The path toward High Performance Computing in High Energy Physics

Federico Carminati for the GeantV project

SISSA, Feb 24-26, 2016

THE GEANTV PROJECT

- G.Amadio (UNESP), Ananya (CERN), J.Apostolakis (CERN) , A.Arora (CERN), M.Bandieramonte (CERN), A.Bhattacharyya (BARC), C.Bianchini (UNESP), R.Brun (CERN), P.Canal (FNAL), F.Carminati (CERN), L.Durhem (intel), D.Elvira (FNAL), A.Gheata (CERN), M.Gheata (CERN), I.Goulas (CERN), R.Iope (UNESP), S.Jun (FNAL), G.Lima (FNAL), A.Mohanty (BARC), T.Nikitina (CERN), M.Novak (CERN), W.Pokorski (CERN), A.Ribon (CERN), R.Sehgal (BARC), O.Shadura (CERN), S.Vallecorsa (CERN), S.Wenzel (CERN), Y.Zhang (CERN)

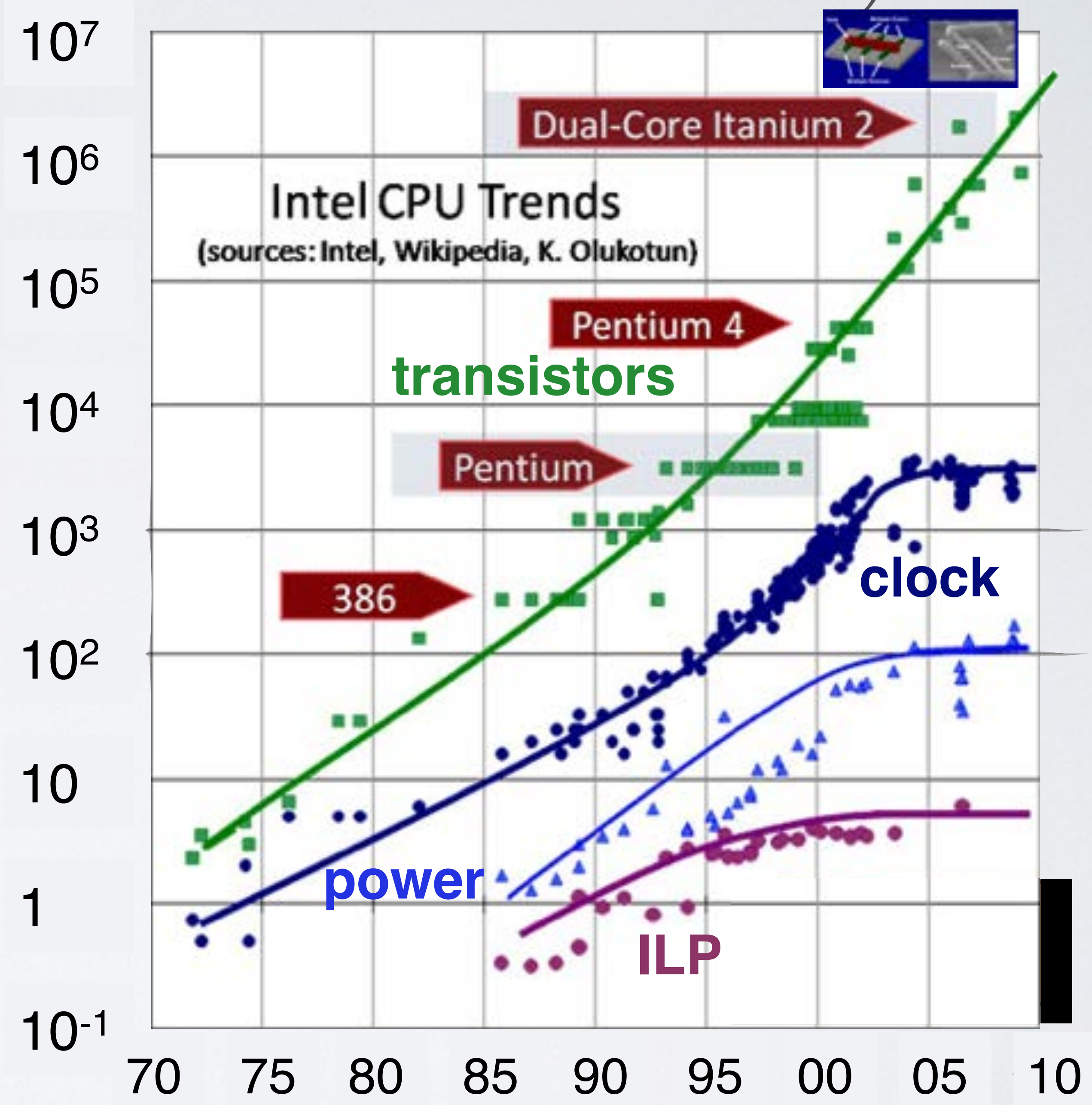
WHY DO WE NEED HPC?

- WLCG
  - 170 computing centres in 42 countries
  - ~500k Cores
  - O(200) PB on disk

MOTIVATIONS

(EVEN IF YOU ARE FAMILIAR WITH THEM)

- Performance of our code almost stagnant
- SIMD is key to achieve the performance we need (10x)
- Portability, better physics and optimisation will be the targets
- Simulation
    - 50% of the LWGC cycles
    - can show how to better exploit CPUs for complex applications



THE EIGHT DIMENSIONS

- The "dimensions of performance"
  - Vectors
  - Instruction Pipelining
  - Instruction Level Parallelism (ILP)
  - Hardware threading
  - Clock frequency
  - Multi-core
  - Multi-socket
  - Multi-node

Micro-parallelism: gain in throughput and in time-to-solution

Very little gain to be expected and no action to be taken

Gain in memory footprint and time-to-solution but not in throughput

Possibly running different jobs as we do now is the best solution

Expected limits on performance scaling

	SIMD	ILP	HW
THEORY	8	4	1.35
OPTIMISED	6	1.57	1.25
HEP	1	0.8	1.25

Expected limits on performance scaling (multiplied)

	SIMD	ILP	HW
THEORY	8	32	43.2
OPTIMISED	6	9.43	11.79
HEP	1	0.8	1

OpenLab@CHEP12

WHY IT IS SO DIFFICULT?

- No clear kernel
- C++XX code generation / optimisation not well understood
- Most of the technology is coming out now
  - Lack of standards
  - Technological risk
- Non professional coders
- Fast evolving code
- No control on hardware acquisition

WHY SIMULATION?

- Simulation can be developed in a "system independent" way
- It can be prototyped with "little I/O" (at the beginning)
- The LHC experiments use extensively G4 as main simulation engine. They have invested in validation procedures
- Experiments develop their own fast MC solution as a full simulation is too slow for several physics analysis
- We need an architecture where fast and full MC can be run together with the highest performance on parallel systems

WHY GEANTV?

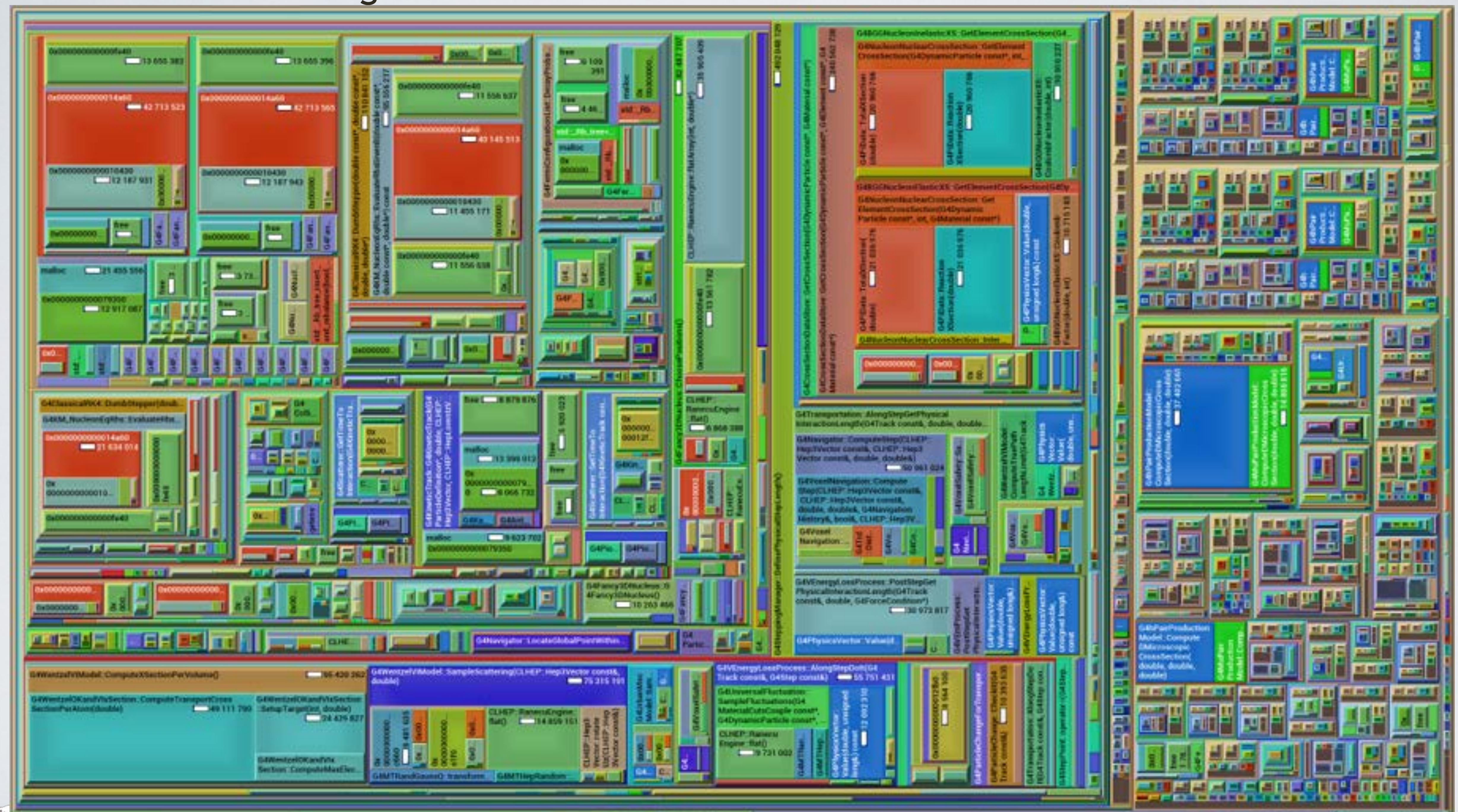
- Simulation is largely experiment independent
- Geant4 is THE simulation code for HEP
- 50% of the WLCG cycles are used by simulation

WHY NOT GEANT4+?

- Extensive prototyping and analysis has convinced us that "vectorisation" of Geant4 was not achievable without a major rewrite of the code
  - No hotspots (!)
  - Virtual table structure very deep and complex (1990's style)
  - Codebase very large and non-homogeneous
- No criticism, but even the best things age

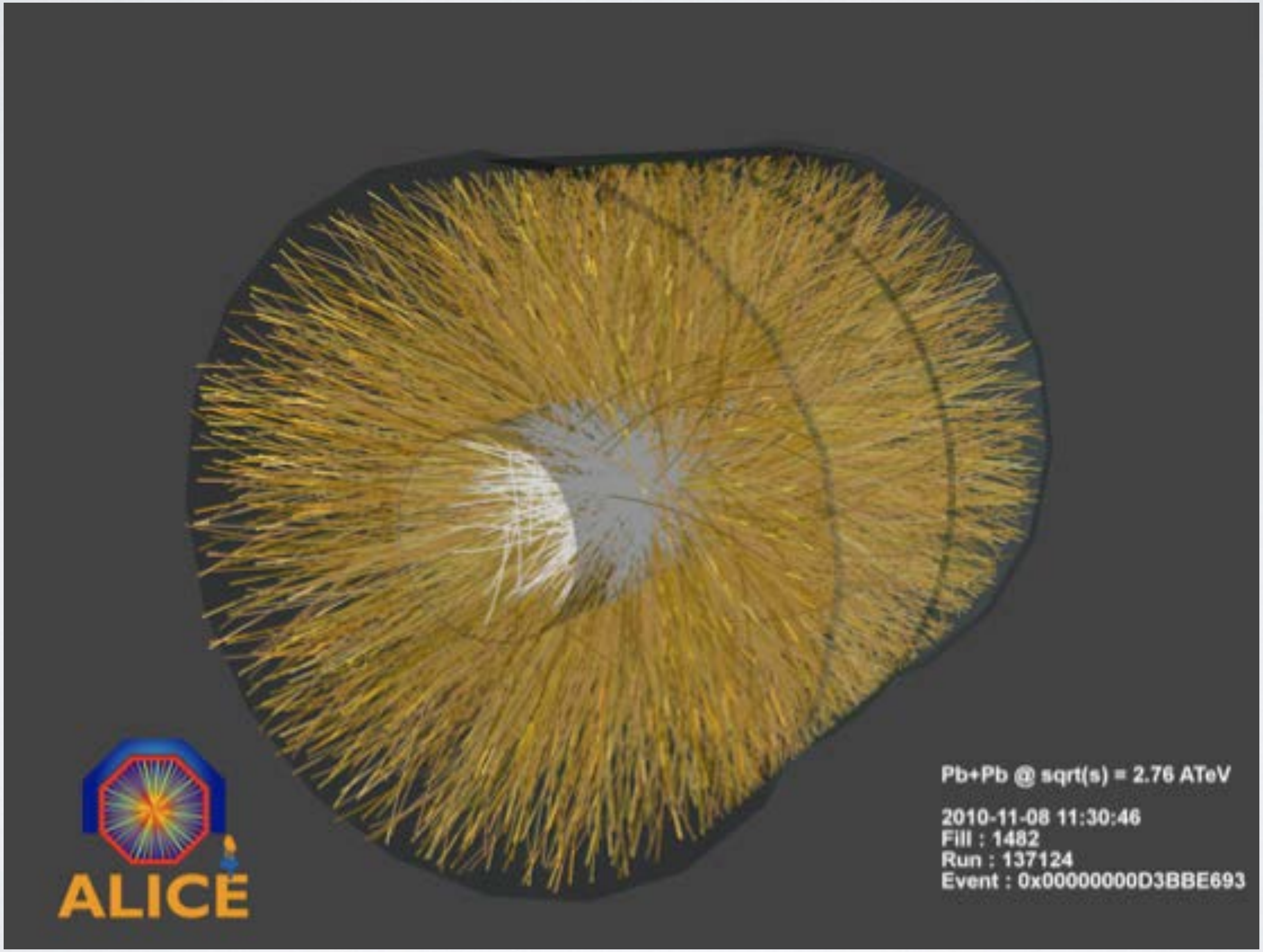
GEANT4 PROFILING EXAMPLE: CALL MAP

valgrind / kcachegrind



Anything above 10% is a pathology!

PARALLELISM EVERYWHERE AGAIN… BUT HOW TO EXPLOIT IT?

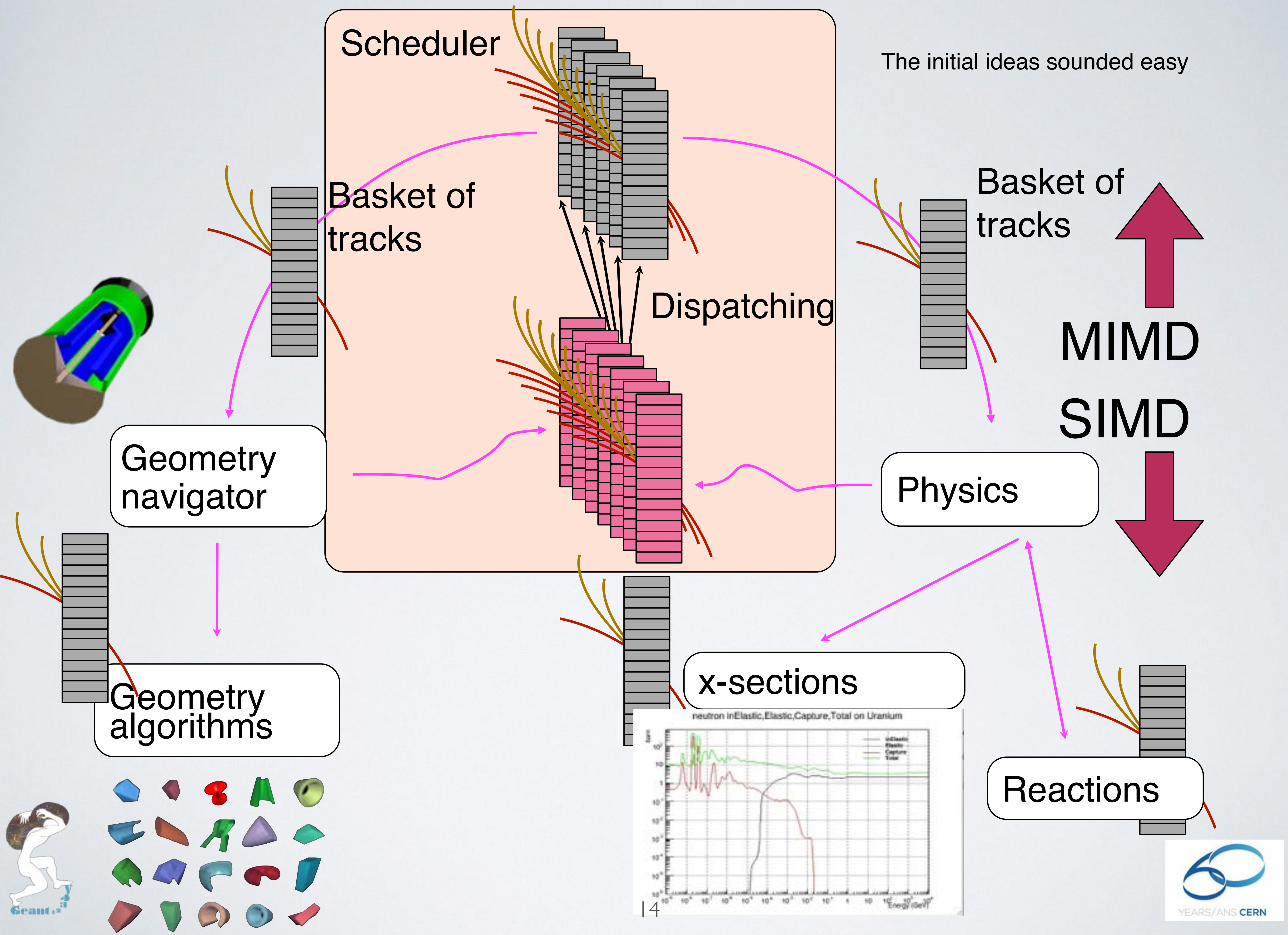


?

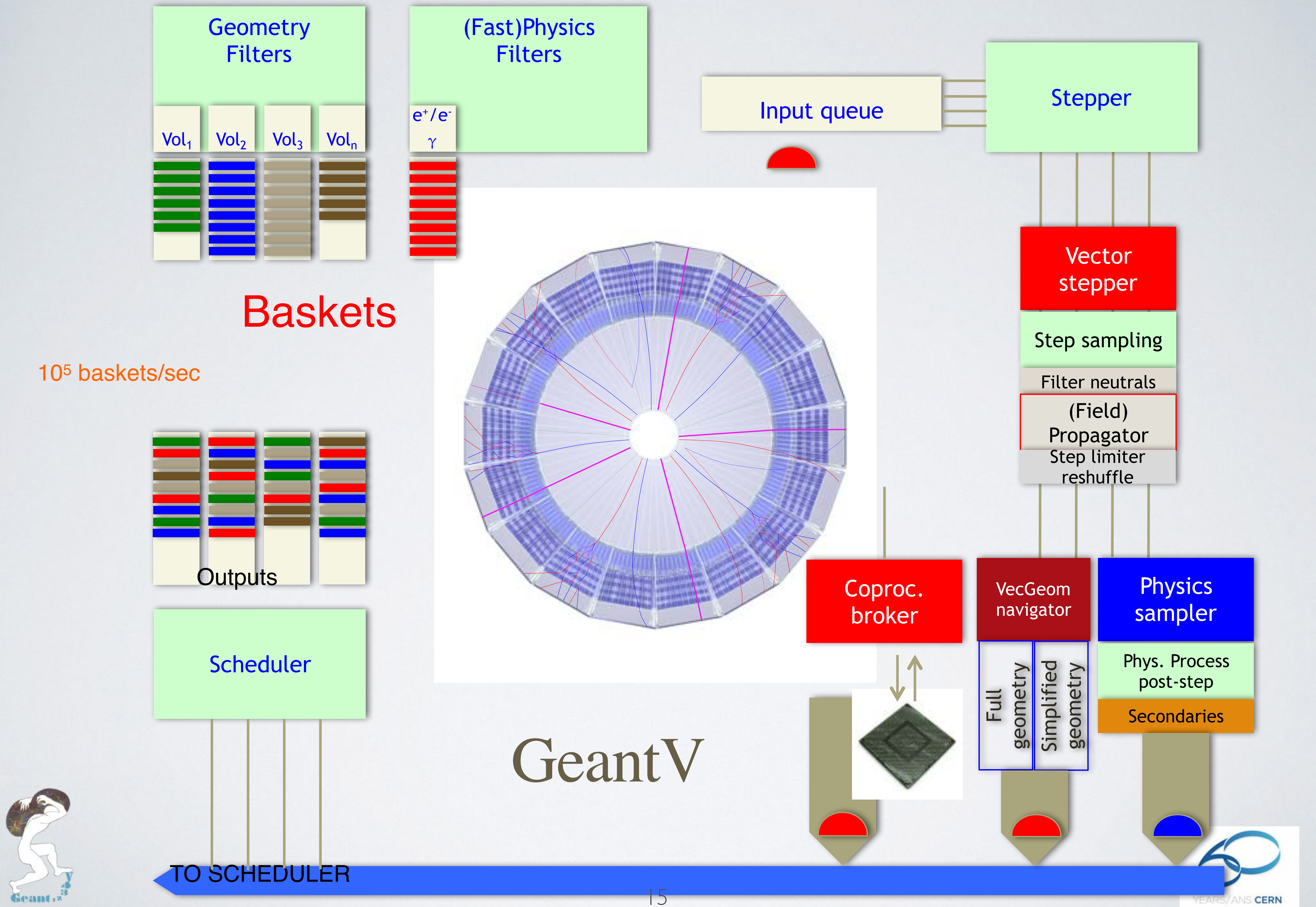
In some sense… but not entirely
(see later)

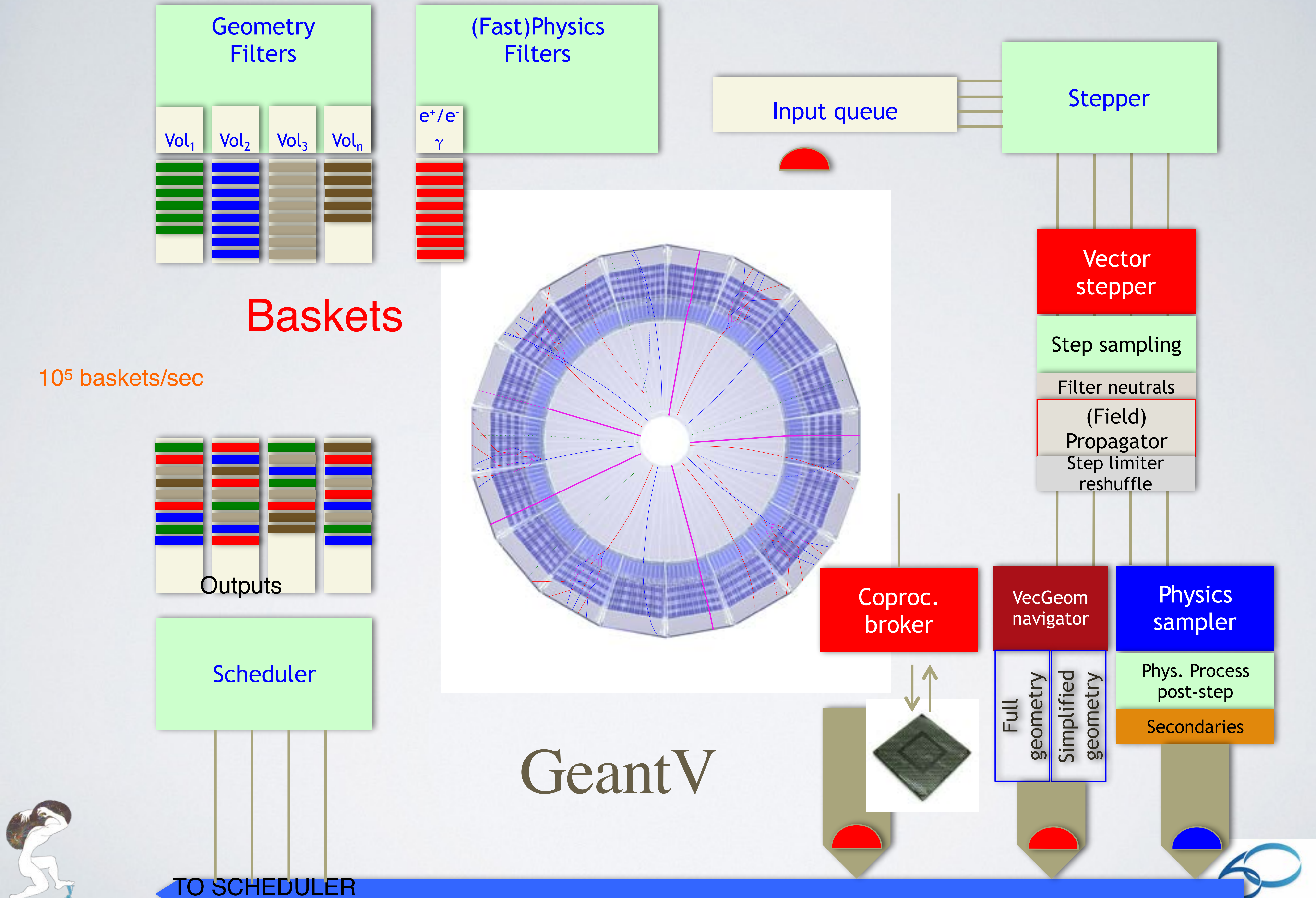
WHAT DO WE WANT TO DO?

- Develop an all-particle transport simulation programme with
  - A code 2-5 times faster than Geant4
  - Continue improvement of physics
  - Full simulation and various fast simulation options
  - Portable on different architectures (CPUs, GPUs and Xeon Phi's)
- Understand the limiting factors for a (10x) improvement



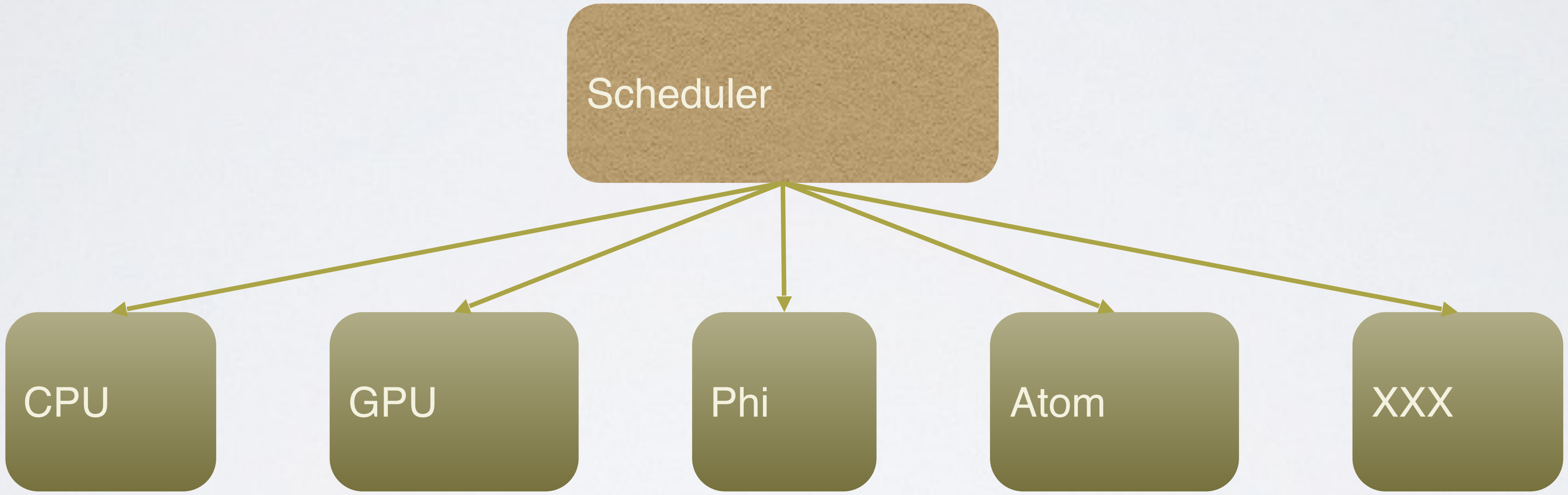
The initial ideas sounded easy
Scheduler
Basket of tracks
Basket of tracks
Dispatching
MIMD
SIMD
Geometry navigator
Physics
Geometry algorithms
x-sections
neutron inElastic,Elastic,Capture,Total on Uranium
Reactions


Geometry Filters
Vol$_1$
Vol$_2$
Vol$_3$
Vol$_n$
(Fast)Physics Filters
e^+/e^-
γ
Input queue
Stepper
Baskets
10^5 baskets/sec
Vector stepper
Step sampling
Filter neutrals
(Field) Propagator
Step limiter reshuffle
Outputs
Scheduler
Coproc. broker
VecGeom navigator
Physics sampler
Full geometry
Simplified geometry
Phys. Process post-step
Secondaries
GeantV
TO SCHEDULER


Geometry Filters
Vol1
Vol2
Vol3
Voln
(Fast)Physics Filters
e+/e-
γ
Input queue
Stepper
Baskets
10⁵ baskets/sec
Vector stepper
Step sampling
Filter neutrals
(Field) Propagator
Step limiter reshuffle
Outputs
Scheduler
GeantV
Coproc. broker
VecGeom navigator
Full geometry
Simplified geometry
Physics sampler
Phys. Process post-step
Secondaries
TO SCHEDULER

CHALLENGES

- Overhead from reshuffling particle lists should not offset SIMD gains
- Exploit the metal at its best, while maintaining portability



- Test from the onset on a "large" setup (LHC-like detector)
  - Toy models tell us very little – complexity is the problem

http://code.compeng.uni-frankfurt.de/projects/vc

PORTABILITY

- Long-term maintainability of the code => write one single version of each algorithm and to specialise it to the platform via template programming and low level optimised libraries (Vc in our case)
- A Xeon Phi specific backend is being developed
- Results are quite encouraging: maybe portable HPC is NOT an oxymoron after all…

Here how

“Backend” is a (trait) struct encapsulating standard types/properties for “scalar, vector, CUDA” programming; makes information injection into template function easy

1 particle API

Many particle API (SIMD)

Common C++ template functions

```
double distance( double );
```

```
Vc::double_v distance( Vc::double_v );
```

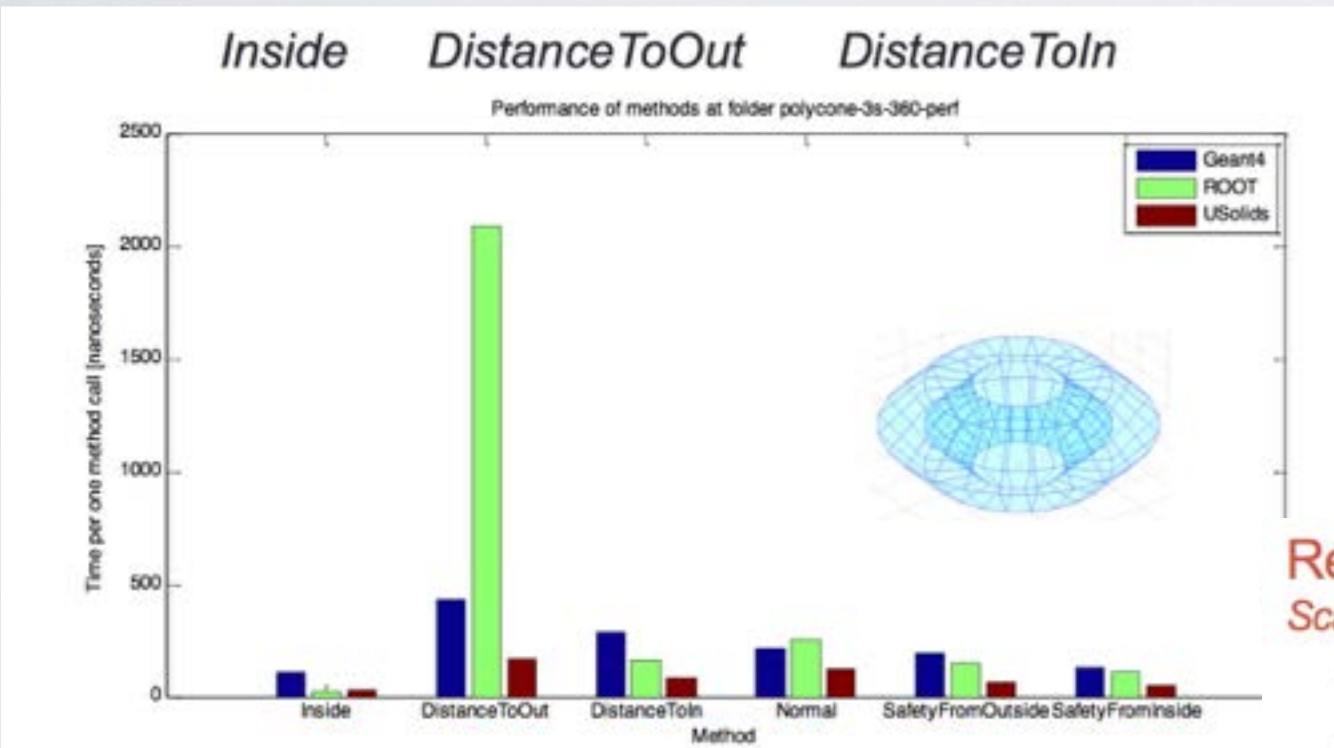
```
template<class Backend>
Backend::double_t
common_distance_function( Backend::double_t input
)
{
   // Algorithm using Backend types
}
```

```
struct ScalarBackend
{
   typedef double double_t;
   typedef bool   bool_t;
   static const bool IsScalar=true;
   static const bool IsSIMD=false;
};
```

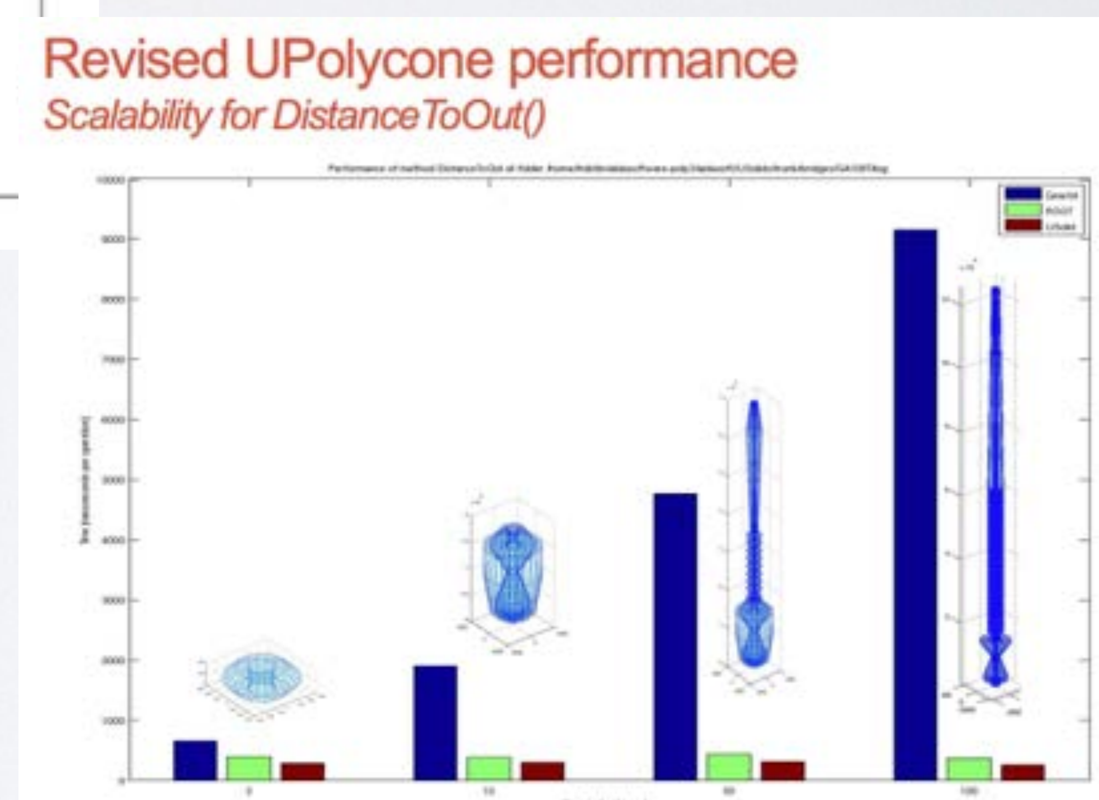
```
struct VectorBackend
{
   typedef Vc::double_v double_t;
   typedef Vc::bool_v bool_t;
   static const bool IsScalar=false;
   static const bool IsSIMD=true;
};
```

ANOTHER EXAMPLE POLYCONE

Speedup factor **3.3x** vs. Geant4, **7.6x** vs. Root • for most performance critical methods, i.e.:



- It is today possible to run Geant4 simulations with USolids shapes replacing Geant4 shapes (seamless to user)
- Geant4 10.1. ships USolids internally optionally one may also compile against external USolids installation



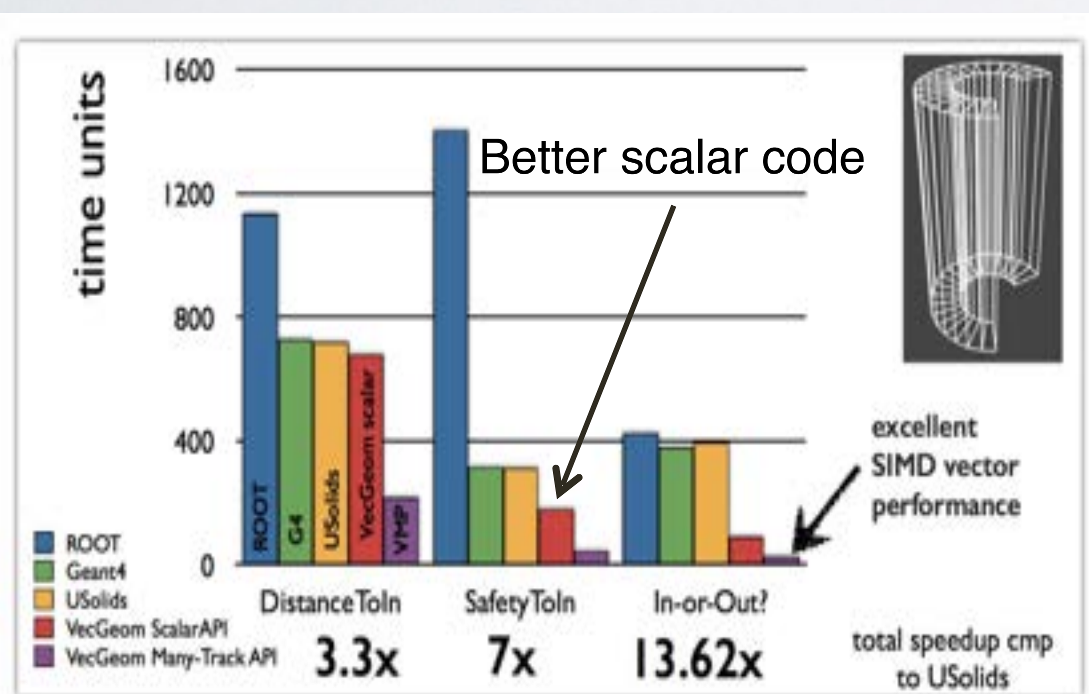
- USolids source code repository: gitlab.cern.ch/VecGeom/VecGeom

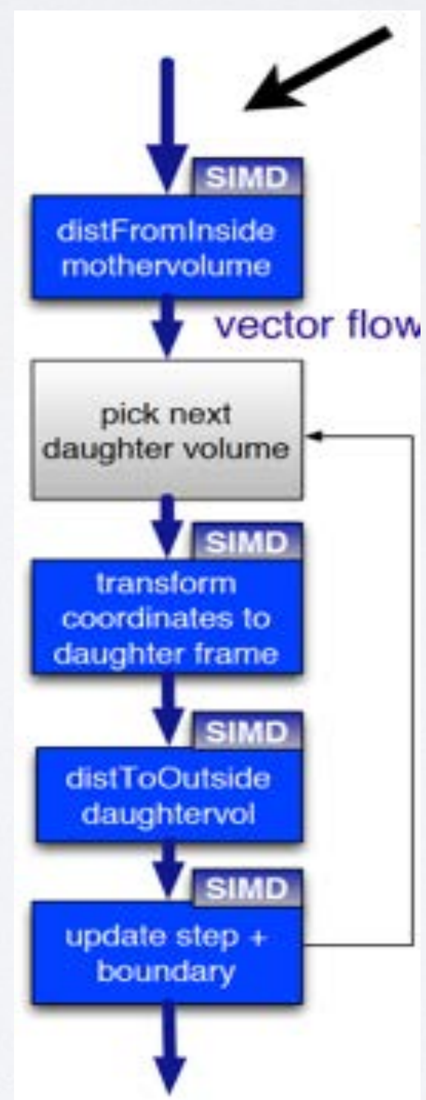
Ugeom/VecGeom is developed by the AIDA project.

GEOMETRY PERFORMANCE (PHI VS XEON)

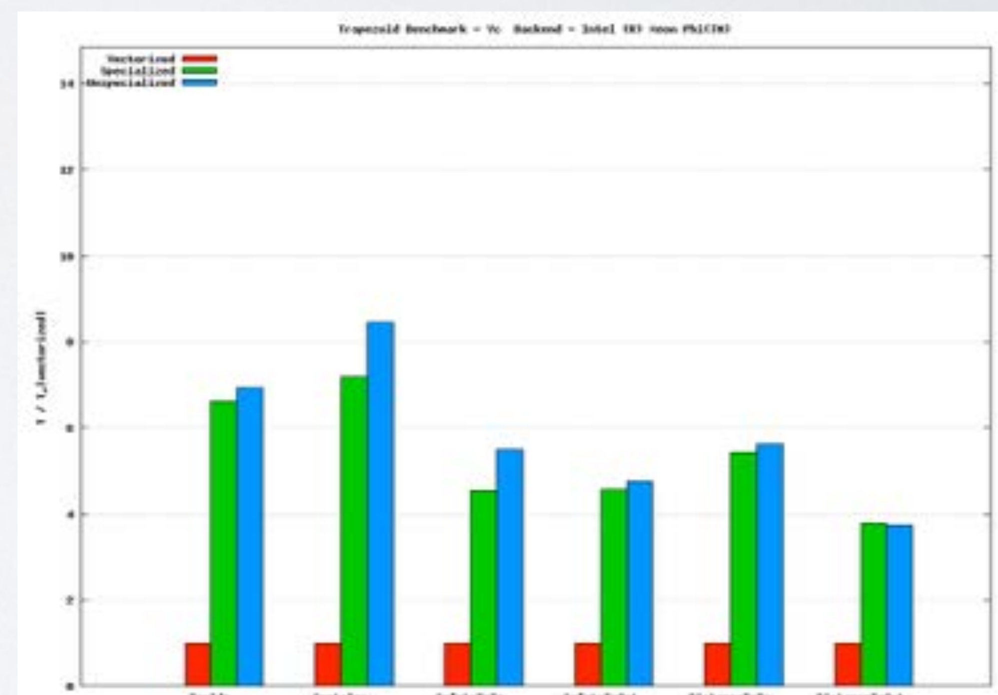
- Geometry is 30-40% of the total CPU time in Geant4
- A library of vectorized geometry algorithms to take maximum advantage of SIMD architectures
- Substantial performance gains also in scalar mode
- Testing the same on on GPU

	16 particles	1024 particles	SIMD max
Intel Ivy-Bridge (AVX)	~2.8x	~4x	4x
Intel Haswell (AVX2)	~3x	~5x	4x
Intel Xeon Phi (AVX-512)	~4.1	~4.8	8x





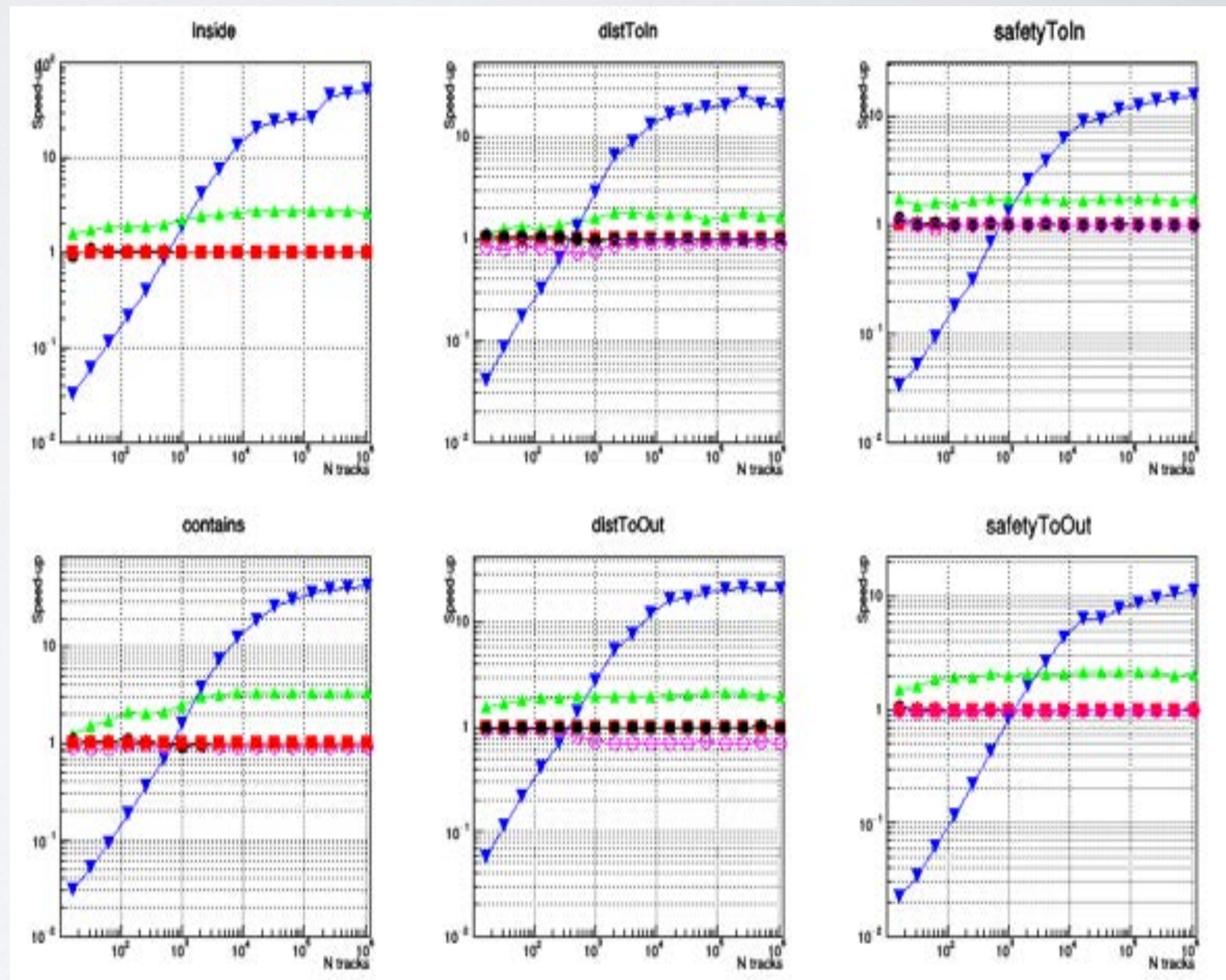
Overall performance for a simplified detector vs. scalar ROOT/5.34.17



Vectorization performance for trapezoid shape navigation (Xeon®Phi® C0PRQ-7120 P)

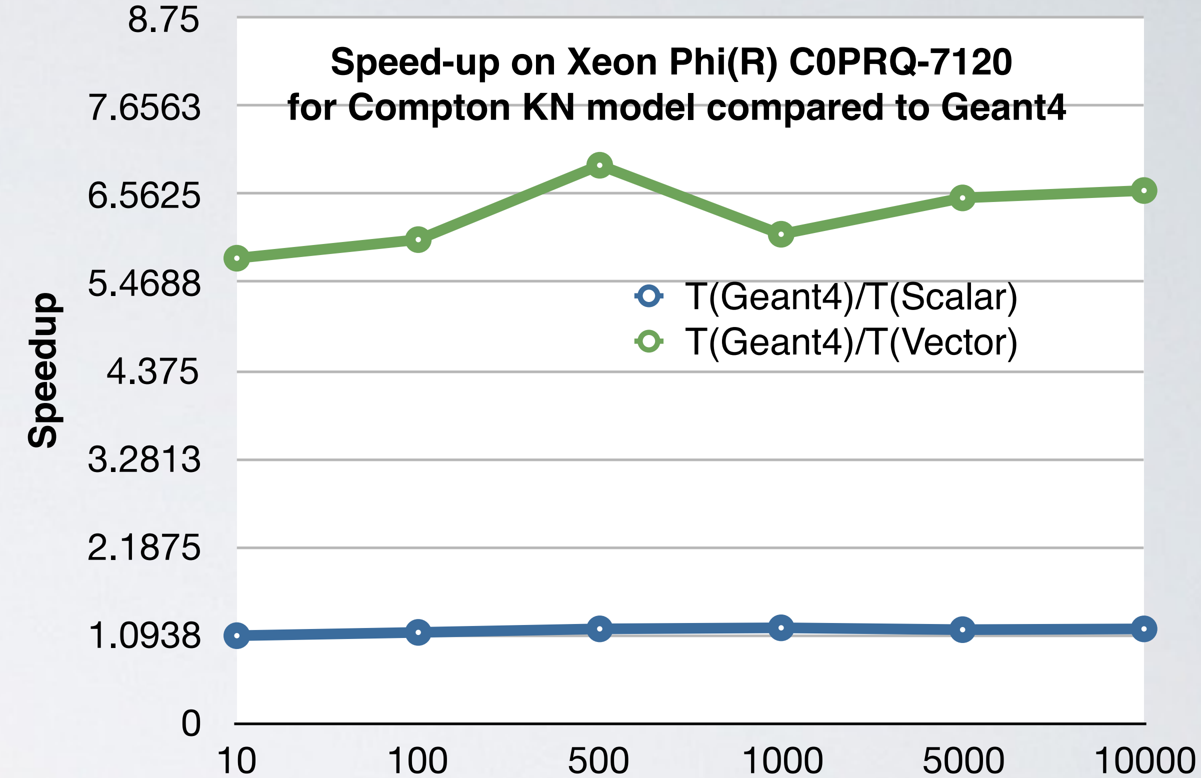
GEOMETRY PERFORMANCE ON K20

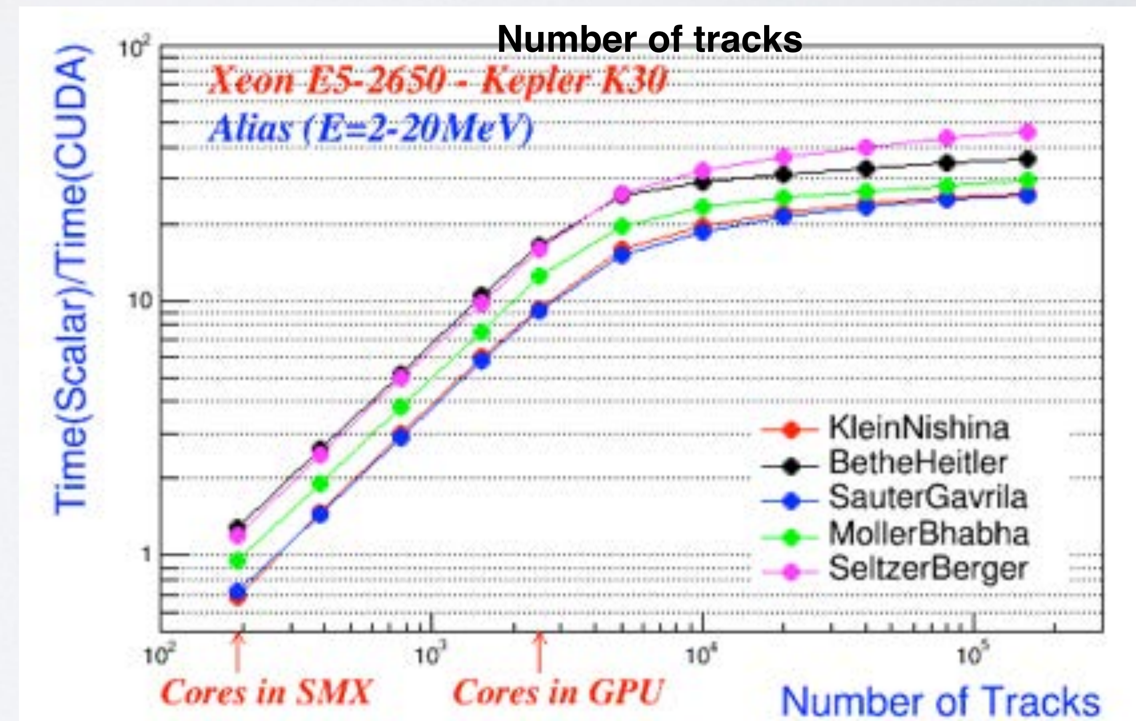
- Speedup for different navigation methods of the box shape, normalized to scalar CPU
  - Scalar (specialized/unspecialized)
  - Vector
  - GPU (Kepler K20)
  - ROOT
- Data transfer in/out is asynchronous
  - Measured only the kernel performance, but providing constant throughput can hide transfer latency
- The die can be saturated with both large track containers, running a single kernel, or with smaller containers dynamically scheduled.
- Just a baseline proving we can run the same code on CPU/accelerators, to be optimized



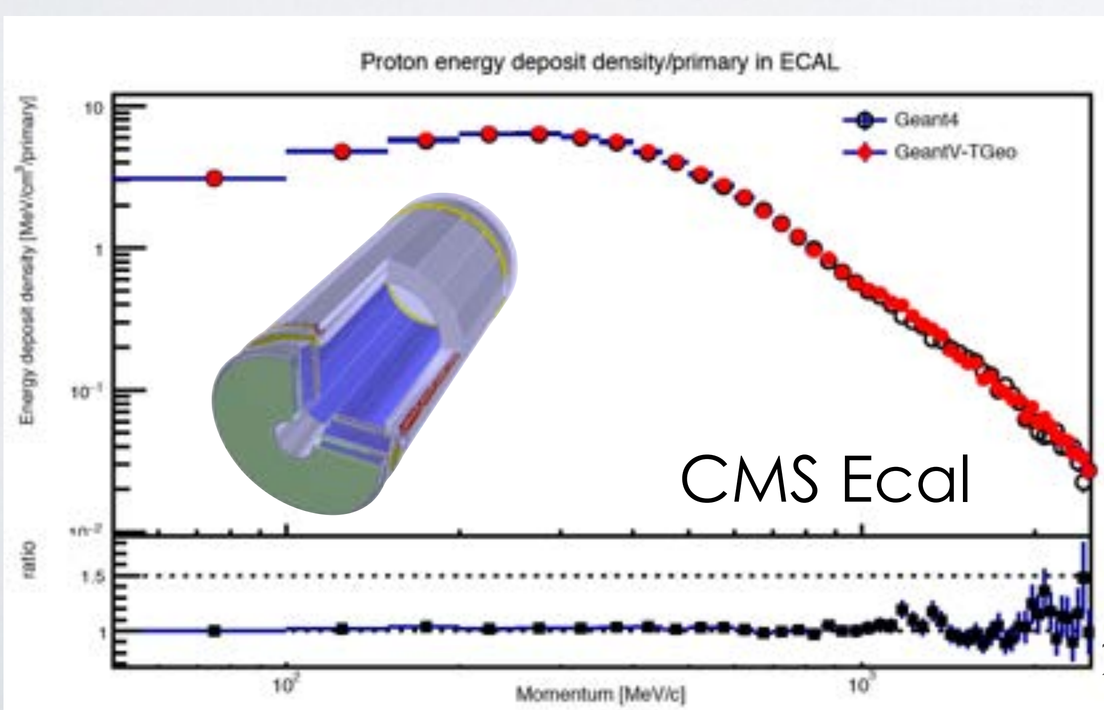
PHYSICS PERFORMANCE

- Objective: a vector/accelerator friendly re-write of physics code
- The vectorised Compton scattering shows good performance gains
- Current prototype able to run an exercise at the scale of an LHC experiment (CMS)
    - Simplified (tabulated) physics but full geometry, RK propagator in field
    - **Preliminary results hint to performance improvements of 3-4**



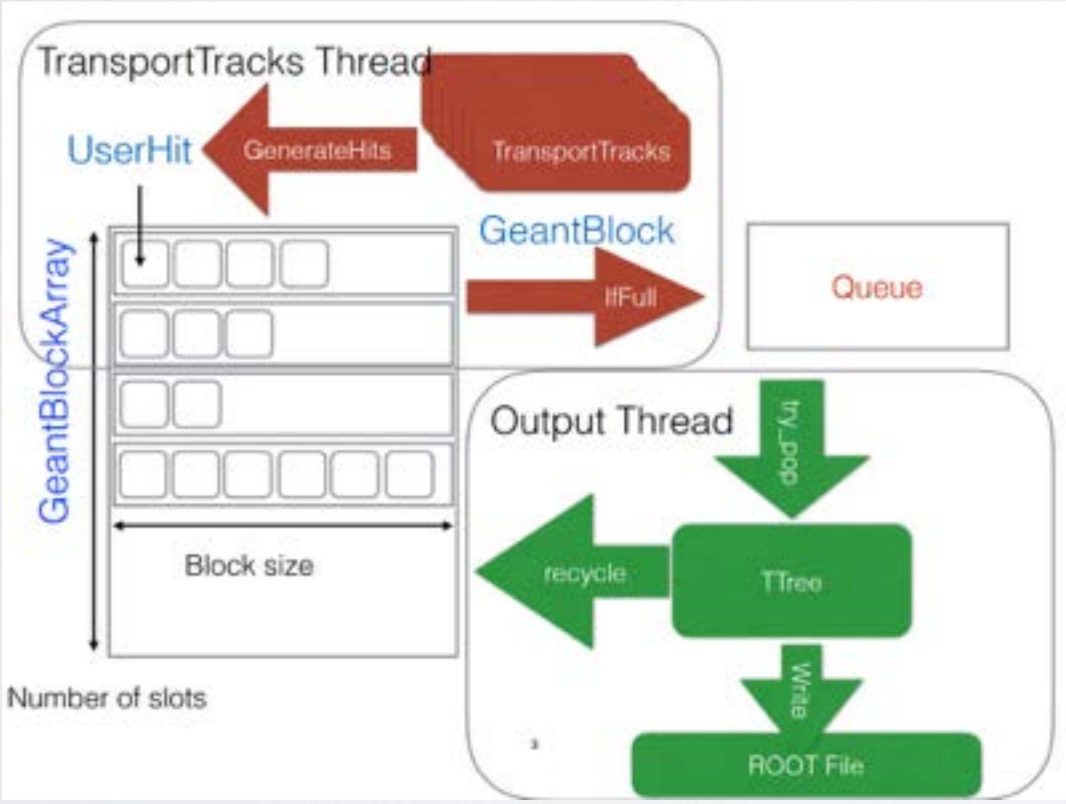


Alias sampling performance on a Kepler K30 (Soon' Y. - EM physics models on parallel computing architectures)



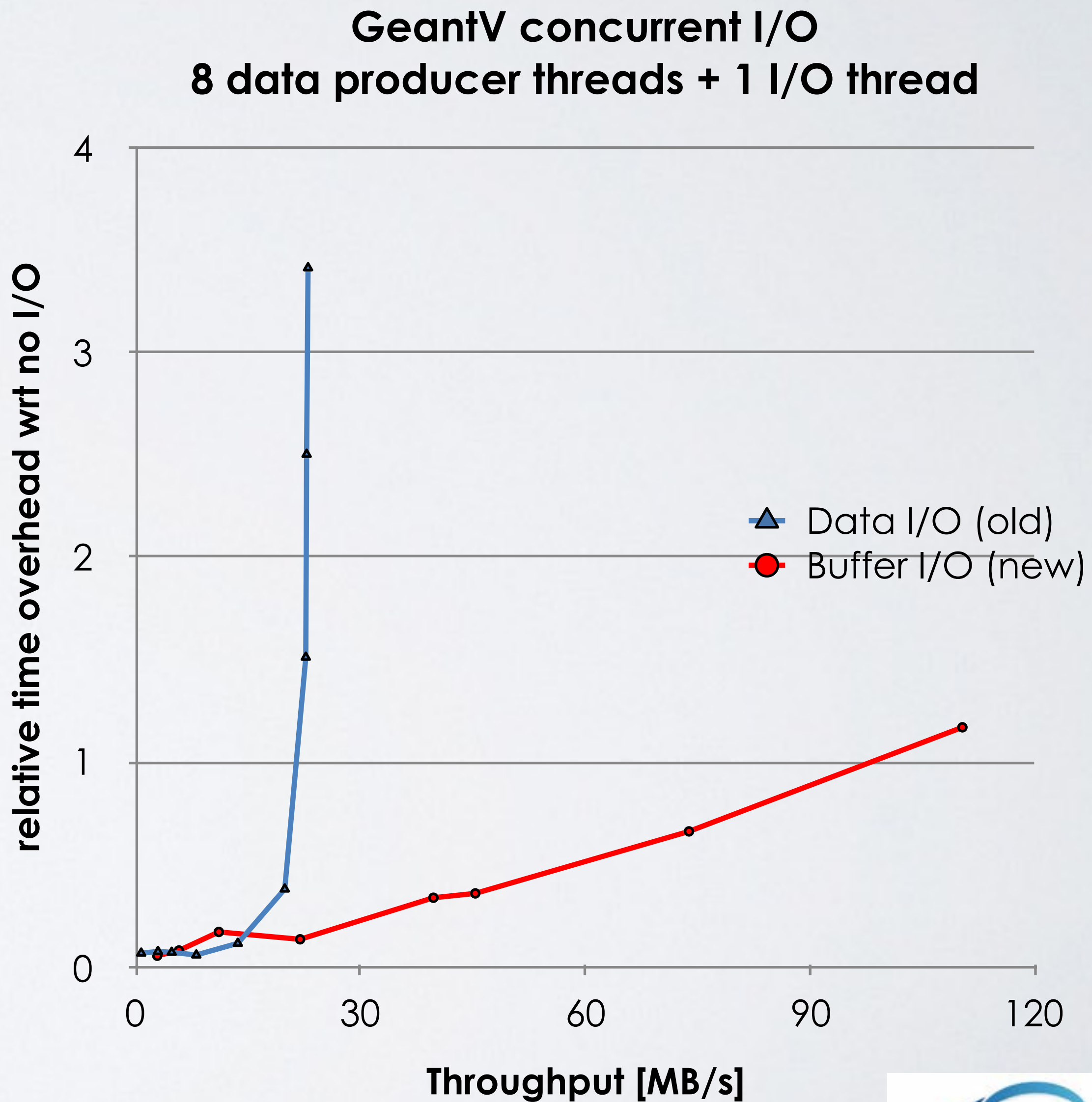
HITS/DIGITS I/O

- "Data" mode
  - Send concurrently data to one thread dealing with full I/O



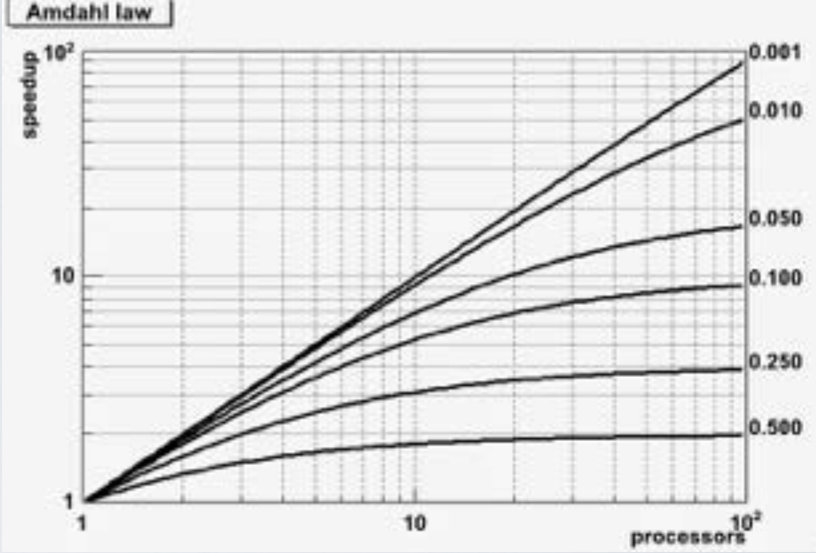
- "Buffer" mode
  - Send concurrently local trees connected to memory files produced by workers to one thread dealing with merging/write to disk

- Integrating user code with a highly concurrent framework should not spoil performance

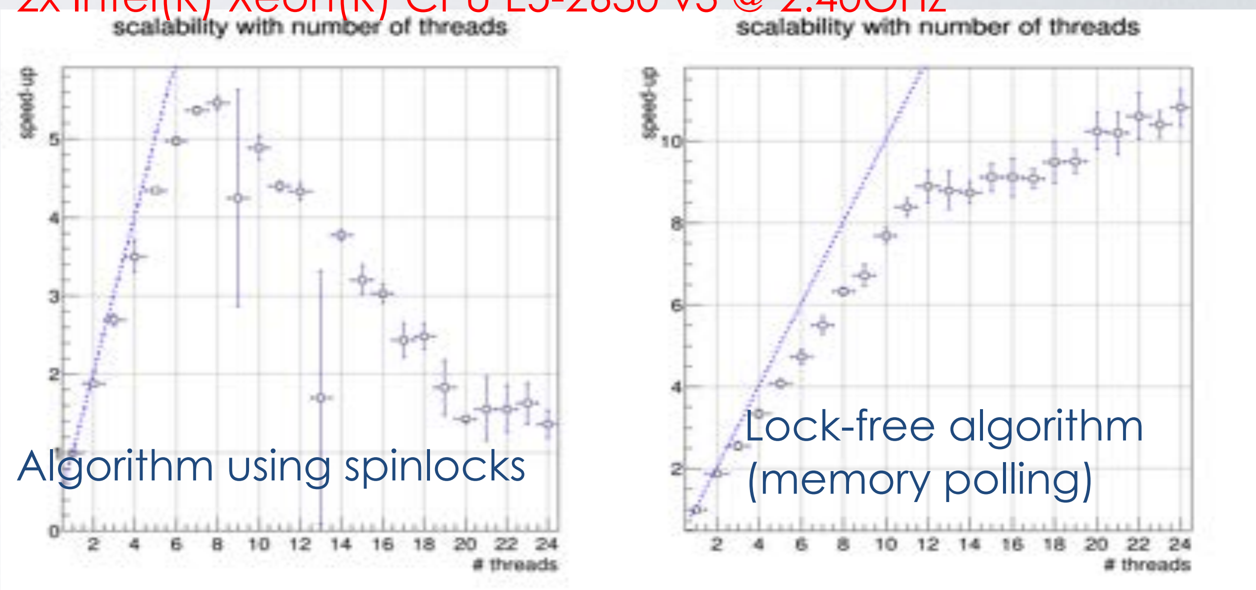


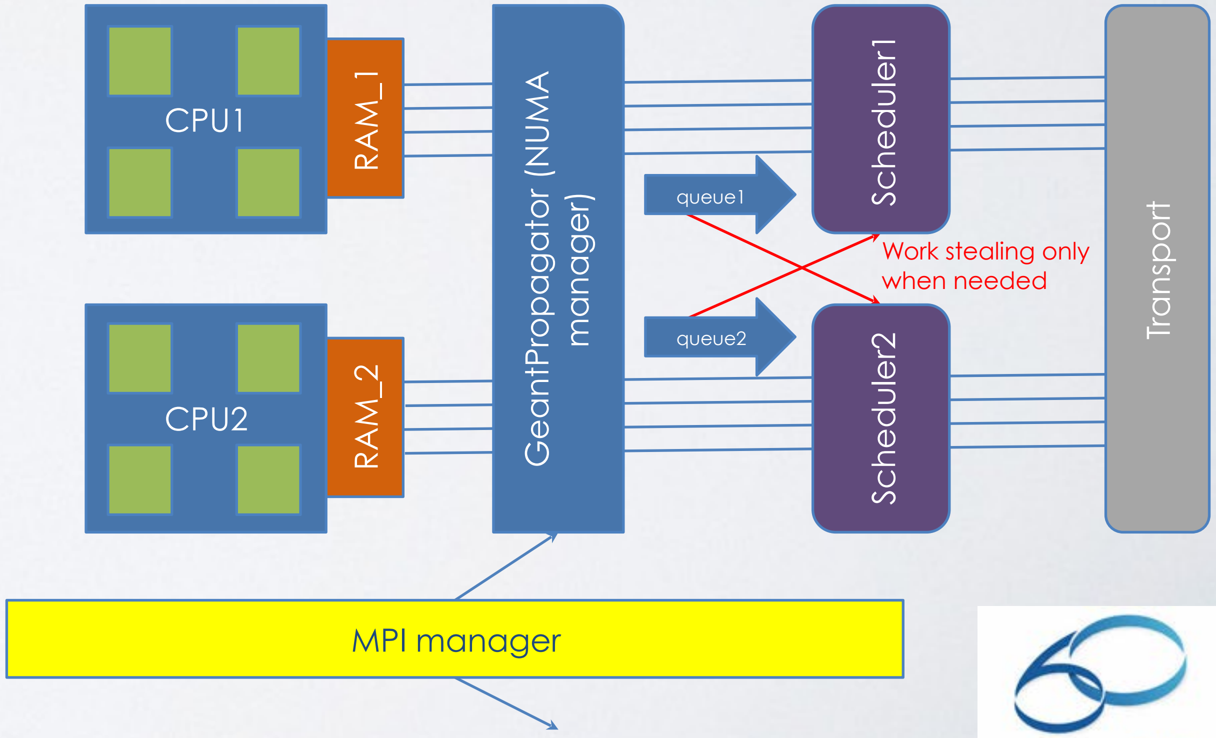
BASKETIZER PERFORMANCE

- Investigated different ways of scheduling & sharing work - lock free queues, ..
  - Changes in scheduler require non-trivial effort (rewrite)
- Amdahl still large, due to high re-basketizing load (concurrent copying)
  - O(10^5) baskets/second on Intel Core i7™
  - Algorithm already lock free
  - Rate will go down with physics processes
- Ongoing work to improve scalability
  - Re-use baskets in the same thread after step if enough particles doing physics-limited steps
  - Clone scheduling in NUMA aware groups, important for many cores (e.g. KNL)

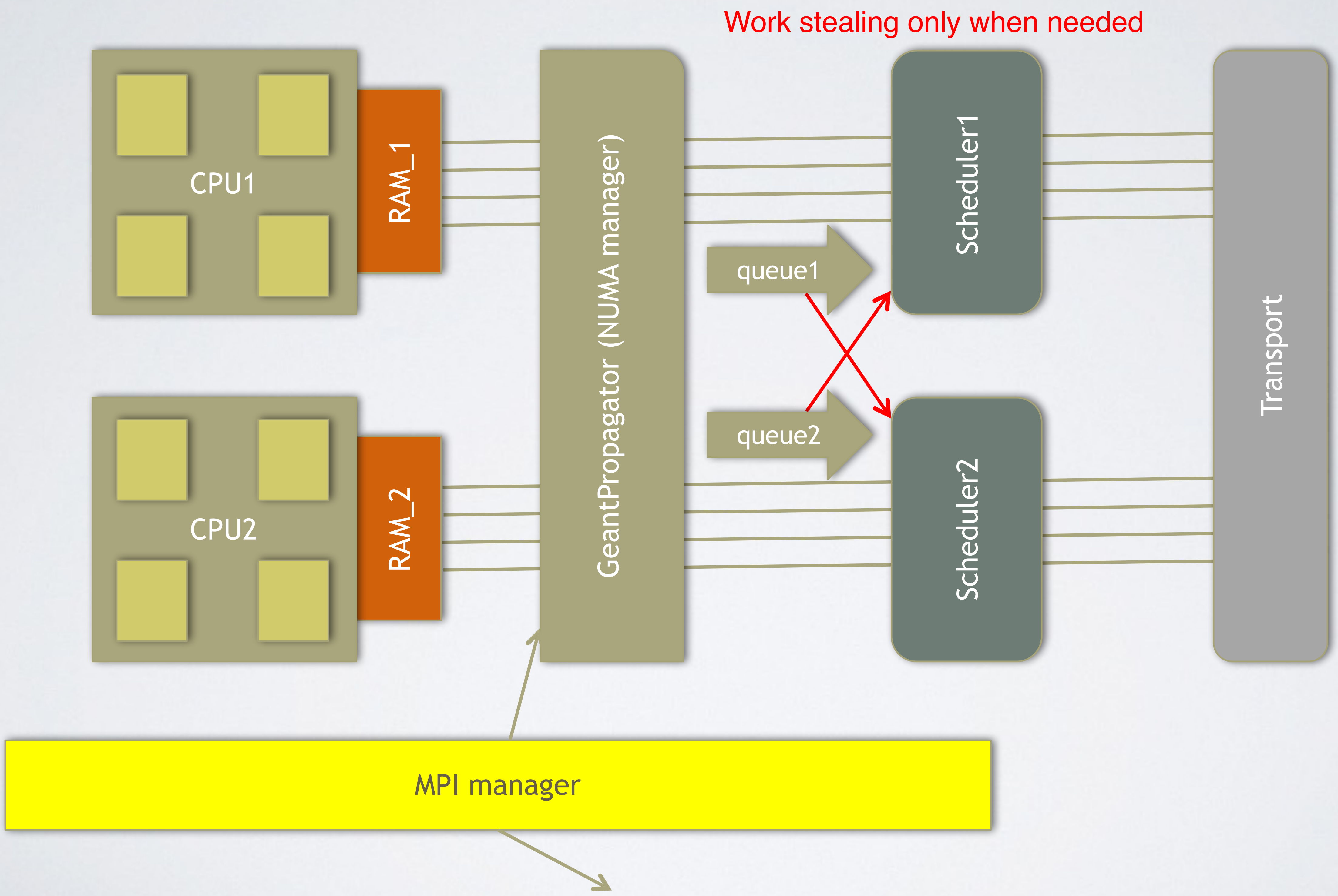


Rebasketizing
2x Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz



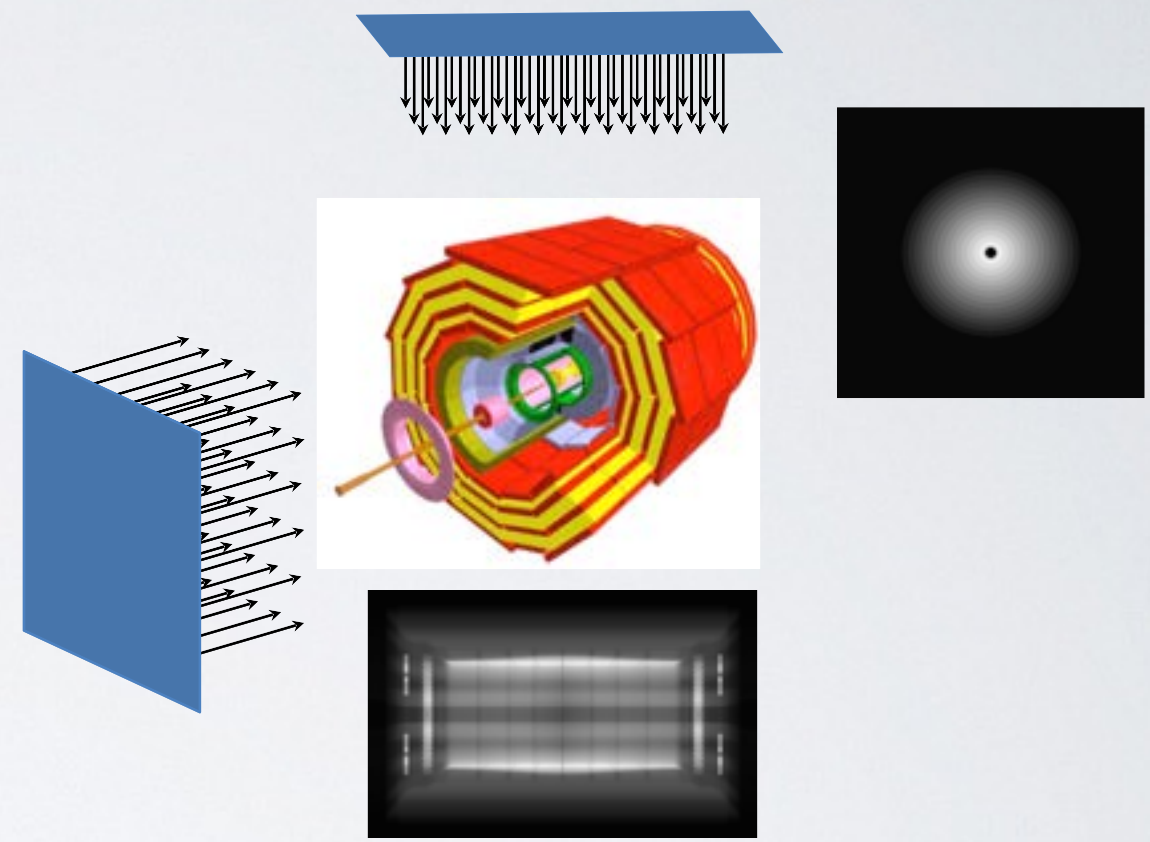


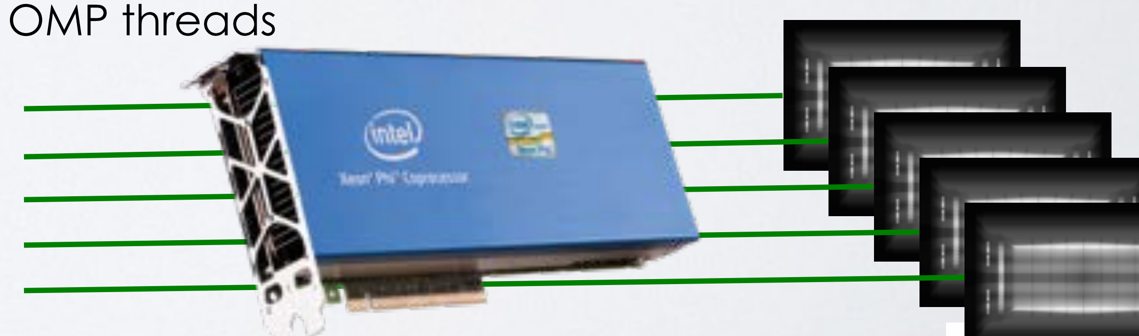
"CLONING" THE SCHEDULER



THE X-RAY BENCHMARK

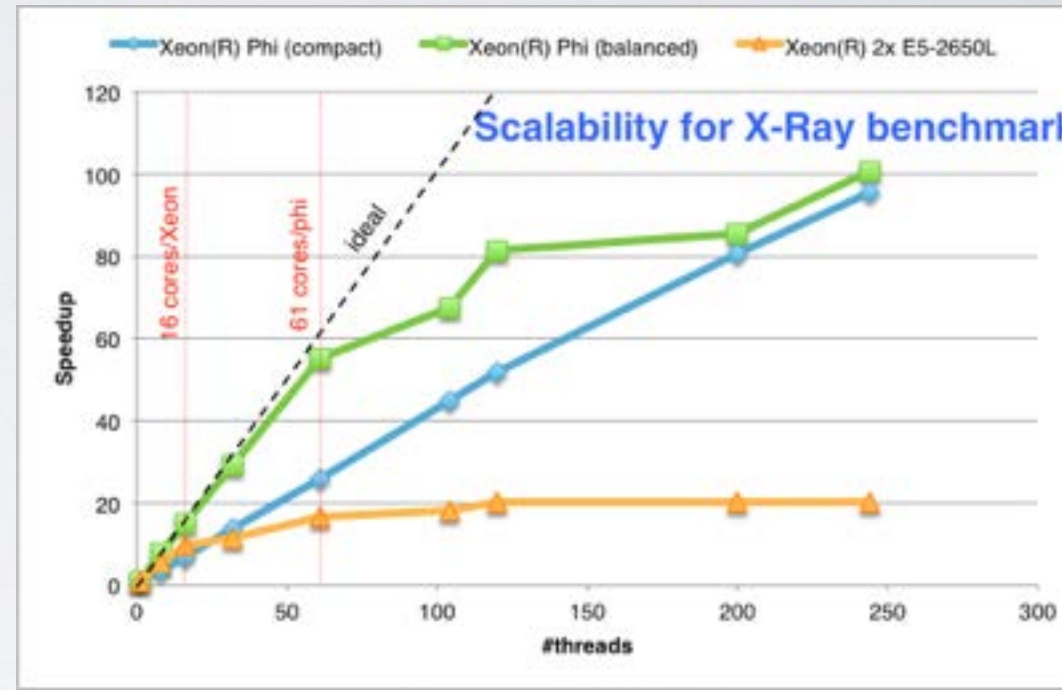
- The X-Ray benchmark tests geometry navigation in a real detector geometry
- *X-Ray* scans a module with virtual rays in a grid corresponding to pixels on the final image
  - Each ray is propagated from boundary to boundary
  - Pixel gray level determined by number of crossings
- A simple geometry example (concentric tubes) emulating a tracker detector used for Xeon©Phi benchmark
  - To probe the vectorized geometry elements + global navigation as task
  - OMP parallelism + "basket" model

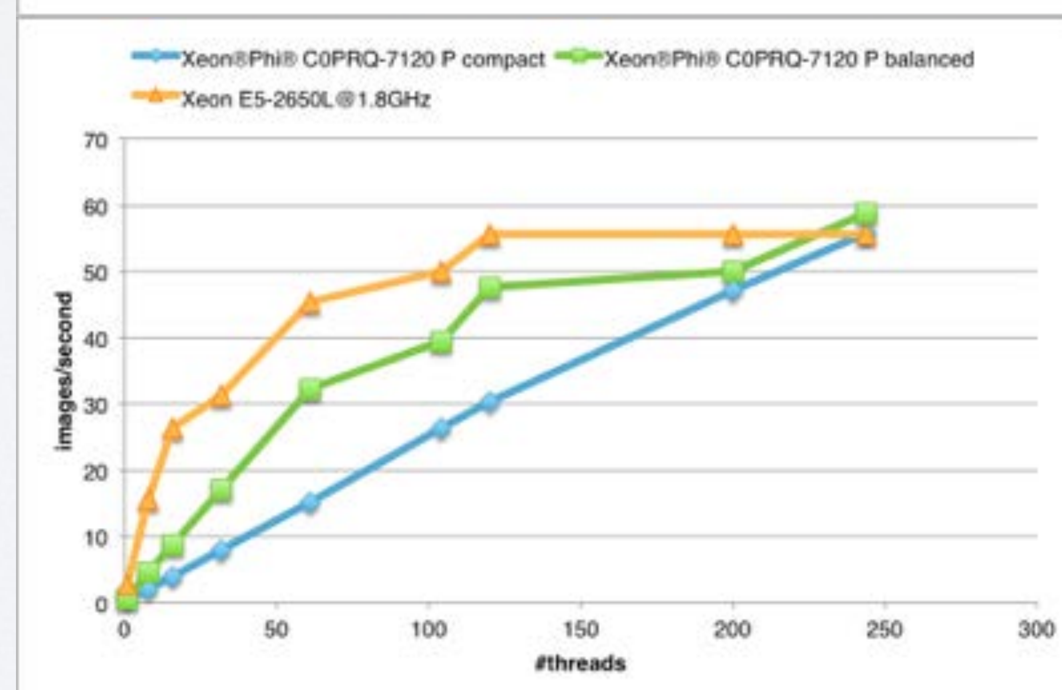




SCALABILITY AND THROUGHPUT

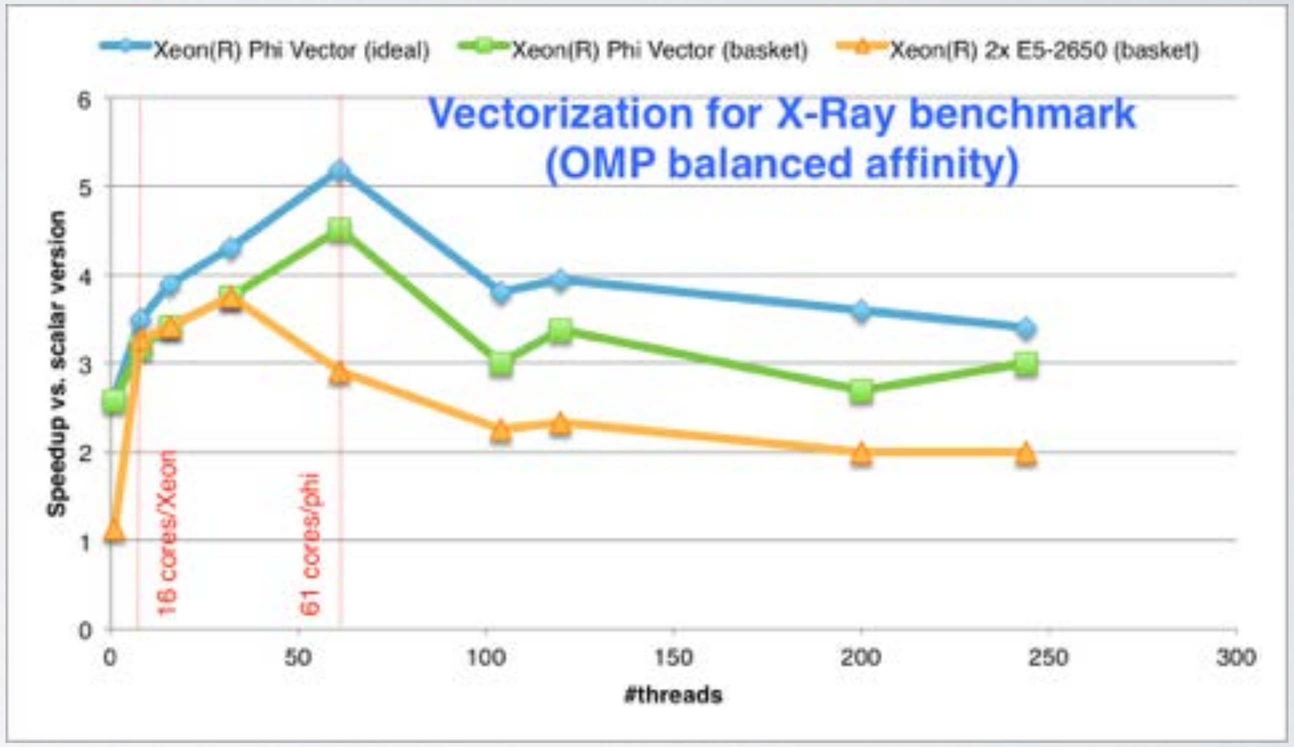
- Better behavior using OMP balanced
  - Approaching well the ideal curve up to native cores count
  - Balanced threading converges towards the compact model as all the thread slots are filled
- It's worth to run Xeon Phi saturated for our application
- The throughput performance for a saturated KNC is equivalent (for this setup) to the dual Xeon E5-2650L@1.8GHz server which hosts the card.





VECTOR PERFORMANCE

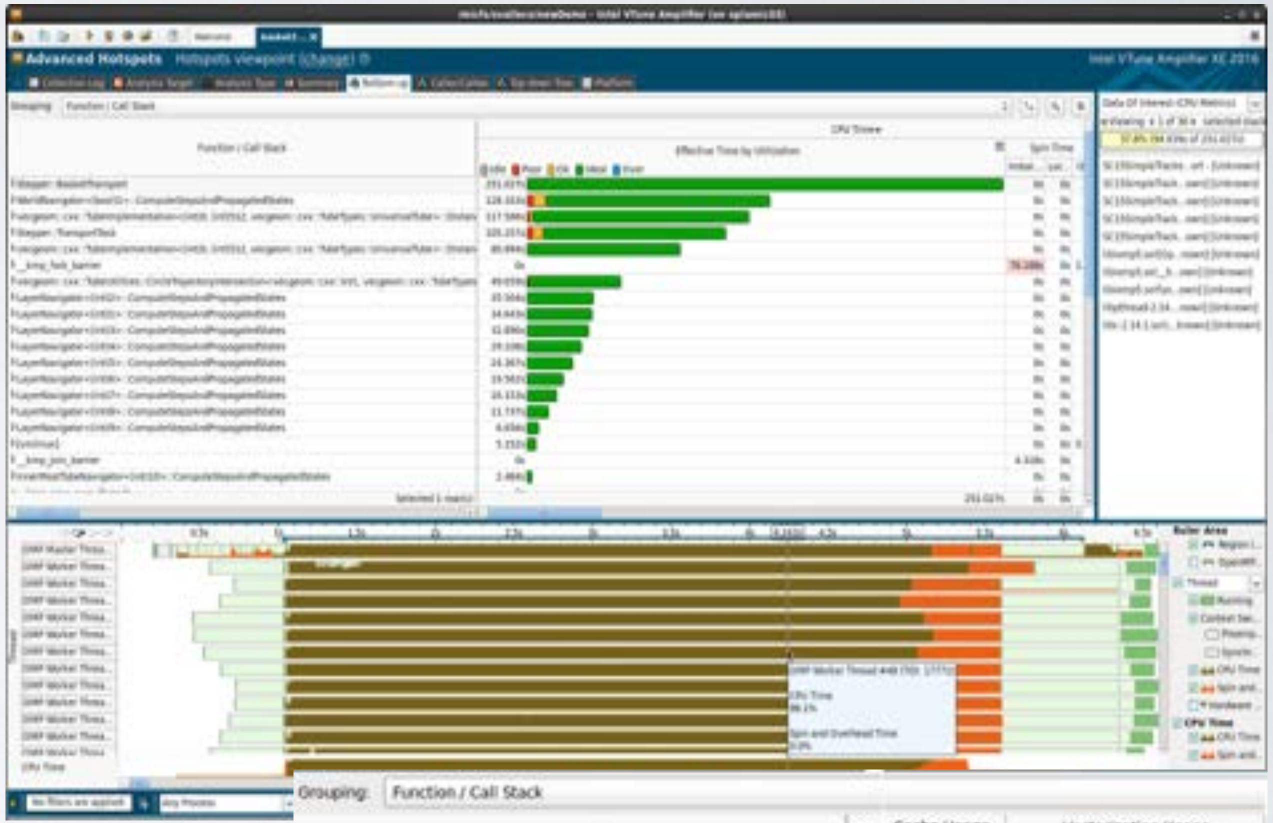
- Gaining up to 4.5 from vectorization in basketized mode
  - Approaching the ideal vectorization case (when no regrouping of vectors is done) .
- Vector starvation starts when filling more thread slots than the core count
  - Performance loss is not dramatic
  - Better vectorization compared to the Sandy-Bridge host (expected)

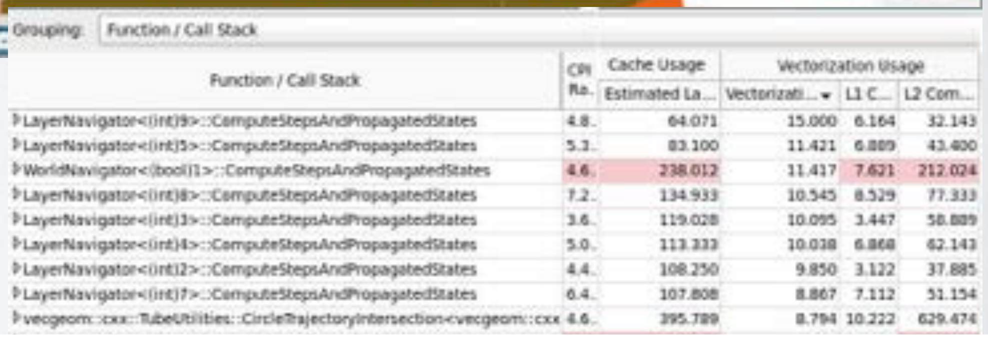


- Scalar case: Simple loop over pixels
- Ideal vectorization case: Fill vectors with N times the same X-ray
- Realistic (basket) case: Group baskets per geometry volume

PROFILING FOR THE X-RAY BENCHMARK

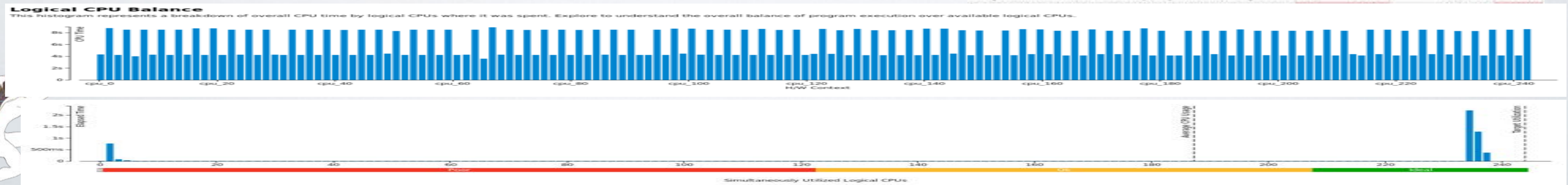
- Good vectorization intensity, thread activity and core usage for the X-Ray basketized benchmark on a Xeon Phi (61 core C0PRQ-7120 P)
- The performance tools gave us good insight on the current performance of GeantV





Grouping: Function / Call Stack

Function / Call Stack	CPI Ra..	Cache Usage	Vectorization Usage		
		Estimated La...	Vectorizati...	L1 C...	L2 Com...
LayerNavigator<(int)9>::ComputeStepsAndPropagatedStates	4.8..	64.071	15.000	6.164	32.143
LayerNavigator<(int)5>::ComputeStepsAndPropagatedStates	5.3..	83.100	11.421	6.889	43.400
WorldNavigator<(bool)1>::ComputeStepsAndPropagatedStates	4.6..	238.012	11.417	7.621	212.024
LayerNavigator<(int)8>::ComputeStepsAndPropagatedStates	7.2..	134.933	10.545	8.529	77.333
LayerNavigator<(int)3>::ComputeStepsAndPropagatedStates	3.6..	119.028	10.095	3.447	58.889
LayerNavigator<(int)4>::ComputeStepsAndPropagatedStates	5.0..	113.333	10.038	6.868	62.143
LayerNavigator<(int)2>::ComputeStepsAndPropagatedStates	4.4..	108.250	9.850	3.122	37.885
LayerNavigator<(int)7>::ComputeStepsAndPropagatedStates	6.4..	107.808	8.867	7.112	51.154
vecgeom::cxx::TubeUtilities::CircleTrajectoryIntersection<vecgeom::cxx	4.6..	395.789	8.794	10.222	629.474



LOOKING FORWARD TO…

- … implementing a “smoking gun” demonstrator combining all prototype features
  - SIMD gains in the full CMS experiment setup
  - Coprocessor broker in action: part of the full transport kernel running on Xeon®Phi® and GPGPU
  - Scalability and NUMA awareness for rebasketizing procedure
  - … achieving these just moves the target a bit further
- … testing and optimizing the workflow on KNL
  - Important architecture to test how flexible our model is
  - Expecting epic “fights” for scaling up the performance
- Complete the porting on GPUs and start performance optimization
- Already working on implementing device-specific scheduling policies and addressing NUMA awareness

SUMMARY

- We are designing the next generation simulation program architecture
    - SIMD exploitation and accelerators are the focus of this effort
- We have a prototype which looks promising toward our goal of a performance gain of a factor 3-5 over current software
- We plan to have a “testable” prototype in a couple of year from now (end 2017)
- We look forward to more communities joining our effort

PERSPECTIVES

- Doing HPC when you do not use blas-rich codes feels like being the “poor relation”
    - Benchmarks are of reduced relevance
    - Your CPI is poor / abysmal
    - There is little guidance on how to go ahead
- Moreover the “far-from-blas” community is sparse and communication is poor
    - HEP for one is seriously affected by NIH syndrome
- There is a nagging feeling of reinventing the (square) wheel
- Communication is here a major problem
    - Big data, ROOT…