Is efficient e/γ calorimeter simulation achievable using native GPU ray tracing ?

J. Apostolakis, CERN EP/SFT
10th June 2020

Why harness GPU ‘native’ libraries ?

Performance

+ raw performance potential,
+ vendor support (migration path)

- tie-in with vendor library
- evolution is unpredictable

Opticks offloaded optical photons to GPU and used full power of recent GPUs:

- Hardware ***Ray Tracing*** (RT) cores for geometry intersections
- Different **‘shader’** routines (CUDA) for **physics** trigerred for cases of intersection or misses (no hit in dist.) with limitations.

Reports impressive acceleration (up to 1500x speedup)*

Goal: investigate how to

- **Adapt** approach to simulate **gammas**
- Extend to **electron** & positron simulation.

* optical photons, seem / are an “ideal” speedup target for GPUs.

What particles / physics to meld with native ray tracing ?

What can we move to ‘shaders’ ?

Can we balance GPU & CPU load ?

>90% steps in LHC Geant4 simul. involve e, γ, n & p

- 65-80% are γ, e^- & e^+

Most e γ steps are in showers in calorimeters, but

- average step number N_γ >> N_e
- γs involve linear propagation
- γs traverse more boundaries
- γs have discrete interactions (no MS, no Eloss)

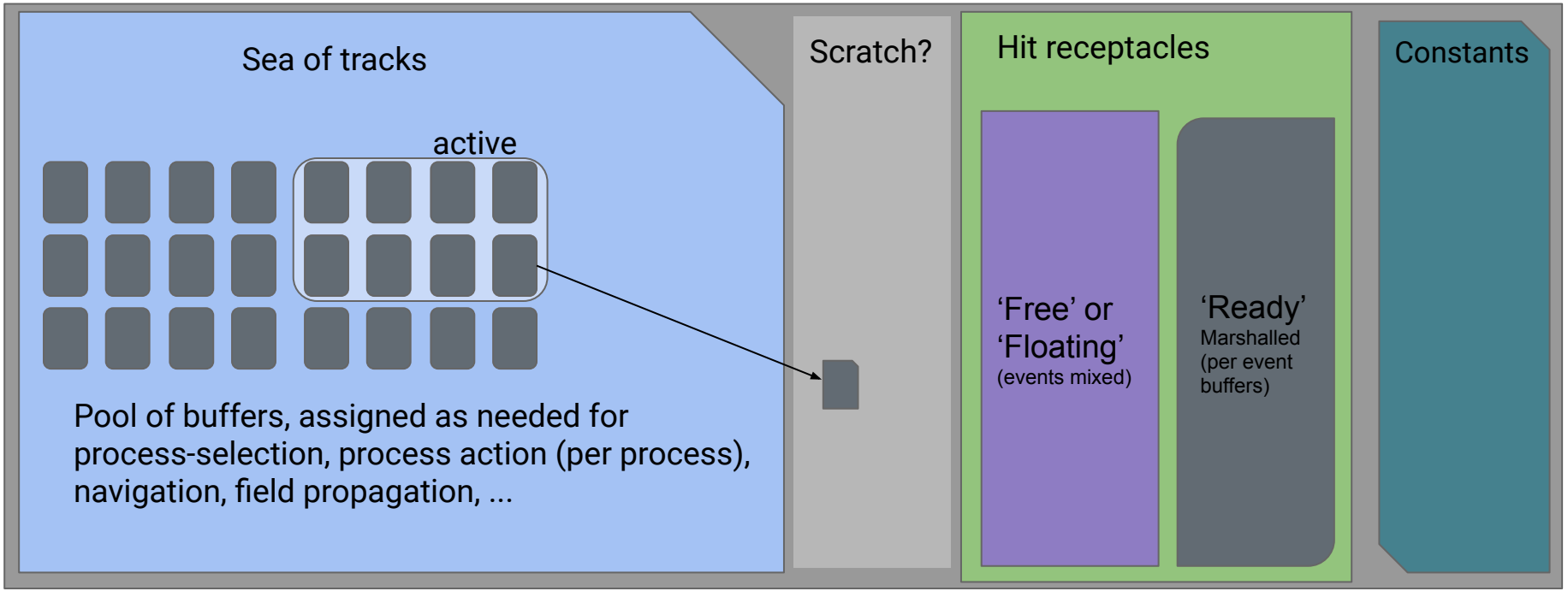
Offload first γ showers, then e-/e+

Full benefit: all γ / e (in calo.) must remain on GPU

Tackle γ models as ‘shaders’ in order of complexity:

- Compton
- Gamma conversion
- Photo-electric

Partitioning GPU memory



Geometry: First steps

&

Challenges

How to deal with copies of large structures?

Large number of subvolumes or repetition inside them

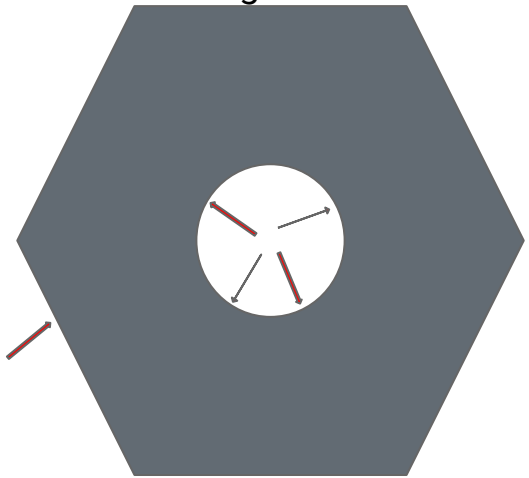
First steps

- Look at EM calo/s from LHC experiments
- Identify necessary volume types
- Select ‘core’ set of solids (later more/most)
- Implement one LHC calorimeter (section?)

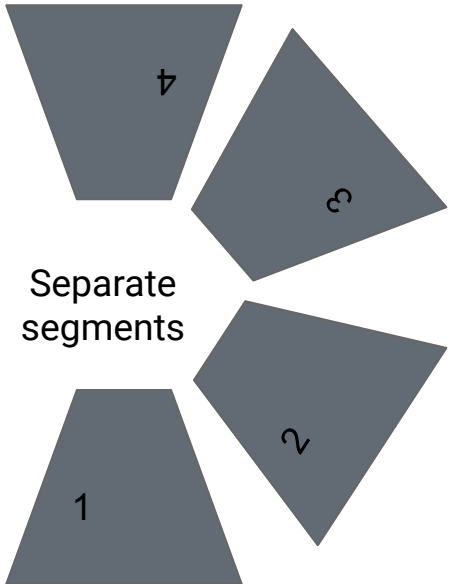
Check results & benchmark

- Number of realistics nav. steps / sec (vs G4 CPU)

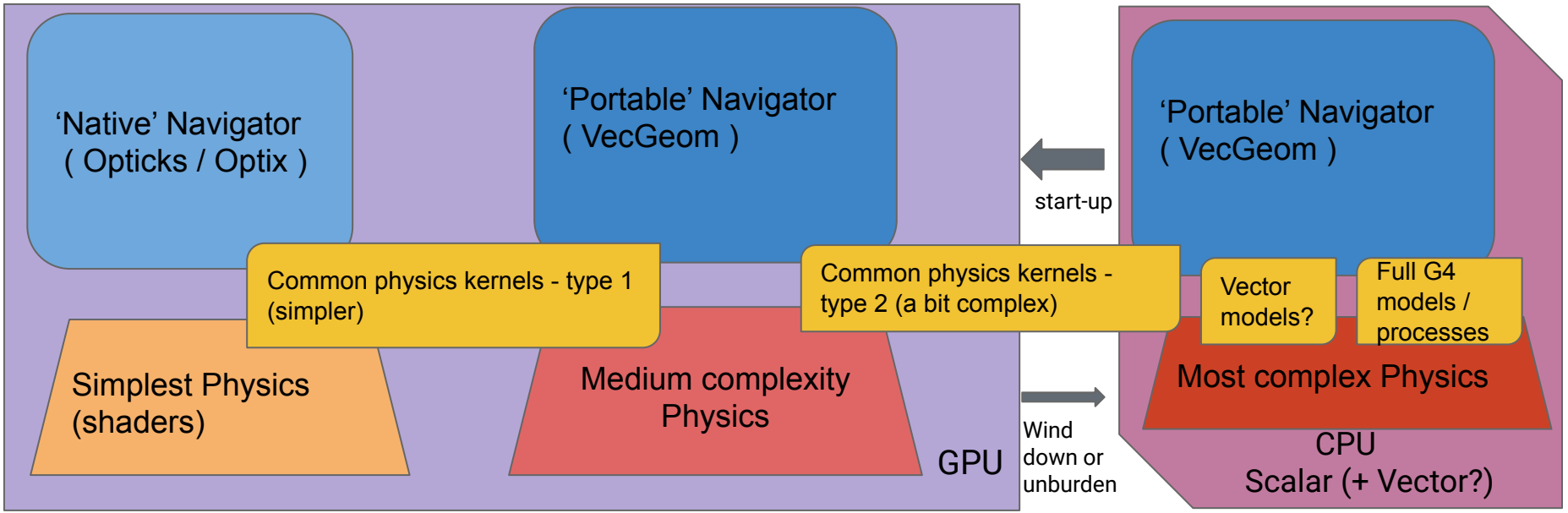
One large volume ?



Separate segments



Share physics kernels & join two approaches



400x (MPEX) - 1500x (Opticks)

Goal: Utilise the power & capabilities of both accelerator (GPU) and CPU

16/32/64 cores

Considerations

- Much to learn from Opticks & Simon
  - expect to benefit greatly both from Simon's pioneering effort and offer of advice (& potentially collaboration)

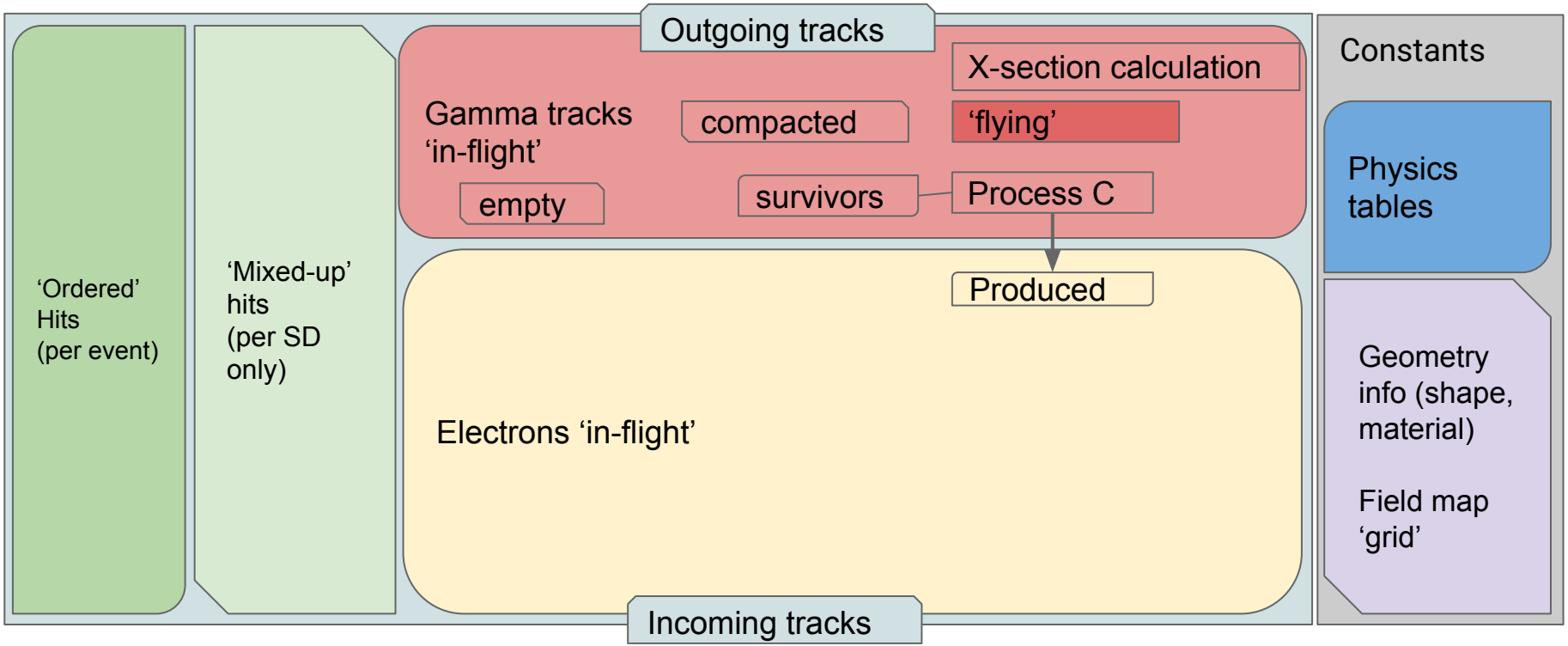
- Need to explore how to share maximally & coordinate
  - exploring 'shader' (native library) approach &
  - the more 'traditional' VecGeom based prototype.

- The initial version of the prototype is a small fraction of the journey
  - Must manage the populations of tracks to avoid drought and flood;
  - Experience shows that a lot of optimisation is needed to get performance

- If we can eventually create a performant prototype, a lot of questions would arise:
  - portability, remaining particles, reproducibility, robustness, code 'understandability' ……

Delving into some details / concerns

Idealised view of GPU memory



Topics for investigation / prototyping

Evaluate capabilities, design of Optix / Opticks

To describe geometry of select LHC calorimeter(s)

- Identify extra capabilities needed ?
- First prototype of navigation
- Benchmarking - with partial geometry if needed

Physics models

- Size of shader routines for **gamma**, e-/e+ interactions
- Physics tables vs (simple) formulae

Curved trajectories (Runge-Kutta) & intersections with ‘native’ geometry

Questions:

What is effect of constraints from ‘shader’ approach

- Evaluate impact on (other) physics models
- Tables or formulae ?

How can we create common kernels with ‘CUDA’ prototype?

- Requires common data types
- Common code for kernel methods
- Steering & auxiliary methods could differ

How to enable mixed-mode simulation?

Common datatype

Tracks must flow freely between

- Between CPU and accelerator, and between
- Between native-library-based & the general VecGeom-based kernels (per particle type?)

Critical: simple, **compact** data structure for track - plain old data

- Global Particle Identifier (event, particle species code, integer counter)
- Global Volume Identifier (32 bits)
- State (E, pVec, t, xVec)
- Nothing more (extra data can live in hash table in the CPU, indexed by particle Identifier)

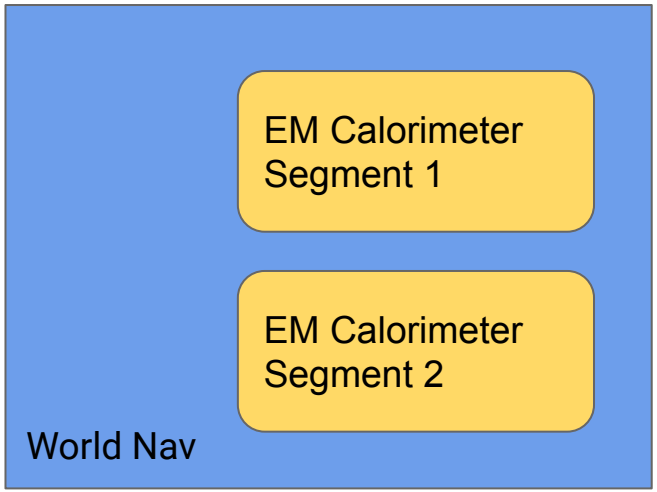
Geometry / Navigation component

Key concerns:
- Flat or hierarchy ?
- Can we afford memory for full tree of ‘touchables’
- Using repetition / regularity as much as possible.

How to use patterns? How is structure regular - i.e. cells in calorimeter, tracker, phantom:
- Is regularity simple: cartesian, phi-slices, hexagonal (?)
- Is a custom scheme needed ?

Assemblies and replicas / divisions can play a key role to identify some regularity even if currently implemented differently.
- Need extension(s) to current Opticks to implement.

One navigator (Optix-based)
Or Optix/VecGeom hybrid?



Possibly accommodated by VecGeom specialised navigation

Collaboration, Licensing & Funding

(Speculative … ?)

It is difficult to bring together prototypes or projects at different stages of maturity.

Geant4 was born by bringing together teams that had worked on ‘similar’ prototypes, agreeing stable interfaces and evolving implementations on one platform (CPU).

The current era is more complex, evolving, and likely will require more frequent ‘architecture’ revisions - evolution of interfaces, data model

Looking 5-10 years ahead we need a strong development team, that brings all developers under a broad umbrella

The challenge of sustainable funding sources must be considered early, and potentially addressed by adopting a licensing model which does not block future options. Concerns to bare in mind: keep principle of free ‘research’/open use via hyper-open license, but seek to identify potential for fee-based licensing for closed-source / commercial applications?