Annual concurrency forum meeting 2-4-2014

Towards a high performance detector geometry library on CPU and GPU

-- present status and future directions --

Sandro Wenzel / CERN-PH-SFT

(for the GPU simulation+ Geant-Vector prototypes)

building on previous talks: 5-6-13 / 9-10-13 / 29-1-14

CERN

intel EMEA High Performance and Throughput Computing

Fermilab

CERN openlab

Outline

Part I ("Status of demonstrator")

"Many particles" SIMD optimizations in geometry

- recap of problem statement
- performance numbers + ingredients how we got there

Part II ("Beyond the demonstrator")

Ideas towards a universal high-performance library for detector geometry

new

- "SIMD everywhere"
- further requirements
- possible solutions

Part III (by Johannes De Fine Licht)

Status of an "abstracted" scalar/SIMD/GPU geometry prototype library

new

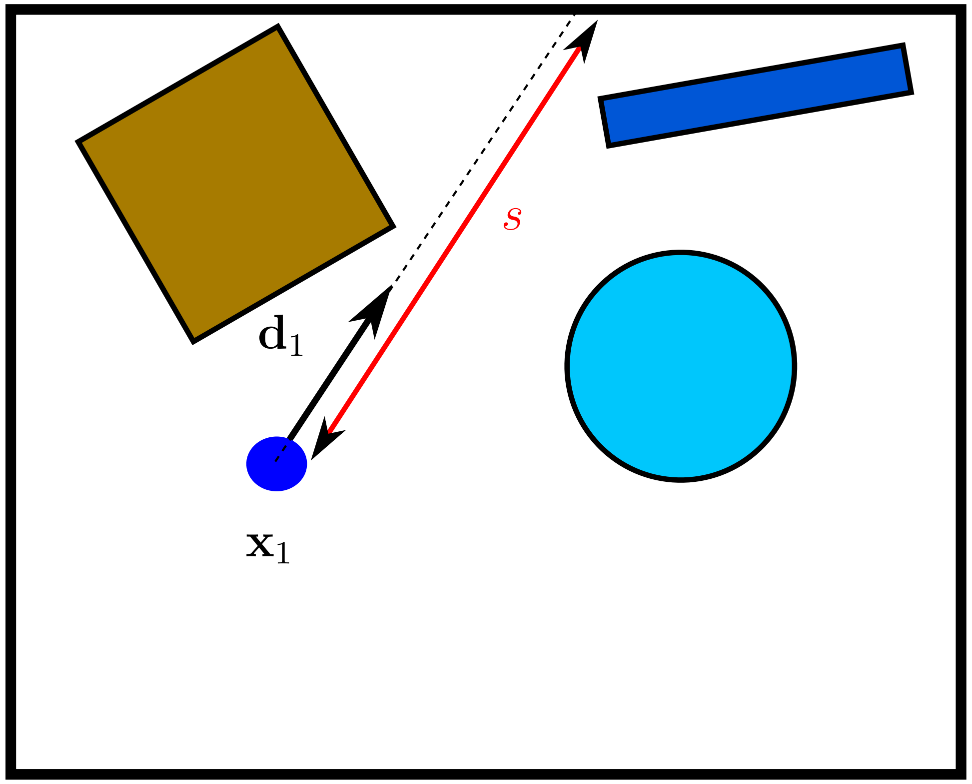
Recap of problem statement and status of many-particle vectorization

with contributions from

Marilena Bandieramonte (University of Catania, Italy)
Georgios Bitzes (CERN Openlab)
Laurent Duhem (Intel)
Raman Sehgal (BARC, India)
Juan Valles (CERN summer student)

The original problem statement in pictures

typical geometry task in particle tracking: **find next hitting boundary and get distance to it**

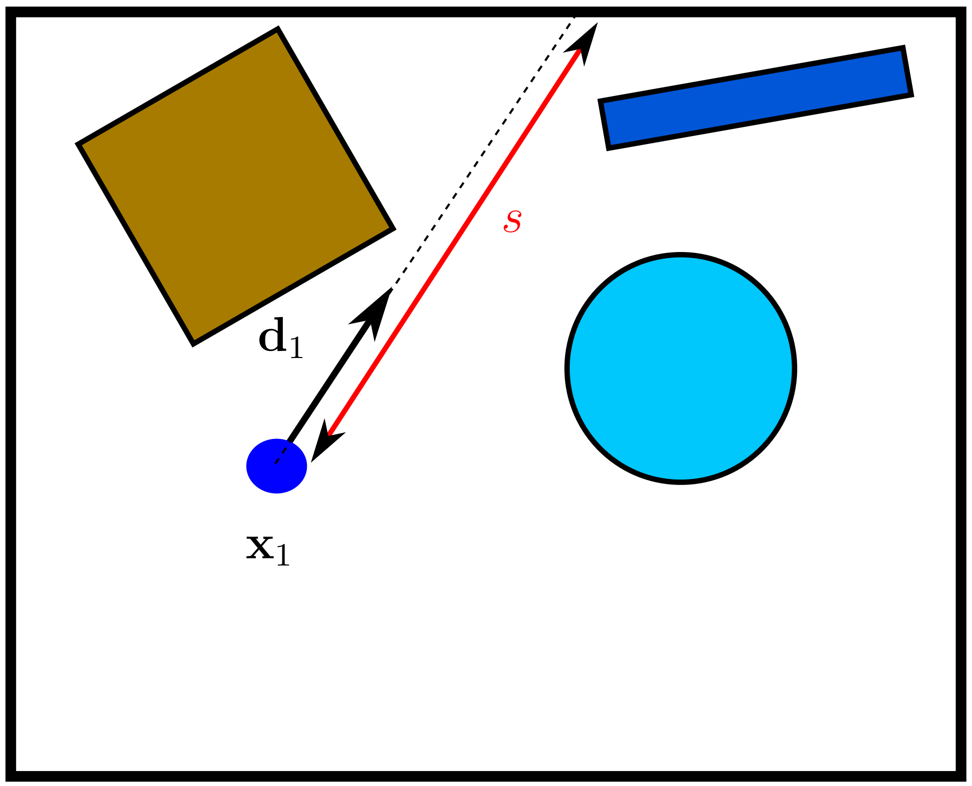


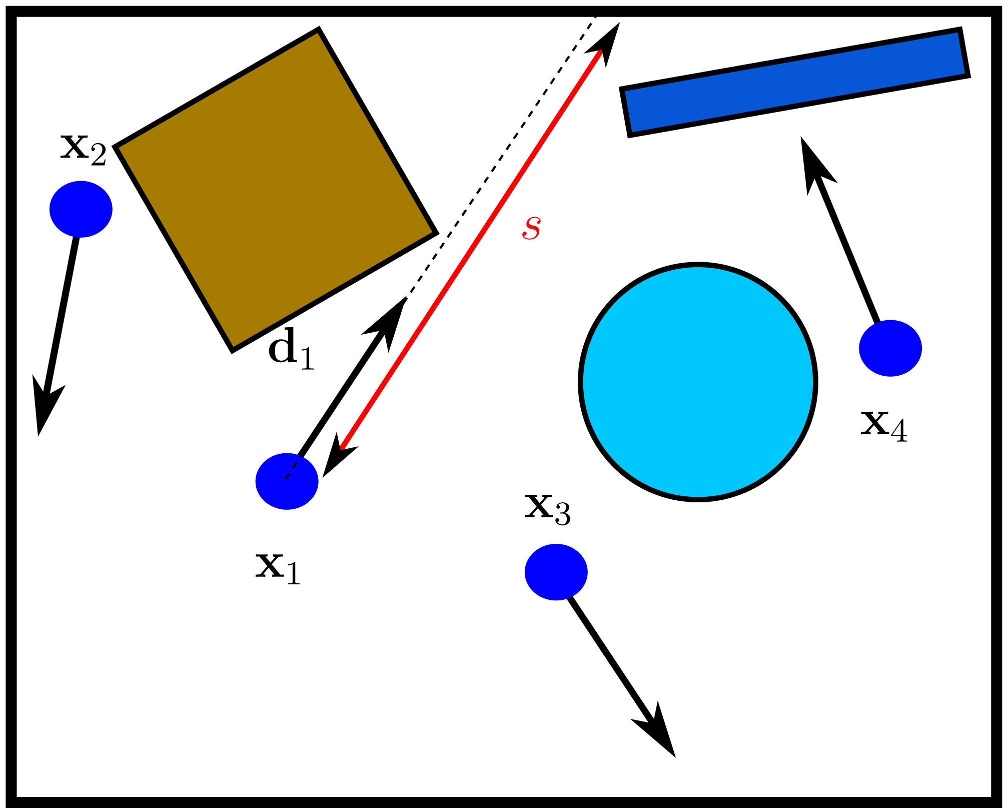
1 particle

functionality provided by existing code (Geant4, ROOT,...)

The original problem statement in pictures

typical geometry task in particle tracking: **find next hitting boundary and get distance to it**





1 particle

functionality provided by existing code (Geant4, ROOT,...)

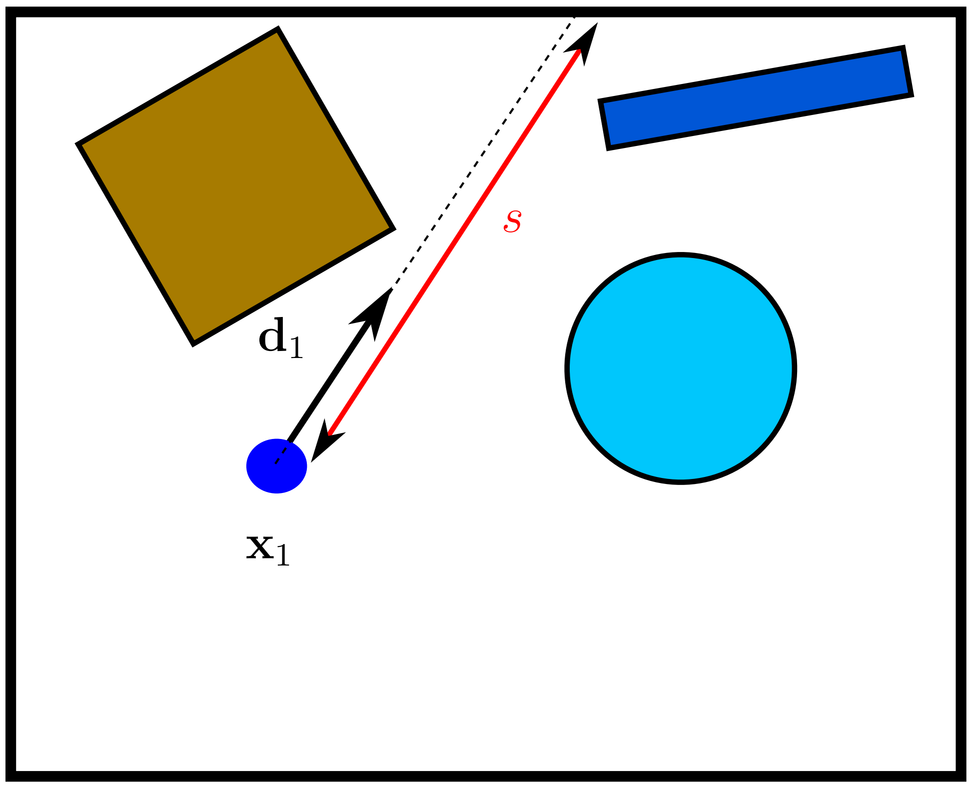
vectors of particles

functionality targeted by future simulation approaches

aim for efficient utilization of current and future hardware

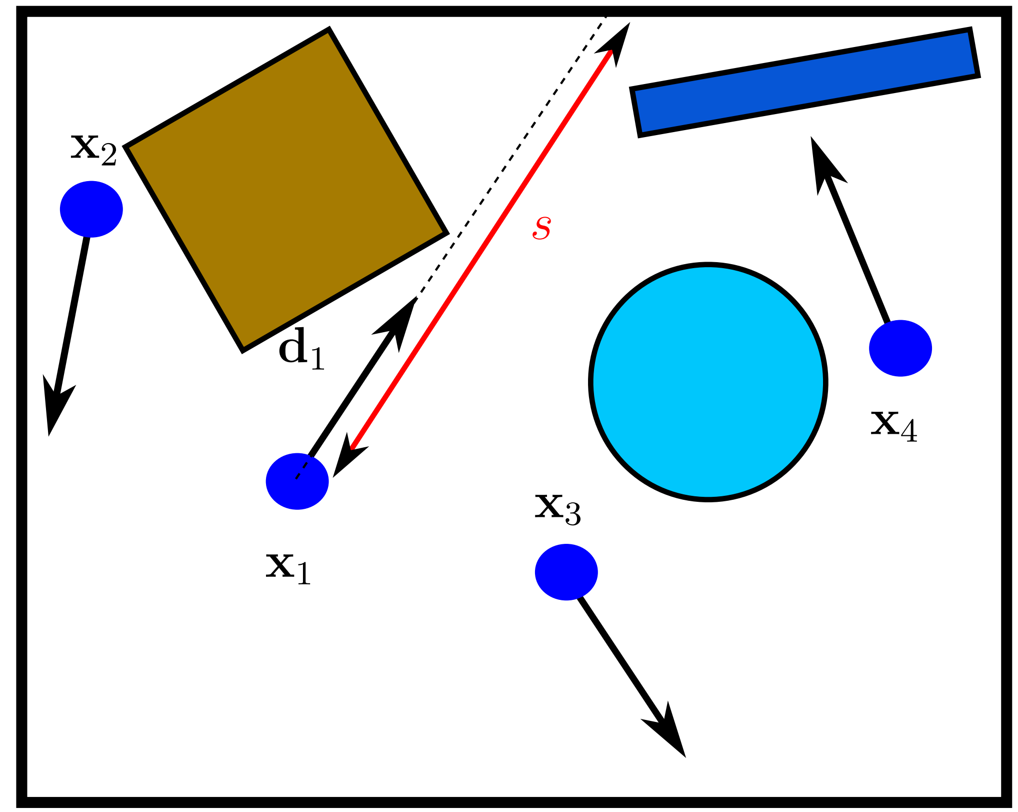
The original problem statement in pictures

typical geometry task in particle tracking: **find next hitting boundary and get distance to it**



1 particle

functionality provided by existing code (Geant4, ROOT,...)



vectors of particles

functionality targeted by future simulation approaches

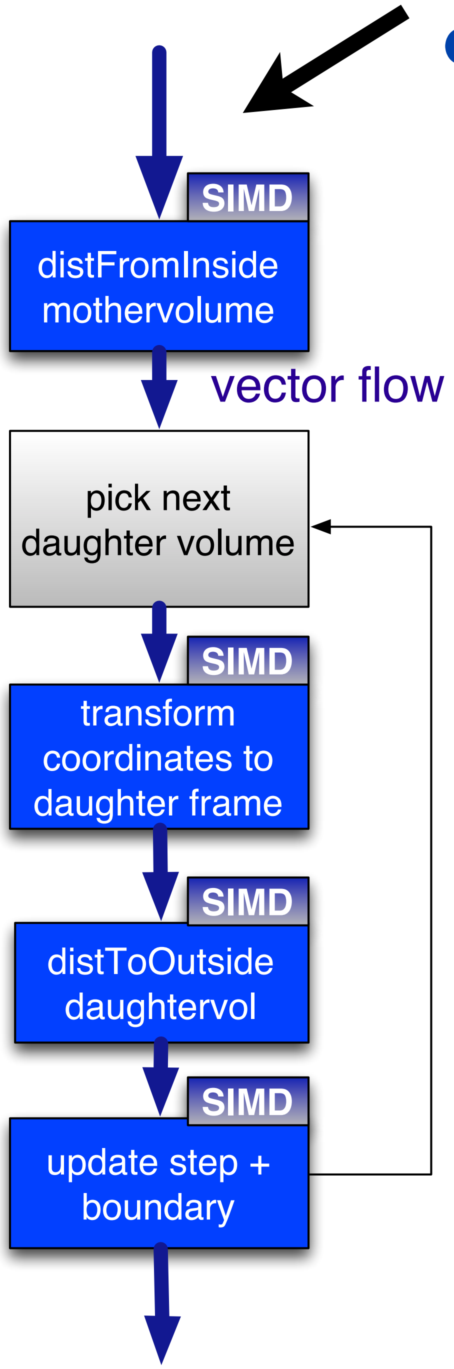
aim for efficient utilization of current and future hardware

➡ **demonstrator started ~04/2013**

Recap of performance status

provided SIMD optimized vector interfaces and algorithms for some elementary solids and geometry base functions (implemented important functions for particle navigation)

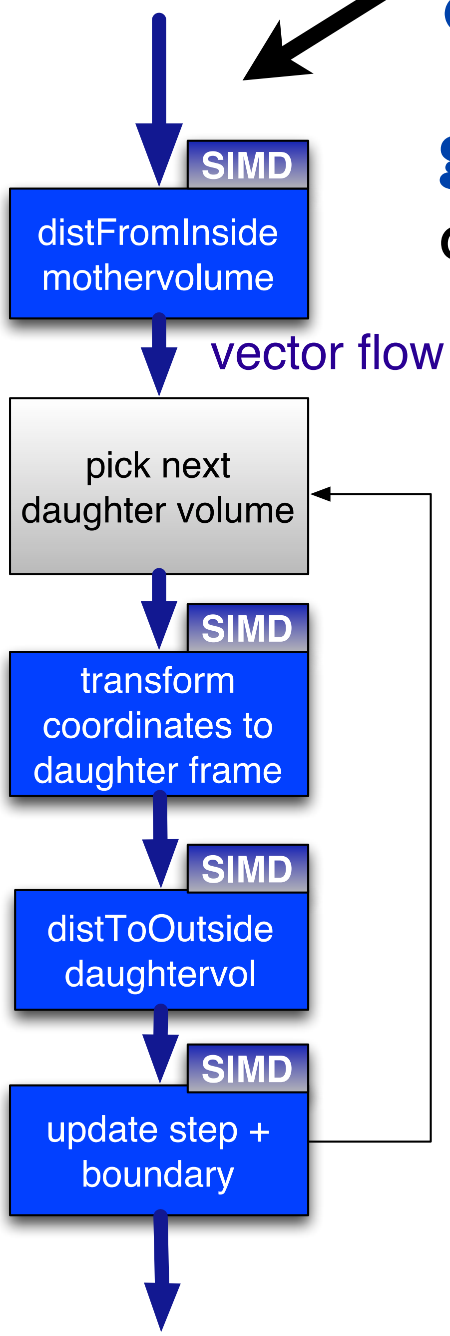
can run chain of algorithms in vector/SIMD mode



CHEP13 paper: http://arxiv.org/pdf/1312.0816.pdf

Recap of performance status

provided SIMD optimized vector interfaces and algorithms for some elementary solids and geometry base functions (implemented important functions for particle navigation)



can run chain of algorithms in vector/SIMD mode

good overall performance gains for such an algorithm (in toy detector with 4 boxes, 3 tubes, 2 cones) - compared to ROOT/5.34.17

	16 particles	1024 particles	SIMD MAX
Intel IvyBridge (AVX)	~2.8x	~4.0x	4x
Intel Haswell (AVX2)	~3.0x	~5.0x	4x
Intel Xeon-Phi (AVX512)	~4.1x	~4.8x	8x

Xeon-Phi and Haswell benchmarks by CERN Openlab (Georgios Bitzes)

gcc 4.8; -O3 -funroll-loops -mavx; no FMA

CHEP13 paper: http://arxiv.org/pdf/1312.0816.pdf

Ingredient 1: SIMD Vectorization

How to (particle) **vectorize existing code** (with many branches...) ?

Option A ("free lunch"):

put code into a loop and let the compiler do the work

- works in very few cases

Ingredient 1: SIMD Vectorization

How to (particle) **vectorize existing code** (with many branches...) ?

Option A ("free lunch"):

put code into a loop and let the compiler do the work

- works in very few cases

Option B ("convince the compiler"):

refactor the code to make it "auto-vectorizer" friendly

- might work but strongly compiler dependent

Ingredient I: SIMD Vectorization

How to (particle) **vectorize existing code** (with many branches...) ?

Option A ("free lunch"):

put code into a loop and let the compiler do the work

- works in very few cases

Option B ("convince the compiler"):

refactor the code to make it "auto-vectorizer" friendly

- might work but strongly compiler dependent

Option C ("use SIMD library"):

refactor the code and perform explicit vectorization using a vectorization library

- always SIMD vectorizes, compiler independent
- excellent experience with the Vc library
- other libraries exist: VectorType (Agner Fog), Boost::SIMD, ...

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

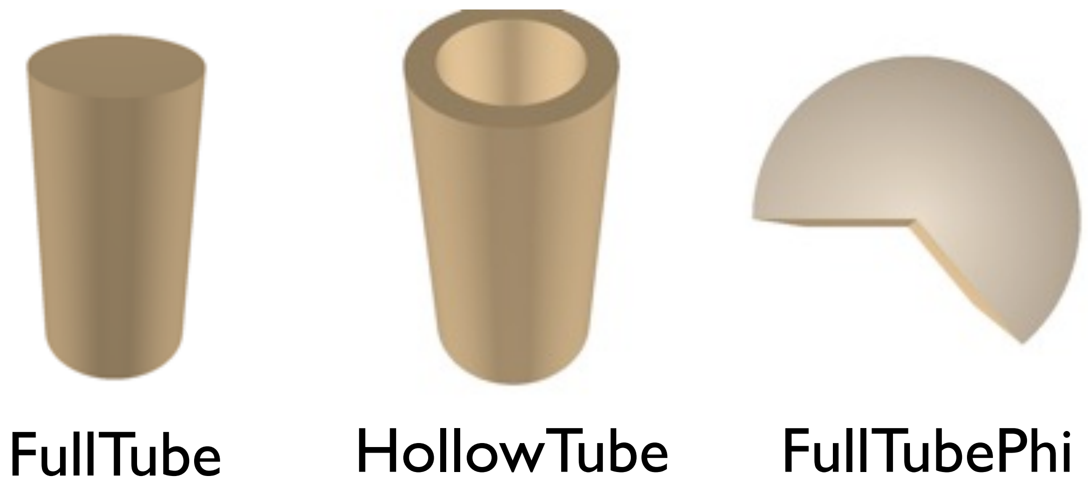
```
// hello world example with Vc-SIMD types
Vc::Vector<double> a, b, c;
c=a+b;
```

Ingredient II: C++ template techniques

"branches are the enemy of vectorization..."

a lot of branches in geometry code just distinguish between "static" properties of class instances

- general "tube solid" class distinguishes at runtime between "FullTube", "Hollow Tube" ...



Ingredient II: C++ template techniques

"branches are the enemy of vectorization..."

a lot of branches in geometry code just distinguish between "static" properties of class instances

- general "tube solid" class distinguishes at runtime between "FullTube", "Hollow Tube" ...

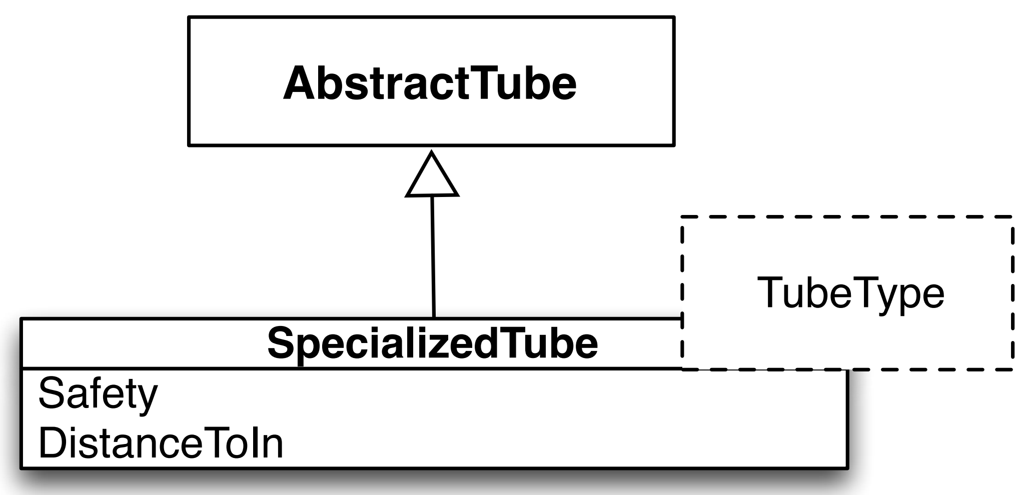
we employ **template techniques** to:

- evaluate and **reduce "static" branches at compile time**
- to **generate binary code specialized to concrete solid** instances
  - ➡ makes vectorization more efficient
  - ➡ allows better compiler optimizations in scalar code

FullTube

HollowTube

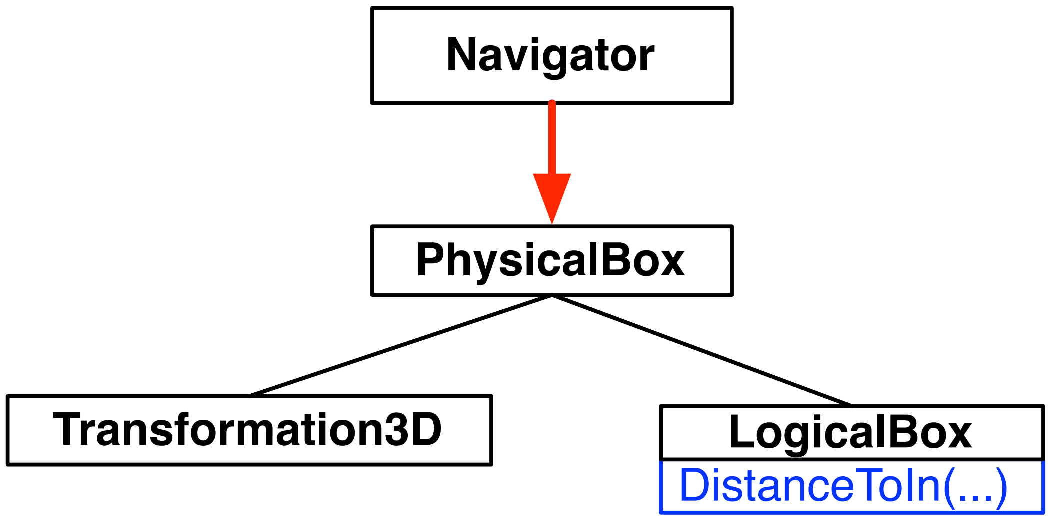
FullTubePhi



see talk (29-1-14) in this forum for further details

new

Ingredient III: Rethink class layout (somewhat)



simplified layouts!

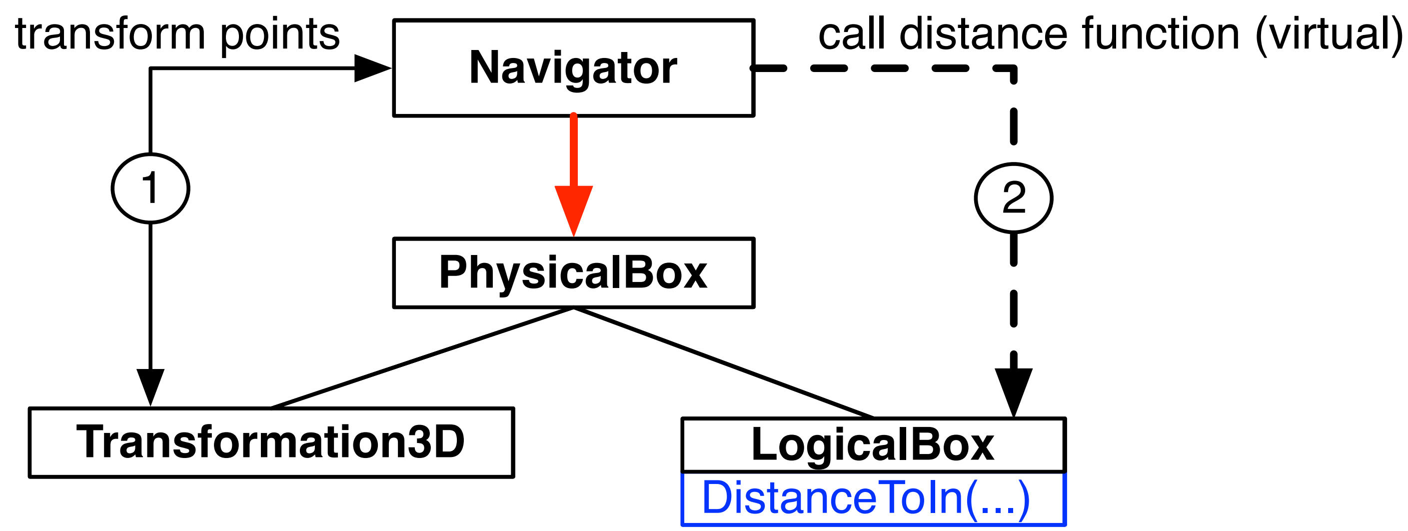
current geometry packages are **"logical volume/solid centric"**

- distance functions declared in logical solids

thanks to discussions with Laurent Duhem (Intel)

Ingredient III: Rethink class layout (somewhat)

new



simplified layouts!

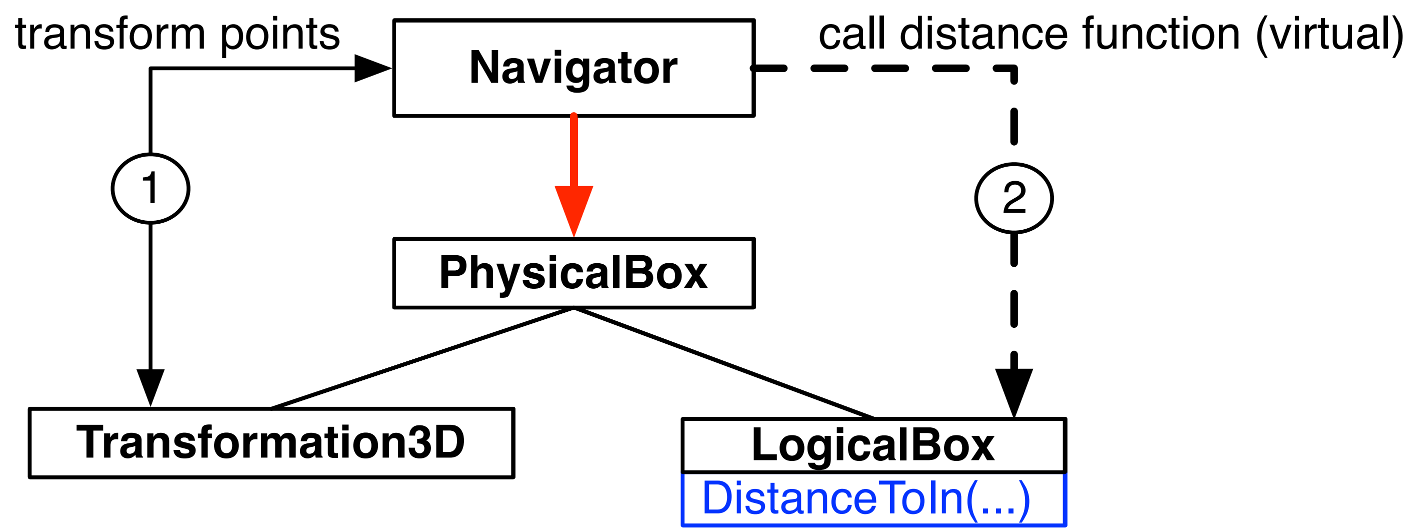
current geometry packages are
"logical volume/solid centric"

- distance functions declared in logical solids
- client code (navigator) requires two-step process to calculate distance to a detector element (PhysicalBox)
- requires temporaries

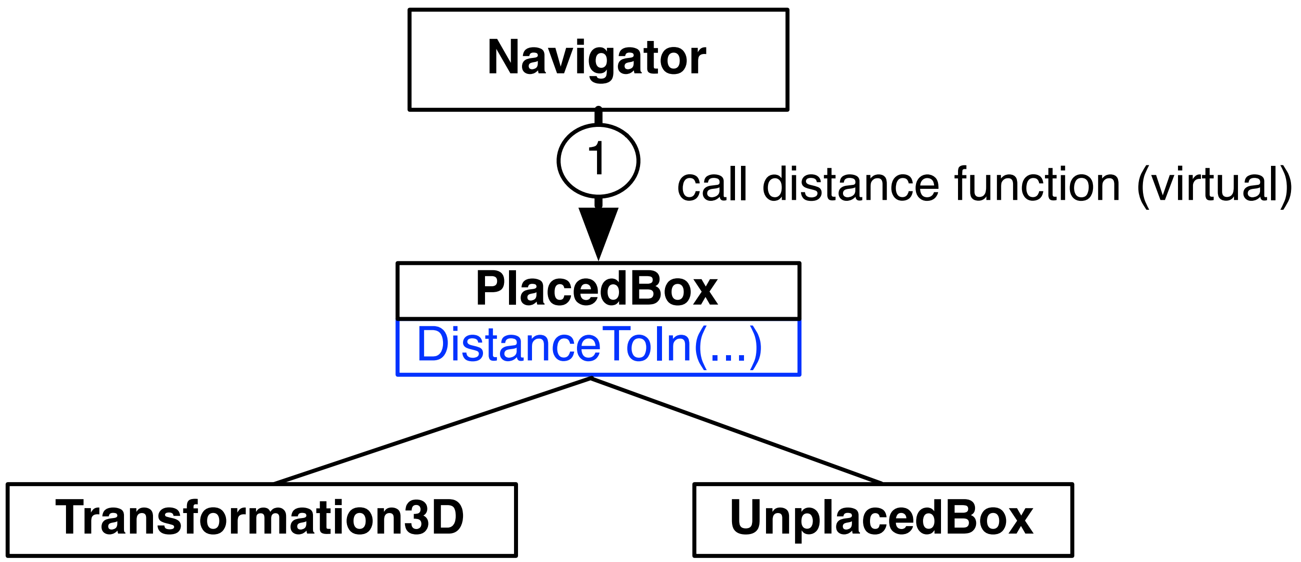
thanks to discussions with Laurent Duhem (Intel)

Ingredient III: Rethink class layout (somewhat)

new



simplified layouts!



current geometry packages are **"logical volume/solid centric"**

- distance functions declared in logical solids
- client code (navigator) requires two-step process to calculate distance to a detector element (PhysicalBox)
- requires temporaries

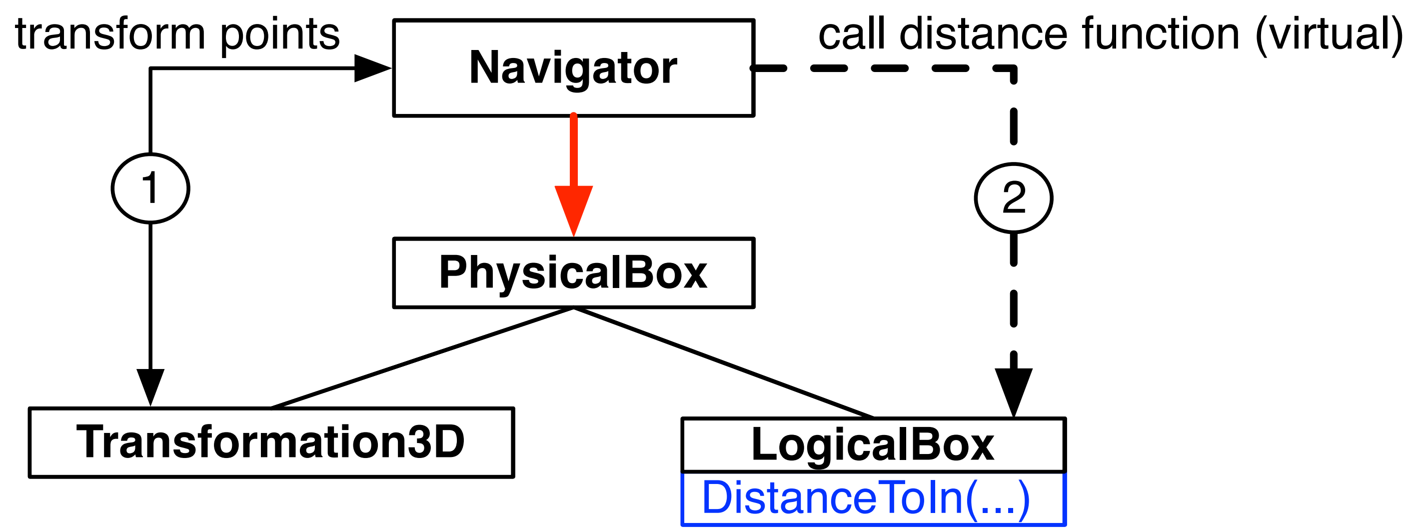
I propose to be more **"placed volume/solid centric"**

- change of resposibility of distance function
- direct call possible; better encapsulation
- no temporaries (unless wanted/needed)

thanks to discussions with Laurent Duhem (Intel)

new

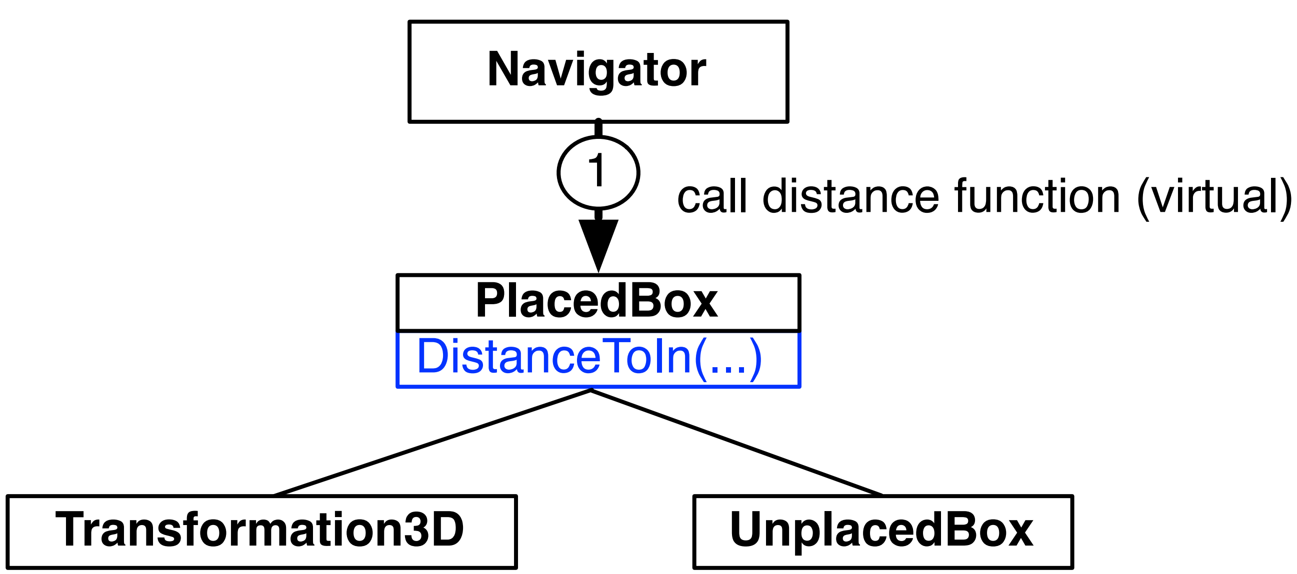
Ingredient III: Rethink class layout (somewhat)



current geometry packages are **"logical volume/solid centric"**

- distance functions declared in logical solids
- client code (navigator) requires two-step process to calculate distance to a detector element (PhysicalBox)
- requires temporaries

simplified layouts!



I propose to be more **"placed volume/solid centric"**

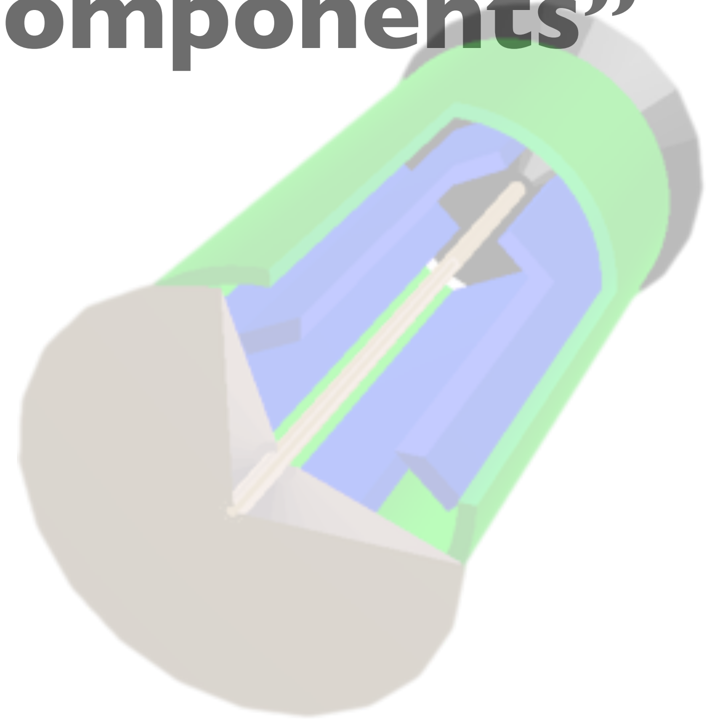
- change of resposibility of distance function
- direct call possible; better encapsulation
- no temporaries (unless wanted/needed)

➡ accounts for 15% of performance gains in SIMD benchmark

thanks to discussions with Laurent Duhem (Intel)

Beyond the demonstrator: Towards a general high performance library for detector geometry

"vectorization everywhere"
"architecture abstraction"
"reusable components"



with contributions from

Georgios Bitzes (CERN Openlab)
Johannes De Fine Licht (CERN technical student)
Guilherme Lima (Fermilab)
Raman Sehgal (BARC, India)

The demonstrator

Goals

Performance

- optimized many particle treatment

The **main points** so far ....

Approach

SIMD

template techniques

algo + class review

- template class specialization / code generation

Implementation

Vc library

Beyond the demonstrator (I)

Goals

Performance

- optimized many particle treatment
- optimized 1-particle functions
- optimized base types / containers

Vectorization is **not limited** to many-particle case

Potential for SIMD optimization even in existing scalar algorithms and base operations

Approach

SIMD

algo + class review

template techniques

- template class specialization / code generation

Implementation

Vc library

Example: Review of 1-particle base classes

3D linear algebra is foundation layer of many simulation/geometry tasks

- particle transport (vector + vector), coordinate transformations (Rotation3D x vector), aggregating transformation (Rotation3D x Rotation3D)
- current library implementations do not support internal vectorization of such operations for single particles (apart from BlazeLib)

Example: Review of 1-particle base classes

3D linear algebra is foundation layer of many simulation/geometry tasks

- particle transport (vector + vector), coordinate transformations (Rotation3D x vector), aggregating transformation (Rotation3D x Rotation3D)
- current library implementations do not support internal vectorization of such operations for single particles (apart from BlazeLib)

Now provided specialized classes that achieve this (using Vc SIMD abstraction)

- can choose optimal memory layout for our use case
- know how matrix going to be used (no need for efficient inverse for instance)
- some memory padding ...

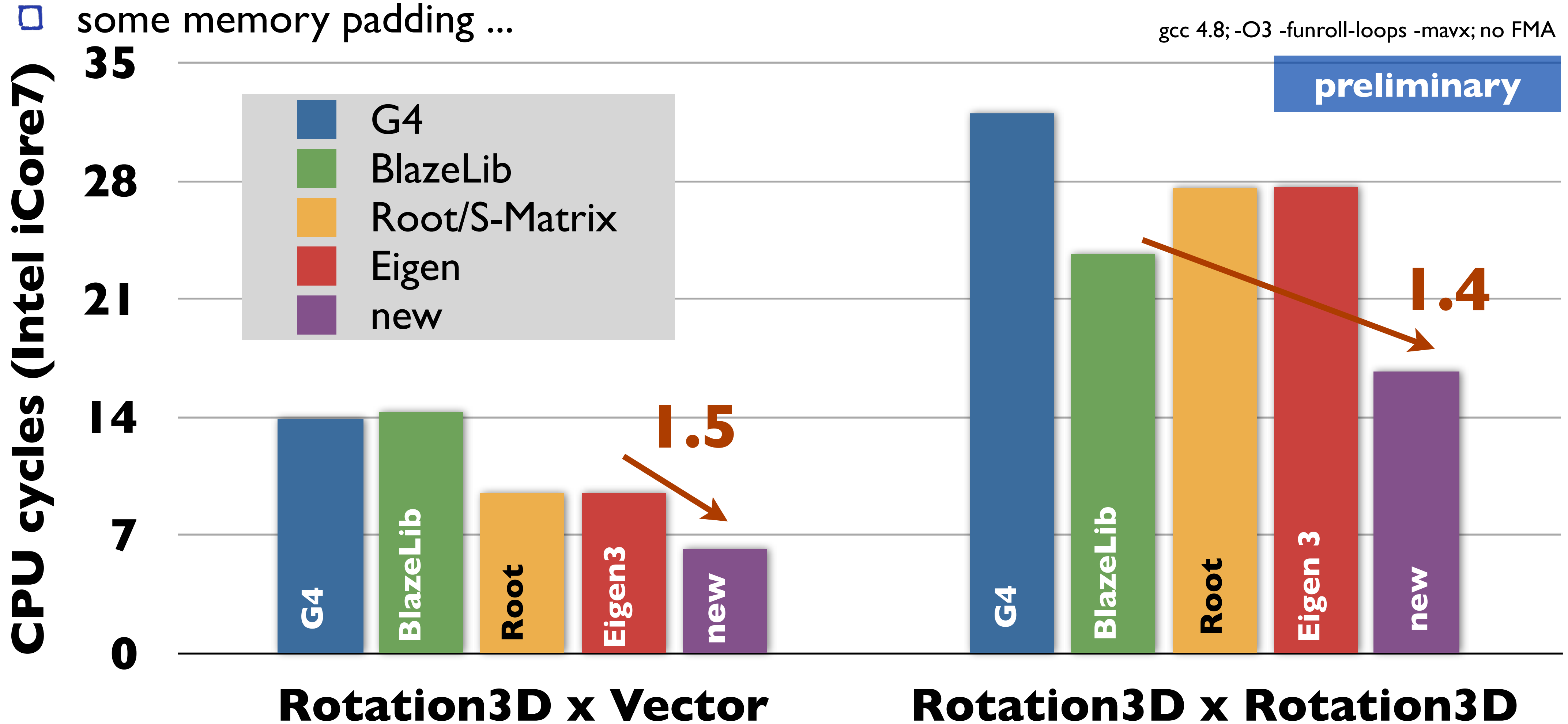
Example: Review of 1-particle base classes

3D linear algebra is foundation layer of many simulation/geometry tasks

- particle transport (vector + vector), coordinate transformations (Rotation3D x vector), aggregating transformation (Rotation3D x Rotation3D)
- current library implementations do not support internal vectorization of such operations for single particles (apart from BlazeLib)

Now provided specialized classes that achieve this (using Vc SIMD abstraction)

- can choose optimal memory layout for our use case
- know how matrix going to be used (no need for efficient inverse for instance)
- some memory padding ...



with Georgios Bitzes (CERN Openlab), Raman Sehgal (BARC, India)

Beyond the demonstrator (II)

Goals

Performance

- optimized many particle treatment
- optimized 1-particle functions
- optimized base types / containers

further wishes:

- try to target GPU
- avoid code duplication for scalar, vector, GPU
- do not depend on a concrete SIMD vectorization technology/library

Approach

SIMD

algo + class review

template techniques

- template class specialization / code generation

Implementation

Vc library

Where we'd like to go

Goals

Performance

- optimized many particle treatment
- optimized 1-particle functions
- optimized base types / containers

Abstraction

- SIMD abstraction
- CPU/GPU abstraction

Code reuse

- reusable components
- same code base for CPU/GPU where appropriate

Approach

SIMD

algo + class review

template techniques

- template class specialization / code generation
- generic programming

Implementation

Vc library | Cilk Plus | autovectorization | ....?

Summary

Part I: promising SIMD results in geometry demonstrator

Part II: we are ready to go beyond the demonstrator and tackle a generic high performance library for detector geometry

Summary

Part I: promising SIMD results in geometry demonstrator

Part II: we are ready to go beyond the demonstrator and tackle a generic high performance library for detector geometry

Extensions:

Extension I: team up with AIDA USolids effort

Extension II: use similar concepts in other simulation areas (physics)

Extensions III: work together in revision of fundamental base classes (Vector3D, Rotation3D); contribute them to core math libraries (ROOT) for common use