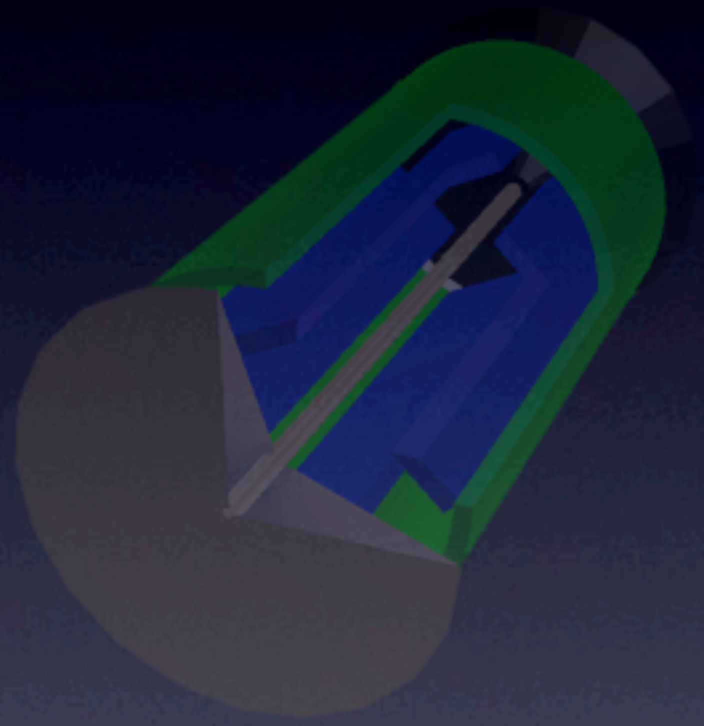
16th International workshop on Advanced Computing and Analysis Techniques in physics research (ACAT); Prague 1.9.-5.9.2014



Towards a high performance detector geometry library on CPU and GPU

for particle-detector simulation

Sandro Wenzel / CERN-PH-SFT

In collaboration with: John Apostolakis (CERN), Marilena Bandieramonte (University of Catania, IT), Georgios Bitzes (CERN), Rene Brun (CERN), Philippe Canal (Fermilab), Federico Carminati (CERN), Gabriele Cosmo (CERN), Johannes Christof De Fine Licht (CERN), Laurent Duhem (Intel), Daniel Elvira (Fermilab), Andrei Gheata (CERN), Soon Yung Jun (Fermilab), Guilherme Lima (Fermilab), Tatiana Nikitina (CERN), Mihaly Novak (CERN), Raman Sehgal (Bhabha Atomic Research Centre), Oksana Shadura (CERN)

Part I ("Geometry in simulation")

- geometry in simulation; typical tasks
- ROOT, Geant4, USolids packages
- the **need to go beyond** current implementations

Part II: Introducing "VecGeom": towards a vectorized and templated geometry library for detector simulation

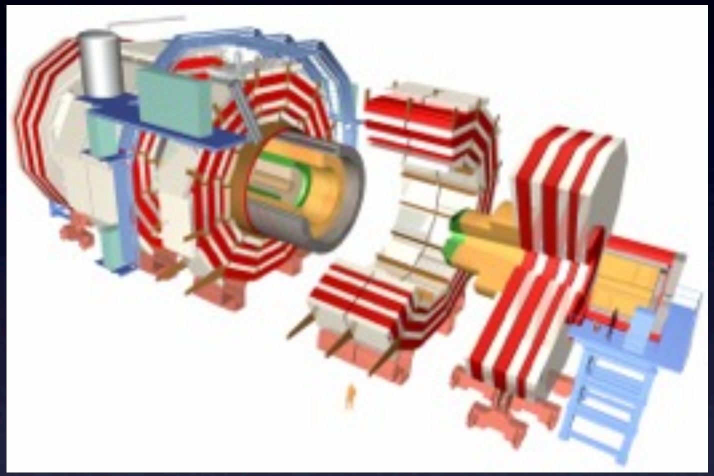
- overview
- walk-through of new features; improvements
- performance examples

Part III: Some words on generic programming approach

- shared scalar/vector (CUDA) kernels

Geometry in simulation

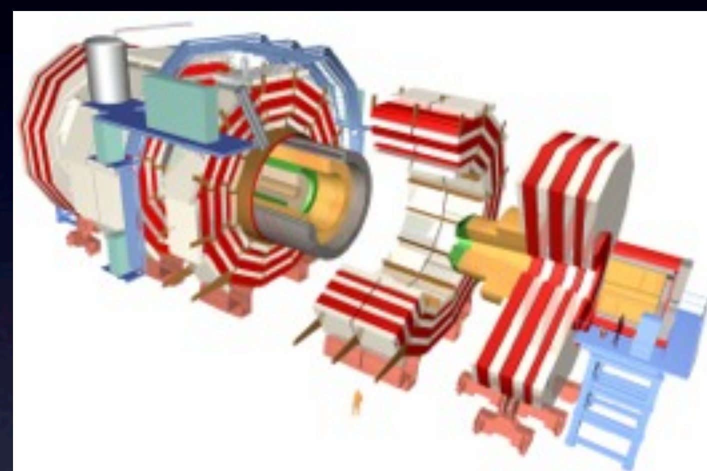
- geometrical model or description of detectors integral part of "particle-detector" simulation, reconstruction etc.;
- detectors usually are modeled as a hierarchy of **shape primitives** containing other shape primitives



CMS detector: boxes, trapezoids, tubes, cones, polycones, ...

Geometry in simulation

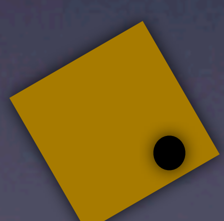
- geometrical model or description of detectors integral part of "particle-detector" simulation, reconstruction etc.;
- detectors usually are modeled as a hierarchy of **shape primitives** containing other shape primitives



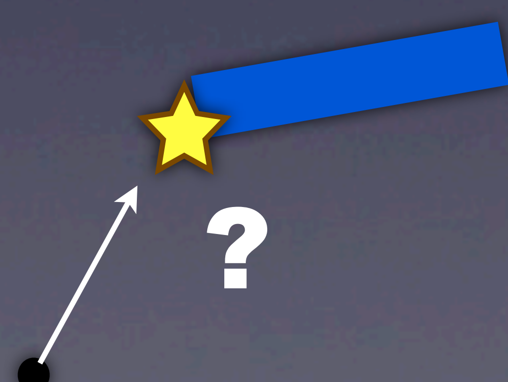
CMS detector: boxes, trapezoids, tubes, cones, polycones, ...

- A geometry library offers an API to ...

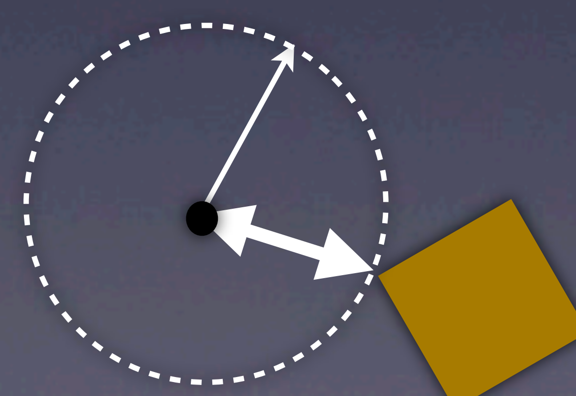
in or out?



collision detection and distance to enter object



minimal(safe) distance to object



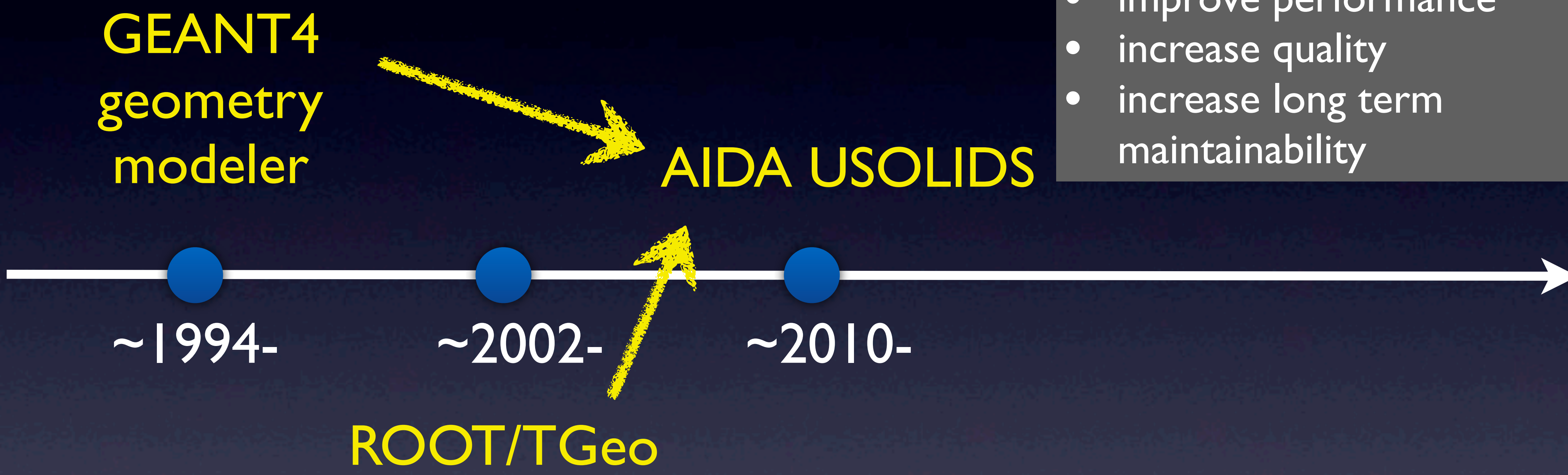
distance to leave object

Geometry/Solid - Packages

very widespread in HEP, medical physics, ...

EU/AIDA funded effort to merge the libraries (**on shape level**):

- merge code base
- pick best implementation
- improve performance
- increase quality
- increase long term maintainability



experiments using virtual Monte Carlo framework (ALICE, FAIR) + ...

Geometry/Solid - Packages

very widespread in HEP, medical physics, ...

GEANT4 geometry modeler

AIDA USOLIDS

EU/AIDA funded effort to merge the libraries (**on shape level**):

- merge code base
- pick best implementation
- improve performance
- increase quality
- increase long term maintainability

~1994-

~2002-

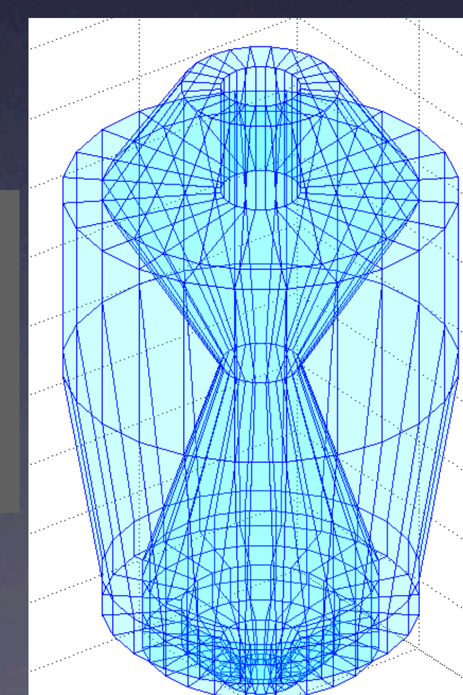
~2010-

ROOT/TGeo

experiments using virtual Monte Carlo framework (ALICE, FAIR) + ...

example for improvement:

- new polycone (**~8 faster than Geant4/Root**)
- multi-union, tesselated solids



New needs/beyond USolids

- USolids made a big step forward improving shape primitive code
- experiments are able to see the benefits now; USolids can be used in Geant4 simulations today!

New needs/beyond USolids

- USolids made a big step forward improving shape primitive code
- experiments are able to see the benefits now; USolids can be used in Geant4 simulations today!

but: **new needs/requirements** not yet addressed by current implementations

- no use of **SIMD vectorization**
- no interfaces to **process many particles** at once
- no use of **HPC features of C++ ("templates")** which could further improve performance
- (no library support **on GPU**)

→ **goals**

Targeting vectorization

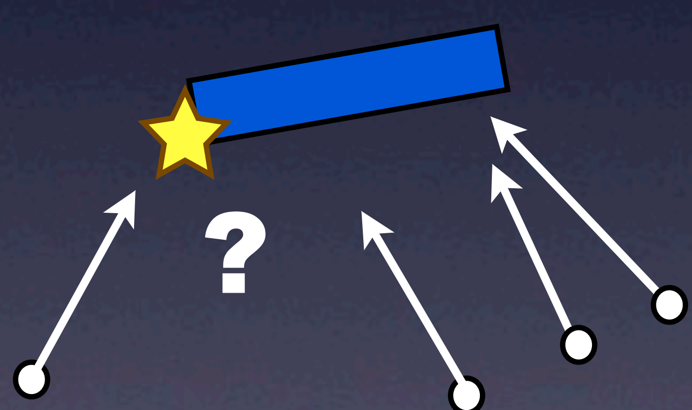
- CPU vector instructions become ever more important; vector registers becoming wider
- these instructions have to be used to efficiently use compute architecture; need to have "vector" data on which we apply the same tasks

Targeting vectorization

- CPU vector instructions become ever more important; vector registers becoming wider
- these instructions have to be used to efficiently use compute architecture; need to have “vector” data on which we apply the same tasks

outer vectorization

“parallel” collision detection



primary target of this investigation; relevant for Geant-V prototype

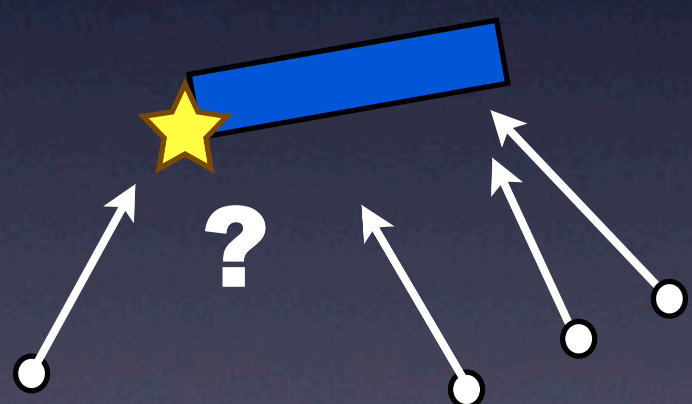
makes “future” code faster

Targeting vectorization

- CPU vector instructions become ever more important; vector registers becoming wider
- these instructions have to be used to efficiently use compute architecture; need to have "vector" data on which we apply the same tasks

outer vectorization

"parallel" collision detection

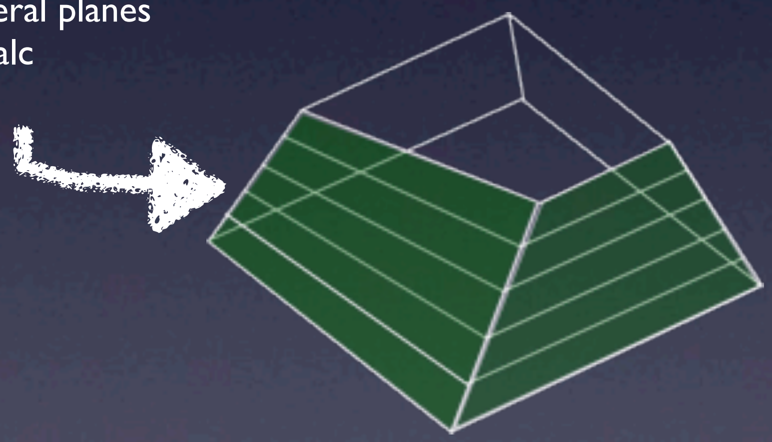


primary target of this investigation; relevant for Geant-V prototype

makes "future" code faster

internal vectorization

internal loop over lateral planes for distance calc



vectorization of inner loops; not common in shape code; but feasible for a couple of shapes (trapezoid)

beneficial for current simulations

Software Challenges implied by goals

- How do we achieve **reliable** vectorization on CPU?
  - easy: we use a specialized C++ vectorization library (Vc!)
  - code in terms of "vector types" instead of scalar types: double vs Vc::double_v

Software Challenges implied by goals

- How do we achieve **reliable** vectorization on CPU?
  - easy: we use a specialized C++ vectorization library (Vc!)
  - code in terms of "vector types" instead of scalar types: double vs Vc::double_v

- How do we cope with the multiplication of interfaces (scalar API, many-particle API, CUDA) ... ?

Box
x,y,z
double DistanceTo(1 particle)
double* DistanceTo(many particles)
bool Contains (1 particle)
bool* Contains (many particles)
double SafeDistance(1 particle)
double* SafeDistance(many particles)
double DistanceToOut (1 particle)
double* DistanceToOut(many particles)

At least ~5 new functions per solid

~20 primitive solid

~100 new functions to maintain (possibly more with CUDA ...)

- In particular: How do we keep the code base small while maintaining good speed + long term maintainability ?

Introducing “VecGeom”

started as prototype project with tight focus to study benefit of vectorization for multi-particle API in geometry

- primarily motivated from GeantV-prototype

now **evolved** to project that **addresses all goals** and challenges presented before

Introducing "VecGeom"

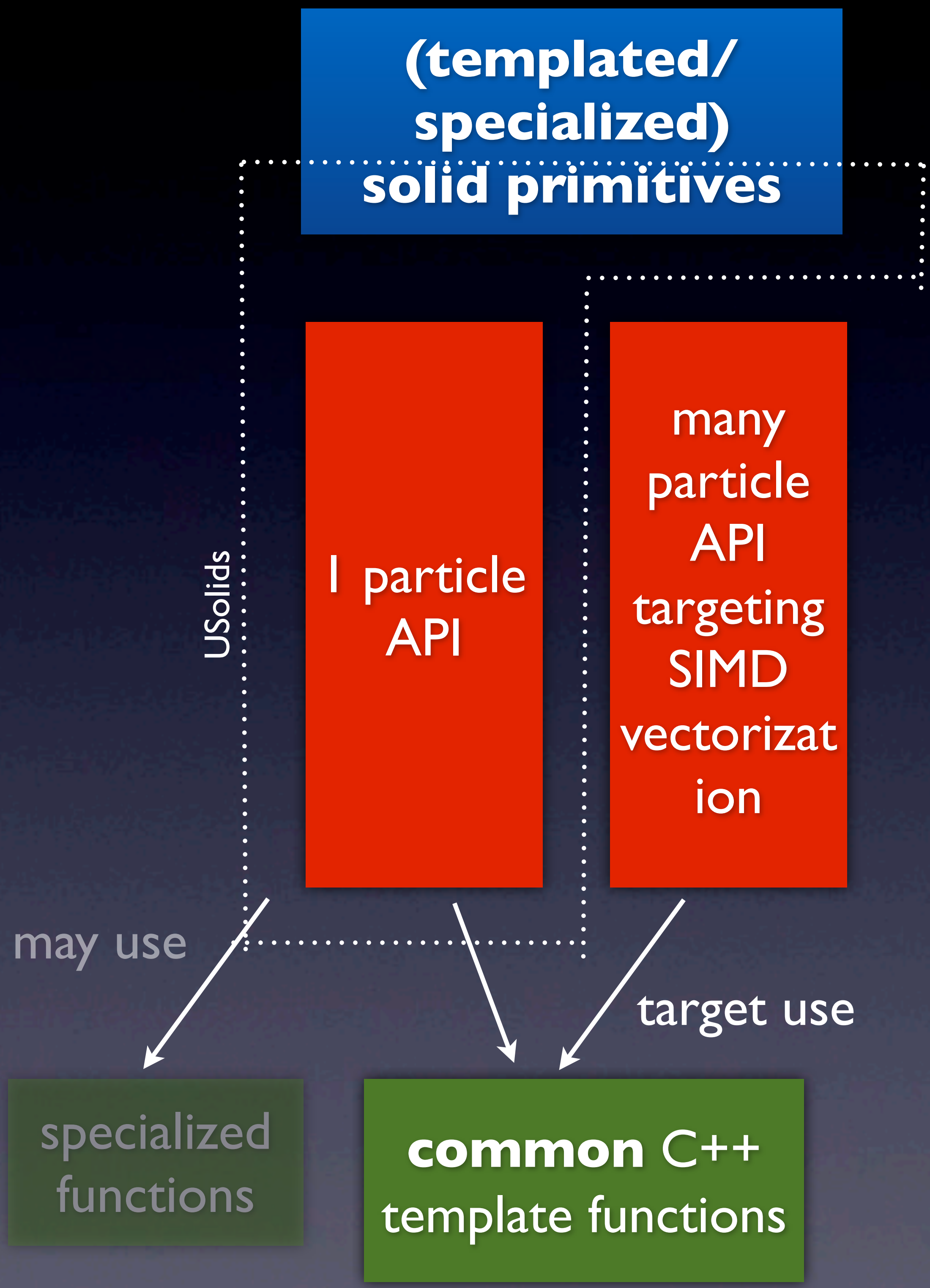
- already developed back-to-back with USolids; sharing a repositions; same interfaces
- solid classes should become natural evolution of USolids library

started as prototype project with tight focus to study benefit of vectorization for multi-particle API in geometry

- primarily motivated from GeantV-prototype

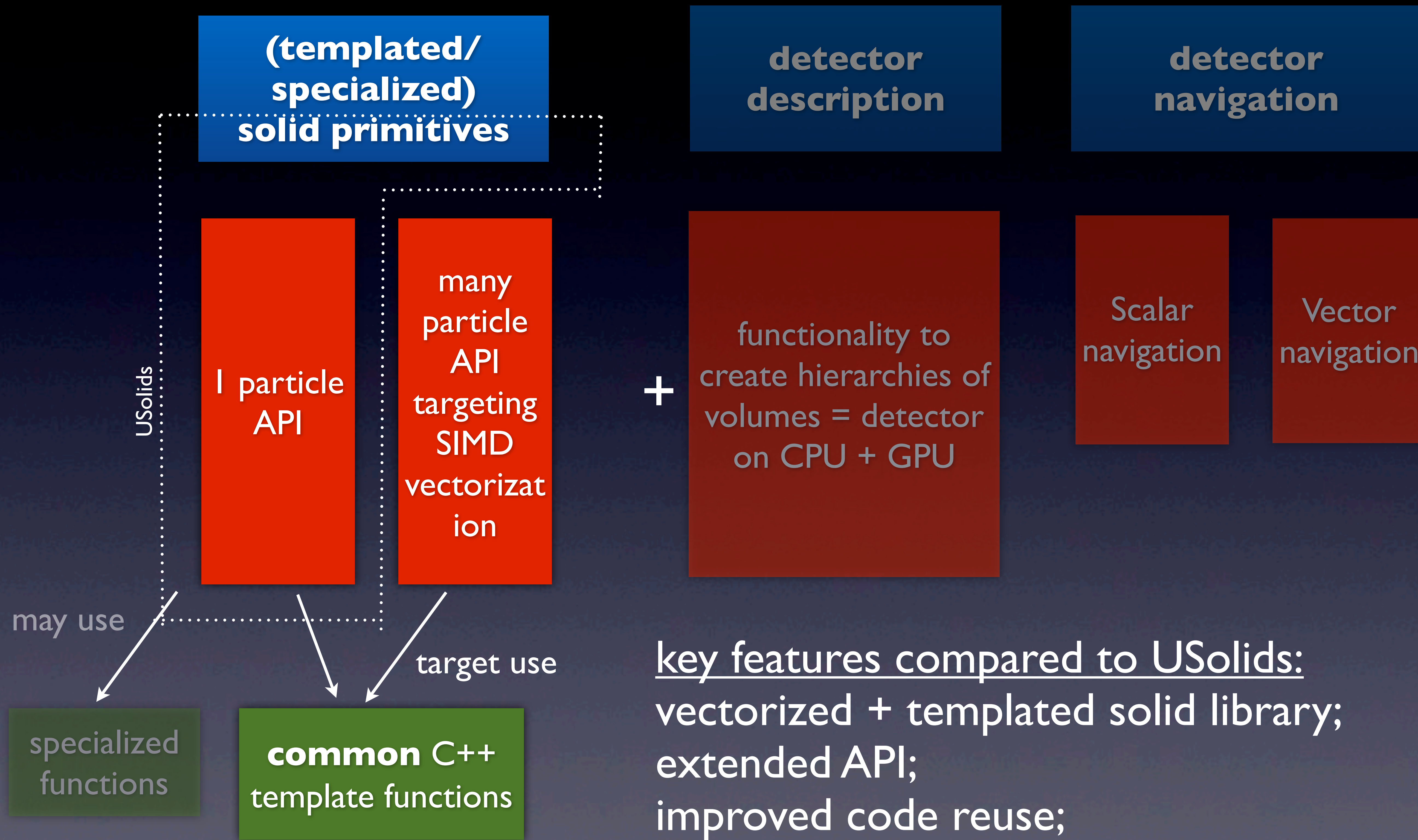
now **evolved** to project that **addresses all goals** and challenges presented before

Overview of “VecGeom”



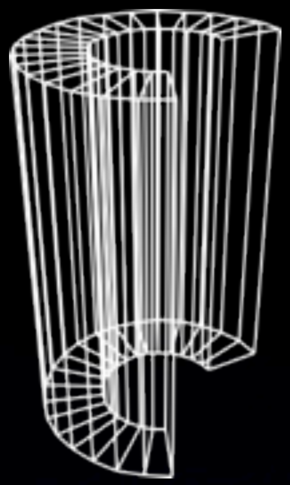
<u>key features compared to USolids:</u>
vectorized + templated solid library;
extended API;
improved code reuse;
further improved algorithms

Overview of "VecGeom"



<u>key features compared to USolids:</u>
vectorized + templated solid library;
extended API;
improved code reuse;
further improved algorithms

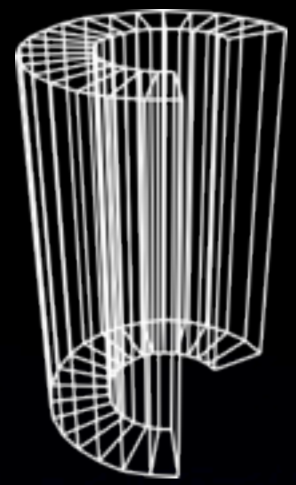
Performance case study: the tube segment



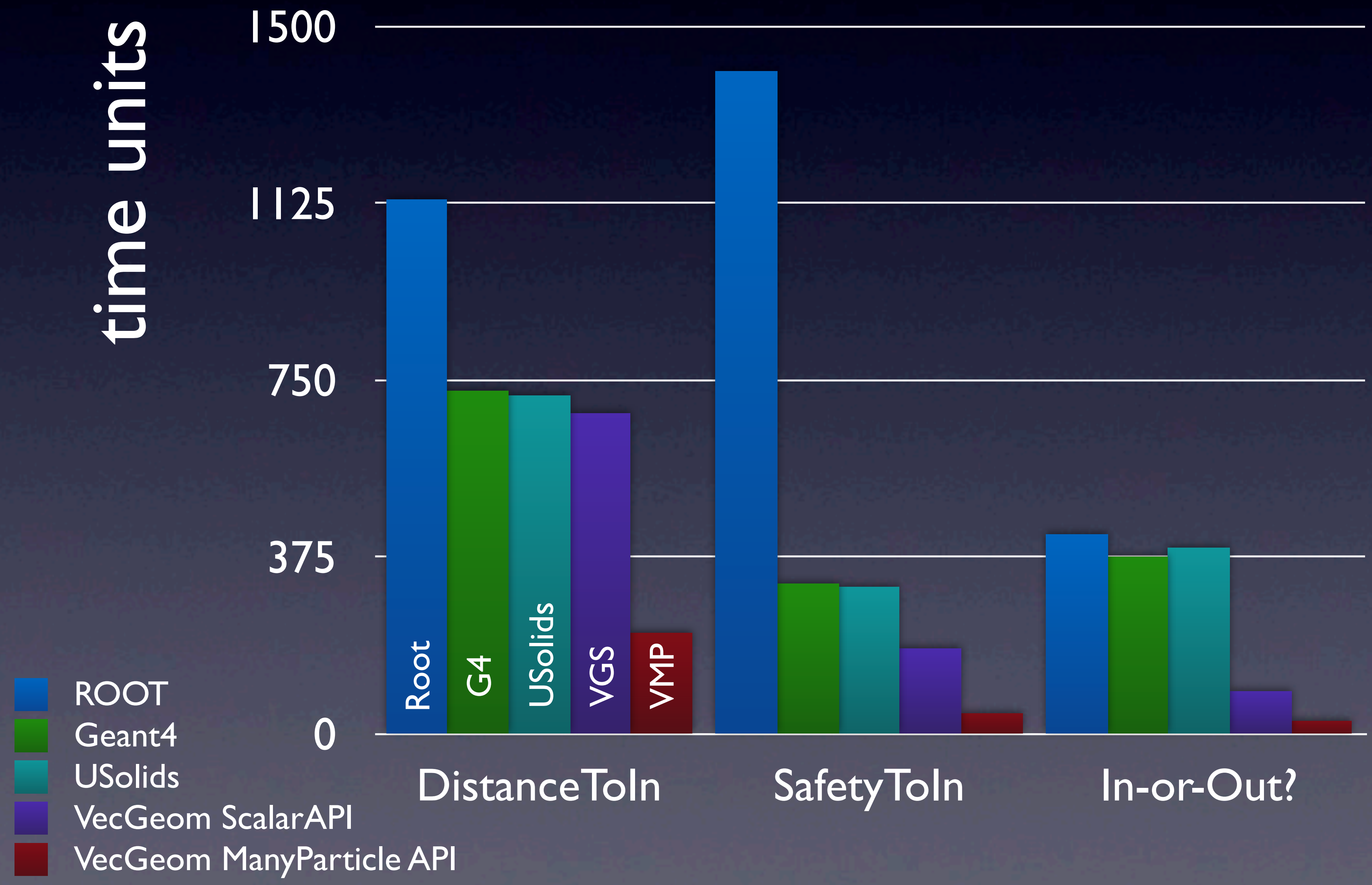
- tube segment one of the most used/important shape primitives
- also integral part of complex shapes: polycone
- **extremely important to be as fast as we can**

gcc 4.7; -O3 -funroll-loops -mavx; no FMA; Geant4 10 (Release); Root 5.34.18 (Release)

Performance case study: the tube segment

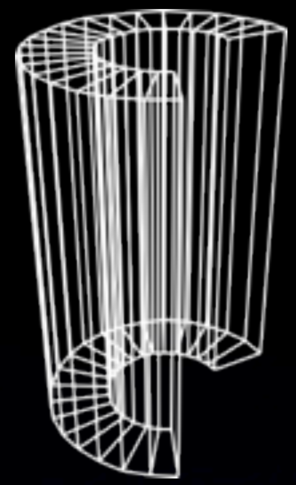


- tube segment one of the most used/important shape primitives
- also integral part of complex shapes: polycone
- **extremely important to be as fast as we can**

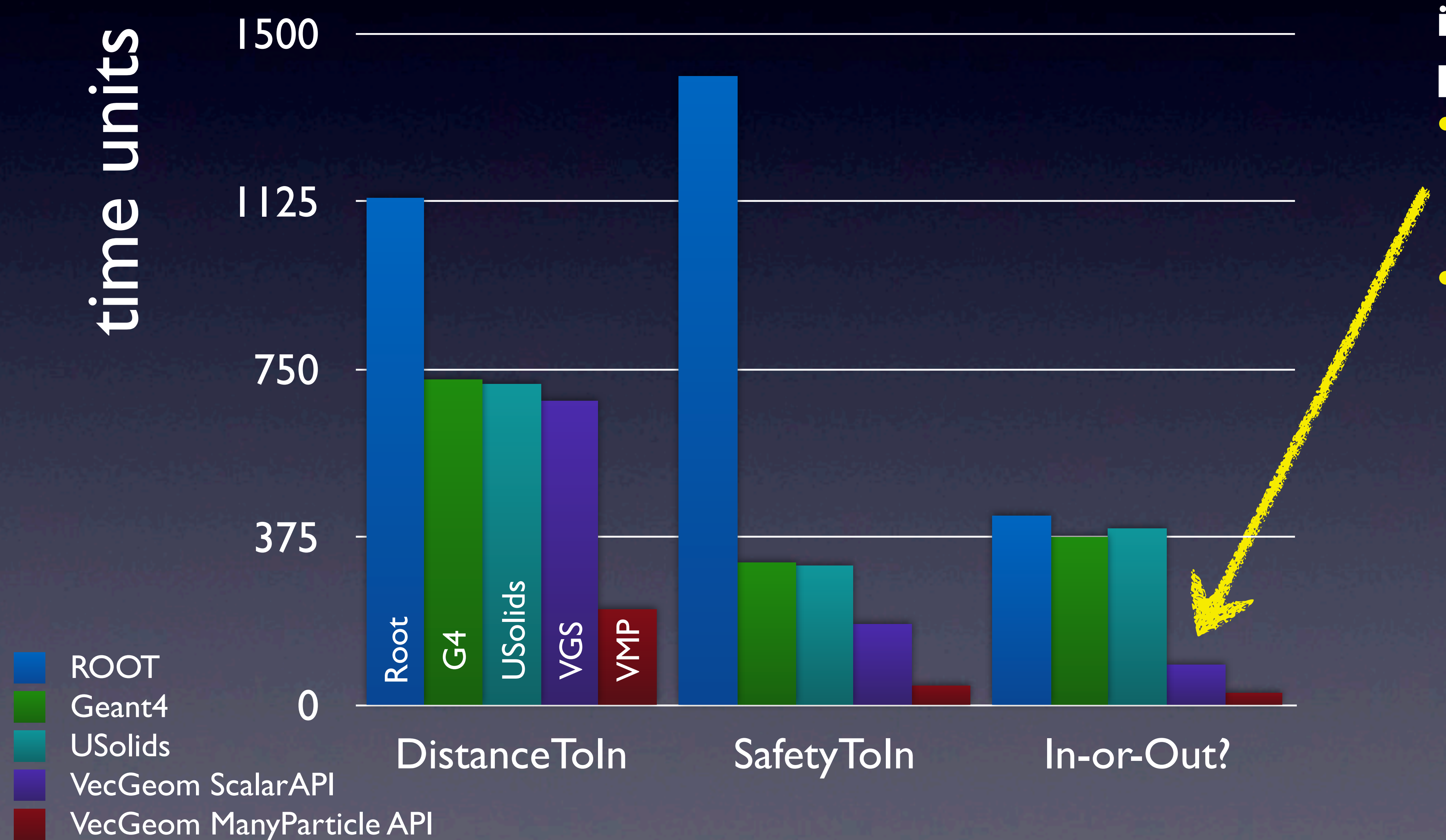


gcc 4.7; -O3 -funroll-loops -mavx; no FMA; Geant4 10 (Release); Root 5.34.18 (Release)

Performance case study: the tube segment



- tube segment one of the most used/important shape primitives
- also integral part of complex shapes: polycone
- **extremely important to be as fast as we can**

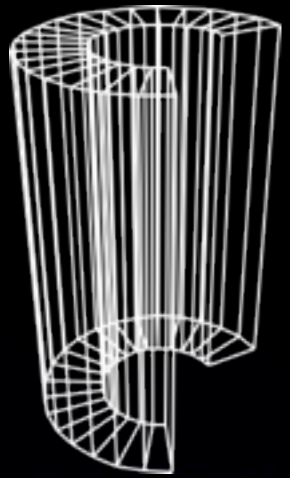


improved scalar performance

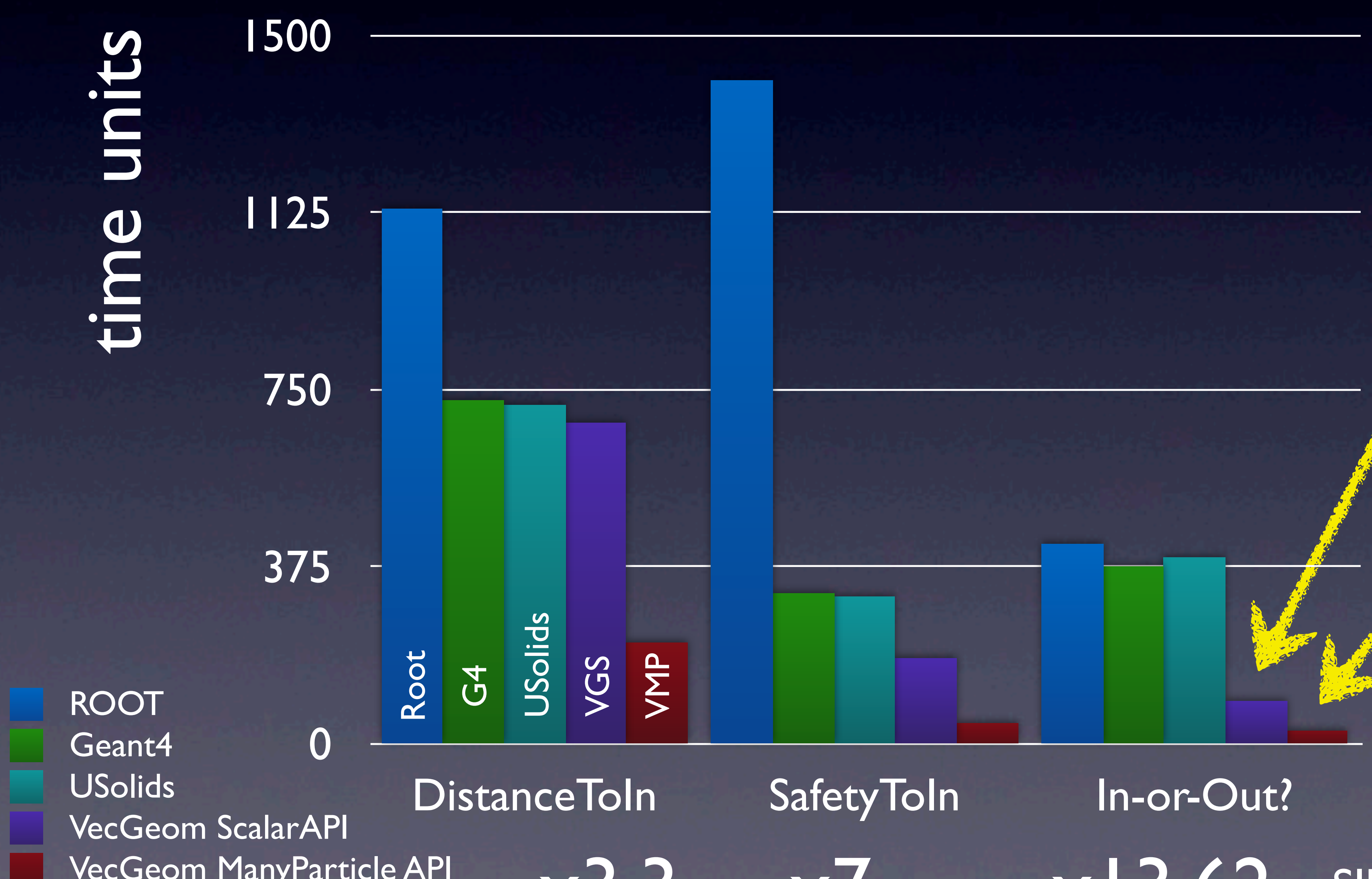
- improved algorithms (avoid atan2)
- template shape specialization

gcc 4.7; -O3 -funroll-loops -mavx; no FMA; Geant4 10 (Release); Root 5.34.18 (Release)

Performance case study: the tube segment



- tube segment one of the most used/important shape primitives
- also integral part of complex shapes: polycone
- **extremely important to be as fast as we can**



improved scalar performance

- improved algorithms (avoid atan2)
- template shape specialization

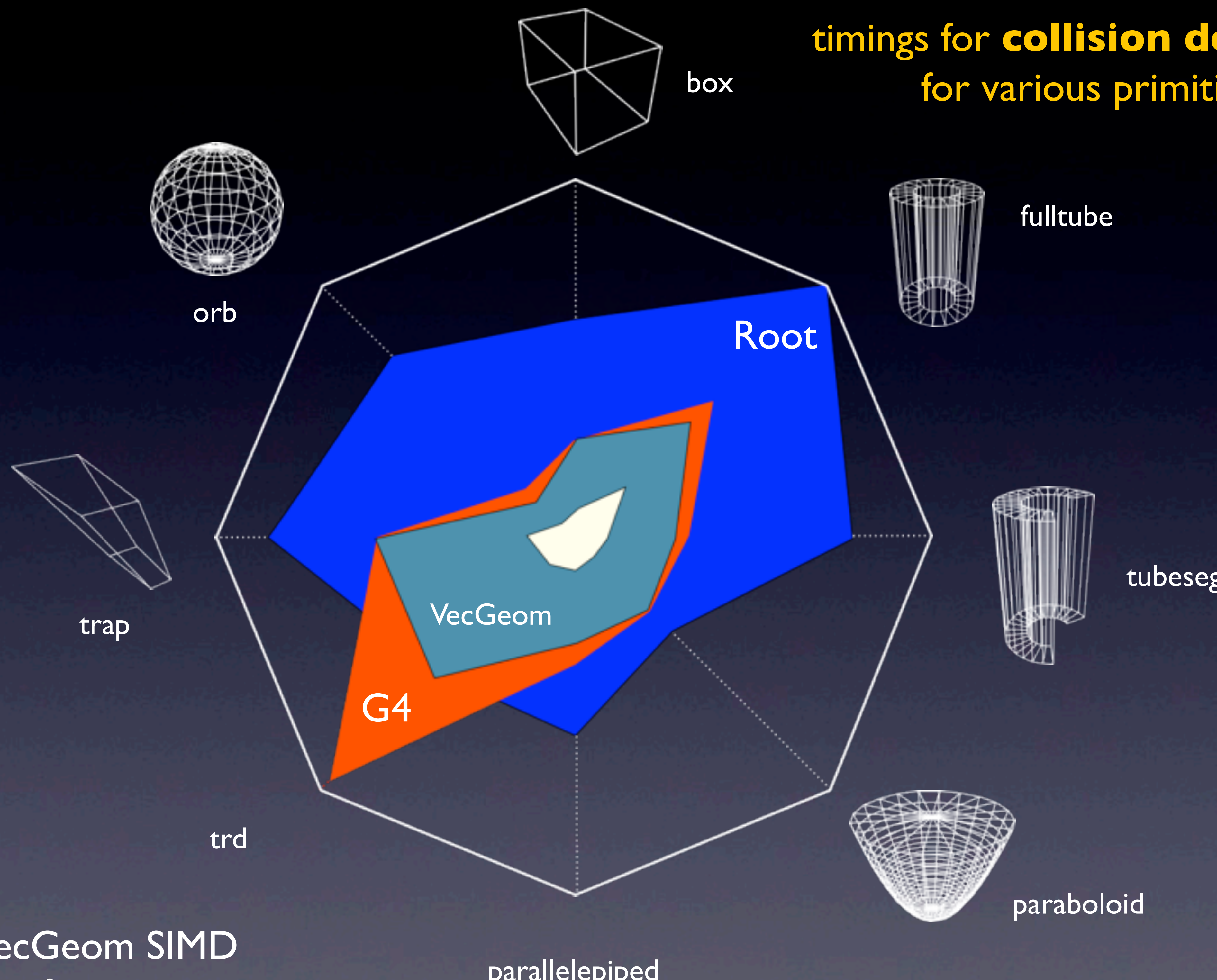
excellent SIMD vector performance

x3.3 x7 x13.62 SIMD speedup cmp to USolids

gcc 4.7; -O3 -funroll-loops -mavx; no FMA; Geant4 10 (Release); Root 5.34.18 (Release)

Solid/shape implementation status; performance

timings for **collision detection**
for various primitives

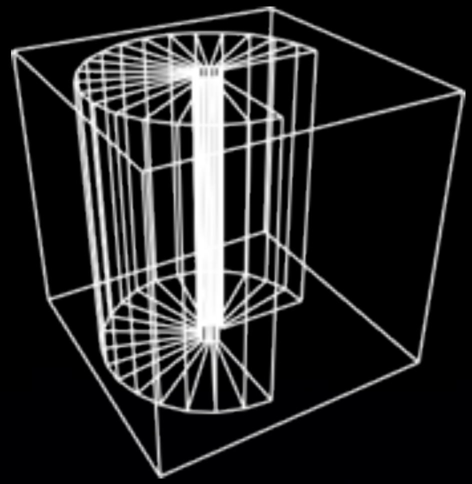



= VecGeom SIMD performance

very generic performance pattern
for all functions

slide under construction!

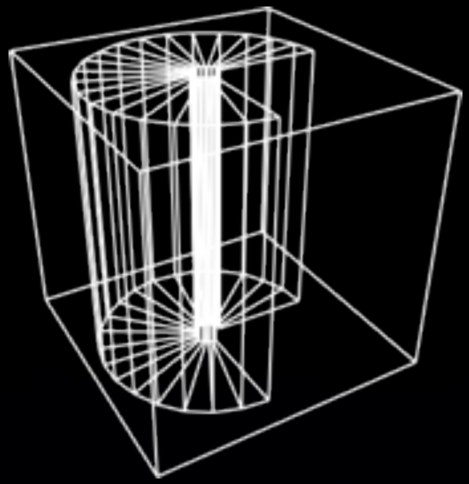
going complex...



- boolean solids are an important element in detector construction (subtraction solid, union solid)
- Geant4+Root offer construction of such objects based on a solid base class and virtual functions

```
SubtractionSolid( AbstractShape * left, AbstractShape * right );
```

going complex...



- boolean solids are an important element in detector construction (subtraction solid, union solid)
- Geant4+Root offer construction of such objects based on a solid base class and virtual functions

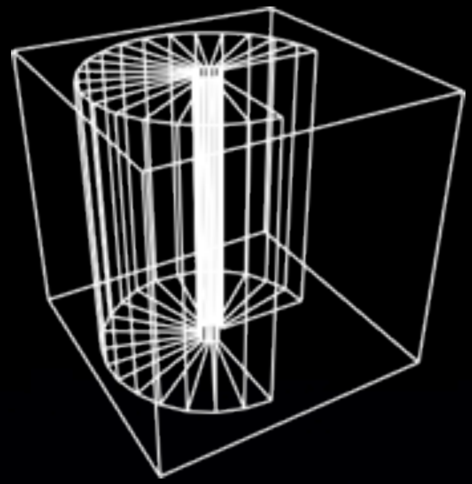
```
SubtractionSolid( AbstractShape * left, AbstractShape * right );
```

- now offer advanced way to combine shapes (ala stl)

```
template <typename LeftSolid, typename RightSolid>
class TSubtractionSolid
{
  TSubtractionSolid( LeftSolid * left, RightSolid * right );
};
```

- compiler can produce optimized code for any combination of primitive shapes (“template-shape specialization”)
- no virtual function calls
- vectorization comes from reusing vector functions of components

going complex (condt)



- performance example for a subtraction solid “box minus tubesegment” (in CMS detector)

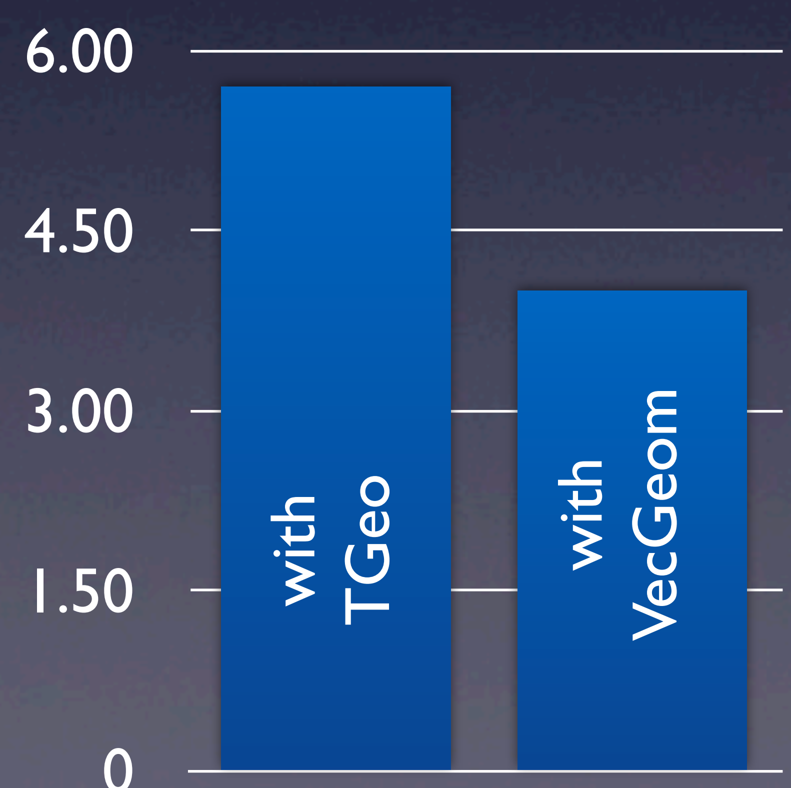
	In-or-Out?	DistanceToOut	SafetyToOut
SIMD/ROOT speedup:	**8x**	**4.6x**	**31x**
SIMD/Geant4 speedup:	**6.6x**	**3.2x**	**17x**

TODO: quantify gain from templates

gcc 4.7; -O3 -funroll-loops -mavx; no FMA; Geant4 10 (Release); Root 5.34.18 (Release)

VecGeom in action ...

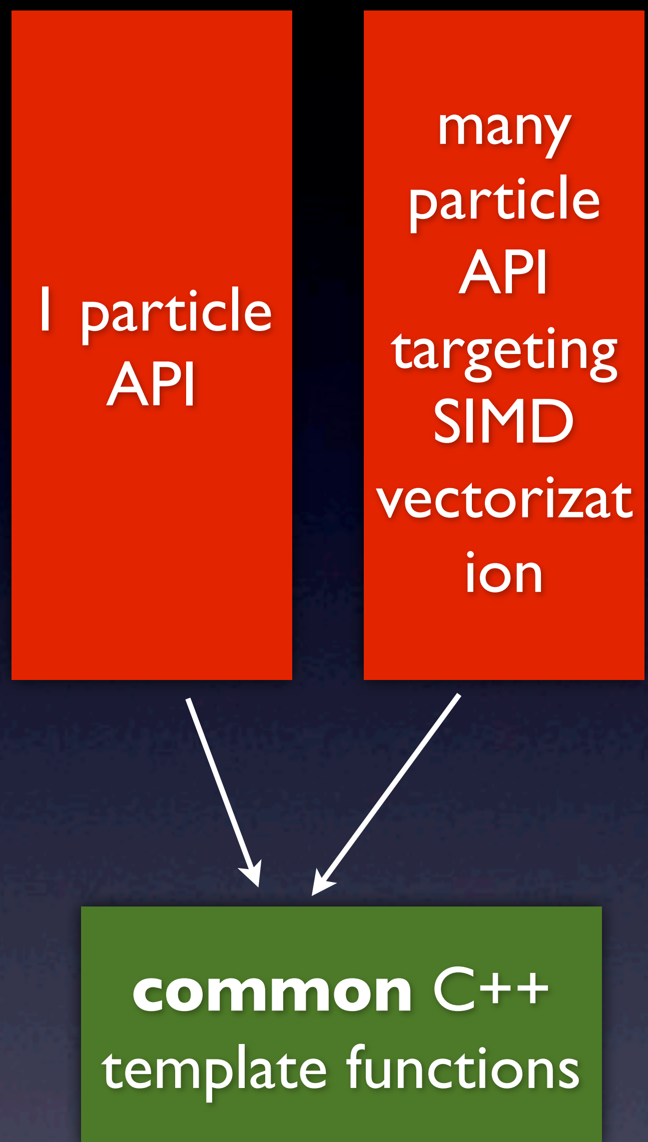
- VecGeom is functionally complete. We can construct detectors and navigate particles on CPU + GPU
- Geant-Vector prototype can run complete first particle-detector simulations using VecGeom
- have the ability to switch between ROOT/TGeo and VecGeom with consistent results
- measured a **total simulation runtime improvement** of **40%** going from TGeo to VecGeom for a simple box-like detector (ExN03 from Geant4)
- should be able to simulate with CMS detector soonish ....



Part III: Some words on programming approch

achieving shared scalar / vector code

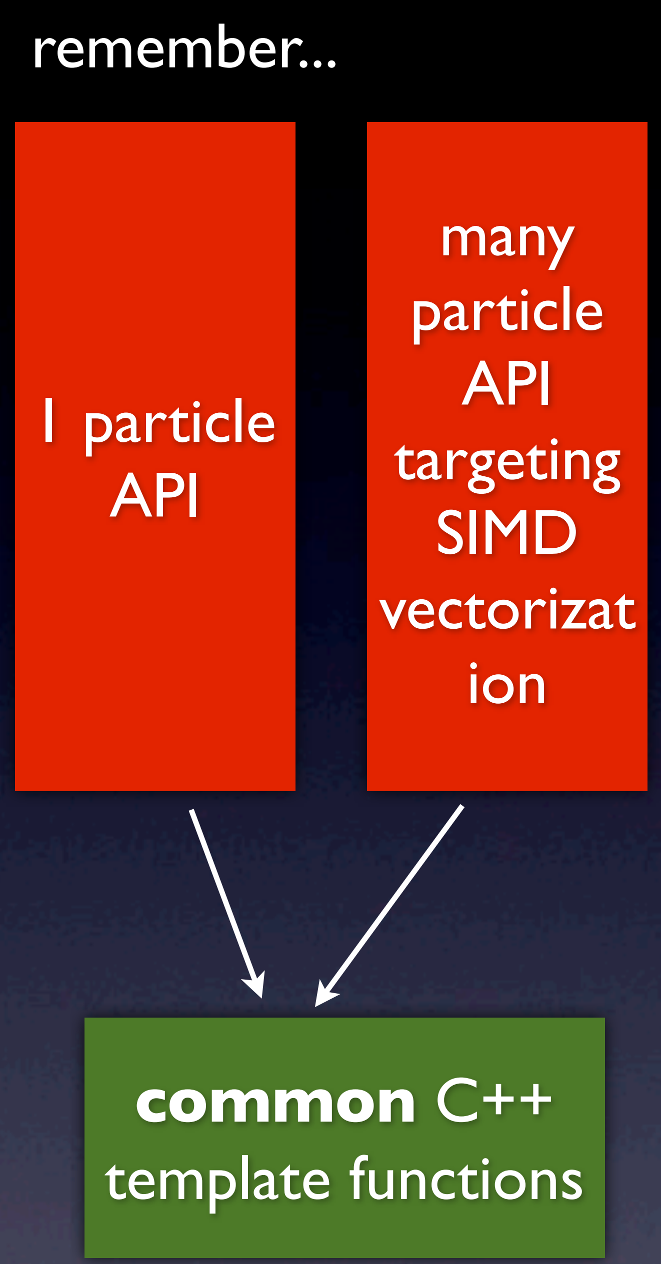
remember...




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

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

achieving shared scalar / vector code



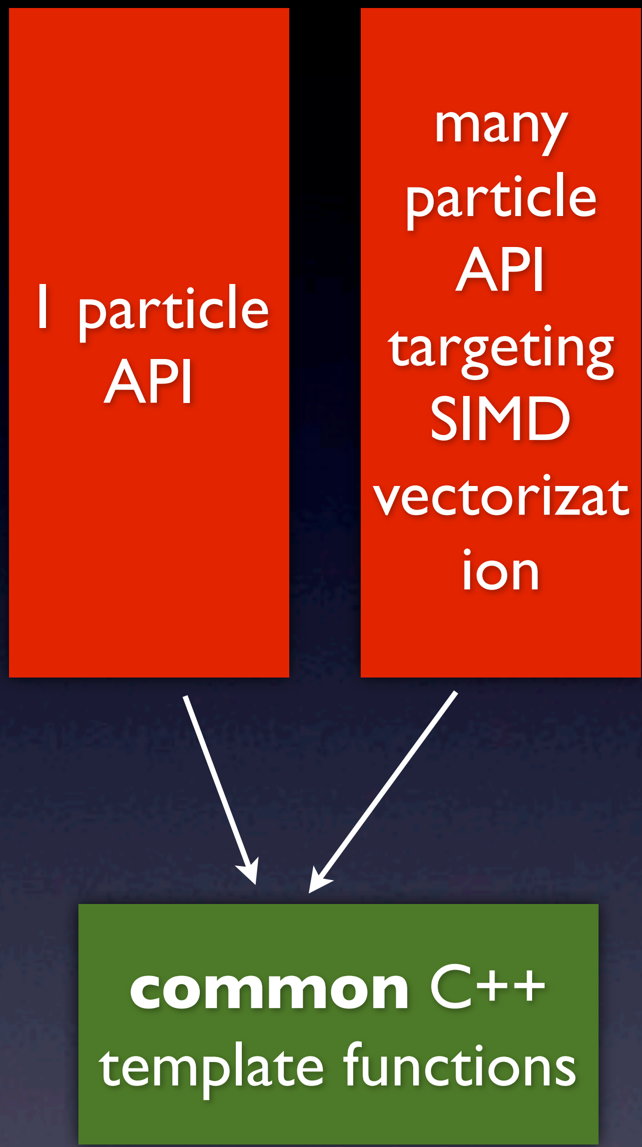

```
double distance( double );
```

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

```
template<class Backend>
Backend::double_t
commonFunction( Backend::double_t const & input )
{
    // complicating code implementing this function
    // using only abstract types that Backend provides
}
```

achieving shared scalar / vector code

remember...




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

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

```
template<class Backend>
Backend::double_t
commonFunction( Backend::double_t const & input )
{
    // complicating code implementing this function
    // using only abstract types that Backend provides
}
```

- “Backend” is a struct encapsulating standard types/ properties for “scalar, vector, CUDA” programming; makes information injection into template function easy

```
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::double_m bool_t;
    static const bool IsScalar=false;
    static const bool IsSIMD=true;
};
```

attention: this is not valid C++ code; need an additional “typename” before Backend

shared scalar-vector code: example

Point
fX, fY, fZ
double Distance(Vector3D<double> …) double_v Distance(Vector3D<double_v> …)

- toy example: calculate distance of particles to a Point represented by class Point with members (fX,fY,fZ)
- Point class offers 2 “distance” interfaces inlining same template function

```
double Point::Distance(Vector3D const& a)
{
    return DistanceKernel<ScalarBackend>( a );
}
```

```
Vc::double_v Point::Distance(Vector3D<Vc::double_v>
const& a)
{
    return DistanceKernel<VectorBackend>( a );
}
```

attention: this is not valid C++ code; need an additional “typename” before Backend

shared scalar-vector code: example

Point
fX, fY, fZ
double Distance(Vector3D<double> …)
double_v Distance(Vector3D<double_v> …)

- toy example: calculate distance of particles to a Point represented by class Point with members (fX,fY,fZ)
- Point class offers 2 "distance" interfaces inlining same template function

```
double Point::Distance(Vector3D const& a)
{
    return DistanceKernel<ScalarBackend>( a );
}
```

```
Vc::double_v Point::Distance(Vector3D<Vc::double_v>
const& a)
{
    return DistanceKernel<VectorBackend>( a );
}
```

produces solid SIMD code

```
template<typename Backend>
inline __attribute__((always_inline))
Backend::double Point::DistanceKernel( Vector3D<Backend::double_t> const & point )
{
  Backend::double_t xp = fX - point.x();
  Backend::double_t yp = fY - point.y();
  Backend::double_t zp = fZ - point.z();
  // might have some Backend specific code
  if( Backend::IsScalar )
  {
      // we are able to diverge the code paths between different backends
  }
  return Sqrt(xp*xp + yp*yp + zp*zp);
}
```

attention: this is not valid C++ code; need an additional "typename" before Backend

Summary

- VecGeom is a detector geometry library which:
  - is **fast**
  - offers **vectorized** multi-particle treatment
  - follows **generic programming approach** to reduce code size
  - (supports CUDA and GPU)

- Now much more confident to tackle vectorization of physics routines

Backup

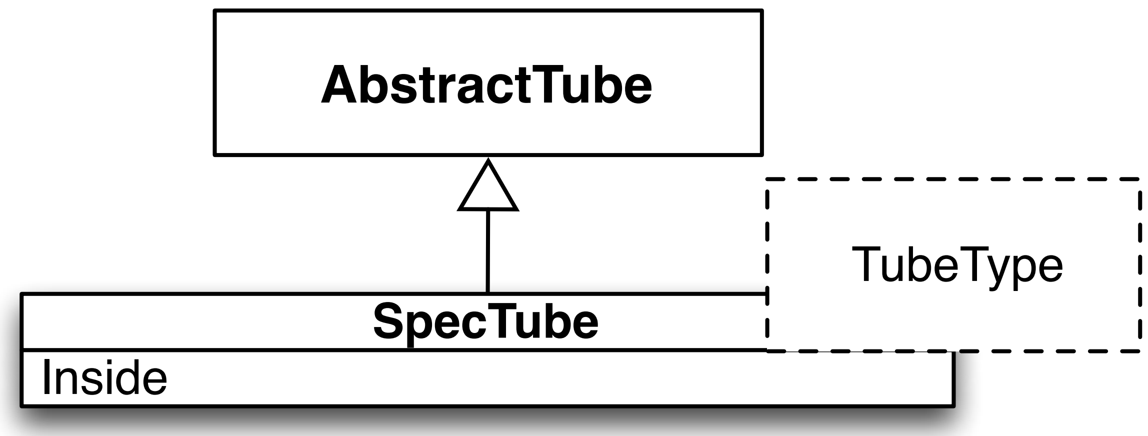
- show generic trap developments (internal vectorization)

Backup

* slides on tube template shape specialization

common code - many realizations

```
template<typename TubeType>
class
SpecTube{
 //  ...
 bool Inside( Vector3D const & ) const;
 //...
};
```



* sharing code between classes with compile-time branches (scalar toy example)

```
template<typename TubeType>
bool SpecTube<TubeType>::Inside( Vector3D const & x) const
{
   // checkContainedZ
   if( std::abs(x.z) > fdZ ) return false;

   // checkContainmentR
   double r2 = x.x*x.x + x.y*x.y;
   if( r2 > fRmaxSqr ) return false;

   if ( TubeType::NeedsRminTreatment )
   {
      if( r2 < fRminSqr ) return false;
   }

   if ( TubeType::NeedsPhiTreatment )
   {
      // some code
   }
   return true;
}
```

we can express **“static” ifs** as **compile-time if statements (e.g. via const properties of TubeType)**

gets optimized away if a certain TubeType does not need this code

compiler creates different binary code for different TubeTypes