GeantV – From CPU to accelerators

Philippe Canal (FNAL) for the GeantV development team

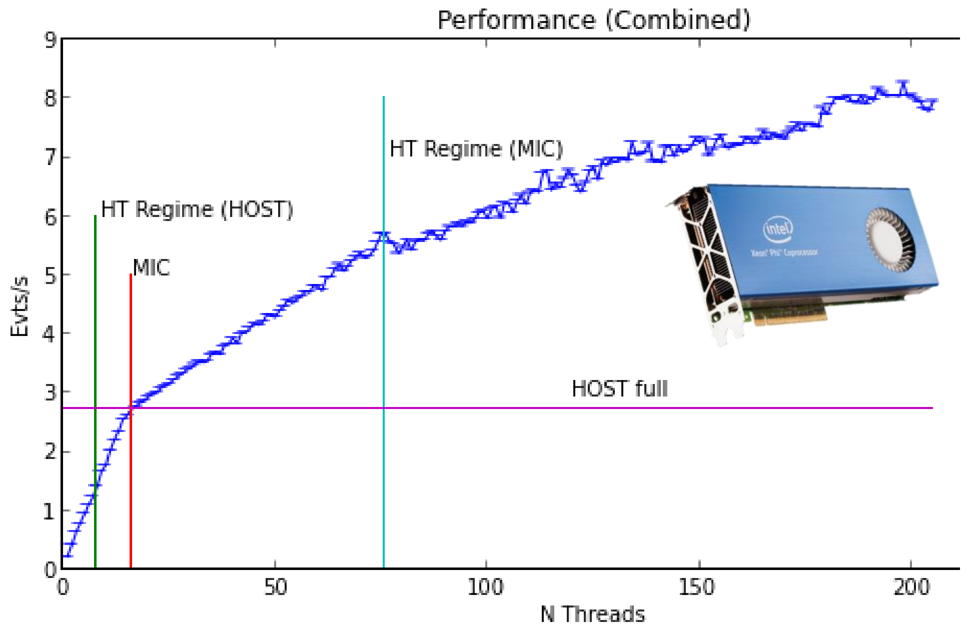
G.Amadio (UNESP), Ananya (CERN), J.Apostolakis (CERN) , A.Arora (CERN), M.Bandieramonte (CERN), A.Bhattacharyya (BARC), C.Bianchini (UNESP), R.Brun (CERN), Ph.Canal (FNAL), F.Carminati (CERN), L.Duhem (intel), D.Elvira (FNAL), A.Gheata (CERN), M.Gheata (CERN), I.Goulas (CERN), F.Hariri (CERN), R.Iope (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)

Outline

- GeantVectorized – an introduction
  - Challenges, ideas, goals
- Main components and performance
  - Design and infrastructure
  - Vectorization: overheads vs. gains
  - Geometry library
  - Physics processes
- Performance benchmarks
- Results, milestones, plans

Geant4 Multi-threading

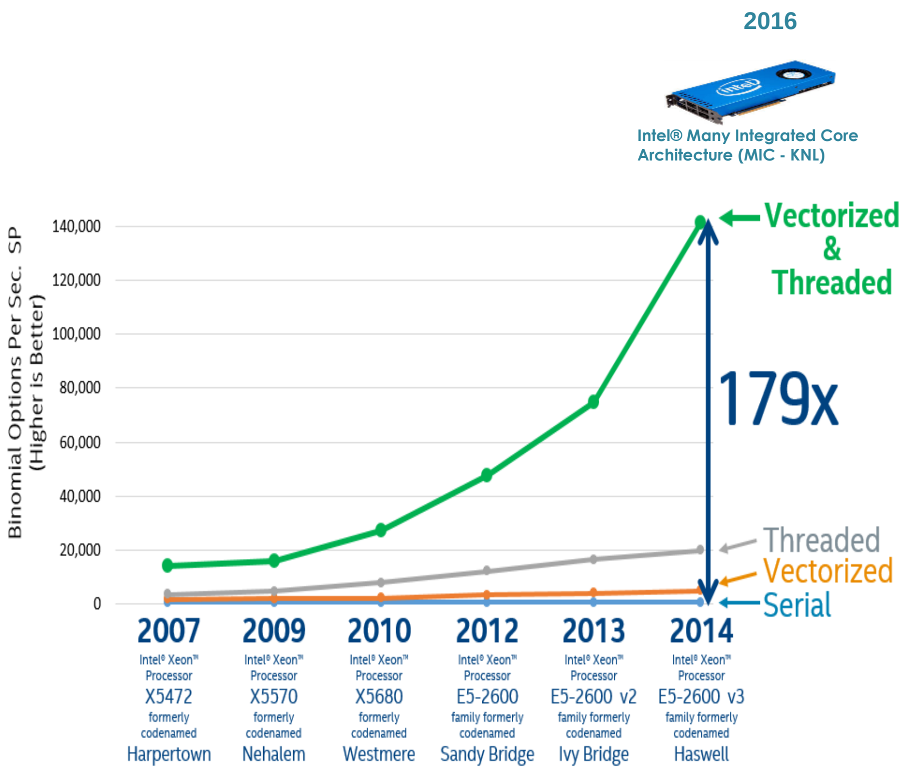
- Event level Parallelism
  - Each thread processes one full event exclusively
  - Part of Geant4 since release 10.0, Dec. 2013

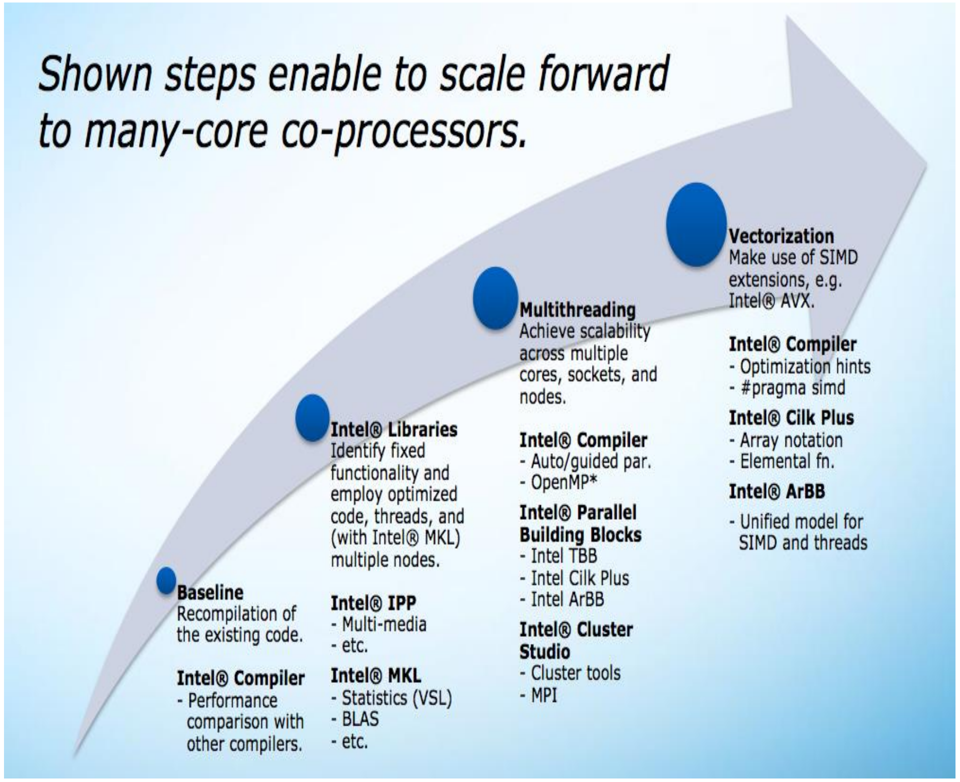


Preliminary, Courtesy of A.Dotti, SLAC

- Demonstrates
  - Linear scaling of throughput with number of threads
- Large savings in memory: 9MB extra memory per thread
- No Performance/Throughput increase

Hardware constraints and promised paths





Shown steps enable to scale forward to many-core co-processors.

Baseline
Recompilation of the existing code.

Intel® Compiler
- Performance comparison with other compilers.

Intel® Libraries
Identify fixed functionality and employ optimized code, threads, and (with Intel® MKL) multiple nodes.

Intel® IPP
- Multi-media
- etc.

Intel® MKL
- Statistics (VSL)
- BLAS
- etc.

Multithreading
Achieve scalability across multiple cores, sockets, and nodes.

Intel® Compiler
- Auto/guided par.
- OpenMP*

Intel® Parallel Building Blocks
- Intel TBB
- Intel Cilk Plus
- Intel ArBB

Intel® Cluster Studio
- Cluster tools
- MPI

Vectorization
Make use of SIMD extensions, e.g. Intel® AVX.

Intel® Compiler
- Optimization hints
- #pragma simd

Intel® Cilk Plus
- Array notation
- Elemental fn.

Intel® ArBB
- Unified model for SIMD and threads

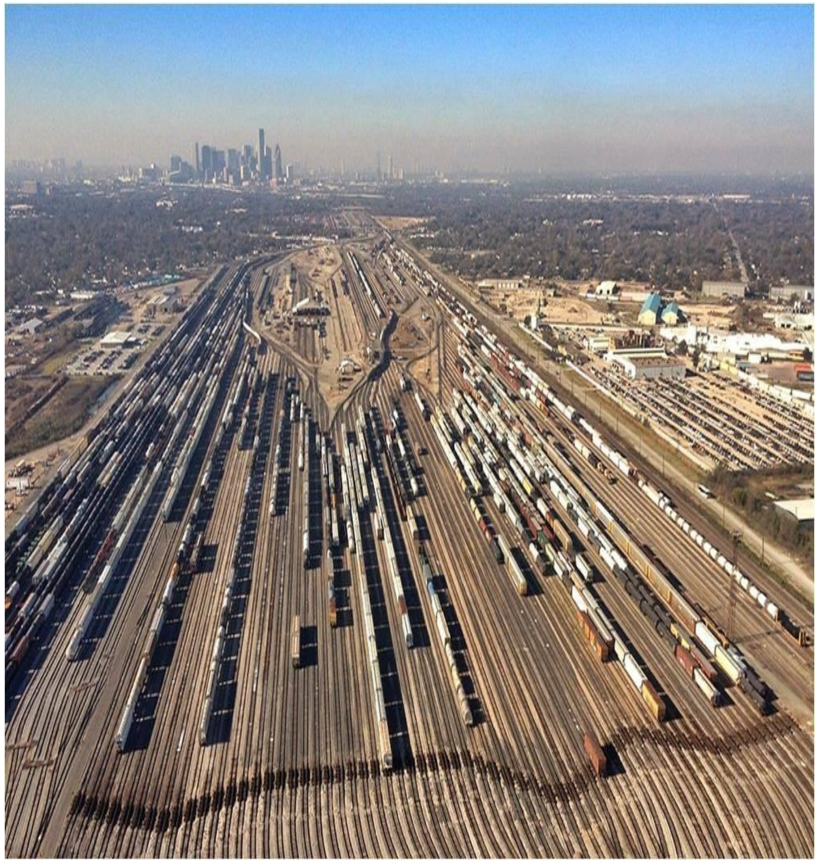
Reality is that refactoring effort towards multi-level parallelism goes way beyond the usage of specific software tools and the effort and end result depend significantly on the workload and design.

What do we want to do?

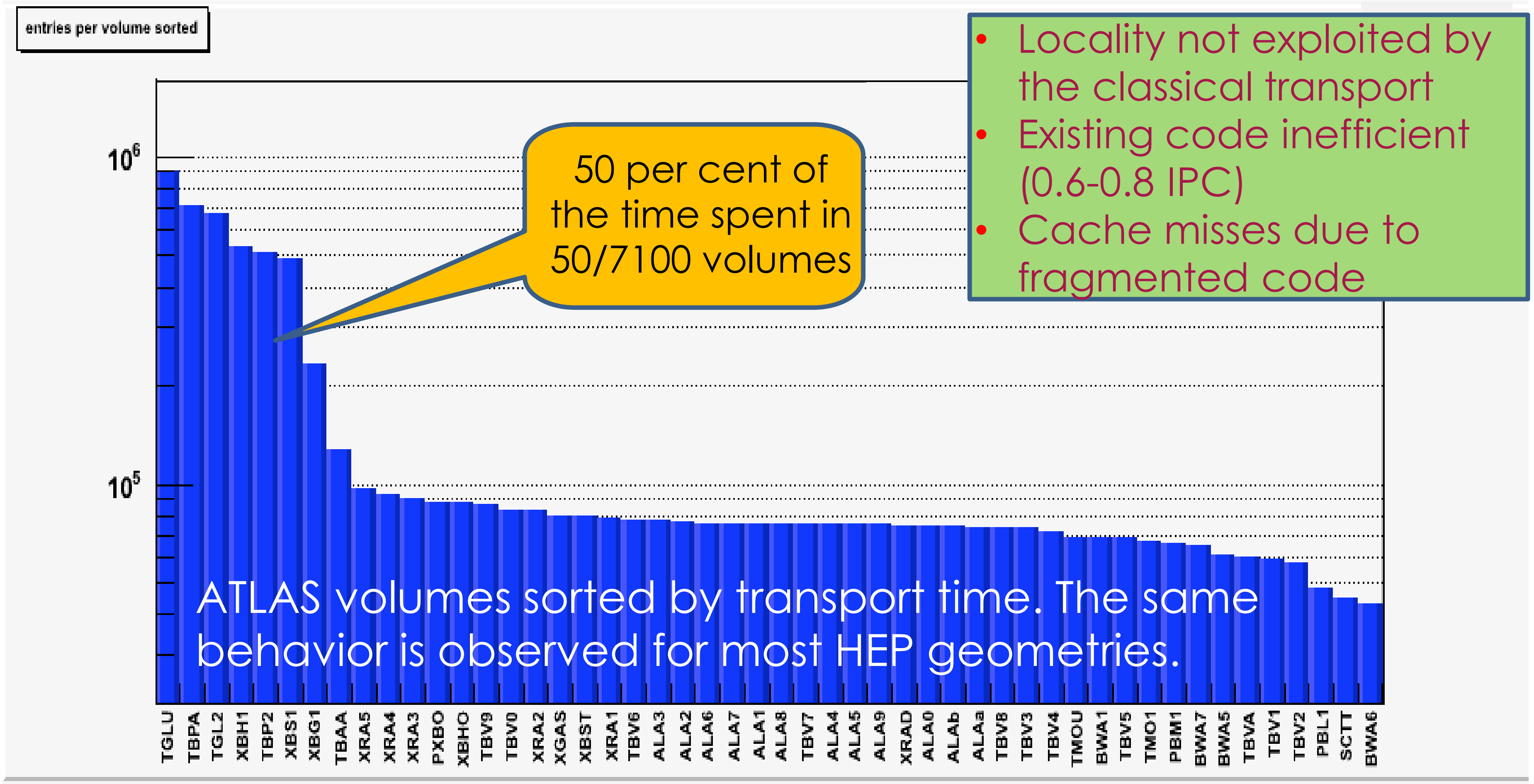
- Develop an all-particle transport simulation software with
  - Geant4 or new improved (where possible) physics models
  - A performance between 2 and 5 times greater than Geant4
  - Full simulation and various options for fast simulation
  - Portable on different architectures, including accelerators (GPUs and Xeon Phi's)
- Understand the limiting factors for a one-order-of-magnitude (10x) improvement

The ideas

- Transport particles in groups (vectors) rather than one by one
  - Group particles by geometry volume or same physics
  - No free lunch: data gathering overheads needs to stay less than vector gains
- Dispatch SoA to functions with vector signatures
  - Use backends to abstract interface: vector, scalar
  - Use backends to insulate technology/library: Vc, Cilk+, VecMic, …
- Redesign the library and workflow to target fine grain parallelism
  - CPU, GPU, Phi, Atom, …
  - Aim for a 3x-5x faster code, understand hard limits for more



HEP transport is mostly local !

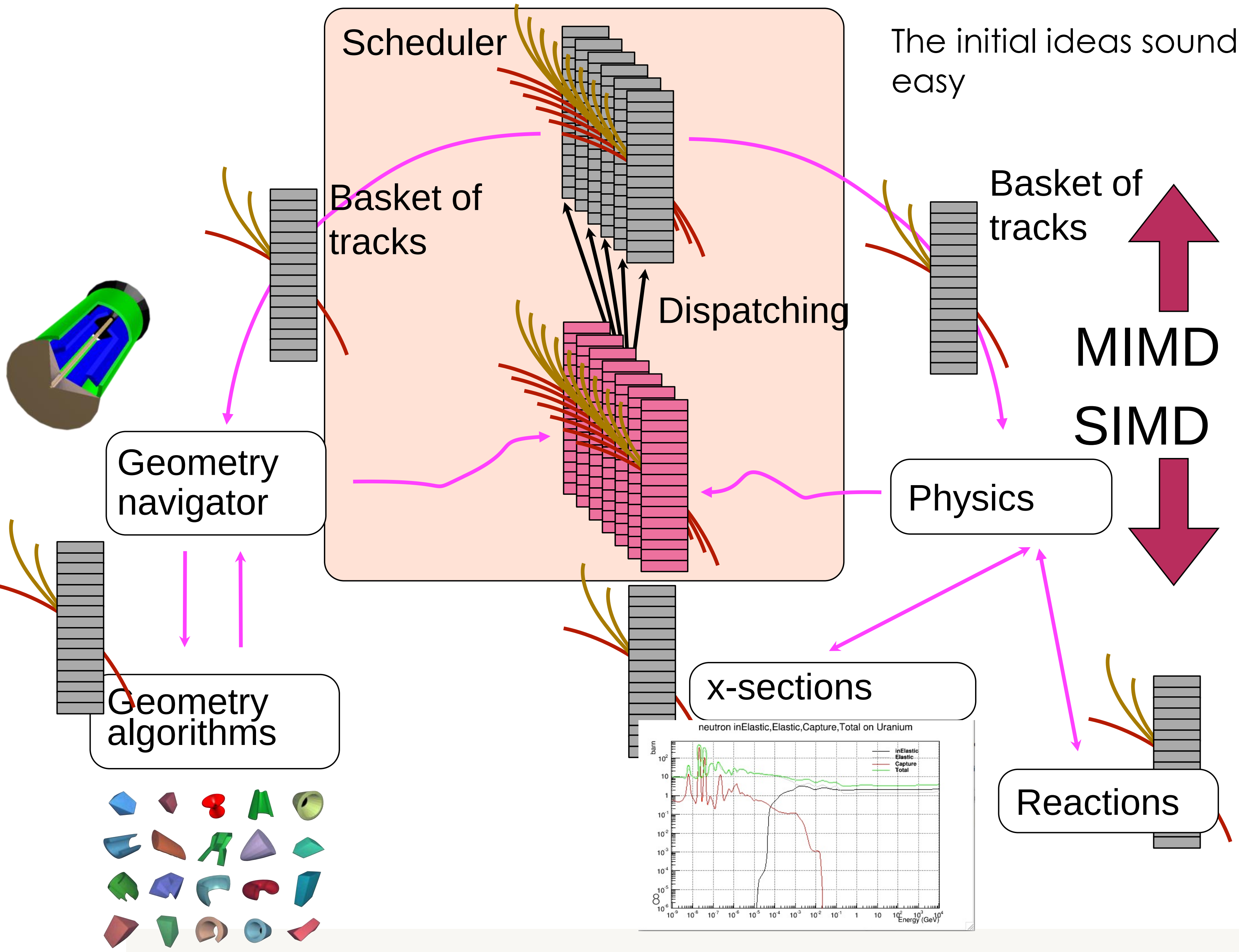


50 per cent of the time spent in 50/7100 volumes

- Locality not exploited by the classical transport
- Existing code inefficient (0.6-0.8 IPC)
- Cache misses due to fragmented code

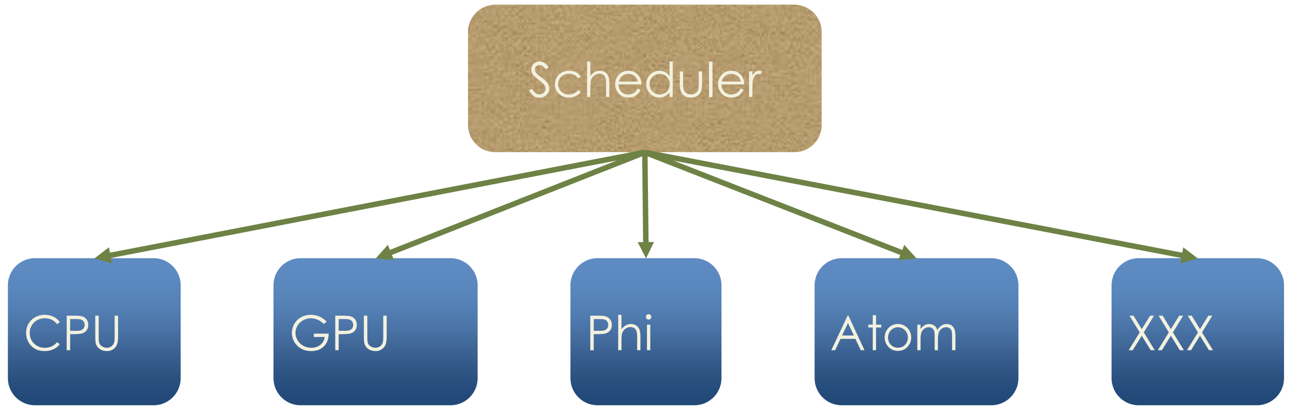
ATLAS volumes sorted by transport time. The same behavior is observed for most HEP geometries.

The initial ideas sounded easy



Challenges

- Overhead from reshuffling particle lists should not offset SIMD gains
- Exploit the hardware at its ***best***, while maintaining ***portability***



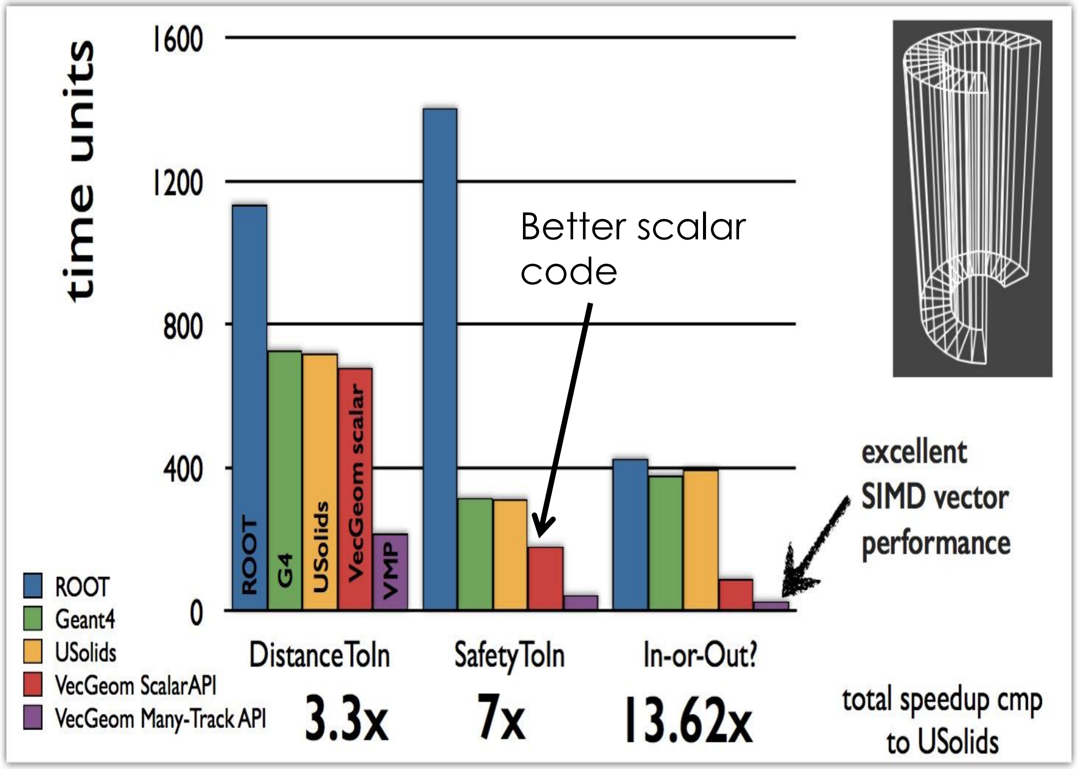
- Test from the onset on a “large” setup (LHC-like detector)
  - Toy models tell us very little – complexity is the problem

Status on GPU

- Broker adapts baskets to the coprocessor
  - Selects tracks that are efficiently processed on coprocessor
  - Gather in chunk large enough (e.g. 4096 tracks on NVidia K20)
  - Transfer data to and from coprocessor
  - Execute kernels
- On NVidia GPU, we are effectively using implicit vectorization
  - Rather than one thread per basket, on GPUs we use 4096 threads each processing one of the tracks in the basket
- Cost of data transfer is mitigated by overlapping kernel execution and data transfer
  - We can send fractions of the full GPU's work asynchronously using streams

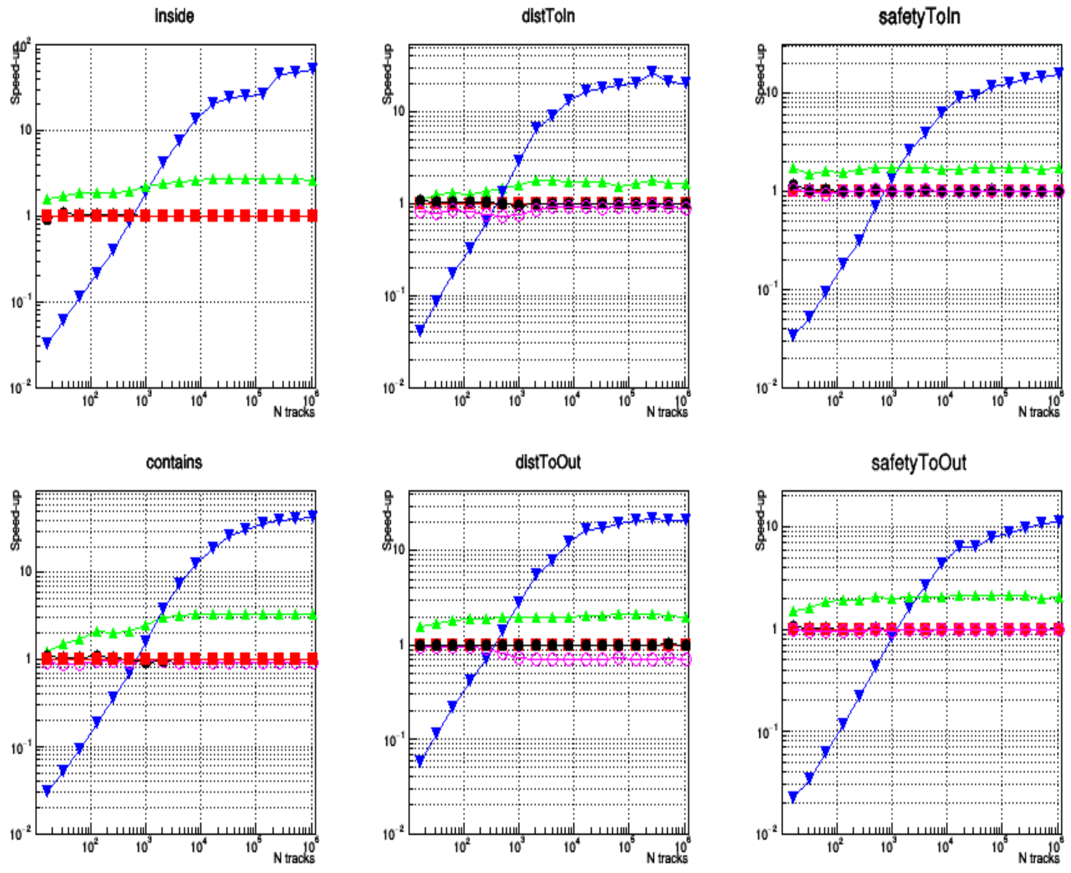
Geometry - VecGeom

- Geometry takes 30-40% CPU time of typical Geant4 HEP Simulation
- A library of vectorised geometry algorithms to take maximum advantage of SIMD architectures
- Substantial performance gains also in scalar mode



Geometry performance on K20

- Speedup for different navigation methods of the box shape, normalized to scalar CPU
  - Scalar (specialized/unspecialized)
  - Vector
  - GPU (Kepler K20)
  - ROOT
- Data transfer in/out is asynchronous
  - Measured only the kernel performance, but providing constant throughput can hide transfer latency
- The die can be saturated with both large track containers, running a single kernel, or with smaller containers dynamically scheduled.
- Just a baseline proving we can



Evolution

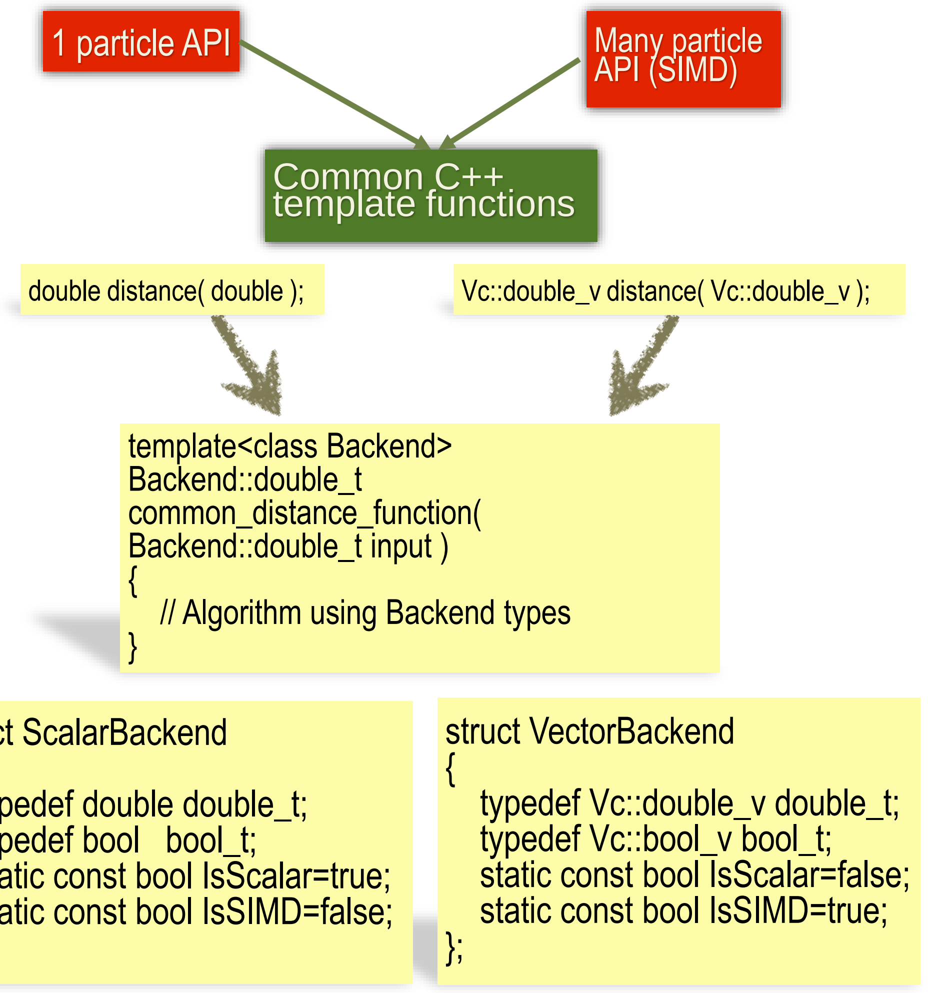
One becomes two, two becomes three, and out of the third comes the one as the fourth. Maria Prophetissa (3rd century AD)

- VecGeom code has been developed for GeantV vectorised transport
- USolids was developed to unify TGeo and Geant4 geometry packages
- Now VecGeom algorithms are retrofitted to USolids and are available both to Geant4 and to TGeo
- VecGeom has the potential to introduce a few percent gain for Geant4 (to be verified)
  - Algorithm improvement and (internal) vectorisation of some shapes
- VecGeom is the consolidation both on the algorithm level and on the developer level of G4-Geo, TGeo, USolid and Vectorization efforts.

Portability

"Backend" is a (trait) struct encapsulating standard types/properties for "scalar, vector, CUDA" programming; makes information injection into template function easy

- Long-term maintainability of the code
  - write one single version of each algorithm and to specialise it to the platform via template programming and low level optimised libraries (Vc in our case)
- A Xeon Phi specific backend is being developed in collaboration with CERN's openlab (UME::SIMD)
- Results are quite encouraging: maybe portable HPC is NOT an oxymoron after all…



```
double distance( double );
```

```
Vc::double_v distance( Vc::double_v );
```

```
template<class Backend>
Backend::double_t
common_distance_function(
Backend::double_t input )
{
   // Algorithm using Backend types
}
```

```
struct ScalarBackend
{
   typedef double double_t;
   typedef bool   bool_t;
   static const bool IsScalar=true;
   static const bool IsSIMD=false;
};
```

```
struct VectorBackend
{
   typedef Vc::double_v double_t;
   typedef Vc::bool_v bool_t;
   static const bool IsScalar=false;
   static const bool IsSIMD=true;
};
```

http://code.compeng.uni-frankfurt.de/projects/vc

Avoiding code duplication

- Support of multiple platforms usually means multiple versions of source code
- What are the differences between the two versions of code shown on the right?
- Primarily: types and their operators, function attributes (__device__), also some higher level functions, e.g. conditional assignment
- Avoid code duplication by abstracting away differences into common types or overloaded functions defined

cuda

```
template <int N>
__device__
double Planes<N>::DistanceToOut(
    double const (&plane)[4][N],
  Vector3D<double> const &point,
  Vector3D<double> const &direction) {

  double bestDistance = kInfinity;
  for (int i = 0; i < N; ++i) {
    double distance;
    distance = -(plane[0][i]*point[0] + plane[1][i]*point[1] +
                 plane[2][i]*point[2] + plane[3][i]);
    distance /= (plane[0][i]*direction[0] + plane[1][i]*direction[1] +
                 plane[2][i]*direction[2]);
    bestDistance = (distance < bestDistance) ? distance : bestDistance;
  }
  return bestDistance;
}
```

Vc

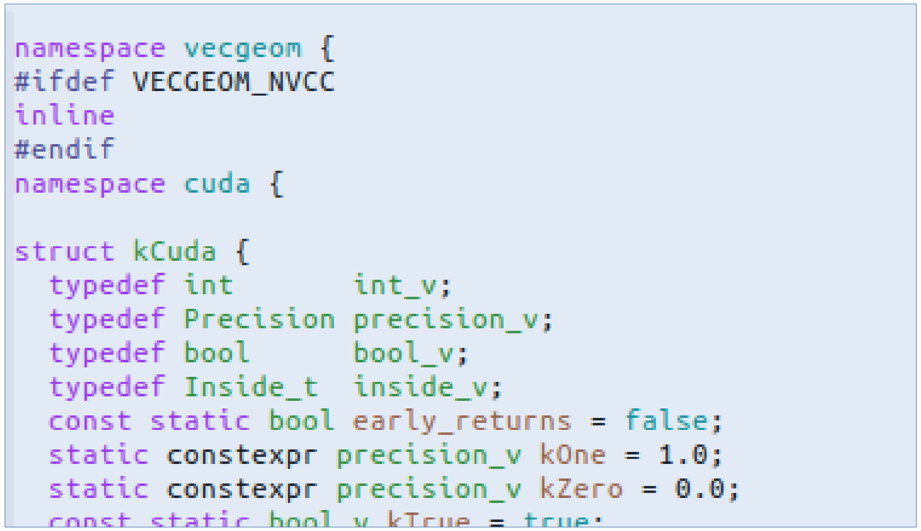
```
template <int N>
Vc::double_v Planes<N>::DistanceToOut(
    double const (&plane)[4][N],
  Vector3D<Vc::double_v> const &point,
  Vector3D<Vc::double_v> const &direction) {

  Vc::double_v bestDistance = kInfinity;
  for (int i = 0; i < N; ++i) {
    Vc::double_v distance;
    distance = -(plane[0][i]*point[0] + plane[1][i]*point[1] +
                 plane[2][i]*point[2] + plane[3][i]);
    distance /= (plane[0][i]*direction[0] + plane[1][i]*direction[1] +
                 plane[2][i]*direction[2]);
    bestDistance(distance < bestDistance) = distance;
  }
  return bestDistance;
}
```

Using traits to avoid code duplication

- Intensive kernels are developed in a generic way, using only trait-defined types and functions.
- Architecture-specific traits are created as needed, to associate generic types and functions with their arch-specific types.
- Appropriate backends are requested by #define

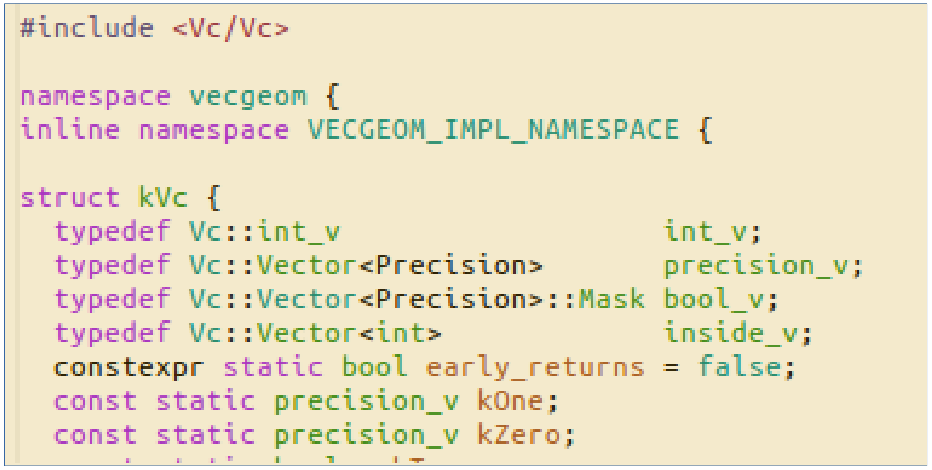
backend/cuda/Backend.h



```
namespace vecgeom {
#ifdef VECGEOM_NVCC
inline
#endif
namespace cuda {

struct kCuda {
  typedef int       int_v;
  typedef Precision precision_v;
  typedef bool      bool_v;
  typedef Inside_t  inside_v;
  const static bool early_returns = false;
  static constexpr precision_v kOne = 1.0;
  static constexpr precision_v kZero = 0.0;
  const static bool_v kTrue = true;
```

backend/vc/Backend.h



```
#include <Vc/Vc>

namespace vecgeom {
inline namespace VECGEOM_IMPL_NAMESPACE {

struct kVc {
  typedef Vc::int_v                          int_v;
  typedef Vc::Vector<Precision>              precision_v;
  typedef Vc::Vector<Precision>::Mask bool_v;
  typedef Vc::Vector<int>                    inside_v;
  constexpr static bool early_returns = false;
  const static precision_v kOne;
  const static precision_v kZero;
```

A generic kernel

```
template <int N>
template <class Backend>
VECGEOM_CUDA_HEADER_BOTH
typename Backend::Float_t Planes<N>::DistanceToOutKernel(
    double const (&plane)[4][N],
    Vector3D<typename Backend::Float_t> const &point,
    Vector3D<typename Backend::Float_t> const &direction) {

  typedef typename Backend::Float_t Float_t;
  typedef typename Backend::bool_v Bool_t;

  Float_t bestDistance = kInfinity;
  Float_t distance[N];
  Bool_t valid[N];
  for (int i = 0; i < N; ++i) {
    distance[i] = -(plane[0][i]*point[0] + plane[1][i]*point[1] +
                    plane[2][i]*point[2] + plane[3][i]);
    distance[i] /= (plane[0][i]*direction[0] + plane[1][i]*direction[1] +
                    plane[2][i]*direction[2]);
    valid[i] = distance[i] >= 0;
  }
  for (int i = 0; i < N; ++i) {
    MaskedAssign(valid[i] && distance[i] < bestDistance, distance[i],
                 &bestDistance);
  }
  return bestDistance;
}
```

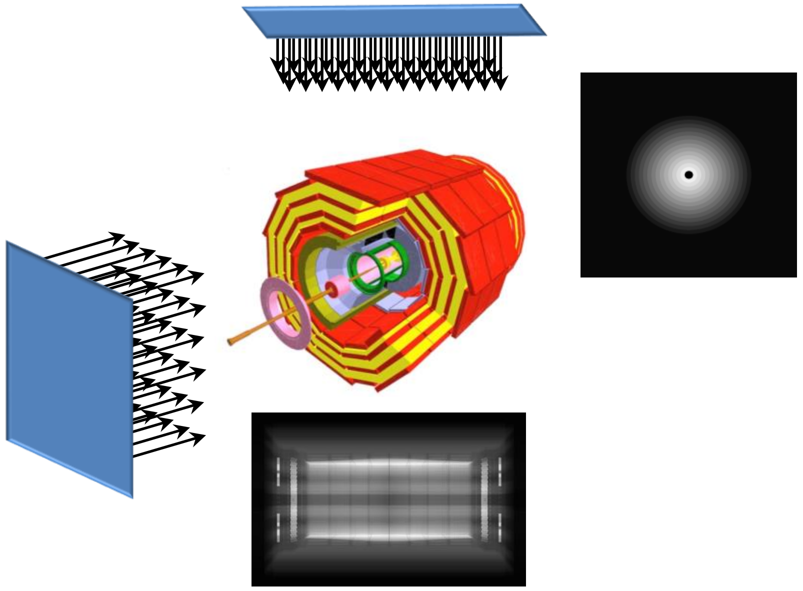
The Backend; can also be the type instead

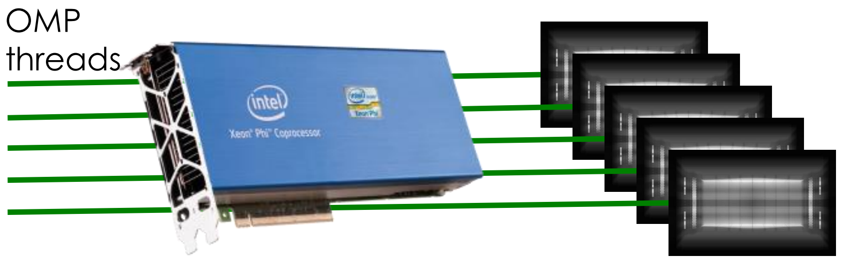
Arithmetics just works!

MaskedAssign( ) is an optimized if( ) replacement

The X-Ray benchmark

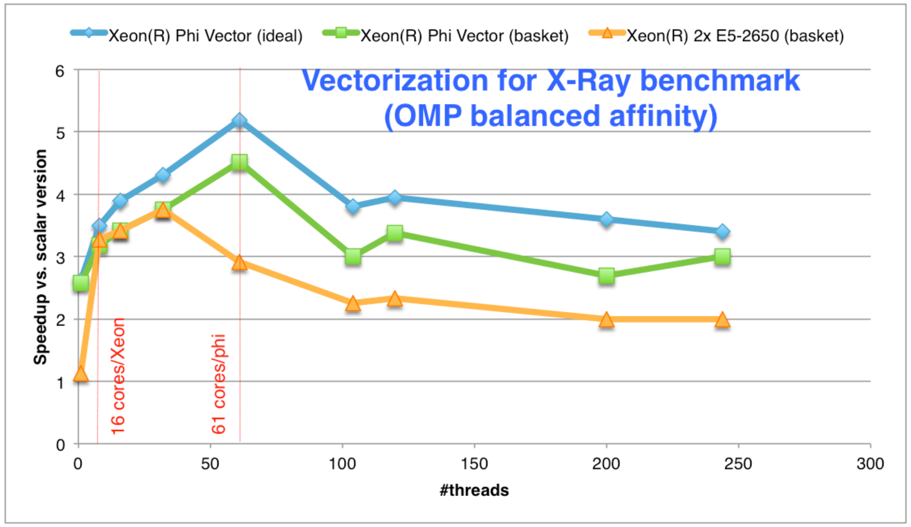
- The X-Ray benchmark tests geometry navigation in a real detector geometry
- X-Ray scans a module with virtual rays in a grid corresponding to pixels on the final image
  - Each ray is propagated from boundary to boundary
  - Pixel gray level determined by number of crossings
- A simple geometry example (concentric tubes) emulating a tracker detector used for Xeon©Phi benchmark
  - To probe the vectorized geometry elements + global navigation as task
  - OMP parallelism + "basket" model





Vector performance

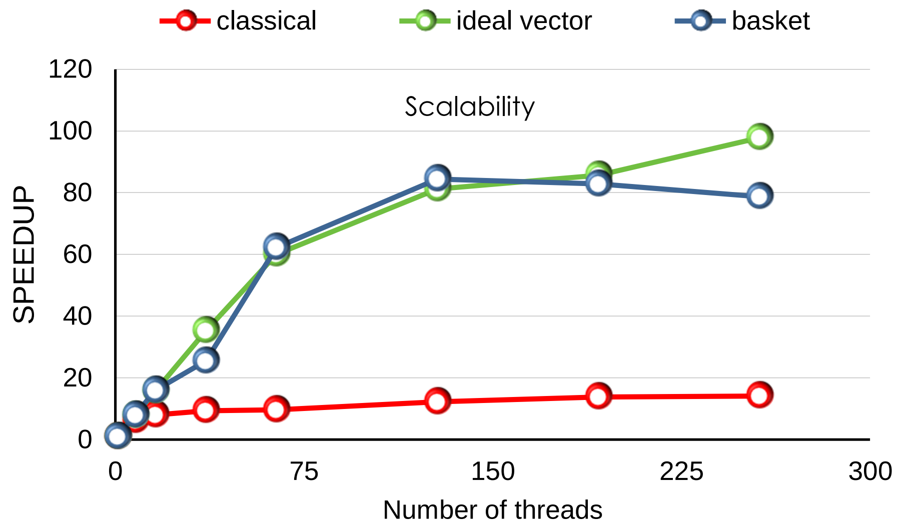
- Gaining up to 4.5 from vectorization in basketized mode
  - Approaching the ideal vectorization case (when no regrouping of vectors is needed)
- Vector starvation starts when filling more thread slots than the core count
  - Performance loss is not dramatic
  - Better vectorization compared to the Sandy-Bridge host (expected)

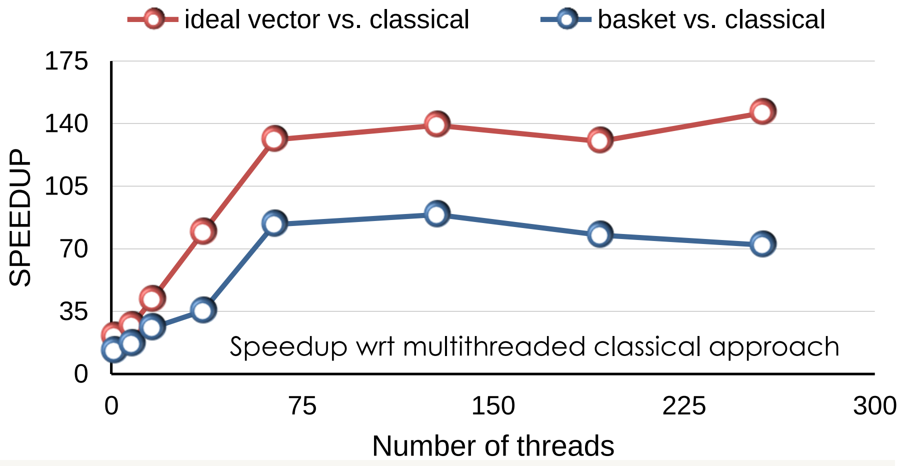


- Scalar case: Simple loop over pixels
- Ideal vectorization case: Fill vectors with N times the same X-ray
- Realistic (basket) case: Group baskets per geometry volume

Performance on KNL

- A first glance of results obtained on *Intel® Xeon Phi™ CPU 7210 @ 1.30GHz*, 64 cores
- Scalability comparable KNC vs. KNL for the ideal and basket versions (~100x)
  - No major difference observed between ”compact” and “balanced” OMP affinity
- GeantV approach gives excellent benefits with respect to the classical one (Geant4/ROOT)



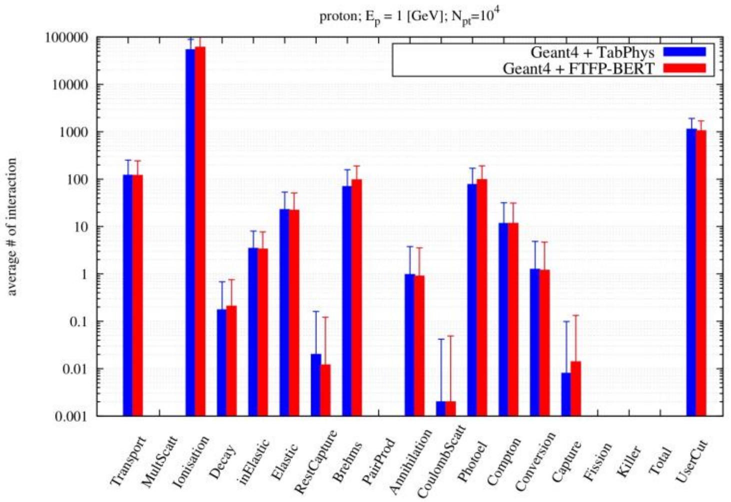


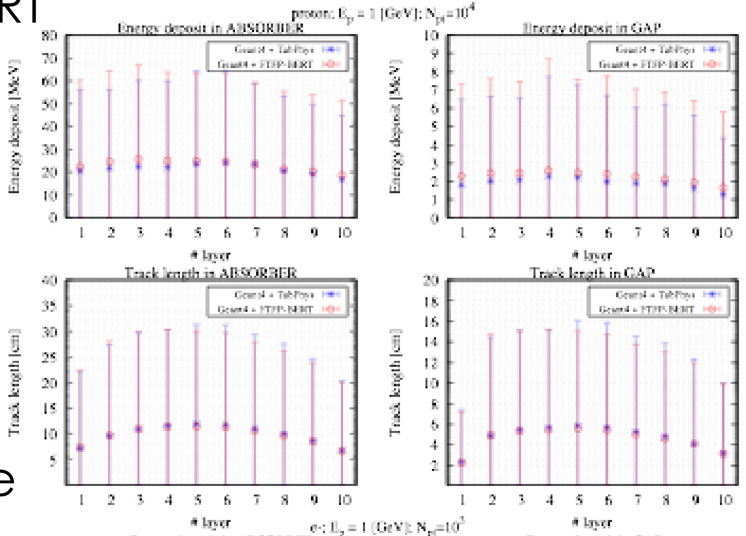
What about physics?

- Needed a “reasonable” shower development
- Developed a library of sampled interactions and tabulated x-sections for GeantV
  - Back ported to Geant4 for verification and comparison
- A quick tool for developing realistic showers
  - Potentially for developing into a fast simulation tool

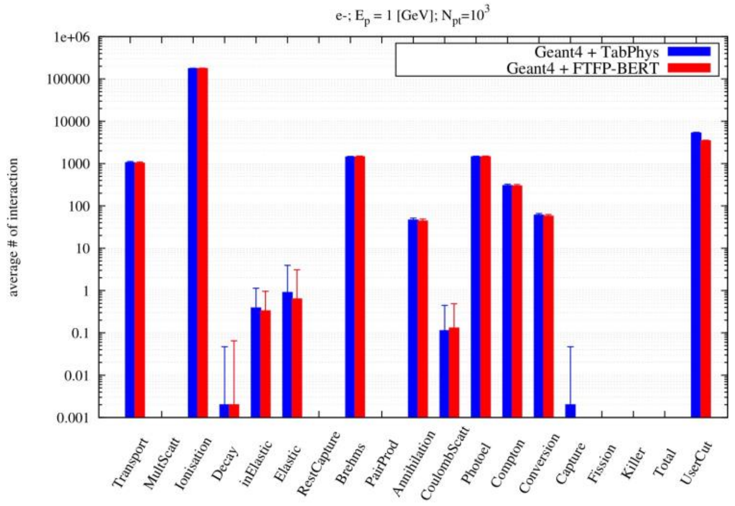
G4 example N03 vs tabulation
(simple calo with many slabs)

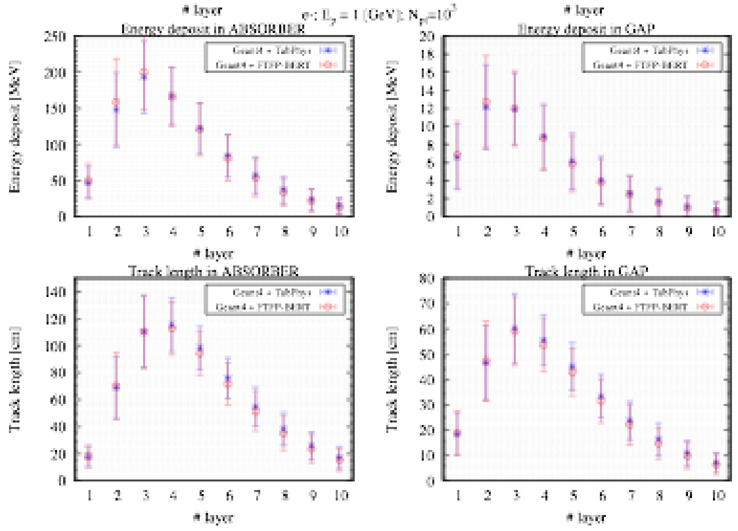
FTFP-BERT





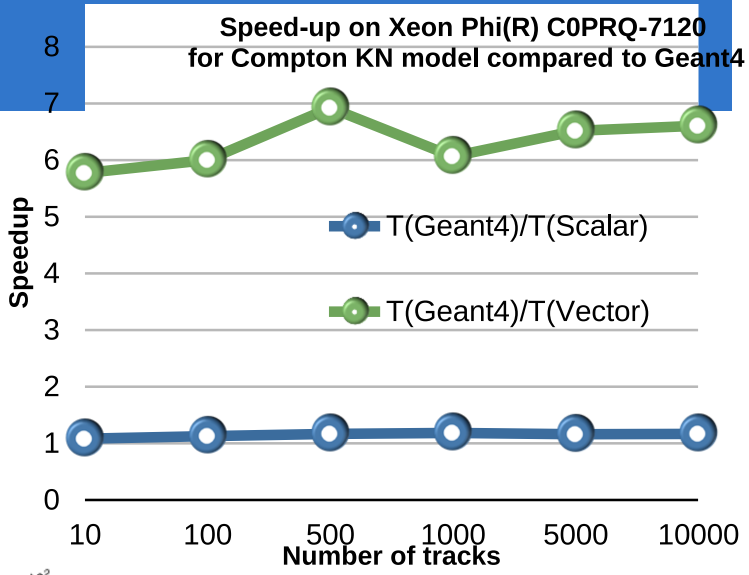
profile histos

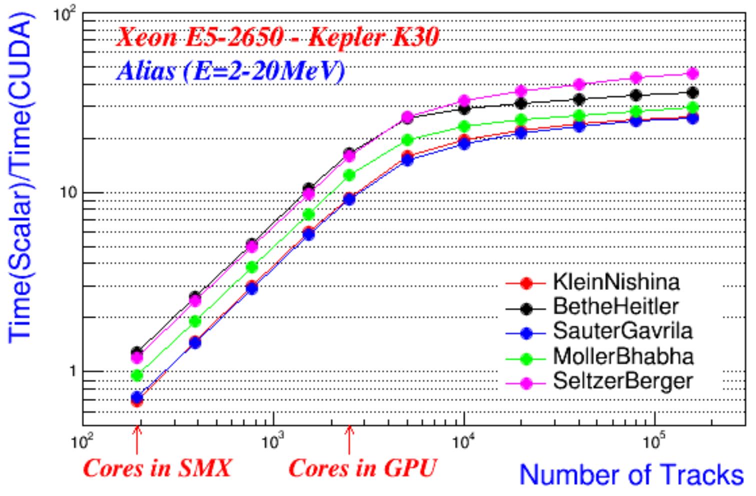


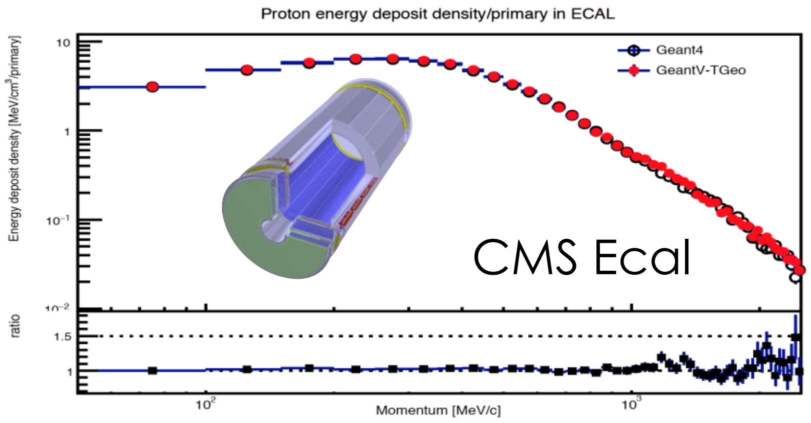


Physics Speed

- Objective: a vector/accelerator friendly re-write of physics code
- Started with the electromagnetic processes
- The vectorised Compton scattering shows good performance gains
- Current prototype able to run an exercise at the scale of an LHC experiment (CMS)
  - Simplified (tabulated) physics but full geometry, RK propagator in field
  - Very preliminary results needing validation, but hinting to performance improvements of factors



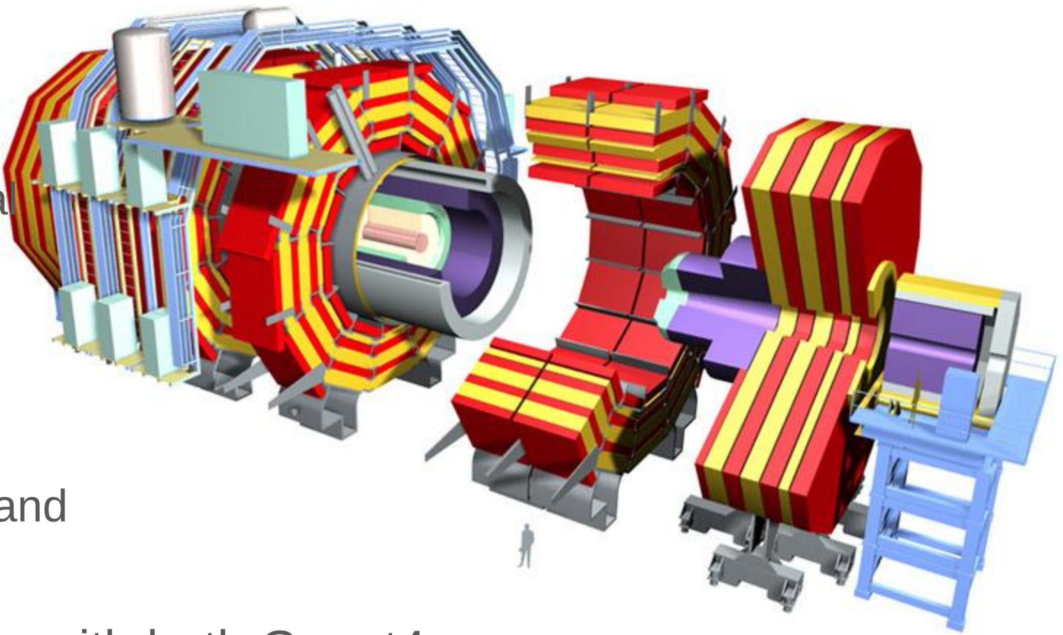




Yardstick: CMS With Tabulated Physics

Realistic Scale Simulation

- pp collisions @ 14TeV minimum bias events produced by Pythia 8
- 2015 CMS detector
- 4T uniform magnetic field
  - Decent approximation of the real solenoidal field
- Low energy cut at 1MeV
- 'Tabulated' Physics
  - Library of sampled interactions and tabulated x-sections
- ***Same*** test (described above) run with both Geant4 and GeantV with various versions of the Geometry library.



Putting It All Together - CMS Yardstick

Semantic changes

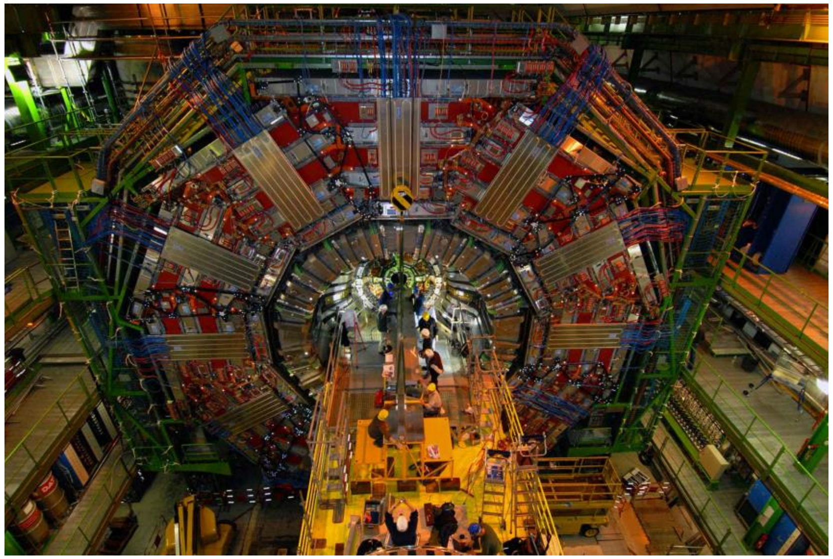
Scheduler	Geometry	Physics	Magnetic Field Stepper
Geant4 only	Legacy G4	Various Physics Lists	Various RK implementations
Geant4 or GeantV	VecGeom 2016 scalar	• Tabulated Physics • Scalar Physics Code	• Helix • Cash-Karp Runge-Kutta
GeantV only	• VecGeom 2015 • VecGeom 2016 vector • Legacy TGeo	Vector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

Putting It All Together - CMS Yardstick

$\overbrace{\text{Physics} \quad \text{Magnetic Field Stepper}}^{\text{Semantic changes}}$

Scheduler	Geometry	Physics	Magnetic Field Stepper
Geant4 only	**Legacy G4**	Various Physics Lists	Various RK implementations
Geant4 or GeantV	**VecGeom 2016 scalar** ③	• **Tabulated Physics** • Scalar Physics Code	• Helix (Fixed Field) • **Cash-Karp Runge-Kutta**
GeantV only	• **VecGeom 2015** ② • VecGeom 2016 vector • **Legacy TGeo** ①	Vector Physics Code	Vectorized RK Implementation

Putting It All Together - CMS Yardstick



- Some of the improvements can be back ported to G4
- Overhead of basket handling is under control
- Ready to take advantage of vectorization throughout.

Improvement Factors (total) with respect to G4

Legacy (TGeo) Geometry library:

- **1.5** → Algorithmic improvements in infrastructure.

2015 VecGeom (estimate)

- **2.4** → Algorithmic improvements in Geometry

Current VecGeom

- **3.3** → Further Geometric algorithmic improvements and some vectorization

Interaction with Frameworks

- Threading technology
  - Currently using std::thread to steer ‘tasks’ (but no thread local storage).
  - The number of threads used is configurable.
  - Testing OpenMP/MPI to steer Xeon Phi in offload mode and with separate processes.
  - Exploring if we can benefit from TBB and how to best coordinate with other uses
- Coprocessors
  - GPU (and Xeon Phi) can be used optionally via plugin
  - Enabling use of the coprocessor in offload mode will be done via a function call.
- Coprocessor sharing
  - Number of CUDA threads and blocks used by GeantV is customizable
  - Newer NVidia hardware support concurrent execution of independent kernels (in addition to the queue mechanism that was supported for a very long time).

Interaction with Frameworks

- Events/Data in and out from Frameworks
  - Still under design
  - Input
    - One or more initial particles coming from one or more events will be passed on
  - Output
    - Upon completion of the propagation of all the particles for an event, a call back will be made (CMSApplication::Digitize for example)
  - Should find a way to recast this flow into one of the TBB task mechanism

- Memory handling
  - Tied to the number of event in flight and the size of the output information
  - High watermark used to trigger a reduction in number of events in flight to limit memory usage to under the watermark

Engagement

- VecGeom alpha release ready to be tested as Geant4 geometry update.
  - Magnetic field code update will also eventually be available to Geant4

- VecCore can be used to develop technology agnostic vector code

- GeantV ready to get out of the laboratory
  - Starting to think about/design interfaces for user actions, digitization, etc.
  - Welcoming early stakeholders to start reviewing the interfaces needed for a full application and develop more realistic tests and prototypes

Restating our case

- We developed the three main components
  - A multithread scheduler to handle the particle baskets
  - A vectorised geometry library and navigator
  - A vectorised Compton scattering and a tabulated physics list
- Our results indicate that
  - Basket handling introduces a minimal overhead
  - SIMD gains half an order of magnitude in performance
- An optimistic prediction based on our results gives an improvement factor ***beyond the 3.3*** currently achieved on CPU
- GPU and Xeon Phi improvement factors are expected to be higher

We are on track with achieving our objectives (see slide 5: a 2 to 5 speedup)

Thank you!

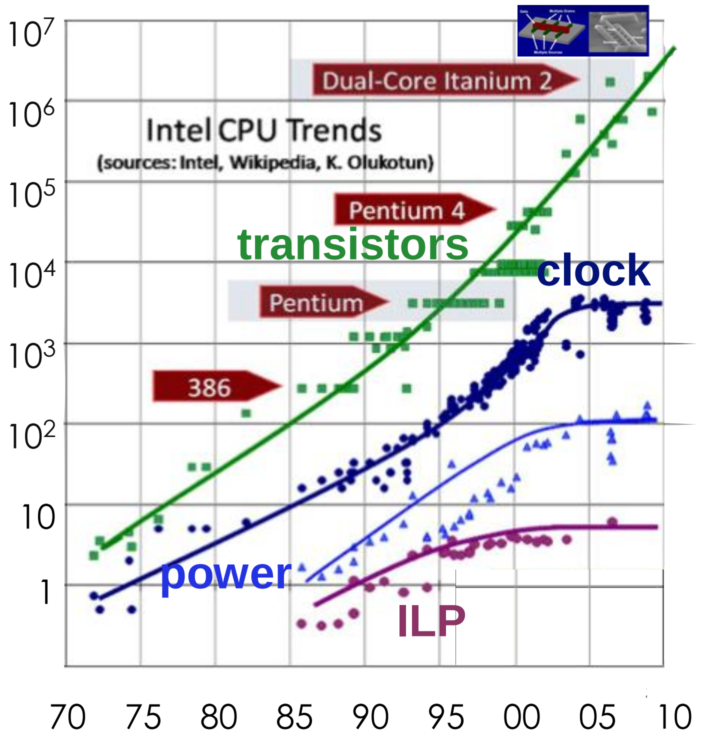
BACKUPS

Next steps

- Repeat the test with the introduction of
  - Vectorised EM physics
  - Vectorised transport in Mag Field
- Develop simple classes for materials and particles to be able to run on coprocessors to enable physics on the GPU and Xeon Phi full CMS yardstick
- … implementing a “preliminary performance yard-stick” combining all prototype features
  - SIMD gains in the full CMS experiment setup
  - Coprocessor broker in action: part of the full transport kernel running on Xeon®Phi® and GPGPU
  - Scalability and NUMA awareness for rebasketizing procedure
  - … achieving these just moves the target a bit further
- … testing scaling up to large node count through MPI, e.g. on CORI
  - Input distribution and Output gathering.

GeantV: (familiar) motivations

- Performance of our code scales with clock cycle (hence is stagnant!)
- Needs will increase more than tenfold and the budget will be constant at best
- HEP code needs to exploit new architectures and to team with other disciplines to share the optimization effort
  - Data & instruction locality and vectorisation
- Portability, better physics and optimization will be the targets
- Simulation can lead the way to show how to exploit today's CPU's resources more effectively in complex applications



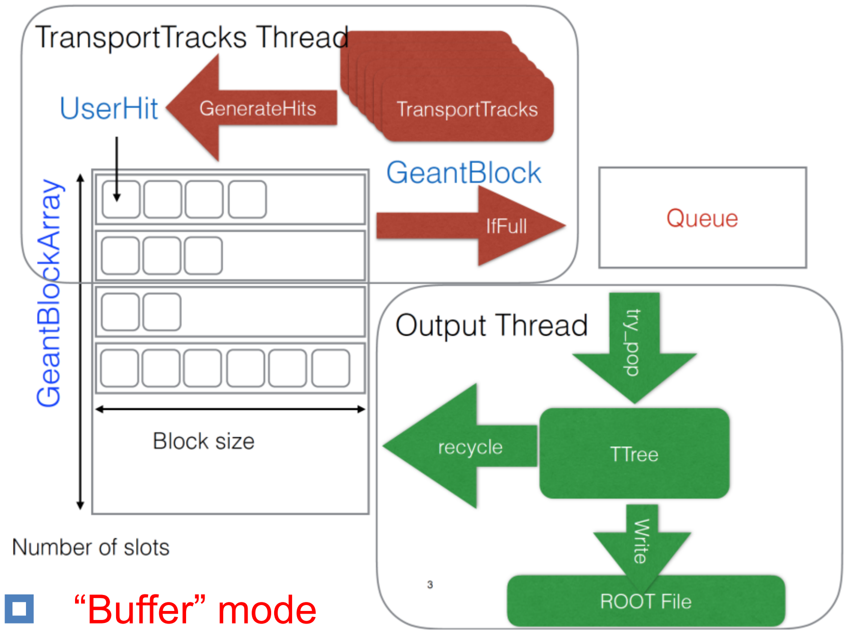
- Seeking ways to write code portable between CPU with vector units or not and accelerators (GPU, Xeon Phi)

GeantV Output

- Physics simulation produces ‘hits’ i.e. energy depositions in the sensitive parts of the detector
- Those hits are produced concurrently by all the simulation (TransportTracks) threads
- Thread-safe queues have been implemented to handle asynchronous generation of hits by several threads
- Dedicated Output thread transfers the data from the output queues to ROOT I/O

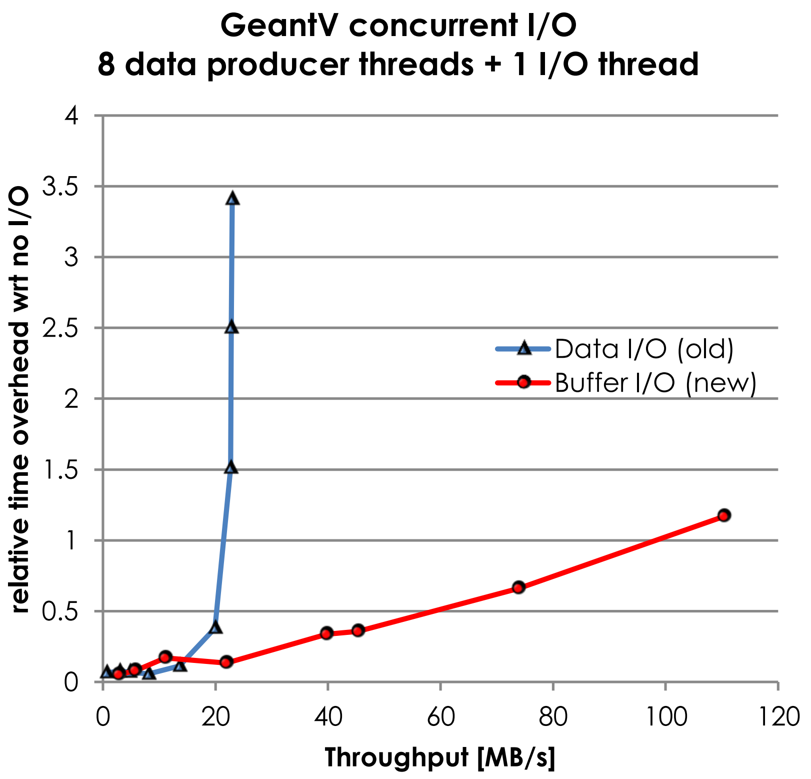
Hits/digits I/O

- "Data" mode
  - Send concurrently data to one thread dealing with full I/O



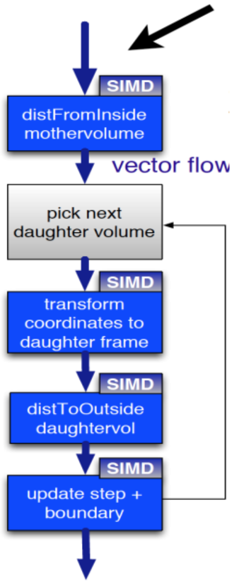
- "Buffer" mode
  - Send concurrently local trees connected to memory files produced by workers to one thread dealing with merging/write to disk

- Integrating user code with a highly concurrent framework should not spoil performance

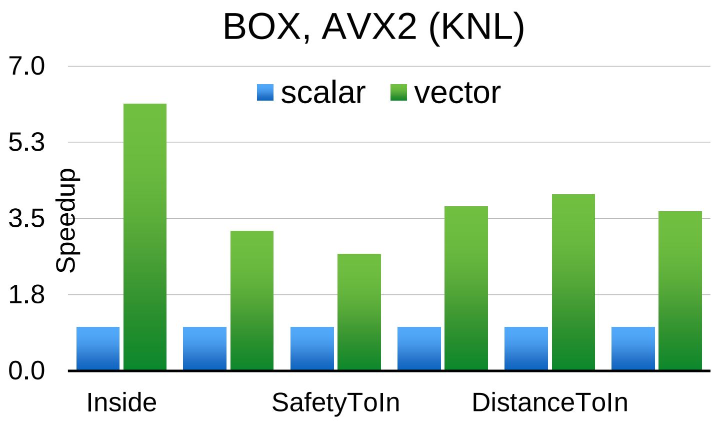


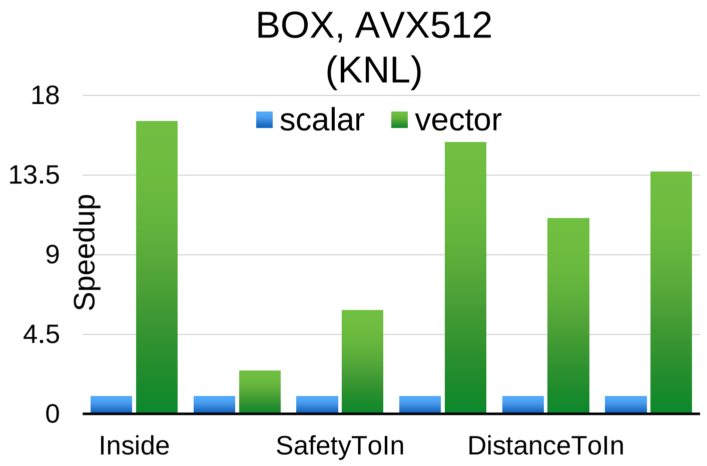
Geometry performance on KNL

- Running set of standard geometry benchmarks using UME::SIMD backend.
- Measuring vector versus scalar speed-up using AVX2 and AVX512, for CPU-intensive geometry navigation methods
- Observe super-linear speedup for some methods
- Investigating if it is compiler-related
  - Vector interface is better than scalar one (~x2 factor) w/o auto-vectorization
  - Found ~10% scalar performance improvement on KNL switching off auto-vectorization and setting different ISA options (AVX512 vs AVX2)


SIMD
distFromInside mothervolume
vector flow
pick next daughter volume
SIMD
transform coordinates to daughter frame
SIMD
distToOutside daughtervol
SIMD
update step + boundary

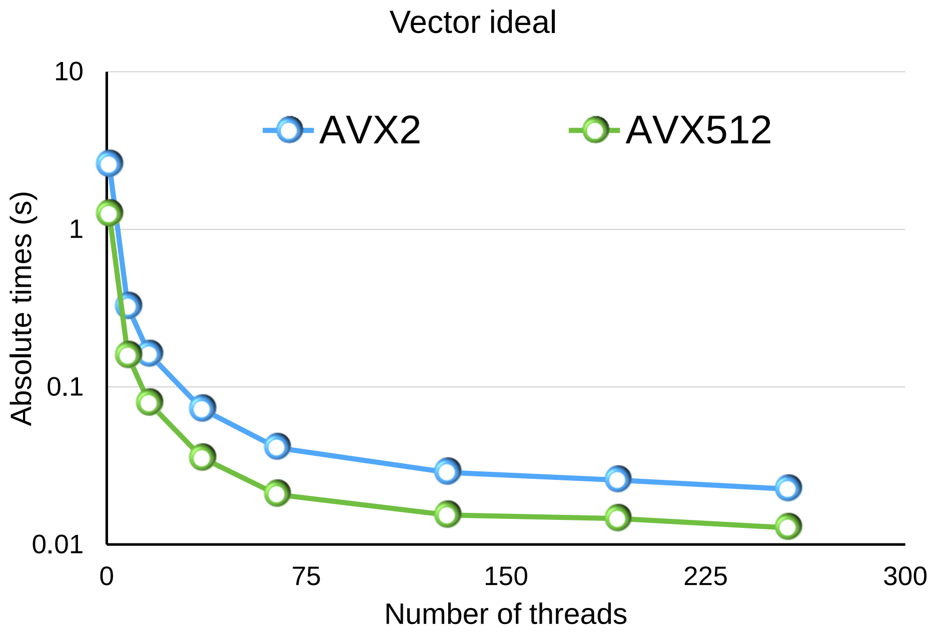
Intel® Xeon Phi™ CPU 7210 @ 1.30GHz, 64 cores

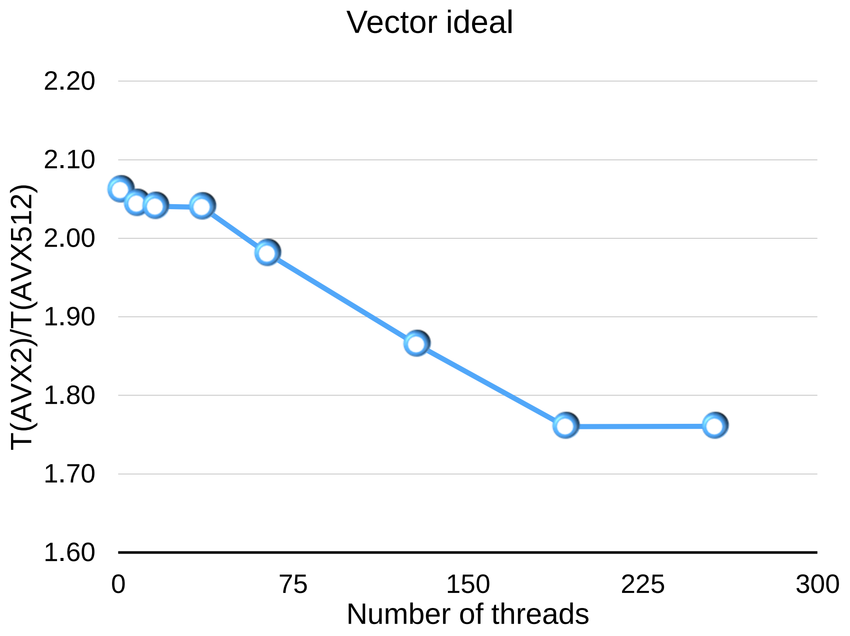




AVX-512 versus AVX2 on KNL

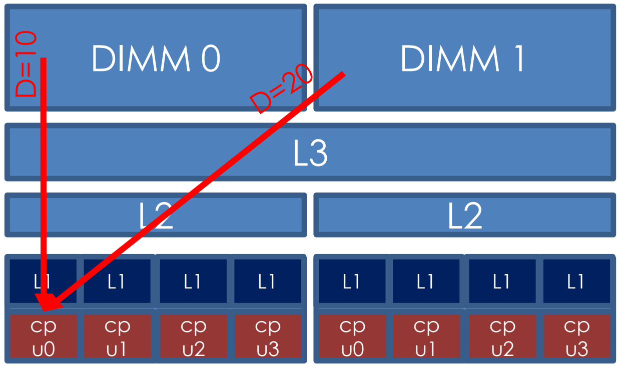
- High vectorization intensity achieved for both ideal and basketized cases
  - AVX-512 brings an extra factor of ~2 to our benchmark

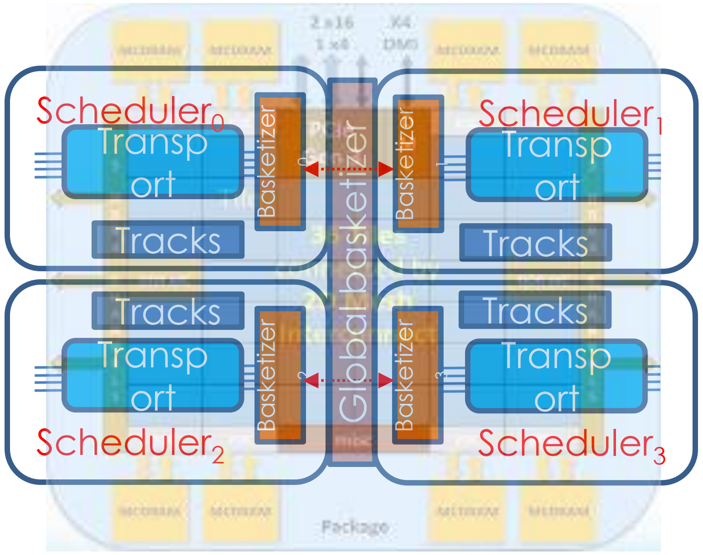




NUMA awareness

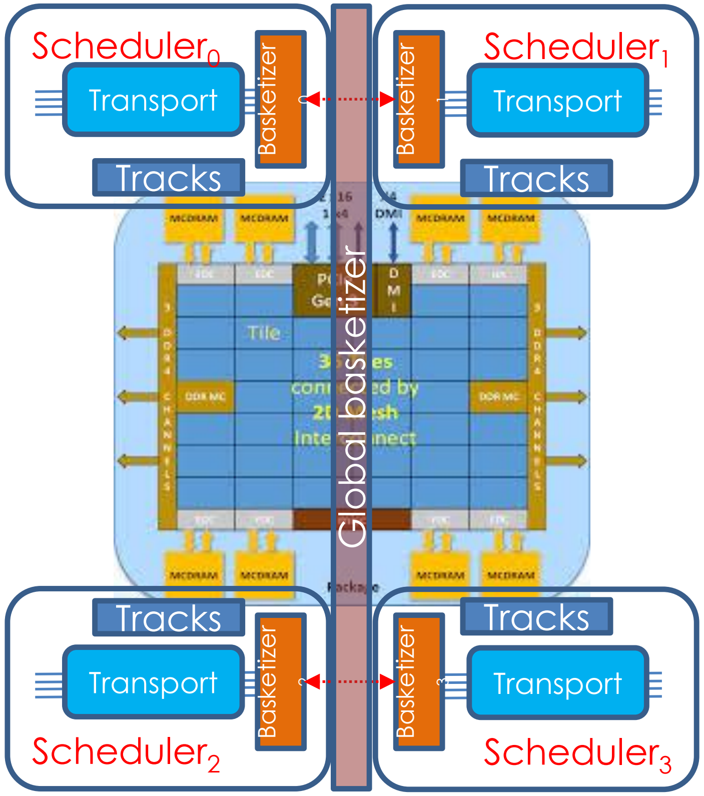
- Latency of memory access depends on "locality distance"
- Libraries used:
  - libnuma, numactl – NUMA memory & thread affinity policies
    - Developed by SUSE Labs & SGI
    - Most linux flavors, LGPL license
  - Portable Hardware Locality (hwloc) – NUMA topology detection, API & tools
    - Developed within Open MPI
    - NewBSD license
- Layer on top of *libnuma* & *libhwloc* to control affinity





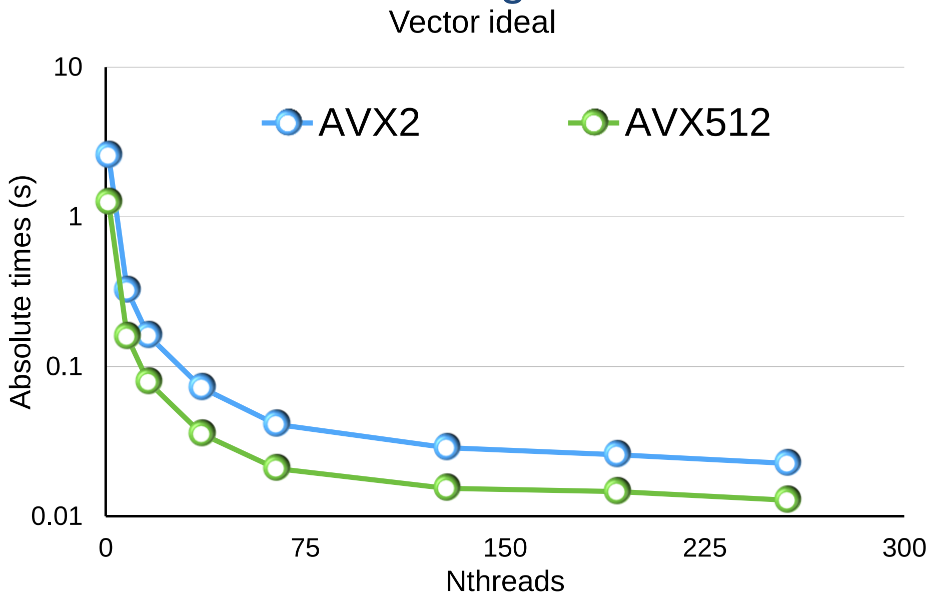
NUMA aware GeantV to be tested on KNL (SNC mode)

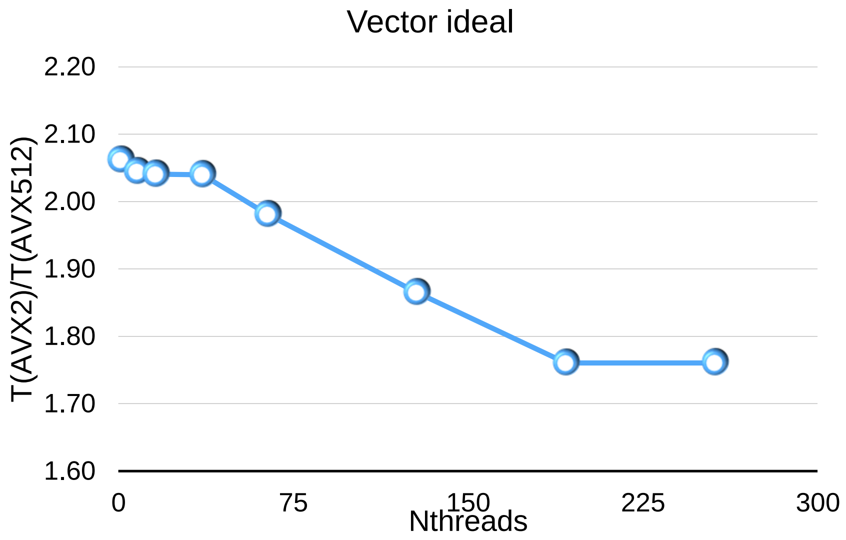
- Replicate schedulers on NUMA clusters
  - One basketizer per NUMA node
- 2 supported modes
  - MPI dispatch running one GeantV process per NUMA node
  - Single process spawning one scheduler per NUMA node
    - Loose communication between NUMA nodes at basketizing step
  - Currently under development



AVX-512 versus AVX2 on KNL

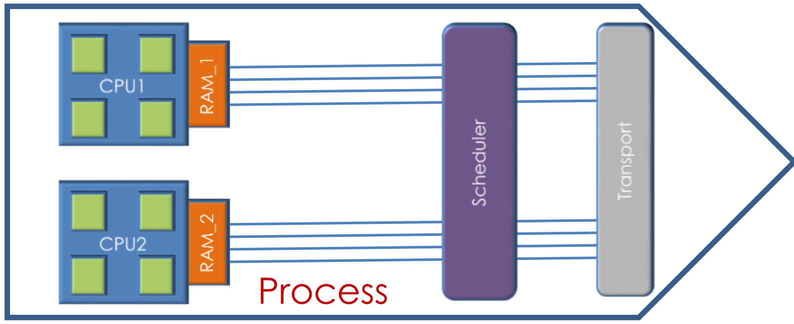
- High vectorization intensity achieved for both ideal and basketized cases
  - AVX-512 brings an extra factor of ~2 to our benchmark



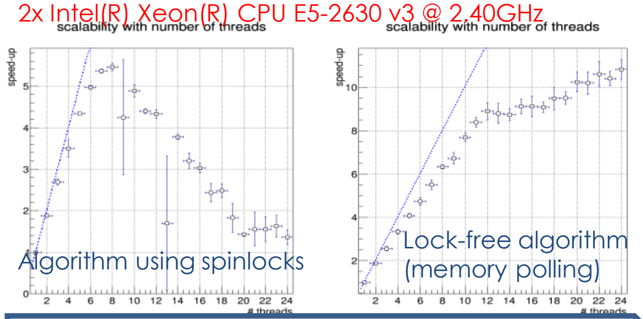


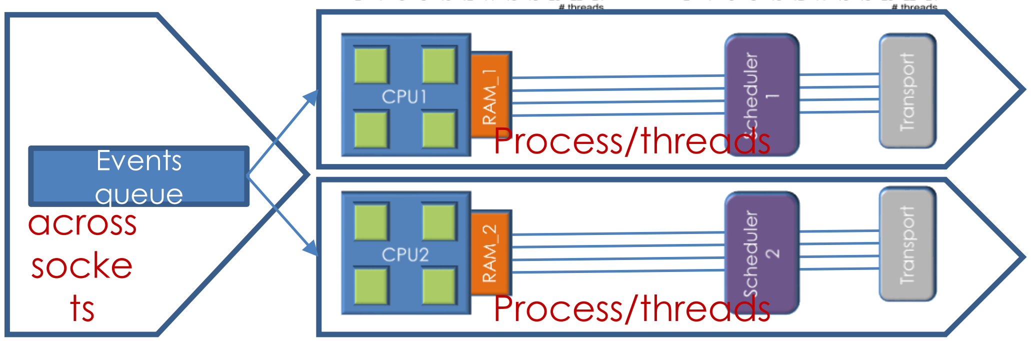
Scalability on many-core

- Fine grain MT preventing to scale to high number of threads
  - Issue for many core architectures
- Split application in (NUMA-aware) clusters and use a common event queue for workload balancing
  - Lightweight/no interaction
  - Memory friendly
  - Possible to extend across sockets, replacing the concurrent queue with an event server using MPI channels



Rebasketizing
2x Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz

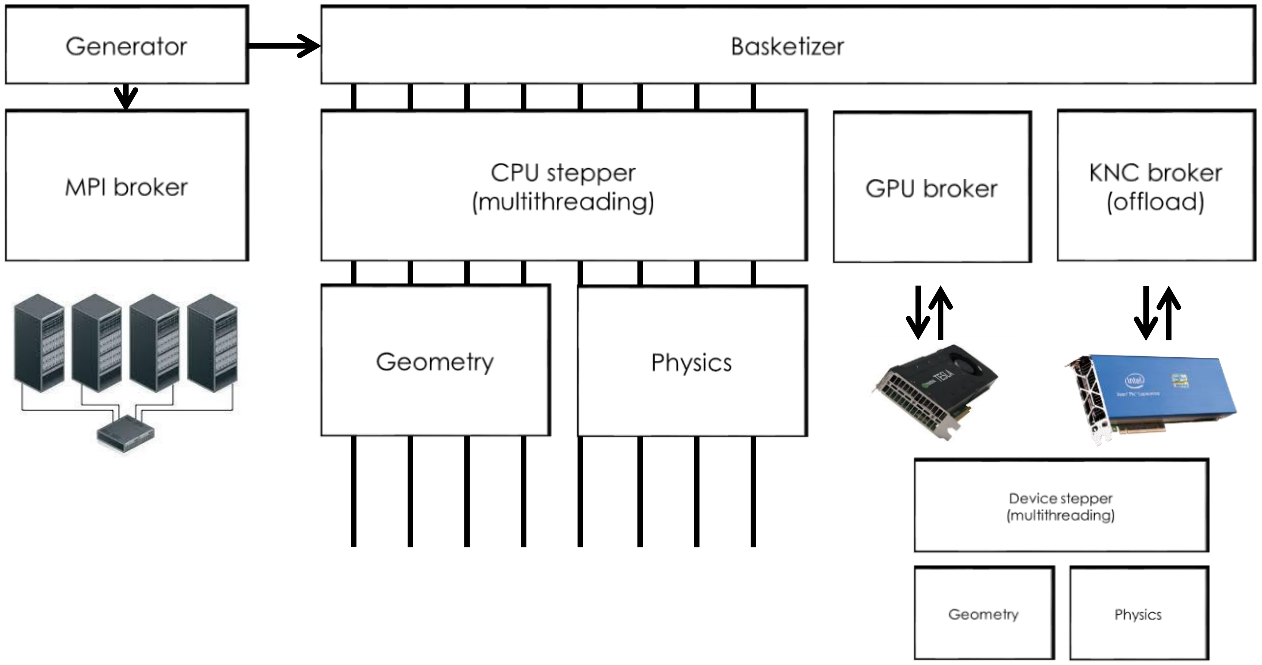




Going CUDA – beyond writing custom kernels

GeantV scheduler can communicate with arbitrary device brokers

- Getting work and processing in native mode or processing steps of work and sending data back to the host
- Implemented so far: CUDA broker, KNC offload interface. KNL will work in native mode



Host code instrumented with __host__ __device__ macros

Instrumented methods compiled in ::cuda namespace

Library compiled for host and device (nvcc)

GPU broker dealing with initialization on device and data copying

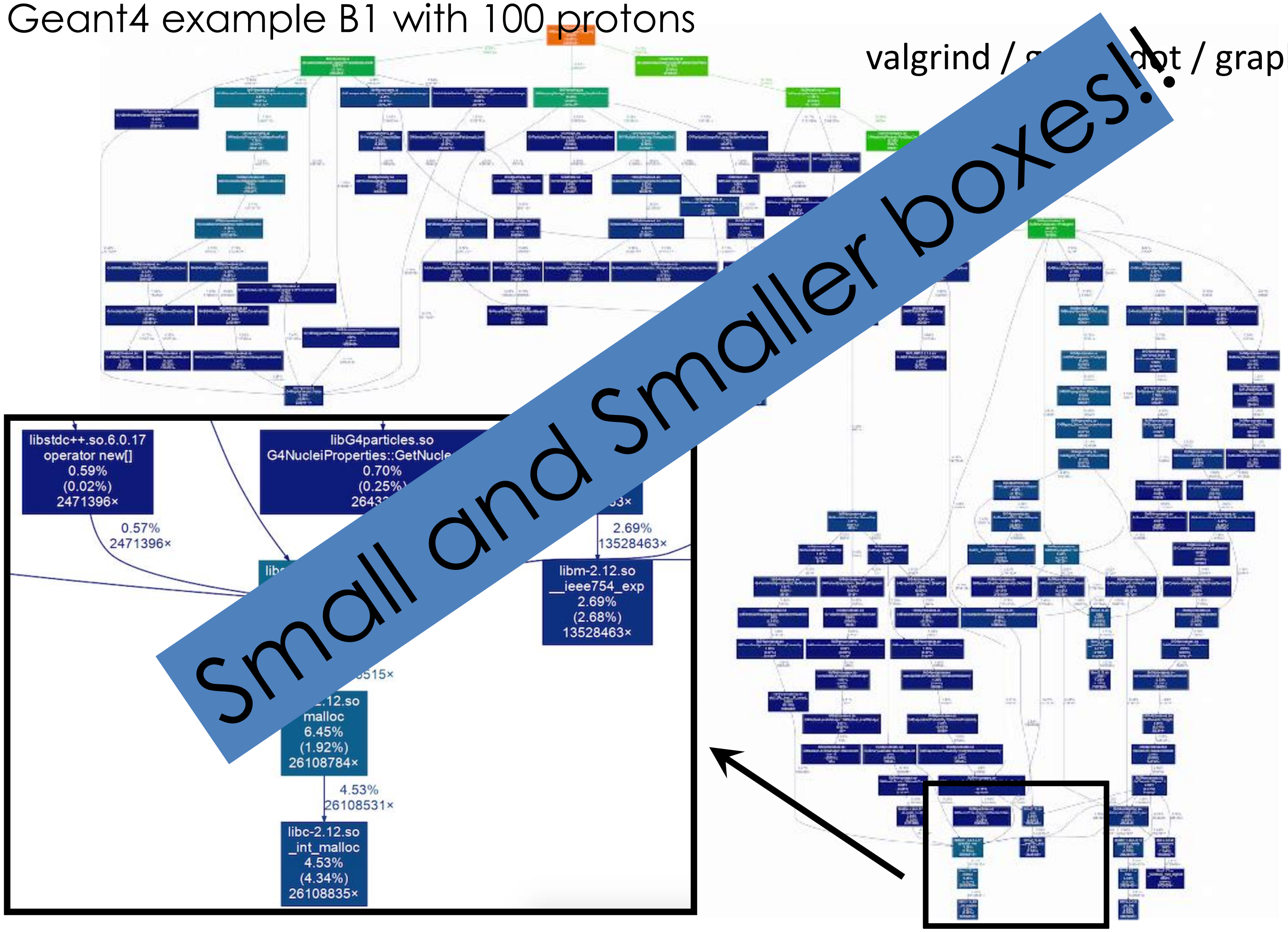
Why not Geant4+?

- Extensive prototyping and analysis has convinced us that “vectorisation” of Geant4 was not achievable without a major rewrite of the code
  - No hotspots (!)
  - Virtual table structure very deep and complex (1990’s style)
  - Codebase very large and non-homogeneous
  - Auto-vectorization can only have small and very localized effect
- No criticism, but even the best things age (born 1994)

Geant4 Profiling Example: Call Graph

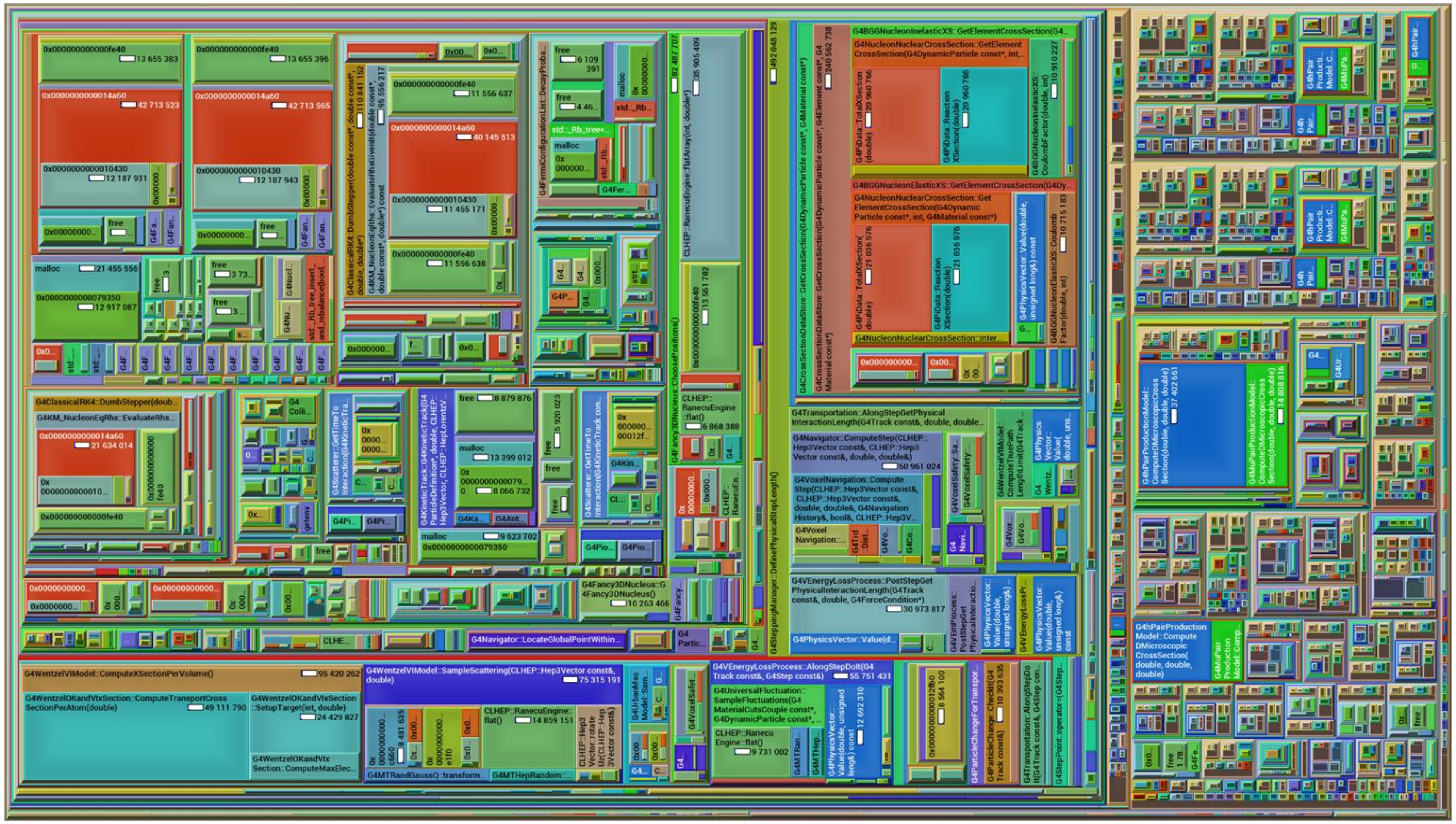
Geant4 example B1 with 100 protons

valgrind / gprof2dot / graphviz



Geant4 Profiling Example: Call Map

valgrind / kcachegrind

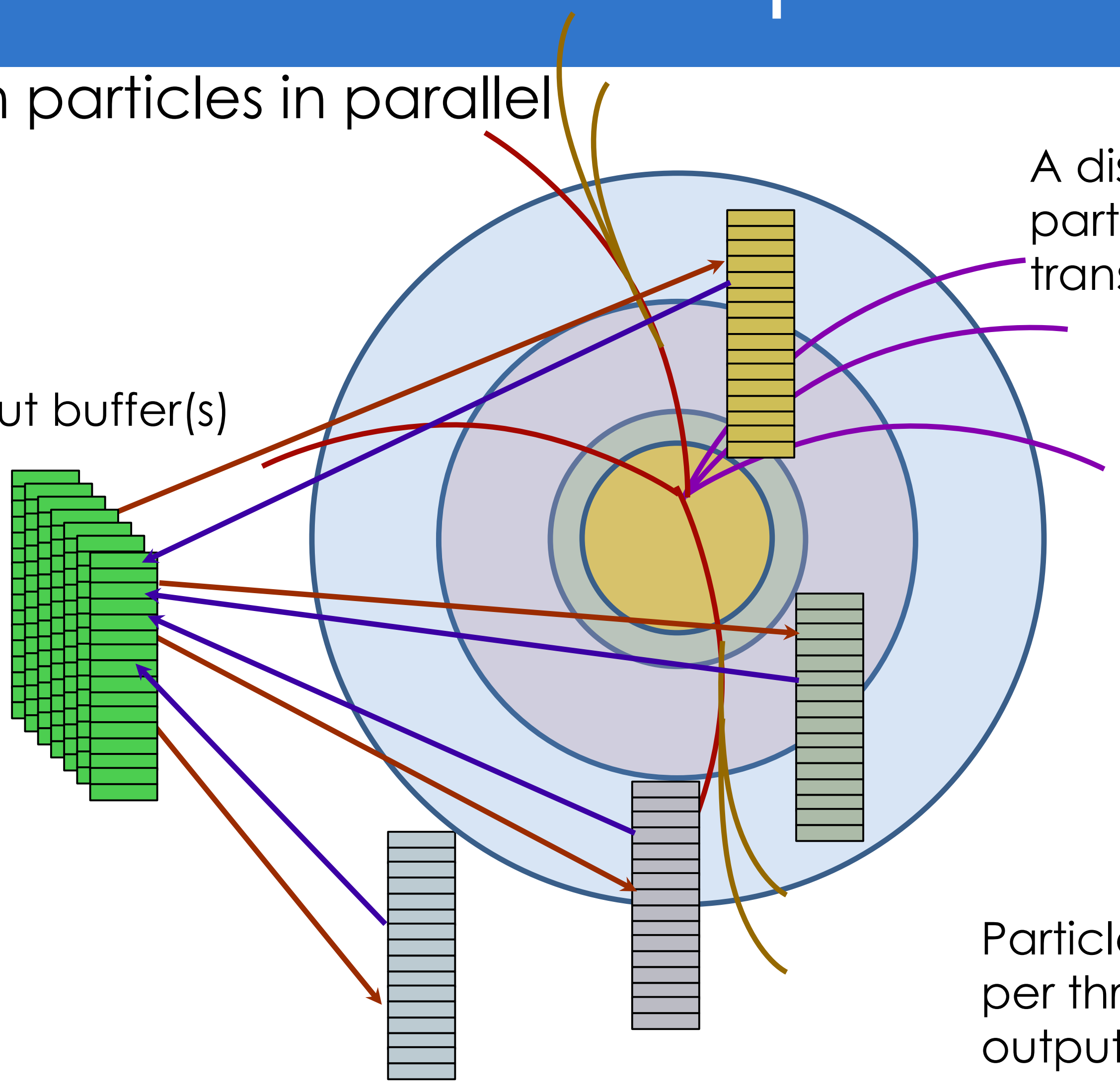


G4SteppingManager::DefinePhysicalStepLength()
G4BGGNucleonInelasticXS::GetElementCrossSection(G4...
G4NucleonNuclearCrossSection::GetElementCrossSection(G4DynamicParticle const*, int,...
G4BGGNucleonElasticXS::GetElementCrossSection(G4Dy...
G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track const&, double, double...
G4Navigator::ComputeStep(CLHEP::Hep3Vector const&, CLHEP::Hep3Vector const&, double, double&)
50 961 024
G4VEnergyLossProcess::PostStepGetPhysicalInteractionLength(G4Track const&, double, G4ForceCondition*)
30 973 817
G4WentzelVIModel::ComputeXSectionPerVolume()
95 420 262
G4WentzelVIModel::SampleScattering(CLHEP::Hep3Vector const&, double)
75 315 191
G4VEnergyLossProcess::AlongStepDoIt(G4Track const&, G4Step const&)
55 751 431
G4WentzelOKandVIxSection::ComputeTransportCrossSectionPerAtom(double)
49 111 790
G4WentzelOKandVIxSection::SetupTarget(int, double)
24 429 827
G4UniversalFluctuation::SampleFluctuations(G4MaterialCutsCouple const*, G4DynamicParticle const*,...
CLHEP::RanecuEngine::flat()
14 859 151
9 731 002
G4ClassicalRK4::DumbStepper(doub...
G4KM_NucleonEqRhs::EvaluateRhs...
G4Navigator::LocateGlobalPointWithin...
G4hPairProductionModel::ComputeDMicroscopicCrossSection(double, double, double)
37 402 661
G4MuPairProductionModel::ComputeDMicroscopicCrossSection(double, double, double)
14 808 816
G4Fancy3DNucleus::G4Fancy3DNucleus()
10 263 466
CLHEP::RanecuEngine::flatArray(int, double*)
malloc
free
492 048 129
82 487 707
35 905 409

No easy to address hotspots

“Basketised” transport

Deal with particles in parallel



A dispatcher thread puts particles back into transport buffers

Everything happens asynchronously and in parallel

The challenge is to minimise locks

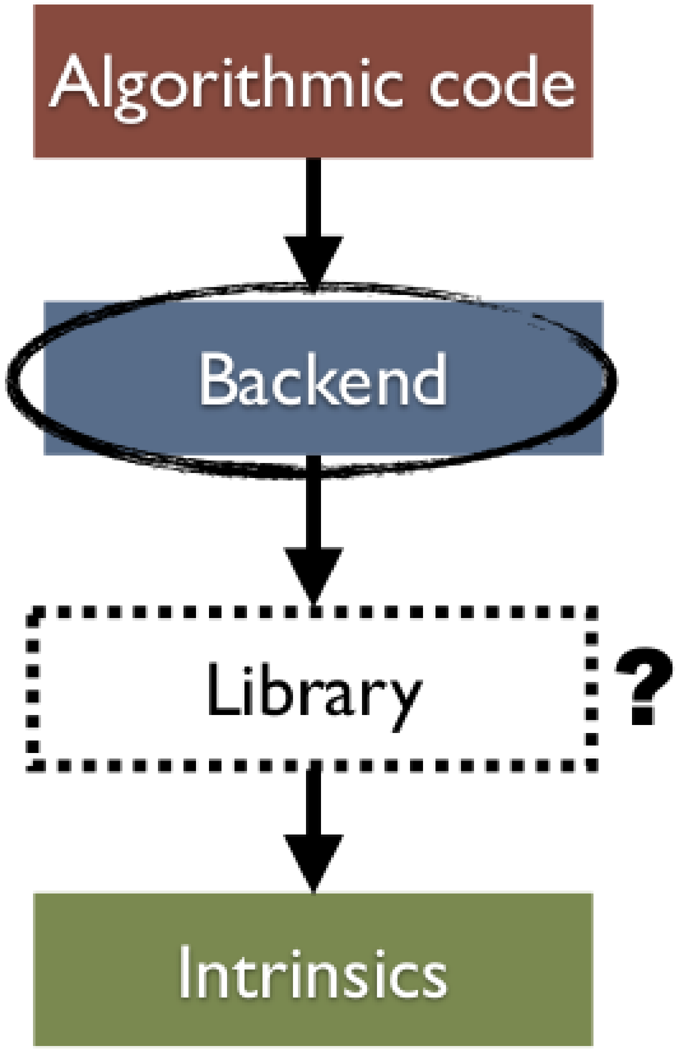
Keep long vectors

Avoid memory explosion

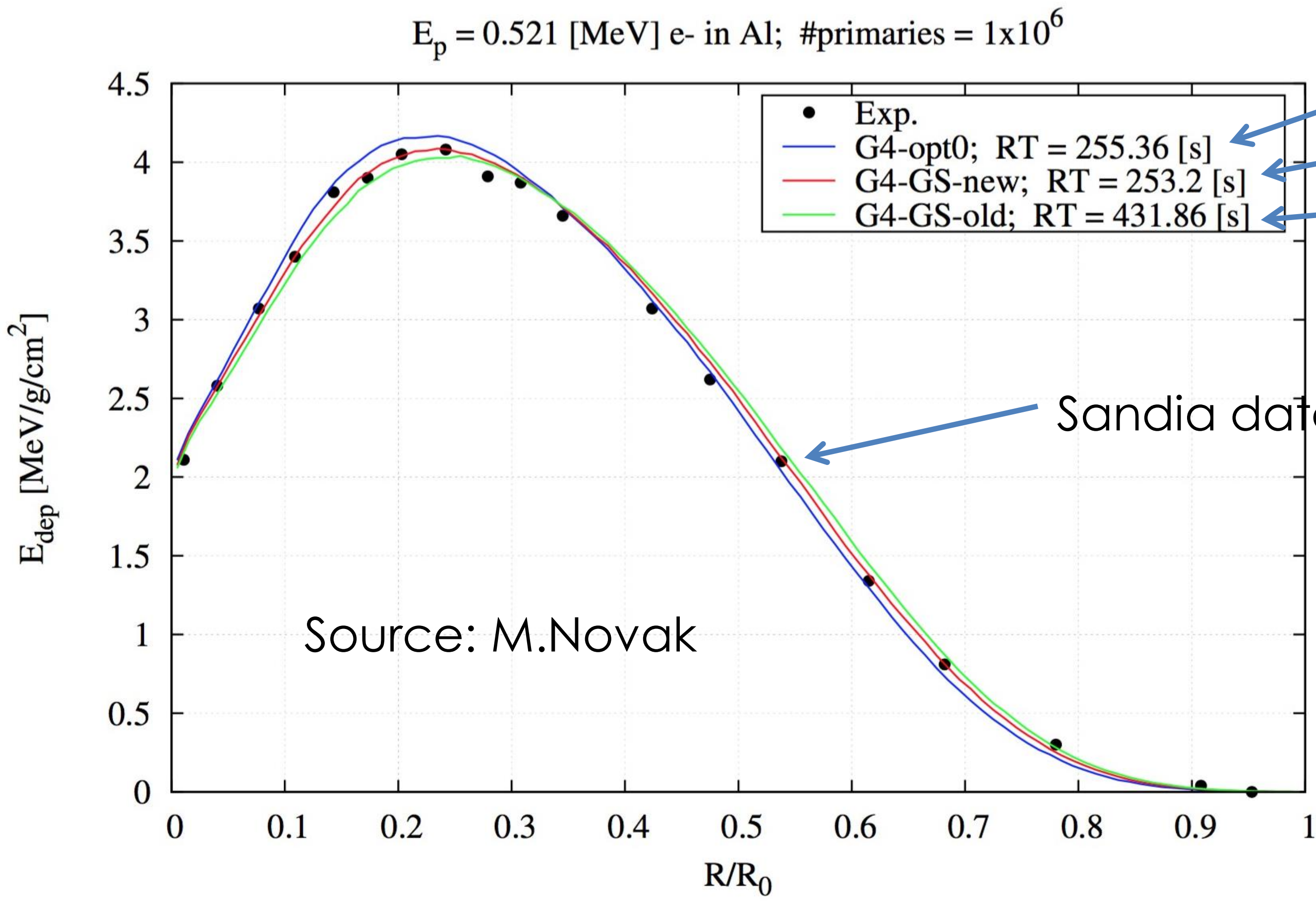
Particles are transported per thread and put in output buffers

Explicit vectorization

- Explicit SIMD vectorization can be implemented directly using intrinsics, but a vectorization library already brings many utilities pre-defined, like common math operators and functions.
- VecGeom currently works with Vc library, by Mathias Kretz, but other libraries can be easily plugged in (Agner Fog's VCL, Intel's VML, Cilk Plus, …).
  A new backend is maybe all that is needed.



Physics developments: Multiple Scattering



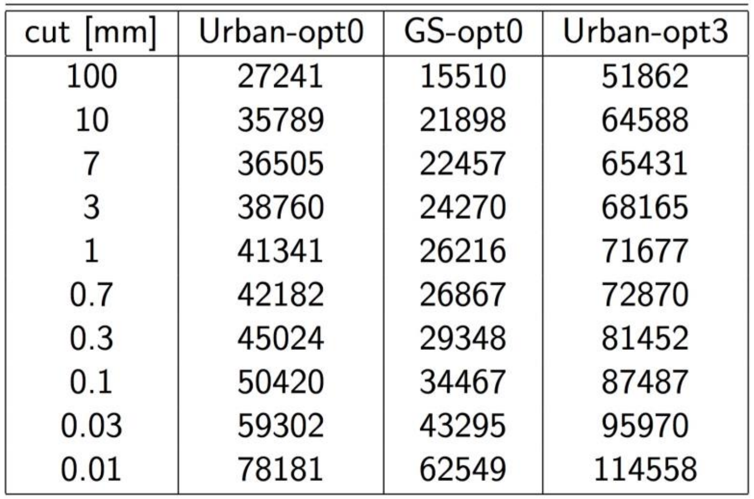
G4 fast

GV

G4 precision

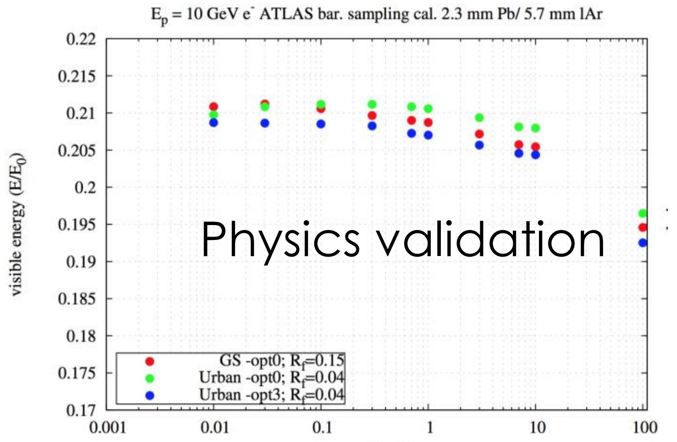
GeantV Multiple Scattering

Less charged steps (30%-50%)

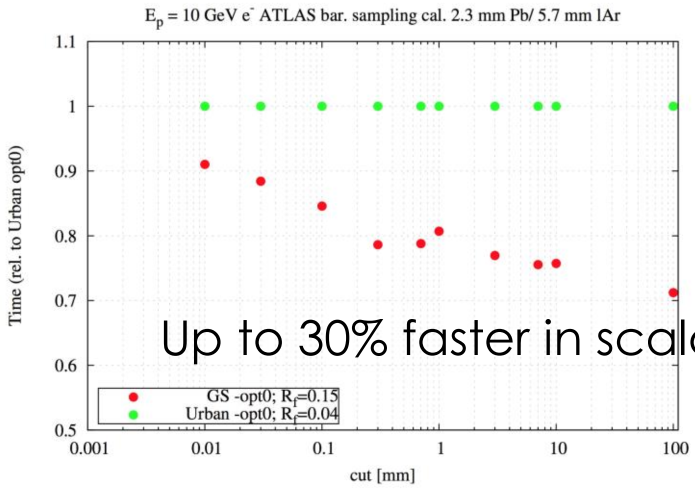


cut [mm]	Urban-opt0	GS-opt0	Urban-opt3
100	27241	15510	51862
10	35789	21898	64588
7	36505	22457	65431
3	38760	24270	68165
1	41341	26216	71677
0.7	42182	26867	72870
0.3	45024	29348	81452
0.1	50420	34467	87487
0.03	59302	43295	95970
0.01	78181	62549	114558

Table: number of charged steps



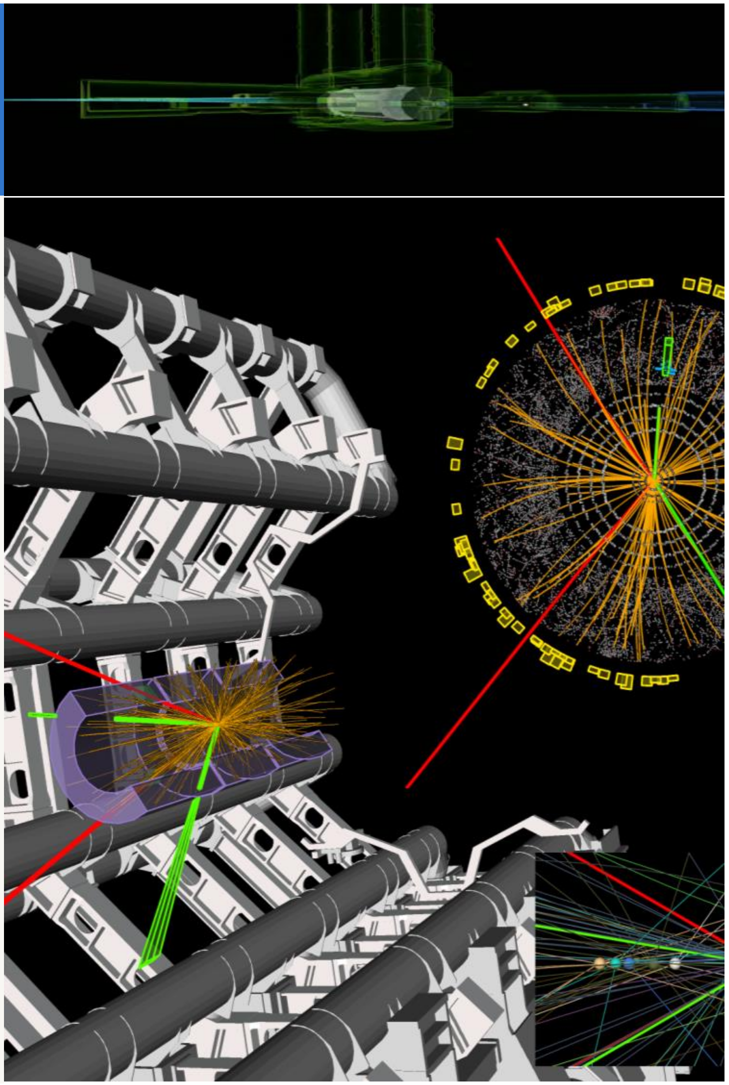
Physics validation



Up to 30% faster in scalar

- The new algorithm is being now vectorised for GeantV
- It is in an experimental physics list for Geant4
  - Candidate to become the default

The problem


http://atlas.ch

Detailed simulation of subatomic particle transport and interactions in detector geometries

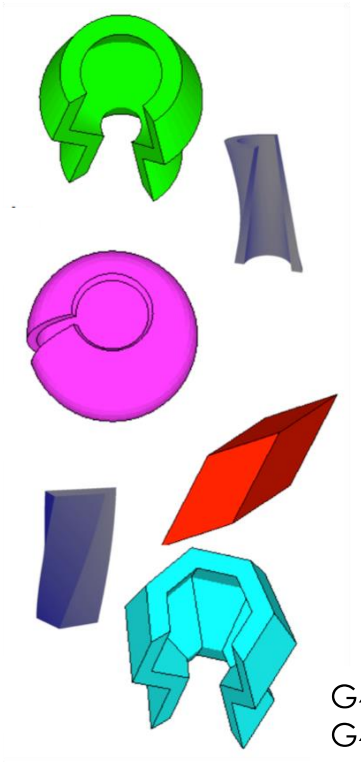
Using state of the art physics models, propagation in electromagnetic fields in geometries having complexities of millions of parts

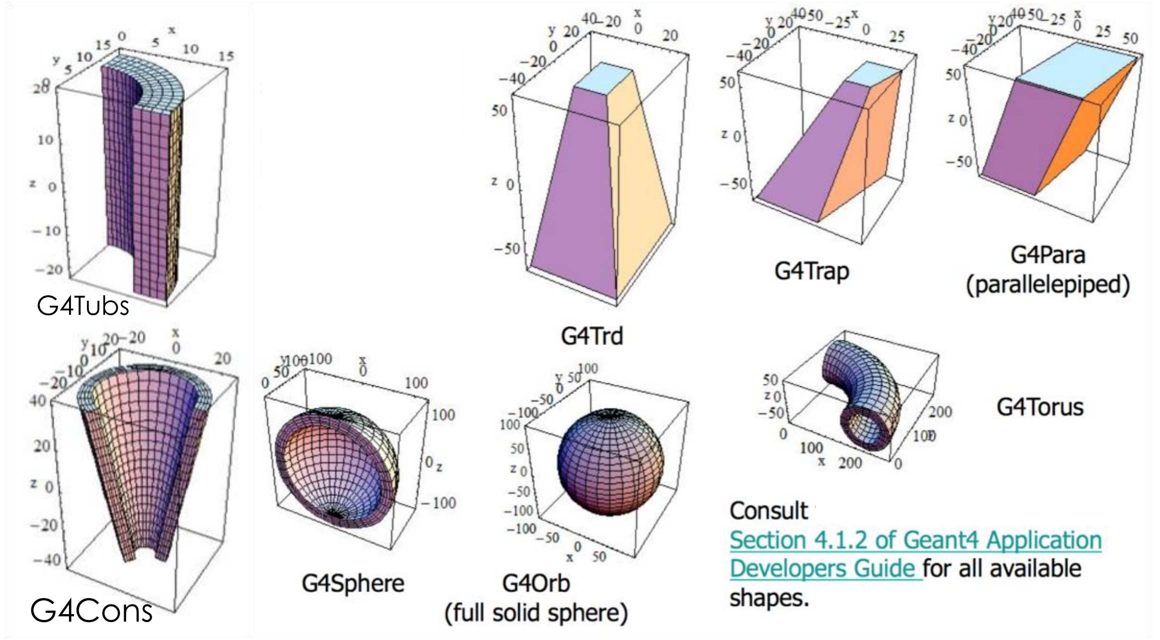
Heavy computation requirements, massively CPU-bound, seeking organically HPC solutions…

The LHC uses more than 50% of its distributed GRID power for detector simulations (~250.000 CPU years equivalent so far)

Geant4 Geometry

A large collection of solids are defined in Geant4:

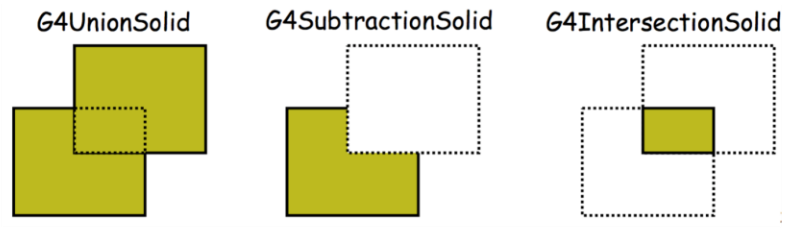




Consult [Section 4.1.2 of Geant4 Application Developers Guide] for all available shapes.

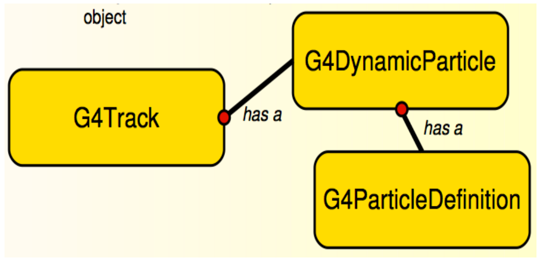
G4Polycone, G4 Polyhedra, G4Hype, G4TwistedTubs, G4TwistedTrap

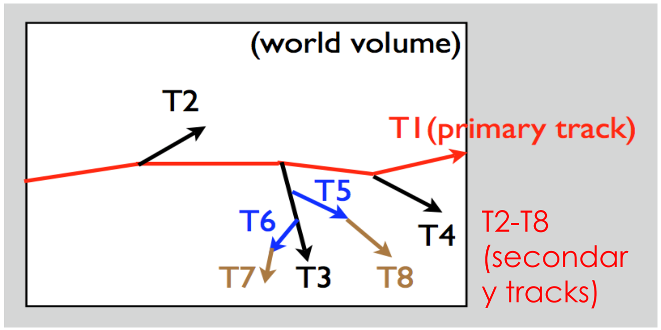
Also Boolean operations such as:



Geant4 Transportation

A G4Track also includes the info for transporting the particle through the detector ➔ in G4 we typically use Particle = Track

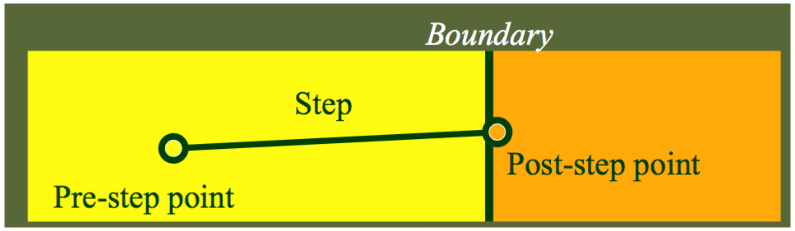




Tracking follows "last in first out" rule: T1->T4->T3->T6->T7 ->T5->T8->T2

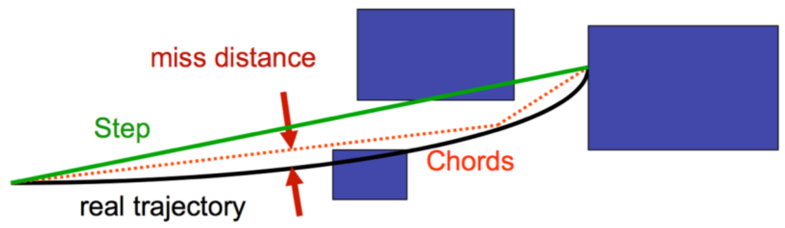
The G4Track information is updated after every G4Step

- A G4Step is a step in the particle (track) propagation
- The user defines a maximum step length but steps also end when a physics process is invoked and at volume boundaries



Geant4 Magnetic Field

Particle propagated in EM field by integration of equation of motion using the Runge-Kutta method (others also available)



- Curved path broken into linear *chord* segments to minimize the *sagitta* (maximum chord-trajectory distance)
- Chords used to interrogate navigator on whether the track has crossed a volume boundary
- *miss distance* parameter used to tune volume intersection accuracy

G4 supports user defined, uniform, and non-uniform (static or time dependent) magnetic fields

Specialized Geometry Library.

- Backward compatibility with ROOT and Geant4
- Continue the already started AIDA USolids project
- Numerical simulation have special requirements on numerical stability (double vs float, no leaks, we transport things ***across*** boundaries often not the case in 3D graphics engines)
- HEP use specialized volumes not existing in external packages (***polycone***) and we can put a lot of domain specific knowledge to accelerate things
- ***Single*** solution for CPU and GPU
- Rely on special functions "***safety***" which might not exist in classical 3D rendering engines (which concentrate on hit detection)
- Need an exact volume representation (and not a triangle approximation).
- Different scale than most 3D graphics engines (that often have far fewer things to treat)