The Fast Simulation Project

28th January 2013

Federico Carminati

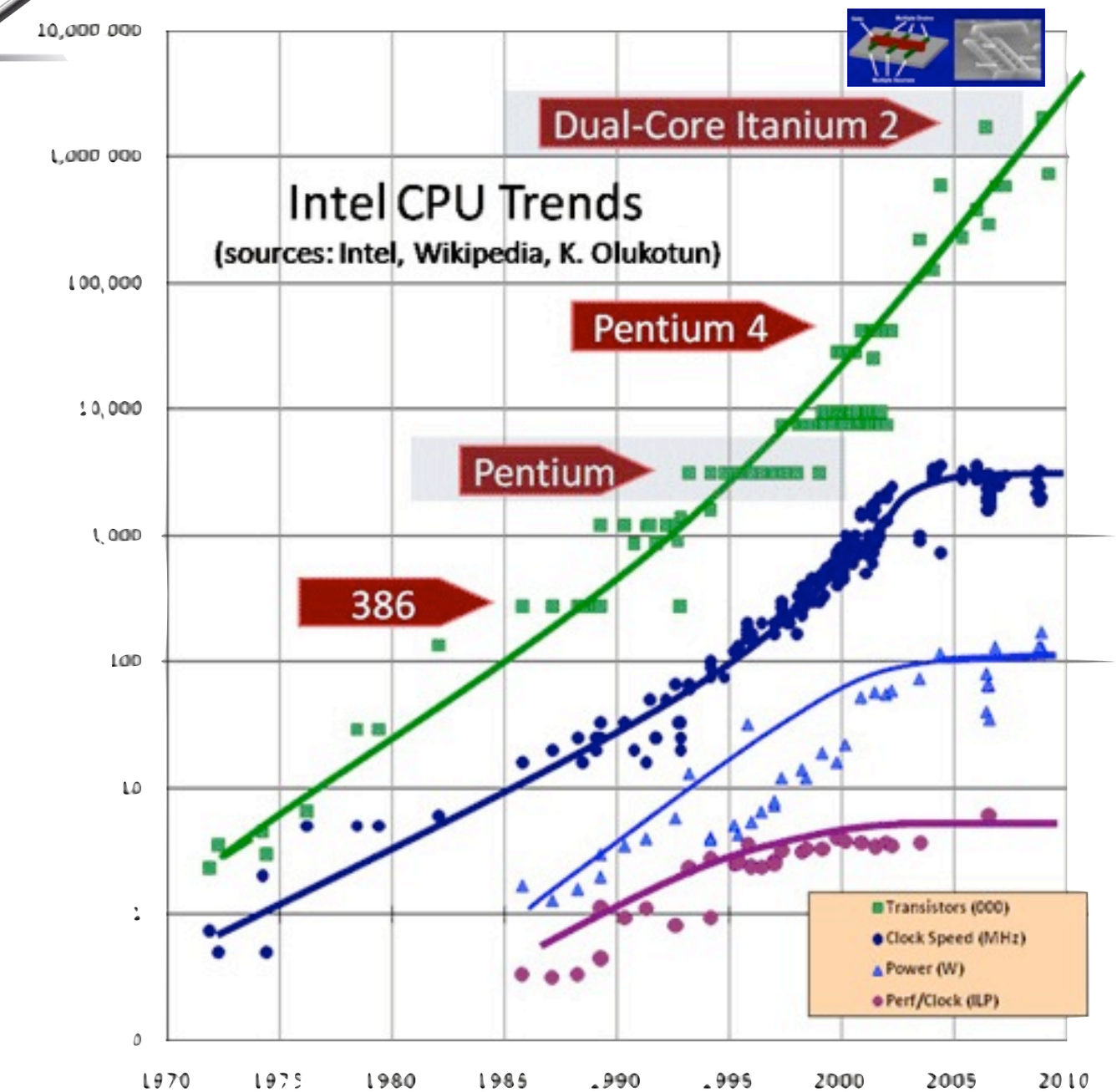
Motivations for the project

- Massive-parallelism is here – Moore's law has to be reinterpreted
  - You do not get more speed for free, you get more optimisation opportunities to exploit
- We "got away" many times, but now we probably can't
  - Difficult to ask Funding Agencies for (much) more computing and, at the same time, confess that we use only part of the "bare iron"
- "Embarrassing parallelism" and "throughput computing" reduced the push to shorten "time to solution"

Motivations for the project

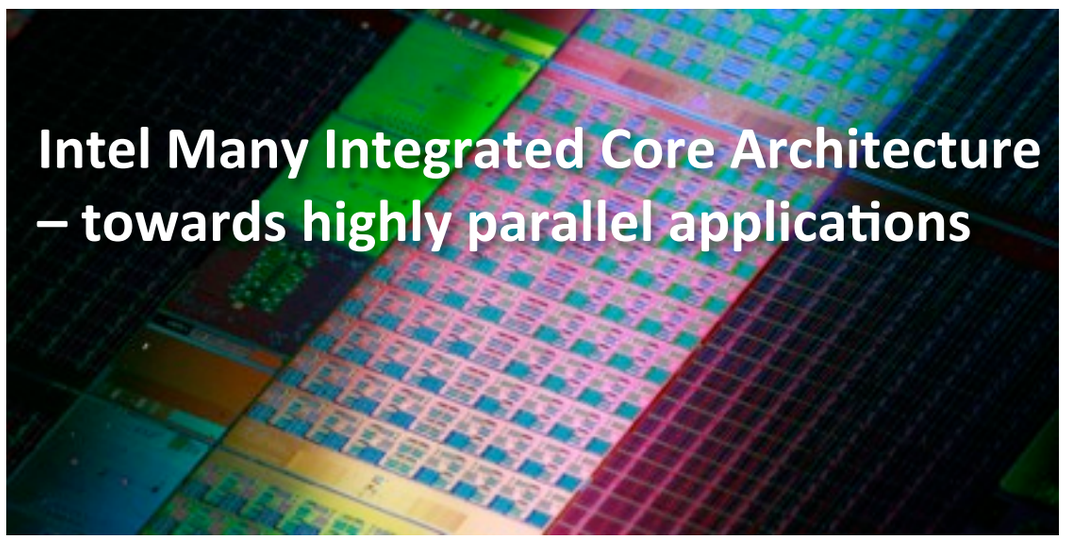
- We have “missed” several trains
  - Vectorisation (IBM VM, Cray X-MP)
  - Low parallelism (IBM VM, Cray X-MP)
  - Moderate parallelism (GPMIMD machine)
  - High parallelism (IBM SP2)
  - Heterogeneous parallelism
- Trivial (job-level) parallelism and evolution of clock cycle was enough
- But now the bangs-per-bucks for us is a monotonic decreasing function and this affects also throughput
- I am afraid this time we have to bite the bullet
  - I think we are all convinced of this

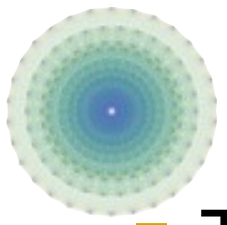
Trends…



While transistors increase was following Moore's law, frequency and power consumption was not…

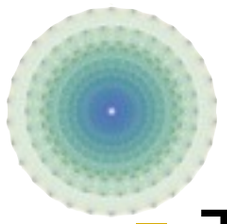
Tesla k10 GPU (NVIDIA) – state of the art in GPU technology





The eight dimensions

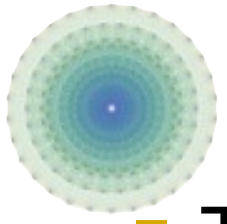
- The “dimensions of performance”
  - Vectors
  - Instruction Pipelining
  - Instruction Level Parallelism (ILP)
  - Hardware threading
  - Clock frequency
  - Multi-core
  - Multi-socket
  - Multi-node



The eight dimensions

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

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

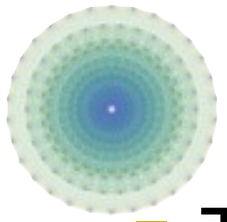


The eight dimensions

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

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

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



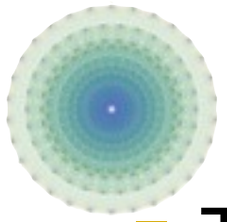
The eight dimensions

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

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



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

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

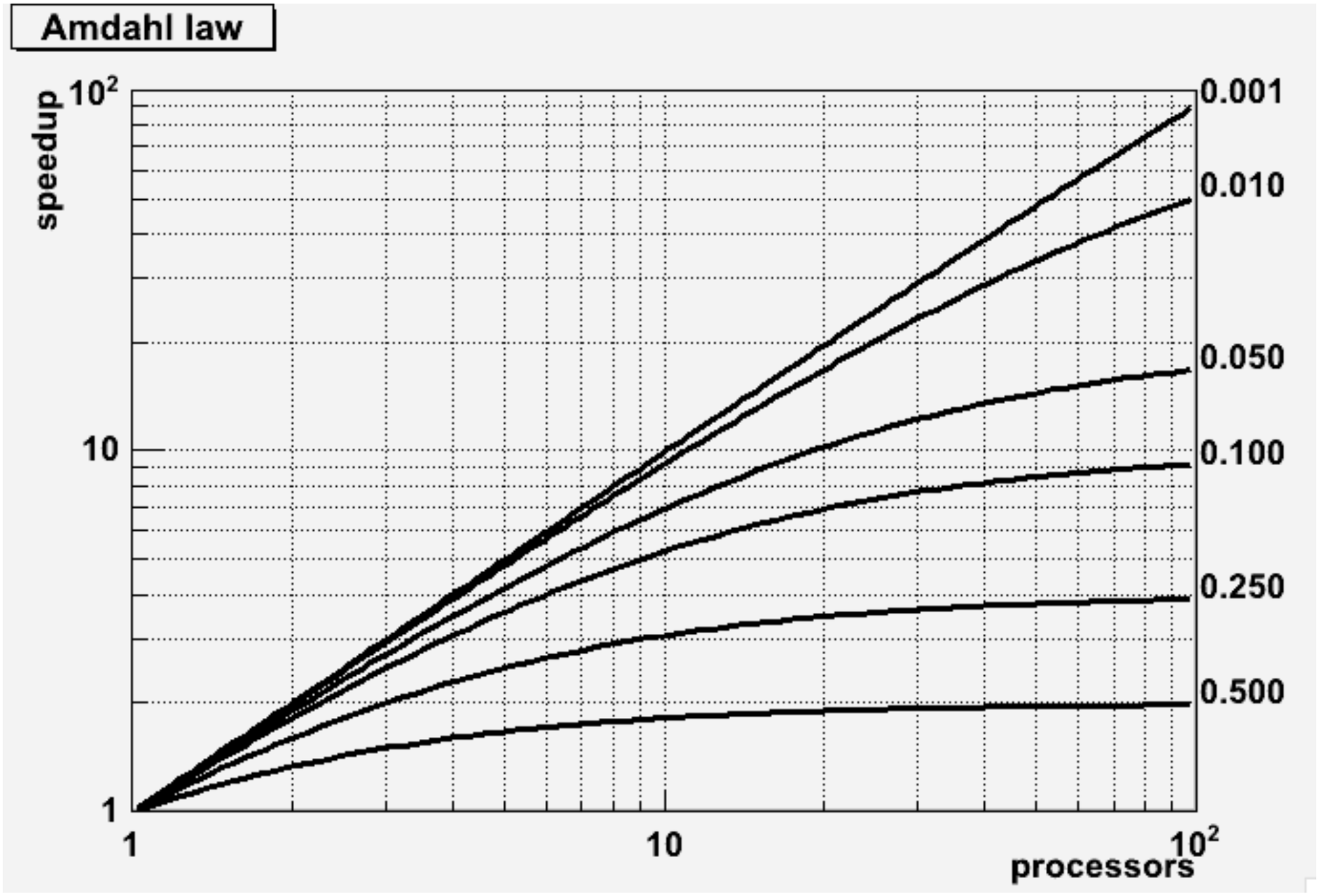
Gain in memory footprint and time-to-solution but not in throughput

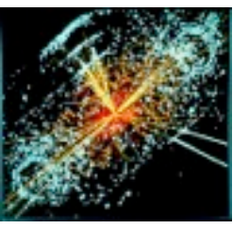
Possibly running different jobs as we do now is the best solution

Expected limits on performance scaling			
	SIMD	ILP	HW THREADS
MAX	8	4	1.35
INDUSTRY	6	1.57	1.25
HEP	1	0.8	1.25
Expected limits on performance scaling (multiplied)			
	SIMD	ILP	HW THREADS
MAX	8	32	43.2
INDUSTRY	6	9.43	11.79
HEP	1	0.8	1

OpenLab@CHEP12

Just a reminder...





Why simulation?

- The most CPU-bound application we have with large room for speed-up
- It is largely experiment independent
  - Unlike reconstruction or analysis
- It is one of the most time-consuming activities in HEP computing
- Precision depends on (sqrt of) the number of events
- Improvements (in geometry for instance) and techniques are expected to feed back into reconstruction

The Grand Strategy

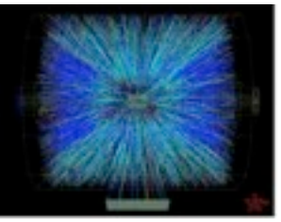
- Explore what can be the gain for simulation with no constraint coming from legacy code
  - There must be at least one project exploring this dimension
- Expose the parallelism intrinsic to the problem at all levels, from coarse granularity to micro-parallelism at the algorithm level
- Integrate from the beginning slow and fast simulation in order to optimise both in the same framework
- Explore if-and-how existing physics code (GEANT4) can be optimised in this framework

The synergies within SFT

- The spectrum of activities within SFT provides an almost ideal environment for this activity
  - GAUDI parallelisation is exploring the possibility of optimising existing reconstruction code
  - G4-MT is exploring how to reduce memory footprint of present simulation code exploiting “embarrassing parallelism”
  - The Fast simulation prototype is exploring how to “do the best we can” with an ab-initio rewrite
  - ROOT is working to provide a thread-safe library for data manipulation and I/O and algorithm optimisation at the micro-parallelism level
  - The concurrency forum offers an established venue for dialogue with the experiments@CERN and with HEP in general
- We have strengths on which we can build

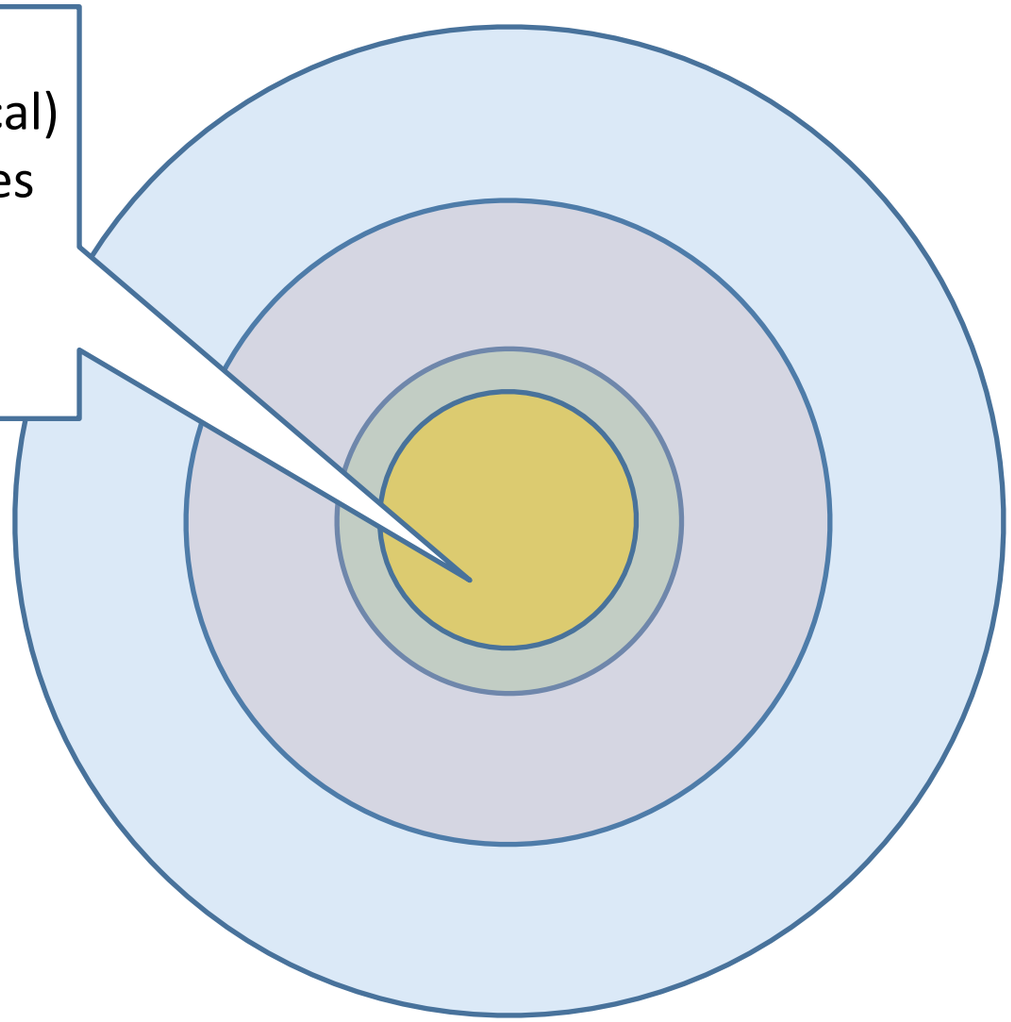
Where are we now?

- Two years of discussions between René, Andrei and fca and JohnA
- One year of (part time) development by Andrei focussed on transport in geometry
- Main conclusions
  - To increase throughput via the exploitation of micro-parallelism we have to expose parallelism at all levels
  - A complete re-thinking of particle transport is necessary for this
  - The ideal limit for vectorisation speedup would be hard to achieve due to the subsetting of particle lists (we have seen this with GEANT3 already)
  - Micro-parallelism is however within reach
- Today's certainties are tomorrow's mistakes...

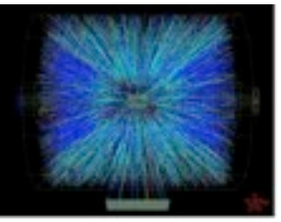


Classical particle transport (per particle)

- Geometry navigation (local)
- Material – X-section tables
- Particle type - physics processes

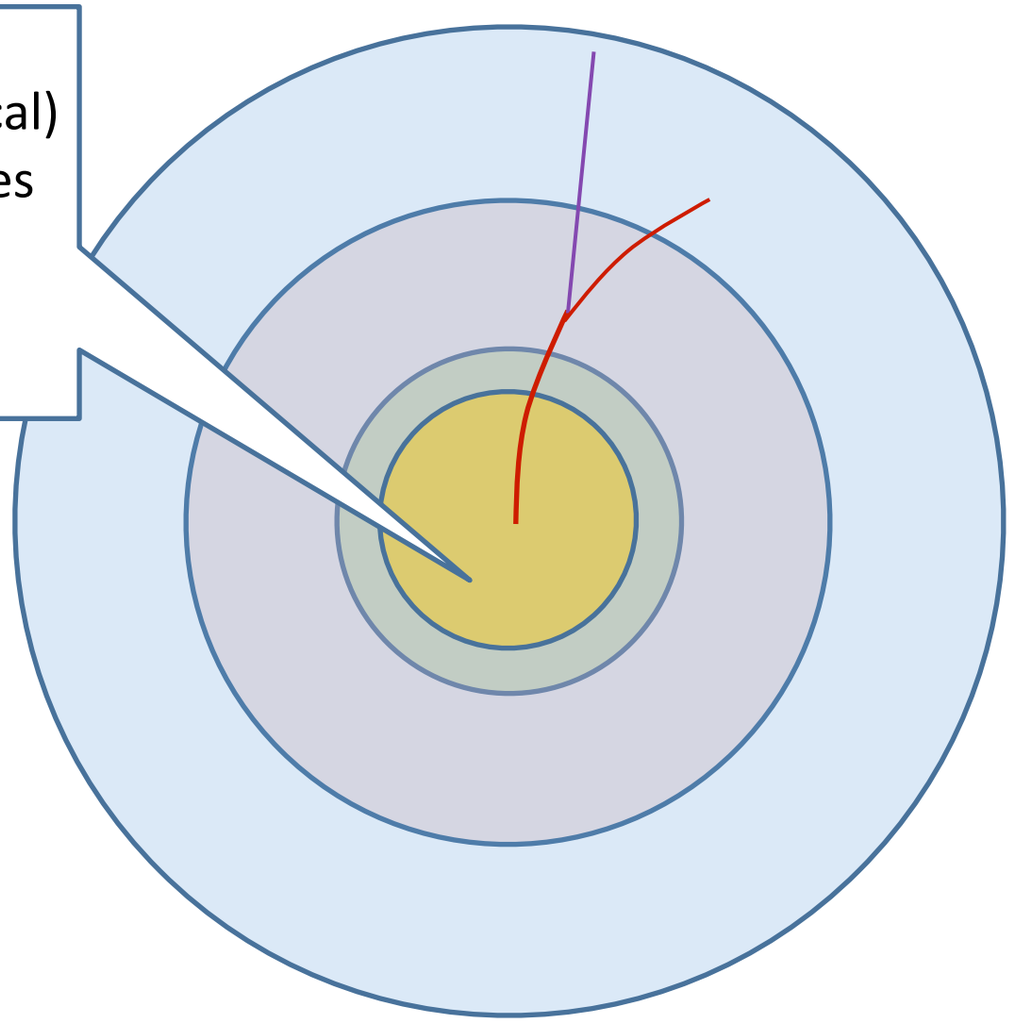


- Navigating very large data structures
- No locality
- OO abused: very deep instruction stack
- Cache misses
- Event-level parallelism will better use resources but won't improve these

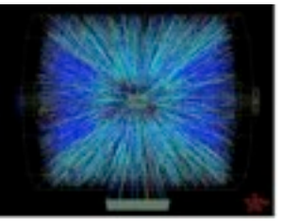


Classical particle transport (per particle)

- Geometry navigation (local)
- Material – X-section tables
- Particle type - physics processes

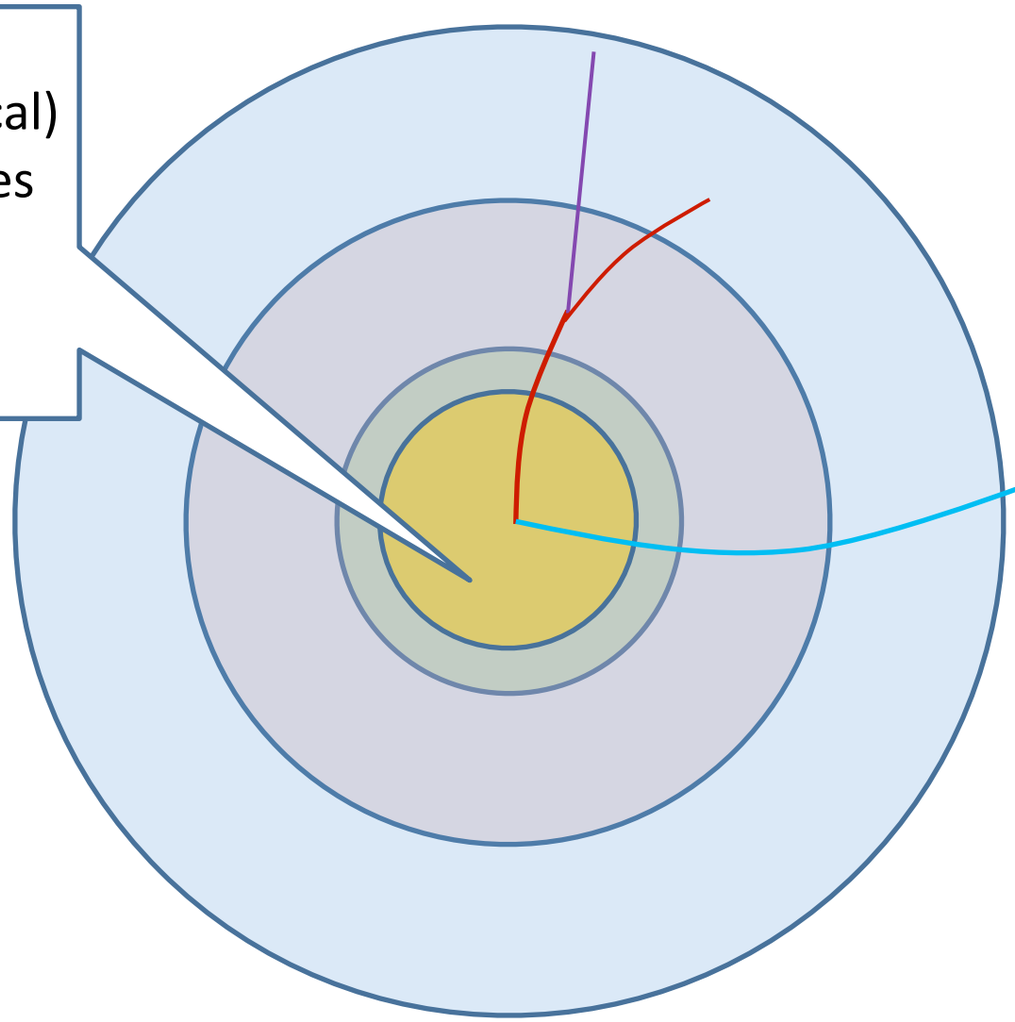


- Navigating very large data structures
- No locality
- OO abused: very deep instruction stack
- Cache misses
- Event-level parallelism will better use resources but won't improve these



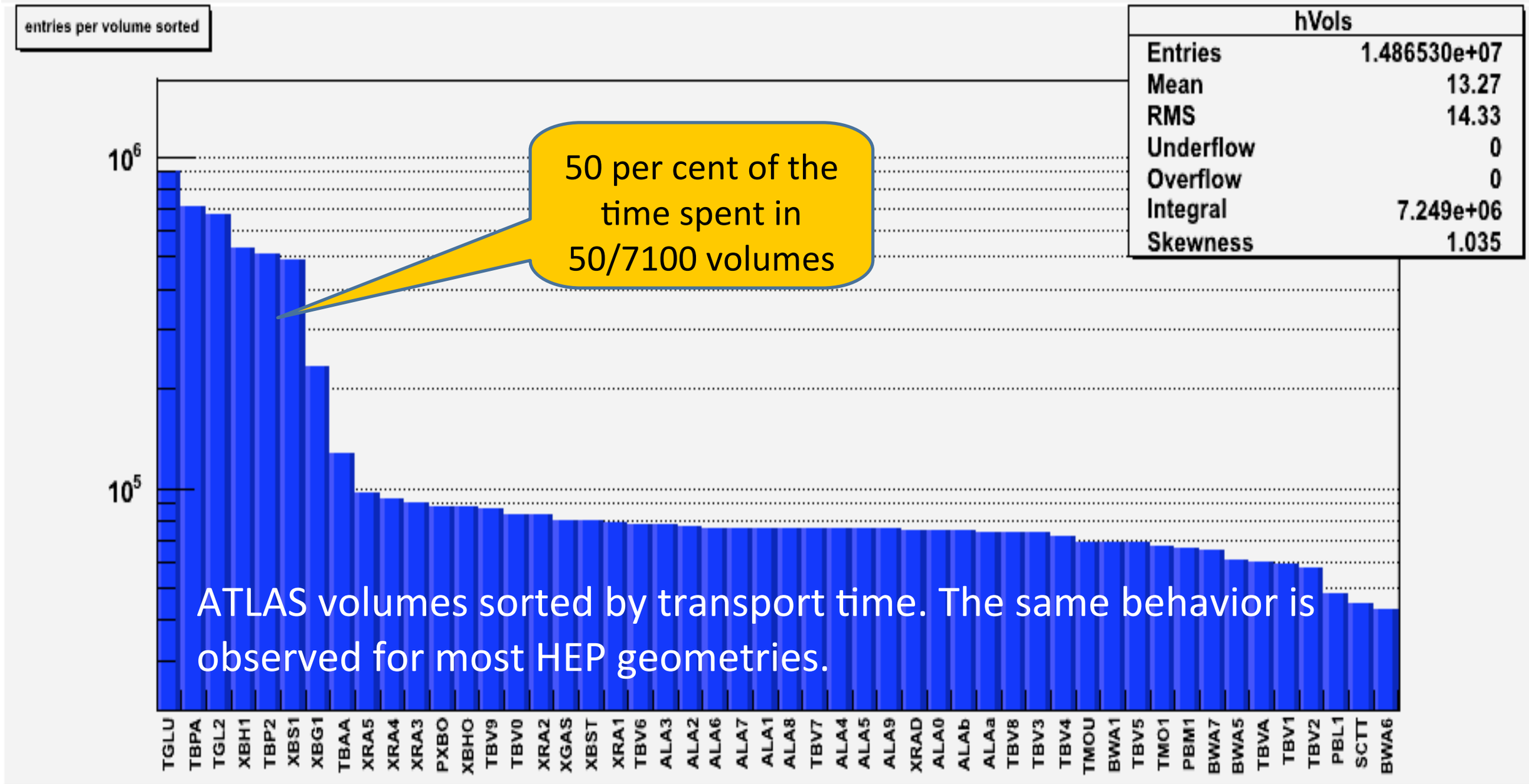
Classical particle transport (per particle)

- Geometry navigation (local)
- Material – X-section tables
- Particle type - physics processes

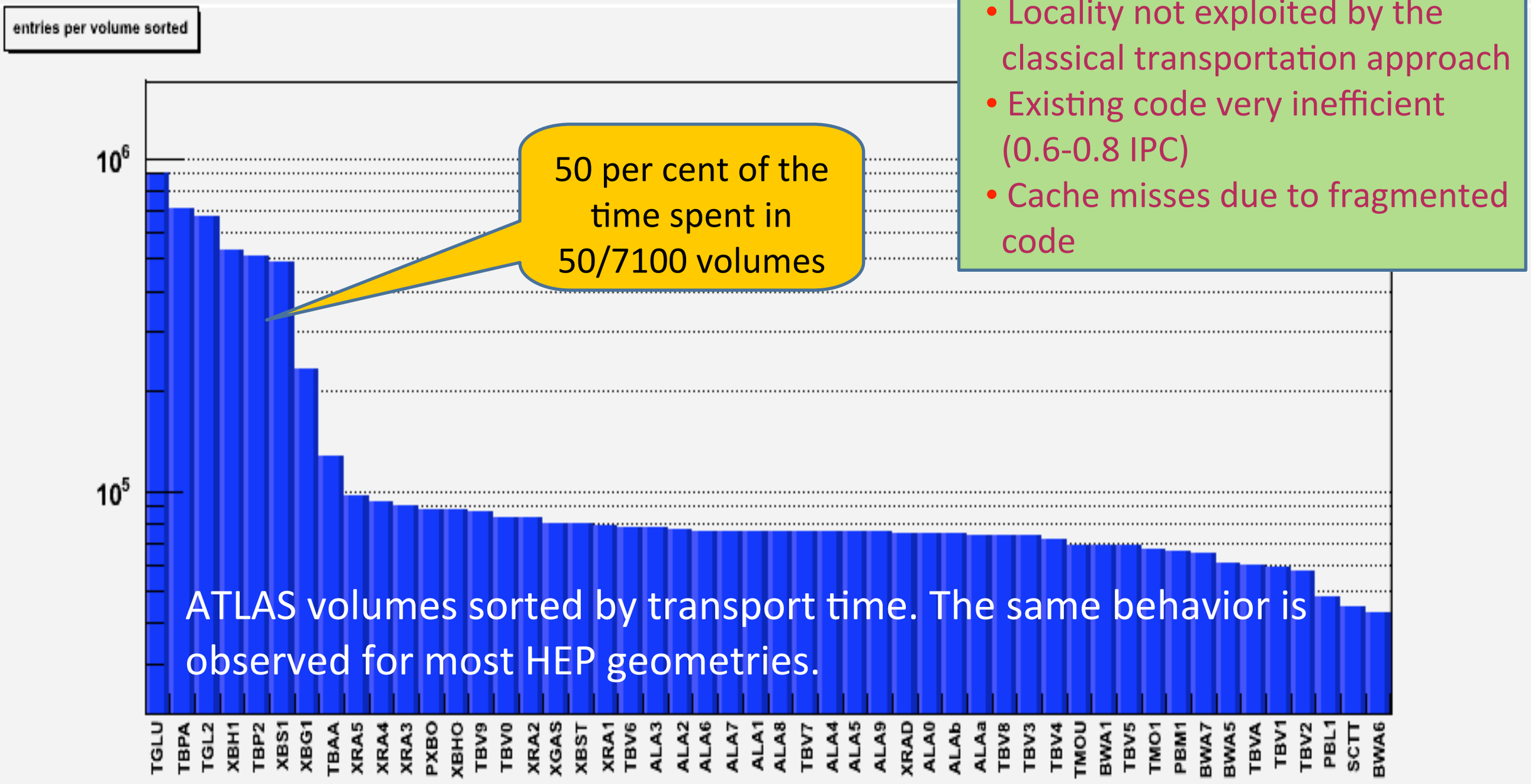


- Navigating very large data structures
- No locality
- OO abused: very deep instruction stack
- Cache misses
- Event-level parallelism will better use resources but won't improve these

Simple observation: HEP transport is mostly local !



Simple observation: HEP transport is mostly local !



- Locality not exploited by the classical transportation approach
- Existing code very inefficient (0.6-0.8 IPC)
- Cache misses due to fragmented code

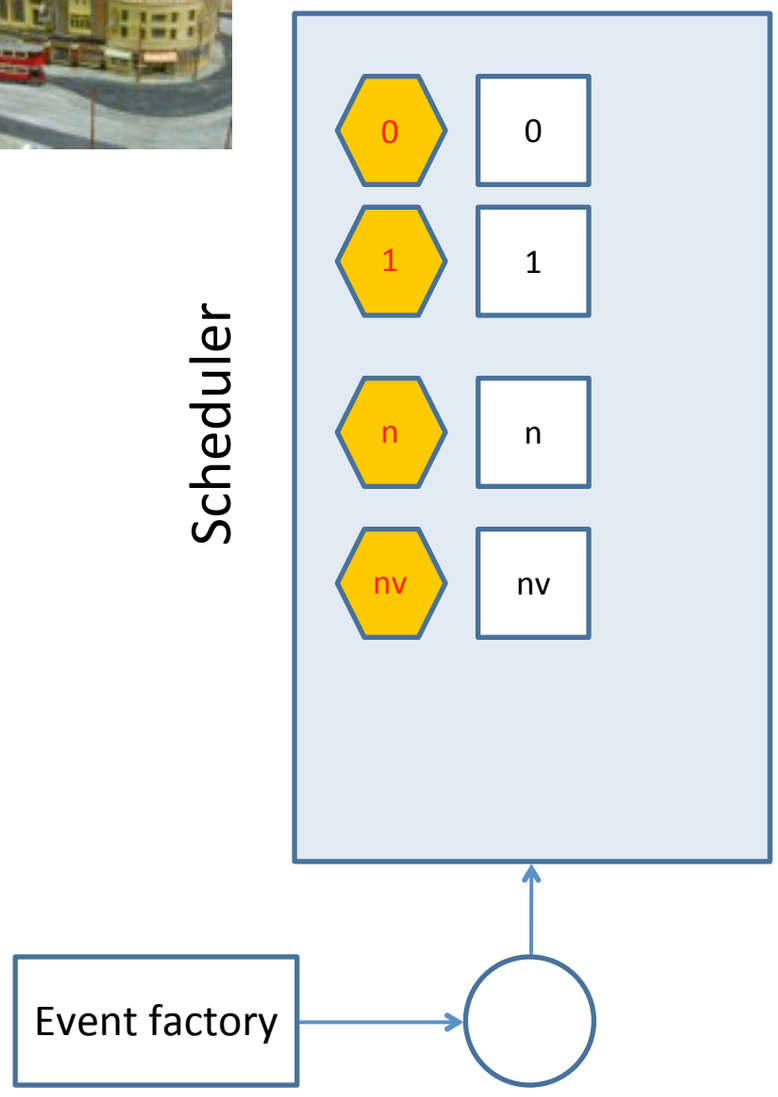
A playground for new ideas

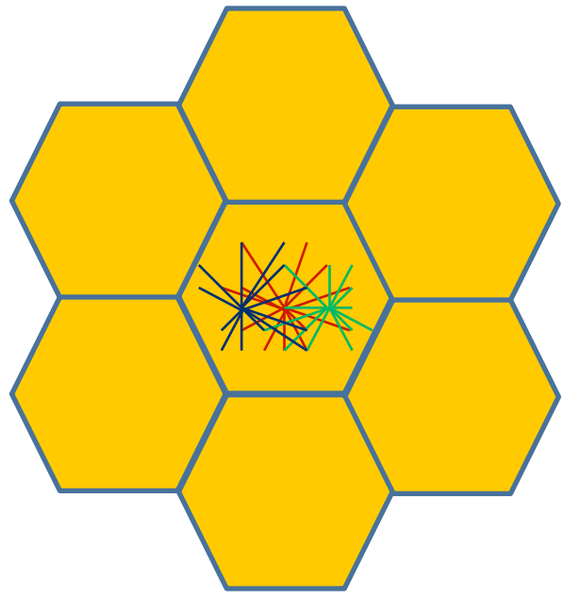
- Simulation prototype in the attempt to explore parallelism and efficiency issues
  - Start with simple physics but realistic geometry: can we implement a parallel transport model on threads exploiting data locality and vectorisation?
  - Ab-initio design of data structures and steering to easily exploit parallel architectures and share main data structures among threads
  - Can we achieve a continuous data flow from generation to digitization and I/O
- Events and primary tracks are independent
  - Transport together a vector of tracks coming from many events
  - Study how does scattering/gathering of vectors of tracks and hits impact on the simulation data flow
    - Can we push useful vectors downstream ?
- Start with toy physics to develop the appropriate data structures and steering code

Volume-oriented transport model

- Particles traversing a given geometry volume are transported together as a vector until the volume gets empty
  - Same volume ➔ local (vs. global) geometry navigation, same material and same cross sections
  - Load balancing: distribute all particles in a logical-volume into work units (baskets) to be transported by one thread
  - Work with vectors to allow for micro-parallelism
- Particles exiting a volume are distributed to baskets of the neighbor volumes until they exit the setup or disappear
  - Like a champagne cascade, but lower glasses can also fill higher ones…
  - Wait for the glass to be full before drinking the champagne…
  - Independent processing once the work is distributed

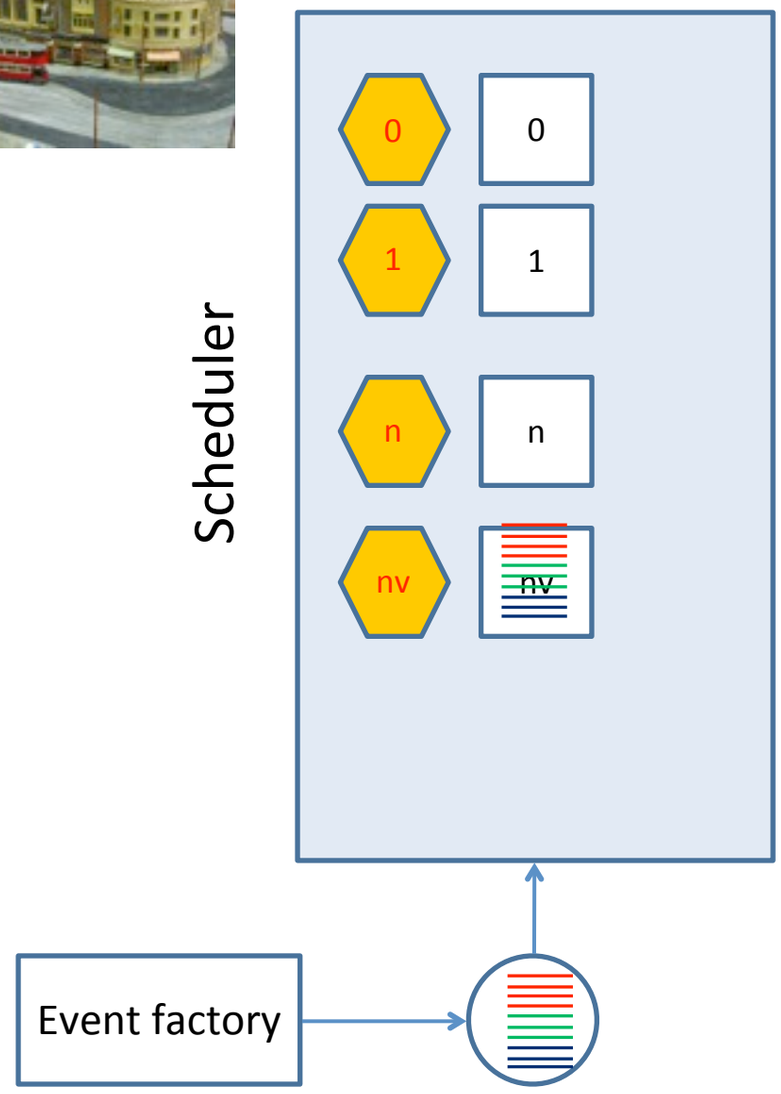
Transport model

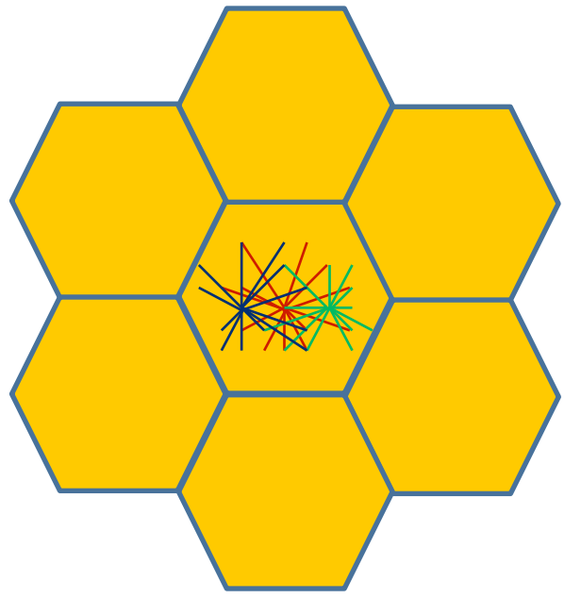




Transport model

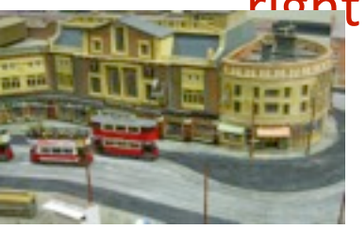
Inject events into a track container taken by a scheduler thread



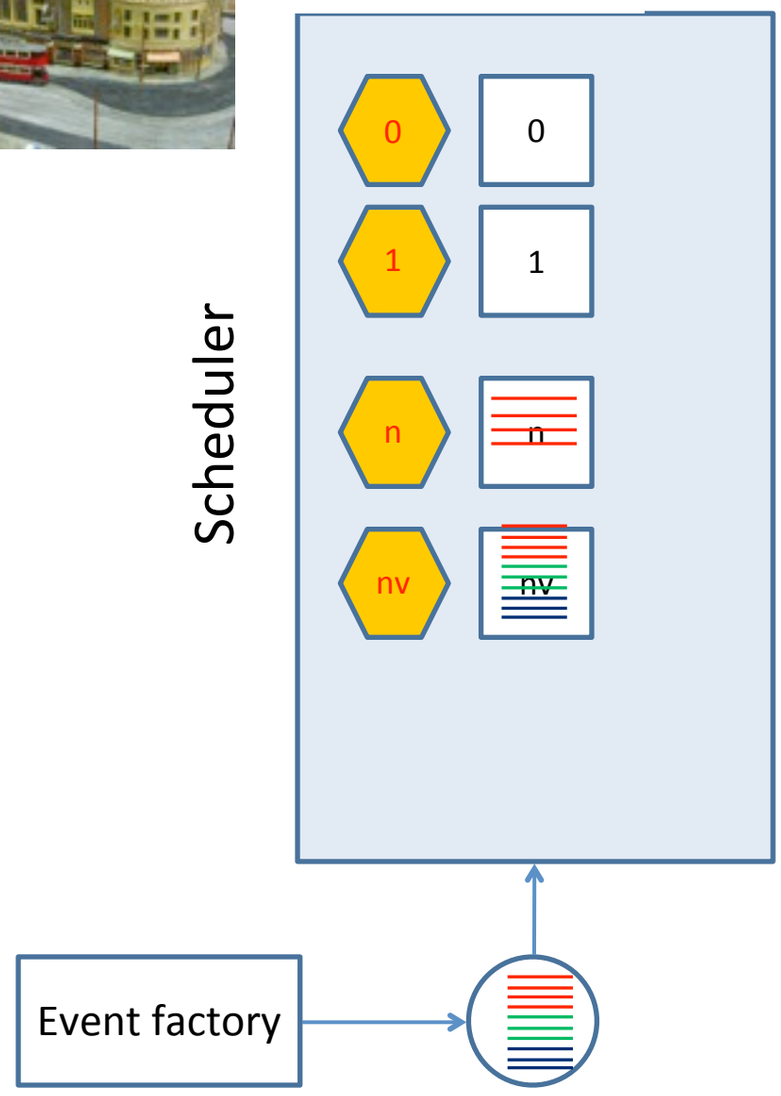


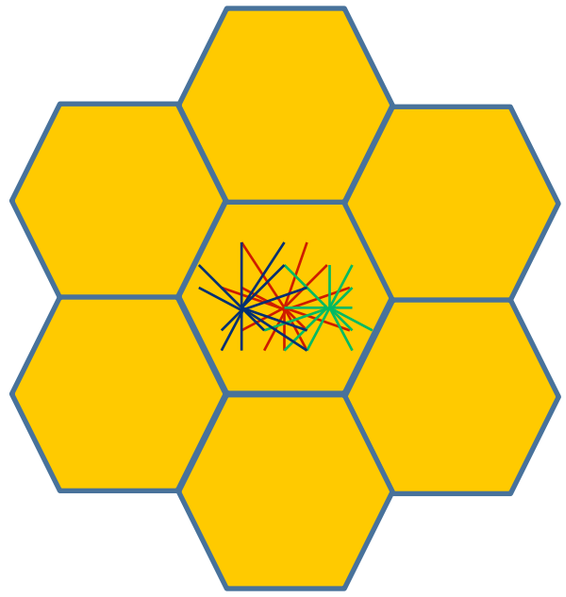
Transport model

The scheduler holds a track "basket" for each logical volume. Tracks go to the right basket.



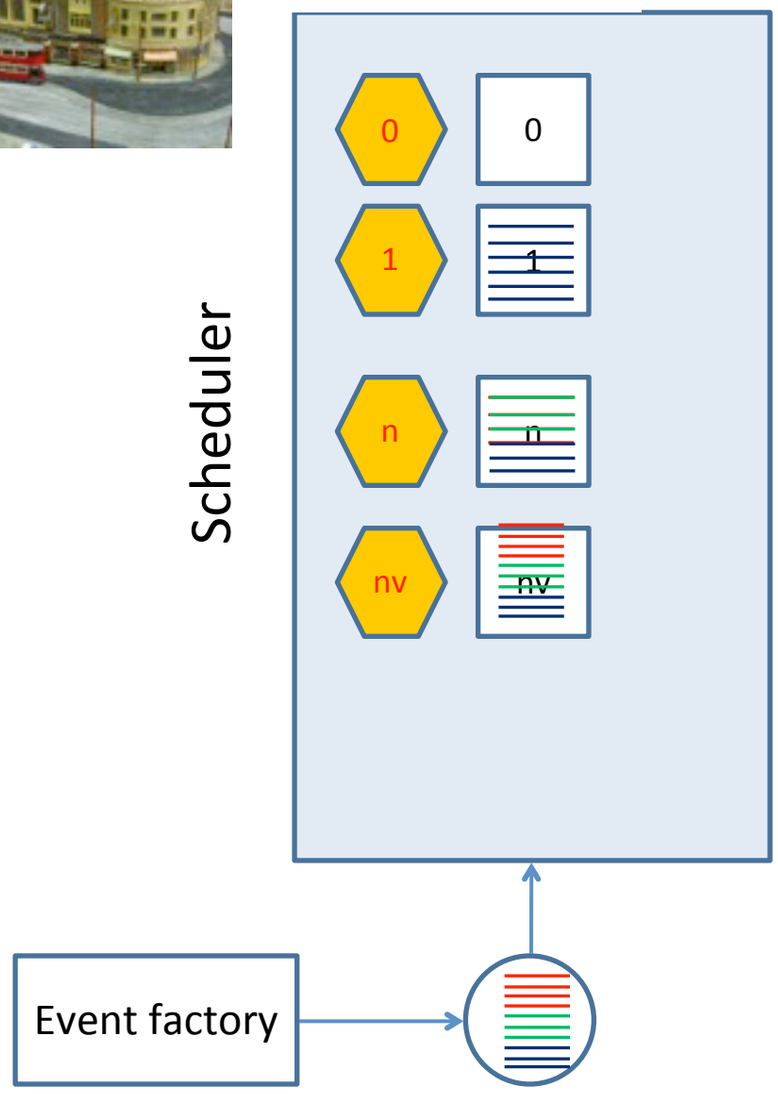
Track baskets (tracks in a single volume type)



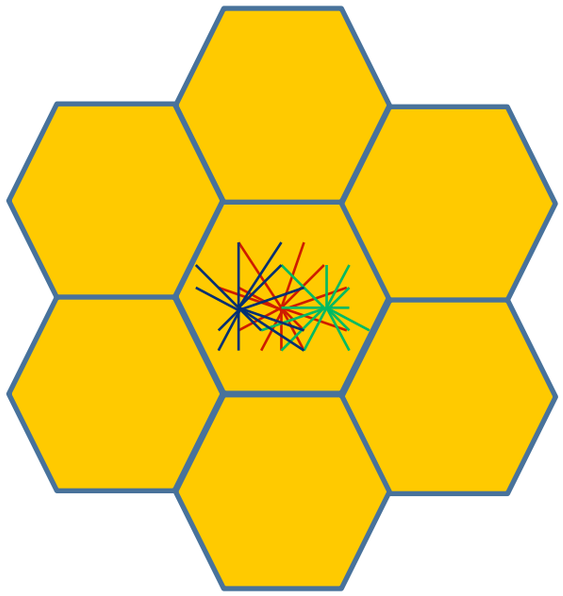


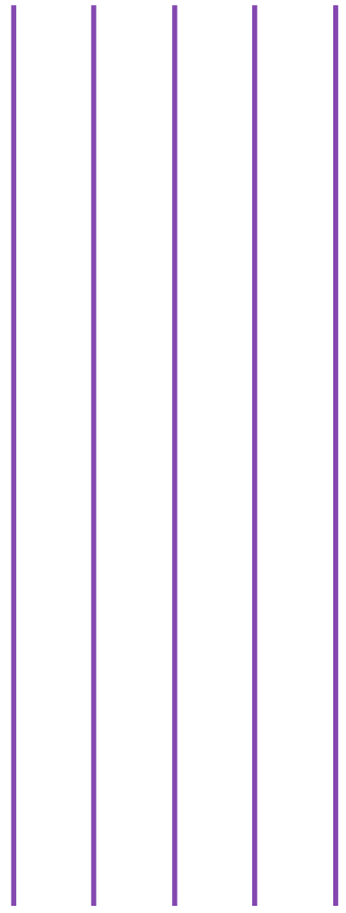
Transport model

Baskets are filled up to a threshold, then injected in a work queue



Track baskets (tracks in a single volume type)

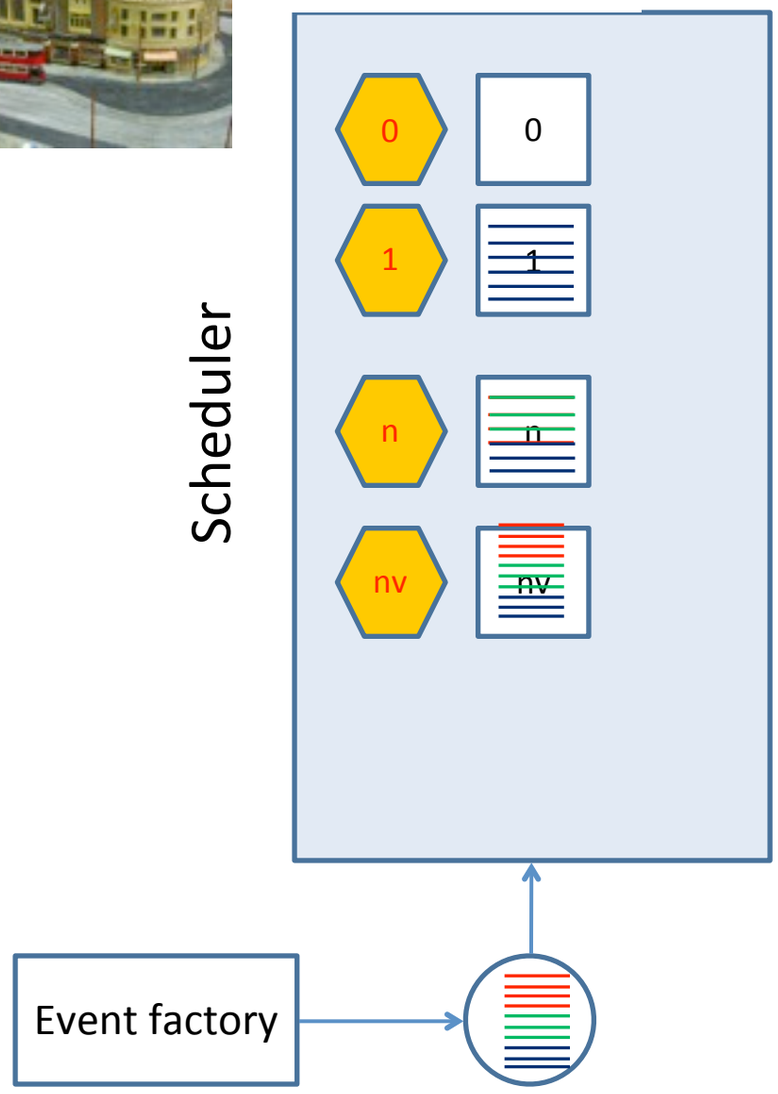


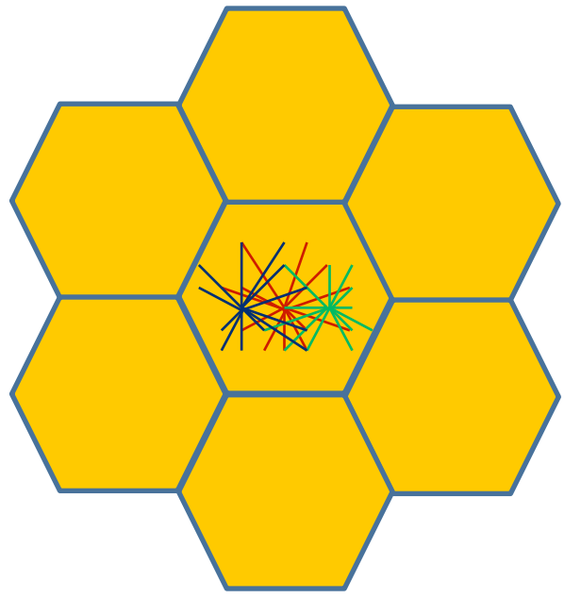


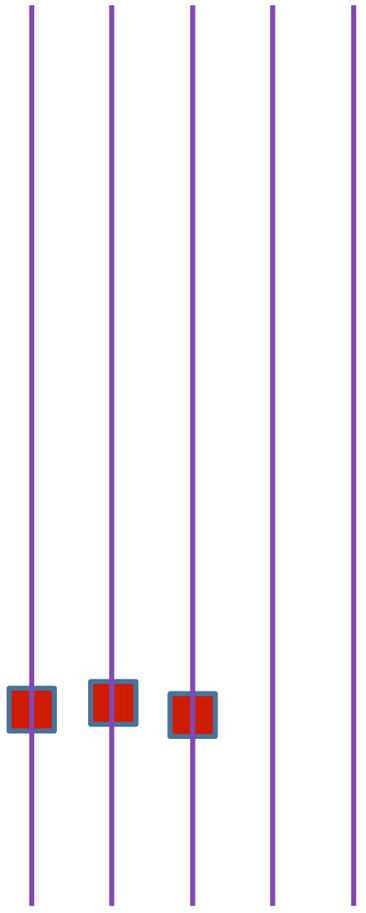
Transport model

Transport threads pick baskets “first in first out”

Track baskets (tracks in a single volume type)



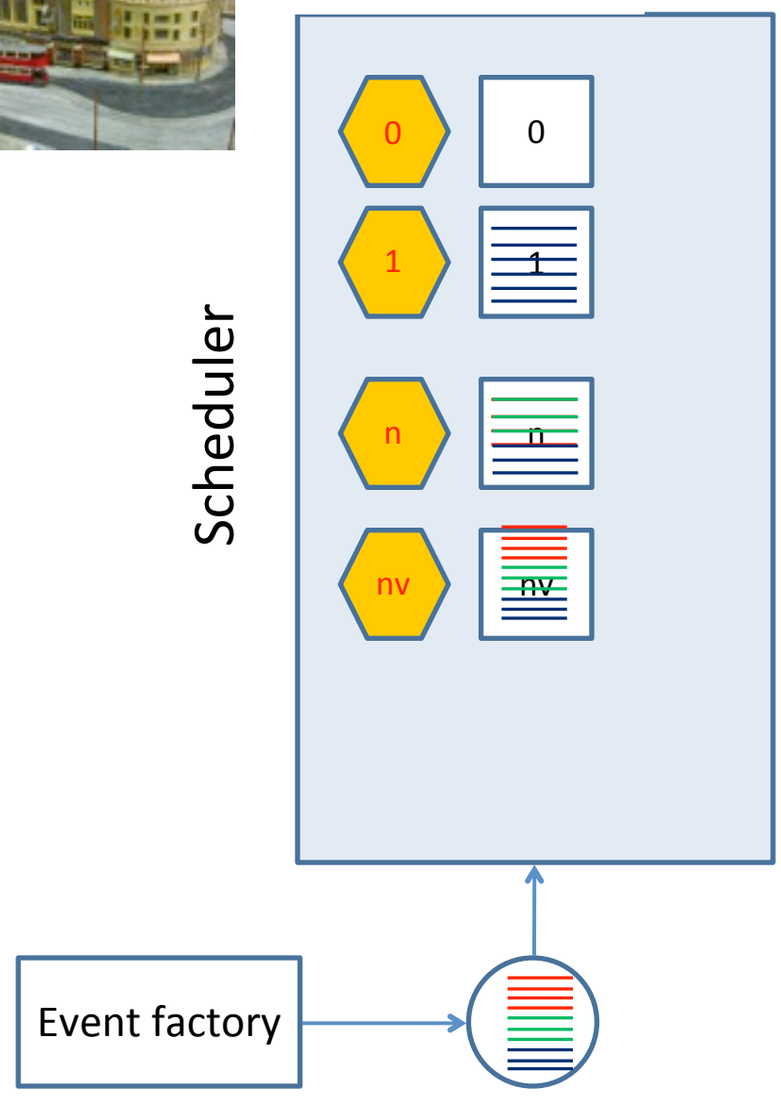


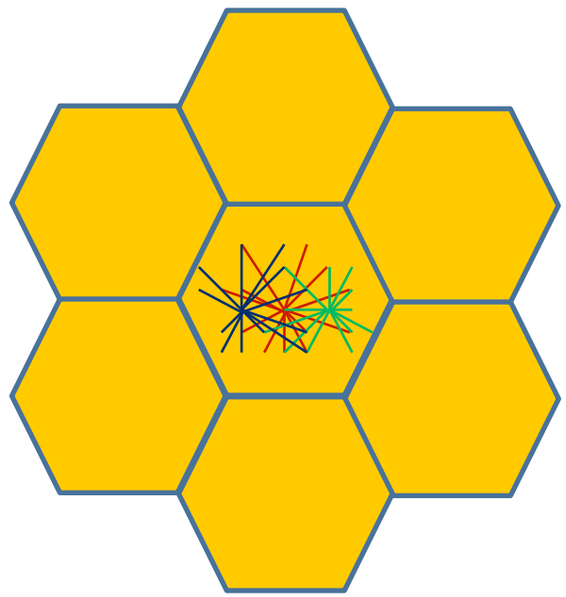


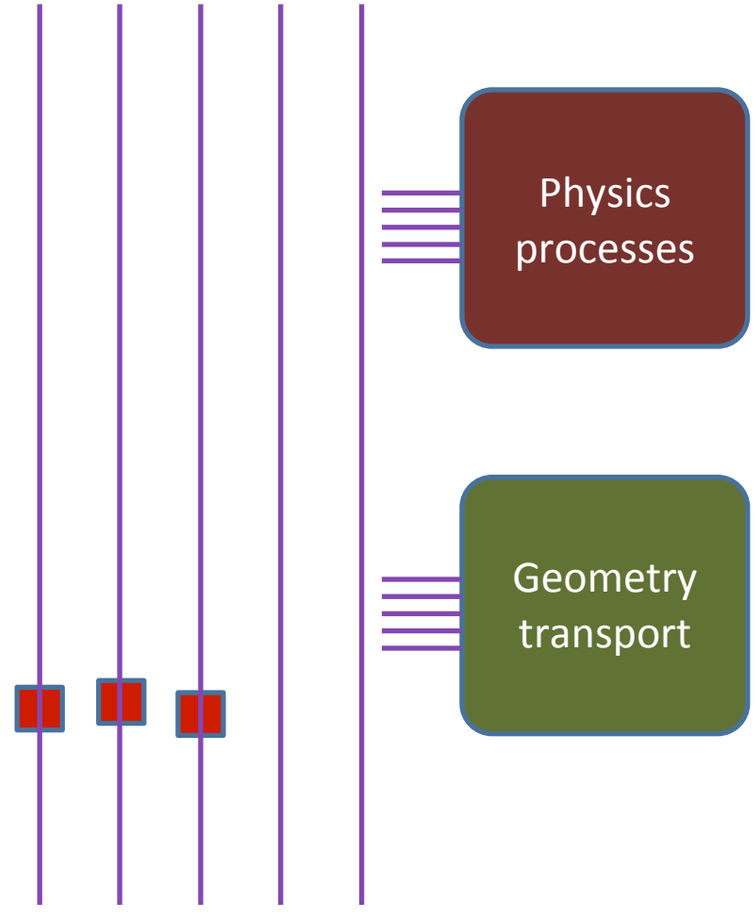
Transport model

Physics processes and geometry transport work with vectors

Track baskets (tracks in a single volume type)



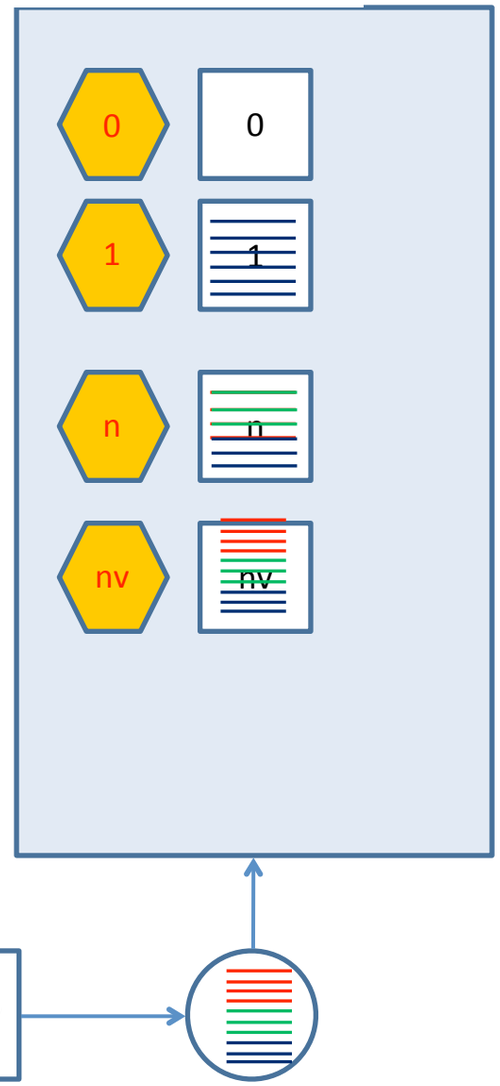


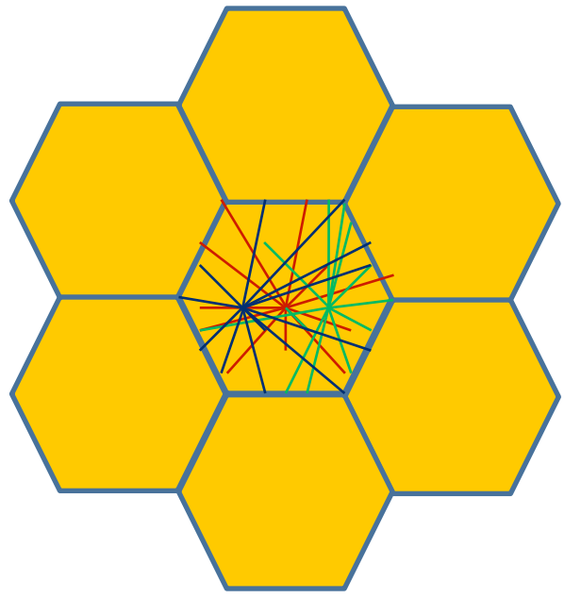


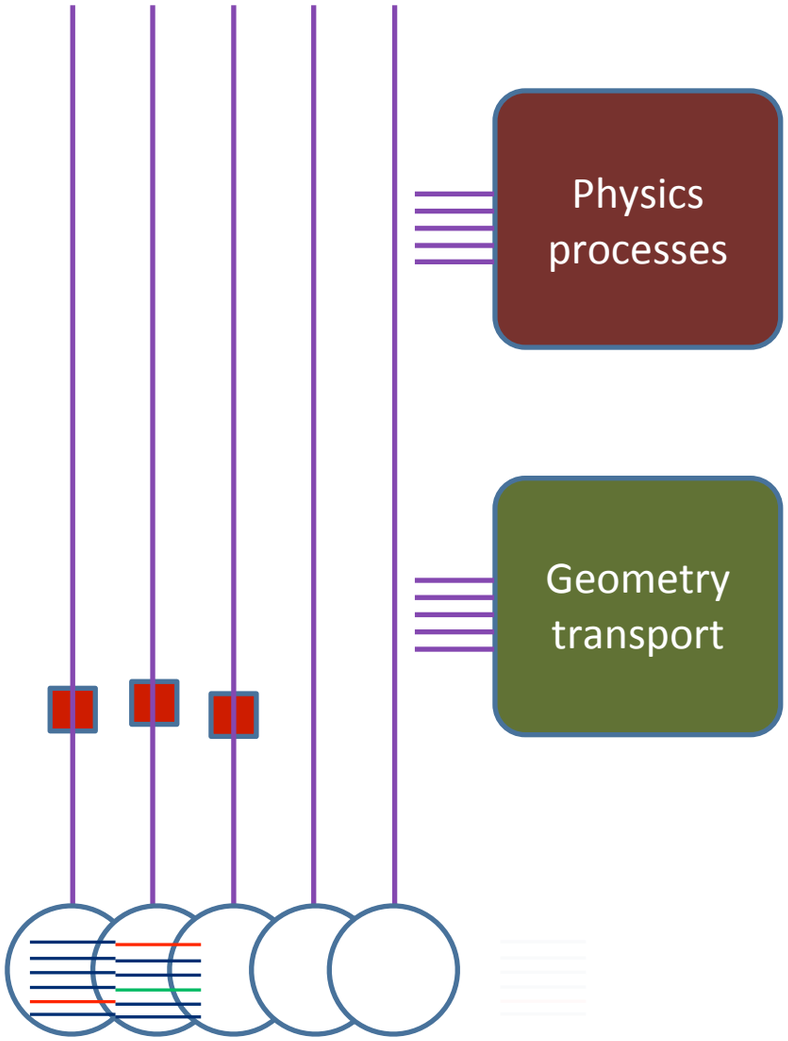
Transport model

Tracks are transported to boundaries, then dumped in a track collection per thread

Track baskets (tracks in a single volume type)



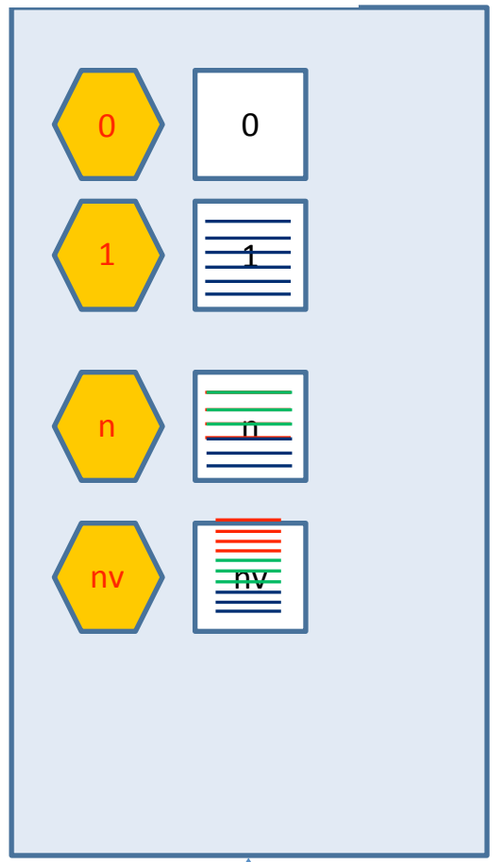


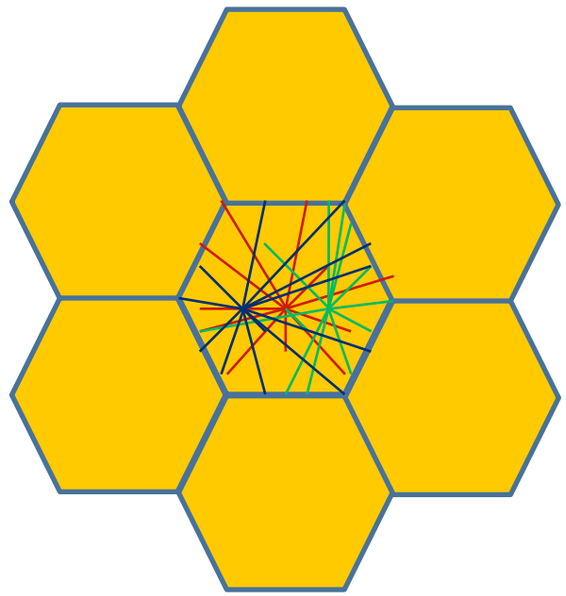


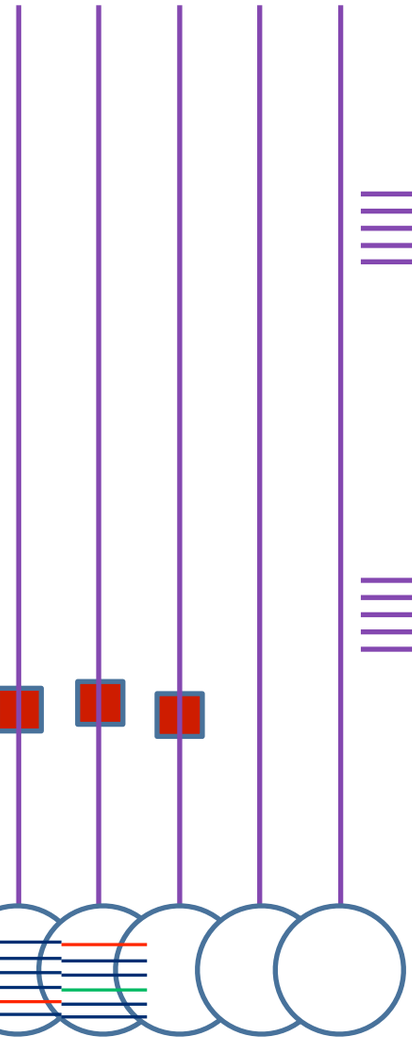
Transport model

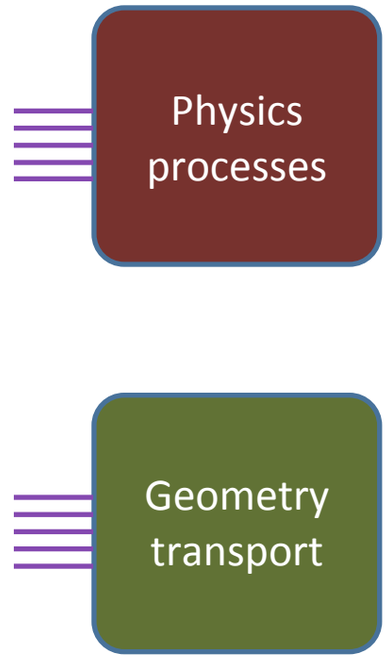
Track collections are pushed to a queue and picked by the scheduler

Track baskets (tracks in a single volume type)





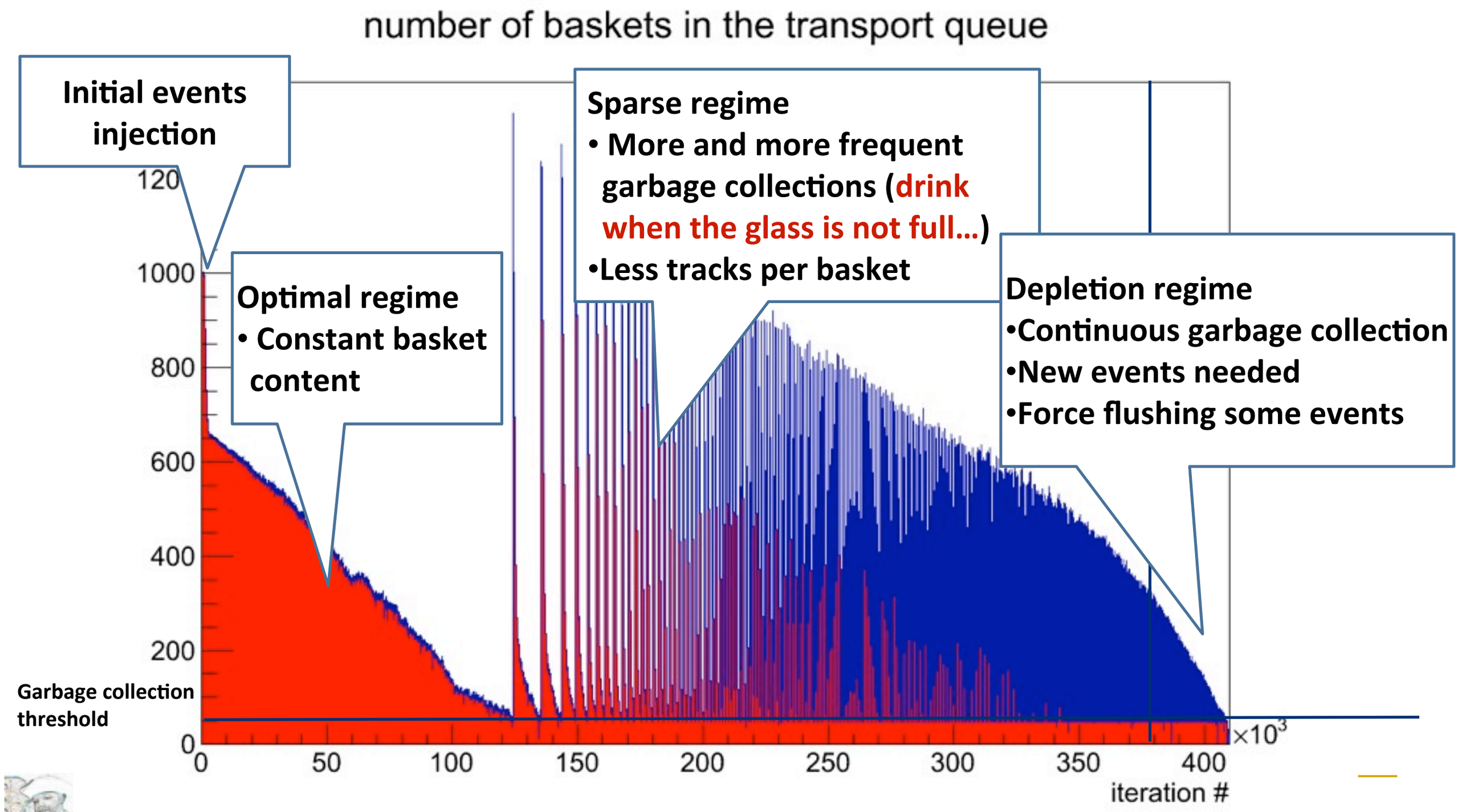




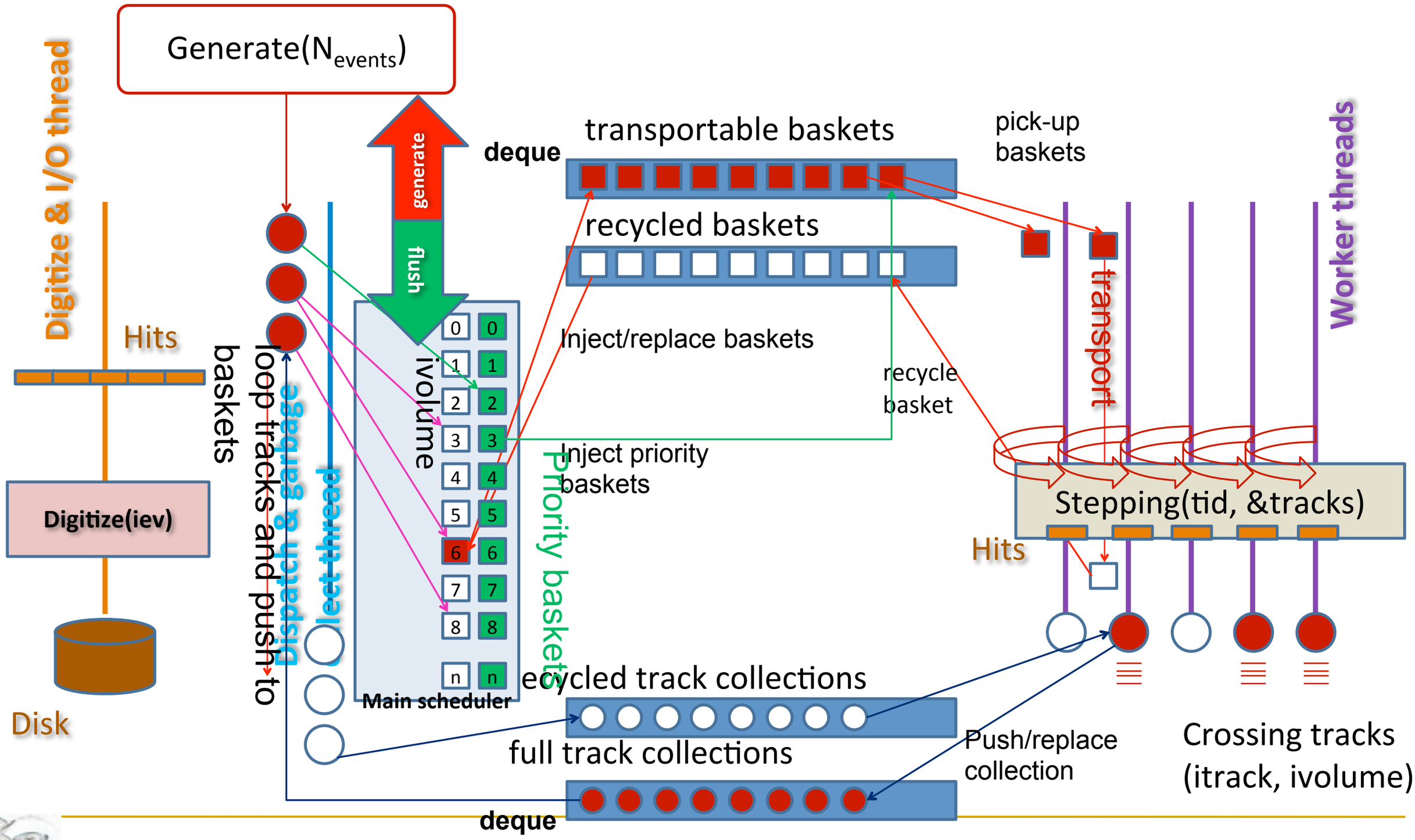
Track containers (any volume)

Event factory

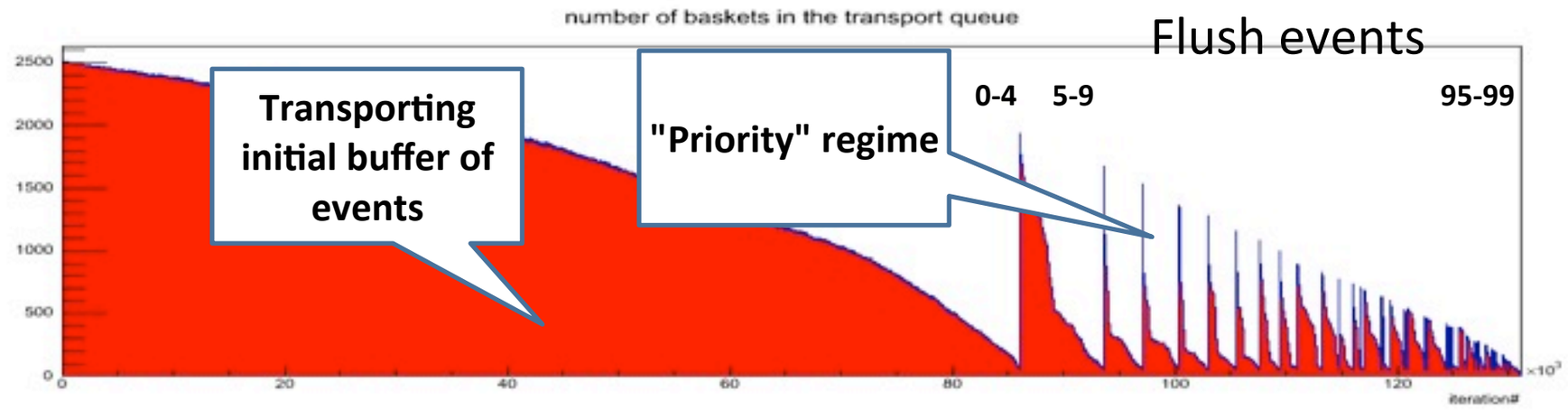
Processing phases – old lessons

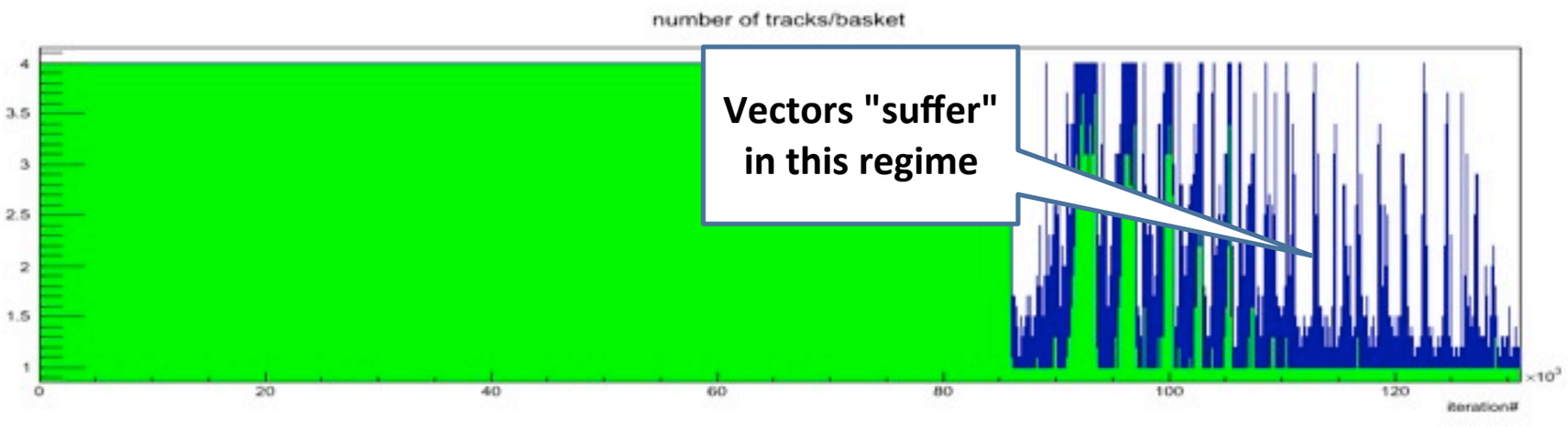


Prioritized transport

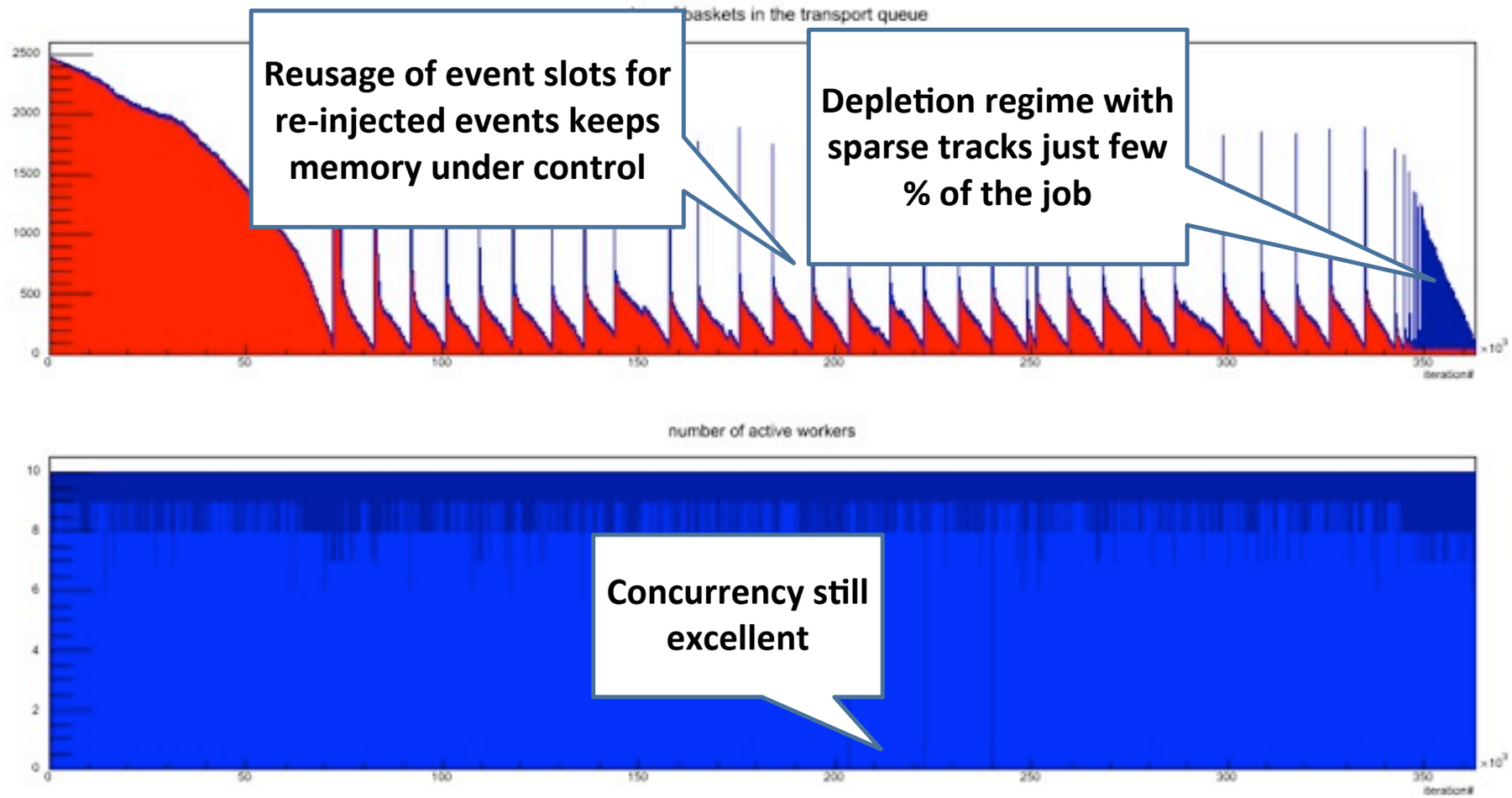


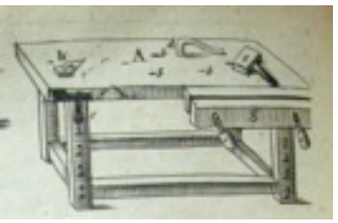
Depletion of baskets (no re-injection)



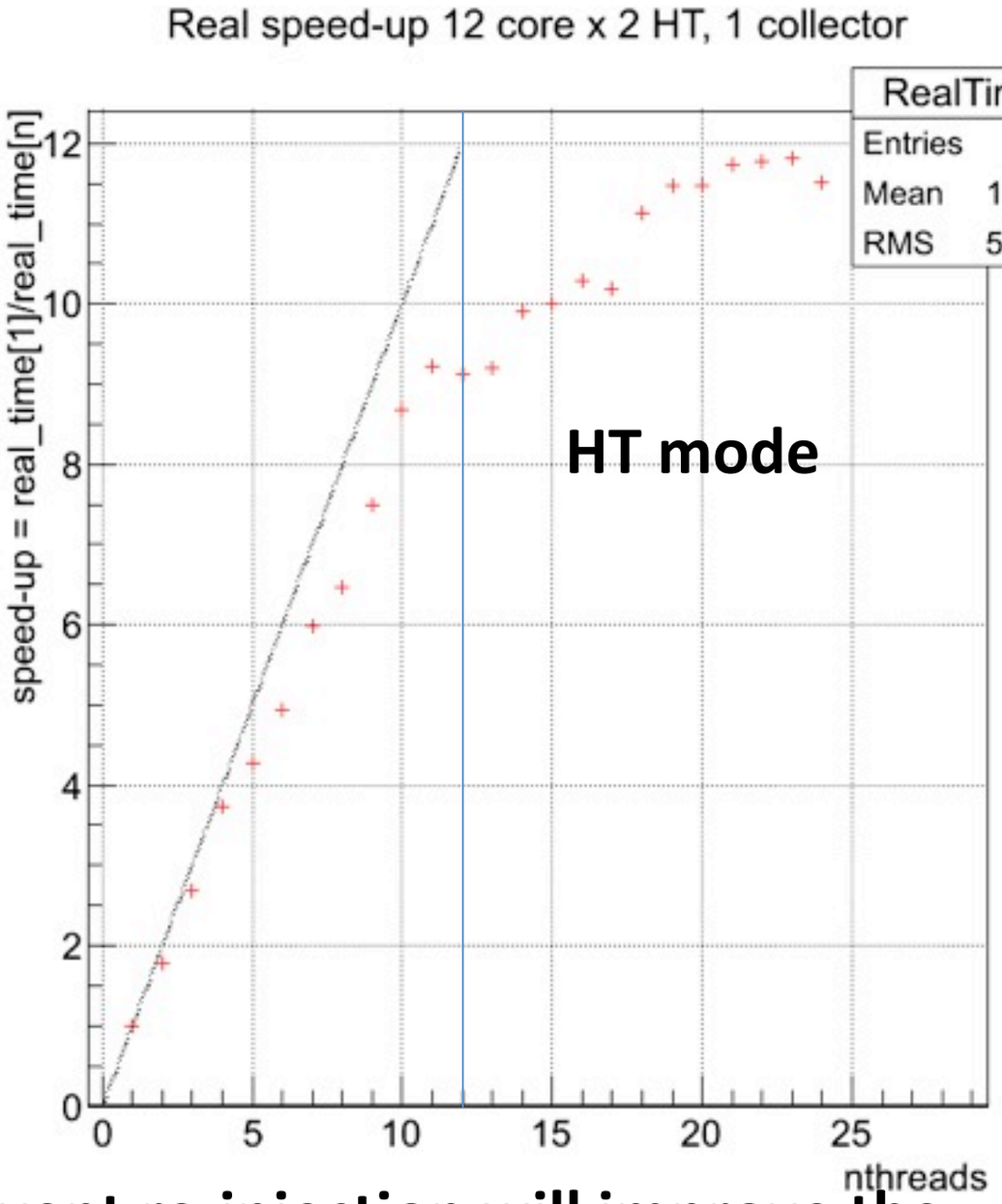


Buffered events & re-injection



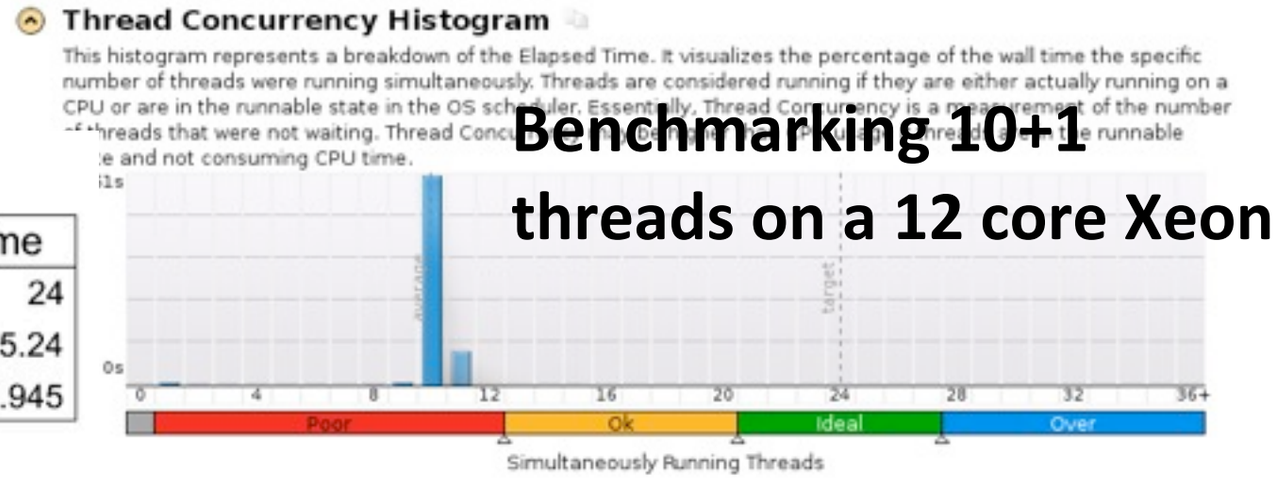


Preliminary benchmarks

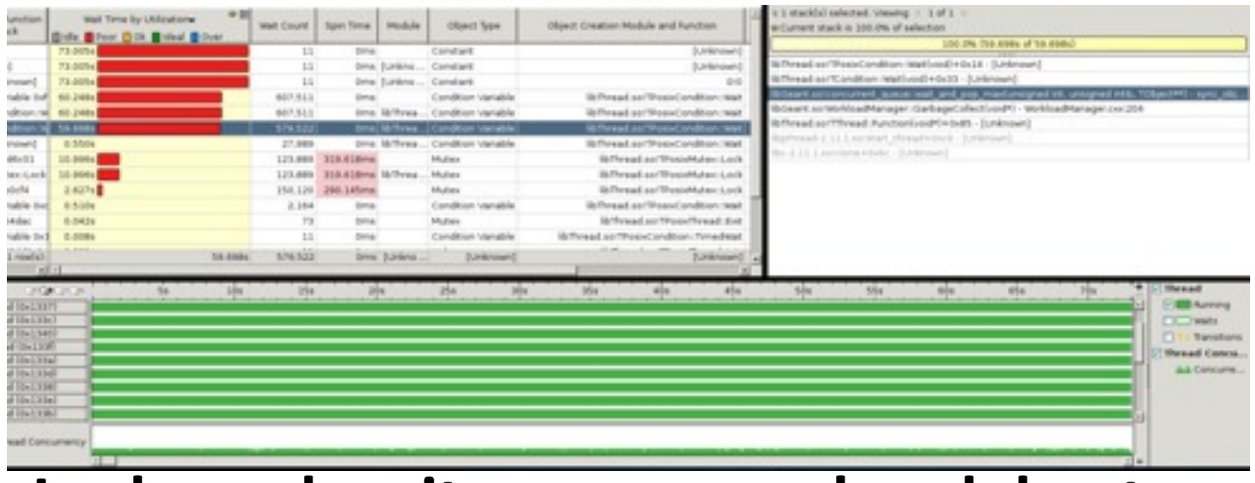


Event re-injection will improve the speed-up

Benchmarking 10+1 threads on a 12 core Xeon



Excellent CPU usage



Locks and waits: some overhead due to transitions coming from exchanging baskets via concurrent queues

Micro-Parallelism Opportunities

- The real gain in speed will come at the end from the exploitation of the (G/C)PU hardware
  - Vectors, Instruction Pipelining, Instruction Level Parallelism (ILP)
- Algorithms will be more appropriate for one or the other of these techniques
  - The idea being to expose the maximum amount of parallelism at the lowest possible granularity level
  - And then explore the optimisation opportunities
- This will give better code anyway even for simple architectures
  - e.g. ARM CPUs

Next steps

- Include hits, digitization and I/O to test the full flow (ROOT)
  - Factories allowing contiguous pre-allocation and re-usage of user structures
    - MyHit *hit = HitFactory(MyHit::Class())->NextHit(); hit->SetP(particle->P());
- Introduce realistic EM physics (GEANT4)
  - Tuning prototype parameters, better estimate memory requirements
- Re-design transport models from a “plug-in” perspective (FORUM, GAUDI)
  - E.g. ability to use fast simulation on per-track basis
- Look further to micro-parallelism options (FORUM, OpenLab)
  - Compilers and extensions: C++11, Intel Cilk array notation, OpenACC, OpenMP...
  - Check impact of vector-friendly data restructuring
    - Vector of objects -> object with vectors

Next steps

- Push “vectors” to lower level (ROOT, FORUM, OpenLab)
  - Geometry and physics
- GPU is a great challenge for simulation code (ROOT, FORUM, OpenLab)
  - Localize hot-spots with high CPU usage and low data transfer requirements
  - Test performance on data flow machines
- Collect requirements & examples for fast simulation (experiments)

Timeline

- Discussion with the experiments about fast MC & plan on how to proceed
  - June
- Basic structure for hits and digits generation
  - June
- Simplified electromagnetic interactions
  - In time for CHEP (October 14 - 18, 2013)
- Optimisation of geometry and physics methods
  - First results September – continuous activity
  - Estimate maximum/realistic gain
  - Gain experience with emerging architectures and devices

And further in the future...

- Variance reducing & biased transport
- Low energy neutrons saga
- Radio-protection applications
- Optical photon propagation
- Collaboration with other fields (health physics, medicine…)
- Coupling with CAD, Finite Elements, structural modelling, heath transport…
- Beam transport dynamics
- Material damage, heating, mechanical impact

Opportunity

- From 1992~2004 there was an intense R&D about future HEP software
  - These were the C++ years
- From 2004 people just sat down to work and put in practice what they learnt
  - Learning much more… and building the current production systems
- I think a new cycle is beginning
  - LHC upgrade, ILC/CLIC studies, FAIR preparation
- Starting the prototype now is the right moment to make an impact
- More generally I believe we should promote a new round of brainstorming between IT, PH, OpenLab and experiments within and without CERN

Thank you!