VecGeom navigators for GPU

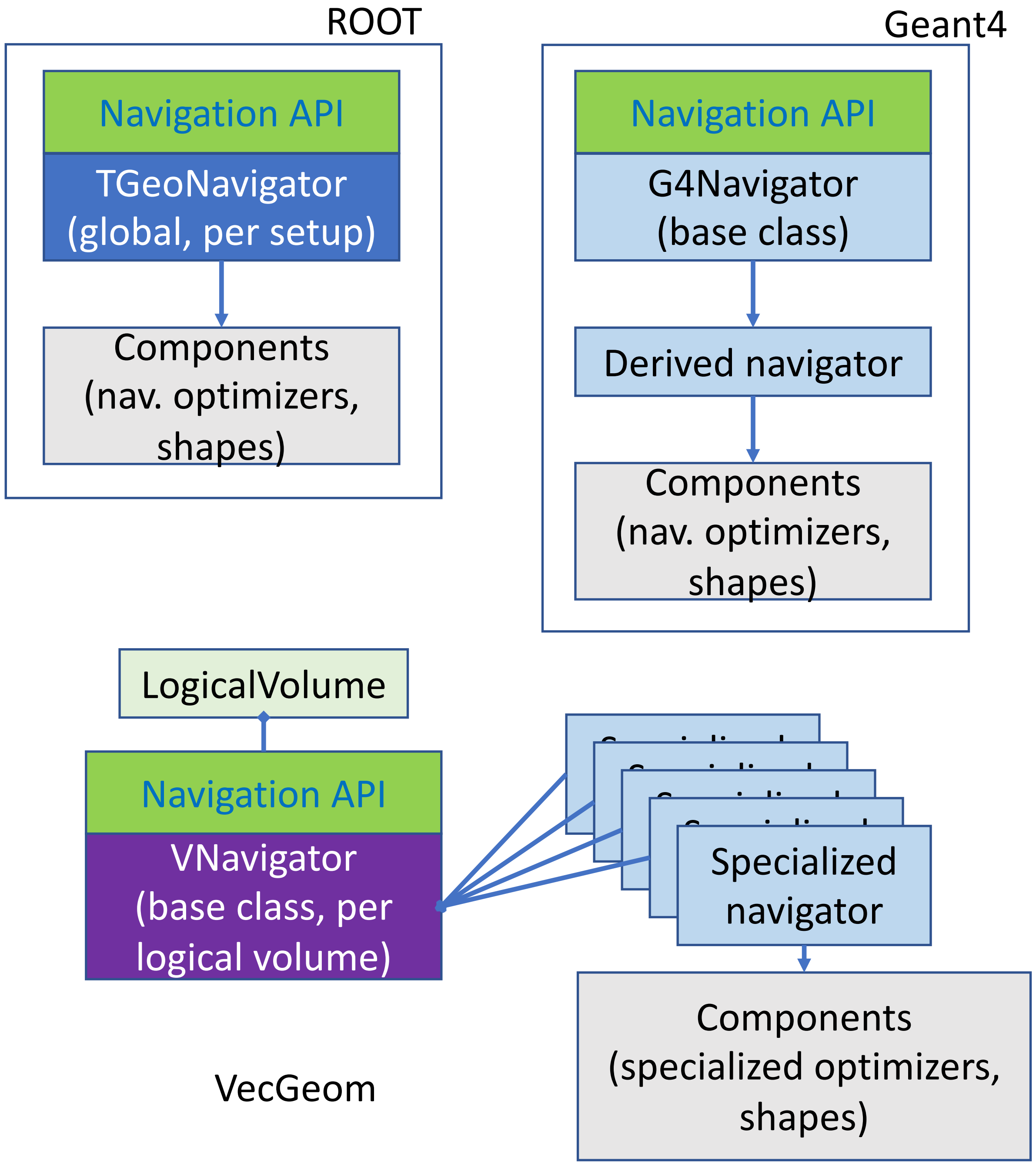
A. Gheata

Preamble

- We want to make simulation GPU-friendly
- ➢Geometry navigation is an important simulation component (%)
- ➢Most geometry components already GPU-aware
  - ➢Except navigation layer
- ➢Work on GPU-friendly navigation
  - ➢Simple example/demonstrator, e.g a raytracer utility taking arbitrary geometry setup

Navigation interface

- VecGeom top navigation layer quite different from ROOT and Geant4
  - Specialized per logical volume topology (complexity) or optimization type (simple loop, SIMD)
- Question:
  - Porting existing navigator for GPU case vs. implementing a GPU-friendly specialization



CUDA-friendliness of VecGeom classes

- Implemented using custom macros (host/device, forward declarations)
- The portable classes are compiled under different namespaces into separate libraries
  - *cxx* for the host compiled with *gcc/clang/icc*, *cuda* for the device, compiled iwith *nvcc*
- The world volume and its content can be streamed over to GPU
  - *CudaManager::LoadGeometry(GetWorld()) // prepare lists to be streamed*
  - *CudaManager::Synchronize(); // actual allocation and copy to GPU*
- For all logical volumes, the navigator getting constructed by default is NewSimpleNavigator (stateless)
  - Implemented navigation as a loop over daughter volumes

Specialized CUDA navigator

- In the first approximation, NewSimpleNavigator could be used
  - Not optimized, just to make a simple demonstrator for global navigation
- Porting existing SIMD-specialized navigators to GPU “as is” pointless
  - The internal data structures organized in SIMD lanes, not matching number of GPU warps
- Parallelism models: per track (top level) versus per feature (internal)
  - Internal parallelism on model features not efficient for long GPU vectors
    - (e.g. one daughter to a warp, one ABBox to a warp, …)
  - In navigation algorithm pipelines, having just few components massively parallelized is not globally efficient

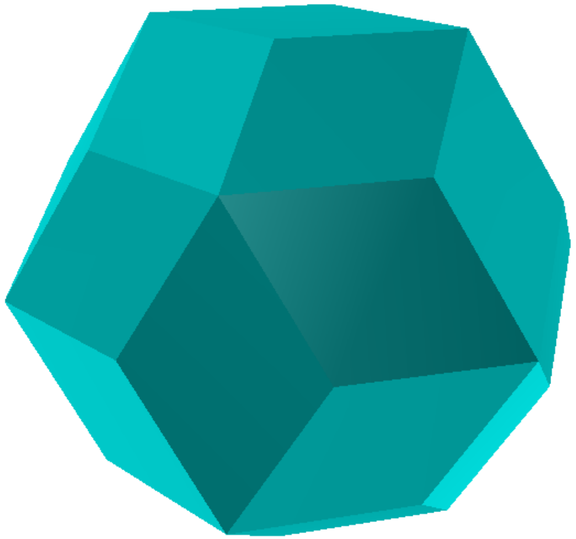
➢ We need optimization structures that work well in scalar mode
  - ➢ Stateless or read-only

A possible plan

- Make a simple example of a global raytracer (setup, not only single volume)
  - CUDA kernel, analogue to Benchmarker.cu
  - Using NewSimpleNavigator in the first implementation
  - Benchmark on GPU vs. CPU
- Implement a bounding box accelerated scalar GPU-friendly navigator
  - Number of BBOX levels and volumes per level optimized for a given volume, not for the GPU architecture
- Benchmark for complex geometry
- Investigate alternative portability libraries for the example (e.g. Alpaka)

Side topic: support for tessellations in ROOT

- Requested by experiments and DD4HEP for conversions Geant4<->VecGeom<->ROOT
- No navigation functionality but:
  - Validation checks (e.g. compacting common vertices, checking facets for degeneration, vertex order definition/flip)
  - Persistence in ROOT/GDML formats
  - Visualization



Triacontahedron as tessellated shape in ROOT