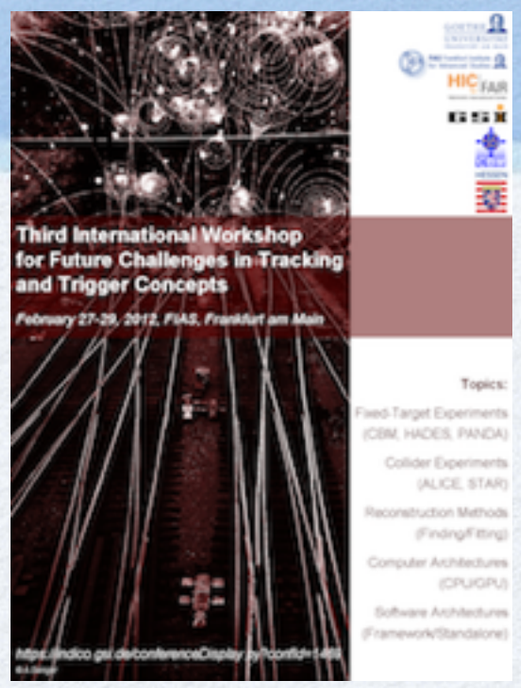
ROOT and Parallelism

Lorenzo Moneta
CERN, PH-SFT



Third International Workshop for Future Challenges in Tracking and Trigger Concepts

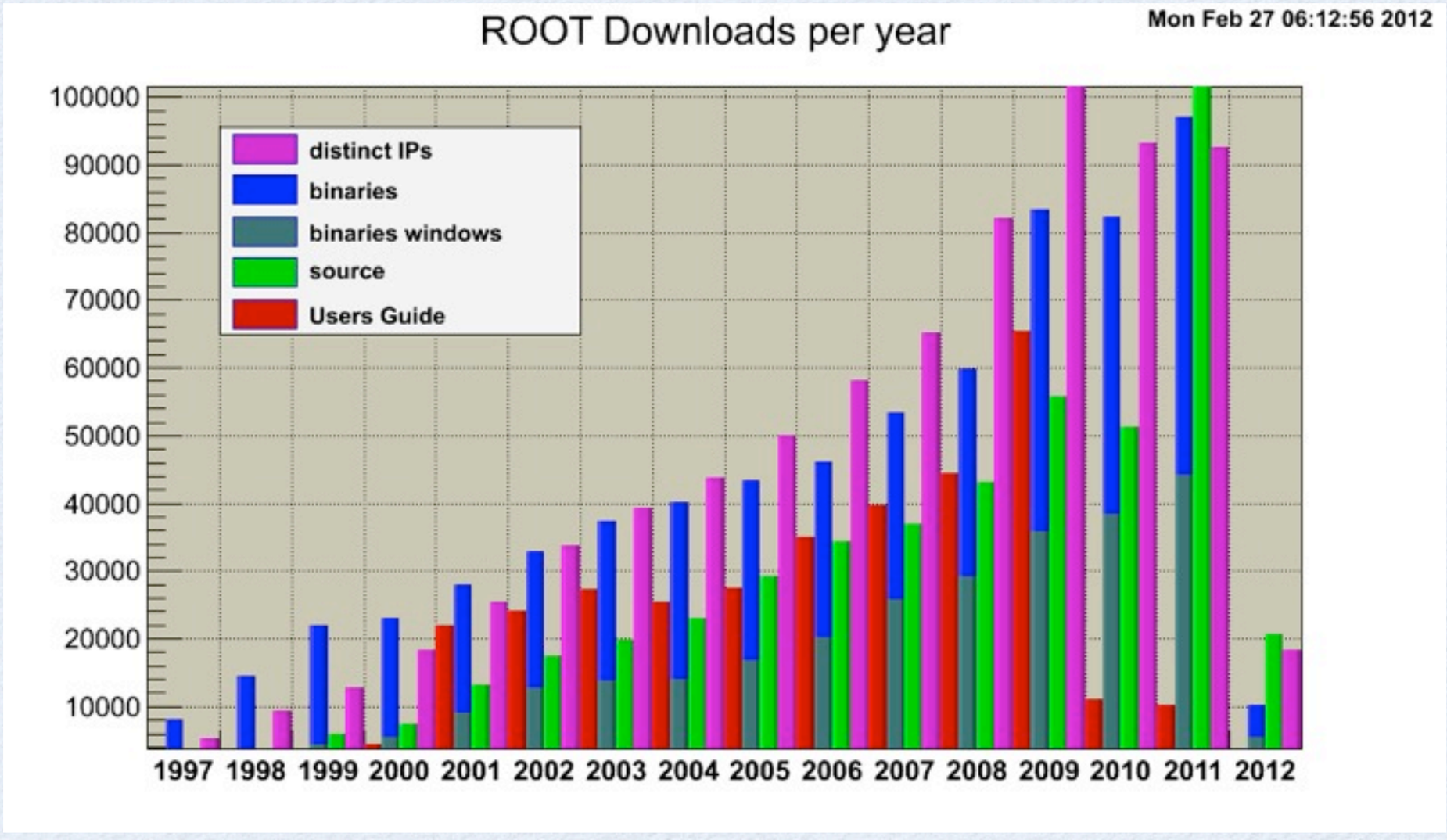
27-29 February 2012

Outline

- ROOT: new and planned developments
- Parallelism in ROOT
  - Parallelism in I/O
  - Parallelism in data analysis (fitting)
- Vectorization
  - vectorization of ROOT matrix and vector libraries
- Parallel random numbers
- PROOF

ROOT Going Strong

- Ever increasing number of users
  - 5600 forum members, 56850 posts, 1300 mailing list members
  - Used by basically all HEP experiments and beyond



New ROOT developments

- New interpreter CLING to replace CINT
- ROOT porting to iOS devices and ROOT iOS browser application
  - new OSX graphics native support
- Javascript based ROOT browser
- I/O performance improvements:
  - Parallel merge of ROOT files
- Improvements in data analysis tools
  - better support for parallelization

Next ROOT Releases

- ROOT 5.34 in the summer
  - with iOS support and JavaScript ROOT browser
- ROOT 6 at the end of the year
  - with Cling based ROOT

New Interpreter

- Replacing good old CINT by new, shining Cling
- Cling is based on LLVM and Clang compiler libraries
  - LLVM/Clang “new” open source compiler suite
  - Default compiler on OSX Lion
- Cling released in July 2011
  - Fully functional C and C++ interpreter (including C++11)
  - Uses Just-In-Time compilation
  - Still a few issues to solve (e.g. reloading of code)
- Integration with ROOT starting now
  - Interfaces via TCling and TClass
  - Use precompiled header files (PCH’s) as dictionary repository (no more huge compiled dictionaries)

Parallelism in I/O

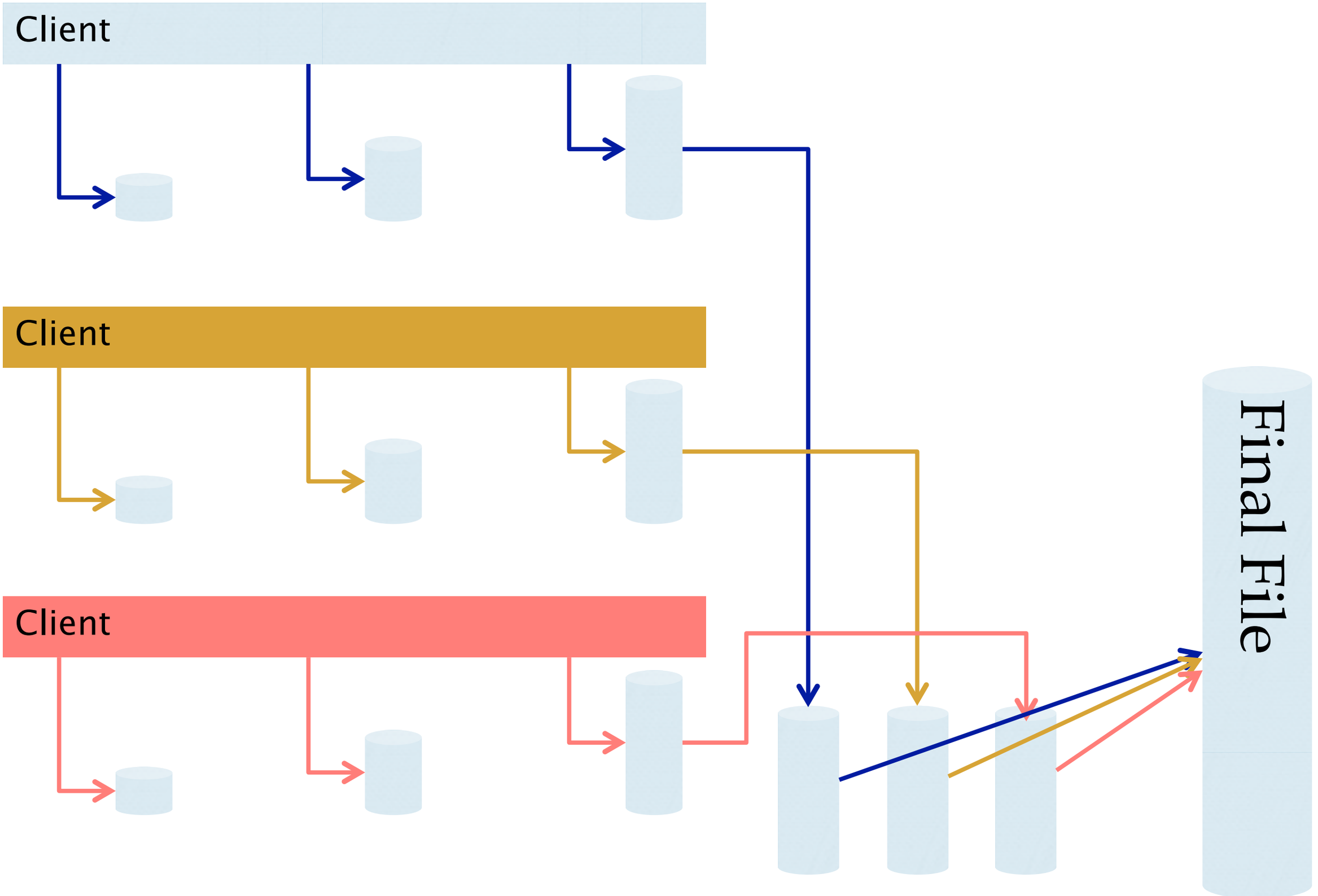
- 3 approaches for parallelizing I/O
  - Parallel Merge via external process
  - Support for one TFile per thread
    - Currently requires meta data (TClass, TStreamerInfo) to be fully build before starting parallel operation.
    - Planning to lifting this restriction in ROOT 6 (requires cling).
  - Internal use of spare cores
    - Ability to read multiple TBranch data in parallel
      - Top level branches can be uncompressed and un-streamed independently.
    - Prefetching of remote file content as a separate thread.

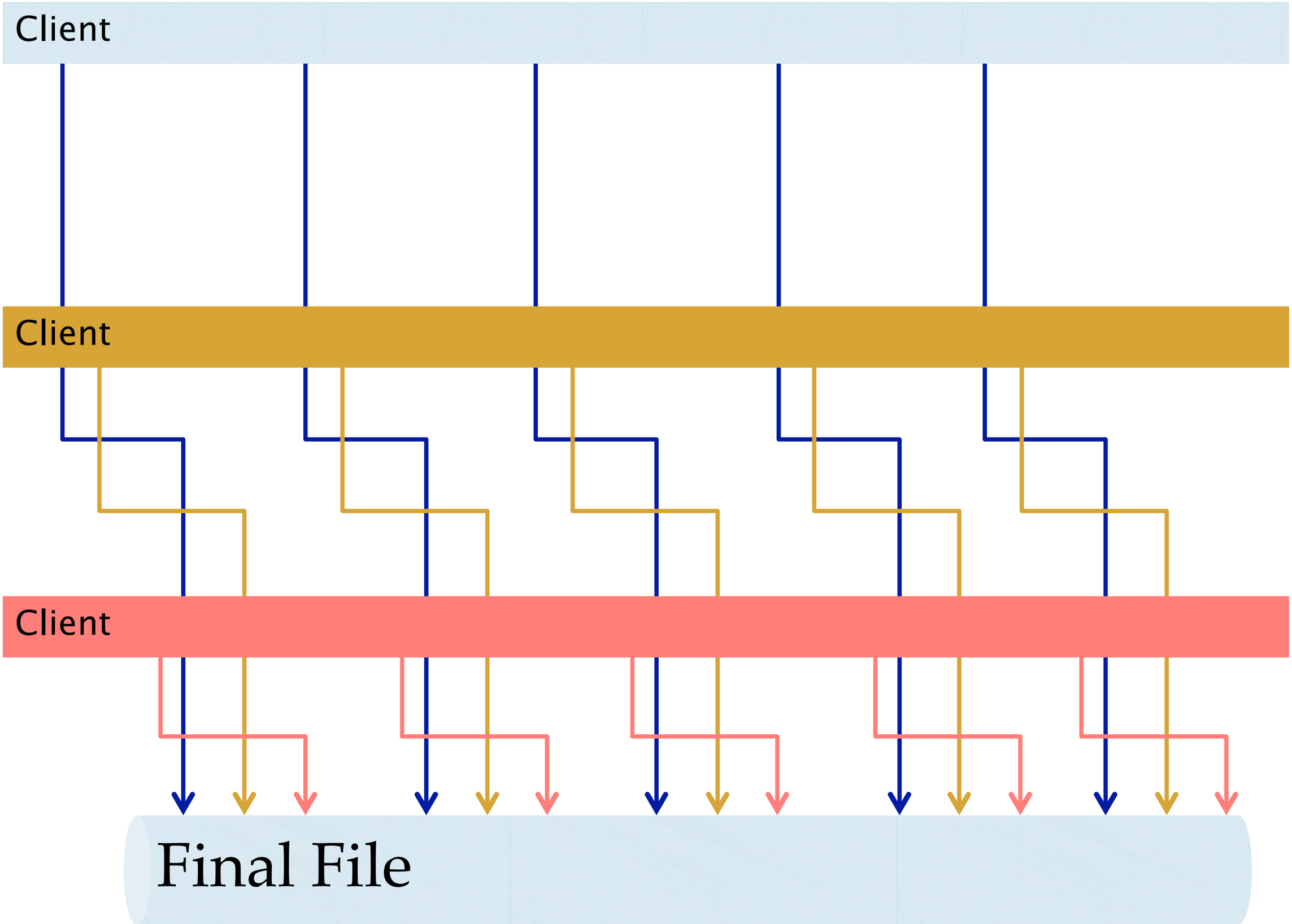
Parallel Merge

- New class TParallelMergingFile
  - A TMemFile that on a call to Write will
    - Upload its current content to a parallelMergerServer
    - Reset the TTree objects to facilitate the new merge.

```
TFile::Open("mergedClient.root?pmerge=localhost:1095","RECREATE");
```

- New daemon parallelMergeServer
  - Receive input from local or remote client and merger into request file (which can be local or remote).
  - Fast merge TTree. Re-merge all histogram at regular interval.


Client
Client
Client
Final File


Client
Client
Client
Final File

Data Analysis Parallelism

- Statistical techniques all based on the likelihood function
  - each event is described by a probability density function (PDF)

$$P(x|\theta) \qquad \text{Likelihood:} \qquad L(x|\theta) = \prod_i P(x_i|\theta)$$

- All statistical methods require evaluation of the likelihood
  - maximum likelihood fit for parameter estimation
  - integral of likelihood for Bayesian methods

$$\int L(x|\mu,\nu)\Pi(\mu,\nu)d\nu$$

  - profile likelihood distribution for frequentist methods (toy generation and fitting)

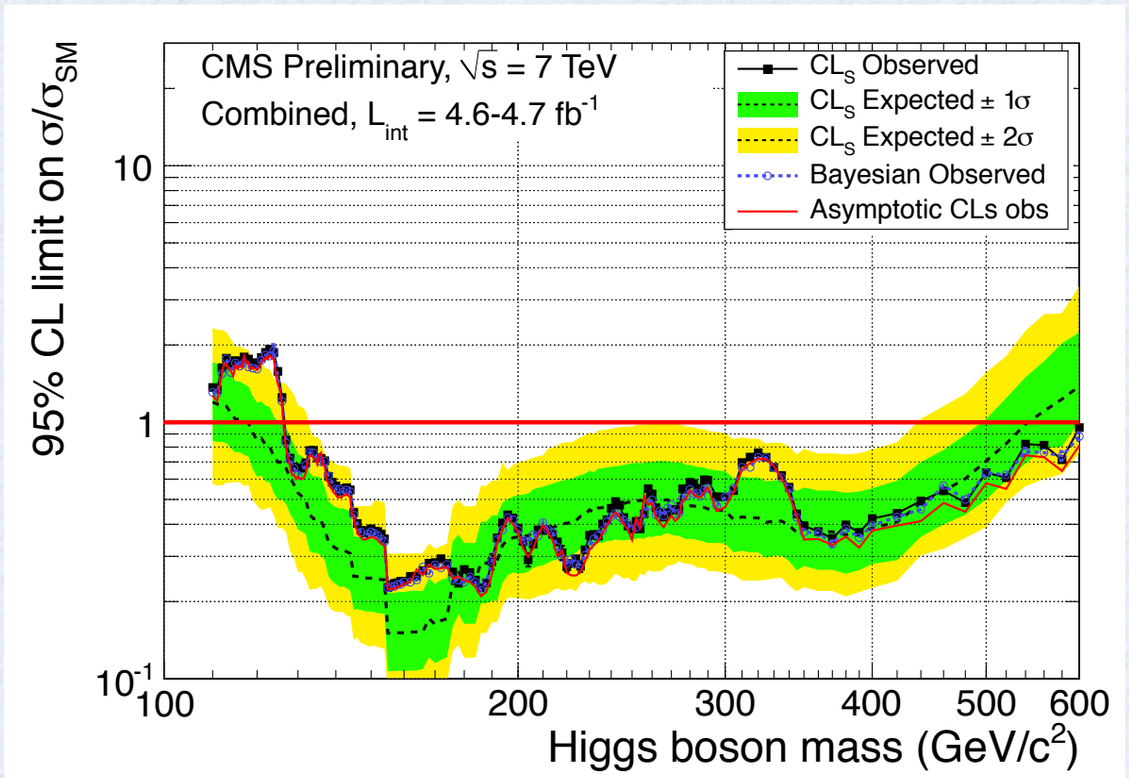
$$\lambda(\mu) = \frac{L(x|\mu,\hat{\hat{\nu}})}{L(x|\hat{\mu},\hat{\nu})}$$

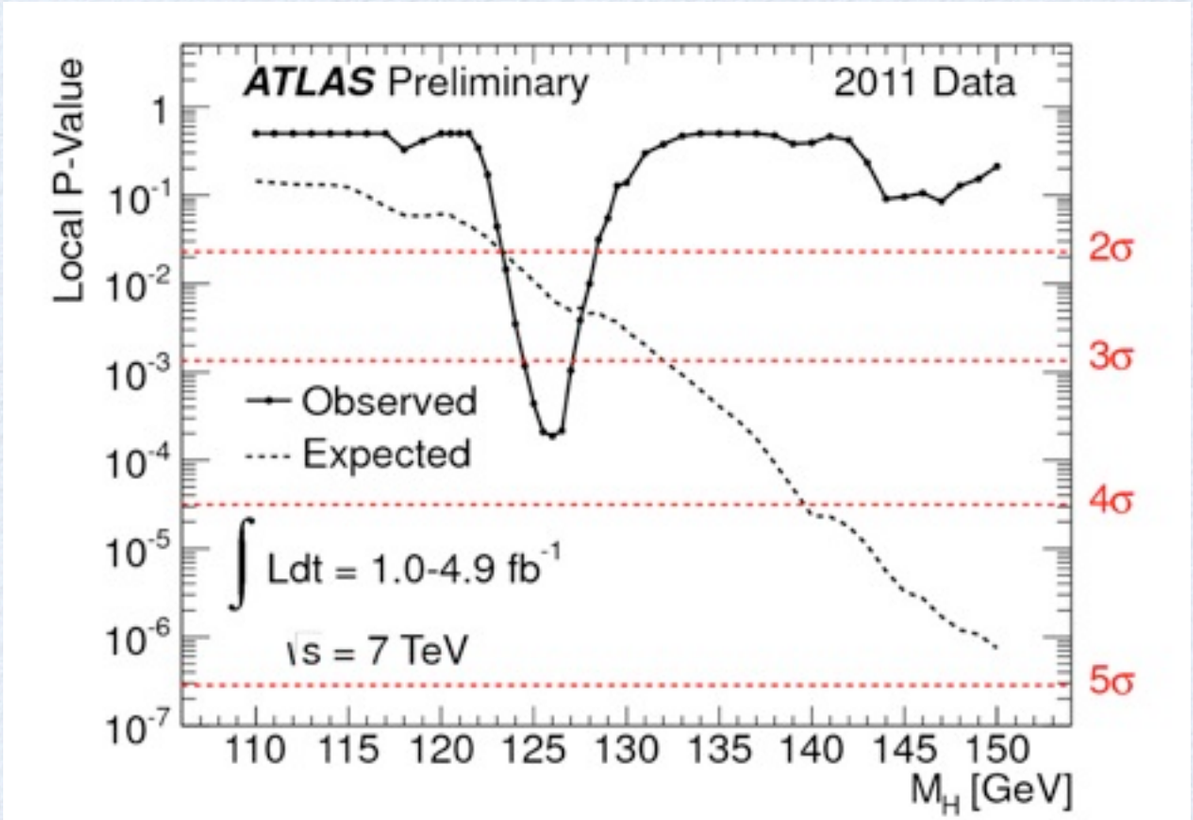
Data Analysis Parallelization

- Use typically RooFit for building complex PDF and RooStats for running statistical analysis
  - models with many PDF, many observables and a lot of parameters
    - e.g. Higgs combination (more than 200 parameters and several channels)
- Possible various level of parallelizations:
  - PDF evaluation
  - Loop on events for computing log-likelihood
  - Algorithms (e.g Minuit) require multiple likelihood evaluations
  - Loop on toy data analysis (on various likelihood minimization)
  - Repetition of same analysis on different inputs (analysis points)

Example: Higgs Searches

- Higgs search results require numerous minimization of complex likelihood functions (> 200 parameters)





Minuit Parallelization

- Parallelization of Migrad minimization algorithm
- Each Migrad iteration consists of:
  - computing function value and gradient to find Newton direction
  - computing step by searching for minimum along the Newton direction
  - if satisfactory improve calculation of Hessian matrix, H
  - invert to get new matrix $V = H^{-1}$
  - repeat iteration until expected distance from minimum smaller than tolerance

Minuit Parallelization

- In case of many parameters (> 10) and complex function evaluation, gradient calculation dominates the process:

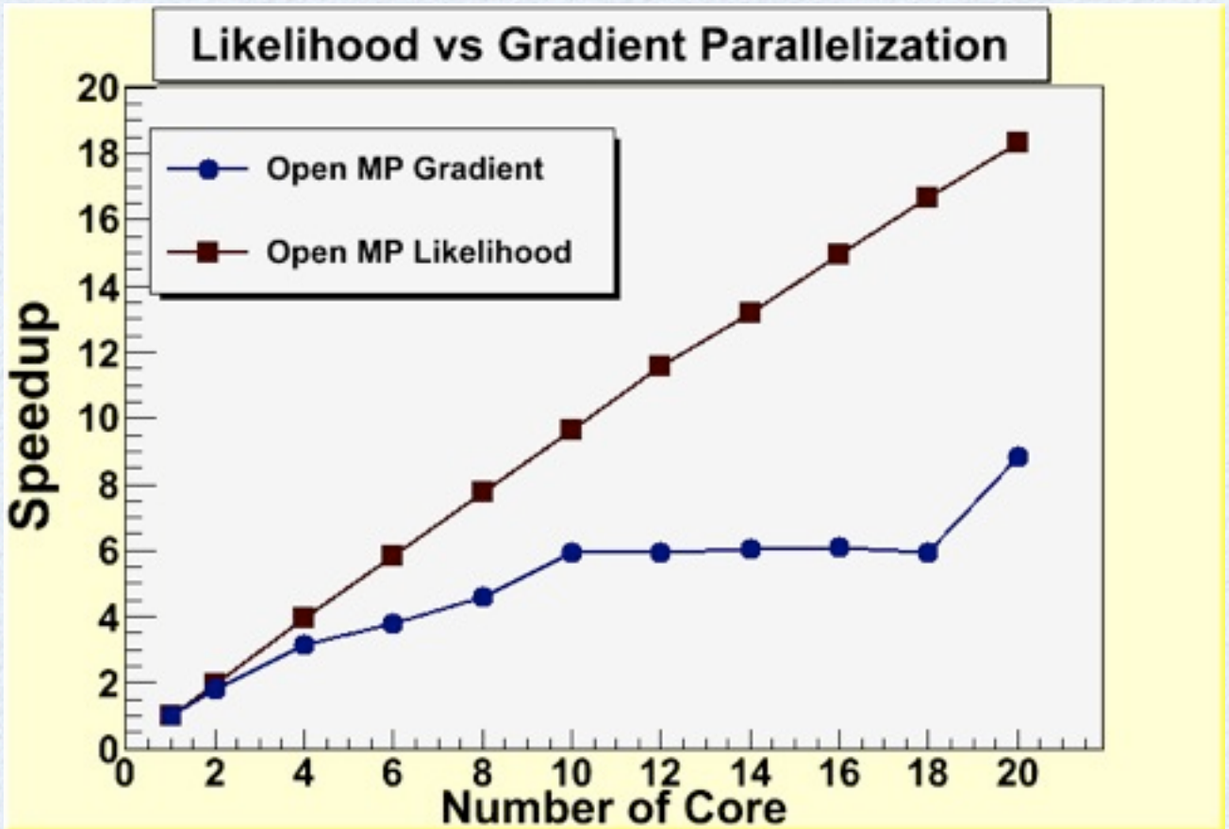
$$\nabla_i(x) = \frac{\partial f}{\partial x_i} \approx \frac{f(x_i + \delta x_i) - f(x_i - \delta x_i)}{2\delta x_i}$$

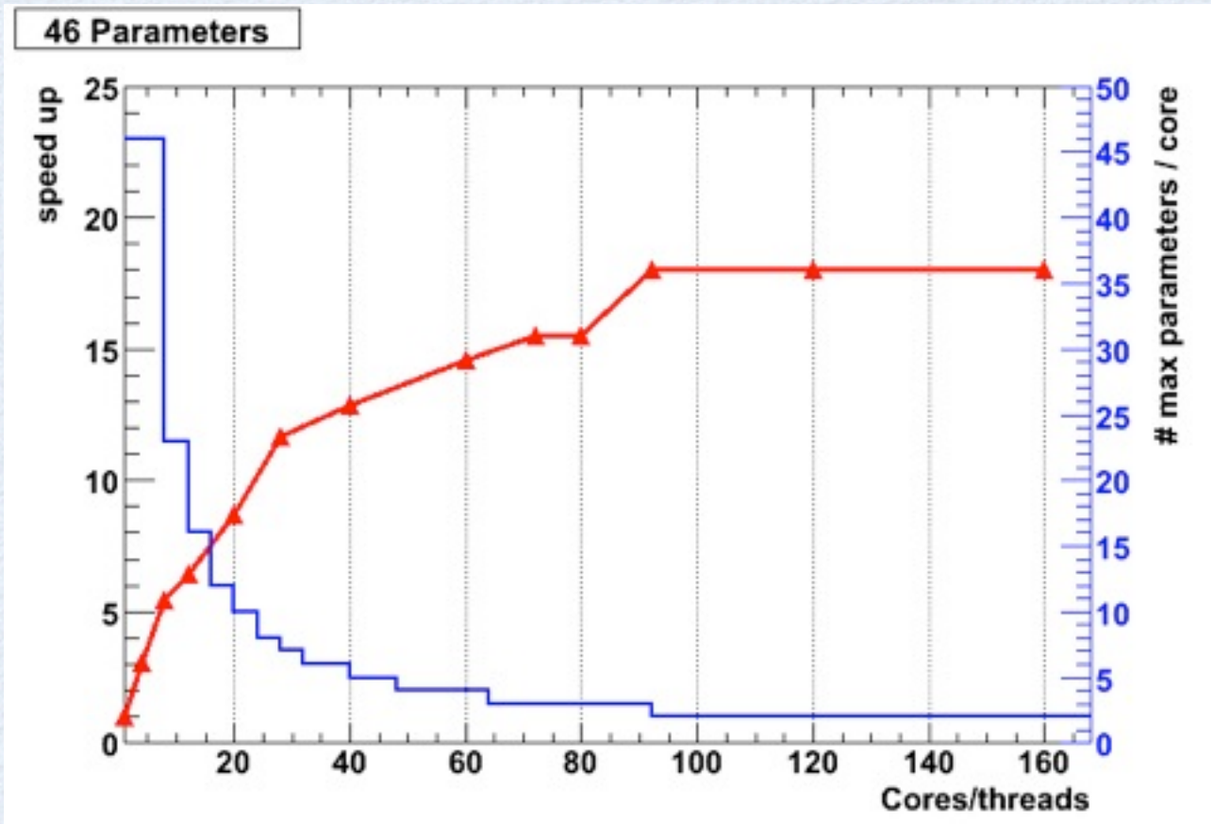
  - al least 2 * NDIM function evaluation are needed

- Parallelize calculations by using a thread for computing each partial derivative
- Use OpenMP (multi-thread) or MPI (multi-processes)
- Available in ROOT for Minuit2 (new C++ version of Minuit)
  - since version 5.22 (3 years ago)

Results for Minuit Parallelization

- unbinned fit with 20 parameters using openMP
- complex BaBar fitting parallelized using MPI (A. Lazzaro)





Minuit Parallelization

- Parallelization in Minuit is independent of user code
  - requirement only of thread safety function evaluation when using multi-thread implementation (OpenMP)
    - problems when function uses cached values (e.g. function normalization)
- Alternatively, one can parallelize evaluation of minimization function (log-likelihood function)
  - more efficient, but requires user to change code or to use a fitting package providing it (e.g. RooFit)
  - ROOT fitting classes are not providing this parallelization, but it is planned to do it

RooFit Parallelization

- RooFit: toolkit for data modeling and fitting (parameter estimation)
- RooFit supports parallelization in evaluating the log-likelihood function
  - multi-process parallelization
  - use fork to parallelize likelihood on multi-processes
    - pdf->fitTo(data, NumCPU(8));
  - Support also for PROOF and PROOFLite
    - useful for multiple likelihood fits (e.g. for toy studies, goodness of fits, etc.)

RooStats Parallelization

- RooStats: advances statistical tools for interval estimation (e.g. limits) and hypothesis tests (estimation of discovery significance)
  - frequentist tools are based on toys generations
- Support for parallelization of toys (generation and fitting) using PROOF
  - results from each toy (ROOT object) are automatically merged and returned to the user as running a serial job
  - PROOFLite found to be very convenient to use on user desktops
  - memory can start to be a problem with very large models and many cores
- Trivial parallelization performed at job level
  - run several jobs on Grid or on cluster each with a small number of toys
  - RooStats provides the tools for merging results but users still needs to do it
  - this is now the most common usage of RooStats for the complex analysis

Vectorization of PDF

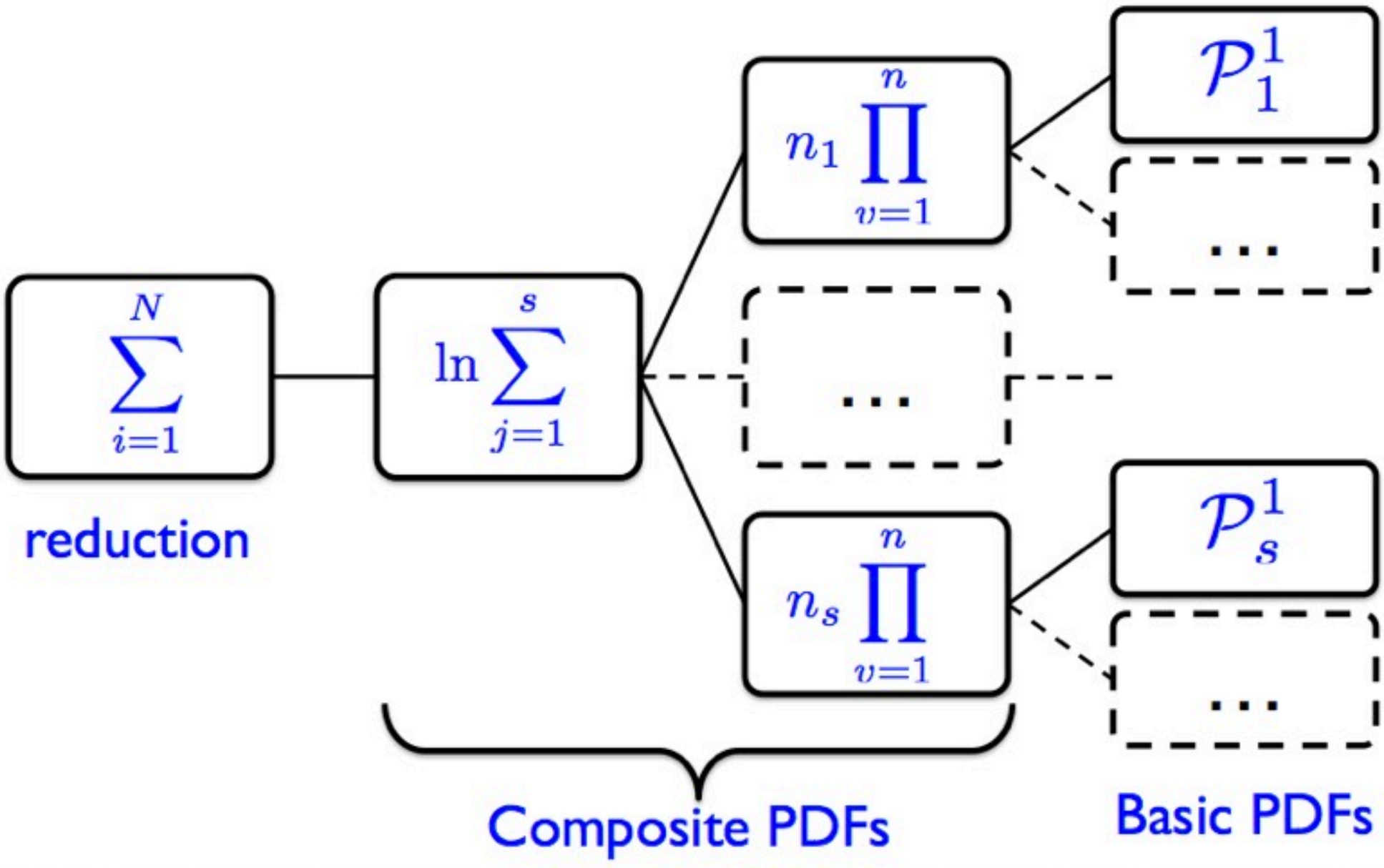
- Openlab fitting prototype
- Organize data (observables) as vectors
- Evaluate PDF not on a single observables but on vector of observables

$$P_i = P(x_i|\theta) \Longrightarrow \vec{P}(\vec{x}|\theta)$$

- Collect data vector of pdf and combine them to evaluate the log-likelihood (e.g. summing the vector values)
- Allows for SIMD vectorization during the pdf evaluation

Openlab Prototype

- We can visualize the NLL evaluation as a tree



Possible various level of parallelization

Openlab Prototype

- Studied parallelization at various levels using multi-threads
  - CPU with OpenMP
  - GPU with CUDA or OPENCL
  - hybrid setup to optimize CPU/GPU load with OpenCL
- Levels:
  - parallelize loop on the single PDF evaluation of the observables
  - parallelize outer loop for summing the final result
- Try to have minimal change in RooFit code
  - results reported in various presentations and reports from Openlab (see for example EPRINT: CERN-IT-2011-012)

Likelihood Parallelization

- Inner loop parallelization:
  - small memory footprint and better for race conditions
  - suffer from OpenMP overhead in having multiple parallel regions
  - require manage a large number of arrays with the evaluation results
  - cache problems when evaluating composite PDF's
    - much better scalability when using processors with larger cache
  - GPU -> CPU communication problems for summing final results

Likelihood Parallelization

- Outer loop parallelization:
  - better scalability
  - suffer from race conditions
  - more difficult to implement, it requires more changes in original code
    - developed prototype has many changes and is difficult to port in RooFit production code
- Conclusions
  - parallelizing existing code is not easy
  - importance of optimizing and redesigning code to have good scalability for many threads
    - this will result also in a faster scalar version of the code

Vectorization

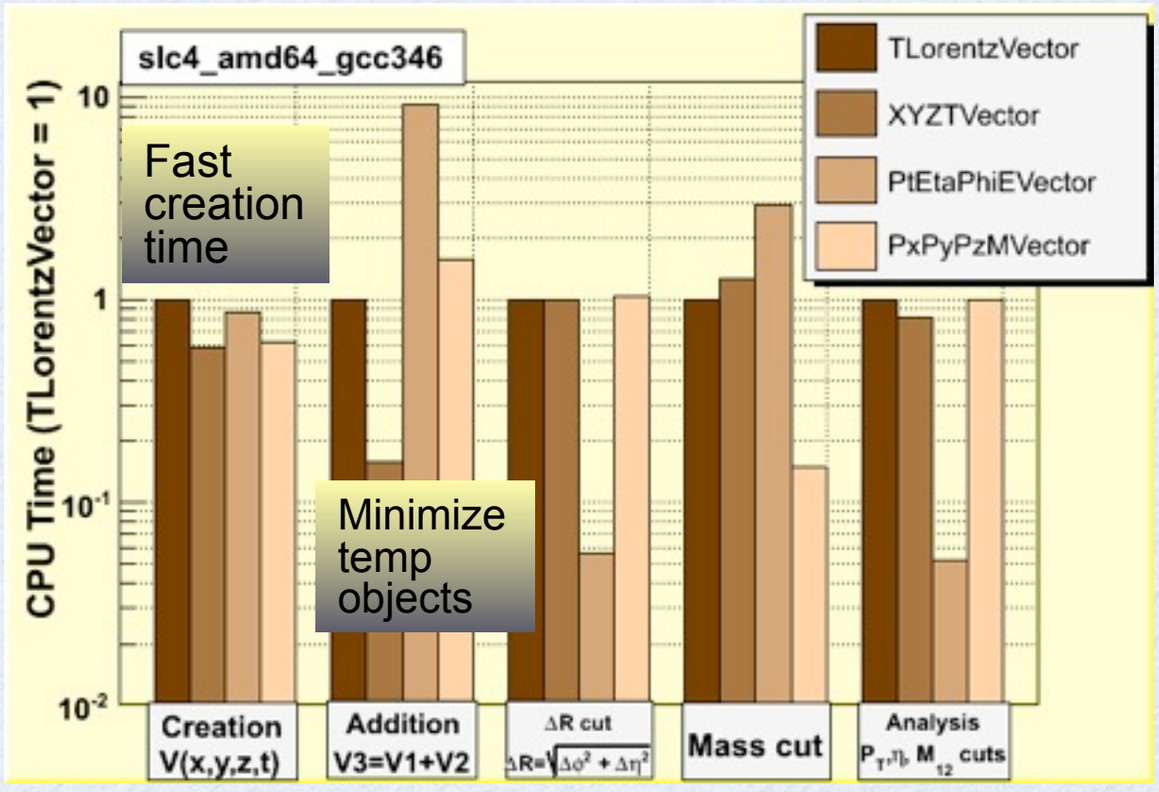
- Another parallelization dimension
  - vector processing using SIMD (Single Instruction Multiple Data)
- Perform numerical operations in parallel
  - size of registers depending on architectures
    - SSE : 128 bits : 2 double's or 4 float's
    - AVX: 256bit : 4 double's or 8 float's)
- Compilers can perform auto-vectorization of loops
  - require data organized in vectors and iteration independence
  - branches (if statement) can break vectorization
  - new compilers (e.g. gcc 4.6) are much better
- Can use special instructions for processors (intrinsic)
  - SSE or AVX instructions
- Libraries exist to hide this complexity to user (e.g. Vc library)

New ROOT Physics Vectors

- Classes for 3D and 4D vectors and their operations (GenVector package)
  - template on contained type
    - i.e. single or double precision
  - template on coordinate system type
    - i.e. cartesian, polar and cylindrical
  - no virtual table

- `LorentzVector<PxPyPzE<double> >`
- `LorentzVector<PtEtaPhiE<double> >`
- `LorentzVector<PxPyPzM<double> >`
- `LorentzVector<PtEtaPhiM<double> >`

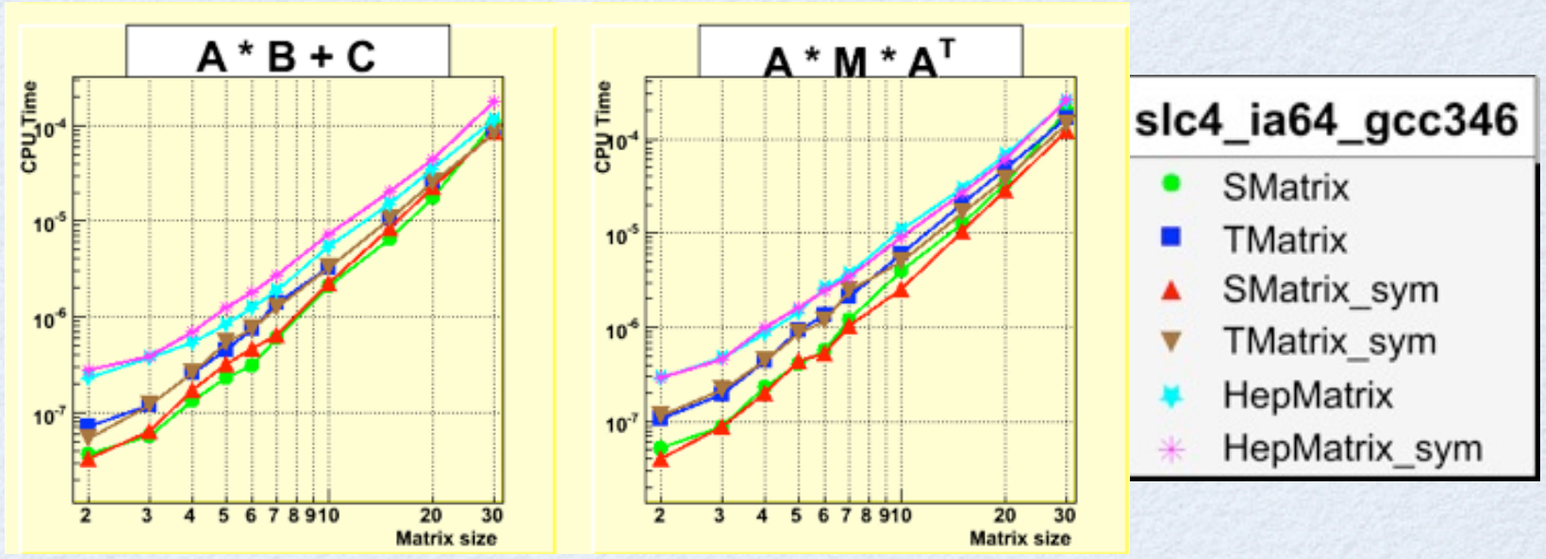
Advantage in performances using GenVector



SMatrix Package

- Matrix and vector classes of arbitrary type and for fixed (not dynamic) matrix and vector sizes (must be known at compiled time)
  - SMatrix< double, N_1,N_2>
  - SVector< double, N>
- Complementary to TMatrix
- Optimized for small sizes (N < 10)
  - use expression templates to avoid temporaries
  - facilitate vectorization and loop un-rooling
- Use by LHC experiments for tracking (Kalman filters)

Large CPU performance gains compared to other matrix packages not using templates



Vc Library

- Vc provides new vector types:
  - Vc::float_v or Vc::double_v
  - float_v::Size will depend on architecture (e.g 8 on AVX)
  - basic operations (+,-,/,*) for these types are supported
  - also basic Math and transcendental functions (sin,cos, log,etc..)
- User can vectorize code without need to use and know the intrinsic instructions

Vc in ROOT

- Tried to use Vc as template argument for physics vectors and for the matrices (SMatrix)
  - SMatrix<Vc::double_v, N> , SVector<Vc::double_v, N>
  - LorentzVector <PxPyPzE< Vc::double_v > >
- when looping on set of vector or matrices, loop size reduced by the size of the Vc type (NITER = NITER / double_v::Size)
- useful to use it in reconstruction (e.g tracking) or simulation applications
  - example: Kalman filter equations for updating error matrix
- Performed some tests a couple years ago
  - mixed results obtained
  - explained as some compiler limitation at that time (gcc 4.4 was used)
- Need to try now with new compiler versions before deciding for inclusion in ROOT
  - if found useful could be added as an optional package to ROOT distribution

AutoVectorization

- Implement code in a way that can be vectorized automatically by the compiler
  - no need to use intrinsic
- CMS is prototype new implementation of mathematical functions (from Cephes)
  - provide vector API
    - double exp(double x) ⇒ void exp_vect(const double *, double *, int)
  - Obtained promising results
    - need to use latest compiler version (4.7) for the vectorization
- These new vector function could be eventually included in ROOT

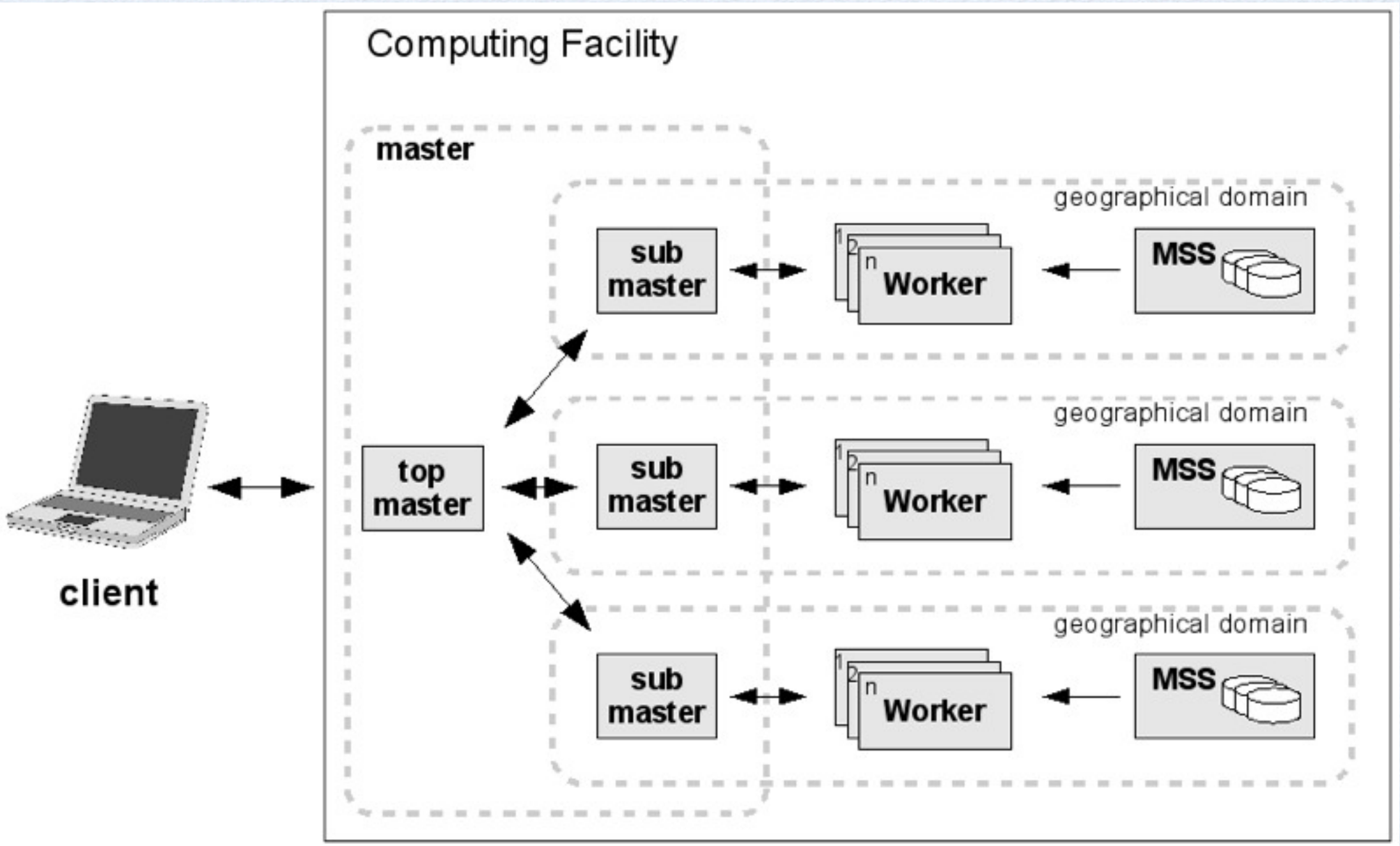
Parallelization of RNG

- Parallelization of pseudo-random numbers generators
  - most used generator are very fast (RanLux is maybe the exception)
  - time in generating random numbers is often not critical in majority of our applications
    - one does much more time consuming things with a random number
- Using the random numbers in parallel application is more problematic
  - many good generators have a very large state
    - e.g. Mersenne and Twister (TRandom3) has state of 624 words (32 bits)
    - this makes them problematic to run on GPU
  - problem in seeding and bookeeping many independent sequences
    - need generator with very long periods, which normally can be obtained only with large states
      - need care in seeding the generators to have really independent states
    - or dedicated parallel generators which allow to jump in the sequence
      - need to know in advance max length of each stream

New Parallel Random Numbers

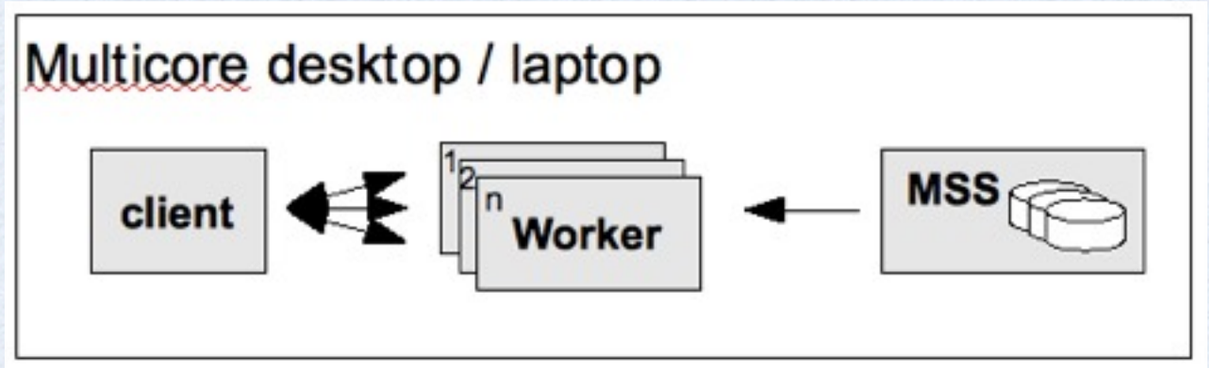
- New class of PRNG based on counters without a state (J. Salmon et al., see *http://www.thesalmons.org/john/random123/papers/random123sc11.pdf*)
  - based on a counter n and key k
    - k : $x_n = f_k(n)$
  - instead of an iterative sequence
    - $x_i \rightarrow x_{i+1} = f(x_i)$
  - they have **no state** (can be easily used in parallel applications)
  - generators derived from algorithms used in cryptography
  - awarded best paper at the SC11 conference
- These new generators pass the most stringent tests
  - BigCrush of TestU01 from L'Ecuyer
- but are empirical generators (lack of mathematical analysis) and based very complex algorithm
- interesting to watch this new development

PROOF Architecture



PROOF Lite

- PROOF optimized for multicore machines
- Zero-config setup (no config files, no daemons)



- Same API, same code as for standard PROOF
- Very popular, especially in ATLAS

PROOF on Demand

- Tool-set developed by A.Manafov (GSI) to setup PROOF on any resource management system
- Uses RMS to start the worker daemons; master runs on a dedicated machine, e.g. the desktop
- Easy installation
- RMS job managers provided via plug-in: gLite, Condor, PBS, OGE, LSF, ssh
- http://pod.gsi.de

PoD

- PoD main ingredient for
  - PEAC *(PROOF Enabled Analysis Cluster)*
    - Cluster management based on PoD
  - PROOF on the GRID
    - via gLite (ATLAS Tier2)
    - full integration (AliEN)

Improvements in PROOF

- Improved connection layout, remove single point of failure
- Result merging optimization (parallel tree merging)
- Packetizer redesign
  - Large number of events of ~same size (analysis)
  - Small number of events of different sizes (reconstruction)
  - Non-homogenous data distribution (local, networked data)
  - Dynamically handle new work, i.e. new files
- Dynamic Multi-Master setup
  - Improve scalability on a large cluster (grid, clouds, many-cores)
  - Elastic sessions
  - Large scale data set management

Summary

- Working in ROOT in improving support for parallel architecture
  - planning to improve thread-safety of code
  - provide support for parallel algorithms
    - example of Minuit, provide a version which can be used in parallel without changing user code
  - opportunity to work on optimize and parallelize algorithm at the same time
- Started to investigate parallelization in vector and matrix classes
  - Vectorization looks promising
  - parallelization of large matrix computations less relevant for our community

New Concurrency Forum

- Parallelization activities happening within a new forum on concurrency model and framework organized by the CERN SFT group
  - participation from Fermilab and LHC experiments
  - regular meeting every two weeks
- Development of various prototypes (demostrators) in 2012
- Adaptation and porting to LHC experiments during 2013 shutdown
  - development of some functional components which can be integrated in current experiment frameworks
- see http://concurrency.web.cern.ch/