Detector Simulation Project Prototype

LPCC Detector Simulation Workshop, CERN 7 Oct 2011

René Brun

Project context

- La Mainaz meeting in Jan 2010->Better synergy between G4&ROOT teams in PH/SFT.
- Many discussions between April and October 2010.
- In November 2010, new Project approuved with more focus on medium and long term.
- First conclusions rapidly reached in January.
- First prototype with important conclusions presented in July.
- Main work so far by Andrei Gheata, Federico Carminati and me.
- Discussions with Atlas (Andi Salzburger) and OpenLab (Alfio/Sverre).

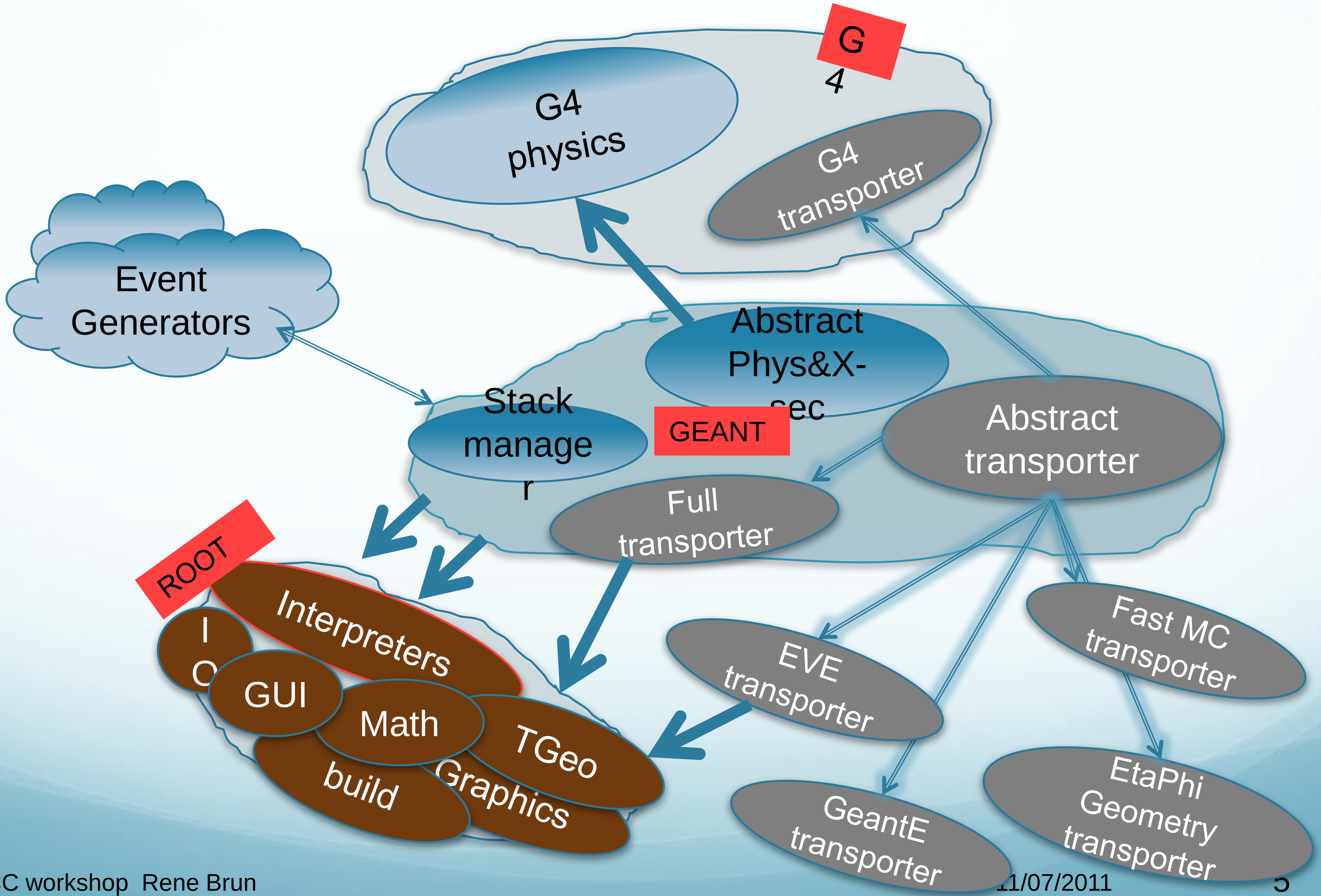
Starting Assumptions

- The LHC experiments use extensively G4 as main simulation engine. They have invested in validation procedures. Any new project must be coherent with their framework.
- One of the reasons why the experiments develop their own fast MC solution is the fact that a full simulation is too slow for several physics analysis. These fast MCs are not in the G4 framework (different control, different geometries, etc), but becoming coherent with the experiments frameworks.
- Giving the amount of good work with the G4 physics, it is unthinkable to not capitalize on this work.

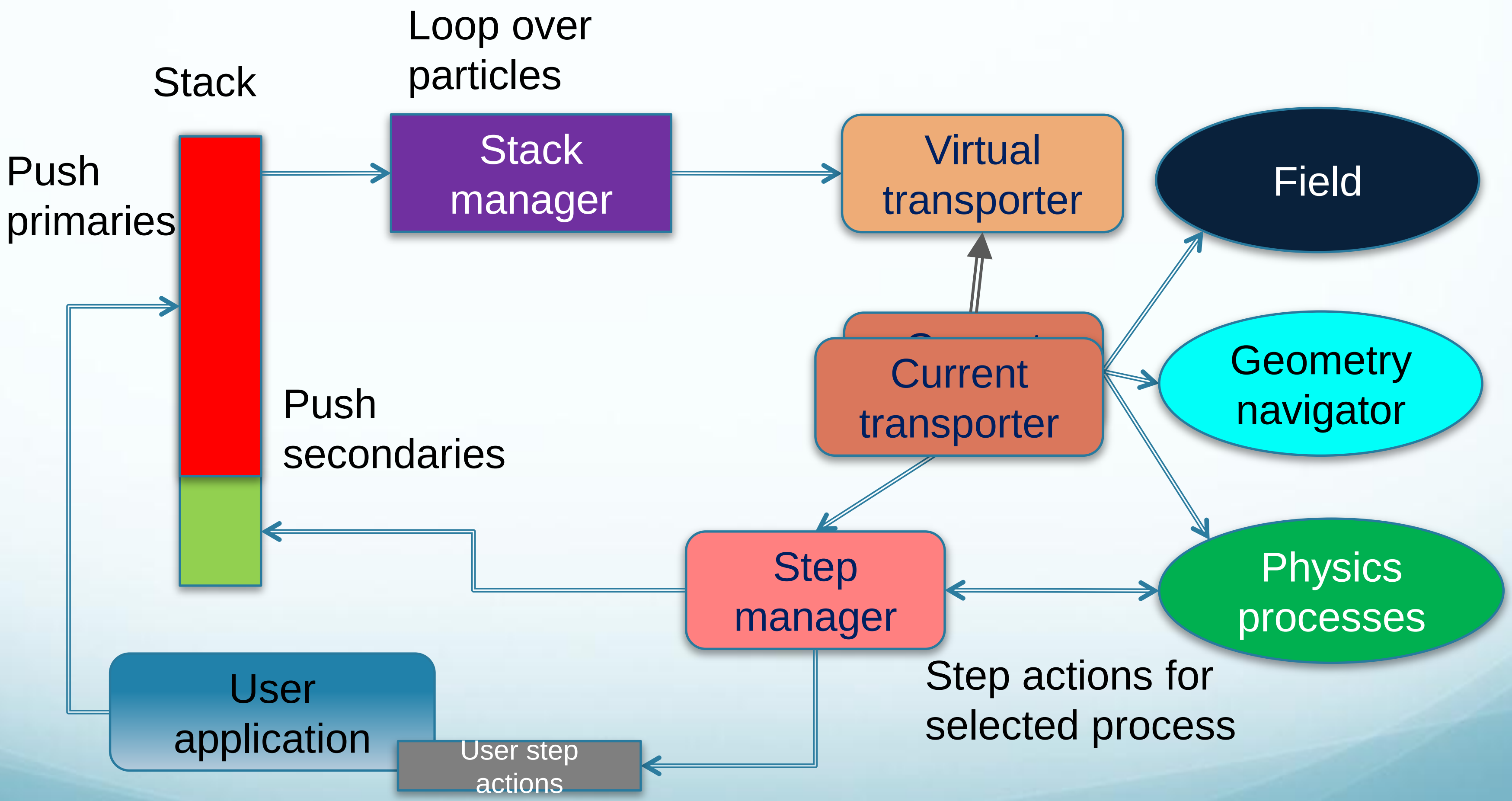
My December talk in a thumbnail

- Increase synergy between G4&ROOT teams.
- Particle stack outside G4.
- Virtual transporters with concrete instances for fast or/and full simulation, reconstruction,visualization.
- Investigation of parallel architectures.

New GEANT in one picture



Event loop and stacking



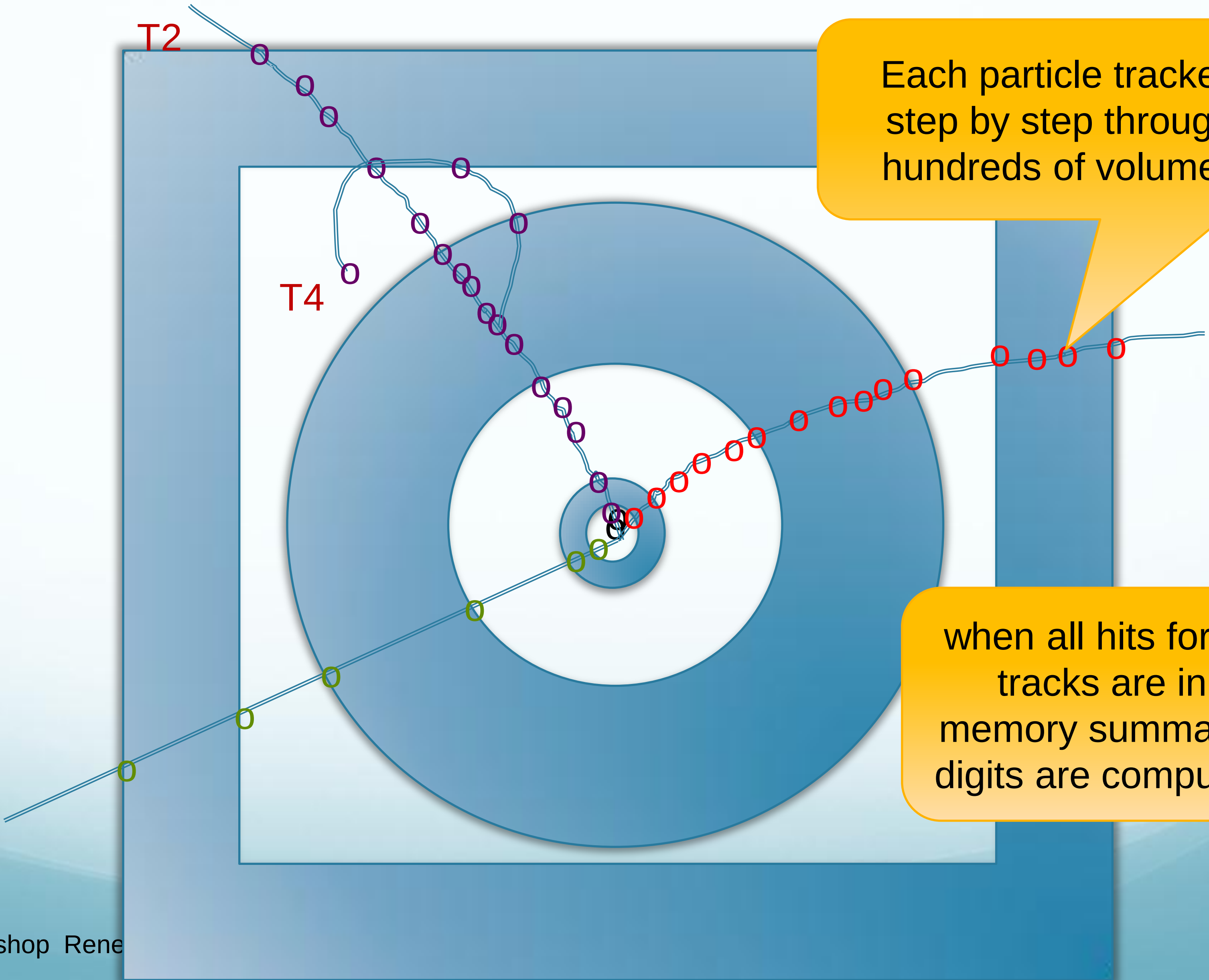
Fast and Full MonteCarlo

- We would like an architecture (via the abstract transporters) where fast and full MC can be run together.
- To make it possible one must have a separate particle stack.
- However, it was clear from the very beginning in January that the particle stack depends strongly on the constraints of parrallelism. Multiple threads cannot update efficiently a tree data structure.

Findings in January

- Decide to concentrate on a very small prototype to test our main ideas.
- No need to import G4 (at least for some time)
- Understanding the geometry of our detectors. We have the real detector geometry of 35 experiments (LHC, LEP, Tevatron, Hera, Babar, etc).
- We rapidly concluded that MASSIVE changes are required in the current simulation strategy to take advantage of the new parallel architectures.
- In this talk, I will discuss mainly the impact of parrallelism.

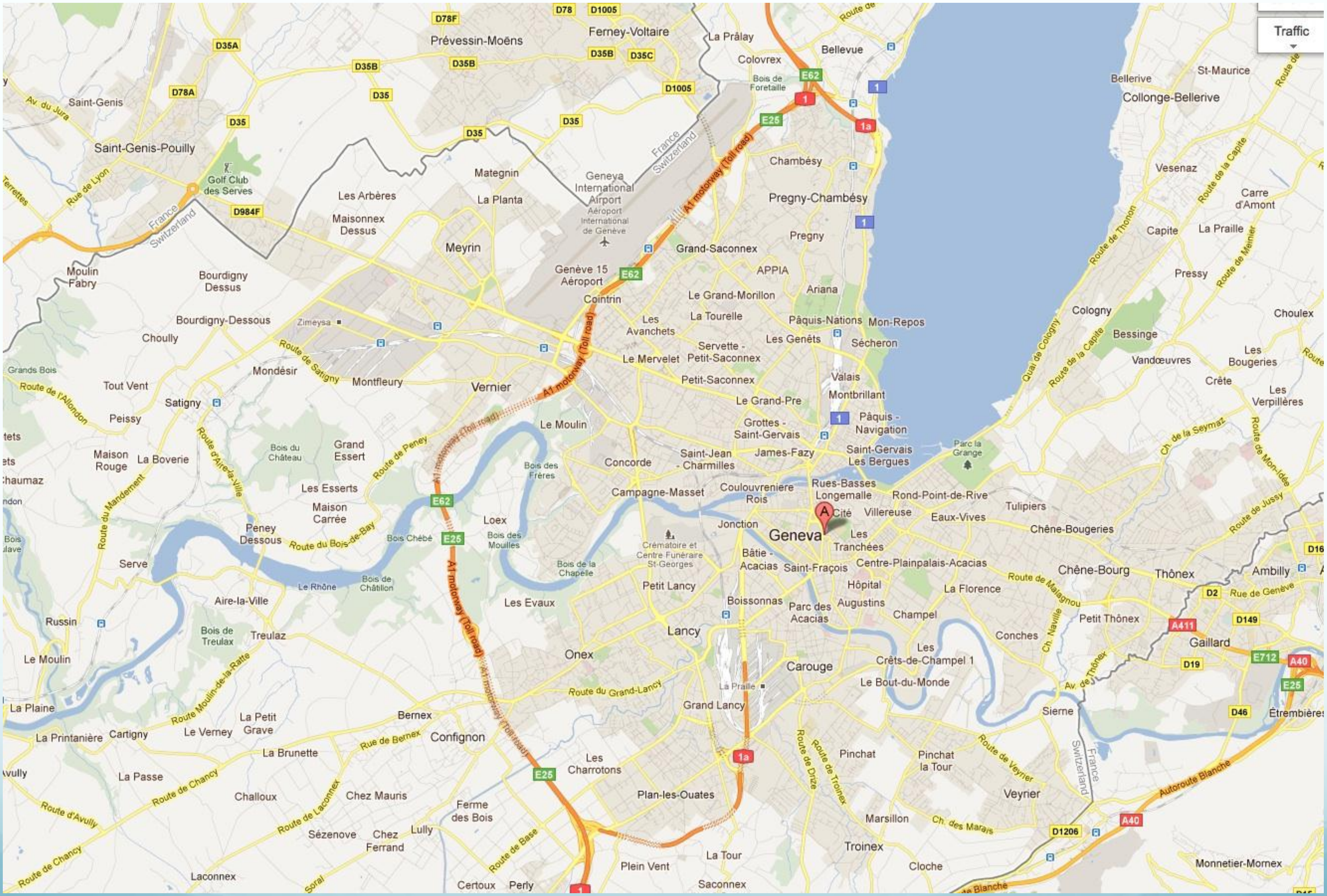
Conventional Transport



Each particle tracked step by step through hundreds of volumes

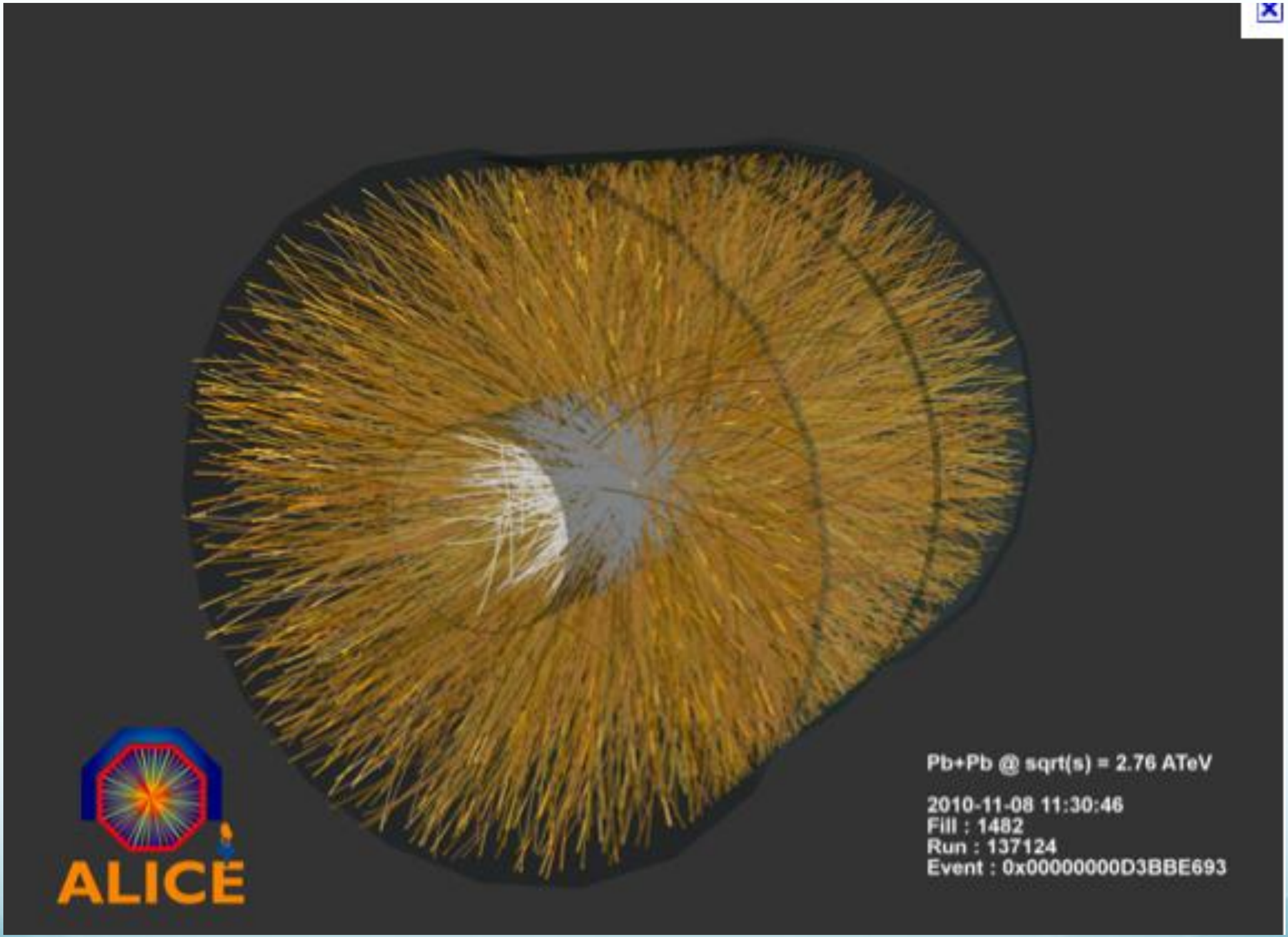
when all hits for all tracks are in memory summable digits are computed

Analogy with car traffic



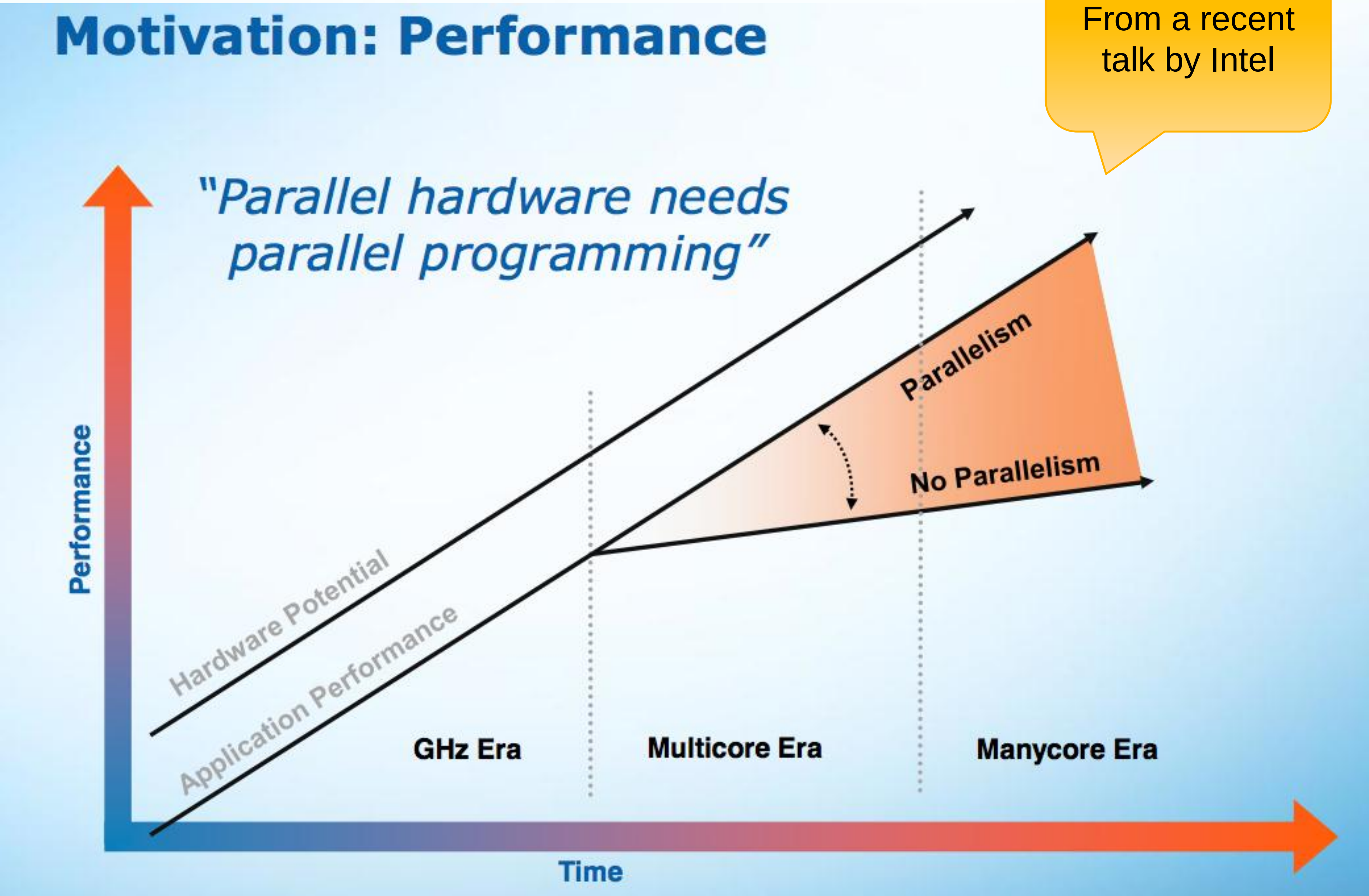
Conventional Transport

- At each step, the navigator *nav has the state of the particle x,y,z,px,py,pz, the volume instance volume*, etc.
- We compute the distance to the next boundary with something like
  - Dist = nav->DistoOut(volume,x,y,z,px,py,pz)
- Or the distance to one physics process with, eg
  - Distp = nav->DistPhotoEffect(volume,x,y,z,px,py,pz)


ALICE
Pb+Pb @ sqrt(s) = 2.76 ATeV
2010-11-08 11:30:46
Fill : 1482
Run : 137124
Event : 0x00000000D3BBE693

parallelism

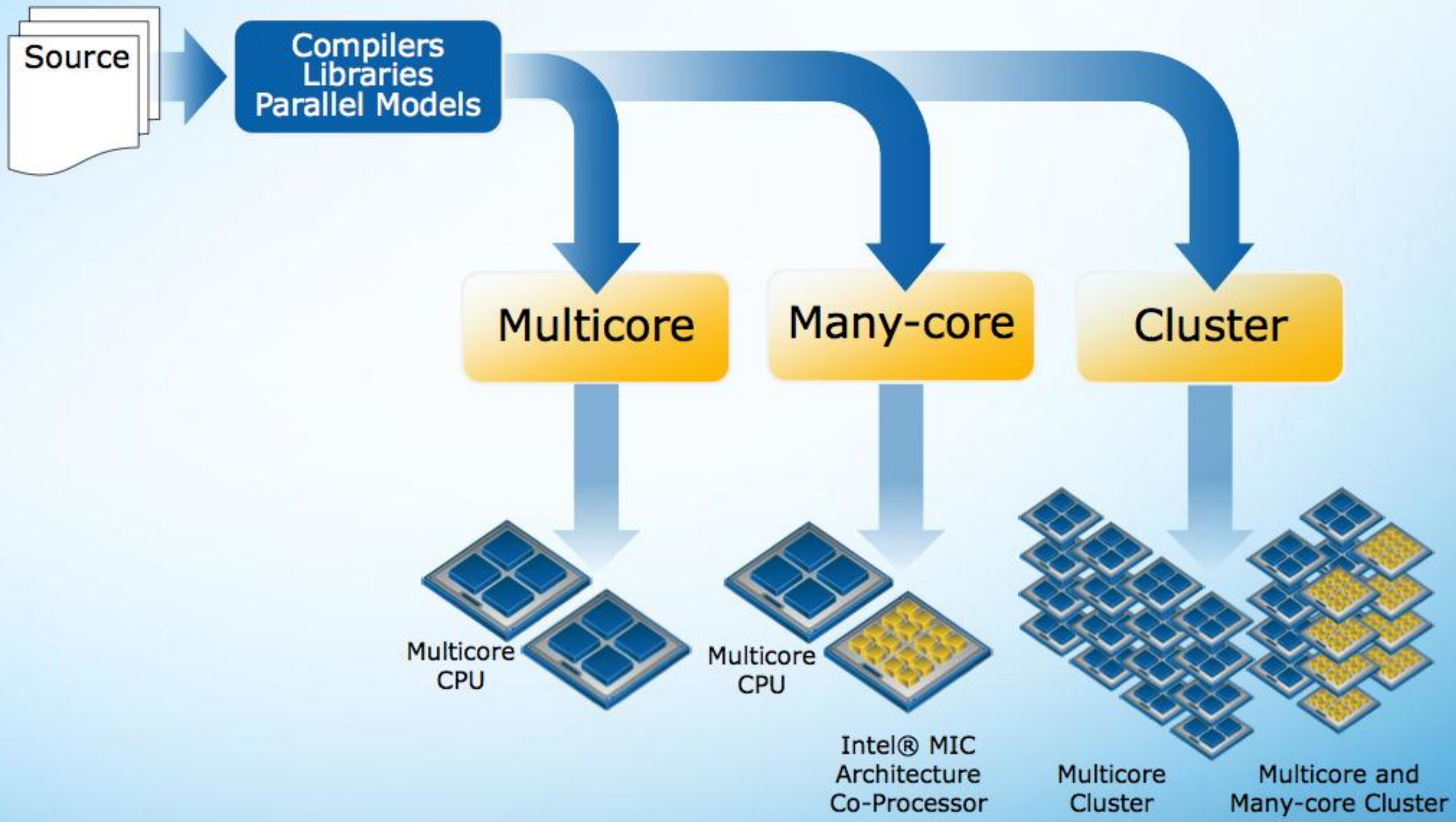
From a recent talk by Intel



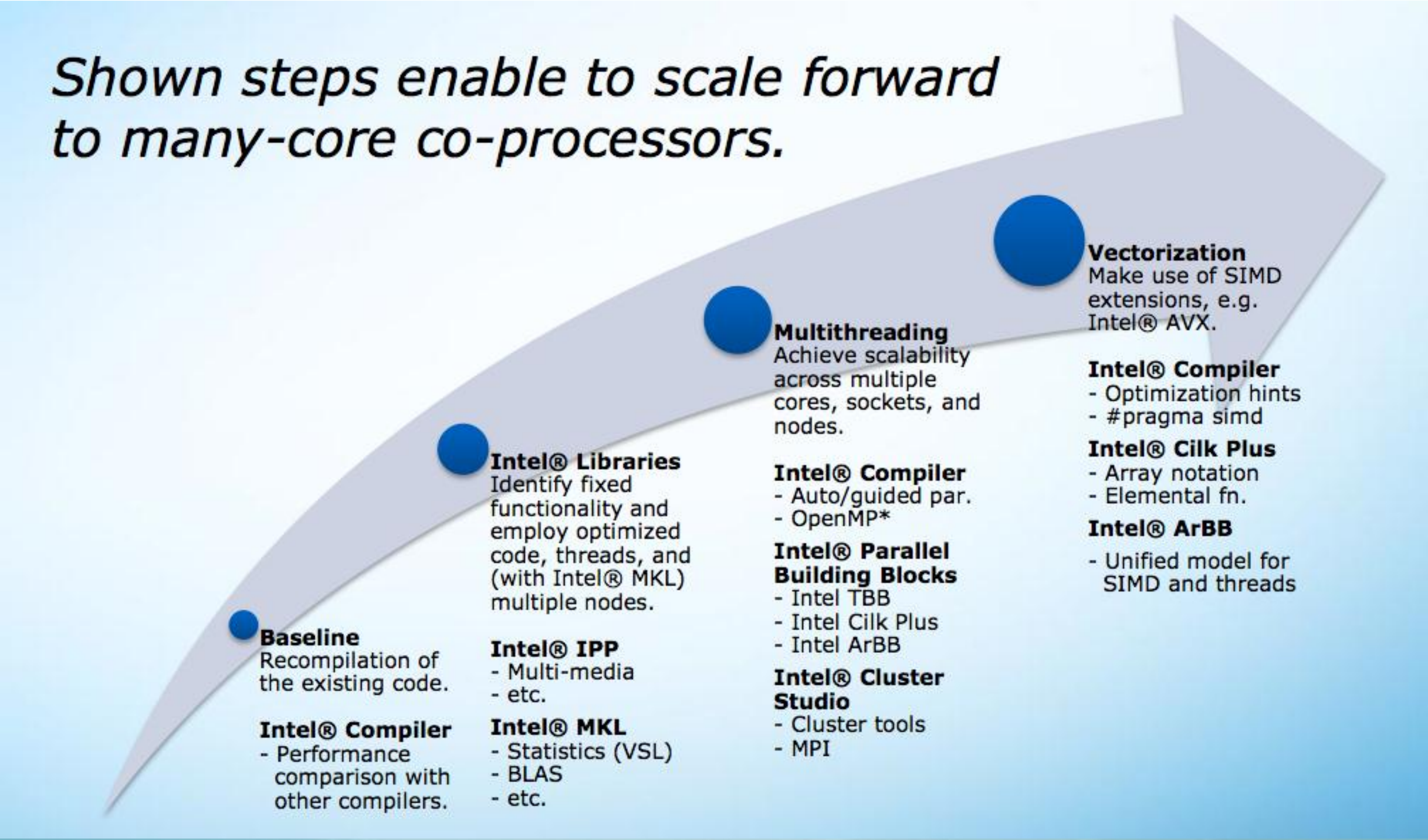
Motivation: Performance

"Parallel hardware needs parallel programming"

If you trust Intel



If you trust Intel 2



Shown steps enable to scale forward to many-core co-processors.

Baseline
Recompilation of the existing code.

Intel® Compiler
- Performance comparison with other compilers.

Intel® Libraries
Identify fixed functionality and employ optimized code, threads, and (with Intel® MKL) multiple nodes.

Intel® IPP
- Multi-media
- etc.

Intel® MKL
- Statistics (VSL)
- BLAS
- etc.

Multithreading
Achieve scalability across multiple cores, sockets, and nodes.

Intel® Compiler
- Auto/guided par.
- OpenMP*

Intel® Parallel Building Blocks
- Intel TBB
- Intel Cilk Plus
- Intel ArBB

Intel® Cluster Studio
- Cluster tools
- MPI

Vectorization
Make use of SIMD extensions, e.g. Intel® AVX.

Intel® Compiler
- Optimization hints
- #pragma simd

Intel® Cilk Plus
- Array notation
- Elemental fn.

Intel® ArBB
- Unified model for SIMD and threads

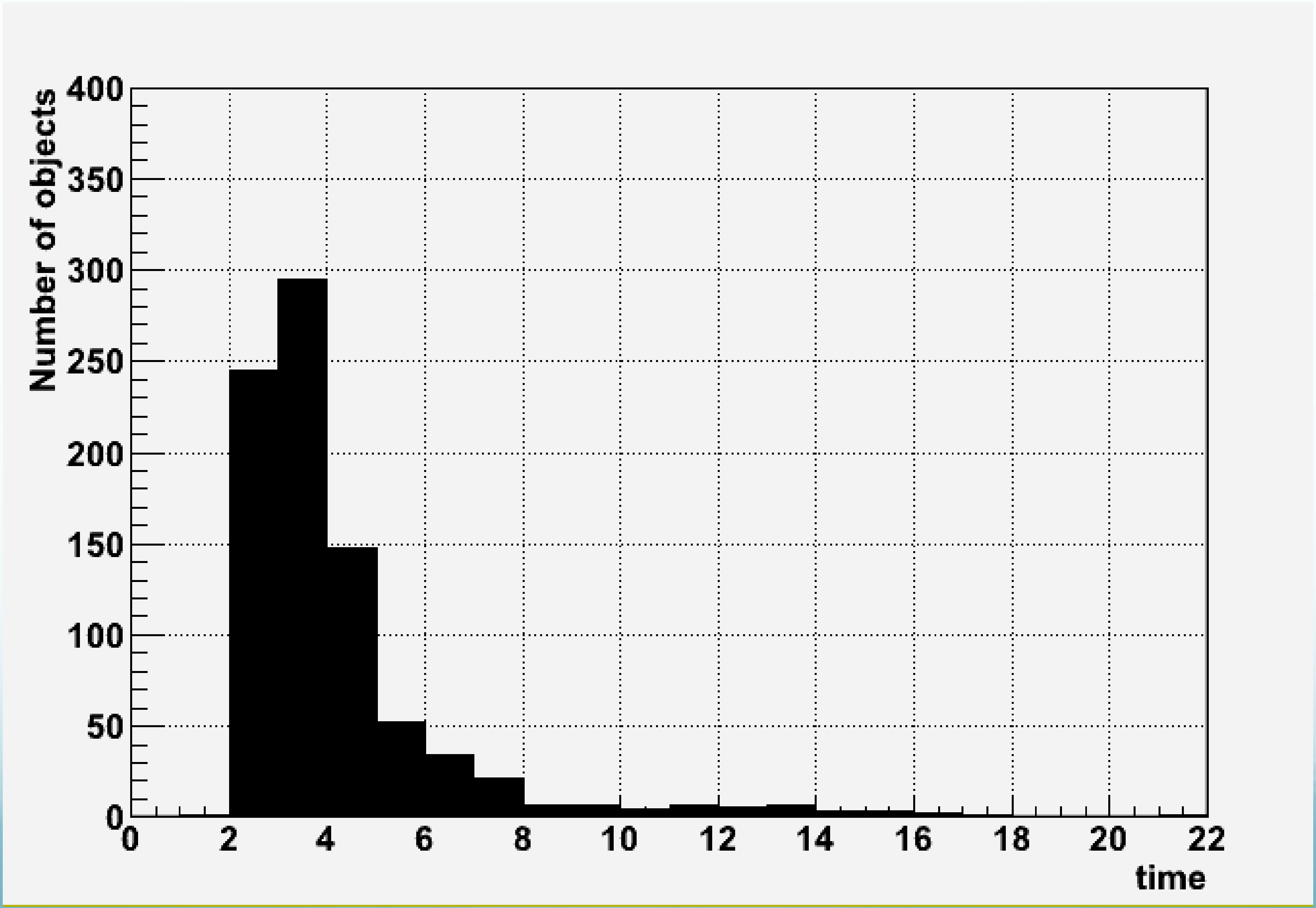
Current Situation

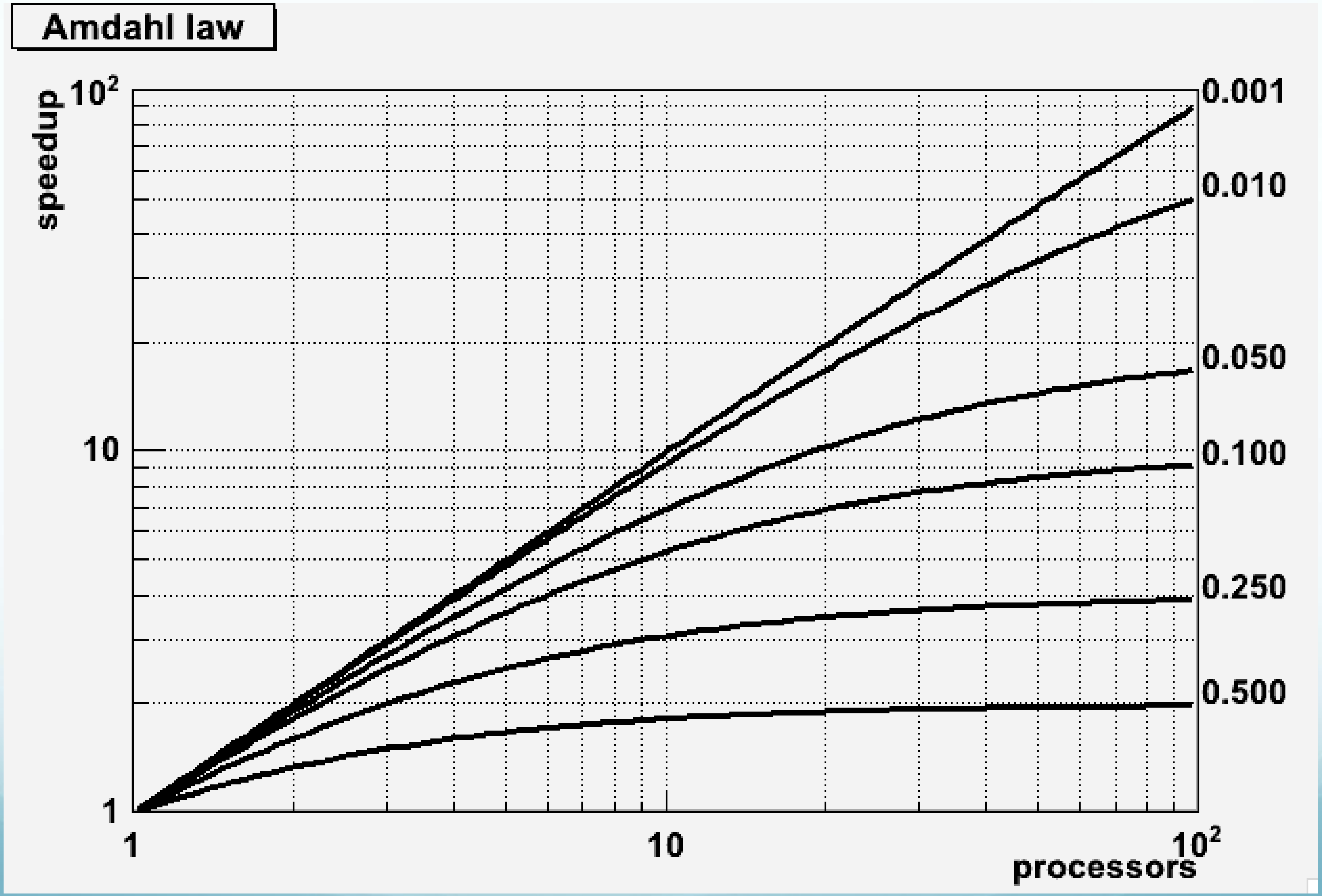
- We run jobs in parallel, one per core.
- Nothing wrong with that except that it does not scale in case of many cores because it requires too much memory.
- A multithreaded version may reduce (say by a factor 2 or 3) the amount of required memory, but also at the expense of performance.
- A multithreaded version does not fit well with a hierarchy of processors.
- So, we have a problem, in particular with the way we have designed some data structures, eg HepMC.

Can we make progress?

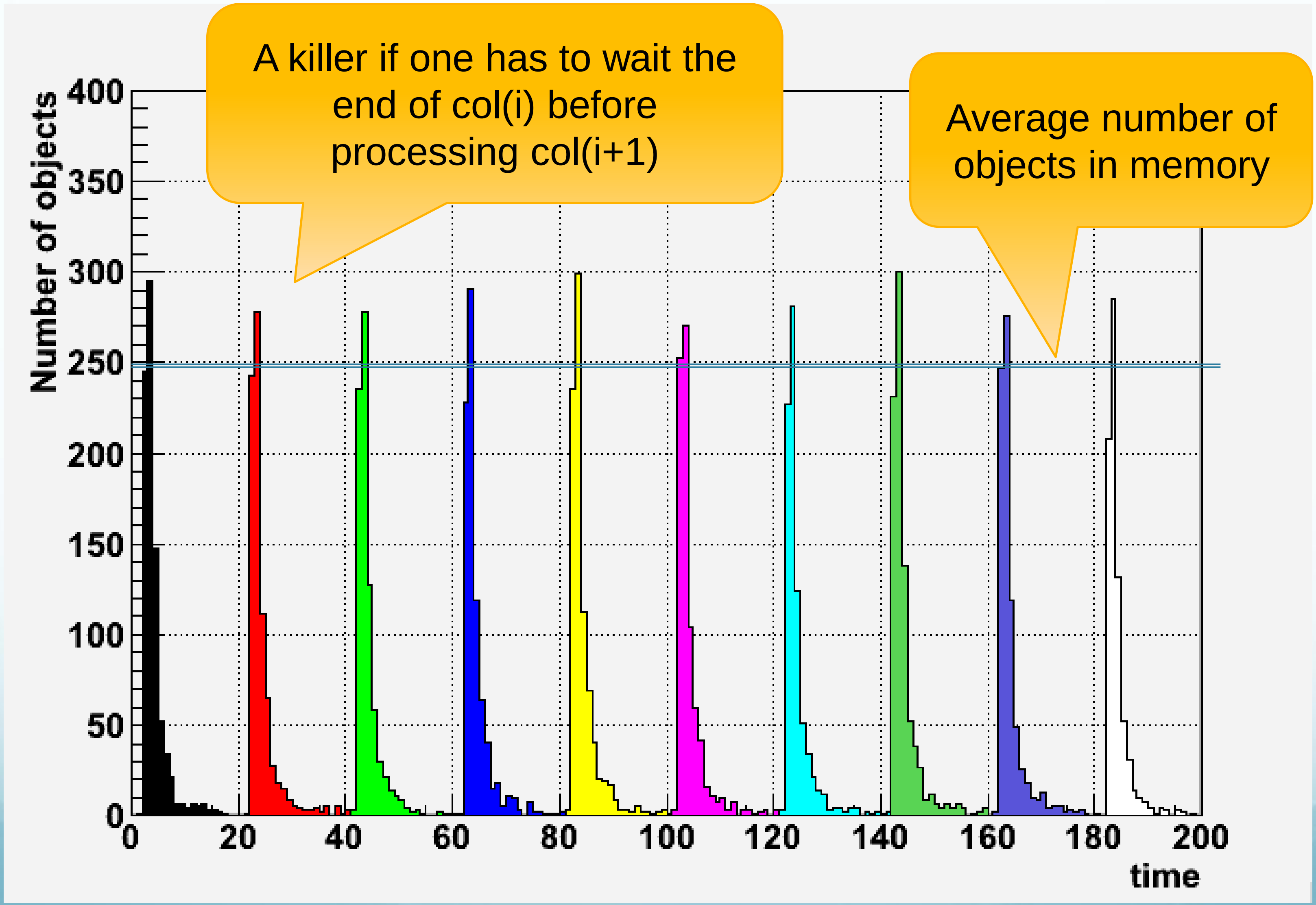
- We need data structures with internal relations only. This can be implemented by using pools and indices.
- When looping on collections, one must avoid the navigation in large memory areas killing the cache.
- We must generate vectors of reasonable size well matched to the degree of parallelism of the hardware and the amount of memory.
- We must find a system to avoid the tail effects

tails, tails, tails

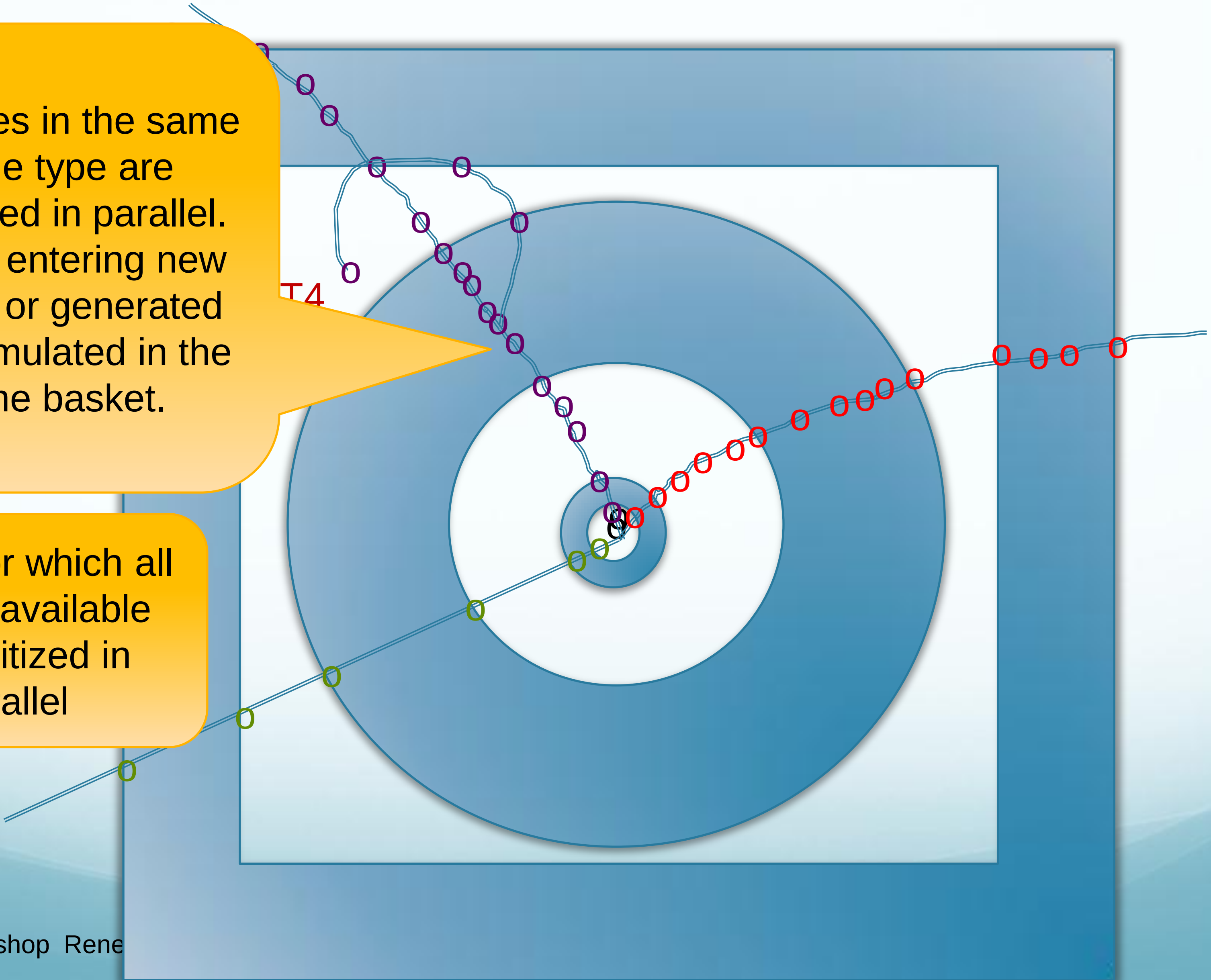



Amdahl law
speedup
10^2
10
1
1
10
10^2
processors
0.001
0.010
0.050
0.100
0.250
0.500

Tails again


A killer if one has to wait the end of col(i) before processing col(i+1)
Average number of objects in memory
Number of objects
time
0
50
100
150
200
250
300
350
400
20
40
60
80
100
120
140
160
180
200

New Transport Scheme



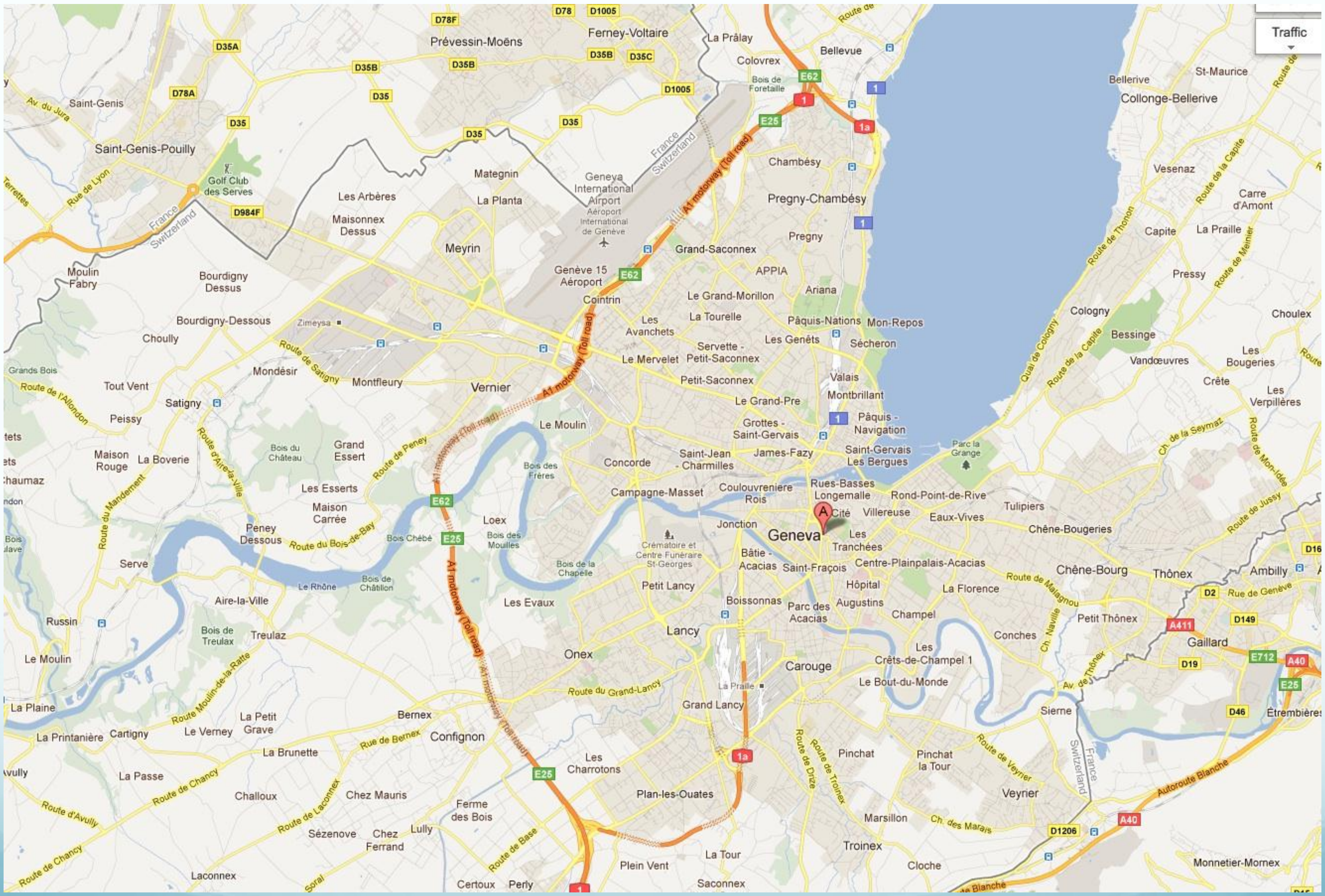
All particles in the same volume type are transported in parallel. Particles entering new volumes or generated are accumulated in the volume basket.

Events for which all hits are available are digitized in parallel

Generations of baskets

- When a particle enters a volume or is generated, it is added to the basket of particles for the volume type.
- The navigator selects the basket with the highest score (with a high and low water mark algorithm).
- The user has the control on the water marks, but the idea that this should be automatic in function of the number of processors and the total amount of memory available. (see interactive demo)

Analogy with car traffic



New Transport

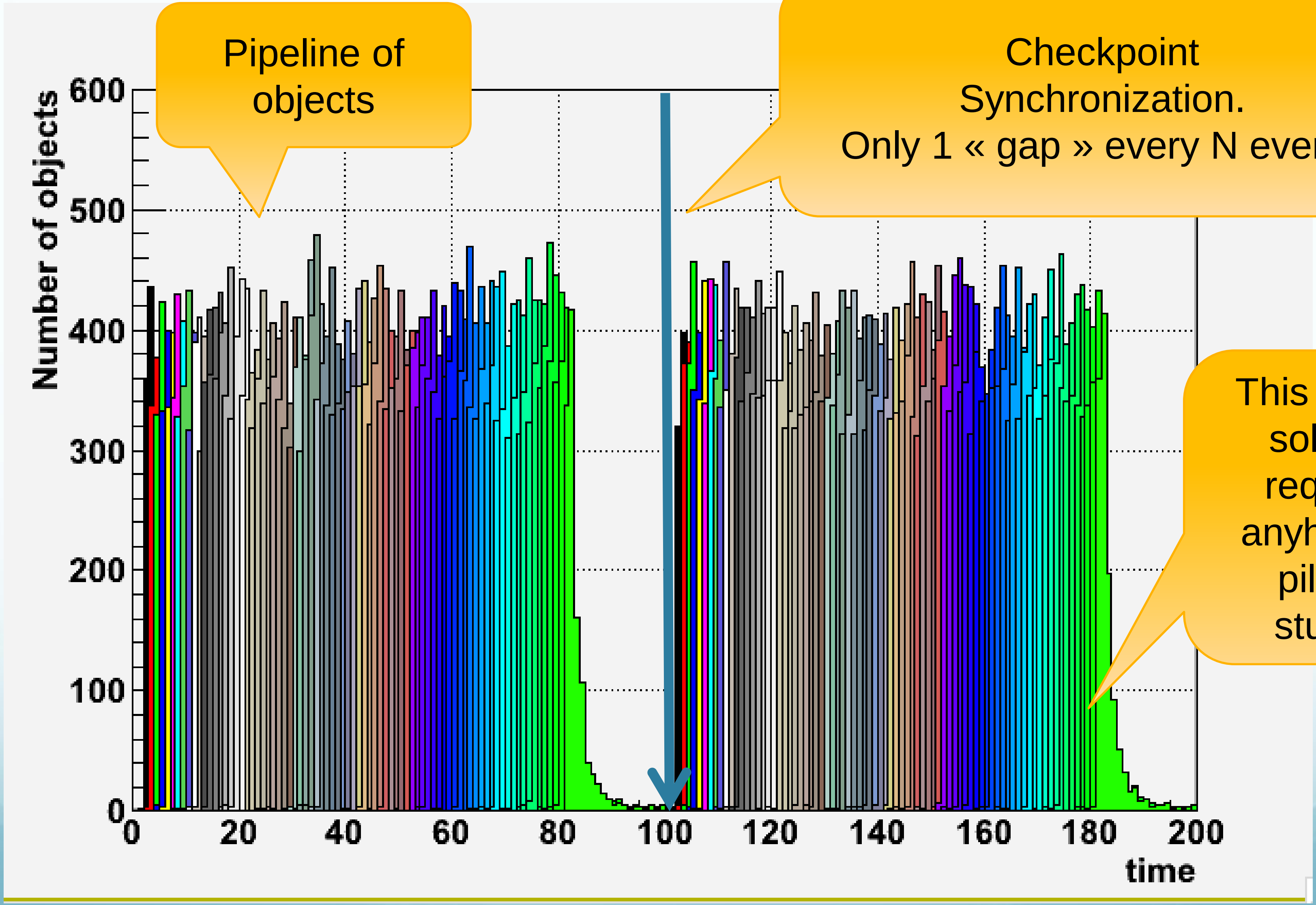
- At each step, the navigator *nav has the state of the particles *x,*y,*z,*px,*py,*pz, the volume instances volume**, etc.
- We compute the distances (array *Dist) to the next boundaries with something like
  - nav->DistoOut(volume,x,y,z,px,py,pz,Dist)
- Or the distances to one physics process with, eg
  - nav->DistPhotoEffect(volume,x,y,z,px,py,pz,DispP)

New Transport

- The new transport system implies many changes in the geometry and physics classes. These classes must be vectorized (a lot of work!).
- Meanwhile we can survive and test the principle by implementing a bridge function like

```
MyNavigator::DisttoOut(int n, TGeoVolume **vol, double *x,..)
{
   for int i=0;i<n;i++)  {
      Dist[i] = DisttoOutOld(vol[i],x[i],…);
   }
 }
```

A better solution

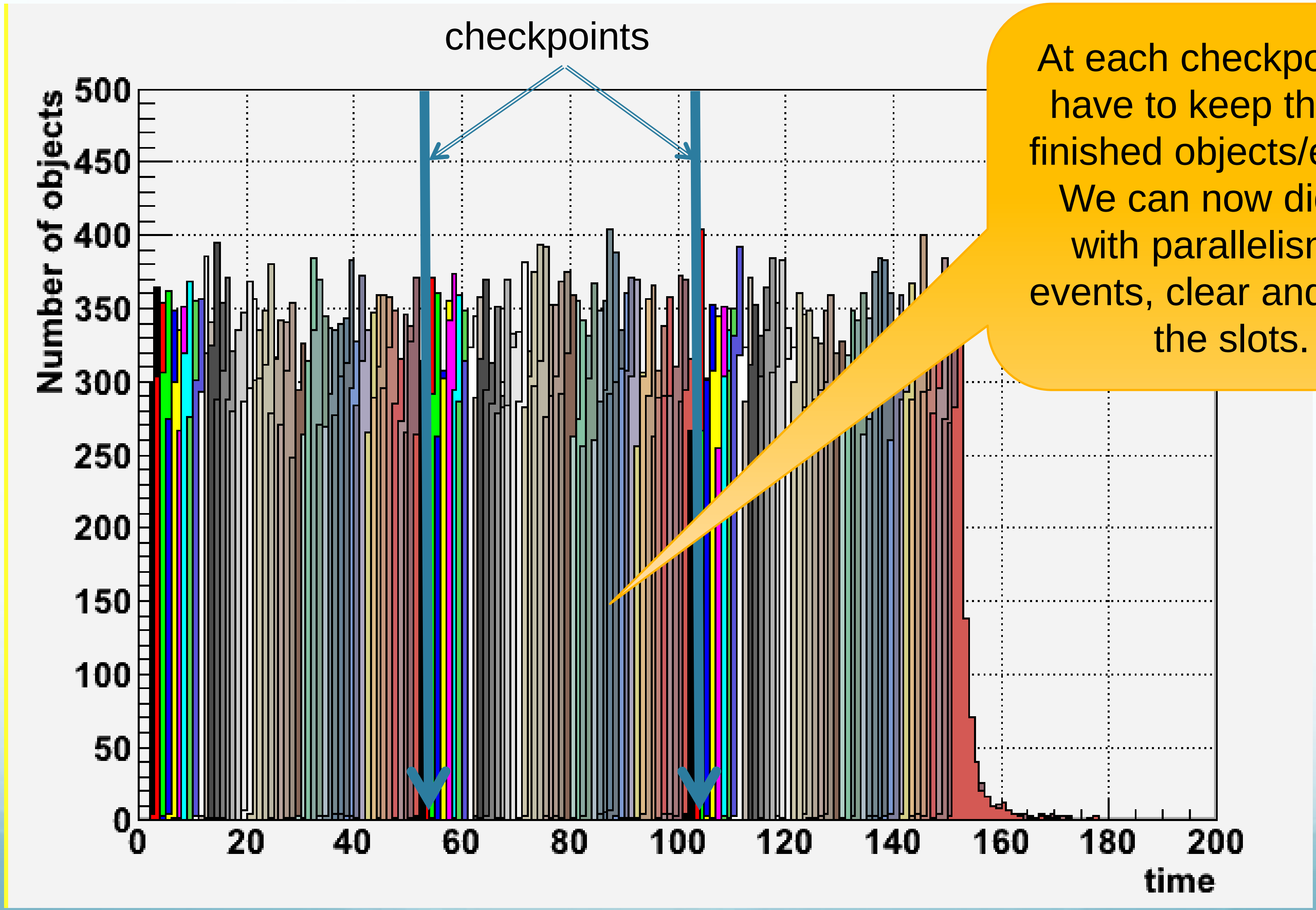


Pipeline of objects

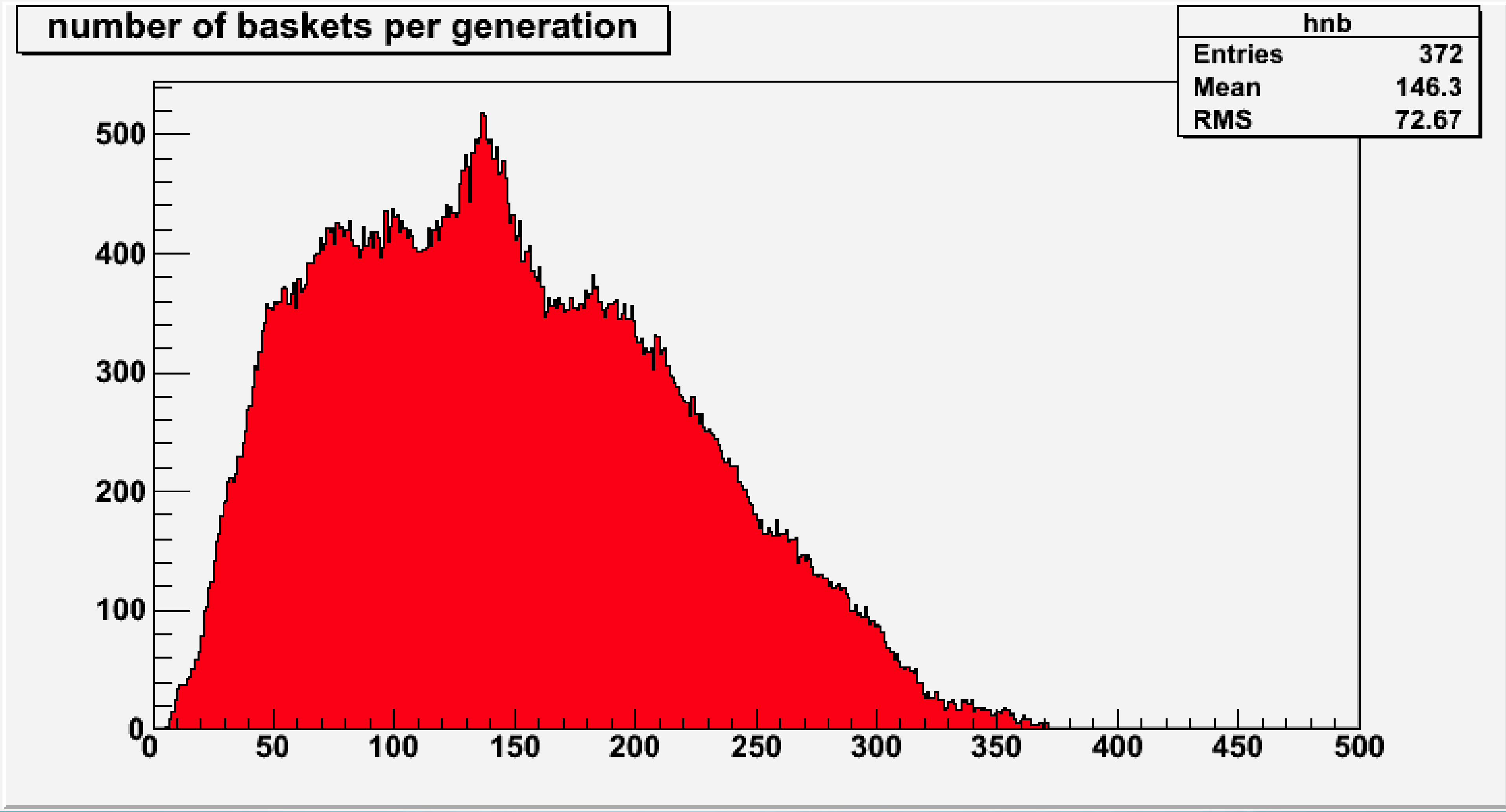
Checkpoint Synchronization. Only 1 « gap » every N events

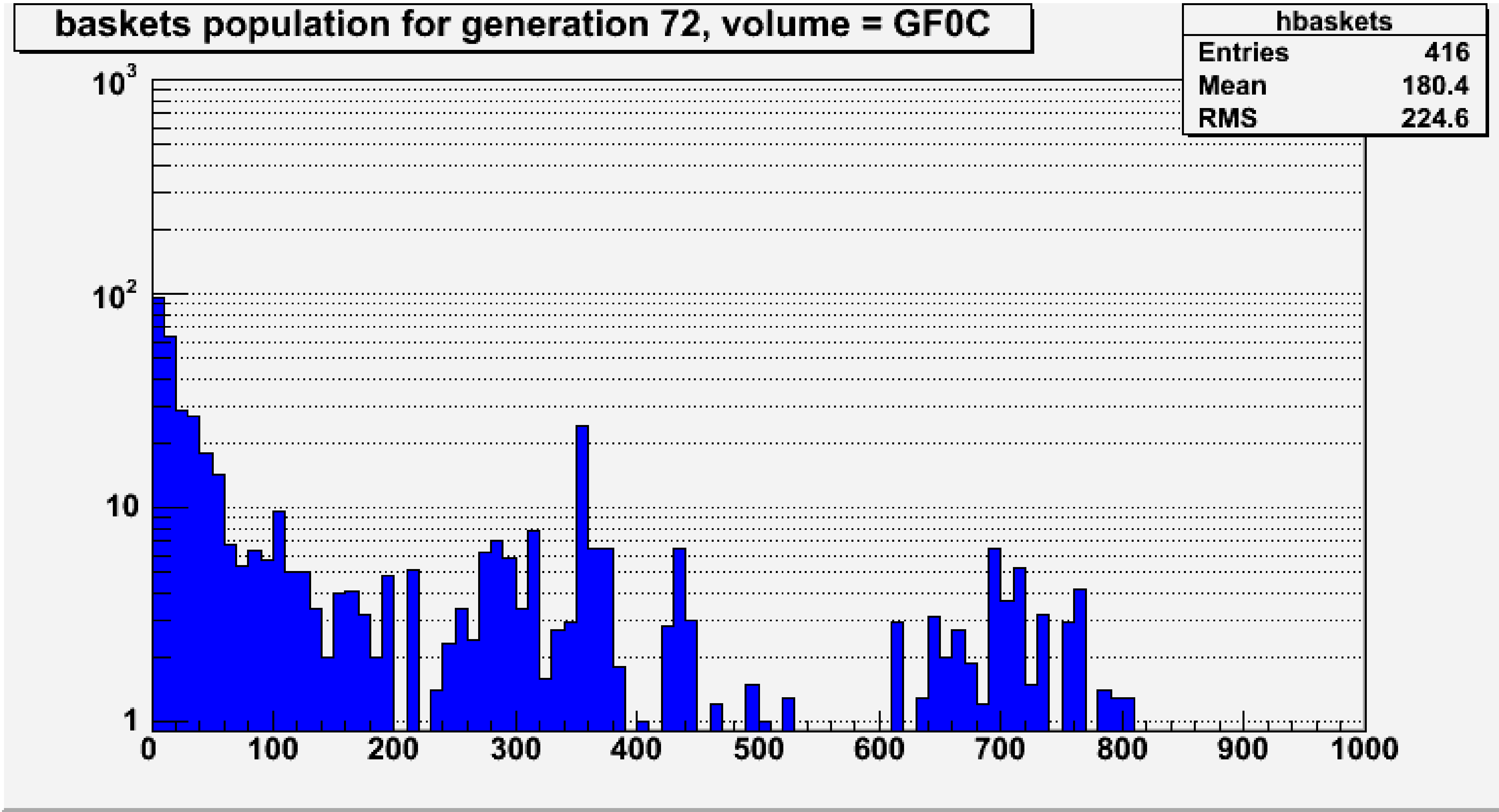
This type of solution required anyhow for pile-up studies

A better better solution



At each checkpoint we have to keep the non finished objects/events. We can now digitize with parallelism on events, clear and reuse the slots.


number of baskets per generation
hnb
Entries 372
Mean 146.3
RMS 72.67
0
100
200
300
400
500
0
50
100
150
200
250
300
350
400
450
500


baskets population for generation 72, volume = GF0C
hbaskets
Entries 416
Mean 180.4
RMS 224.6
10^3
10^2
10
1
0
100
200
300
400
500
600
700
800
900
1000

Vectorizing the geometry (ex1)

```
Double_t TGeoPara::Safety(Double_t *point, Bool_t in) const
{
   // computes the closest distance from given point to this shape.
   Double_t saf[3];
   // distance from point to higher Z face
   saf[0] = fZ-TMath::Abs(point[2]); // Z

   Double_t yt = point[1]-fTyz*point[2];
   saf[1] = fY-TMath::Abs(yt);       // Y
   // cos of angle YZ
   Double_t cty = 1.0/TMath::Sqrt(1.0+fTyz*fTyz);

   Double_t xt = point[0]-fTxz*point[2]-fTxy*yt;
   saf[2] = fX-TMath::Abs(xt);       // X
   // cos of angle XZ
   Double_t ctx = 1.0/TMath::Sqrt(1.0+fTxy*fTxy+fTxz*fTxz);
   saf[2] *= ctx;
   saf[1] *= cty;
   if (in) return saf[TMath::LocMin(3,saf)];
   for (Int_t i=0; i<3; i++) saf[i]=-saf[i];
   return saf[TMath::LocMax(3,saf)];
}
```

Huge performance gain expected in this type of code where shape constants can be computed outside the loop

Vectorizing the geometry (ex2)

```
G4double G4Cons::DistanceToIn( const G4ThreeVector& p,
                               const G4ThreeVector& v   ) const
{
  G4double snxt = kInfinity ;      // snxt = default return value
  const G4double dRmax = 100*std::min(fRmax1,fRmax2);
  static const G4double halfCarTolerance=kCarTolerance*0.5;
  static const G4double halfRadTolerance=kRadTolerance*0.5;

  G4double tanRMax,secRMax,rMaxAv,rMaxOAv ;  // Data for cones
  G4double tanRMin,secRMin,rMinAv,rMinOAv ;
  G4double rout,rin ;

  G4double tolORMin,tolORMin2,tolIRMin,tolIRMin2 ; // `generous' radii squared
  G4double tolORMax2,tolIRMax,tolIRMax2 ;
  G4double tolODz,tolIDz ;

  G4double Dist,s,xi,yi,zi,ri=0.,risec,rhoi2,cosPsi ; // Intersection point vars

  G4double t1,t2,t3,b,c,d ;    // Quadratic solver variables
  G4double nt1,nt2,nt3 ;
  G4double Comp ;

  G4ThreeVector Normal;

  // Cone Precalcs

  tanRMin = (fRmin2 - fRmin1)*0.5/fDz ;
  secRMin = std::sqrt(1.0 + tanRMin*tanRMin) ;
  rMinAv  = (fRmin1 + fRmin2)*0.5 ;

  if (rMinAv > halfRadTolerance)
  {
    rMinOAv = rMinAv - halfRadTolerance ;
  }
  else
  {
    rMinOAv = 0.0 ;
  }
  tanRMax = (fRmax2 - fRmax1)*0.5/fDz ;
  secRMax = std::sqrt(1.0 + tanRMax*tanRMax) ;
  rMaxAv  = (fRmax1 + fRmax2)*0.5 ;
  rMaxOAv = rMaxAv + halfRadTolerance ;

  // Intersection with z-surfaces

  tolIDz = fDz - halfCarTolerance ;
  tolODz = fDz + halfCarTolerance ;

……  //here starts the real algorithm
```

All these statements are independent of the particle !!!

Huge performance gain expected in this type of code where shape constants can be computed outside the loop

Vectorizing the Physics

- This is going to be more difficult when extracting the physics classes from G4. However important gains are expected in the functions computing the distance to the next interaction point for each process.
- There is a diversity of interfaces and we have now sub-branches per particle type.

Status and next Steps

- Consolidation of the prototype.
- Implementation of the sliding objects.
- Web site construction with a description of the current status and goals. (now)
- Thread safety of TGeo (now in a good shape)
- Vectorization of TGeo (at least a critical subpart)
- Discussion with the G4 team about the consequences for the G4 physics classes.