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

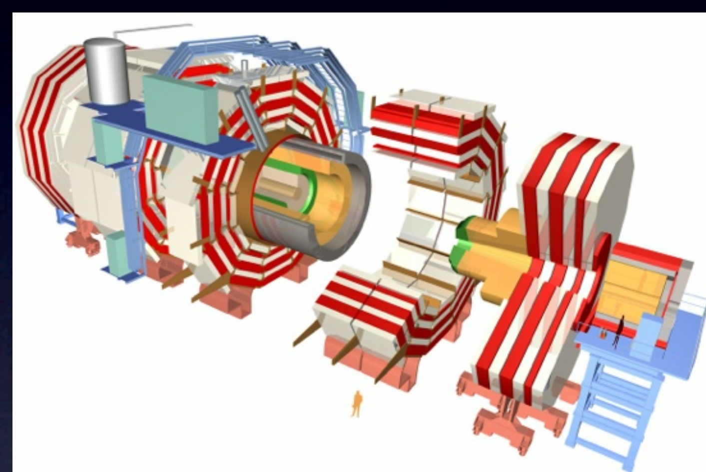
# Towards a high performance geometry library for particle-detector simulation


## Sandro Wenzel / CERN-PH-SFT

In collaboration with: J. Apostolakis (CERN), M. Bandieramonte (University of Catania, IT), G. Bitzes (CERN), R. Brun (CERN), P. Canal (Fermilab), F. Carminati (CERN), G. Cosmo (CERN), J. De Fine Licht (CERN), L. Duhem (Intel), D. Elvira (Fermilab), A. Gheata (CERN), S. Yung Jun (Fermilab), G. Lima (Fermilab), T. Nikitina (CERN), M. Novak (CERN), R. Sehgal (Bhabha Atomic Research Centre), O. Shadura (CERN)

# Geometry in simulation

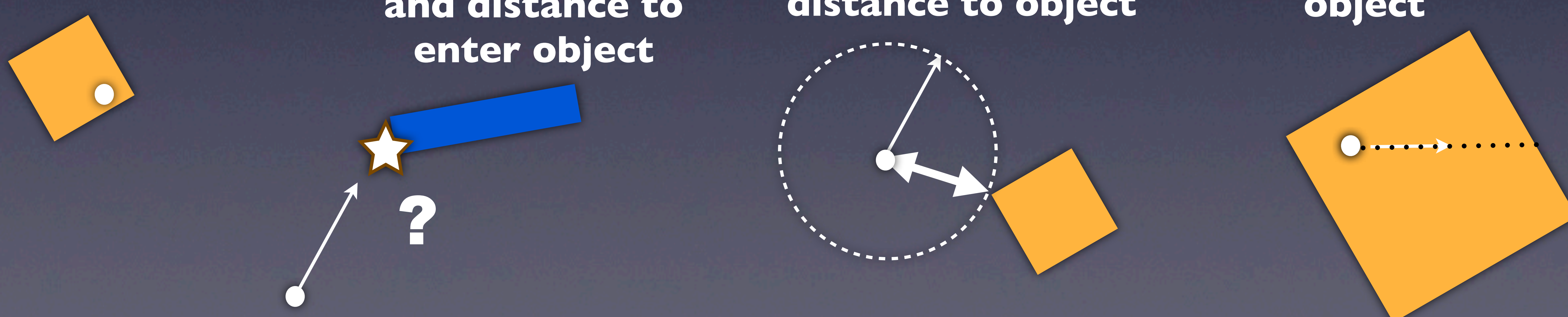
- geometry tasks are a major consumer of CPU cycles in detector simulation
- most of time spent in interaction with **shape primitives** which make up a detector



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

- For shape primitives, 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**



# Outline

## Part I: Geometry in simulation

- review of ROOT, Geant4, USolids packages
- the **need to go beyond** current implementations
- software challenges

## Part II: Introducing "VecGeom"

- overview
- performance and status update

## Part III: Some details on generic programming approach

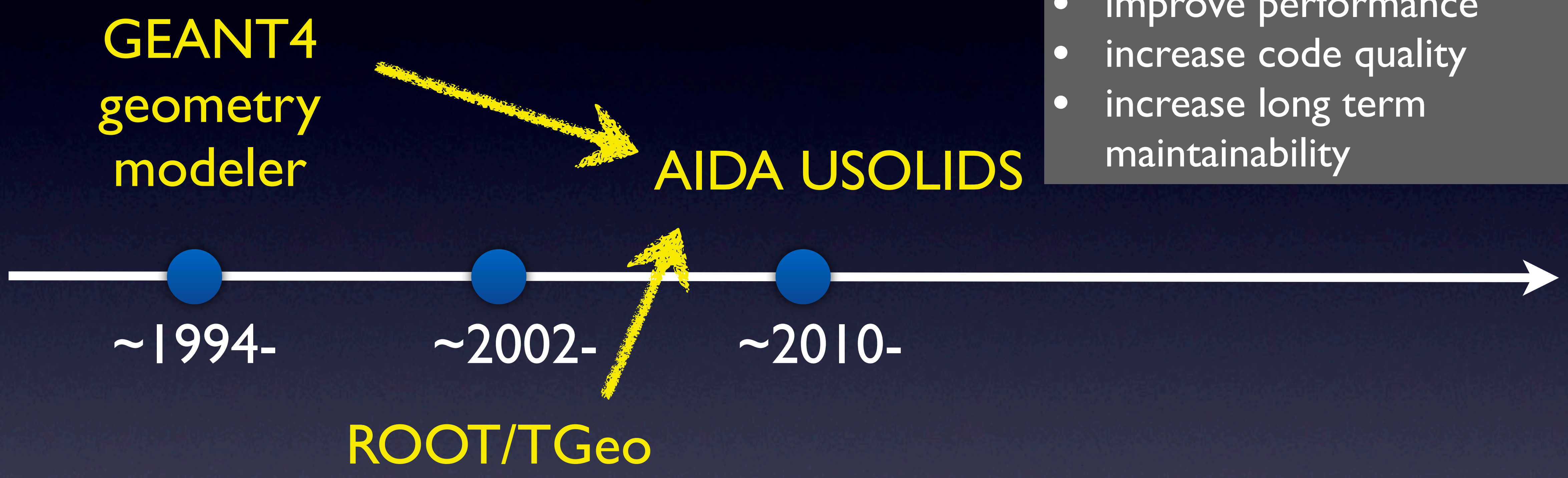
- shared scalar/vector (CUDA) kernels

# 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 code quality
- increase long term maintainability




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

# 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 code quality
- increase long term maintainability

GEANT4 geometry modeler

AIDA USOLIDS

~1994-

~2002-

~2010-

ROOT/TGeo

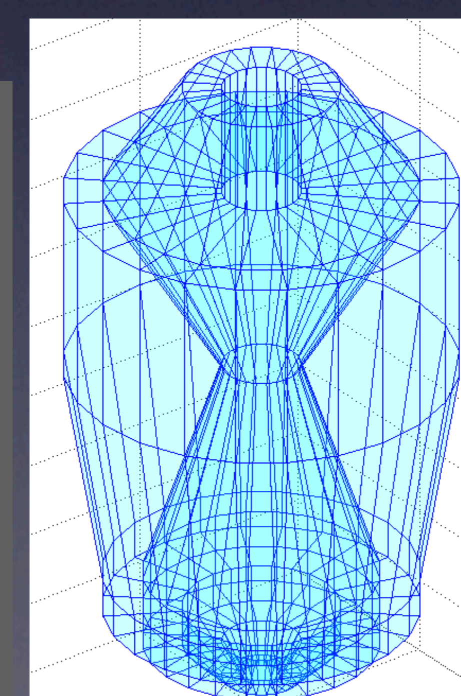
experiments using virtual Monte Carlo framework (ALICE, FAIR) + ...

**improvements:**

- new polycone (**~8x** faster than Geant4/Root)

**completely new features:**

- 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! **PLEASE TRY !!**

# 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! **PLEASE TRY !!**

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

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

**goals**

# Targeting vectorization

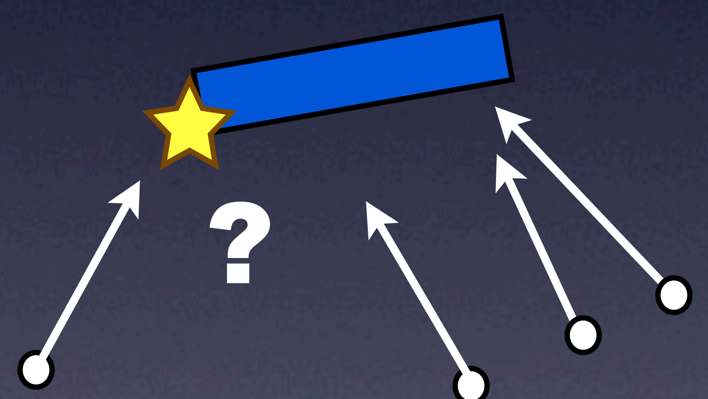
- vector instructions getting more important; vector registers becoming wider
- these instructions have to be used to efficiently use compute architecture; need to have “vector” data on which we apply the same tasks

# Targeting vectorization

- vector instructions getting 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



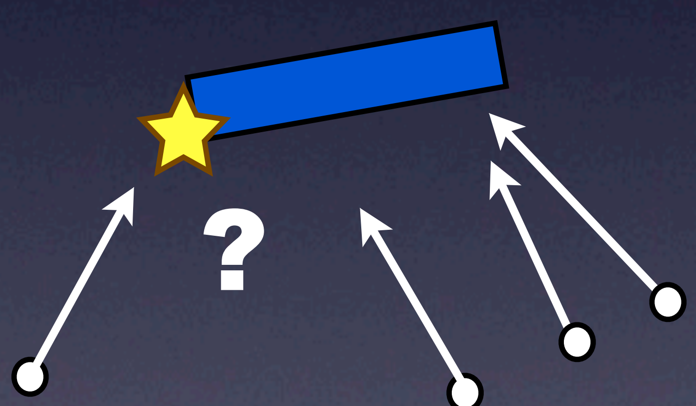

makes "future" code faster

# Targeting vectorization

- vector instructions getting 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

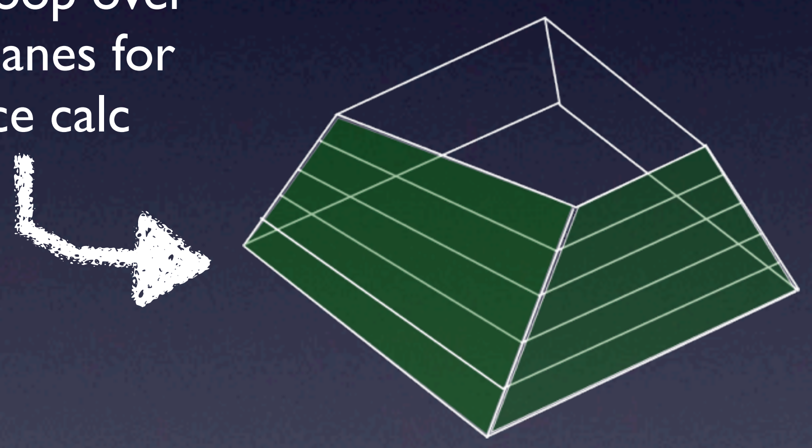
"parallel" collision detection




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 **??**

# Software challenges implied by goals

- How do we achieve **reliable** vectorization on CPU **??**
- How do we **cope with the multiplication of interfaces** ... ?

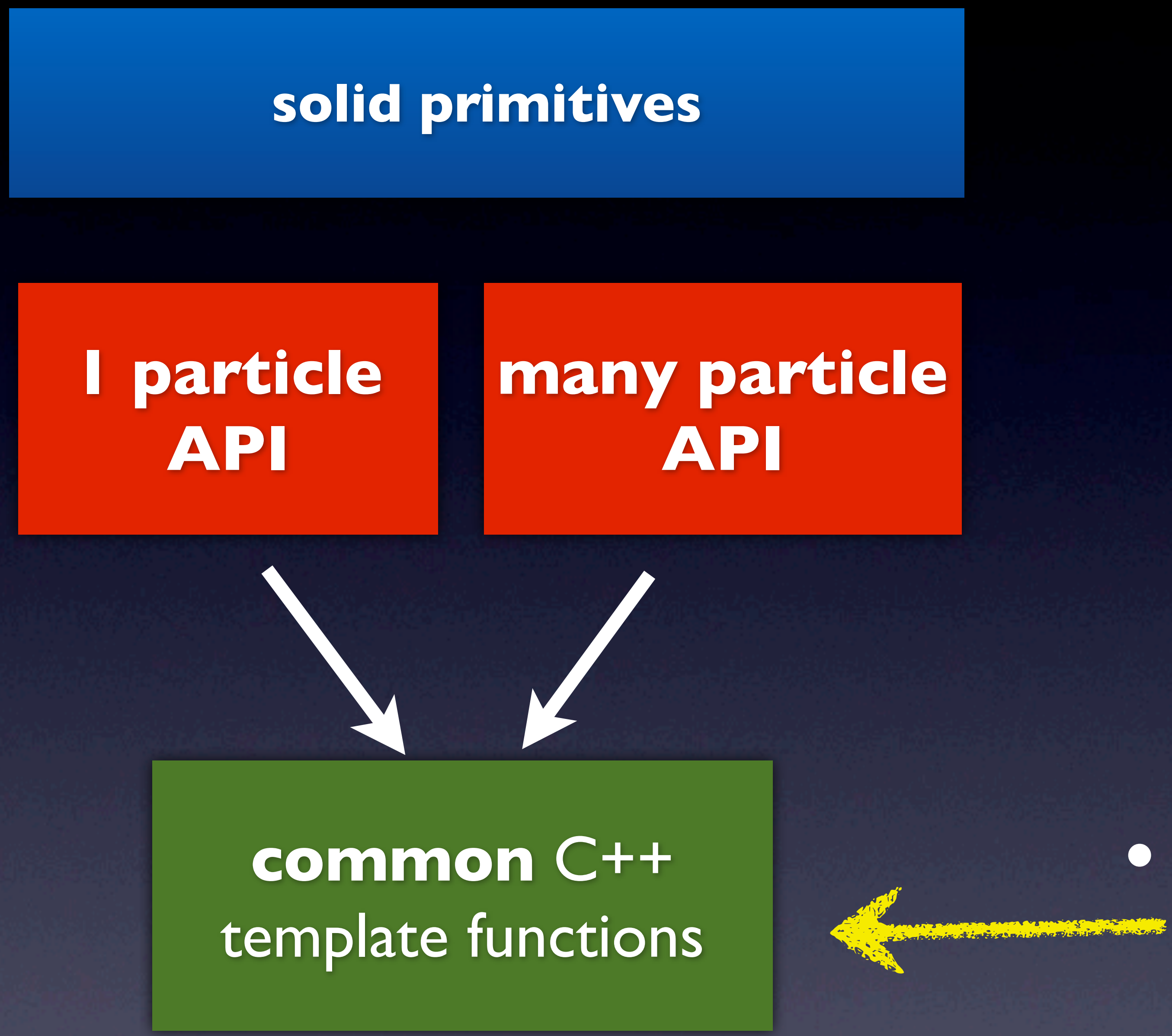
| **Box** |
|---|
| x,y,z |
| double DistanceToIn( 1 particle )<br>**double* DistanceToIn( many particles )**<br>bool Contains ( 1 particle )<br>**bool* Contains ( many particles )**<br>double SafetyToIn( 1 particle )<br>**double* SafetyToIn( many particles )**<br>double DistanceToOut ( 1 particle )<br>**double* DistanceToOut( many particles )**<br>**....** |

**>4 new functions per solid**

**~20 primitive solids**

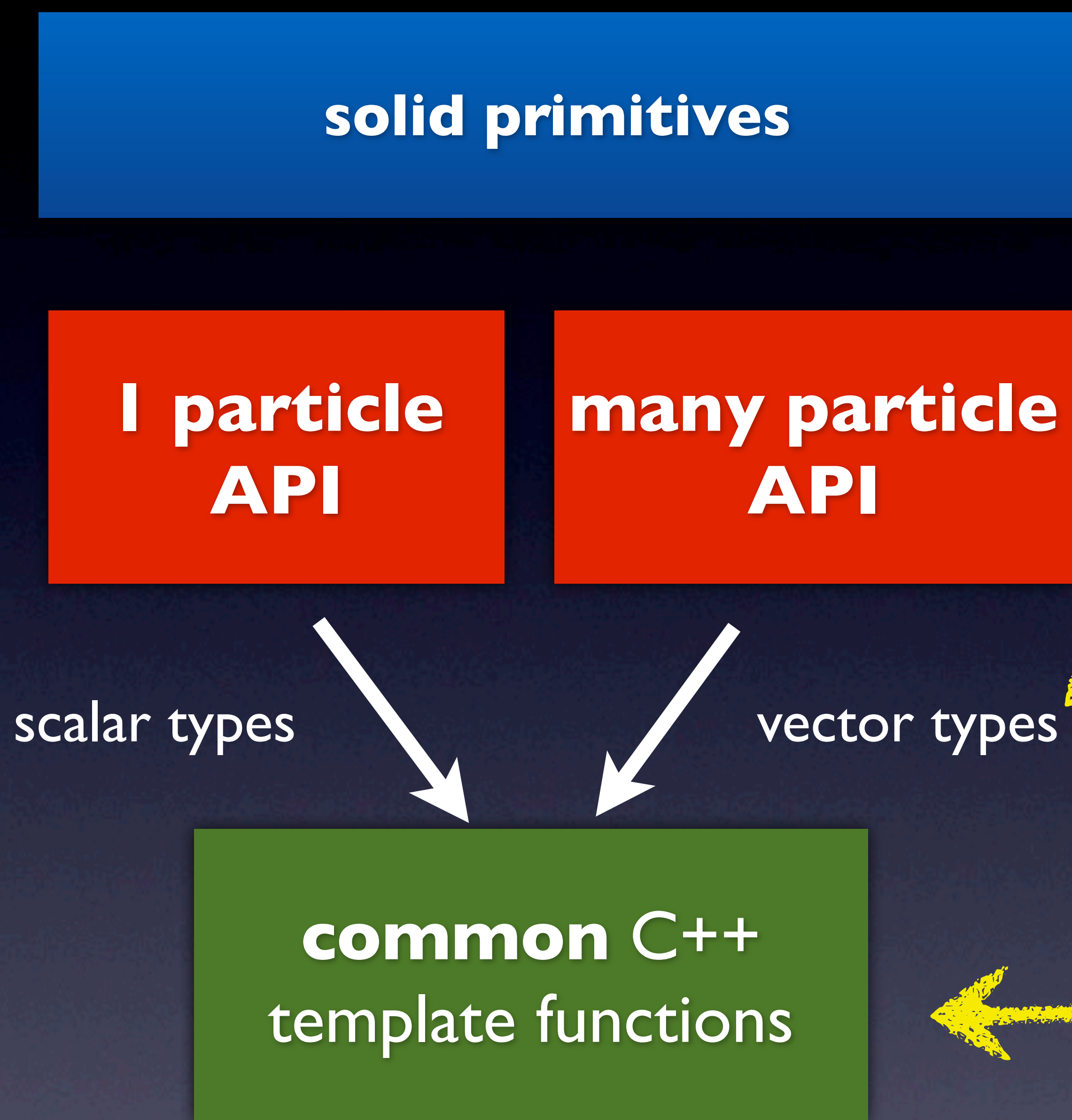
**~100 new functions to maintain ( not including CUDA yet ... )**

# Approach to target software challenge




- template C++ programming solves code multiplication issue

# Approach to target software challenge

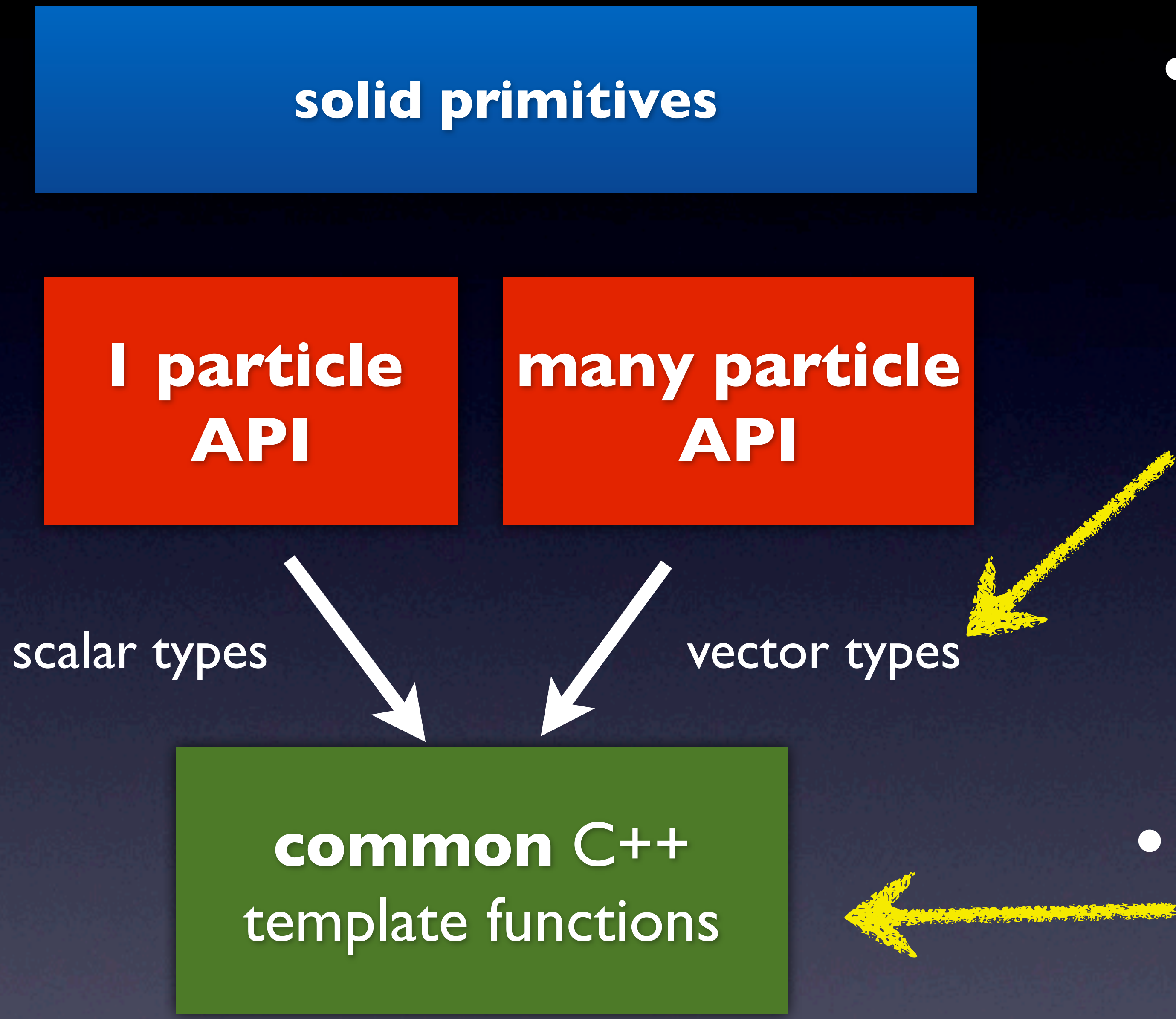



- reliable efficient SIMD vectorization achieved by using vector libraries (e.g. Vc) providing C++ approach to explicit vectorization

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

- template C++ programming solves code multiplication issue

# Approach to target software challenge




- reliable efficient SIMD vectorization achieved by using vector libraries (e.g. Vc) providing C++ approach to explicit vectorization

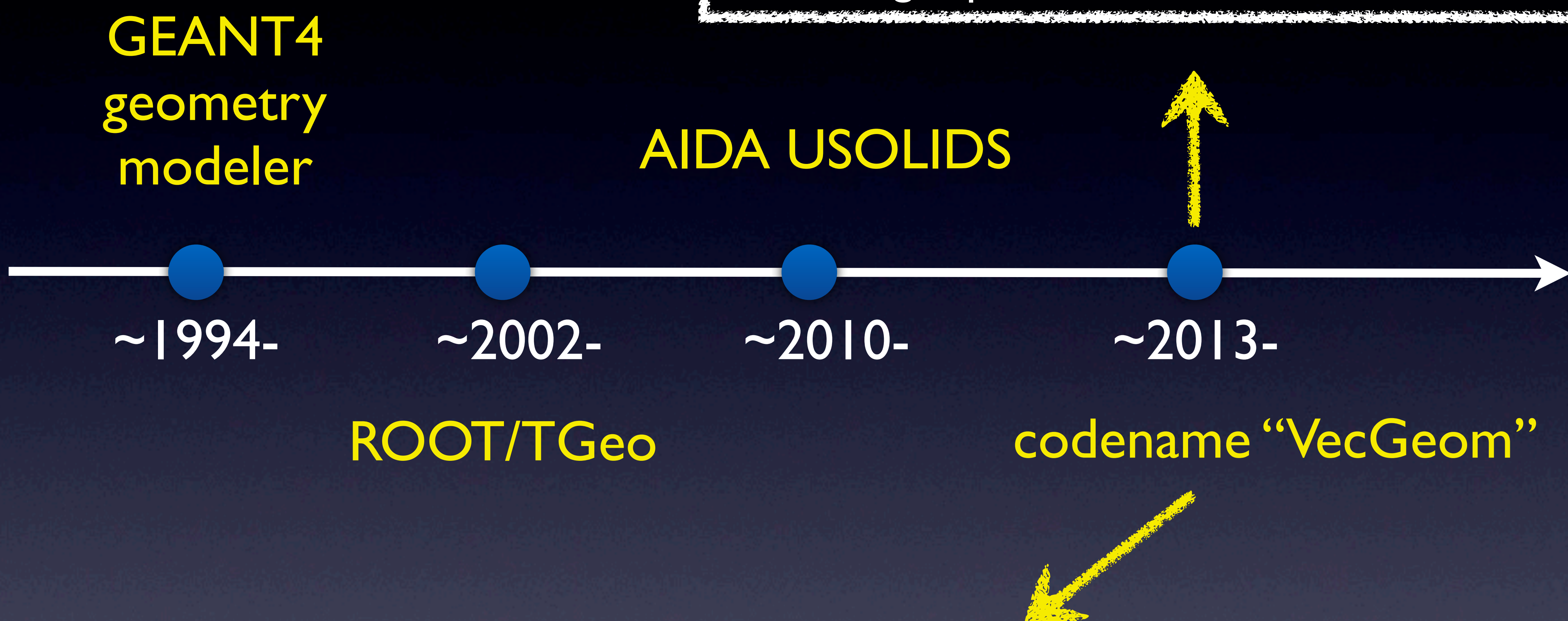
  http://code.compeng.uni-frankfurt.de/projects/vc

- template C++ programming solves code multiplication issue

- **nothing here is specific to geometry !!!**

# “VecGeom”

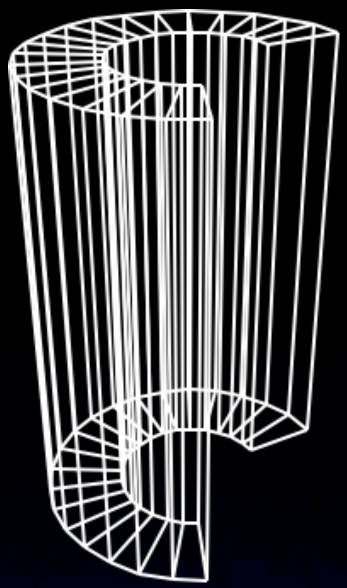
- started as feasibility study of vectorization in geometry
- now “evolved” to project addressing all goals and challenges presented




- geometry primitive code development is now seen as long-term **evolution of USolids**
- already developed back-to-back with USolids; **sharing a repository; same interfaces**

# Part II: Status + Performance

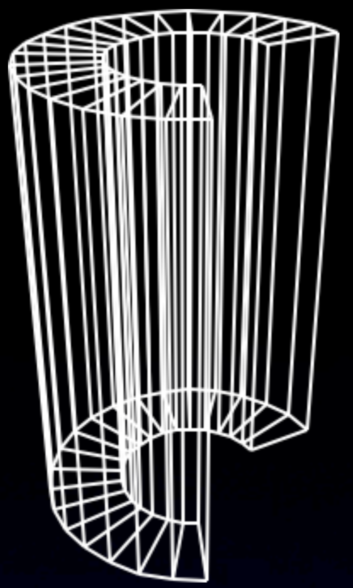
# Performance case study: the tube segment



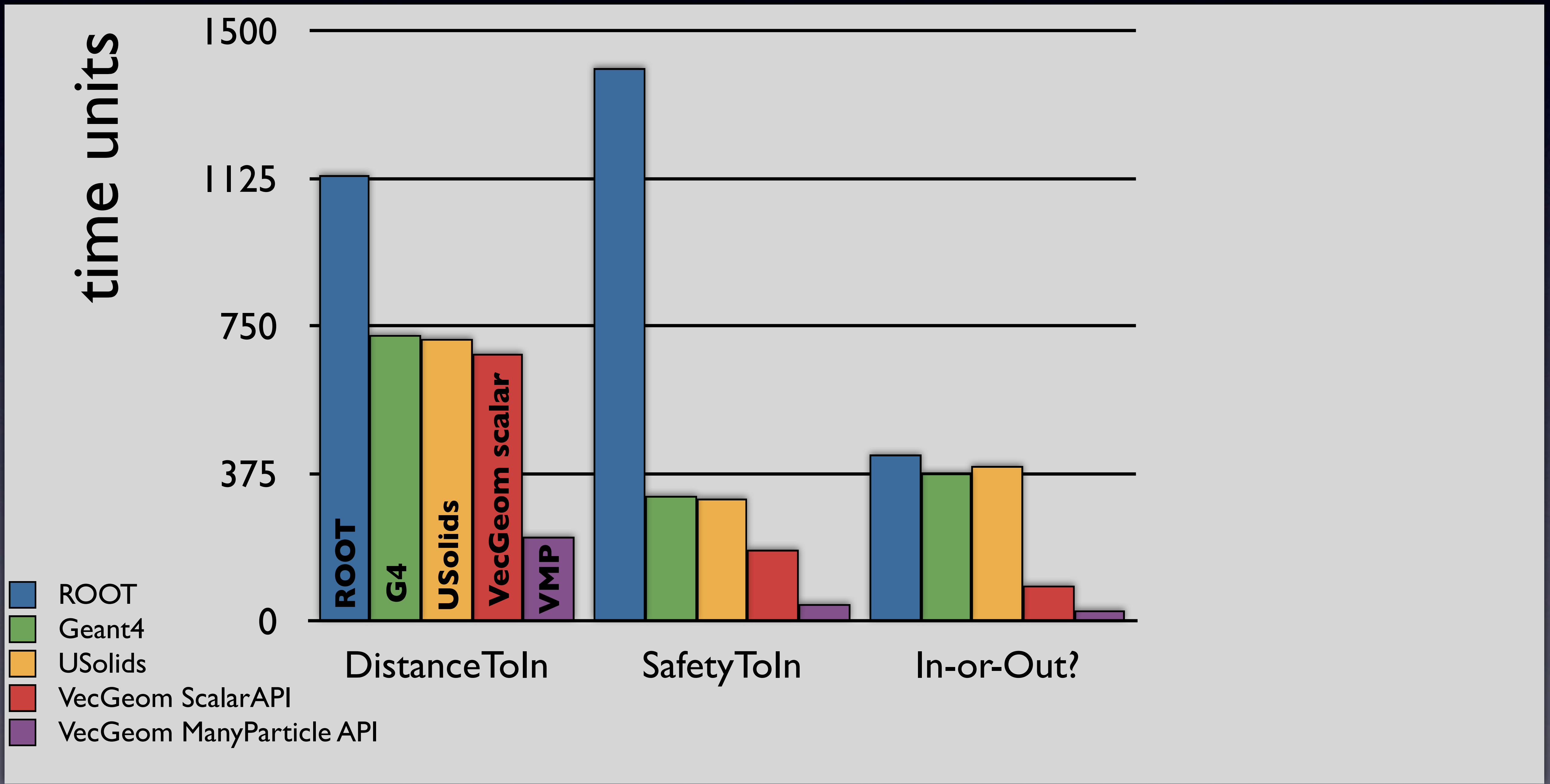
- **most used/important** shape primitive
- also **integral part** of complex shapes: polycone
- extremely **important to be as fast as we can**

gcc 4.7; -O3 -funroll-loops -mavx; no FMA; Geant4 10 (Release); Root 5.34.18 (Release); benchmark with 1000 particles

# Performance case study: the tube segment

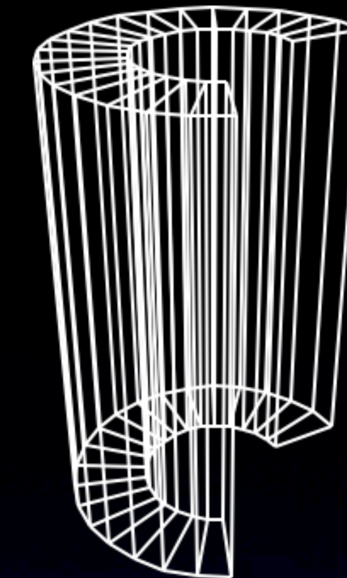


- **most used/important** shape primitive
- also **integral part** of complex shapes: polycone
- extremely **important to be as fast as we can**

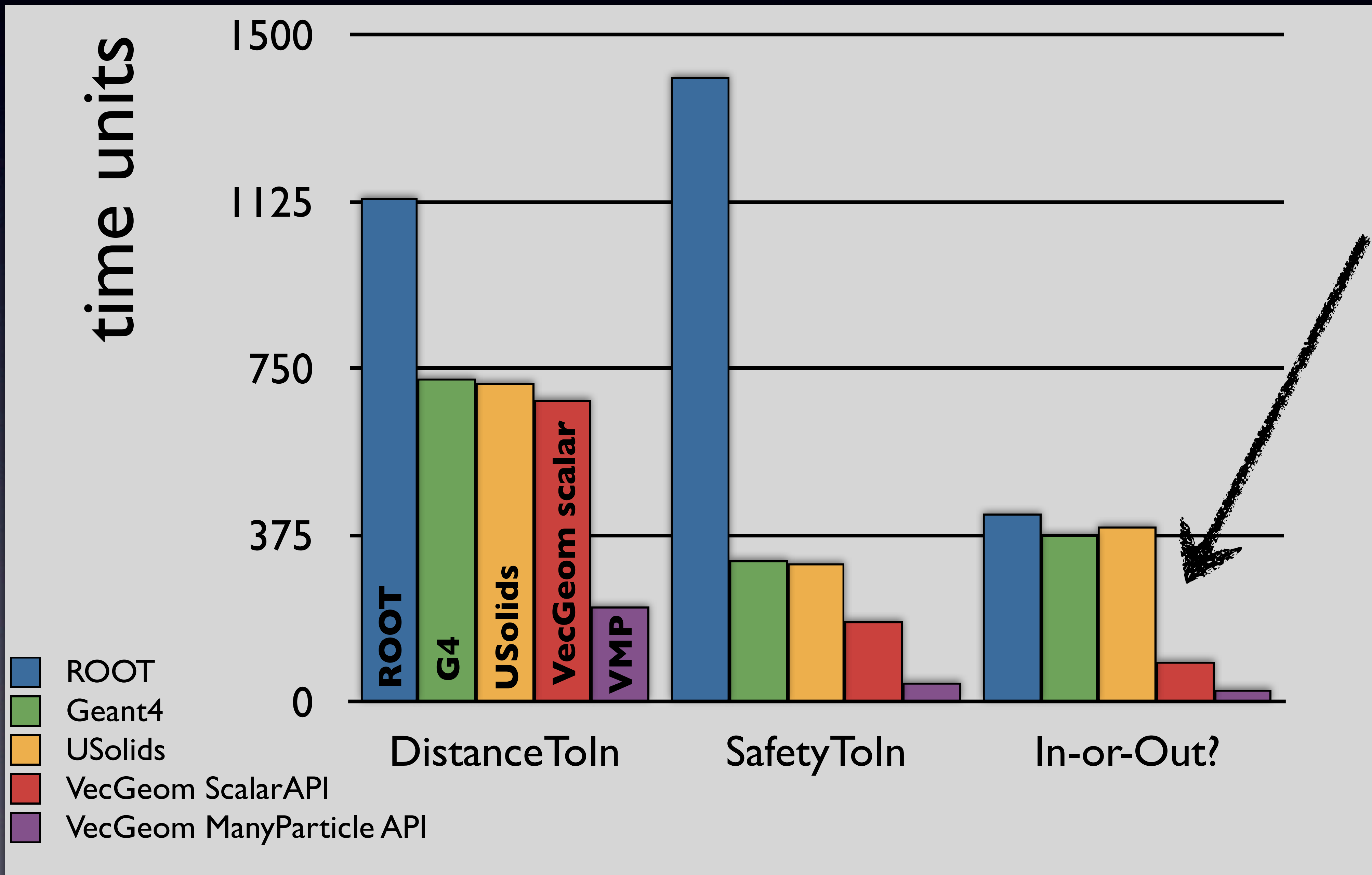



gcc 4.7; -O3 -funroll-loops -mavx; no FMA; Geant4 10 (Release); Root 5.34.18 (Release); benchmark with 1000 particles

# Performance case study: the tube segment



- **most used/important** shape primitive
- also **integral part** of complex shapes: polycone
- extremely **important to be as fast as we can**

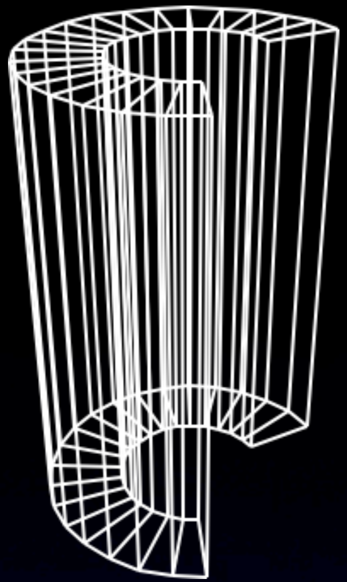



**improved scalar performance**

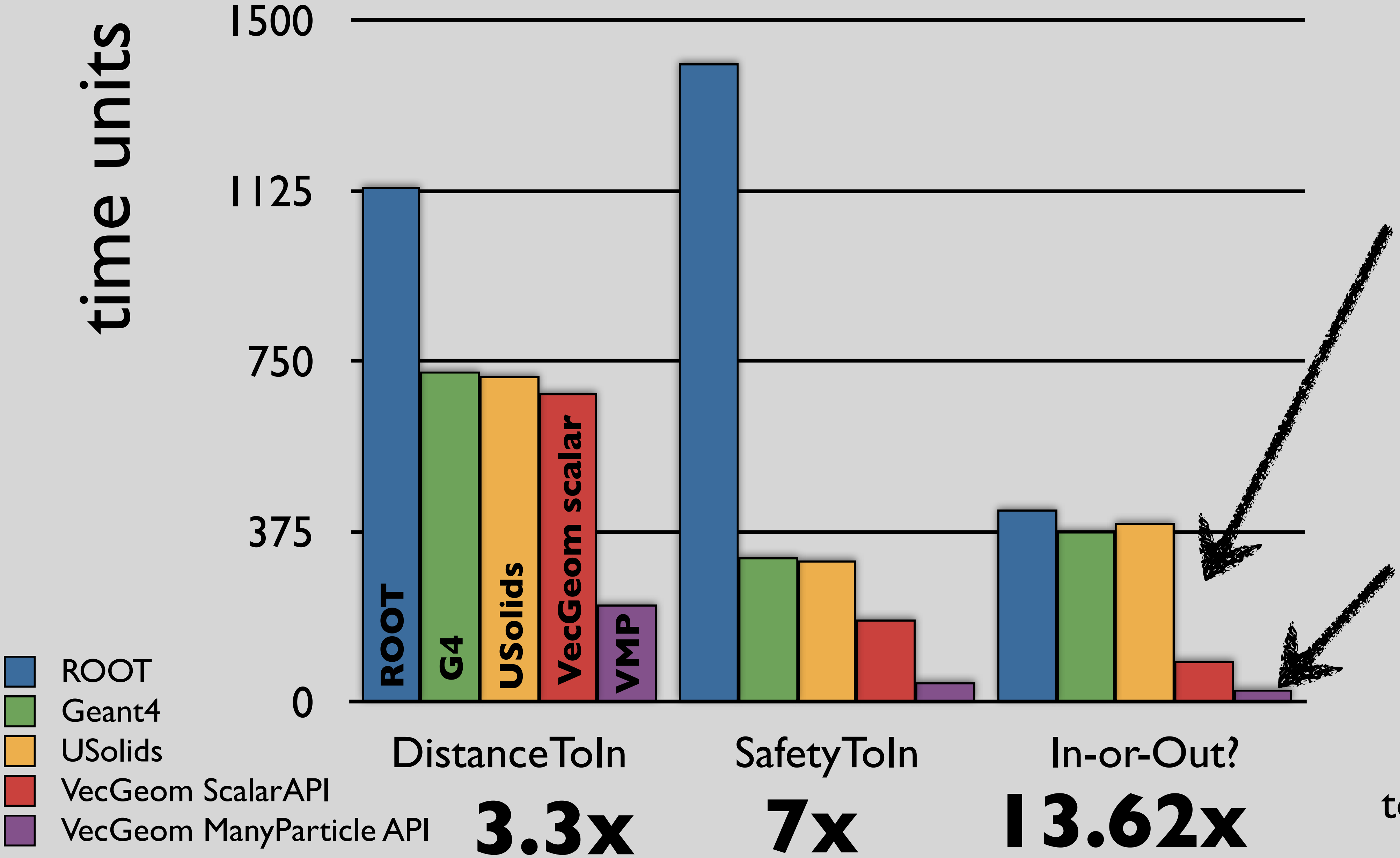
- <u>improved algorithms</u> (avoid atan2)
- <u>template shape specialization</u>

gcc 4.7; -O3 -funroll-loops -mavx; no FMA; Geant4 10 (Release); Root 5.34.18 (Release); benchmark with 1000 particles

# Performance case study: the tube segment



- **most used/important** shape primitive
- 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

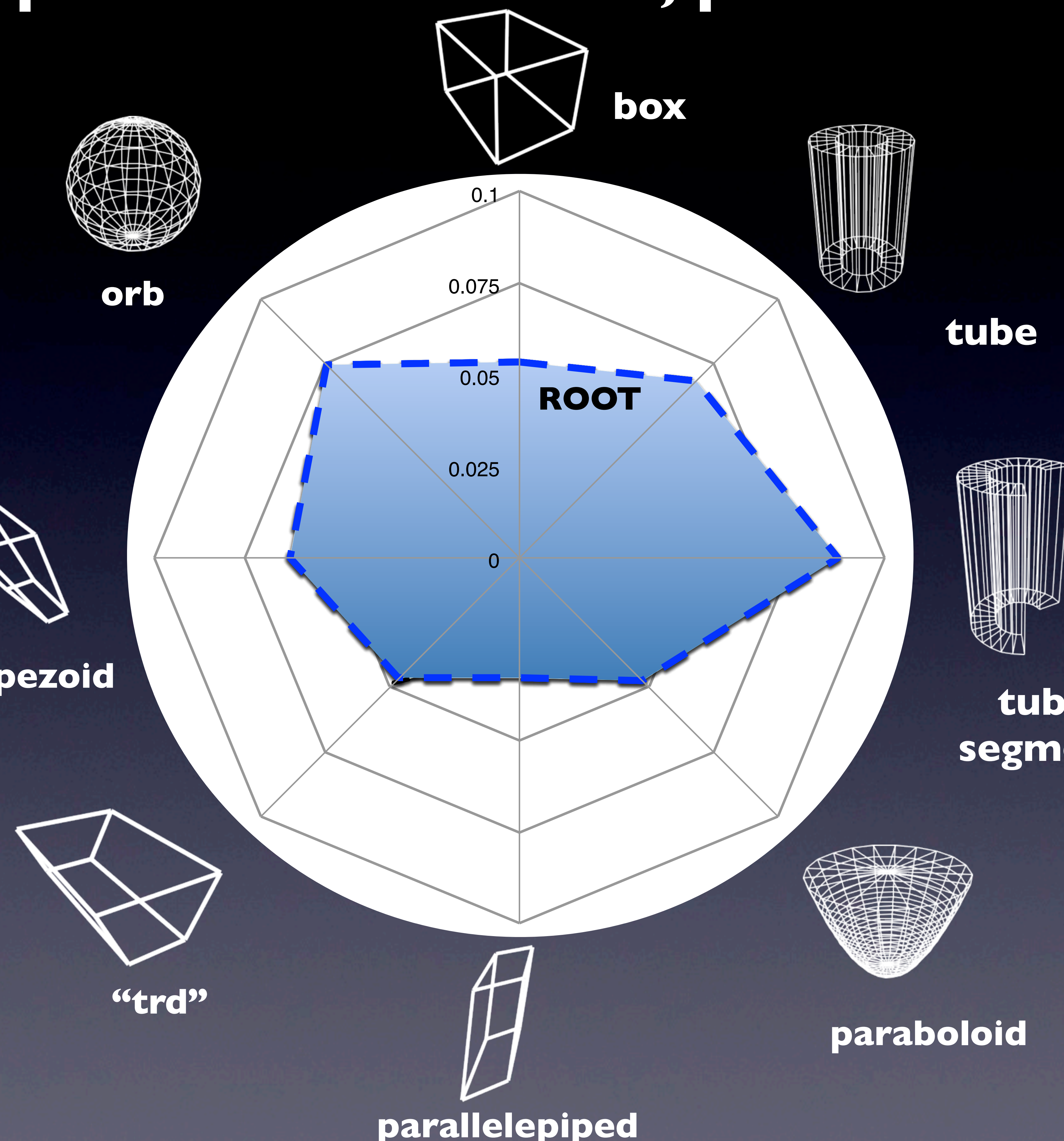
**excellent SIMD vector performance**

total speedup cmp to USolids

gcc 4.7; -O3 -funroll-loops -mavx; no FMA; Geant4 10 (Release); Root 5.34.18 (Release); benchmark with 1000 particles

# Solid/shape implementation status; performance

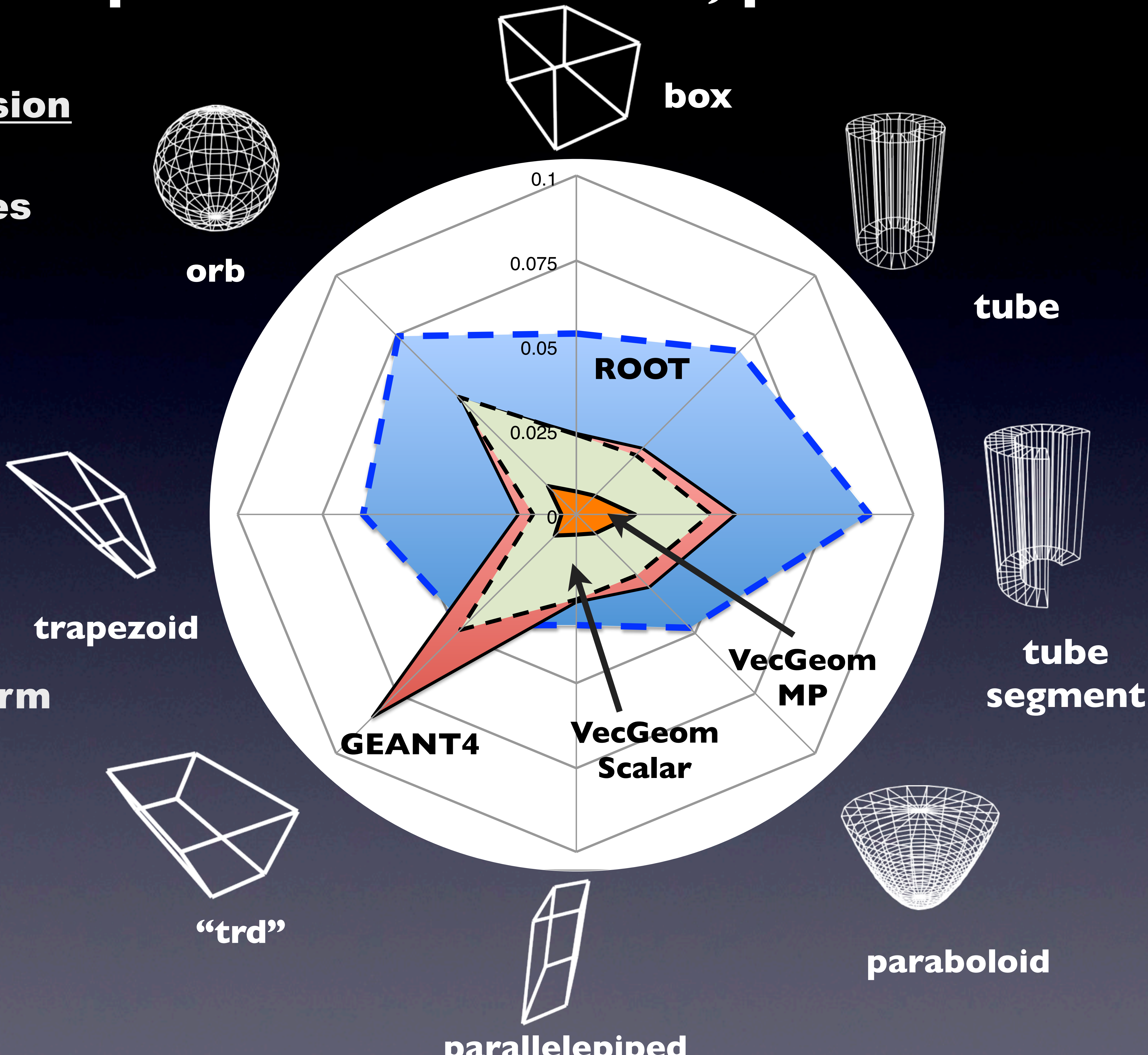
timings for <u>collision detection</u> for various primitives




timing points form a polygon per library

# Solid/shape implementation status; performance

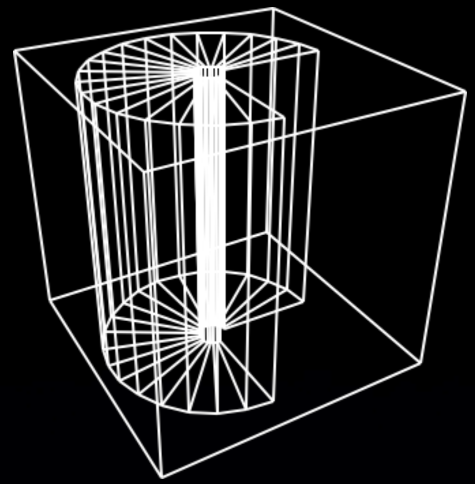
timings for <u>collision detection</u> for various primitives




timing points form a polygon per library

smaller area = better library performance

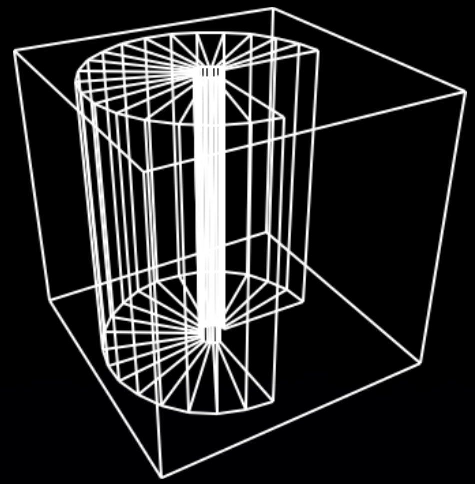
# going complex...



- 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

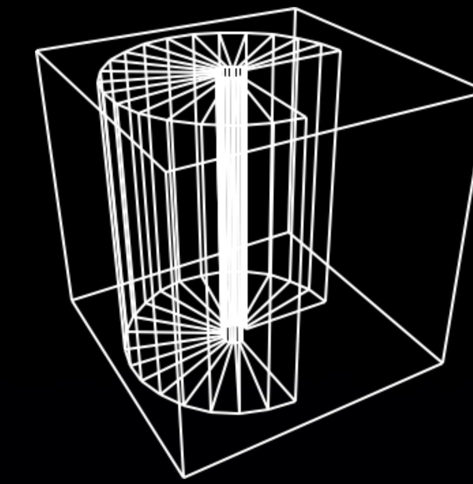
```
SubtractionSolid( AbstractShape * left, AbstractShape * right );
```

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

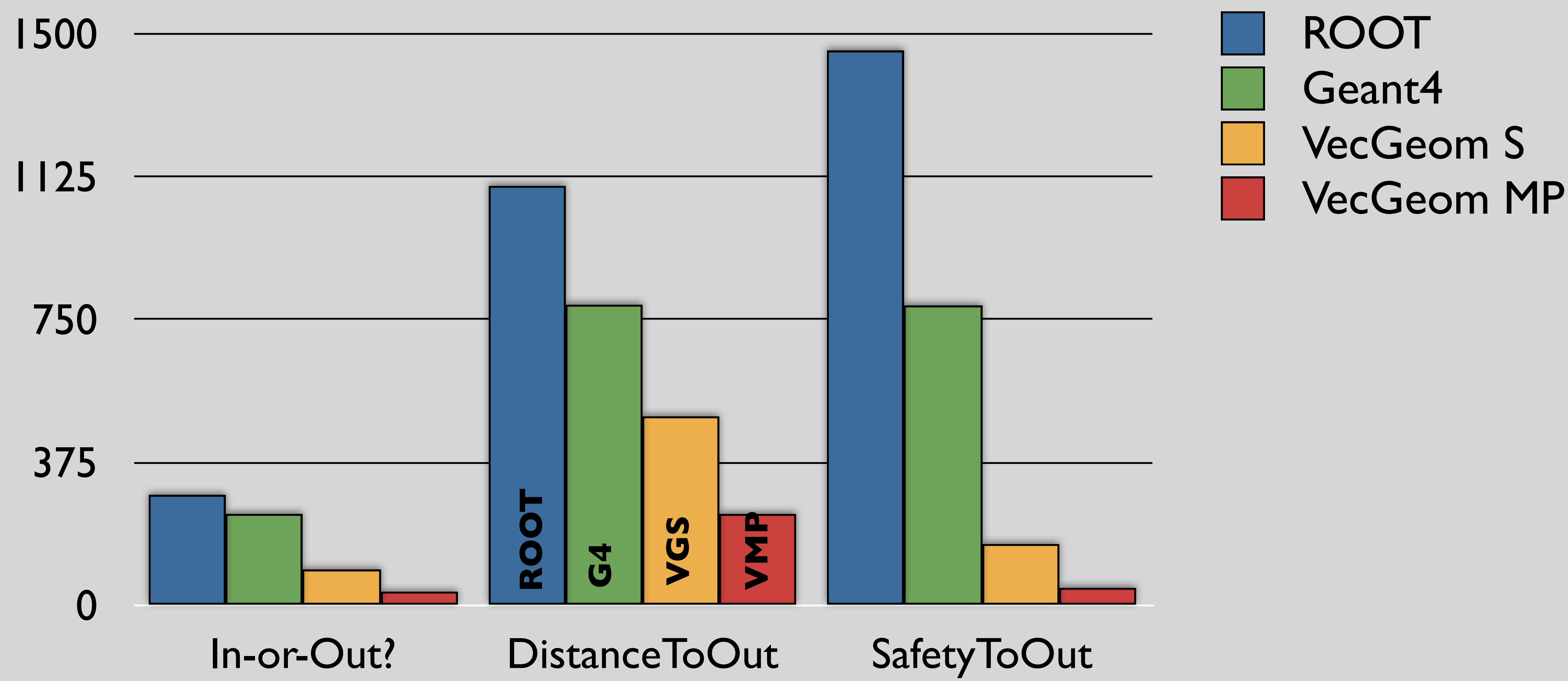
```
template <typename LeftSolid, typename RightSolid>
class TSubtractionSolid
{
  TSubtractionSolid( LeftSolid * left, RightSolid * right );
};
```

- 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 )




SIMD/ROOT speedup: **8x** **4.6x** **31x**

SIMD/Geant4 speedup: **6.6x** **3.2x** **17x**

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

# “VecGeom” and Geant-V

**(templated/ specialized) solid primitives**

1 particle API

many particle API targeting SIMD vectorization

target use

**common** C++ template functions

**detector description**

functionality to create hierarchies of volumes = detector on CPU + GPU

**detector navigation**

Scalar navigation

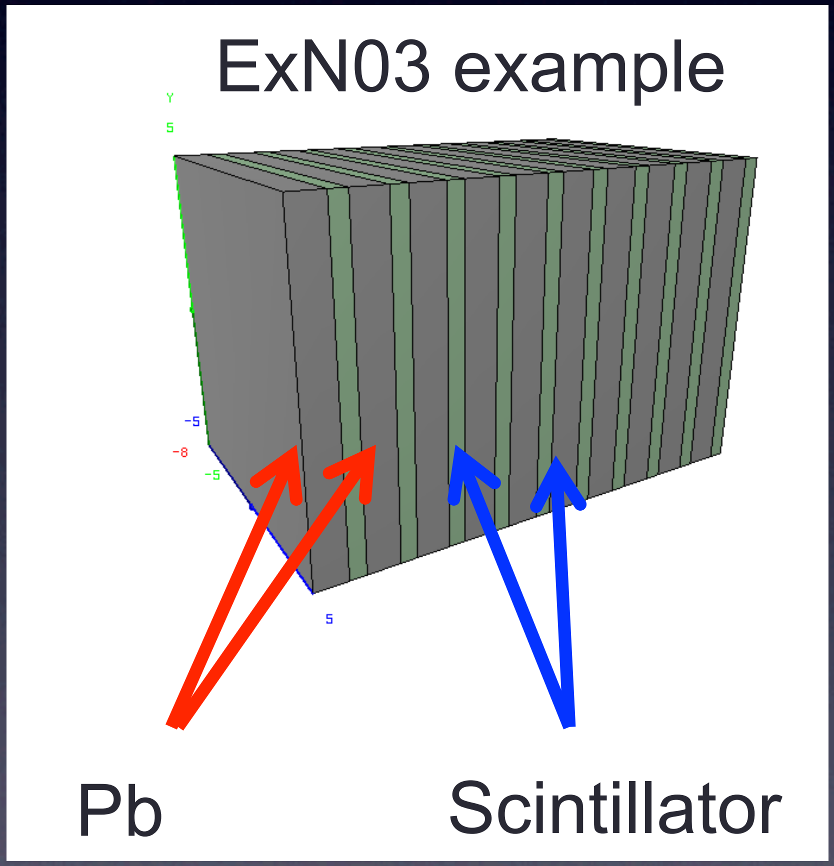
Vector navigation

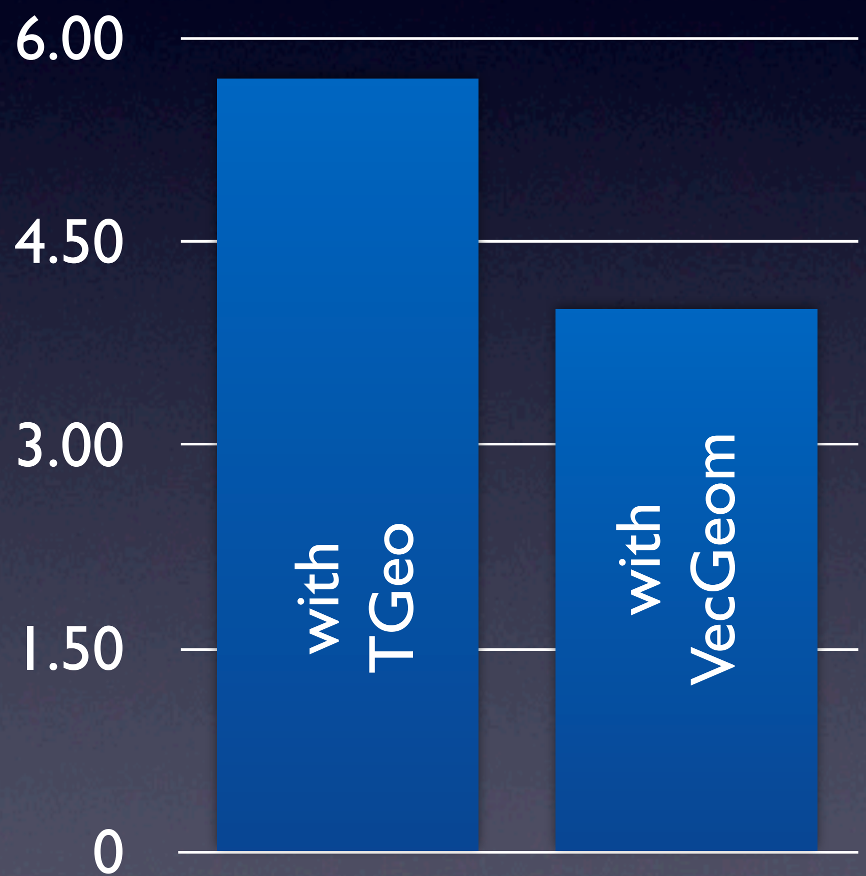
Geant-V / GPU prototype need additional library components to fully use vectorized shapes:

- shape hierarchies on CPU + GPU
- vector navigator

# “VecGeom” in action

- Geant-Vector prototype can run complete first particle-detector simulations using VecGeom ( or with ROOT/TGeo )
- measured a **total simulation runtime improvement** of **40%** going from ROOT/TGeo to VecGeom for small example



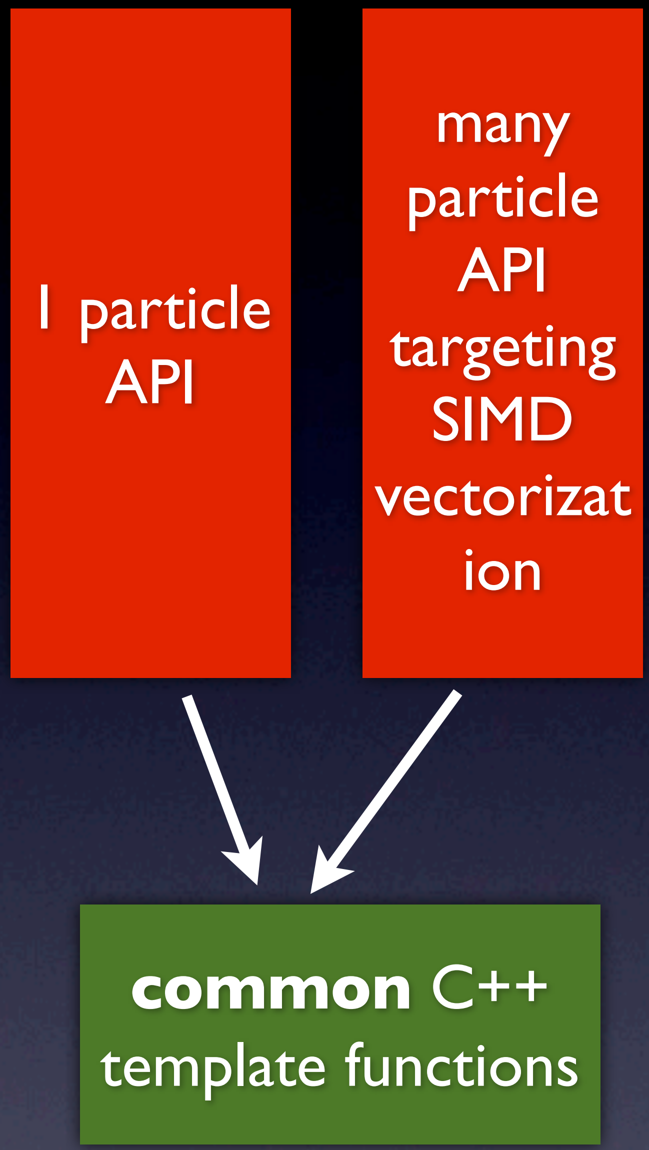




- should be able to simulate with CMS detector soonish ....

# Part III: Some details on programming approach

# achieving shared scalar / vector code

remember...




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

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

# achieving shared scalar / vector code

remember...

1 particle API

many particle API targeting SIMD vectorization

**common** C++ template functions

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

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

```
template<class Backend>
Backend::double_t
common_distance_function( Backend::double_t input )
{
    // complicated code implementing this function
    // using  abstract types that Backend provides
}
```

# achieving shared scalar / vector code

remember...

1 particle API

many particle API targeting SIMD vectorization

**common** C++ template functions

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

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

```
template<class Backend>
Backend::double_t
common_distance_function( Backend::double_t input )
{
    // complicated code implementing this function
    // using  abstract types that Backend provides
}
```

- "Backend" is a (trait) 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 boolIsScalar=false;
    static const bool IsSIMD=true;
};
```

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

# shared scalar-vector code: example

- 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

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

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

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

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

# shared scalar-vector code: example

- 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

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

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

```
Vc::double_v
Point::Distance(Vector3D<Vc::double_v> 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

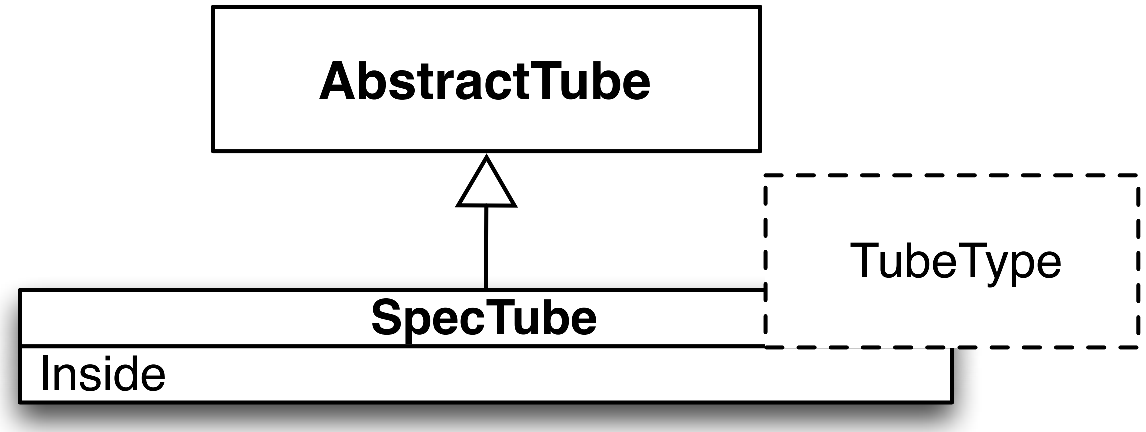
# Summary

- 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)
- development model could be extended to other components of Geant-V prototype

# Backup

# Shape specialization by example

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




- if statements (“branches”) in **generic** code can be compiled away

```
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