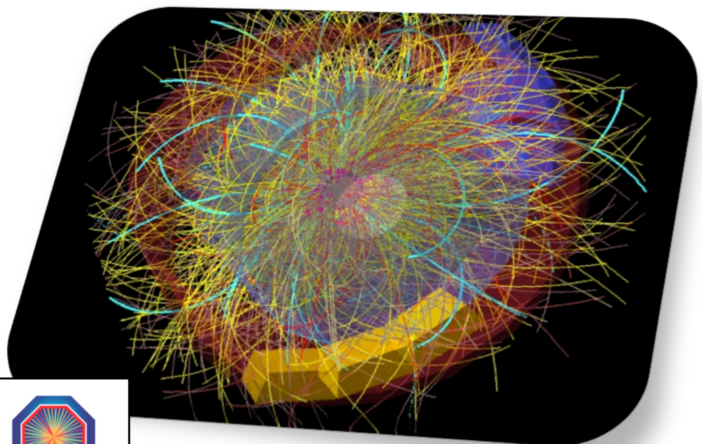
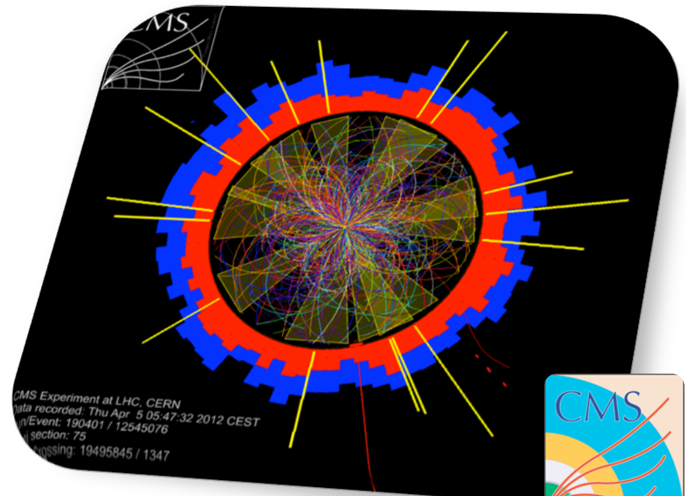
Introduction to HEP numerical computing

Challenges in Data reconstruction and simulation





Danilo Piparo – CERN, PH-SFT

4th Openlab Numerical Computing Workshop

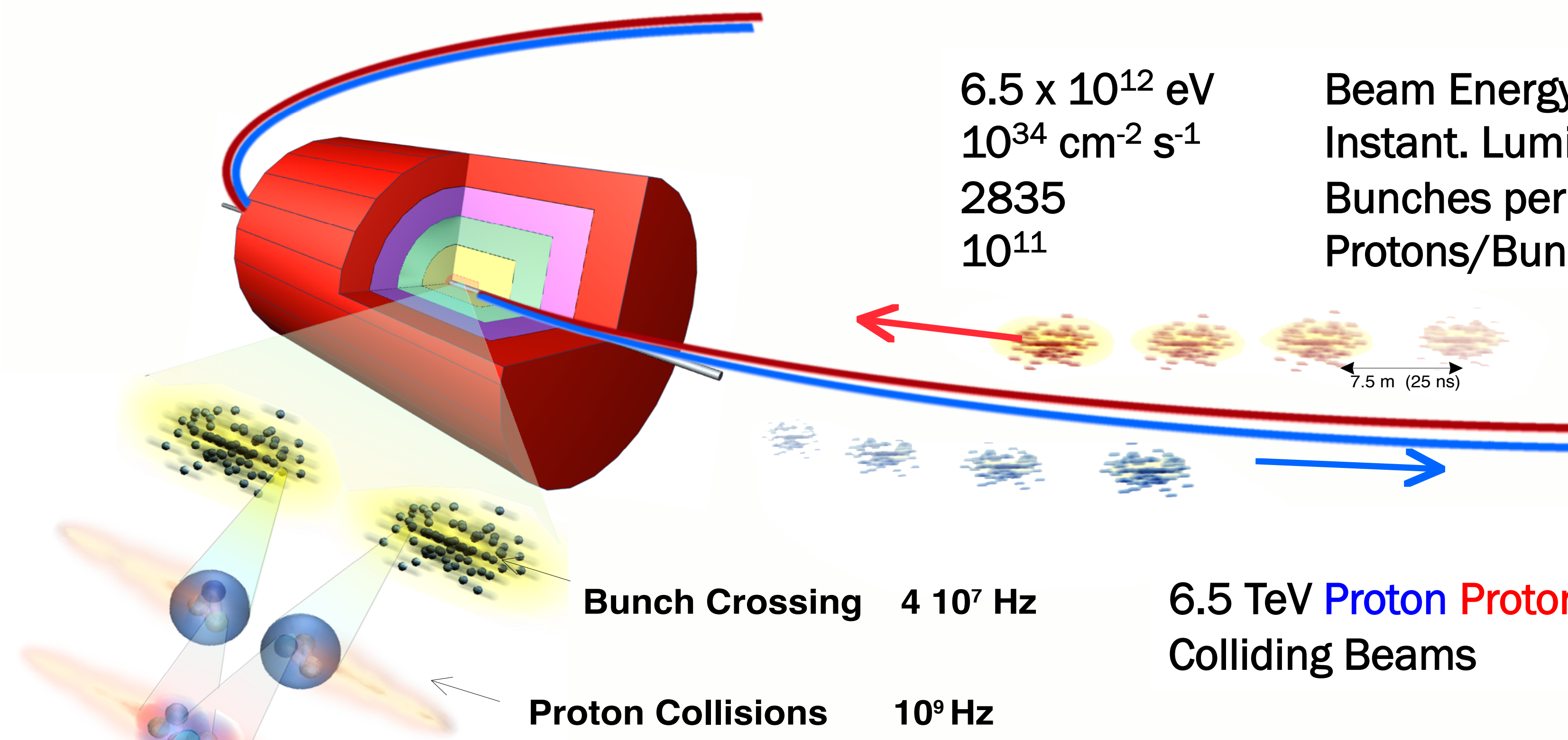
About this talk

- A HEP discovery in a nutshell
- Floating point in HEP algorithms
  - Focus on mathematical functions
- Floating point in data

A big thanks goes to Vincenzo Innocente for contributing to these slides both with ideas and concrete material.

A HEP discovery in a nutshell

Collisions at LHC: a summary



6.5×10^{12} eV	Beam Energy*
10^{34} cm^{-2} s^{-1}	Instant. Luminosity
2835	Bunches per Beam
10^{11}	Protons/Bunch

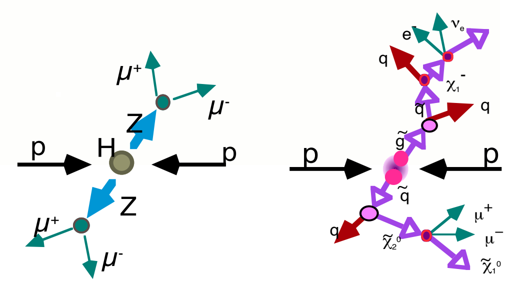
6.5 TeV Proton Proton
Colliding Beams

Bunch Crossing **$4\ 10^{7}$ Hz**

Proton Collisions **10^{9} Hz**

Parton Collisions

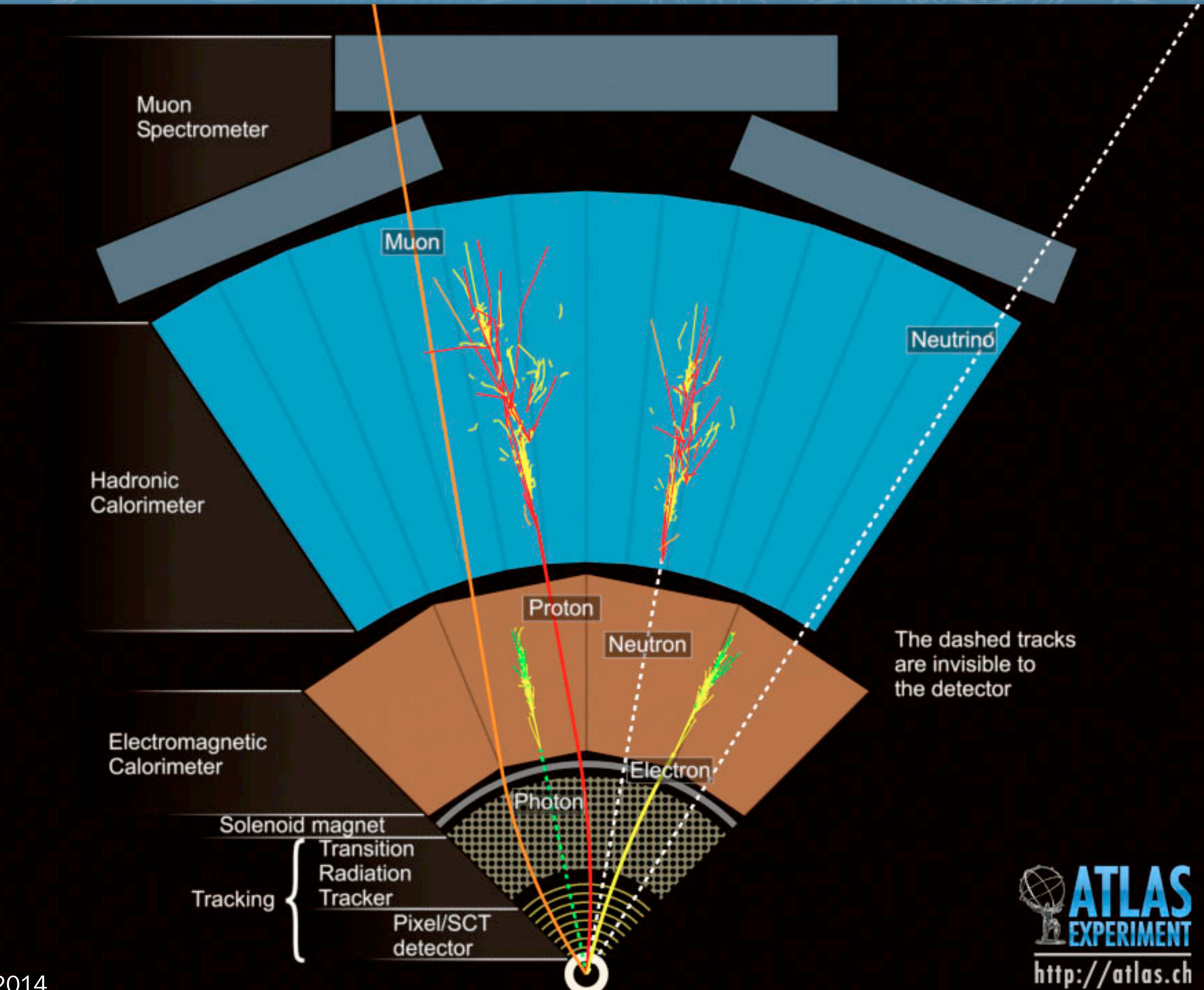
New Particle Production **10^{-5} Hz**
(Higgs, SUSY, ...)



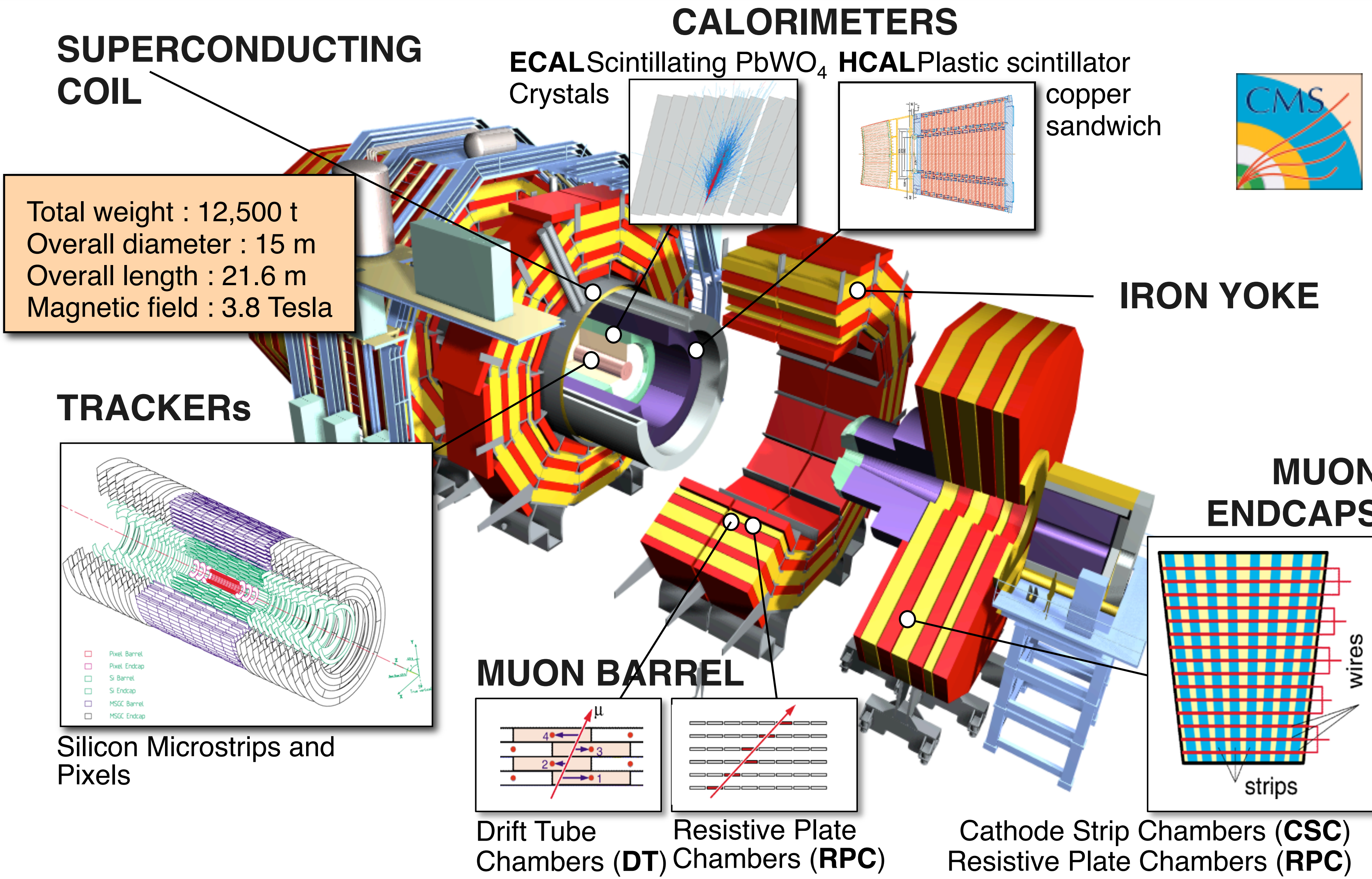
Selection of 1 event in 10,000,000,000,000

* Estimate for the value at the machine re-start, not the official numbers

Typical “Onion structure”

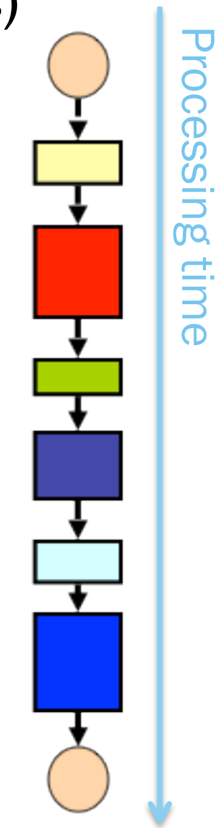


The CMS experiment



Data acquisition and processing

- HEP main data: **statistically independent Events** (particle collisions)
- Simulation, Reconstruction and Analysis: process "one Event at the time"
  - **Event-level parallelism** (success of the Grid!)
  - Landscape is changing: advent of parallel data processing frameworks
- Applications **composed of several algorithms** to:
  - **Create simulated "raw" event data** (event generation+simulation of passage of particles through matter+simulation of detector response to such energy depositions)
  - **Select and transform measured/simulated "raw" event data into "particles"**
- Final result: **statistical data** (histograms, distributions, etc.)
  - Typically**: comparison between simulation and data**
- All of these algorithms:
  - **Are mainly developed by "Physicists"**
  - May require additional "detector conditions" data (e.g. calibrations, Geometry, etc)

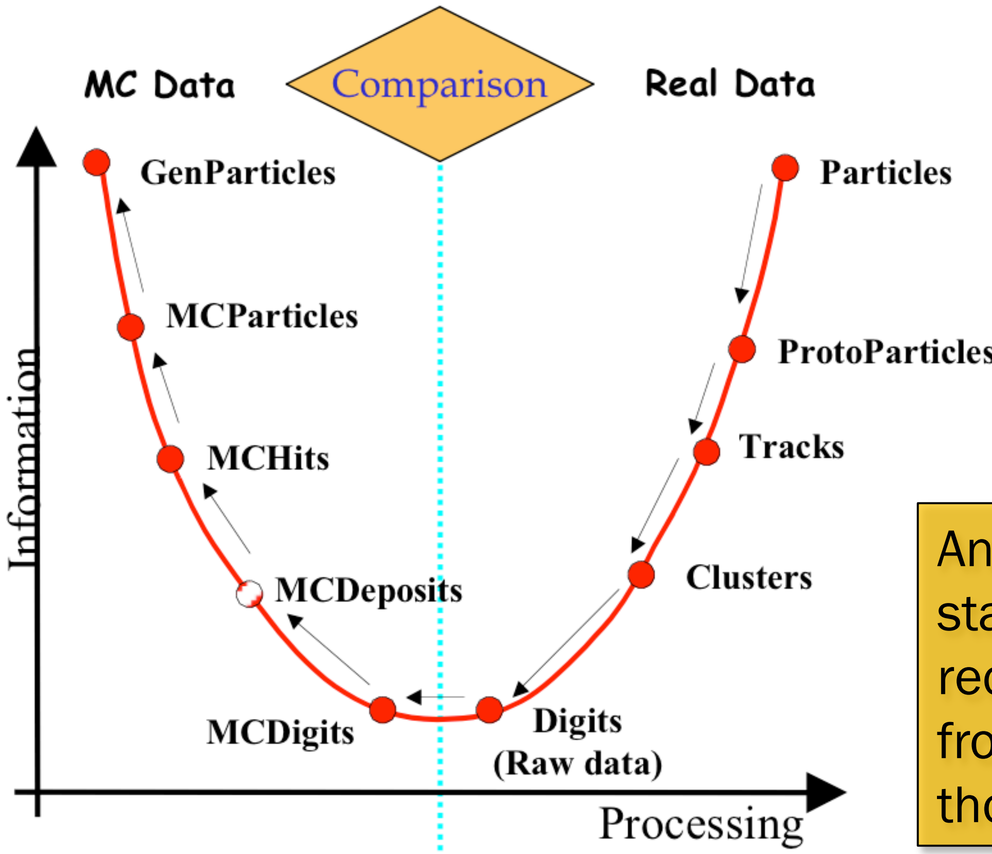


Raw data "definition": readout of the ADC of the subdetectors' frontends

High Energy Physics analysis model

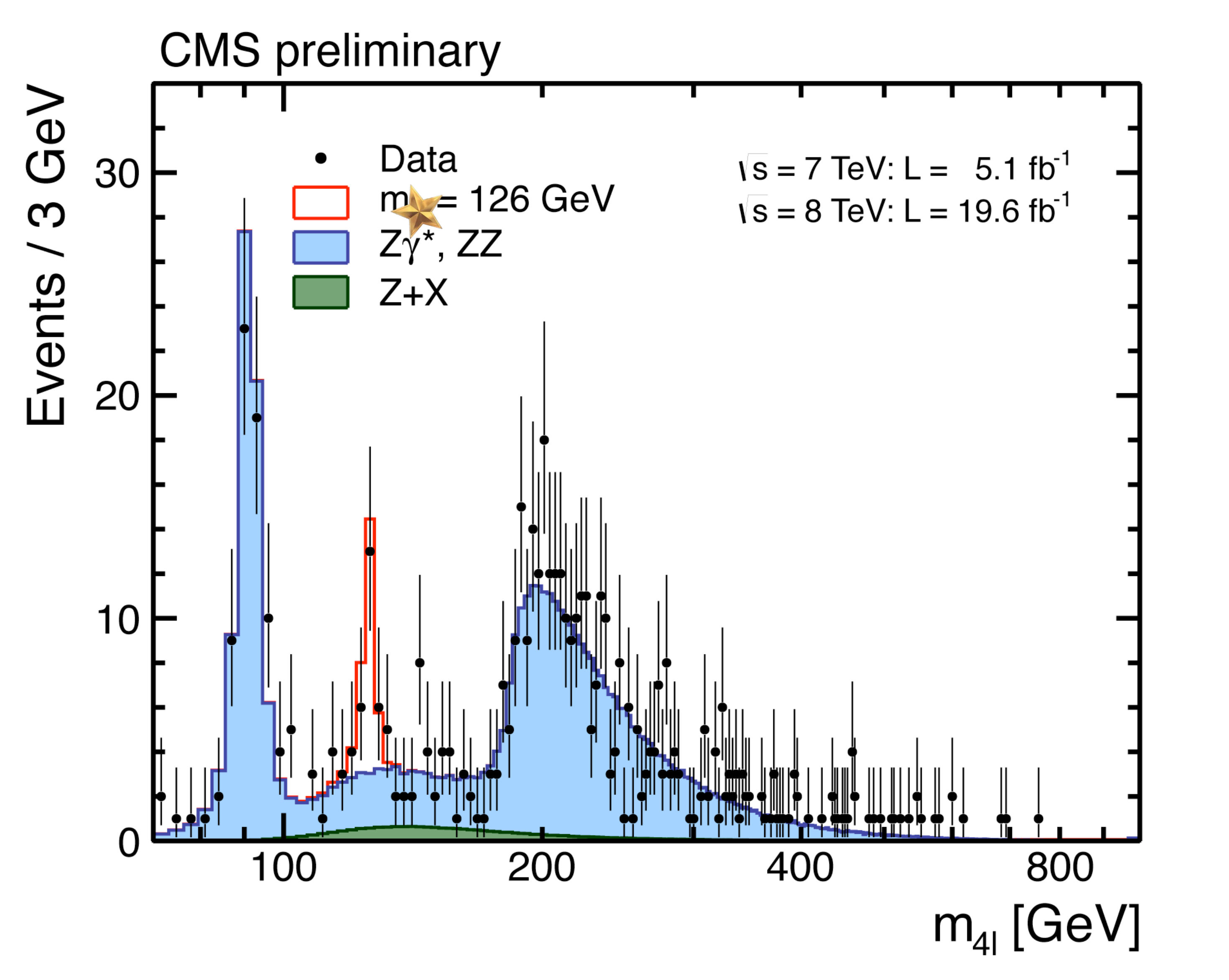
Monte Carlo Simulation follows the evolution of physics processes from collision to digital signals

Reconstruction "goes back in time" from digital signals to the original particles produced in the collision

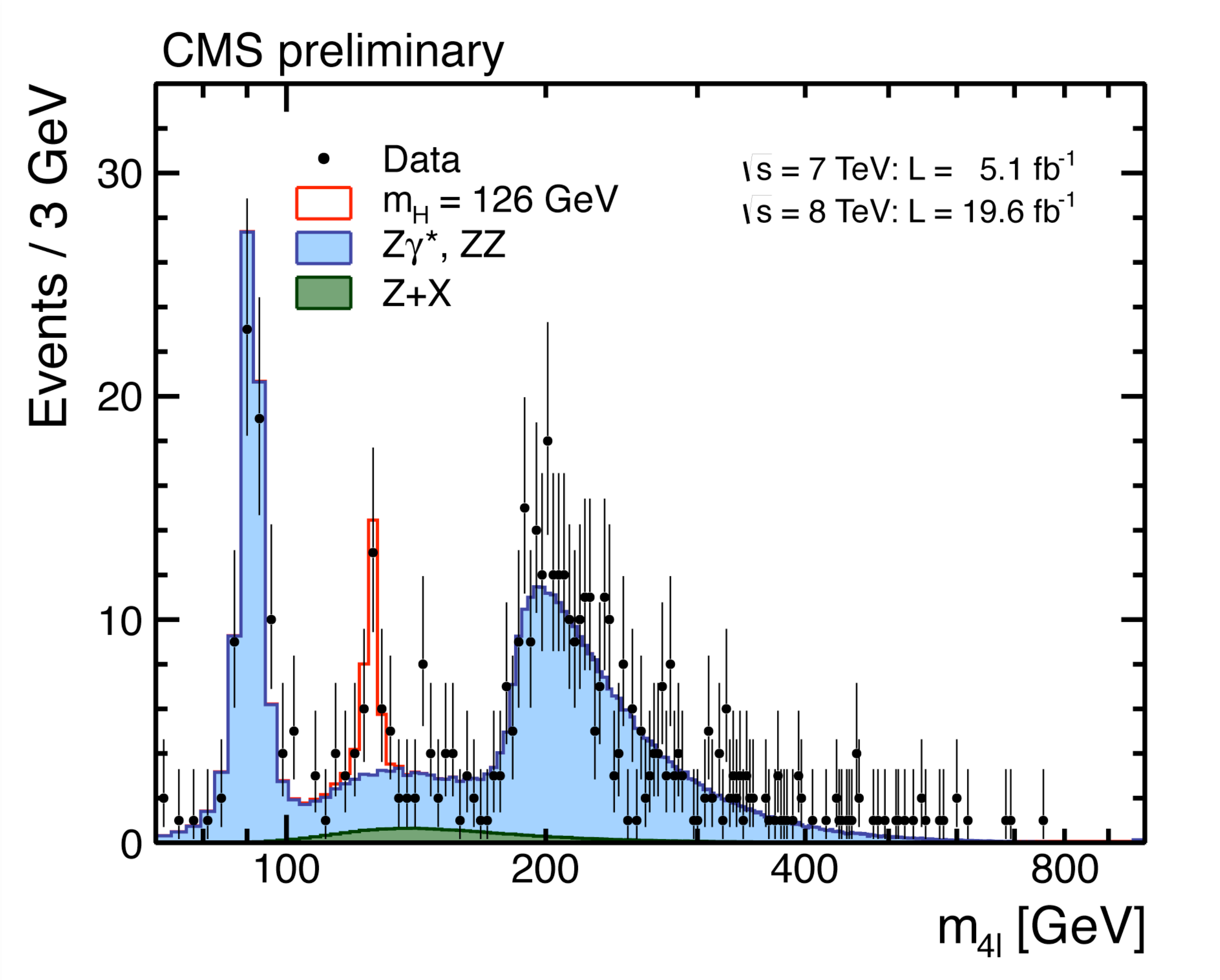


Analysis compares (at statistical level) reconstructed events from real data with those from simulation

A well known example



A well known example



Analogies with Industry

- **Signal/image processing**
  - DAC (including calibrations)
  - Pattern recognition, "clustering"
- **Topological problems**
  - Closest neighbour, minimum path, space partitioning
- **Navigation/Avionics (Kalman filtering)**
  - Tracking in a force field in presence of "noise"
  - Trajectory identification and prediction
- **Gaming**
  - "walk-through" complex 3D geometries
  - Detection of "collisions"

We are not alone and we should always look from inspiration outside!

Floating point in HEP algorithms

Accuracy and Precision

- Measurements themselves require modest precision (16,24 bits)
  - Originally they were output of electronic frontends
- Geometry/Materials often known at per-cent level
  - Cross section of reactions for simulation not rarely at ~10% level (e.g. hadronic)

BUT

- Dynamic range, when converted in natural units, often requires a high precision FP representation
  - Energy range from hundreds of KeV to hundreds of GeV: $>10^9$!
  - Position: µm over 20m (precise silicon tracker/detector length)
- Many conversions back and forth various coordinate/measurement systems
- Uncertainties manipulation (including correlations)
  - Squared quantities: each transformation requires two matrix multiplications

FP Operations and their Costs

op	instruction	sse s	sse d	avx s	avx d
+,-	ADD,SUB	3	3	3	3
== < >	COMISS CMP..	2,3	2,3	2,3	2,3
f=d d=f	CVT..	3	3	4	4
|,&,^	AND,OR	1	1	1	1
*	MUL	5	5	5	5
/,sqrt	DIV, SQRT	10-14	10-22	21-29	21-45
1.f/ , 1.f/sqrt	RCP, RSQRT	5		7	
=	MOV	1,3,...	1,3,...	1,4,....	1,4,...

Typical Applications of FP Operations

- Signal calibration
  - Ideal for vectorisation (more about this later)
    - Unfortunately lookups to calib constants required ☹
    - Calib params may depend on "reconstructed quantities"
- "Geometry" transformations
  - Trigonometry (also log/exp – e.g. physicists like pseudo-rapidity)
  - Small matrices (max 5x5, 6x6)
- Translation of formulas from literature (include all sorts of mathematical functions)
  - Energy losses, scattering

A typical profile: Low Level

CMS reconstruction, spotlight on μ−operations

CPI (cycle per instruction): 0.964

load instructions %: 30.58%
store instructions %: 13.74%
branch instructions % (approx): 17.06%
resource stalls % (of cycles): 30.63%
divider busy % (of cycles): 12.11%
% of branch instr. mispredicted: 2.25%
% of L3 loads missed: 2.09%

% of SIMD in all uops: 19.22%

breakdown:	%of all uops	% of all SIMD
PACKED_DOUBLE:	0.663%	3.449%
PACKED_SINGLE:	0.613%	3.190%
SCALAR_DOUBLE:	**13.485%**	70.159%
SCALAR_SINGLE:	4.038%	21.010%

- Tons of loads/stores
- Divisions are evil for CPUs
- Extensive usage of doubles (only partially justified)
- Very little vectorisation!

CMS simulation at 8 TeV

Obtained with IgProf
http://igprof.org

Total %	Self	Symbol name
4.98	36.22	G4Mag_UsualEqRhs::EvaluateRhsGivenB(double const*, dc
3.12	22.67	G4PhysicsVector::Value(double, unsigned long&) const
3.01	21.93	G4hPairProductionModel::ComputeDMicroscopicCrossSecti
2.45	17.86	G4ClassicalRK4::DumbStepper(double const*, double con
2.37	17.27	G4Navigator::LocateGlobalPointAndSetup(CLHEP::Hep3Vec
2.21	16.10	__ieee754_exp
2.17	15.83	G4PolyconeSide::DistanceAway(CLHEP::Hep3Vector const&
1.93	14.03	_init
1.83	13.32	sim::Field::GetFieldValue(double const*, double*) con
1.30	9.44	G4ElasticHadrNucleusHE::HadrNucDifferCrSec(int, int,
1.25	9.12	G4UniversalFluctuation::SampleFluctuations(G4Material
1.25	9.07	__ieee754_atan2
1.22	8.88	G4PropagatorInField::ComputeStep(G4FieldTrack&, doub
1.18	8.60	G4VEmProcess::PostStepGetPhysicalInteractionLength(G
1.18	8.55	G4VoxelNavigation::ComputeStep(CLHEP::Hep3Vector con
1.11	8.08	G4MagInt_Driver::QuickAdvance(G4FieldTrack&, double c
1.11	8.07	G4MuPairProductionModel::ComputeDMicroscopicCrossSect
1.07	7.77	G4SteppingManager::DefinePhysicalStepLength()

Cut at 1% of the total runtime

- No major offender
- Mathematical functions: clearly visible

CMS performance optimisations may have made this measurement not actual

How to Improve: Precision, Accuracy

- Double precision often required to keep under control coordinate system transformations (in particular for the error matrices)
  - Develop more robust algorithms
  - Avoid back&forth
  - Choose (dynamically?) units (metrics) to avoid too large dynamic-ranges
- Tune precision to the required accuracy in parameterization
  - Use a math-lib allowing control of precision

How Can We Improve: Math Lib

- Cost of a sin/cos/exp high and includes overhead of an indirect function call
  - **Inline math functions**
    - Help vectorisation too
- Choice of the "right" precision
- **Architecture specific implementation**
- Significant time spent in range reductions and limit/ exceptions checking/setting
  - Our angles are ALL in [-pi,pi] range (sometime less)
  - Arguments of log/exp often in a limited range
    - **Special version for reduced ranges**

How Can We Improve: Math Lib

With some exceptions, the default mathematical library used for HEP calculations is **Libm** (glibc implementation)

Running on linux powered machines

- **A rock-solid reference!**
- **Always focussed on accuracy rather than performance**
- **Not architecture specific, no limited ranges, no inlining, one implementation only**

A Selection of Alternatives

Different products are available, for example:

- Intel's SVML, IMF, MKL (commercial)
- AMD Libm (free, closed source)
- VDT (VectoriseD maTh: free and open source)
- Yeppp… any other?

Differences in the implementations but common underlying principle:

Trade off between accuracy and speed of execution

What is VDT?

- An **open source** math library library, LGPL3 licence
- Inspired by the good old Cephes (and videogames)
- Single/Double precision of (a)sin, (a)cos, sincos, (a)tan, atan(2), log, exp and 1/sqrt
- **Fast, approximate, inline**
- Symbols names are different from traditional ones: vdt::fast_<name>
  - Do not force drop-in replacement, allow full control
- **Functions usable in autovectorised loops**
  - **Array signatures** available: calculate on multiple elements conveniently
- **C++ code only,** no intrinsics: **portability guaranteed**
  - The compiler adapts the code to the target architecture
  - ARM, x86, GPGPUs, Xeon Phi, <future microarchitecture>

https://svnweb.cern.ch/trac/vdt

Padé Approximants

- VDT (and Cephes) double precision functions: Padé Approximants
- Single Precision: polynomials

The “best” **approximation of a function by a rational function** of a given order → Better approximation than a truncated Taylor series

Padé approximantimant of *f(x)* of order [m/n] is the function

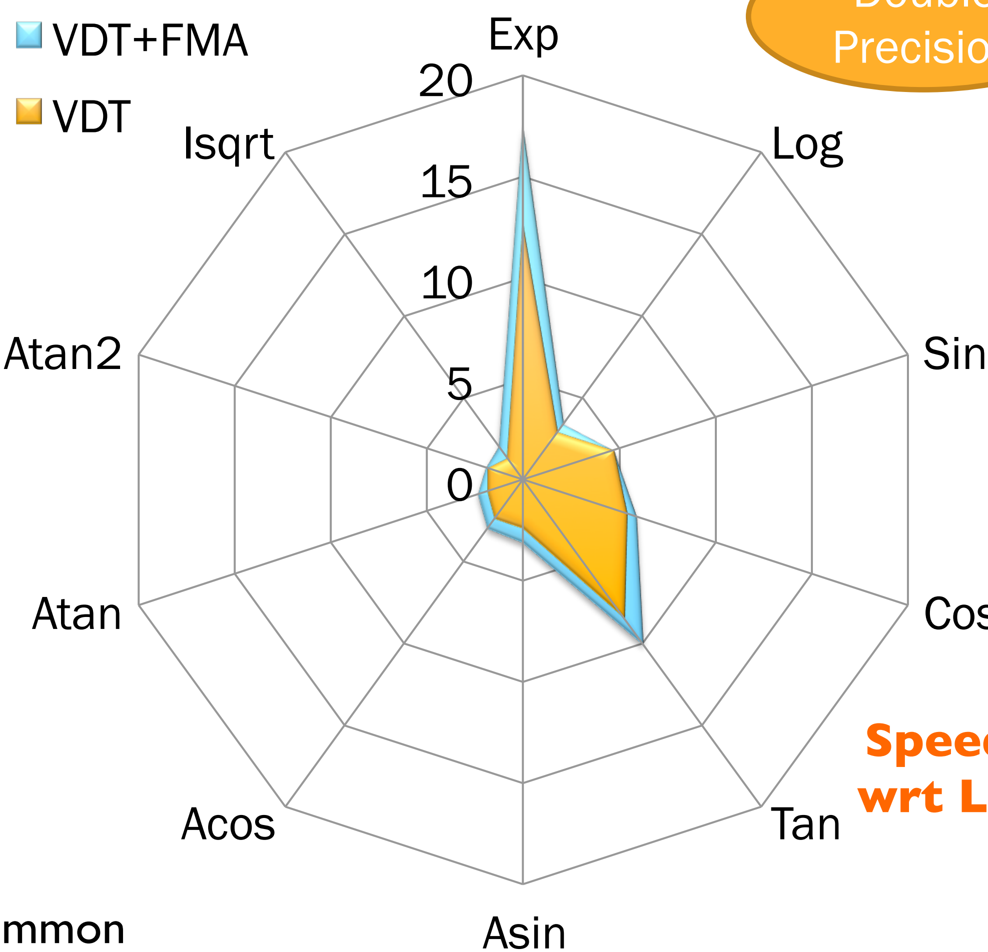
$$R(x) = \frac{\sum_{j=0}^{m} a_j x^j}{1 + \sum_{k=1}^{n} b_k x^k} = \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m}{1 + b_1 x + b_2 x^2 + \cdots + b_n x^n}$$

Speed: VDT Vs Libm

Fnc.	Libm	VDT	VDT-FMA
Exp	102	8	5.8
Log	33.3	11.5	9.8
Sin	77.8	16.5	16.5
Cos	77.6	14.4	13.2
Tan	89.7	10.6	8.9
Asin	21.3	8.9	6.9
Acos	21.6	9.1	7.3
Atan	15.6	8.4	6.7
Atan2	36.4	19.9	18.9
Isqrt	5.7	4.3	2.8

Time in **ns** per value calculated

Double Precision



Speedup wrt Libm

FMA: Fused Multiply Add $d = a + b \times c$

- Operative input range: [-5k, 5k]
- **Speedup factors of >5** not uncommon
- **Effect of FMA clearly visible**
  - **A waste not to profit from it!**

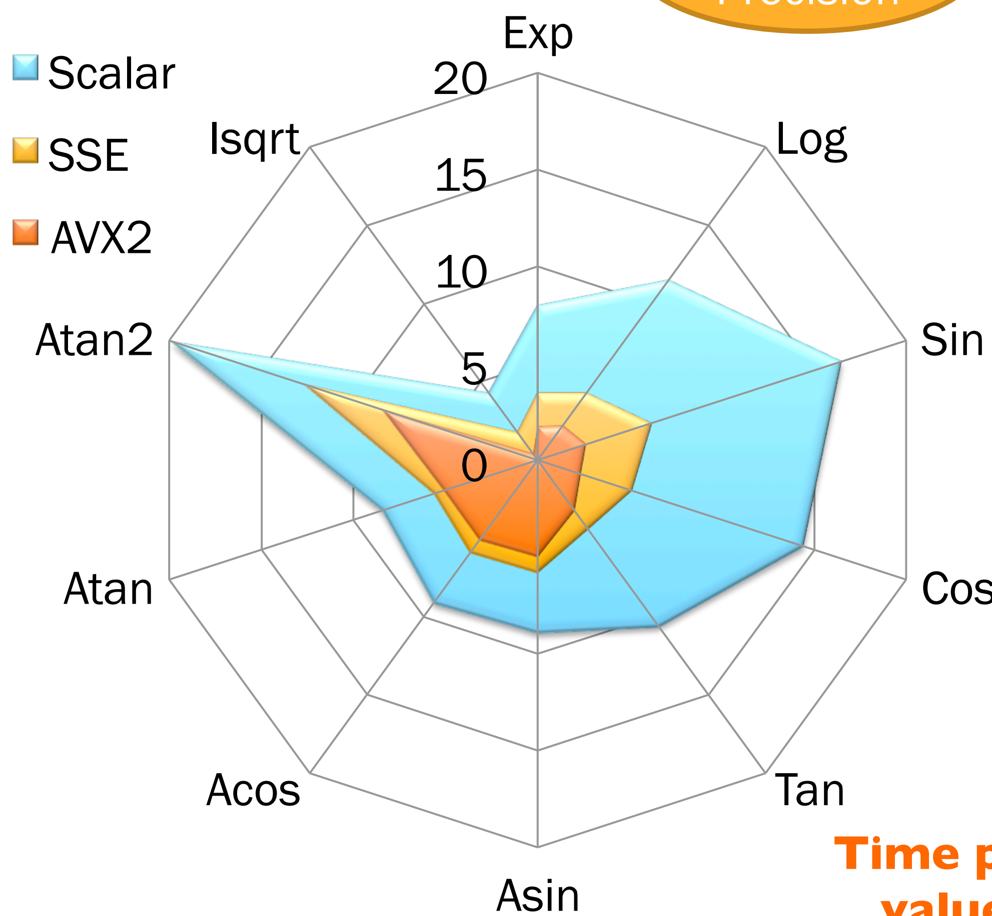
Testbed:
SLC6-GCC48, i7-4770K at 3.50GHz Haswell
glibc 2.12-1.107.el6_4.4 and VDT v0.3.6

Speed: VDT Vectorisation

Fnc.	Scalar	SSE	AVX2
Exp	8	3.5	1.7
Log	11.5	4.3	2.2
Sin	16.5	6.2	2.6
Cos	14.4	5.1	2.3
Tan	10.6	4.4	3.2
Asin	8.9	5.8	5
Acos	9.1	5.9	5.1
Atan	8.4	5.6	5.1
Atan2	19.9	12.7	8.4
Isqrt	4.3	1.8	0.4

Time in **ns** per value calculated

Double Precision



Time per value calculated

- **Effect of vectorisation clearly visible**

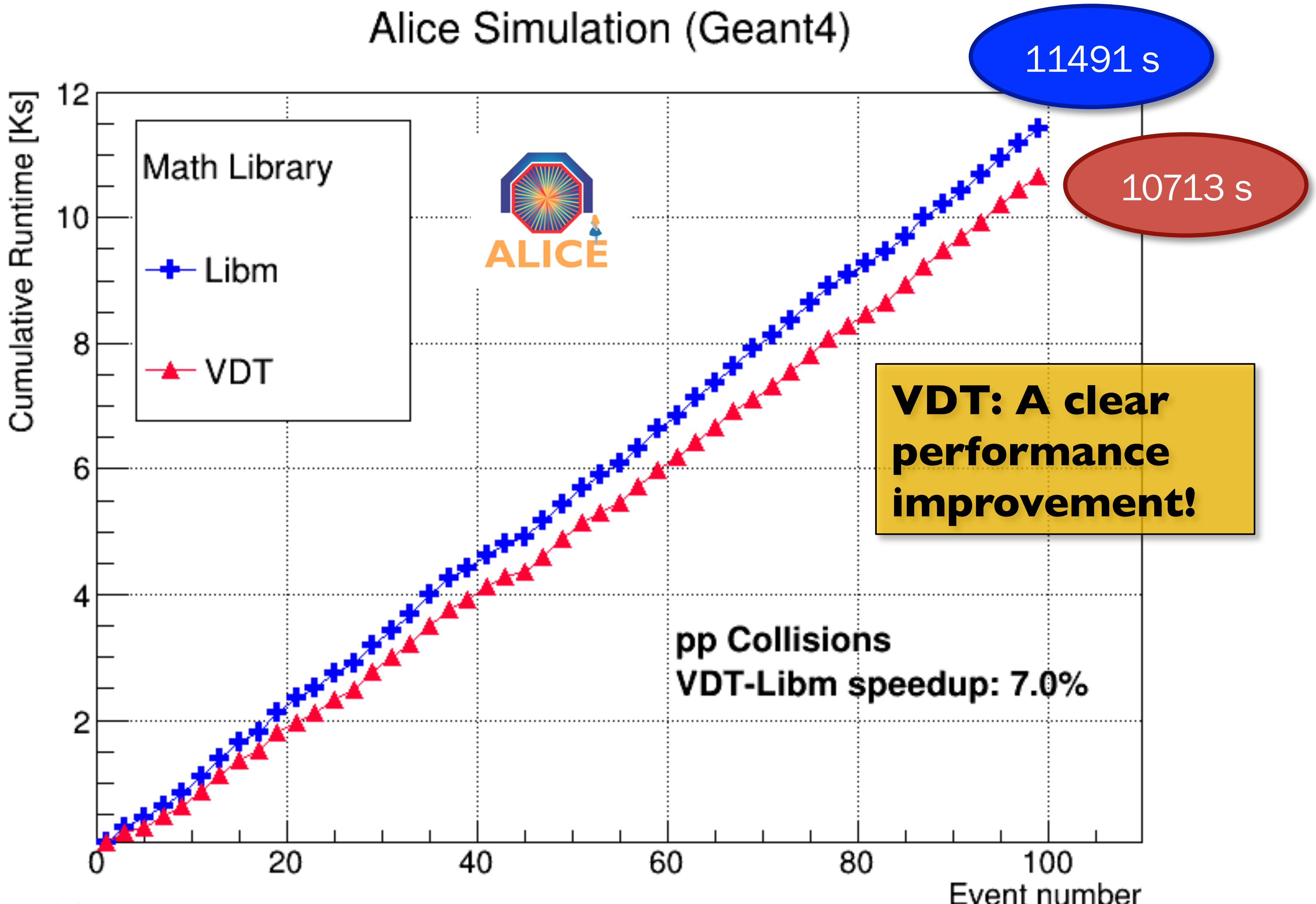
Accuracy: An Example

- Accuracy was measured comparing the results of **Libm and VDT bit by bit with the same input**
- **Differences quoted in terms of most significant different bit**
- In the end they are just 32 (64) bits which are properly interpreted (sign, exponent, mantissa)!

Double Precision	MAX VDT	AVG VDT
Exp	2	0.14
Log	2	0.42
Sin	2	0.25
Cos	2	0.25
Tan	2	0.35
Asin	2	0.32
Acos	8	0.39
Atan	1	0.33
Atan2	2	0.27
Isqrt	2	0.45

Only slight difference present: already enough for many applications

Alice Simulation: Switching to VDT



Courtesy of S. Wenzel

Being More Ambitious

- Can a “traditional” mathematical library be our best solution?
- What about a veritable “MetaLibM”?
  - Automatic generation of functions’ code
  - Platform specific implementation
  - Limitation in range
  - Choice of precision
- Specific approximations (polynomial/Pade) of full formulas?

How to Improve: Concrete example

Multiple scattering algorithm in CMS

```
double ms(double radLen, double m2, double p2) {
  constexpr double amscon = 1.8496e-4;    // (13.6MeV)**2
  double e2      = p2 + m2;
  double beta2  = p2/e2;
  double fact = 1.f + 0.038f*log(radLen);  fact *=fact;
  double a = fact/(beta2*p2);
  return amscon*radLen*a;
}
```

Already an approximation

Material density, thickness, track angle Known at percent?

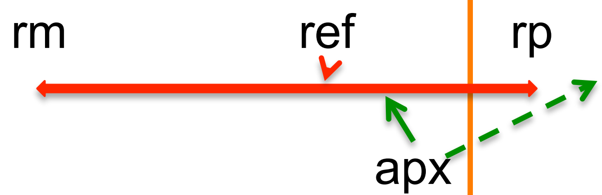
```
float msf(float radLen, float m2, float p2) {
  constexpr float amscon = 1.8496e-4;    // (13.6MeV)**2
  float e2      = p2 + m2;

  float fact = 1.f + 0.038f*dirtylogf<2>(radLen); fact /= p2;
  fact *=fact;
  float a = e2*fact;
  return amscon*radLen*a;
}
```

2nd order polynomial by FdD

Exciting times for curious physicists {or,and} programmers: **a single person can make the difference**

The Accuracy of this Approximation




```
float ref = ms(rl,m2,p2);
float rp = ms(rl*1.001,m2,p2);   // 0.1% positive
float rm = ms(rl*0.999,m2,p2);  // 0.1% negative
float apx = msf(rl,m2,p2);   // fast approximation

// look if approximation inside uncertainty-interval
int dd = std::min(abs(diff(rm,ref)),abs(diff(rp,ref)));
dd -= abs(diff(apx,ref)); // negative if apx-ref is larger than the uncer-interval
dm = std::min(dm,dd);

da = std::max(da,abs(diff(apx,ref)));  // maximum “error” by approx
di = std::max(di,abs(diff(rp,ref)));
di = std::max(di,abs(diff(rm,ref)));    // maximum uncertantly
// ditto for minimum
```

diff is in “bits”

- 0.1% accuracy corresponds to a difference of 13-14 bits
- **Maximum error of the approximation is ~12 bits**
- “dm” always positive

How to Improve: Simulation

```
G4double
G4HadronCrossSections::GetCaptureCrossSection(const
G4DynamicParticle* aParticle, G4int ZZ)
 { [...]
  G4double ekx = std::max(ek, 1.e-9);
  if (ekx != lastEkx) {
   lastEkx = ekx;
   lastEkxPower = std::pow(ekx*1.e6, 0.577); }

  G4int izno = ZZ;
  if (izno > 100) izno = 100;     // Not in GHESIG
 izno = izno - 1;     // For array indexing
 G4double sigcap = 11.12*cscap[izno]/lastEkxPower;

sigcap = sigcap*millibarn;
  return sigcap; }
```

Argument of pow is at most 1e3.

Probably double precision is not needed.

Look-up table

Modernisation!

- Many algorithms coded in the '80 (even '70)
- Programmer's heuristics still based on x87 math and sequential processing
- Advent of "extreme" architectures (GPUs etc) is an opportunity to modernize algorithms for ALL architectures!

A paradigm shift?

From the CERNLIB manual

Title of program: VAVILOV

Catalogue number: AAUJ

Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland (see application form in this issue)

Computer: CDC 6600; *Installation*: CERN, Geneva

Operating system: CDC Scope

Programming language used: FORTRAN IV

High speed storage required: 3246 words

No. of bits in a word: 60

Overlay structure: None

No. of magnetic tapes required: None

Other peripherals used: Card reader, pine printer

No. of cards in combined program and test deck: 636

Card punching code: BCD

Keywords: Nuclear, Vavilov distribution, energy loss, thin absorber, random number generation.

Floating point in HEP data

HEP Data Structures

- High granularity “naïve” object model
  - Innermost loop often not the longest!
- Fragmentation in several libraries (plugin model)
  - Link time optimisation does not help
- “Linear thinking” conditional code

Vectorisation possible only with proper layouts in memory

- Only a massive redesign of data-structures (and not only algorithms) can make vectorisation effective
  - Not alone: see
    - http://research.scee.net/files/presentations/gcapaustralia09/Pitfalls_of_Object_Oriented_Programming_GCAP_09.pdf
    - http://www.slideshare.net/DICEStudio/introduction-to-data-oriented-design

OO Pitfalls

We moved all of the HEP code from FORTRAN to C++.

Now, are objects good?

- Well, yes
- And no

Keyword: Data Oriented Design (re-design?)

Almost copied from Tony Albrecht: Pitfalls of Object Oriented Programming (see previous slide)

I/O on disk

- Reduce precision within calculations requires in-depth studies
- What about persistent representation of data structures (e.g. data on disk) ?
  - Maintain a full precision reference
  - Can we reduce precision of some data formats (e.g. analysis?)
  - Responsibility of the toolkit used for I/O
- Existing example: Alice AOD data & ROOT
  - Massive usage of Double32_t opaque typedefs
  - Reduced precision on disk (e.g. float) but double in memory!

Take Away Message

- FP: big weight in HEP calculations (~20% of reconstruction)
  - Mostly double: for no good reason sometimes?
  - Not easy to vectorise as it stands
  - Large use of std math-functions
    - glibm: excellent reference, overkill for many applications?
- Opportunities for improvements
  - Data Oriented (re-)Design
  - Use parameterizations also for non-elementary functions
  - Use fast (less precise, limited-range) math-functions
    - Plenty of appealing alternatives available!
  - Use metrics allowing the use of floats
  - Systematically verify required accuracy
    - Face the algorithms: **you** can make the difference!
  - Save disks/tapes: reduced precision in data persisted for analysis

Backup

Speed: VDT On ARM

Double Precision

Fnc.	Libm	VDT
Exp	155	71.4
Log	153	64.6
Sin	202	57.9
Cos	199	54.9
Tan	290	96.4
Asin	99.2	77.9
Acos	95.4	78.9
Atan	127	75.4
Atan2	187	89.7
Isqrt	24.7	52.0

Time in **ns** per value calculated

- ARM Cortex A9, arm-v7 Odroid
- **VDT: Portable and very convenient**
- **Simple implementation pays also on a simple architecture!**