Installing GAMBIT

Go here to download: https://www.hepforge.org/downloads/gambit
Please get version 1.1.1

Multiple external packages are needed. See installation instructions!

Once those packages are installed, go to the GAMBIT directory and run the following (on Linux):

- First change line 26 of `Backends/include/gambit/Backends/default_bossed_versions.hpp` to `#define Default_pythia 8_212_EM`
- `mkdir build`
- `cd build`
- `cmake ..`
- `make scanners`
- `cmake ..`
- `make -jn gambit` (n is number of cores, takes a while)
- `make -jn backends` (also takes a while, if time is limited, just do `make superiso` and `make micromegas`)

Slightly more complicated for MacOS. See installation instructions!

Last time, we used GAMBIT as a tool to do Wilson coefficient fits…

Effective field theory

$$\mathcal{H}_{\text{eff}} = -\frac{4G_F}{\sqrt{2}} V_{tb} V_{ts}^* \sum_{i=1}^{10} \Big(C_i(\mu)\mathcal{O}_i(\mu) + C_i'(\mu)\mathcal{O}_i'(\mu) \Big)$$

Wilson Coefficients

$$\mathcal{O}_1 = (\bar{s}\gamma_\mu T^a P_L c)(\bar{c}\gamma^\mu T^a P_L b)$$

$$\mathcal{O}_2 = (\bar{s}\gamma_\mu P_L c)(\bar{c}\gamma^\mu P_L b)$$

$$\mathcal{O}_3 = (\bar{s}\gamma_\mu P_L b) \sum_q (\bar{q}\gamma^\mu q)$$

$$\mathcal{O}_4 = (\bar{s}\gamma_\mu T^a P_L b) \sum_q (\bar{q}\gamma^\mu T^a q)$$

$$\mathcal{O}_5 = (\bar{s}\gamma_{\mu_1}\gamma_{\mu_2}\gamma_{\mu_3} P_L b) \sum_q (\bar{q}\gamma^{\mu_1}\gamma^{\mu_2}\gamma^{\mu_3} q)$$

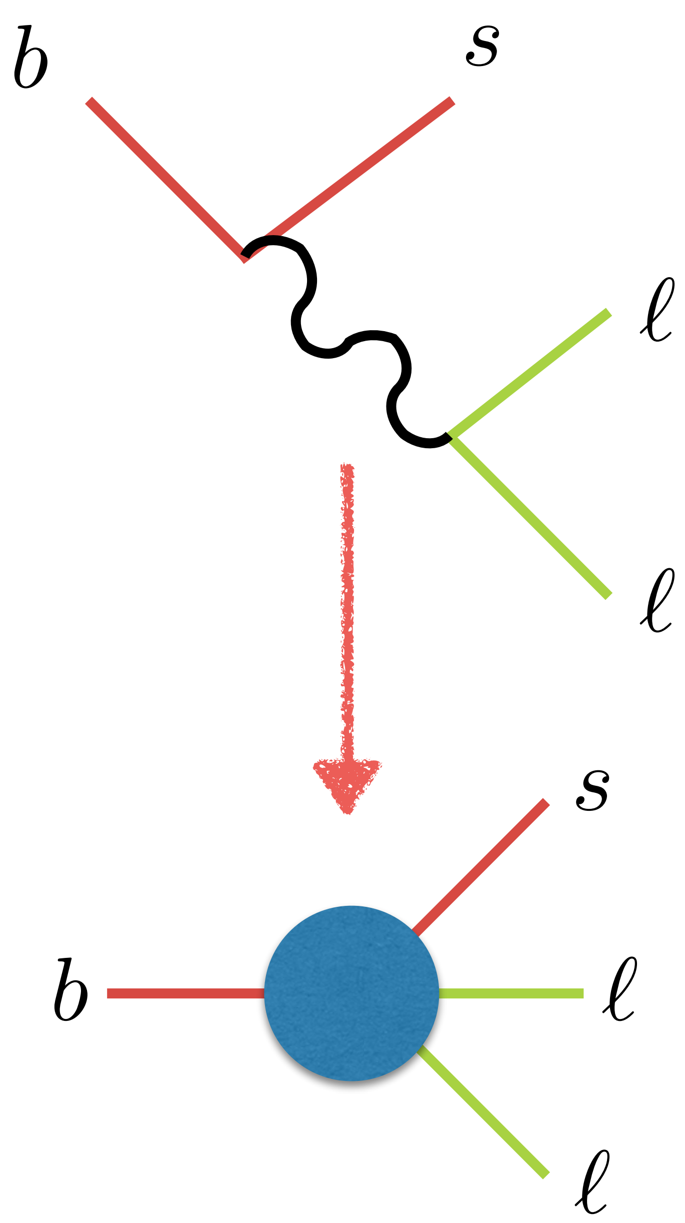
$$\mathcal{O}_6 = (\bar{s}\gamma_{\mu_1}\gamma_{\mu_2}\gamma_{\mu_3} T^a P_L b) \sum_q (\bar{q}\gamma^{\mu_1}\gamma^{\mu_2}\gamma^{\mu_3} T^a q)$$

$$\mathcal{O}_7 = \frac{e}{(4\pi)^2} m_b (\bar{s}\sigma^{\mu\nu} P_R b) F_{\mu\nu}$$

$$\mathcal{O}_8 = \frac{g}{(4\pi)^2} m_b (\bar{s}\sigma^{\mu\nu} T^a P_R b) G^a_{\mu\nu}$$

$$\mathcal{O}_9 = \frac{e^2}{(4\pi)^2} (\bar{s}\gamma^\mu P_L b)(\bar{\ell}\gamma_\mu \ell)$$

$$\mathcal{O}_{10} = \frac{e^2}{(4\pi)^2} (\bar{s}\gamma^\mu P_L b)(\bar{\ell}\gamma_\mu \gamma_5 \ell)$$



The least global global fit ever...

- 2D Wilson coefficient fit

$$\Delta C_x \equiv C_{x,BSM} - C_{x,SM}$$

- Free parameters: ΔC_7 `Re_DeltaC7`

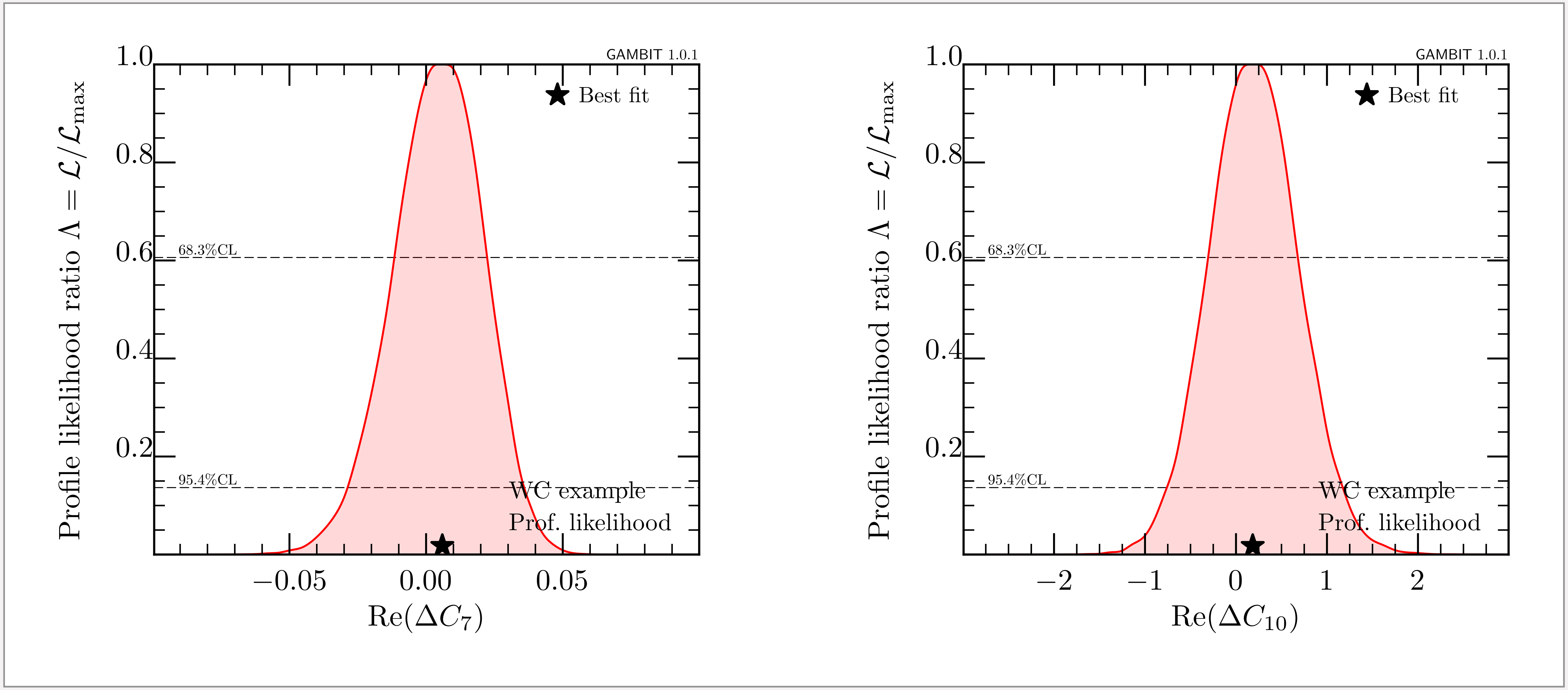
  ΔC_{10} `Re_DeltaC10`

- Observables: $BR(B \to X_s \gamma)$ `b2sgamma`

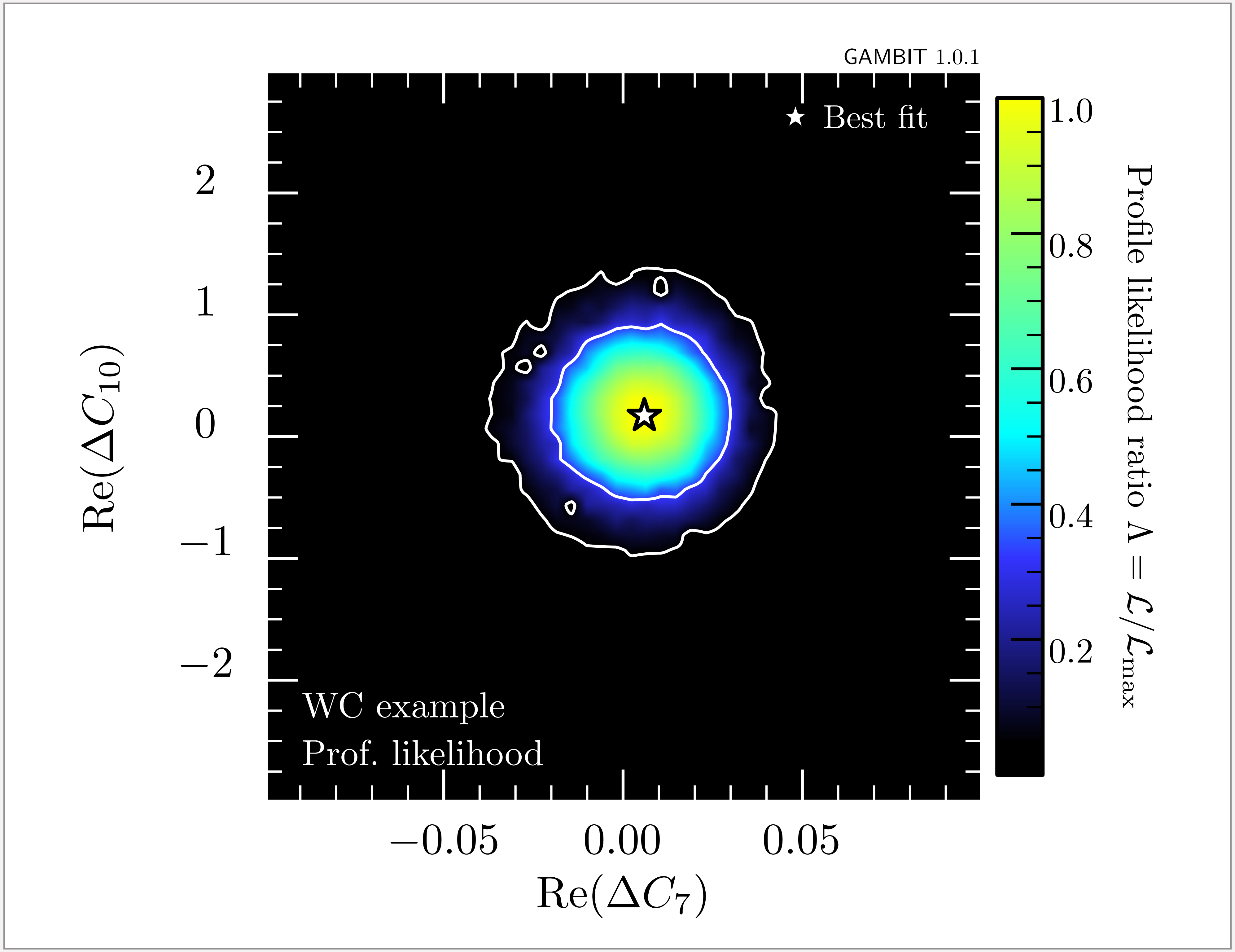
  $BR(B_d \to \mu^+ \mu^-)$

  $BR(B_s \to \mu^+ \mu^-)$ `b2ll`

Results — Diver scan

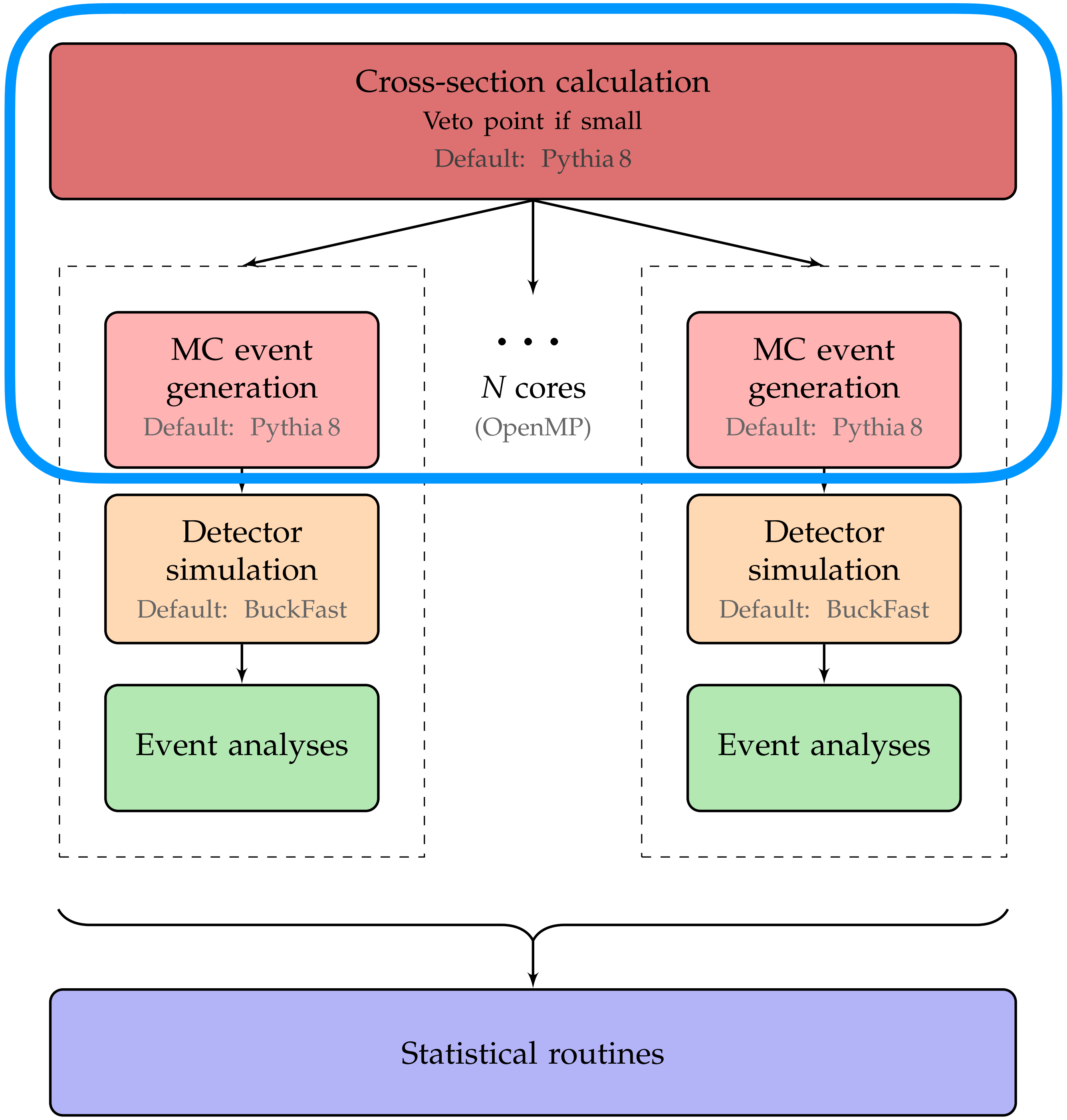


Results — Diver scan



GAMBIT is much more than flavour physics!

ColliderBit — LHC

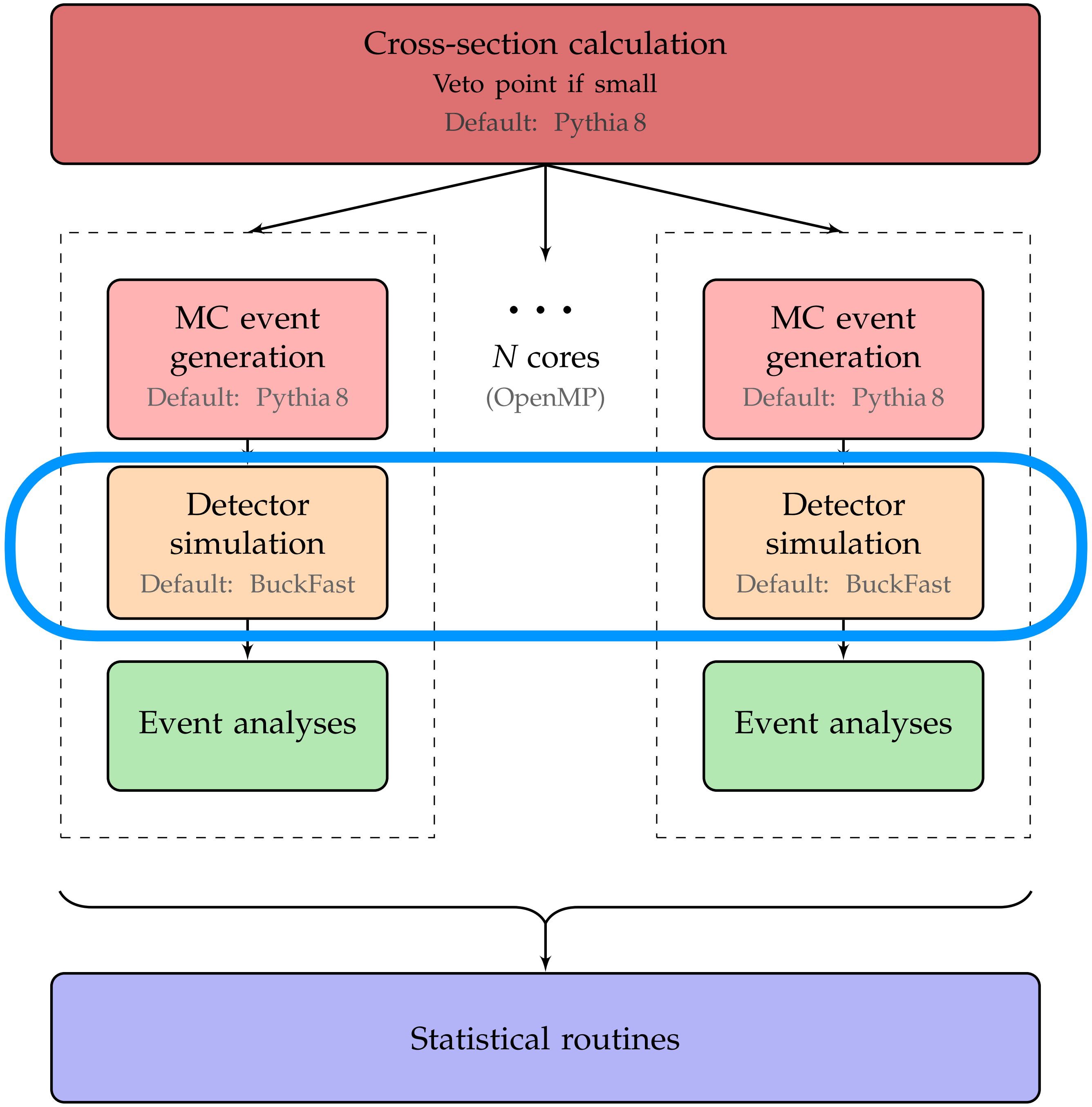


ColliderBit's recast chain is designed with a focus on speed:

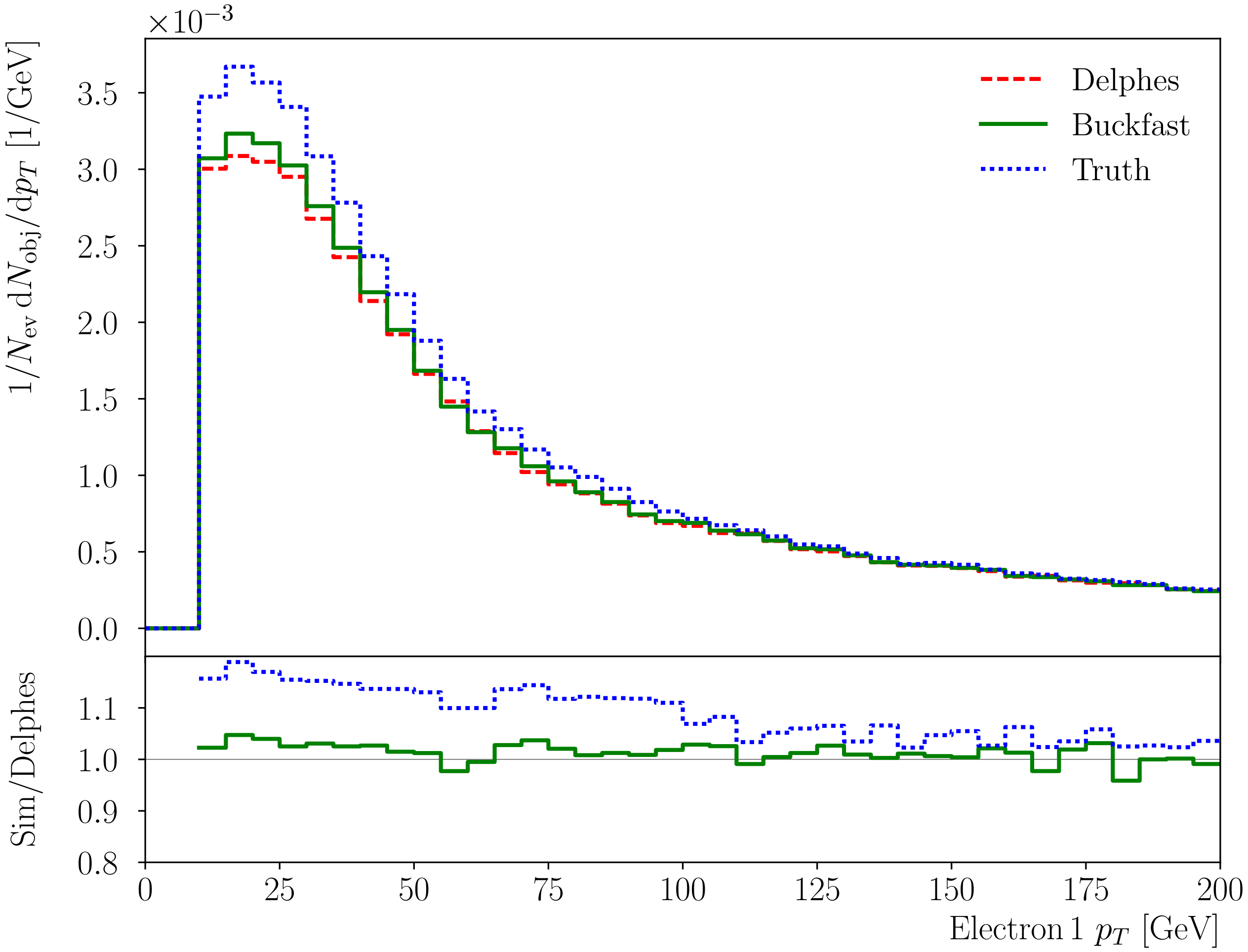
- Cross-sections are calculated with Pythia (interfaces to MadGraph/CalcHEP in development)
- If cross section is too low, point is vetoed
- Pythia has been parallelized and some options have been turned off

Configuration	t (10^5 events)	Speed-up
All	1,841 sec	1
$\hookrightarrow$ −MPI	671 sec	2.7
$\hookrightarrow$ −τ correlations	531 sec	3.5

ColliderBit – LHC



We have written a custom fast detector simulator, **BuckFast**, based on four-vector smearing, which we use by default. (Delphes is also available).



ColliderBit LHC Likelihoods

- We use a Poissonian likelihood marginalized over a rescaling parameter ξ to account for systematic uncertainties:

$$\mathcal{L}(n|s,b) = \int_0^\infty \frac{[\xi(s+b)]^n\, e^{-\xi(b+s)}}{n!} P(\xi)\mathrm{d}\xi$$

$$P(\xi|\sigma_\xi) \approx \frac{1}{\sqrt{2\pi}\sigma_\xi}\frac{1}{\xi}\exp\left[-\frac{1}{2}\left(\frac{\ln\xi}{\sigma_\xi}\right)^2\right]$$

where

$$\sigma_\xi^2 = \sigma_s^2 + \sigma_b^2$$

- *n*, *s* and *b* are for signal region *expected* to give the strongest limit

- Currently available analyses (all 8 TeV):
  - ATLAS SUSY searches (0 lepton*, 0-1-2 lepton stop, b jets + MET, 2 lepton EW, 3 lepton EW)
  - CMS DM searches (top pair + MET, mono-b, mono-jet)
  - CMS multilepton SUSY search

*13 TeV also available

ColliderBit LHC Likelihoods

- We use a Poissonian likelihood marginalized over a rescaling parameter ξ to account for systematic uncertainties:

$$\mathcal{L}(n|s,b) = \int_0^\infty \frac{[\xi(s+b)]^n \, e^{-\xi(b+s)}}{n!} P(\xi)\mathrm{d}\xi$$

$$P(\xi|\sigma_\xi) \approx \frac{1}{\sqrt{2\pi}\sigma_\xi}\frac{1}{\xi}\exp\left[-\frac{1}{2}\left(\frac{\ln\xi}{\sigma_\xi}\right)^2\right]$$

where

$$\sigma_\xi^2 = \sigma_s^2 + \sigma_b^2$$

- and *b* are for signal region *expected* to give the strongest limit
- ble analyses (all 8 TeV):
  - ATLAS ton*, 0-1-2 lepton stop, b jets + MET, 2 EW)
  - CMS DM searches (top pair jet)
  - CMS multilepton SUSY search

  *13 TeV also available

More 13 TeV searches currently being implemented

LEP and Higgs Likelihoods

LEP

- L3, ALEPH and OPAL limits on production cross sections of sleptons, neutralinos, and charginos are also available.

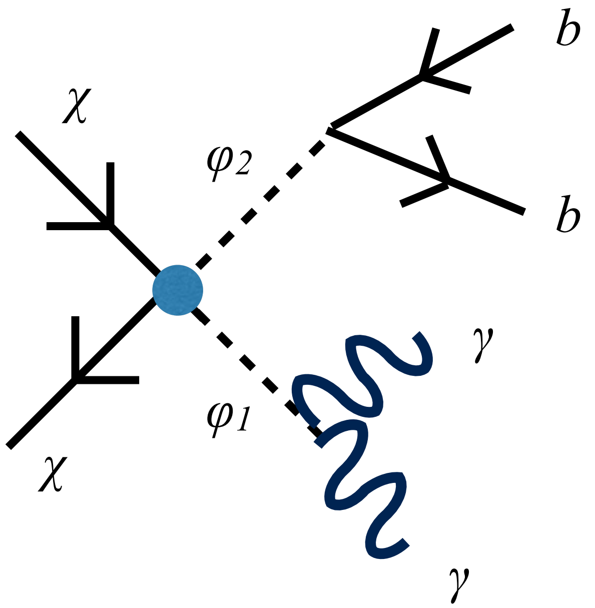
Higgs

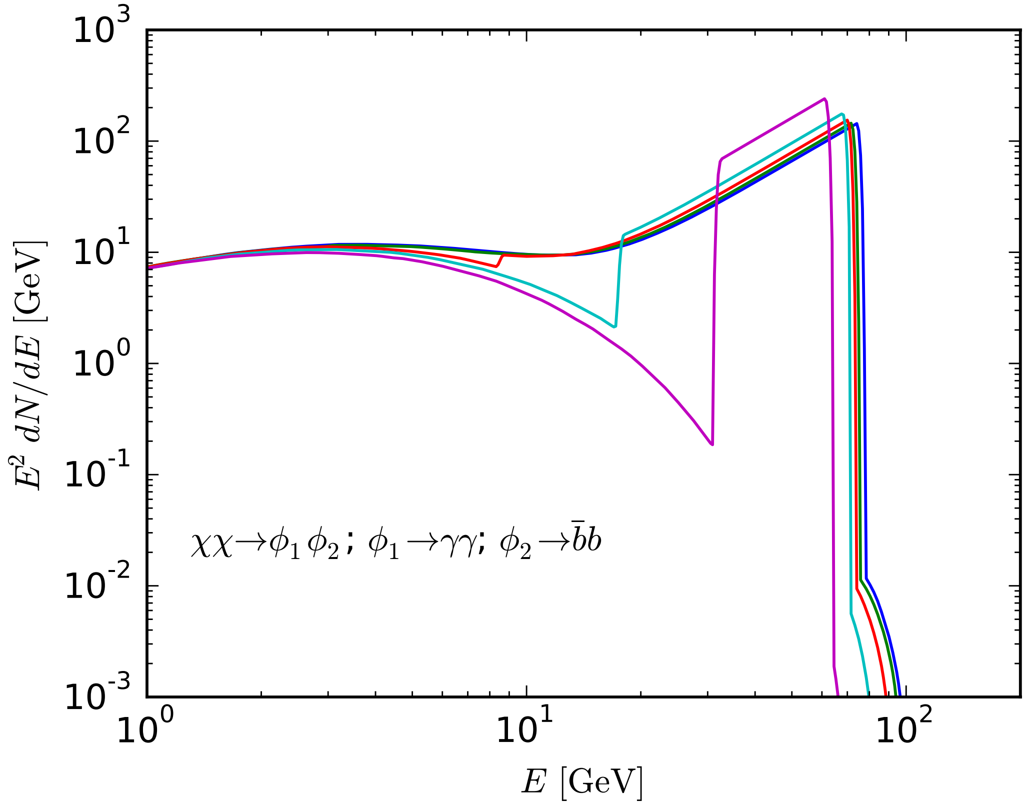
- Interfaces to HiggsBounds and HiggsSignals provide likelihoods from Higgs searches at LEP and measurements of Higgs properties at LHC.

DarkBit: Indirect Detection

Gamma rays:

- Theoretical spectra calculated using branching fractions and tabulated gamma-ray yields
- Non-SM final state particles and Higgs are decayed on the fly with cascade Monte Carlo
- gamLike (gamlike.hepforge.org): New standalone code with likelihoods for DM searches from Fermi-LAT (dwarf spheroidals, galactic centre) and H.E.S.S. (galactic halo)



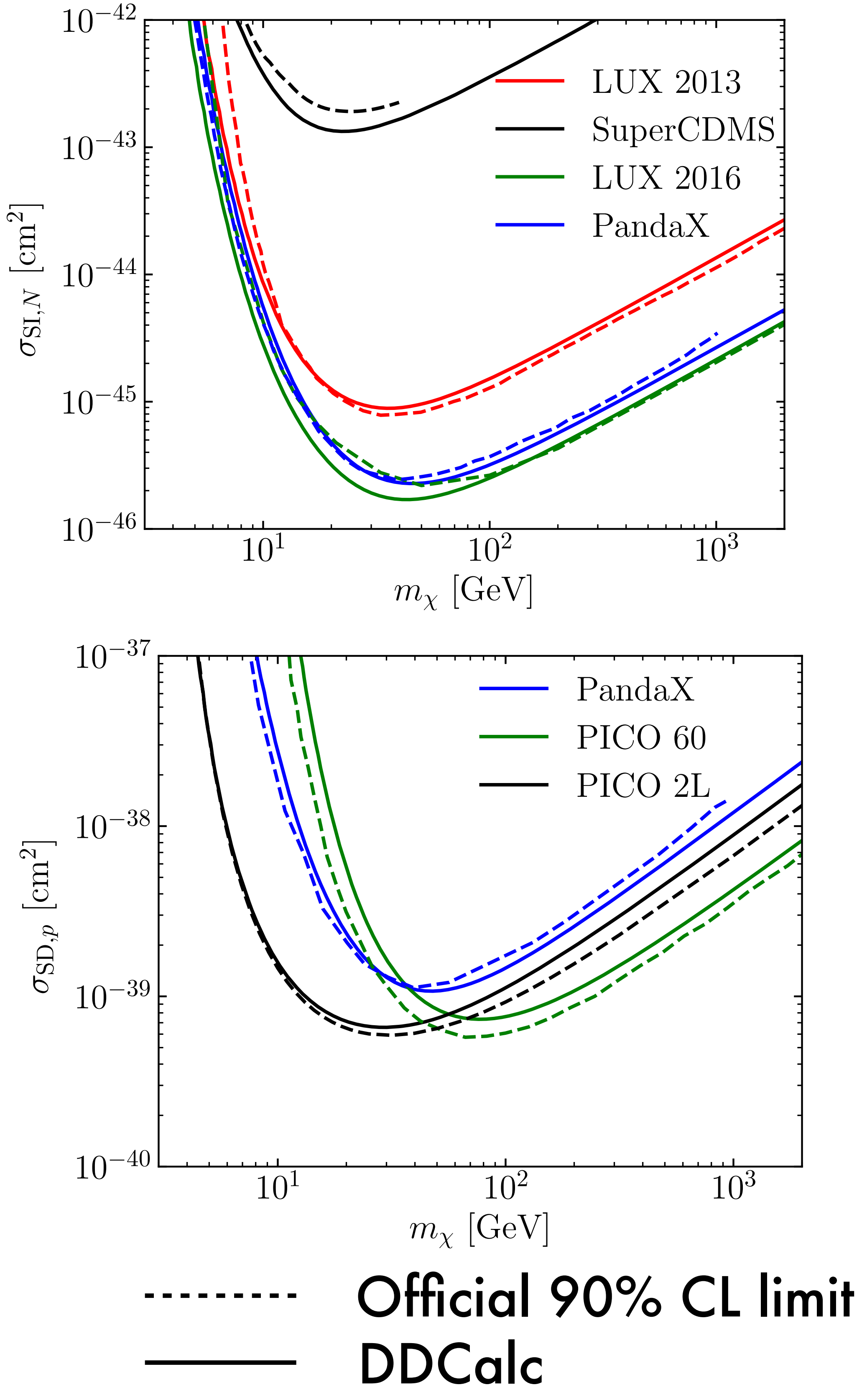


Solar neutrinos:

- Yields from DM annihilation in sun calculated by DarkSUSY. IceCube likelihoods contained in nulike (nulike.hepforge.org) standalone code.

DarkBit: Direct Detection

- In parallel with GAMBIT, we introduce *DDCalc* (ddcalc.hepforge.org), a tool to calculate event rates and complete likelihood functions for direct detection experiments taking into account:
  - A mix of both spin-independent and dependent contributions to the scattering rate.
  - Halo parameters (local density, DM velocity dispersion, etc.) chosen by the user.
- We currently have implemented likelihoods for Xenon(1T, 100), LUX, PandaX, SuperCDMS, PICO(60, 2L), and SIMPLE



The Constrained MSSM

- 5 *model* parameters, defined at GUT scale:
  - $m_0, m_{1/2}$ Unified scalar, gaugino mass parameters
  - A_0 Universal trilinear coupling
  - $\tan\beta$ $\text{sign}(\mu)$ Higgs sector parameters

- 5 *nuisance* parameters:
  - α_s, m_t SM parameters: strong coupling and top mass
  - ρ_0 Local DM density
  - σ_s, σ_l Hadronic matrix elements

- Many *likelihoods*:
  - Relic density (upper limit)
  - Fermi-LAT DM searches in dwarf spheroidals
  - Direct Detection (LUX, PandaX, etc..)
  - IceCube limits on DM scattering from solar neutrinos
  - Higgs invisible width
  - LHC run 1 SUSY searches (and 0 lepton run 2 search)
  - LHC Higgs constraints
  - Flavor physics limits from LHCb
  - *g*-2 of the muon

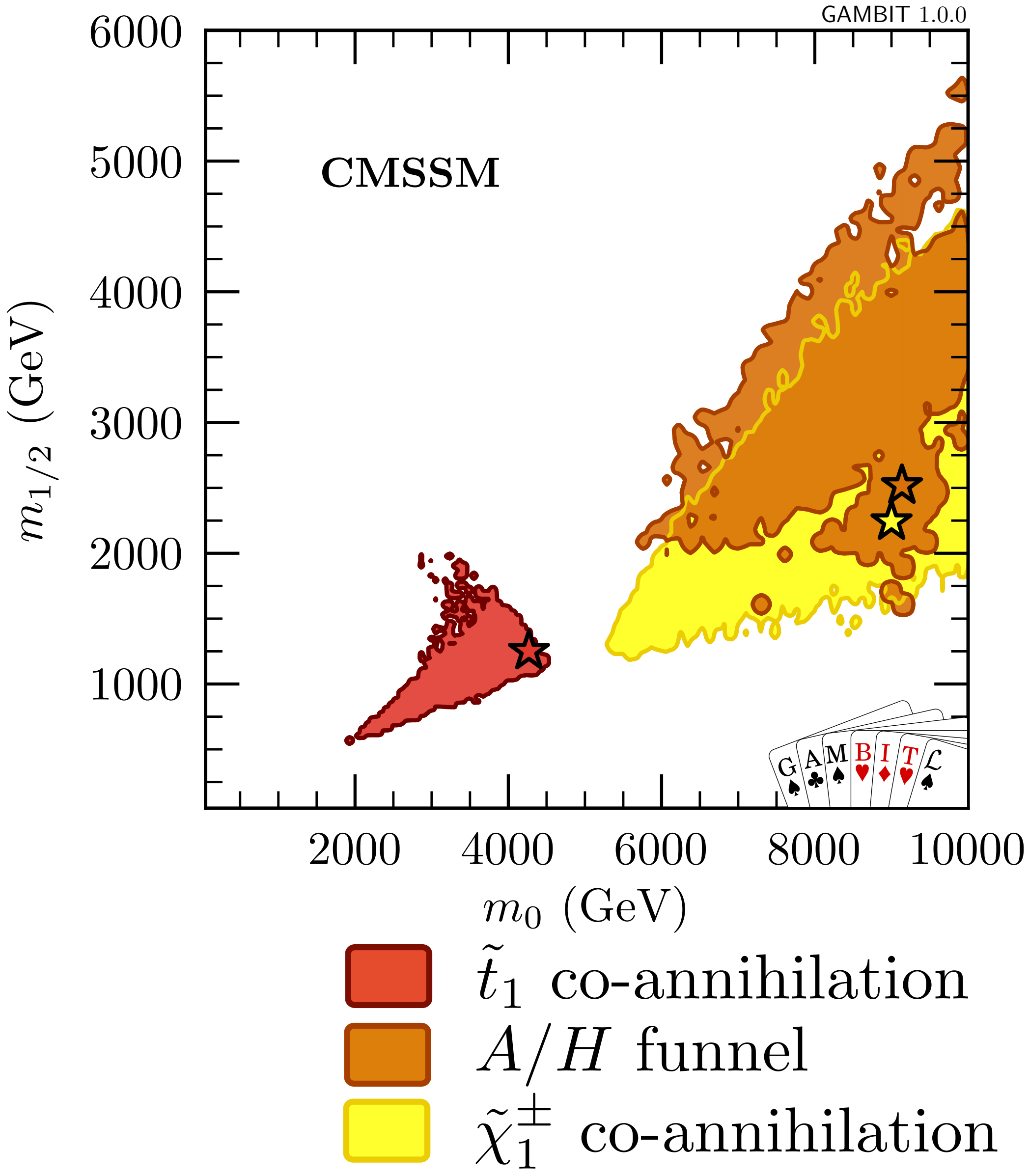
The Constrained MSSM

- The YAML file that runs this scan is `yaml_files/CMSSM.yaml`. Please open it.
- The scan defined in that YAML does not include the H.E.S.S. galactic center likelihoods or the ATLAS 13 TeV 0 lepton analysis. Add them! (Look in the other example YAML files in `yaml_files` for hints.)
- Set `debug` to `true` in the `KeyValues` section of the YAML file and `silenceLoop` in the options of the `operateLHCLoop` function to `false`.
- Try to start a CMSSM scan with `./gambit -f yaml_files/CMSSM.yaml`.

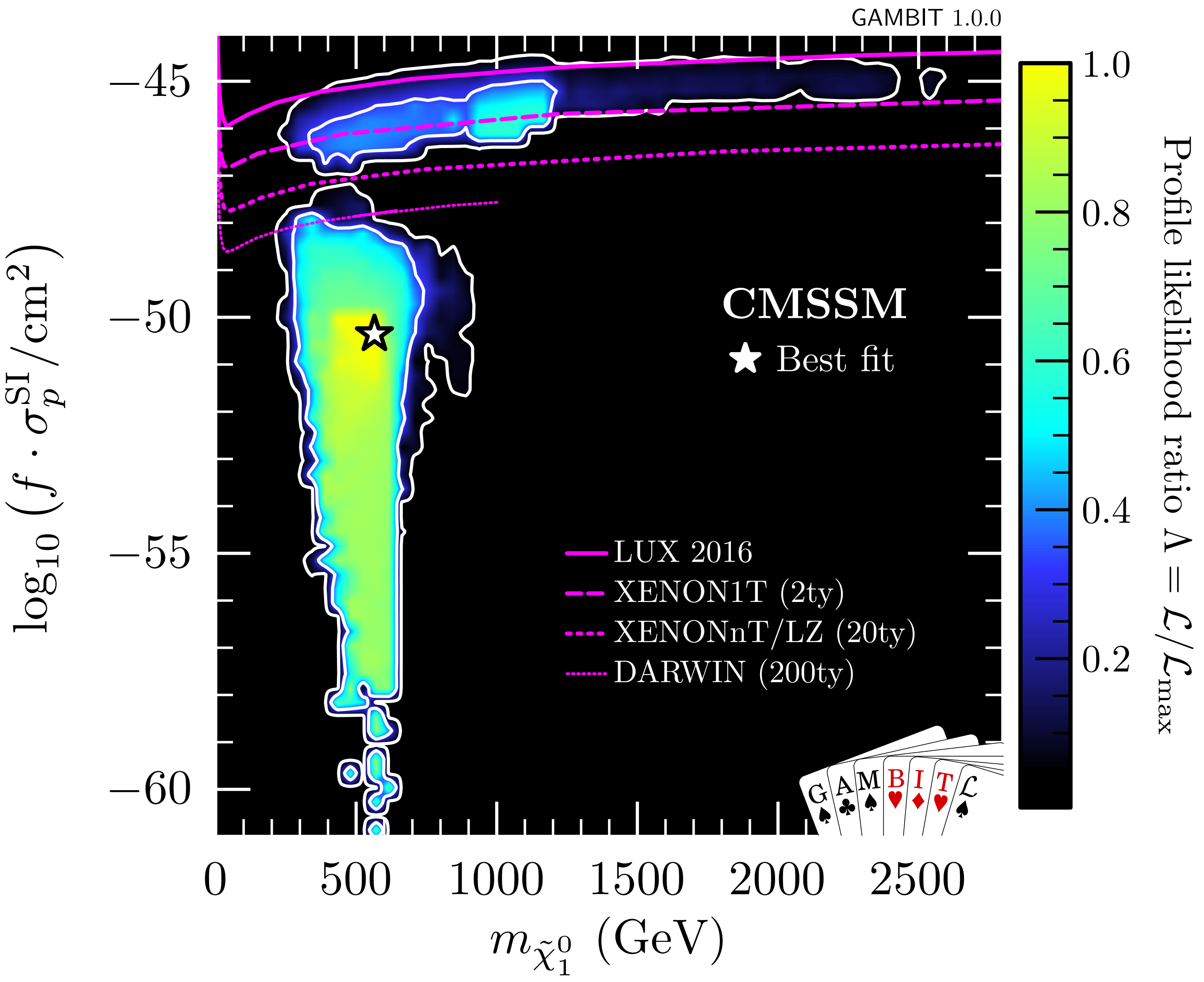
The Constrained MSSM

If you have a large enough cluster and enough computer time, you can make plots like these:

95% CL preferred regions of parameter space



How can future DD searches probe the viable regions?



arXiv:1705.07935

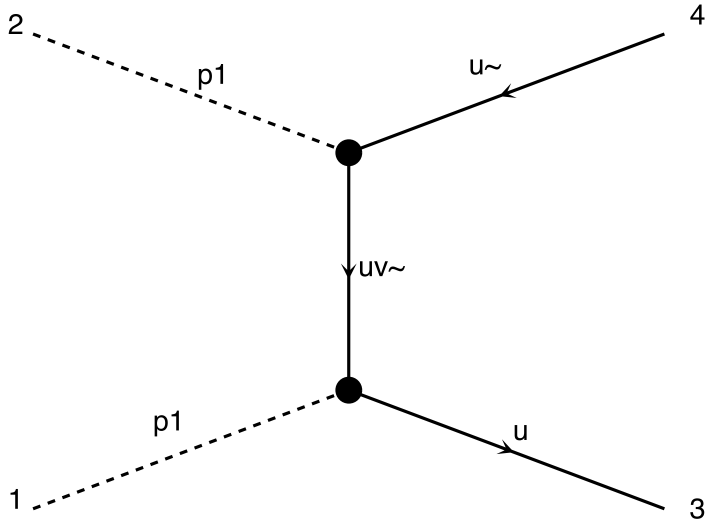
Adding a New Model into GAMBIT (and calculating DM likelihoods for it)

The Model

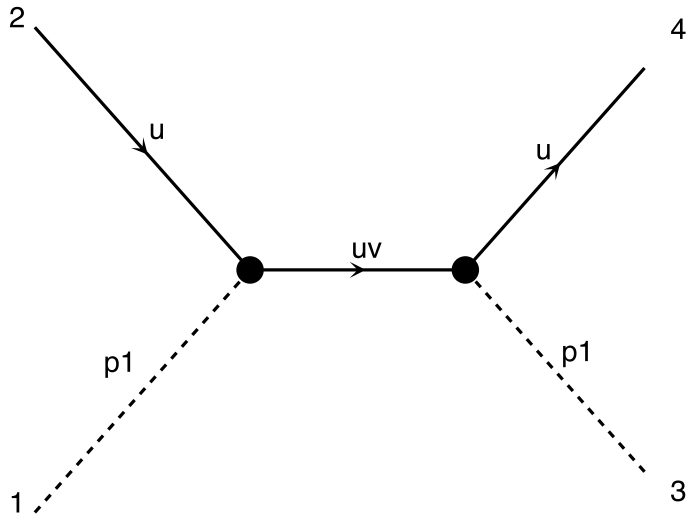
An example with two new particles: $\mathcal{L}_{\rm int} = \lambda_1 \phi_1 \bar{U} u_R + h.c.$

- ϕ_1, real singlet scalar, mass m_1, DM candidate
- U, colored fermion, SU(2) singlet, mass M_U

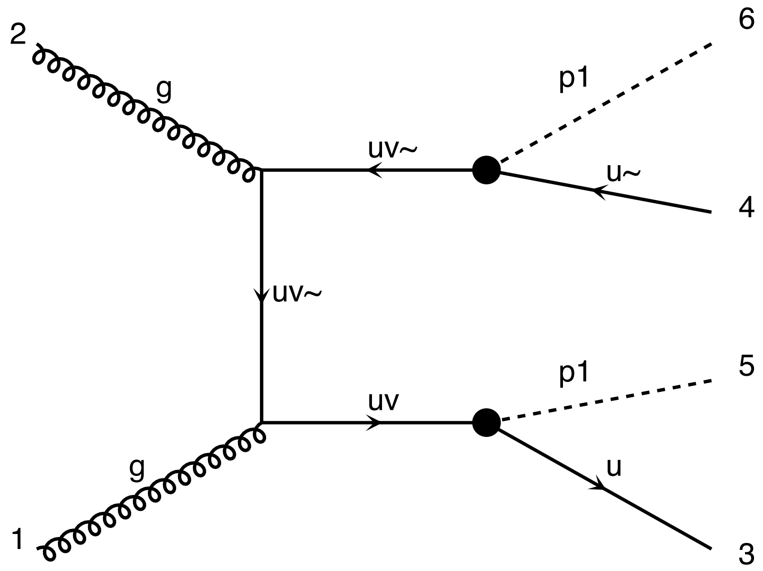
DM annihilation: relic density, indirect detection



DM-quark scattering: direct detection



U production at LHC

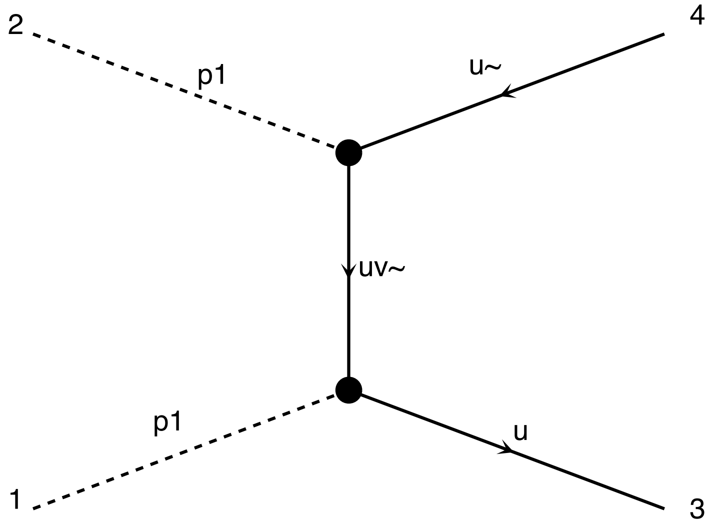


The Model

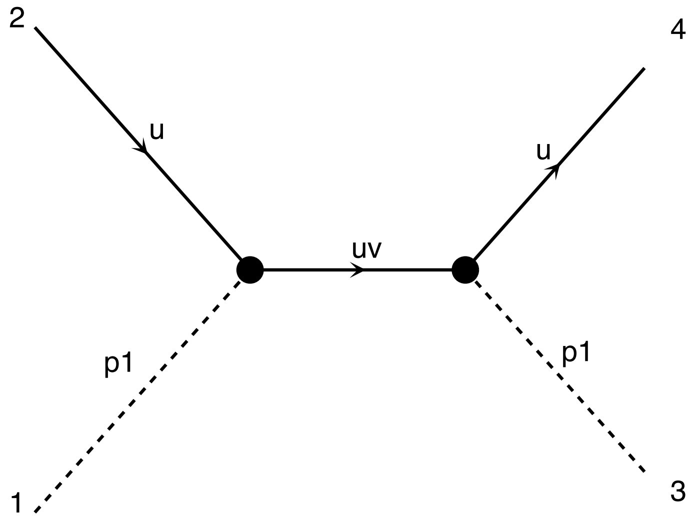
An example with two new particles: $\mathcal{L}_{\text{int}} = \lambda_1 \phi_1 \bar{U} u_R + h.c.$

- ϕ_1, real singlet scalar, mass m_1, DM candidate
- U, colored fermion, SU(2) singlet, mass M_U

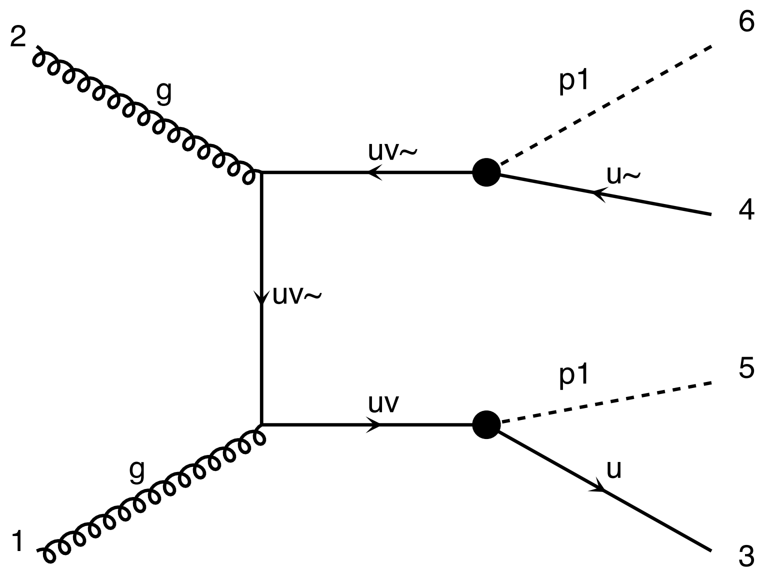
DM annihilation: relic density, indirect detection



DM-quark scattering: direct detection



U production at LHC



In the remaining part of this tutorial, we will complete a simple implementation of the model into GAMBIT, and find the best fit regions of parameter space in light of the these constraints.

Expansion: adding new models

1. Add the model to the **model hierarchy**:
   - Choose a model name, and declare any **parent model**
   - Declare the model's parameters
   - Declare any **translation function** to the parent model

```
#define MODEL NUHM1
#define PARENT NUHM2
  START_MODEL
  DEFINEPARS(M0,M12,mH,A0,TanBeta,SignMu)
  INTERPRET_AS_PARENT_FUNCTION(NUHM1_to_NUHM2)
#undef PARENT
#undef MODEL
```

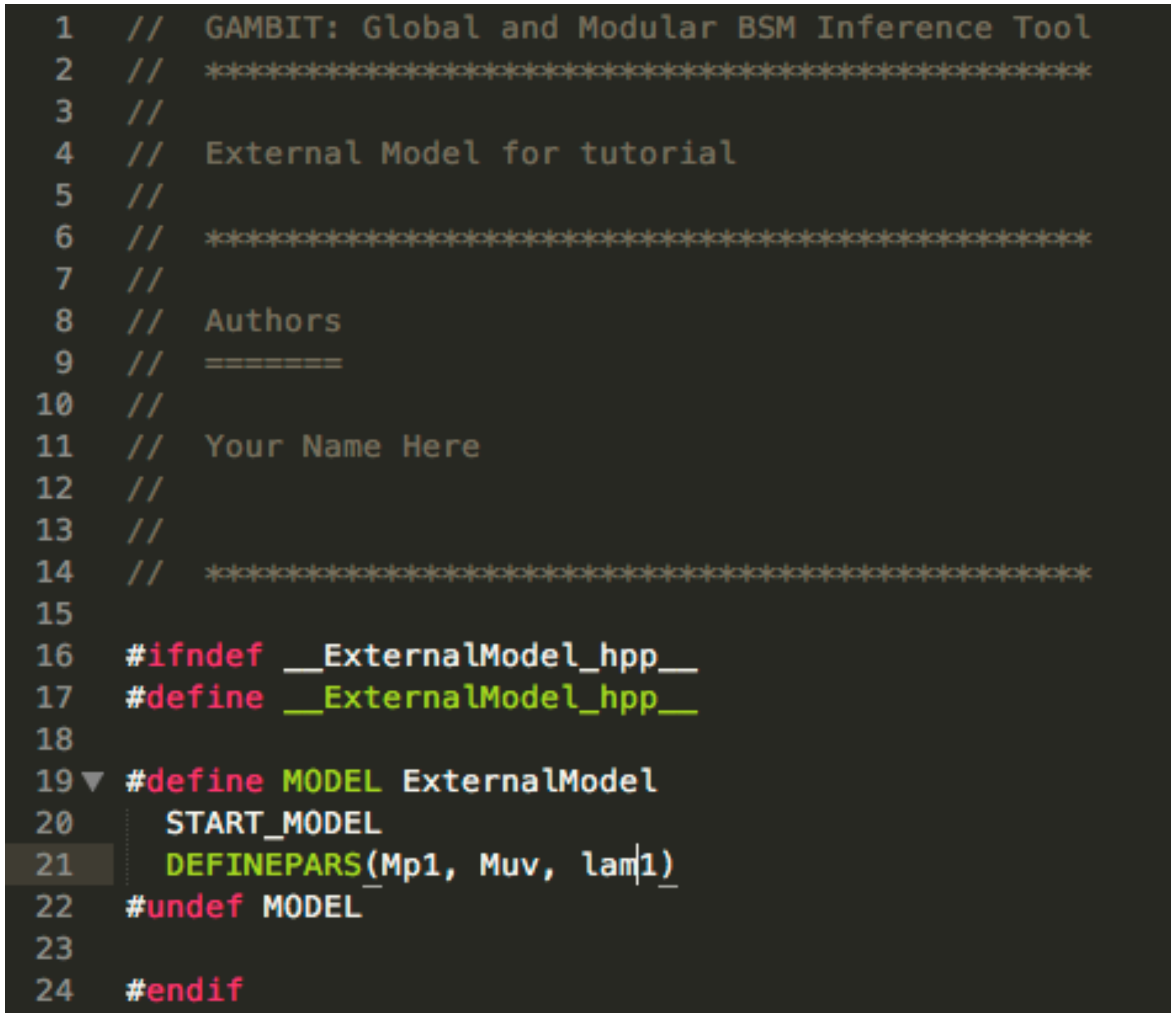
2. Write the translation function as a standard C++ function:

```
void MODEL_NAMESPACE::NUHM1_to_NUHM2 (const ModelParameters &myP, ModelParameters &targetP)
{
   // Set M0, M12, A0, TanBeta and SignMu in the NUHM2 to the same values as in the NUHM1
   targetP.setValues(myP,false);
   // Set the values of mHu and mHd in the NUHM2 to the value of mH in the NUHM1
   targetP.setValue("mHu", myP["mH"]);
   targetP.setValue("mHd", myP["mH"]);
}
```

3. If needed, declare that existing module functions work with the new model, or add new functions that do.

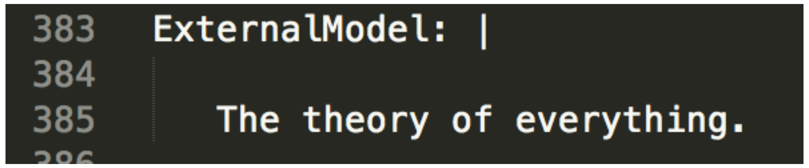
Step 1 - Add a new model file

Add the file `ExternalModel.hpp` to the `Models/include/gambit/Models/models` directory.



```
//   GAMBIT: Global and Modular BSM Inference Tool
//   *********************************************
//
//   External Model for tutorial
//
//   *********************************************
//
//   Authors
//   =======
//
//   Your Name Here
//
//
//   *********************************************

#ifndef __ExternalModel_hpp__
#define __ExternalModel_hpp__

#define MODEL ExternalModel
  START_MODEL
  DEFINEPARS(Mp1, Muv, lam1)
#undef MODEL

#endif
```

You also need to add a description of the model to the GAMBIT diagnostic system, otherwise GAMBIT will judge you and refuse to run. Add something like this to `config/models.dat`:



```
ExternalModel: |

  The theory of everything.
```

Step 2 - Implement the model into micrOMEGAs

- Go to the directory `Backends/installed/micromegas/3.6.9.2/`

- Run `./newProject ExternalModel` to make a micrOMEGAs directory for the new model.

- Enter the `./ExternalModel` directory. Copy all of the provided CalcHEP files for this model into the `./work/models` directory.

- Copy the provided `Makefile` into the `ExternalModel` directory.

- Run `make sharedlib main=main.c`. This should create a library `libmicromegas.so`, which GAMBIT will use to run micrOMEGAs functions.

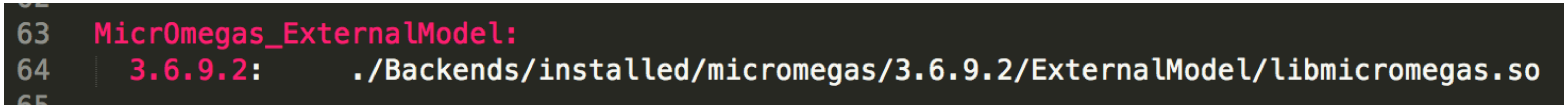
Step 3 – Implement a new micrOMEGAs frontend into GAMBIT

- Open the provided files `MicrOmegas_ExternalModel_3_6_9_2.*`. These files make GAMBIT aware of various functions and variables in micrOMEGAs, as well as providing an initialization function for the backend.

- Copy `MicrOmegas_ExternalModel_3_6_9_2.hpp` to `Backends/include/gambit/Backends/frontends` and `MicrOmegas_ExternalModel_3_6_9_2.cpp` to `Backends/src/frontends`

Step 4 – Tell GAMBIT where to find the new micrOMEGAs library

Copy `config/backend_locations.yaml.default` to `config/backend_locations.yaml`. Add the following lines to the file:



```
MicrOmegas_ExternalModel:
  3.6.9.2:     ./Backends/installed/micromegas/3.6.9.2/ExternalModel/libmicromegas.so
```

Expansion: adding new observables and likelihoods

Adding a new module function is easy:

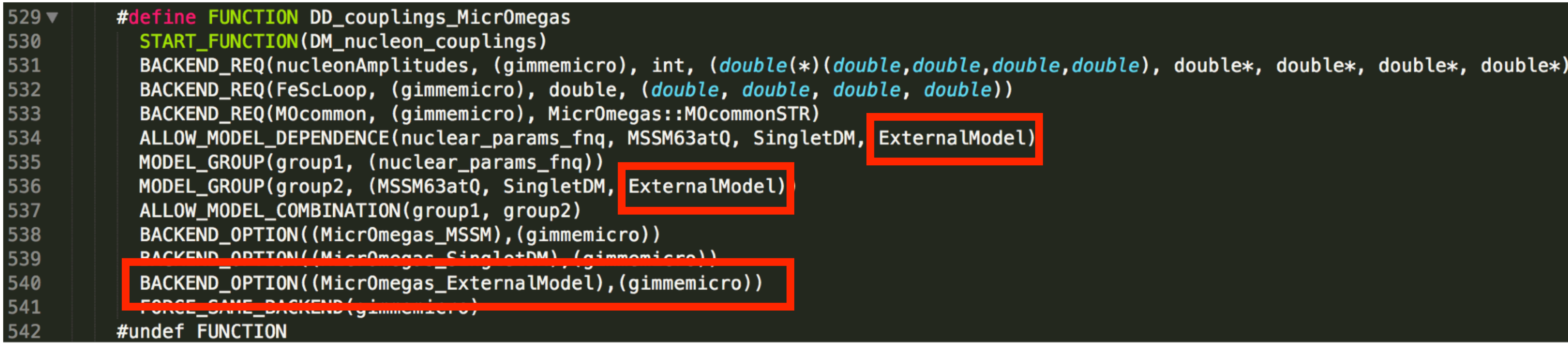
1. Declare the function to GAMBIT in a module's **rollcall header**
   - Choose a capability
   - Declare any **backend requirements**
   - Declare any **dependencies**
   - Declare any specific **allowed models**
   - other more advanced declarations also available

```
#define MODULE FlavBit                           // A tasty GAMBIT module.
START_MODULE

  #define CAPABILITY Rmu                         // Observable: BR(K->mu nu)/BR(pi->mu nu)
  START_CAPABILITY
    #define FUNCTION SI_Rmu                      // Name of a function that can compute Rmu
    START_FUNCTION(double)                       // Function computes a double precision result
    BACKEND_REQ(Kmunu_pimunu, (my_tag), double, (const parameters*)) // Needs function from a backend
    BACKEND_OPTION( (SuperIso, 3.6), (my_tag) )                      // Backend must be SuperIso 3.6
    DEPENDENCY(SuperIso_modelinfo, parameters)   // Needs another function to calculate SuperIso info
    ALLOW_MODELS(MSSM63atQ, MSSM63atMGUT)        // Works with weak/GUT-scale MSSM and descendents
    #undef FUNCTION
  #undef CAPABILITY
```

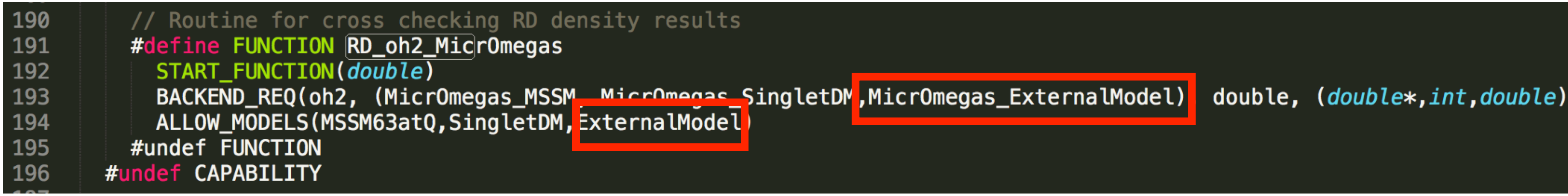
2. Write the function as a standard C++ function (one argument: the result)

Step 5 – Make existing DarkBit functions aware of the new model

- Our implementation of the this model into micrOMEGAs allows us to calculate relic density and direct detection likelihoods for it using DarkBit. To make the existing DarkBit functions aware of this new model and backend, make the following changes to `DarkBit/include/gambit/DarkBit/DarkBit_rollcall.hpp`:



```
    #define FUNCTION DD_couplings_MicrOmegas
      START_FUNCTION(DM_nucleon_couplings)
      BACKEND_REQ(nucleonAmplitudes, (gimmemicro), int, (double(*)(double,double,double,double), double*, double*, double*, double*)
      BACKEND_REQ(FeScLoop, (gimmemicro), double, (double, double, double, double))
      BACKEND_REQ(MOcommon, (gimmemicro), MicrOmegas::MOcommonSTR)
      ALLOW_MODEL_DEPENDENCE(nuclear_params_fnq, MSSM63atQ, SingletDM, ExternalModel)
      MODEL_GROUP(group1, (nuclear_params_fnq))
      MODEL_GROUP(group2, (MSSM63atQ, SingletDM, ExternalModel))
      ALLOW_MODEL_COMBINATION(group1, group2)
      BACKEND_OPTION((MicrOmegas_MSSM),(gimmemicro))
      BACKEND_OPTION((MicrOmegas_SingletDM),(gimmemicro))
      BACKEND_OPTION((MicrOmegas_ExternalModel),(gimmemicro))
      FORCE_SAME_BACKEND(gimmemicro)
    #undef FUNCTION
```

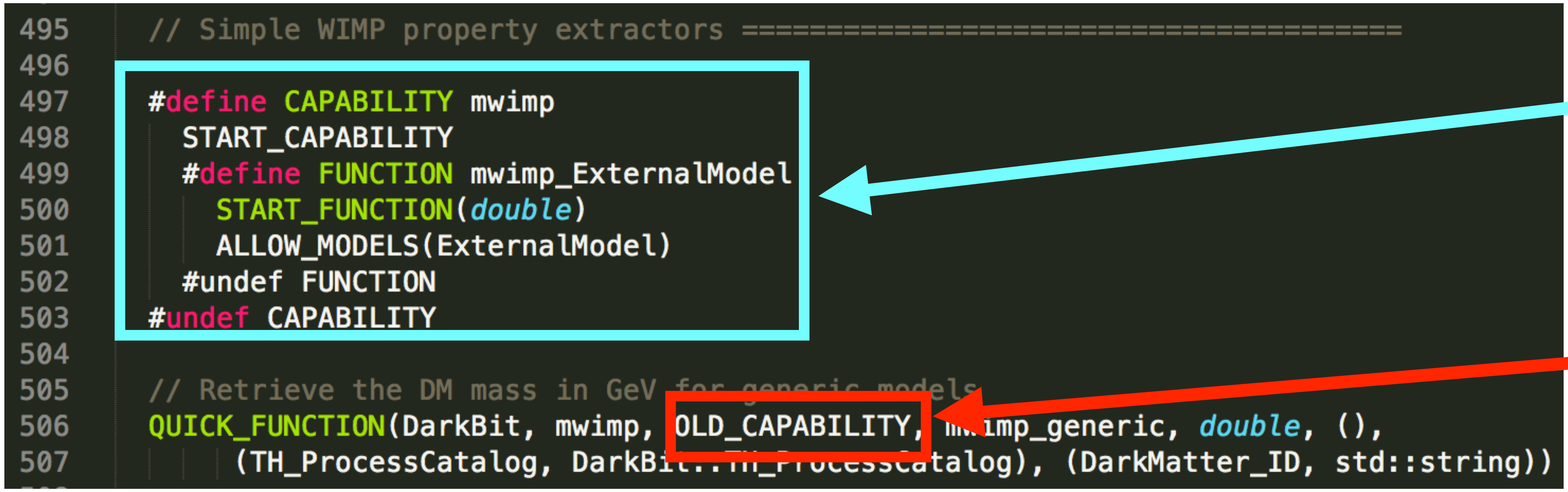


```
    // Routine for cross checking RD density results
    #define FUNCTION RD_oh2_MicrOmegas
      START_FUNCTION(double)
      BACKEND_REQ(oh2, (MicrOmegas_MSSM, MicrOmegas_SingletDM,MicrOmegas_ExternalModel)  double, (double*,int,double)
      ALLOW_MODELS(MSSM63atQ,SingletDM,ExternalModel)
    #undef FUNCTION
  #undef CAPABILITY
```

Step 6 – Add a function to satisfy the *mwimp* capability

The direct detection functions have a dependency on the *mwimp* capability. Add a new function to satisfy this dependency for the new model:

- First register the function in `DarkBit/include/gambit/DarkBit/DarkBit_rollcall.hpp`:

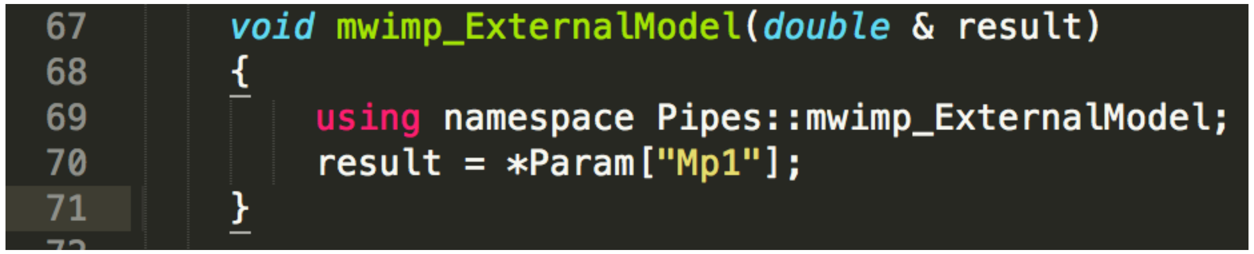


```
  // Simple WIMP property extractors ========================================

  #define CAPABILITY mwimp
    START_CAPABILITY
    #define FUNCTION mwimp_ExternalModel
      START_FUNCTION(double)
      ALLOW_MODELS(ExternalModel)
    #undef FUNCTION
  #undef CAPABILITY

  // Retrieve the DM mass in GeV for generic models
  QUICK_FUNCTION(DarkBit, mwimp, OLD_CAPABILITY, mwimp_generic, double, (),
       (TH_ProcessCatalog, DarkBit::TH_ProcessCatalog), (DarkMatter_ID, std::string))
```

The new function.

Be sure to change this from what is there now!

- Then add a new function to `DarkBit/src/DarkBit.cpp`:



```
    void mwimp_ExternalModel(double & result)
    {
        using namespace Pipes::mwimp_ExternalModel;
        result = *Param["Mp1"];
    }
```

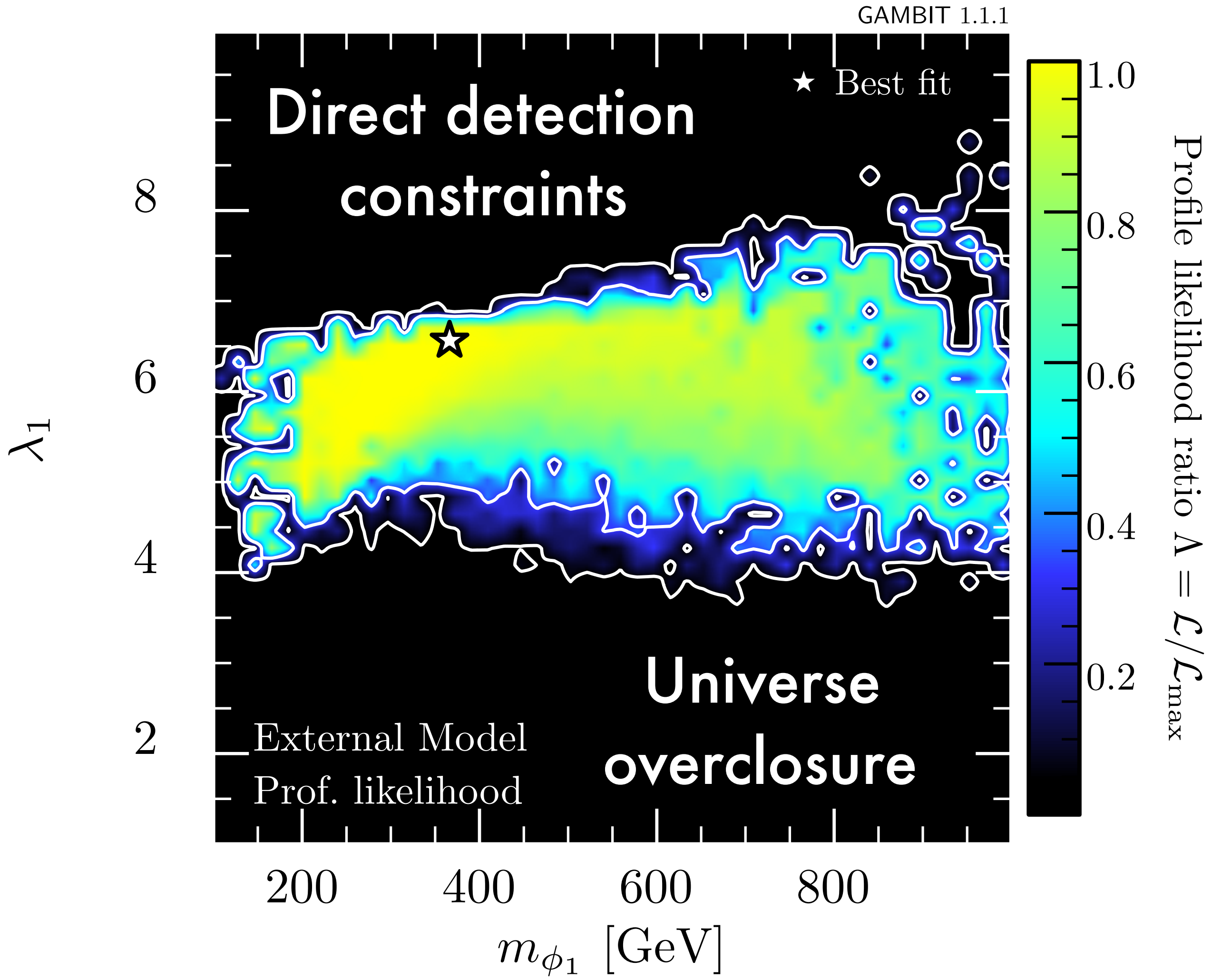
This function just returns the dark matter mass from the model parameters.

Step 7 – Recompile and Scan

- Run `make clean` and then `make -jn gambit` in the build directory to incorporate all of your changes. (This takes a long time, sorry :( )
- Run `./gambit backends` and `./gambit models`. You should see the backend and model you added there. Run `./gambit ExternalModel` to learn more about our new model.
- Use the `ExternalModel_DM.yaml` file to run a scan of the new model! You can stop GAMBIT from complaining about missing descriptions of new capabilities with the `--developer` flag.

Scan results

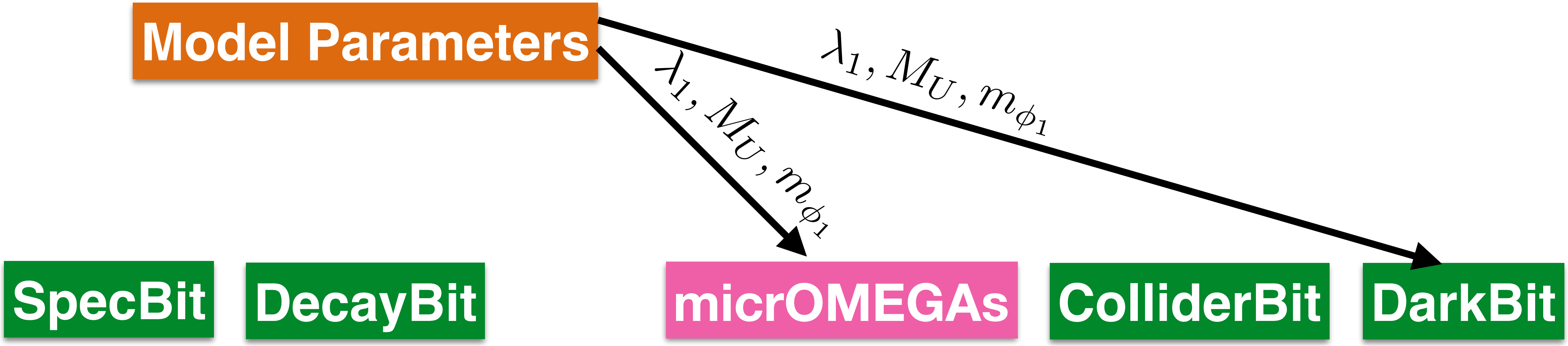
Allowing M_U to be greater than 400 GeV, I find this best fit region of the parameter space when scanning over M_U, m_1, and λ_1



Can you reproduce this? How does the plot change if M_U is restricted to higher masses?

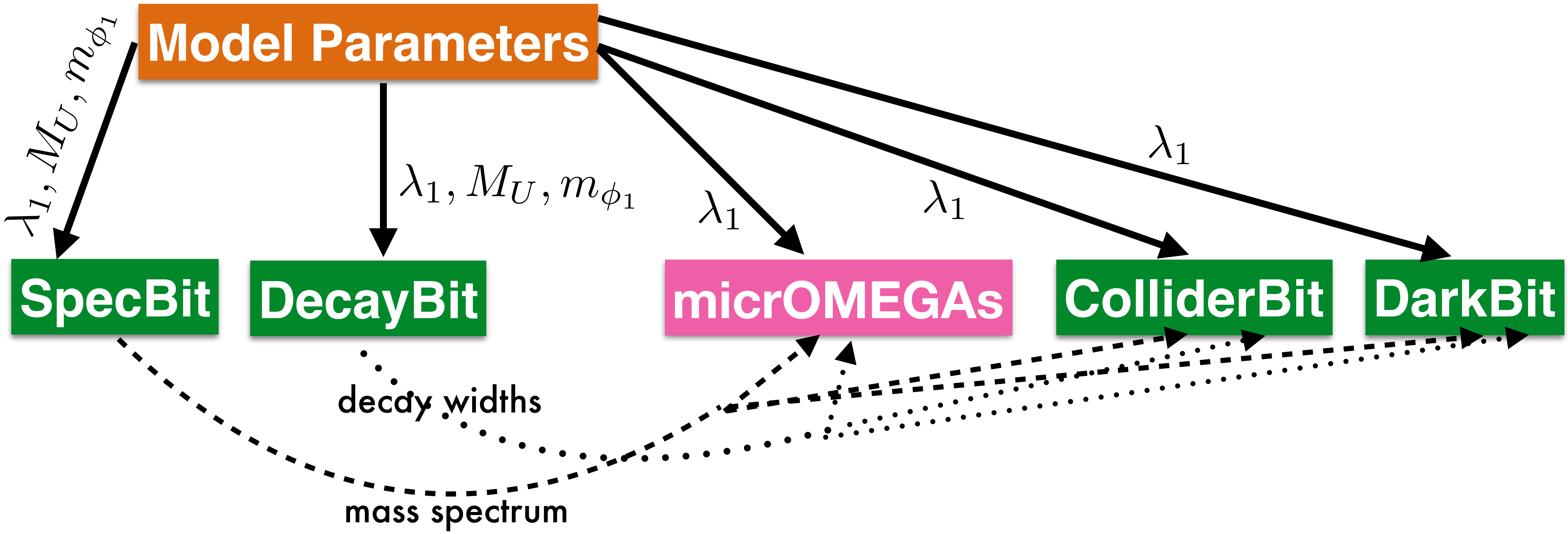
One Last Caveat

In this quick and dirty example, we bypassed SpecBit and DecayBit and passed model parameters directly to our backend:



One Last Caveat

Many of the observable calculations in GAMBIT require spectrum objects from SpecBit and decay tables from DecayBit, so the more canonical implementation would take this form:



For more details, see the SpecBit paper.