Floating Point in Experimental HEP Data Processing

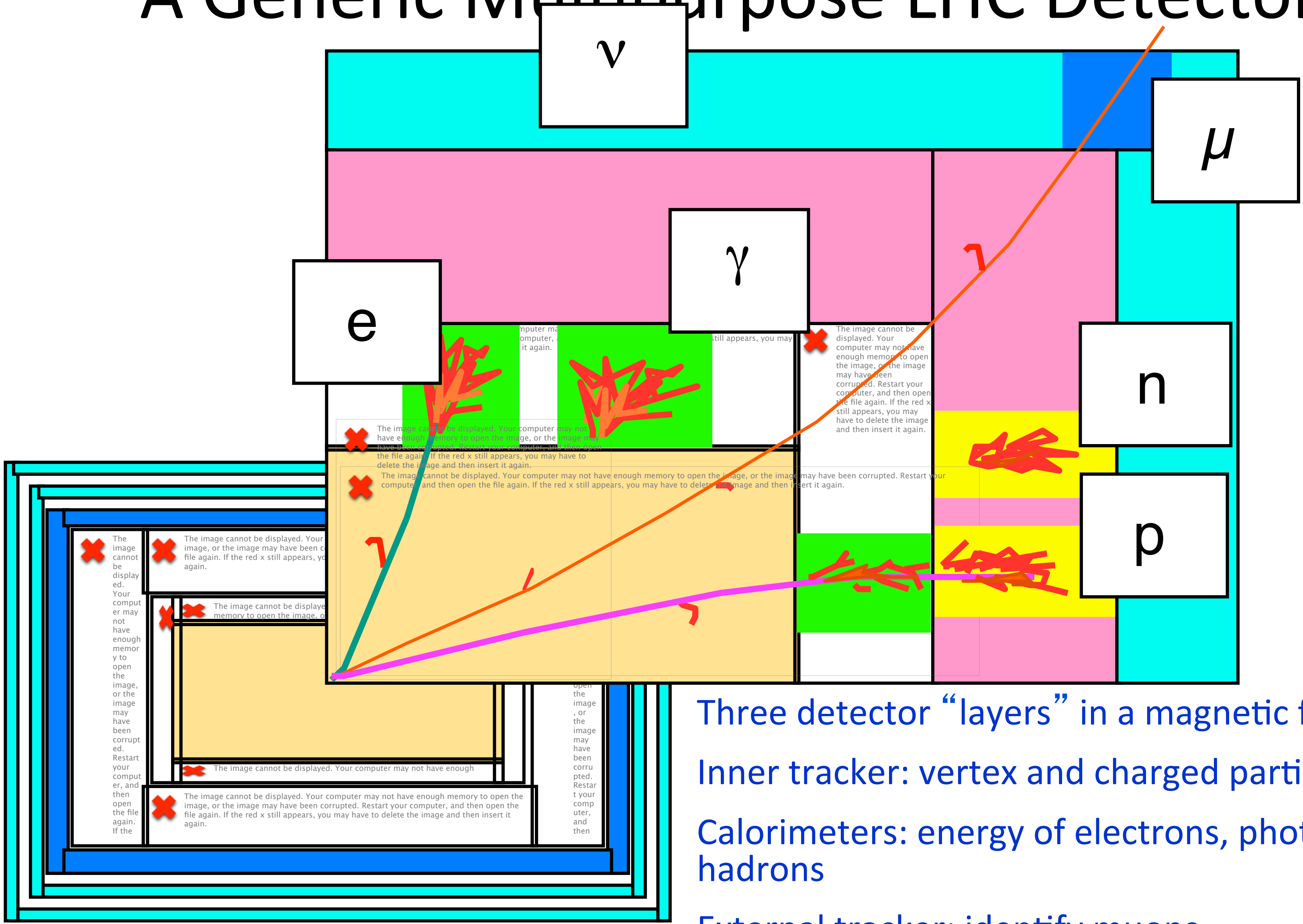
(aka Reconstruction)

Vincenzo Innocente

CERN

PH/SFT & CMS

A Generic Multipurpose LHC Detector



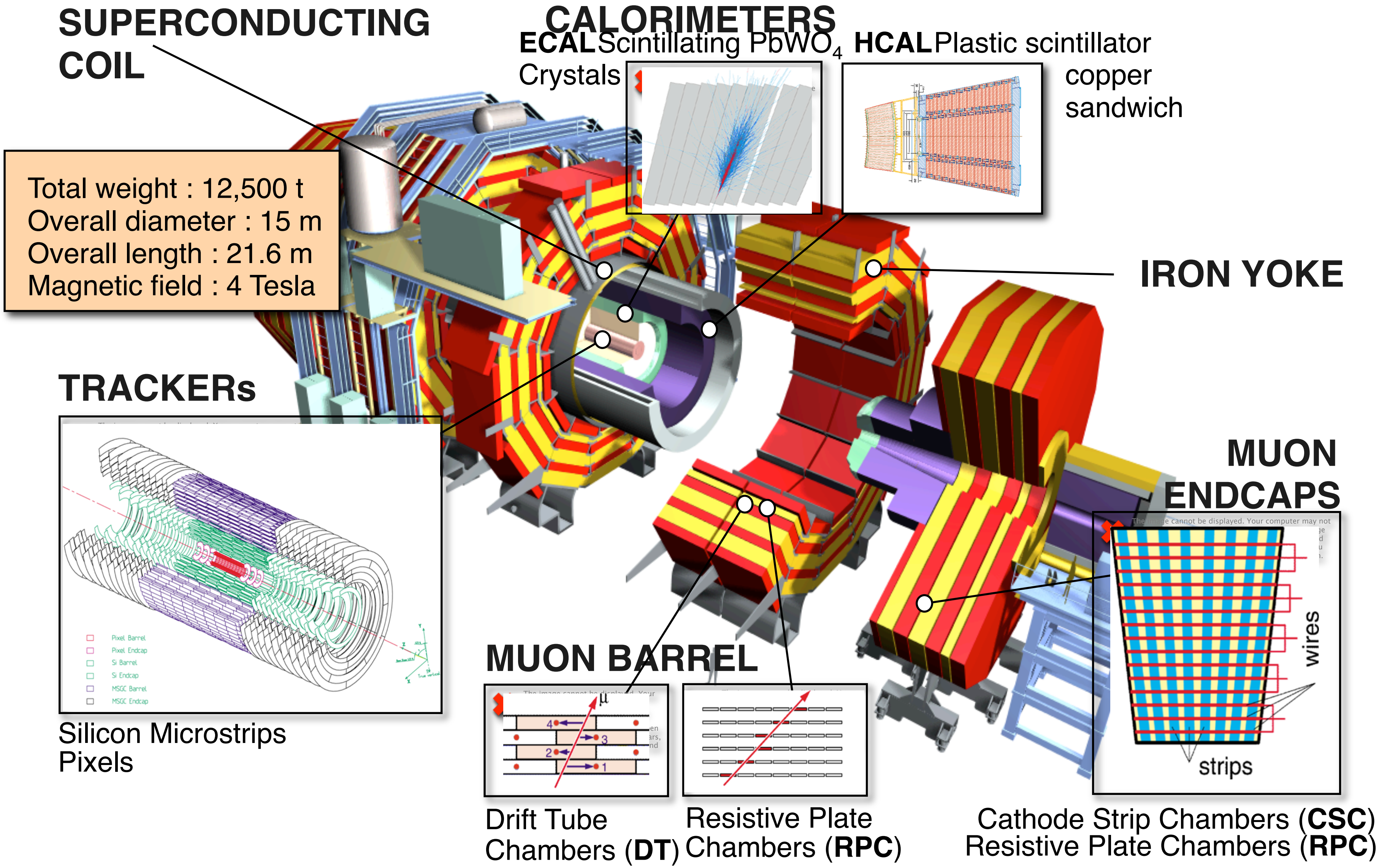
Three detector "layers" in a magnetic field

Inner tracker: vertex and charged particles

Calorimeters: energy of electrons, photons, hadrons

External tracker: identify muons

An experiment: CMS

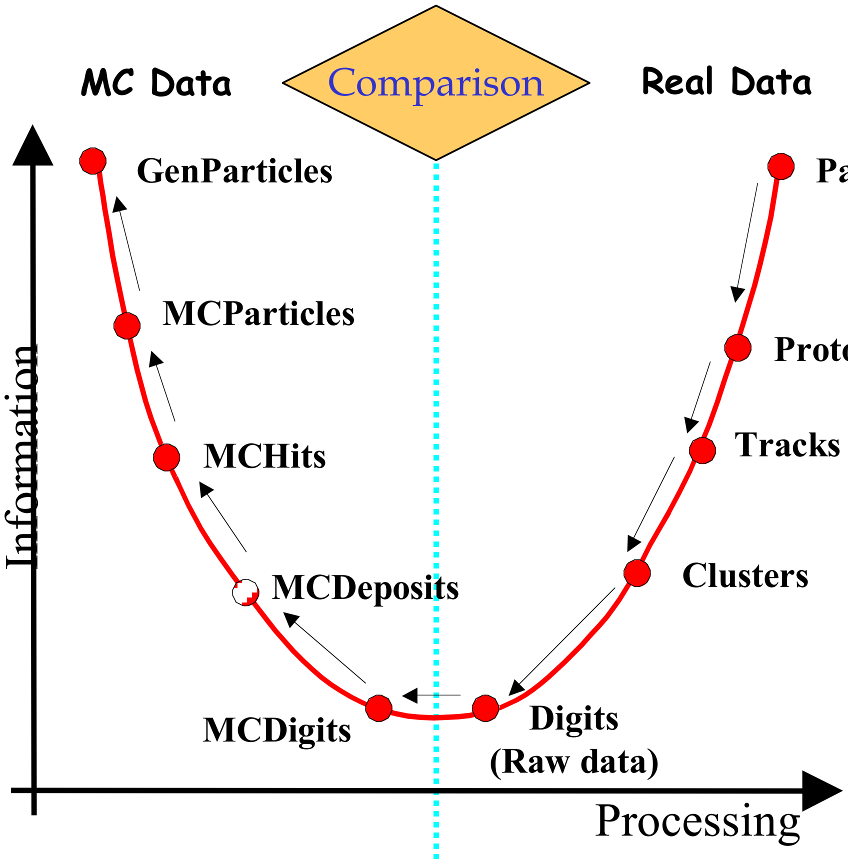


Data and Algorithms

- HEP main data are organized in *Events* (particle collisions)
- Simulation, Reconstruction and Analysis programs process "one Event at the time"
  - Events are fairly independent of each other
  - Trivial parallel processing
- Event processing programs are composed of a number of Algorithms selecting and transforming "raw" Event data into "processed" (reconstructed) Event data and statistics
  - Algorithms are mainly developed by "Physicists"
  - Algorithms may require additional "detector conditions" data (e.g. calibrations, geometry, environmental parameters, etc.)
  - Statistical data (histograms, distributions, etc.) are typically the final data processing results

High Energy Analysis Model

MonteCarlo 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

Data Hierarchy

"RAW, ESD, AOD, TAG"

~2 MB/event	RAW	Triggered events recorded by DAQ	Detector digitisation
~100 kB/event	ESD/RECO	Reconstructed information	Pseudo-physical information: Clusters, track candidates
~10 kB/event	AOD	Analysis information	Physical information: Transverse momentum, Association of particles, jets, (best) id of particles,
~1 kB/event	TAG	Classification information	Relevant information for fast event selection

Analogies with Industry

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

Accuracy, Precision

- Measurement themselves require a modest precision (16,24 bits)
- Geometry/Materials often known at per-cent level
- Dynamic range, when converted in natural units, often requires a high precision FP representation
  - Enengy range $>10^9$
  - Position: micron over 20m
- Many conversions back and forth various coordinate/measurement systems
- Error manipulation (including correlations)
  - Squared quantities: each transformation requires two matrix multiplications

FP operations in reconstruction

- Signal calibration
  - Ideal for vectorization
    - (if was not that calib requires lookup!)
    - Calib-params may depend on “reconstructed quantities”
- “Geometry” transformation
  - Trigonometry (also log/exp!)
  - Small matrices (max 5x5, 6x6)
- Many logs, exp coming from parameterizations

Vectorization?

- Current code design and implementation often hinder vectorization
  - High granularity "naïve" object model
  - Fragmentation in several libraries (plugin model)
    - lto will not help
  - "Linear thinking" conditional code
- Only a massive redesign of data-structures and algorithms will make vectorization 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

Typical Profile (today)

CPI (cycle per instruction): 0.9636
load instructions %: 30.577%
store instructions %: 13.737%
load and store instructions %: 44.314%
resource stalls % (of cycles): 30.631%
branch instructions % (approx): 17.065%
% of branch instr. mispredicted: 2.247%
% of L3 loads missed: 2.087%
computational x87 instr. %: 0.038%

% of SIMD in all uops: 19.22%
% of comp. SIMD in all uops: 10.17%

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%
VECTOR_INTEGER:	0.421%	2.192%

More details (see next page):
Function where time is spent most

- *No hot-spot: top 30 each between 2.5% and 0.5% of total*
- Trig/trans functions
- div/sqrt latency

BR_INST_EXEC.INDIRECT_NON_CALL		UOPS_RETIRED.STALL_CYCLES		ARITH.CYCLES_DIV_BUSY		Function
9.5e+07	5.30 %	8.1e+09	41.41 %	2e+09	10.07 %	__ieee754_exp
3.5e+08	13.71 %	8.1e+09	45.49 %	0	0.00 %	arena_malloc_small
6.7e+06	0.23 %	7.5e+09	47.55 %	3.8e+09	24.31 %	__ieee754_atan2
6.6e+07	46.92 %	9.9e+09	63.11 %	4.2e+09	26.82 %	void TkGluedMeasurementDet::doubleMatch< ...
1.9e+08	15.15 %	4.9e+09	33.67 %	0	0.00 %	arena_dalloc_bin
1.4e+08	7.66 %	9.6e+09	68.94 %	5.9e+09	42.28 %	ThirdHitPredictionFromCircle::phi(double ...
3.4e+07	1.05 %	6e+09	43.11 %	3.6e+09	25.47 %	atanf
3.9e+08	17.85 %	7.8e+09	58.89 %	0	0.00 %	free
4.4e+07	2.68 %	8.5e+09	65.22 %	2.4e+09	18.60 %	__ieee754_acos
2.5e+07	2.56 %	4.3e+09	34.11 %	1.1e+08	0.90 %	ROOT::Math::SMatrix<double, (unsigned in ...
1.1e+07	11.71 %	4.4e+09	41.21 %	0	0.00 %	cms::TrackListMerger::produce(edm::Event ...
8.5e+07	204.00 %	8.6e+09	81.25 %	4.2e+09	39.96 %	magfieldparam::TkBfield::Bcyl(double, do ...
6.2e+06	0.59 %	4.6e+09	46.46 %	5.6e+08	5.70 %	__ieee754_log
1.7e+06	0.99 %	4.9e+09	53.99 %	5.6e+07	0.61 %	<unknown(s)>
1.8e+08	7.49 %	5.1e+09	59.85 %	2.8e+07	0.33 %	strcmp
2.6e+08	20.20 %	5.5e+09	67.64 %	2.6e+09	32.26 %	PixelTripletLargeTipGenerator::hitTriple ...
0	0.00 %	4.3e+09	57.80 %	1.1e+08	1.51 %	do_lookup_x
9.3e+07	11.99 %	4.9e+09	66.54 %	3.9e+09	53.23 %	DAClusterizerInZ::update(double, std::ve ...
3.4e+07	11.88 %	3.5e+09	48.00 %	3.1e+08	4.22 %	sincos
1.3e+08	24.73 %	2.5e+09	41.40 %	4.2e+08	6.82 %	PixelTripletHLTGenerator::hitTriplets(Tr ...
4.8e+07	19.87 %	4.7e+09	77.57 %	4.5e+08	7.34 %	tan
0	0.00 %	2.5e+09	45.01 %	0	0.00 %	<unknown(s)>
7.3e+07	8.77 %	2.1e+09	37.74 %	5.9e+08	10.71 %	__ieee754_atan2f
9.8e+06	5.74 %	3.9e+09	71.26 %	2e+09	37.42 %	AnalyticalCurvilinearJacobian::computeFu ...
8.4e+06	9.26 %	3.4e+09	64.46 %	1.5e+09	28.77 %	JacobianCurvilinearToLocal::JacobianCurv ...
7.3e+06	9.85 %	1.7e+09	32.66 %	0	0.00 %	SiStripRecHit2D::sharesInput(TrackingRec ...
6.7e+07	24.80 %	3.1e+09	62.12 %	1.2e+09	23.72 %	StripCPEfromTrackAngle::localParameters(...
2.4e+07	17.47 %	2.9e+09	62.58 %	7e+08	15.34 %	std::pair<bool, double> Chi2MeasurementE ...
1.6e+08	13.06 %	1.7e+09	36.84 %	0	0.00 %	arena_malloc
0	0.09 %	5.3e+08	12.62 %	0	0.00 %	PixelHitMatcher::compatibleSeeds(std::ve ...
6.6e+07	23.53 %	2.9e+09	69.80 %	2e+09	47.86 %	ThirdHitPredictionFromCircle::angle(doub ...
2.8e+05	5.50 %	1.8e+09	43.09 %	1.7e+09	41.04 %	RectangularPlaneBounds::inside(Point3DBa ...
2.8e+05	0.04 %	1.1e+09	28.79 %	0	0.00 %	inflate_fast
0	0.00 %	2.3e+09	59.12 %	0	0.00 %	fesetenv

Cost of operations (in cpu cycles)

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,...

Cost of functions (in cpu cycles i7sb)

	Gnu libm		Cephes scalar		Cephes autovect		Cephes handvect	Approx (16bits)	Intel svml		Amd libm	
	s	d	s	d	s	d	s		s	d	s	d
sin,cos large x	55 ***>500***	100	30	50	11	30	20		12	30	25	45
sincos	70		40		15		22				50	
atan2	50	100	30		13				17	52	67	87
exp	***650***	65	42	55	10	23	27		12	26	16	36
log	50	105	37	42	11	28	24	12	12	30	27	59

SET_RESTORE_ROUND_NOEXF (FE_TONEAREST);

Where/how can we improve?

- Cost of a sin/cos/exp close to div/sqrt and to the overhead of an indirect function call
  - **Inline math functions**
    - Help autovectorization too
- Trig-funs spend not negligible time in range reduction
  - Our angles are ALL in [-pi,pi] range
    - **Special version for reduced range?**
    - **Move to "fractional-pi" instead of radiant**

Where/how can we improve?

- 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
- Arguments of log/exp often in a limited range
  - Use specialized implementation
- rsqrt/rcp (+ “tunable” Newton-Raphson)
  - C-implementation in double precision faster than sse!

Example: multiple scattering

```
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?

2nd order polynomial by FdD

```
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;
}
```

Verify accuracy of 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

One More example

- In CMS the Vavilov distribution is used to compute the probability of a cluster in a Silicon Detector to come from a m.i.p.
  - It is then encoded in an 8-bit quality word
- Precision tuned-down while verifying that the final result (the 8-bits!) do not change
- Speed up of a factor 3…

Summary

- FP accounts for ~20% of HEP reconstruction
  - Mostly double (for no good reason?)
  - Not easy to vectorize as it stands
  - Large use of std math-function
- glibm: excellent full-precision reference
  - An overkill for any practical application
- Opportunities for improvements
  - Move to Data-oriented-Design
  - Reduce branches and indirect-calls
  - Use fast (less precise, limited-range) math-fun
  - Use metrics that will allow the use of floats
  - Systematically verify required accuracy