FlexibleSUSY – A spectrum generator generator for supersymmetric models

Alexander Voigt

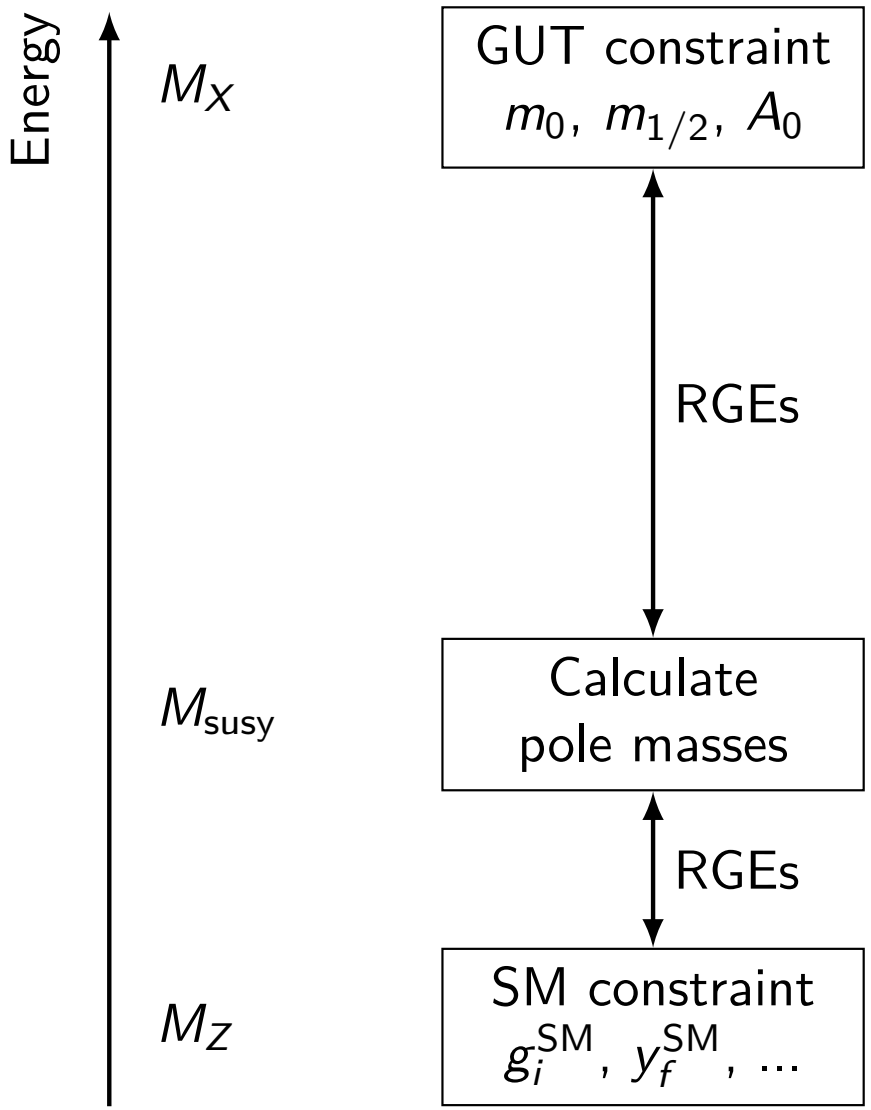
Technische Universität Dresden
Institut für Kern- und Teilchenphysik

DPG Spring Meeting in Mainz
26 March 2014, Session T 77.6

In collaboration with Peter Athron, Jae-hyeon Park and Dominik Stöckinger

Content

Motivation – What is a spectrum generator?

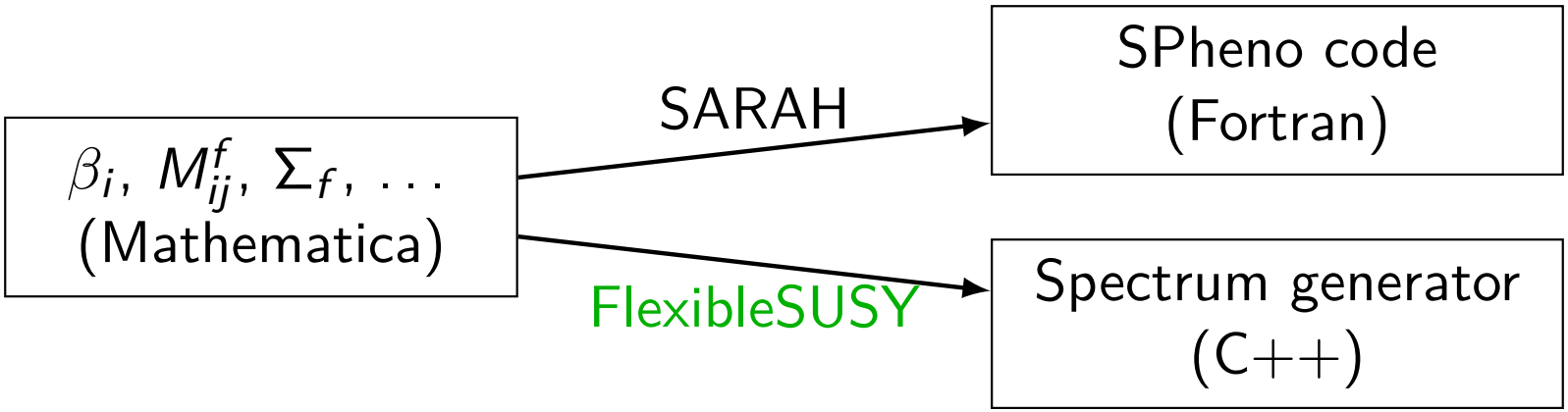


Motivation – Available spectrum generators

Model	Spectrum generator
MSSM	ISASUSY[1], Softsusy[2], SPheno[3], SuSeFlav[4], SuSpect[5]
NMSSM	NMSPEC[6], Softsusy[2], SPheno[3]
CE_6SSM	cE6SSM_SpecGen
any susy model	SARAH[7], FlexibleSUSY[8]

[1] http://nhn.nhn.ou.edu/~isajet/
[2] https://softsusy.hepforge.org
[3] https://spheno.hepforge.org
[4] http://cts.iisc.ernet.in/Suseflav/main.html
[5] http://www.coulomb.univ-montp2.fr/perso/jean-loic.kneur/Suspect/
[6] http://www.th.u-psud.fr/NMHDECAY/nmssmtools.html
[7] https://sarah.hepforge.org
[8] https://flexiblesusy.hepforge.org

Motivation – Why creating a new one?



Motivation:

- large variety of supersymmetric models

$\rightarrow$ user customization desired

- convergence problems in certain parameter regions

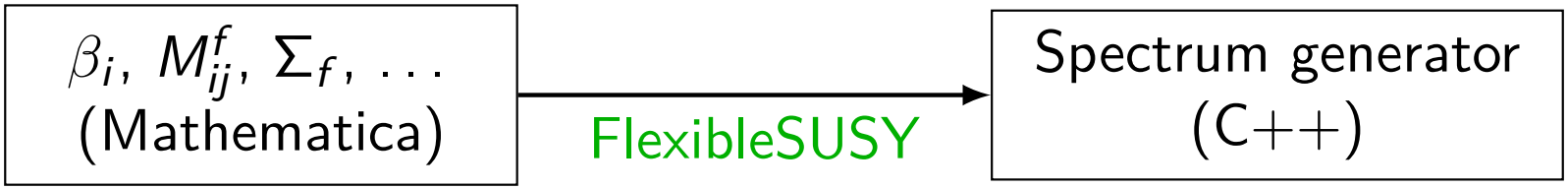
$\rightarrow$ provide alternative RG solvers

- High-dimensional parameter space

$\rightarrow$ short run-time desired

Content

Design goals



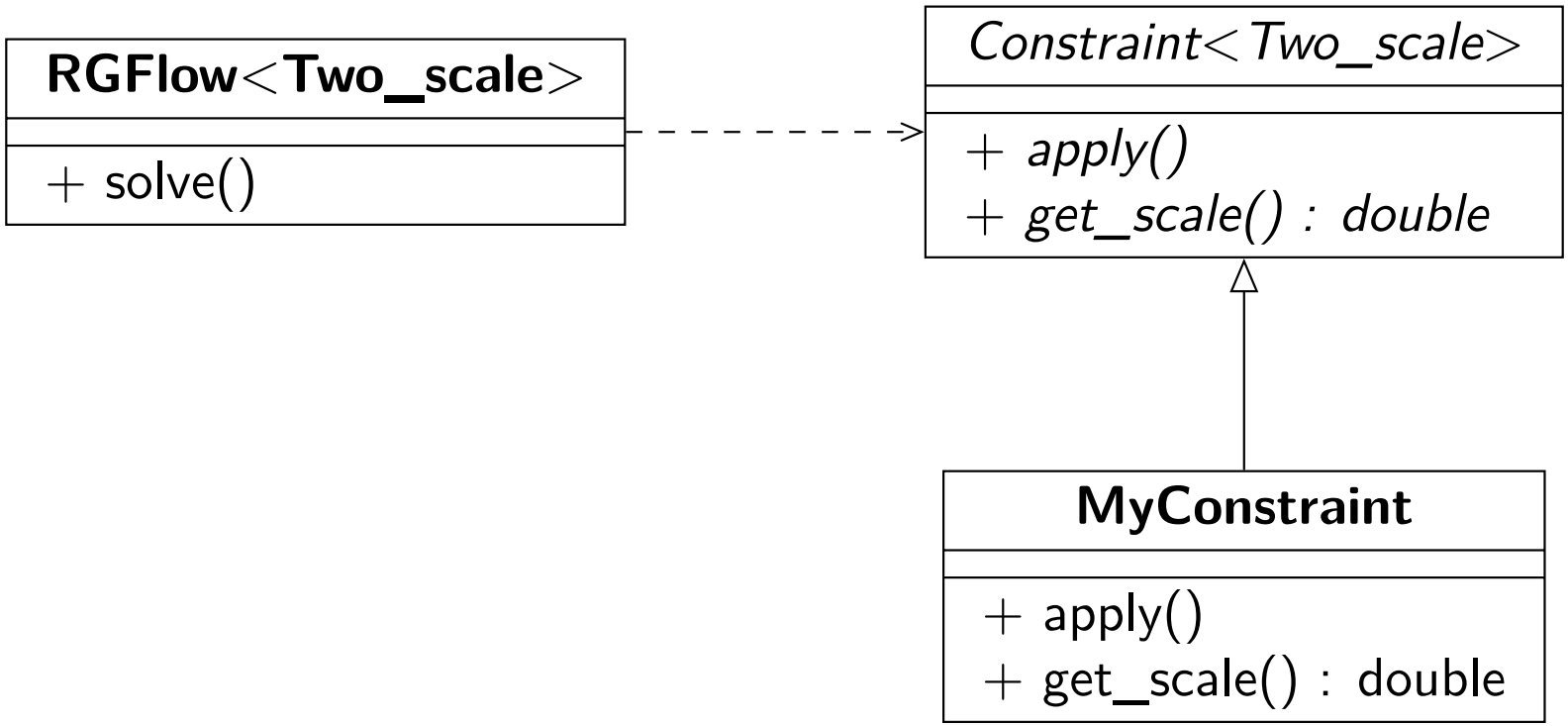
Design goals:

- modular, object oriented C++ code $\Rightarrow$ easy to hack! ✓
- multiple RGE solvers:
  - two-scale running (adaptive Runge-Kutta) ✓
  - lattice method + variants (Jae-hyeon Park) ✓
- fast (smart linear algebra, multithreading) ✓
- SARAH-like user interface ✓
- tower of effective field theories ✗

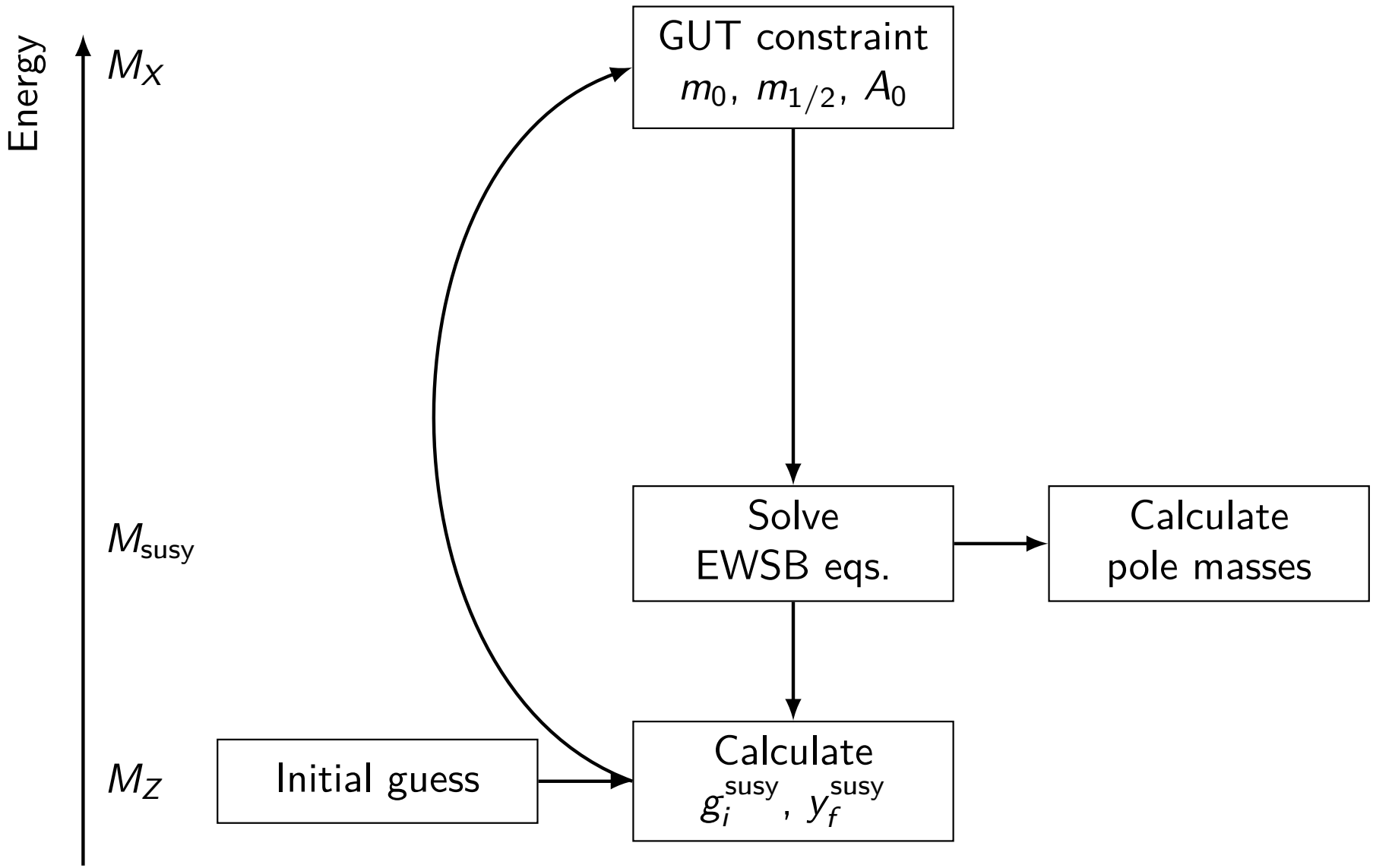
Design goals – Modularity – generated C++ code

```
typedef Two_scale T; // or Lattice
MSSM<T> mssm;
MSSM_input_parameters input;

std::vector<Constraint<T>*> constraints = {
  new MSSM_low_scale_constraint<T>(input),
  new MSSM_susy_scale_constraint<T>(input),
  new MSSM_high_scale_constraint<T>(input)
};

// solve RG eqs. with the above constraints
RGFlow<T> solver;
solver.add_model(&mssm, constraints);
solver.solve();

mssm.calculate_spectrum();
```

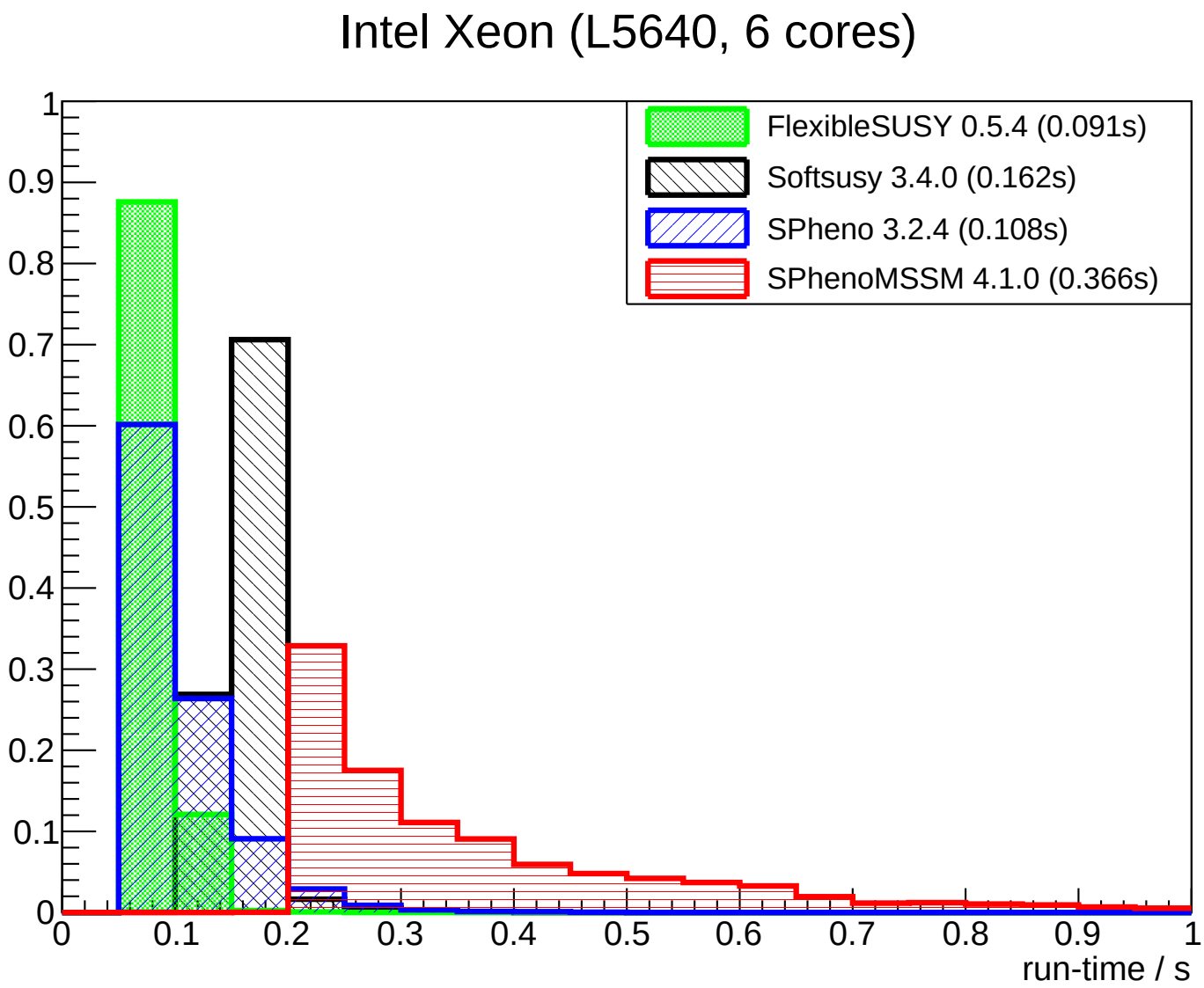
Design goals – Modularity – Constraint interface



Design goals – RGE solvers – two-scale algorithm



Design goals – Speed – run-time comparison CMSSM



g++ 4.8.0, ifort 13.1.3 20130607

Content

Usage example

Get the source code from http://flexiblesusy.hepforge.org

Create a NMSSM spectrum generator:

```
$ ./createmodel --name=NMSSM
$ ./configure --with-models=NMSSM
$ make
```

Run it:

```
$ ./models/NMSSM/run_NMSSM.x
```

Run it with the SLHA interface:

```
$ ./models/NMSSM/run_NMSSM.x \
   --slha-input-file=input.slha \
   --slha-output-file=output.slha
```

Conclusions

FlexibleSUSY http://flexiblesusy.hepforge.org

- is **Modular** (C++ classes, easy to modify and extend)
- is **Fast** (CMSSM run-time $\approx 0.1s$)
- Provides **different RGE solvers**
  - two-scale running (adaptive Runge-Kutta)
  - lattice method + variants

Currently supported models:

- MSSM, NMSSM, SMSSM, UMSSM, E6SSM, MRSSM

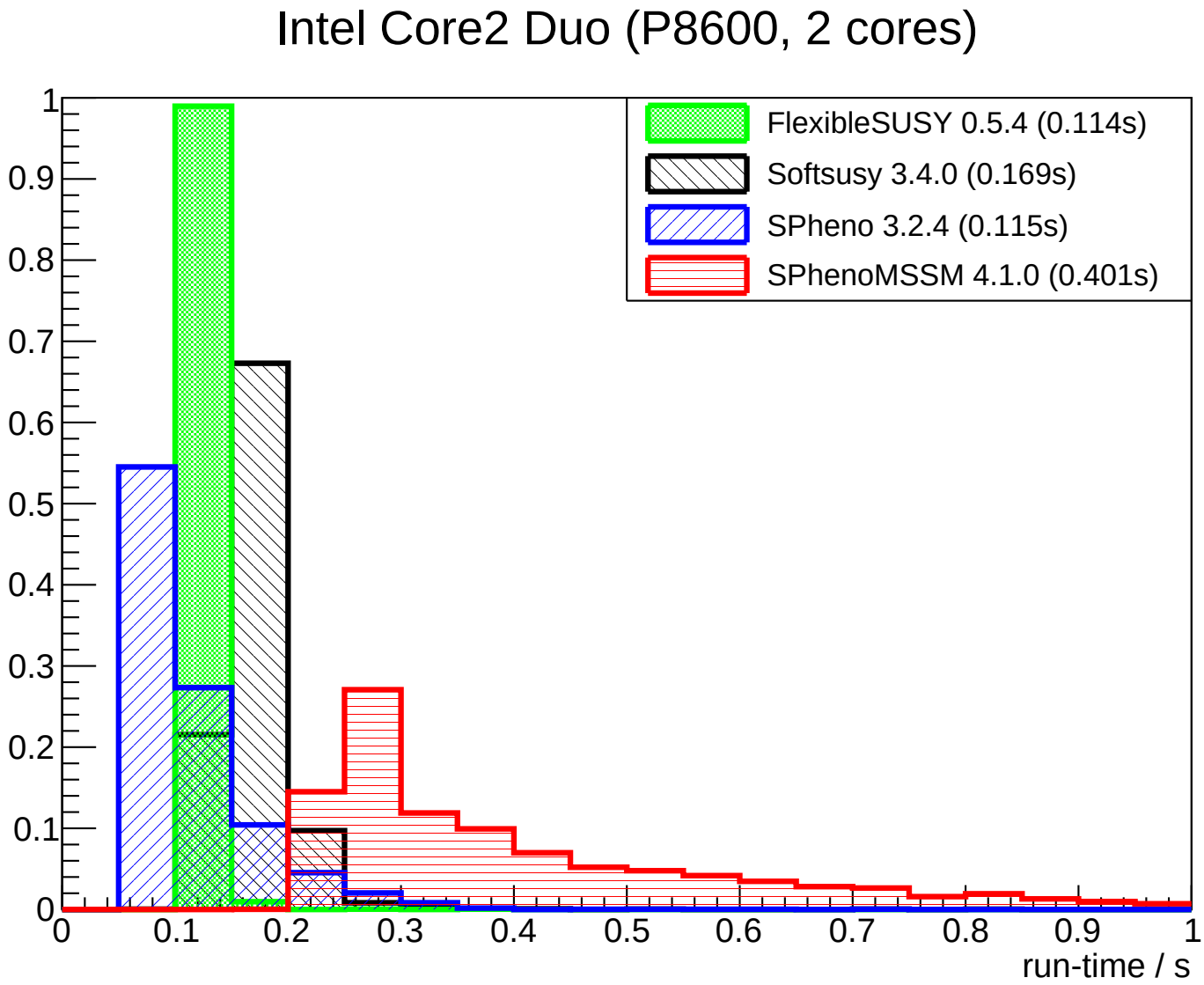
Future plans:

- tower of effective field theories
- calculate decays

FlexibleSUSY model file

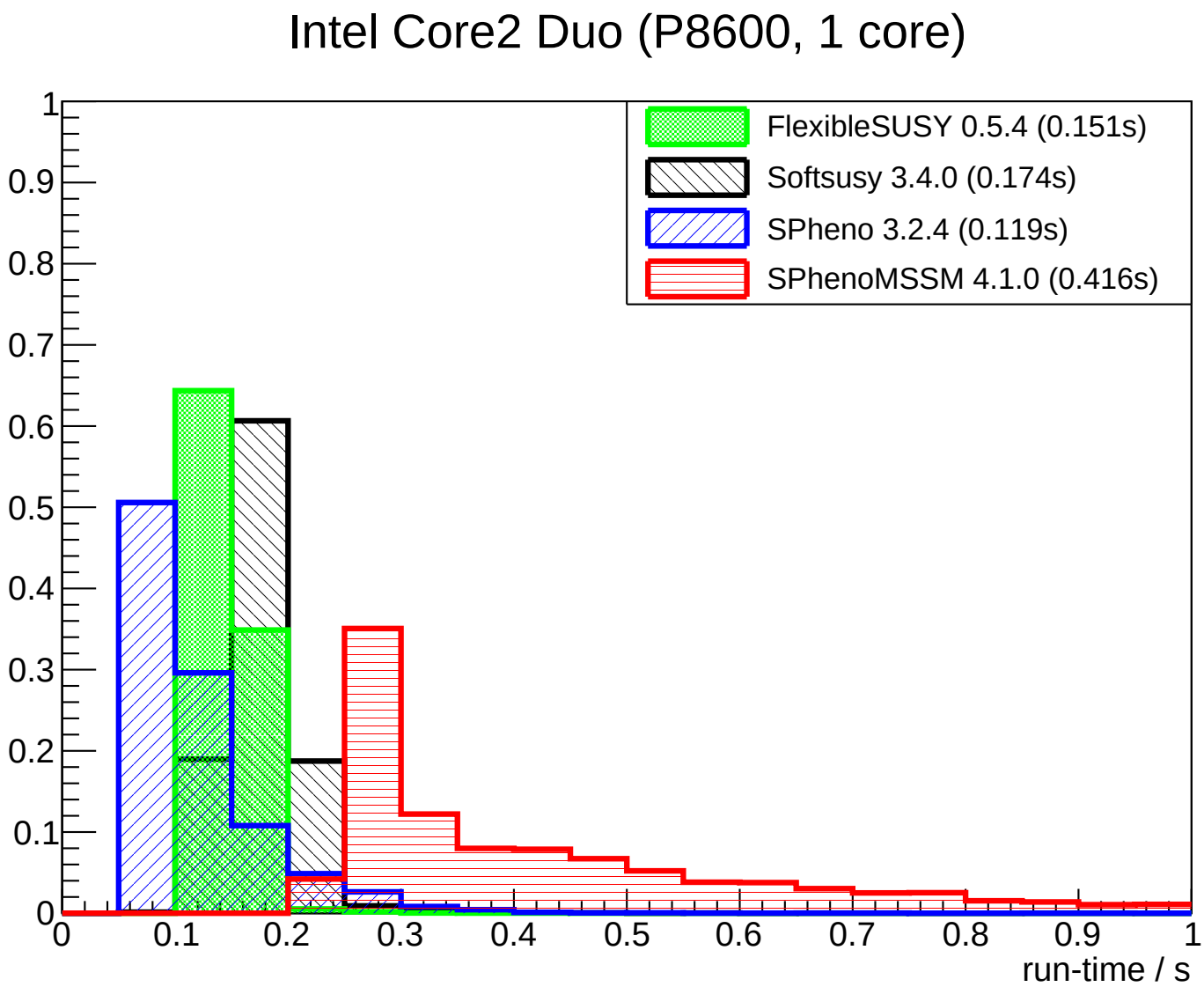
```
FSModelName = "MSSM";

MINPAR = { {1, m0},
           {2, m12},
           {3, TanBeta},
           {4, Sign[Mu]},
           {5, Azero} };

EWSBOutputParameters = { B[Mu], Mu };

SUSYScale = Sqrt[M[Su[1]]*M[Su[6]]];

HighScale = g1 == g2;

HighScaleInput = {
   {mHd2, m0^2}, {mHu2, m0^2}, {mq2, UNITMATRIX[3] m0^2},
   ...
};

LowScale = SM[MZ];

LowScaleInput = { ... };
```

Run-time comparison CMSSM – 2 cores



g++ 4.8.0, ifort 13.1.3 20130607

Run-time comparison CMSSM – 1 core



g++ 4.8.0, ifort 13.1.3 20130607