Simulation For High-Energy Physics

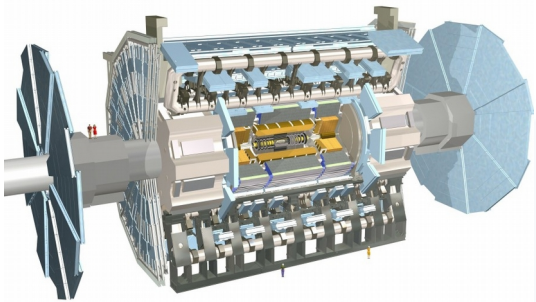
Ben Smart

Science and Technology Facilities Council

Introduction

- This lecture will cover simulation for high-energy physics (HEP).
  - But what is that?
- This lecture is aimed at PhD students,
  to provide them with the basic knowledge needed to pursue a simulation-based project.
- I also want to provide some lessons “from the front lines of HEP simulation”.
  - Really, tips/tricks/gotchas learned, that might be worth passing on.

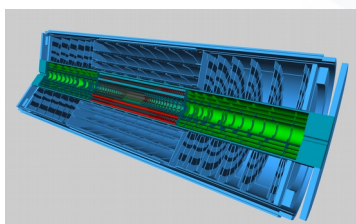
Outline



- What is simulation in high-energy physics?
  - General introduction and big picture.
- Geant4
  - The ‘main’ tool used for HEP simulation.
  - How to build simulations, and how to work with this tool.

- Fluka
  - A useful tool to compliment Geant4.

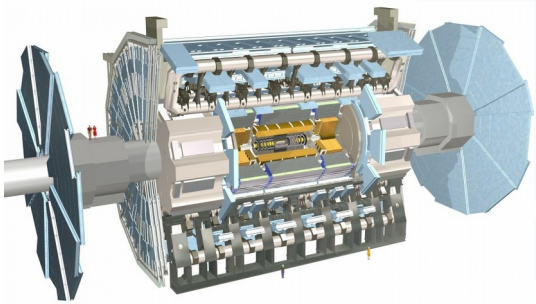
- Lessons from the front lines of HEP simulation (ITk).



What Is Simulation In High-Energy Physics?

- Simulation, in HEP, is the modelling of particles interacting with matter.

- Typically this means modelling of our particle detectors,
  how particles move through them,
  and interact with them.

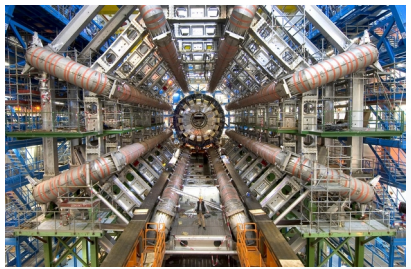


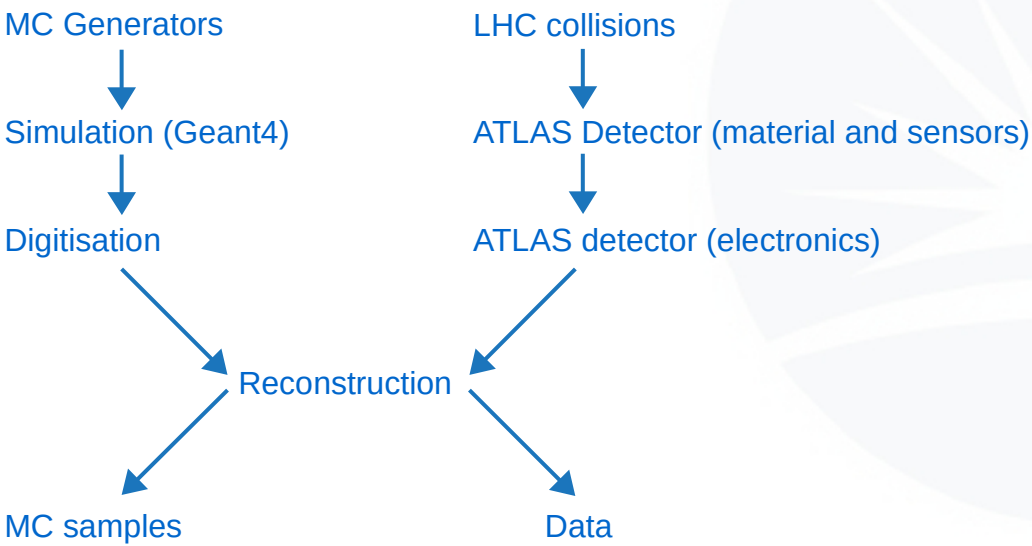
- Simulation is one step in the ‘Monte Carlo (MC) full chain’:
  - More on this next.

- We can also better understand our detector by comparing data to simulation results.
  - Working in simulation, you get to see all the ‘oddities’ in the detector...

The Monte Carlo Full Chain

- Simulation is one part of the Monte Carlo chain.
- It is used to model our detectors, how particles pass through and interact with them, and which sensors are hit, (and where and with how much energy etc.)
- Computationally expensive, (so ways to speed it up have been developed).



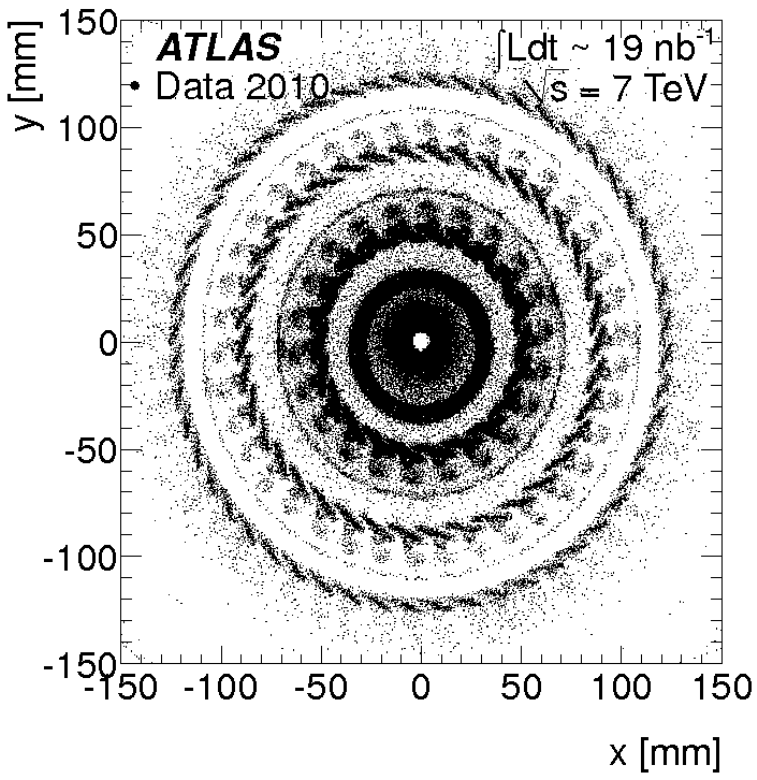


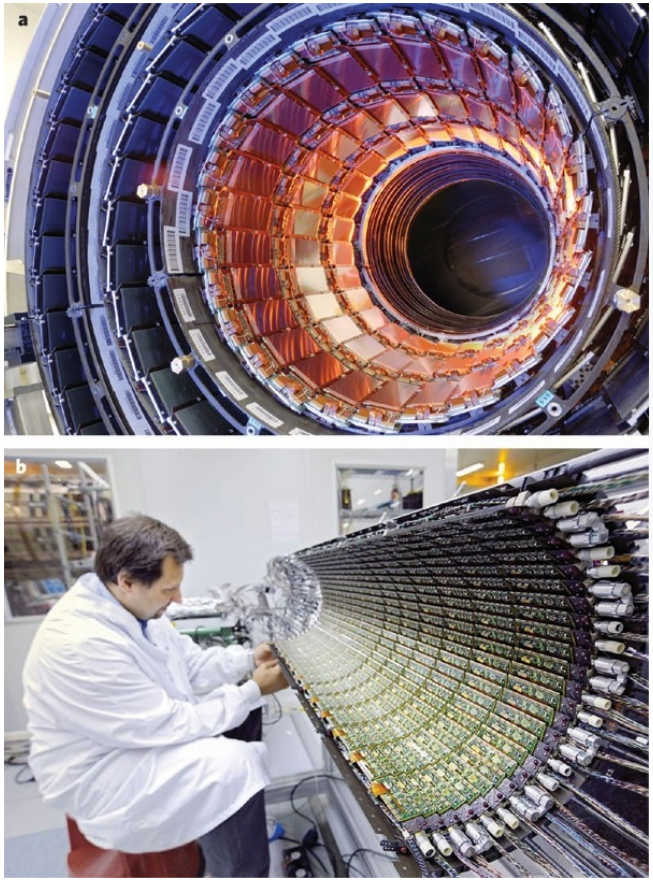
Data / MC Comparison

- In order to validate and debug simulation models,
  data from the real detector and simulated MC samples can be compared.
  - Need to disentangle mismodelling in simulation from mismodelling in MC generators.
  - Generally we look at inclusive data (minimum bias) and simple parameters:
    - Track momentum spectra, positions of reconstructed secondary vertices.
    - The more material a particle has to pass through,
      the more likely it is to interact with material and decay,
      which will produce a secondary vertex.
    - We can then use secondary vertex location to map material...

Data / MC Comparison

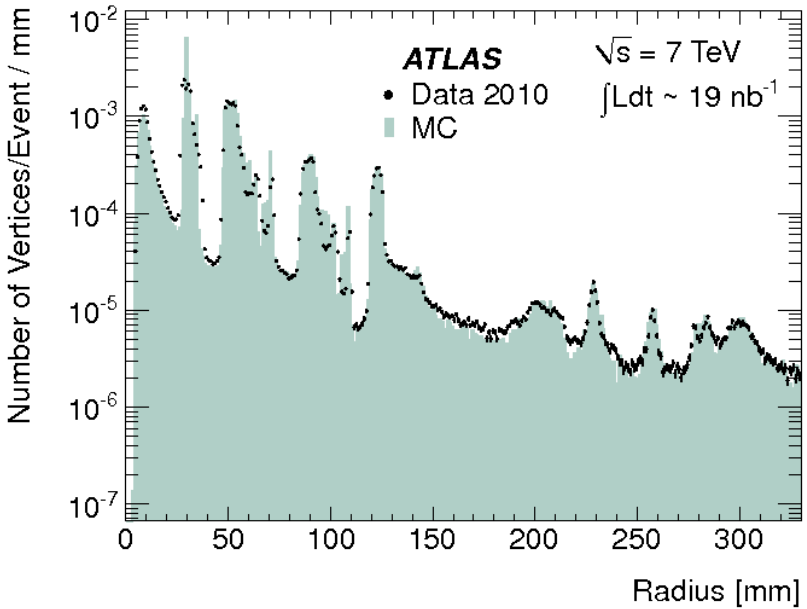
- In practice the (x,y,z) positions of secondary vertices are not enough to debug simulation models, but you get some nice images...

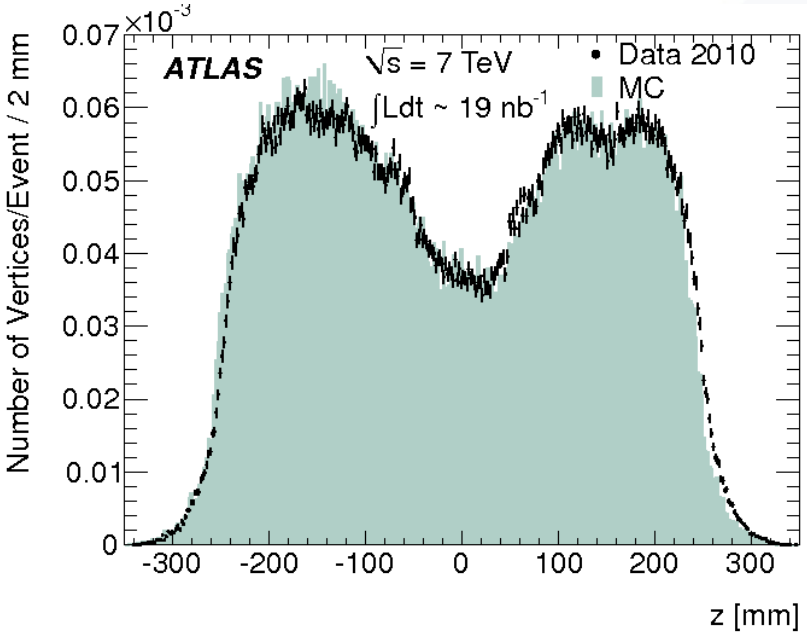




Data / MC Comparison

- Left: Radial positions of reconstructed vertices with |z|<300 mm.
- Right: z positions of reconstructed vertices in the radial range 29-35 mm.



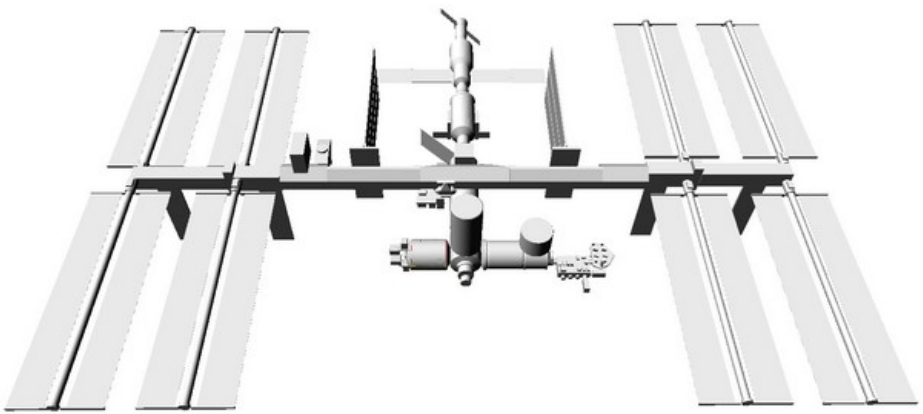


- Once you identify an issue, you try different modifications to your simulation model until you get a better match to data.

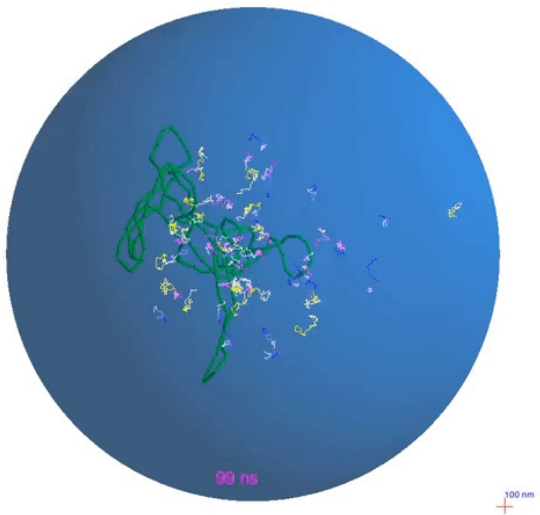
HEP Simulation Tools Are Not Just For HEP...

- Simulation, in HEP, is the modelling of particles interacting with matter.
  - But HEP is not the only field interested in modelling particles interacting with matter.

Geant4 is used by the European Space Agency for modelling radiation (cosmic ray) effects on the International Space Station.



Geant4 is used in medicine for modelling radiation radiation damage to DNA.



- Geant4 is an open-source c++ software package for simulating the passage of particles through matter.
- Can be used in a standalone package – you write the c++ main() function to run it.
- Often used in HEP in an experiment's computing framework, (ie. Athena).
- History: Originated from CERN and international collaboration in 1993.
  - It was an evolution of older simulation software 'Geant3' (which was written in FORTRAN), but using 'new' computing techniques such as Object Oriented programming.

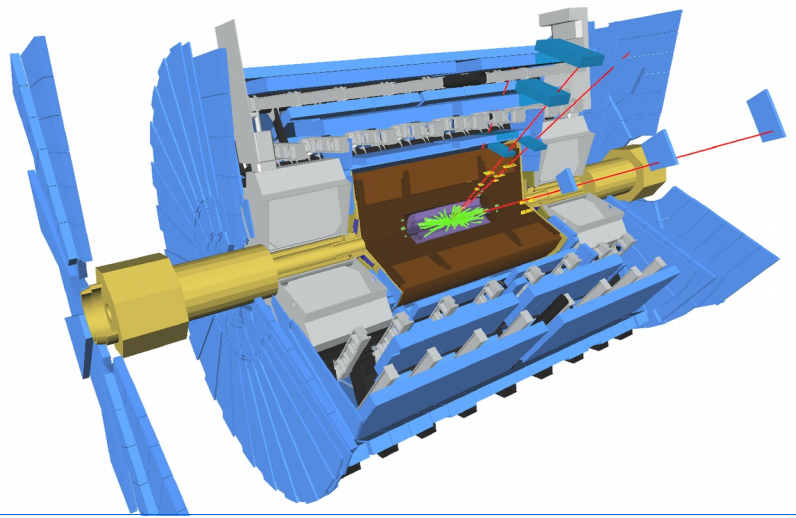
- It is the workhorse of HEP simulation, and has code to handle:

  - the geometry of the system,
  - the materials involved,
  - the fundamental particles of interest,
  - the generation of primary events,
  - the tracking of particles through materials and electromagnetic fields,
  - the physics processes governing particle interactions,
  - the response of sensitive detector components,
  - the generation of event data,
  - the storage of events and tracks,
  - the visualization of the detector and particle trajectories, and
  - the capture and analysis of simulation data at different levels of detail and refinement.

- Excellent documentation for new users on the website: https://geant4.web.cern.ch/
  Introduction: http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/IntroductionToGeant4/html/index.html
  Read ‘Getting Started’ here: http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/ForApplicationDeveloper/html/Introduction/introduction.html

Geant4 Parts

- We will cover:
  - Model building / geometry
  - Materials
  - Particles
  - Physics lists
    - Particle transport and steps
      - Particle Transport And Geometry Volume Hierarchy
  - Electromagnetic fields
  - Sensitive regions (sensors) and ‘hits’
  - Inputs to simulations
  - Hooks
  - Validation and visualisation
  - Alternative usage
    - Fast simulation
    - Truth smearing
    - Geant4MT



Model Building And Geometry

- Simulation models in Geant4 are built out of ‘volumes’;
  3D shapes such as boxes and cylinders and such.

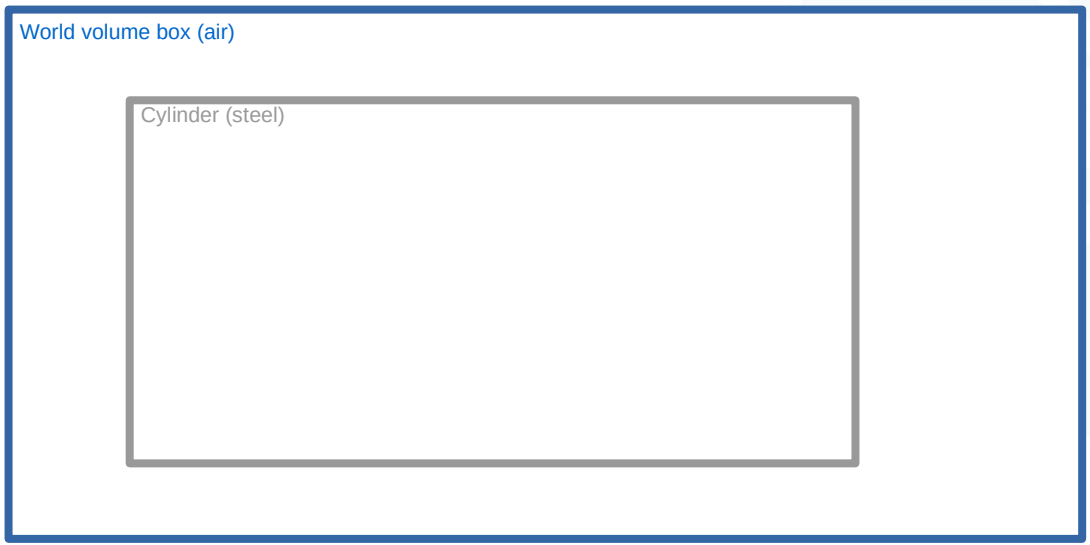
- These volumes have dimensions, positions, orientations,
  and properties (materials associated with them, for example).

- The volumes are organised
  in a hierarchical way:
  - Each volume lives inside another volume, its ‘parent’.
  - Each ‘parent’ can have several ’children’.
  - Only the ‘world volume’ has no parent volume.

- Simulation models in Geant4 are built out of ‘volumes’;
  3D shapes such as boxes and cylinders and such.

- These volumes have dimensions, positions, orientations,
  and properties (materials associated with them, for example).

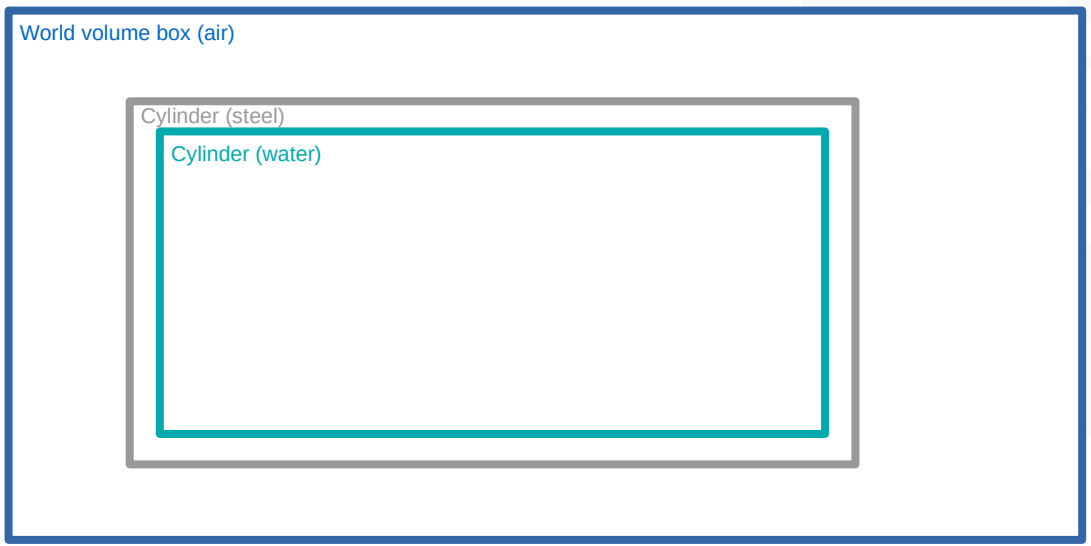
- The volumes are organised
  in a hierarchical way:
  - Each volume lives inside another volume, its ‘parent’.
  - Each ‘parent’ can have several ’children’.
  - Only the ‘world volume’ has no parent volume.



- Simulation models in Geant4 are built out of ‘volumes’; 3D shapes such as boxes and cylinders and such.

- These volumes have dimensions, positions, orientations, and properties (materials associated with them, for example).

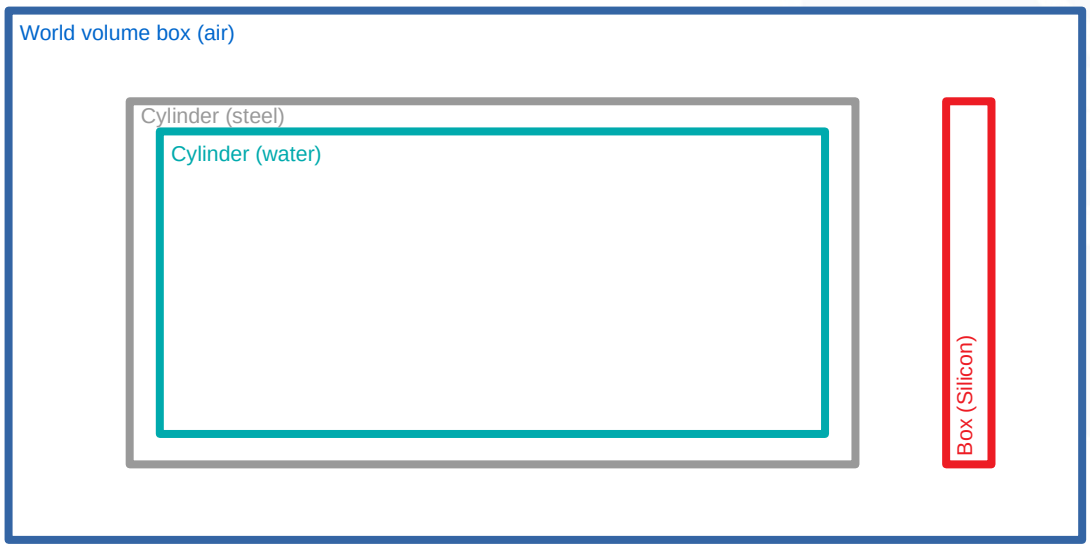
- The volumes are organised in a hierarchical way:
  - Each volume lives inside another volume, its ‘parent’.
  - Each ‘parent’ can have several ’children’.
  - Only the ‘world volume’ has no parent volume.



- Simulation models in Geant4 are built out of ‘volumes’; 3D shapes such as boxes and cylinders and such.

- These volumes have dimensions, positions, orientations, and properties (materials associated with them, for example).

- The volumes are organised in a hierarchical way:
  - Each volume lives inside another volume, its ‘parent’.
  - Each ‘parent’ can have several ’children’.
  - Only the ‘world volume’ has no parent volume.



- In the code, the geometry functionality is handled with sets of classes:
  - shapes
  - logical volumes
  - physical volumes

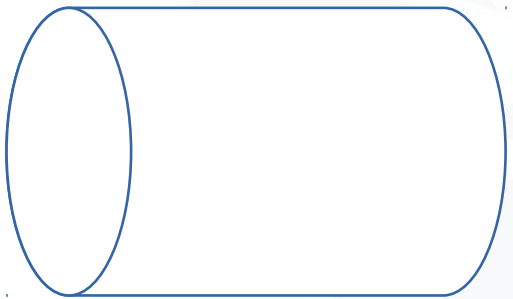
- **Shapes** describe the shape and size of the volume, (box, with width, height, and depth) (cylinder, with radius and length).

```
G4double world_hx = 3.0*m;
G4double world_hy = 1.0*m;
G4double world_hz = 1.0*m;

// This will make a box of 6m by 2m by 2m!
// All dimensions are from the centre of the shape (its origin)
G4Box* worldBox = new G4Box("World", world_hx, world_hy, world_hz);
```

- In the code, the geometry functionality is handled with sets of classes:
  - shapes
  - logical volumes
  - physical volumes

- **Shapes** describe the shape and size of the volume, (box, with width, height, and depth) (cylinder, with radius and length).

```
G4double innerRadius = 0.*cm;
G4double outerRadius = 60.*cm;
G4double halfLengthAlongZ = 25.*cm;
G4double startAngle = 0.*deg;
G4double spanningAngle = 360.*deg;

// This will make a cylinder 50cm long
G4Tubs* metalTube = new G4Tubs("Tracker",
                               innerRadius,
                               outerRadius,
                               halfLengthAlongZ,
                               startAngle,
                               spanningAngle);
```

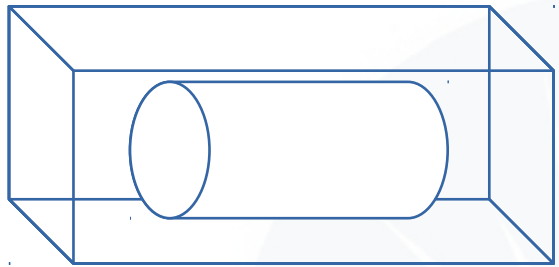


- In the code, the geometry functionality is handled with sets of classes:
  - shapes
  - logical volumes
  - physical volumes

- **Logical volumes** attach properties to shapes, (material, and physical properties such as radiation length which are automatically calculated from the material data.)

```
G4LogicalVolume* worldLog = new G4LogicalVolume(worldBox, air, "World");

G4LogicalVolume* metalTubeLog = new G4LogicalVolume(metalTube, steel, "MetalTube");
```

- In the code, the geometry functionality is handled with sets of classes:
  - shapes
  - logical volumes
  - physical volumes



- **Physical volumes** give a position* and orientation* to logical volumes, and set which other logical volume will be their parent.
  - *Relative to their parent volume, ie. in the coordinate system of their parent.
  - The centre of each object is its origin.

```
G4double pos_x = -1.0*meter;
G4double pos_y =  0.0*meter;
G4double pos_z =  0.0*meter;

G4VPhysicalVolume* metalTubePhys = new G4PVPlacement(0,                                // no rotation
                                                     G4ThreeVector(pos_x, pos_y, pos_z), // translation position
                                                     MetalTubeLog,                     // its logical volume
                                                     "MetalTube",                      // its name
                                                     worldLog,                         // its mother (logical) volume
                                                     false,                            // no boolean operations
                                                     0);                               // its copy number
```

Materials

- Materials in Geant4 are built up from elements of the periodic table.
- Materials are defined in terms of densities, and components in terms of fraction-by-mass.

```
G4double z, a, fractionmass, density;
G4String name, symbol;
G4int ncomponents;

G4Element* elN  = new G4Element(name="Nitrogen",symbol="N" , z= 7., a= 14.01*g/mole);
G4Element* elO  = new G4Element(name="Oxygen"  ,symbol="O" , z= 8., a= 16.00*g/mole);

G4Material* Air = new G4Material(name="Air  ", density=1.290*mg/cm3, ncomponents=2);
Air->AddElement(elN, fractionmass=70*perCent);
Air->AddElement(elO, fractionmass=30*perCent);
```

- Geant4 has various materials already available.

```
G4NistManager* man = G4NistManager::Instance();

G4Material* H2O  = man->FindOrBuildMaterial("G4_WATER");
G4Material* air  = man->FindOrBuildMaterial("G4_AIR");
```

- You can also build new materials from combinations of other materials.

```
Air->AddMaterial(H2O, fractionmass=0.005); // Roughly the humidity in Oxford today
```

- If the sum of the fractions provided does not add up to 1, Geant4 will rescale all fractions.

- In Geant4, you have to specify which particles can exist in your simulations.
  If you don't list a particular particle, it wont appear in your simulations.
  - You specify which particles to include by creating a class, derived from G4VuserPhysicsList, and implementing in it the method ConstructParticle()

```
class MyPhysicsList: public G4VUserPhysicsList
{
  public:
    MyPhysicsList();
    ~MyPhysicsList();

  protected:
    // Construct particle and physics process
    void ConstructParticle();
    void ConstructProcess();
}
```

- Particles can be individually added, or added as a group with a built-in constructor.

```
void MyPhysicsList::ConstructParticle()
{
  G4Proton::ProtonDefinition();
  G4Geantino::GeantinoDefinition(); // made-up non-interacting particle for debugging

  // Construct all leptons
  G4LeptonConstructor pConstructor;
  pConstructor.ConstructParticle();
}
```

- You can create your own particles, or modify existing ones. Be careful!

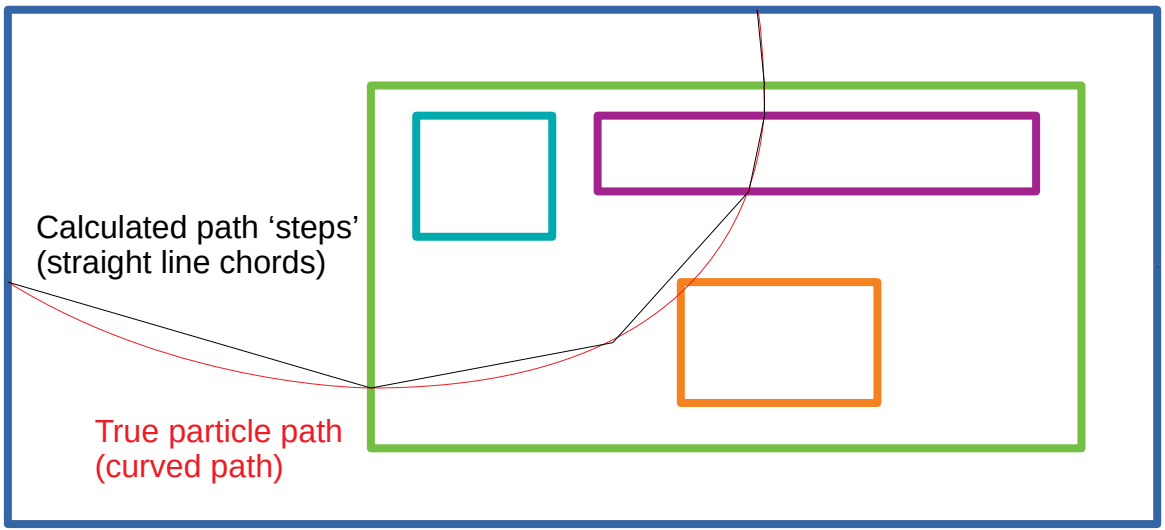
- In Geant4, you need to specify all the physics process to simulate in a ‘physics list’.
  - You do this by creating the method ConstructProcess() seen on the previous slide.

- Transportation is the moving of particles through the geometry – more on this next.

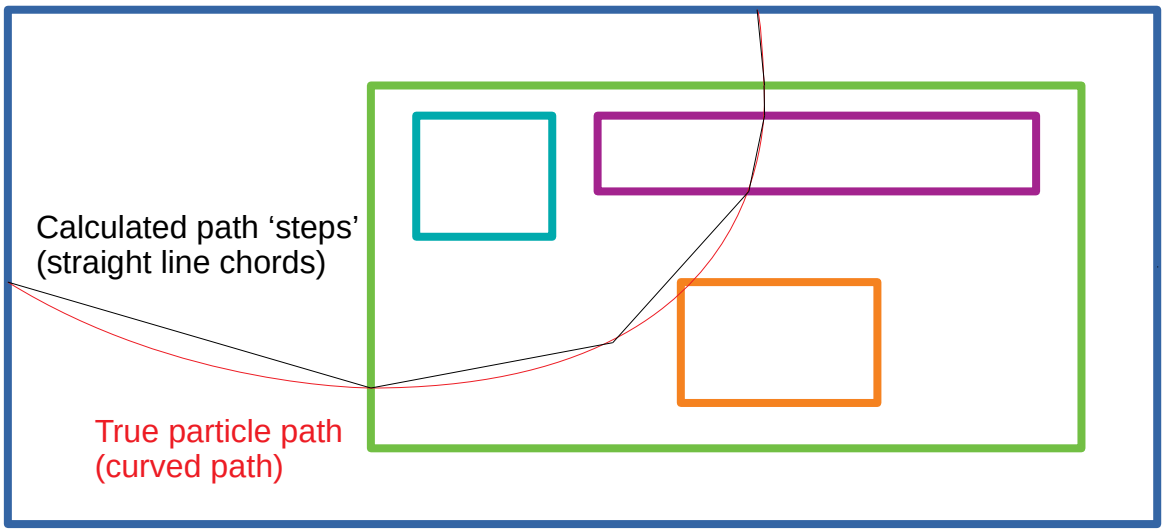
- Geant4 provides physics lists for you to use or modify.
  - ATLAS uses the Geant4 physics list ‘QGSP_BERT’ and adds transition radiation (needed for our tracker simulations).

```
void MyPhysicsList::ConstructProcess()
{
  // Define transportation process
  AddTransportation();
  // Electromagnetic processes
  ConstructEM();
}

void MyPhysicsList::ConstructEM()
{
  // Get pointer to G4PhysicsListHelper
  G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();

  // Get pointer to photon
  G4ParticleDefinition* particle = G4Gamma::GammaDefinition();

  // Construct and register processes for photon
  ph->RegisterProcess(new G4PhotoElectricEffect(), particle);
  ph->RegisterProcess(new G4ComptonScattering(), particle);
  ph->RegisterProcess(new G4GammaConversion(), particle);
  ph->RegisterProcess(new G4RayleighScattering(), particle);
}
```

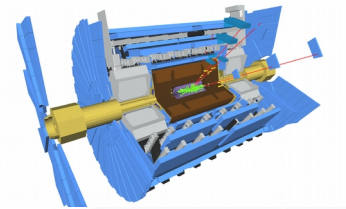
- Geant4 moves particles through the simulated geometry in steps.
  - It does this using path integration algorithms to integrate the equations of motion.
  - These algorithms are referred to as ‘steppers’.
  - By default it uses a 5th-order Runge-Kutta (Dortmund and Prince) stepper.
  - Other steppers can be used (for example where analytical solutions are known).
  - Steppers can be adjusted for accuracy/speed.



- Each step is limited to being in only one volume.
  - When the particle enters a new volume, a new step is started.
  - Thus each step is only in one material – easier for physics process calculations.
  - At each step, Geant4 must consider if the particle will enter other volumes, or exit the volume it is currently in.



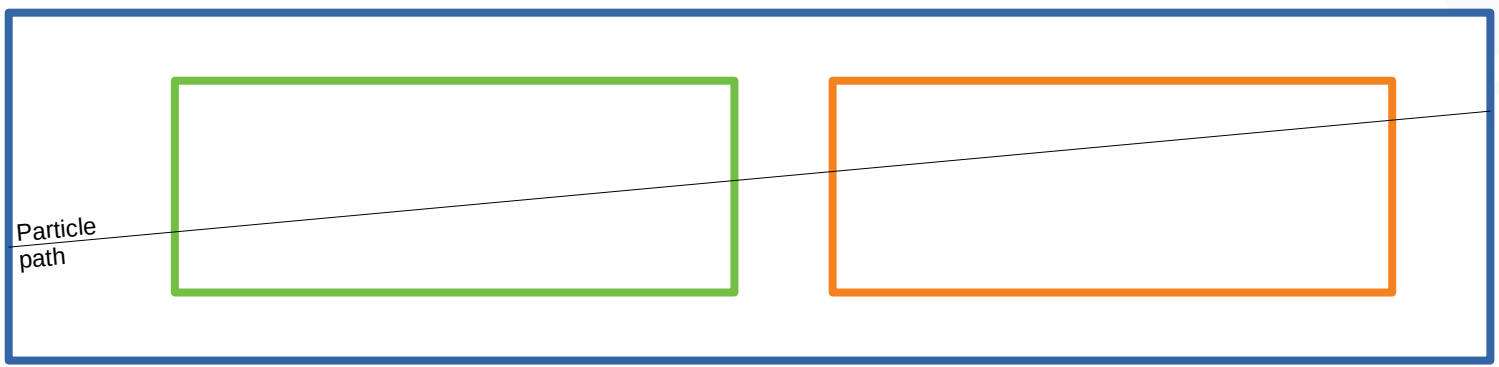
Particle Transport And Geometry Volume Hierarchy



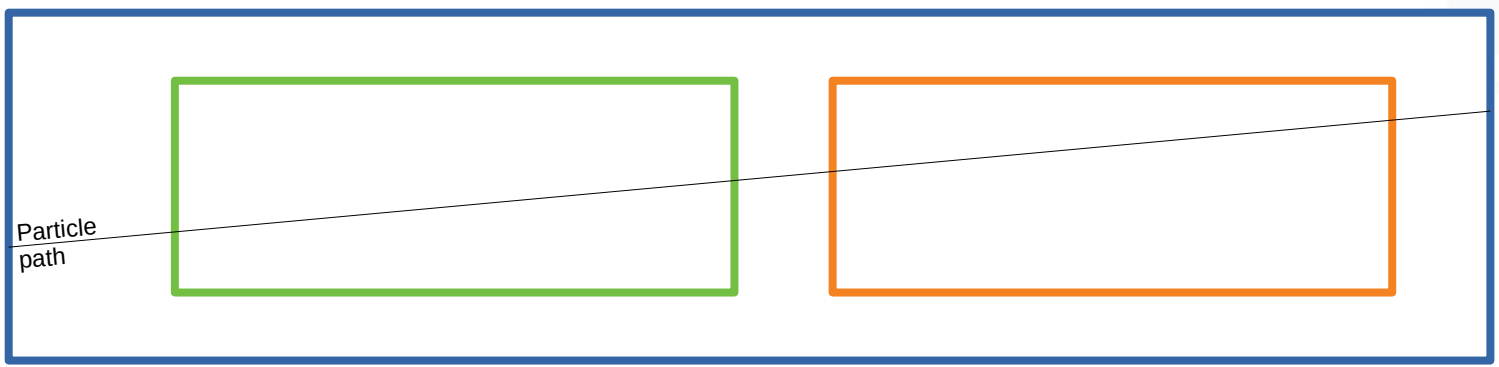
- A detailed detector description can easily end up containing many volumes, (the full ATLAS Detector simulation contains more than 4.8M volumes).

- At each step, Geant4 must consider if the particle will enter other volumes, or exit the volume it is currently in.

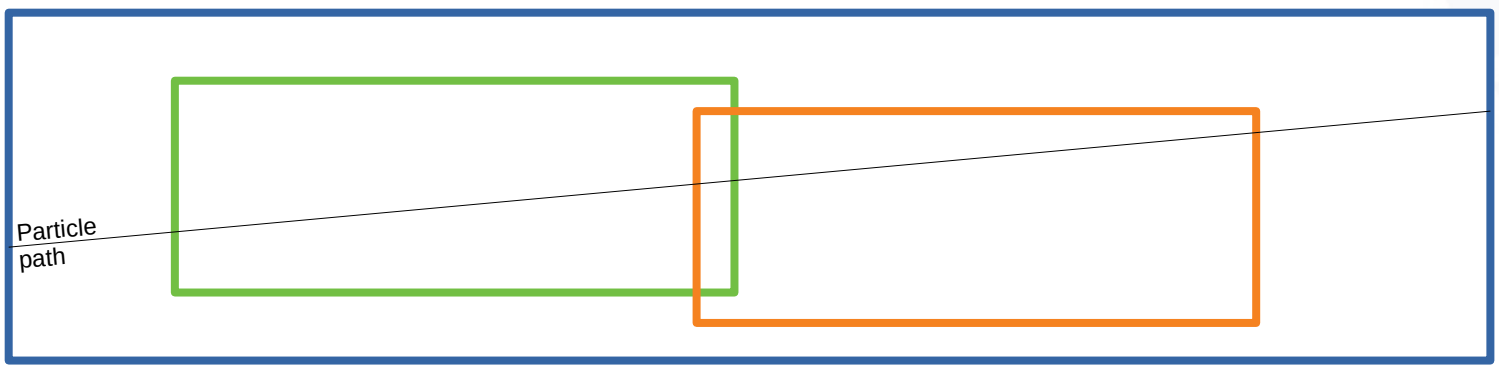
  - It is impractical to consider all possible volumes at each step.
  - Thus Geant4 only considers the **current volume** the particle is in, and **daughter volumes of the current volume**.
  - Specifically, Geant4 only checks if the particle will encounter the **surfaces** of volumes.



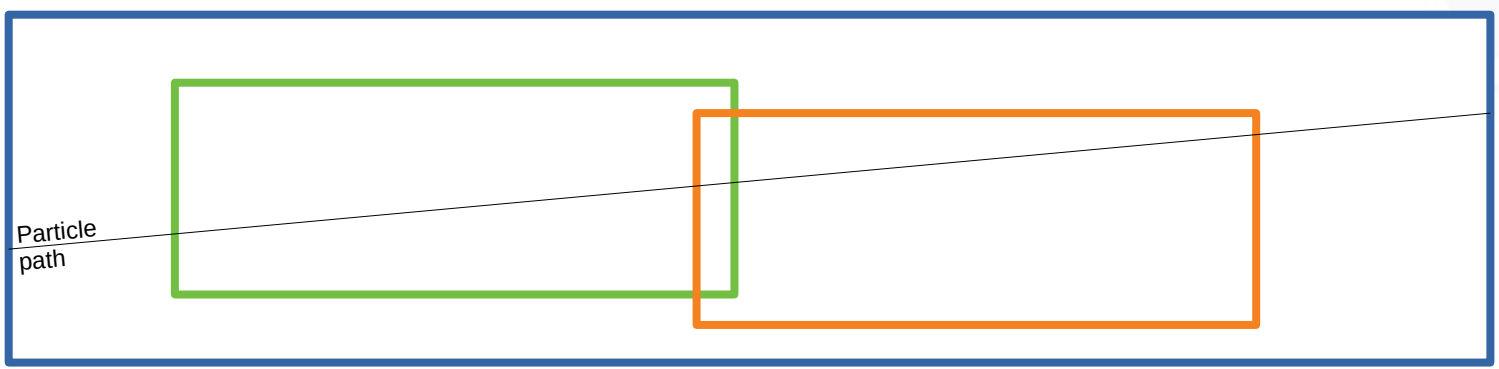
- This is fine.
- Green and orange volumes are daughters of the blue volume.
- Geant4 will calculate 5 steps, (can you identify where they will be?).



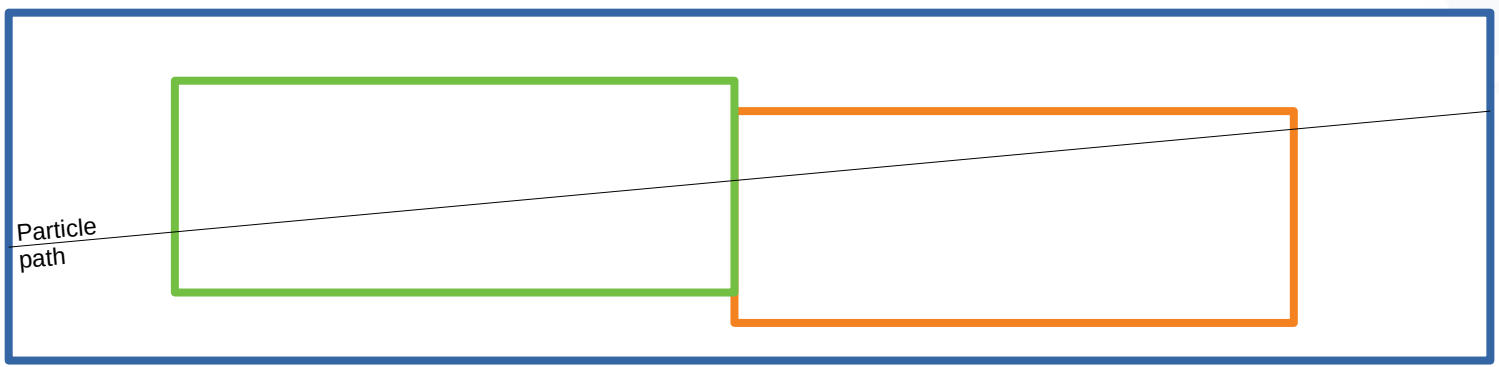
- But what if we misplace the orange volume?
- Remember: green and orange volumes are daughters of the blue volume.
- How many steps will Geant4 calculate?



- But what if we misplace the orange volume?
- Remember: green and orange volumes are daughters of the blue volume.
- How many steps will Geant4 calculate?
  - 4 steps.
  - The orange volume has effectively become invisible.
  - This is called a volume overlap.



- But what if we misplace the orange volume?
- Remember: green and orange volumes are daughters of the blue volume.
- How many steps will Geant4 calculate?
  - 4 steps.
  - The orange volume has effectively become invisible.
  - The same thing happens if the volumes are touching (surfaces at the same location).



- Electromagnetic fields can be created in Geant4.
  - These can be uniform/non-uniform, static/time-varying etc.
  - If you create a field, you just need to tell Geant4 to use it, and set up path integration tools.
    - If you want to use the default tools, then this is simple.

```
#include "G4SystemOfUnits.hh"

G4MagneticField *magField;
// Uniform field
magField = new G4UniformMagField(G4ThreeVector(0.,0.,3.0*kilogauss));
// Non-uniform field
magField = new G4QuadrupoleMagField( 1.*tesla/(1.*meter) );

// Tell the field manager to use this field
G4FieldManager* fieldMgr = G4TransportationManager::GetTransportationManager()->GetFieldManager();
fieldMgr->SetDetectorField(magField);

// Short-cut!
// Set up objects required for particle transport calculations in a field (using default Runge-Kutta stepper)
fieldMgr->CreateChordFinder(magField);
```

- If you want more control over the path integration in your field, for speed or accuracy, you can set this up:

```
#include "G4EqMagElectricField.hh"
#include "G4UniformElectricField.hh"
#include "G4DormandPrince745.hh"

// Create electric field
G4ElectricField* pEMfield = new G4UniformElectricField(G4ThreeVector(0.0,100000.0*kilovolt/cm,0.0));

// Create an equation of motion for this field
G4EqMagElectricField* pEquation = new G4EqMagElectricField(pEMfield);

G4int nvar = 8;
// Create the Runge-Kutta stepper
auto pStepper = new G4DormandPrince745( pEquation, nvar );

// Tell the field manager to use this field
G4FieldManager* fieldMgr = G4TransportationManager::GetTransportationManager()->GetFieldManager();
fieldMgr->SetDetectorField(pEMfield);

G4double minStep = 0.010*mm ; // minimum step size of 10 microns

// Set up path integration tools with your stepper
auto pIntgrationDriver = new G4IntegrationDriver<G4DormandPrince745>(minStep, pStepper, nvar);
pChordFinder = new G4ChordFinder(pIntgrationDriver);
fieldManager->SetChordFinder( pChordFinder );
```

Sensors, Sensitive Regions, And Hits

- We want to know how our detector would respond to physics that we simulate.
- We can specify certain logical volumes to be ‘sensitive detectors’ (those volumes representing the active regions of our sensors).
- When a particle passes through a volume marked as a sensitive detector, a ‘hit’ will be registered, and information related to the step (path integration step) can be recorded:
  - the position and time of the step,
  - the momentum and energy of the track,
  - the energy deposition of the step,
  - geometrical information,
- You need to make classes (inherited from abstract base classes in Geant4) for:
  - A **hit**, to store the desired information for one hit
  - A **hit collection**, a vector to store your hits in.
  - A **sensitive detector**, to assign to a logical volume and create hit objects.
- You need to define classes for each type of hit you want (each sensor type in your detector).

Inputs To Simulations

- When running simulations, you will simulate a series of ‘events’. A series of events is a ‘run’.
  - An ‘event’ is defined by some initial conditions and input particles, which are then evolved in time (a simulation).
- An event could be one proton-proton collision at the LHC, for example.
  - In this case, you probably want to use the output particles from a Monte Carlo generator (see Andy Buckley’s lecture on Tuesday) as the input particles to your simulation.
  - Geant4 provides an interface for this.
- You can also generate input particles in Geant4 using Geant4’s G4GeneralParticleSource.
  - For example, G4ParticleGun generates one particle per event.
  - With G4ParticleGun you specify the initial particle’s position, momentum, etc.

Hooks

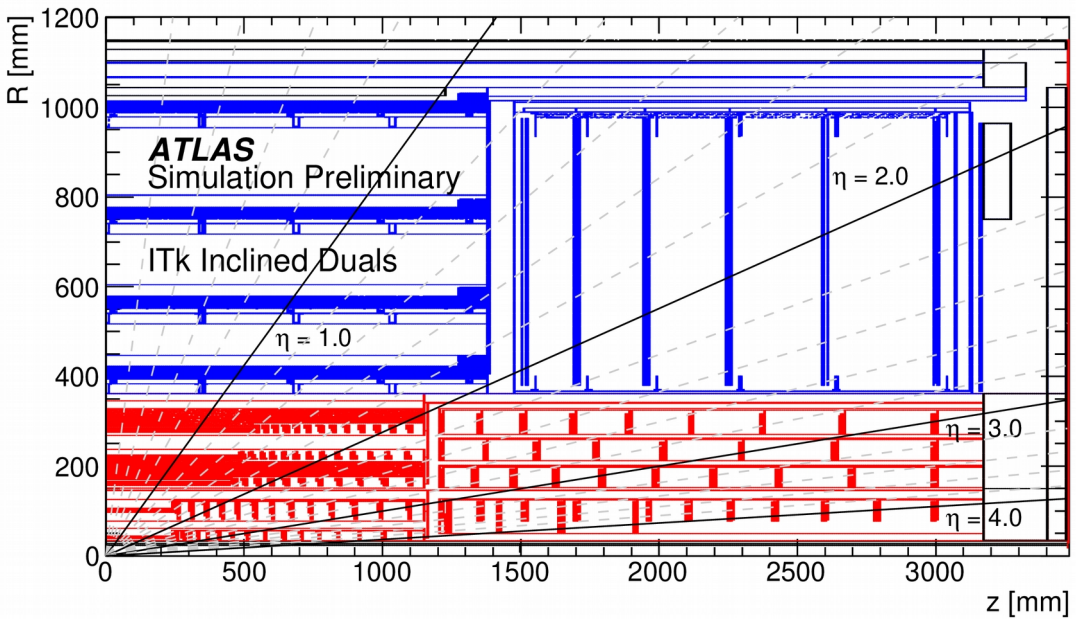
- Hooks are ways for developers to add code into Geant4 processes to add functionality.
  - You can create and fill histograms.
  - You can inspect and store information during a simulation.
  - You can do whatever you want, really.
  - You can even modify the simulation in progress, (Danger!).

- 5 classes of hooks are available:
  - **G4UserRunAction**
  - **G4UserEventAction**
  - **G4UserStackingAction**
  - **G4UserTrackingAction**
  - **G4UserSteppingAction**

Hooks – Example

- You could: use **G4UserRunAction** to create a vector for histograms at the start of the run.

- Use **G4UserSteppingAction** to access each step (pointer to each G4Step object) as soon as it is processed.
  - Can store the positions of steps in a 2D histogram.
  - Useful for debugging models.

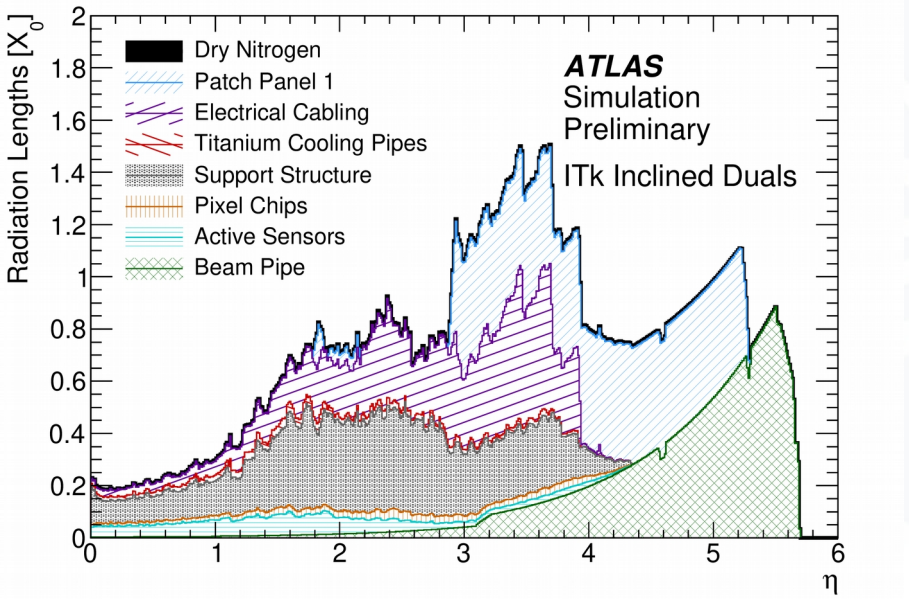


- Use **G4UserRunAction** to write out all histograms to a file at the end of the run.

Hooks – Example

- You could: use **G4UserRunAction** to create a vector for histograms at the start of the run.

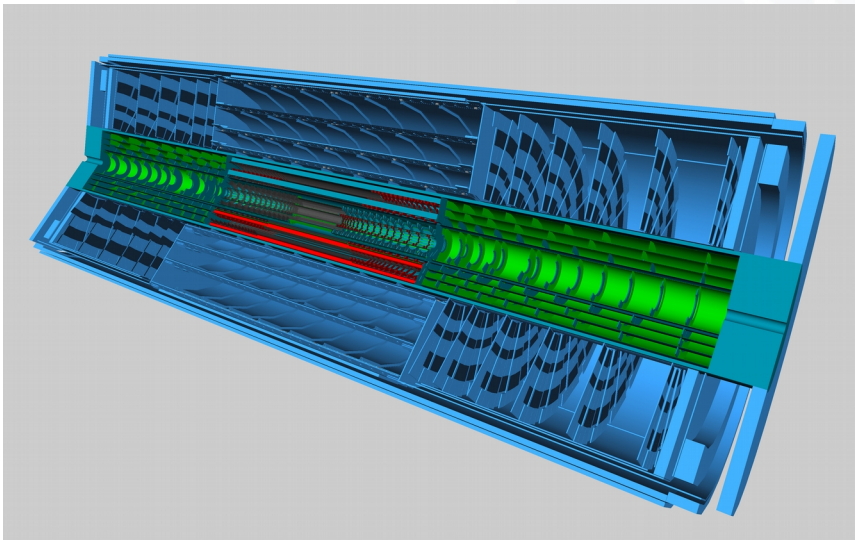
- Use **G4UserSteppingAction** to access each step (pointer to each G4Step object) as soon as it is processed.
  - Find out the step length d, and the radiation length X_0 of the material the step is in.
  - Calculate how many radiation lengths particles have to pass through (d/X_0) for each material.
  - Store this in a histogram, (one histogram for each material).



- Use **G4UserRunAction** to write out all histograms to a file at the end of the run.

Validation And Visualisation

- Validation is crucial when building simulations.
- Ideally you would compare you simulations to data, but this is not always possible.
- There are various validation methods – use them all!
- Visualisation can be a useful tool for inspecting and debugging a simulation model, as well as a way of making nice images for talks and papers.
  - 2D material maps as seen previously.
  - Geant4 includes 3D visualisation tools.
  - Powerful custom tools also exist, (VP1 if you're on the ATLAS experiment).
  - You can also export a model geometry, and import it into Blender for rendering.



- Other validation methods include:
  - Overlap tests:
    - Geant4 provides a tool to check for volume overlaps,
      via the **G4PVPlacement** and **G4PVParameterised** classes:

```
G4bool CheckOverlaps(G4int res=1000, G4double tol=0., G4bool verbose=true);
```

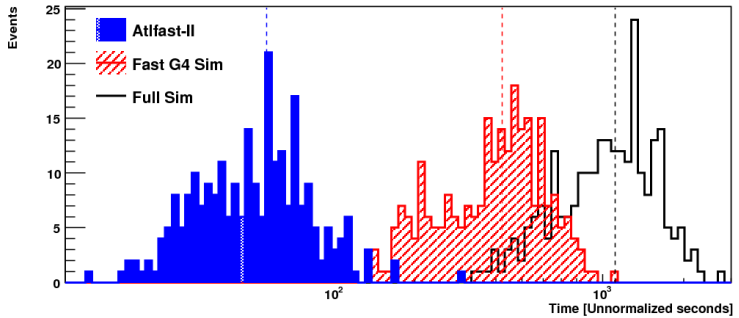
    - Can be very CPU intensive – only switch on for debugging.
  - Mass checks:
    - Often you will know the intended mass of detector components, (from engineering).
    - Geant4 does not store the masses of volume,
      but you can get the volume (in g/mm^3) of a volume,
      and the density of its material.
    - You will need to write a mass calculation tool that loops over volumes in the **G4PhysicalVolumeStore** and returns all their masses, for comparison.

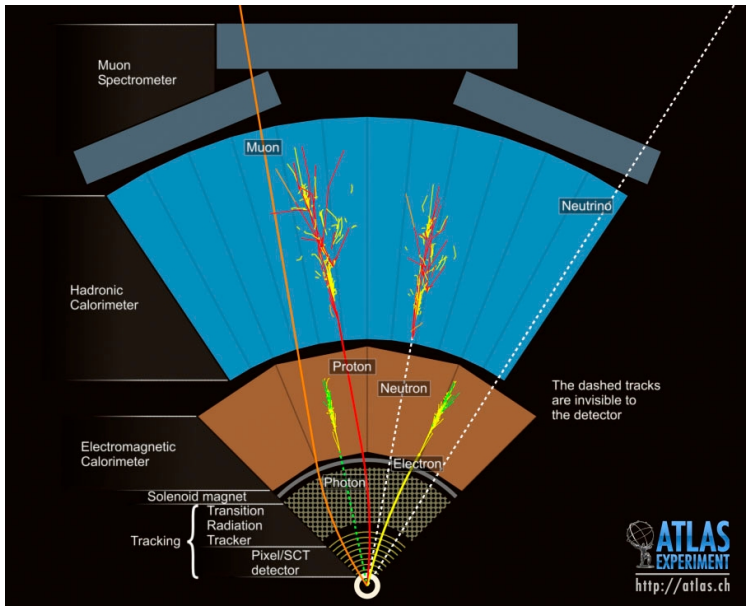
Alternative Usage – Geant4MT

- There are alternative ways of running and using Geant4. Here we'll briefly cover a few.
  - Geant4MT is multi-threaded Geant4.
  - You can take regular Geant4 code, and modify it to initialise multi-threading, and specify which bits can be run in parallel, and which bits must not be parallelised.
  - Multi-threading overhead is small, so Geant4MT **should** be faster than Geant4.
  - Certain actions can only be performed by one thread at once, so if multiple threads try to perform this action, all threads will be locked and will effectively run in series.
    - In this case, Geant4MT can be slower than regular Geant4.
    - Histogram filling is the usual culprit of this.

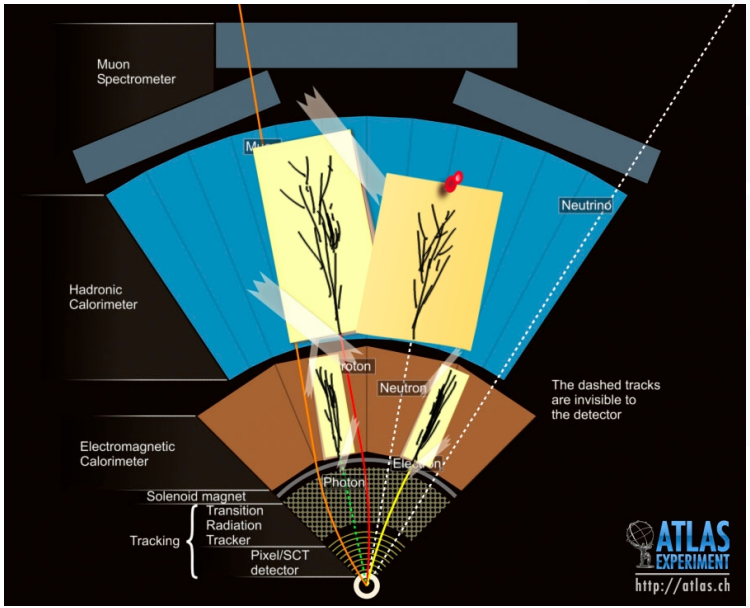
Alternative Usage – Fast Simulation

- In fast simulation, instead of fully simulating things such as hadronic and electromagnetic showers, when the start of a shower is detected, a suitable pre-simulated shower is chosen at random from a selection and ‘pasted’ into the simulation results.





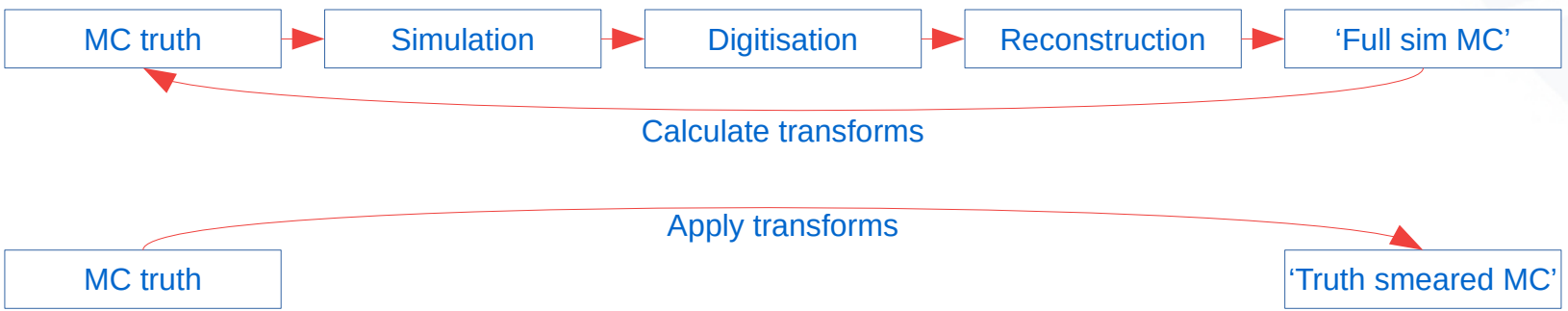
‘Full Simulation’



‘Fast Simulation’

Alternative Usage – Truth Smearing

- For some scenarios, it will be impractical to simulate events.
  - For example, events with 200 pile-up collisions are very computationally expensive, so it is not yet practical to fully simulate and digitise these events for all physics analyses.
- What can be done instead is that, for some events, the output of monte carlo generators (called ‘MC truth’) can be fully simulated, digitised, and reconstructed.
- Transforms can then be calculated that map from MC truth directly to reconstructed objects.
- These transforms can be applied to other MC truth events to create pseudo-reconstructed objects. This process is called ‘truth smearing’.
- In ATLAS we have the ‘Upgrade Performance Functions’ available for truth smearing.



A Note On Digitisation And Pile-Up

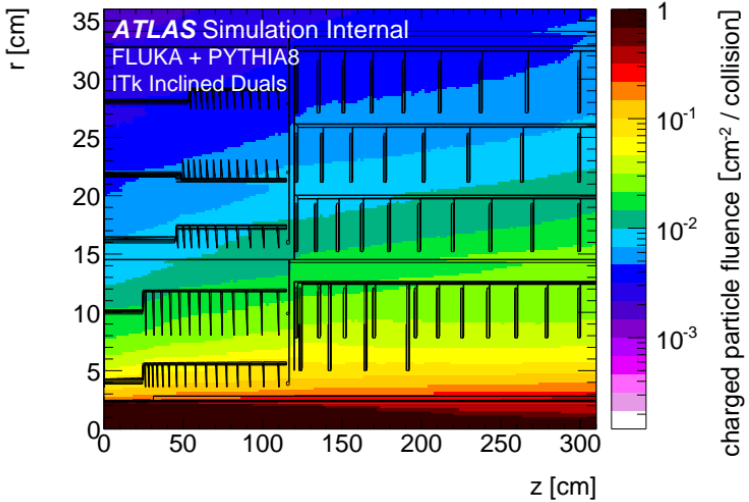
- Digitisation is the process of taking simulation hit data
  and turning it into signals/data equivalent to what the real detector would output.
    - This essentially means modelling, in a simple way, the detector electronics.

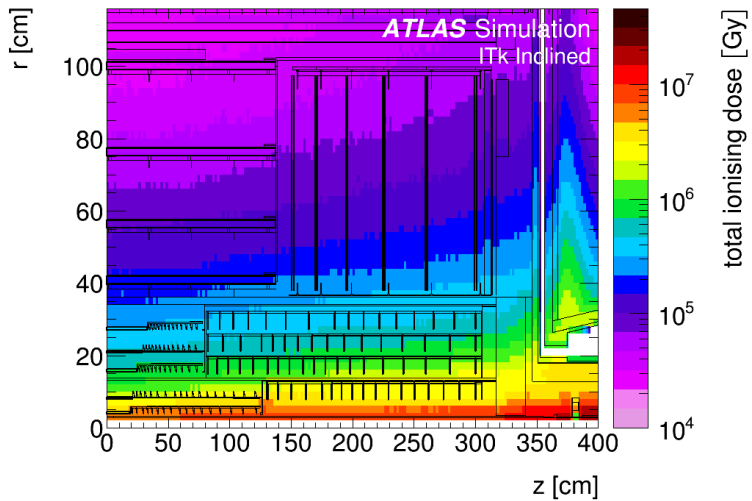
- Pile-up is when there are multiple proton-proton collisions per bunch crossing.
    - Rather than simulate all the proton-proton collisions at the same time in one simulation event, we can simulate:
        - One ‘signal’ event (a process of interest)
        - Many ‘minimum bias’ events, each corresponding to one proton-proton collision.
    - We can then load all these events into digitisation, and digitise them all at once.
        - This requires a lot of RAM!

FLUKA

- FLUKA is a different simulation program.

- It's very good for calculating things like fluences and total integrated ionising radiation dose.
- It generally requires simpler models/geometries than Geant4.
- FLUGG (FLUKA + Geant4 Geometry) allows you to import your Geant4 models.





Lessons From The Front Lines Of HEP Simulation

- I’d like to include some hints and tips, from ATLAS ITk (future tracking detector) simulation.

- Xml input
- Models generated on the fly
- Geometries stored in databases - software version independent
- Mass inputs - from engineering
- Sources of error
- Things to focus on in models, and things to simplify
- Uncertainty calculations (material, misalignment)
- Validation, validation, validation

- Xml input:
- Current ATLAS simulation stores all model information directly in an SQL database.

LArMaterials:

MATERIAL	DENSITY	COMPONENT	FRACTION
AnodeMat	3.06	LAr::FR4	.519
		Copper	.481
BoardsEnvelope	.932	LAr::FEBoards	.4147
		std::Air	.0008
		std::Water	.0736
		Aluminium	.5109
BoratedPolyethelyne	.99	Boron	.05
		Carbon	.813
		Hydrogen	.137

- For the ITk, we have moved to having models described in human-readable xml.
  - The xml is read in at the start of a simulation run, and the model is built on-the-fly.
  - Models can be changed in the xml, and simulation re-run, without recompiling code.
  - Xml files can be stored in the database, (as SQL CLOBs).
  - This has made it easier for non-expert users to develop and cross-check simulation models.

- Models generated on the fly:
- Instead of defining all volumes exactly in the input xml,
  you can write code to determine volume sizes, locations, and positions at build time,
  based on other inputs.
  - Example: modelling electrical cables and cooling pipes.
    - Paths are defined for these, “along the back of each stave”,
      “from the end of each stave, out to the radius of object X”, etc.
    - This meant that when a different detector layout is tested,
      (different stave length or position, for example),
      the cables and pipes will automatically be recomputed.
    - Allows much faster development, but requires very careful debugging!

- Geometries stored in a database – software version independence:
- By storing model inputs and geometries in a database,
  you can disentangle geometry versions from software versions.
    - This means that you can retest older models/geometries with newer software.

- Mass inputs - from engineering:

- When you create simulation models,
  you will often be basing these models on engineering designs.

  - These designs (or the engineers who make them) will typically give the masses of components, (not the densities of components).
  - Also, if you're modelling something that exists,
    it's easier to weigh it than calculate its density.

  - Geant4 specifies materials by their density.

  - A way around this is to specify the mass and material of a volume/object,
    and then write code that, when the model is being built,
    will ask the G4Volume what its volume is (in mm3),
    and will create a new material (a copy of the given material)
    with the correct density to give the desired total mass,
    and then assign this new material to the logical volume.

- Sources of error:
- It is worth thinking about how information passes through the full design chain.
  - Engineers → Physicists:
    - Engineers explain designs to physicists.
    - There may be a language/jargon barrier, which could lead to misunderstandings.
  - Physicists → input files:
    - Have the designs the physicists intended/understood been correctly implemented in the input files?
  - Input files → simulation model:
    - Does the simulation model match what the input files represent?
    - (Are there bugs in your code?)

- Things to focus on in models, and things to simplify:
- The more complex your model, the slower it will run, and the harder it will be to debug.
- Think about which materials and components will have a large impact on physics.
  - Cables and services can typically be simplified into single blocks containing the required elements, and of the correct total mass.
  - Electronics can be modelled as a volume of fibreglass (PCB) and a volume of copper (thin layer). Be sure to get the correct total mass of copper!
  - Materials in front of sensors will typically have larger effects, and must be more accurately modelled.
  - Be weary of any electronics or cables in front of sensors that are under development:
    - Revisit engineering designs periodically.
    - Designs tend to get heavier over time, rarely lighter.

- Uncertainty calculations (material, misalignment):

- Misalignments are nothing new – nothing is ever built with infinite precision.
  - Think what misalignments might affect physics results.
  - How can you introduce misalignments into your simulation model to test this?

- Until a detector is built, materials and masses can only be estimated.
- Designs will change over time, and usually they get heavier.
  - Think about which materials/masses will most affect physics.
  - Vary the masses/densities of these components up and down, to understand how uncertainties on materials and component designs will affect physics performance.
  - This can tell you which components need to have their designs tightly controlled to maximise physics performance.

- Validation, validation, validation:
- Have multiple people double-check information-flow at each stage of the design chain.
- Use all debugging tools at your disposal.
  - Once you feel a model is complete:
    1. Volume overlap checks are useful to start with.
    2. Visual and material/X_0 checks can tell if anything is missing.
    3. Mass comparisons with engineering estimates can highlight missing material.
    4. Basic performance studies can highlight missing hits, odd performance.
       Are features and changes in performance understood?
    5. Have changes to your model (your subdetector, say)
       affected simulations of neighbouring subdetectors?
       Talk to people to check.
- You can never do enough validation! (so at some point you have to decide to stop)

Coda

- Simulation is a powerful tool for high energy physics and beyond.
  - If you go down into all the depths of it, it can be quite a complicated beast.
  - But typically you don’t need to deal with all the complexities.
  - With all it can do, you can do a lot of studies and developments.

- Working in simulation gives you a wide overview of an experiment.
  - Worth noting – if you like modelling / 3D design / building things, you’ll likely really enjoy HEP simulation. (My observation)

“The life of a designer is no parade of victories.
There are innumerably more failures.
But they must not stop us.”

- ostensibly Sergei Korolev / Public Service Broadcasting