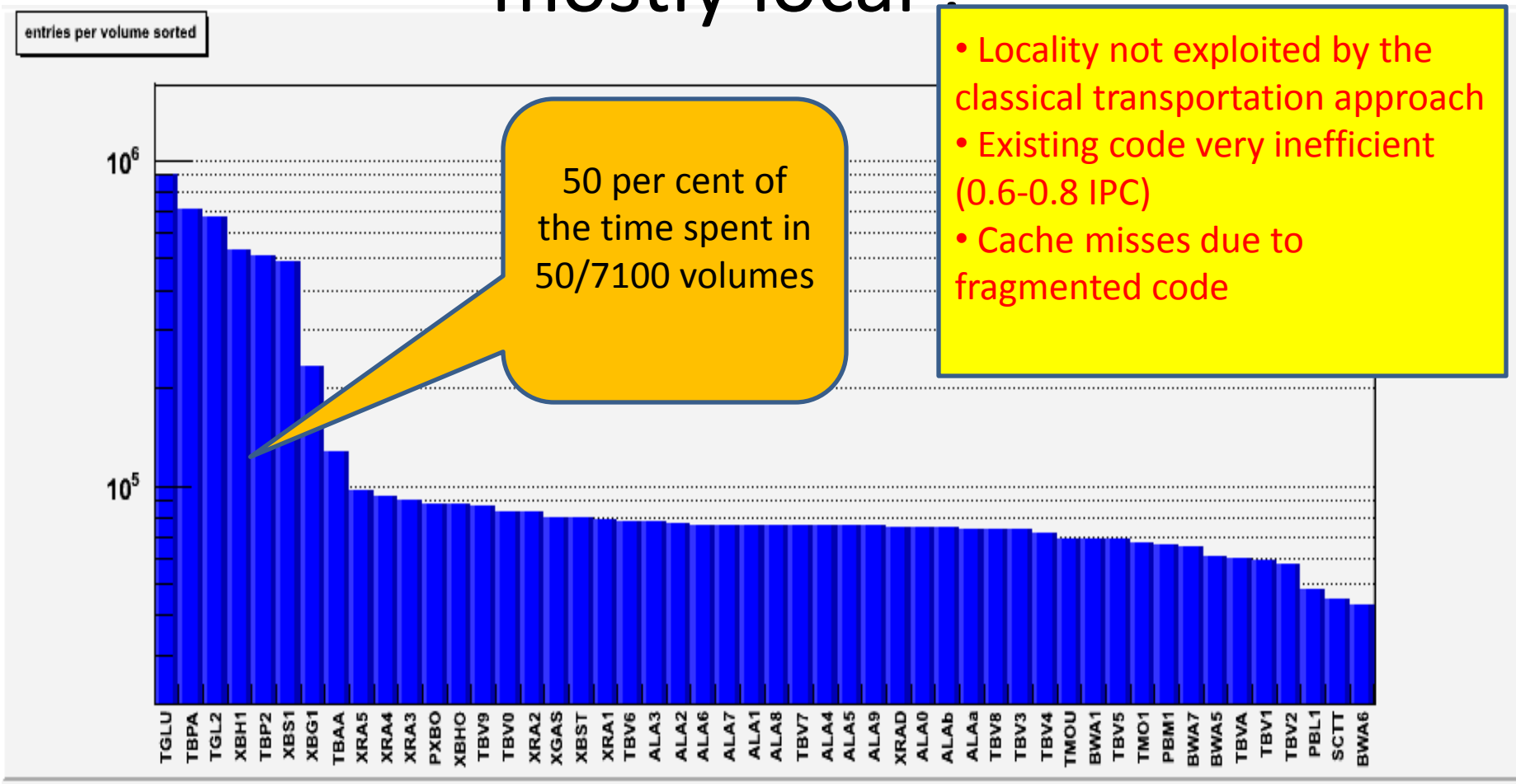
Report on Vector Prototype

J.Apostolakis, R.Brun, F.Carminati,
A. Gheata

10 September 2012

Simple observation: HEP transport is mostly local !



ATLAS volumes sorted by transport time. The same behavior is observed for most HEP geometries.

A playground for new ideas

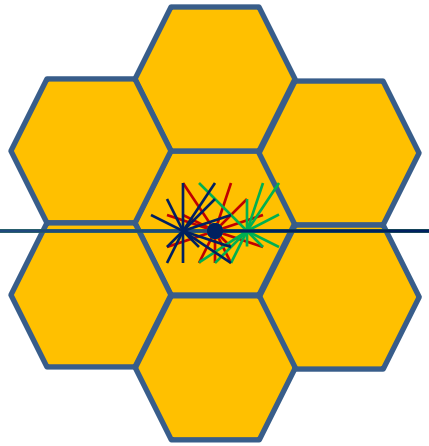
- Simulation prototype in the attempt to explore parallelism and efficiency issues
  - Basic idea: simple physics, realistic geometry: can we implement a parallel transport model on threads exploiting data locality and vectorisation?
  - Clean re-design of data structures and steering to easily exploit parallel architectures and allow for sharing all the main data structures among threads
  - Can we make it fully non-blocking 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
- Start with toy physics to develop the appropriate data structures and steering code
  - Keep in mind that the application should be eventually tuned based on realistic numbers

Volume-oriented transport model

- We implemented a model where all 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 from a volume type into smaller work units called **baskets**, give a basket to a transport thread at a time
  - Steering methods working with vectors, allowing for future auto-vectorisation
- Particles exiting a volume are distributed to baskets of the neighbor volumes until exiting the setup or disappearing
  - Like a champagne cascade, but lower glasses can also fill top ones…
  - No direct communication between threads to avoid synchronization issues

Event injection

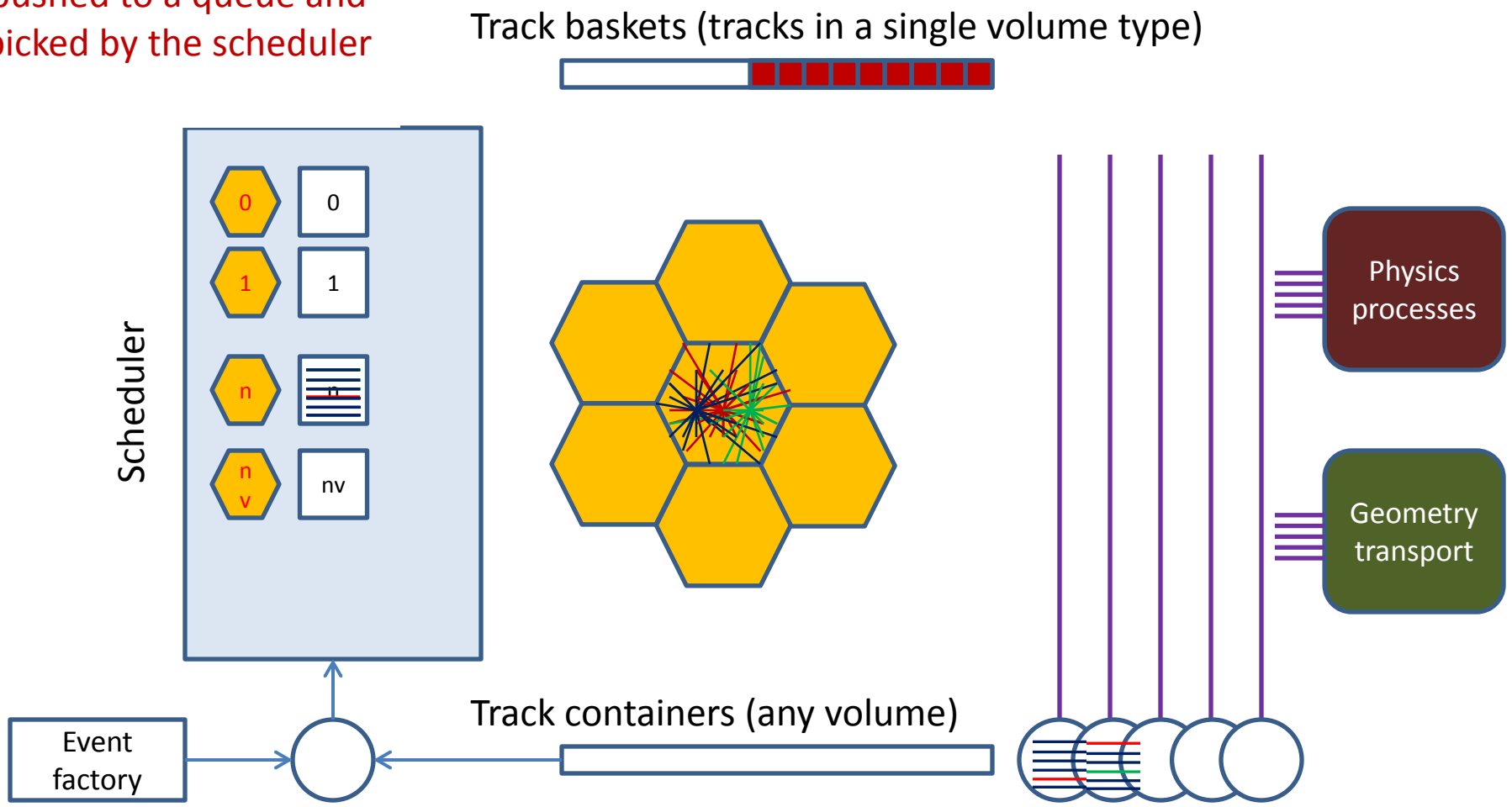
More events better to cut event tails and fill better the pipeline !



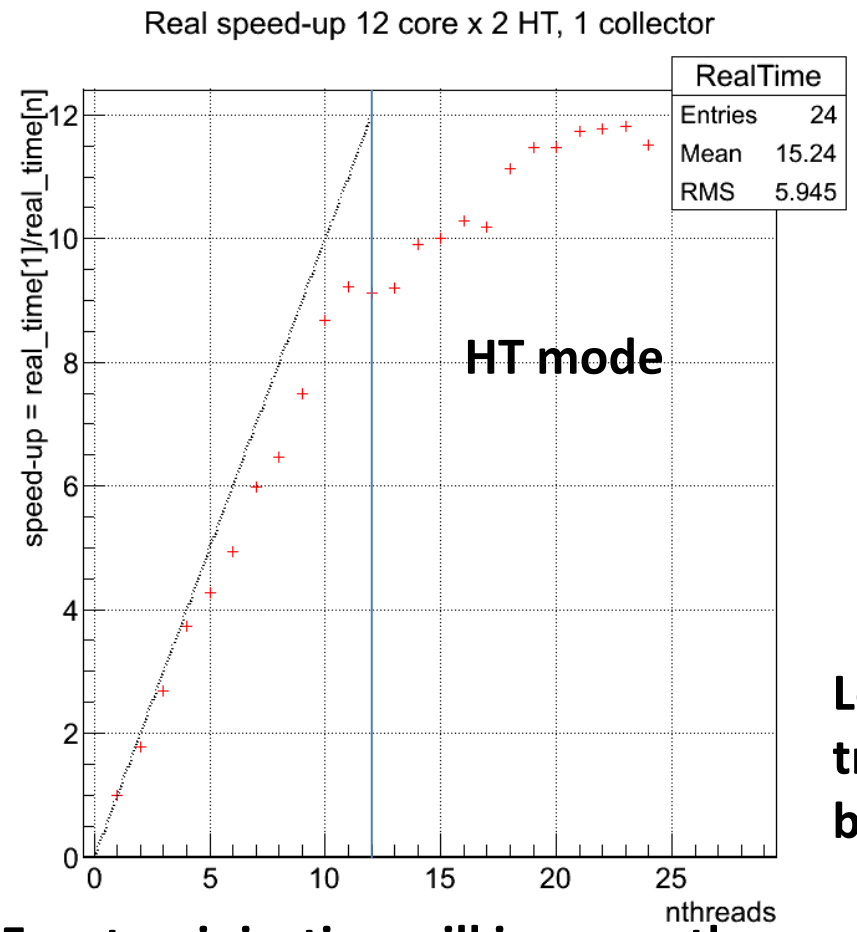
Realistic geometry + event generator

Transport model

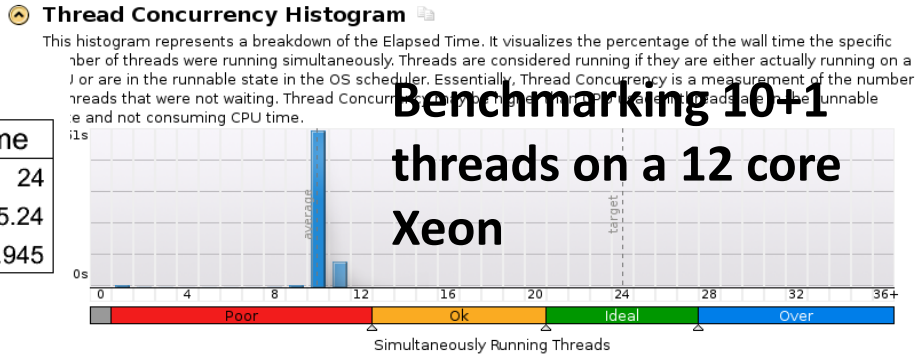
Track collections are pushed to a queue and picked by the scheduler

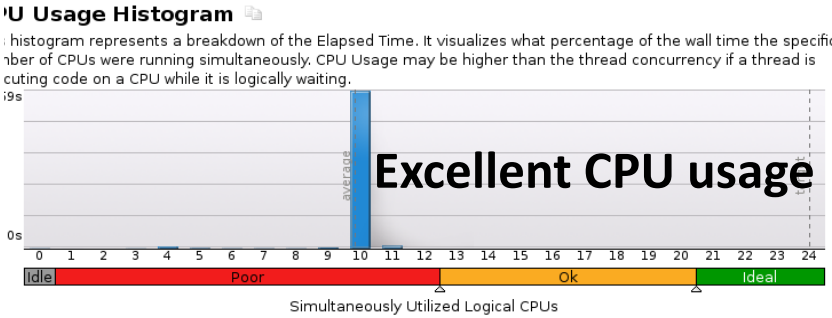


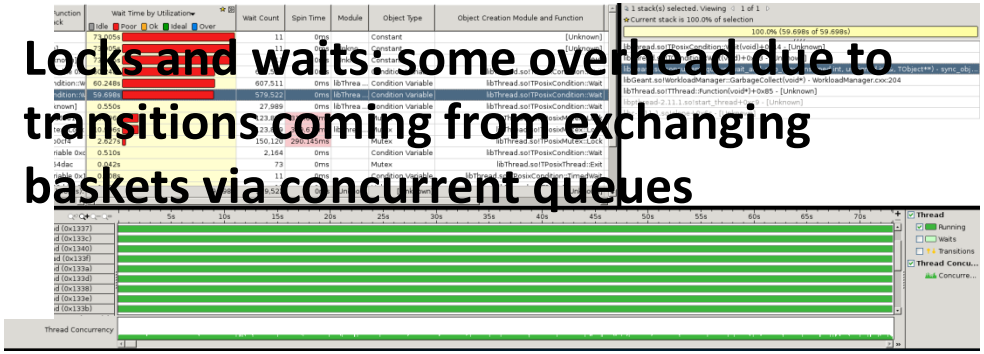
Preliminary benchmarks



Event re-injection will improve the speed-up

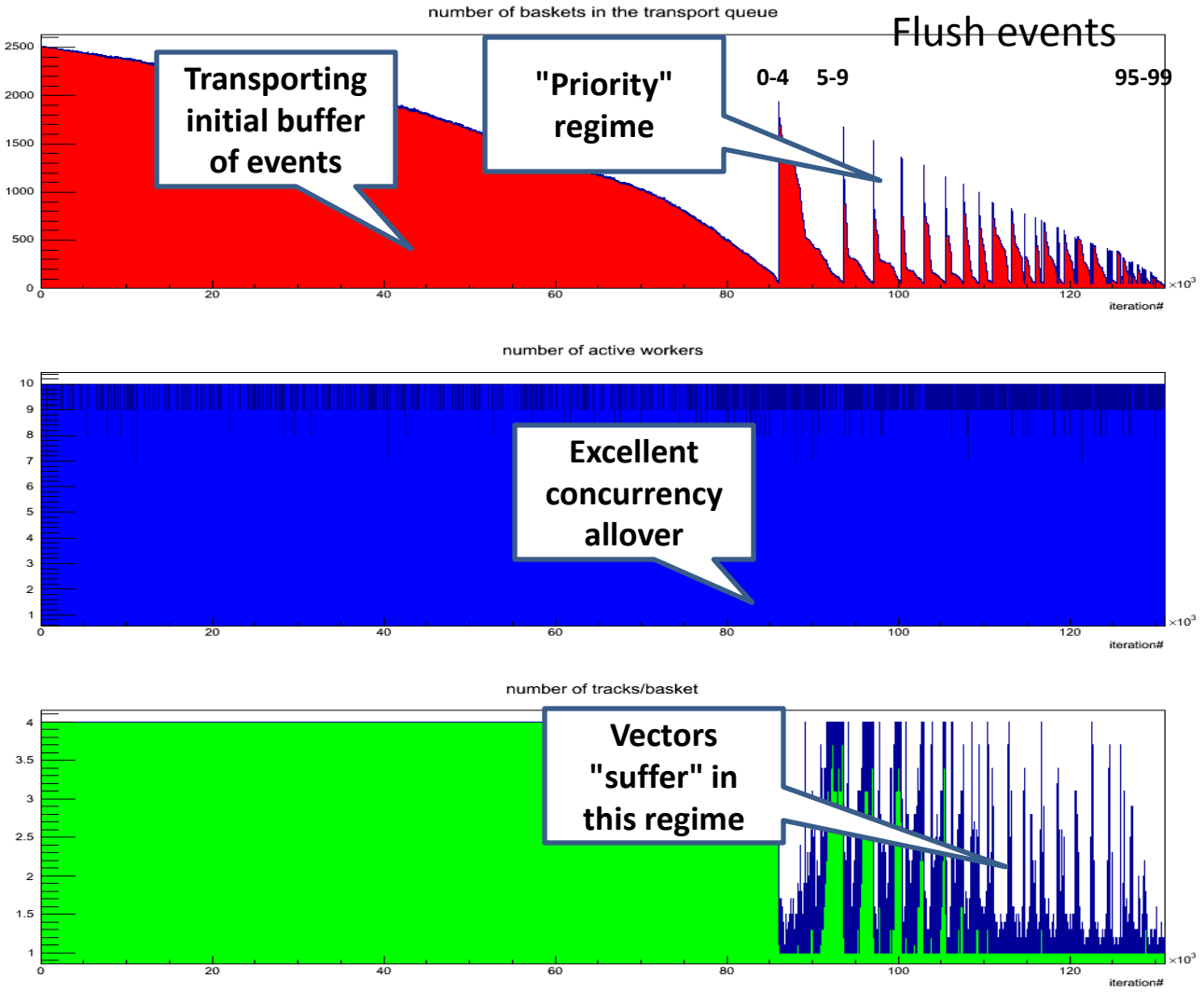




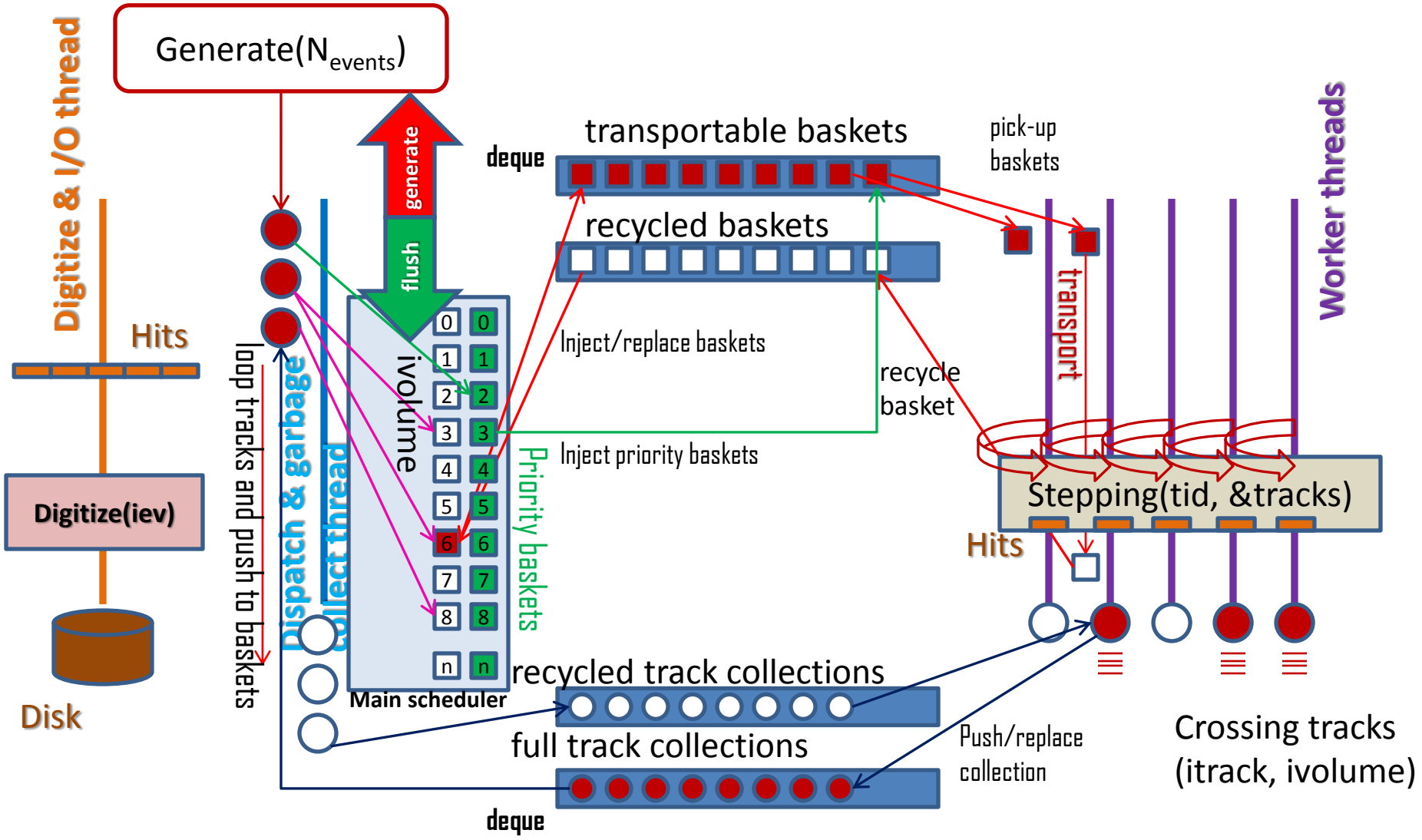


Locks and waits: some overhead due to transitions coming from exchanging baskets via concurrent queues

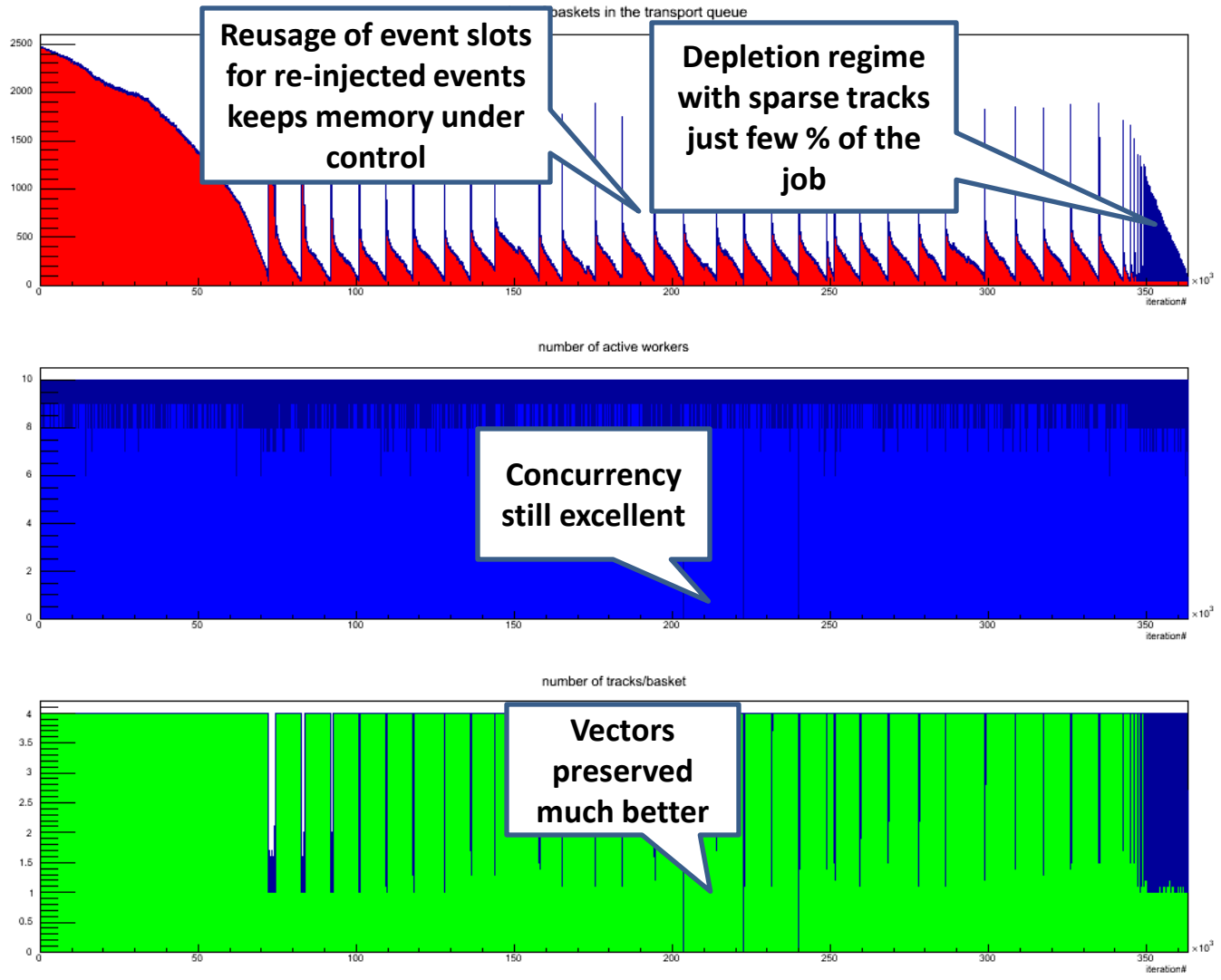
Evolution of populations



Prototype implementation



Buffered events & re-injection



Next steps

- Include hits, digitization and I/O in the prototype
  - 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
  - Tuning model parameters, better estimate memory requirements
- Re-design transport models from a “plug-in” perspective
  - E.g. ability to use fast simulation on per-track basis
- Look further to auto-vectorization options
  - New compiler features, Intel Cilk array notation, ...
  - Check impact of vector-friendly data restructuring
    - Vector of objects -> object with vectors
- Push vectors lower level
  - Geometry and physics models as main candidates
- GPU is a great challenge for simulation code
  - Localize hot-spots with high CPU usage and low data transfer requirements
  - Test performance on data flow machines

Outlook

- Existing simulation approaches perform badly on the new computing infrastructures
  - Projecting this in future looks much worse...
  - The technology trends motivate serious R&D in the field
- We have started looking at the problem from different angles
  - Data locality and contiguity, fine-grain parallelism, data flow, vectorization
  - Preliminary results confirming that the direction is good
- We need to understand what can be gained and how, what is the impact on the existing code, what are the challenges and effort to migrate to a new system…

Thank you!