-   PDG code : 13   Charge : -1
  = Original momentum : -22.464989 -38.451706 20.864853 GeV
    Vertex : 4.11461
      Current traject
    Point[0] Positio
    Point[1] Positio
    Point[2] Positio
    Point[3] Positio
    Point[4] Positio
    Point[5] Positio
    Poin
    Poin
    Poin
    Poin
    Poin
    Poin
    Poin
    Poin
    Poin
    Poin
    Poin
    Poin
  =
  =
  =
    Point[24] Position= (-233.79372,-400.12944,217.10408)
```

```
TrackID 6782   Position (-126.11431,-215.85917,117.12988)      : 1878.8831 [keV]
TrackID 6782   Position (-127.89383,                          4.56017 [keV]
TrackID 6782   Position (-151.33864,                          83.9962 [keV]
TrackID 6782   Position (-176.56317,-302.20101,163.96689)     : 1776.6378 [keV]
TrackID 6782   Position (-201.7911,-345.36988,187.38871)     : 2413.8986 [keV]
TrackID 6782   Position (-227.01961,-388.53841,210.81469)     : 550.67792 [keV]
TrackID 6782   Position (-227.70865,-389.71739,211.45445)     : 638.57593 [keV]
TrackID 6782   Position (-228.6702,-391.36253,212.34721)      : 778.03992 [keV]
```

```
Source track ID 6782 (mu-,49.162515[GeV]) at (-252.24762,-431.70723,234.23766)
Original primary track ID 1 (unknown,335.96305[GeV])
Cell[11,31]     0.028283647 [GeV]
Cell[12,31]     0.039822296 [GeV]
Cell[13,31]     0.050185748 [GeV]
Cell[14,31]     0.049883344 [GeV]
Cell[15,31]     0.041446764 [GeV]
Cell[16,31]     0.06386168 [GeV]
Cell[16,32]     0.0036926015 [GeV]
Cell[17,32]     7.2955385e-05 [GeV]
Cell[17,31]     0.0043463898 [GeV]
Cell[15,32]     0.010138473 [GeV]
Cell[14,32]     0.0018386352 [GeV]
Cell[13,32]     0.0018836759 [GeV]
Cell[12,32]     0.00036846059 [GeV]
###  Total energy deposition in calorimeter by a source track in 13 cells : 0.29582467 (GeV)
```

Trajectory of track6782

Tracker hits of track6782

Calorimeter hits of track6782

Energy deposition includes not only muon itself but also all secondary EM showers started inside the calorimeter.

RE01RegionInformation

- RE01 example has three regions, i.e. default world region, tracker region and calorimeter region.
  - Each region has its unique object of RE01RegionInformation class.

```
class RE01RegionInformation : public G4VUserRegionInformation
{
  …
  public:
   G4bool IsWorld() const;
   G4bool IsTracker() const;
   G4bool IsCalorimeter() const;
  …
};
```

- Through step->preStepPoint->physicalVolume->logicalVolume->region->regionInformation, you can easily identify in which region the current step belongs.
  - Don’t use volume name to identify.

Use of RE01RegionInformation

```
void RE01SteppingAction::UserSteppingAction(const G4Step * theStep)
{ // Suspend a track if it is entering into the calorimeter
  // get region information
  G4StepPoint* thePrePoint = theStep->GetPreStepPoint();
  G4LogicalVolume* thePreLV = thePrePoint->GetPhysicalVolume()->GetLogicalVolume();
  RE01RegionInformation* thePreRInfo
   = (RE01RegionInformation*)(thePreLV->GetRegion()->GetUserInformation());
  G4StepPoint* thePostPoint = theStep->GetPostStepPoint();
  G4LogicalVolume* thePostLV = thePostPoint->GetPhysicalVolume()->GetLogicalVolume();
  RE01RegionInformation* thePostRInfo
   = (RE01RegionInformation*)(thePostLV->GetRegion()->GetUserInformation());

  // check if it is entering to the calorimeter volume
  if( !(thePreRInfo->IsCalorimeter()) && (thePostRInfo->IsCalorimeter()) )
  { theTrack->SetTrackStatus(fSuspend); }
}
```

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- User limits
- Attaching user information to G4 classes
- Accumulating event data for a run

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science
JSA

Extract useful information

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”.
  - You have to do something to extract information useful to you.
- There are three ways:
  - Built-in scoring commands
    - Most commonly-used physics quantities are available.
  - Use scorers in the tracking volume
    - Create scores for each event
    - Create own Run class to accumulate scores

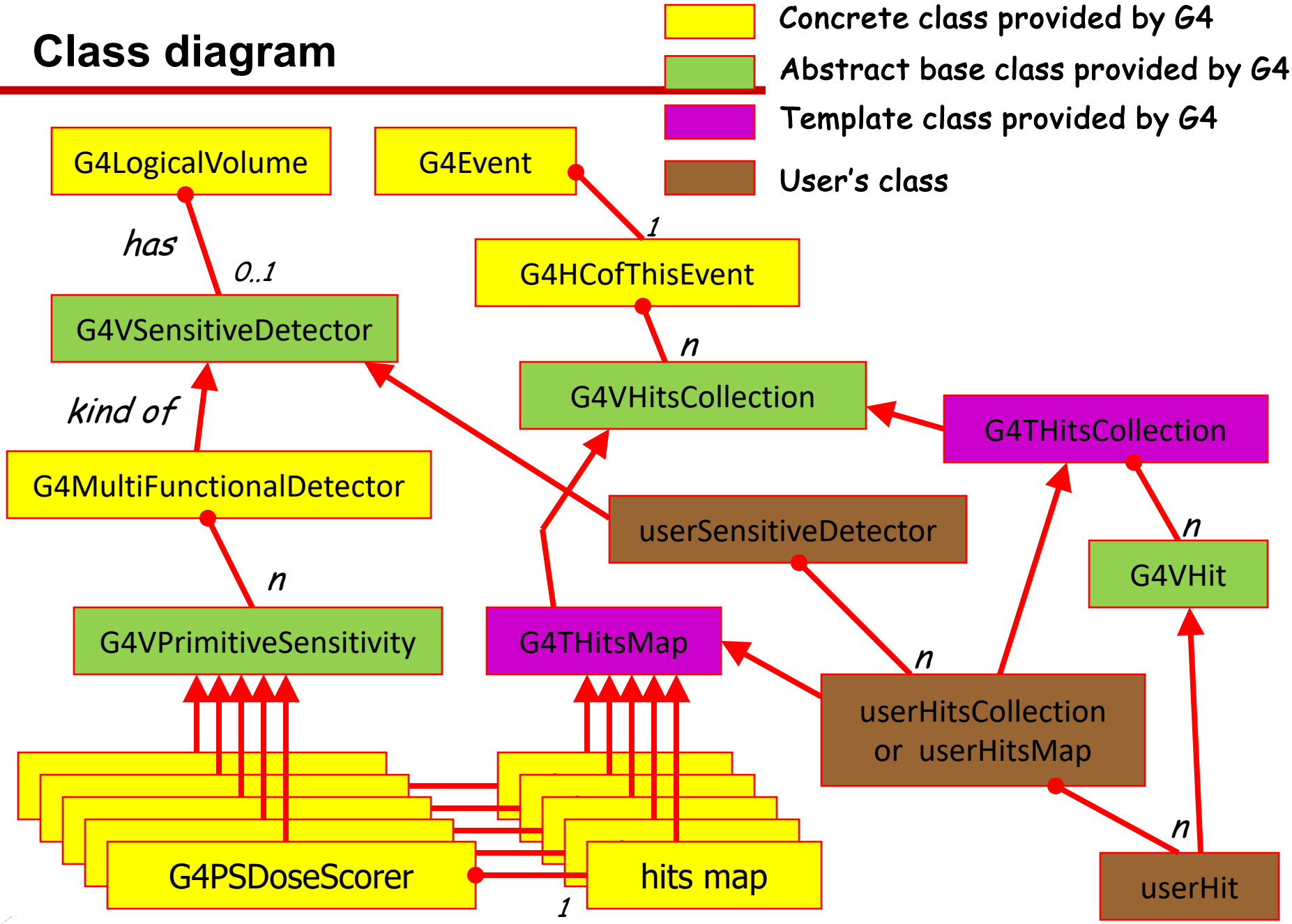
    This talk
  - Assign G4VSensitiveDetector to a volume to generate “hit”.
    - Use user hooks (G4UserEventAction, G4UserRunAction) to get event / run summary
- You may also use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
  - You have full access to almost all information
  - Straight-forward, but do-it-yourself

example…

```
MyDetectorConstruction::ConstructSDandField()
{
    G4MultiFunctionalDetector* myScorer = new
      G4MultiFunctionalDetector("myCellScorer");
    G4VPrimitiveSensitivity* totalSurfFlux = new G4PSFlatSurfaceFlux("TotalSurfFlux");
    myScorer->Register(totalSurfFlux);
    G4VPrimitiveSensitivity* protonSufFlux = new G4PSFlatSurfaceFlux("ProtonSurfFlux");
    G4VSDFilter* protonFilter = new G4SDParticleFilter("protonFilter");
    protonFilter->Add("proton");
    protonSurfFlux->SetFilter(protonFilter);
    myScorer->Register(protonSurfFlux);

    G4SDManager::GetSDMpointer()->AddNewDetector(myScorer);
    SetSensitiveDetector("myLogVol",myScorer);
}
```

Class diagram



Score == G4THitsMap<G4double>

- At the end of successful event, G4Event has a vector of G4THitsMap as the scores.
- Create your own Run class derived from G4Run, and implement two methods.
- RecordEvent(const G4Event*) method is invoked in the worker thread at the end of each event. You can get all output of the event so that you can accumulate the sum of an event to a variable for entire run.
- Merge(const G4Run*) method of the run object in the master thread is invoked with the pointer to the thread-local run object when an event loop of that thread is over. You should merge thread-local scores to global scores.
- Your run class object should be instantiated in GenerateRun() method of your *UserRunAction*.
  - This UserRunAction must be instantiated both for master and worker threads.

Customized run class

```
#include “G4Run.hh”
#include “G4Event.hh”
#include “G4THitsMap.hh”
Class MyRun : public G4Run
{
  public:
    MyRun();
    virtual ~MyRun();
    virtual void RecordEvent(const G4Event*);
    virtual void Merge(const G4Run*);
  private:
    G4int nEvent;
    G4int  totalSurfFluxID,  protonSurfFluxID,  totalDoseID;
    G4THitsMap<G4double> totalSurfFlux, protonSurfFlux, totalDose;
  public:
    … access methods …
};
```

Implement how you accumulate event data

Implement how you merge thread-local scores

Customized run class

```
MyRun::MyRun()
{
  G4SDManager* SDM = G4SDManager::GetSDMpointer();
  totalSurfFluxID = SDM->GetCollectionID("myCellScorer/TotalSurfFlux");
  protonSurfFluxID = SDM->GetCollectionID("myCellScorer/ProtonSurfFlux");
  totalDoseID = SDM->GetCollectionID("myCellScorer/TotalDose");
}
```

name of G4MultiFunctionalDetector object

name of G4VPrimitiveSensitivity object

Customized run class

```
void MyRun::RecordEvent(const G4Event* evt)
{
  G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
  G4THitsMap<G4double>* eventTotalSurfFlux
           = (G4THitsMap<G4double>*)(HCE->GetHC(totalSurfFluxID));
  G4THitsMap<G4double>* eventProtonSurfFlux
           = (G4THitsMap<G4double>*)(HCE->GetHC(protonSurfFluxID));
  G4THitsMap<G4double>* eventTotalDose
           = (G4THitsMap<G4double>*)(HCE->GetHC(totalDoseID));
  totalSurfFlux += *eventTotalSurfFlux;
  protonSurfFlux += *eventProtonSurfFlux;
  totalDose += *eventTotalDose;

  G4Run::RecordEvent(evt);
}
```

No need of loops.
+= operator is provided !

Don't forget to invoke base class method!

Customized run class

```
void MyRun::Merge(const G4Run* run)
{
  const MyRun* localRun = static_cast<const MyRun*>(run);

  totalSurfFlux += *(localRun . totalSurfFlux);
  protonSurfFlux += *(localRun . protonSurfFlux);
  totalDose += *(localRun . totalDose);

  G4Run::Merge(run);
}
```

Cast !

No need of loops.
+= operator is provided !

Don't forget to invoke base class method!

RunAction with customized run

```
G4Run* MyRunAction::GenerateRun()
{ return (new MyRun()); }
void MyRunAction::EndOfRunAction(const G4Run* aRun)
{
  const MyRun* theRun = static_cast<const MyRun*>(aRun);

  if( IsMaster() )
  {
    // … analyze / record / print-out your run summary
    // MyRun object has everything you need …
   }
}
```

IsMaster() returns true for the RunAction object assigned to the master thread. (also returns true for sequential mode)

- As you have seen, to accumulate event data, you do NOT need
  - Event / tracking / stepping action classes
- All you need are your Run and RunAction classes.