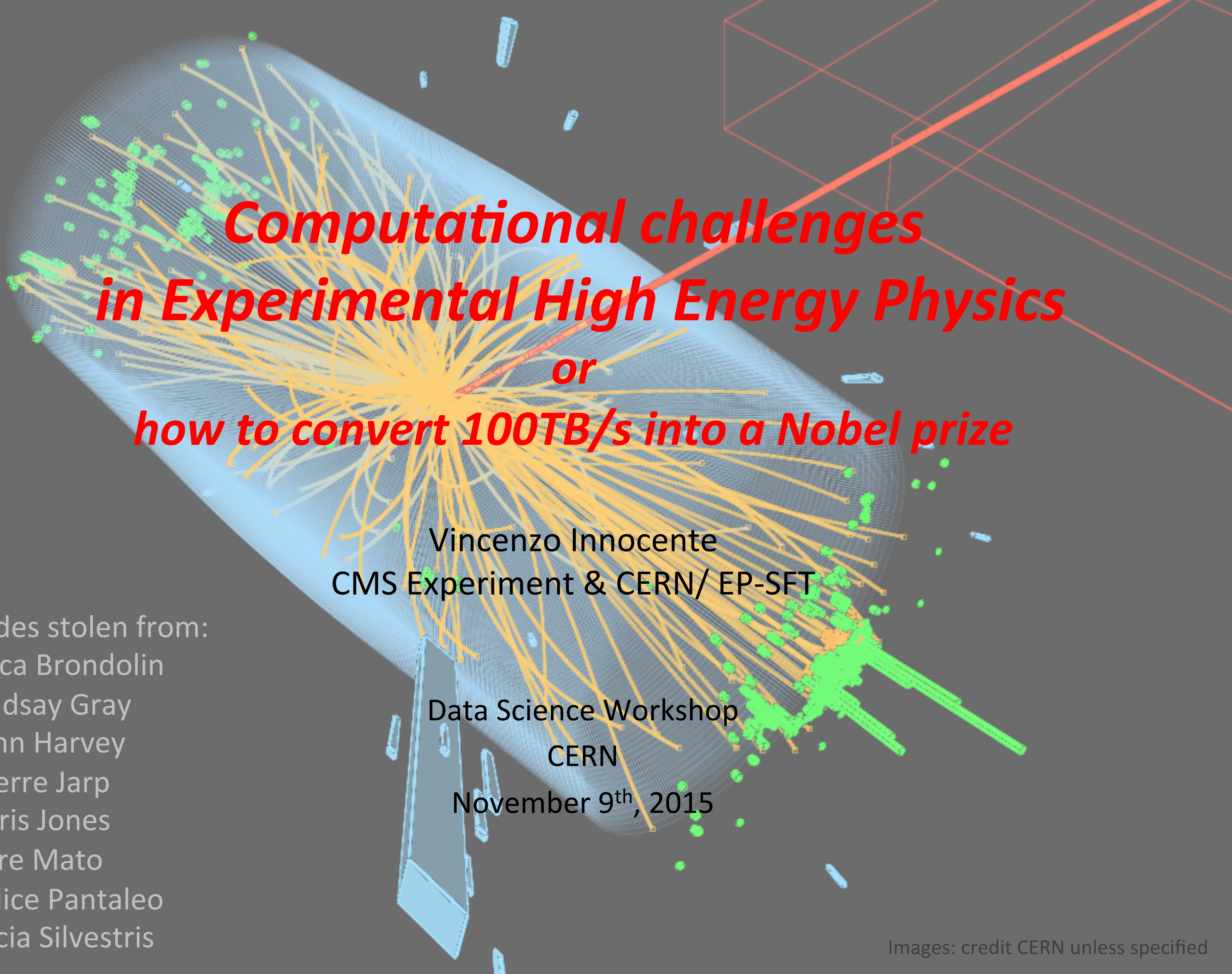


Computational challenges in Experimental High Energy Physics
or
how to convert 100TB/s into a Nobel prize

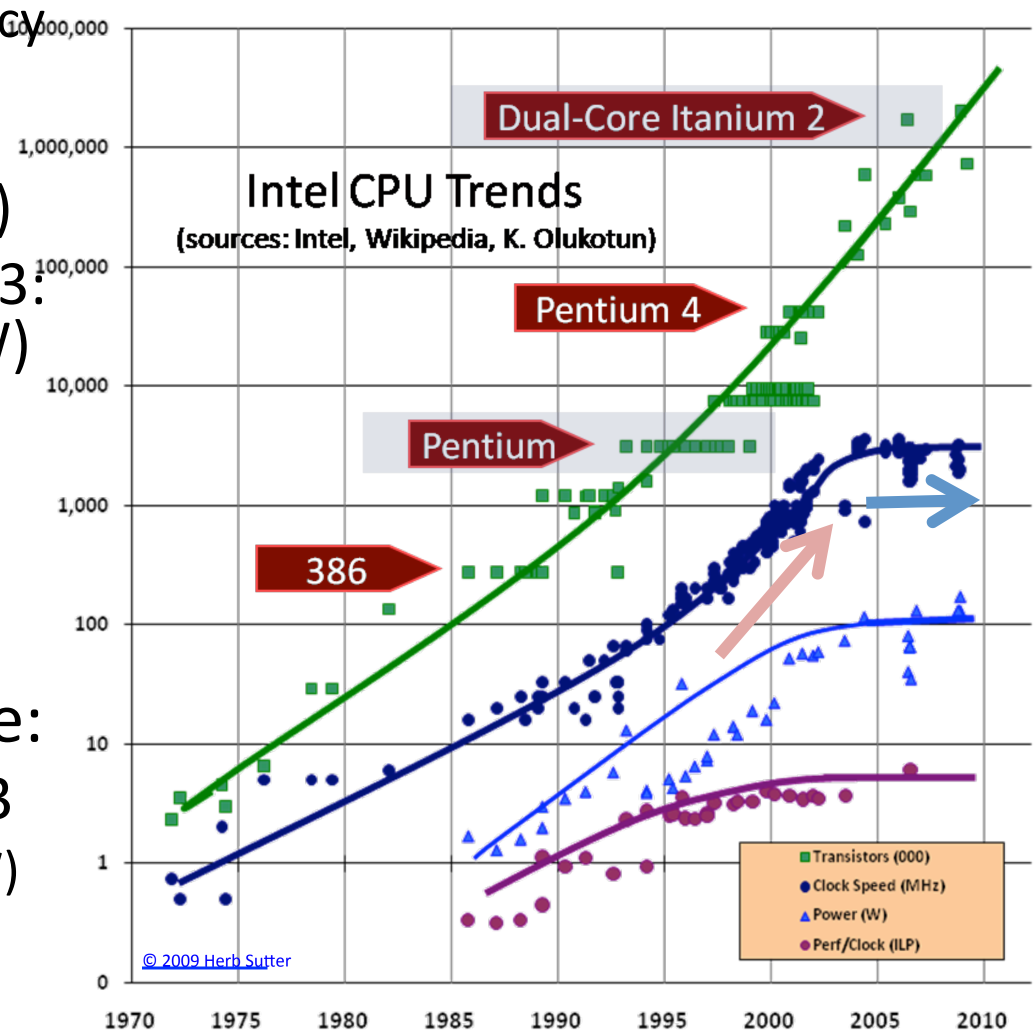
Vincenzo Innocente
CMS Experiment & CERN/ EP-SFT

Data Science Workshop
CERN
November 9th, 2015

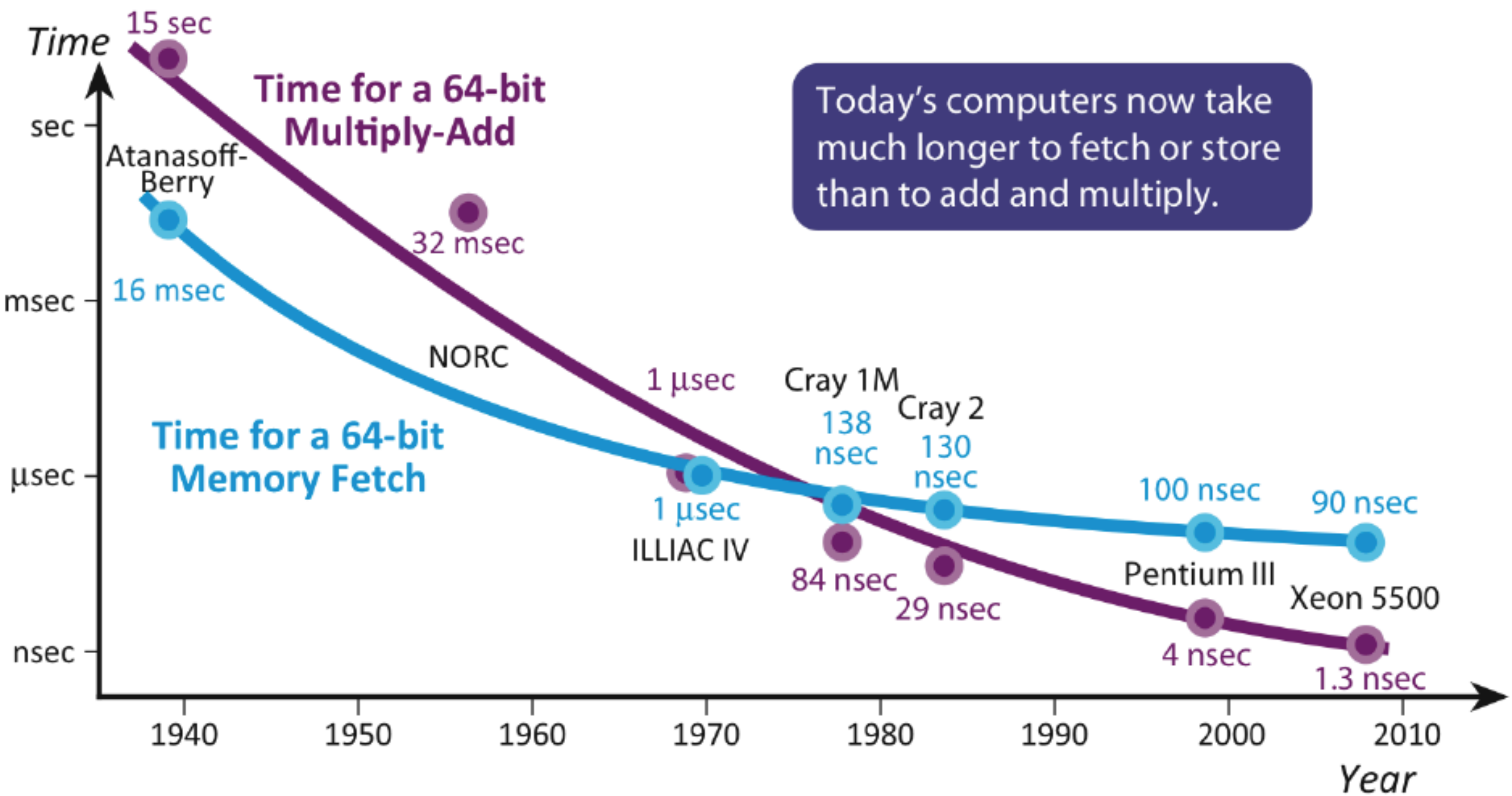
Slides stolen from:
Erica Brondolin
Lindsay Gray
John Harvey
Sverre Jarp
Chris Jones
Pere Mato
Felice Pantaleo
Lucia Silvestris

Why are we here today?

- The 7 "fat" years of frequency scaling:
  - The Pentium Pro in 1996: 150 MHz (12W)
  - The Pentium 4 in 2003: 3.8 GHz (~25x) (115W)
- Since then
  - Core 2 systems:
    - ~3 GHz
    - Multi-core
- Recent CERN purchase:
  - Intel Xeon E5-2630 v3
    - "only" 2.40 GHz (85W)
    - 8 core

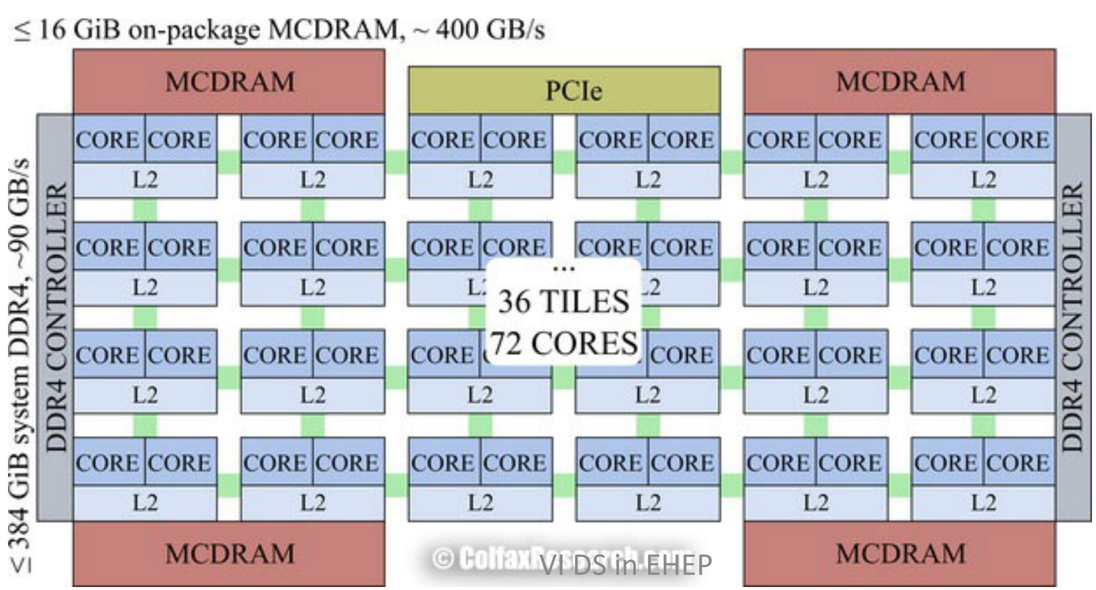


Memory Latency

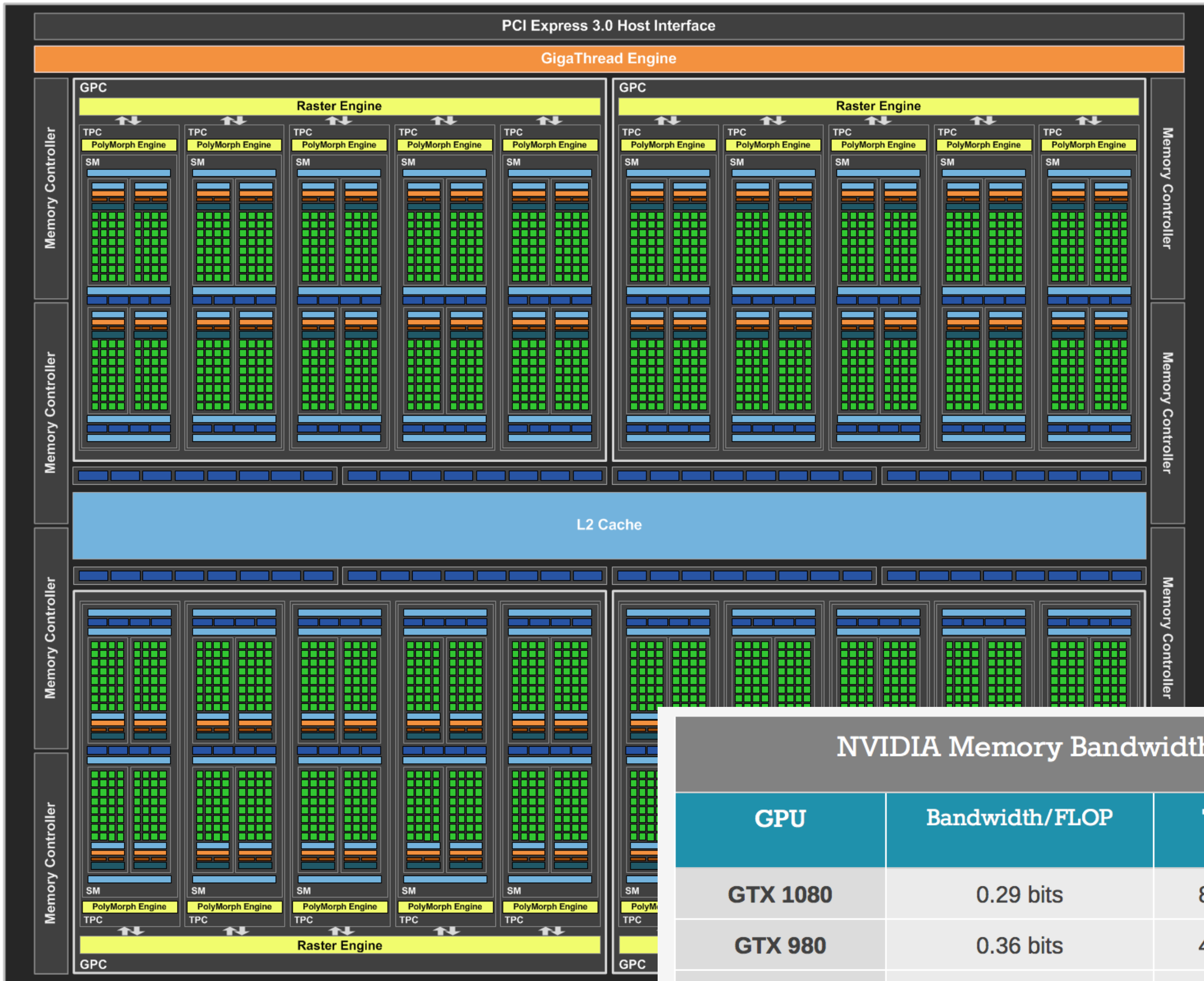


Simple, but illustrative example

- KNL has ~64 cores @1.30GHz, 2FMA port (VPU) each, 4-way hardware threading, hardware vectors of size 8 (Double Precision), 16GB of fast memory:
- 3TFLOPS DP for 400GB/s = 0.5bit/flop-sp
  – 60 fp-ops = 1 fp-load



Streaming Multiprocessor Architecture



credit AnandTech

NVIDIA Pascal
32 CUDA core
x4 x5 x4 = 2560
Floating Point Units
@1.7GHz
8GB fast memory

Require 110 fp-ops
to compensate
one memory access!

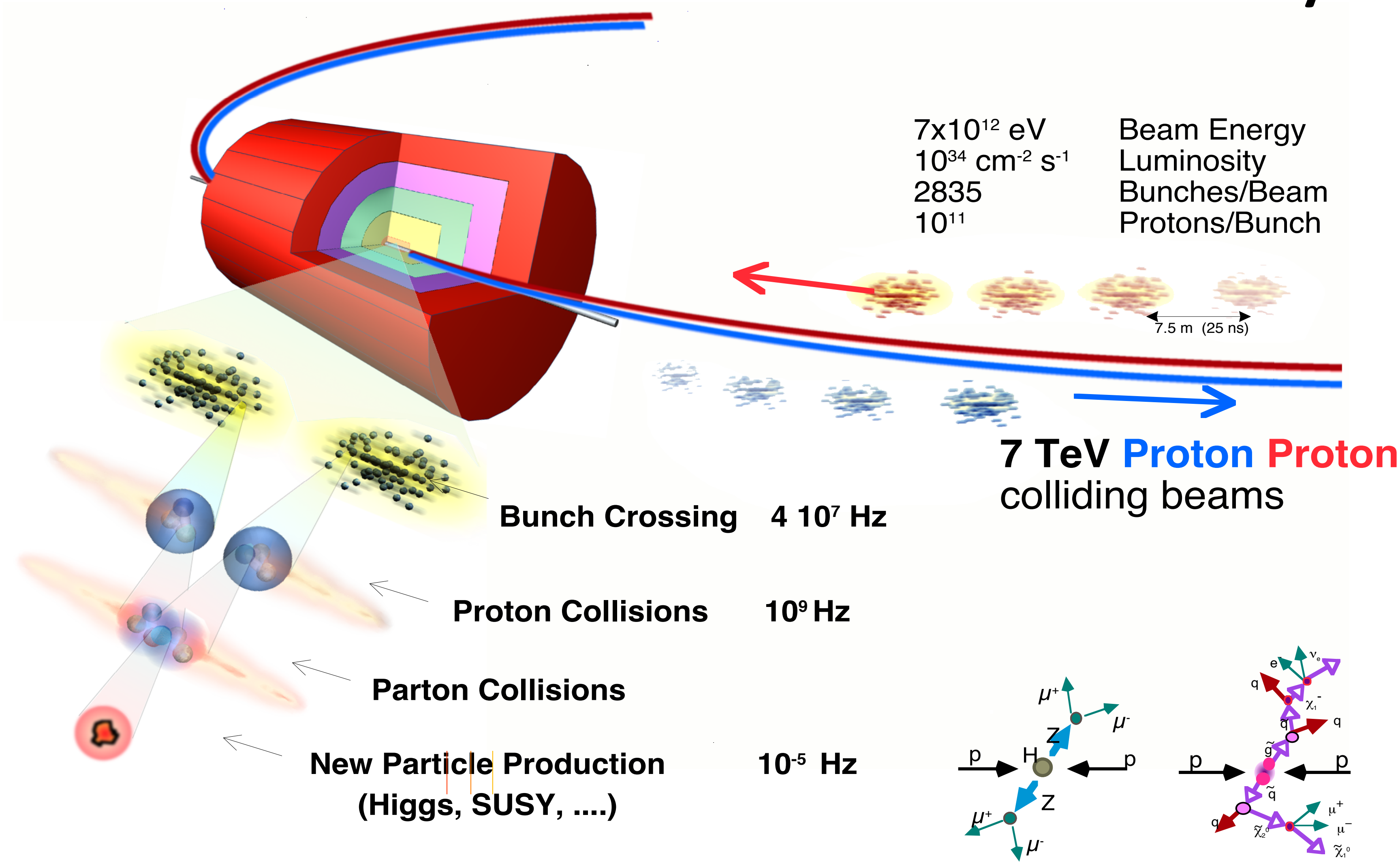
NVIDIA Memory Bandwidth per FLOP (In Bits)

GPU	Bandwidth/FLOP	Total FLOPs	Total Bandwidth
GTX 1080	0.29 bits	8.87 TFLOPs	320GB/sec
GTX 980	0.36 bits	4.98 TFLOPs	224GB/sec
GTX 680	0.47 bits	3.25 TFLOPs	192GB/sec
GTX 580	0.97 bits	1.58 TFLOPs	192GB/sec

Conclusions

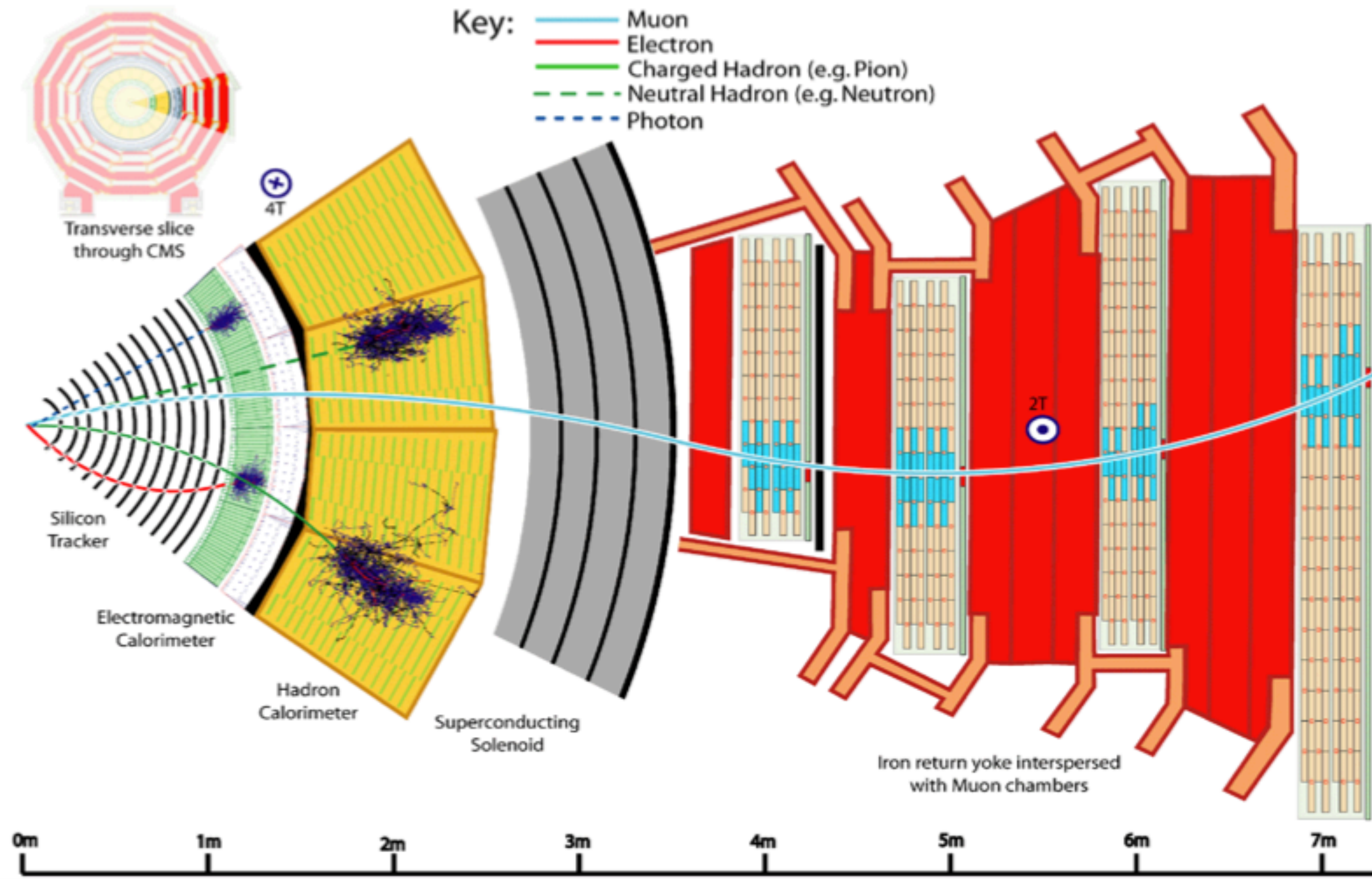
- Improving throughput or latency requires exploiting optimal massive parallelization at all levels
- Speeding up algorithms will not pay up if memory access is not reduced

Collisions at the LHC: summary

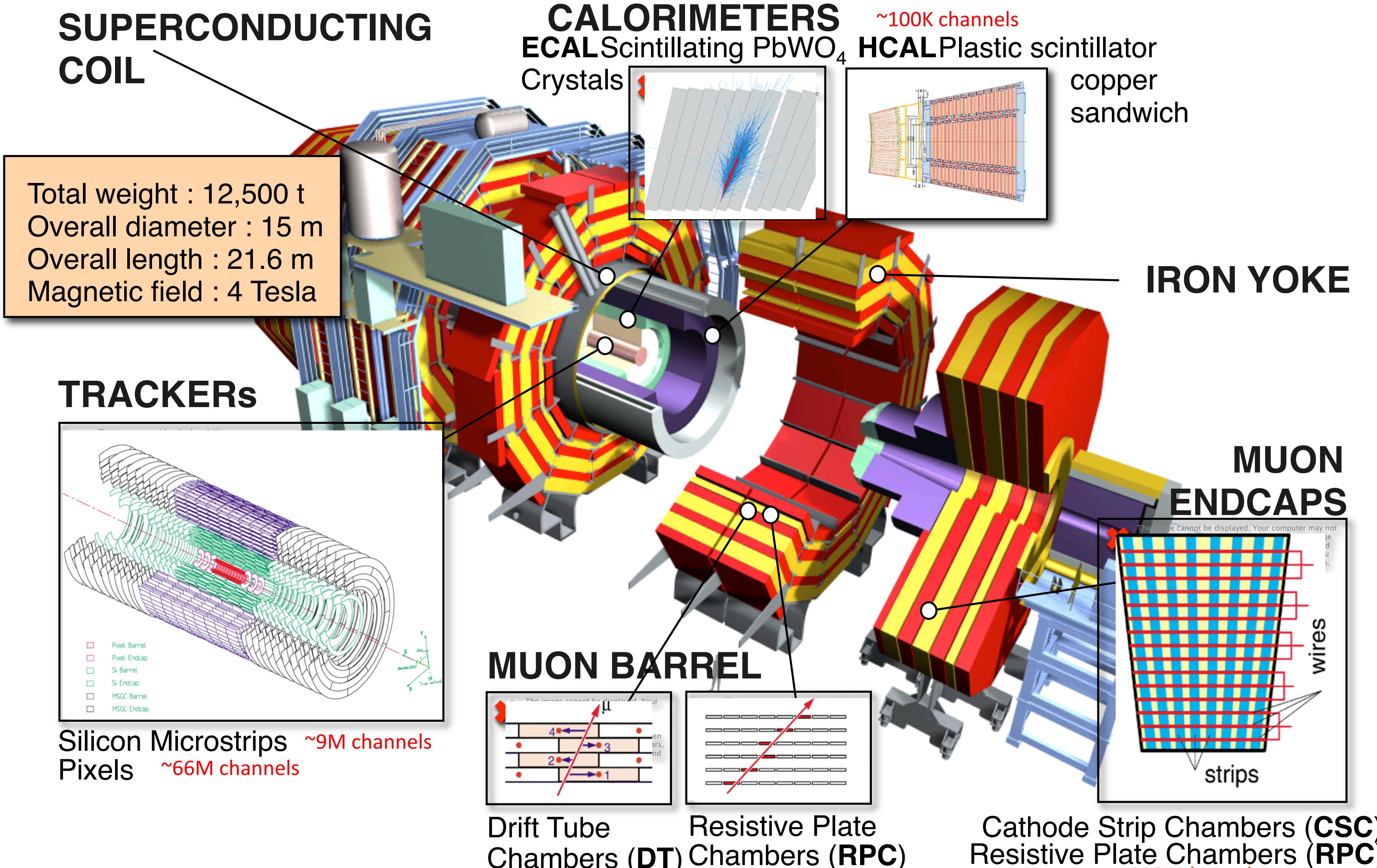


Selection of 1 event in 10,000,000,000,000

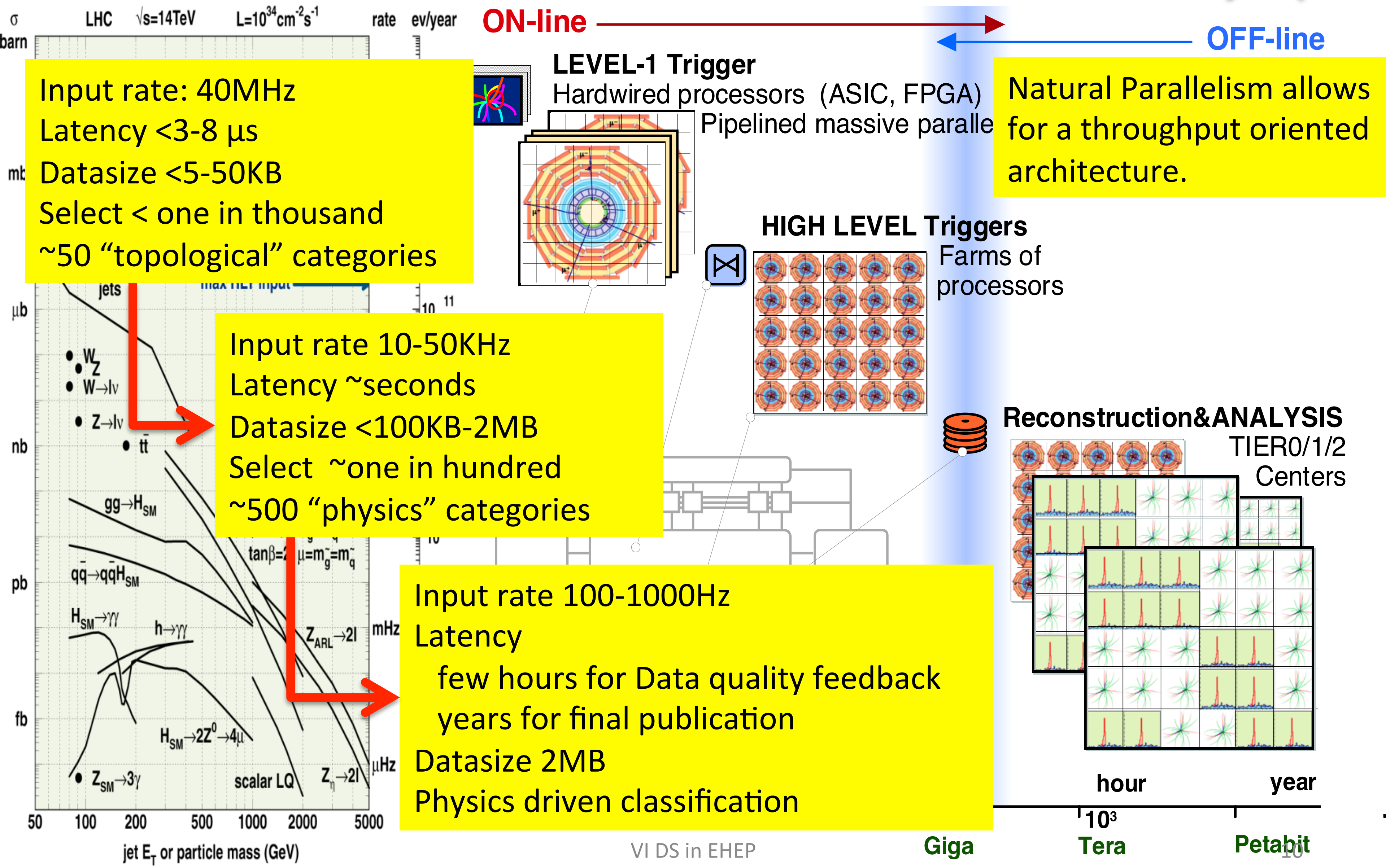
Detector "onion" structure



An experiment: CMS



Data Flow



ON-line
OFF-line
LEVEL-1 Trigger
Hardwired processors (ASIC, FPGA)
Pipelined massive paralle
Natural Parallelism allows for a throughput oriented architecture.
Input rate: 40MHz
Latency <3-8 μs
Datasize <5-50KB
Select < one in thousand
~50 "topological" categories
HIGH LEVEL Triggers
Farms of processors
Input rate 10-50KHz
Latency ~seconds
Datasize <100KB-2MB
Select ~one in hundred
~500 "physics" categories
Reconstruction&ANALYSIS
TIER0/1/2 Centers
Input rate 100-1000Hz
Latency
few hours for Data quality feedback
years for final publication
Datasize 2MB
Physics driven classification
hour
year
10³
Giga
Tera
Petabit

Toward 2023

- High Luminosity: proton collisions per bunch-crossing (PU) 40 -> 200
  - x5 more occupancy in detectors
  - **Access to new corners of phase-space**
    - **High Mass, Boosted topologies**
      - **Dense environment**
- New Detectors
  - New Tracker
    - Higher granularity (x4), extended coverage, hardware trigger capability
  - CMS: New High granularity Calorimeter
  - Timing information
- First Level Trigger
  - Include Tracking information
  - Output Rate up to 1MHz
- High Level Trigger
  - More use of tracking
  - Detailed analysis in search of new signals
  - Output Rate up to 10KHz
- Offline
  - Not just do as well as today but at PU 200
  - **More precision to look for tiny signals of New Physics**

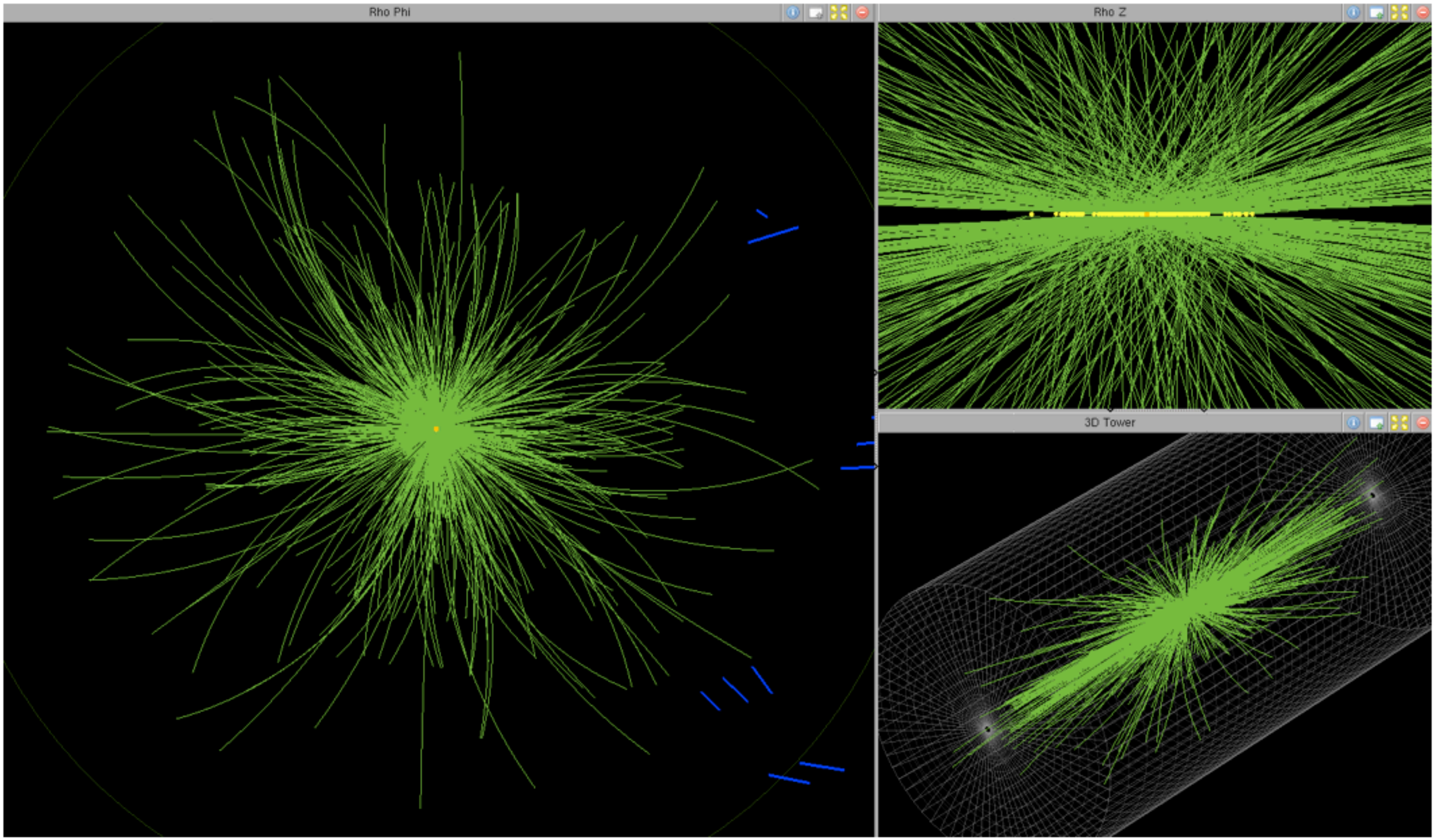
Data Hierarchy: Our solution to BigData

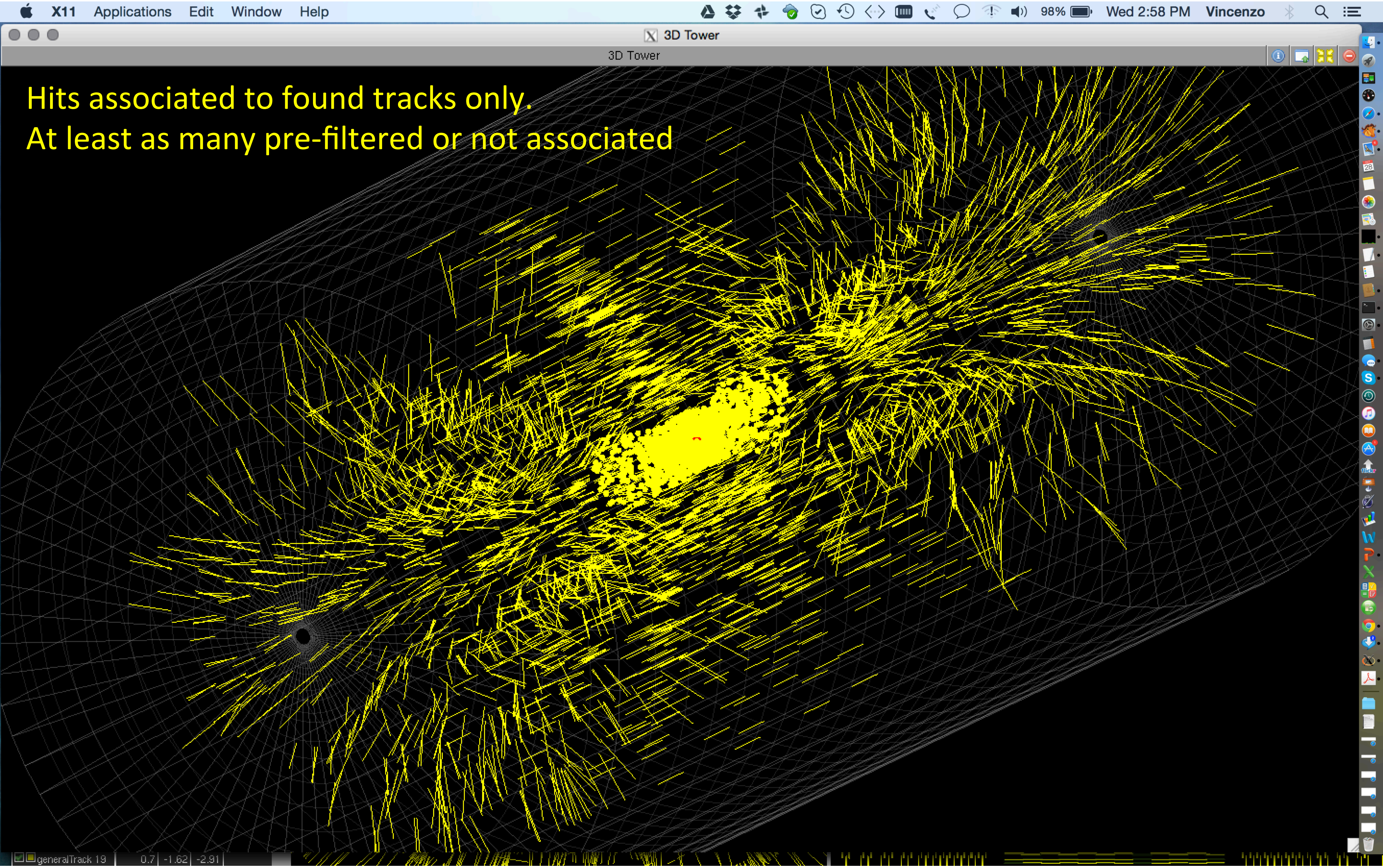
"RAW, ESD, AOD, TAG"

~2 MB/event ~9MB with Sim ~80MB at PU200	RAW	Triggered events recorded by DAQ	Detector digitisation
~100 kB/event ~2MB with Sim ~23MB at PU200	ESD/RECO	Reconstructed information	Pseudo-physical information: Hits, Clusters, track candidates
~10 kB/event <1kB prune/compress	(mini)AOD	Analysis information	Physical information: Transverse momentum, Association of particles, jets, (best) id of particles,
~1 kB/event	TAG/tuple	Classification information	Relevant information for "fast" event selection

Reconstruction of CMS Simulated Event

$t\bar{t}$ event at <PU>=140 (94 vertices, 3494 tracks)

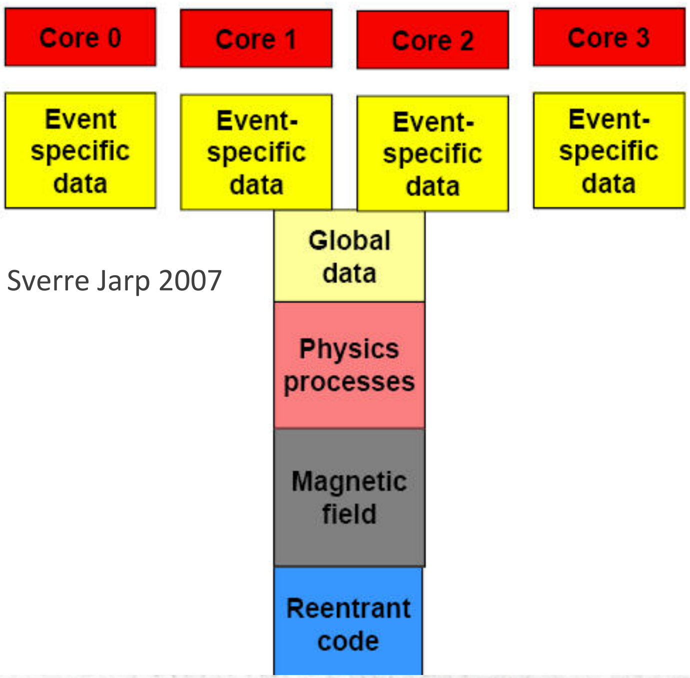



X11 Applications Edit Window Help
98% Wed 2:58 PM Vincenzo
3D Tower
3D Tower
Hits associated to found tracks only.
At least as many pre-filtered or not associated
generalTrack 19
0.7 -1.62 -2.91

Event parallelism

Opportunity: Reconstruction Memory-Footprint shows large condition data

How to share common data between different process?

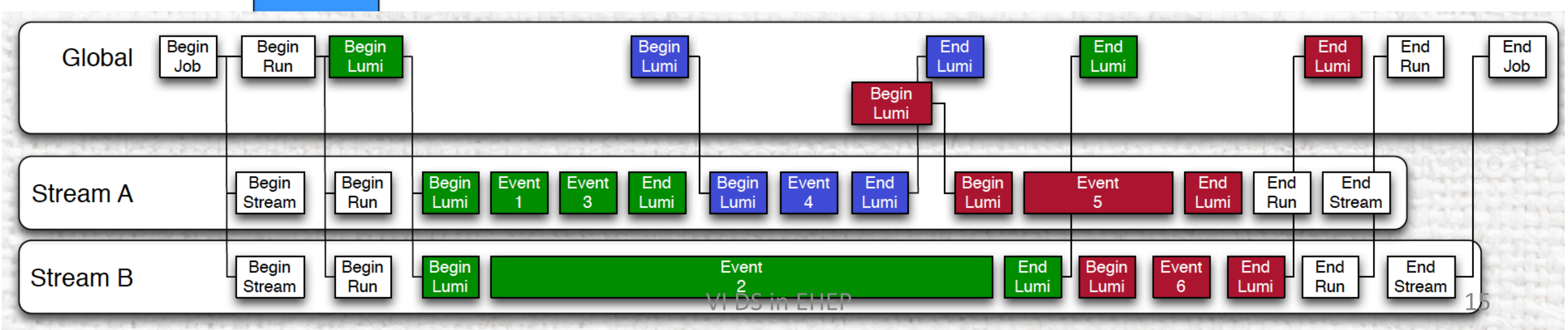


Sverre Jarp 2007

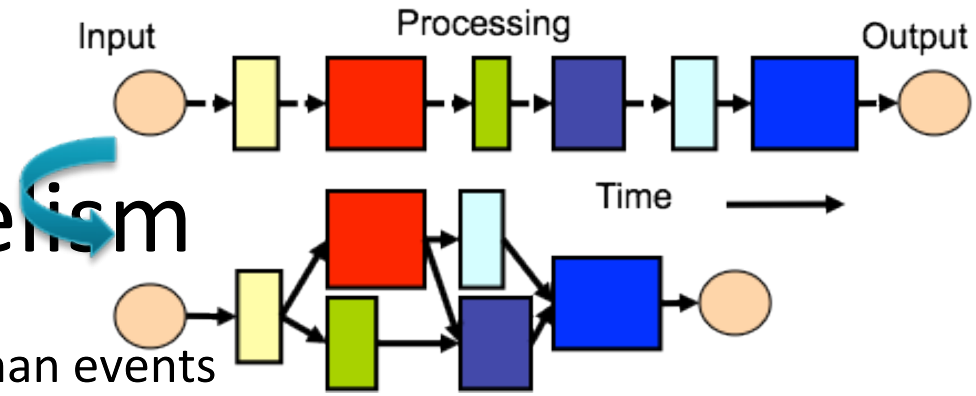
→ multi-process and multi-thread applications are now in production

→ CMS simulation and reconstruction runs on KNL with 126 threads well within the16GB of fast memory

→I/O remains a problem...



Beyond event-level parallelism



- Why?
  - » We may endup with more core than events
  - » Resources (shared access to memory, to disk) may be scarce
    - Typical example is a KNL used as a cluster of ~256 cpus
- Parallelize a DAG workflow is relatively easy including the management of a mild overcommit to mitigate starvation issues
  - » All concurrent framework implements it (or plan to implement it)
  - » To work well it requires a reasonably balanced workflow:
    - a single long pipeline may easily defeat its purpose!
  - » Iterative tracking is the most striking example of long pipeline (50% of reco time spent in it for CMS…)
- NB: up to this point data-processing is fully reproducible independently of the order of execution and granularity of concurrency

Outer loop parallelization

- Typically each processing module has an “outer loop” on its input collection
  - The most trivial concurrency model is to parallelize it
    - “For loop” parallelization is a well established practice
- In CMS proven to work “almost” out of the box for both seed and track building
  - Seed building is fully combinatorial, no reproducibility issues
  - Track building includes “cleaning passes” to remove already used hits
    - Introduces a sequential dependency and therefore an irreproducibility in case of parallel processing
- Current implementation
  - Avoid “cleaning” and pay the price

In-Out parallelization

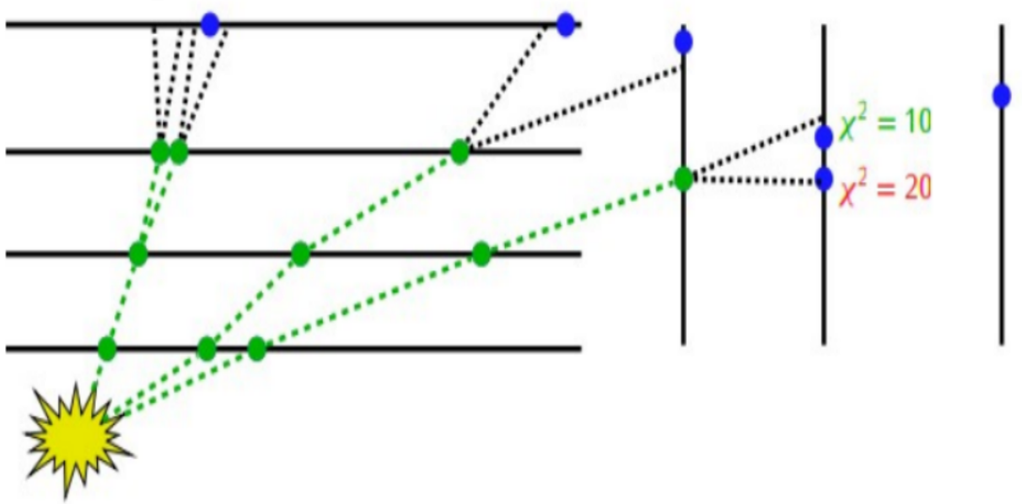
- Out-In parallelization will allow to overcome the limitation of traditional batch processing, exploiting new (heterogeneous) concurrent hardware (SIMD/SIMT) will require a completely new approach, most probably a full rethinking of algorithms, data structures and even of the workflow decomposition
- By definition SIMD/SIMT applies to the innermost loop
  - Either directly or by code transformation
- w/r/t multi-threading, effective concurrency is "broken" in SIMD/SIMT by pretty common patterns such as
  - Branch predication
  - Random memory access
  - Recursion
- SIMD/SIMT algorithms are fragile
  - Supporting a new use case (even adding some protections or a minor variant) may destroy efficient parallelism
  - Often better to duplicate code and/or to partition data and manage conditionals at a higher level (which is not necessarily a bad thing even in general!)
  - Runtime polymorphism is out-of-question: has to be managed outside.
- Mitigation strategies do exist, still for a full efficient use of these architectures a dedicated, specialized software effort is required
  - Think parallel
  - Think local

Making the code SIMD/SIMT friendly

- Several "success stories" in CMS: pattern very similar
  - Transform storage representation in algorithm specific data
    - SOA to AOS, variable transformation, sorting, filtering, re-indexing etc
  - Move all constant components outside
  - Devirtualize, Use explicit RTTI, inline
    - Move from generic to specific
    - Limit the number of use-cases to the few known
  - Make functions to act on collections not on single objects
- The net effect is a significant speed up just from such code transformation
  - In many cases vectorization itself adds little
    - Short inner loops
    - Little computations
    - Branch predication

Traditional track building

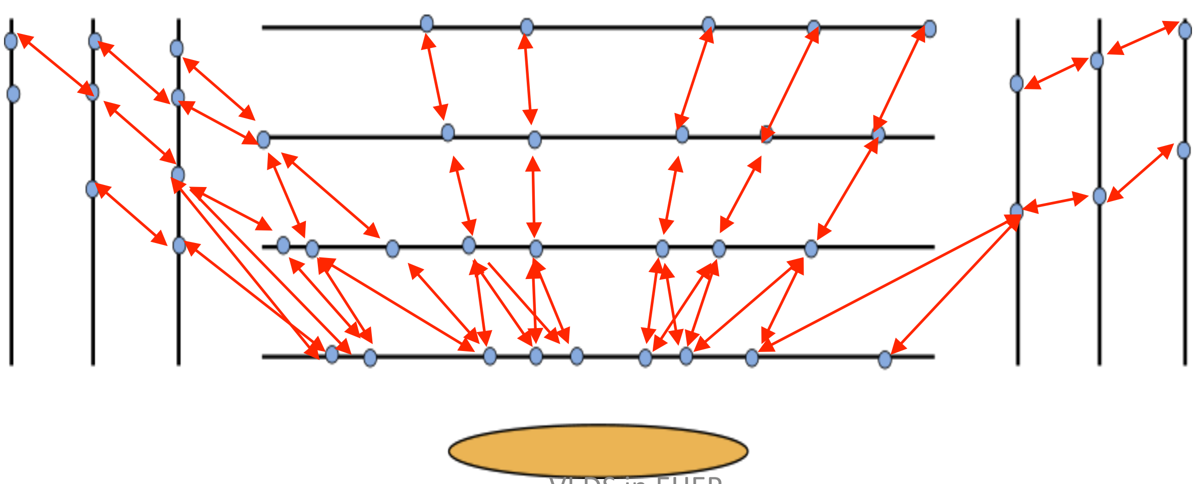
1. Build doublets
2. "Propagate" doublets to third layer and search for compatible hits (open search window on target layer)
3. Propagate 1-2-3 triplet to 4th layer and search for compatible hits



Highly divergent code, optimized to bail out asap.
Easy to parallelize "Outermost Loop", amost impossible to vectorize

Cellular Automaton (CA)

- The CA is a track seeding algorithm designed for parallel architectures
- It requires a list of layers and their pairings
  - A graph of all the possible connections between layers is created
  - Doublets aka Cells are created for each pair of layers (compatible with a region hypothesis)
    - Doublet building identical to traditional approach
  - "Connect" cells that share hit
  - Fast computation of the compatibility between two connected cells
    - Vectorized loop of floating point operations
  - No knowledge of the world outside adjacent neighboring cells required, making it easy to parallelize



Current Performance

- Plan to use Cellular Automaton in its sequential implementation at the HLT already in 2017

Algorithm	time per event [ms]
Traditional Triplets	29
Traditional Quadruplets	72
CPU Cellular Automaton	14
GPU Cellular Automaton	1.2

On GPU CA is Memory-Bandwidth limited
(on CPU as well…)

- Hardware: Intel Core i7-4771@3.5GHz , NVIDIA GTX 1080

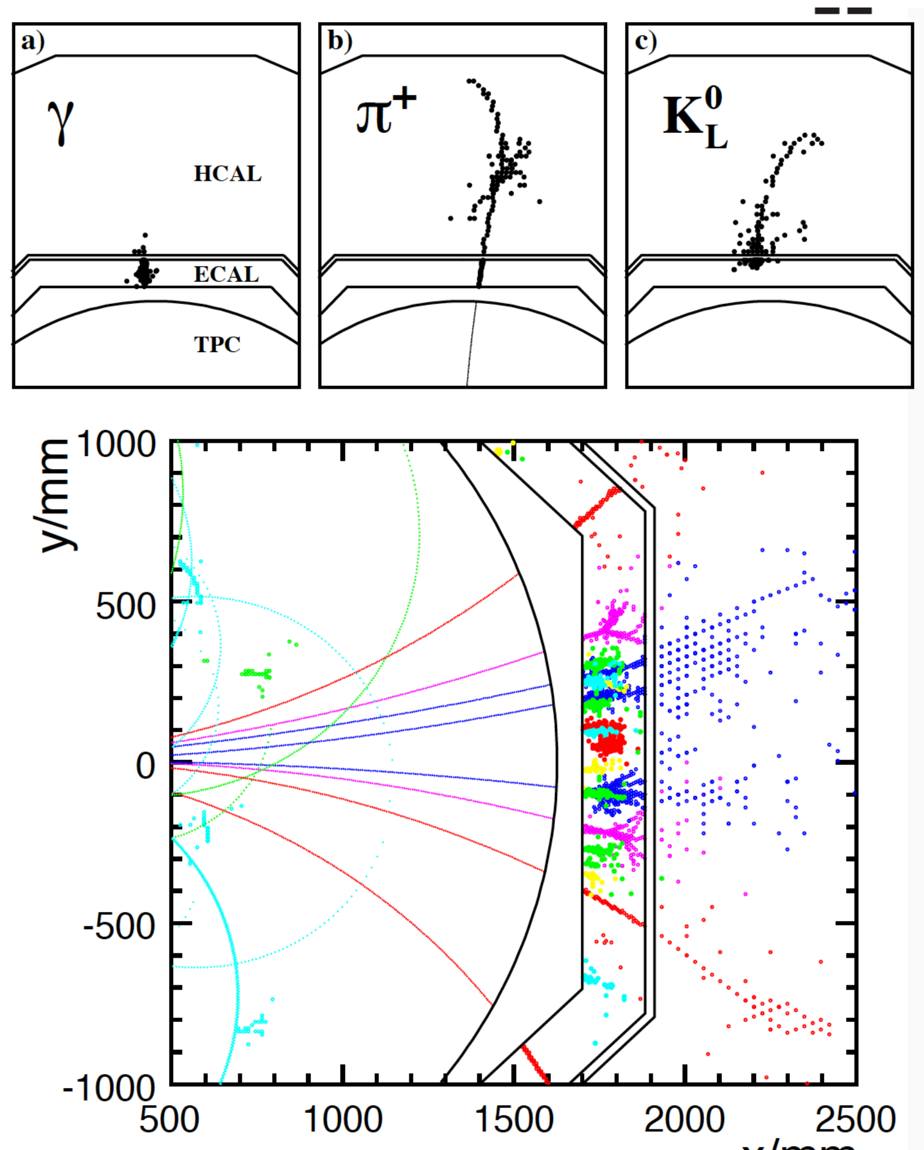
The dream of every experimental HEP Physicist:

Identify and measure each single particle produced in a collision

This may need high resolution calorimetry that will compete with trackers in complexity and data volume

Still, using current data-processing approach, most of this information will reach the physicists only in a very condensed form

Difficult to estimate the real impact of such a detector on physics analysis w/o a new data-processing paradigm



http://arxiv.org/abs/0907.3577

Big Question

- Can a “new” Paradigm make the difference?
  - Artificial Intelligence
    - Used already for classification
  - Dedicated Specialized Hardware
    - In use in First Level Trigger since ever
      - CMS Track trigger demonstrated with latency < 4us
  - Smart data mining
    - Analysis currently limited to a single data-tier level

CMS simulation & data processing Software "Legacy"

- ~10k "modules"
- ~1000 "data processing" modules
- Code (SLOC)
  - C++: 3,558,032 (68.86%)
  - python: 1,240,801 (24.02%)
    - Used only in initialization
  - fortran: 277,857 (5.38%)
    - Interface to physics simulation code
- Total size of TEXT sections : 229,246,680 bytes
  - + ~220MB of "external software"

Conclusions

- Free lunch is over
  - To improve the efficiency of software we need to increase the granularity of parallelism, optimize data access patterns and make use of heterogeneous resources
- Waiting for the definitive standard to emerge we need to develop our own infrastructure to support the implementation of concurrent algorithms able to exploit parallelism on heterogeneous hardware
- Recent work shows that
  - An efficient concurrent schedule of algorithms is feasible
  - With huge effort it is possible to make current algorithm implementations free from data-race (thread safe)
  - Making use of parallelism in algorithms requires a total re-implementation
- More R&D is required to tackle the challenges of
  - Exploiting heterogeneity
  - Efficient parallelize algorithms
  - Efficient utilization of memory hierarchy
  - Efficient utilization of the few developers left

BACKUP

The real issue: maximize throughput

Theoretical peak throughput: the maximum amount of data that a kernel can read **and** produce in the unit time.

$$\text{Throughput}_{peak}\ (\text{GB/s}) = 2 \text{ x access width (byte) x mem\_freq (GHz)}$$

This means that if your device comes with a memory clock rate of 3GHz DDR (double data rate) and a 384-bit wide memory interface, the amount of data that a kernel can process and produce in the unit time is at most:

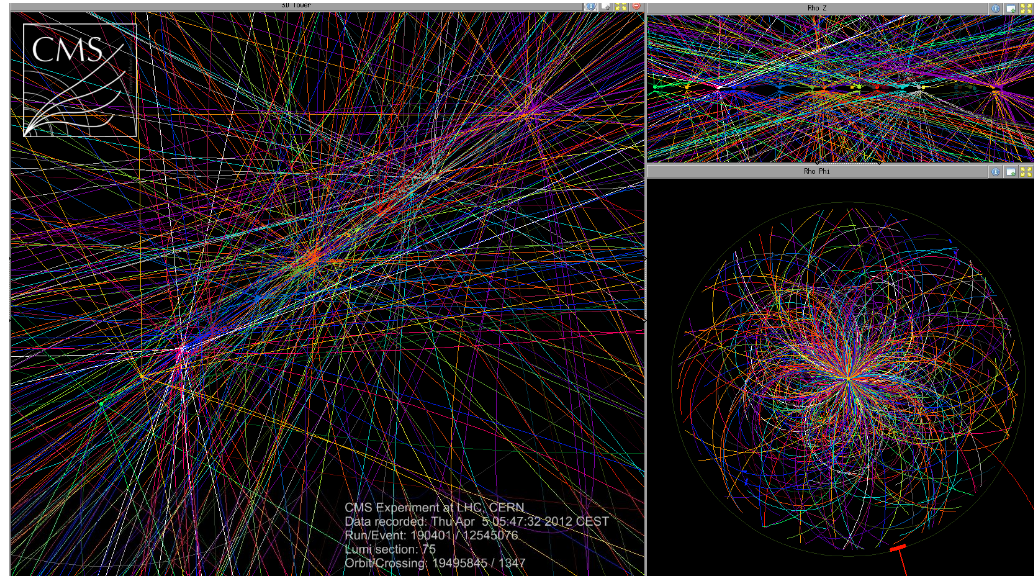
$$\text{Throughput}_{peak}\ (\text{GB/s}) = 2 \text{ x } (384/8)(\text{byte}) \text{ x } 3\ (\text{GHz}) = \mathbf{288\ GB/s}$$

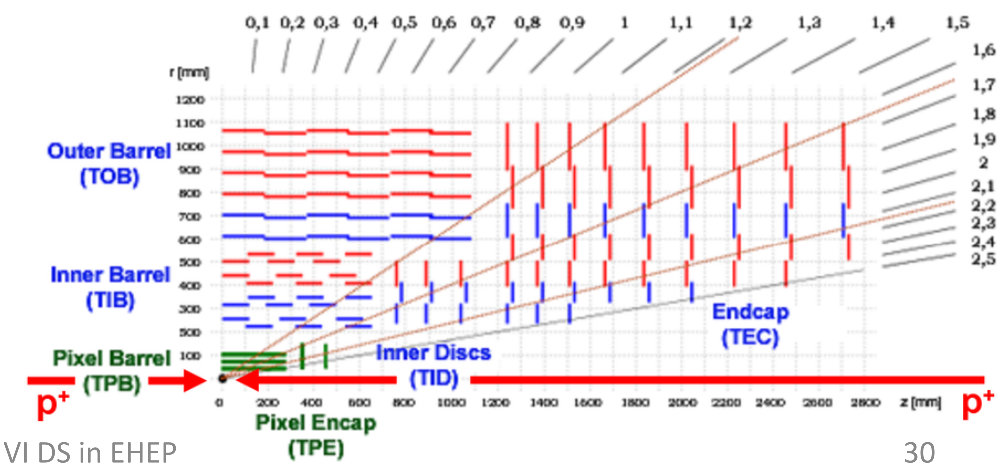
Consequence: cpu starvation!

- NVIDIA TESLA Kepler K40:
  - **1.4 TFLOPS DPFP** peak throughput
  - 288 GB/s peak off-chip memory access bandwidth
    - 36 G DPFP operands per second
- In order to achieve peak throughput, a program must perform 1,400/36 = **~39 DPFP** arithmetic operations for each operand value fetched from off-chip memory
  - In most of current code is **0.5** (fetch two operands, never use them again)!

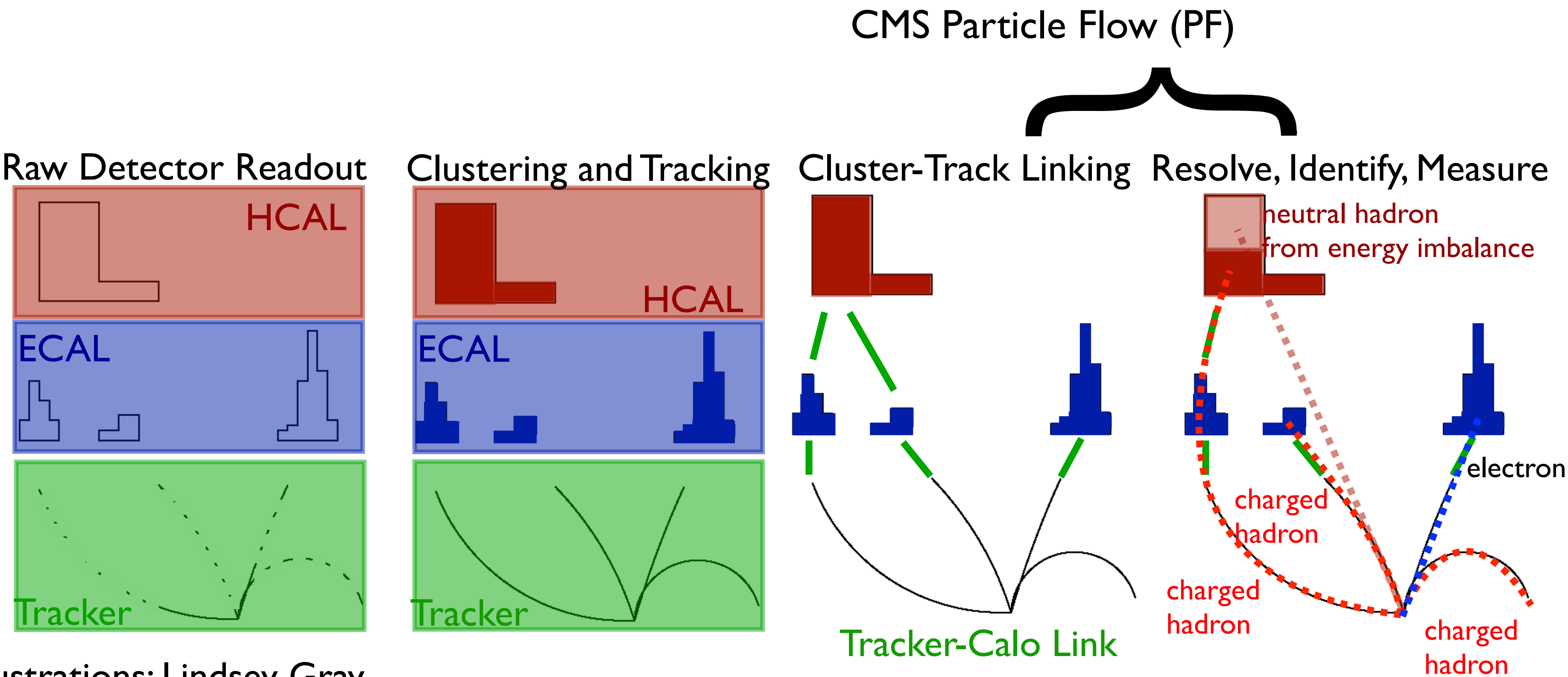
Tracking at CMS

- Particles produced in the collisions leave traces (hits) as they fly through the detector
- The innermost detector of CMS is called **Tracker**
- **Tracking**: the art of associate each hit to the particle that left it
- The collection of all the hits left by the same particle in the tracker along with some additional information (e.g. momentum, charge) defines a **track**
- **Pile-up**: # of p-p collisions per bunch crossing





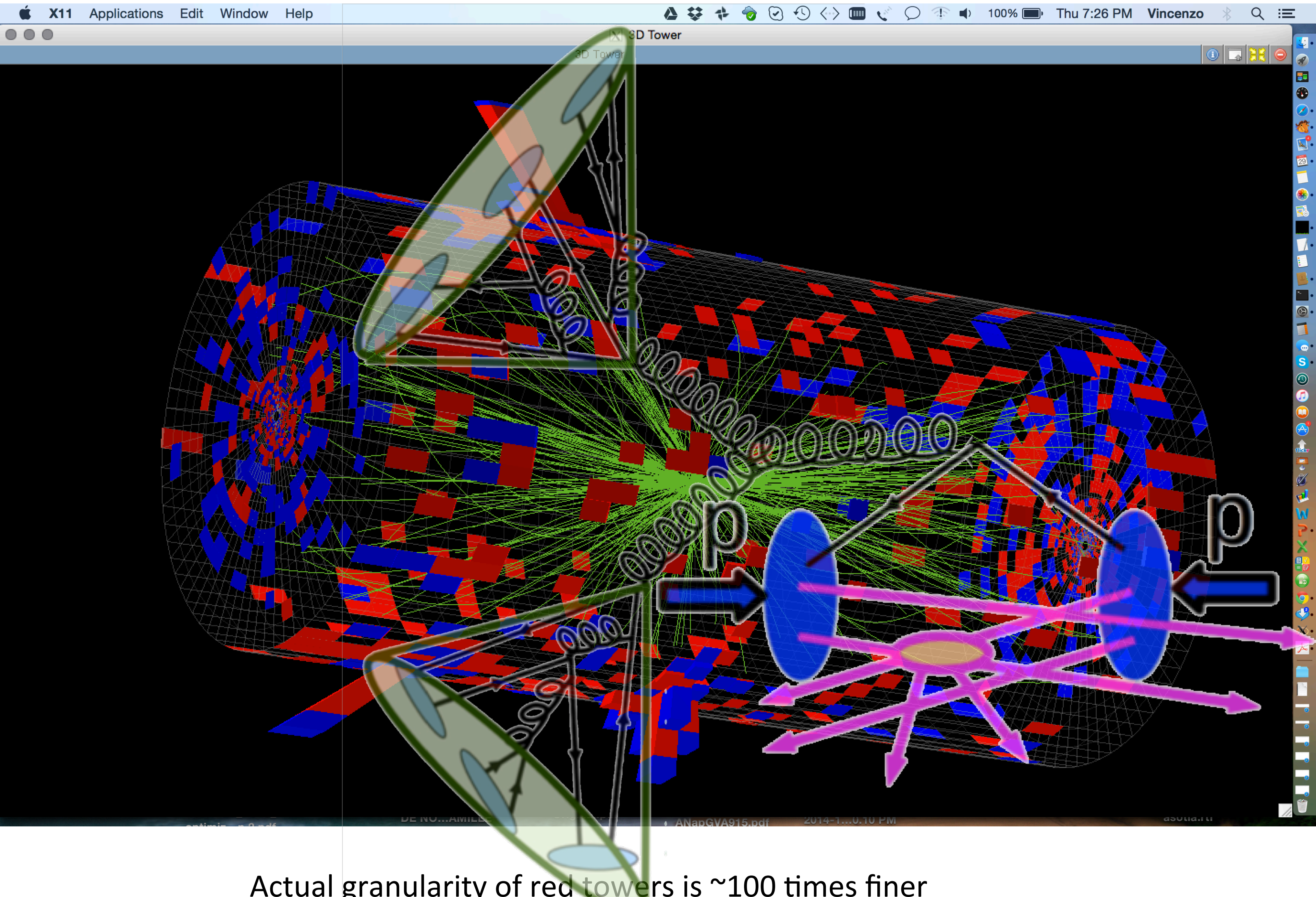
Reconstructing Jet Constituents



Illustrations: Lindsey Gray

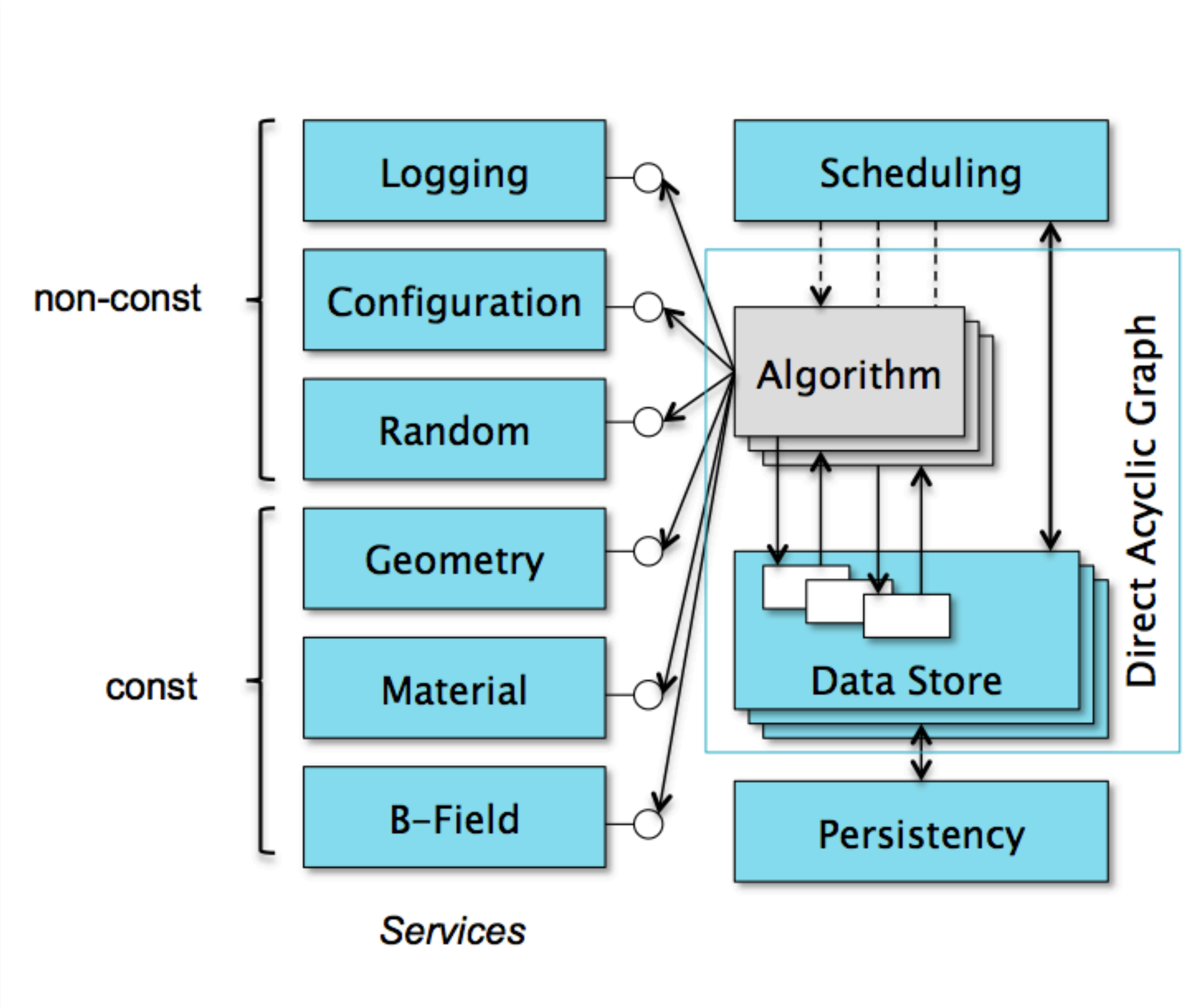
Non trivial regression to compute best estimation of particle energy combining all available information taking into account non-uniformity in detector response

Based on intensive, iterative statistical analysis of data themselves to extract alignment and calibration constants



Actual granularity of red towers is ~100 times finer

HEP Applications



Algorithms read and write from/to the event-data store and the "services"

Only interfaces are defined (with no "cost" associated)

Algorithms are in turn based on a large set of utilities and foundation libraries

A real application (LHCb Brunel)