# Software for HL-LHC

**David Rousseau, LAL-Orsay,**
**for the TDOC preparatory group**
**3 Oct 2013**

# TDOC Membership

- ALICE: Pierre Vande Vyvre, Thorsten Kollegger, Predrag Buncic
- ATLAS: David Rousseau, Benedetto Gorini, Nikos Konstantinidis
- CMS: Wesley Smith, Christoph Schwick, Ian Fisk, Peter Elmer
- LHCb: Renaud Legac, Niko Neufeld

# More credits

- Also to be credited for material to this talk specifically: Sebastien Binet, Ian Bird, Federico Carminati, Markus Elsing, Benedict Hegner, Thorsten Kollegger, Rocco Mandrysch, Bernd Panzer-Steindel, Pere Mato, Elmar Ritsch, Andreas Salzburger, Elizabeth Sexton-Kennedy, Rolf Seuster, Nick Styles, I Ueda

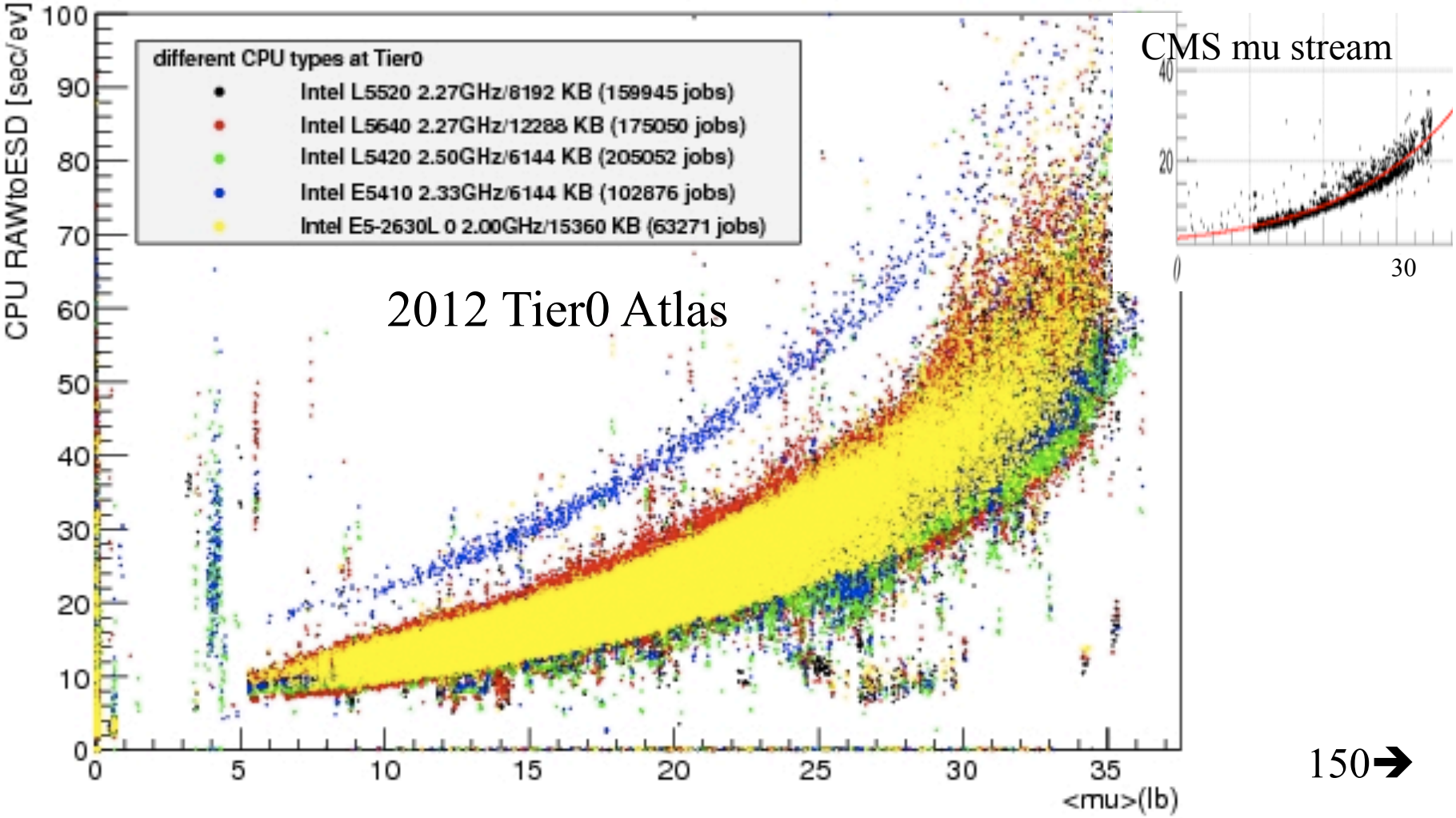
- Mistakes, inaccuracies, oversimplifications are mine…

# LHC Context

- HL-LHC events:
  - more complex (pile-up ~150 rather than ~25 in run 1, also energy x ~2)
    - No reliable estimate today of the impact on CPU, as existing code shows non linear divergence. Indicatively, multiplicity increases by a factor 8.
  - higher read-out rates (factor ~10)
- Flat resources (in euros) and Moore's law give us a factor 10 in CPU power (if and only if we can use the processors as efficiently as today!)
- ➔handling HL-LHC event added complexity, and maintenance/improvement of processor efficiency rely on software improvements. If not, impact on physics.
- Run 2 already has some of these issues in a milder way

# Impact of pileup on reco

Reconstruction CPU time at Tier0 vs pileup (Jet stream)

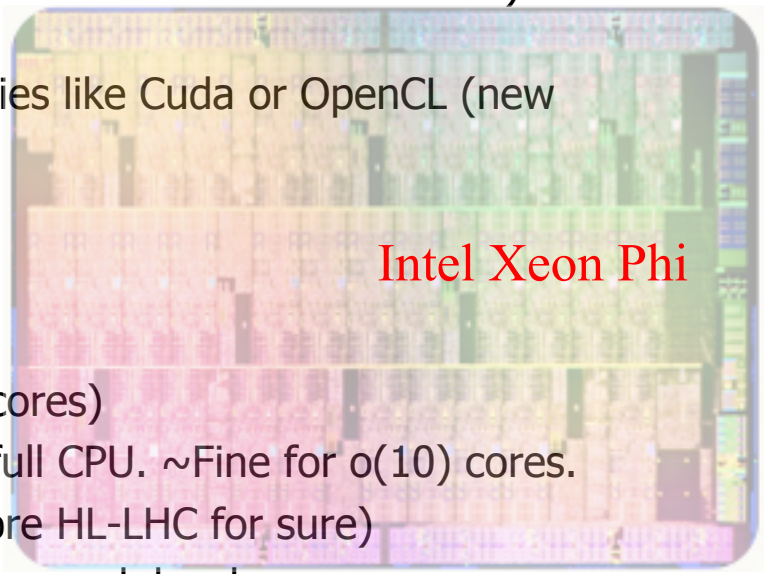



150➔

# Major crisis ?

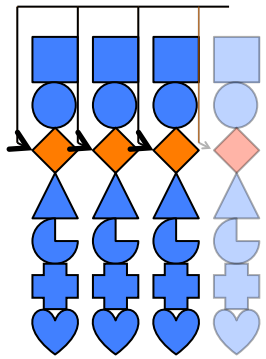
# CPU Context

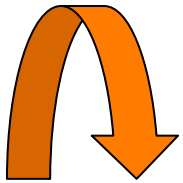
- Remember: no more CPU frequency gain since ~2005
- Two orthogonal avenues (in addition to traditional algorithm improvement):
- Micro-parallelism
  - Modern cores are not used efficiently by HEP software. Many cache misses. SIMD (Single Instruction Multiple Data), ILP (Instruction Level Parallelism) etc… to be used
  - Specialised cores like GPU require use of libraries like Cuda or OpenCL (new languages effectively)
  - Expert task. Focus on hot spots.
  - Immediate benefit on performance
- Macro-parallelism
  - More cores per CPU (and possibly specialised cores)
  - So far largely ignored : treat one core as one full CPU. ~Fine for o(10) cores.
  - Will break down for o(100) cores (when ? before HL-LHC for sure)
  - ➔calling for macro-parallelism, handled at framework level
  - Mitigates impact on sw developers if framework is smart enough
  - However does not help throughput if enough I/O and memory
  - Should be on-going already now given the large code/people base involved



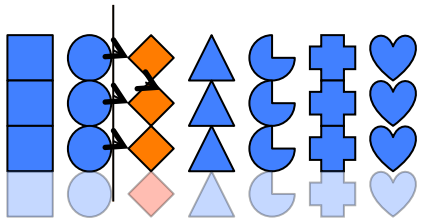
# Note on data organisation

Array of struct





Struct of arrays



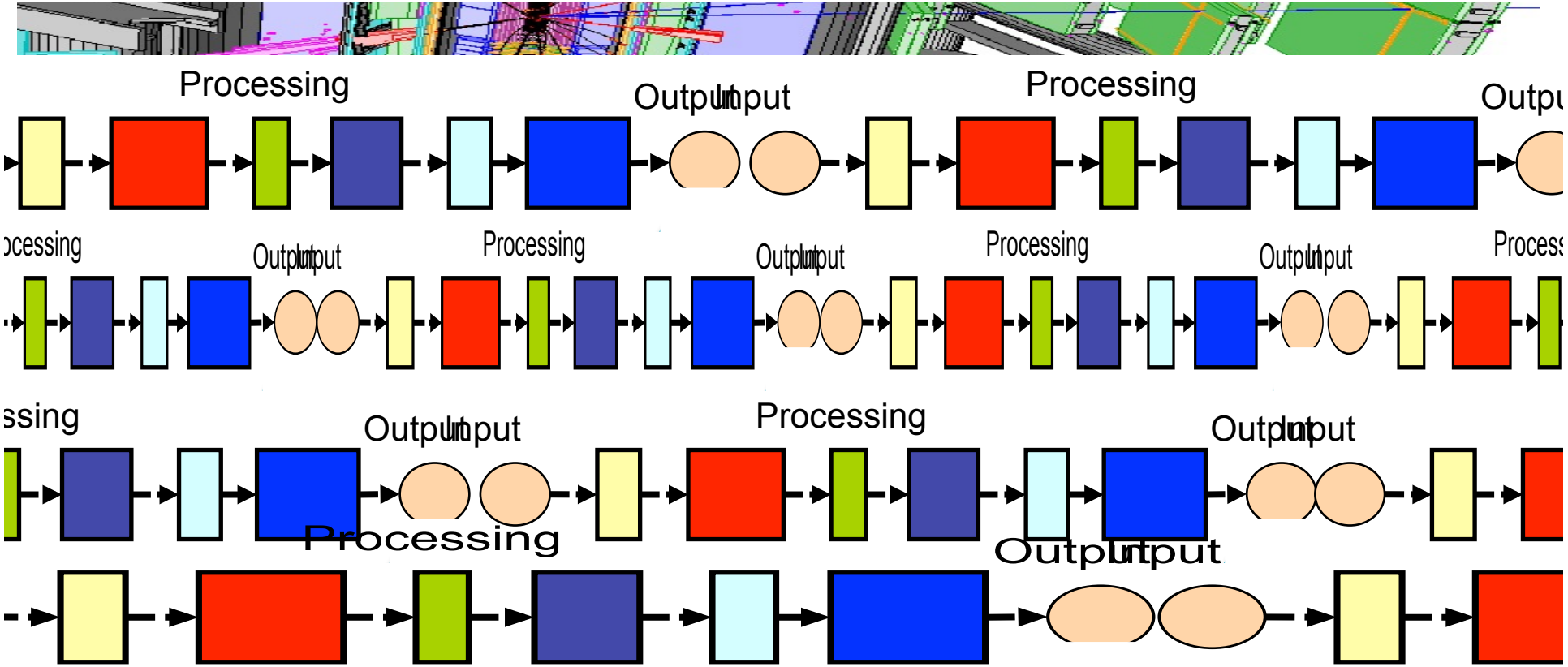
➔ More suitable for vectorisation

- Data organisation often need to be completely revisited prior to algorithm vectorisation
- (may improve performance even without vectorisation due to better locality (less cache misses))

# LHC experiments code base

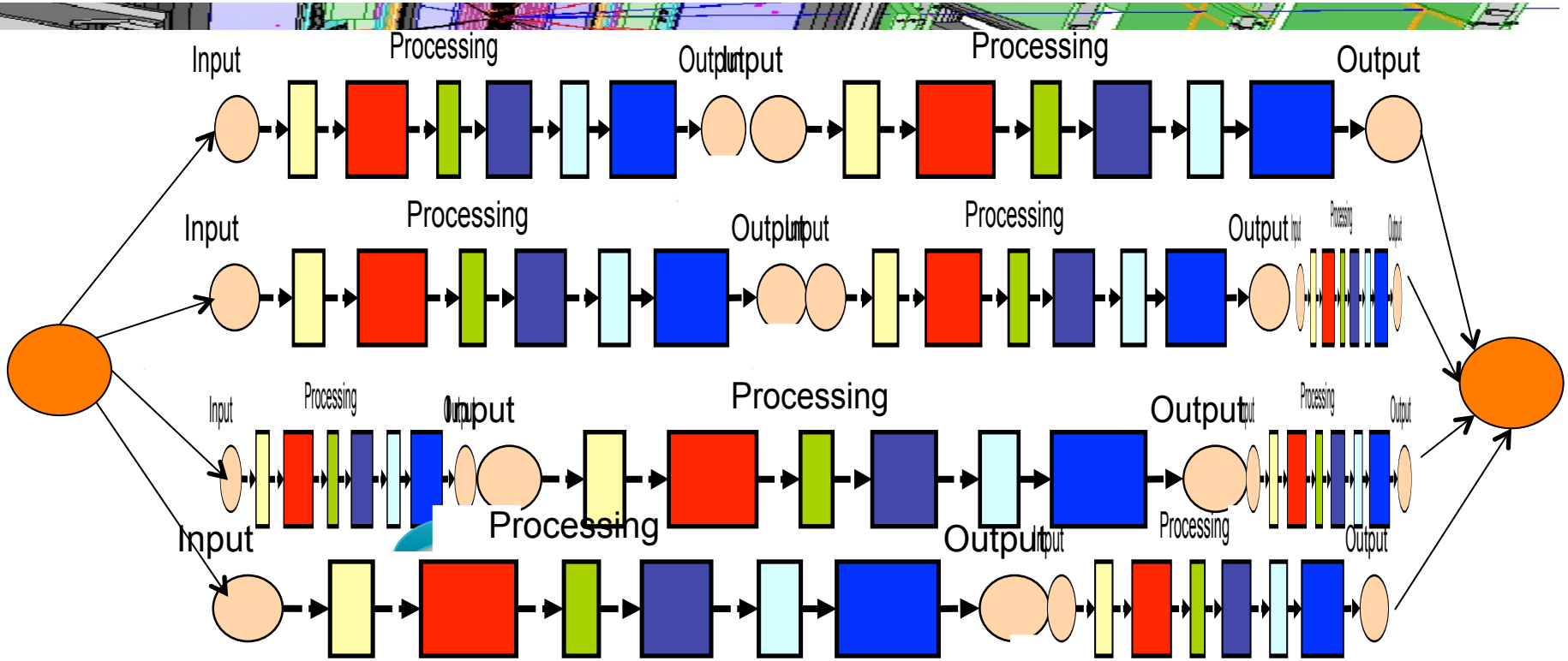
- LHC experiments code base
  - ~5 millions line of code per experiment
  - written by ~ 1000 people per experiment since ~15 years
  - up to 300 people active per experiment (but dropping in 2011 2012 2013) (new blood needed)
- Who are they ?
  - Very few software engineers
  - Few physicists with very strong software expertise
  - Many physicists with ad-hoc software experience
- All these people need take part to the new transition

# One core one job



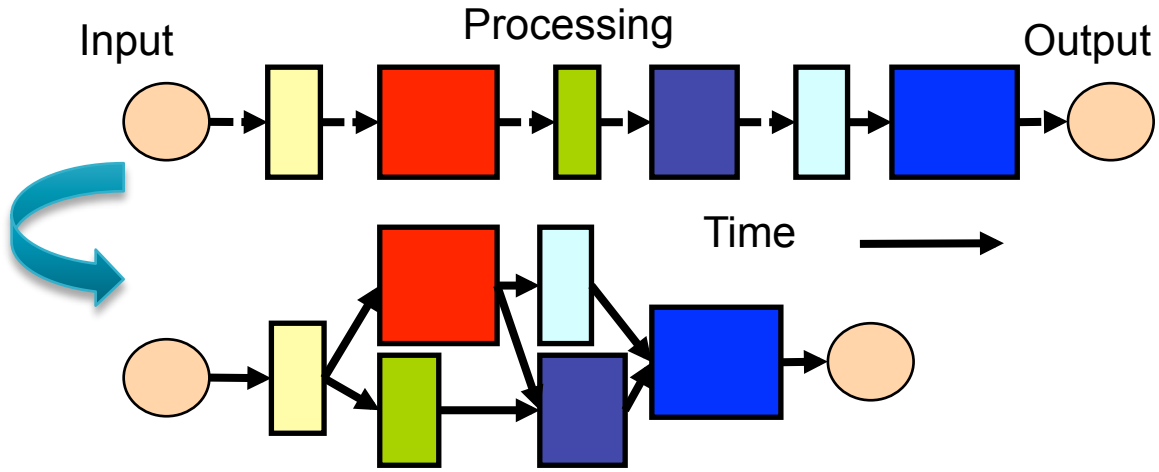

- Today, typical grid workhorse is a 16GB memory, 8 core CPU (2GB/core)
- Each core is adressed by the batch system as a separate processor
- Each job process event one by one, running one by one a finite number of algorithms
- One processor may handle simultaneously e.g. one Atlas reco job, 3 CMS simulation job, and 4 LHCb analysis jobs
- This works (today), however disorganised competition for resources like memory, I/O (Already memory issues for Atlas Run 2 reconstruction, requires 3GB memory per core.)

# One processor one job




- Available today (GaudiMP, AthenaMP) but not used in production yet
- One job goes to one processor (which is completely free)
- The framework distributes event processing to all cores, while sharing common memory (code, conditions,…) using Copy-on-Write
- No change to algorithmic code required (in principle)
- ˜50% reduction of memory achieved (w.r.t. independent jobs)

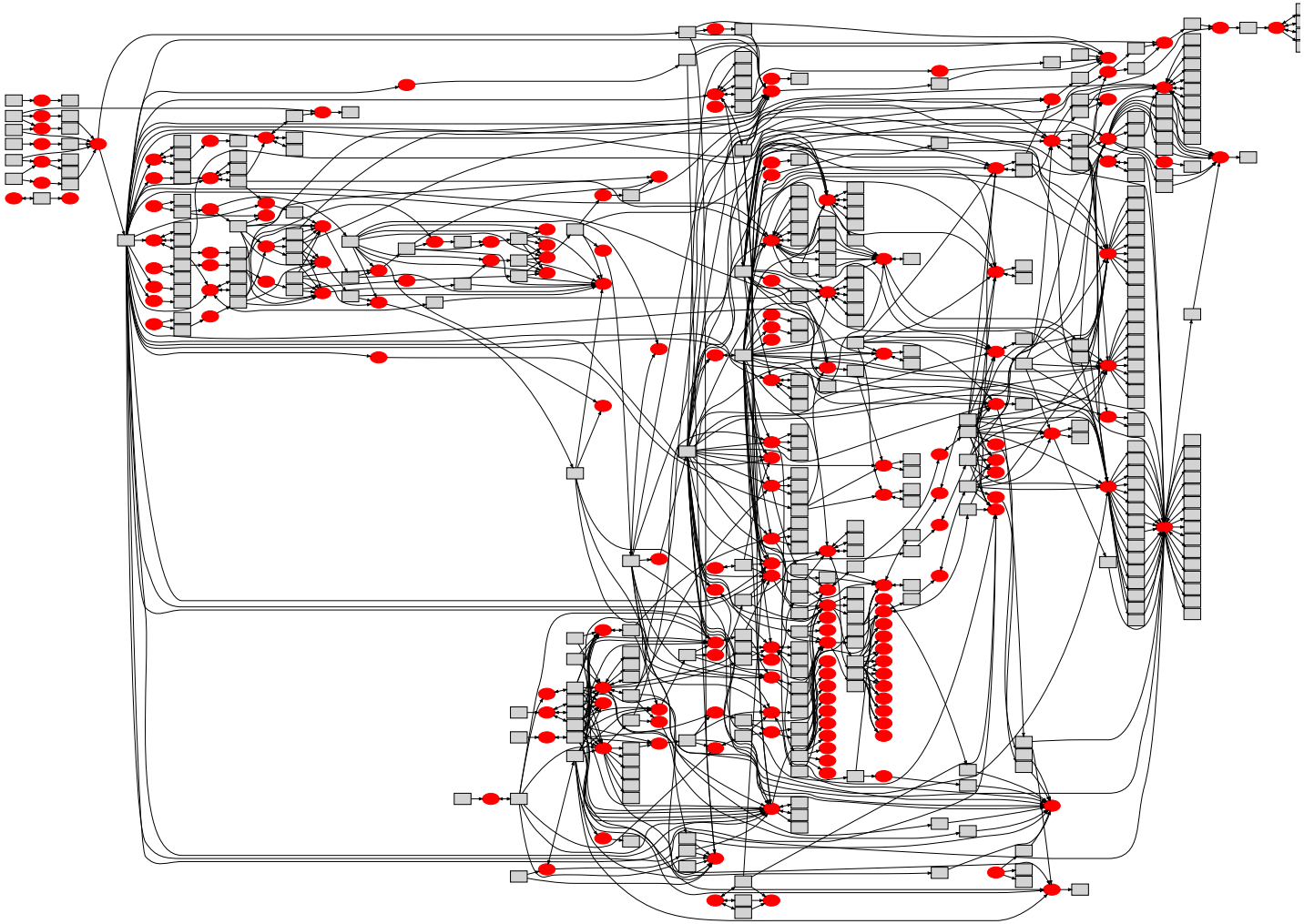
# Event level parallelism




- framework schedules intelligently the algorithms from their dependency graph
- e.g. run tracking in parallel with calorimeter, then electron ID
- in practice too few algorithms can run in parallel
- ➔most cores remain idle

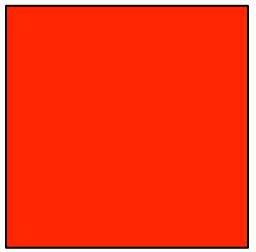
# Real life

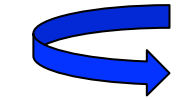
- Direct Acyclic Graph extracted from real reco job
- Today, algorithms run sequentially

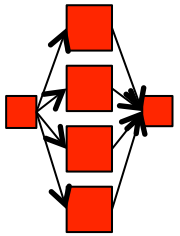


# Note on multi-threading

- One possible answer is to use multi-threading within algorithms
  - E.g. the tracking algorithm spawns multiple threads, each one reconstructing tracks in an eta-phi region
  - Test jobs on empty processor will effectively run (much) faster
  - However, in a grid environment, with one job per core, the multiple threads will compete with the other jobs running on the same processor➔no good!



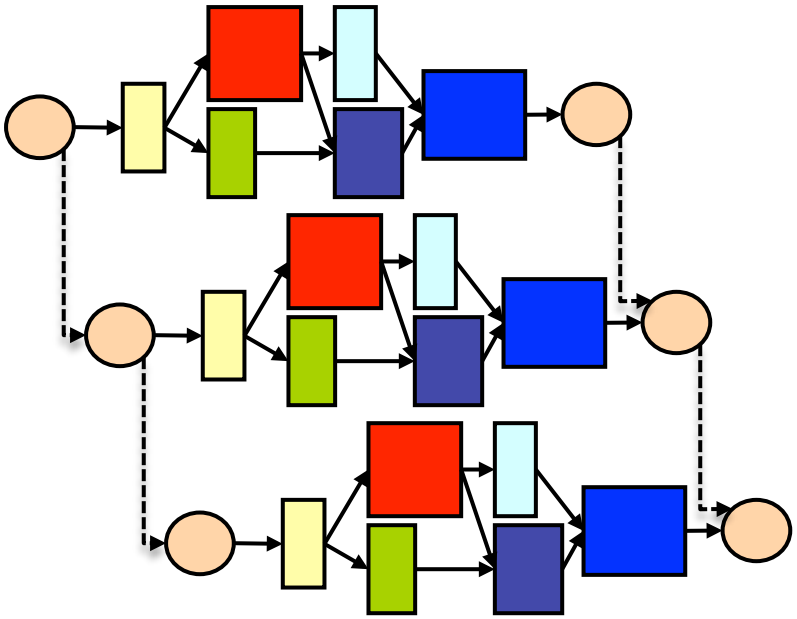




- Multi-threading useful if done at framework level, in an organised way (➔)

# Event level concurrent event processing

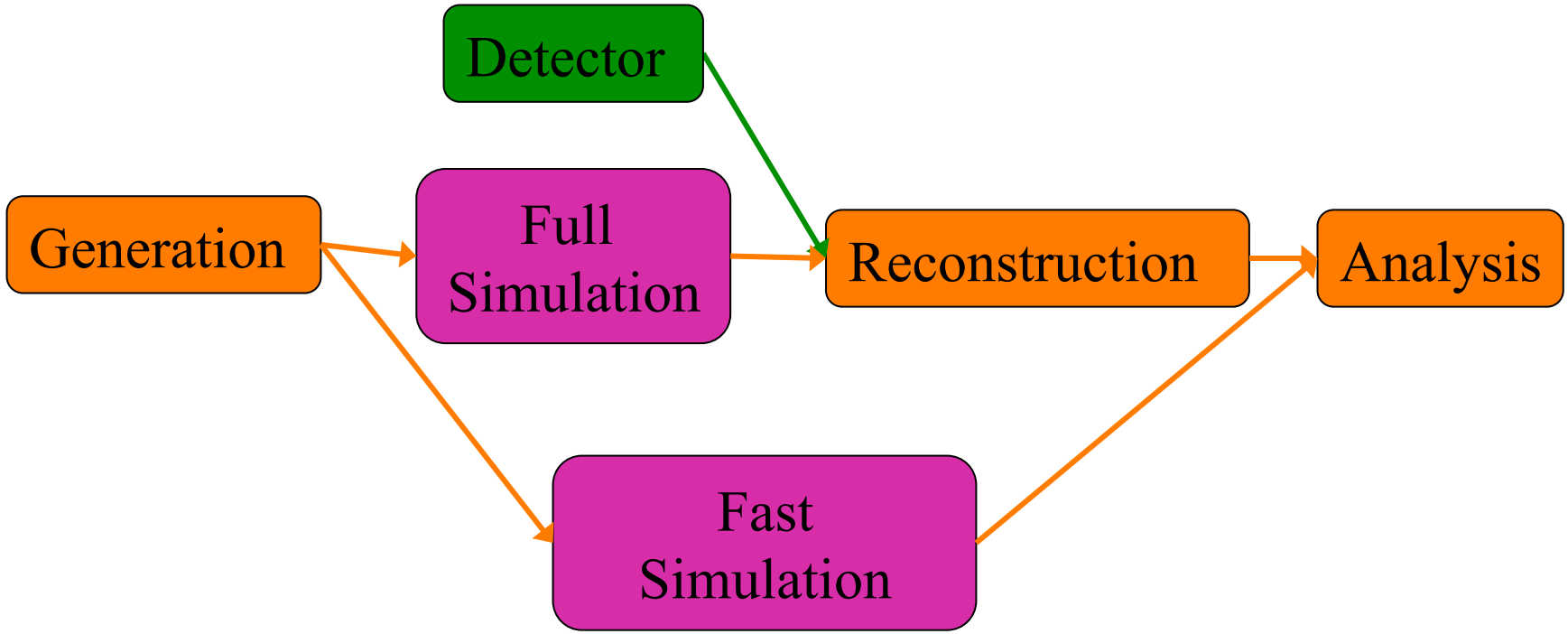
a.k.a the Holy Grail




- The framework processes several events simultaneously…
- …distributes intelligently algorithms to cores
- can allocate more cores to slowest algorithms
- can optimise use of specialised cores

- In addition to algorithm scheduling, the framework provides services to pipeline access to resources (I/O, conditions, message logging…)
- Algorithms should be thread safe : no global object (except through the framework), only use thread safe services and libraries
- Algorithms do not need to handle threads themselves
- ➔regular software physicist with proper training can (re)write algorithms

# Processing steps

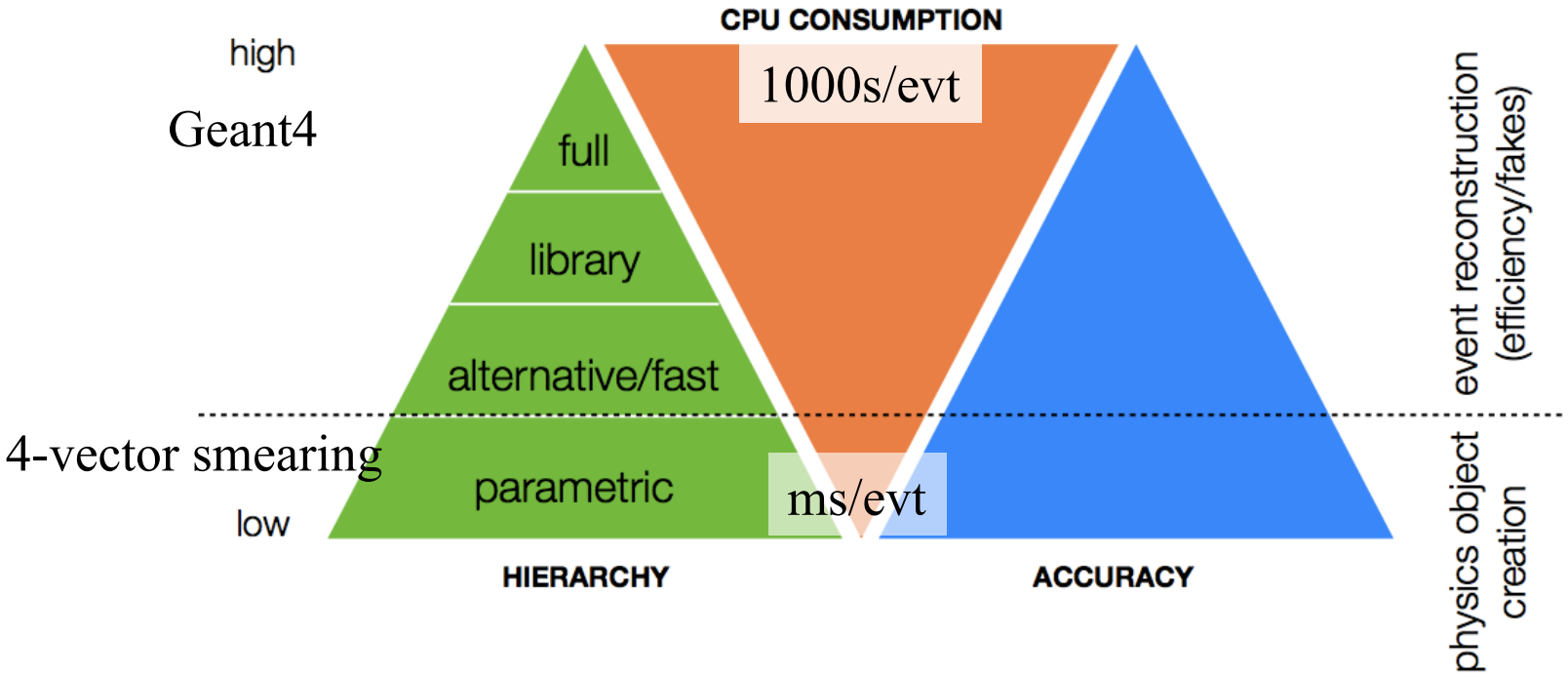


# Generators

- Each generator is a different beast, developed by a very small team (theorists usually)
- Some commonalities: HepData, HepMC event record
- Significant work in each experiment to interface each generator to the experiment framework
- Many generators N…NLO needed for many channels even if "just" for systematics (e.g.˜40 generators interfaced in Atlas)
- Overall CPU consumption not negligible (up to 10% of total grid)
- Little input, little output, no database connection: candidate for parasitic use of HPC

# Simulation

- Dominates CPU consumption on the grid
- HL-LHC : 10x read-out rate➔10x n events simulated ? Even more due to increased requirement on precision
- Continue effort on Geant4 optimisation:
  - G4 10.0 multi-threaded to be released Dec 2013
  - Re-thinking core algorithms with vectorisation in mind
- Rely on blend of G4/Fast sim/Parametric. Challenge : the optimal blend is very analysis dependent. But only one pot of resources.

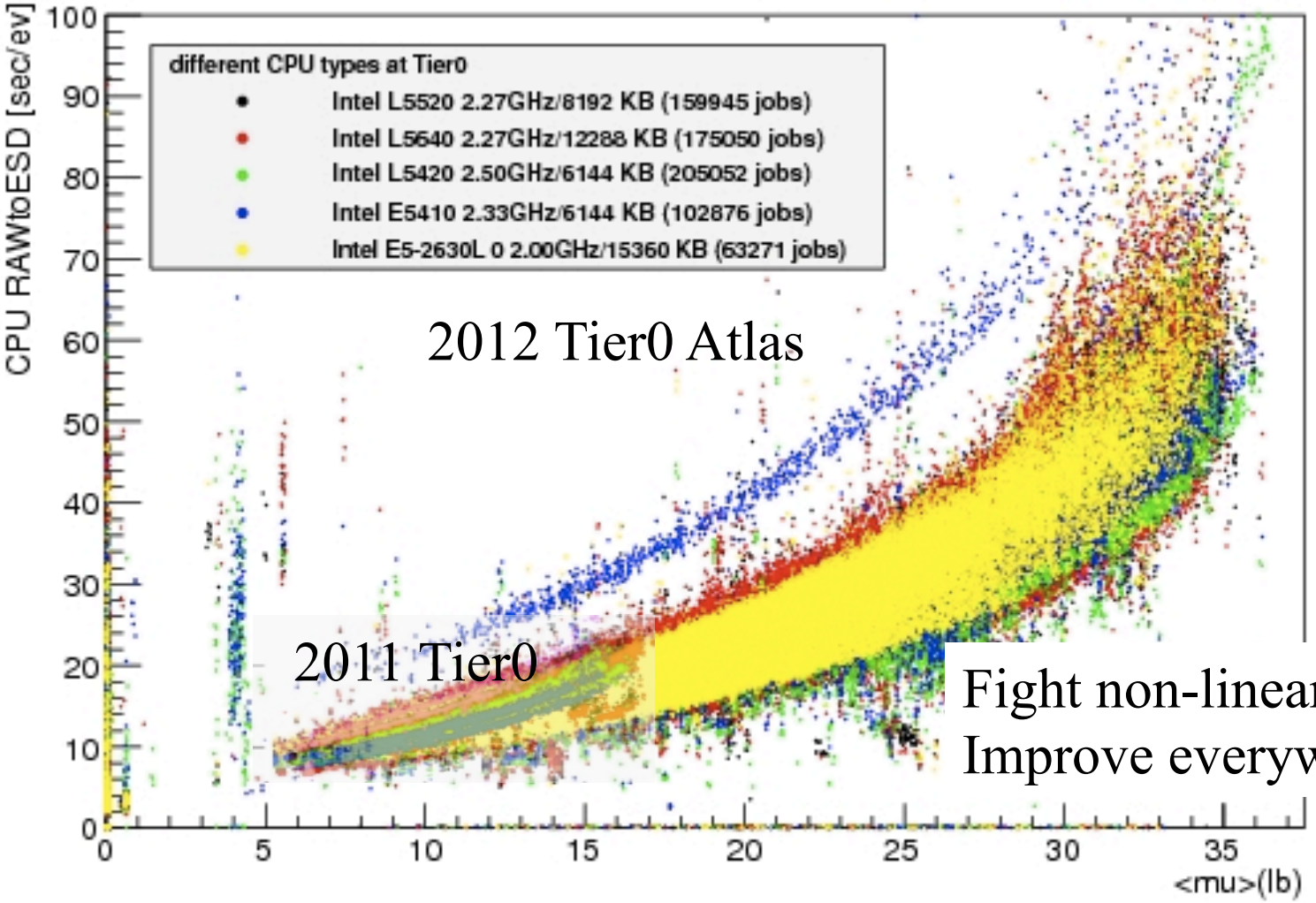


# Reconstruction

- Reconstruct analysis objects from raw data
- High Level Trigger code is looking more and more like (offline) reconstruction code:
  - access to more info on the event
  - desire to have the exact same algorithms online and offline for better control of systematics
  - still one major difference w.r.t. CPU optimisation: trigger optimised to reject fast bad events rather than reconstruction of good events
- If cpu for reconstruction is a problem (and it is), one possible idea (CMS) is to focus on interesting events (active dataset) and archive the rest (i.e. like a trigger with mercy).
- But let's have a look first at the CPU issue➔

# Impact of pileup on reco

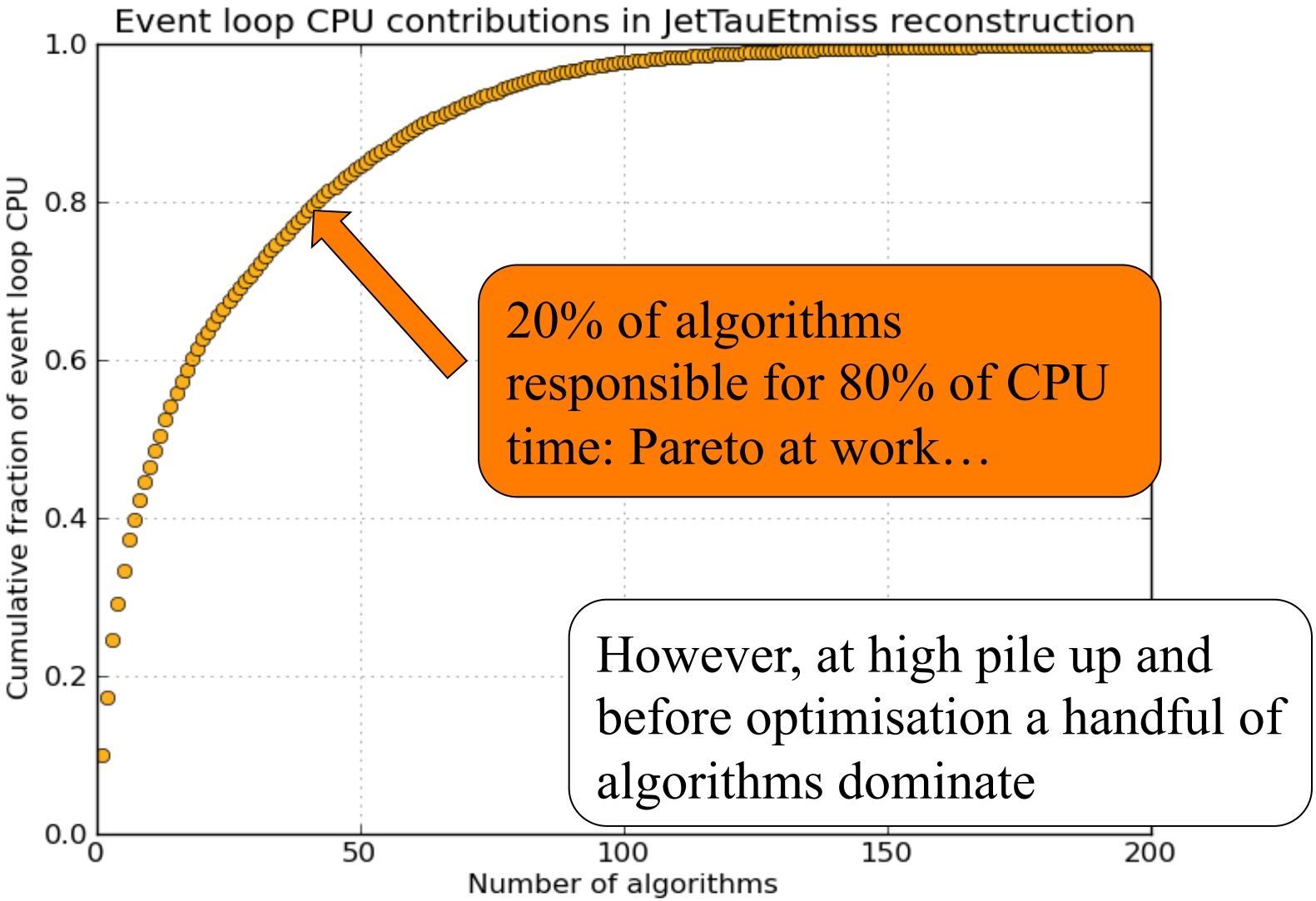
Reconstruction CPU time at Tier0 vs pileup (Jet stream)




Fight non-linearities
Improve everywhere

150➔

# CPU per algorithm (reco)



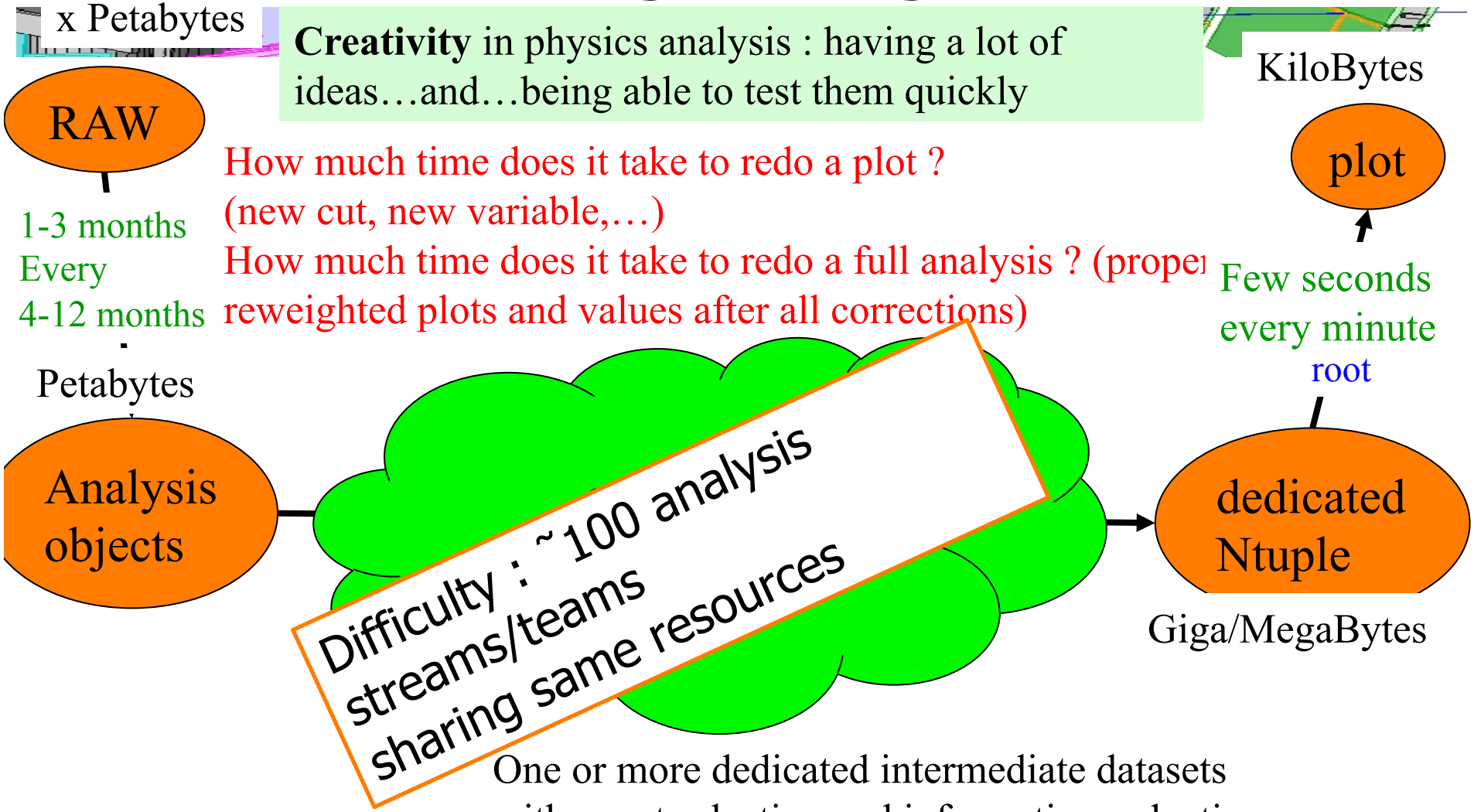
# CPU per domain (reco)


- Tracking dominant in absolute and also non linear behavior
- However, the sum of other domains (calorimeter, jets, muons,…) is similar
- ➔need to improve across the board
- (CMS is slightly more tracking dominated)

# Analysis software

- Two levels of analysis (often mixed up)
  - Event combinatorics (e.g. compute Higgs candidate mass per event):
    - Need explicit event loop
    - Root or dedicated frameworks CMSSW, Gaudi/Athena
  - Final analysis
    - no explicit event loop
    - TTree->Draw
    - histogram manipulations
    - Limit/signal setting RooFit/RooStat
    - Root definitely the framework here.
- Order kB per event, a small fraction of events used, not CPU intensive➔I/O bounded jobs
- In the future, x10 larger disks, x10 larger bandwidth but disk access rate unchanged to a few 100Hz (still hard-drives, no SSD)
- ➔even more sophisticated data organisation/access methods will be needed (experiment Event Data Model and Root persistence)

# Analysis cycle




One or more dedicated intermediate datasets with event selection and information reduction. Balance between completeness, **resource** usage, speed to use and speed to reproduce the dataset.

# Software for GPU

- Graphic co processor massively parallel, up to x100 speed-up on paper
- In practice, task must be prepared by traditional CPU and transferred to GPU
- Successfully used in HEP for very focussed usage e.g. Alice HLT tracking (gain factor 3 in farm size), now also in other experiments
- Code need to be written from scratch using libraries such as Cuda, etc…
- …and largely rewritten/retuned again for different processors, generation
- Need to maintain a second, traditional, version of the code for simulation
- Usage on the grid unlikely/difficult due to the variety of hardware
- In the future, expect progress in generic libraries (e.g. OpenCL) which would ease maintenance (one code for all processors) at an acceptable loss in performance

# Common software

- Can we have more common software ? (beyond flagships Geant4, Root)
- One monster software with `if (CMS) do_this(); if (LHCb) do_that();` ? certainly not…
- Still, we can most likely do more that what we are doing right now
- Note that we are largely running on the same Grid (even the same processor can run at one time some cores with Atlas jobs, some with CMS, some with LHCb)
- Three angles:
  - Framework : introducing parallelism at different levels
  - Foundation libraries
  - Highly optimised HEP Toolboxes
- Even if we do not share software, it is essential we share experience, developer tools and infrastructure (already happening since 2012 in the "concurrency forum", which scope is to be increased)
- We should share tutorials, e.g. do's and dont's to write thread safe code, proper use of vector libraries etc…

# Frameworks

- Hard to argue different HEP experiments have really different needs w.r.t their framework
- In practice, the LHC experiments are using different frameworks (except Atlas and LHCb with different flavours of Gaudi)
- Not the place to dwell on the many reasons for this
- Also changing to a new framework is a huge change (compared to a drop-in replacement of e.g. minimisation algorithm)
- Still if not code, many lessons can be shared

# Toolbox : FastJet example

- FastJet success due to simultaneous occurrence of:
  - Very clever sequential-recombination algorithm (order nxlog(n) vs n^3) « just » a speed-up but a huge one at LHC multiplicities
  - New physics algorithm AntiKt became a de facto standard
  - Many algorithms and plug-ins added later on
  - Easy to interface as a drop-in replacement of previous in house jet algorithms
- No attempt to oversell: FastJet library still interfaced with experiment specific code
  - 1) Mapping EDM back and forth
  - 2) Preparing input: calibration, noisy channel masking
  - 3) Massaging output : more calibration
  - 2) and 3) are physics loaded, depends of the detector peculiarities and high level choices (e.g. calorimeter based jet vs particle flow jet)

# Toolboxes

- The algorithm should be sufficienty CPU intensive
- Interfacing to and fro (from experiment specific representation to the Toolbox representation) should use negligible resource (CPU and memory) and little manpower
- Most efficient data representation to be used (e.g. adjacent in memory)
- Micro-parallelism : vectorisation and specific processor instruction used as much as possible (as few experts can do it)
- Optimisation to the latest compilers and processors instruction set
- Possible examples:
  - Track extrapolating in inhomogeneous magnetic field, track fitting (on the other hand pattern recognition most often require experiment specific implementation to maximise both CPU and physics performance)
  - Calorimeter clustering, different flavours e.g. sliding window, topological cluster
- Benefit:
  - Not necessarily breakthrough algorithm like FastJet, but it could be that algorithms that have been disregarded in the past are more suitable for parallelisation (e.g. track fitting with Kalman filtering being essentially iterative is not very parallelisable)
  - For LHC experiments where these algorithms already exist, share the CPU optimisation manpower (experts)
  - For new experiment, no need to reinvent the wheel
- Has been attempted in the past, more compelling arguments now

# Foundation libraries

- Study (semi) drop-in replacement of low level libraries like (examples):
  - Arithmetic functions
  - Memory management
  - Random number generators
  - Geometry (e.g. CLHEP) and 4-vectors (e.g. TLorentzVector)
- Sometimes, even the right set of compiler/linker options can bring a few percent for free

# Summary

- HL-LHC : high pile-up and high read-out rate

➔ large increase of processing needs

- With flat resource (in euros), and even with Moore's law holding true (likely, provided we maintain/improve efficient use of processors), this is not enough (by ½ to one order of magnitude)

➔large software improvement needed

- Future evolution of processors: many cores with less memory per core, more sophisticated processors instructions (micro-parallelism), possibility of specialised cores➔
  - Optimisation of software to use high level processors instructions, especially in identified hot spots (expert task)
  - Parallel framework to distribute algorithms to cores, in a semi-transparent way to regular physicist software developer
- LHC experiments code base more than 15 millions of line of code, written by more than 3000 people➔a whole community to engage, starting essentially now, new blood to inject
- We are sharing already effort and software. We can do much more: concurrency forum http://concurrency.web.cern.ch