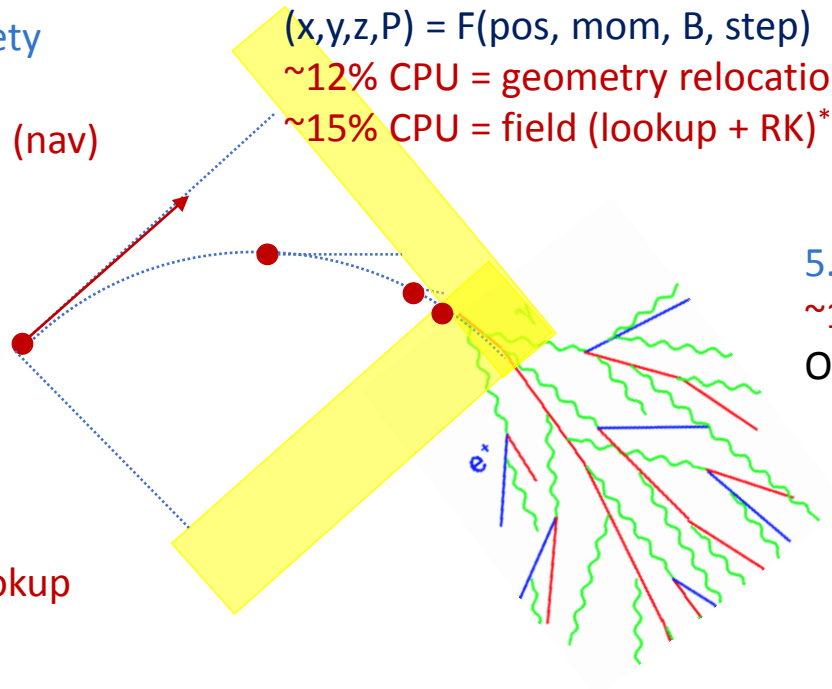
# Simulation on GPU?

Andrei Gheata

# Budget for a simulation step (CMS simulation)

4. Propagate with selected step
(x,y,z,P) = F(pos, mom, B, step)
~12% CPU = geometry relocation O($10^{3-4}$) branching LOC
~15% CPU = field (lookup + RK)* O($10^{3}$) LOC

2. Find next boundary + safety
snext = F(pos, dir, geom)
~12% CPU = branching code (nav)
O($10^{4}$) branching LOC




1. Sample interaction length:
phys_step = F(Xsec(energy, material))
~12% CPU = table lookup + interpolation
O($10^{2}$) LOC

5. Post-propagation MSC step correction
~10% CPU = FP calculation*
O($10^{2-3}$) LOC

3. Sample MSC
~6% CPU = table lookup
O($10^{2}$) LOC

6. Continuous processes (ioni)
Eloss, P'
~2% CPU = FP calculation
O($10^{2-3}$) LOC

Do **stages** 1 to 8 for every track step

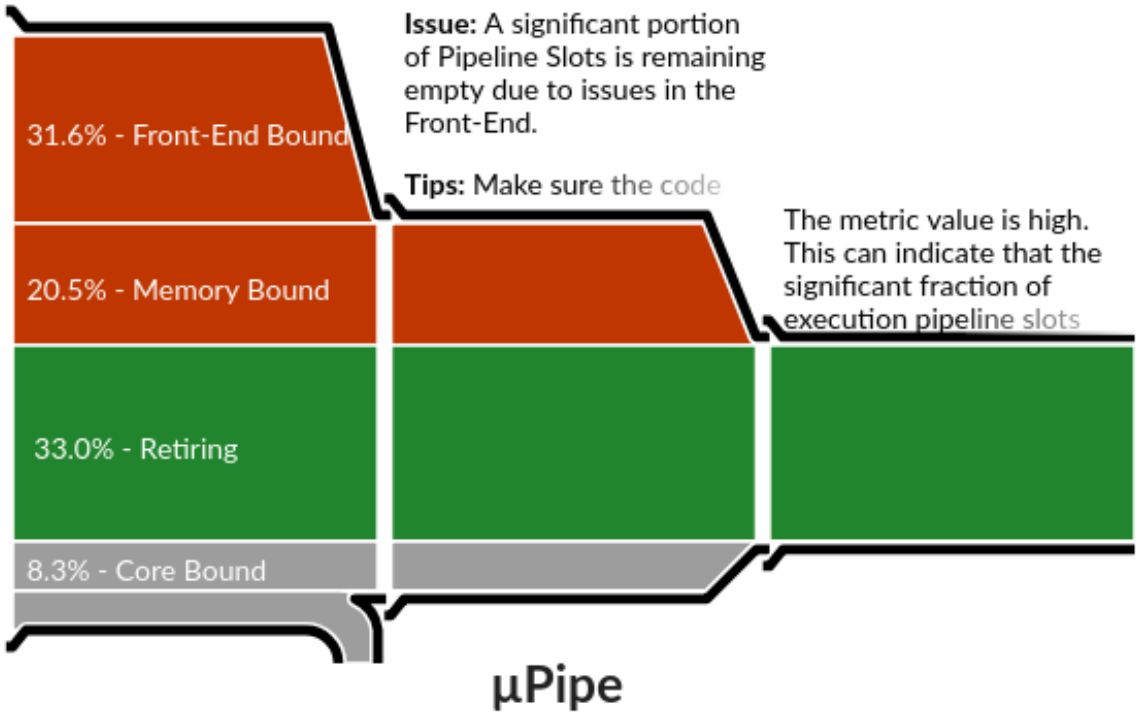
7. Sample discrete process + at rest
$N_{sec}$ = f(process, …)
~10% CPU = FP calc. split between >10 models*
O($10^{3-4}$) LOC

%CPU vary depending on many simulation parameters. The remaining to 100% is management, overheads, …
#LOC is a rough estimate for the "weight" of the module

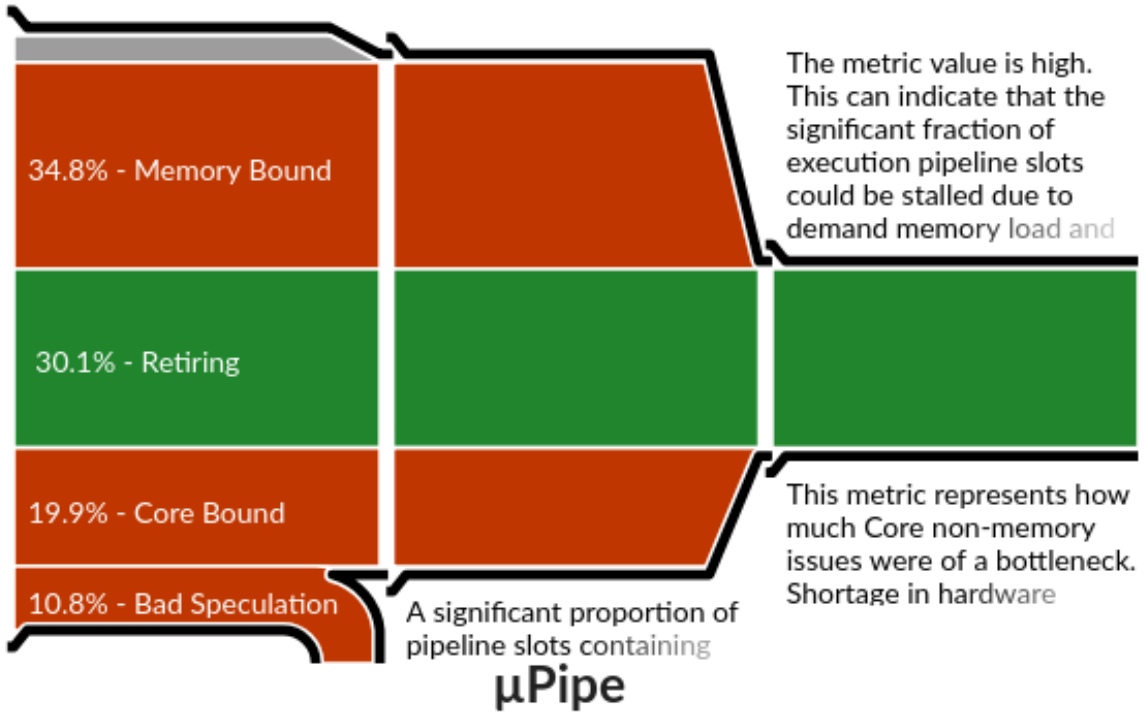
8. Stepping actions (accounting, user scoring)

# CMS Simulation Application µPipe

Geant4




GeantV



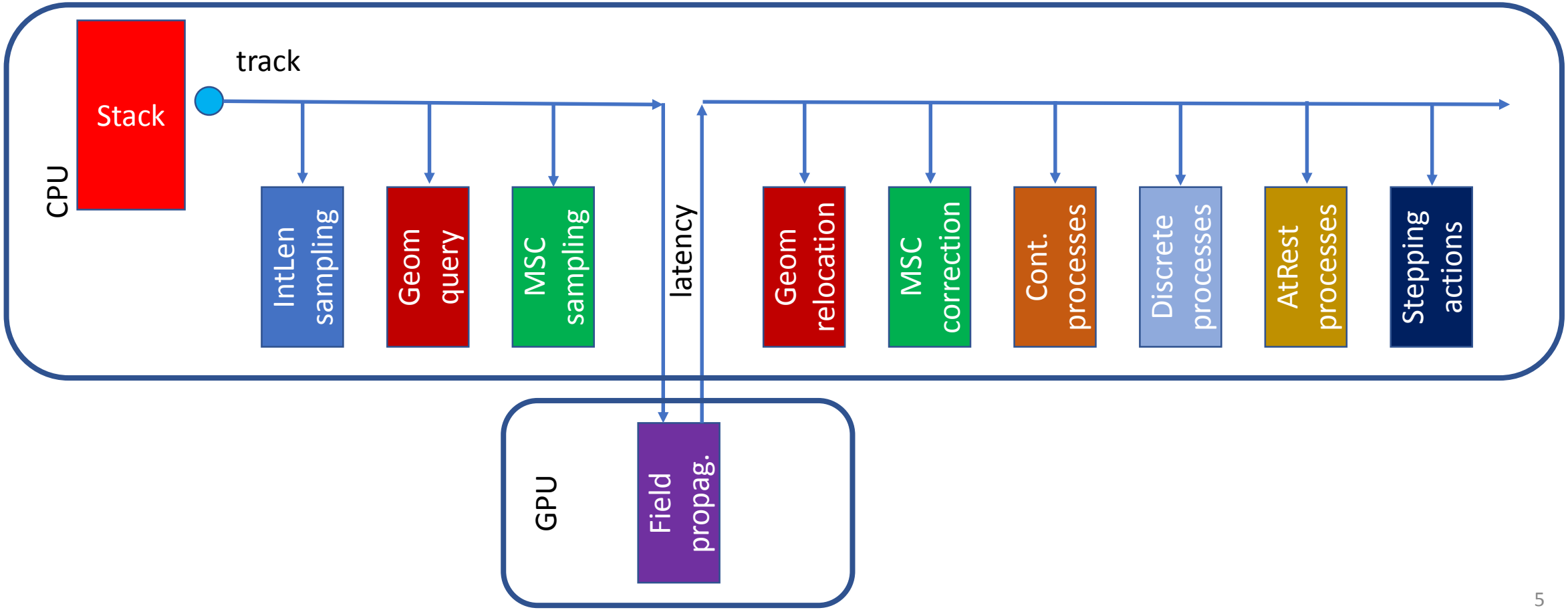

Doesn’t look good…

# GPU considerations

- Architecture very different compared to CPU
  - CPU: huge ALU, caches and control units – minimize memory access latency
  - GPU: many small ALU and control units w. small caches – latency is an issue
    - Good for code independent on data values (small branching)
- Portability: possible, but big issue for large code base
  - Can we run full simulation on modern GPUs?
  - What is the migration effort?
- Limited pipelines for 64bit operations – using just fraction of the GPU
  - Which parts of simulation can be made 32-bit friendly?
- What is the benefit/cost for migrating some FP-intensive module to GPU?

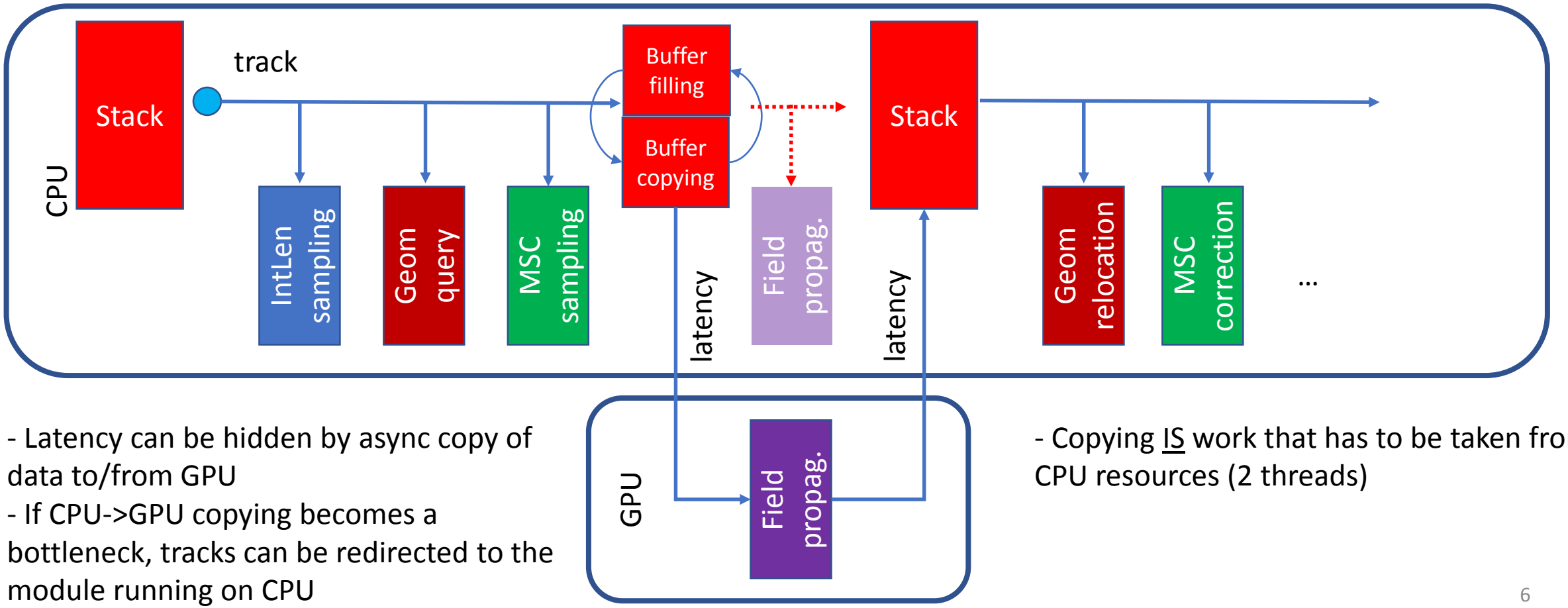
# A possible workflow (1)

- A single track stepping cannot fill the GPU, latency hinders throughput gains



# A possible workflow (2)

- Buffer tracks for a module, 2 threads copy async, step follow-up from new stack




- Latency can be hidden by async copy of data to/from GPU
- If CPU->GPU copying becomes a bottleneck, tracks can be redirected to the module running on CPU

- Copying IS work that has to be taken from CPU resources (2 threads)

# Some prerequisites

- Stateless simulation: all state is embedded in track, tracks are passed via interfaces
  - Issues: interface changes, caching state takes more memory (per track)
  - May need supporting “last produced tracked first” policy
- Insertion of a vector particle flow in the stepping loop, using intermediate stacks
  - We know how to do it, but will it be efficient?
- The idea could be prototyped
  - Minimal effort: use GeantV as testbed
  - Stateless Geant4 + VectorFlow integration ongoing, but will take more time