How can HEP applications benefit from vectorization?

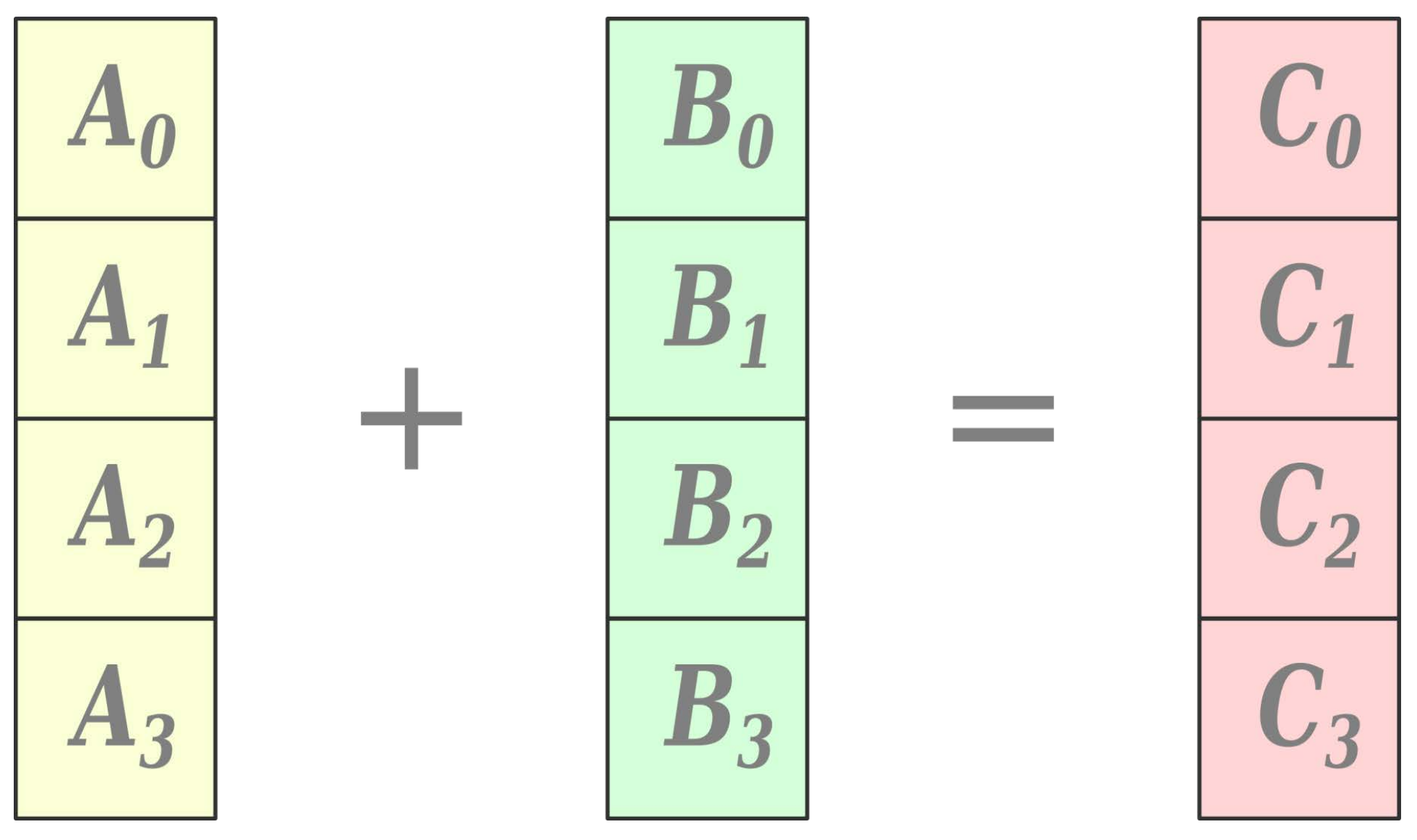
an example from the Geant-V prototype

by Georgios Bitzes
georgios.bitzes@cern.ch

May 2014

What is vectorization?

All modern commodity processors support performing basic operations on vectors, as opposed to single elements. As an example on 256-bit AVX, 4 64-bit double additions can be performed in parallel using a single instruction – no need to loop through all elements and add them individually.

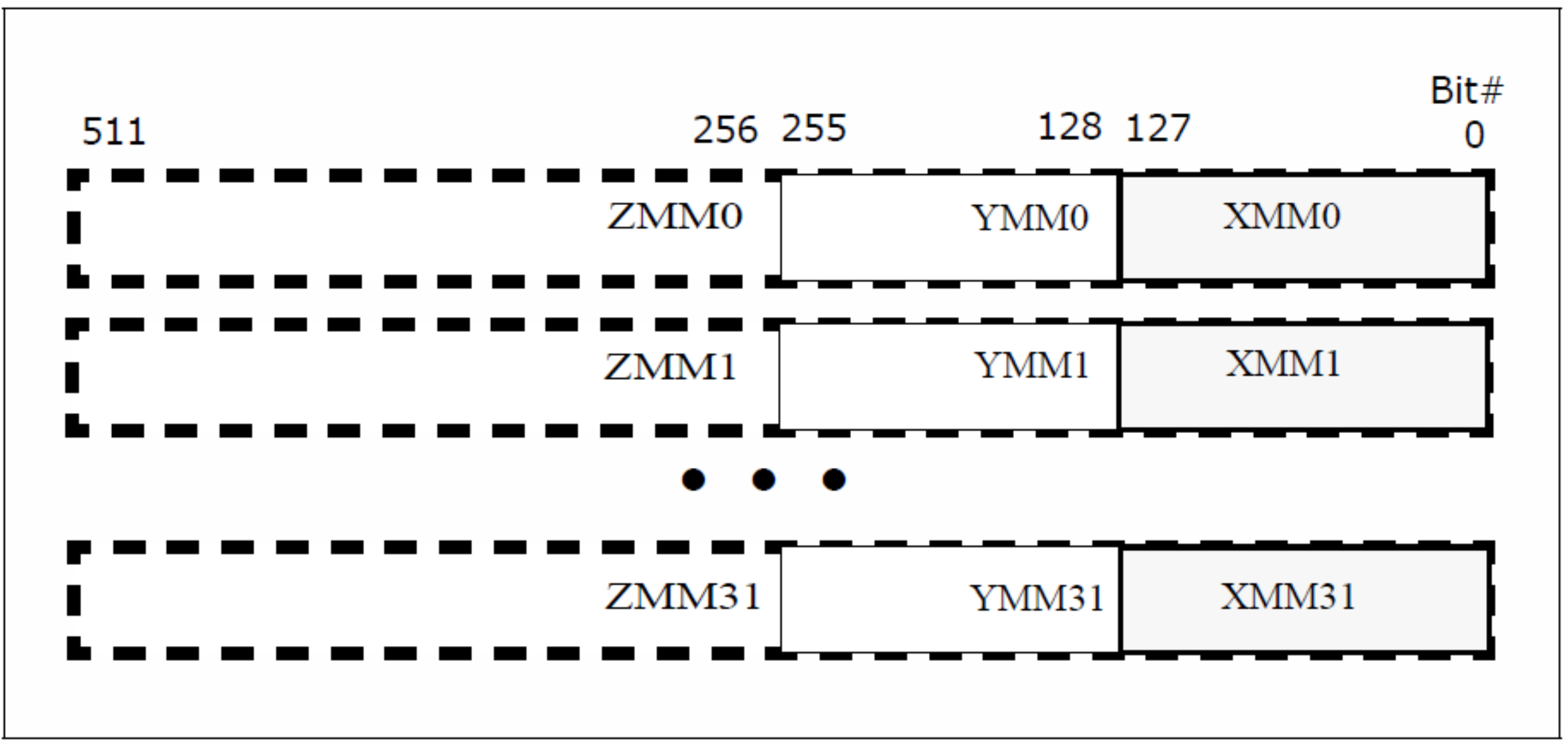


The ever-increasing vector size

The introduction of AVX in Intel "Sandy Bridge" has brought a vector size of ***256 bits***, soon to increase to ***512 bits*** in a future Xeon microarchitecture.

This will effectively double the number of elements we can process in parallel. What used to be a 2x – 4x potential performance gap will soon grow to 8x – the benefits of vectorization are becoming harder and harder to ignore!

The following picture shows how the size of the vector registers is evolving over time.



How to use?

Currently there are a few appealing options:

- Letting the compiler autovectorize the loops in the code
- Using Intel® Cilk™ Plus language extensions
- Using an external SIMD library such as Vc, VectorType or Boost::SIMD

Simple loops are easily vectorizable...

```
for(int i = 0; i < N; i++) {
    c[i] = a[i] + b[i]
}
```

...but the compiler could have trouble with more complicated ones, in which case an external library might be preferable.

Cilk+ introduces new array notations which extend the syntax of C/C++, helping the compiler figure out how to vectorize even in more complicated cases.

```
c[0:N] = a[0:N] + b[0:N]
```

Particle simulation

Whether it's predicting how a new proposed particle detector will behave, calculating cosmic ray induced doses for electronics used in space missions, or estimating radiation doses of cancer treatments, particle simulation is an invaluable tool for many fields in science.

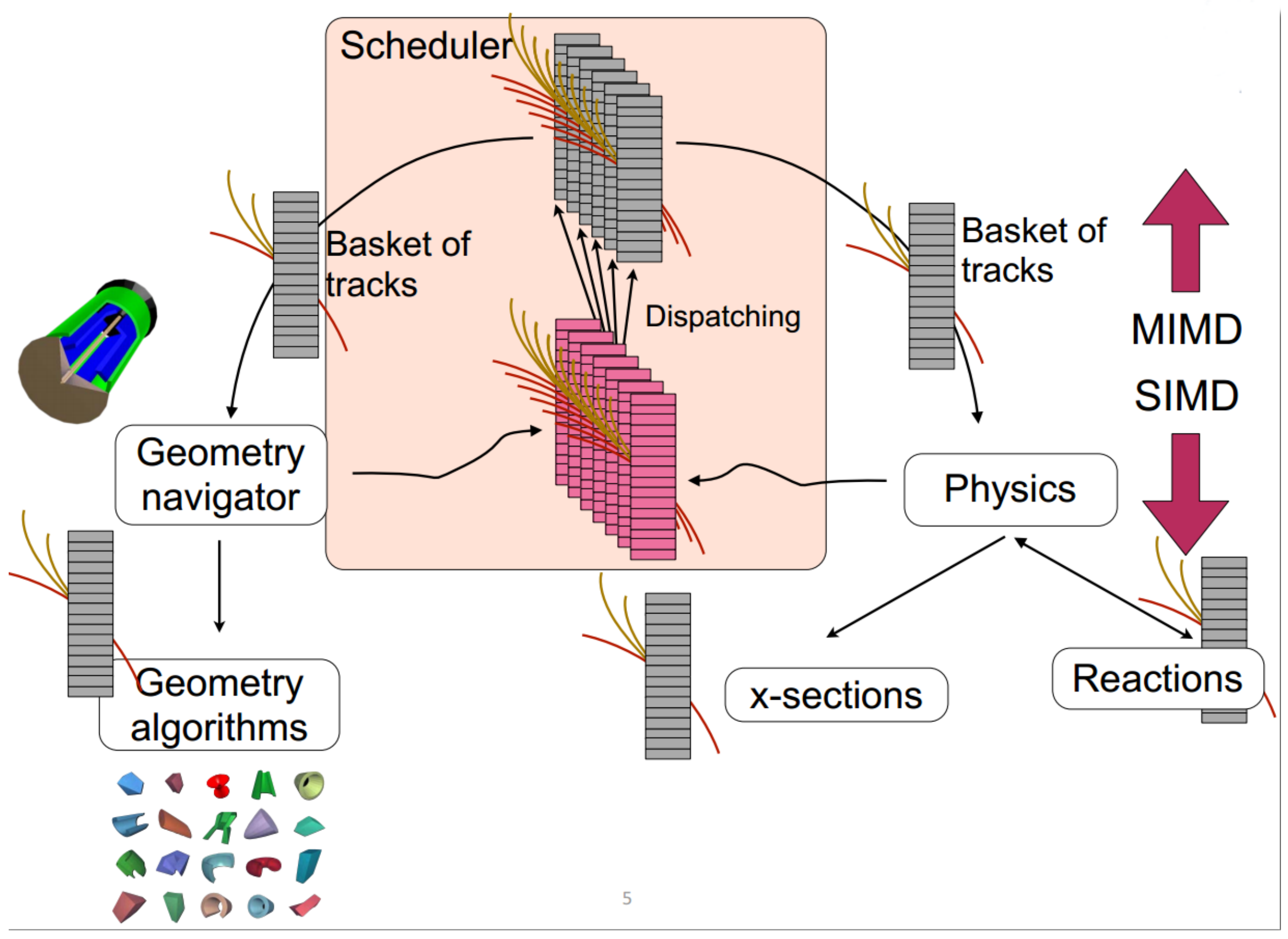
The CPU-intensive nature of this process has triggered the search for increased efficiency – current state of the art software is multi-threaded but not yet vectorized.

Geant-V

A new particle simulation toolkit being built

The goal is to build a high-performance toolkit that exploits all the capabilities a modern processor has to offer – including high core counts and vectorization.

Particles residing in the same logical volume are grouped together into baskets, which are then processed in parallel.



Performance gains

Our implementation with SIMD has showed good performance gains in the geometry navigation. The benchmark is based on a toy detector using 4 boxes, 3 tubes and 2 cones and is in comparison to ROOT/5.34.17. Double floating-point precision is used. Further improvements except from vectorization (caching, template specialization) lets us reach speedups exceeding the maximum predicted from SIMD alone, in some cases.

	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™	~4.1x	~4.8x	8x

Participants

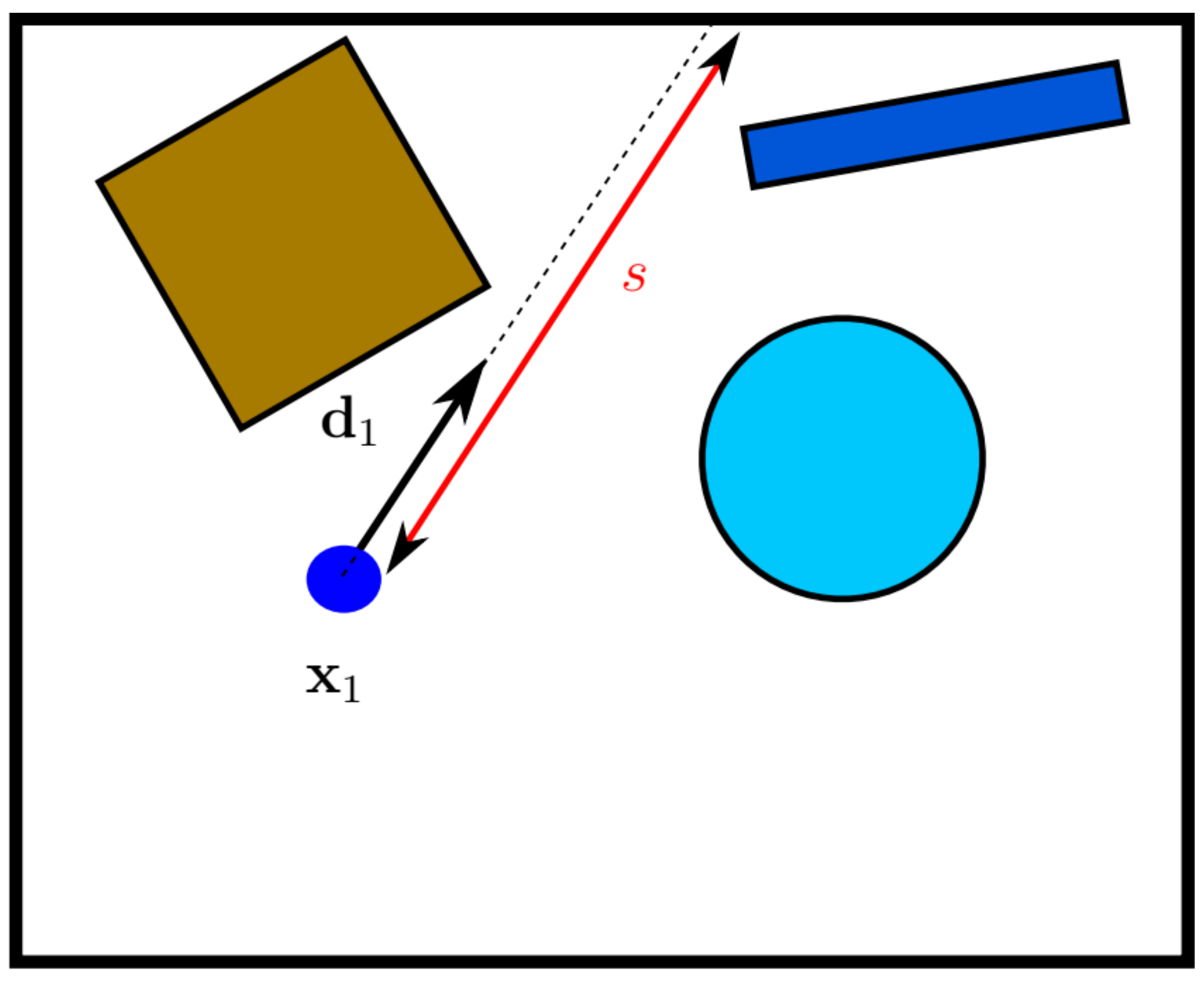
Geant-V is a broad collaboration with participation from CERN, Fermilab, University of Catania, BARC, India, CERN openlab and Intel.

Some material in this poster courtesy of Sandro Wenzel (CERN).

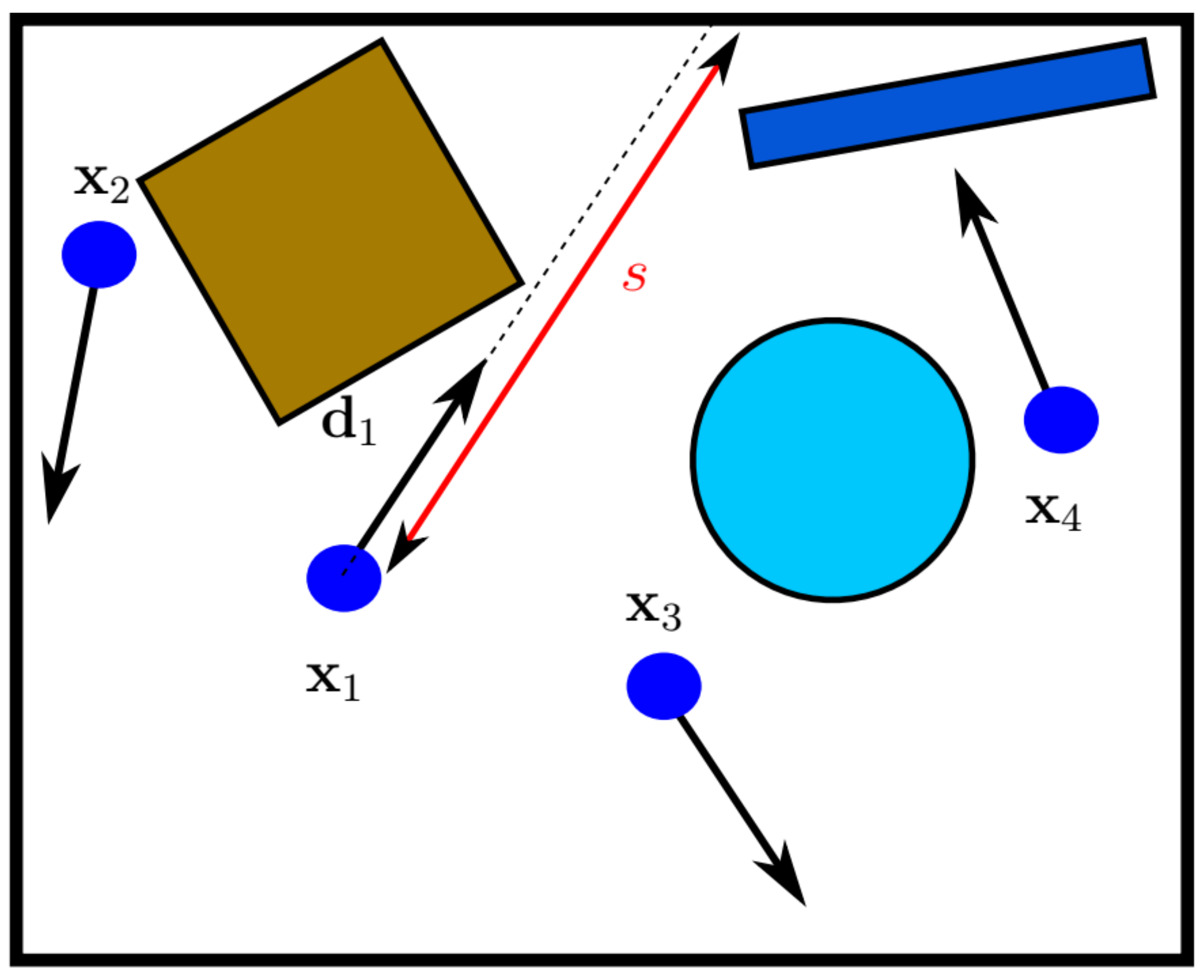
Vectorized geometry transport

A large percentage of CPU time is spent transporting particles, calculating their trajectory in the presence of a magnetic field and deciding which parts of the detector are hit.

A typical task is to find the next hitting boundary and get distance to it – using vectorized geometry kernels we can perform this operation for multiple particles at a time.



1 particle



vectors of particles

Challenges in the geometry

Computational geometry is notorious for having many special cases – different arithmetic calculations depending on the positions of the shapes and the particles.

As an example, a particle hitting the top face of a tube requires different treatment than a particle hitting the main cylinder sideways.

This poses a challenge in applying vectorization – not all particles in a vector will fall under the same case! We need to use vector masks to track which calculations each particle requires.