Ben Farmer
(on behalf of the
GAMBIT collaboration)

Overview

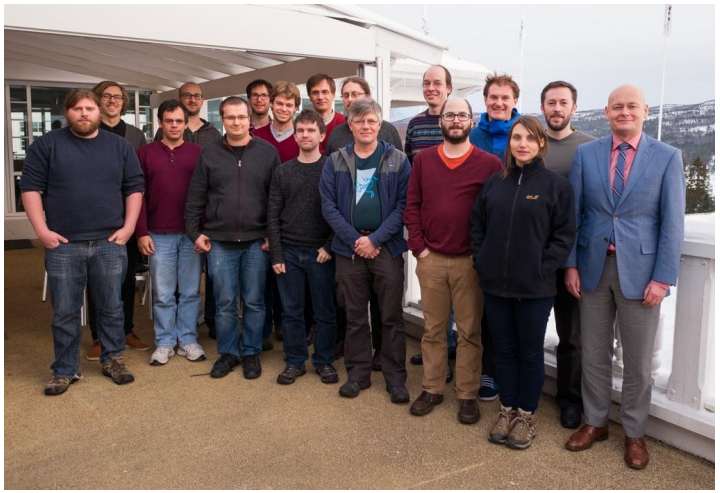
- **The GAMBIT project**
- **Why global fits?**
- **GAMBIT overview**
  - Design philosophy
  - Model hierarchy
  - Backend system
  - Generic container objects
  - Generic scanner interface (ScannerBit)
  - (For recasting LHC limits, see Martin White's talk)
  - Comparing tools and preliminary results
- **Summary**

GAMBIT: The Global And Modular BSM Inference Tool

gambit.hepforge.org

- Fast definition of new datasets and theoretical models
- Plug and play scanning, physics and likelihood packages
- Extensive model database – not just SUSY
- Extensive observable/data libraries
- Many statistical and scanning options (Bayesian & frequentist)
- *Fast* LHC likelihood calculator
- Massively parallel
- Fully open-source

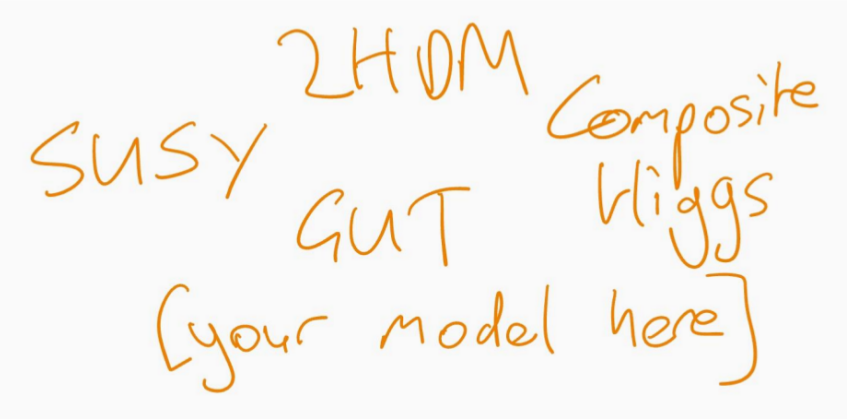
ATLAS	A. Buckley, P. Jackson, C. Rogan, M. White,
LHCb	M. Chrząszcz, N. Serra
Belle-II	F. Bernlochner, P. Jackson
Fermi-LAT	J. Conrad, J. Edsjö, G. Martinez, P. Scott
CTA	C. Balázs, T. Bringmann, J. Conrad, M. White
HESS	J. Conrad
IceCube	J. Edsjö, P. Scott
XENON/DARWIN	J. Conrad, R. Trotta
Theory	P. Athron, C. Balázs, T. Bringmann, J. Cornell, J. Edsjö, B. Farmer, T. Gonzalo, S. Hoof, F. Kahlhoefer, A. Krislock, A. Kvellestad, M. Pato, F. Mahmoudi, J. McKay, A. Raklev, R. Ruiz, P. Scott, R. Trotta, C. Weniger, M. White



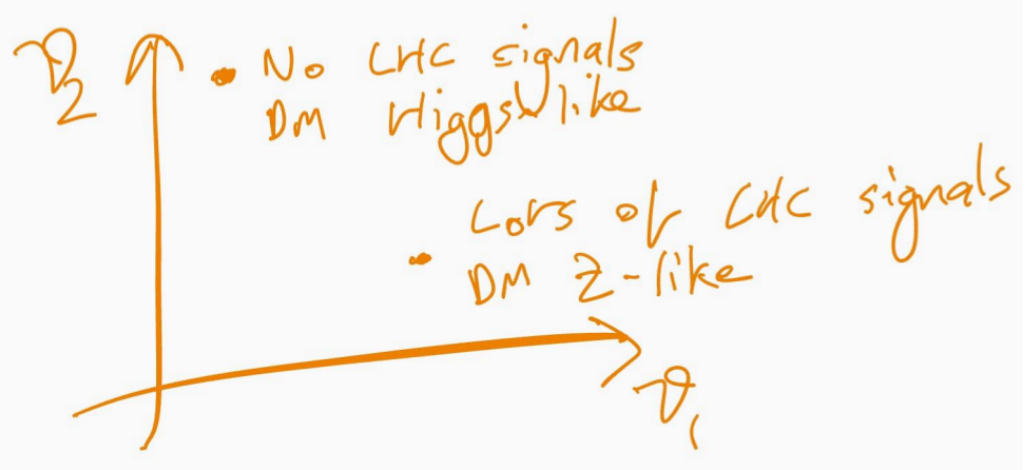
27 Members, 9 Experiments, 4 major theory codes, 10 countries

Why global fits?

- Lots of theories of TeV scale physics



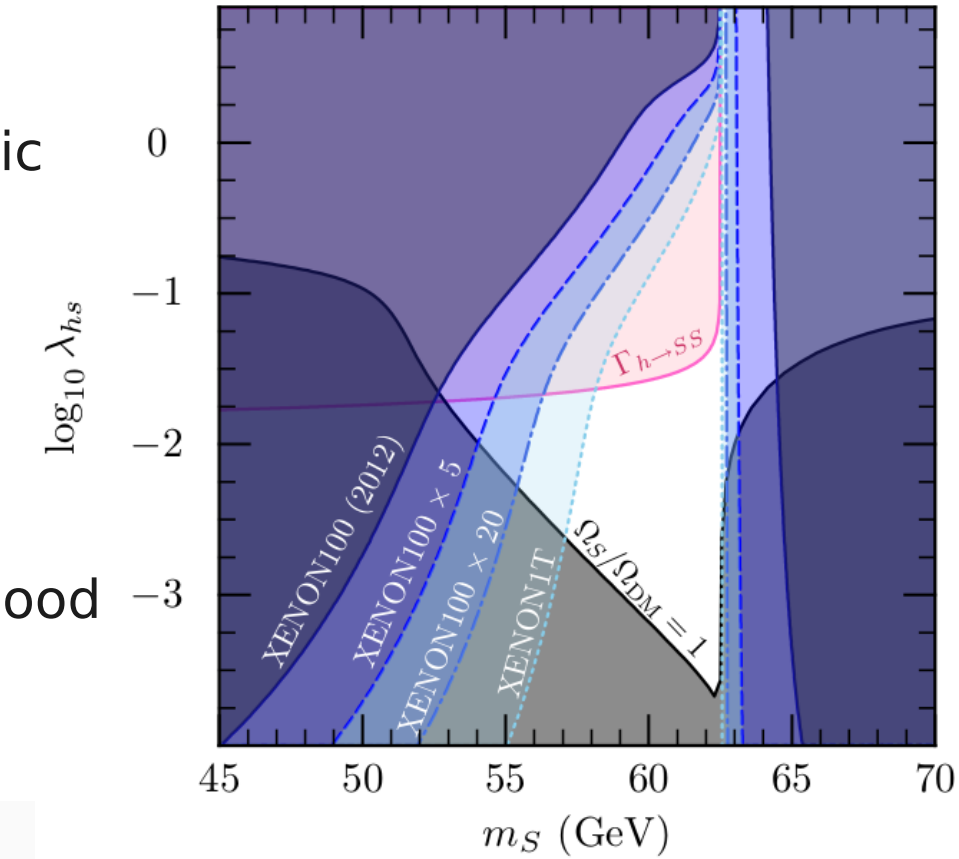
- For each theory, a parameter space of varying phenomenology



- What new physics scenarios are preferred / ruled out? - Compare to data!

Why global fits?

- In/out cuts based on 95% confidence limits are useful to get a quick heuristic picture.
- But ultimately, we want to make valid statistical inferences.
- To do this, we must perform full likelihood calculations, and sample parameter spaces in a statistically valid way.



Cline, Kainulainen, Scott & Weniger, PRD, 1306.4710

$$\mathcal{L} = \mathcal{L}_{\text{Collider}}\,\mathcal{L}_{\text{Higgs}}\,\mathcal{L}_{\text{DM}}\,\mathcal{L}_{\text{EWPO}}\,\mathcal{L}_{\text{Flavour}} \ldots$$

But it is a hard problem

- **Even for a comprehensive in/out analysis, the problem is multi-faceted:**
  - Pick a model
  - Compute predictions for all physical observables of interest
  - Compare these predictions to experimental limits
    - Sometimes limits already exist directly on the relevant quantities for a model.
    - Usually need to recast limits
      - May involve numerically intensive simulations of specific experiments, e.g. simulating LHC collisions, neutrino events (IceCube), gamma ray events (Fermi-LAT), WIMP-nucleon scattering.
      - Limits often dependent on background model assumptions (e.g. dark matter halo models, cosmology, simplified model assumptions)

But it is a hard problem

- **For full statistically valid global fit, even harder**
  - Need likelihood calculations for many experiments – can be numerically intensive
  - Need computations at many (millions - hundreds of millions) of model points, for moderately complex models (e.g. MSSM7-30)

A lot of effort just for one model family.

Organisation is essential

- **It requires a huge amount of manpower to prepare analyses of this kind even for one model class (e.g. MSSM).**

- **Need to reuse as many calculations and as much "book-keeping" code as possible.**

GAMBIT overview

Key design points

- Reuse calculations to the maximum theoretically permissible extent
- Graph-based dependency resolution for run-time “plug and play”
  - Keep calculations as modular as possible so that any piece can be easily “swapped out” for an alternate calculation.
  - Seamless integration of new calculations as they become available (minimise “hacking” of existing code).

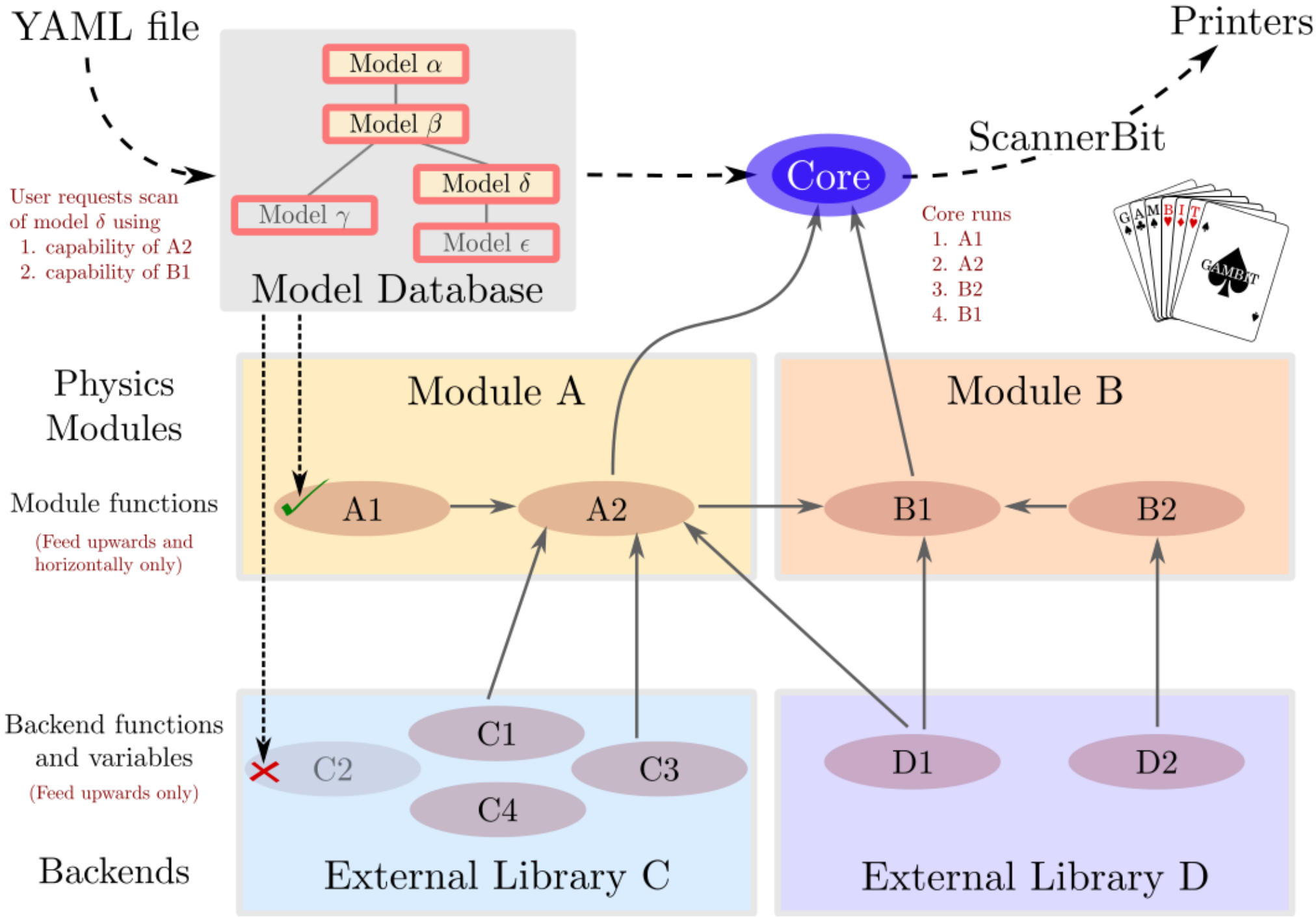
Modular structure

Physics modules

- DarkBit – dark matter observables (relic density, direct +indirect detection)
- ColliderBit – collider observables inc. Higgs + SUSY searches from ATLAS, CMS + LEP
- FlavBit – flavour physics inc. $g-2$, $b \to s\gamma$, B decays (new channels, angular obs., theory uncerts, LHCb likelihoods)
- SpecBit – generic BSM spectrum object, providing RGE running, masses, mixings, etc via interchangeable interfaces to different RGE codes
- DecayBit – decay widths for all relevant SM & BSM particles
- PrecisionBit – SM likelihoods, precision BSM tests (W mass, $\Delta\rho$ etc)

Each consists of a number of **module functions** that can have **dependencies** on each other.

+ScannerBit: manages stats, sampling and optimisation


YAML file
Printers
Model α
Model β
Model γ
Model δ
Model ϵ
Model Database
ScannerBit
Core
User requests scan
of model δ using
1. capability of A2
2. capability of B1
Core runs
1. A1
2. A2
3. B2
4. B1
Physics
Modules
Module A
Module B
Module functions
(Feed upwards and
horizontally only)
A1
A2
B1
B2
Backend functions
and variables
(Feed upwards only)
C1
C2
C3
C4
D1
D2
Backends
External Library C
External Library D

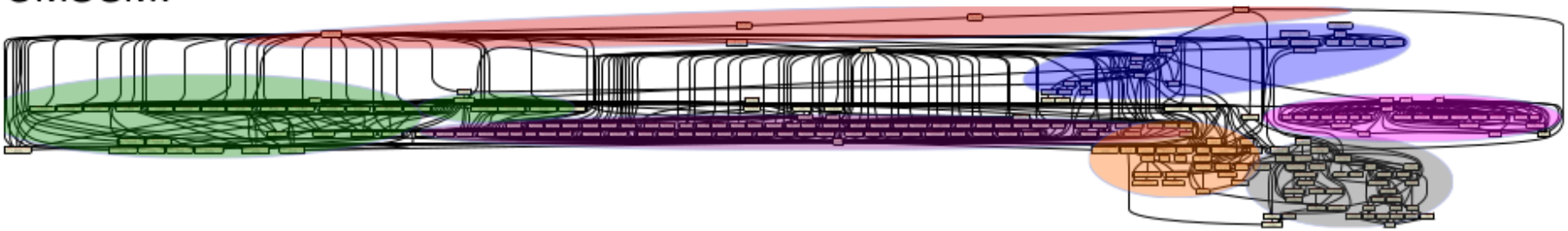
Module function declaration

```
#define CAPABILITY MSSM_spectrum
START_CAPABILITY

  // FlexibleSUSY compatible maximal CMSSM generalisation (MSSM with GUT boundary conditions)
  #define FUNCTION get_MSSMatMGUT_spectrum
  START_FUNCTION(Spectrum)
  ALLOW_MODELS(MSSM63atMGUT)
  DEPENDENCY(SMINPUTS, SMInputs) // Need SLHA2 SMINPUTS to set up spectrum generator
  #undef FUNCTION

#undef CAPABILITY
```

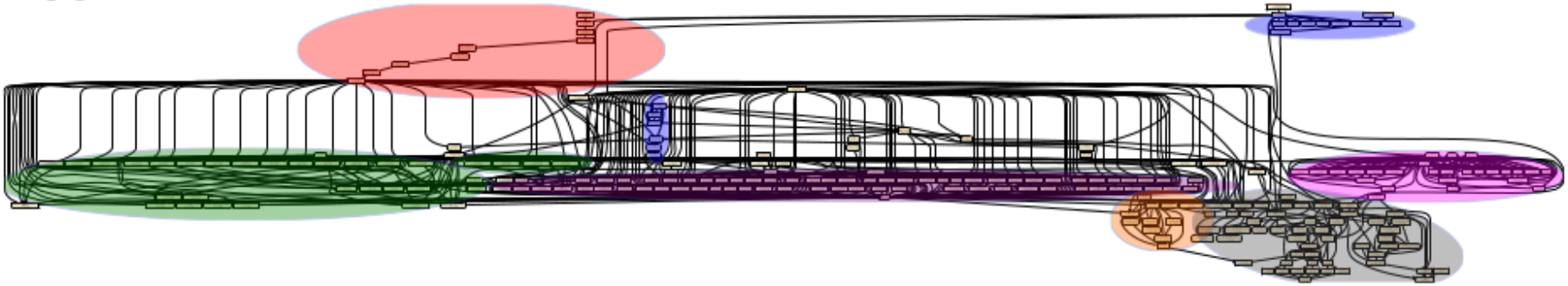
```
// FeynHiggs SUSY masses and mixings
#define CAPABILITY FH_MSSMMasses
START_CAPABILITY
  #define FUNCTION FH_MSSMMasses
  START_FUNCTION(fh_MSSMMassObs)
  BACKEND_REQ(FHGetPara, (libfeynhiggs), void, (int&,int&,
              Farray<fh_real, 1,2, 1,5, 1,3>&, Farray<fh_complex, 1,2, 1,2, 1,5, 1,3>&,
              Farray<fh_real, 1,6, 1,5>&, Farray<fh_complex, 1,36, 1,5>&,
              Farray< fh_real,1,2>&, Farray< fh_complex,1,4>&,
              Farray< fh_complex,1,4>&, Farray< fh_real,1,4>&,
              Farray< fh_complex,1,16>&, fh_complex&, fh_real&,
              Farray< fh_real,1,4>&, fh_real&))
  BACKEND_OPTION( (FeynHiggs, 2.11.2, 2.11.3), (libfeynhiggs) )
  ALLOW_MODELS(MSSM63atQ, MSSM63atMGUT)
  #undef FUNCTION
#undef CAPABILITY
```

Example dependency graphs

CMSSM:

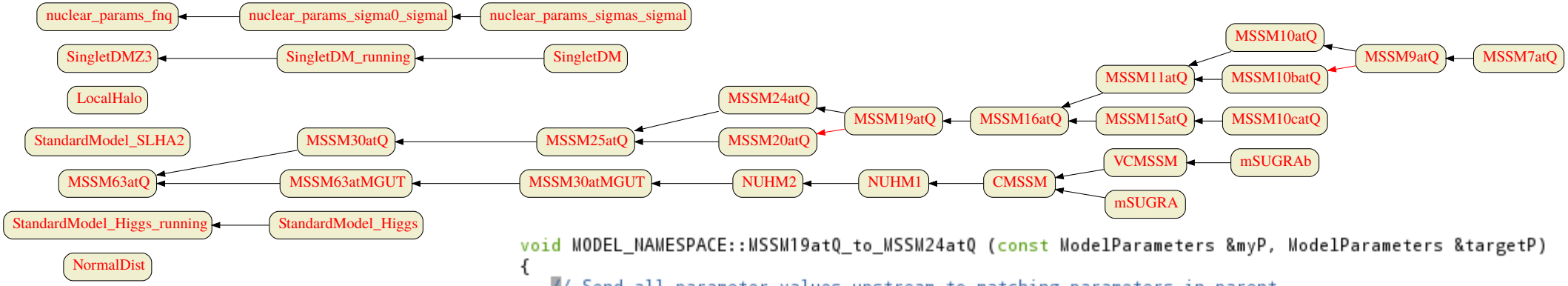


MSSM7:



Red: Model parameter translations
Blue: Precision calculations
Green: LEP rates+likelihoods
Purple: Decays
Orange: LHC observables and likelihoods
Grey: DM direct, indirect and relic density
Pink: Flavour physics

Hierarchical Model Database




```
#define MODEL MSSM19atQ
#define PARENT MSSM24atQ
  START_MODEL
  DEFINEPARS(Qin,TanBeta,SignMu,
             mHu2,mHd2,M1,M2,M3)
  DEFINEPARS(mq2_12, mq2_3)
  DEFINEPARS(ml2_12, ml2_3)
  DEFINEPARS(md2_12, md2_3)
  DEFINEPARS(mu2_12, mu2_3)
  DEFINEPARS(me2_12, me2_3)
  DEFINEPARS(Ae_3)
  DEFINEPARS(Ad_3)
  DEFINEPARS(Au_3)
  INTERPRET_AS_PARENT_FUNCTION(MSSM19atQ_to_MSSM24atQ)
  INTERPRET_AS_X_FUNCTION(MSSM20atQ, MSSM19atQ_to_MSSM20atQ)
#undef PARENT
#undef MODEL
```

```
void MODEL_NAMESPACE::MSSM19atQ_to_MSSM24atQ (const ModelParameters &myP, ModelParameters &targetP)
{
   // Send all parameter values upstream to matching parameters in parent.
   // Ignore that some parameters don't exist in the parent, as these are set below.
   targetP.setValues(myP,false);
   // RH squark soft masses, gen 1 and 2
   targetP.setValue("mq2_1",  myP["mq2_12"] ); // mq2_11 in MSSM63
   targetP.setValue("mq2_2",  myP["mq2_12"] ); // mq2_22   " "
   // RH slepton soft masses, gen 1 and 2
   targetP.setValue("ml2_1",  myP["ml2_12"] ); // ml2_11 in MSSM63
   targetP.setValue("ml2_2",  myP["ml2_12"] ); // ml2_22   " "
   // LH down-type squark soft masses
   targetP.setValue("md2_1",  myP["md2_12"] ); // ml2_11 in MSSM63
   targetP.setValue("md2_2",  myP["md2_12"] ); // ml2_22   " "
   // LH up-type squark soft masses
   targetP.setValue("mu2_1",  myP["mu2_12"] ); // mu2_11 in MSSM63
   targetP.setValue("mu2_2",  myP["mu2_12"] ); // mu2_22   " "
   // LH charged slepton soft masses
   targetP.setValue("me2_1",  myP["me2_12"] ); // me2_11 in MSSM63
   targetP.setValue("me2_2",  myP["me2_12"] ); // me2_22   " "
}
```

Mix and match backends

BACKENDS	VERSION	PATH TO LIB	STATUS	#FUNC	#TYPES	#CTORS
DDCalc0	0.0	Backends/installed/DDCalc/0.0/libDDCalc0.so	OK	62	0	0
DarkSUSY	5.1.1	Backends/installed/DarkSUSY/5.1.1/lib/libdarksusy.so	OK	68	0	0
FastSim	1.0	Backends/installed/fastsim/1.0/libfastsim.so	absent/broken	1	0	0
FeynHiggs	2.11	Backends/installed/FeynHiggs/2.11.2/lib/libFH.so	OK	14	0	0
HiggsBounds	4.2.1	Backends/installed/HiggsBounds/4.2.1/lib/libhiggsbounds.so	OK	10	0	0
HiggsSignals	1.4	Backends/installed/HiggsSignals/1.4.0/lib/libhiggssignals.so	OK	11	0	0
LibFarrayTest	1.0	Backends/examples/libFarrayTest.so	OK	9	0	0
LibFirst	1.0	Backends/examples/libfirst.so	OK	8	0	0
	1.1	Backends/examples/libfirst.so	OK	15	0	0
LibFortran	1.0	Backends/examples/libfortran.so	OK	6	0	0
MicrOmegas	3.5.5	Backends/installed/micromegas/3.5.5/MSSM/MSSM/libmicromegas.so	OK	15	0	0
MicrOmegasSingletDM	3.5.5	Backends/installed/micromegas/3.5.5/SingletDM/SingletDM/libmicromegas.so	OK	13	0	0
Pythia	8.186	Backends/installed/Pythia/8.186/lib/libpythia8.so	absent/broken	0	27	105
	8.209	Backends/installed/Pythia/8.209/lib/libpythia8.so	OK	0	28	107
SUSYPOPE	0.2	no path in config/backend_locations.yaml	absent/broken	3	0	0
SUSY_HIT	1.5	Backends/installed/SUSY-HIT/1.5/libsusyhit.so	OK	55	0	0
SuperIso	3.4	Backends/installed/SuperIso/3.4/libsuperiso.so	OK	32	0	0
gamLike	1.0.0	Backends/installed/gamLike/1.0.0/lib/gamLike.so	OK	3	0	0
nulike	1.0.0	Backends/installed/nulike/1.0.0/lib/libnulike.so	OK	4	0	0

Generic container objects

Mass spectra:

Generic interface functions:

```
double MZ = spec.get(Pole_Mass,"Z0");

spec.RunToScale(MZ);
```

Backend wrapper:

```
tmp_map["~d"] =  FInfo1( &Model::get_MSd_pole_slha, i012345 );
tmp_map["~u"] =  FInfo1( &Model::get_MSu_pole_slha, i012345 );
tmp_map["~e-"] = FInfo1( &Model::get_MSe_pole_slha, i012345 );
tmp_map["~nu"]=  FInfo1( &Model::get_MSv_pole_slha, i012 );
tmp_map["h0"] =  FInfo1( &Model::get_Mhh_pole_slha, i01 );
tmp_map["~chi+"] = FInfo1( &Model::get_MCha_pole_slha, i01 );
tmp_map["~chi0"] = FInfo1( &Model::get_MChi_pole_slha, i0123 );

map_collection[Par::Pole_Mass].map1 = tmp_map;
```

PDG code convenience overloads:

```
spectrum.get(Pole_Mass,"~e-",1);
spectrum.get(Pole_Mass,"~e-_1");
spectrum.get(Pole_Mass,std::make_pair("~e-",1));
spectrum.get(Pole_Mass,1000011,0);
spectrum.get(Pole_Mass,std::make_pair(1000011,0));

spectrum.get(Pole_Mass,"~e+",1);
spectrum.get(Pole_Mass,"~e+_1");
spectrum.get(Pole_Mass,std::make_pair("~e+",1));
spectrum.get(Pole_Mass,-1000011,0);
spectrum.get(Pole_Mass,std::make_pair(-1000011,0));
```

Decays:

```
// Declare a set of decays for the W+
DecayTable::Entry Wplus;

// Set the total width
Wplus.width_in_GeV = 2.085;

// Set the branching fraction for W → e νe
Wplus.set_BF(0.1071, 0.0016, "e+", "nu_e");

// Find the partial width for W → e νe
double pw = Wplus.width_in_GeV*Wplus.BF("e+","nu_e");
```

```
// Declare a DecayTable
DecayTable BosonDecays

// Add entries ("here are some I prepared earlier")
BosonDecays("W+")    = Wplus;
BosonDecays("W-")    = Wminus;
BosonDecays("Z0")    = Z;
BosonDecays("h0_1") = SM_higgs;

// Get the branching fraction for W → e νe
double enueBF = BosonDecays("W+").BF("e+", "nu_e");
```

ProcessCatalogue (DarkBit) – Decay and annihilation processes
(used for dark matter annihilation rates, gamma-ray and neutrino yields for indirect searches)

Multi-format output system

Current plugins: ASCII and HDF5

Results of module function calculations automatically streamed to output system, in format selected at runtime.

HDF5 support added to Pippi (“parse it, plot it” https://github.com/patscott/pippi)

But HDF5 interface exists in many languages (C, Fortran, Java, C++, Python)

e.g. Plotting in Python is easy:

```
import h5py
import numpy as np

f = h5py.File("runs/spartan_multinest_hdf5/samples/results.hdf5",'r')
group = f["/gambit_data"]

mu = group["#NormalDist_parameters @NormalDist::primary_parameters::mu"]
```

Uniform interface to scanning algorithms and priors (ScannerBit)

- **External algorithms (or native C++) wrapped into plugins:**

```
scanner_plugin(random_sampler, version(1, 0, 0))
{
    like_ptr LogLike;
    int num, dim;

    plugin_constructor
    {
        LogLike = get_purpose(get_inifile_value<std::string>("like"));
        num = get_inifile_value<int>("point_number", 10);
        dim = get_dimension();
    }

    int plugin_main ()
    {
        std::vector<double> a(dim);

        std::cout << "Entering random sampler." << "\n\tnumber of points to calculate:  " << num << std::endl;

        for (int k = 0; k < num; k++)
        {
            for (int i = 0; i < dim; i++)
            {
                a[i] = Gambit::Random::draw();
            }
            LogLike(a);

            if (k%1000 == 0)
                std::cout << "points:  " << k << " / " << num << std::endl;
        }

        return 0;
    }
}
```

Other tools:

CMake build system to organise compilation of many tools and modules.

Scanners: MultiNest, Diver (diff. evolution), twalk (population MC), GreAT (MCMC)

Parallelisation: Mixed mode MPI + openMP, mostly automated

Backends: dynamic loading of C++ classes from backends (BOSS)

POSIX signal handling: for safe early shutdown/resuming

LHC likelihoods

- MC generation: Pythia8 parallelised with OpenMP + other speed tweaks
- Detector simulation: fast simulation based on 4-vector smearing $\rightarrow$ matches DELPHES results very closely (but much faster)
- Cross-sections: LO + LL from MC generator by default (fast NLO in works for SUSY)
- Analysis framework: custom event-level, independent of experiment or simulation
- Likelihood: inline systematic error marginalisation (via `nulike`)
- Initially shipping with:
  - ATLAS SUSY searches (0ℓ, $0/1/2\ell$ $\tilde{t}$, b jets + MET, $2/3\ell$ EW)
  - CMS multi-ℓ SUSY
  - CMS DM (t pair + MET, mono-b, monojet)

For more details, see talk by Martin White
(1630 Thursday, Experimental and Collider Aspects of SUSY session)

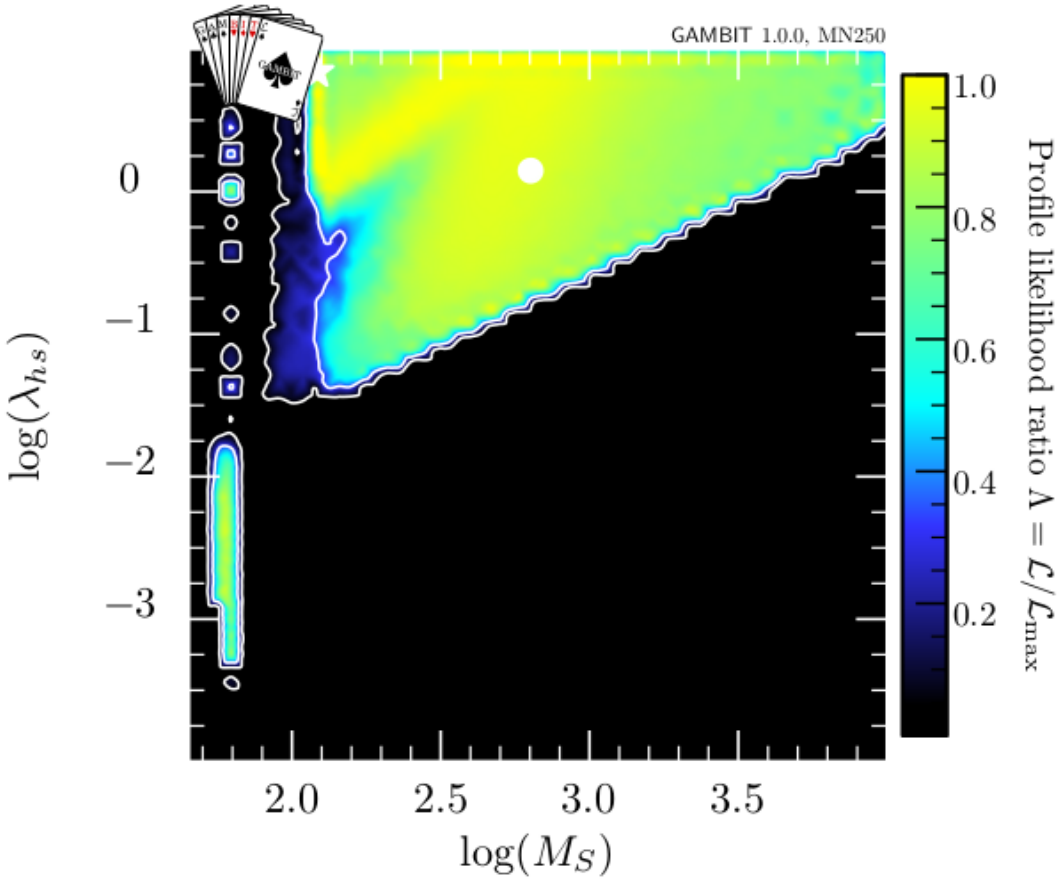
Comparison of tools

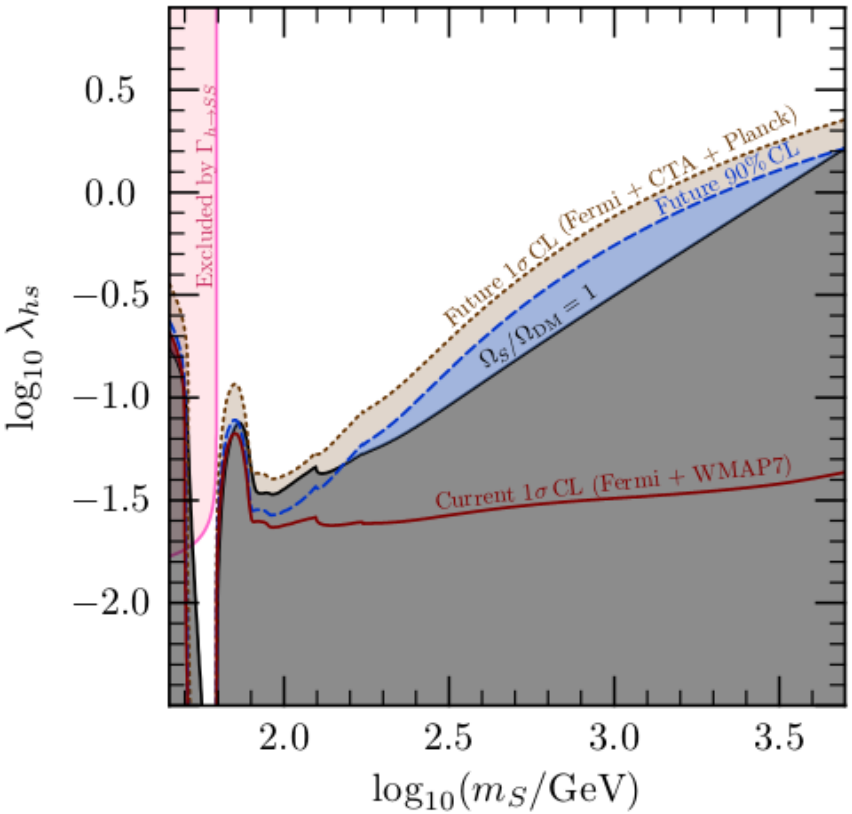
Model point description	Ωh^2		$\sigma_{\mathrm{SI},p}$ $[10^{-46}\mathrm{cm}^2]$		$\sigma_{\mathrm{SD},p}$ $[10^{-40}\mathrm{cm}^2]$	
	DarkSUSY	micrOMEGAs	DarkSUSY	micrOMEGAs	DarkSUSY	micrOMEGAs
Resonant annihilation via A^0, gaugino-like neutralino	0.08110	0.07877	4.317	4.778	10.20×10^{-6}	9.381×10^{-6}
Resonant annihilation via A^0, mixed neutralino	0.08067	0.08521	1.827	1.937	5.713	5.422
Resonant annihilation via A^0, Higgsino-like neutralino	0.08063	0.08543	0.7372	0.6826	1.564×10^{-4}	1.485×10^{-4}
Sfermion coannihilations, gaugino-like neutralino	0.1775	0.2461	1.667	1.716	4.209×10^{-3}	24.59×10^{-3}
Sfermion coannihilations, mixed neutralino	0.09823	0.1112	404.1	378.5	0.2476	0.2125
Sfermion coannihilations, Higgsino-like neutralino	7.549×10^{-3}	0.1095	18.74	29.87	1.686×10^{-3}	1.464×10^{-3}
Chargino coannihilations	0.09655	0.09997	0.5142	0.4848	16.92	16.05
Chargino coannihilations	0.08138	0.08720	4.147	3.954	9.122	8.657

Table 20: A table showing the dark matter relic density and proton scattering cross-sections, both spin-independent and spin-dependent, for a range of MSSM model points. The points were chosen to have different types of processes contribute to the relic density calculation as shown in the model point description column. All quantities were calculated through `DarkBit_standalone_MSSM` using both the DarkSUSY and micrOMEGAs backends.

Comparison of scanning algorithms

Scalar singlet dark matter
(Relic density + direct detection + Multinest scanner)

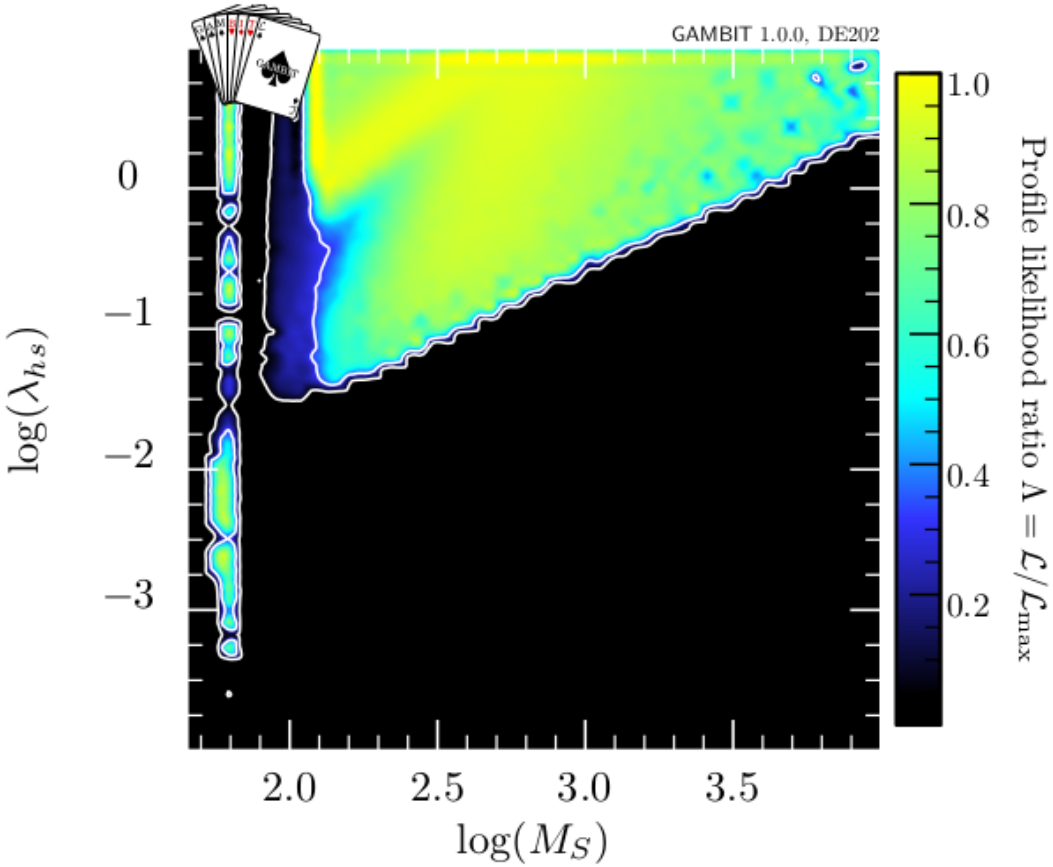


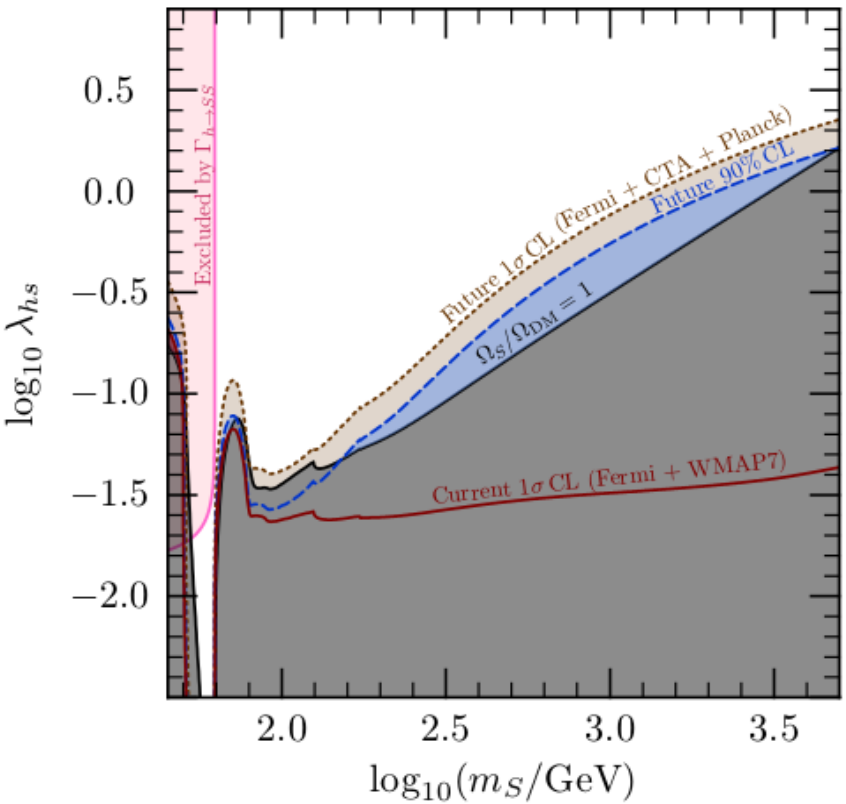


Cline et al. 2013

Comparison of scanning algorithms

Scalar singlet dark matter
(Relic density + direct detection + Diver scanner)





Cline et al. 2013

Summary

- GAMBIT to be released later this year
- As well as likelihoods and physics calculations for popular models, offers a robust framework and numerous helpful utilities for systematically including more and more models, likelihoods for new experiments, and scanning algorithms over time (we want to backend your codes!)
- We are very happy to talk to prospective users!

  (some other GAMBITers here at SUSY2016 to talk to:

  - Peter Athron
  - Csaba Balazs
  - Paul Jackson
  - Chris Rogan
  - Martin White)