CPU Performance of ALICE Simulation: Current Activities and Look Into Future

Sandro Wenzel (ALICE - CERN)

ALICE OFFLINE WEEK - 3 November 2016

Outline

- **Part I: Status update on simulation CPU performance**
  - Report on recent optimization activities + results
  - Pointers to hotspots to work on in future

- **Part II: Opportunities from VecGeom**
  - Short intro to VecGeom library
  - Performance boost from VecGeom shape primitives
  - Performance boost from VecGeom navigation

Motivation

- **Monte Carlo** (simulation/digitization + reconstruction) uses **major fraction of our grid resources** (ca. 70% of total CPU hours)
  - 3/4 simulation/digitisation
  - 1/4 reconstruction
- **Natural desire to make software faster**
  - faster throughput, shorter turnaround times
  - preparation to handle higher-luminosity problems
- **A dedicated campaign was started March 2016 to assess the current situation and address the most easily accessible problems**
- **Start with simulation/digitization part; Tackle reconstruction next**

Approach

- Revision of compiler options for some recipes
- Profile Pb-Pb benchmark simulation
  - valgrind/callgrind + igprof/VTune
- Identify and fix “low-hanging fruit” hotspots in code for now
- Attack actual algorithms in a later optimization pass

CPU-Usage map (callgrind) : Identifying problems

CPU-Usage map (callgrind) : Identifying problems

CPU-Usage map (callgrind) : Identifying problems

CPU-Usage map (callgrind) : Identifying problems

CPU-Usage map (callgrind) : Identifying problems

CPU-Usage map (callgrind) : Identifying problems

The Main Message: 20% Gain

- **~20% gain in total runtime achieved (from ~2359s to ~1966s) until now**
  - for Pb-Pb benchmark running with Geant3 (3 events; Intel(R)-Core(TM) i7-5930K; CentOs7)
  - Gains from pure simulation part + digitization
- **Activity ongoing … more gains to be expected**

	Original	Tuned compiler flags	Code optimizations in AliRoot/ROOT
RunSimulation	1462s	1367s	1182s
RunSDigitization	683s	692s	585s
Total	2359s	2274s	1966s

Actions taken in AliRoot/ROOT

- Replace time-critical ***dynamic_casting*** with ***static_casts***
- Optimized element access to ***TMatrixT***
- Optimized access to other ROOT containers (such as ***TArrayI***, …)
- Optimize ***TLorentzVector*** class which had non-optimal constructors and element access (patch accepted by ROOT)
- **Inlining (+de-virtualization) campaign** for smaller functions called very often
- Provide an optimized sort algorithm on ***TClonesArrays*** using templates
- Avoid repeated access to thread-local variables by caching a reference
- … (see commit list on backup slide)

dynamic-cast "problem"

- There is an overuse of expensive dynamic casting in AliRoot
- Origin probably related to non-type strictness/awareness of ROOT containers (wouldn't exist with STL containers)
- Problematic pattern in AliROOT:

```
TObjArray *fModules; // holds elements of type AliModule

AliModule *module;
if (module = dynamic_cast<AliModule*>(fModules[i]){
  module->PreTrack();
}
```

Declaration of an essentially read-only container

Repeated type-checking at every Monte-Carlo step !

- Use of **static_casts** is often ok (may do dynamic type checking when writing to container)

TMatrix element access

- TMatrixT class used heavily in inner loops of (TPC) digitization
- ROOT does **not provide a fast way** to fetch an element from a TMatrixT
  - unavoidable explicit bound check at each access
  - unavoidable internal assert (even in release mode)
  - access operator not inline

```
TMatrixF signal;
element = signal(i,j);
```

TMatrix element access

- TMatrixT class used heavily in inner loops of (TPC) digitization
- ROOT does **not provide a fast way** to fetch an element from a TMatrixT
  - unavoidable explicit bound check at each access
  - unavoidable internal assert (even in release mode)
  - access operator not inline

```
TMatrixF signal;
element = signal(i,j);
```

- A fast accessor function was implemented in AliRoot to circumvent this problem (temporarily)

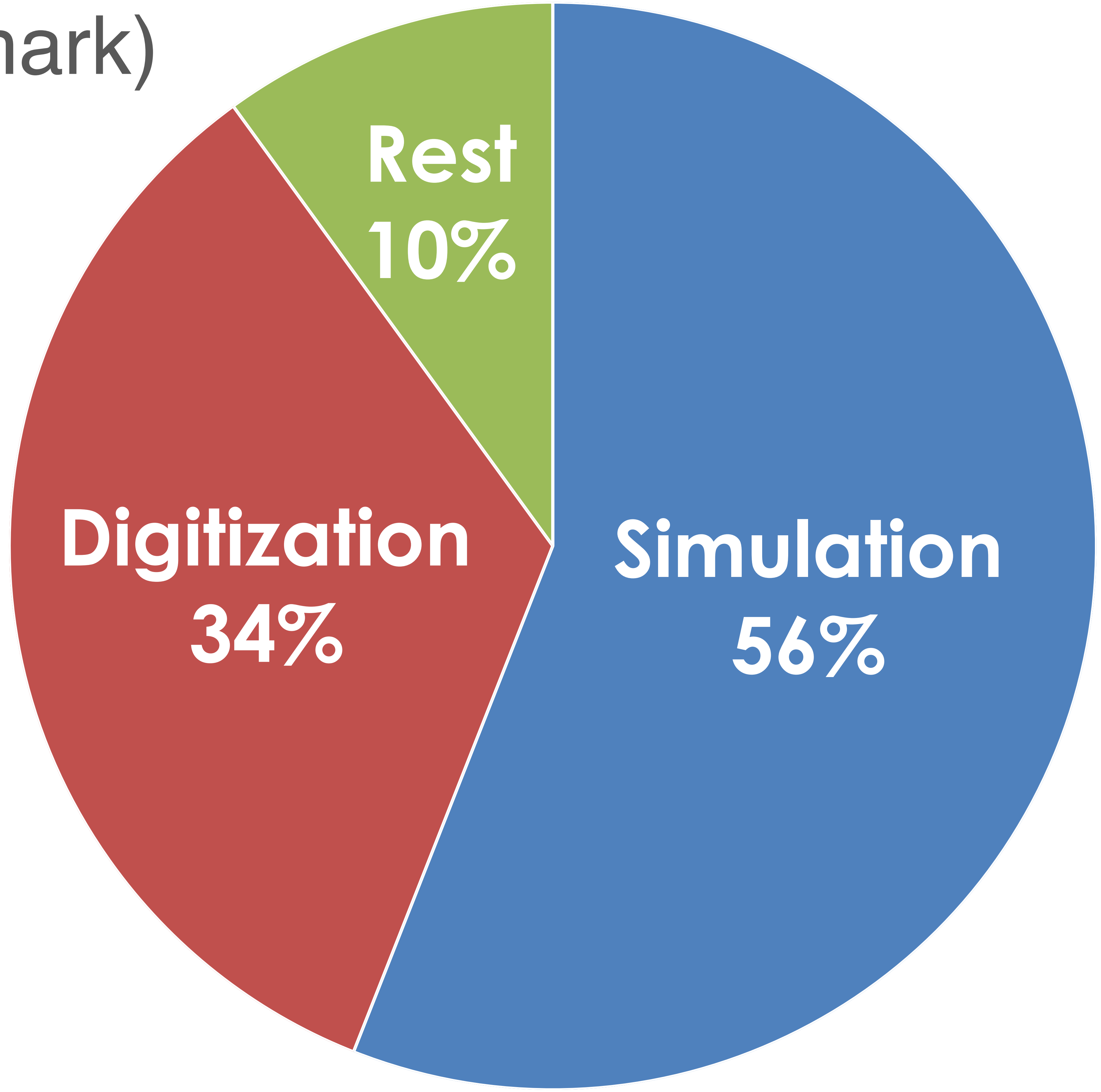
```
TMatrixF signal;
element = TMatrixFastAt<Float_t>(signal,i,j);
```

- Similar problems exist for other “deprecated” ROOT containers (TArrayI, …)
- Should consider modernizing our types…

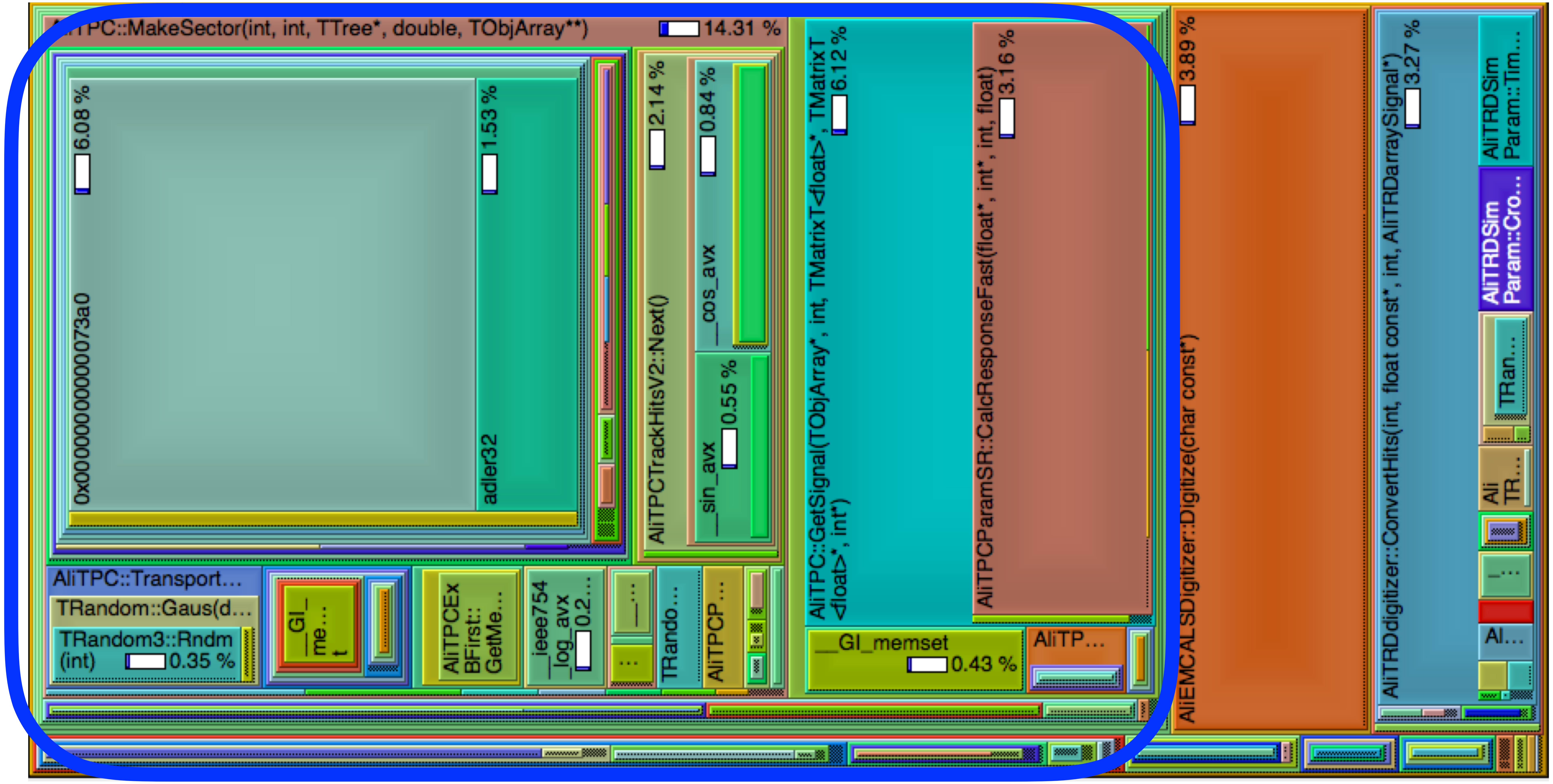
JIRA ROOT-5472 opened a while ago

Updated Situation

- With some low-hanging fruits grabbed … take another view on the real algorithmic hotspots
- Current high-level situation (from G3 Pb-Pb benchmark)
- Most important “components”:
  - Simulation ~56%
  - Digitization ~34%
  - Rest (HLT; QA) ~10%

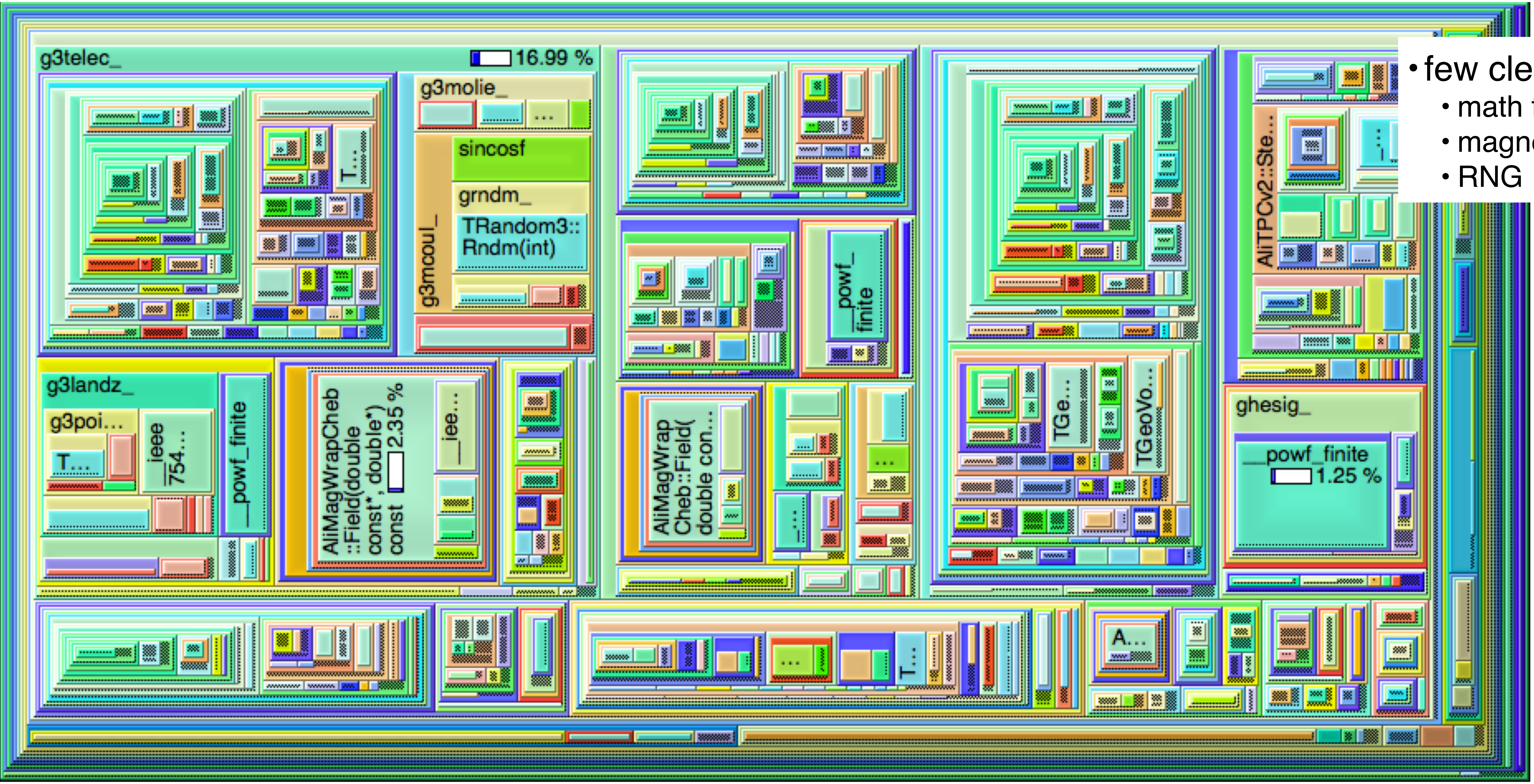


Digitization: Next Challenges/Opportunities?



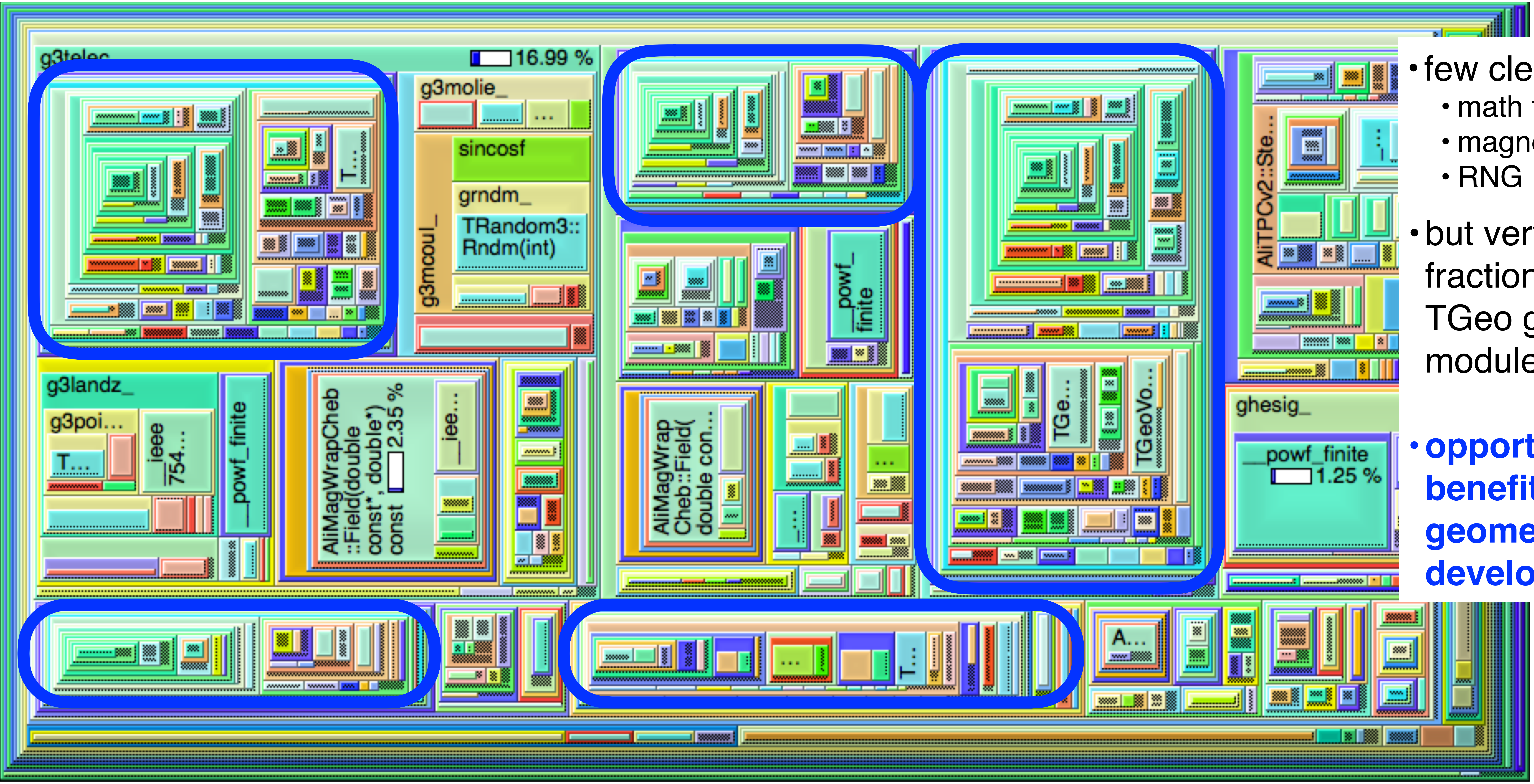
- clearly visible hotspots
- largest component is TPC digitization (blue frame)
- very large fraction inside TPC due to ROOT TTree reading
- considerable improvement potential! (IO problem; SIMD processing ?)

Simulation: Next Opportunities?



- few clear hotspots
  - math functions
  - magnetic field
  - RNG

Simulation: Next Opportunities?



- few clear hotspots
  - math functions
  - magnetic field
  - RNG
- but very large fraction (40%) in TGeo geometry module
- **opportunity to benefit from geometry library developments**

Part II: Opportunities from VecGeom

gitlab.cern.ch/VecGeom/VecGeom

Selection from recent material presented at

@Geant4 collaboration workshop

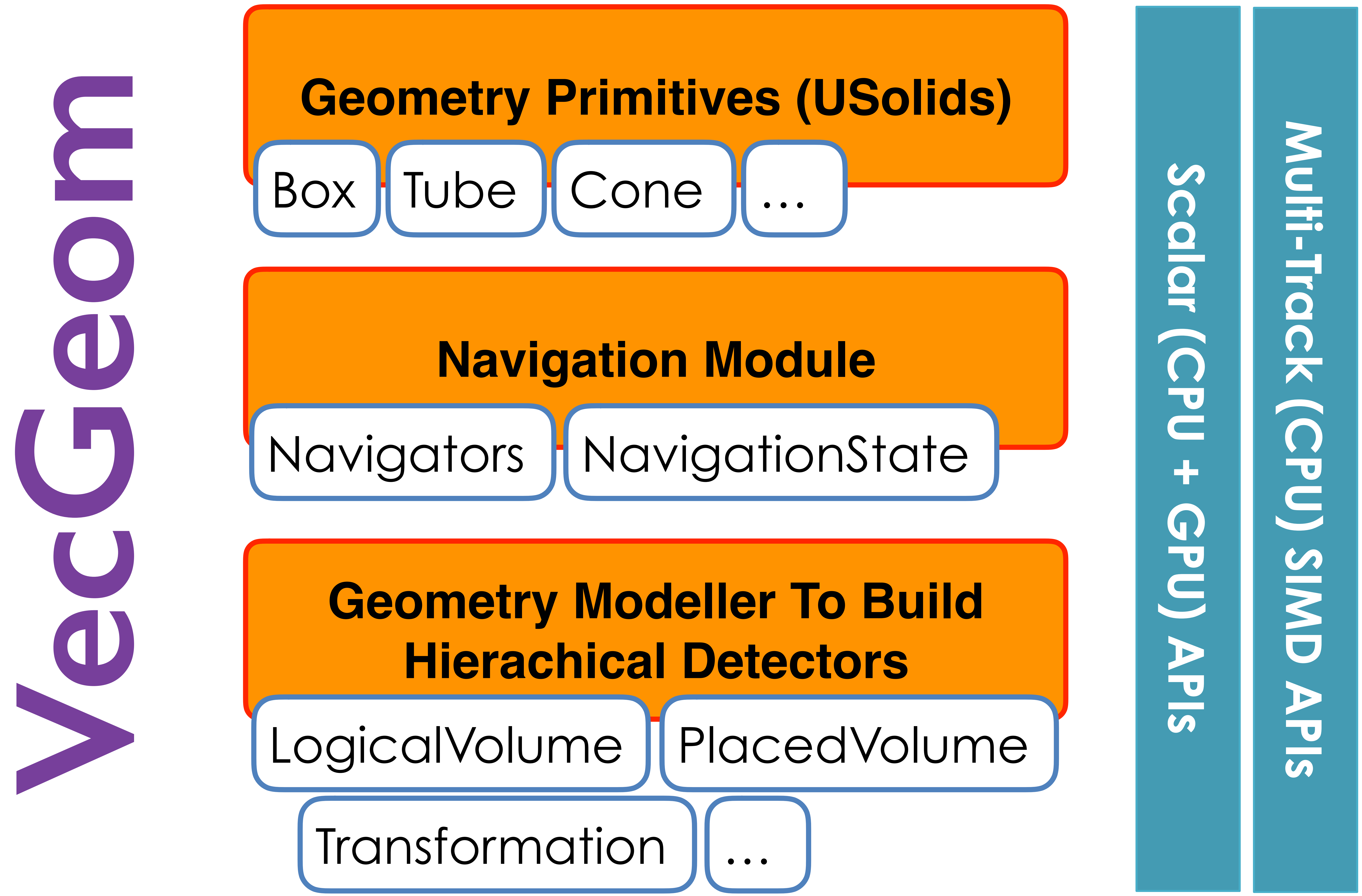
@CHEP16

@GeantV HSF "community meeting"

Intro to VecGeom

- **VecGeom is a geometry modeller like TGeo**
- **Originally founded to satisfy additional needs of GeantV project …**
  - designed for any heavily multi-threaded frameworks, allowing for rapid track/context switches
  - able to handle groups (baskets) of tracks in all algorithmic parts (basket API)
- **Target Performance + SIMD acceleration in all aspects**
  - Review / Refactoring / Modernizing / Extension of existing algorithms
- **Not bound to GeantV !! VecGeom could be used as the geometry component by production simulation frameworks (Geant4, Geant3, …)**

VecGeom: Component Overview



Shape-Primitives Status: The ALICE Use-Case

- VecGeom now has **all shape-primitives** to satisfy needs of most HEP experiments (Xtruded added recently)
- For ALICE simulations (Pb-Pb), demonstrate that VecGeom offers very **significant performance** gains for **the most CPU relevant shape-primitives** even in scalar track mode

Primitive	Safety	Dist2In	Dist2Out	Contains	CPU% Sum
Pgon	2.05	2.52	0.18	1.18	**5.93**
Xtru	0.56	0.68	0.20	1.81	**3.25**
Pcon	1.07	0.32	0.05	0.13	**1.57**

% of CPU cost of shape primitives (TGeo) in typical ALICE Pb-Pb simulation

Shape-Primitives Status: The ALICE Use-Case

- VecGeom now has **all shape-primitives** to satisfy needs of most HEP experiments (Xtruded added recently)
- For ALICE simulations (Pb-Pb), demonstrate that VecGeom offers very **significant performance** gains for **the most CPU relevant shape-primitives** even in scalar track mode

up to **9x faster** than existing code

Primitive	Safety	Dist2In	Dist2Out	Contains	CPU% Sum
Pgon	2.05	2.52	0.18	1.18	**5.93**
Xtru	0.56	0.68	0.20	1.81	**3.25**
Pcon	1.07	0.32	0.05	0.13	**1.57**

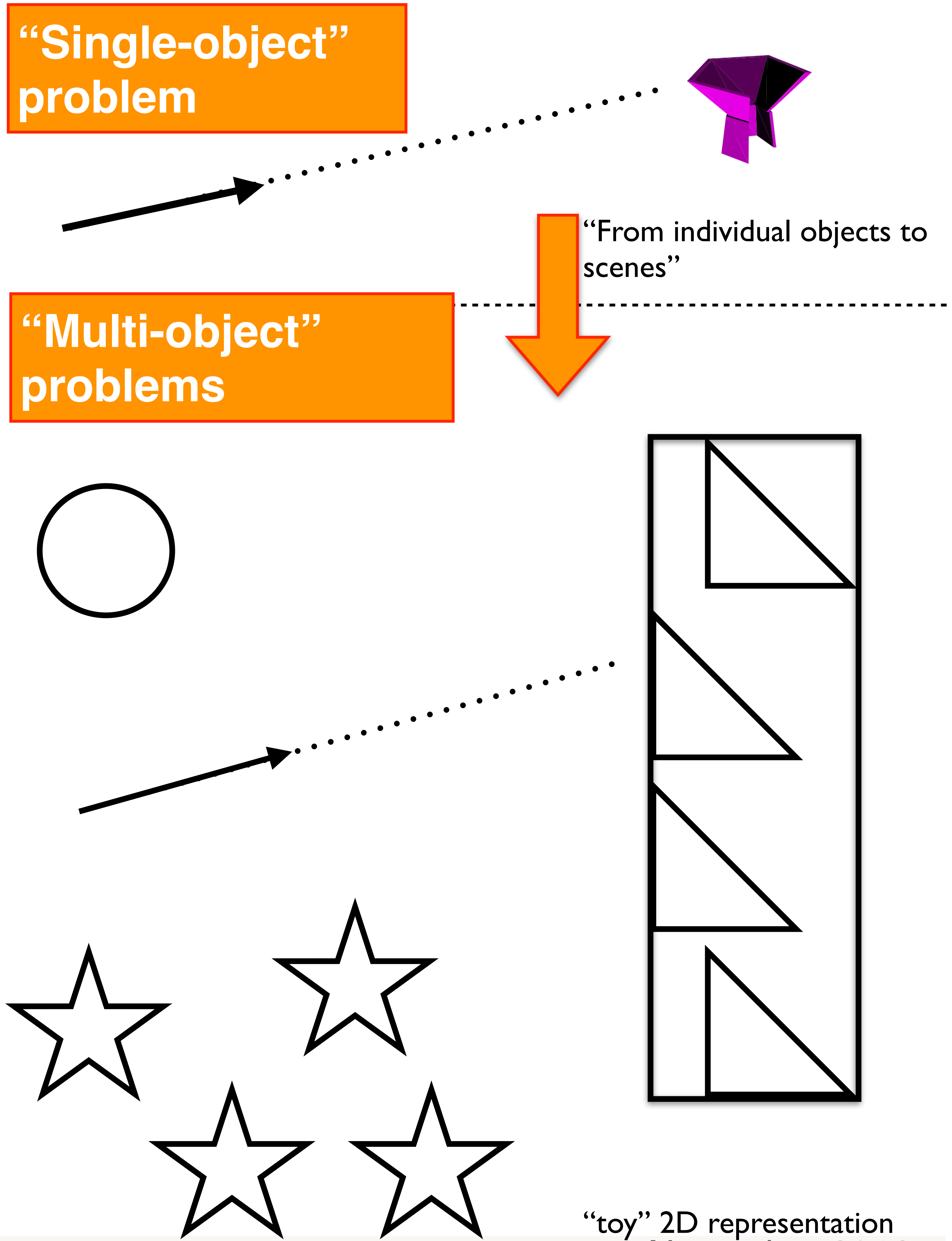
% of CPU cost of shape primitives (TGeo) in typical ALICE Pb-Pb simulation

- Depending on experiment, **a few % in CPU simulation cost gainable** by switching to VecGeom primitives (integration effort into G4/TGeo under way)

The Navigation Module

- Geometry primitives provide algorithms for simple ray - shape problems (focus on individual object)

- **Navigation module** provides **"multi-object"** algorithms:
  - provides next colliding object + distance in a "multi-object" scene
  - provide object after the next boundary crossing
  - highest level interface used in simulation (ALICE ~40% with TGeo, similar in CMS, …)

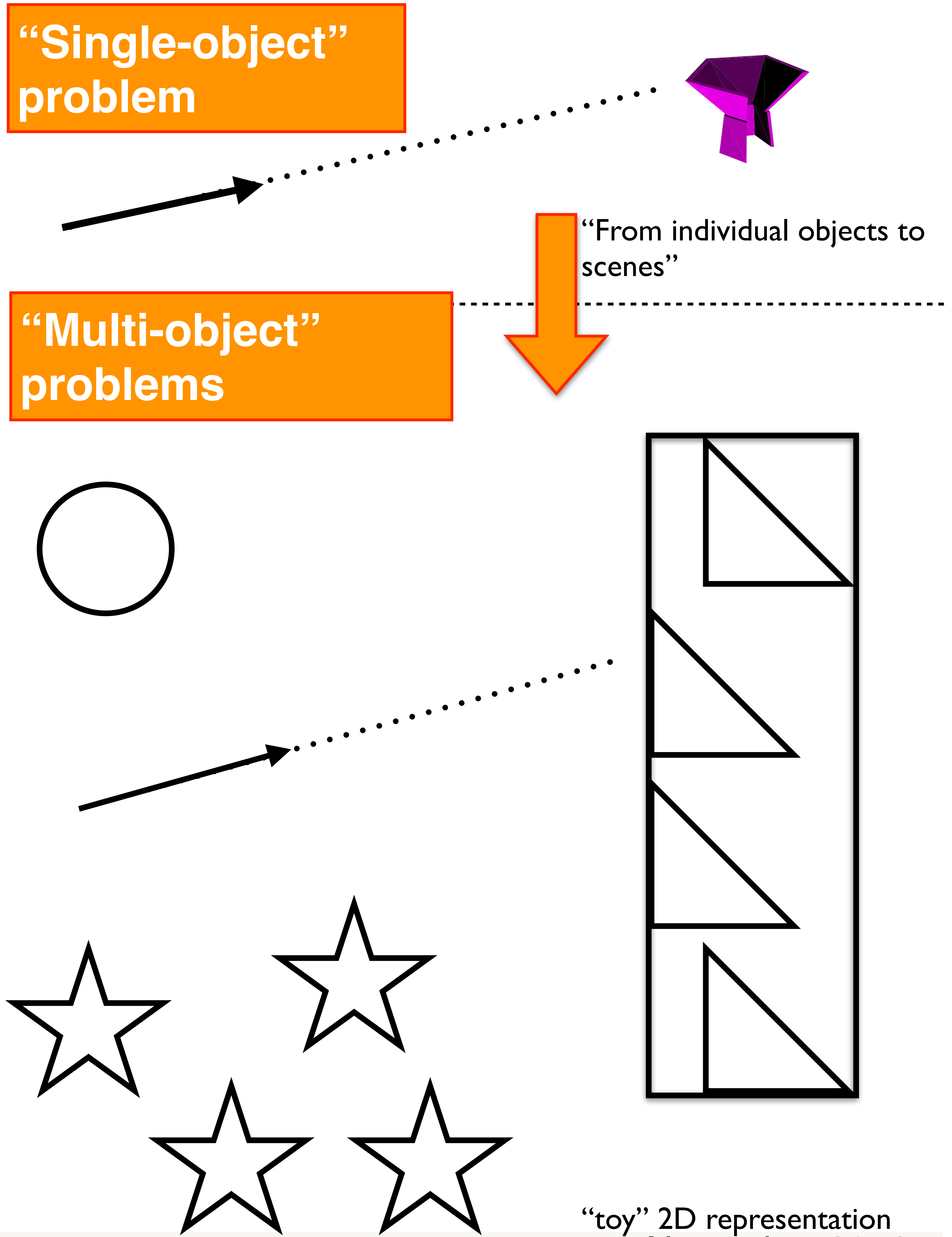


The Navigation Module

- Geometry primitives provide algorithms for simple ray - shape problems (focus on individual object)

- **Navigation module** provides “**multi-object**” algorithms:
  - provides next colliding object + distance in a “multi-object” scene
  - provide object after the next boundary crossing
  - highest level interface used in simulation (ALICE ~40% with TGeo, similar in CMS, …)

- **Recent Goals / Targets:**
  - Implement navigation system in VecGeom scaling to many particles and many threads
  - **Implement acceleration structures for fast candidate rule-out (scaling ~log(N) - see voxel techniques of G4/ TGeo)**
  - **Target explicit SIMD acceleration**



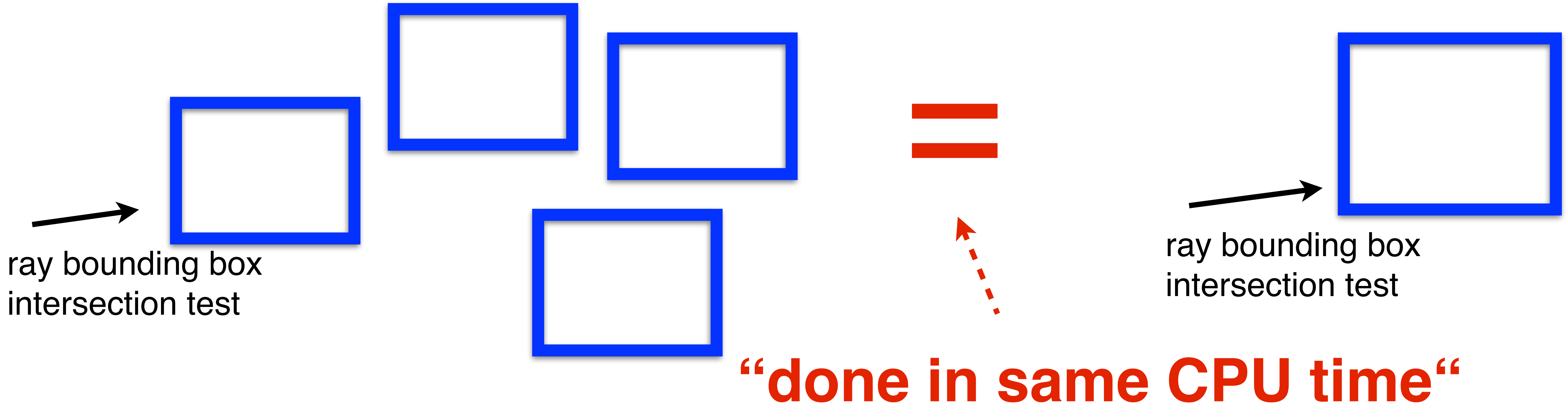
“toy” 2D representation

SIMD Acceleration of *Voxel* Navigation

- Canonical solution for fast hit-detection: tree structures, lookup structures, bounding boxes, …
- **How to combine this with SIMD paradigm?**

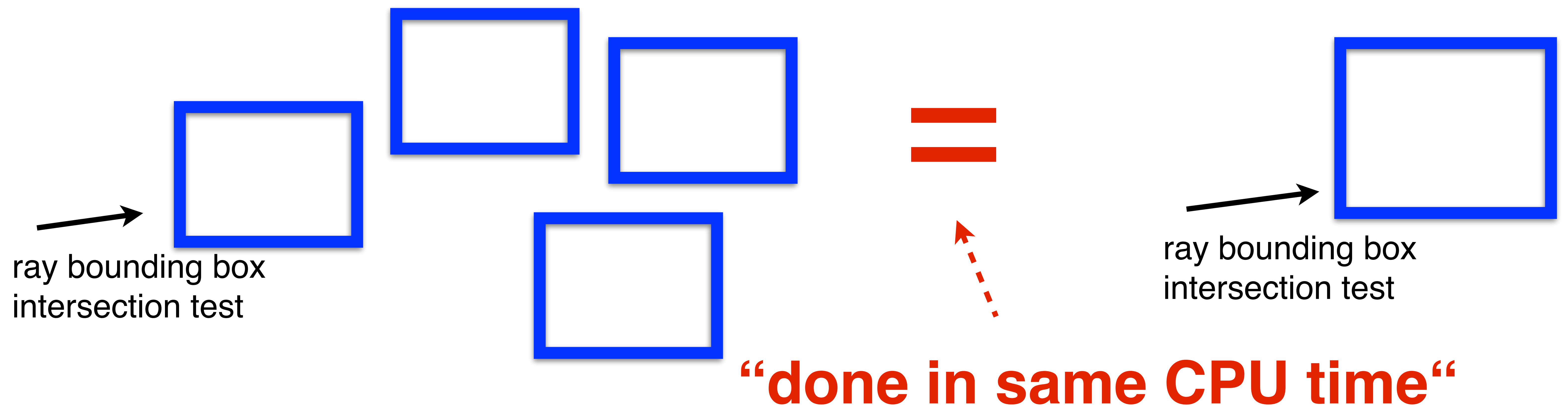
SIMD Acceleration of *Voxel* Navigation

- Canonical solution for fast hit-detection: tree structures, lookup structures, bounding boxes, …
- **How to combine this with SIMD paradigm?**
- Followed idea based on using (aligned) bounding boxes of geometry objects to filter good hit candidates



SIMD Acceleration of *Voxel* Navigation

- Canonical solution for fast hit-detection: tree structures, lookup structures, bounding boxes, …
- **How to combine this with SIMD paradigm?**
- Followed idea based on using (aligned) bounding boxes of geometry objects to filter good hit candidates



get **SIMD gain** from treating **group of boxes** in concert

get **scaling** from **hierarchies** of bounding box groups (forming **regular trees**)

Inspired from e.g.: [Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays]() + CPU ray-tracing libraries: Intel Embree, …

Regular Tree Building via Clusterization

Basic algorithm:

- let S == elements in SIMD register
- cluster objects into groups of S elements (we use a variation of k-means)
- identify bounding boxes of grouped objects as daughters of a tree node
- iterate this process

Algorithm illustrated here for SSE (= 2 double numbers per register)

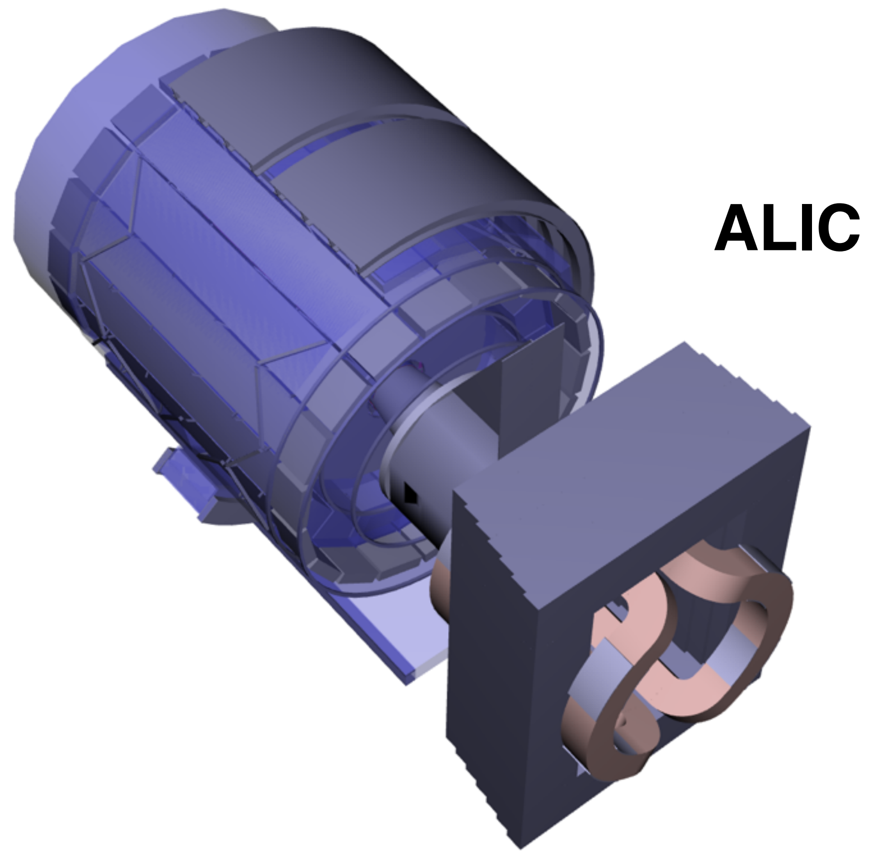
SIMD-Trees: Status + Local Benchmark

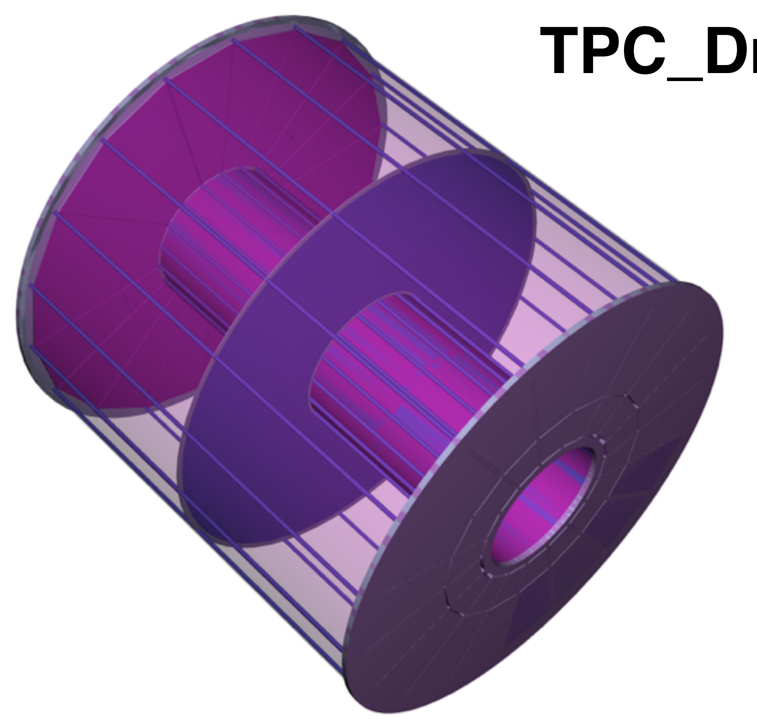
- Test approach on various detector volumes
  - most important complex volumes from ALICE: ALIC + TPC_Drift
  - a complex volume from CMS: MBWheel (~600 daughter volumes)
- Perform local navigation benchmark: One step + boundary crossing in the given volume for 0.5 million different tracks

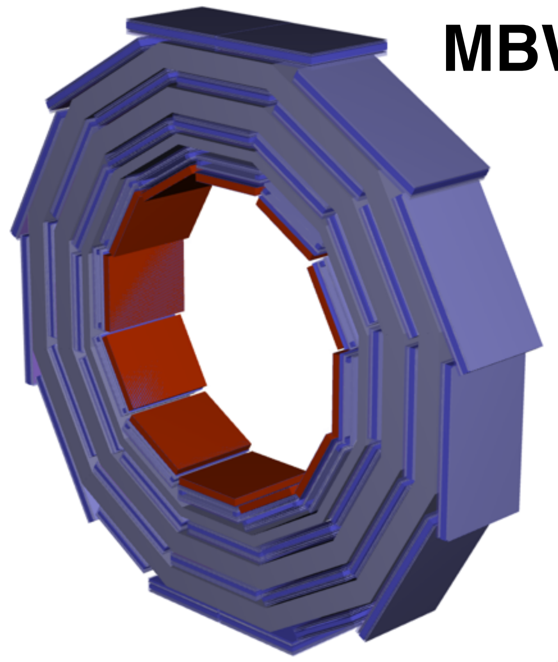
Volume	Daughters	G4	TGeo	VecGeom (SSE4.2)	VecGeom (AVX2)
ALIC (ALICE)	65	0.74	1.07	0.30	0.23
TPC_Drift (ALICE)	641	14	2.20	1.20	0.90
MBWheel (CMS)	~600	0.84	1.09	0.49	0.35

numbers are time in seconds; **worst is red**; **best is blue**

- Demonstrating **overall speedup >2** compared to existing solutions + **gain from SIMD unit** (see change SSE4.2 to AVX)



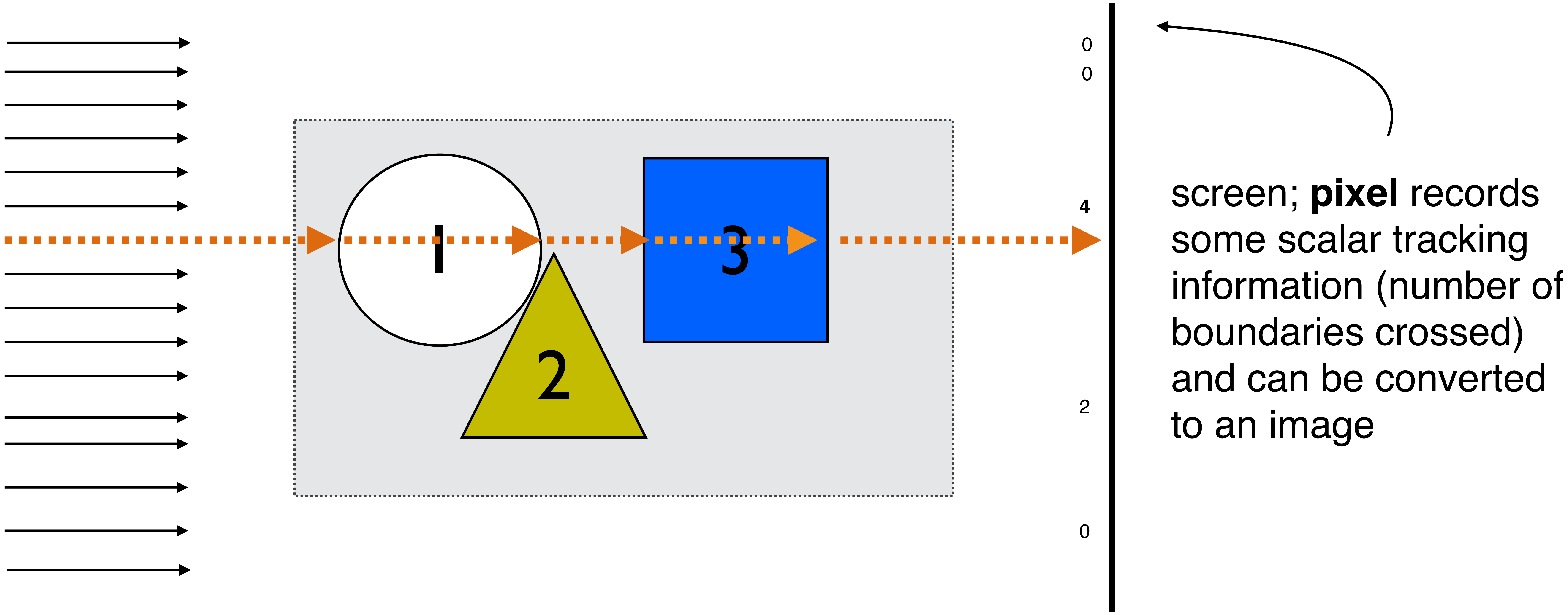




Test/Global Benchmark of VecGeom Navigation

- Evaluate VecGeom (solids + navigation) on complex modules for **multiple steps**

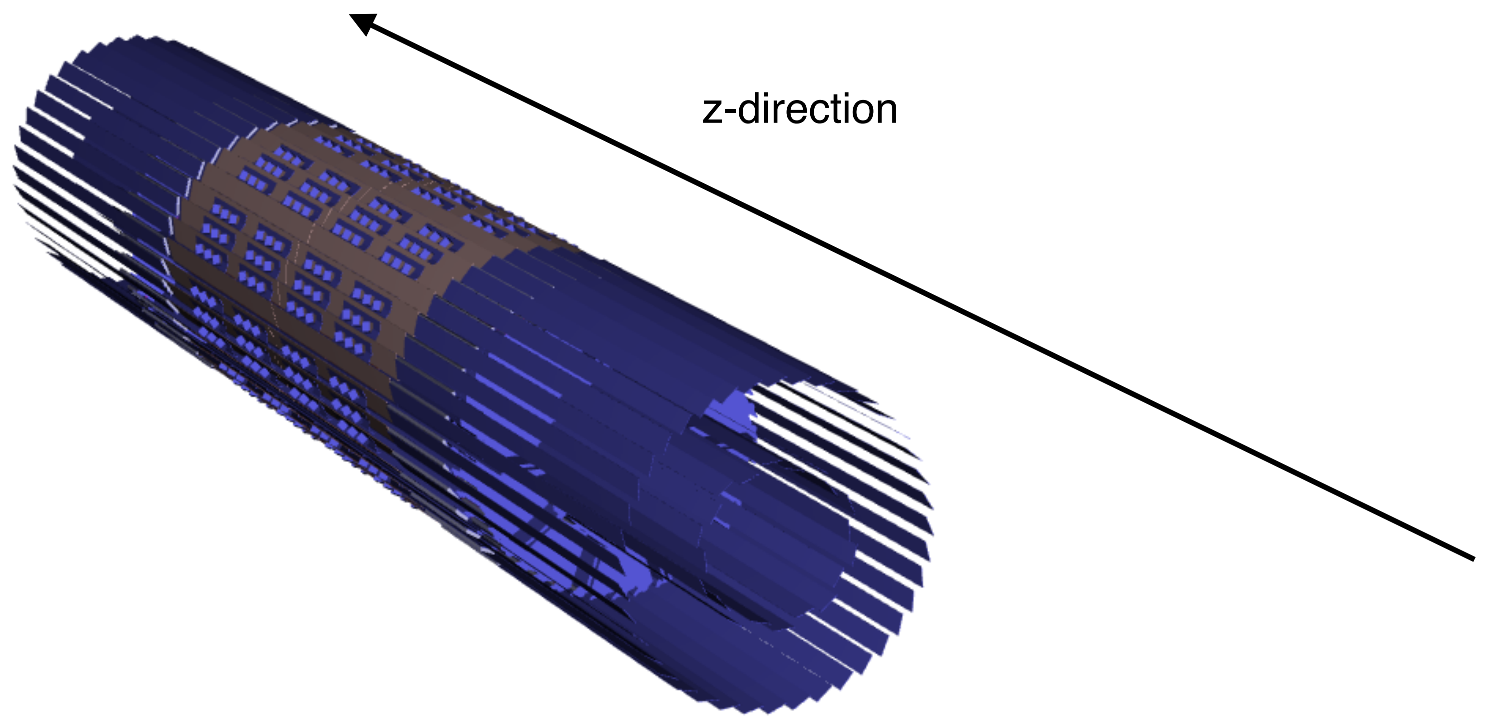
- **XRay-Benchmarker**:
  - follow geantinos through geometry pixel-by-pixel
  - record some information on screen



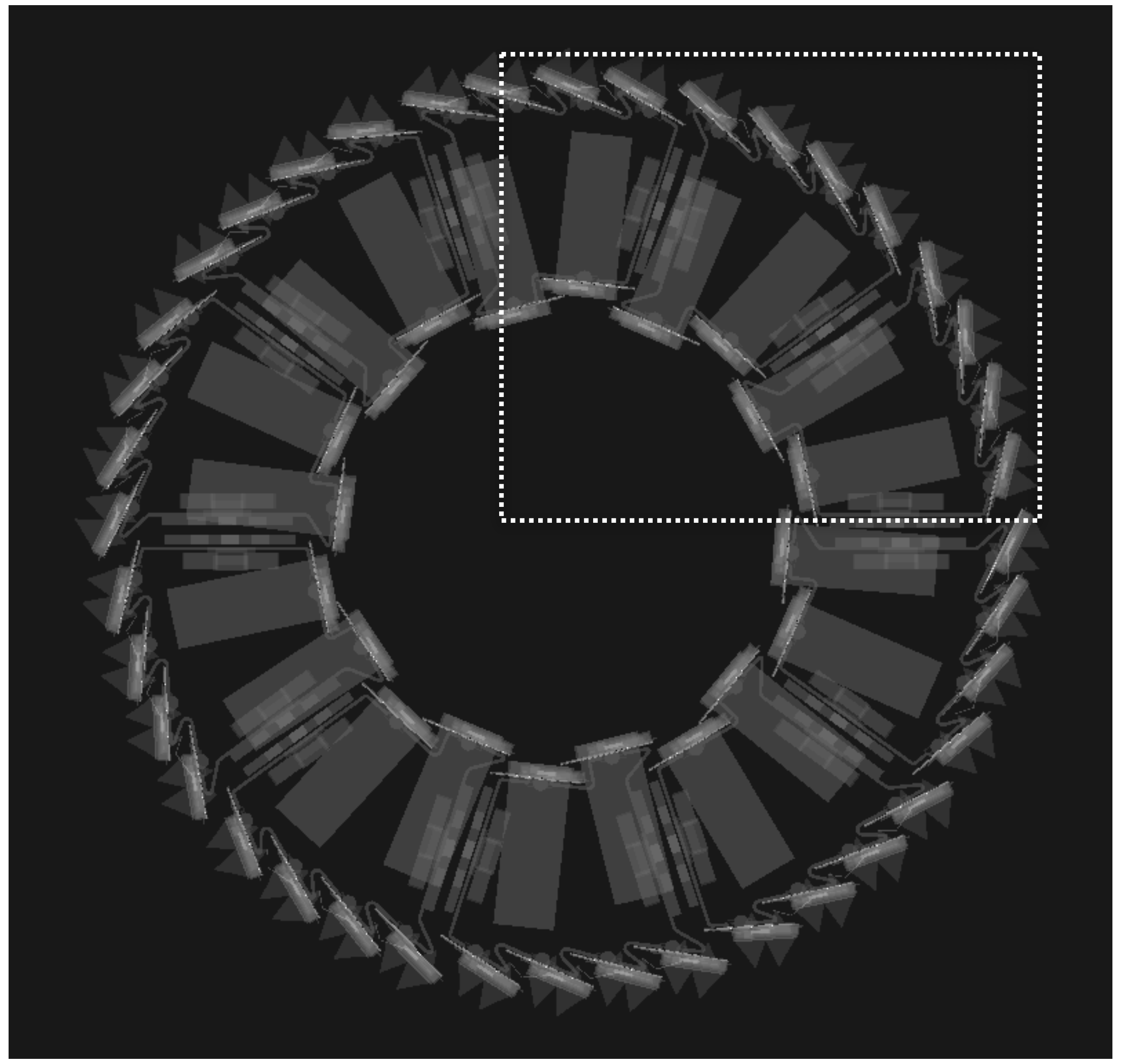
- **Perfect for validating** navigation algorithms (can do same with G4/TGeo)

- **Good to get a global idea of library performance**

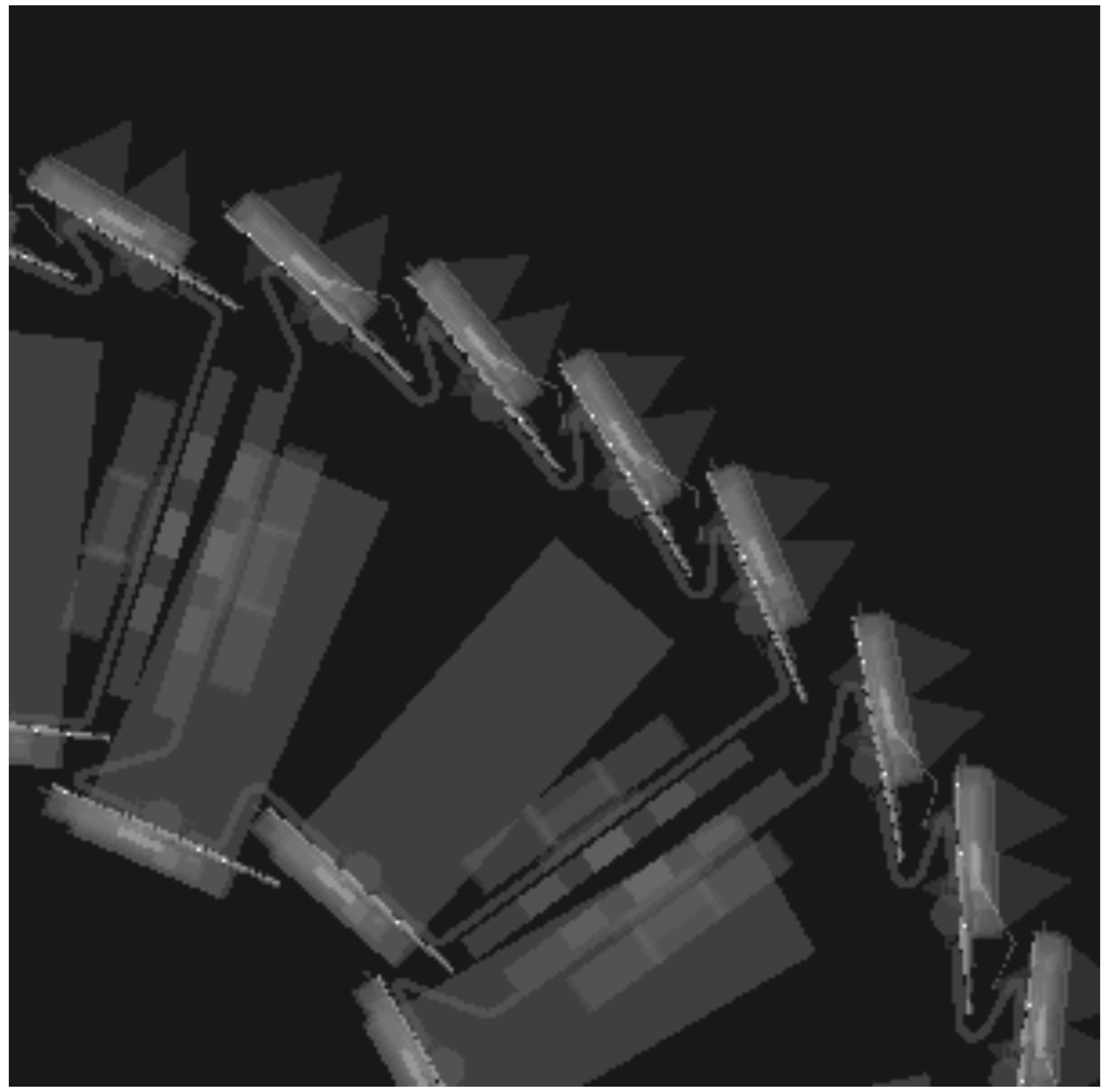
Geantino-XRay: Global Performance



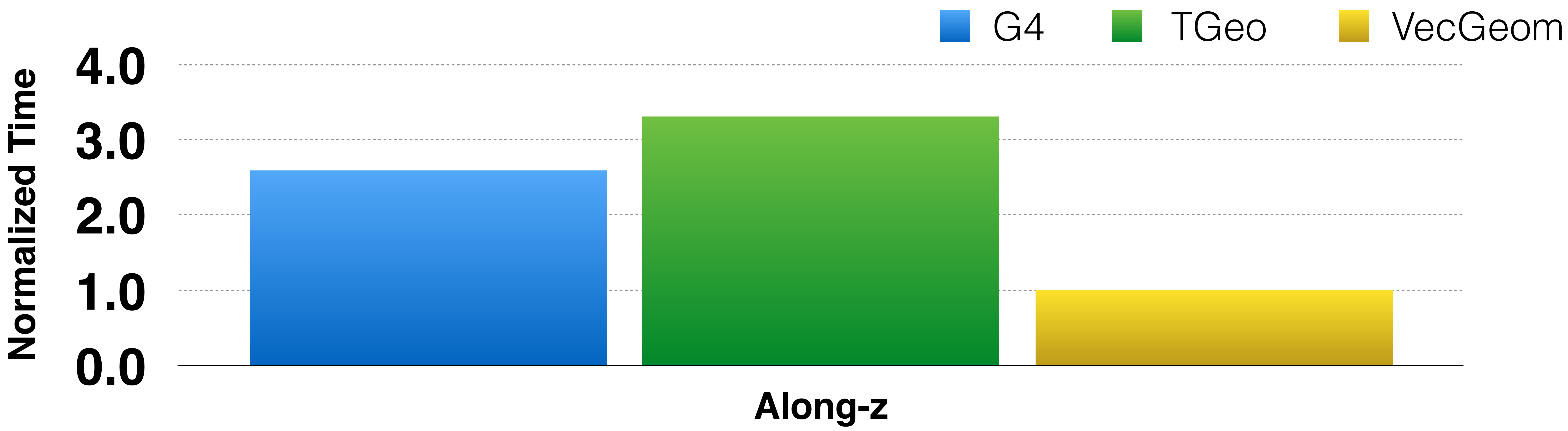
view along z-direction



zoom of white rectangle



- Example for ALICE ITSSPD
- Perfect agreement between G4/TGeo/VecGeom
- Observe generally **factors > 2.6x** speed improvement against other packages
- Another indication of global performance advantage of VecGeom



Summary VecGeom

- For ALICE, VecGeom is an alternative geometry library to TGeo with the potential to noticeably speedup simulation / reconstruction
  - faster geometry shape primitives
  - faster navigation algorithms
- “Estimated” speedup: ~3x over TGeo
- Yet to be integrated … (2 major options):
  - native connection to Geant3 / Geant4 (such as done with TGeo)
  - “hide behind” TGeo interface

Backup: List of optimization commits in AliRoot

1. ed8ba787b9d5ca75d3ba07bf5370ab4ec804147a
2. c972c5cc6fb40e711f2d3fa7ac6fc3eb93385afc
3. 2f02baa13c3fdd75f5502cb528e87435595bba4f
4. 474f2663bf8c7450ffd4b5015d760da60cfabd07
5. 732e917e096564000b77787995808f69fad1a60c
6. c972c5cc6fb40e711f2d3fa7ac6fc3eb93385afc
7. 2f02baa13c3fdd75f5502cb528e87435595bba4f
8. 474f2663bf8c7450ffd4b5015d760da60cfabd07
9. 732e917e096564000b77787995808f69fad1a60c
10. 2f77a2994330bf984e9da6bba8dad453cba3422f
11. f4c66d6f586d7dcb89ff007537eeefb9a803b88f
12. 96eb120e62aa78889f1abf1d9482af1202e157f1
13. 3ec02e2e62047ea971f363effa9a3950a9f79d7b
14. 157986870c996e392a4d28444cdc52e90827e6ff
15. c4fac0b2e3f963526d4099df83a87295d7a1c3e3
16. 1f0fc29ba0217b0358a72b3ec41296f6b71c7900
17. 1afe378e4b94a4bb4396d5d69226364138fa2b9a

Backup: Revision of build options for G3

- **Compile options for G3 found to be rather conservative**
  - due to standard release flags **-O2** of **gfortran** known to lead to unstable builds for G3 (numerical problems)
- **Action taken:**
  - Automated scan and compilation of G3 using increasing subsets of all -O2 options to identify exact cause of problem
  - now building + running reliably with release flags **“-O2 -fno-strict-overflow"** by default
  - see: https://gcc.gnu.org/wiki/FAQ#The_compiler_optimized_away_my_overflow_checks.21_What_is_going_on.3F
- **Few percent ~4% overall accountable to this change**
  - **gain will be larger … the more digitization part becomes smaller**