GeantV - Simulation in future Particle Physics Experiments

V. Daniel Elvira, SCD-Simulation Department Head, SCD-CMS Department Head
Fermi National Accelerator Laboratory

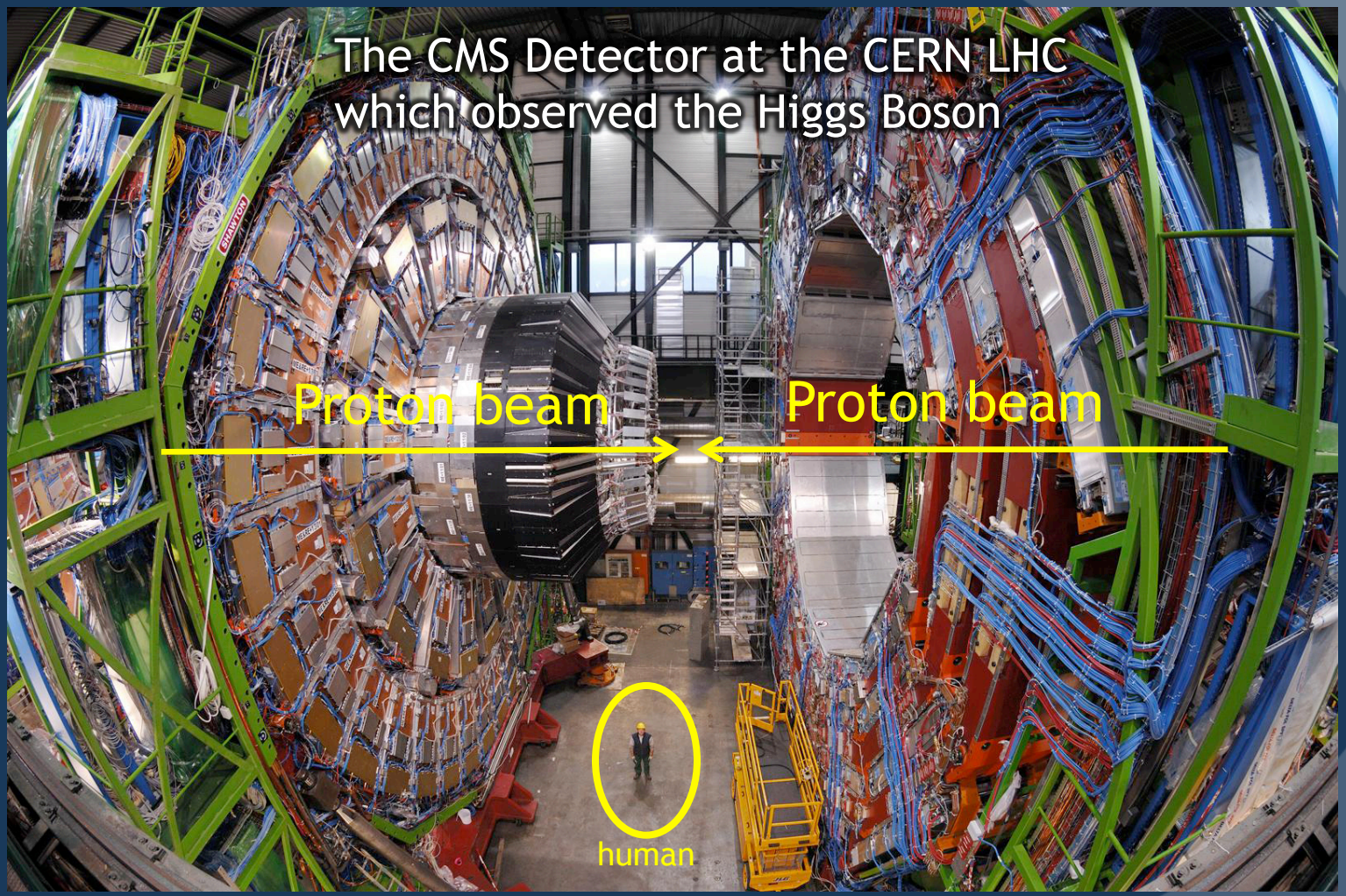
This talk is about …

… a *multidisciplinary* project (physicists and computer scientists)

to

re-design from the grounds up the most popular detector simulation software tool in the world: "Geant"

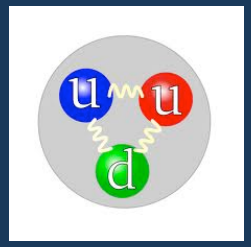
High Energy Physics Detectors



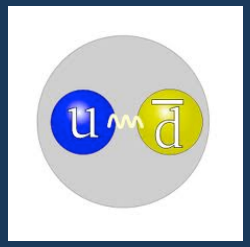
Particles and Interactions

Protons, neutrons, electrons form the atoms

Hadrons (Baryons – protons, neutrons), mesons made of 2, 3 quarks):



Baryon
(i.e. proton)



Meson
(i.e. pion)

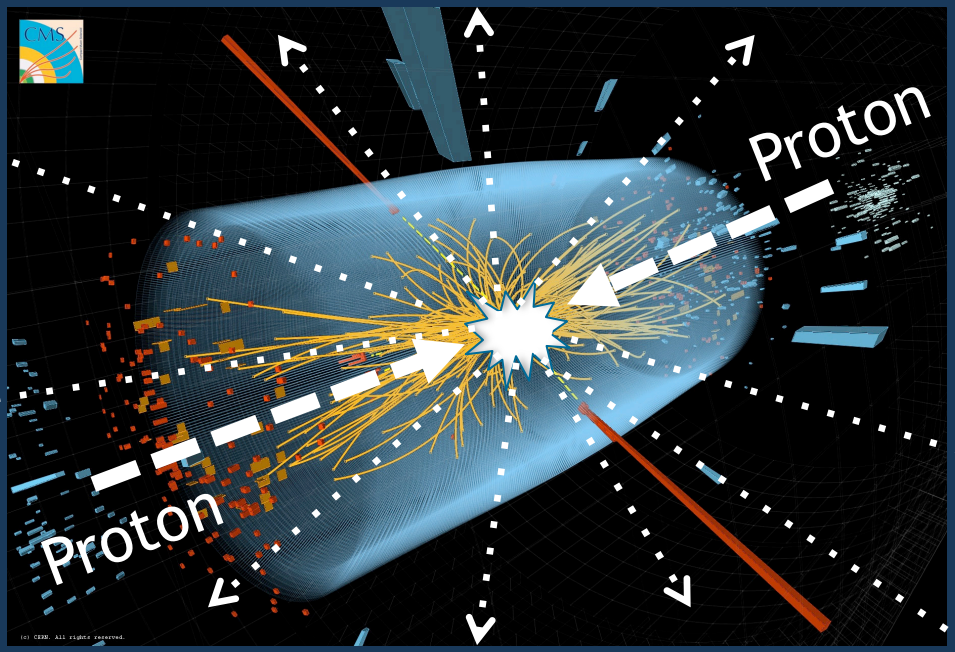
Muons, pions, electrons, protons, neutrons, neutrinos, … are found in cosmic ray showers, and produced in particle accelerators

Particles with electric charge, photons ➔ EM interactions

Particles with quarks (and gluons) ➔ Nuclear interactions

A High Energy Physics Event

Example:
CMS Experiment



Higgs

Quark jets

Gluon jets

Photon

Dark Matter ?

Z Boson

W Boson

Electron

Muon

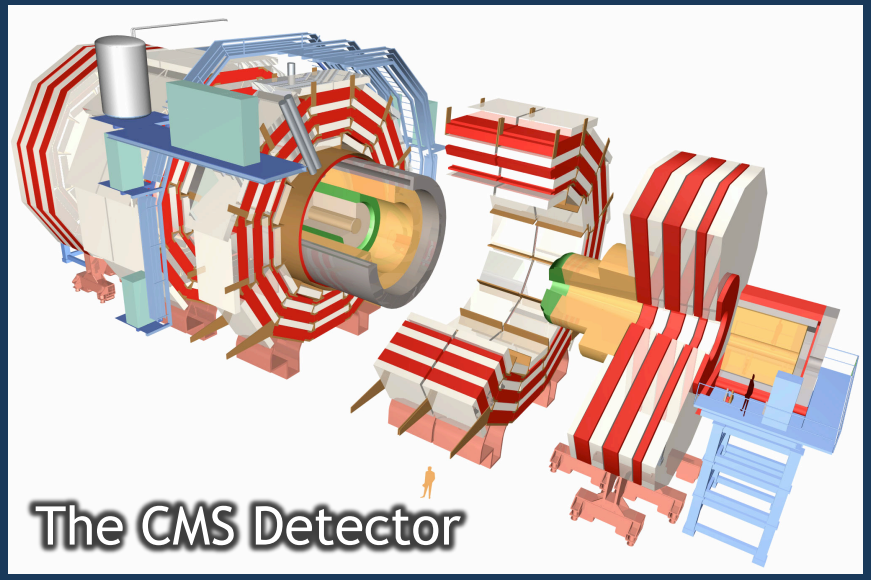
Tau

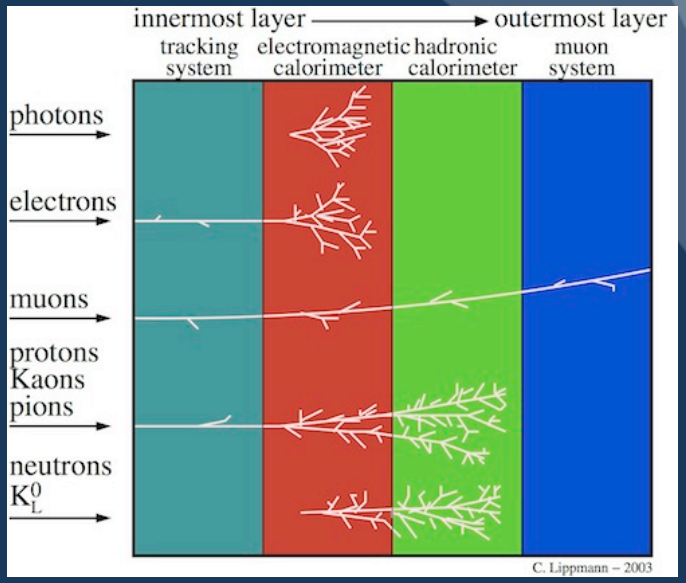
Collider event: all the detector data associated with a single pp collision

Tens to hundreds of these “primary particles”

Particle interactions in Detectors

Incoming particle interactions with the detector results in:



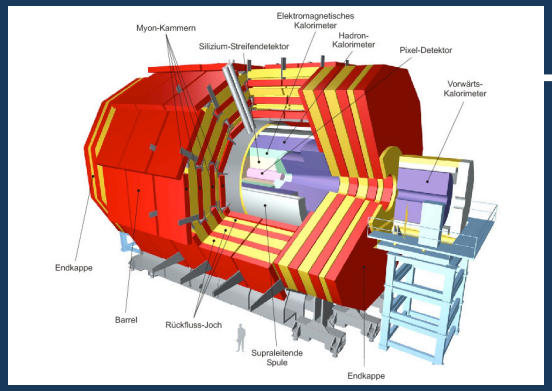


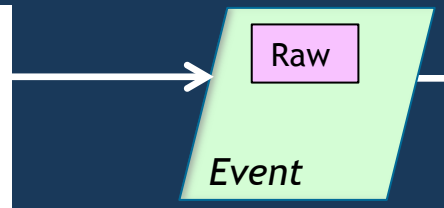
Energy loss, gas ionization, "showers" of secondary particles produced through EM and nuclear interactions

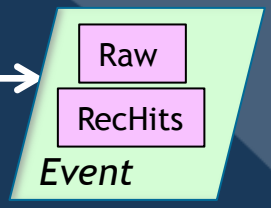
Charges and voltages collected by millions of electronic channels ➔ positions, momenta, energy, time measurements

Data Processing

Example:
CMS Experiment

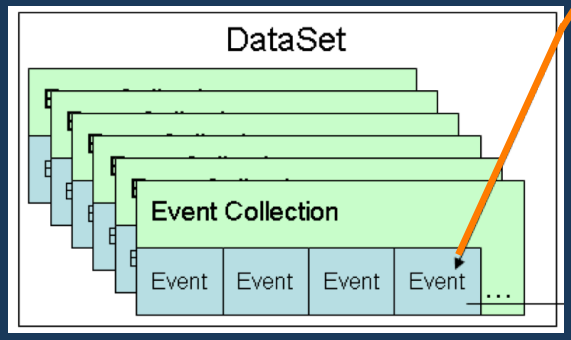






Voltages/charges - individual electronic channels

Calibration – V/Q to distance, energy, momentum



Software algorithms transform voltages/charges into reconstructed particles (electrons, photons, muons, hadrons & mesons, jets of particles)

A Reconstructed Event contains the data (position, momentum, energy, time) associated with all particles produced in the (pp) collision

Why Simulate Detectors ?

Simulation saves time and money and improves the quality and accuracy of physics measurements

- Detector design and optimization
- Software development, including reconstruction algorithms
- Design of methods for calibration and data analysis
- Detector studies of new particles as predicted by theories

Simulation helps us design the optimal detector, for best physics at a given cost, even before we fasten the first screws !

Ideal detector simulation software – produce output data identical to what comes out of the real detector

The Geant4 Simulation Toolkit

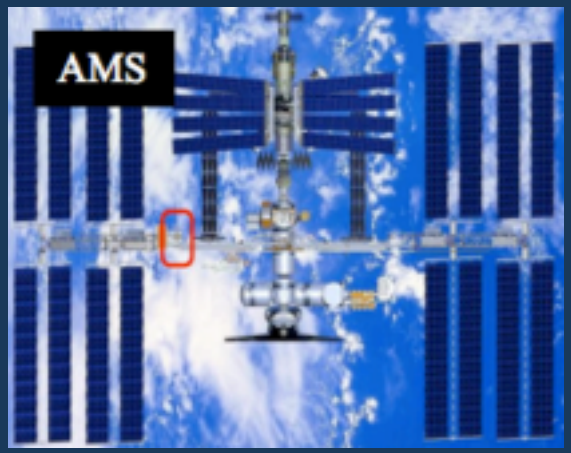
Geant 4 is a software toolkit to simulate the passage of particles through matter and EM fields

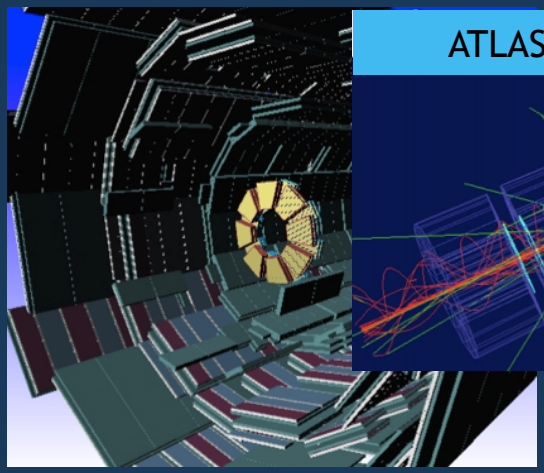
- International Collaboration of tens of institutions and ~120 physicists and Computer Scientists, including FNAL/CERN/SLAC
- Written in OO C++, > 1 million lines of code, > 2000 classes
- Used by almost all HEP experiments (10,000 users), space, and medical applications

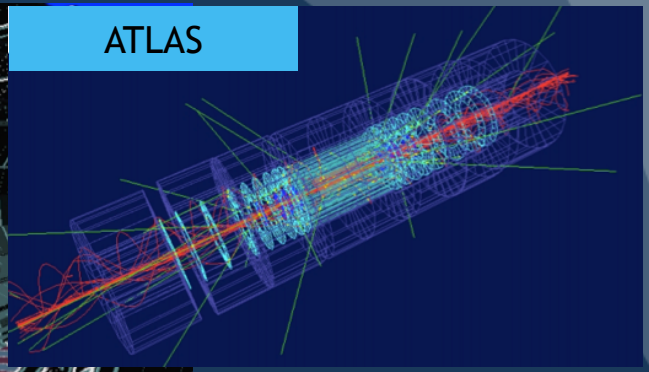
20th G4 Collaboration meeting at Fermilab, USA (2015)

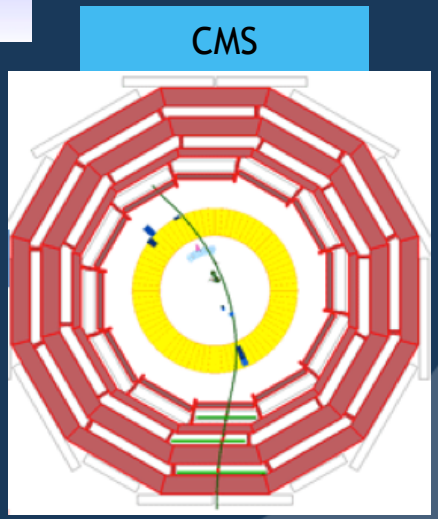
21st G4 Collaboration meeting in Ferrara, Italy (2016)

The Geant4 Simulation Toolkit





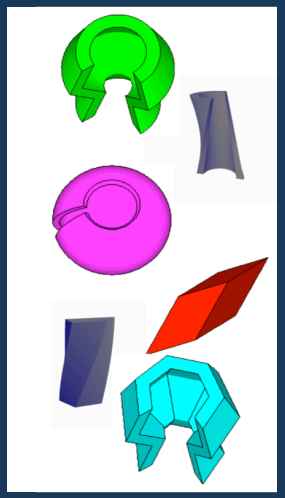




The Geant4 Simulation Toolkit

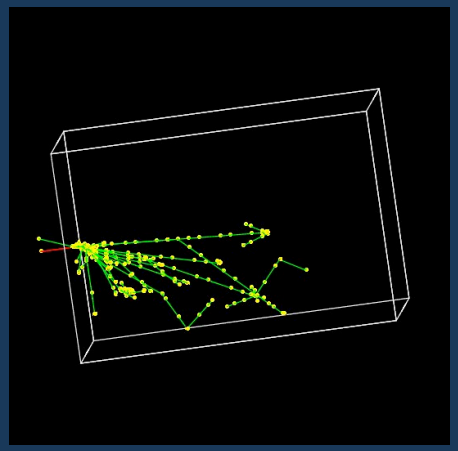
Experiments develop a simulation package for their detector using Geant4 by assembling each of the following elements:

Detector geometry (shapes and material)



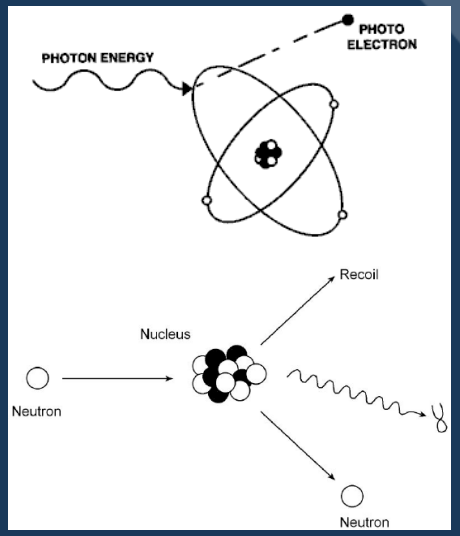
+

Particle Propagation through geometry and EM fields



+

Physics Processes



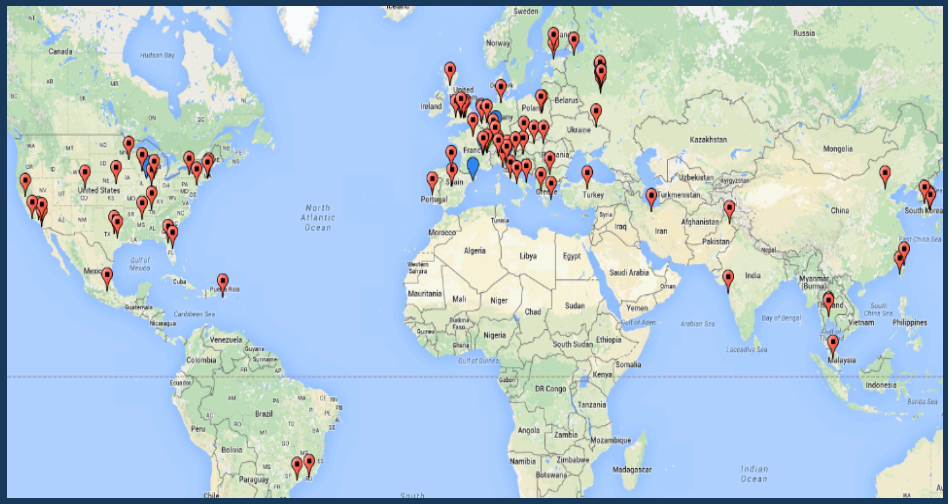
Simulation: accurate modeling of the detector ➔ accurate simulation of the experiment physics observables

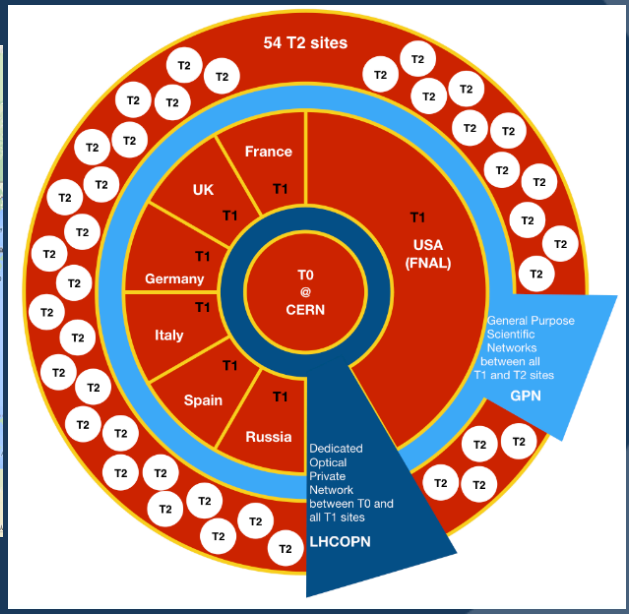
Goal: simulation output indistinguishable from real data

Computing in High Energy Physics

HEP experiments are not just large devices that collect signals with fancy electronics, they also involve:

Large distributed computing systems running *millions of lines of code*, processing *many petabytes of data* every year which are transferred for physics analysis to every corner of the world





A HEP Experiment in Numbers

HEP experiment data is BIG DATA ! For example, in CMS:

- ~10 billion real collider events and ~10 billion simulated events were reconstructed and stored during 2010-2012

  (150 times more data expected through 2030)
- Event size: ~ 1 MB for real data, ~ 1.5 MB for simulated data
- CPU time for simulated event ~ 15 sec – 3 min (~1 min for typical event), plus ~30 sec for reconstruction
- Memory usage/event ~ 2 GB, data transfers ~ 2 PB/week

(A total of ~150k cores around the world, running SL5.X, C++ code, i.e. Intel Sandy Bridge X86 machines)

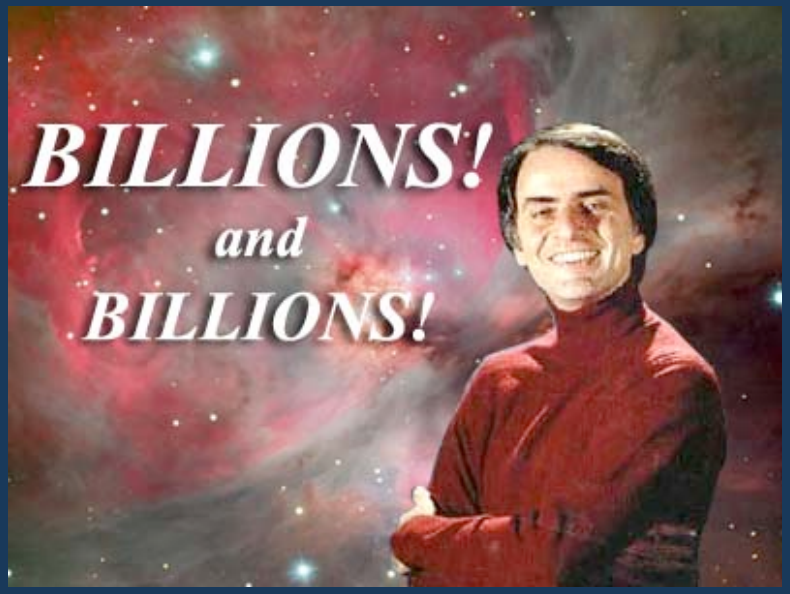
LHC experiments in 2025-2035 will need 10-100 times more computing, depending on the solutions implemented for data processing/handling

Houston, we have a problem …

Not enough money, computers not fast enough …

Houston, we have a problem …

… to generate and reconstruct …



… of simulated events needed for physics analysis

Explore the benefits of new computing and software technology: not a choice but a necessity

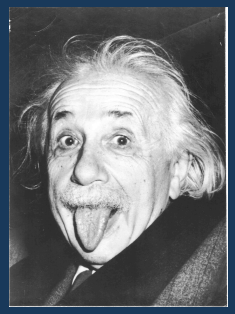
Increasing Demand in a Flat Budget

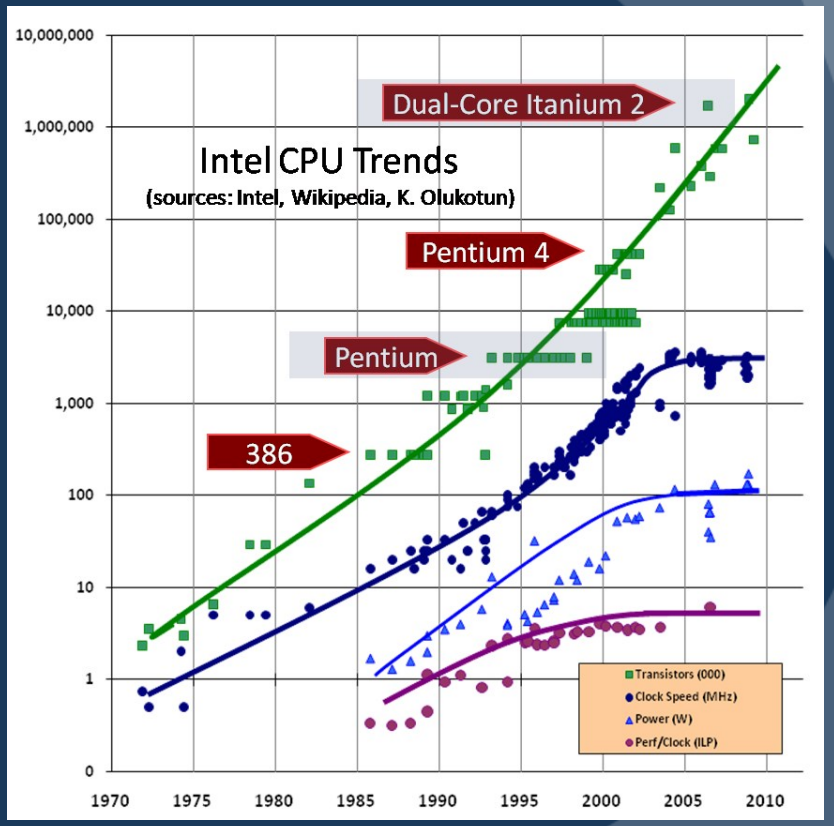
Transistor density growth is keeping up (with Moore's law – doubling every 2 yrs) but clock speed has been flat since ~2003

Leverage from growth in:

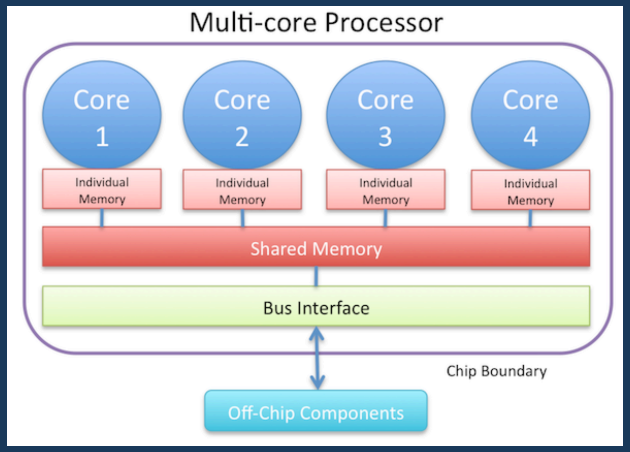
- Core count (multi-core machines)
- Coprocessors
- Concurrency and parallel computing

What does this mean?





New Era in Computing Technology



Multi-core processors

Multiple cores (CPUs) in a single chip package, 2-10 cores, individual and shared memory – *parallel and concurrent programming*

- Multi-processing refers to the execution of many processes of the same program in *parallel* (at the same time) on different CPU cores or *concurrently* (alternating) in a single core

  One event per process per core in parallel

- Multi-threading refers to the execution of multiple threads of a single process in one core or in multiple cores.

  one event per thread per core in parallel. Shared memory (instructions, detector geometry, physics data)

Coprocessors and Hybrid Computing

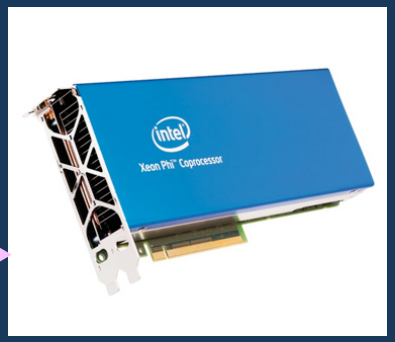
Coprocessors

CPU core supplemented with *coprocessors (accelerators) that specialize in operations* such as floating point math, graphics, etc.

- A *CPU host core* manages memory, fetches instructions, does flow control and i/o operations
- Examples are Intel Xeon Phi architecture, NVIDIA Graphics Processing Unit (GPU) – called GPGPU when used for general purposes

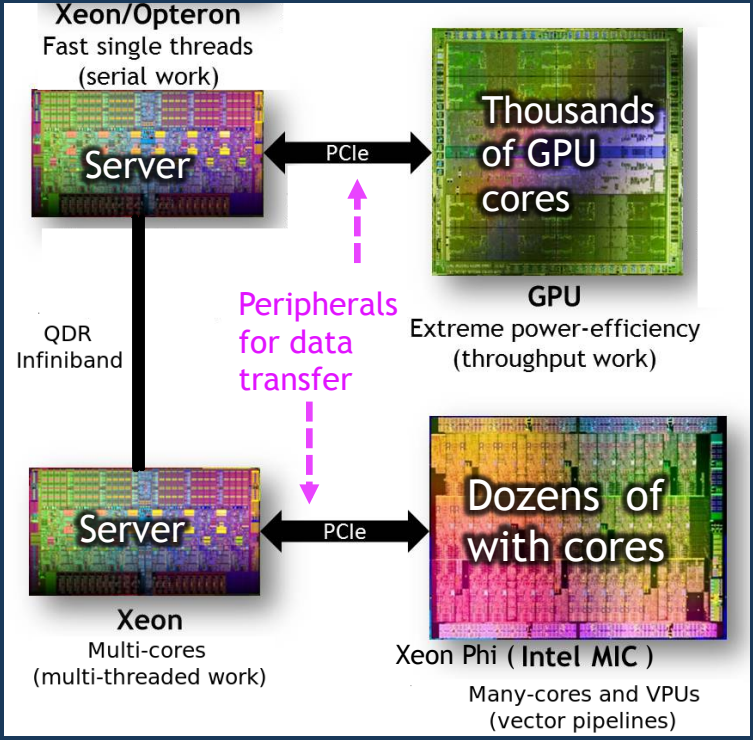
NVIDIA Tesla GPGPU

Intel Xeon Phi



Hybrid Computing Model

Hybrid computing means sharing work across a mixture of computers with different architectures



Servers control GPU and Xeon Phi devices, cooperate with each other

GPU, Xeon Phi share memory within each device, offload FP operations from CPU

Software frameworks for writing programs that execute across heterogeneous platforms (CPU, GPGPU)

CUDA by NVIDIA,
OpenCL by Khronos group

Simulation in HPC Systems

How can simulation benefit from HPC systems ?
Non-trivial question (R&D) ➔ What we know:

- Event level multi-processing ➔ time performance improvement directly proportional to the money spent
- Event level multi-threading ➔ memory savings (G4.10 series)
- *Coprocessors* may bring additional time/memory savings for the same money
- *Geant4 was not designed* to benefit from vector organization of data, data locality, concurrency and parallelism.

GeantV is designed from the grounds up for optimal performance in new computing architectures

The GeantV Project

R&D to explore the use of modern computing architecture for G4 re-engineering started in 2012

- At a small scale, at FNAL and CERN independently
- In 2014, FNAL and CERN joined forces to develop GeantV

GeantV - Next generation detector simulation toolkit

- At least 2-5 times faster than Geant4
- Improved physics accuracy
- Full simulation and various fast simulation options
- Portable to new generation of computing architectures (CPUs, GPUs, Xeon Phi's, etc.)

Understand limiting factors for 10x time performance gain

The GeantV Team

G.Amadio (UNESP), A.Ananya (CERN), J.Apostolakis (CERN) , A.Arora (CERN), M.Bandieramonte (CERN), A.Bhattacharyya (BARC), C.Bianchini (UNESP), R.Brun (CERN), Ph.Canal (FNAL), F.Carminati (CERN), L.Duhem (intel), D.Elvira (FNAL), A.Gheata (CERN), M.Gheata (CERN), I.Goulas (CERN), F.Hariri (CERN), R.Iope (UNESP), S.Y.Jun (FNAL), H.Kumawat (BARC), G.Lima (FNAL), A.Mohanty (BARC), T.Nikitina (CERN), M.Novak (CERN), W.Pokorski (CERN), A.Ribon (CERN), R.Sehgal (BARC), O.Shadura (CERN), S.Vallecorsa (CERN), S.Wenzel (CERN), Y.Zhang (CERN)

Institutions:

- Bhabha Atomic Research Centre (BARC, India)
- European Center of Nuclear Research (CERN)
- Fermi National Accelerator Laboratory (USA)
- Universidade Estadual Paulista (UNESP, Brazil)

The GeantV Software

Designed to take the opportunities offered by modern computing architectures

Paradigm shift – Particle-level parallelization (not event-level like in G4). One thread may process particles from different events !

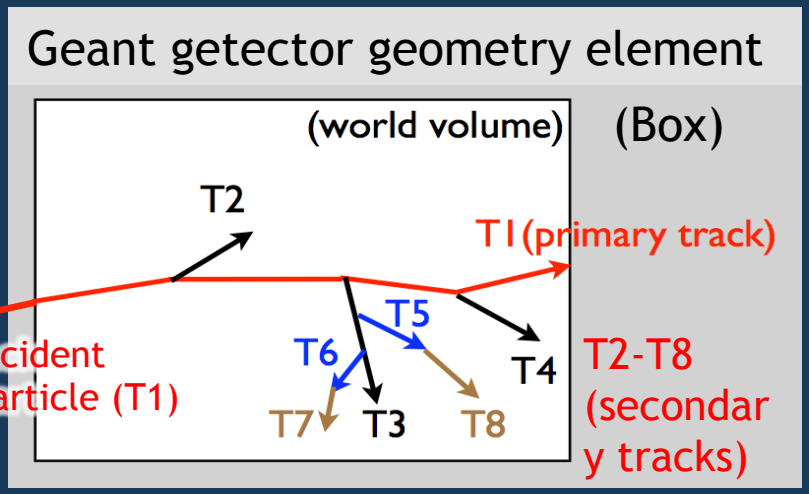
Use parallelization techniques

- *Instruction pipelining*
- Explicit *vectorization*, *Single Instruction Multiple Data (SIMD)*
- *Multiple Instruction Multimple Data (MIMD)*
- *Data locality*

GeantV Basketized Transport

Particle (track) level parallelization, including primary and secondary tracks

Organize *tracks in baskets* sorted by particle type, volume traversed, or kinematic properties (same or different event !)



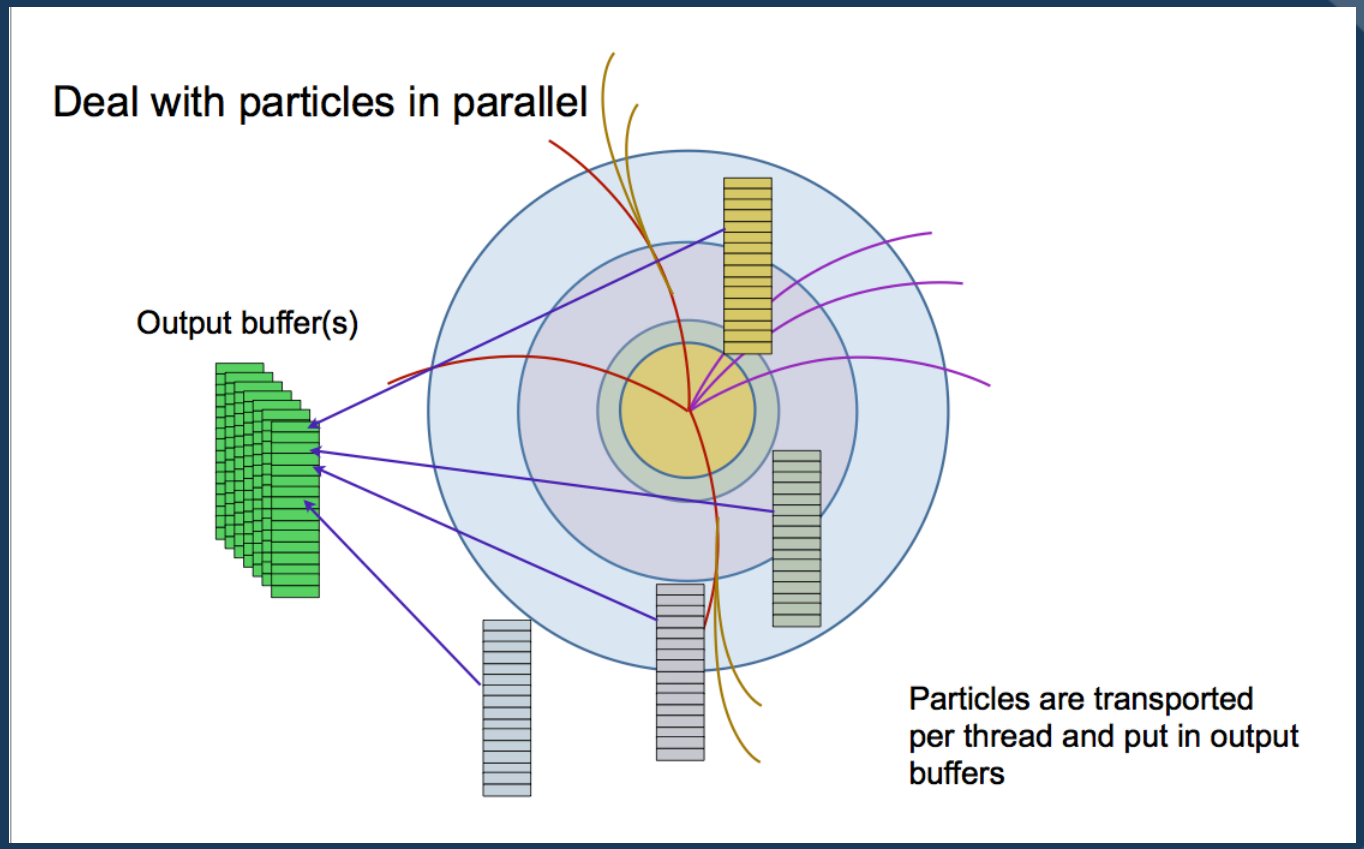
GeantV: entire track baskets processed *in parallel*, one per thread

Geant4: entire event processed *in parallel*, one per thread

A *Scheduler* sorts the *baskets* to be dispatched to a hybrid system (CPU, GPGPU, etc.)

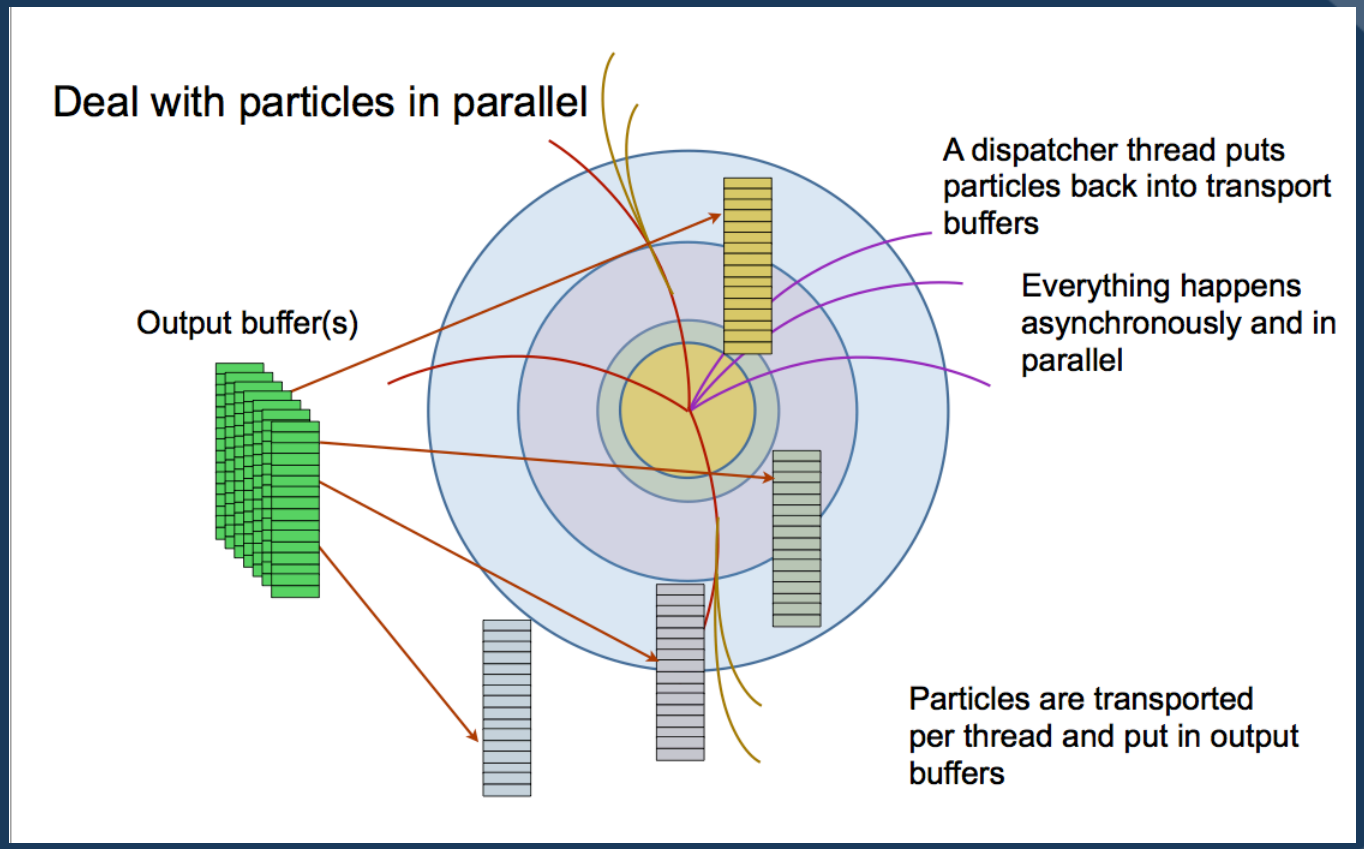
GeantV Basketized Transport

Particles grouped in baskets by common characteristics: volume traversed, physics process, particle type



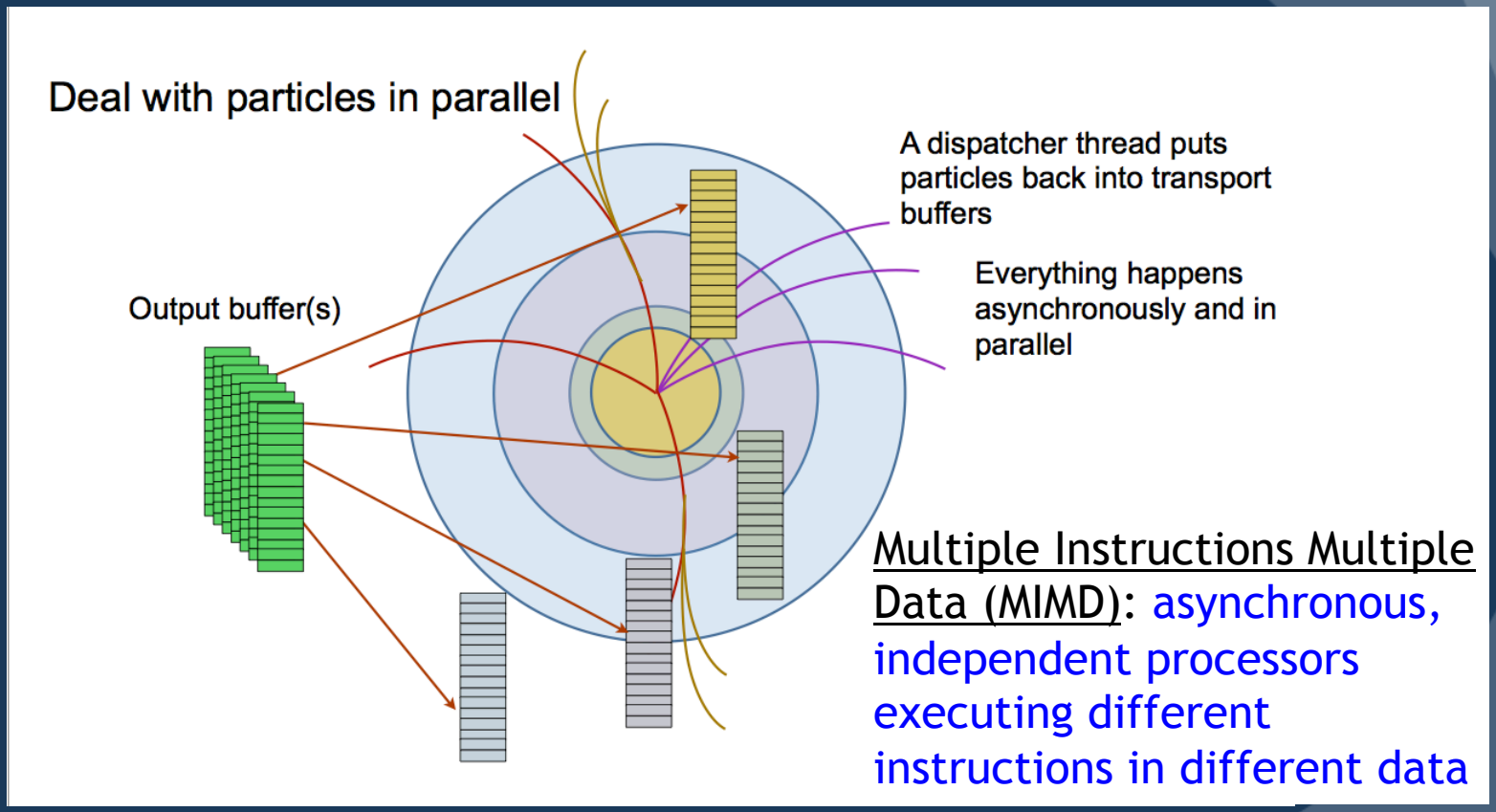
GeantV Basketized Transport

Particles grouped in baskets by common characteristics: volume traversed, physics process, particle type



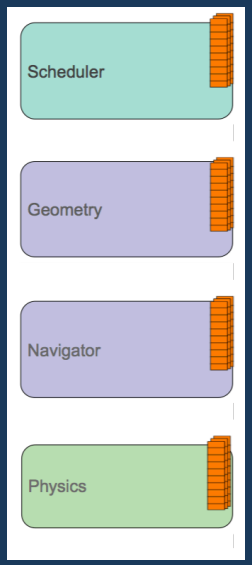
GeantV Basketized Transport

Particles grouped in baskets by common characteristics: volume traversed, physics process, particle type



Multiple Instructions Multiple Data (MIMD): asynchronous, independent processors executing different instructions in different data

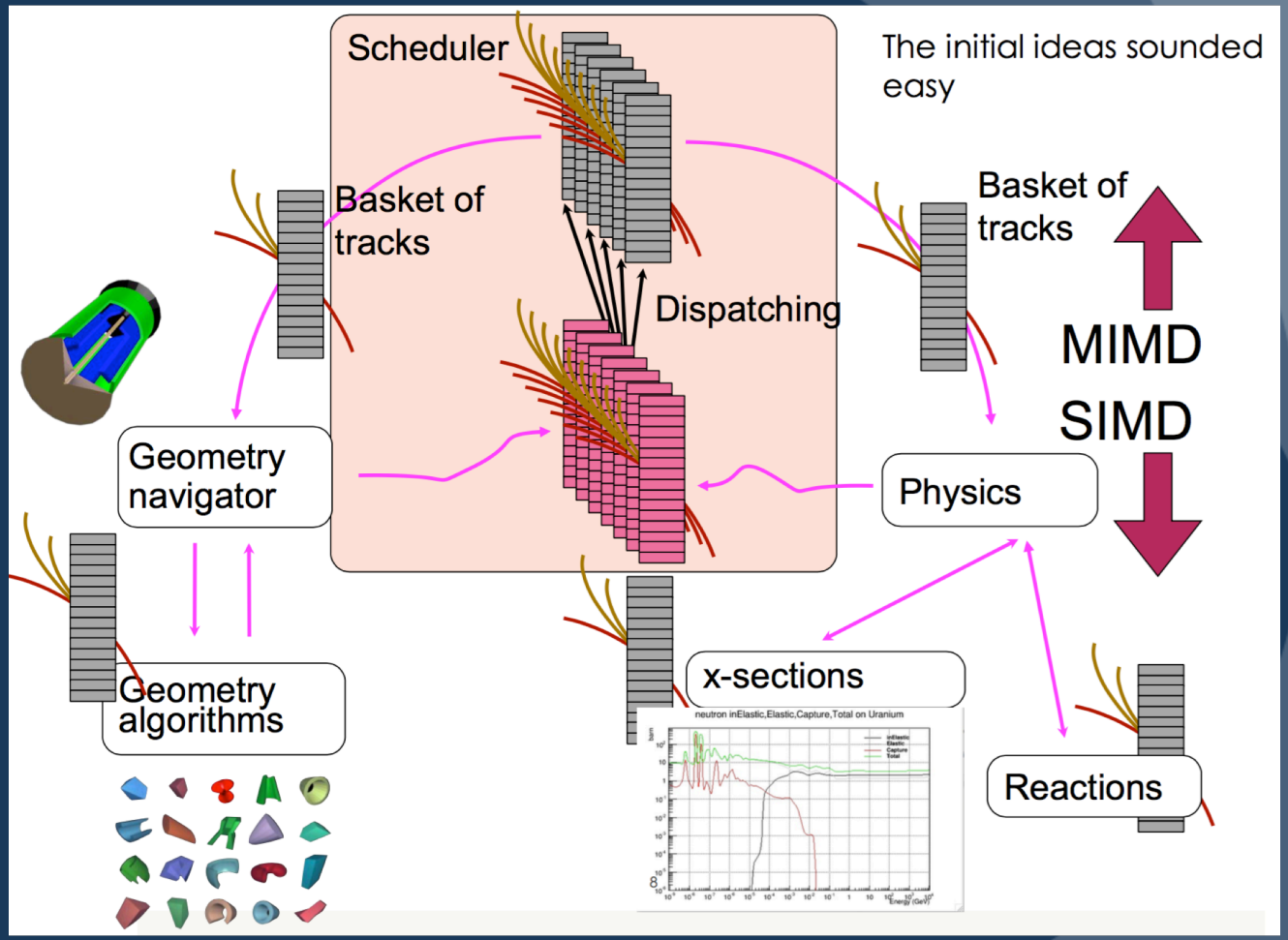
GeantV Software Components



All elements of a simulation toolkit

↓

Complete system



Instruction Pipelining

Available in most processors

Instruction processing not sequential but *concurrent* and *in parallel*

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

IF: instruction fetching
ID: instruction decoding
EX: instruction execution
MEM: memory access
WB: register write back

(programmer must avoid conditional branching!)

Newer hardware architectures utilize instruction pipelining in an increasingly more efficient way

Vectorization and SIMD

Single Instruction Multiple Data (SIMD) vectorization

Code *vectorization* consists of organizing data in vectors to be stored simultaneously in processor *registers*

Example: a single (same) instruction may be applied in one clock cycle to track data from different events

Apply same single instruction to these multiple data (SIMD)

Compilers have options for auto-vectorization, i.e.

```
g++ -mavx   or   -msse4.2
```

but …

Vectorization and SIMD

Single Instruction Multiple Data (SIMD) vectorization

Compilers can organize data in vectors:

```
// Example 12.1a. Automatic vectorization
const int size = 1024;
int a[size], b[size];
// ...
for (int i = 0; i < size; i++) {
   a[i] = b[i] + 2;
}
```

but cannot deal with branches:

```
// Example 12.4a. Loop with branch

// Loop with branch
void SelectAddMul(short int aa[], short int bb[], short int cc[]) {

   for (int i = 0; i < 256; i++) {
      aa[i] = (bb[i] > 0) ? (cc[i] + 2) : (bb[i] * cc[i]);
   }
}
```

Vectorization by hand:

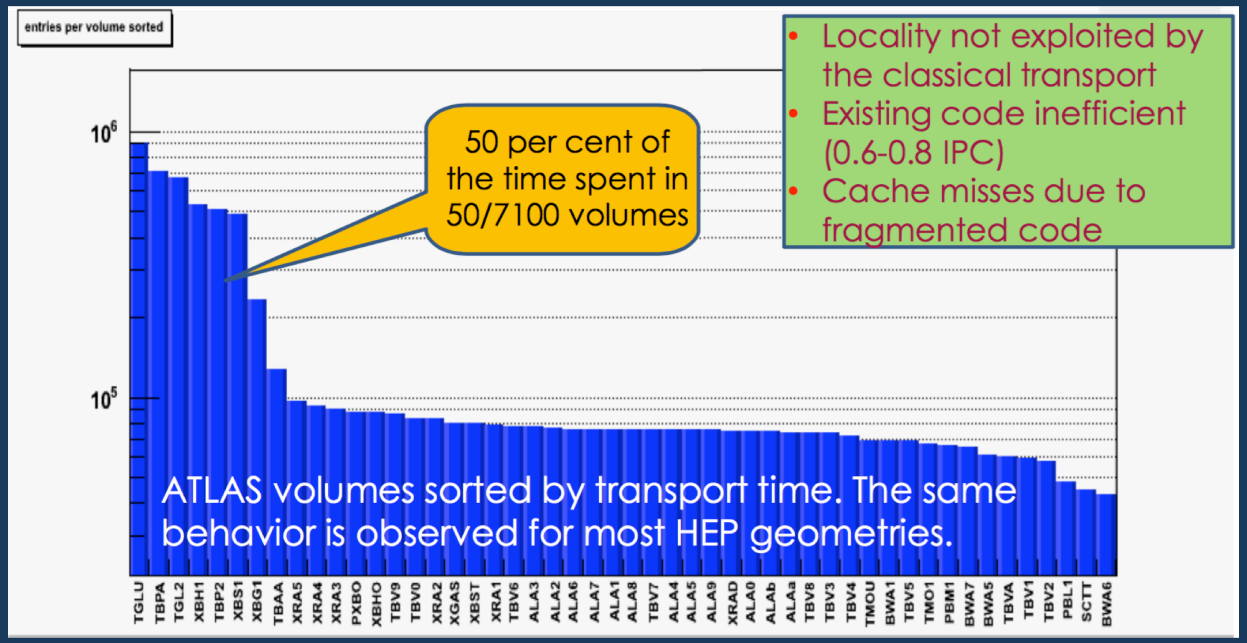
```
// Branch/loop function vectorized:
void SelectAddMul(short int aa[], short int bb[], short int cc[]) {

   // Make a vector of (0,0,0,0,0,0,0,0)
   __m128i zero = _mm_set1_epi16(0);
   // Make a vector of (2,2,2,2,2,2,2,2)
   __m128i two  = _mm_set1_epi16(2);

   // Roll out loop by eight to fit the eight-element vectors:
   for (int i = 0; i < 256; i += 8) {
      // Load eight consecutive elements from bb into vector b:
      __m128i b = LoadVector(bb + i);
      // Load eight consecutive elements from cc into vector c:
      __m128i c = LoadVector(cc + i);
      // Add 2 to each element in vector c
      __m128i c2 = _mm_add_epi16(c, two);
      // Multiply b and c
      __m128i bc = _mm_mullo_epi16 (b, c);
      // Compare each element in b to 0 and generate a bit-mask:
      __m128i mask = _mm_cmpgt_epi16(b, zero);
      // AND each element in vector c2 with the bit-mask:
      c2 = _mm_and_si128(c2, mask);
      // AND each element in vector bc with the inverted bit-mask:
      bc = _mm_andnot_si128(mask, bc);
      // OR the results of the two AND operations:
      __m128i a = _mm_or_si128(c2, bc);
      // Store the result vector in eight consecutive elements in aa:
      StoreVector(aa + i, a);
   }
}
```

Vectorization applied to geometry, navigation, physics, …

Data locality and SIMD

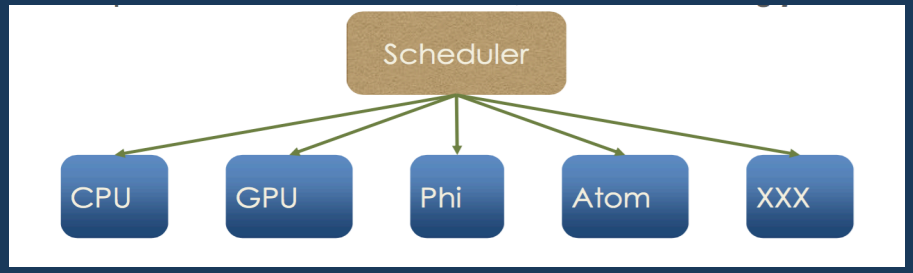
Baskets of particles of same type, same volume traversed, or similar kinematic properties ➔

Same set of geometry or physics instructions



Basket data and instructions local to processor executing thread

GeantV Portability



Hybrid system, different architectures for different tasks

One single version of each algorithm, specialized to platform via template programing and low level optimized libraries

distance(double &);

distance(vector_type &);

Scalar interface

Vector interface

Dual interfaces

```
template<typename Real_v>
static void
common_distance_function( Vector3D<Real_v>
input,                    Real_v &output )
{
   // Single kernel algorithm using Backend types
}
```

Kernels

Backends

```
Scalar.h
template <typename T =
Real_s>
class ScalarT {
public:
  using Real_v   = T;
  using Double_v = Double_s;
  …
}
// Functions operating with
backend types
```

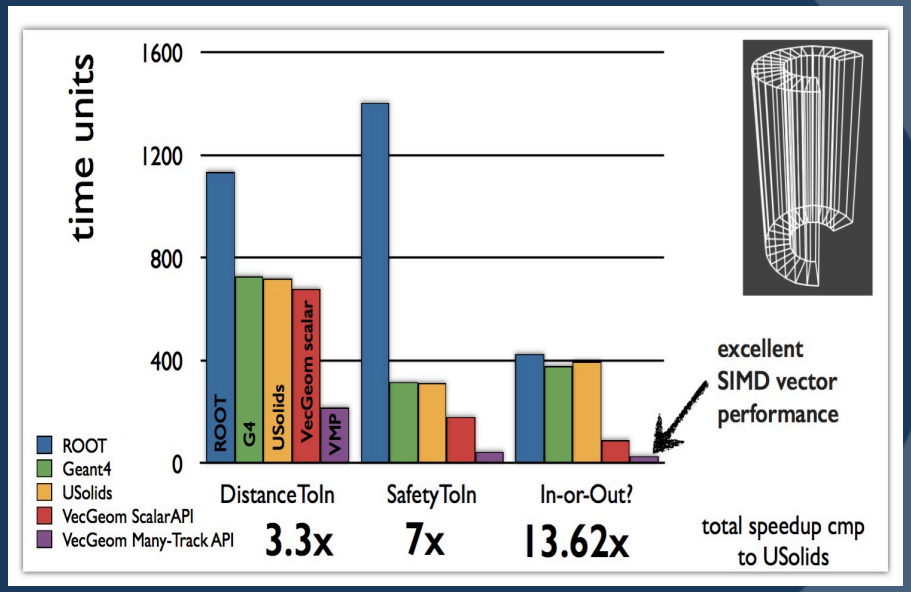
```
UMESimd.h
{
using Real_v   =
UME::SIMD::SIMDVec<Real_
s, SIMDWidth<Real_s>()>;
 using Double_v =
UME::SIMD::SIMDVec<Doubl
e_s,
SIMDWidth<Double_s>()>;
  …
};
// Functions operating with
backend types
```

```
VcSimdArray.h
{
using Real_v   =
Vc::SimdArray<Real_s,
N>;
using Double_v =
Vc::SimdArray<Double_s,
N>;
  …
};
// Functions operating with
backend types
```

The VecGeom Geometry Package

VecGeom is GeantV vectorized geometry package, optimized for SIMD architectures, but also compatible with Geant4

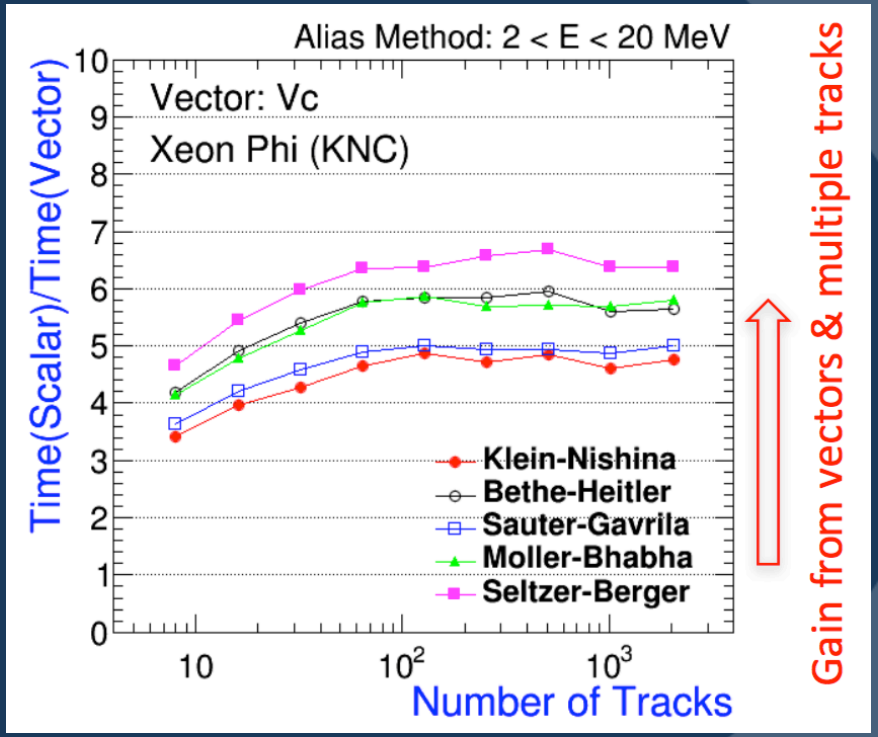
- Geometry takes 30-40% of the time in a typical G4 HEP simulation
- Speedup of main geometry functions in the range of a 3-13 factor
- Substantial gains also in scalar mode, will benefit Geant4



The Vectorized EM Package

Improved multiple scattering algorithm, faster (vectorized) code, optimal for GeantV and beneficial to Geant4

- Physics takes 40% of the time in a typical Geant4 HEP simulation (30% goes to EM processes)
- Vector code performs 3.3-6x faster than scalar code (Xeon Phi - KNC)

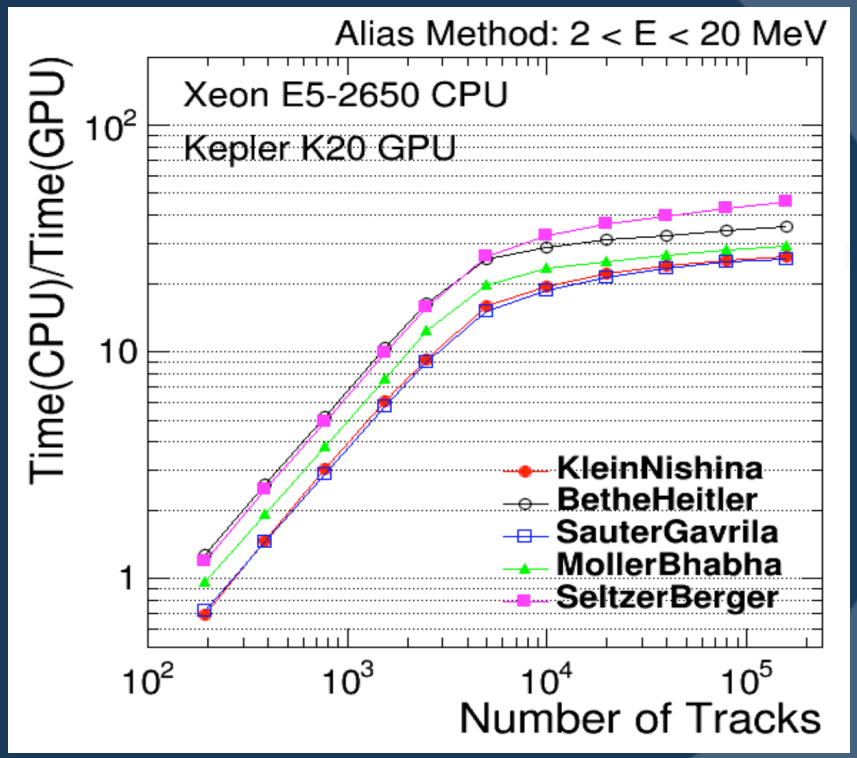


The Vectorized EM Package

Improved multiple scattering algorithm, faster (vectorized) code, optimal for GeantV and beneficial to Geant4

- Physics takes 40% of the time in a typical Geant4 HEP simulation (30% goes to EM processes)
- Code performs 30x faster in GPU than in CPU (NVIDIA-Kepler vs. Intel Xeon)

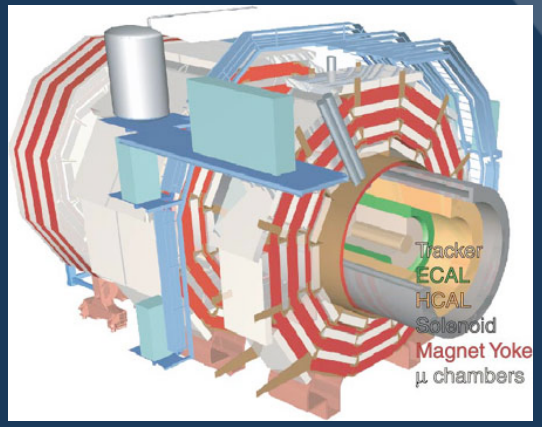
  (Need 10^4 tracks)



CMS Simulation Tests with GeantV

Realistic simulation of complex detector for Geant4-GeantV comparison (with some approximations)

- Proton-proton collisions at 14 TeV, min-bias events (Pythia)
- 2015 CMS detector (Run II)
- 4T uniform magnetic field (decent approximation of a solenoidal field)
- Low energy cut at 1 MeV (secondary particles)
- Tabulated Physics. Library of sampled G4 interactions and tabulated x-sections



Same test (described above) is run with both Geant4 and GeantV with various versions of the Geometry library

CMS Simulation Tests with GeantV

Scheduler	Geometry	Physics	Magnetic Field Stepper
Geant4 only	Legacy G4 geometry	Various Physics Lists	Various RK implementations
Geant4 or GeantV	VecGeom 2016 (scalar mode)	• Tabulated Physics • Scalar Physics Code	• Helix (Fixed Field) • Cash-Karp Runge-Kutta
GeantV only	• VecGeom 2015 • VecGeom 2016 vector • Legacy Tgeo (ROOT geometry)	Vector Physics Code	Vectorized RK Implementation

CMS Simulation Tests with GeantV

Geant4/GeantV execution time ratios as of February 2016 – much progress since then

- GeantV (TGeo): G4/GV = 1.5
  - Algorithm improvements in infrastructure software
- GeantV (VecGeom 2015): G4/GV = 2.4
  - Algorithm improvements in geometry package (no vectorization)
- GeantV (VecGeom 2016): G4/GV = 3.3
  - Further algorithm improvements and some vectorization

New test results will be available by the end of October 2016 (next week)

Timeline, Community Testing

- Design, development, improvement needs to be done in close collaboration with the user community (i.e. HEP experiments)
  - "HEP software community meeting on GeantV R&D", CERN, Oct. 25th-27th 2016
  - LHC experiments expressed interest to test the GeantV engine and its components, such as VecGeom, as they become available
- GeantV first production quality version expected in 2018
  - In time for consideration by HL-LHC experiments and DUNE, to start running in ~2025
- Same as Geant4, GeantV user communities potentially extend to
  - Particle and nuclear physics experiments
  - Medical research and treatment
  - Space exploration

Outlook

A GeantV prototype with all three main components is operational – Scheduler (basketization), Geometry, Physics

The code is portable to various modern computing architectures – CPUs, GPGPU, Xeon Phi

The GeantV development team has started to work closely with the HEP user community

A first production quality version is expected by 2018

The GeantV team is well on track to deliver a next generation simulation package, at least 2-5 time faster than Geant4, portable to modern computing architectures, by 2018