J. APOSTOLAKIS, D. SOROKIN

RECENT DEVELOPMENTS IN MAGNETIC FIELD

OVERVIEW

- Recent development - in 10.6 and 10.7.beta
- Open issues, requirements
- Recent investigations
- Ongoing development for 10.7

- A small window into an investigation of a replacement for TLS in G4 geometry

EVOLUTION OF DRIVERS FOR RUNGE-KUTTA INTEGRATION

- The driver calls a stepper and controls its error.
  - Until 10.3 there was a single driver class; a base class was introduced to deal with different methods: Burlisch-Stoer (non-RK) / plain RK / FSAL / Interpolation
  - Old code: G4Mag_IntDriver has a pointer to G4MagIntegratorStepper
  - G4IntegrationDriver<StepperType> knows stepper type- avoids virtual call
  - G4InterpolationDriver<StepperType> interpolates (see Dmitry's report in 2019.)
-

EVOLUTION OF DEFAULT CONFIGURATION

- *G4ChordFinder* creates as default a new hybrid driver since release 10.6 (*G4BFieldIntegrationDriver*)
  - *G4InterpolatingDriver*<DormandPrince 4/5th order> for ‘small’ steps
  - ‘simple driver’ with Helix based stepper for ‘large’ steps - using *G4IntegrationDriver<>*
- It was testing in 10.5-ref09/10 and proved reliable & stable.
  - As CMS reported it was slightly **faster**, we chose to keep it as default in 10.6.
- Potential for future refinements - due to its implementation & ***extra evaluation*** of B-field,
  - refine implementation to avoid **virtual calls** to the two “sub-drivers”
  - investigate how to reuse field value (non-trivial redesign?)

DEALING WITH LOOPERS

- One key problem is that ‘looping’ tracks can go kilometres in gases
  - A track that is a tight helix ($R_{helix} < \delta_{chord}$) can take forever with little benefit
- Allowed relaxation of δ_{chord} after a fixed number of iterations
  - By default after every 100 iterations δ_{chord} is doubled
  - Reset at end of step - of course.
- Of course there is a risk to miss a small volume – so users encouraged to check
- The number of iterations is a changeable parameter

ATLAS: PORTING ISSUE

- Issue in porting B-field integration to 10.4+
  - relied on ex-unique driver (G4MagInt_Driver)
  - Not trivial to adapt using release notes & documentation
- Provided help to
  - identify reason & fix crash
  - create robust code, which exploits new drivers in recent G4 versions – as a result depends on G4 version
- Key migration message:
  - Advanced use case of changing stepper is no longer works the same (old) way:

```
G4MagInt_Driver* driver= chordFinder()->GetIntegrationDriver();
driver->RenewStepperAndAdjust(ptrStepper);
```

    as *Get* now returns a G4VIntegrationDriver* (not G4MagInt_Driver*)
  - A user must construct a driver from scratch in order to assign it:

```
auto driver= G4IntegrationDriver<StepperType>(…)
chordFinder->SetIntegrationDriver(driver);
```

New Integration Driver/Steppers in Geant4

- In Geant4 10.3 and earlier the type of driver in G4ChordFinder (.hh) was
  `G4MagInt_Driver*fIntgrDriver;`
- The default driver in Geant4 10.6 is now a (very complicated) new templated type that mixes IntersectionDriver<G4DormandPrince745> for small steps and a helix-based RK-type method for long steps.
- Any code which assumes the type of the G4VIntegrationDriver is no longer safe. In particular :

```
magDriver = static_cast<G4MagInt_Driver*>(fieldMgr->GetChordFinder()->GetIntegrationDriver());
```

- If we continue to use G4MagInt_Driver, we would have virtual calls at both levels
  - Even in the case of using G4IntegrationDriver<G4MagIntegratorStepper> we will have virtual calls at both levels.
  - Only if using G4IntegrationDriver<ConcreteStepperType> we get rid of virtual call to stepper

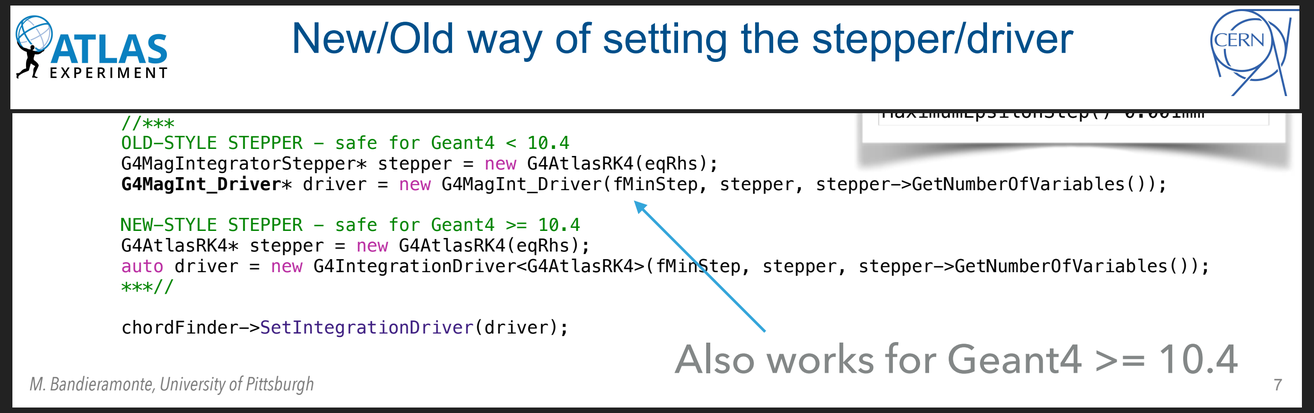
Issues reported & proposed fixes (of user code) verified in ATHENA by Marilena Bandieramonte & Miha Muskinja

Note on driver's RenewStepperAndAdjust() method:

- design revision in 10.4 missed this method in G4VIntegrationDriver
  - this forced use of a cast to G4MagInt_Driver*
- from 10.5 it became a virtual method, usable only for G4MagInt_Driver
- we should suppress this method in release 11.0
  - thus we need to add 'obsolete' warning in 10.7

ATLAS: PERFORMANCE OF DRIVERS

- ATLAS Performance problem using 10.4
  - New templated driver ~10% slower than older
  - With charged geantinos 2x slower
- We recommended to check with 10.6 (& started investigating)
- Latest (preliminary) results with FullSimLight
  - For G4 10.6 the new templated driver is faster than the old



- Demonstrates utility & importance of new FullSimLight
  - Lightweight, standalone
  - faster turnaround

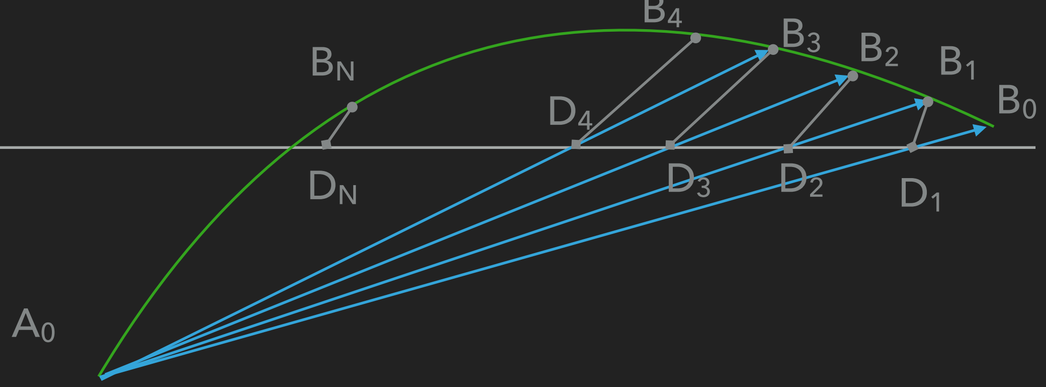
FullSimLight / Marilena B. benchmarks 23.09.2020	10.4	10.5	10.6
Old way: G4MagInt_Driver with virtual stepper	5143+- 20	5300+- 31	5080+-29
New way: InterpolationDriver<G4ATLASRK4>	5303+- 24	5265+-21	4999+-23

- Thanks to Marilena Bandieramonte (Univ. of Pittsburgh, ATLAS)

ISSUES WITH INTERSECTION FINDING

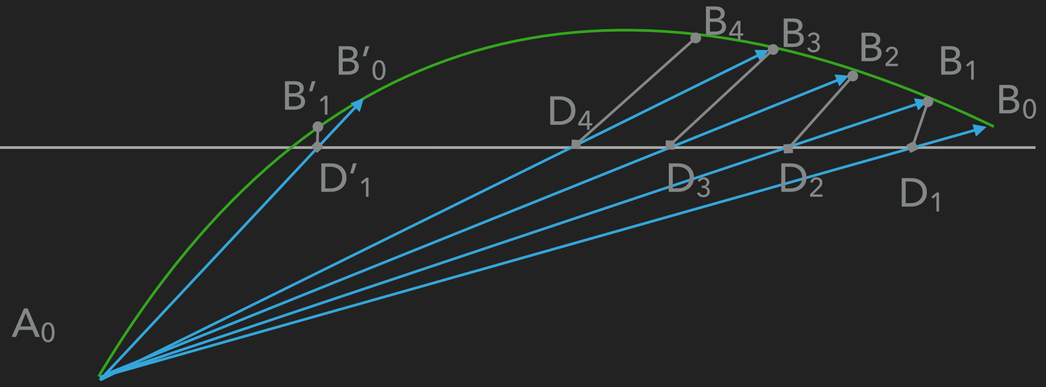
- CMS and Belle reported issues with zero steps
  - CMS issue was an intermittent crash due to “out of order” points
  - Dennis Wright (Belle) reported multiple “zero step” issues
- Both clearly involved the ‘Intersection Finder’ capabilities in the propagation in field
  - diagnosis required code that can be ‘dropped in’ to report when a problem occurs

LOCATING AN INTERSECTION



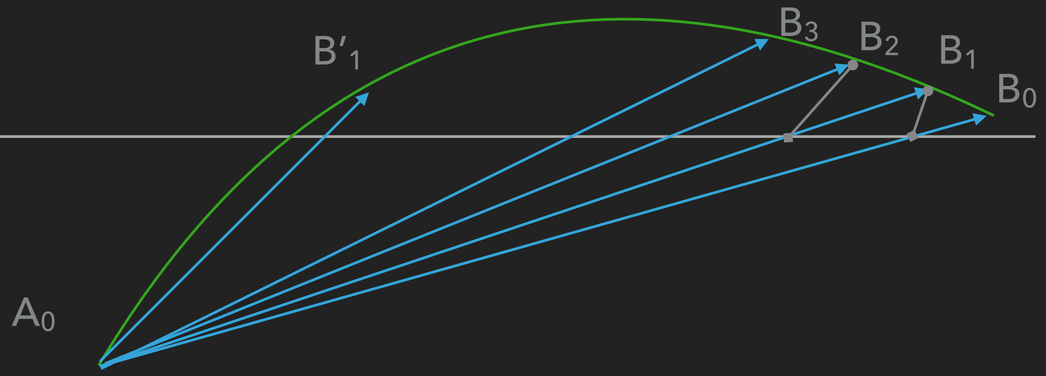
- G4VIntersectionLocator has the *EstimateIntersectionPoint*() virtual method
  - All use chord intersection (D_0, D_1, ..) to locate nearby 'candidate' curve point
    - the intermediate points found are B_1, B_2, B_3, ..
  - All methods iterate until curve & chord intersection points are 'close enough' – their distance $|B_N D_N| < \delta_{intersection}$
- Variants:
  - Simple (as above, slow convergence),
  - Multi-Level (next slide),
  - 'Brent': uses a faster converging method – current implementation is unnecessarily complex & not robust

MULTI-LEVEL INTERSECTION LOCATOR



- G4VIntersectionLocator has the *EstimateIntersectionPoint*() virtual method
  - All use chord intersection (D_0, D_1, ..) to locate nearby 'candidate' curve point
    - the intermediate points found are B_1, B_2, B_3, ..
  - ***G4MultiLevelLocator*** interrupts this sequence after 5 trials (substeps)
    - Splits current segment into two (up to 'depth' of 10 times)
  - Iterates until $|B_{j/k}D_{j/k}| < \delta_{intersection}$
  - It also returns a 'D' point - a *biased* estimation of true intersection (see later)

CREATING A “DRIVER RECORDER”



- G4MultiLevelLocator is the production code for finding an intersection
  - Uses candidate intersection to locate intermediate curve point
  - Iterates until curve & chord intersection point are ‘close enough’ < $\delta_{intersection}$
  - Very complex logic - but tried & tested
- Need to identify code path taken when exceptions occur.
  - Decided to create a whiteboard-like ‘recorder’ class

A ‘DRIVER REPORTER’ WHITEBOARD

Create a new Class G4DriverReporter for use by G4MultiLevelLocator (and potentially other types of G4VIntersectionLocator)

```
=============================================================================
Change#  Iter          CodeLocation          Length-A (start)          Length-B (end)
=============================================================================
      1     0    2    Initialising                         0          5.329505672195954
      2     0    3    IntersectsAF                                    1.314059689653547
      3     0    6   RecalculatedB                                    1.314059689653547
      4     1    3    IntersectsAF                                   0.3132607321997748
      5     1    6   RecalculatedB                                   0.3132607321997748
```

Thanks to Gabriele who ‘protected’ it, configuring it to occur if requested

- Whiteboard-type class to record iteration, ‘code’ and a value
  - It records value when turned on (currently in navigator ‘check’ mode.)
  - When asked it can print the full set of iterations / decisions
- The trigger is typically an error condition (added already in key *G4Exceptions*)
  - For debugging / investigations could use another condition such as an ‘overflow’ (a lot of iterations)
- Feedback obtained from Belle by Dennis (shared earlier) shows what triggered zero step (not yet resolved - sorry)
- CMS not tried it to date (afaik) - it is now available in 10.7beta or later dev releases.

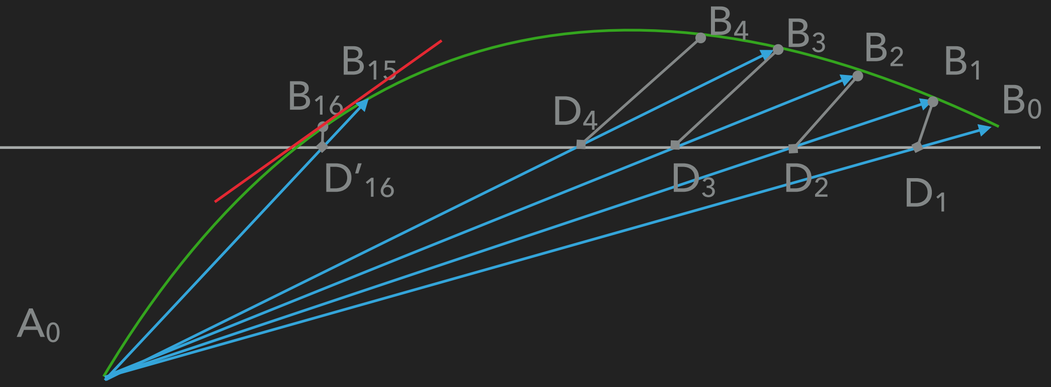
USER FEEDBACK

- ALICE requested in 2019 that we make default values adequate ‘out of the box’ for most HEP experiment use cases (supported by CMS, others.)
  - Will require tightening epsilon parameters – potentially simplifying to one relative accuracy parameter + potentially a new ‘acceptable integration’ error distance (in place of eps_min, eps_max +
- CMS recently reported a bias in high-energy forward muons
  - Bias of about 2.5-3.0 um, in opposite directions for mu+ / mu- after about 10 boundary crossing
  - Changing all field parameters (ε_{min}, ε_{max}, $\delta_{intersection}$) by x10 reduced bias, by x100 eliminated it.
  - Potential performance impact => Will propose ‘best’ values for studies and ‘good’ values for production
- Need to revisit ‘accuracy’ parameters. Plan is to
  - first revisit the decision to use a biased estimator for the step endpoint (bounded by $\delta_{intersection}$)
  - review / tighten the default values of the key accuracy parameters (ε_{min}, $\delta_{intersection}$) and eliminate ε_{max}
  - likely revise the set of user-configurable parameters - either to further simplify them, or to recast them in more user-oriented form.

USER FEEDBACK

- ALICE requested in 2019 that we make default values adequate ‘out of the box’ for most HEP experiment use cases (supported by CMS, others.)
  - Will require tightening epsilon parameters
- CMS recently reported a bias in high-energy forward muons
- We plan to
  - review / revise the default values of the key accuracy parameters
  - revisit the decision to use a biased estimator for the step endpoint (bounded by $\delta_{\text{intersection}}$): test different choice of intersection point and/or convergence criterion

REVISING INTERSECTION TO ELIMINATE BIAS



- Clearly we need to move towards providing an unbiased estimate of the intersection point. But how?
- expect to keep using iterative methods - until curve & chord intersection points are ‘close enough’ < $\delta_{intersection}$
- Options:
  - “quick & dirty”: force a very small $\delta_{intersection}$ to seek undetectable bias
  -

USABILITY : STEPPERS & DRIVERS

- Simplifying selection of ‘standard’ integration drivers / steppers
  - Create a ‘factory’ for use in multiple applications - and by messengers, examples
- Improving & classifying steppers
  - ‘Production’ class - e.g. RK: modern FSAL steppers of ‘medium’ order 4/5
  - ‘Advanced’: steppers with more benefits (higher order, interpolation, ..)
  - ‘Legacy’: G4ClassicalRK4
  - ‘Experimental’: move less tested steppers into separate location (sub-directory?)
- Provide more steppers with ‘Interpolation’ capability - beyond G4DormandPrince745 (4/5th order, 7 stages = total evaluations of equation / field - one at the start).
- Potentially part of 2021 developments

SUMMARY / OUTLOOK

- Field propagation remains a critical area for LHC & other applications
  - both accuracy (‘out of the box’) and high performance (esp. with tuning) are key needs
- Issues keep being reported – some ‘standard’, others much trickier
  - It is critical to create tools to identify the source of problems (ongoing)
  - Will tighten values of key parameters to make them more robust for typical HEP applications (esp. collider experiments)
- There is certainly further room for improvement (usage, simplification, accuracy, performance)
  - some should be relatively painless (adjustments & small improvements)
  - others will require substantial investigation & new implementations (e.g. eliminating intersection bias)

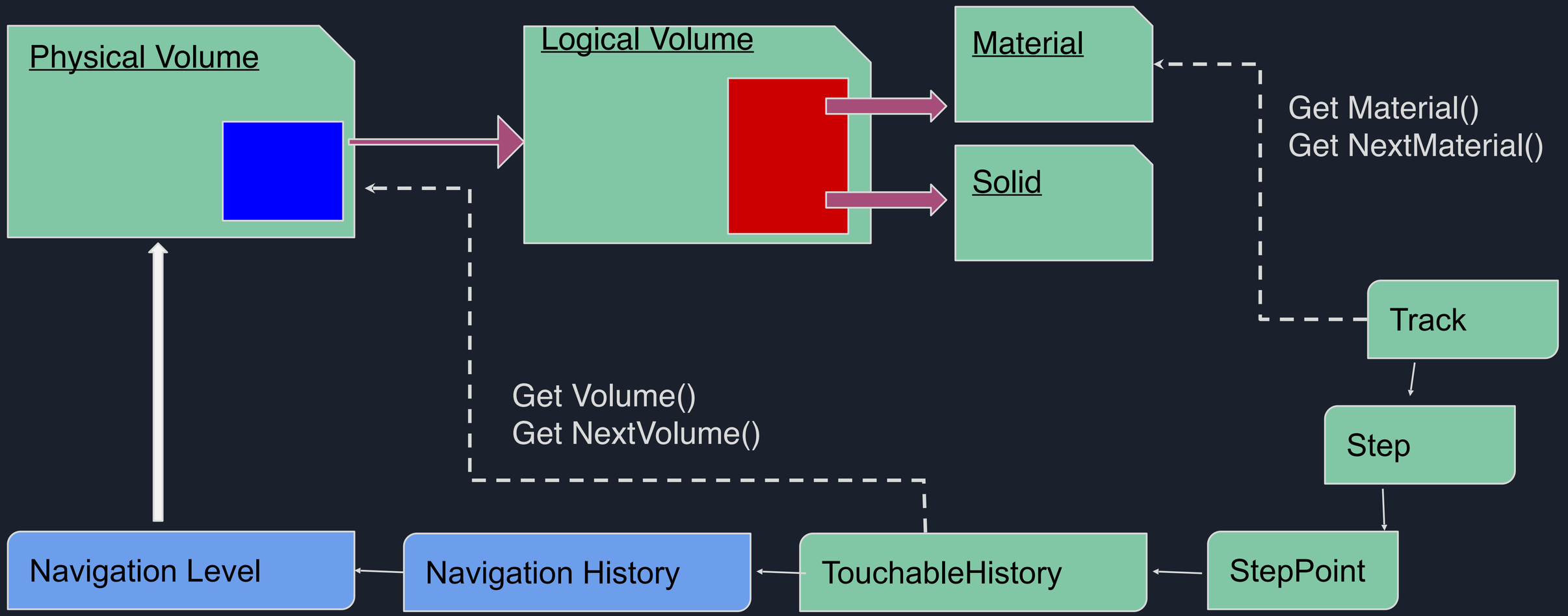
A DIFFERENT TOPIC —— NOT FIELD RELATED

GEOMETRY STATE AND MULTITHREADING

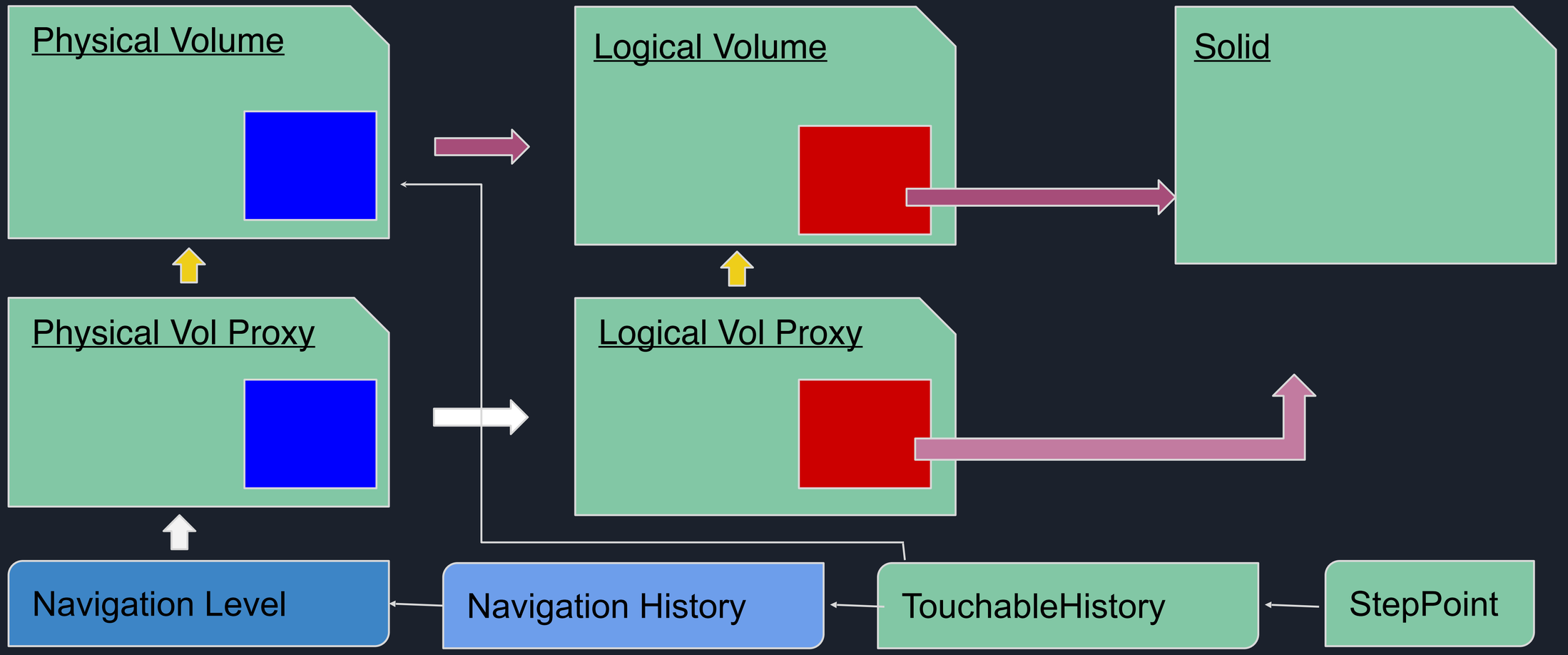
COMPLEXITY & OVERHEAD OF TLS IN GEANT4

- Can we make the state of the Geometry fully thread-safe and also more performant ?
  - Investigated during spring 2020
- Next: A small taste of what was involved
- More / full information in presentations in VecGeom Developer meetings
  - Overview: 11 May 2020
  - Update: 25 May 2020

Current design (Geant4 / Geant4-MT)



The core idea: proxies to hold changing data



REPLACING ‘THREAD-LOCAL’ ELEMENTS OF G4 (LOGICAL) VOLUMES

- The state of the live Geant4 geometry includes
  - the integer copy number stored in a G4Replica object (or TLS ‘extension’)
  - the solid (full object) & material pointer of a Parameterised volume.
- They must be different in each thread – so each class has a ‘split’ personality: part of it lives in a separate TLS ‘side’ object, different in each thread.
- So for these and a number of others there exist ‘parallel’ objects which
  - hold the thread-dependent state
  - ‘transparently’ answer the relevant methods
  - involve a cost in looking up the thread-id
  - are organised into ‘workspaces’ per thread for initialisation, dynamic creation, …