netherlands eScience center

Automated Parallel Calculation of Collaborative Statistical Models in RooFit

Patrick Bos

IEEE eScience, Amsterdam, 31 October 2018

Physics: Wouter **Verkerke** (PI), Vince **Croft**, Carsten **Burgard**

eScience: Patrick **Bos** (yours truly), Inti **Pelupessy**, Jisk **Attema**

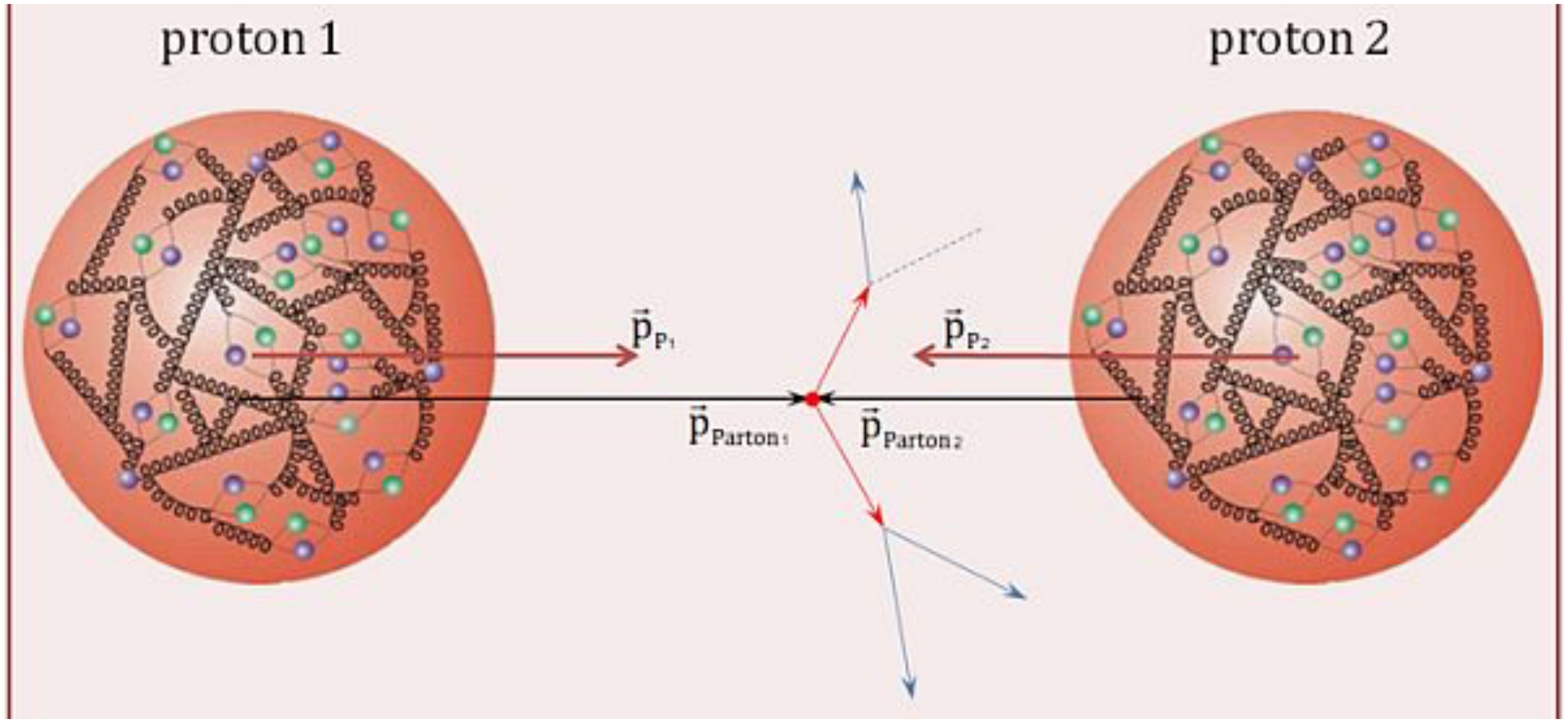


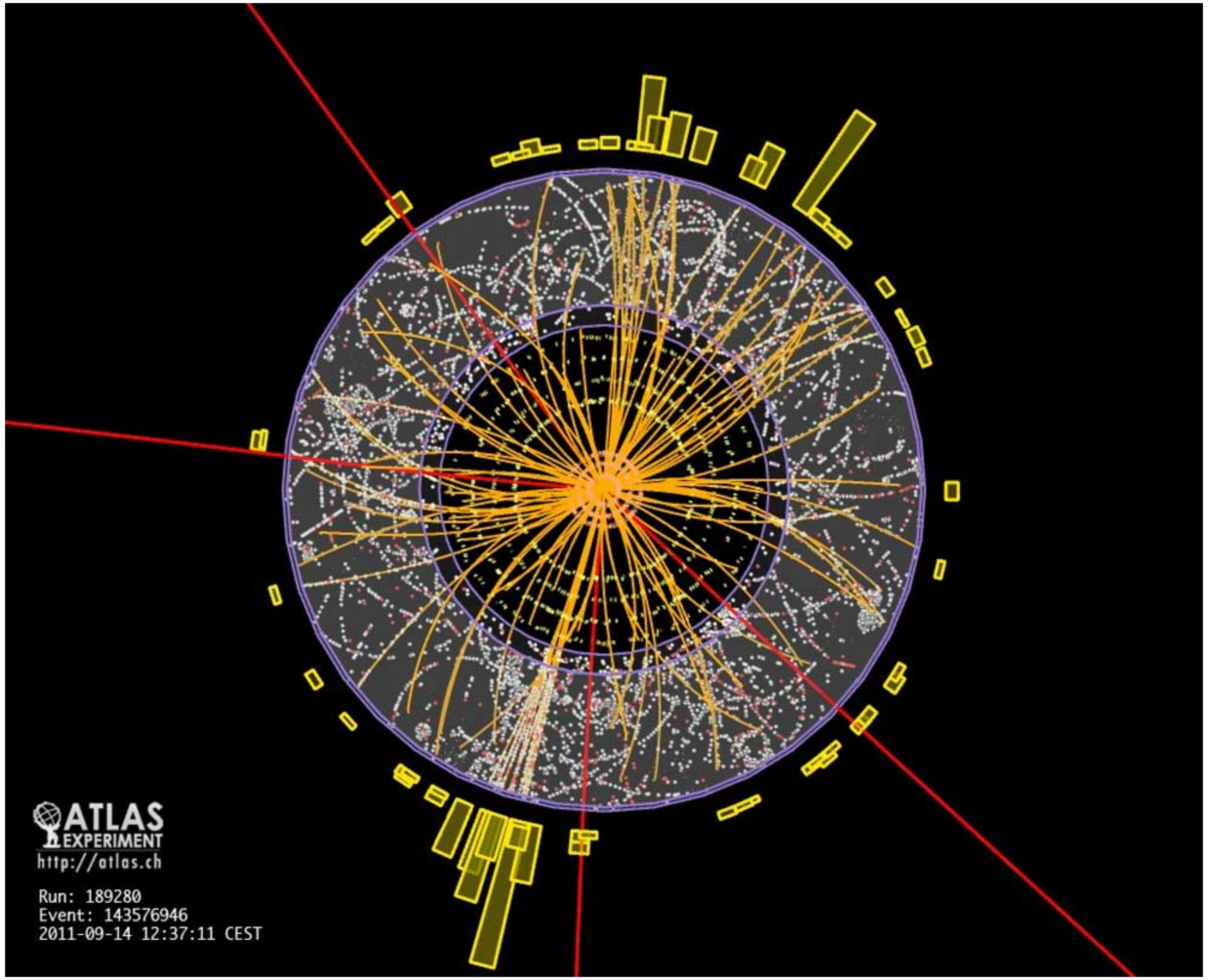


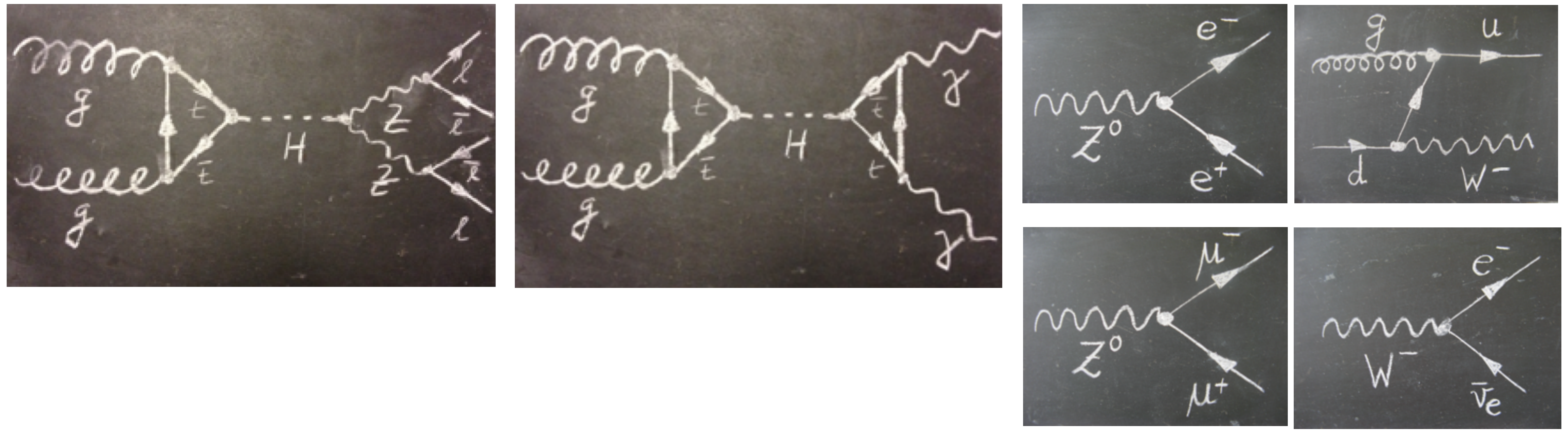

Nikhef ATLAS EXPERIMENT dianahep netherlands eScience center

Particle physics

High energy proton collisions





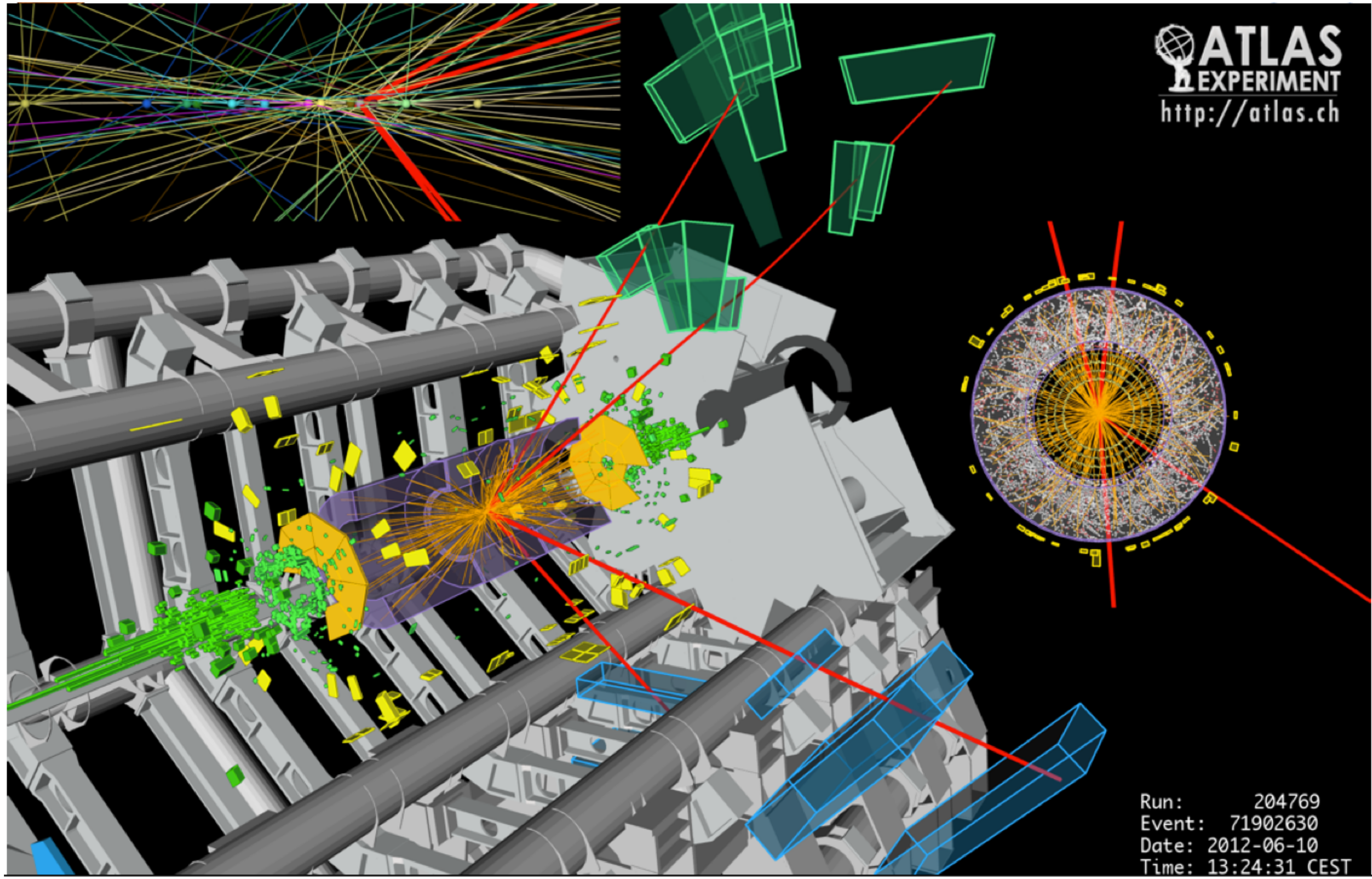


Images from http://atlas.physicsmasterclasses.org

LHC @ CERN

- ATLAS, CMS
- LHCb

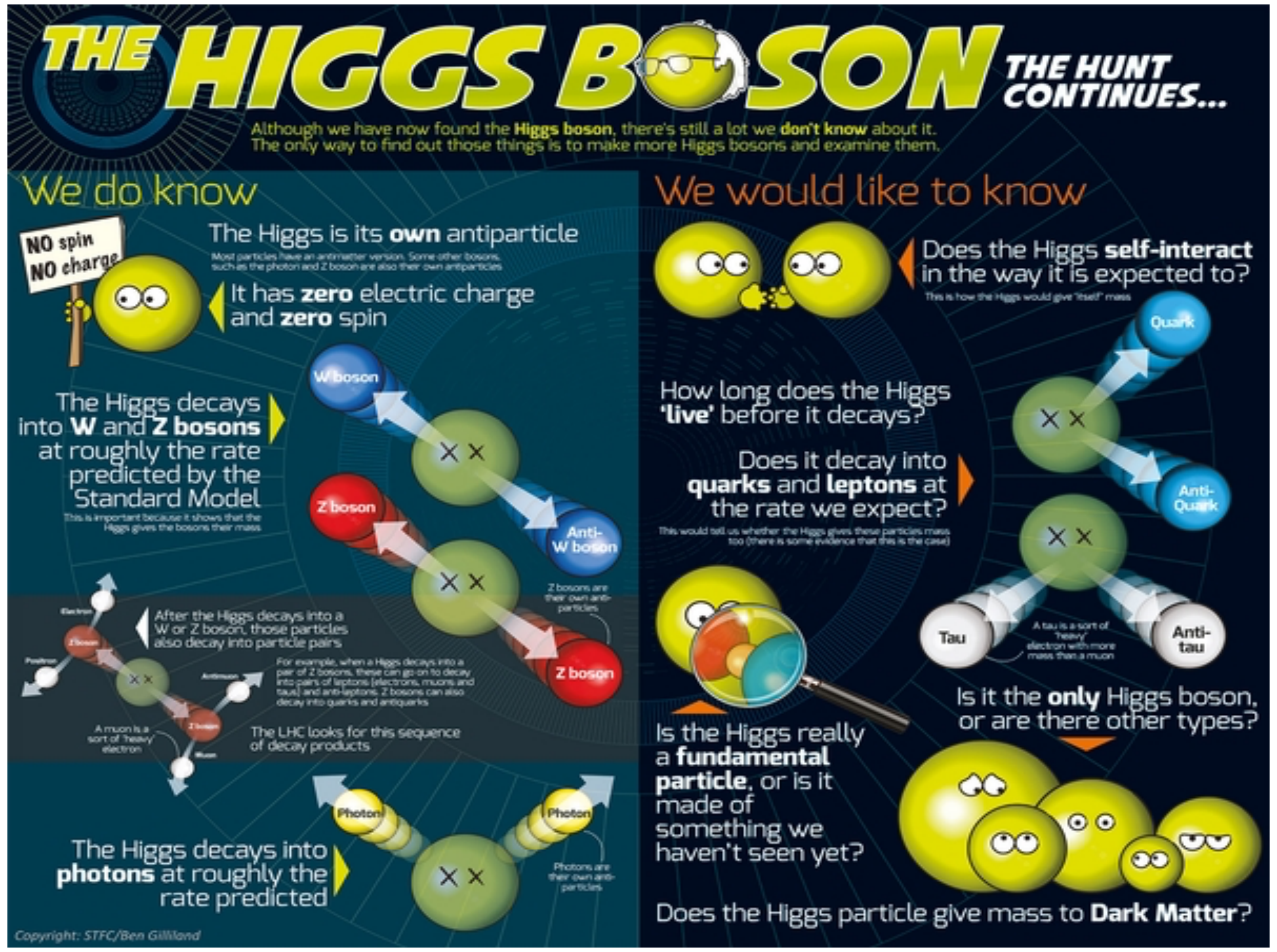
10 PB/yr p-p
Reduced to kB-MBs binned & unbinned events

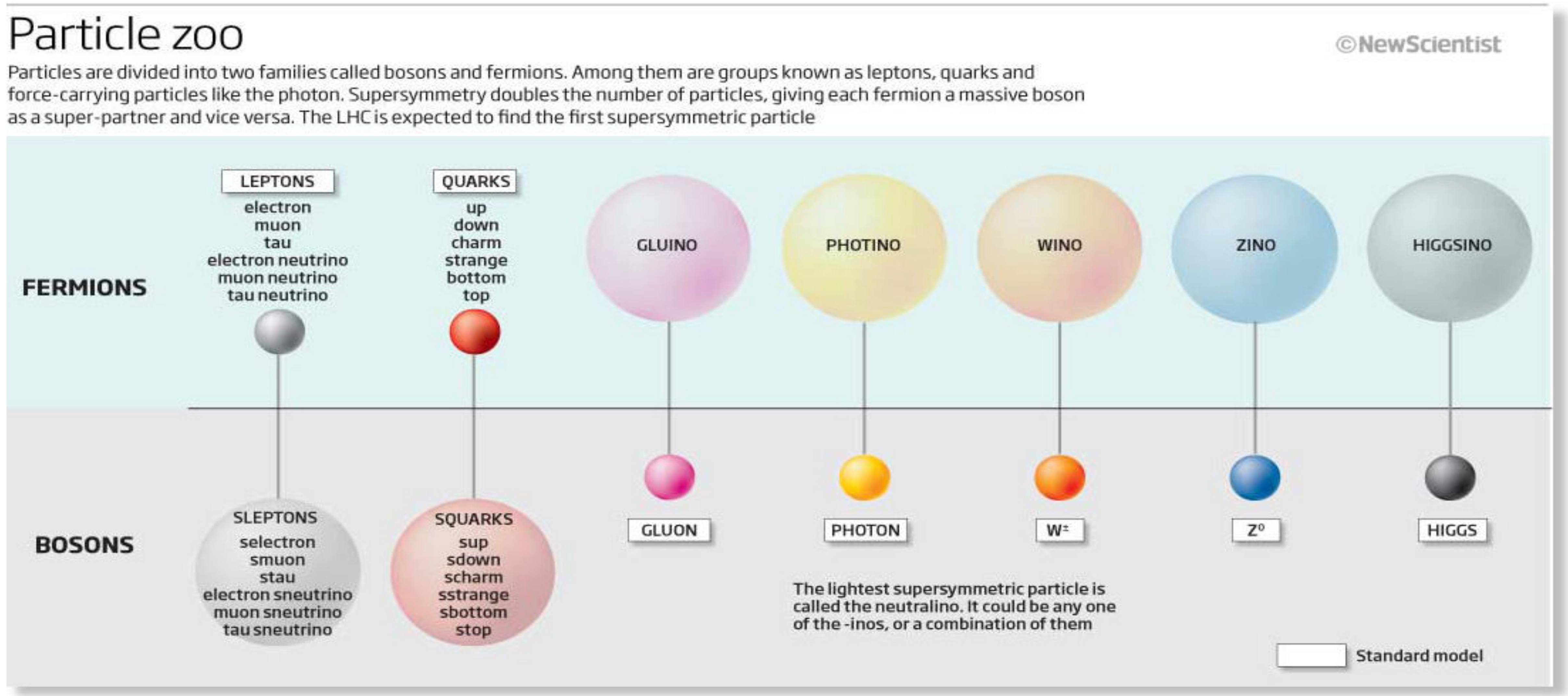


Higgs properties

Physics beyond the Standard Model

- Supersymmetry?
- Dark matter?
- …



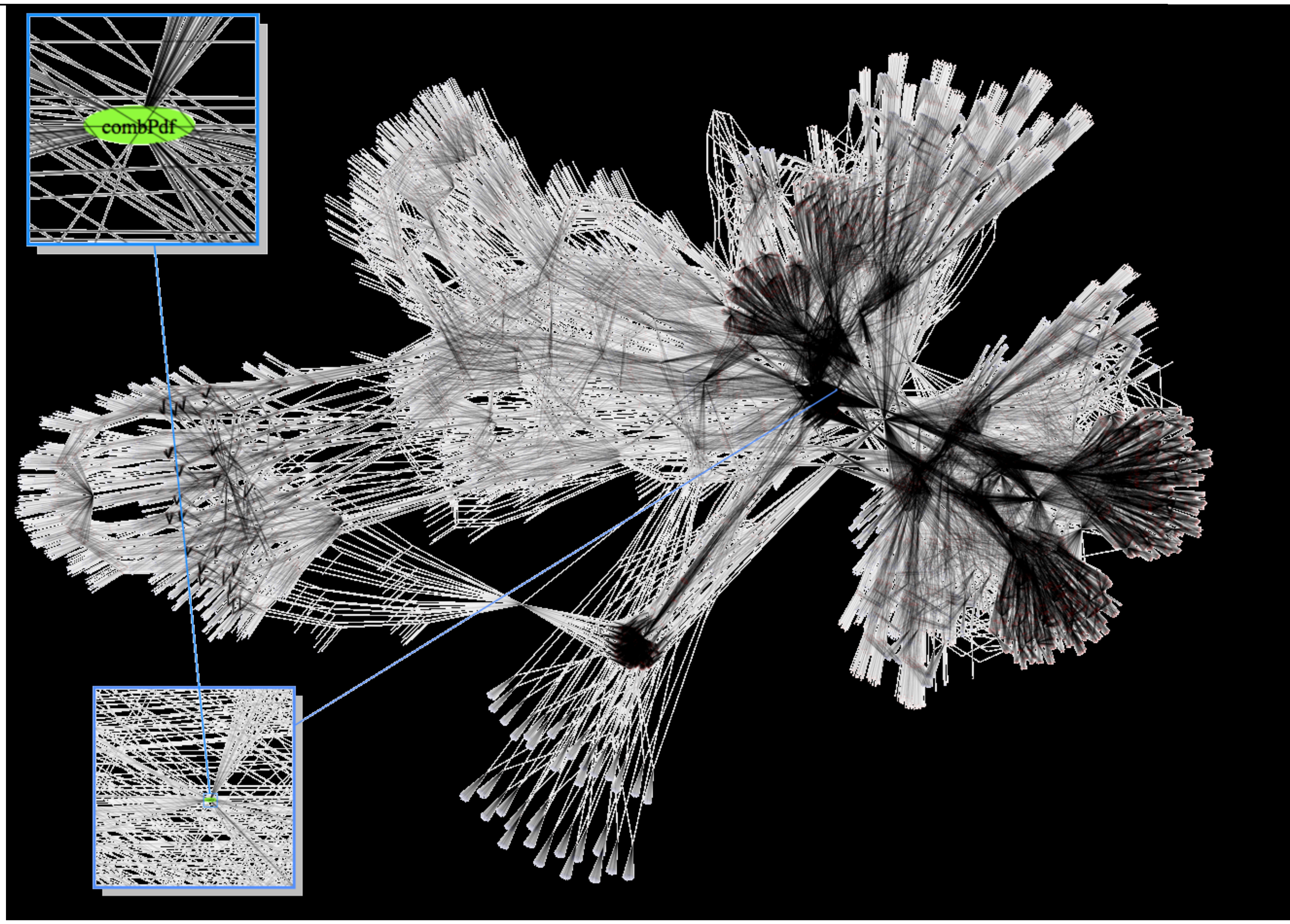


RooFit: Collaborative Statistical Modeling

Collaborative Statistical Modeling

- RooFit: build models together
  - Teams 10-100 physicists
  - Collaborations ~3000
    → ~100 teams
  - Exascale collaboration
    - 10^{15} synaptic connections x 10^{3} brains = 10^{18} (exa)
  - 1 goal
  - Pretty impressive to an outsider

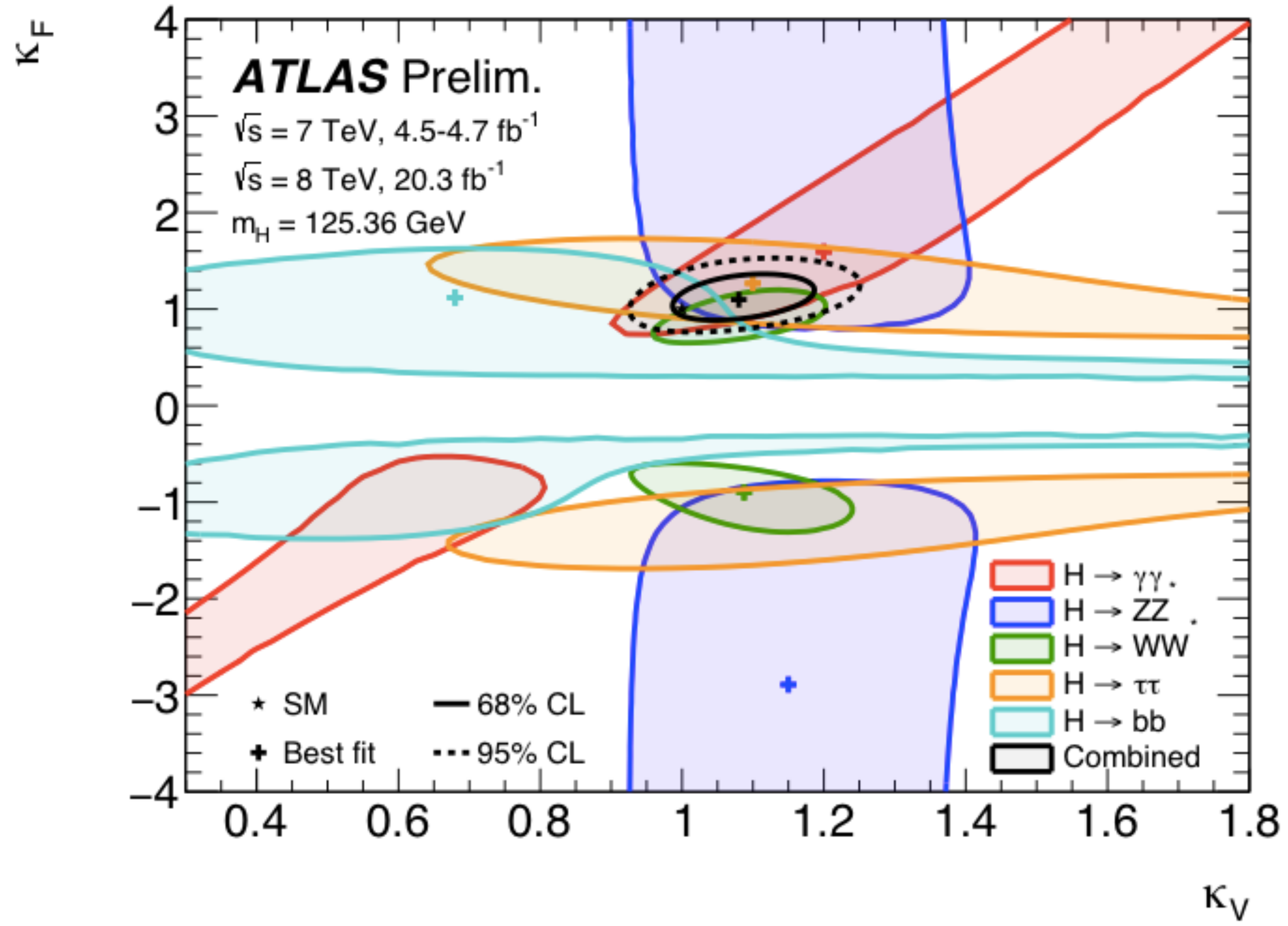
Higgs @ ATLAS
20k+ nodes, 125k hours
Expression tree of C++ objects for mathematical components (variables, operators, functions, integrals, datasets, etc.)
Couple with data, event "observables"



Making RooFit faster (~30x; ~h → ~m)

- More **efficient collaboration**
  - Faster iteration/debugging
  - Faster feedback between teams
- Next level physics modeling ambitions, retaining **interactive workflow**
  1. Complex likelihood models, e.g.
     a) Higgs fit to all channels, ~200 datasets, O(1000) parameter, now O(few) hours
     b) EFT framework: again 10-100x more expensive
  2. Unbinned ML fits with very large data samples
  3. Unbinned ML fits with MC-style numeric integrals

exascale complexity!



Goals and Design: Make fitting in RooFit faster using automated parallel calculation

Serial:

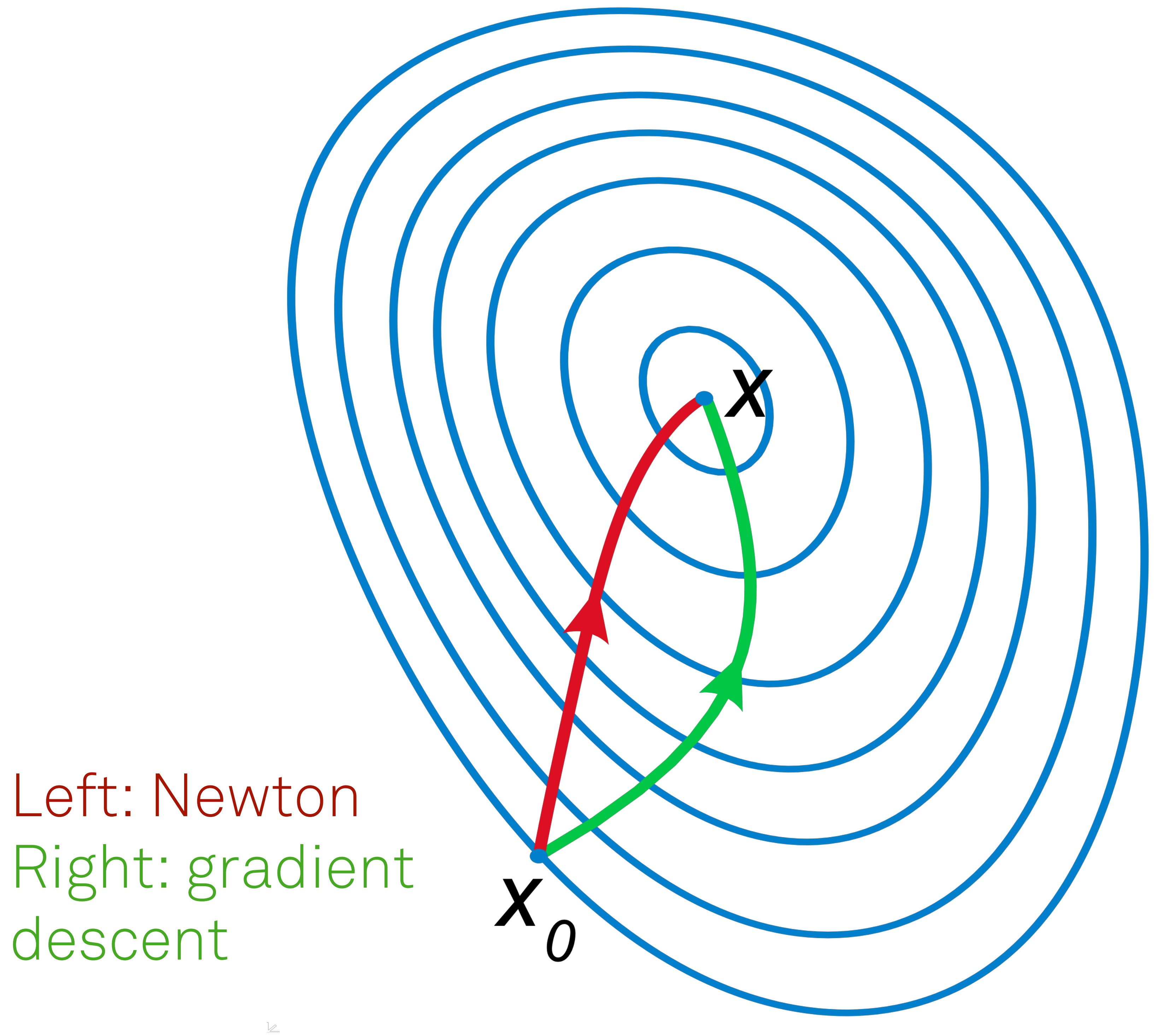
- benchmarks show no obvious bottlenecks
- RooFit already highly optimized (pre-calculation/memoization, MPFE)

Parallel

Fitting method: Quasi-Newton minimization (+ handle multiple minima, physics allowed ranges, confidence intervals)

`Minuit`: minimize PDF $f(x;p)$:

- Quasi-Newton `MIGRAD` method



- Gradient + line-search:
  - gradient for N parameters p: $\frac{df}{dp} \approx \frac{f(p-dp)-f(p)}{dp}$ — 2N f calls → parallelize $\frac{df}{dp}$
  - line-search: descend along gradient direction — 2-3 f calls → parallelize f

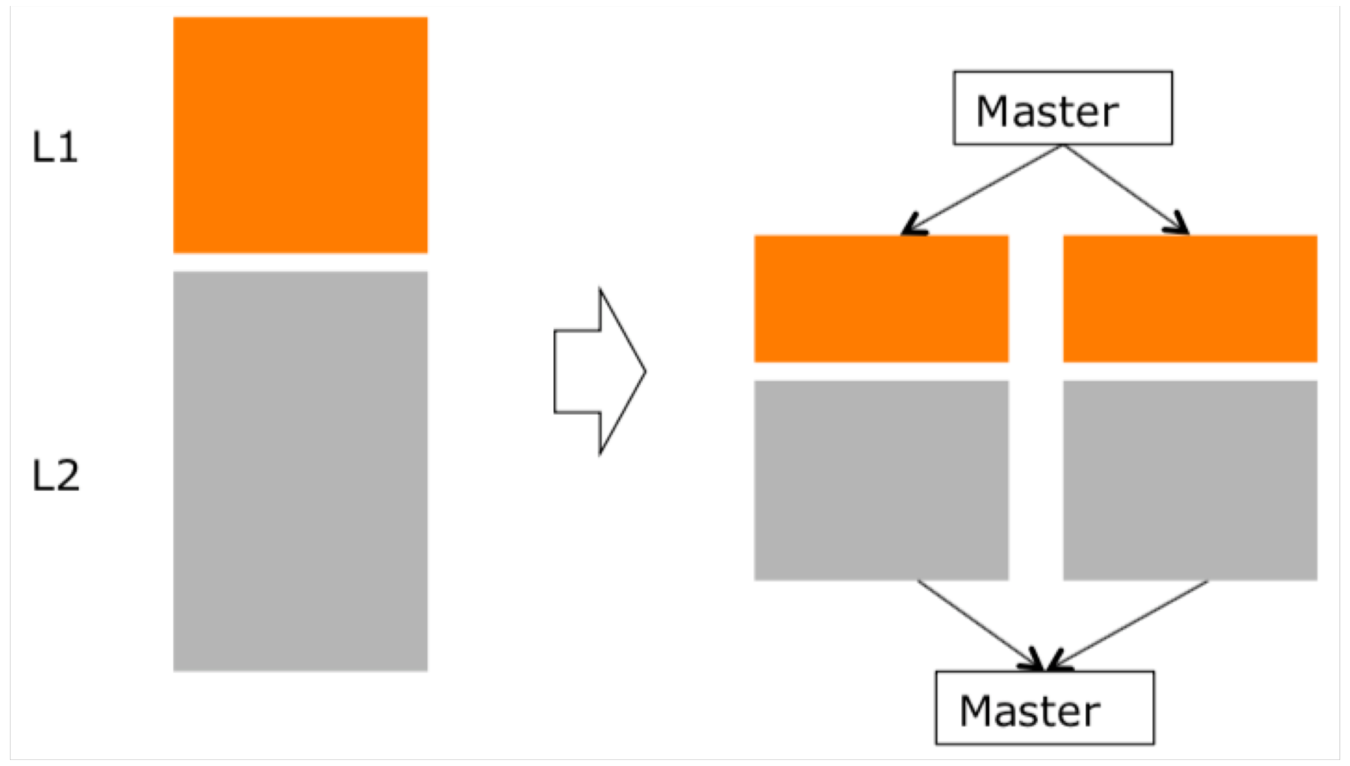
Faster fitting: (how) can we do it?

Levels of parallelism

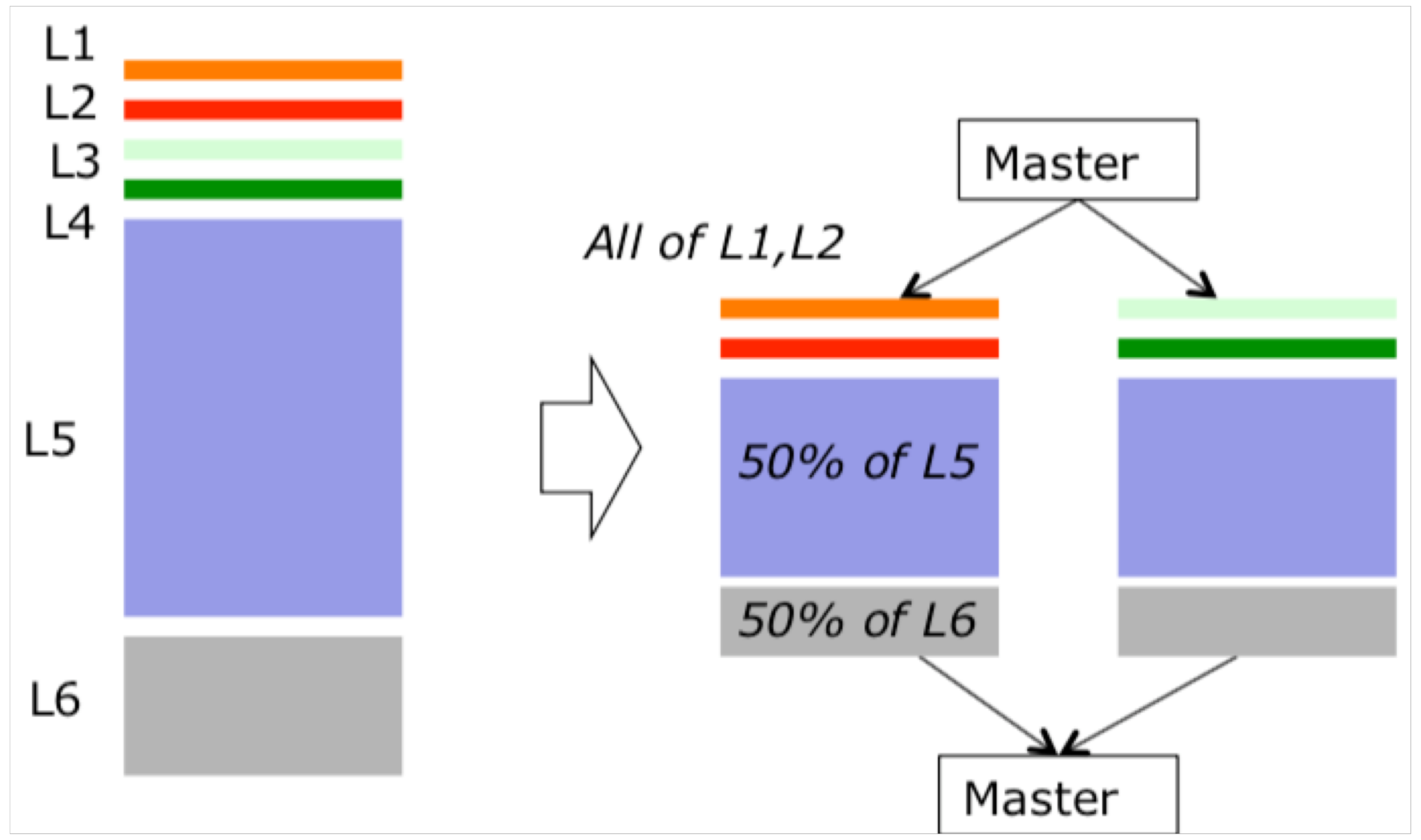
"Vector"

1. Gradient (parameter partial derivatives) in minimizer
2. Likelihood (f)
3. Integrals (normalization) & other expensive shared components

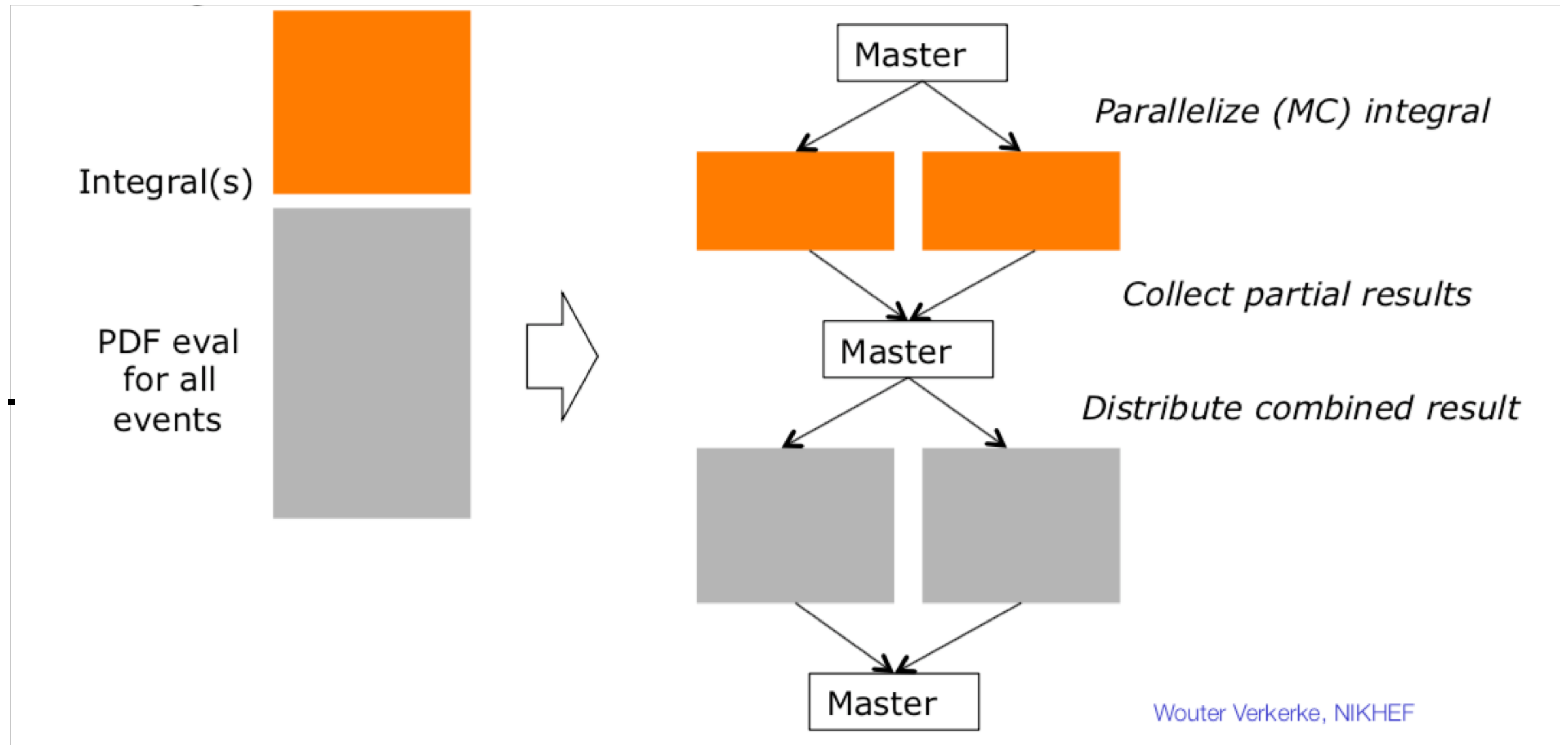
likelihood: events



likelihood: (unequal) components



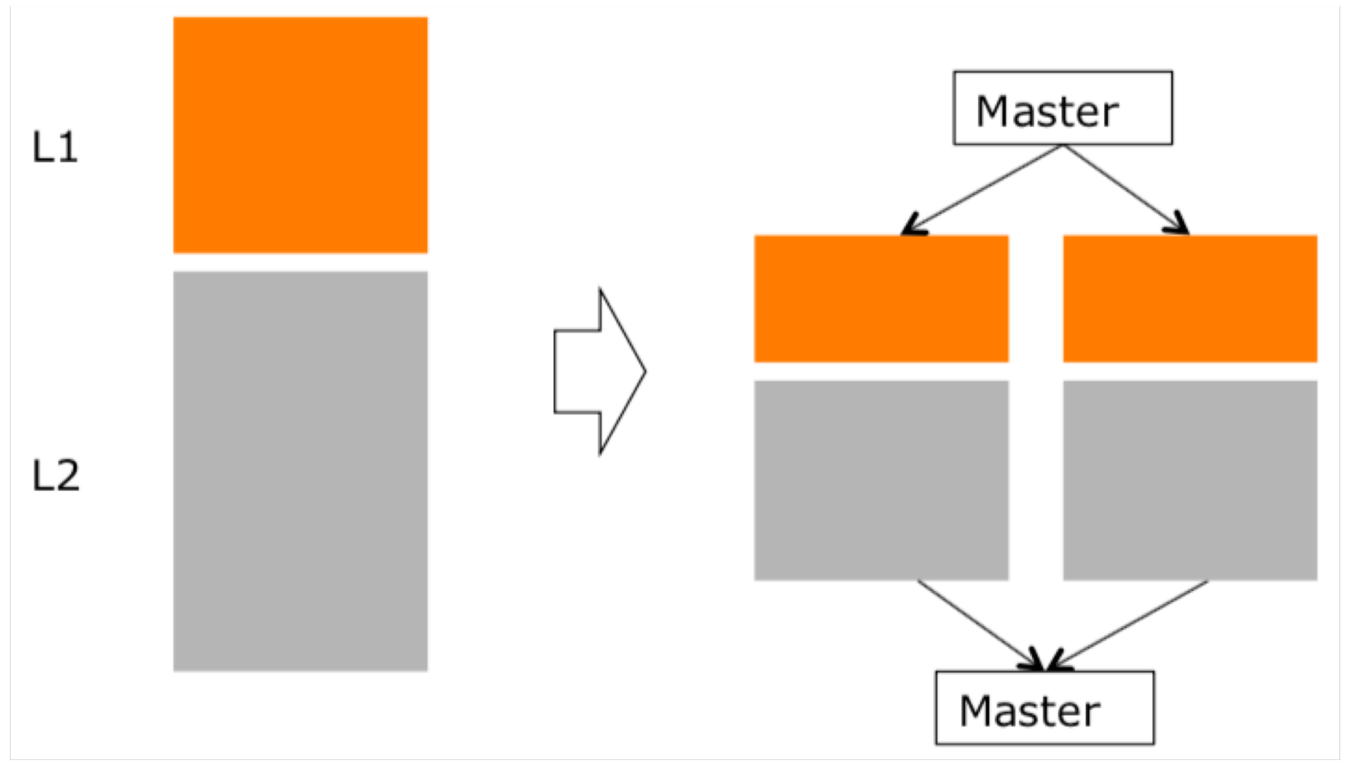
integrals etc.



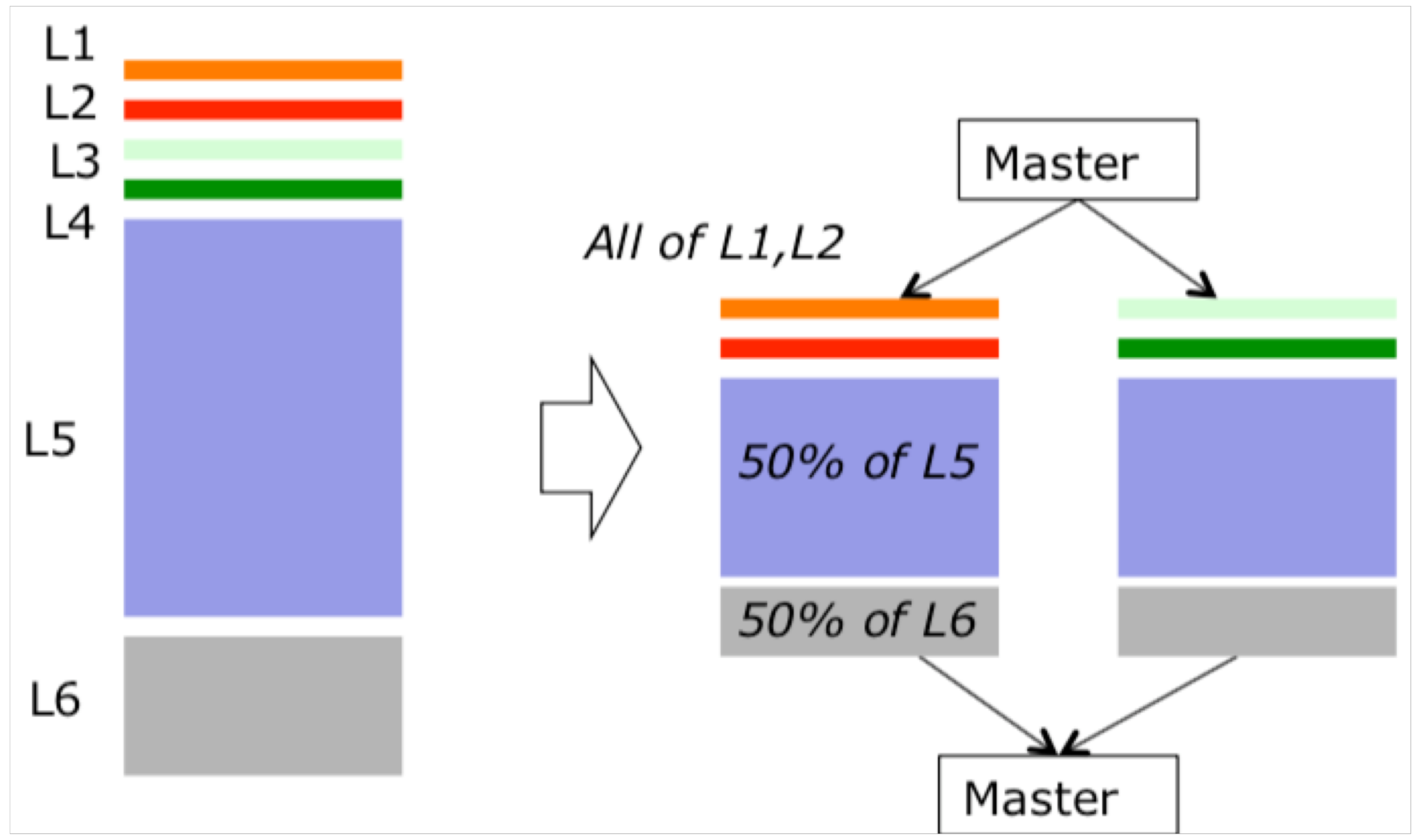
Heterogeneous: sizes, types

- Multiple strategies
- How to split up?
  - Small components → **need low latency/overhead**
  - Large components as well…
  - Run time depends on optimizations, differs per parameter, hard to predict
- How to divide over cores?
  - Load balancing → **task-based approach: work stealing**
- … both for likelihood-level and gradient-level parallelization

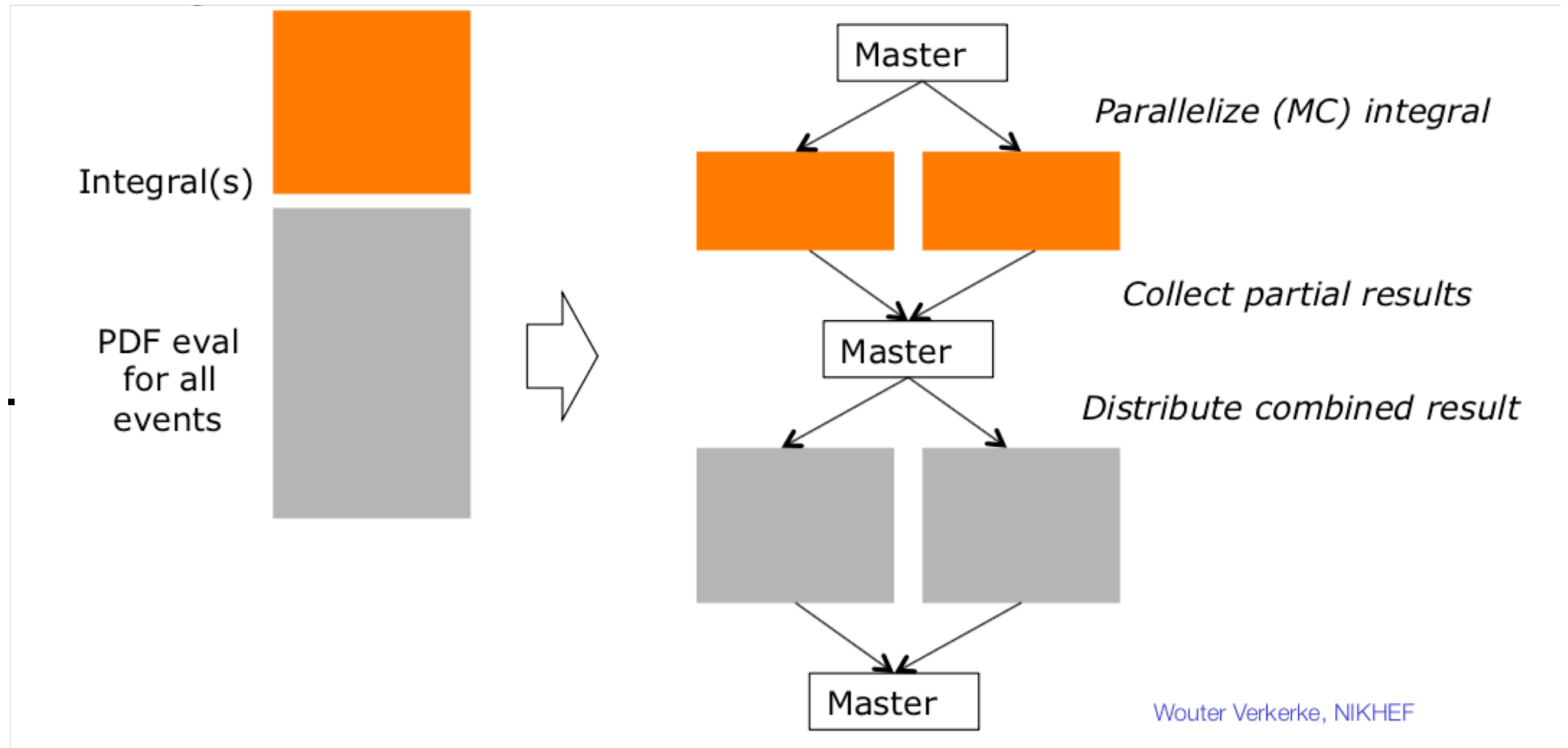
likelihood:
events



likelihood:
(unequal)
components



integrals etc.

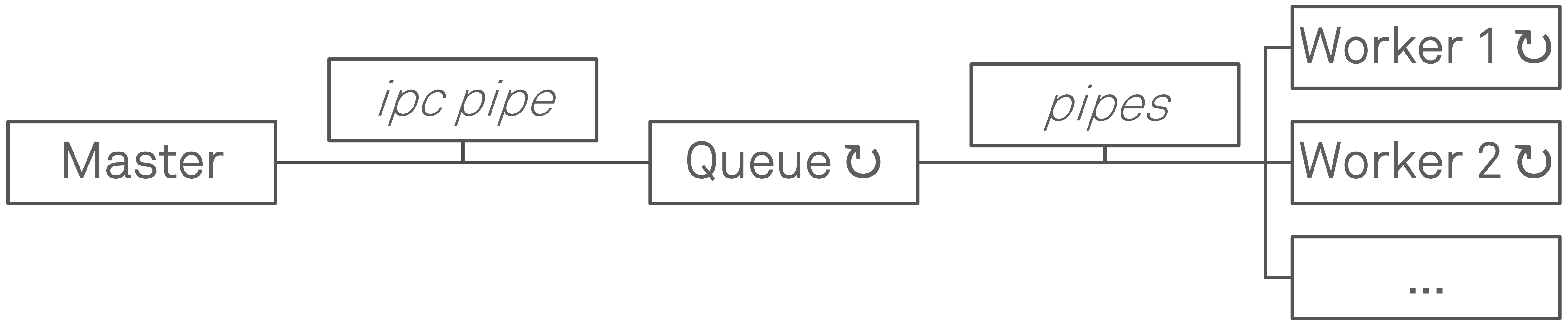


Task-stealing, worker pool, executes **Job** tasks

Job = likelihood component, $\frac{df}{dp}$, ...

No threads, process-based:

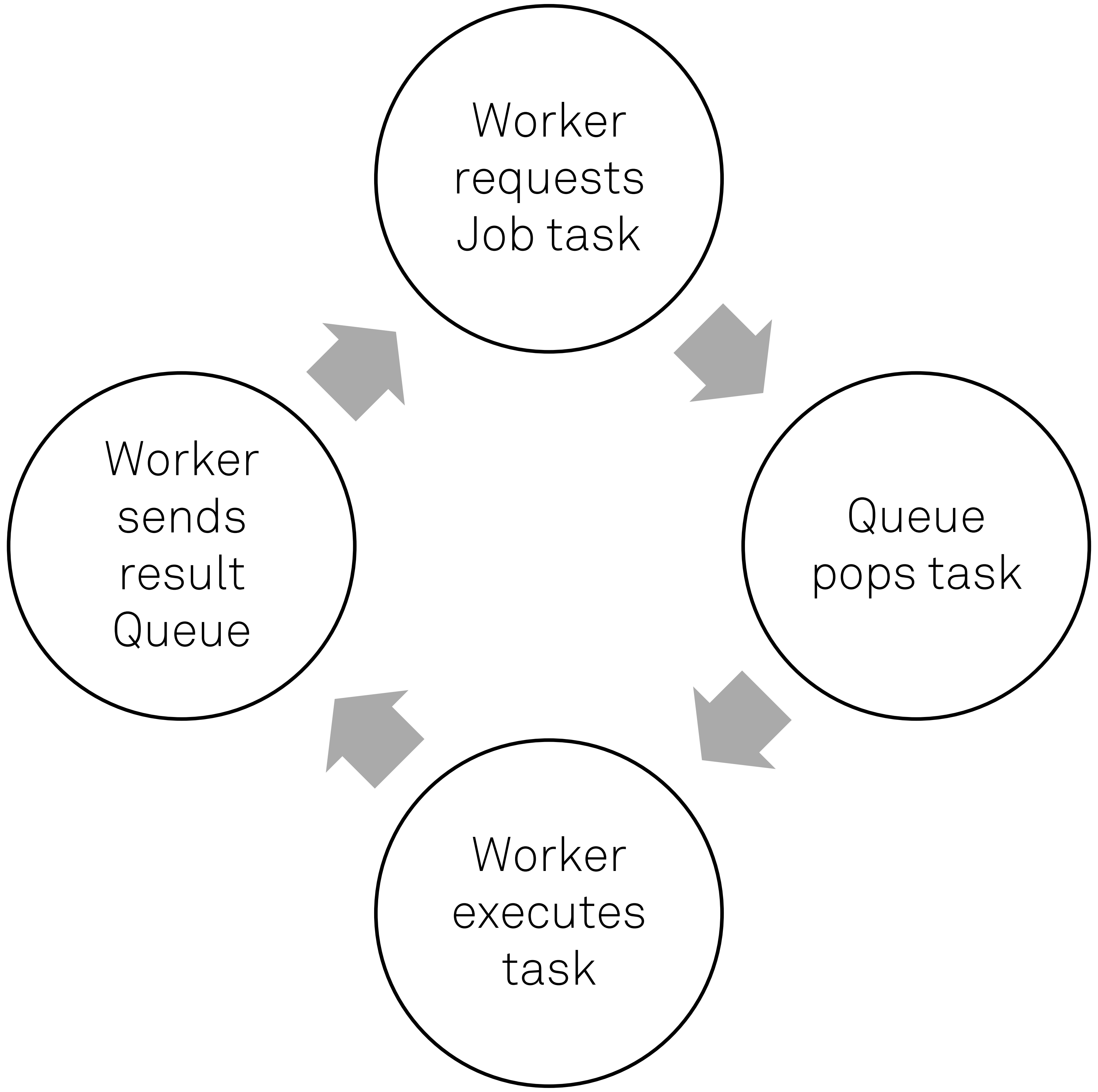
`BidirMMapPipe` handles fork, mmap, pipes



Master: main RooFit process, submits Jobs to queue, waits for results (or does other things in between)

Queue loop: act on input from Master or Workers (mainly to avoid loop in Master / user code) --- collect/distribute Jobs and results

Worker loop:



...until Job done then Queue sends results back to Master on request

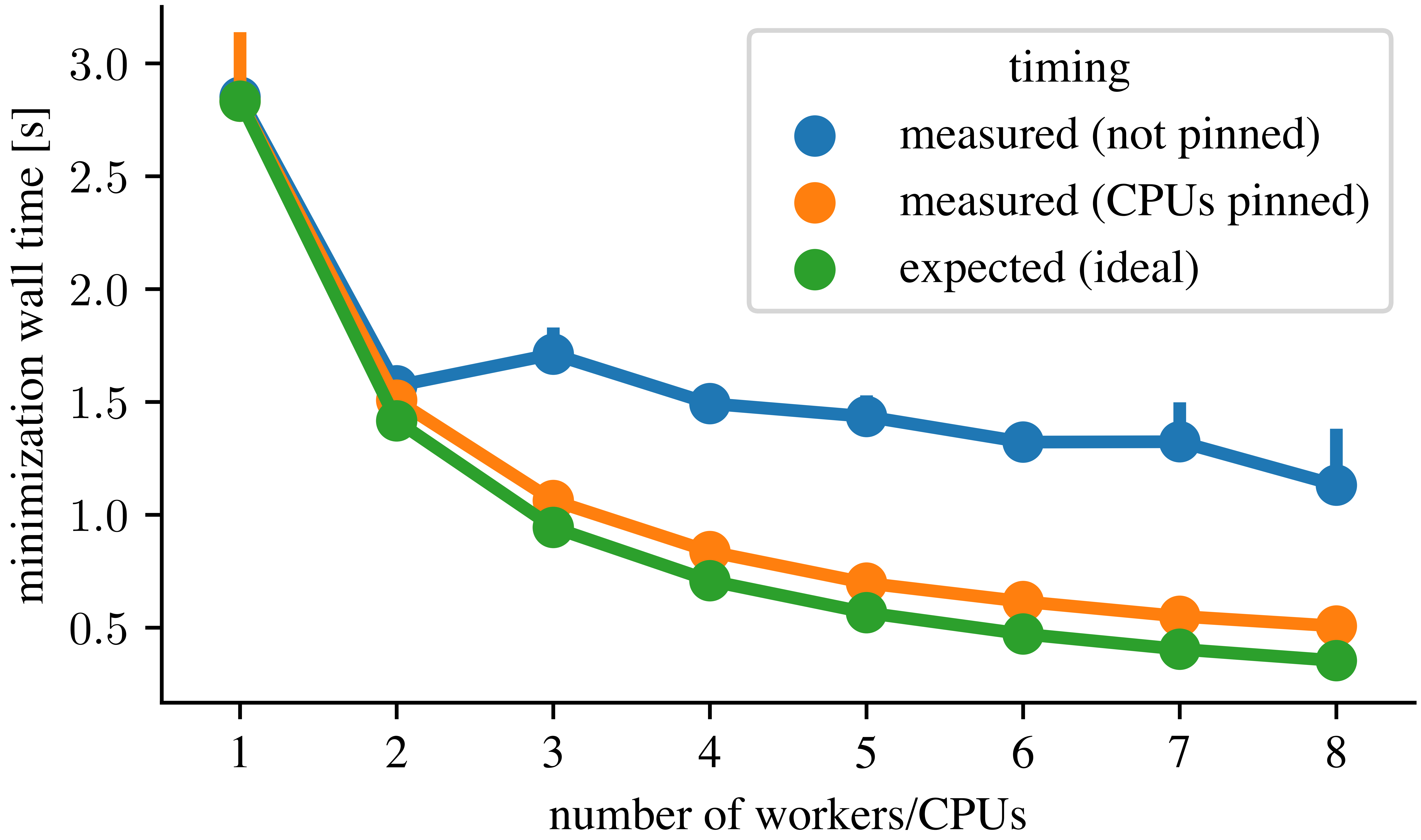
Parallel performance (MPFE & MP)

- Likelihood fits (unbinned, binned)
- Gradients

Run-time vs N(cores): simple N-dim Gaussian, many events

Before: max ~2x

Now (with CPU pinning fixed): max ~20x (more for larger fits)

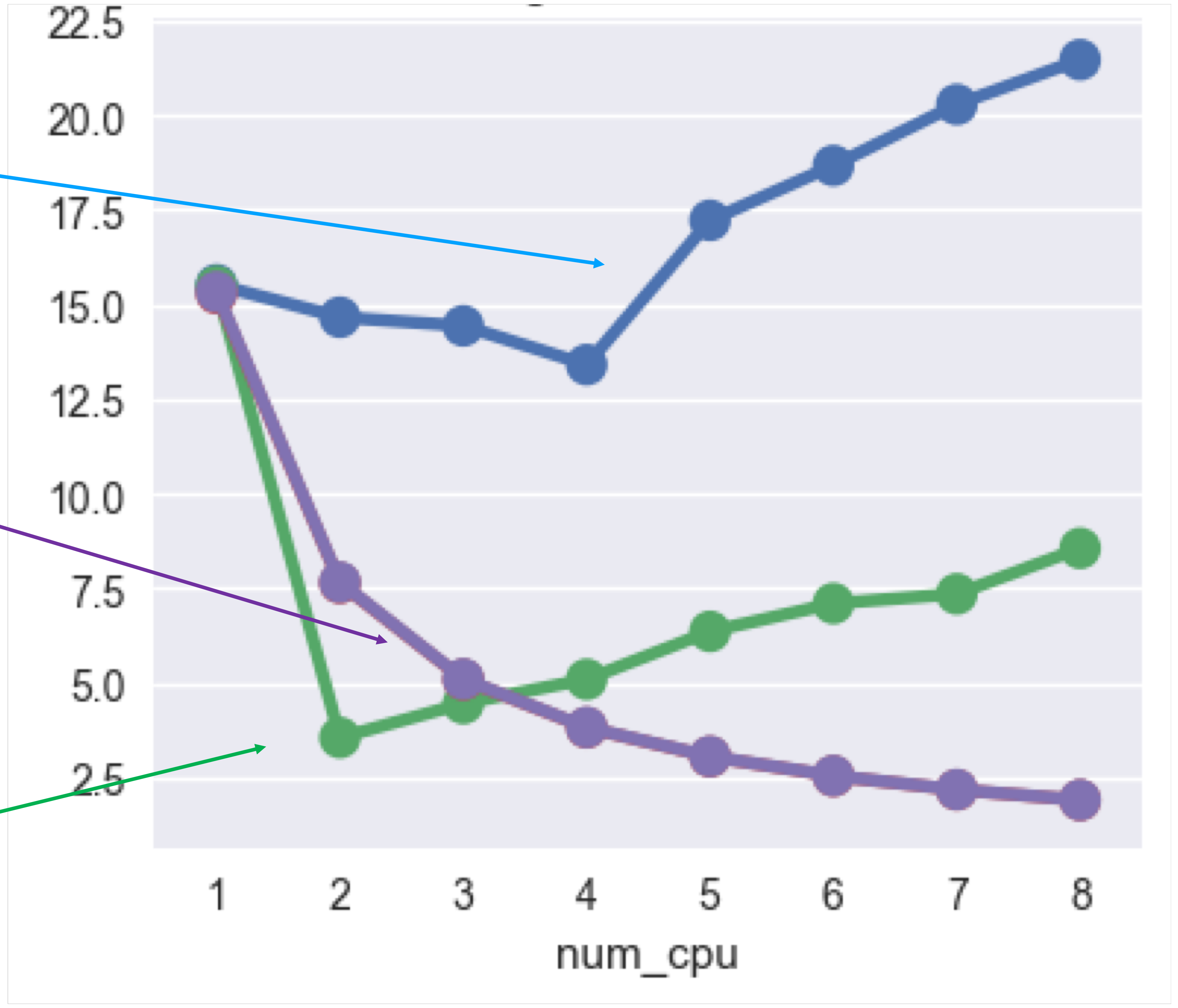


Run-time vs N(cores): certain types of binned fits

Actual performance

Expected performance (ideal parallelization)

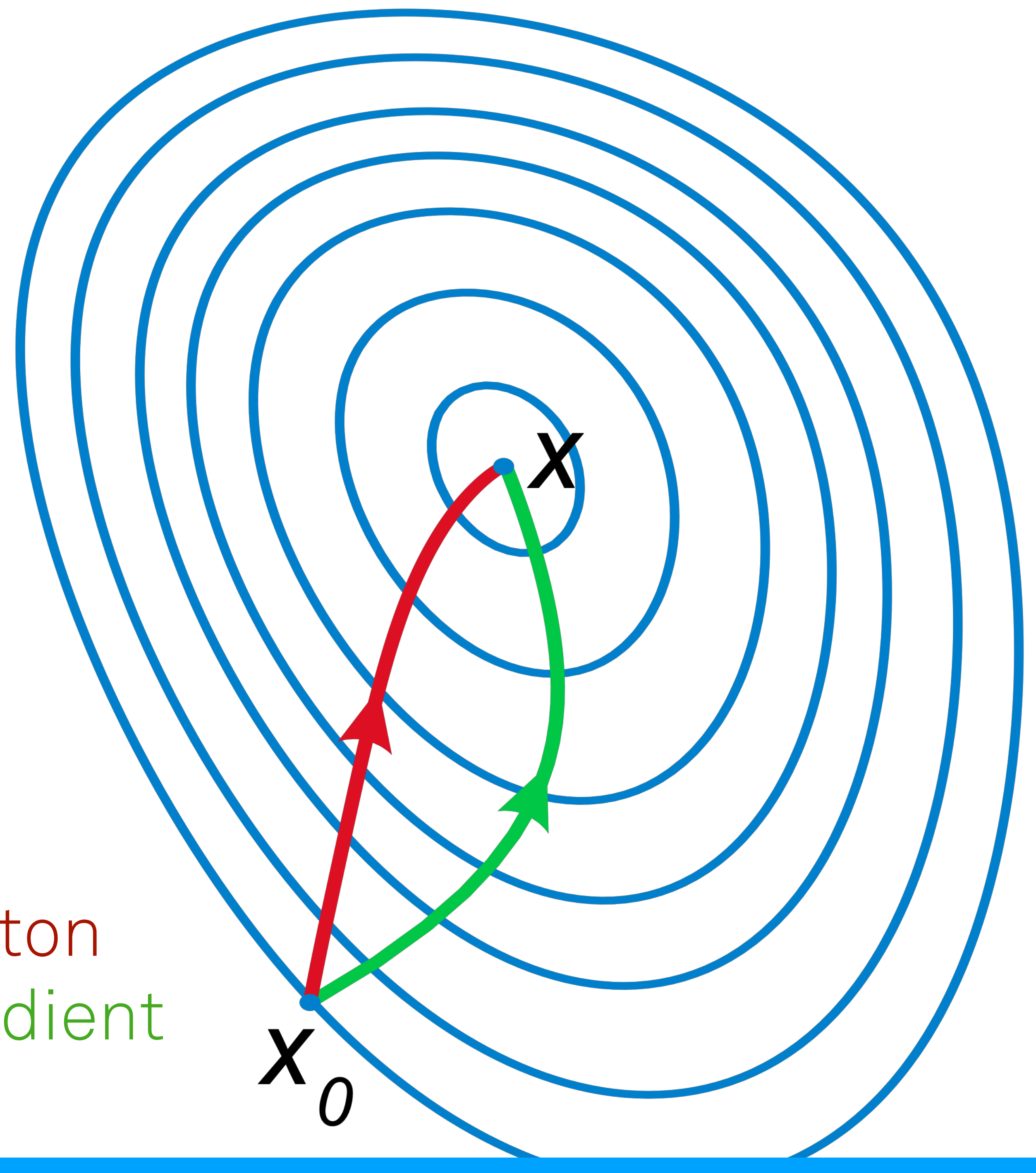
CPU time (single core)



under investigation

`Minuit`: minimize PDF $f(x;p)$:

- Quasi-Newton `MIGRAD` method



Left: Newton
Right: gradient descent

- $\boxed{\text{Gradient}}$ + line-search:
  - gradient for N parameters p: $\frac{df}{dp} \approx \frac{f(p-dp)-f(p)}{dp}$
  - line-search: descend along gradient direction

$\boxed{2N\ f \text{ calls} \rightarrow \text{parallelize } \frac{df}{dp}}$

- Important: serial & parallel results same
  - non-trivial, Minuit internal transformations

First benchmarks:

- "*ggF* model" (gluon-gluon fusion → Higgs boson), `MIGRAD` fit
  - realistic, non-trivial (265 parameters)
- scaling not perfect and erratic (+/- 5s)
  - likely caused by communication protocol - under investigation

RooMinimizer	MultiProcess GradMinimizer					
-	1 worker	2 workers	3 workers	4 workers	6 workers	8 workers
28s	33s	20s	15s	14s	17s (...)	11s

Conclusions

Interactive study of complex LHC physics fits (e.g. Higgs) requires parallelization

We improved scaling performance of likelihood-level parallelization

- Bottlenecks still exist for certain classes of models

New flexible framework: multi-level parallelization (likelihood, gradient)

- First working version, now analysis and tuning performance

Let’s stay in touch

+31 (0)6 10 79 58 74
p.bos@esciencecenter.nl
www.esciencecenter.nl
egpbos
linkedin.com/in/egpbos
blog.esciencecenter.nl

Encore

Load balancing

PDF timings change dynamically due to RooFit precalculation strategies

... not a problem for numerical integrals

Analytical derivatives (automated? `CLAD`)

Minuit confidence intervals

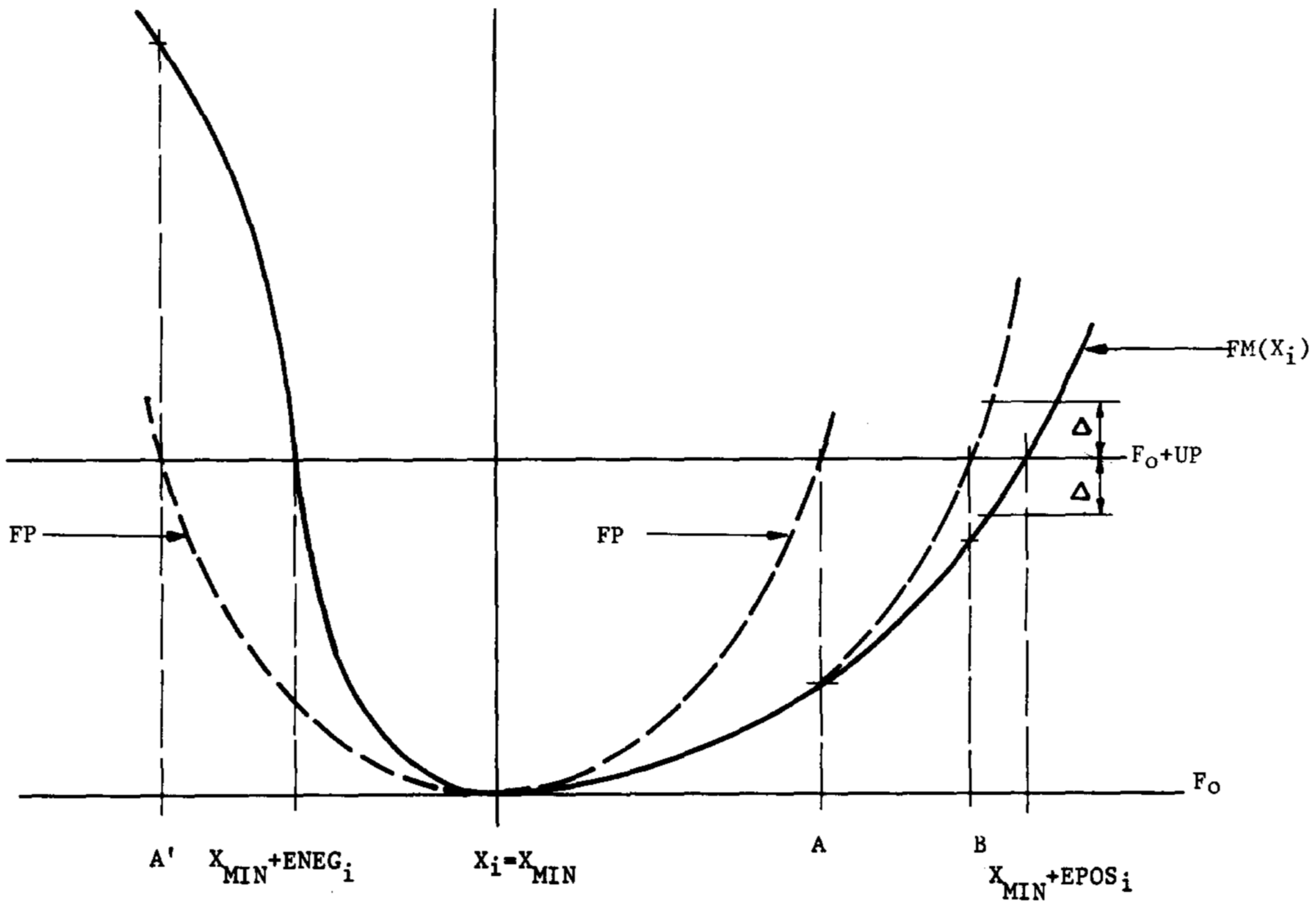


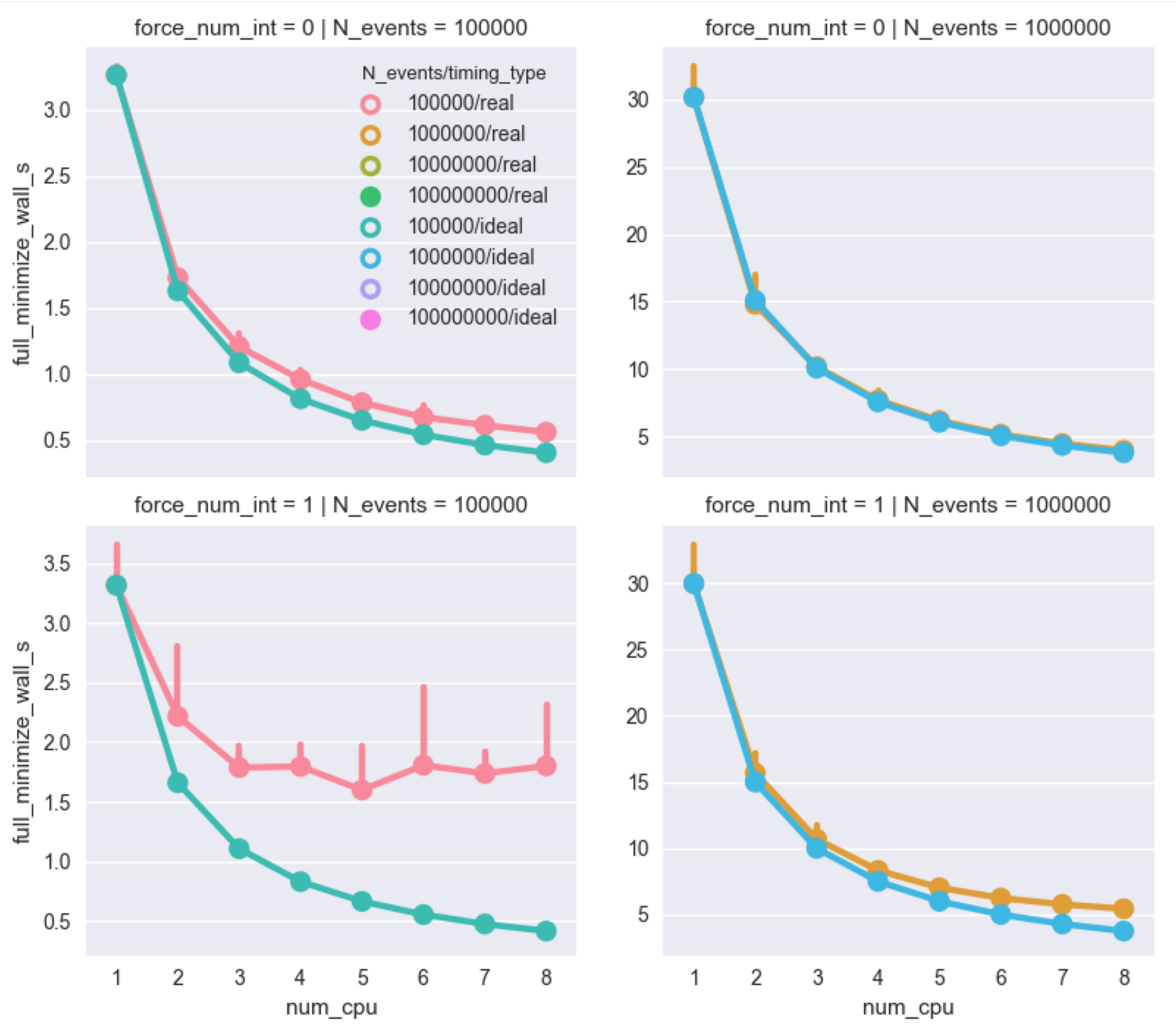
Fig. 2. Calculation of MINOS errors of parameter *i*. The (symmetric) dotted parabola FP is predicted from the covariance matrix, but the nonlinearity of the problem results in the solid curve FM which gives the asymmetric errors EPOS and ENEG (see text).

Numerical integrals

“Analytical” integrals

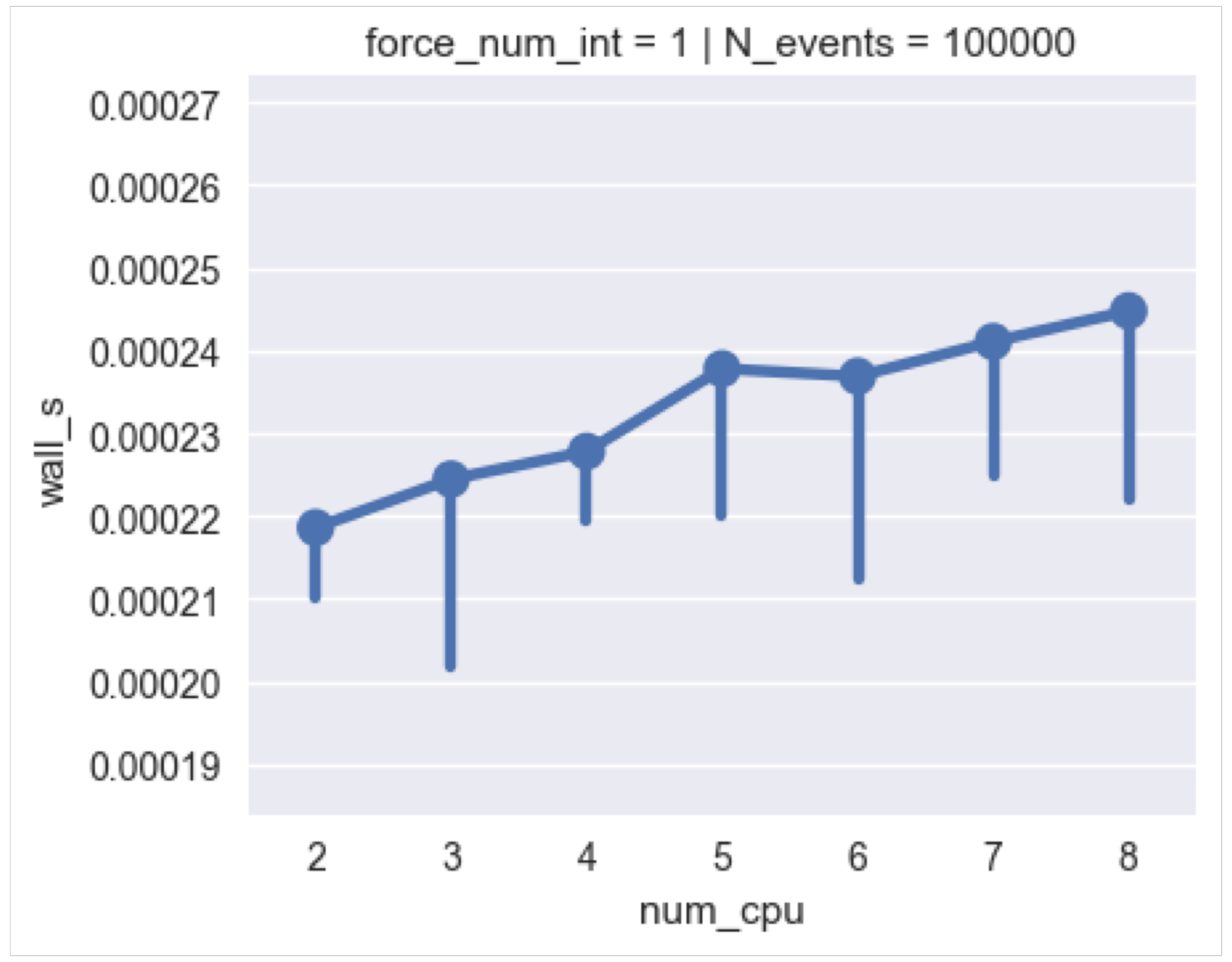
Forced numerical (Monte Carlo) integrals

(Higgs fits didn’t have them)

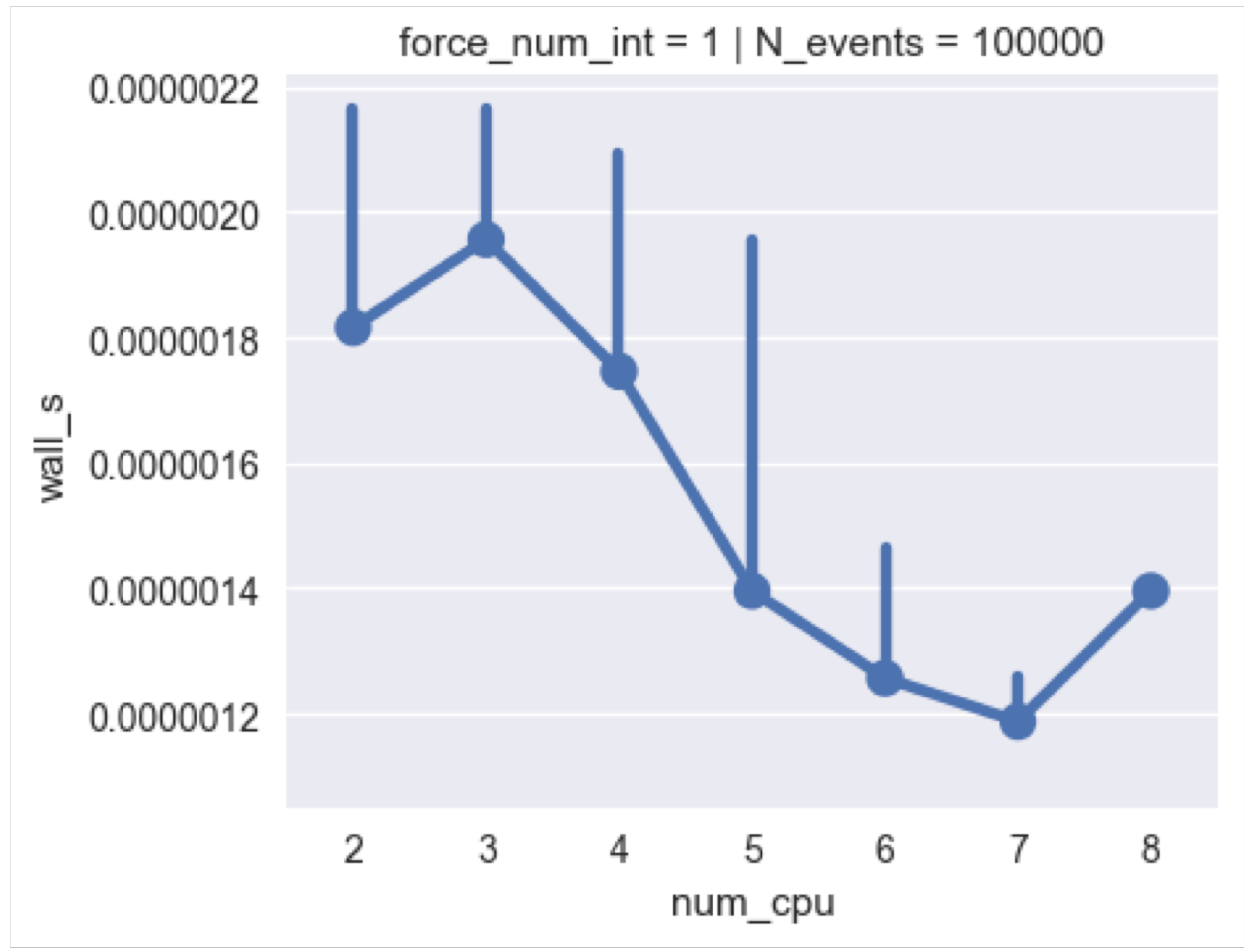


Individual NI timings
(variation in runs and iterations)

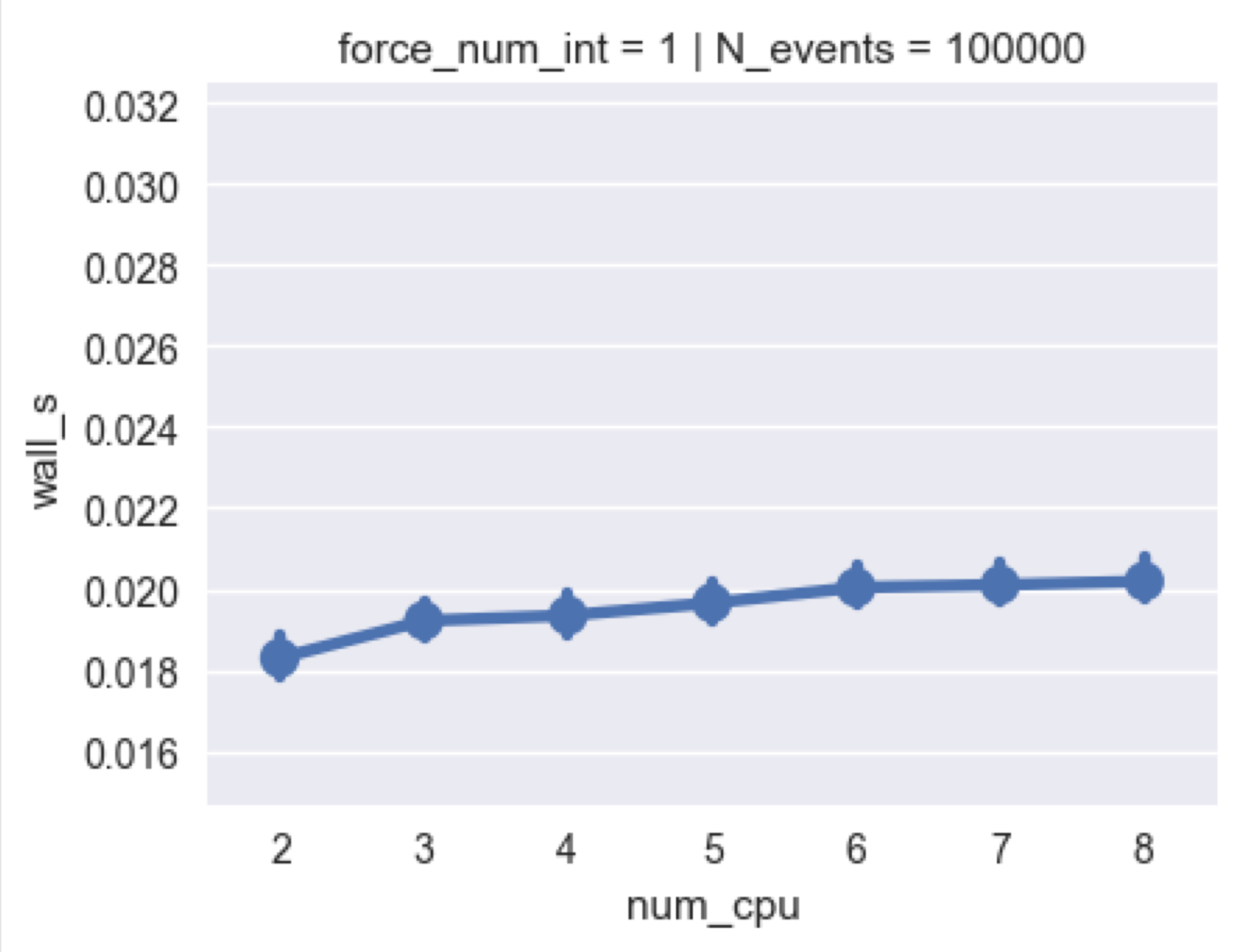
Maxima



Minima



Sum of slowest integrals/cores
per iteration over the entire run



(single core total runtime: 3.2s)

Faster fitting: MultiProcess design

`RooFit::MultiProcess::Vector<YourSerialClass>`

Serial class: likelihood (e.g. `RooNLLVar`) or gradient (Minuit)

Interface: subclass + MP

Define ”vector elements”

Group elements into tasks (to be executed in parallel)

`RooFit::MultiProcess::SharedArg<T>`

`RooFit::MultiProcess::TaskManager`

`RooFit::MultiProcess::Vector<YourSerialClass>`

`RooFit::MultiProcess::SharedArg<T>`

Normalization integrals or other shared expensive objects

Parallel task definition specific to type of object

… design in progress

`RooFit::MultiProcess::TaskManager`

Faster fitting: MultiProcess design

RooFit::MultiProcess::Vector<YourSerialClass>

RooFit::MultiProcess::SharedArg<T>

RooFit::MultiProcess::TaskManager

Queue gathers tasks and communicates with worker pool

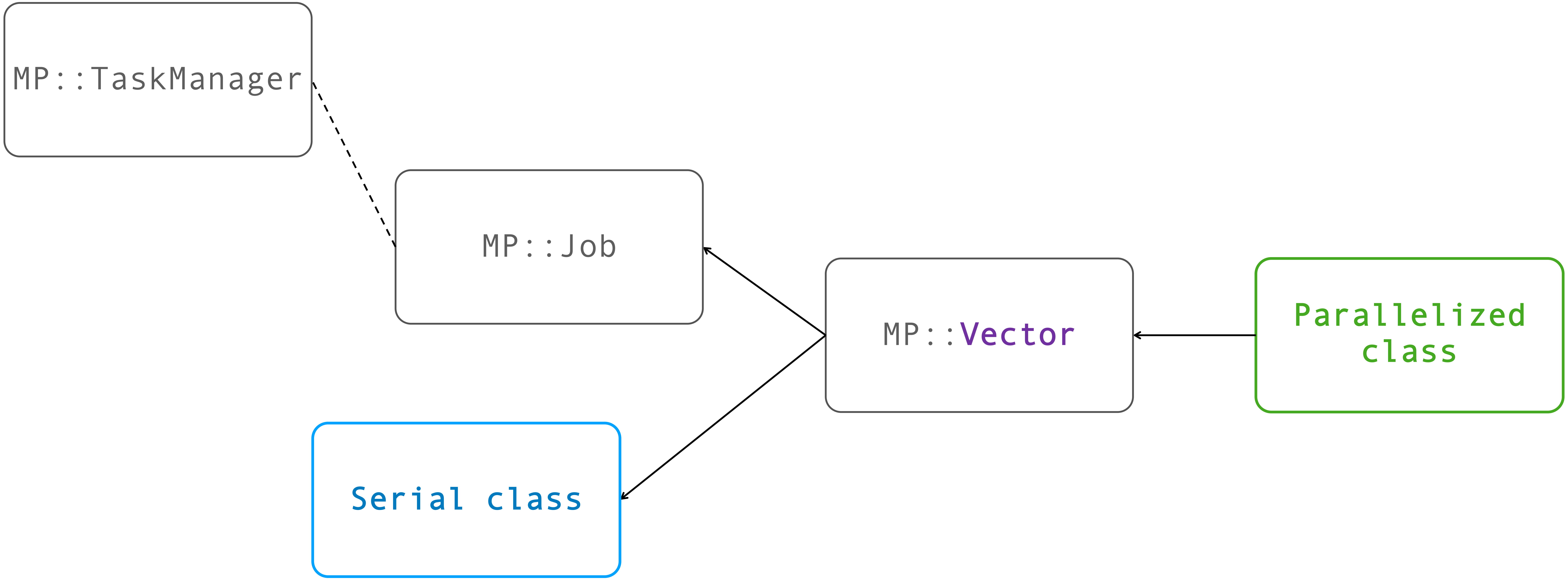
Workers steal tasks from queue

Worker pool: forked processes (BidirMMapPipe)

- performant and already used in RooFit
- no thread-safety concerns
- instead: communication concerns
- … flexible design, implementation can be replaced (e.g. TBB)

```
vector<double> x {1, 4, 5, 6.48074};

xSquaredSerial xsq_serial(x);

size_t N_workers = 4;
xSquaredParallel xsq_parallel(N_workers, x);

// get the same results, but now faster:
xsq_serial.get_result();
xsq_parallel.get_result();

// use parallelized version in your existing functions
void some_function(xSquaredSerial* xsq);

some_function(&xsq_parallel);  // no problem!
```

MultiProcess usage for devs




```
template <class T> class MP::Vector : public T, public MP::Job
class Parallel : public MP:Vector<Serial>
```

MultiProcess usage for devs

```
class xSquaredSerial {
 public:
  xSquaredSerial(vector<double> x_init)
          : x(move(x_init))
          , result(x.size()) {}

  virtual void evaluate() {
    for (size_t ix = 0; ix < x.size(); ++ix) {
      x_squared[ix] = x[ix] * x[ix];
    }
  }

  vector<double> get_result() {
    evaluate();
    return x_squared;
  }

 protected:
  vector<double> x;
  vector<double> x_squared;
};
```

```
class xSquaredParallel
  : public RooFit::MultiProcess::Vector<xSquaredSerial> {
 public:
  xSquaredParallel(size_t N_workers, vector<double> x_init) :
      RooFit::MultiProcess::Vector<xSquaredSerial>(N_workers, x_init)
  {}

 private:
  void evaluate_task(size_t task) override {
    result[task] = x[task] * x[task];
  }

 public:
  void evaluate() override {
    if (get_manager()->is_master()) {
      // do necessary synchronization before work_mode

      // enable work mode: workers will start stealing work from queue
      get_manager()->set_work_mode(true);

      // master fills queue with tasks
      for (size_t task_id = 0; task_id < x.size(); ++task_id) {
        get_manager()->to_queue(JobTask(id, task_id));
      }

      // wait for task results back from workers to master
      gather_worker_results();

      // end work mode
      get_manager()->set_work_mode(false);

      // put gathered results in desired container (same as used in serial class)
      for (size_t task_id = 0; task_id < x.size(); ++task_id) {
        x_squared[task_id] = results[task_id];
      }
    }
  }
};
```

```
template <class T> class MP::Vector : public T, public MP::Job
```

Single core profiling and improvements

Higgs ggf & 9 channel fits (workspaces by Lydia Brenner)

Most time spent on:

1. Memory access → RooVectorDataStore::get() (4% / 32%), 0.3% LL cache misses (expensive!)
   - Row-wise access pattern on column-wise data store (and std::vector<std::vector>)
2. Logarithms: 12%
3. Interpolation → RooStats::HistFactory::FlexibleInterpVar (10%)

RooLinkedList::findArg: ~ 5% of memory access instructions

RooLinkedList::At took considerable time in Gaussian test fit (*Vince*)

std::vector lookup → 1.6x speedup! WIP

Reorder tree evaluation → CPU cache use, vectorization

Smarter fitting (stochastic minimizer, analytical gradient, CLAD)

Front-end / back-end separation (e.g. TensorFlow back-end)

- profiling functions & classes
  - valgrind
  - gprof
  - Instruments
  - … etc.
- profiling objects (e.g. call-trees, e.g. RooFit…)
  - … DIY?

More Multi-Core

`RooRealMPFE` / `BidirMMapPipe`

Custom multi-process message passing protocol

- POSIX `fork`, `pipe`, `mmap`

Communication “overhead” (delay between sending and receiving messages): ~ 1e-4 seconds

- `serverLoop` waits for message & runs server-side code
- messages used sparingly
- data transfer over memory-mapped pipes

	RooFit (MINUIT)	TensorFlow (BFGS)
Unbinned fit	0.1s	**0.01 - 0.1s** (dep. on precision)
Binned fit	**0.7ms**	2.3ms

Fits on identical model & data (single i7 machine)

TensorFlow: No pre-calculation / caching!

- Major advantage of RooFit for binned fits (e.g. morphing histograms)
- (feature request for memoization https://github.com/tensorflow/tensorflow/issues/5323)

N.B.: measured before CPU affinity fixing

- RooFit now even faster (but limited to running one machine)