RE-THINKING PARTICLE TRANSPORT IN THE MANY-CORE ERA

J.APOSTOLAKIS, R.BRUN, F.CARMINATI,A.GHEATA
CHEP 2012, NEW YORK, MAY

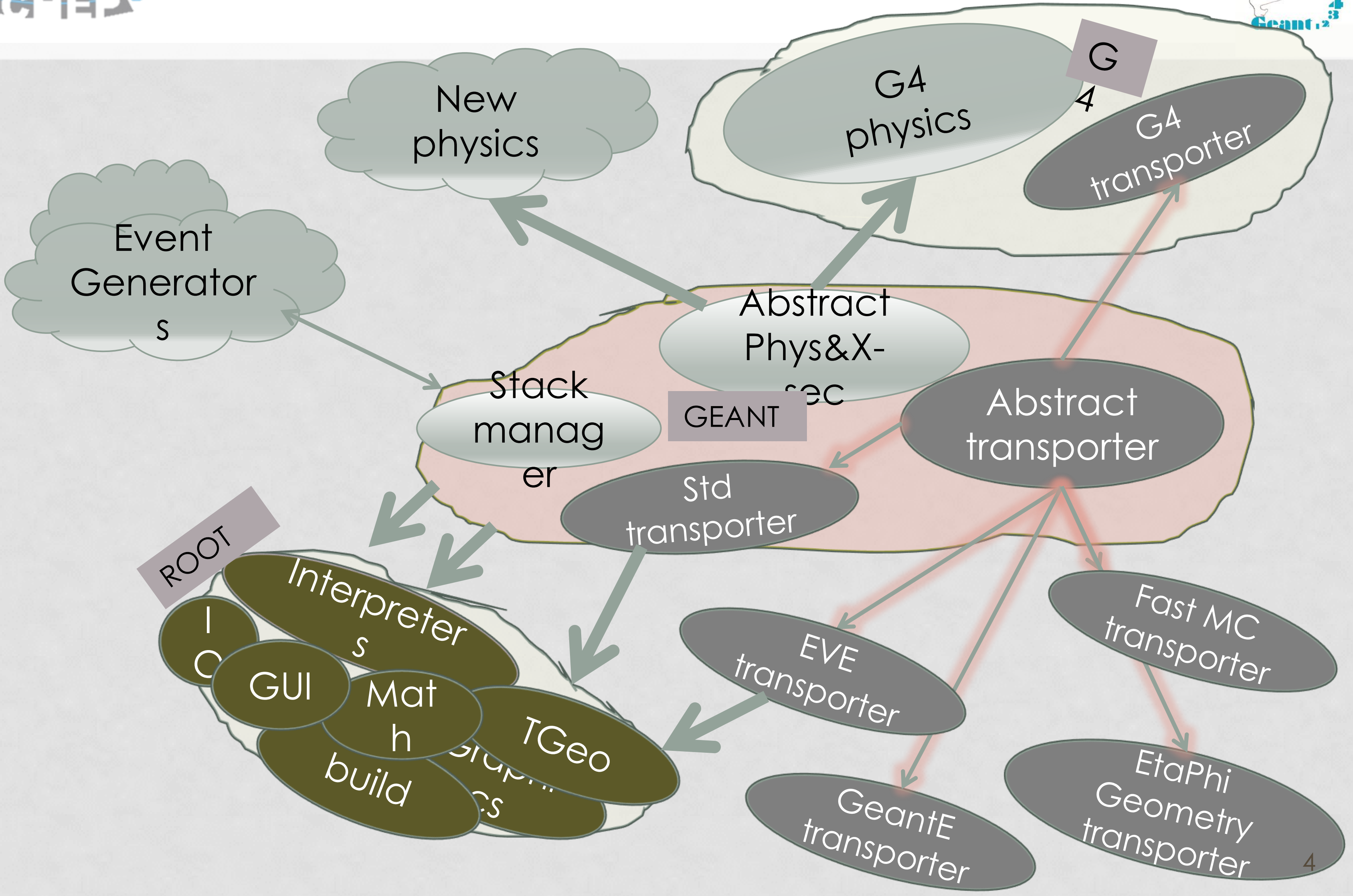
A STEP FORWARD… (FUNCTIONALITY)

- The LHC experiments have expressed the requirement of a continuum spectrum from very detailed to fast simulation
- The idea is to develop a new framework integrating various levels of fast and detailed simulation
  - Keeping services, geometry, I/O and scoring the same (as far as possible)
  - The model is the one of ROOT VMC, but substantially extended
- This will be essential for the new generation of experiments

A STEP FORWARD… (SPEED)

- We have been running Monte Carlo simulations for decades
  - A very large experience has been gained, reflected in the quality of the physics, the complexity of the setups and the precision of the transport
  - State of the art full simulation in a sequential approach…
- CPU frequency has an upper limit, technology has made a turnaround
  - Not yet followed by simulation software
- After a serious investigation, it becomes clear that a rather complete rethinking of the code becomes necessary to exploit the new hardware

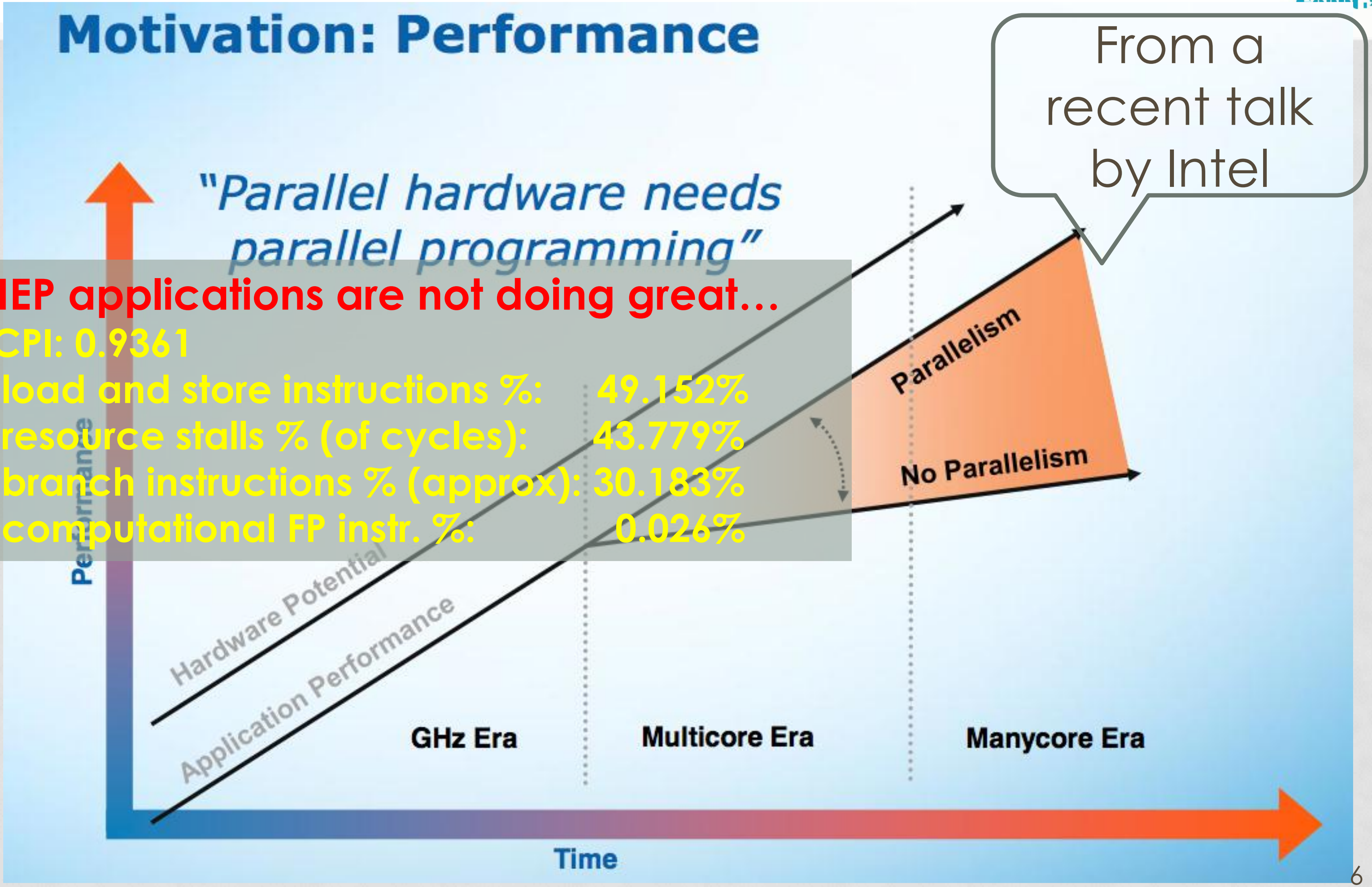
GLOBAL VIEW


New physics
G4 physics
G4
G4 transporter
Event Generators
Abstract Phys&X-sec
Stack manager
GEANT
Abstract transporter
Std transporter
ROOT
Interpreters
IO
GUI
Math
Graphics
build
TGeo
EVE transporter
Fast MC transporter
GeantE transporter
EtaPhi Geometry transporter

MOTIVATION FOR LOOKING INTO PARALLELISM

- Parallel architectures are evolving fast
  - Task parallelism in ever growing hybrid configurations
  - Instruction level parallelism (ILP) exploited more and more
    - 4 FLOP/cycle on modern processors
- HEP applications are using very inefficiently these resources
  - Running in average 0.6 instructions/cycle
  - Bad or inefficient usage of C++ leads to scattered data structures and cache misses (both instruction and data)
- We have evaluated the impact of changing the transport strategy on parallel architectures, trying to spot the weak points and a possible winning strategy

DOING NOTHING WILL ENLARGE THE GAP

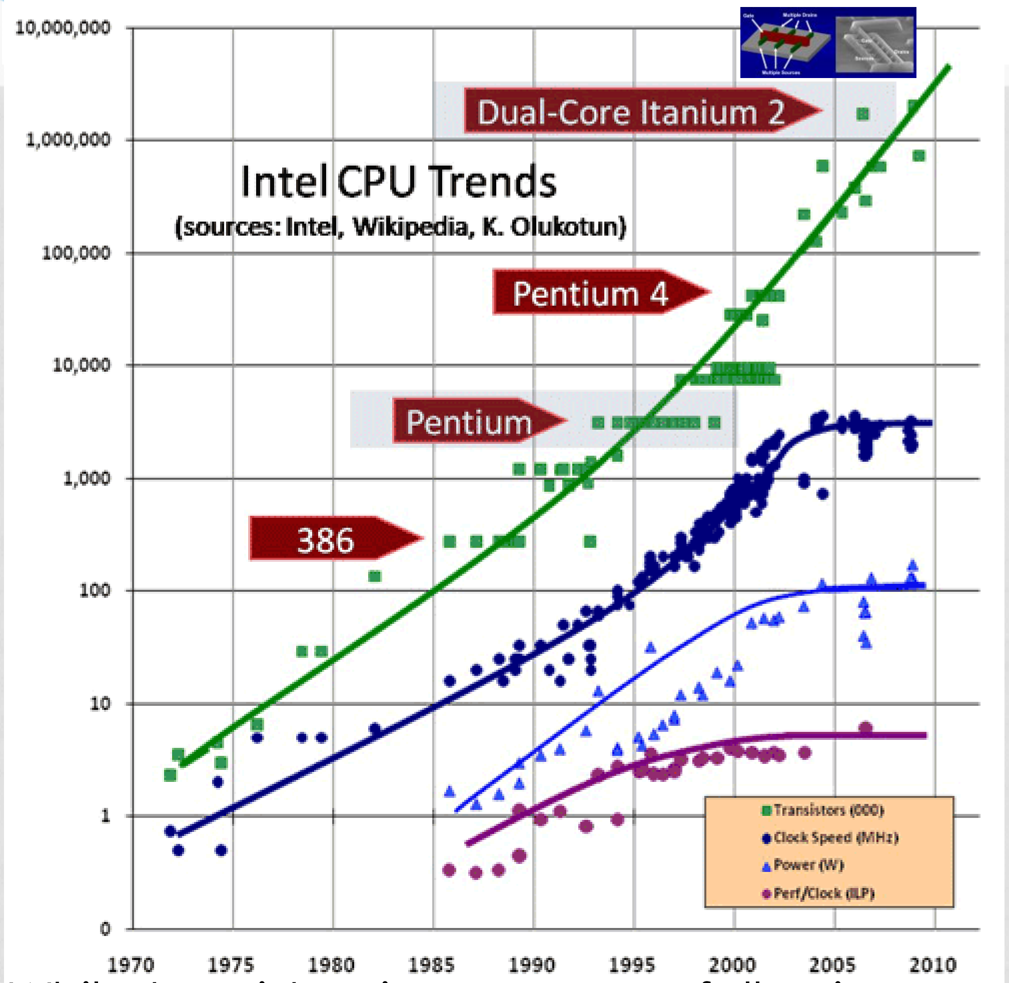


From a recent talk by Intel

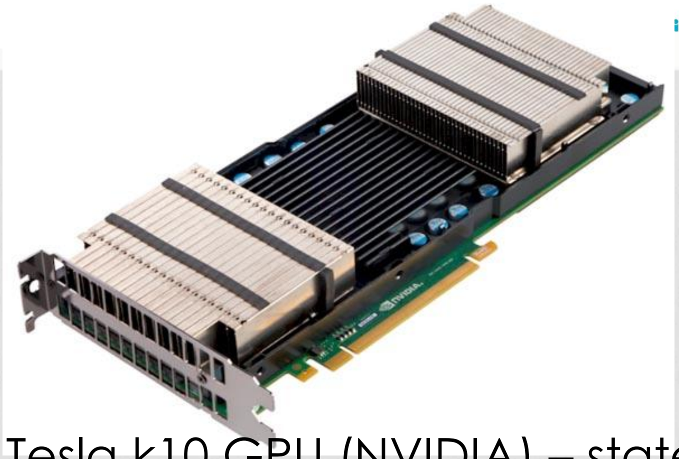
HEP applications are not doing great…

CPI:	0.9361
load and store instructions %:	49.152%
resource stalls % (of cycles):	43.779%
branch instructions % (approx):	30.183%
computational FP instr. %:	0.026%

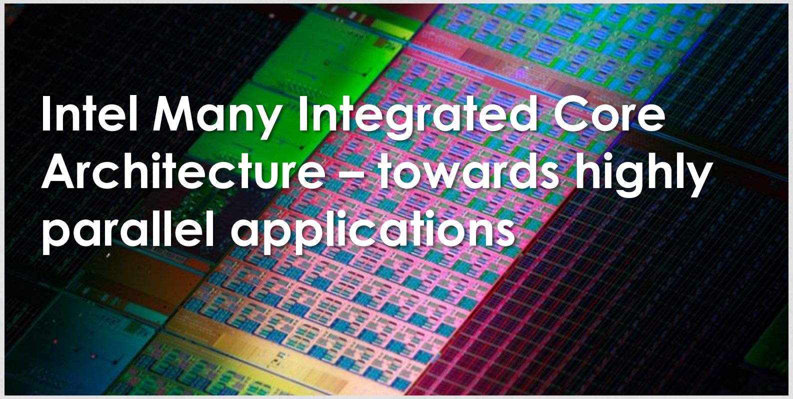
TRENDS…



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



Intel Many Integrated Core Architecture – towards highly parallel applications

EMBARRASSING PARALLELISM ?

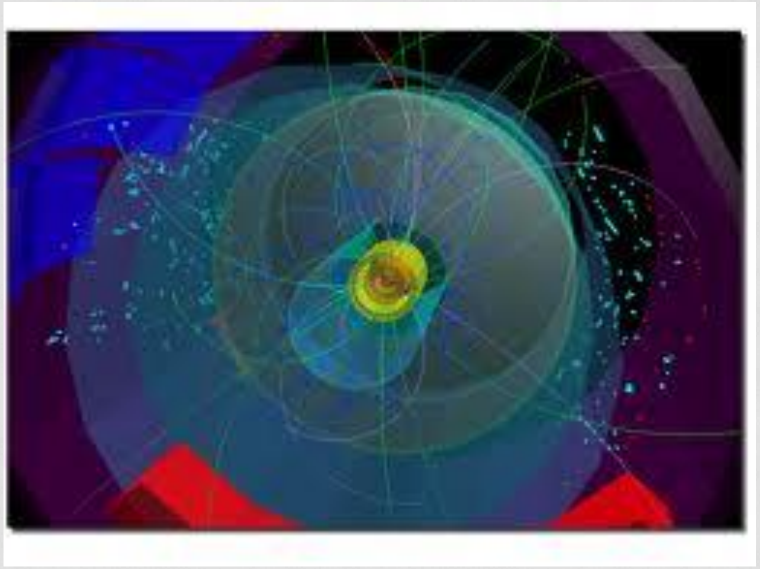
- Data parallelism on GRID was a savior for HEP, but…
  - Resources get short when one needs to simulate x10 the size of LHC data and uses just a tiny fraction of the CPU power
  - Fast Monte Carlo is a getaway, but cannot help in many performance studies
- Event and track level parallelism: share the code and most of the RO data structures
  - Already a step forward, but does not make jobs more efficient…
- There is an additional need to merge the outputs
  - The process may take longer across different machines than the simulation itself

RE-THINKING THE SIMULATION APPLICATION

- Vectorizing CPU-critical parts
  - Pushing common computation outside loops and optimizing CPU-intensive loops
  - Using virtualization at high levels but keeping simple structures for low level loops
  - Using efficiently vectors to minimize the scatter/gather overhead
- Parallell/O
  - Multiple buffers to output stream -> new ROOT developments
  - Asynchronous I/O using dynamic scheduling that balances the use of resources
- Using better the memory and minimizing cache misses
  - Data and code locality: transporting a single particle uses now most of the code and data
  - Needs reviewing the code and algorithms: towards a local particle transport
- Using transparently extra resources when available (GPU kernels)
  - Evolution of C++ and code instrumentation languages
- Having in mind all stages of the simulation chain from a parallel perspective
  - Including digitization and I/O
  - Avoiding synchronization issues and aiming for automatic load balancing

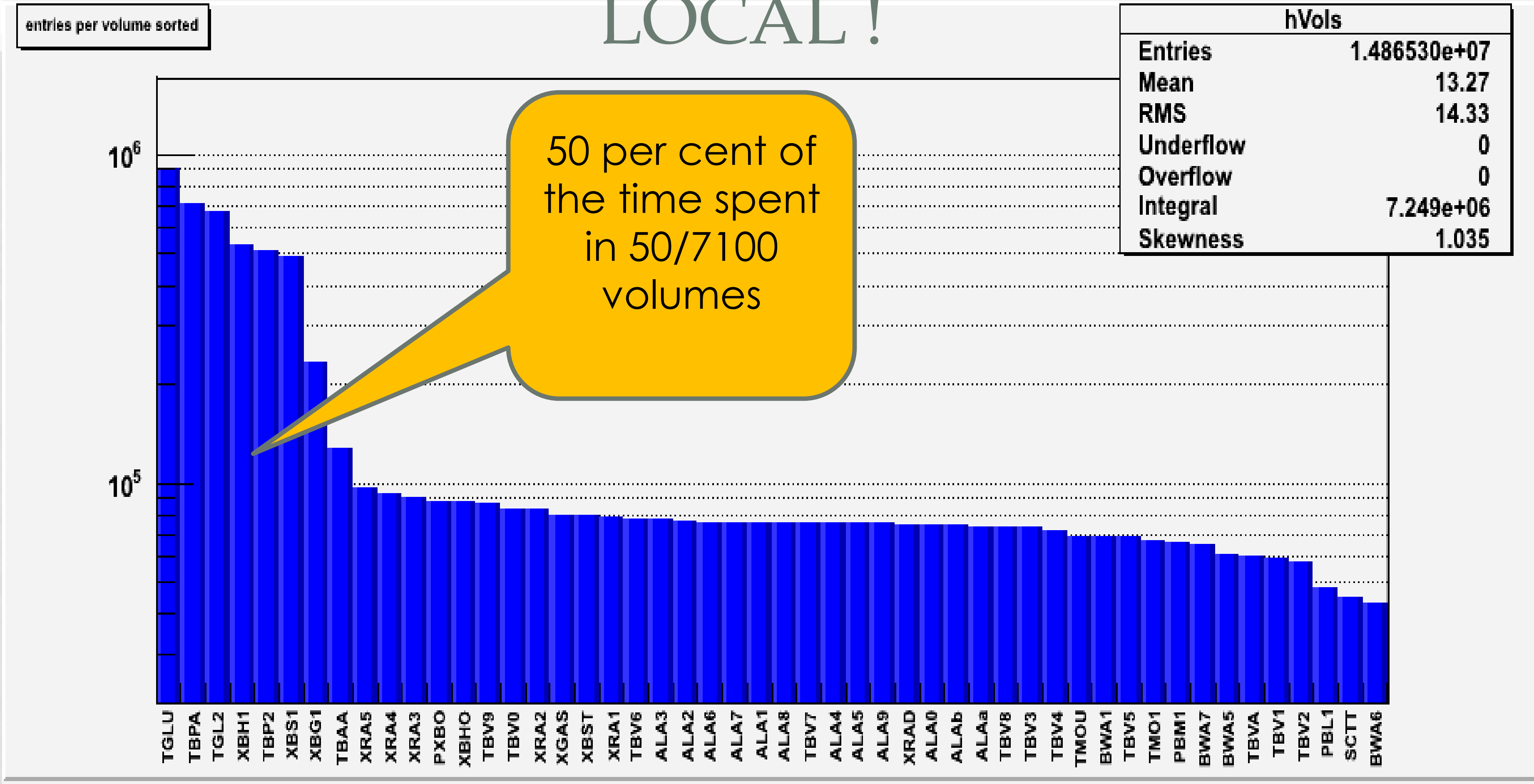
A deep re-organization of particle transport system and a complete re-design of the steering mechanisms and data structures are needed

THE CLASSIC TRANSPORT APPROACH



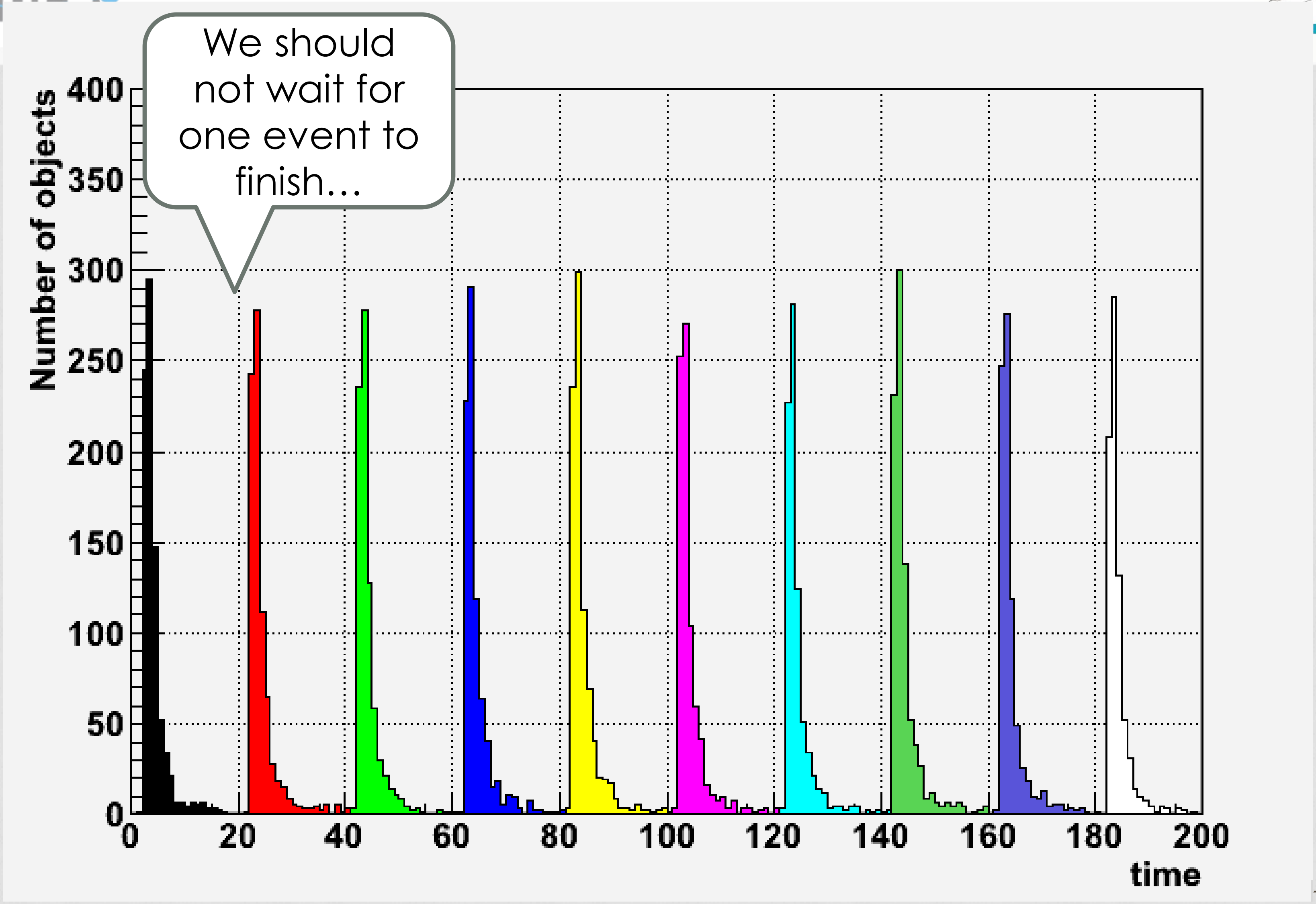
- There is a main event loop pushing primary tracks to the transport stack
- Track loop where particles are being extracted from the stack followed one by one to the exit of the detector, by simulating the physics processes along the way
- During each transport step the user code for hit generation is called
- At the end of each event, a digitization procedure is using the specific detector response to convert the hit structure into digits, which are typically dumped to file

GOOD NEWS: HEP TRANSPORT IS MOSTLY LOCAL !



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

BAD NEWS: TAILS… TAILS…



A PLAYGROUND FOR NEW IDEAS

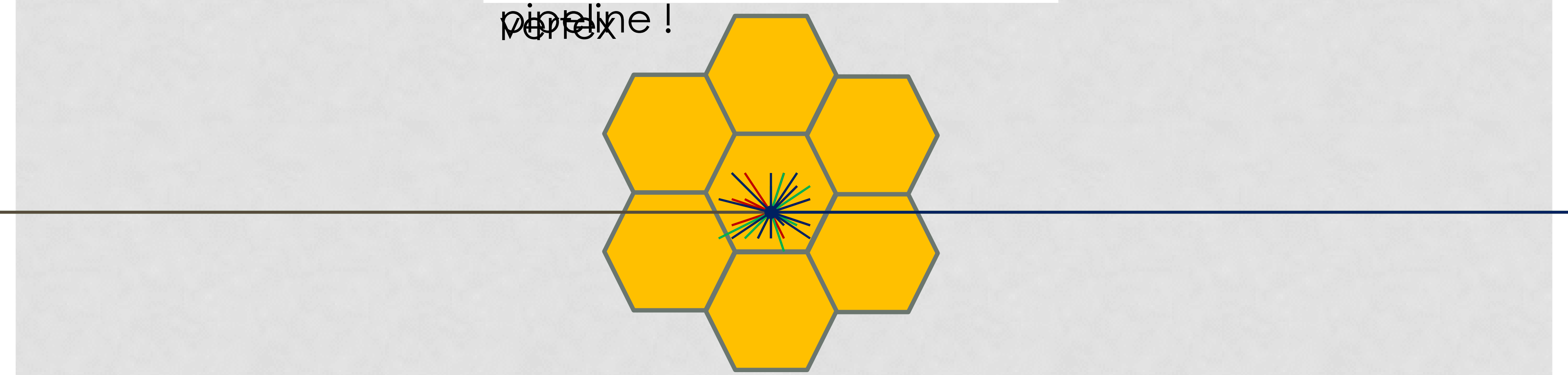
- We started with a simple prototype to help exploring some of these issues
  - Simple idea: minimal physics to start with, realistic HEP geometry: can we implement a parallel transport model exploiting locality ?
  - Base work unit: a vector of particles
- Events and tracks are independent
  - Mixing tracks from different events to avoid tails and have reasonably-sized vectors
  - Study how does scattering/gathering impact the simulation data flow ?
  - Can we achieve a good load balancing ?
- Toy physics at first, more realistic EM processes to be integrated soon
  - The application should be tuned based on realistic numbers
- New transport model more "detector element"-oriented, profiting from the cached data structures
  - geometry and x-section wise
- Re-design the particle stack and the I/O
- Re-design transport models from a "plug-in" perspective
  - E.g. ability to use fast simulation on per track basis

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
- Particles exiting a volume are distributed to baskets of the neighbor volumes until exiting the setup or disappearing
  - Like a champaign cascade, but lower glasses can also fill top ones…
  - No direct communication between threads to avoid synchronization issues

THE TRANSPORT PROTOTYPE

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

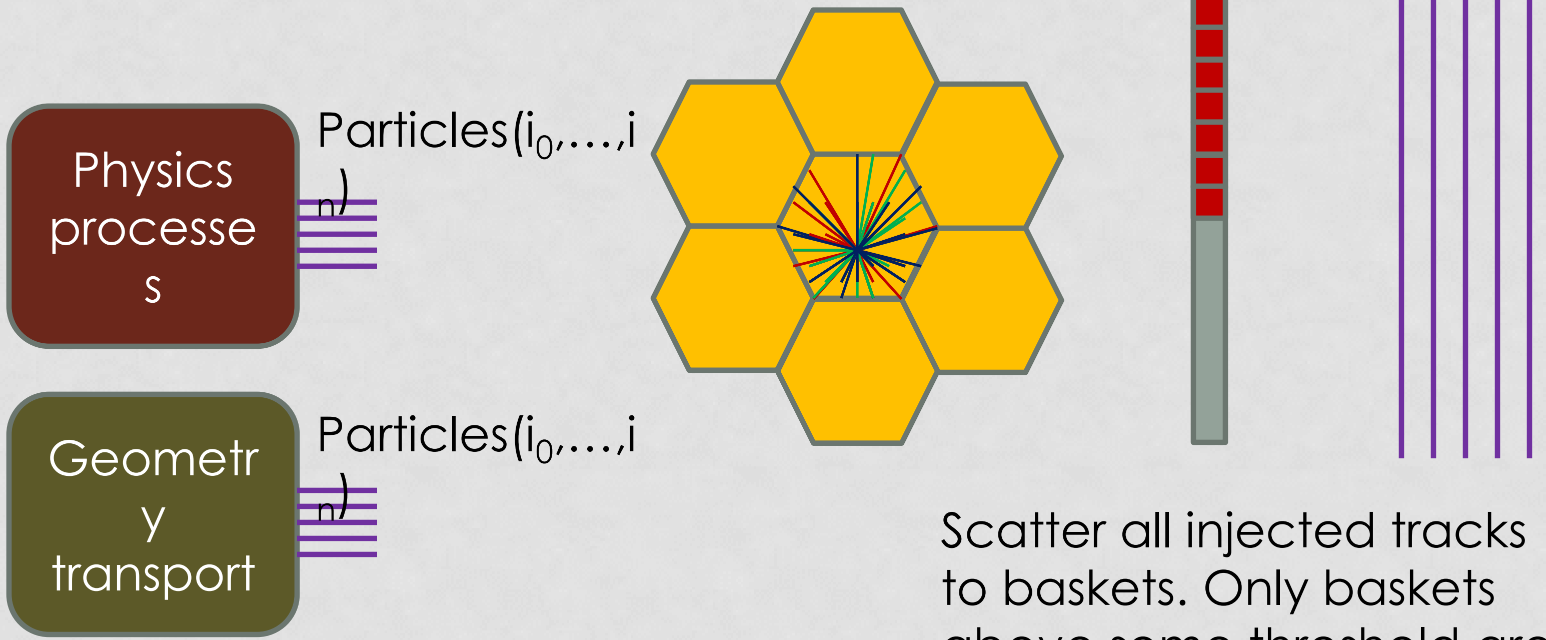


Associate a set of "baskets" to each geometrical logical volume

A FIRST APPROACH

Each thread transports its basket of tracks to the boundaries of the current volume
Move crossing tracks to a buffer, then picks-up the next basket from the queue

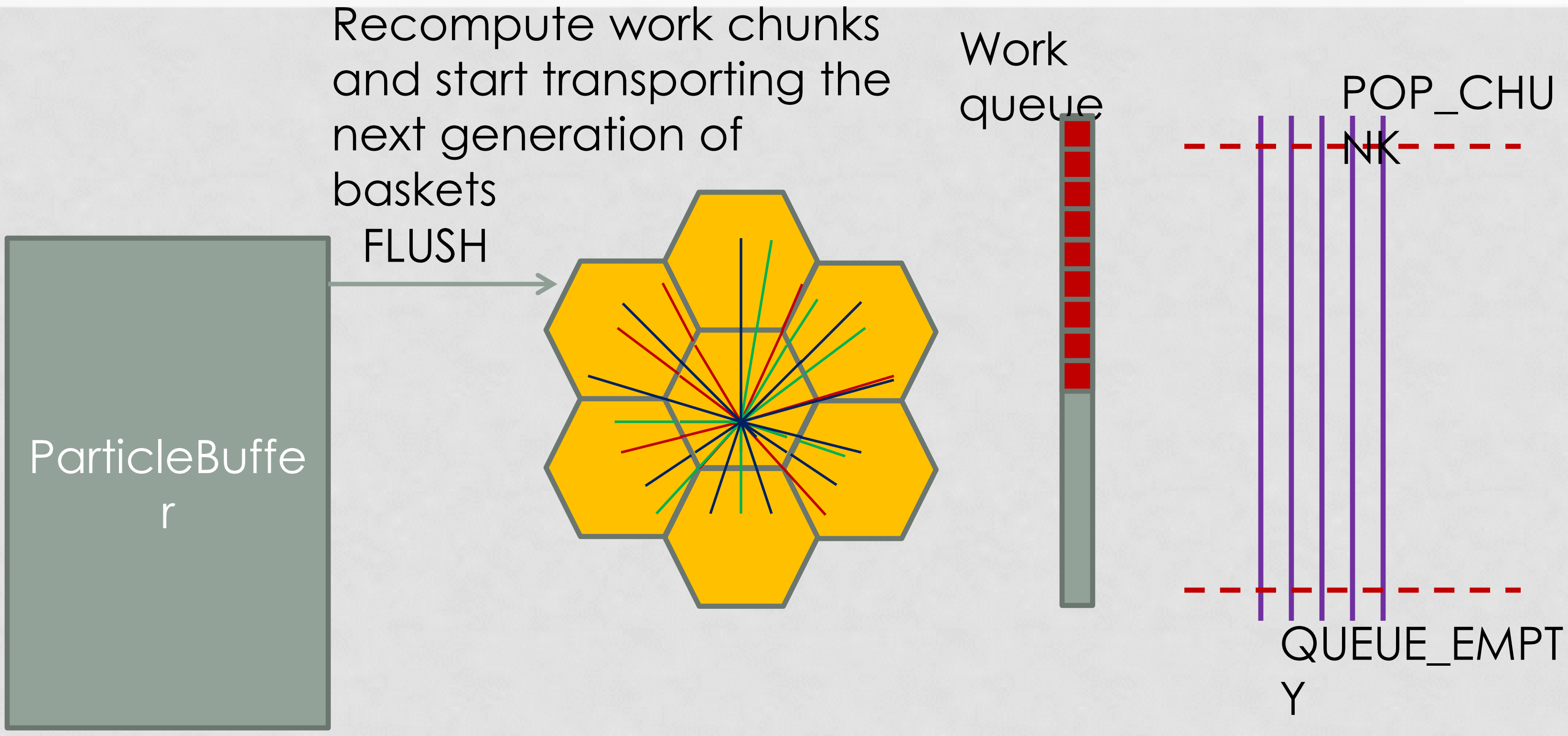
Transport threads pick-up baskets from the work queue



Scatter all injected tracks to baskets. Only baskets above some threshold are transported.

Physics processes and geometry transport called with vectors of particles

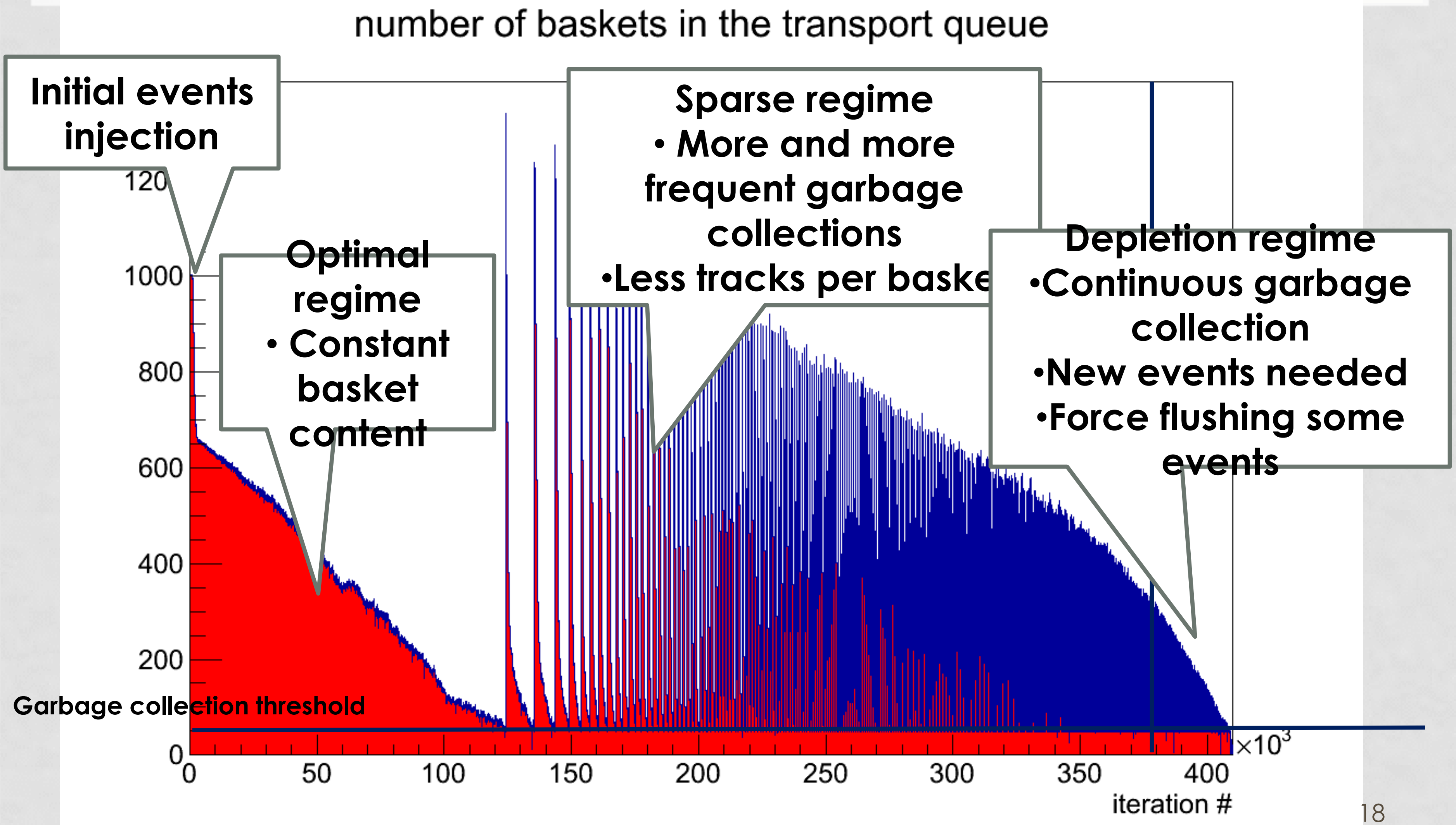
FIRST VERSION REQUIRED SYNCHRONIZATION…


Recompute work chunks and start transporting the next generation of baskets
FLUSH
ParticleBuffer
Work queue
POP_CHUNK
QUEUE_EMPTY

Synchronization point: flush transported particle buffer and sort baskets according content

Generation = Pop work chunks until the queue is empty

BASKET POPULATION EVOLUTION



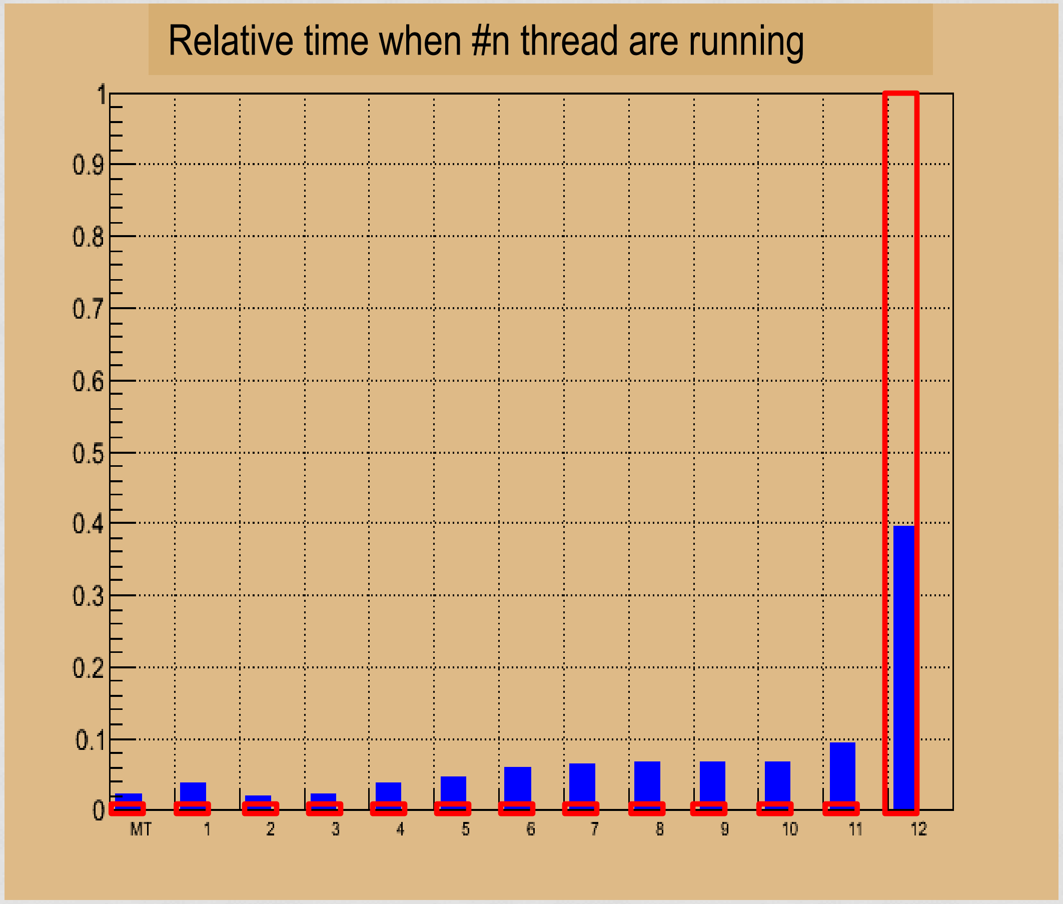
LESSONS LEARNED

- A model requiring synchronization stages is not appropriate
  - Very large Amdahl effects, increasing in the track depletion stage
- Balancing basket populations in time is not trivial
  - Events need to be injected in the system to compensate the basket inefficiencies
    - Will cumulate hits and make memory grow
  - Hits from early introduced events need to be evacuated to the digitization and I/O threads
- We need a good estimate of the percentage of work that can be done with "efficient" vectors
  - A model including realistic physics, digitization and I/O will be needed

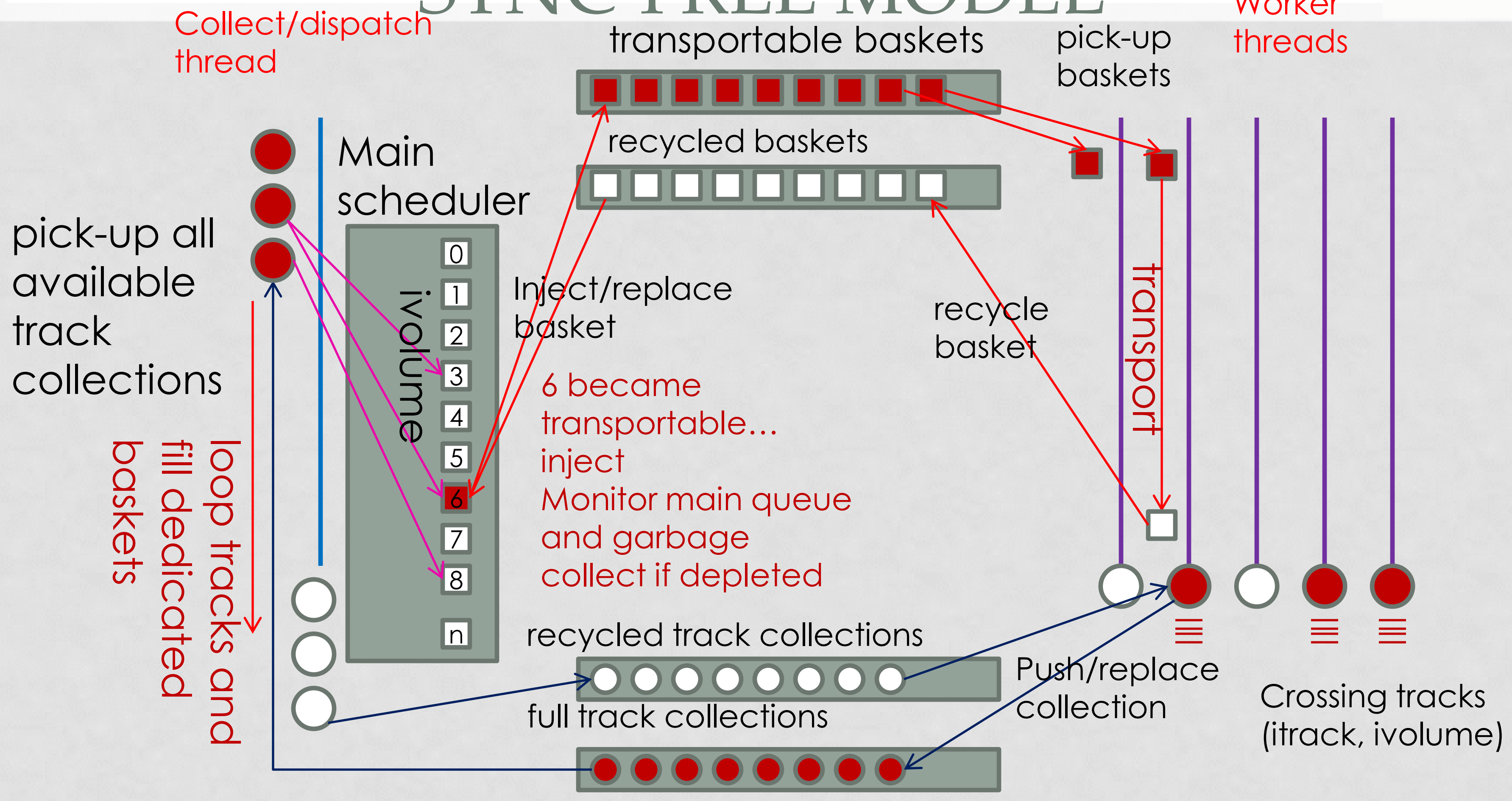
CONCURRENCY IN THE FIRST APPROACH

- Ideally all workers should be in running state during the processing phase, so the distribution should peak for the number of workers

- Synchronization becomes critical during the particle depletion regime, when particle baskets are non-optimally filled and garbage collections more often

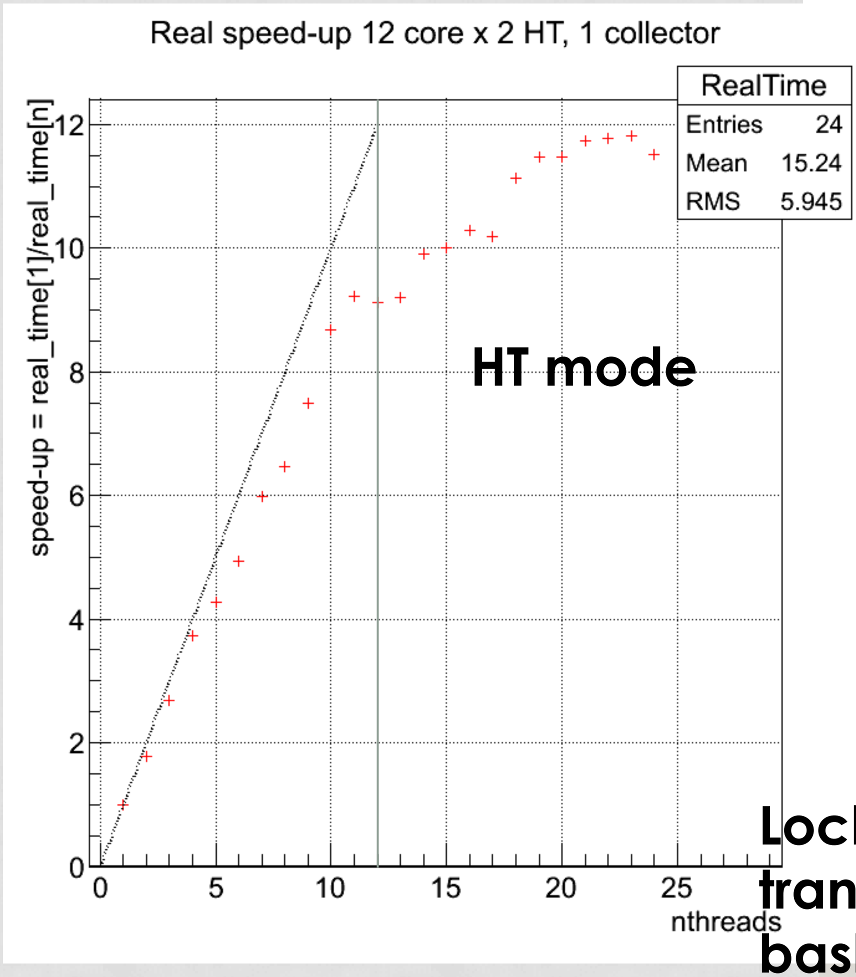


EVOLUTION OF THE PROTOTYPE TO A SYNC-FREE MODEL

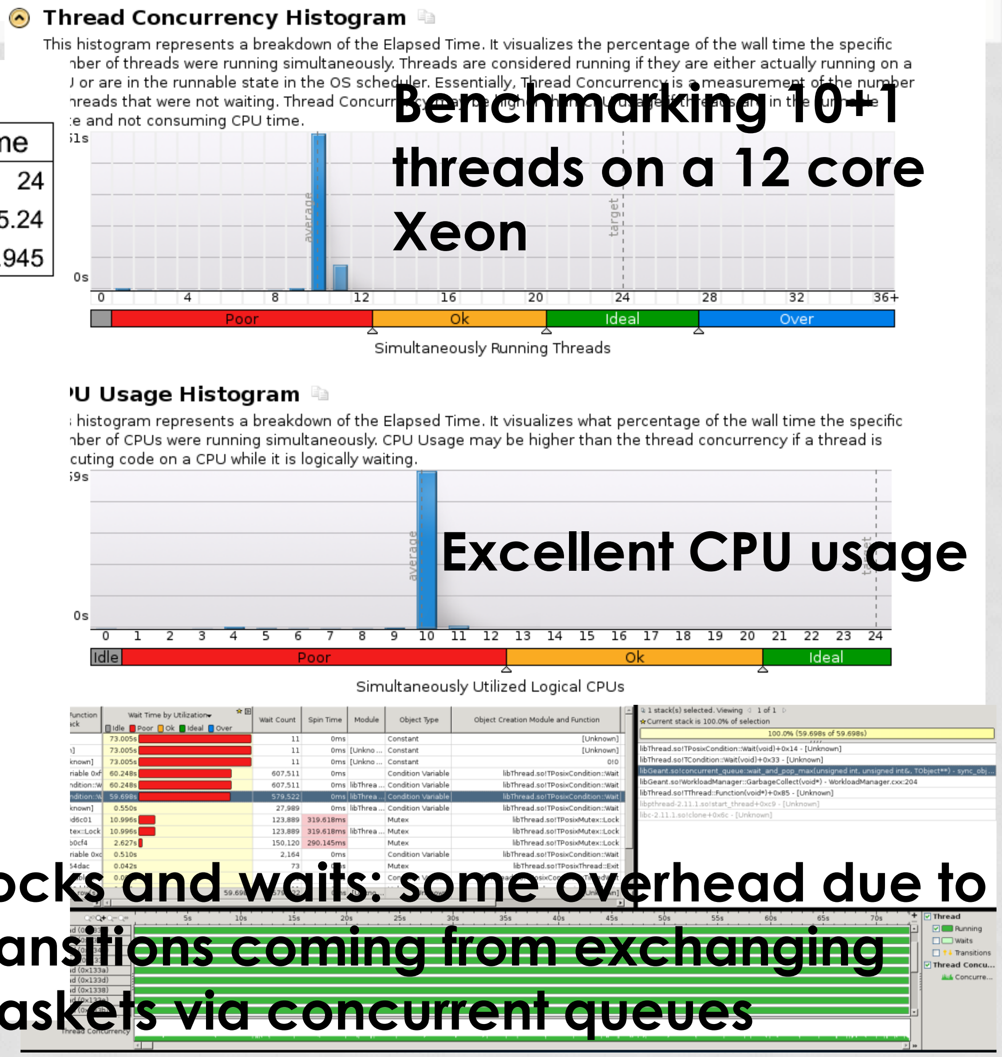


Collect/dispatch thread
transportable baskets
pick-up baskets
Worker threads
Main scheduler
recycled baskets
pick-up all available track collections
ivolume
0
1
2
3
4
5
6
7
8
n
Inject/replace basket
recycle basket
transport
6 became transportable… inject
Monitor main queue and garbage collect if depleted
loop tracks and fill dedicated baskets
recycled track collections
full track collections
Push/replace collection
Crossing tracks (itrack, ivolume)

PRELIMINARY BENCHMARKS



HT mode



Thread Concurrency Histogram

This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific
mber of threads were running simultaneously. Threads are considered running if they are either actually running on a
J or are in the runnable state in the OS scheduler. Essentially, Thread Concurrency is a measurement of the number
hreads that were not waiting. Thread Concurr
e and not consuming CPU time.

U Usage Histogram

histogram represents a breakdown of the Elapsed Time. It visualizes what percentage of the wall time the specific
mber of CPUs were running simultaneously. CPU Usage may be higher than the thread concurrency if a thread is
cuting code on a CPU while it is logically waiting.

Function ack	Wait Time by Utilization (🟫 Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over)	Wait Count	Spin Time	Module	Object Type	Object Creation Module and Function
	73.005s	11	0ms		Constant	[Unknown]
n]	73.005s	11	0ms	[Unkno ...	Constant	[Unknown]
known]	73.005s	11	0ms	[Unkno ...	Constant	0!0
riable 0xf	60.248s	607,511	0ms		Condition Variable	libThread.so!TPosixCondition::Wait
ndition::W	60.248s	607,511	0ms	libThrea ...	Condition Variable	libThread.so!TPosixCondition::Wait
ndition::W	59.698s	579,522	0ms	libThrea ...	Condition Variable	libThread.so!TPosixCondition::Wait
known]	0.550s	27,989	0ms	libThrea ...	Condition Variable	libThread.so!TPosixCondition::Wait
d6c01	10.996s	123,889	319.618ms		Mutex	libThread.so!TPosixMutex::Lock
tex::Lock	10.996s	123,889	319.618ms	libThrea ...	Mutex	libThread.so!TPosixMutex::Lock
b0cf4	2.627s	150,120	290.145ms		Mutex	libThread.so!TPosixMutex::Lock
riable 0xc	0.510s	2,164	0ms		Condition Variable	libThread.so!TPosixCondition::Wait
54dac	0.042s	73	[illegible]		Mutex	libThread.so!TPosixThread::Exit

1 stack(s) selected. Viewing 1 of 1
Current stack is 100.0% of selection
100.0% (59.698s of 59.698s)

```
libThread.so!TPosixCondition::Wait(void)+0x14 - [Unknown]
libThread.so!TCondition::Wait(void)+0x33 - [Unknown]
libGeant.so!concurrent_queue::wait_and_pop_max(unsigned int, unsigned int&, TObject**) - sync_obj...
libGeant.so!WorkloadManager::GarbageCollect(void*) - WorkloadManager.cxx:204
libThread.so!TThread::Function(void*)+0x85 - [Unknown]
libpthread-2.11.1.so!start_thread+0xc9 - [Unknown]
libc-2.11.1.so!clone+0x6c - [Unknown]
```

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

A CONSIDERABLE EFFORT TO MOVE FORWARD…

- HEP simulation cannot afford ignoring the technology trends
  - LHC computing needs largely demonstrate it
  - The software inefficiency becomes more and more expensive
- The move towards efficiency is long and tedious
  - Even if all the components (physics, geometry, …) are written
  - We started to look under the hood, and tried out few ideas, but most of the work is still to be done
- We need to constantly monitor the progress
  - GEANT 4 MT will be a good reference system once the physics will be comparable
  - Even between different versions of the prototype

FUTURE PLANS

- Continue the investigation of parallel event transport
- Develop more realistic physics models
- Integrate fast simulation
- Aiming at a working prototype in 2015
- The activity will start in earnest in September 2012
- As with other large successful projects (ROOT, G4), this will be an international collaboration
  - We count on the HEP community (LHC, ILC, CLIC, FAIR, ...) for help and feedback

CONCLUSIONS

- The new generation (Geant5) of detector simulation programs will have to
  - Integrated seamlessly fast and detailed simulation at different levels
  - Make efficient use of parallelism at different levels
  - Capitalizing on the large Geant 1-4 experience
- A prototype is being built at CERN, which will require collaboration with the HEP Community at large
- The first results are interesting and our learning curve very steep (!)
- Stay tuned…