RICH Status

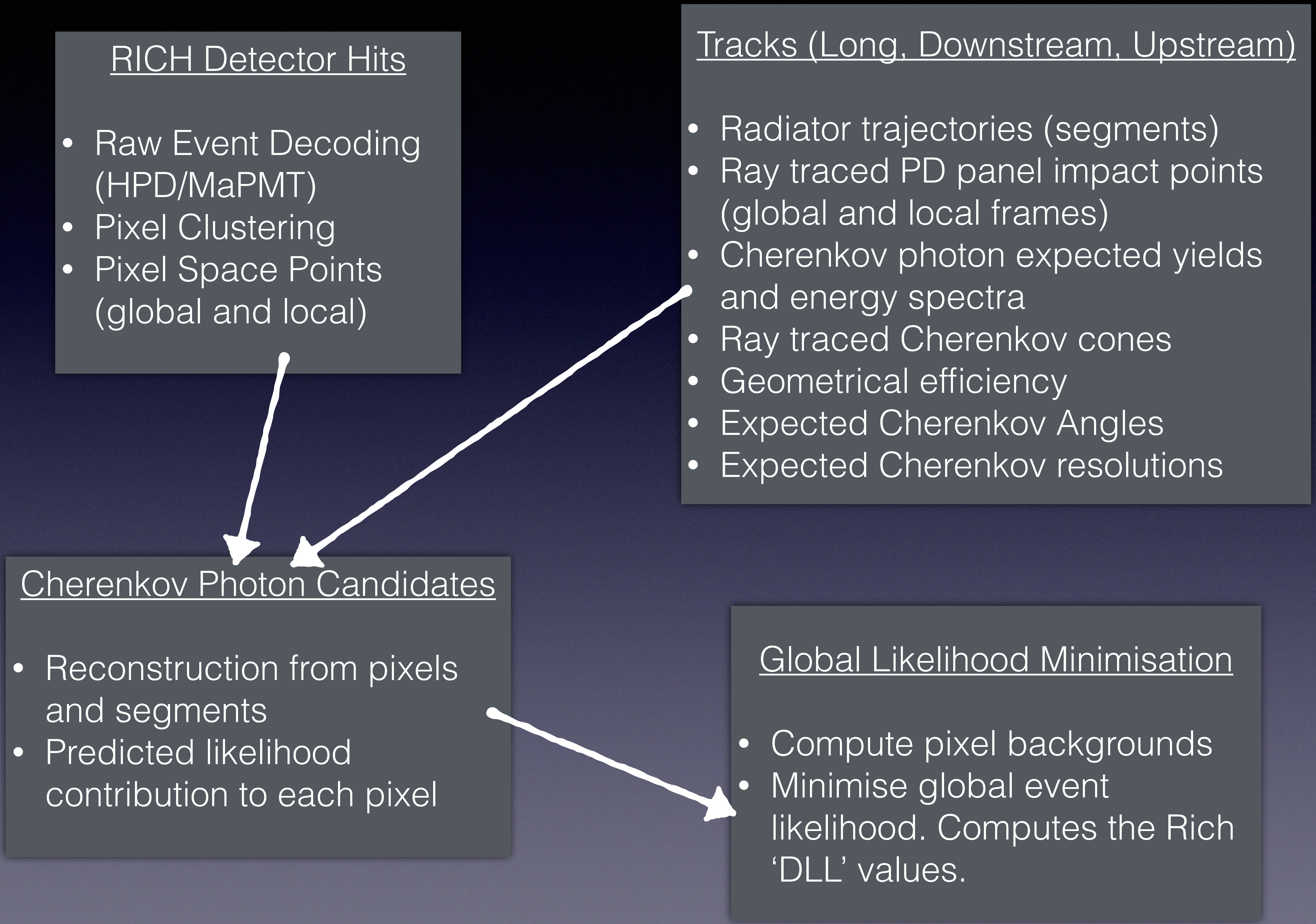
C.Jones

UNIVERSITY OF CAMBRIDGE

Introduction

- Focus on reconstruction developments.
    - Work still ongoing in the simulation, but not covered in detail here.
- RICH in the upgrade has :-
    - ***Upgraded photon detectors*** (MaPMTs replacing existing HPDs)
    - ***Updated RICH1 optics*** (re-optimisation taking into account Aerogel removal).
- ***Fundamentally though, works in the same way***. Same algorithms (currently) envisaged.
    - Have had an ‘upgrade’ sequence running based on update detector description for a long while. Formed basis of MC design work and TDR results.
- Present here recent developments using the ***upgrade Gaudi Framework***.
    - Migration to the ‘**Functional**’ framework
    - CPU improvements to the RICH photon reconstruction
    - First push towards fully utilising **SIMD vectorisation**
        - Ready now and being deployed for 2017 data taking.

RICH Reconstruction Basics



RICH Detector Hits

- Raw Event Decoding (HPD/MaPMT)
- Pixel Clustering
- Pixel Space Points (global and local)

Tracks (Long, Downstream, Upstream)

- Radiator trajectories (segments)
- Ray traced PD panel impact points (global and local frames)
- Cherenkov photon expected yields and energy spectra
- Ray traced Cherenkov cones
- Geometrical efficiency
- Expected Cherenkov Angles
- Expected Cherenkov resolutions

Cherenkov Photon Candidates

- Reconstruction from pixels and segments
- Predicted likelihood contribution to each pixel

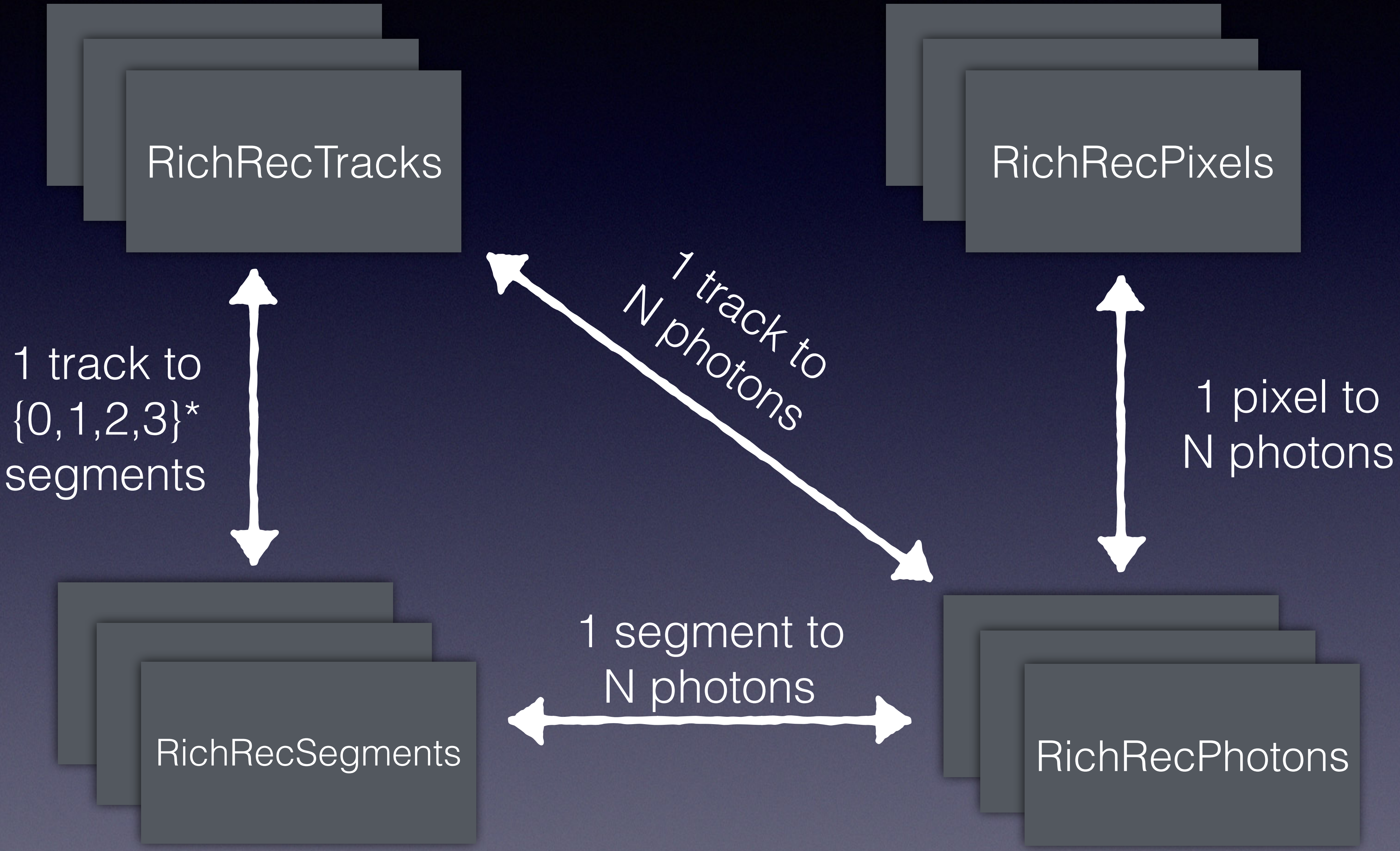
Global Likelihood Minimisation

- Compute pixel backgrounds
- Minimise global event likelihood. Computes the Rich ‘DLL’ values.

Original Software Design

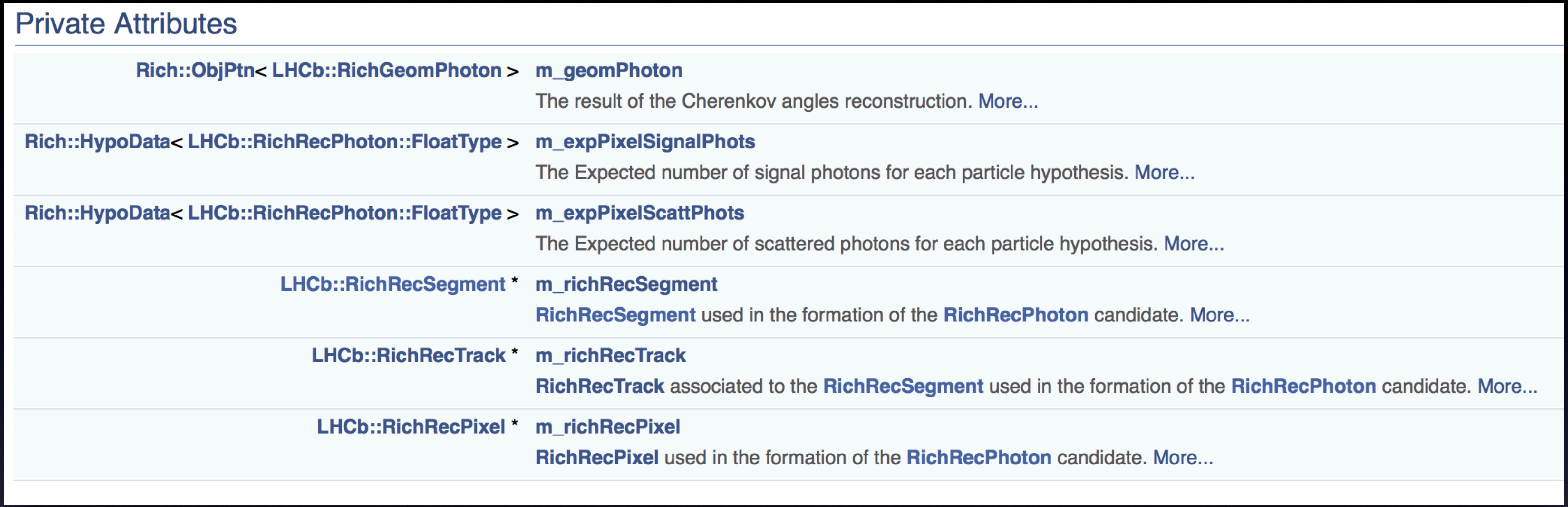
- Current C++ reconstruction dates back ***at least 10 years***.
- Designed with very different ideas in mind to now
  - ***Object oriented design*** was ***the next big thing***.
  - ***Parallel execution never entered the picture***.
  - Basic ***SIMD instructions*** may have been available, but never discussed.
- Different constraints from the framework.
  - Originally ***no python configuration***. Flat ‘job options’ files. Very limited.
- Lead to a software model which has served well, but is now showing its age.
- Two main parts to RICH design :-
  - Reconstruction event model. **Tracks**, **Segments**, **Pixels** and **Photons**.
  - **Data provider tools**. Event model object cache data and tool compute and fill on-demand (‘lazy’ evaluation model).

“RichRecEvent”



* Only {0,1,2} since Run2 due to Aerogel removal

Data Cache Example - ‘RichRecPhoton’



Private Attributes

Rich::ObjPtn< LHCb::RichGeomPhoton >	**m_geomPhoton** The result of the Cherenkov angles reconstruction. More...
Rich::HypoData< LHCb::RichRecPhoton::FloatType >	**m_expPixelSignalPhots** The Expected number of signal photons for each particle hypothesis. More...
Rich::HypoData< LHCb::RichRecPhoton::FloatType >	**m_expPixelScattPhots** The Expected number of scattered photons for each particle hypothesis. More...
LHCb::RichRecSegment *	**m_richRecSegment** RichRecSegment used in the formation of the RichRecPhoton candidate. More...
LHCb::RichRecTrack *	**m_richRecTrack** RichRecTrack associated to the RichRecSegment used in the formation of the RichRecPhoton candidate. More...
LHCb::RichRecPixel *	**m_richRecPixel** RichRecPixel used in the formation of the RichRecPhoton candidate. More...

- **m_geomPhoton** : Encapsulates the photon reconstruction
- **m_expPixel{Signal,Scatt}Phots** : Cached data on expected detector signals.
- **m_richRec{Segment,Pixel,Track}** : Relations back to associated data objects

- Data only related to each other in that they all are associated to the same photon candidates.
- RichRecPhoton a small cache ! Segment class much more extensive.

Moving to the Future…

- ***Make the algorithms fully thread safe.***
- ***Address design choices that hinder modern compiler optimisations.***
    - ***Algorithms that operate on const vector<Foo>& rather than tools on Foo*****. Create vector<Baa>.
    - Move towards SOA rather than AOS designs (so small Foo and Baa objects, often just float).
    - ***Remove monolithic event data model 'cache' objects***. More, smaller, more focused data objects. Composition when required using Range v3 library.
- ***Make use of AnyDataHandle*** to remove DataObject inheritance constraint.
- ***Move processing to algorithms***. Use tools only where appropriate.
- ***Remove all monitoring*** (even basic tallies) from reconstruction algorithms. Keep streamlined.
- ***Simplify python configuration***.
- ***Lower barrier for new people*** to contribute
- ***Conclusion - Start from scratch.***

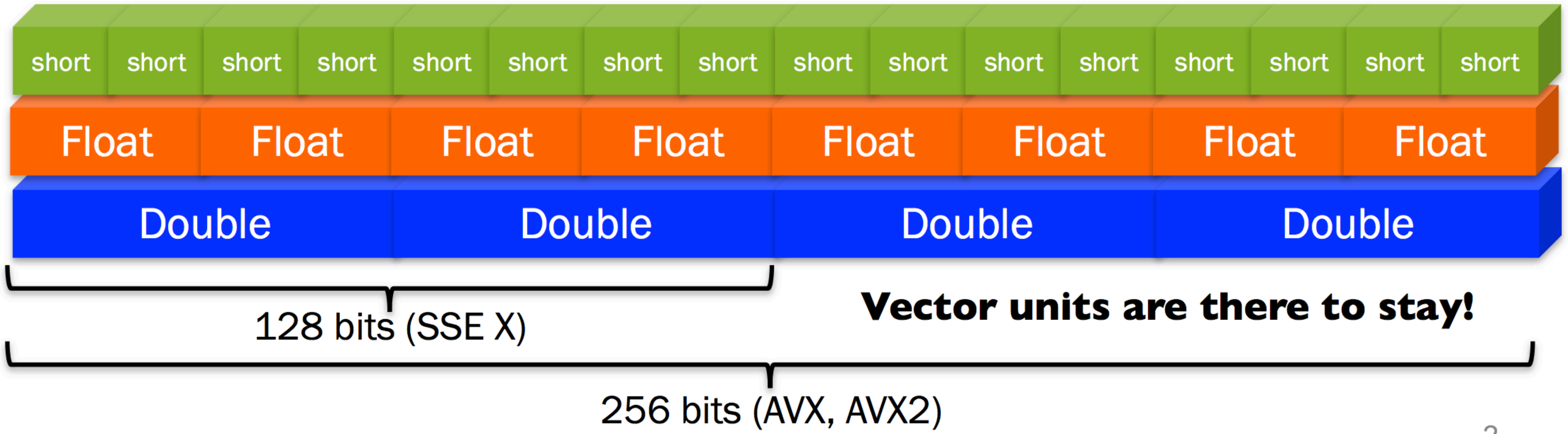
Status of Functional Migration

- Fully functional RICH reconstruction sequence using the new framework available in latest Brunel release.
    - In fact, is the default processing sequence for this years data taking.
    - Old sequence still available, as ‘backup’.
- Timing wise, ***new sequence is ~2.5 times faster**** than that in the Brunel released for the start of the 2016 data taking.
    - Largely due to the adoption of guidelines to better format the data how the CPU wants it, not the user. e.g. utilise SIMD vectorisation.
    - Also in part due to some specific code improvements.

* Comparing Brunel v50r1 to v52r2, using same machine and 1000 event sample

SIMD (Single Instruction Multiple Data)

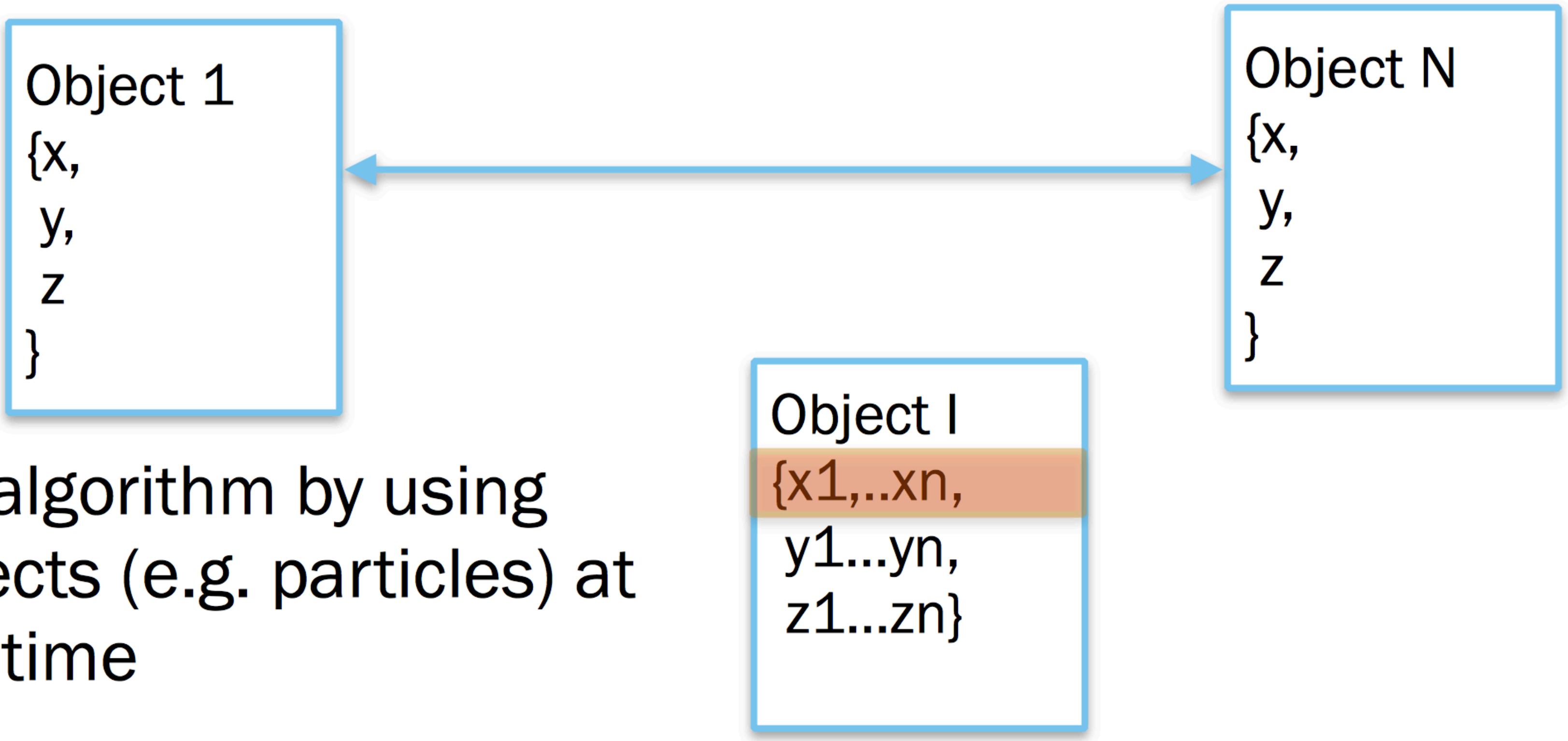
- Hardware vendors raise computational power of today's CPUs with increasing support for parallelism:
  - More cores (beyond the scope of this talk)
  - Larger vector units, richer vector instruction sets
- Vector units: perform same operation on multiple data
  - Data parallelism at instruction level
- Peak performance achievable only if vector units are properly used
  - Especially for "extreme" architectures like the Xeon Phi


short short short short short short short short short short short short short short short short
Float Float Float Float Float Float Float Float
Double Double Double Double
128 bits (SSE X)
256 bits (AVX, AVX2)

Vector units are there to stay!

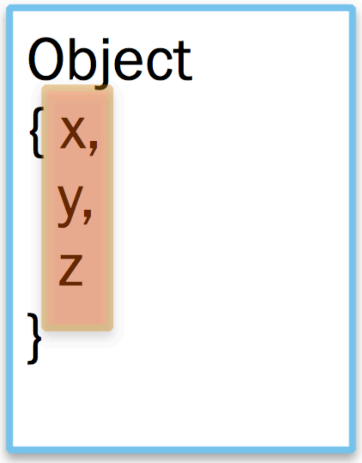
https://indico.cern.ch/event/258092/contributions/1588517/attachments/454175/629578/ACAT2014_ModernisingROOT_SWenzel.pdf

- **Horizontal (external) vectorisation:**



vectorize algorithm by using many objects (e.g. particles) at the same time

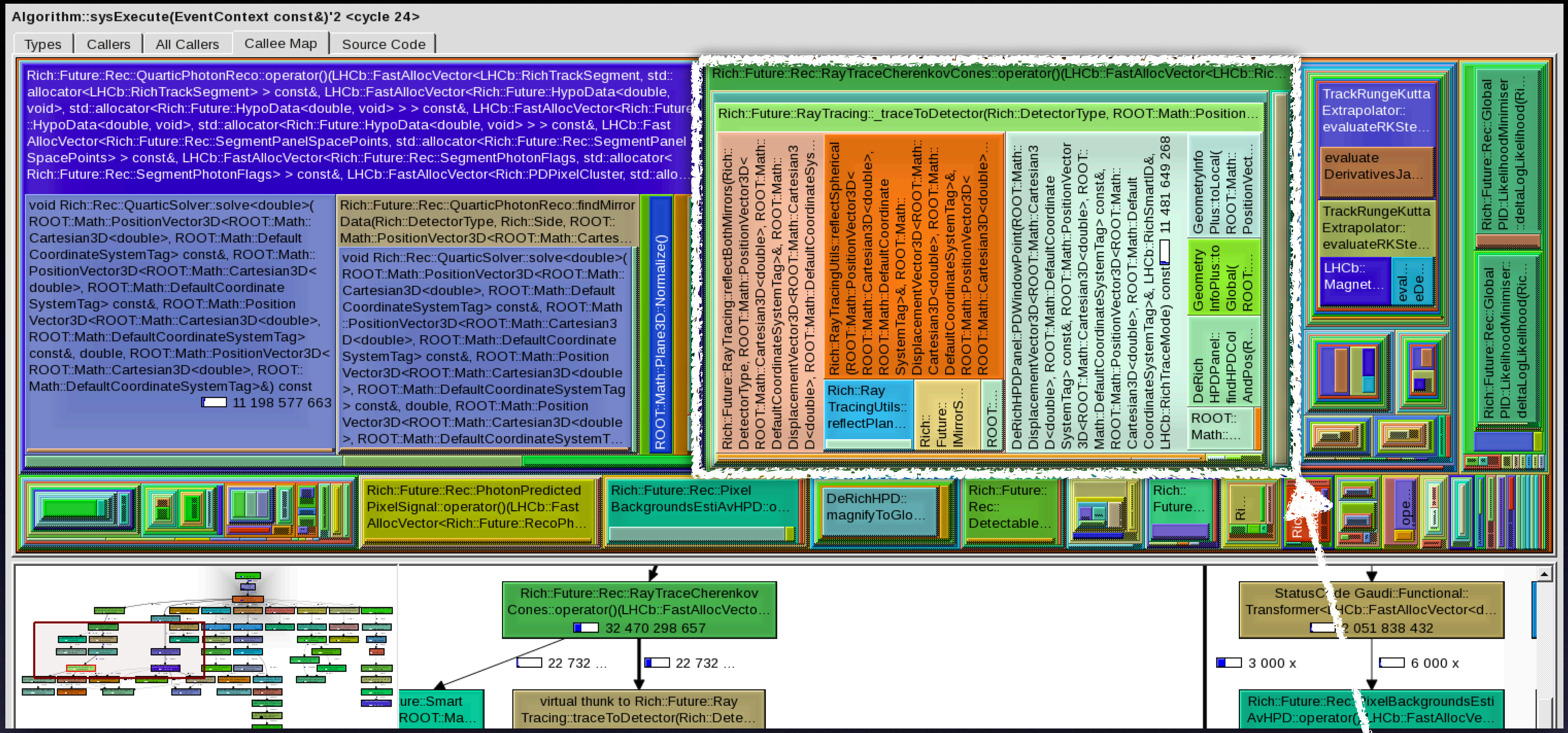
- **Vertical (internal) vectorisation:**



vectorize internally the algorithm operating on a single object
- example: for loop over internal data
Object data member (e.g. x,y,z) must be stored in a vector

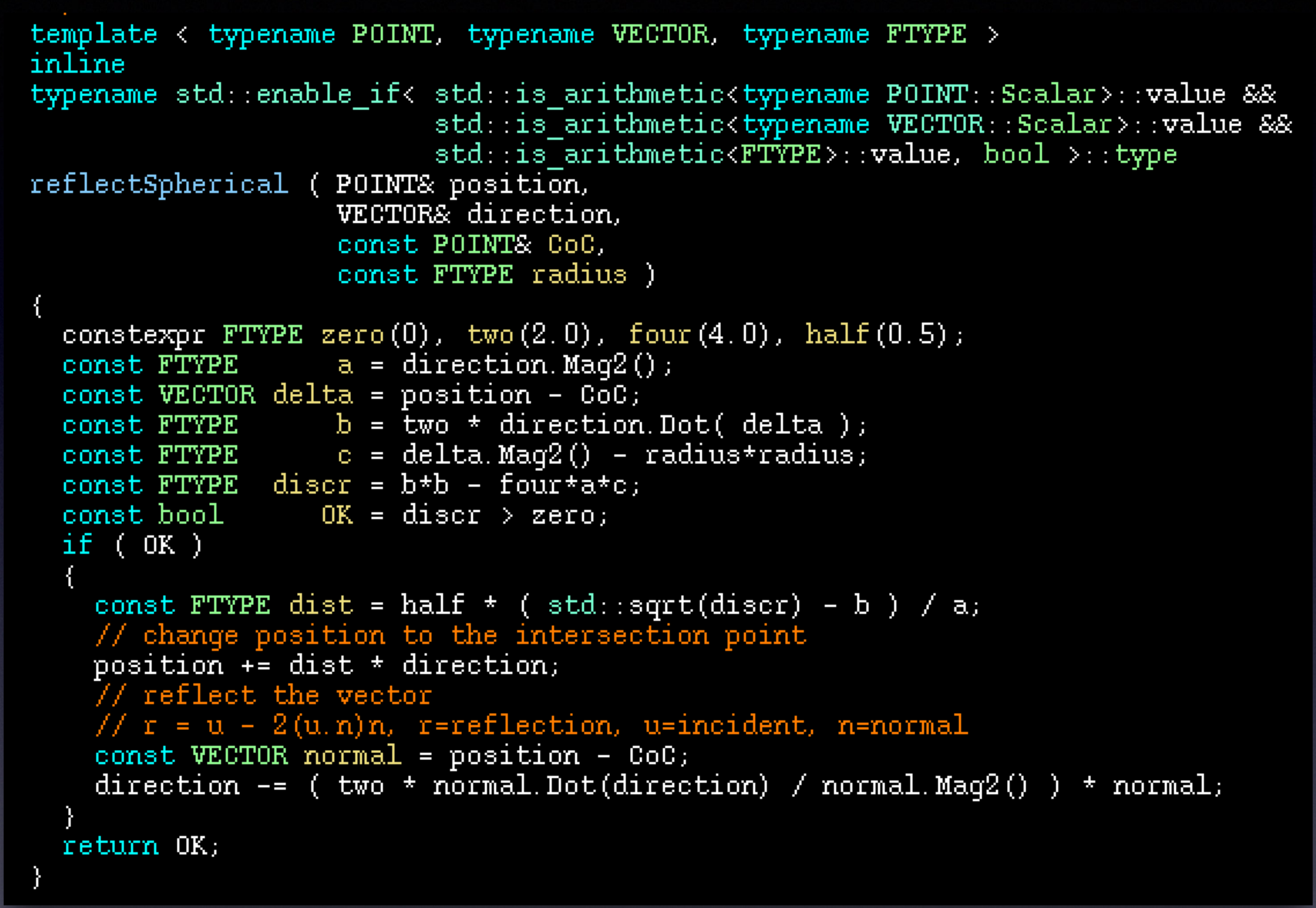
https://indico.cern.ch/event/258092/contributions/1588517/attachments/454175/629578/ACAT2014_ModernisingROOT_SWenzel.pdf

Ray Tracing Photons


Algorithm::sysExecute(EventContext const&)'2 <cycle 24>
Types
Callers
All Callers
Callee Map
Source Code
Rich::Future::Rec::QuarticPhotonReco::operator()(LHCb::FastAllocVector<LHCb::RichTrackSegment, std::allocator<LHCb::RichTrackSegment> > const&, LHCb::FastAllocVector<Rich::Future::HypoData<double,
Rich::Future::Rec::RayTraceCherenkovCones::operator()(LHCb::FastAllocVector<LHCb::Ric…
Rich::Future::RayTracing::_traceToDetector(Rich::DetectorType, ROOT::Math::Position…
void Rich::Rec::QuarticSolver::solve<double>(…
Rich::Future::Rec::QuarticPhotonReco::findMirrorData(Rich::DetectorType, Rich::Side, ROOT::Math::PositionVector3D<ROOT::Math::Cartes…
11 198 577 663
11 481 649 268
TrackRungeKutta Extrapolator::evaluateRKSte…
Rich::Future::Rec::PhotonPredictedPixelSignal::operator()(LHCb::FastAllocVector<Rich::Future::RecoPh…
Rich::Future::Rec::PixelBackgroundsEstiAvHPD::o…
Rich::Future::Rec::RayTraceCherenkovCones::operator()(LHCb::FastAllocVecto…
32 470 298 657
22 732 …
virtual thunk to Rich::Future::RayTracing::traceToDetector(Rich::Dete…
StatusCode Gaudi::Functional::Transformer<LHCb::FastAllocVector<d…
2 051 838 432
3 000 x
6 000 x
Rich::Future::Rec::PixelBackgroundsEstiAvHPD::operator()(LHCb::FastAllocVe…

- Ray tracing photons through RICH mirror system to detector plane major CPU usage for the RICH
- Simple geometrical calculation repeated many times. Should be easy to vectorise.
- Uses ROOT GenVector library. (The ubiquitous **Gaudi::XYZVector** and **Gaudi::XYZPoint** types.)
- Have previously tried internally vectorising (vertical) using libraries like Eigen. Results not too impressive. Difficult to see how this approach can fully utilise SIMD capabilities.
- Look instead at using Vc library. Provides **Vc::float_v**, **Vc::double_v** types that behave (more or less) like **float**, **double**, but are vectors(arrays) of N values (N depends on SIMD level). Horizontal vectorisation.

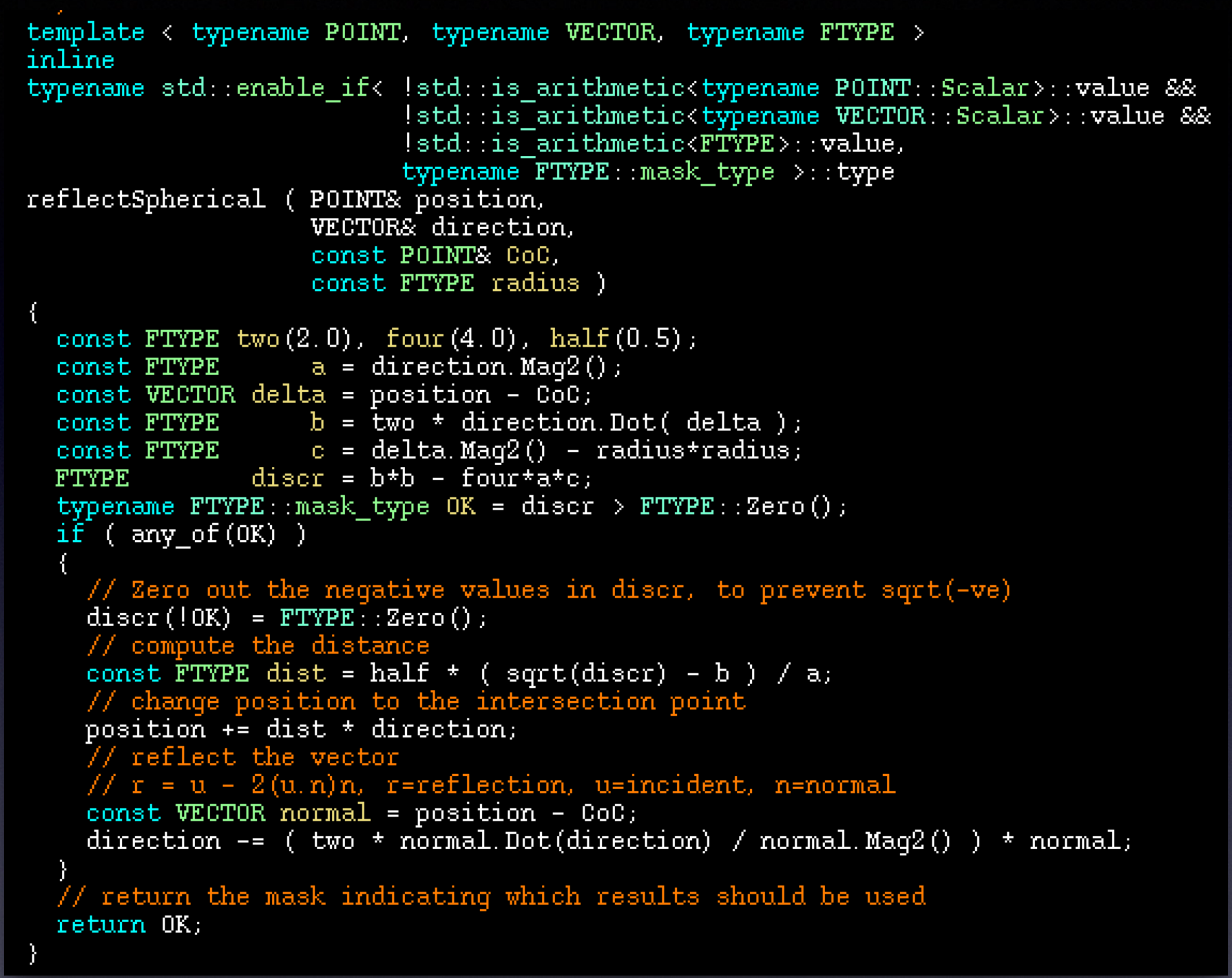
Reflect Spherical - Scalar



```
template < typename POINT, typename VECTOR, typename FTYPE >
inline
typename std::enable_if< std::is_arithmetic<typename POINT::Scalar>::value &&
                         std::is_arithmetic<typename VECTOR::Scalar>::value &&
                         std::is_arithmetic<FTYPE>::value, bool >::type
reflectSpherical ( POINT& position,
                   VECTOR& direction,
                   const POINT& CoC,
                   const FTYPE radius )
{
  constexpr FTYPE zero(0), two(2.0), four(4.0), half(0.5);
  const FTYPE      a = direction.Mag2();
  const VECTOR delta = position - CoC;
  const FTYPE      b = two * direction.Dot( delta );
  const FTYPE      c = delta.Mag2() - radius*radius;
  const FTYPE  discr = b*b - four*a*c;
  const bool      OK = discr > zero;
  if ( OK )
  {
    const FTYPE dist = half * ( std::sqrt(discr) - b ) / a;
    // change position to the intersection point
    position += dist * direction;
    // reflect the vector
    // r = u - 2(u.n)n, r=reflection, u=incident, n=normal
    const VECTOR normal = position - CoC;
    direction -= ( two * normal.Dot(direction) / normal.Mag2() ) * normal;
  }
  return OK;
}
```

- POINT and VECTOR are just template types for, e.g. Scalar XYZPoint<**double**> and XYZVector<**double**>

Reflect Spherical - Vectorised



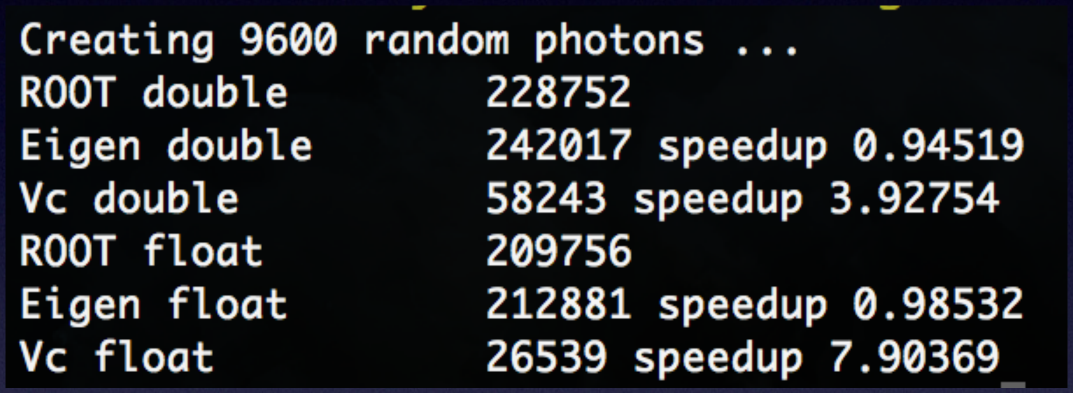
```
template < typename POINT, typename VECTOR, typename FTYPE >
inline
typename std::enable_if< !std::is_arithmetic<typename POINT::Scalar>::value &&
                         !std::is_arithmetic<typename VECTOR::Scalar>::value &&
                         !std::is_arithmetic<FTYPE>::value,
                         typename FTYPE::mask_type >::type
reflectSpherical ( POINT& position,
                   VECTOR& direction,
                   const POINT& CoC,
                   const FTYPE radius )
{
  const FTYPE two(2.0), four(4.0), half(0.5);
  const FTYPE      a = direction.Mag2();
  const VECTOR delta = position - CoC;
  const FTYPE      b = two * direction.Dot( delta );
  const FTYPE      c = delta.Mag2() - radius*radius;
  FTYPE        discr = b*b - four*a*c;
  typename FTYPE::mask_type OK = discr > FTYPE::Zero();
  if ( any_of(OK) )
  {
    // Zero out the negative values in discr, to prevent sqrt(-ve)
    discr(!OK) = FTYPE::Zero();
    // compute the distance
    const FTYPE dist = half * ( sqrt(discr) - b ) / a;
    // change position to the intersection point
    position += dist * direction;
    // reflect the vector
    // r = u - 2(u.n)n, r=reflection, u=incident, n=normal
    const VECTOR normal = position - CoC;
    direction -= ( two * normal.Dot(direction) / normal.Mag2() ) * normal;
  }
  // return the mask indicating which results should be used
  return OK;
}
```

- POINT and VECTOR are ***vectorised*** e.g., XYZPoint<**Vc::double_v**> and XYZVector<**Vc::double_v**>
- Effectively, think of each as being an 'array' of **Vc::double_V::Size** entries, operated on in parallel

Vectorised Ray Tracing - Performance

- Idealised test application - photon reflection of first primary (spherical) and then secondary (flat) mirrors - shows almost 'perfect SIMD' speed up.

AVX2 (4 double / 8 float)

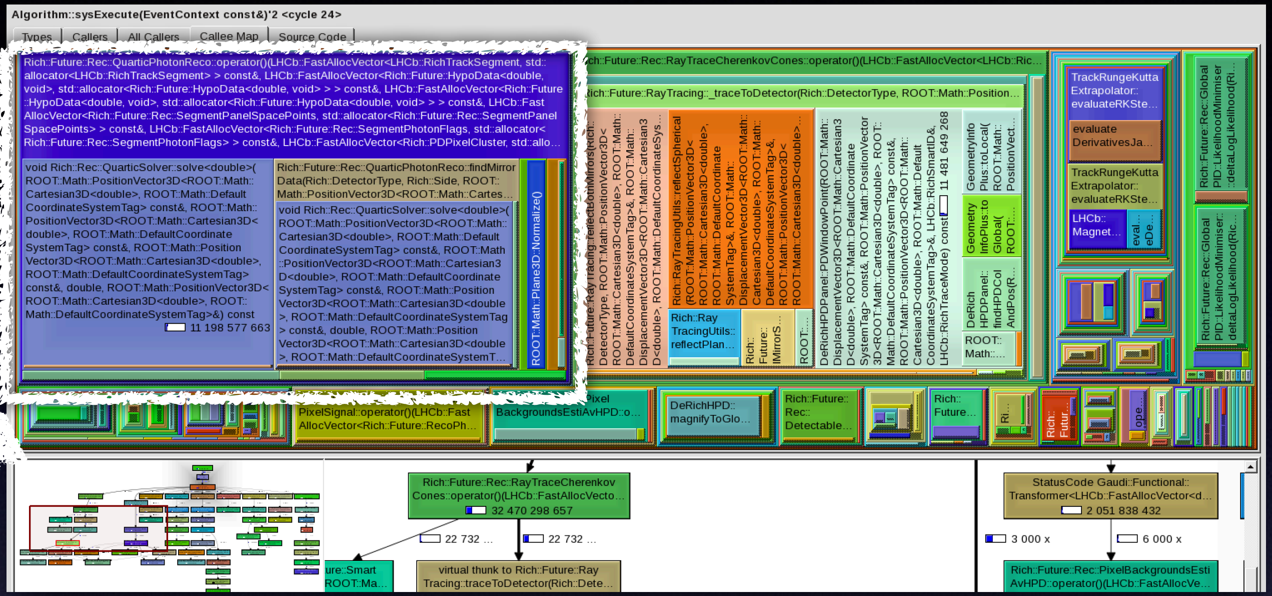


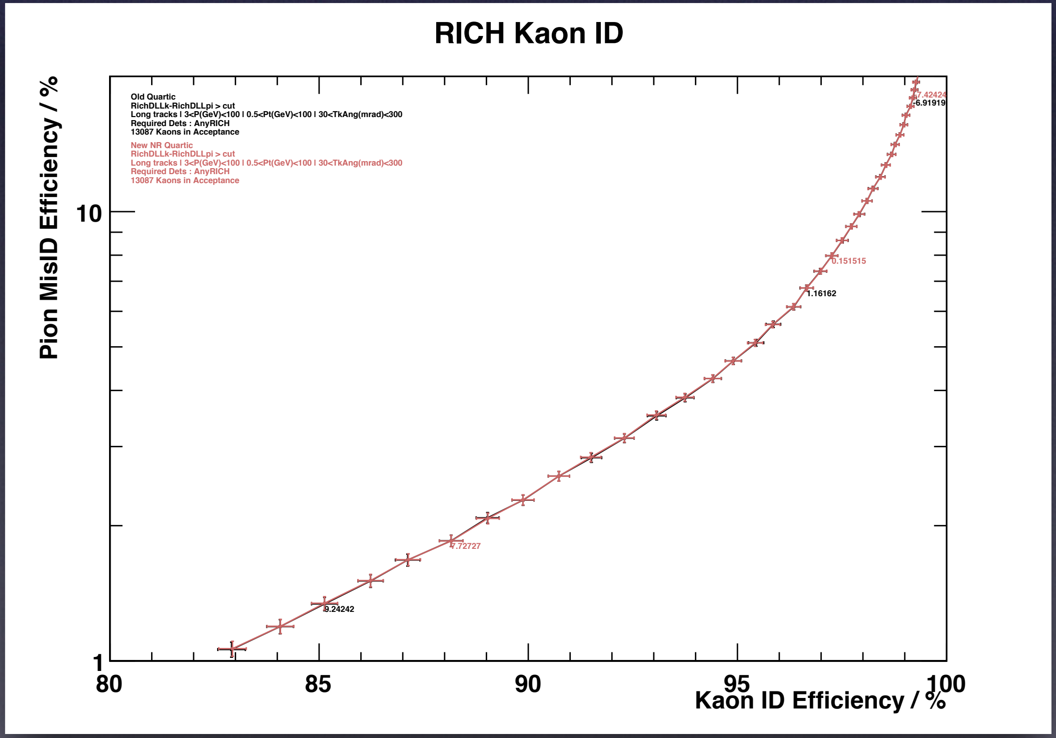
```
Creating 9600 random photons ...
ROOT double         228752
Eigen double        242017 speedup 0.94519
Vc double           58243 speedup 3.92754
ROOT float          209756
Eigen float         212881 speedup 0.98532
Vc float            26539 speedup 7.90369
```

- Real world deployment, somewhat more complicated…
    - Multiple mirror segments, HPD/MaPMT intersection. Not fully vectorised yet.
    - Nevertheless, already see ~33% speed up.
- First steps… Will significantly improve once we learn how to work with SIMD vectorisation better.

Ongoing ‘Parallel’ Efforts

- Studies ongoing to port expensive RICH tasks (photon reconstruction, ray tracing) to various ‘highly parallel’ architectures (R.Schwemmer, C.Quast, C.Faerber)
  - KL (Knights Landing) - ‘Regular’ x86 CPU with many cores and SIMD vector units.
  - FPGAs. Very (very) parallel…
- Work ongoing, but results already encouraging and have resulted in some improvements which could be incorporated in the last release.
  - Speed up in the quartic reconstruction algorithm due to replacement of an analytic solution with a ***Newton Raphson approach***.
  - ***O(40%) speed up !***
- *Clear example of the advantage of getting new people involved.*


Algorithm::sysExecute(EventContext const&)'2 <cycle 24>
Types
Callers
All Callers
Callee Map
Source Code
Rich::Future::Rec::QuarticPhotonReco::operator()
void Rich::Rec::QuarticSolver::solve<double>
Rich::Future::Rec::QuarticPhotonReco::findMirrorData
11 198 577 663
Rich::Future::Rec::RayTraceCherenkovCones::operator()
32 470 298 657
22 732 …
22 732 …
virtual thunk to Rich::Future::Ray Tracing::traceToDetector
StatusCode Gaudi::Functional::Transformer
2 051 838 432
3 000 x
6 000 x
Rich::Future::Rec::PixelBackgroundsEstiAvHPD::operator()
TrackRungeKutta Extrapolator::evaluateRKSte…



Performance statistically identical

Conclusions

- **RICH Reconstruction ported to the upgrade ‘functional’ framework.**
    - **Deployed and ready for 2017 data taking.**
    - Directly, and indirectly, has resulted in a significant improvement in CPU time.
    - **A factor of 2.5 w.r.t. 2016 reconstruction**.
        - RICH now a rather small fraction of overall reconstruction time.
- Just the start…
    - More improvements expected through further improved utilisation of CPU/ cache friendly data structures.
- So far have only touched ‘internal’ RICH data structures. Eventually the **public event model** (RichPID, tracks, Particles) will need to be heavily updated.