Differentiating RooFit likelihoods with Clad

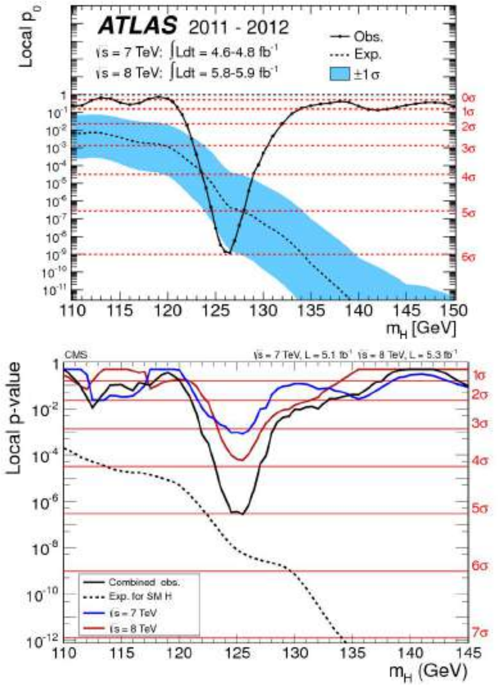
Jonas Rembser (CERN EP-SFT)

CaaS Monthly Meeting – 05 June 2025

Introduction to RooFit

- **RooFit**: C++ library for statistical data analysis in ROOT
  - provides tools for model building, fitting and statistical tests
- Recent development focused on:
  - **Performance** boost (preparing for larger datasets of **HL-LHC**)
  - More **user friendly** interfaces and high-level tools

In **this presentation** we’re summarizing the RooFit developments that integrate Automatic Differentiation (AD) using Clad.

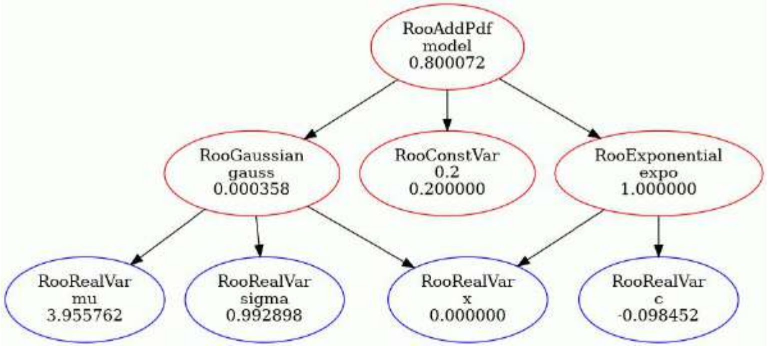


RooFit was used in the Higgs boson discovery!

Why RooFit?

RooFit is serving the HEP community well because of several **key features**:

- Likelihood functions **highly optimized** for the context of minimization with Minuit
- It takes care of **analytical normalization** integrals where possible
- **User-extensible** framework that can cover a wide range of use cases
  - *Binned* likelihood fits
  - *Unbinned* likelihood fits
- Sharing of **statistical workspaces** thanks to ROOTs powerful IO system



Conditional pdf example:

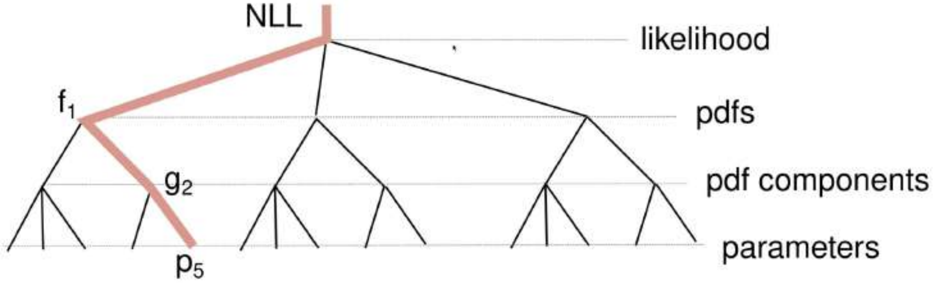
$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(x,y)}{\int p(x,y)dx}$$

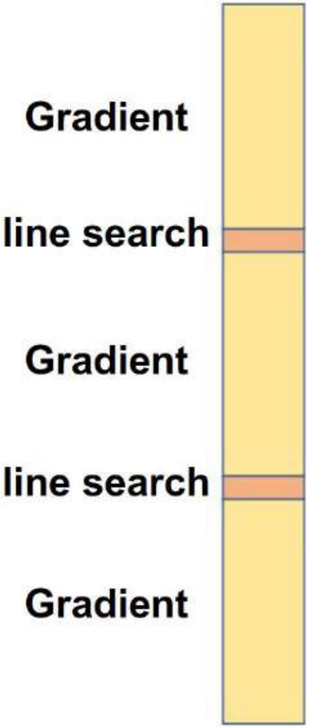
Observable subdomain example:

$$p(x|\text{subrange}) = p(x)\frac{\int_{\text{full}} p(x)dx}{\int_{\text{subrange}} p(x)dx}$$

Numeric minimization of RooFit Likelihoods

- By default, RooFit uses **numerical differentiation**: Minuit 2 changes parameters **on-at-the-time** to get the full gradient
- One key concept of RooFit: **caching of intermediate results** to minimize redundant computations in gradient evaluation
- Still, gradient dominates minimization time *(see also the ICHEP 2022 RooFit presentation)*





Our goal: make evaluating gradients cheap with **Automatic differentiation (AD)**

Typical bottlenecks in RooFit + Minuit 2

The bottlenecks in likelihood minimization with RooFit are typically:

- **Function evaluation**:
  - e.g. if many *events* (dataset entries to iterate)
- **Gradient evaluation**:
  - in case you have many *parameters*
  - This is the bottleneck that we’re addressing with AD
- **RooFit bookkeeping of** what needs reevaluation:
  - in case you have *deep computation graphs*
  - Important for caching in numerical gradient calculations
- **Linear algebra in Minuit 2**:
  - If you have *many parameters*, but the function and gradients are cheap and the computation graphs are shallow

Automatic differentiation engine for RooFit

- RooFit is a framework to build **computation graphs for function minimization**, similar to the ML frameworks **TensorFlow** or **PyTorch**
- Different from other frameworks, RooFit didn’t have an **automatic differentiation engine**
- However, the other frameworks are generally not optimized for HEP use cases and workflows

TensorFlow PyTorch

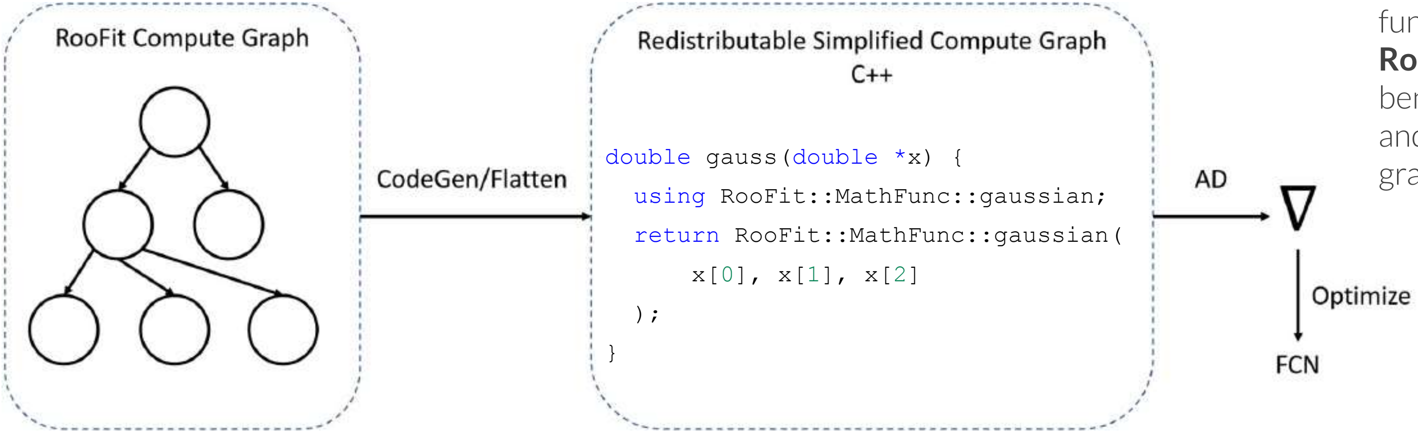
ROOT
Data Analysis Framework
Clad

Therefore, we have added a differentiation engine based on **Clad** and **C++ code generation** to RooFit.

Automatic Differentiation with Clad

RooFit uses Clad to get analytic gradients: **Code generation** (aka. “codegen”)
More detail in [last month’s ROOT blog post]

1. **Mathematical** concept
2. **RooFit** user code
3. **Automatic translation** of RooFit model to simple C++ code
4. **Gradient** of C++ code **automatically generated** with **Clad**
5. Gradient code **wrapped** back into RooFit object



```
double gauss(double *x) {
  using RooFit::MathFunc::gaussian;
  return RooFit::MathFunc::gaussian(
      x[0], x[1], x[2]
  );
}
```

Note: for the **nominal NLL** function, we **still use RooFits CPU backend** to benefit from vectorization and caching outside the gradients.

Implementation Details

- There are four ingredients in RooFit to make the “codegen” happen:
  a. A collection of free functions for the **math of a given RooFit class**
  b. The CodegenContext that is has to **visit each graph node** and collects the code snippet for each node
  c. A codegen library with **one free function for each RooFit primitive** that generates the actual code snippet, e.g.:

```
void codegenImpl(RooGaussian &arg, CodegenContext &ctx) {}
// The dispatching is done by downcasting in Cling:
// no virtual functions needed!
```

  d. The RooFuncWrapper that **manages the code generation** and AD. To the outside it looks like any other RooAbsArg.
- Our developer documentation explains this in more detail.

Implementation Details (combined fits)

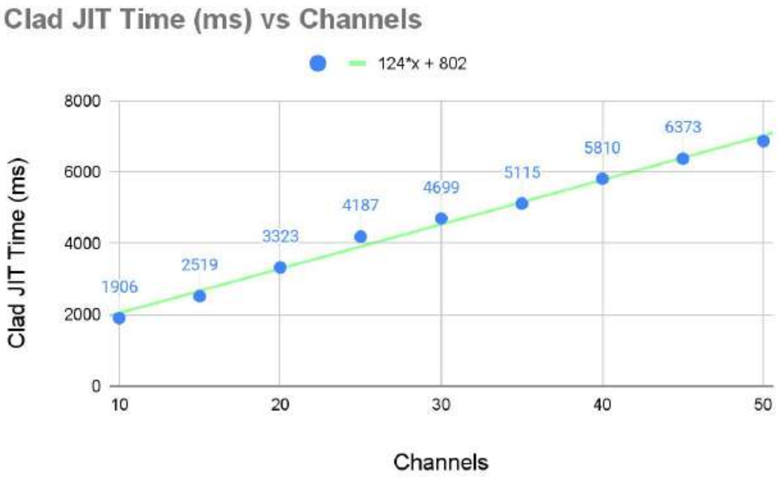
- One exception to translating the whole computation graph to one function:
  - **Combined** fits (likelihood is sum of likelihoods for different “channels”, with shared parameters)
- This ensures total JIT time is proportional to the number of channels, and that the **used stack memory is constant** with the number of channels

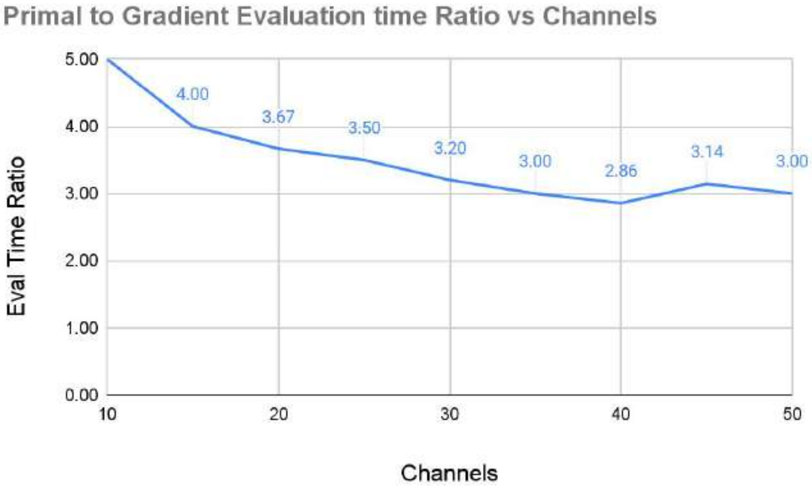
```
double nll_channel_0(
  double *params, // parameters
  const double *obs, // observed data
  const double *xlArr // auxiliary constants
                      // (e.g. histogram data)
) {...}
…
double nll_channel_<n>(...) { … }

double combined_nll(double *params,
  const double *obs,
  const double *xlArr) {
  // sum over all channel nlls...
  res += nll_channel_0(params, obs, xlArr);
  // .. plus parameter constraints
  // from auxiliary measurement
}
```

Structure of generated code for combined likelihoods. The user doesn’t have to deal with this: everything is done in the RooFit implementation details.

JIT time and evaluation time of the gradients

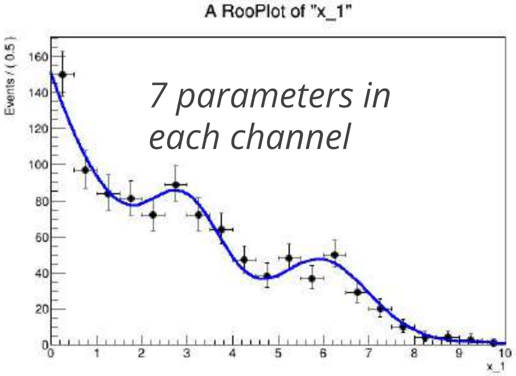




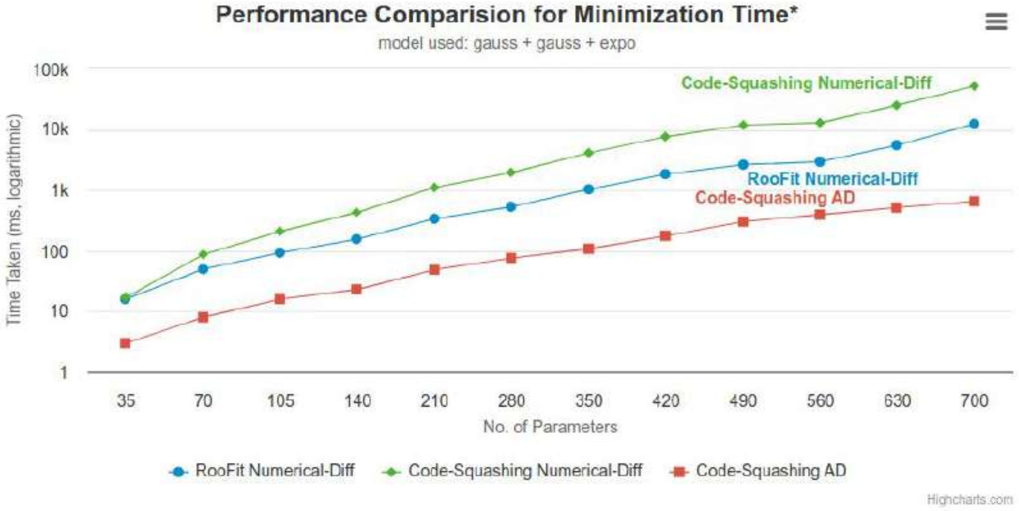
- Indeed, **JIT time** for an ATLAS example is **scaling linearly** with #channels
- Splitting up the gradient in multiple functions **doesn’t negatively affect performance**
- Also, **memory consumption** of gradient evaluation is very low compared to the python/ML based frameworks
  - Constant factor of the consumption by primal function

Scaling Study From CHEP 2023

- In the CHEP 23 conference, we have presented a scaling study as a function of the number of channels, with a simple fit of two Gaussians plus exponential to a histogram in each channel
- Many **things have changed since then** in RooFit, Clad and Minuit 2
- It’s worth to **redo the study** to see where we stand

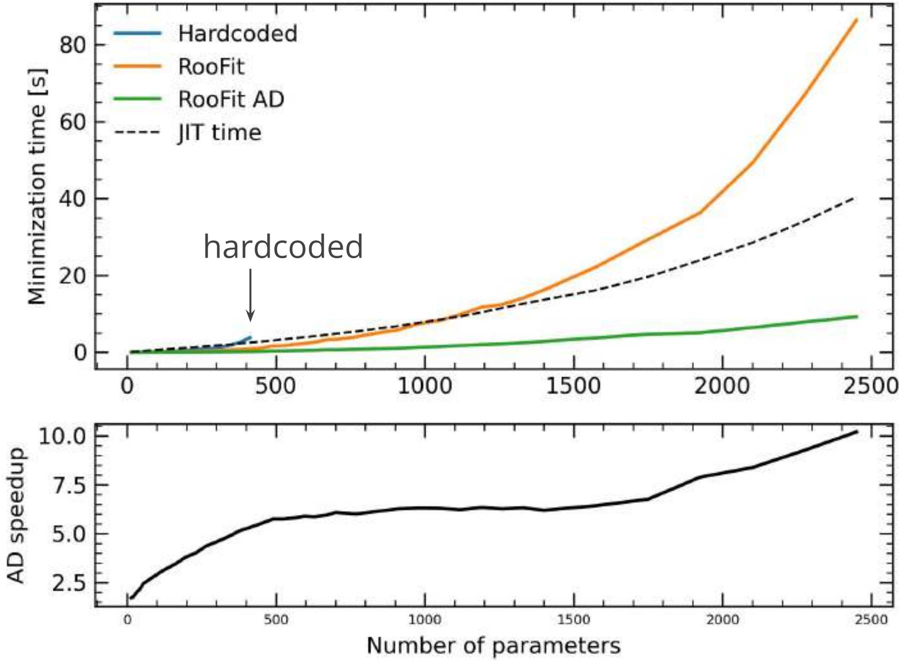


7 parameters in each channel



Updated scaling study

- Gradient **bottleneck disappears** with RooFit AD
- New bottleneck according to profiling: **linear algebra in Minuit 2**
  - expected because function is cheap (simple model)
- Although jitting is slow, for many parameters it is amortized even after a *single minimization*
- Speedup reduced compared to CHEP 2023 result because of optimizations in numeric gradients in Minuit 2 with ROOT 6.36
- Still: **impressive speed-up** that **scales well**!



Note that the hardcoded likelihood fails minimization for ~400 parameters or more, because of missing offsetting.

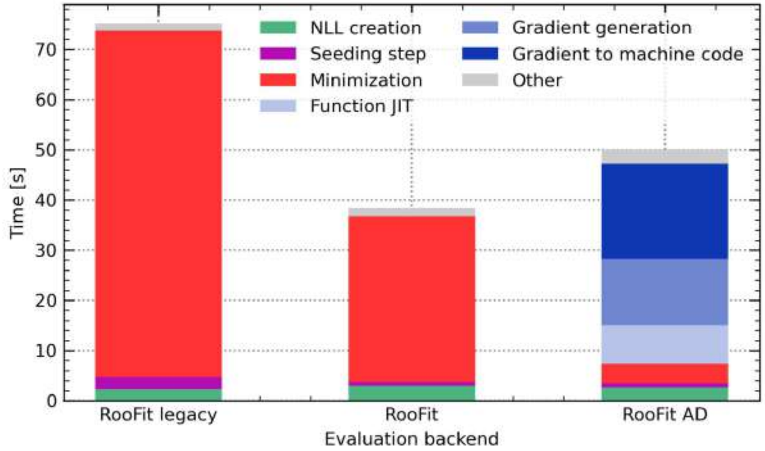
Higgs Combination Benchmark - ATLAS

- Using analytic gradients significantly **reduces minimization time** for many-parameter fits with
  - ATLAS HistFactory benchmark on the right
- Also **numerically more stable**: no tricks required to get better precision on numeric gradients *(e.g. likelihood offsetting)*
- *Caveat:* potentially long time for gradient generation
  - To benefit, workflow needs to reuse likelihood (e.g. **toy studies** or **profile likelihood scans**)

More detail in ICHEP 2024 presentation.

ATLAS fit

Jit time can be amortized by re-using likelihood!



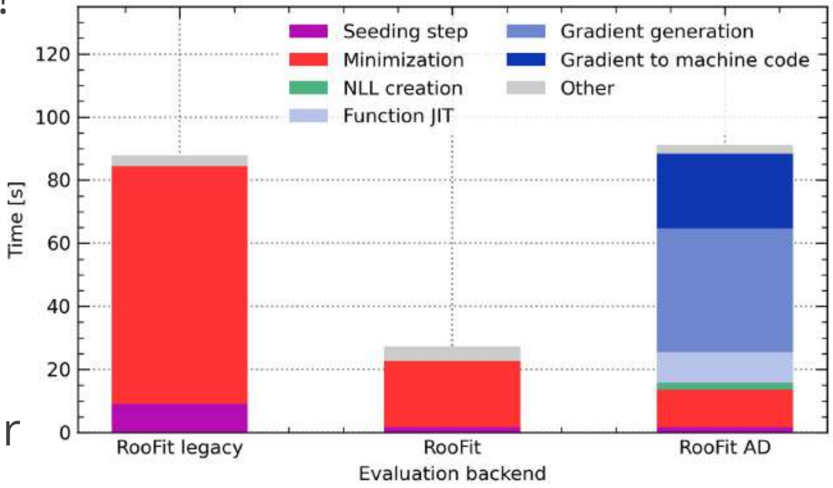
Detailed breakdown of minimization time for ATLAS Higgs combination benchmark with different RooFit backends (49 HistFactory channels, 739 parameters in total, in rootbench)

Higgs Combination Benchmark - CMS

- **Breaking news in April 2024**: CMS published RooFit-based Higgs observation likelihood!
- Very heterogeneous likelihood: **672 parameters** in **102 channels** with
  - Template histogram fits
  - Analytical shape fits, numerical integration necessary in some cases
- **Perfect example** to test the new RooFit developments
- Results can be reproduced with the master branch of the CMS combine tool

More detail in ICHEP 2024 presentation.

CMS fit



Profiling RooFit - not a black box!

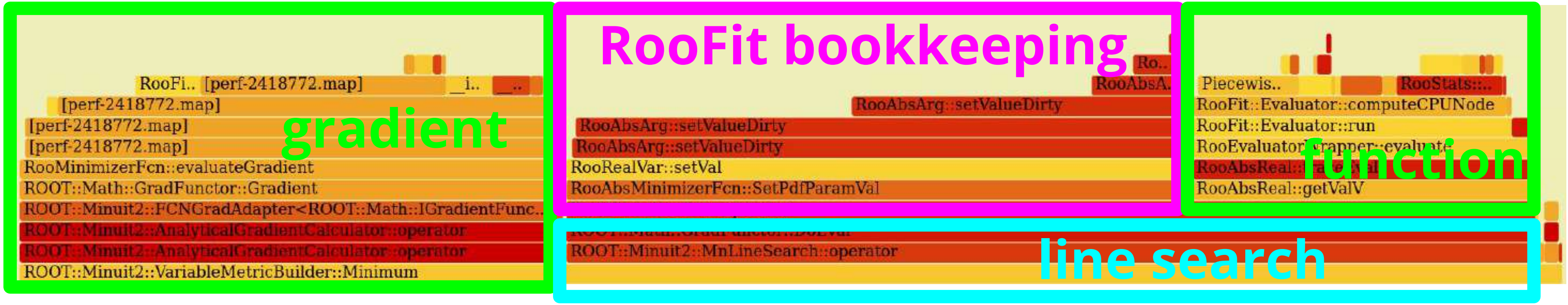
- RooFit serves many use cases and users hit **different bottlenecks**
- Since written in C++, RooFit code is convenient to profile
- Flamegraphs often inspire **significant performance improvements** in RooFit
- Guarantees that RooFit continues to scale well for cutting edge fits

Example workflow to profile ROOT macro with `perf` *and* `flamegraph.pl`:

- *Make sure ROOT is built with debug info but not in debug mode (*`-DCMAKE_BUILD_TYPE=RelWithDebInfo`*)*
- *Macro needs a* `main()` *function so it can be compiled*

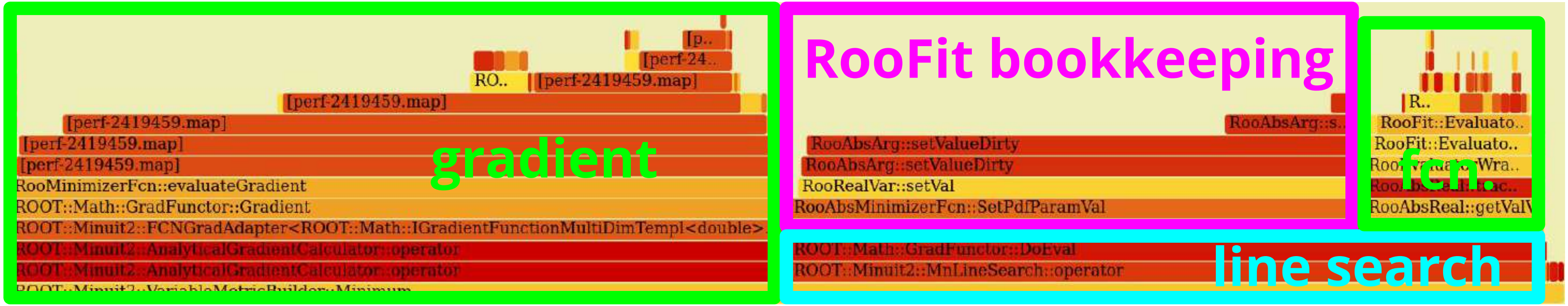
```
g++ $(root-config --cflags --libs) -g \
   -lRooFitCore -lRooFit -o fit_macro \
   fit_macro.C
perf record -F 99 -g -- "./fit_macro"
perf script | stackcollapse-perf.pl >out.perf-folded
flamegraph.pl out.perf-folded > flamegraph.svg
```

Profiling the minimization - ATLAS



- Profiling **ATLAS** minimization (full flamegrah)
- With RooFit AD, gradient is **not the bottleneck** anymore
- New bottleneck is the RooFit parameter bookkeeping in the line search
  - In theory, it’s possible to completely eliminate that overhead: bookkeeping of changed parameters is *unnecessary for line search*, because all parameters change anyway

Profiling the minimization - CMS



- Profiling **CMS** minimization (full flamegraph)
- Likelihoods in CMS Combine are very optimized, so the **RooFit bookkeeping overhead** is relatively larger
- Once RooFit bookkeeping overhead is gone, further optimizing the gradient could be worth it

Conclusions

- With **Clad**, RooFit can make use of a powerful engine for **Automatic Differentiation** (AD)

- Using AD to get analytical gradients in RooFit, the gradients are **no longer the bottleneck** in the minimization
  - The price to pay is JIT time in the beginning, but this can be amortised if the likelihood is re-used for multiple fits (e.g. in toy studies or profile likelihood scans)

- There is still work to do in terms of:
  - RooFit **feature coverage** of codegen
  - Higher order derivatives (**Hessians**)
  - **Integration** in LHC experiment frameworks