Hybrid Parallelization of Maximum Likelihood Fitting with MPI and OpenMP

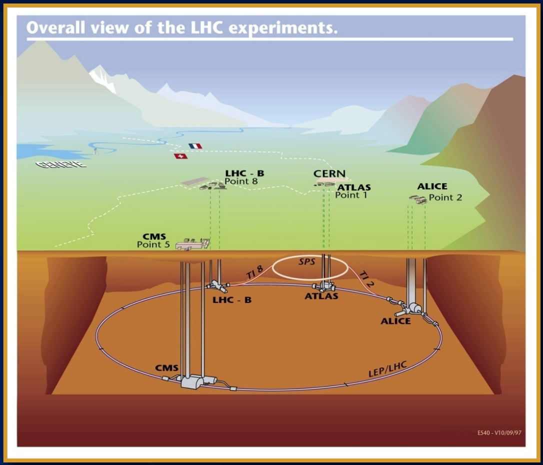
Sverre Jarp, *Alfio Lazzaro*,
Andrzej Nowak, Liviu Valsan

CERN openlab

INTERNATIONAL
SUPERCOMPUTING CONFERENCE
ISC'12 The HPC Event

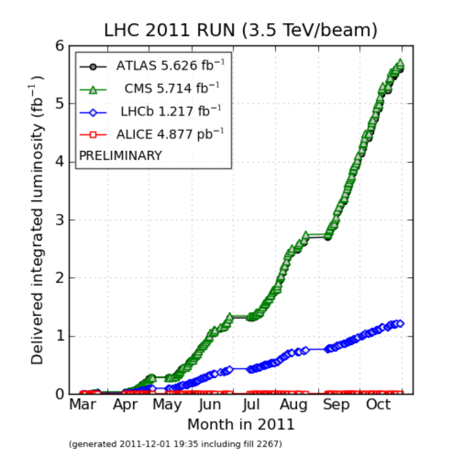
Large Hadron Collider (LHC)

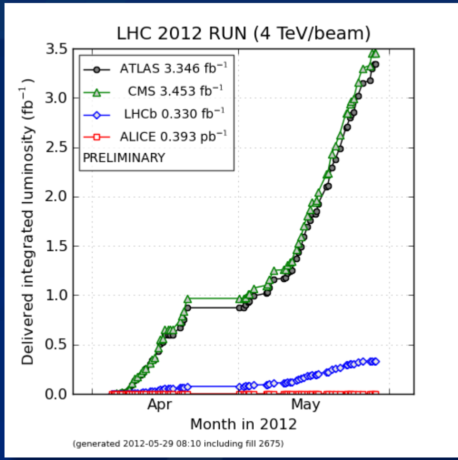
- **The biggest machine ever built**
  - 27 km, 100 meters below ground



- **Activities started in 2009**
  - Highest energy in an accelerator
  - Large data sample of recorded collisions (events) available for high energy physics (HEP) measurements

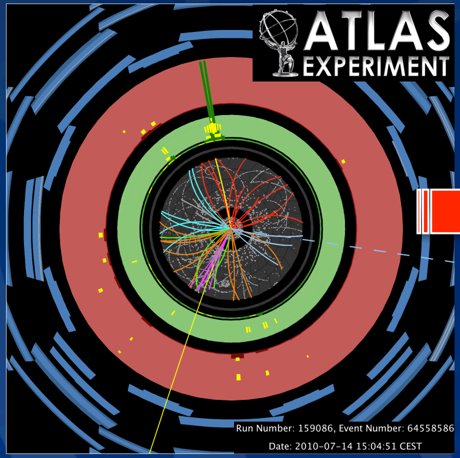
> 10^7 collisions per seconds, about 200-300 events recorded per second per experiment: **~300 MB/s (~5 PB/year)**

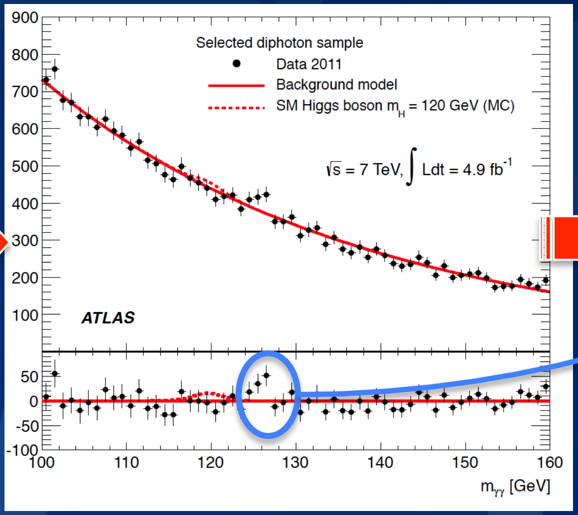


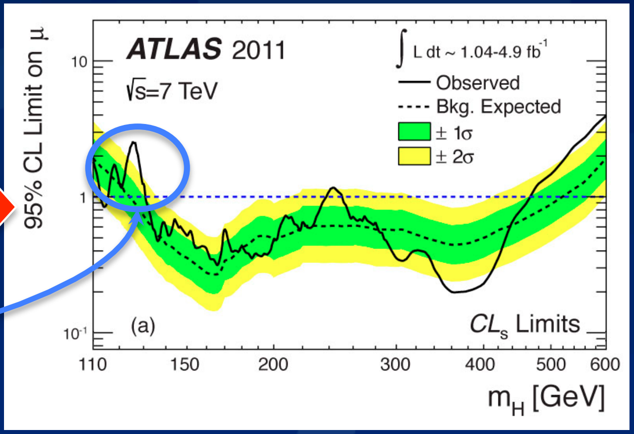


Data Analysis

- **Huge quantity of data collected, but most of events are due to well-know physics processes**
  – New physics effects expected in a tiny fraction of the total events: few tens
- **Crucial to have a good discrimination between interesting events and the rest, i.e. different species**
  – Data analysis techniques play a crucial role in this “fight”







Likelihood-based analysis

- **Specific variables (observables) combined by using multivariate analysis techniques, e.g. *Likelihood-based***
  - Each observable described by a probability density function $\mathcal{P}$
- **HEP package to build likelihood function models: ROOT/RooFit (http://root.cern.ch/drupal/content/roofit)**
  - C++ code
  - All data in the calculation are in double precision floating point numbers
- **We present the results based on a prototype of RooFit, developed at CERN openlab, that enables several optimizations and parallelization techniques applied to Maximum Likelihood fits**

Maximum Likelihood Fits

- **For estimating parameters over a data sample, by minimizing the Negative Log-Likelihood (NLL) function**

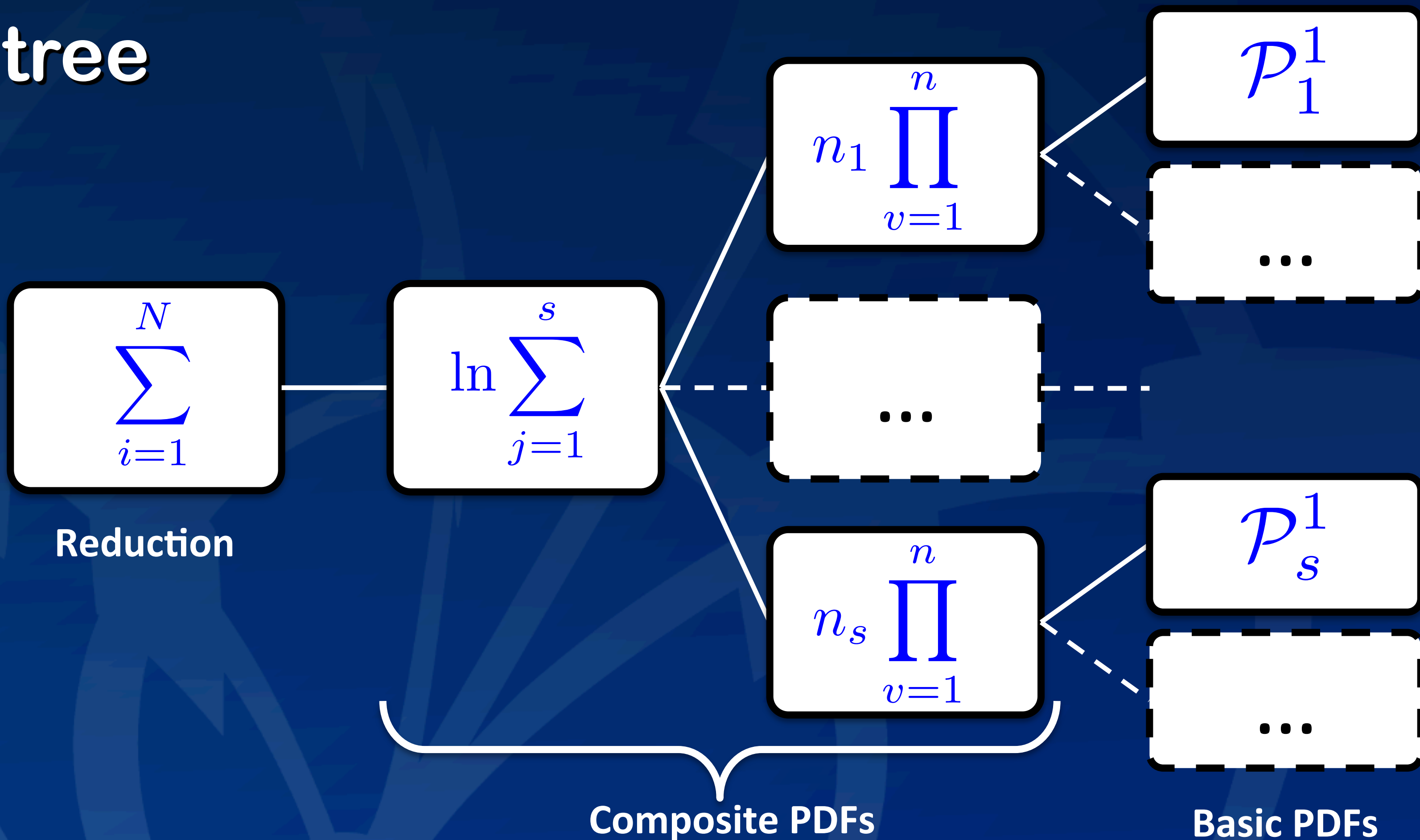
$$NLL = \sum_{j=1}^{s} n_j - \sum_{i=1}^{N} \left[\ln \sum_{j=1}^{s} \left(n_j \prod_{v=1}^{n} \mathcal{P}_j^v (x_i^v | \hat{\theta}_j) \right) \right]$$

N number of events
$\hat{x}_i$ set of observables for the event i
$\hat{\theta}$ set of parameters
n observables
s species
n_j number of events belonging to the species j

- **The procedure of minimization can require several evaluation of the NLL**
  - Depending on the complexity of the function, the number of observables, the number of free parameters, and the number of events, the entire procedure can require long execution time
  - **Mandatory to speed-up the evaluations of the NLL**

Algorithm Tree

- We can visualize the NLL evaluation as a tree

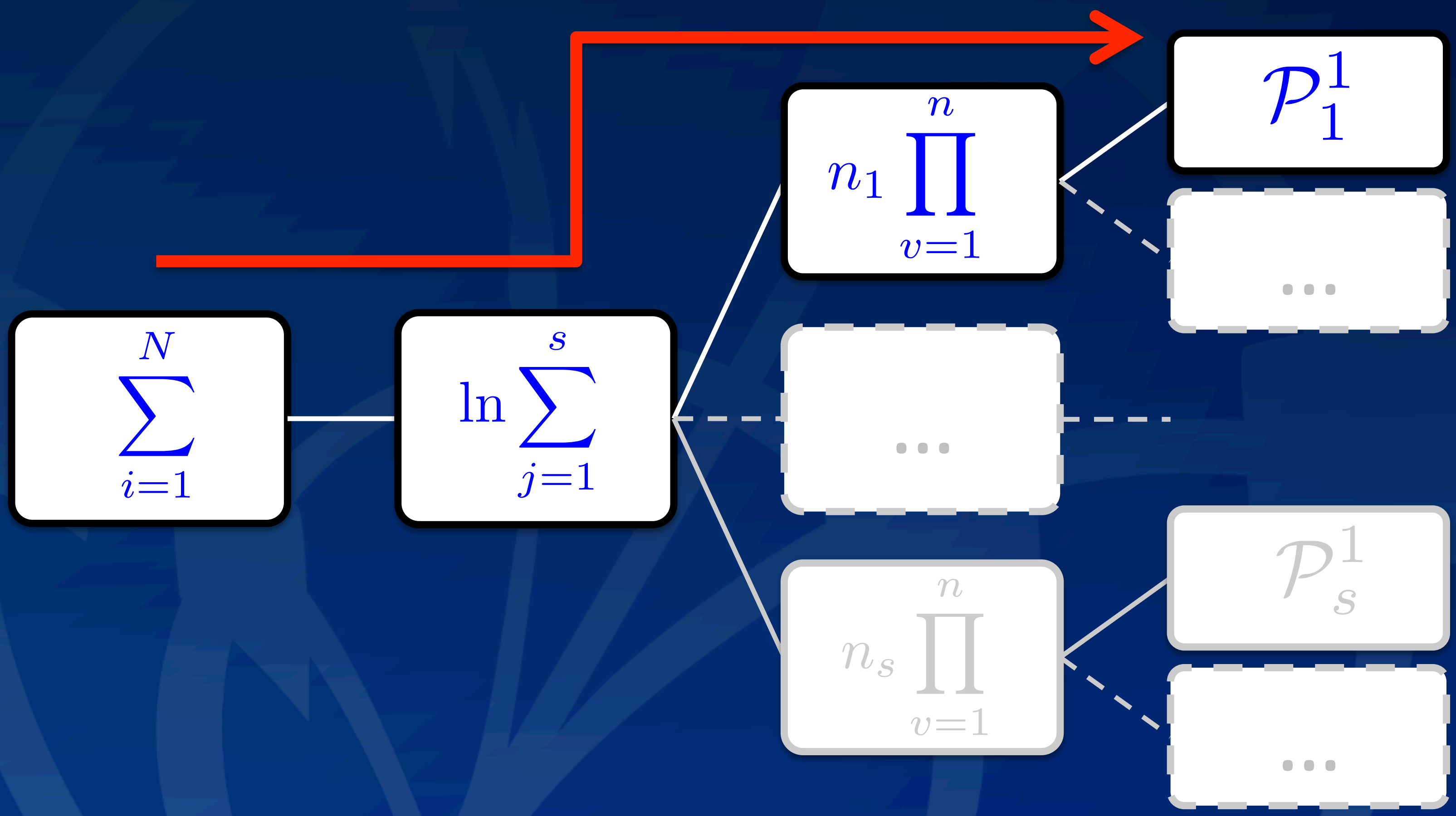


Algorithm Description

- **Data are organized in memory in vectors**
  - A vector for each observable
  - Read-only during the NLL evaluation
- **Values of the PDFs evaluated with loops**
  - One loop per each PDF over the values of the observables
    - A loop iteration per each input event
  - Use Intel compiler for the **auto-vectorization** of the loops (using Intel SVML library)

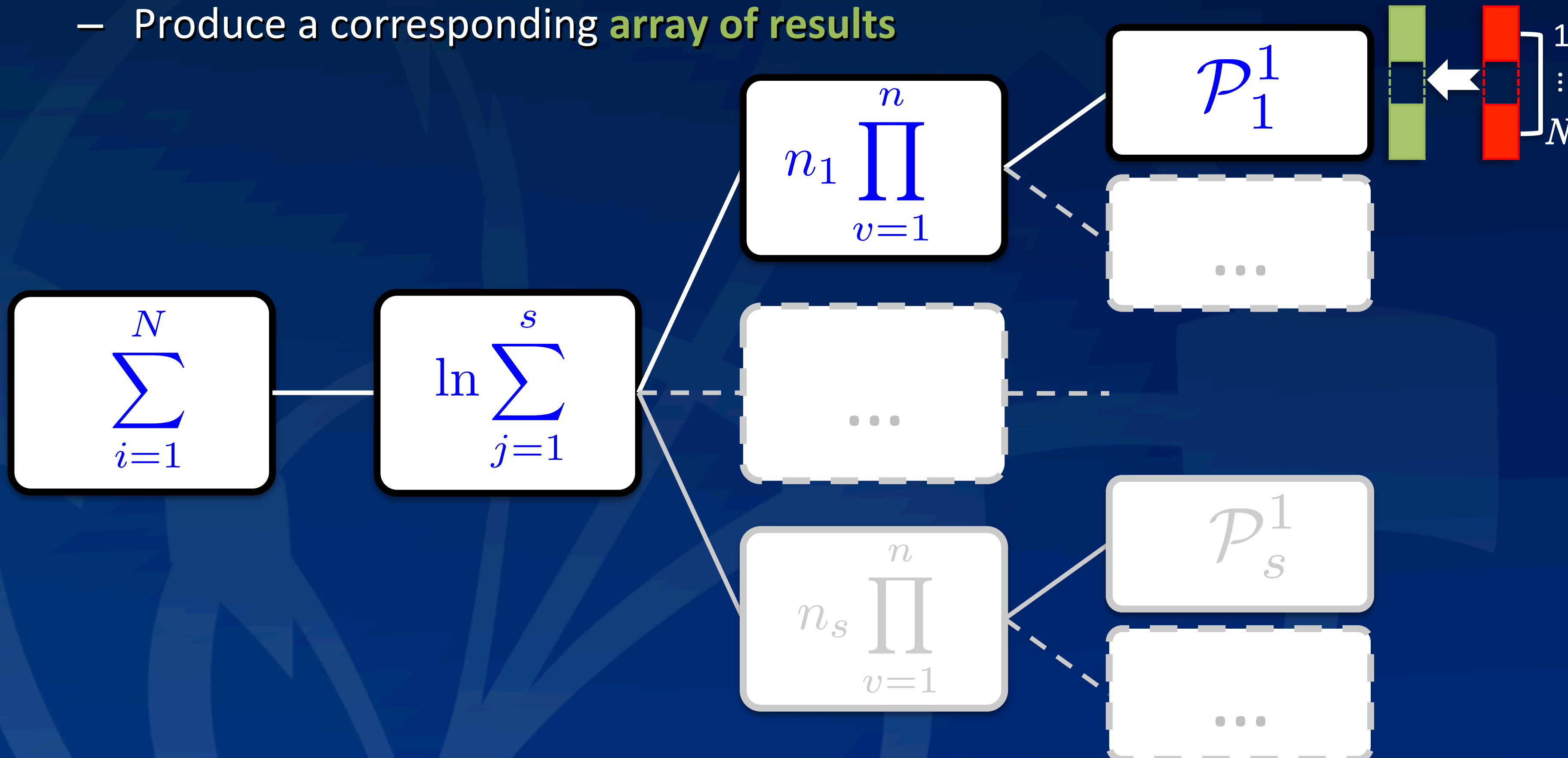
Algorithm Evaluation

1. Traverse the NLL tree up to the first leaf (basic PDF)



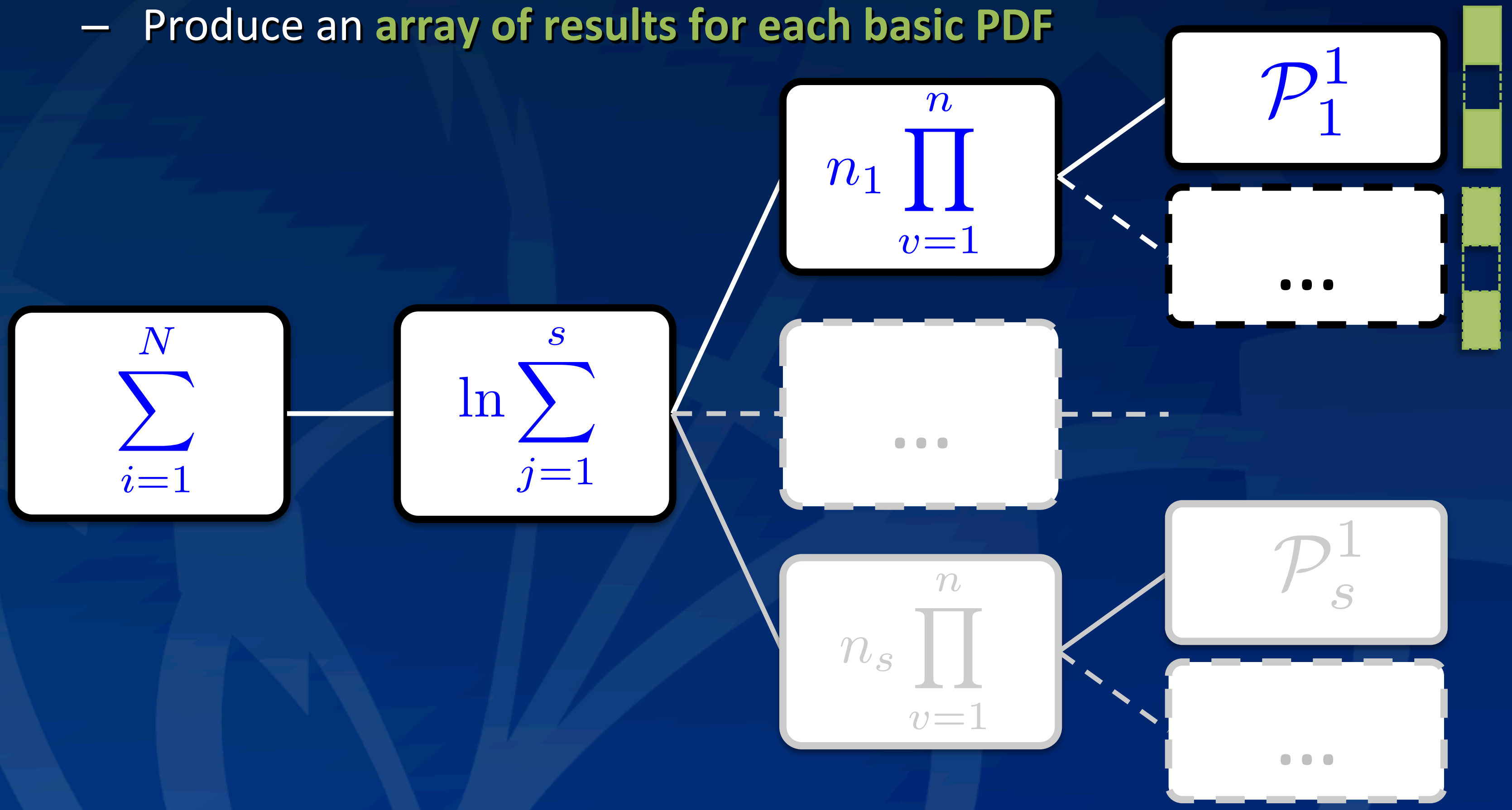
Algorithm Evaluation

2. Loop over the *N* **events** and evaluate the PDF for each event
   - Produce a corresponding **array of results**



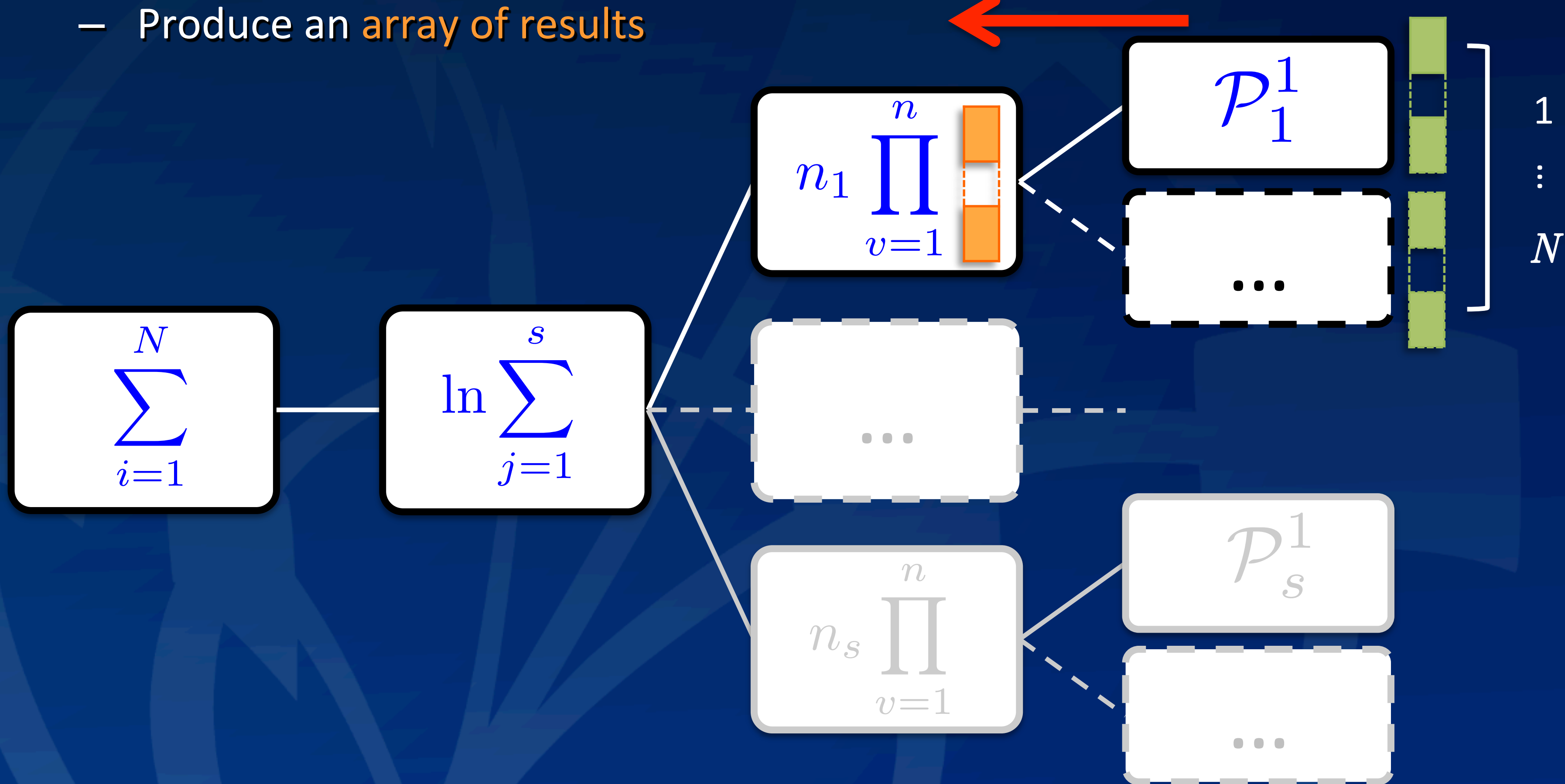
Algorithm Evaluation

3. **Repeat the evaluation for all basic PDF in a composite PDF**
   – Produce an **array of results for each basic PDF**



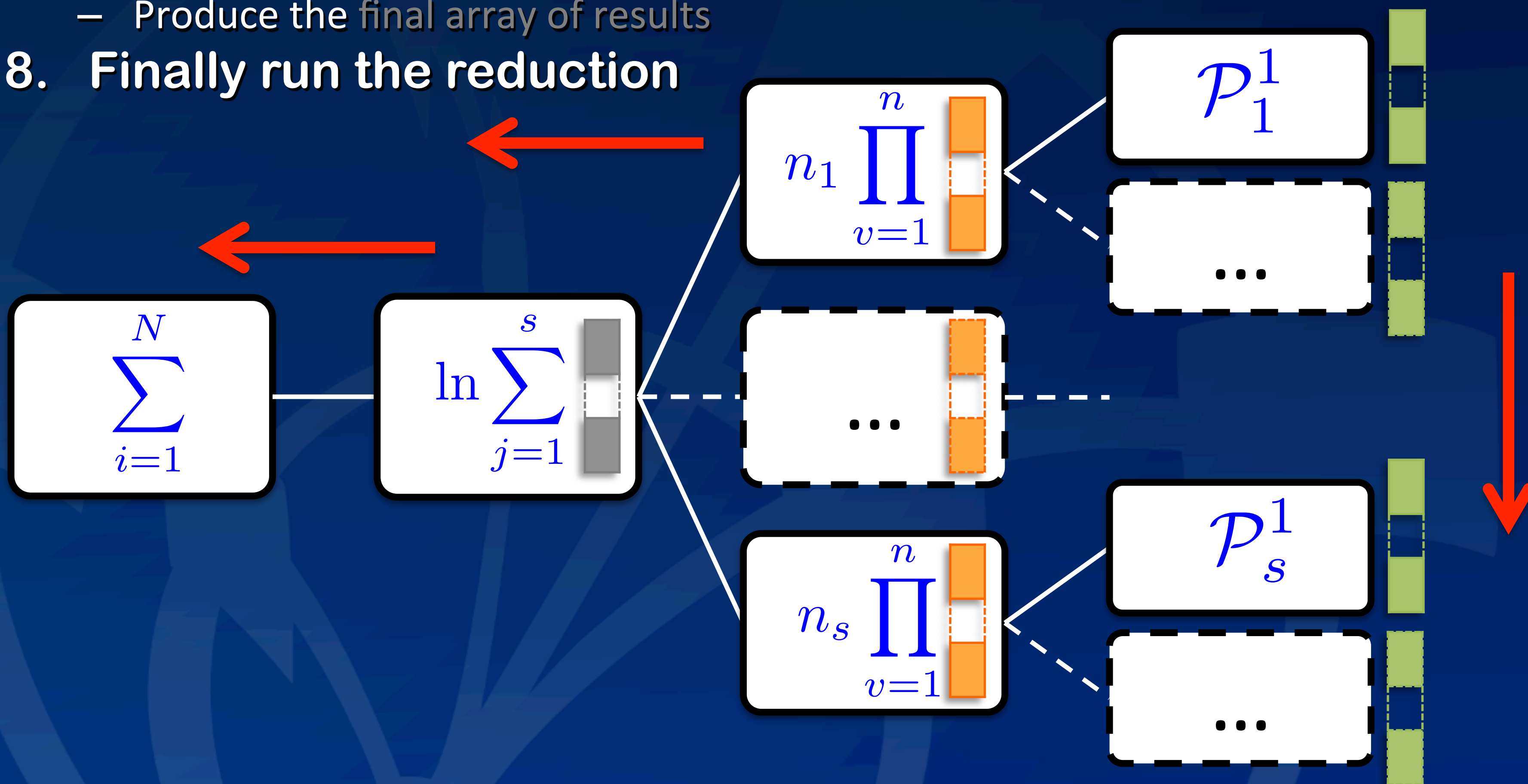
Algorithm Evaluation

4. **Combine the array of results for the composite PDF**
   - Loop over the array of results of the basic PDF
   - Produce an array of results



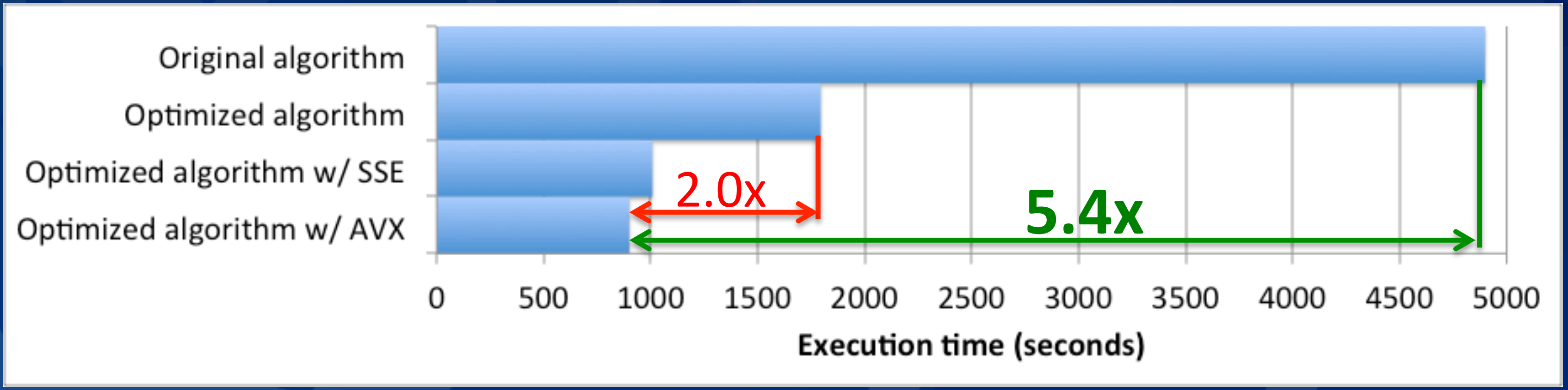
Algorithm Evaluation

5. Repeat for all composite PDFs
7. Loop over the array of results
   – Produce the final array of results
8. Finally run the reduction



Sequential performance

- **Optimization with respect to original RooFit algorithm**
  - Reduce the number of virtual functions calls
  - Inlining of the functions
  - **Prefer data-flow rather than control-flow**
- **Testing on dual-socket Sandy Bridge-EP server, CPU E5-2680 @ 2.7GHz (Turbo OFF), dual socket, 8*2 cores**
- **Intel C++ compiler version 12.1.0**
- **Input data is composed by 1,000,000 events per 3 observables, for a total of about 24MB; results are stored in 29 vectors of 1,000,000 values, i.e. about 230MB**



Parallelization: MPI+OpenMP

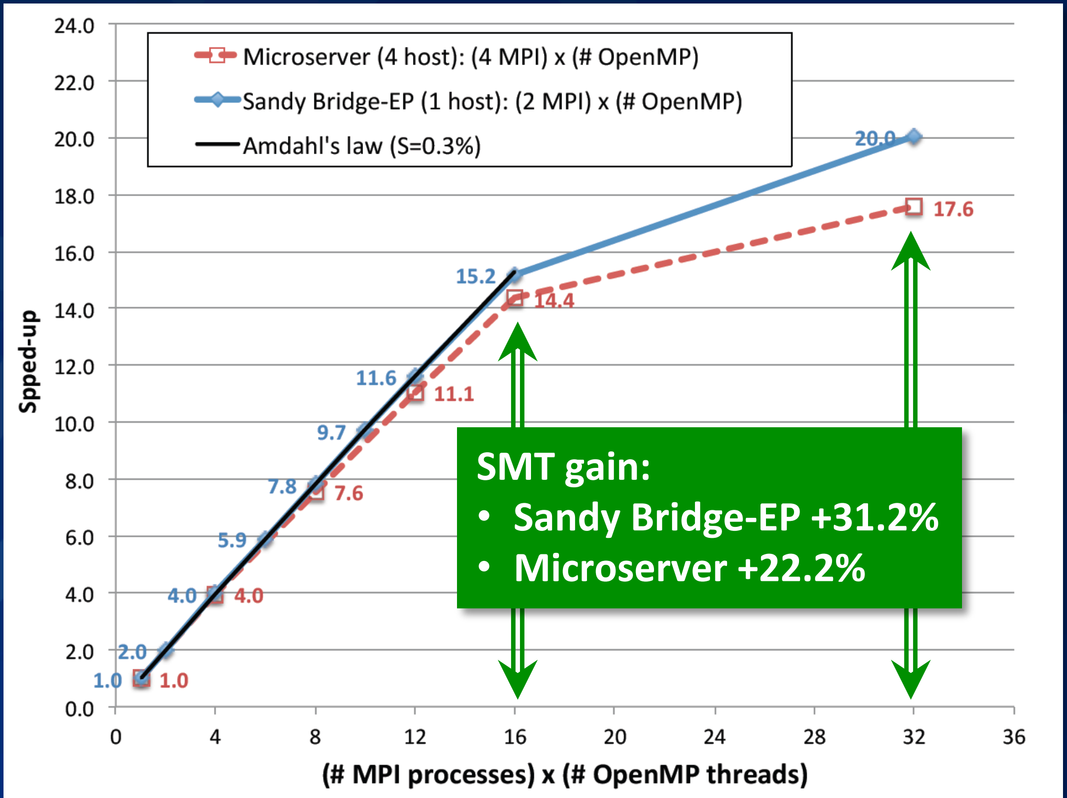
- **Each MPI process runs several OpenMP threads**
- **Decomposition of the input events (and corresponding loop iterations) in chunks**
  - Easy to balance: each chunk is composed by an equal number of events (maximum one event of difference)
  - Decomposition in two steps:
    - Step 1 for the MPI processes
    - Step 2 for the OpenMP threads belonging to each MPI process. A single OpenMP parallel region in common for all loops for each NLL evaluation
- **Input data are shared in memory between OpenMP threads**
- **Parallel reduction in two steps:**
  - Step 1 for the OpenMP threads belonging to each MPI process
  - Step 2 between the MPI processes (the only MPI communication)

Parallel performance

- **Same example as before**
  - Sequential portion 0.3%
- **Intel MPI v4.0.3**
- **Testing on DELL C5220 Microserver, 4 hosts single-socket Sandy Bridge, CPU E3-1280 @ 3.50GHz (Turbo OFF), 4 cores, 8MB L3 cache (2MB per core)**
  - One Ethernet link per host @ 1Gb
- **Process topology to maximize the number of hosts, with a single MPI process per each host**
- **Comparison of the performance with the Sandy Bridge-EP system**
  - Same number of total cores (16)
  - 2 MPI processes with corresponding OpenMP threads pinned within the sockets
  - **Smaller L3 cache size on the CPU version (20MB, 2.5MB per core)**

Parallel performance

- **Perfect scalability: 14x-15x with 16 threads**
  - Using SMT threads: 20x for Sandy Bridge-EP, 18x for Microserver
- **Main limitation to scalability comes from the L3 cache size**
  - Negligible penalty for the Sandy Bridge-EP
  - Microserver: −4.5% per 12 threads, −5.5% per 16 threads
- **Analysis of the MPI communication time shows no penalty to the scalability**



Parallelization on Intel MIC (Knights Corner)

- **Openlab involved in the evaluation since the beginning of the project**
  - Evaluation of the previous Intel MIC “Knights Ferry”
- **Very easy porting of the application**
  - Just recompile it for MIC…
  - Use only the OpenMP parallelization directly on the card
- **Able to exploit the new 512-bit wide SIMD registers for vectorization**
  - Auto-vectorization by the Intel compiler
- **Perfect scalability, very close to the expectation**
  - Able to exploit all cores and hardware threads

Conclusion

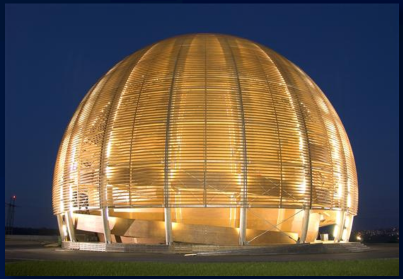
- **Redesign the algorithm to exploit optimizations**
  - Data-flow versus control-flow
- **Vectorization is crucial to get performance**
  - A good compiler can help a lot
- **Good scalability, close to the expectation**
  - Low impact by MPI and OpenMP overheads
- **Code under validation by the HEP community**
- **Porting on Intel MIC using MPI as host-device communication approach**
- **Some references:**
  - S. Jarp *et al.*, *Evaluation of the Intel Sandy Bridge-EP server processor*, March 2012, CERN-IT-Note-2012-005
  - S. Jarp *et al.*, *Parallel Likelihood Function Evaluation on Heterogeneous Many-core Systems*, August 2011, CERN-IT-2011-012

THANK YOU

Q & A

CERN
openlab

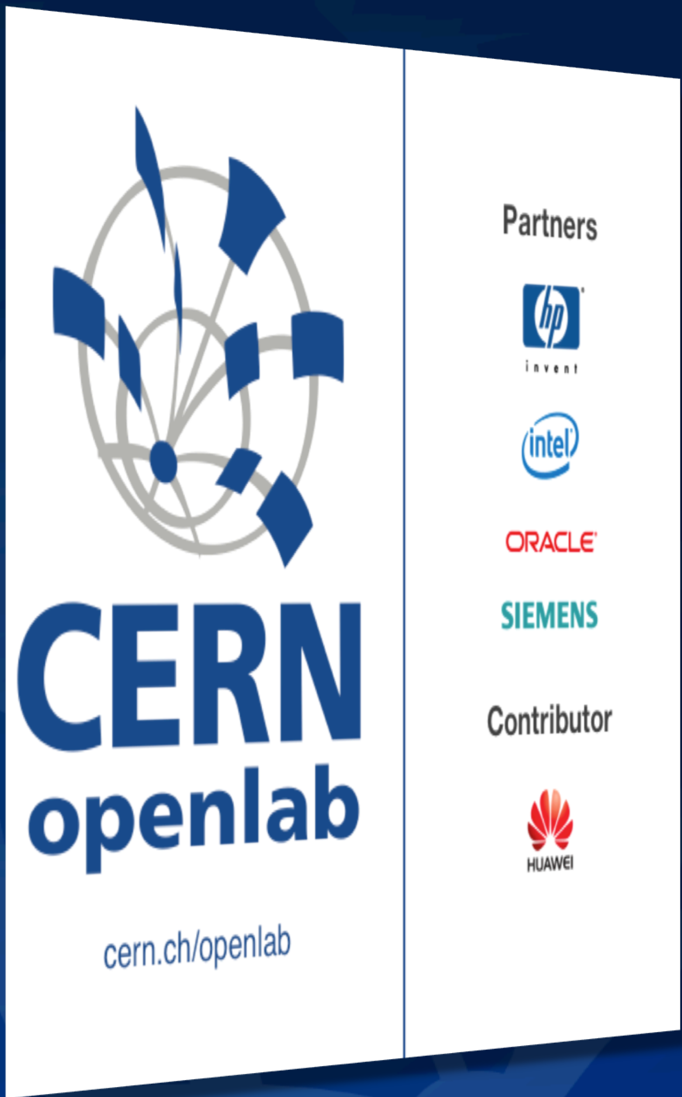
About CERN



- **CERN is the European Organization for Nuclear Research in Geneva**
  - Particle accelerators and other infrastructure for high energy physics (HEP) research
  - Worldwide community
    - 20 members states (+ 3 foreseen members)
    - Observers: Turkey, Russia, Japan, USA, India
    - About 2300 staff
    - >10,000 users (about 5000 on-site)
    - Budget (2011) 1,000MCHF

- **Birthplace of the World Wide Web**

CERN openlab



- **CERN openlab is a partnership between CERN and leading ICT companies**
  - Its mission is to accelerate the development of cutting-edge solutions to be used by the worldwide LHC community
- **The Platform Competence Center of the CERN openlab has worked closely with Intel for the past decade and focuses on:**
  - Many-core scalability
  - Performance tuning and optimization
  - Benchmarking and thermal optimization
  - Teaching