Parallel RooFitting: Put your CPUs to **use**

Patrick Bos *(NL eScience Center)*,
Zef Wolffs *(Nikhef)*,
Wouter Verkerke *(Nikhef)*,
et al.

netherlands eScience center

Live demo… on your laptop!

- Requirements:
  - ROOT 6.28+ built with `-Droofit_multiprocess=ON`
- Check your build!
- We'll come back to the actual demo later

sneak preview:
https://gist.github.com/egpbos/03003b273b8bb2407aa64a575a99a25b
(or go to https://gist.github.com/egpbos, the top one)

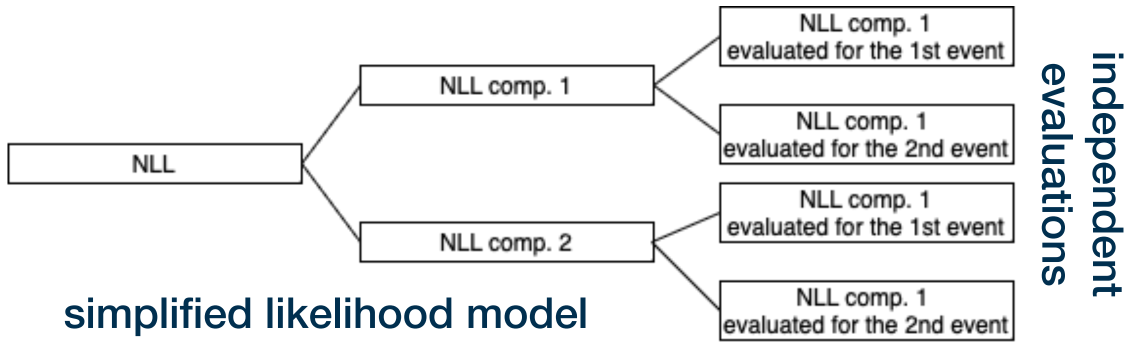
netherlands eScience center

Background

- In high energy physics, hypothesis testing is done by fitting likelihood models to datasets
- **In principle**, parallelizing this problem is not hard, remember the likelihood model

$$-\log L(\theta|\mathbf{x}) = -\log \prod_{i=0}^{N} p(\mathbf{x}_i|\theta) = -\sum_{i=0}^{N} log(p(\mathbf{x}_i|\theta)) = \underbrace{-log(p(\mathbf{x}_1|\theta))}_{\textbf{parallel task 1}} \underbrace{-\, log(p(\mathbf{x}_2|\theta))}_{\textbf{parallel task 2}} - \ldots$$

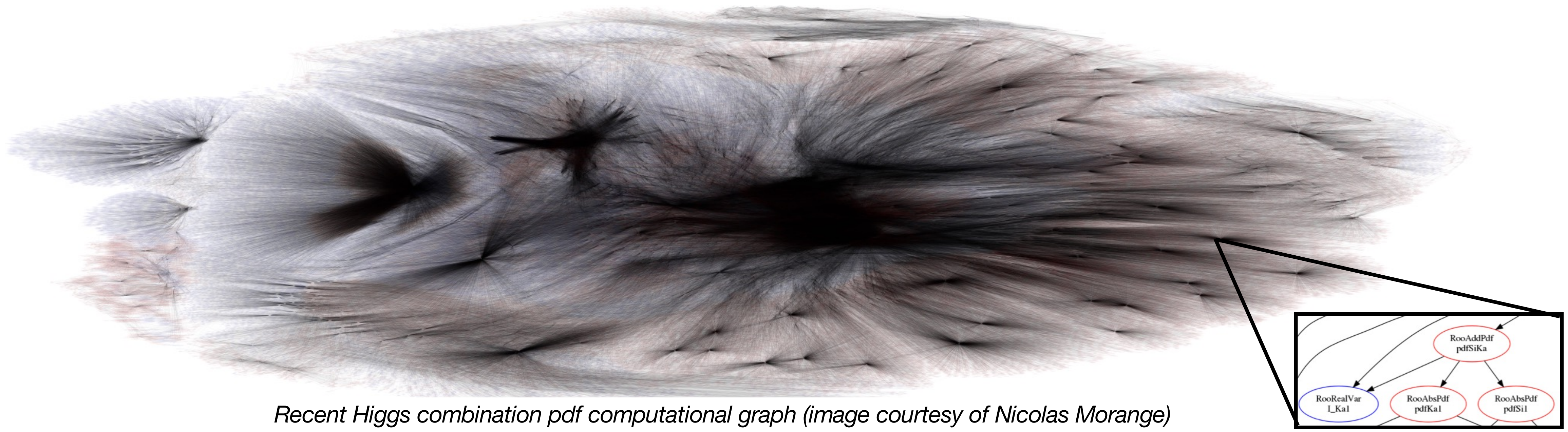
  - The evaluation of each event can be calculated fully independently and thus in parallel
  - Even more so, likelihood models in high energy physics are generally also constructed from independent components which could also be evaluated in parallel


NLL
NLL comp. 1
NLL comp. 2
NLL comp. 1 evaluated for the 1st event
NLL comp. 1 evaluated for the 2nd event
NLL comp. 1 evaluated for the 1st event
NLL comp. 1 evaluated for the 2nd event
independent evaluations

simplified likelihood model

Background

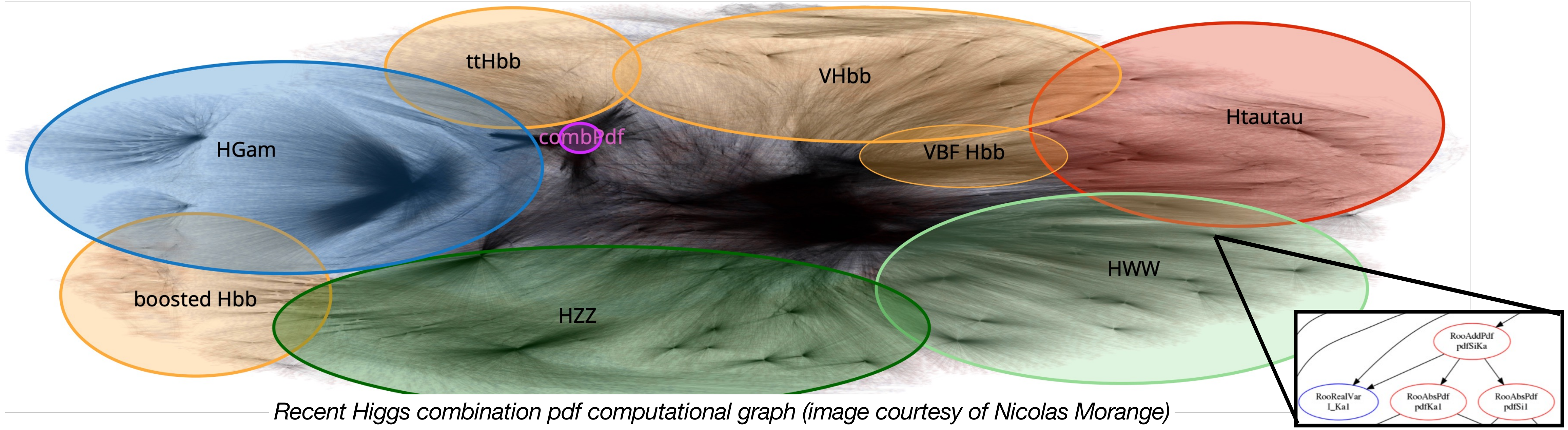
- In practice though, models quickly grow quite convoluted, Higgs combination fits for example incorporate hundreds of smaller likelihood models with varying structures and data



Recent Higgs combination pdf computational graph (image courtesy of Nicolas Morange)

Background

- In practice though, models quickly grow quite convoluted, Higgs combination fits for example incorporate hundreds of smaller likelihood models with varying structures and data
  - This makes it hard to find any general parallelization strategy with optimal load balancing



Recent Higgs combination pdf computational graph (image courtesy of Nicolas Morange)

- The above likelihood models are those with the longest fit durations, currently taking hours
- The challenge at hand: **Developing a multiprocessing strategy to significantly speed up these complex (Higgs comb type) fits while not compromising on robustness**

Parallelization strategy

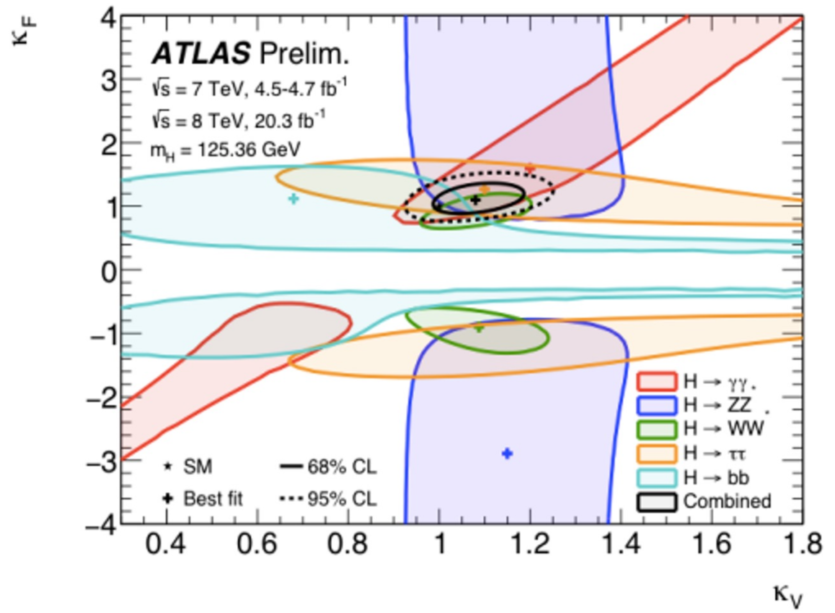
Old & new parallelization implementation

- Original RooFit implements simple parallel strategy ("`NumCPU`")
  - Split calculation of each likelihood call in N equal pieces
  - Load balancing scales poorly for workspaces with many component likelihoods of different sizes and types (binned/unbinned)

- New initiative to parallelize RooFit started ± 7 years ago [1]
  - **Parallelize at level of gradient calculations**, rather than at level of likelihood evaluation
  - New strategy improves load balancing and minimizes communication overhead

- Also overhaul of both internal and user interface classes for likelihood component calculations

- Back-end available from ROOT 6.26, in public interfaces since 6.28



Example Higgs combination fit result, these fits currently easily require many hours to complete

[1] Bos, E. G. P., Burgard, C. D., Croft, V. A., Hageboeck, S., Moneta, L., Pelupessy, I., ... & Verkerke, W. (2020). Faster RooFitting: Automated parallel calculation of … (see last slide)

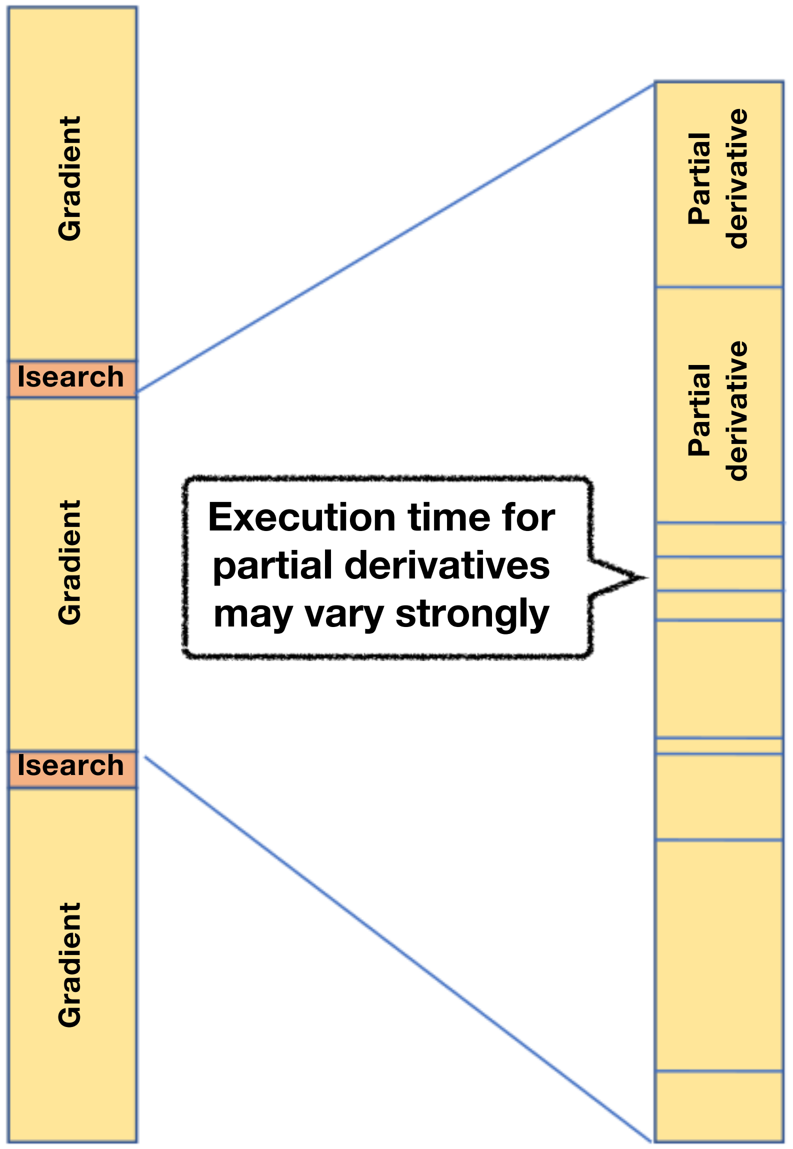
A Brief Reminder on Likelihood Minimisation

- The principle behind most minimization routines consists of

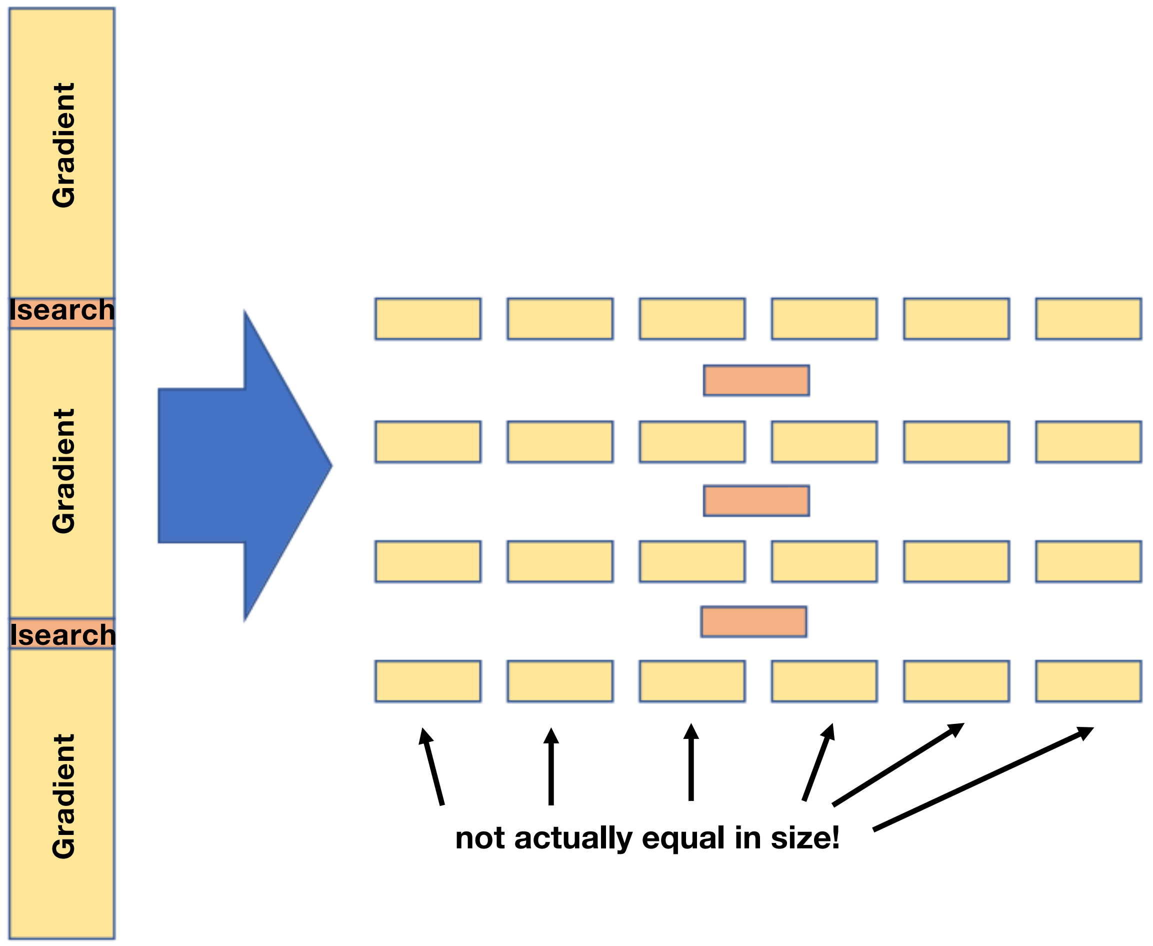
$$\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda\mathbf{p} \text{ such that } f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i)$$

  until some stopping condition is satisfied

- For Minuit2, the minimization routine that RooFit uses, the following holds
  - **p** is the step direction, determined by the variable metric method, the most expensive part of which is the **calculation of the gradient (O(N) likelihood evals)**
  - λ is the step size in the given direction, determined by a line search step, the most expensive part of which is the **evaluation of the full likelihood (O(3) likelihood evals)**



Gradient Parallelisation



- `RooFit::TestStatistics` splits the gradient into individual partial derivative tasks

- The task (partial derivatives) sizes may vary strongly due to
  - Most components only being dependent on subset of parameters, thus not all components need evaluation for every partial derivative
  - Varying likelihood component calculation complexity

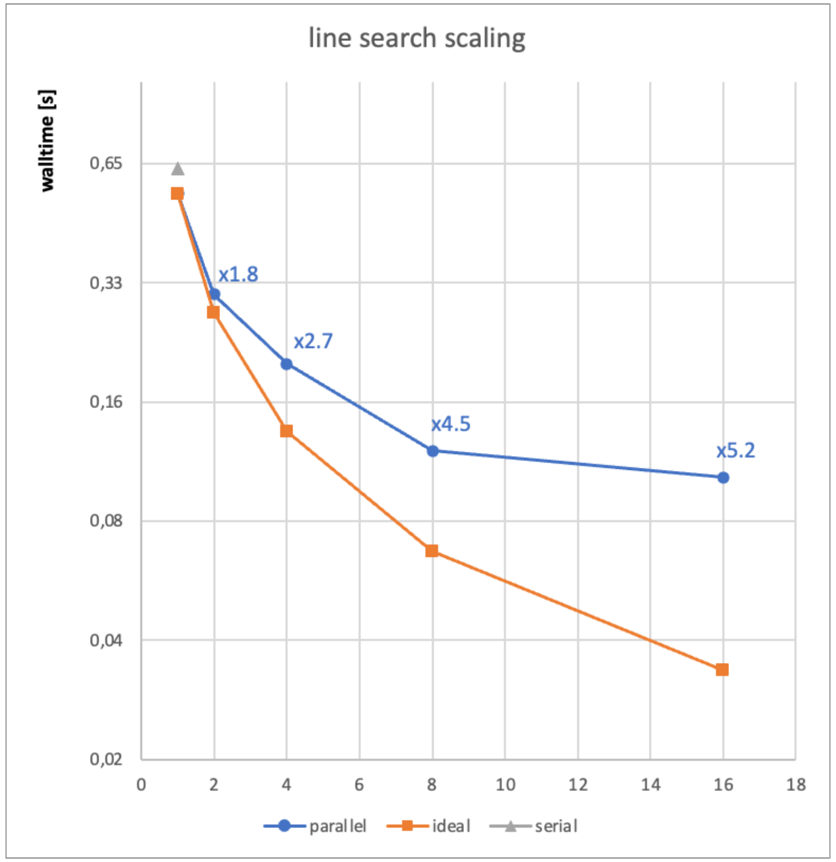
- Dynamic load balancing is crucial and is currently addressed by
  - "Work stealing" algorithm
  - Task ordering by duration

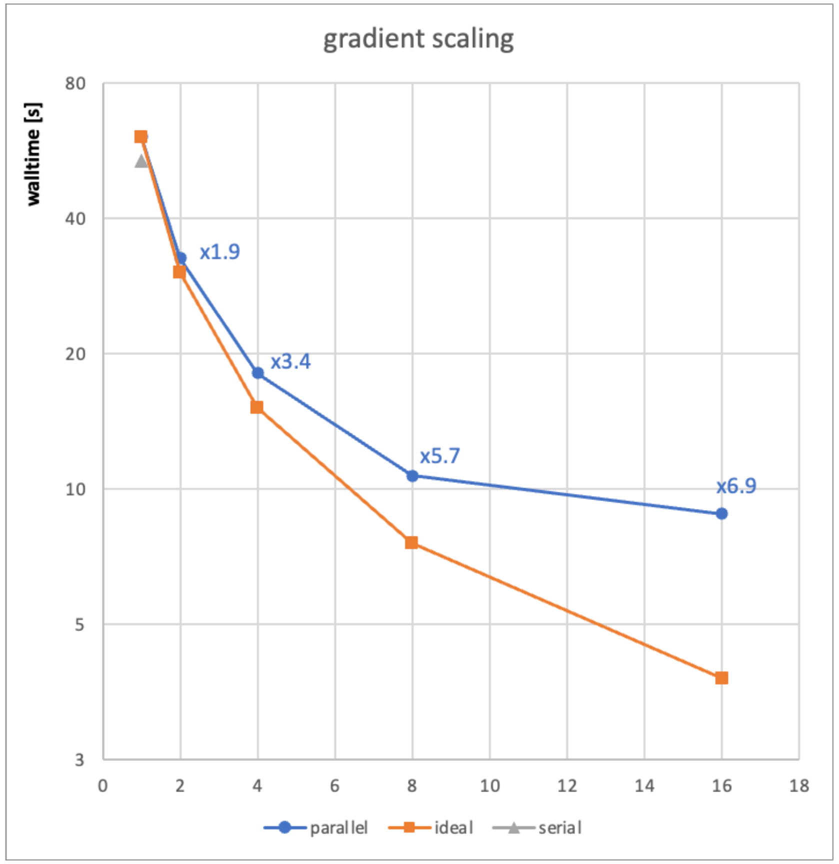
RooFit::MultiProcess + ::TestStatistics

- **Back-end** implementation choices (hidden for most users):
  - `MultiProcess`: parallel **processes**, not threads
    - Bypasses thread safety concerns
    - Requires communication (ZeroMQ) → overhead → **best for large parallel tasks**
    - In theory allows extension towards multiple machines (not currently planned)
  - `TestStatistics`: new classes for likelihoods, separate statistics concepts from computational details
    - Refactoring of the RooNLLVar – RooAbsOptTestStatistic – RooAbsTestStatistic tree
      - Ease maintenance and extensibility
    - Functional, open for testing and feedback
      - Consolidation with old RooNLLVar infra to be planned

Benchmarks

Scaling of Line Search and Gradient





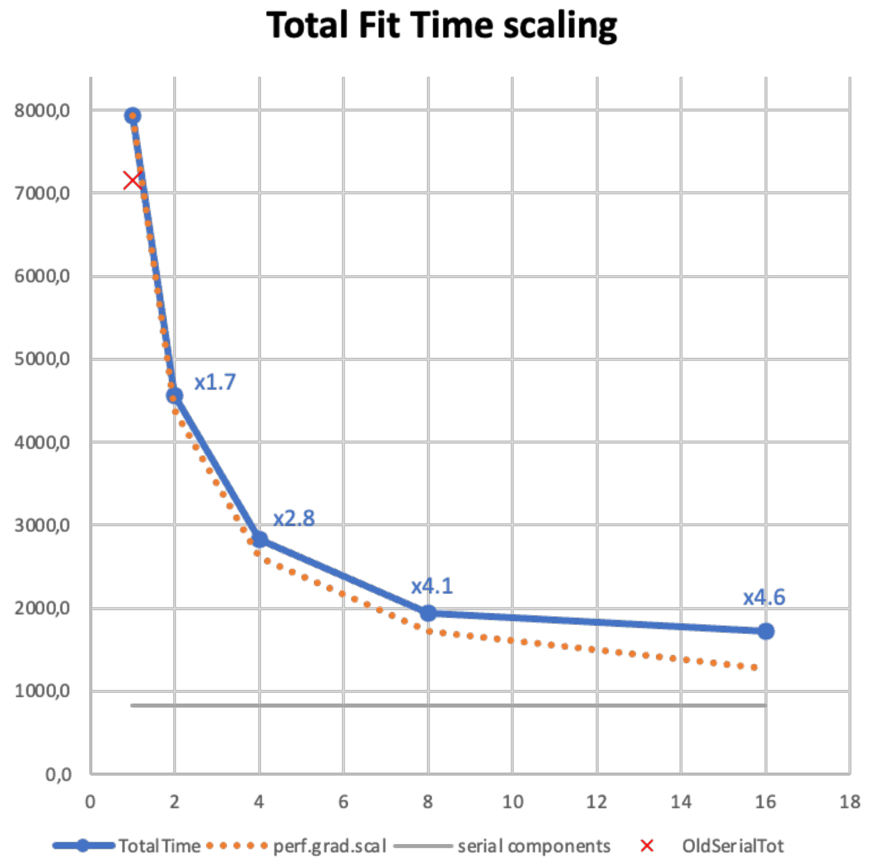
- Used recent Higgs combination workspace produced for 10 year Higgs anniversary paper [2]
- The line search parallelization is still in testing, gradient can be used out of the box since ROOT 6.28
  - For the line search timings $H \rightarrow \gamma\gamma$ was removed from the combination workspace

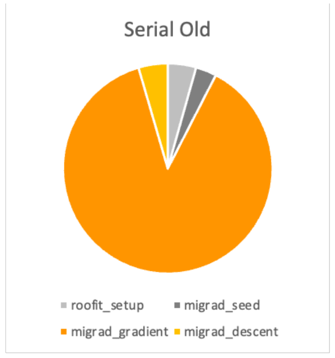
[2] The ATLAS Collaboration. A detailed map of Higgs boson interactions by the ATLAS experiment ten years after the discovery. Nature 607, 52–59 (2022). https://doi.org/10.1038/s41586-022-04893-w

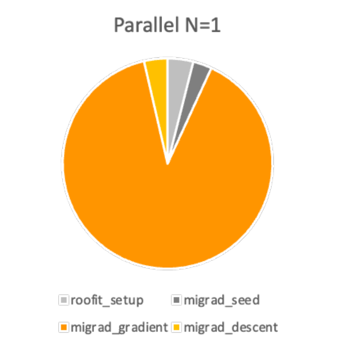
Full Higgs Combination Fit Scaling (2022)

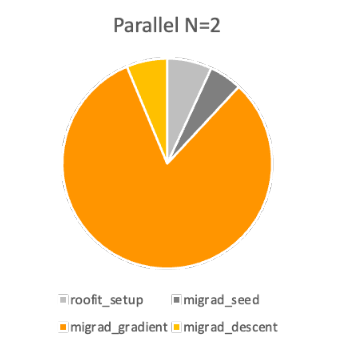
- With gradient parallelization the achieved speedup with 16 workers is 4.6, including all serial components
  - Walltime down from **2h12m** to **29m**
  - At that point, nearly half of the walltime is spent in serial parts

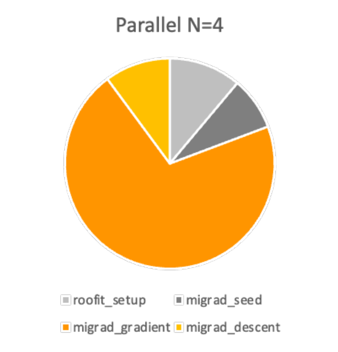
- With line search parallelization fully integrated we can **reasonably expect** to reach a total speedup of 5.3
  - Would bring walltime down to 25 minutes

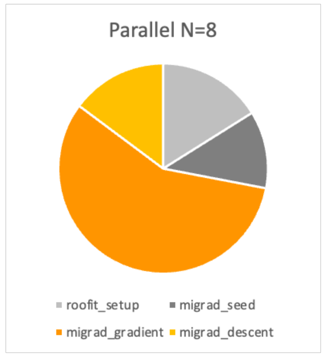


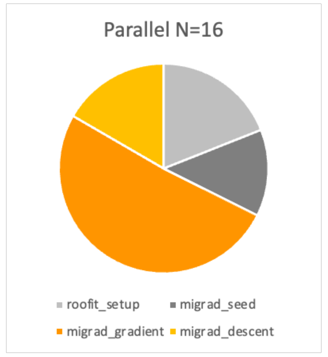


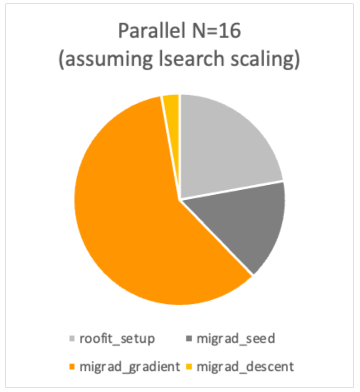






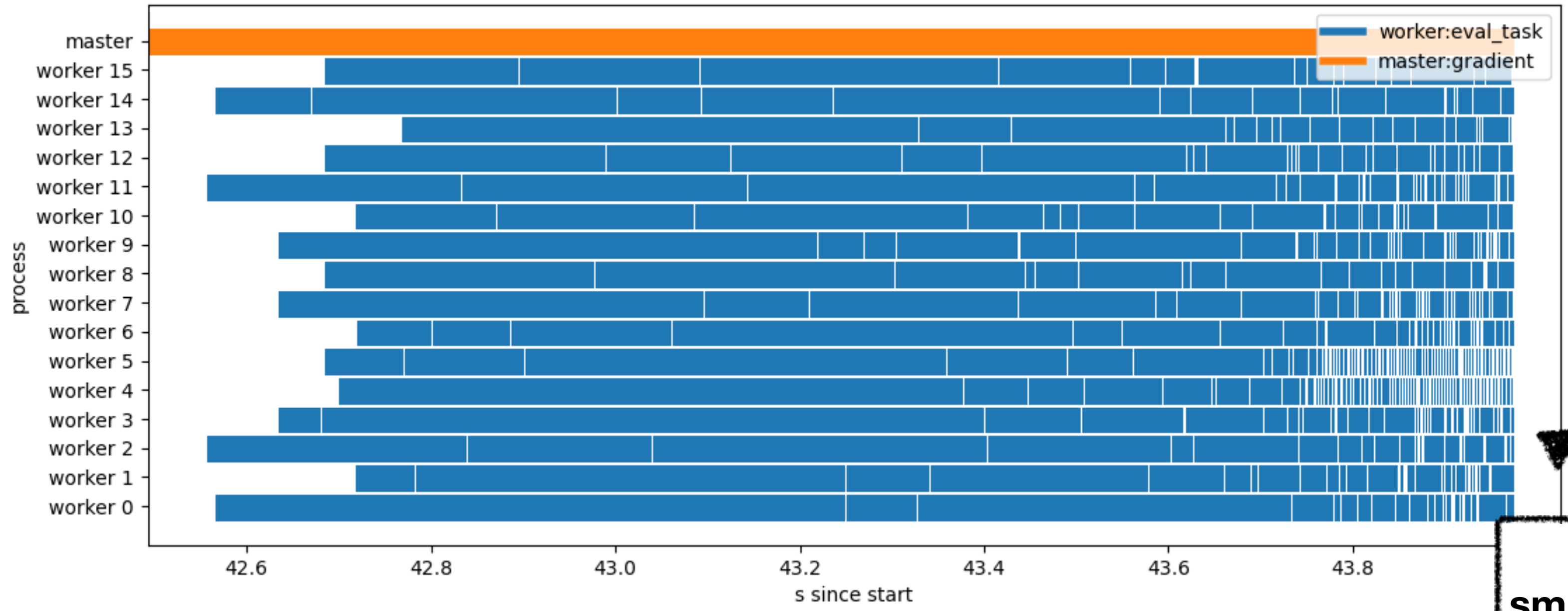






Optional: Task Ordering Optimisation

- The ordering of parallel tasks can significantly impact the total runtime of a parallel program
  - Suboptimal ordering in cases where task duration varies strongly can cause processes to idle

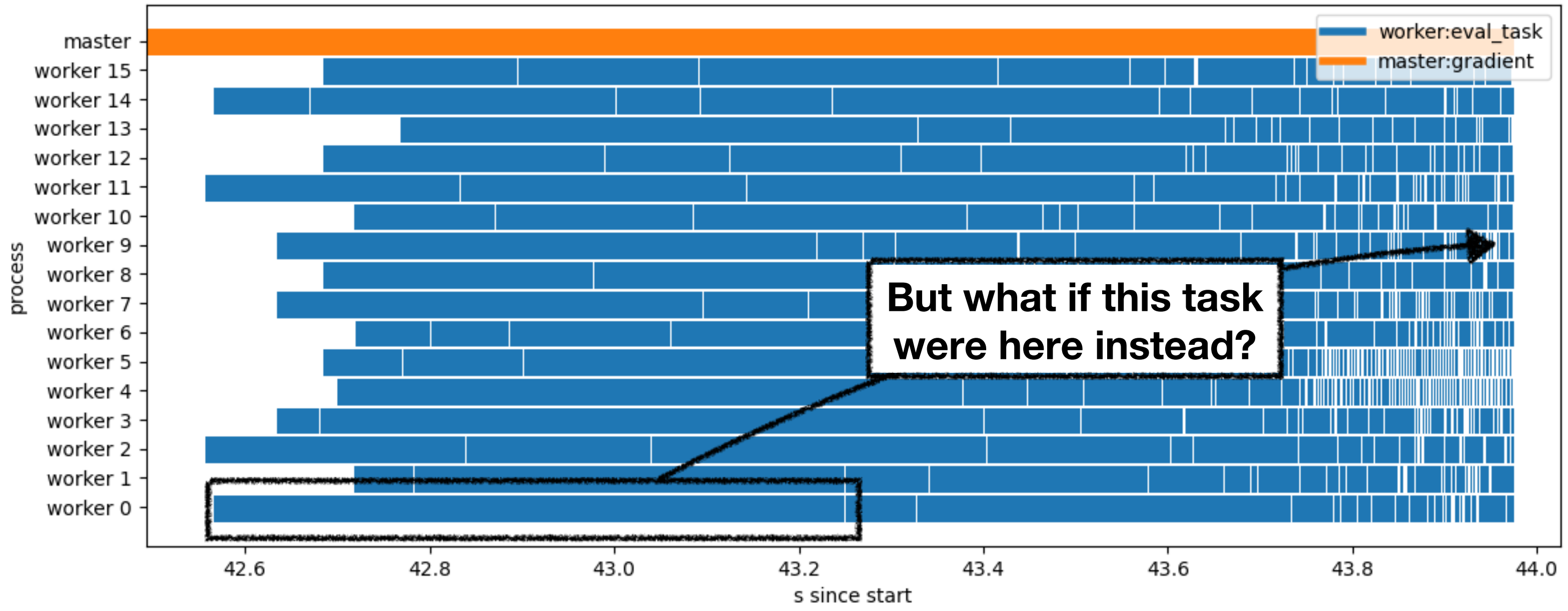


When ending with the smallest jobs workers do not have to wait for each other

- `RooFit::MultiProcessing` implements custom task ordering
  - Can be dynamically updated with timing information as the variable metric steps progress
  - Reduces gradient calculation time by more than 5% for 10 workers “for free”

Optional: Task Ordering Optimisation

- The ordering of parallel tasks can significantly impact the total runtime of a parallel program
  - Suboptimal ordering in cases where task duration varies strongly can cause processes to idle



- `RooFit::MultiProcessing` implements custom task ordering
  - Can be dynamically updated with timing information as the variable metric steps progress
  - Reduces gradient calculation time by more than 5% for 10 workers “for free”

Demo time!

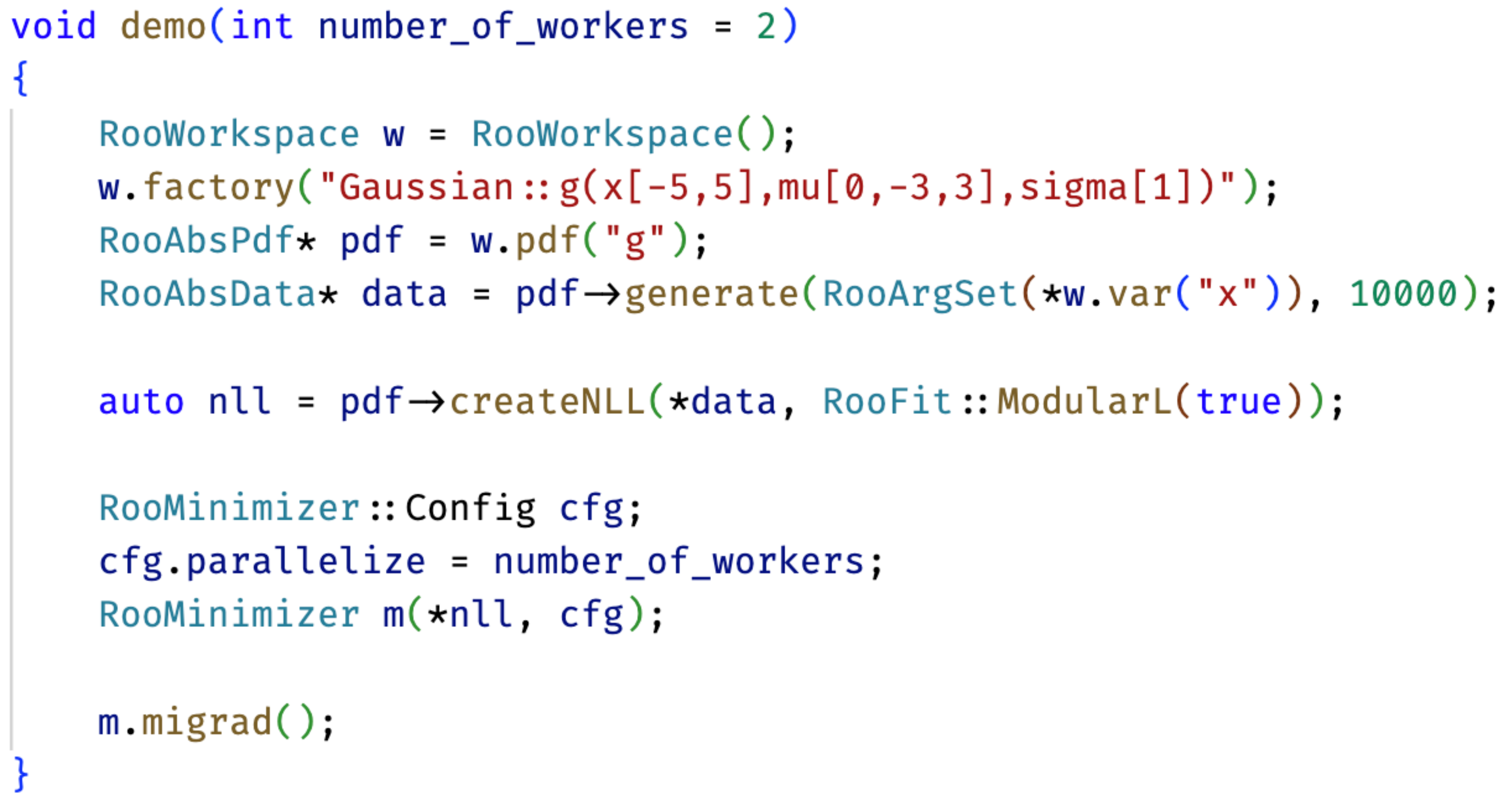
… hope your ROOT compilation completed yet

https://gist.github.com/egpbos/03003b273b8bb2407aa64a575a99a25b
(or go to https://gist.github.com/egpbos, the top one)

Try it out!

Requirements:

- ROOT 6.28+ built with `-Droofit_multiprocess=ON`
- … that's it!

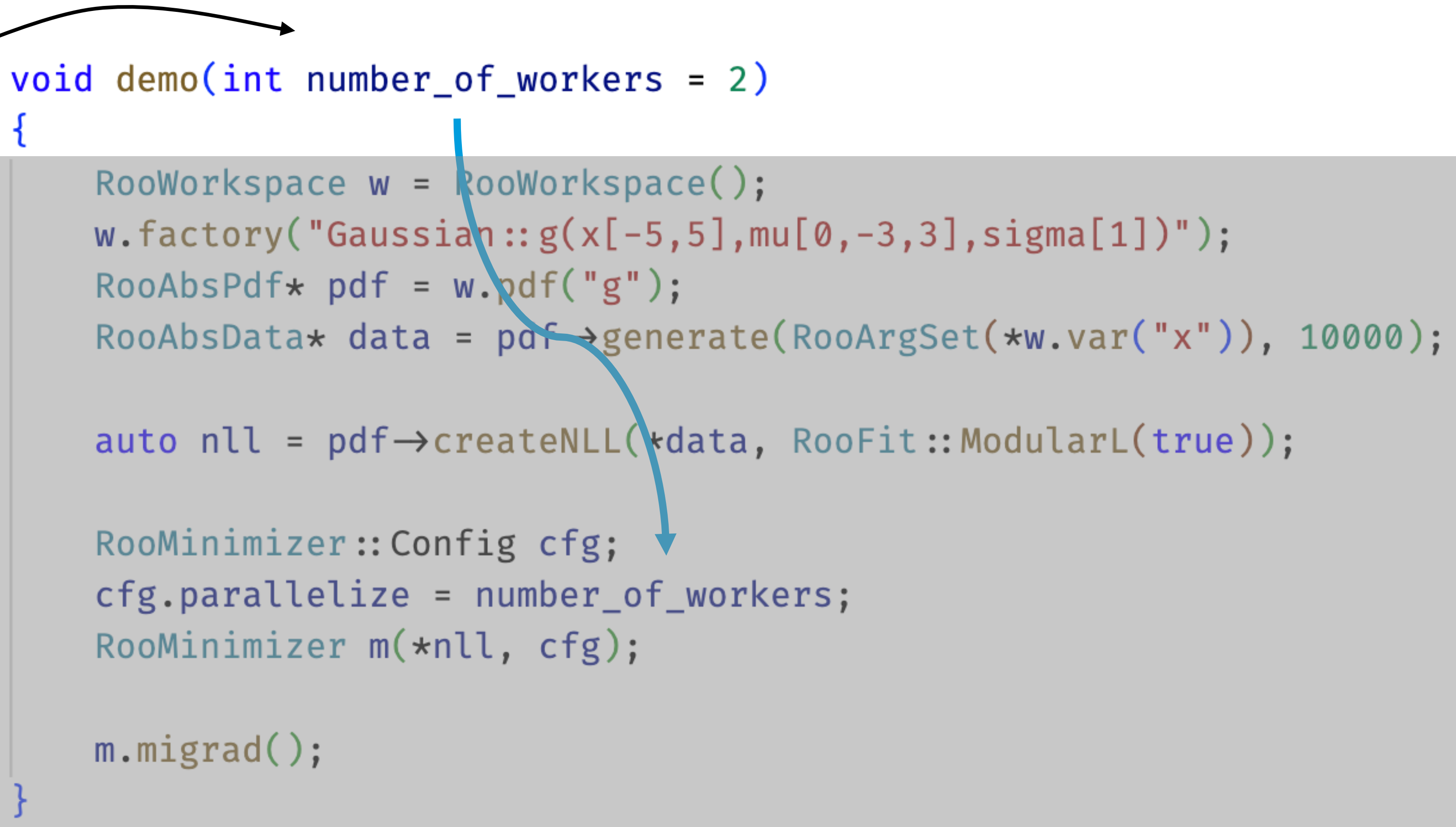


```
void demo(int number_of_workers = 2)
{
    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf* pdf = w.pdf("g");
    RooAbsData* data = pdf->generate(RooArgSet(*w.var("x")), 10000);

    auto nll = pdf->createNLL(*data, RooFit::ModularL(true));

    RooMinimizer::Config cfg;
    cfg.parallelize = number_of_workers;
    RooMinimizer m(*nll, cfg);

    m.migrad();
}
```

https://gist.github.com/egpbos/03003b273b8bb2407aa64a575a99a25b
(or go to https://gist.github.com/egpbos, the top one)

Try it out!

Specify number of workers to use



```
void demo(int number_of_workers = 2)
{
    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf* pdf = w.pdf("g");
    RooAbsData* data = pdf→generate(RooArgSet(*w.var("x")), 10000);

    auto nll = pdf→createNLL(*data, RooFit::ModularL(true));

    RooMinimizer::Config cfg;
    cfg.parallelize = number_of_workers;
    RooMinimizer m(*nll, cfg);

    m.migrad();
}
```

https://gist.github.com/egpbos/03003b273b8bb2407aa64a575a99a25b
(or go to https://gist.github.com/egpbos, the top one)

Try it out!

Specify number of workers to use

Create or import a workspace
(try your own!)

```
void demo(int number_of_workers = 2)
{
    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf* pdf = w.pdf("g");
    RooAbsData* data = pdf->generate(RooArgSet(*w.var("x")), 10000);

    auto nll = pdf->createNLL(*data, RooFit::ModularL(true));

    RooMinimizer::Config cfg;
    cfg.parallelize = number_of_workers;
    RooMinimizer m(*nll, cfg);

    m.migrad();
}
```

netherlands eScience center

https://gist.github.com/egpbos/03003b273b8bb2407aa64a575a99a25b
(or go to https://gist.github.com/egpbos, the top one)

Try it out!

Specify number of workers to use

Create or import a workspace

Build a likelihood with option `RooFit::ModularL(true)` (instantiates a `RooRealL`).

```
void demo(int number_of_workers = 2)
{
    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf* pdf = w.pdf("g");
    RooAbsData* data = pdf->generate(RooArgSet(*w.var("x")), 10000);

    auto nll = pdf->createNLL(*data, RooFit::ModularL(true));

    RooMinimizer::Config cfg;
    cfg.parallelize = number_of_workers;
    RooMinimizer m(*nll, cfg);

    m.migrad();
}
```

https://gist.github.com/egpbos/03003b273b8bb2407aa64a575a99a25b
(or go to https://gist.github.com/egpbos, the top one)

Try it out!

Specify number of workers to use

Create or import a workspace

Build a likelihood with option `RooFit::ModularL(true)` (instantiates a `RooRealL`).

Create the minimizer, using the Config object (has more options)

```cpp
void demo(int number_of_workers = 2)
{
    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf* pdf = w.pdf("g");
    RooAbsData* data = pdf->generate(RooArgSet(*w.var("x")), 10000);

    auto nll = pdf->createNLL(*data, RooFit::ModularL(true));

    RooMinimizer::Config cfg;
    cfg.parallelize = number_of_workers;
    RooMinimizer m(*nll, cfg);

    m.migrad();
}
```

netherlands eScience center

https://gist.github.com/egpbos/03003b273b8bb2407aa64a575a99a25b
(or go to https://gist.github.com/egpbos, the top one)

Try it out!

Specify number of workers to use

Create or import a workspace

Build a likelihood with option `RooFit::ModularL(true)` (instantiates a `RooRealL`).

Create the minimizer, using the Config object (has more options)

Minimize!

```
void demo(int number_of_workers = 2)
{
    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf* pdf = w.pdf("g");
    RooAbsData* data = pdf->generate(RooArgSet(*w.var("x")), 10000);

    auto nll = pdf->createNLL(*data, RooFit::ModularL(true));

    RooMinimizer::Config cfg;
    cfg.parallelize = number_of_workers;
    RooMinimizer m(*nll, cfg);

    m.migrad();
}
```

https://gist.github.com/egpbos/03003b273b8bb2407aa64a575a99a25b
(or go to https://gist.github.com/egpbos, the top one)

Try it out!

(on ROOT 6.28+ built with `-Droofit_multiprocess=ON`)

Specify number of workers to use

Create or import a workspace

Build a likelihood with option `RooFit::ModularL(true)` (instantiates a `RooRealL`).

Create the minimizer, using the Config object (has more options)

Minimize!

```
void demo(int number_of_workers = 2)
{
    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf* pdf = w.pdf("g");
    RooAbsData* data = pdf->generate(RooArgSet(*w.var("x")), 10000);

    auto nll = pdf->createNLL(*data, RooFit::ModularL(true));

    RooMinimizer::Config cfg;
    cfg.parallelize = number_of_workers;
    RooMinimizer m(*nll, cfg);

    m.migrad();
}
```

Conclusions

Conclusions

- A new parallel implementation of RooFit was developed that parallelizes at the level of **gradient calculations** and optionally over events or components during **line search**
  - Scales well through dynamic load-balancing

- RooFit speed improvements in multiple directions
  - Automatic differentiation
  - New computation back-ends: CPU vectorization, GPU
  - Multiprocessing

- Consolidation of these efforts is an important next step on the agenda
  - For example, multiprocessing and vectorized computations optimize at a different level and could be used simultaneously

Bonus: caching ftw

TensorFlow experiments

	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)

From ACAT 2019 talk

netherlands eScience center

Try it out!

And grab/call me if it doesn't work

Thank you for your attention!

https://gist.github.com/egpbos/03003b273b8bb2407aa64a575a99a25b

p.bos@esciencecenter.nl

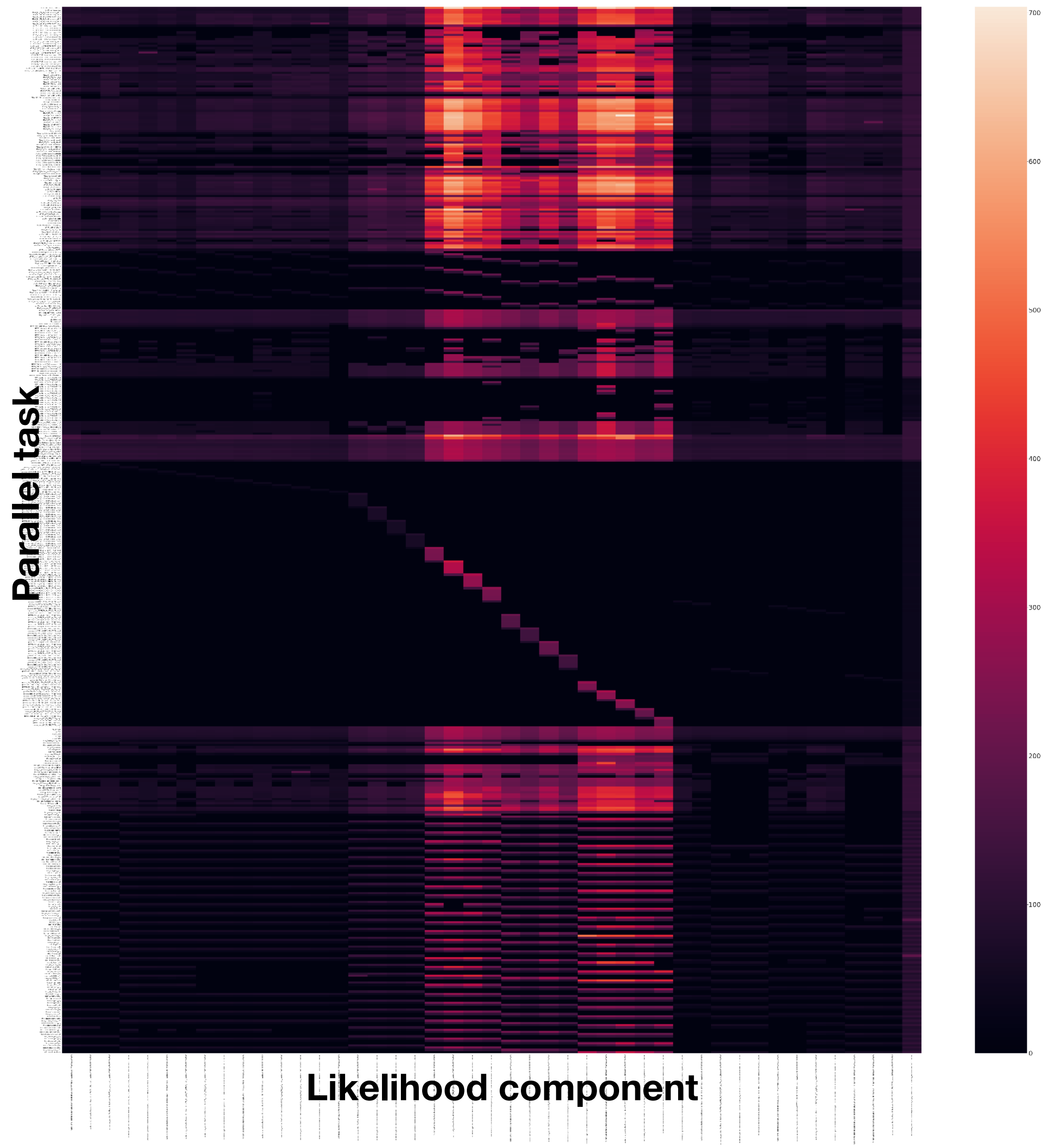
+31 6 10795874

Moar cool stuff!

Likelihood fit benchmarking tools by Zef Wolffs

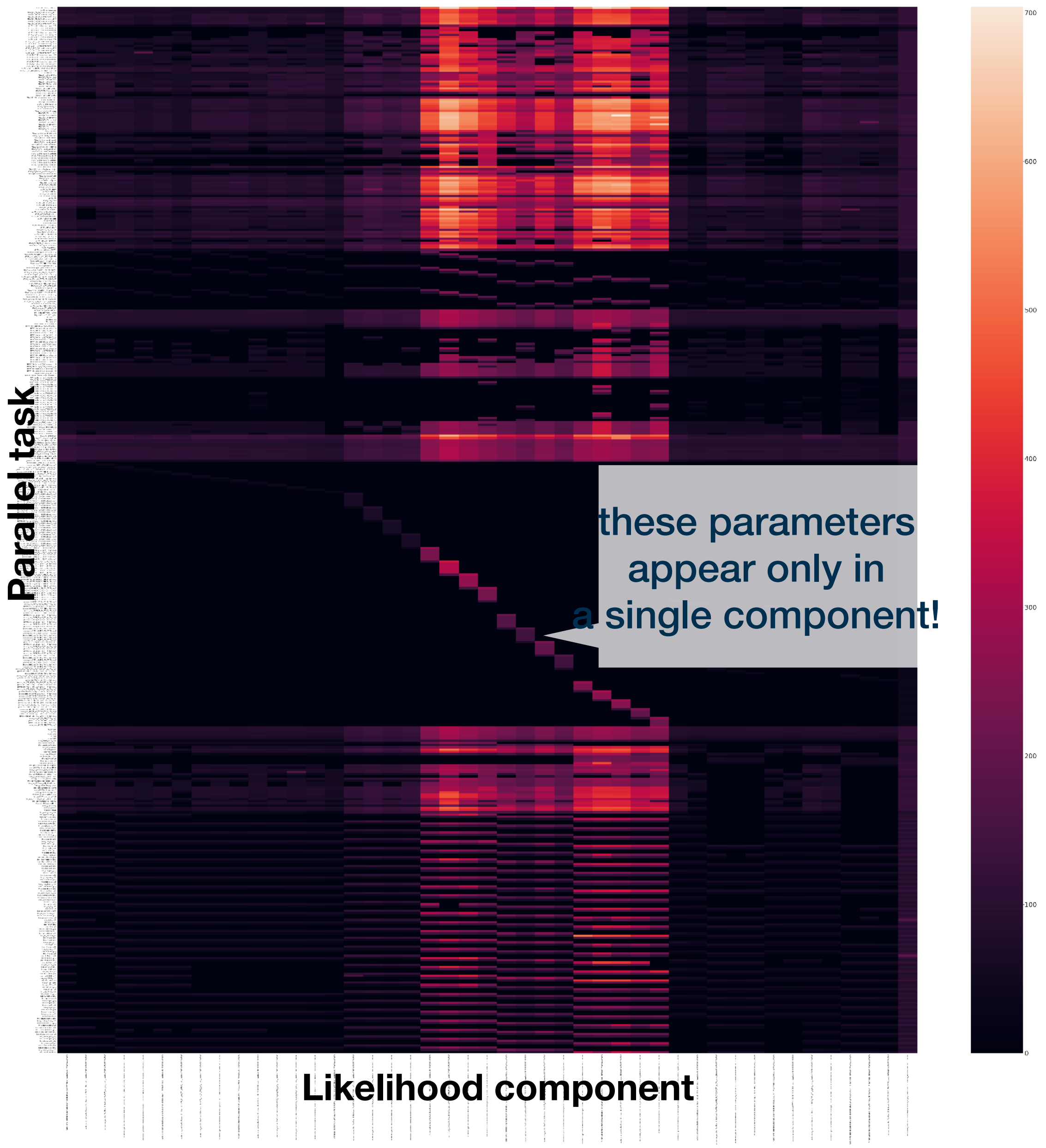
Gradient Parallelisation - Benchmarking

- Not all parameters present in all likelihood components
  - If this is the case, no evaluation is necessary and the result is returned immediately
  - Explains the black regions in heatmap
- Benchmarking tools now available in RooFit
  - `TimingAnalysis` argument in RooMinimizer enables profiling
  - `RooFit::MultiProcess::HeatmapAnalyzer()` to create a heatmap



Gradient Parallelisation - Benchmarking

- Not all parameters present in all likelihood components
  - If this is the case, no evaluation is necessary and the result is returned immediately
  - Explains the black regions in heatmap
- Benchmarking tools now available in RooFit
  - `TimingAnalysis` argument in RooMinimizer enables profiling
  - `RooFit::MultiProcess::HeatmapAnalyzer()` to create a heatmap

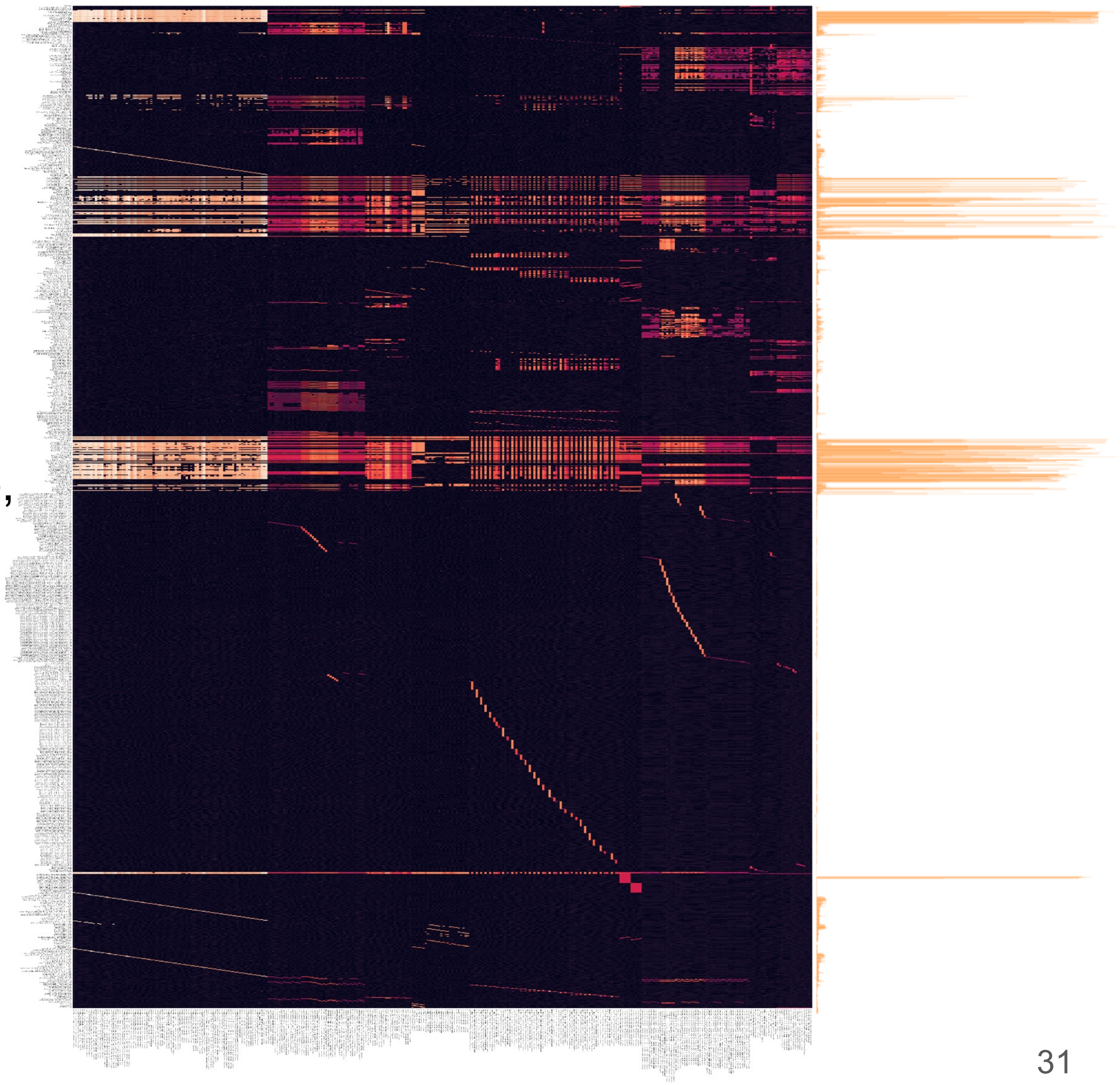


Future Work - Job Scheduling

- This heatmap contains the time expenditures for a Higgs combination workspace, the histogram next to it displays the time expenditure per partial derivative

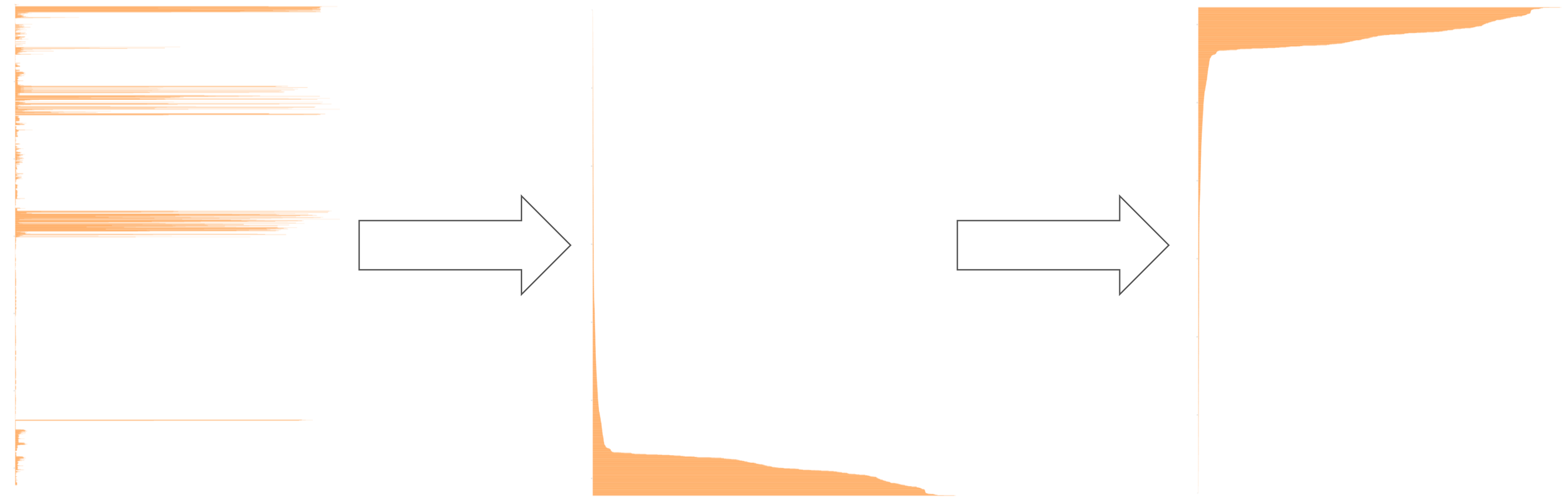
- Note that each partial derivative constitutes a job, i.e. a task to be executed by a worker. In this case, there is a large difference in time needed per task

- Current job scheduling strategy lets workers pick jobs from queue, currently the queue is ordered with the same order as the heatmap



Future Work - Job Scheduling

- Running a simple simulation shows that times can vary based on ordering of jobs, this effect scales directly with the differences in job sizes
  - These simulations were run with real time expenditures from previous slide and assuming no communication overhead



6.6 min. per gradient

6.5 min. per gradient

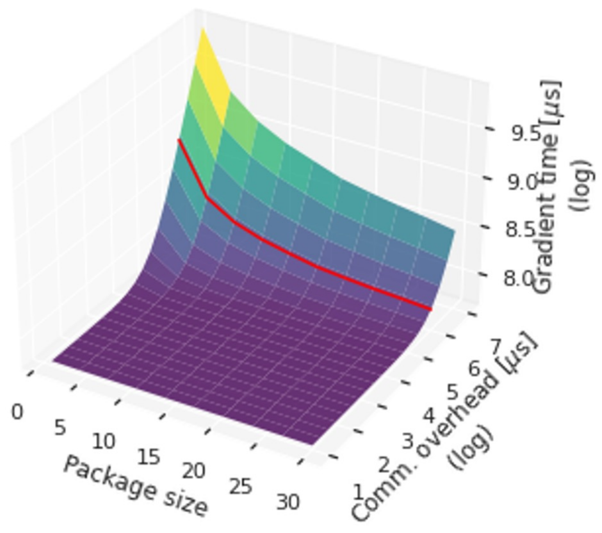
5.5 min. per gradient

Nikhef

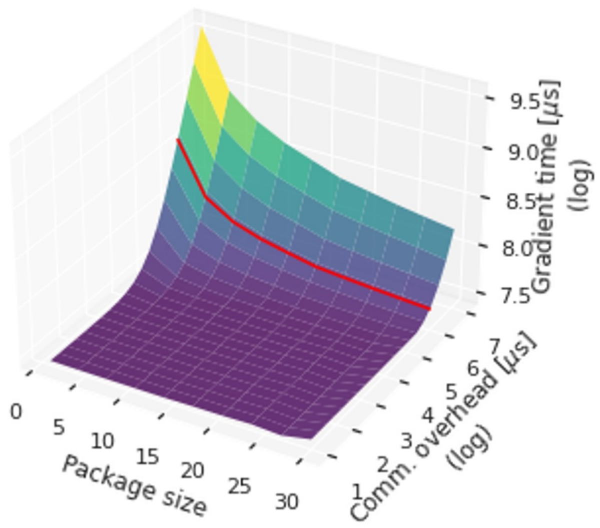
ROOT Data Analysis Framework

Future Work - Job Scheduling

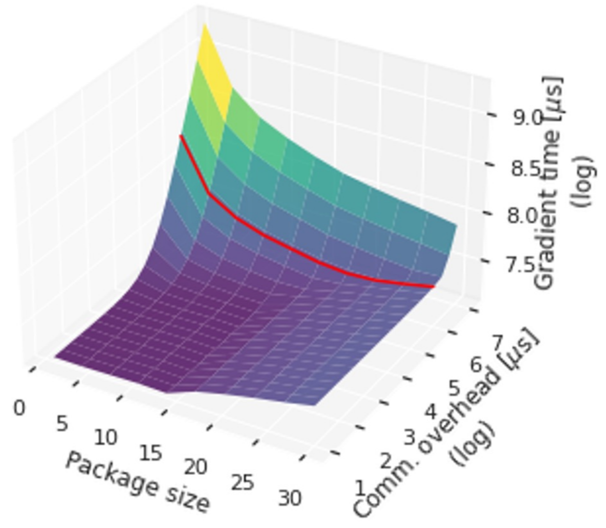
- If communication overhead is dominating, we want to limit the number of times that workers communicate. This can be achieved by sending packages of partial derivatives per job simultaneously to workers.
    - Reduces the number of times communication is done, but increases the time spent per job
- Below plots show the simulated time expenditures per gradient as a function of package size (number of partial derivatives per job) and communication overhead



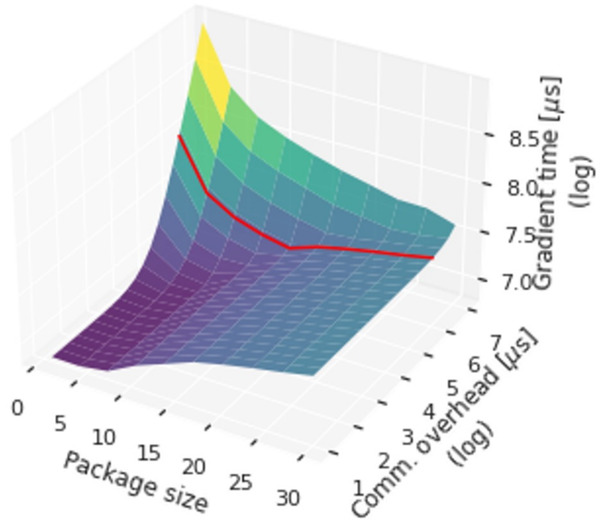
4 workers



8 workers



16 workers



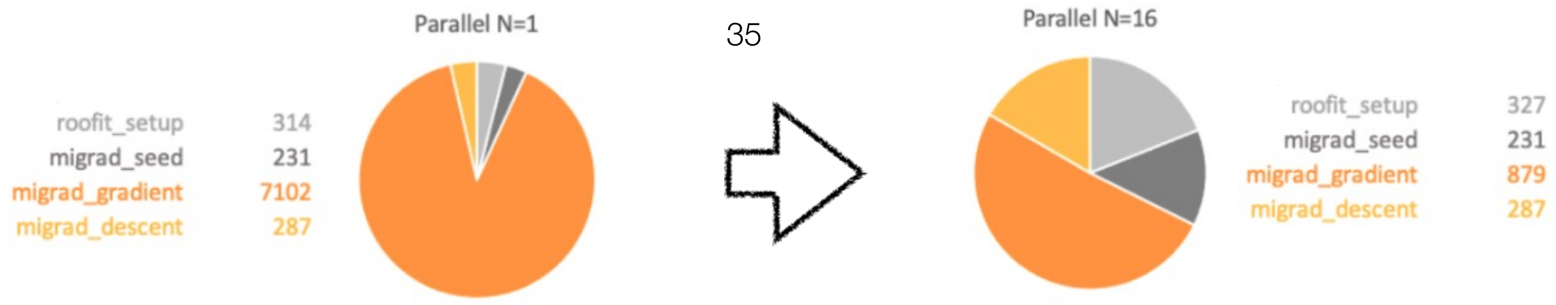
32 workers

Moar details!

the Nitty Gritty™

Likelihood Parallelisation

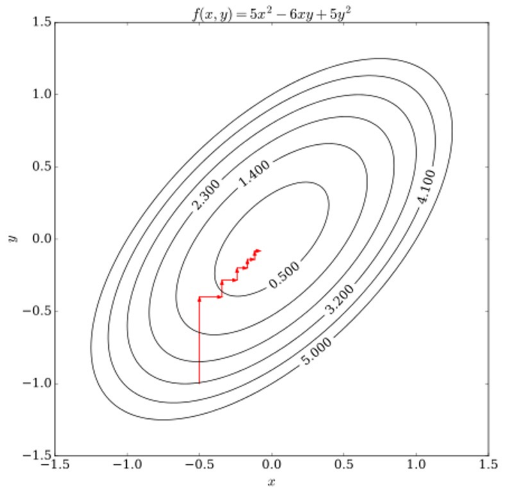
- In some cases, evaluation of the likelihood can be the bottleneck, for example in the calculation of the line search step
  - During the line search step all parameters are typically changed two or three times, requiring an evaluation of all components of the likelihood
  - With the gradient sufficiently optimised, this can become the bottleneck for an entire fit

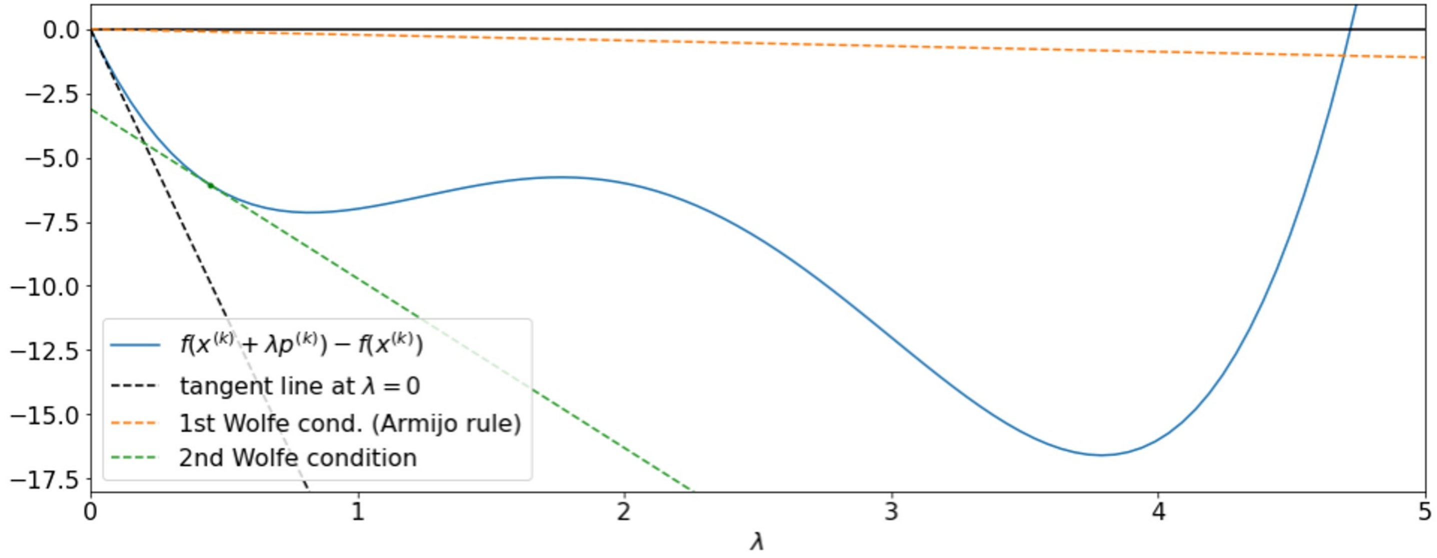


- `RooFit::TestStatistics` has two options for splitting likelihood evaluation into tasks
  - By events: each task is defined by an event range to execute
  - By components: each task is defined by a set of components to execute

Background - Line Search

- The idea is to find an optimal step size λ, given a calculated step direction **p,** that minimizes f(**x**), i.e. $\min_{\lambda} f(\mathbf{x} + \lambda \mathbf{p})$
  - This is essentially another single-dimensional minimization problem, which could be solved again by something like gradient descent
  - Line search in MIGRAD is "inexact", this means that rather than finding the exact minimum of f(**x**) along the line spanned by λ**p**, a sufficient decrease in f(**x**) is found
    - This is because finding the exact minimum might cost a lot of minimization steps, and finding a new step direction may then be preferable
    - Instead of finding global minimum, minimize until conditions are satisfied (e.g. Wolfe conditions)





RooFit::MultiProcess

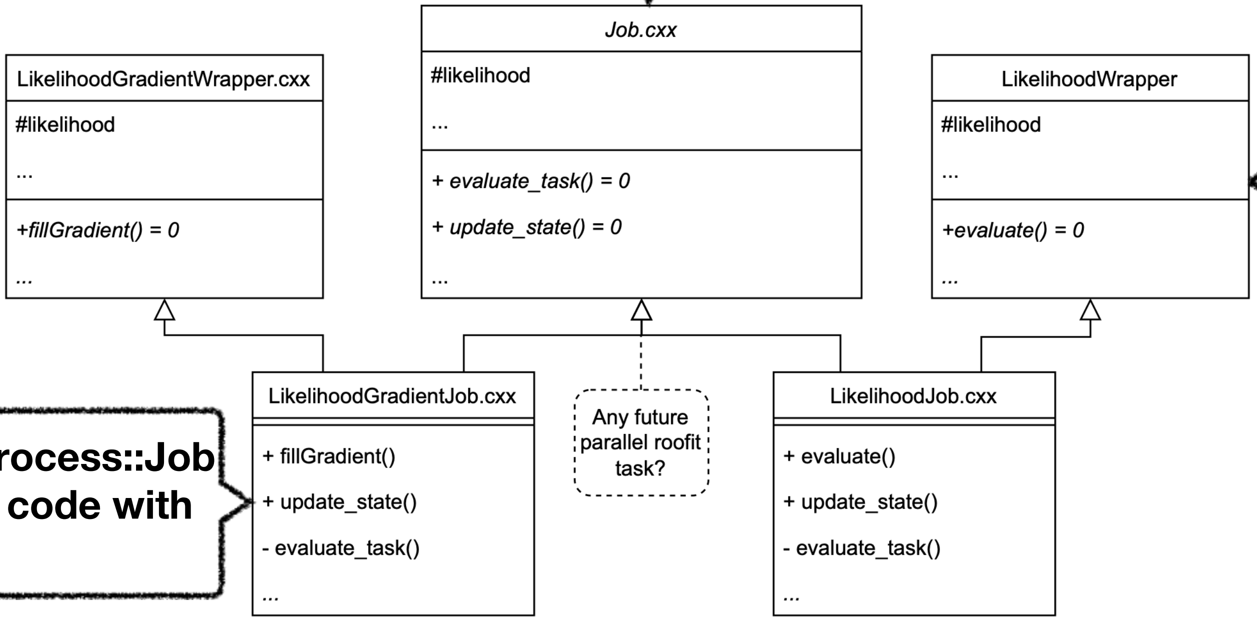
Implementation details

netherlands eScience center

A General Parallel RooFit Framework

- A general parallel framework `RooFit::MultiProcess` was written to serve as a foundation for any RooFit parallelisation efforts
  - Uses ZeroMQ for inter-process communication
  - Interfaces with rest of RooFit through `RooFit::MultiProcess::Job`

RooFit::MultiProcess knows how to interact with "Job" base classes



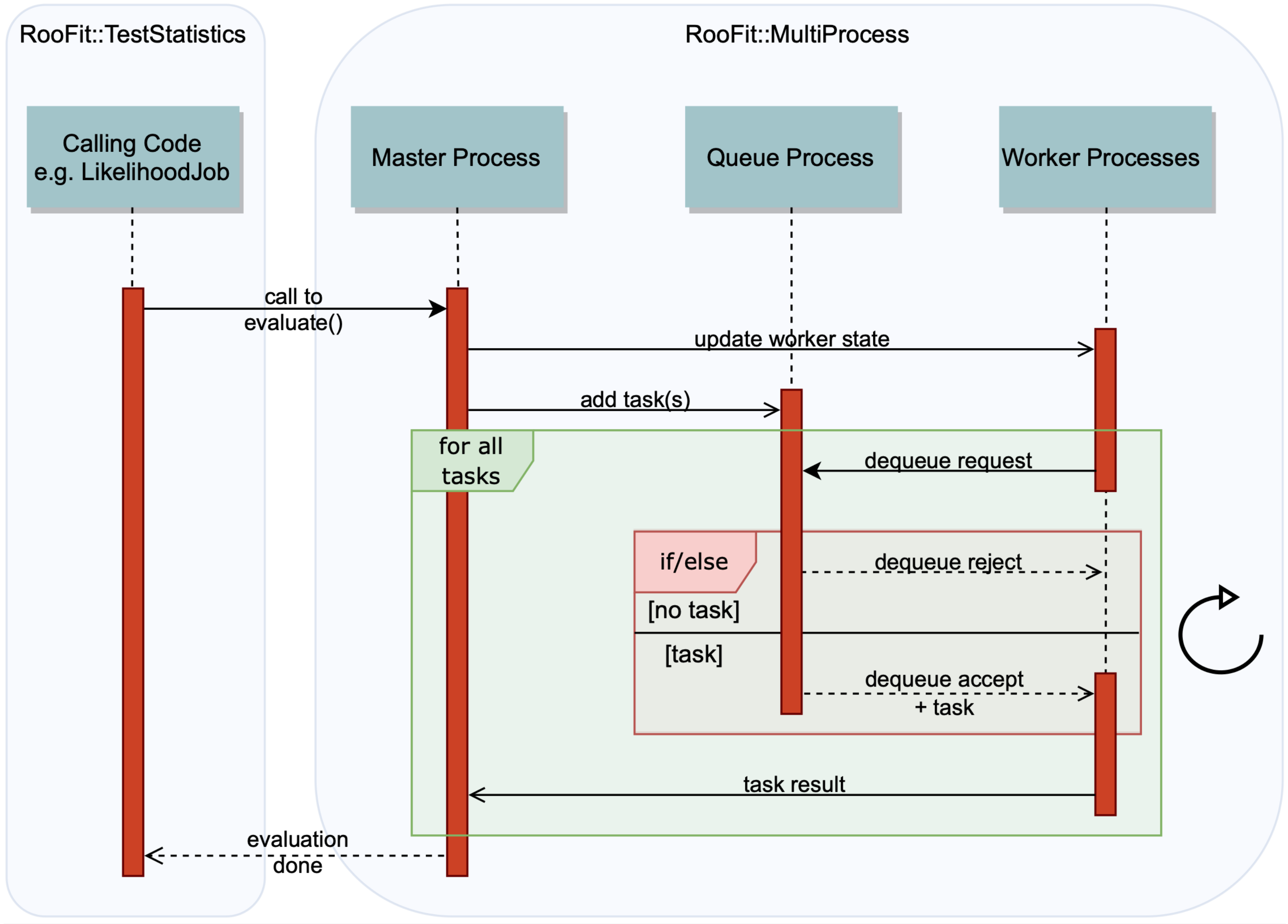
The rest of RooFit knows how to interact with likelihoods, gradients, etc…

Classes that inherit from MultiProcess::Job thus interface multiprocessing code with the rest of RooFit

A General Parallel RooFit Framework

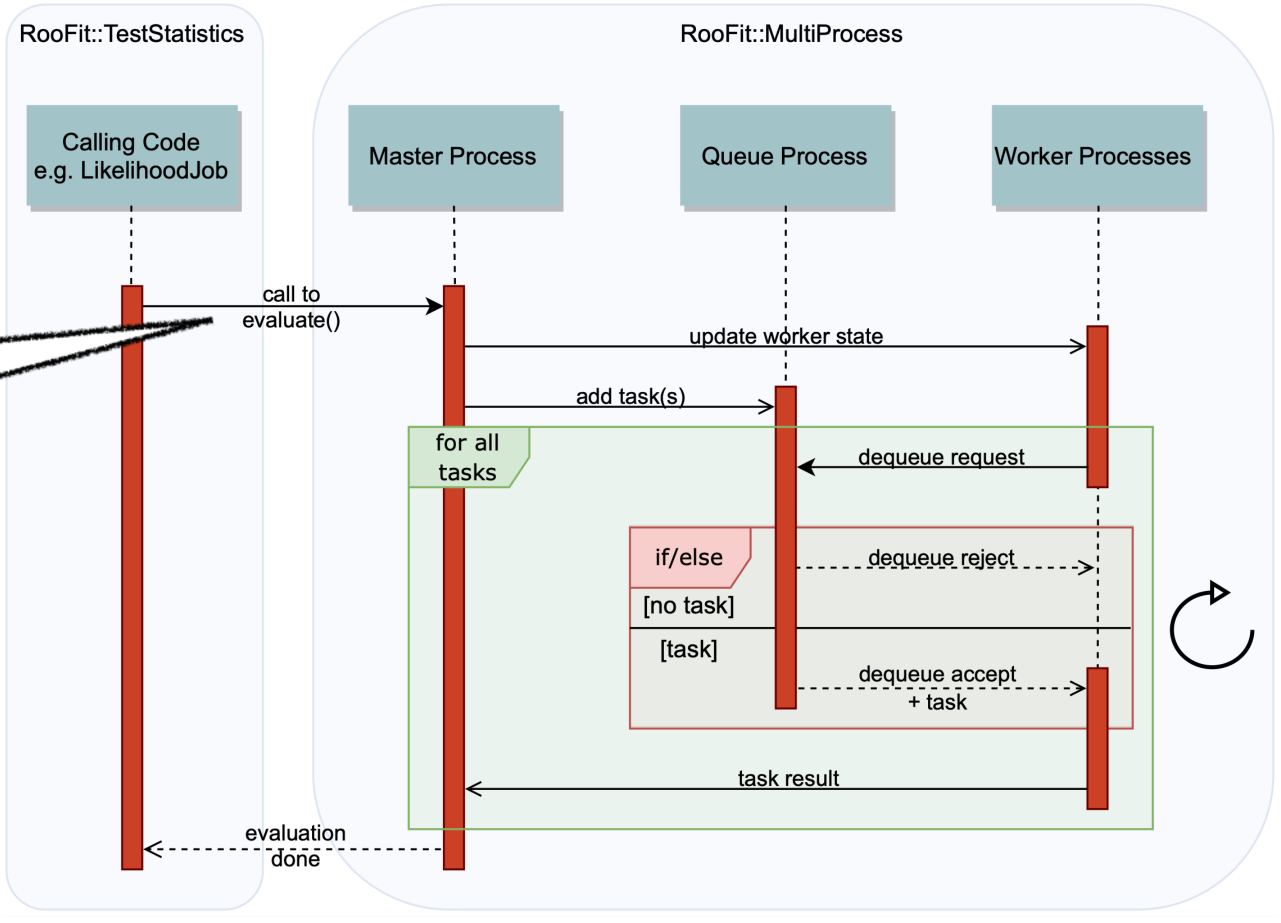
- The UML sequence diagram included on the right displays a simplified version of the `RooFit::MultiProcessing` execution flow

- Much more detailed UML diagrams of `RooFit::MultiProcessing` can be found in previous CHEP proceedings [1]



[1] Bos, EG Patrick, et al. "Faster RooFitting: Automated parallel calculation of collaborative statistical models." Journal of Physics: Conference Series. Vol. 1525. No. 1. IOP Publishing, 2020.

A General Parallel RooFit Framework

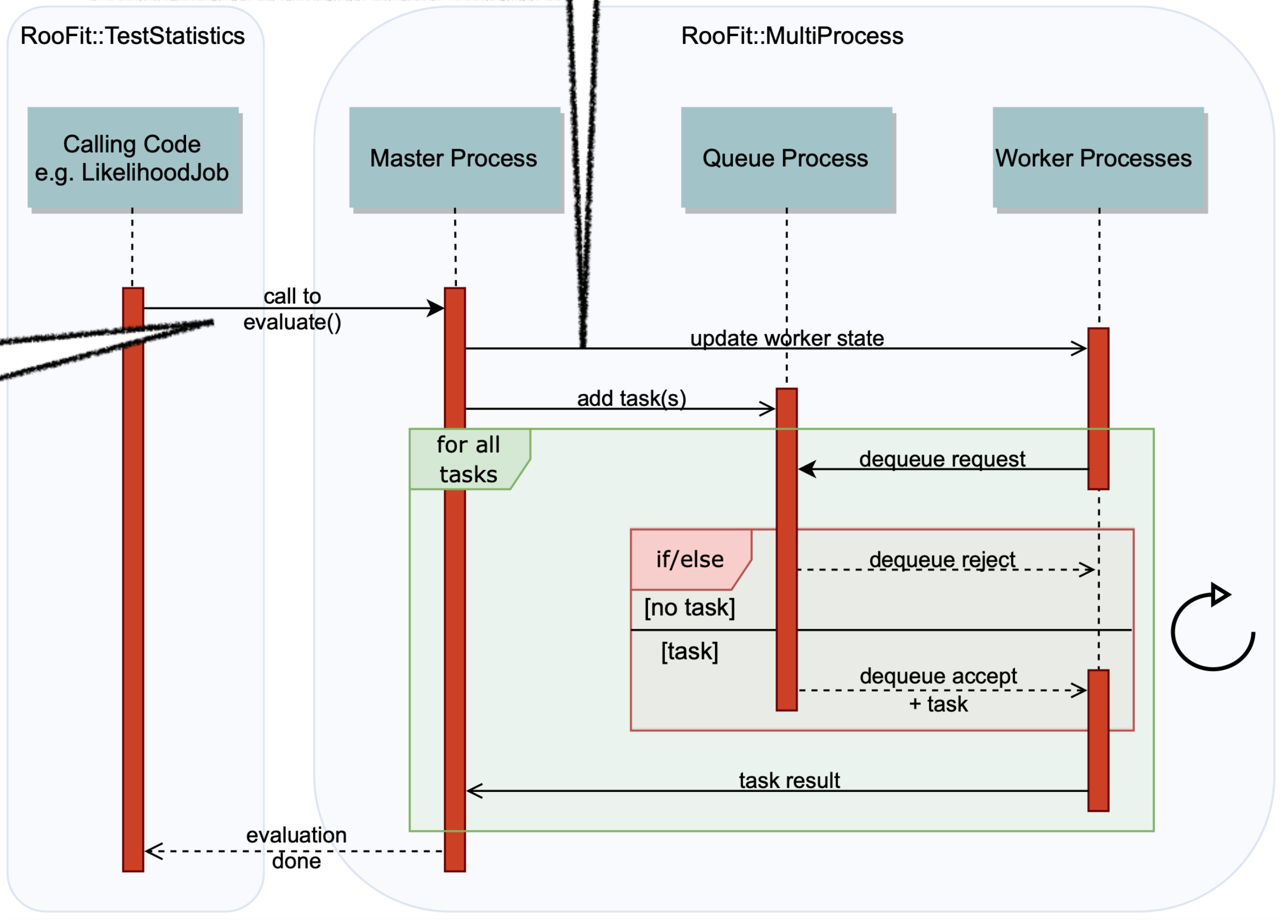


A parallel evaluation starts with some derived class of Job requesting parallel evaluation

A General Parallel RooFit Framework

Master tells workers how to synchronise their state

A parallel evaluation starts with some derived class of Job requesting parallel evaluation

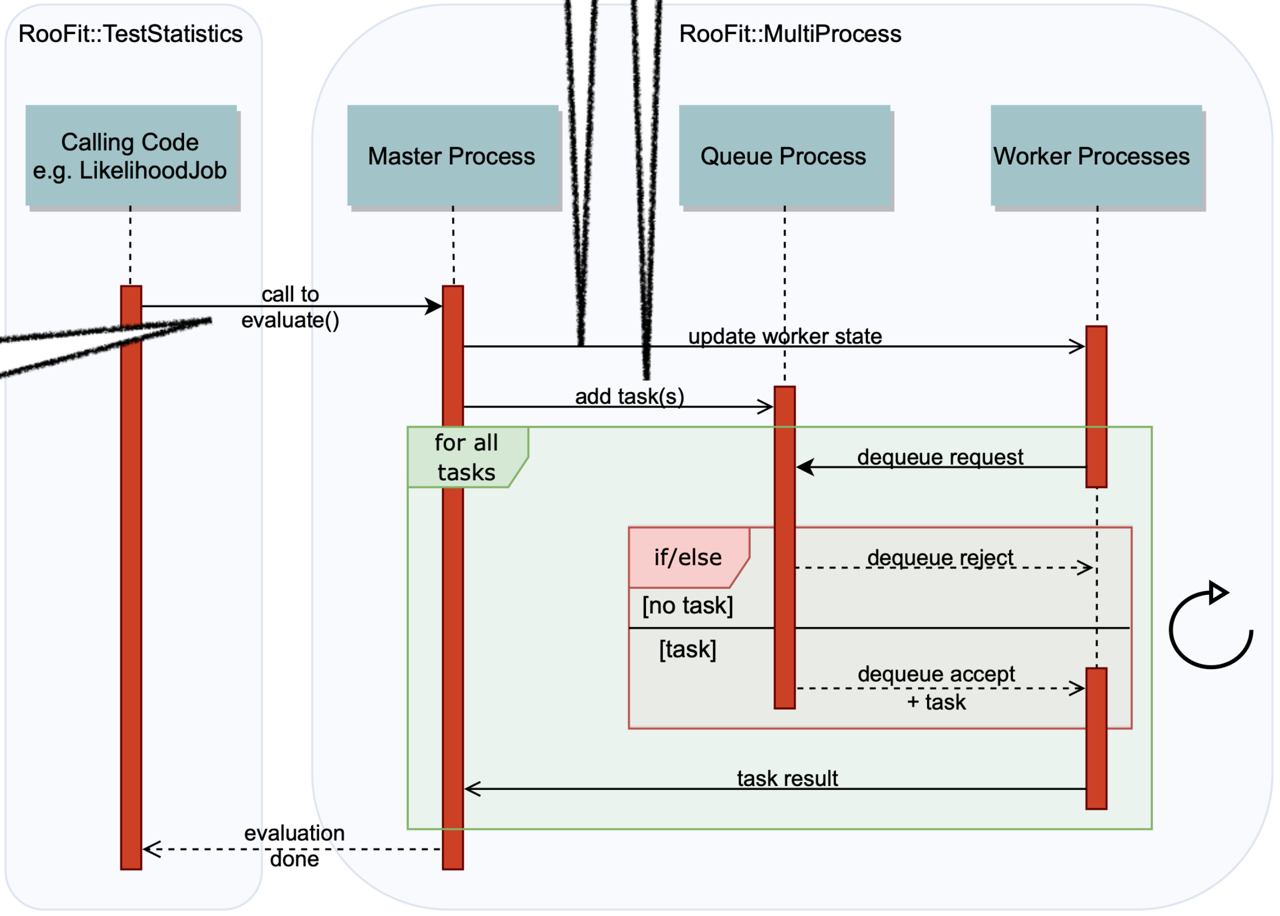


A General Parallel RooFit Framework

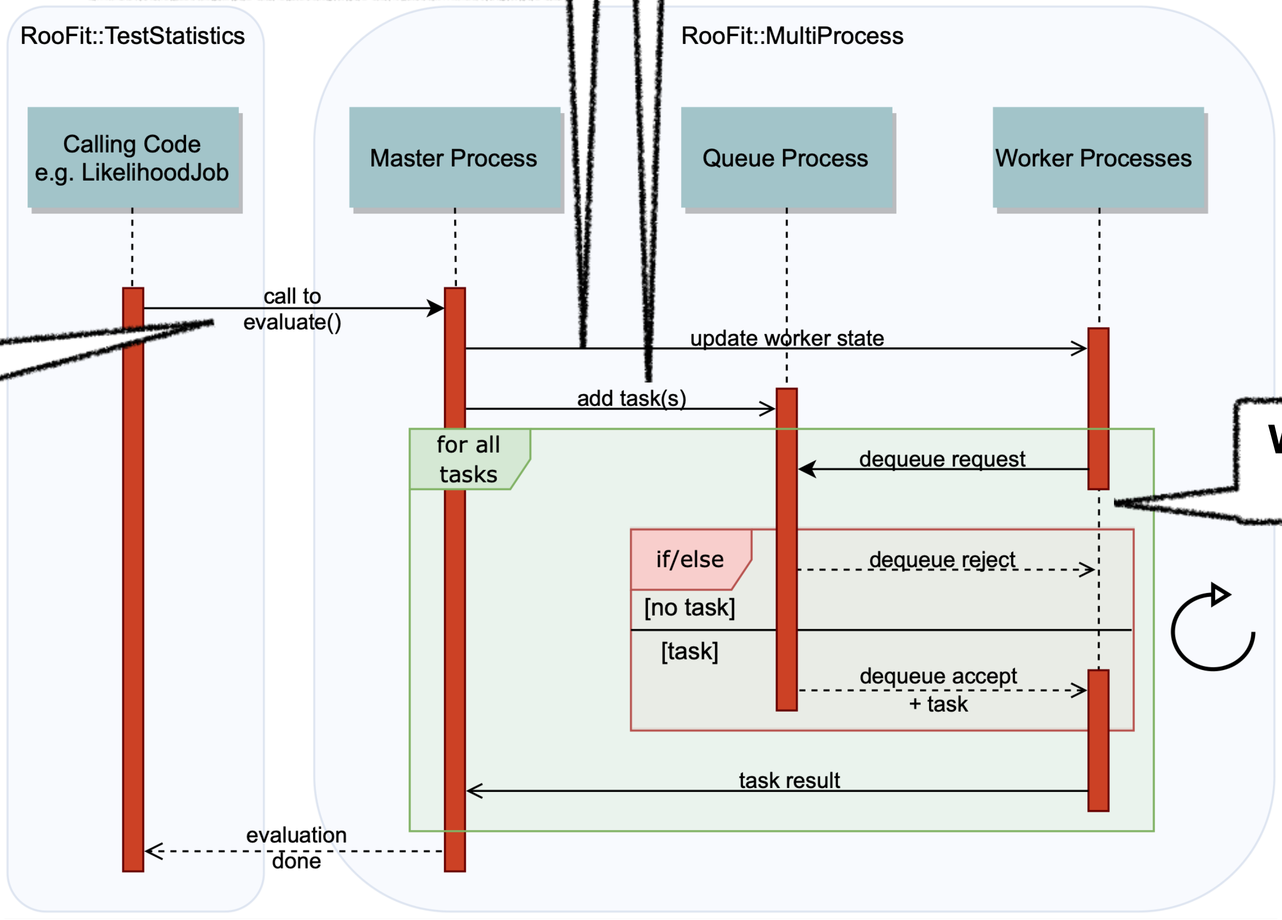
Master tells workers how to synchronise their state

Master submits parallel tasks to queue

A parallel evaluation starts with some derived class of Job requesting parallel evaluation



A General Parallel RooFit Framework



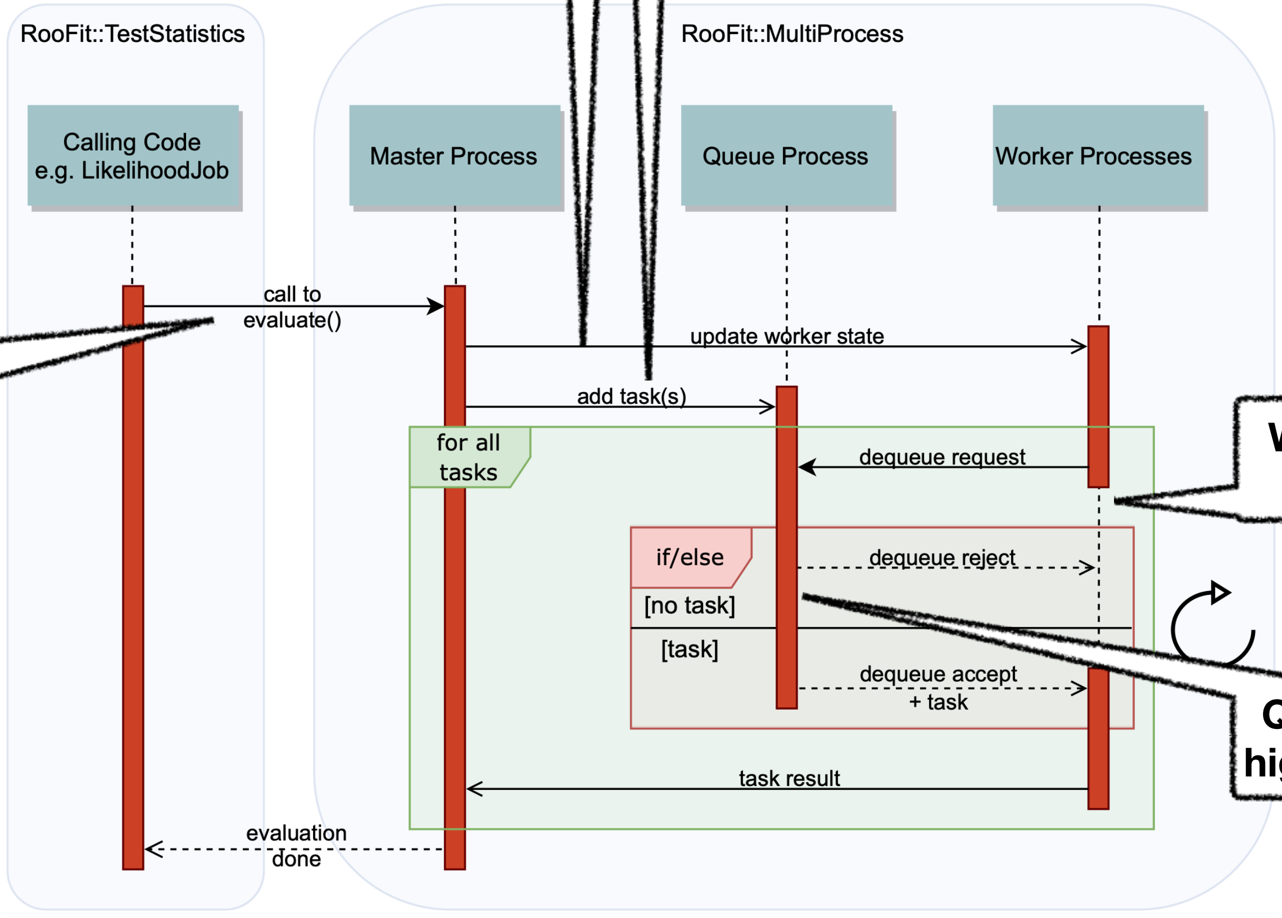
Master tells workers how to synchronise their state

Master submits parallel tasks to queue

A parallel evaluation starts with some derived class of Job requesting parallel evaluation

Workers request tasks from queue in loop

A General Parallel RooFit Framework



Master tells workers how to synchronise their state

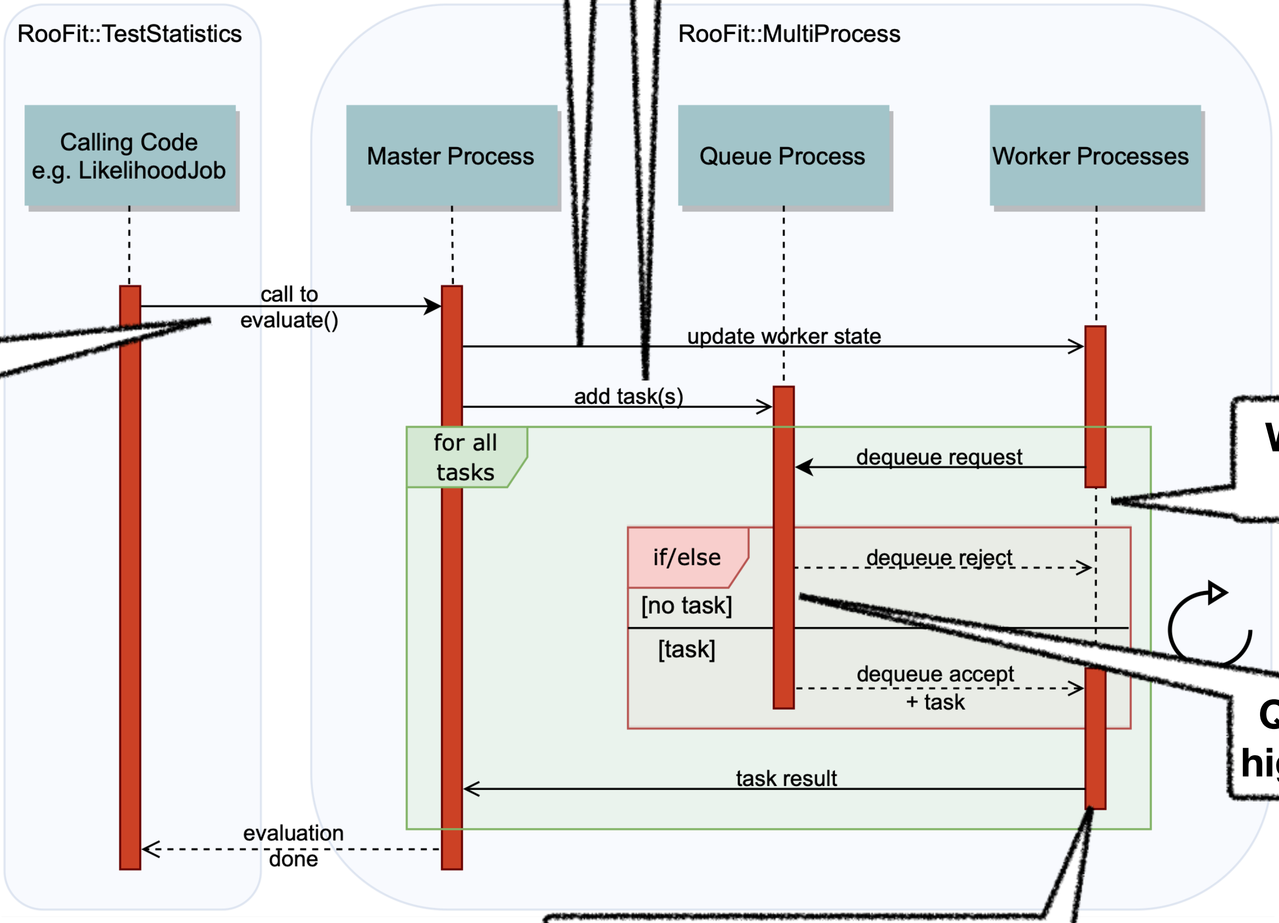
Master submits parallel tasks to queue

A parallel evaluation starts with some derived class of Job requesting parallel evaluation

Workers request tasks from queue in loop

Queue assigns task with highest priority if available

A General Parallel RooFit Framework



Master tells workers how to synchronise their state

Master submits parallel tasks to queue

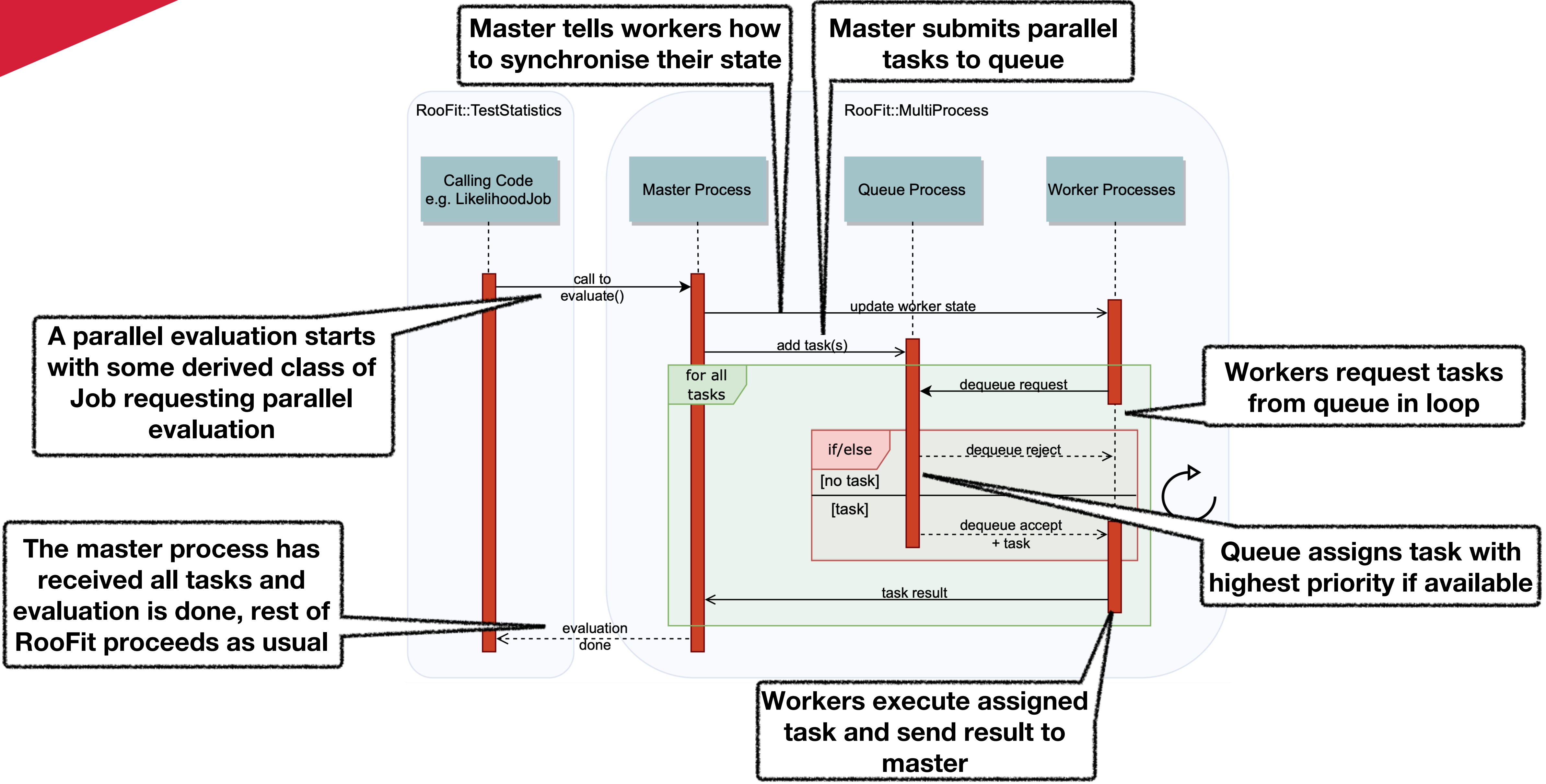
A parallel evaluation starts with some derived class of Job requesting parallel evaluation

Workers request tasks from queue in loop

Queue assigns task with highest priority if available

Workers execute assigned task and send result to master

A General Parallel RooFit Framework



Master tells workers how to synchronise their state
Master submits parallel tasks to queue
RooFit::TestStatistics
RooFit::MultiProcess
Calling Code e.g. LikelihoodJob
Master Process
Queue Process
Worker Processes
call to evaluate()
update worker state
add task(s)
for all tasks
dequeue request
if/else
dequeue reject
[no task]
[task]
dequeue accept + task
task result
evaluation done
A parallel evaluation starts with some derived class of Job requesting parallel evaluation
Workers request tasks from queue in loop
Queue assigns task with highest priority if available
The master process has received all tasks and evaluation is done, rest of RooFit proceeds as usual
Workers execute assigned task and send result to master