Beyond SM Monte Carlo with FeynRules

Claude Duhr

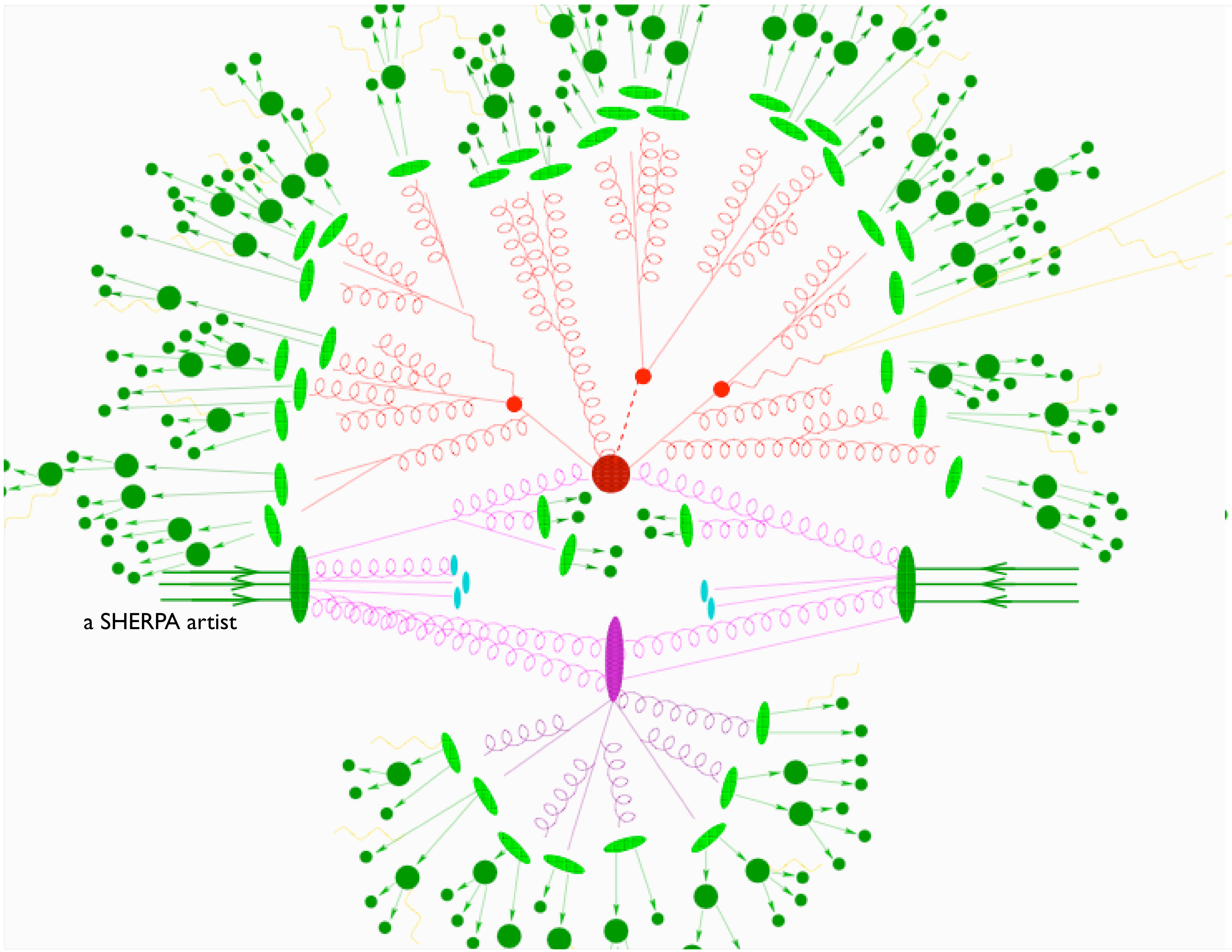
2011 IPMU-YITP School on Monte Carlo Tools for LHC
YITP, 08 September 2011

Going Beyond SM

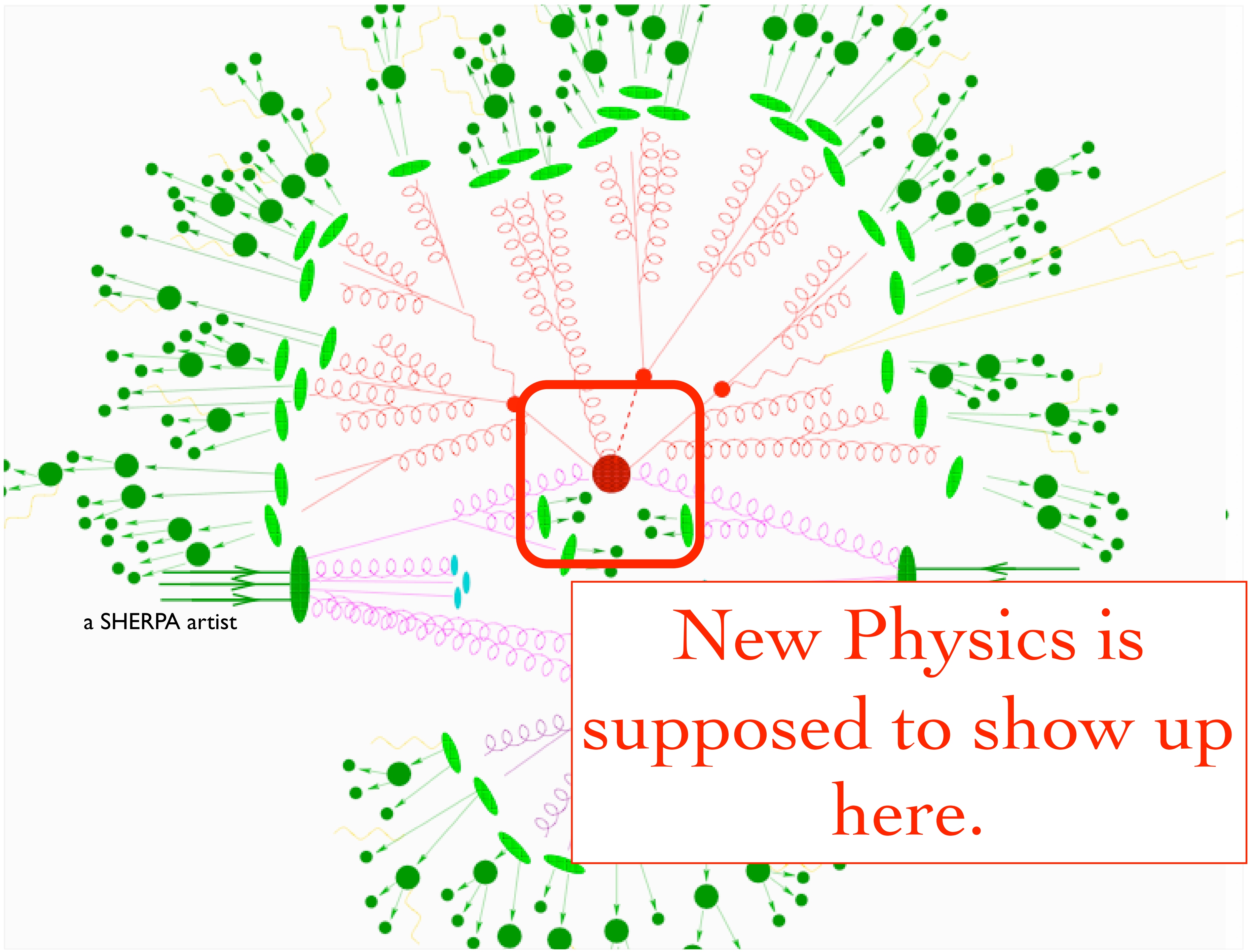
- So far in this School:
  - ➡ How to use/run Monte Carlo event generators to obtain physics results.
  - ➡ Focus was mostly on SM physics.

- Aim of this lecture:
  - ➡ How to obtain events for a BSM model that is *not yet* implemented into any of the existing MC codes.
  - ➡ In other words, what is an efficient way to implement a BSM model into a matrix element generator?

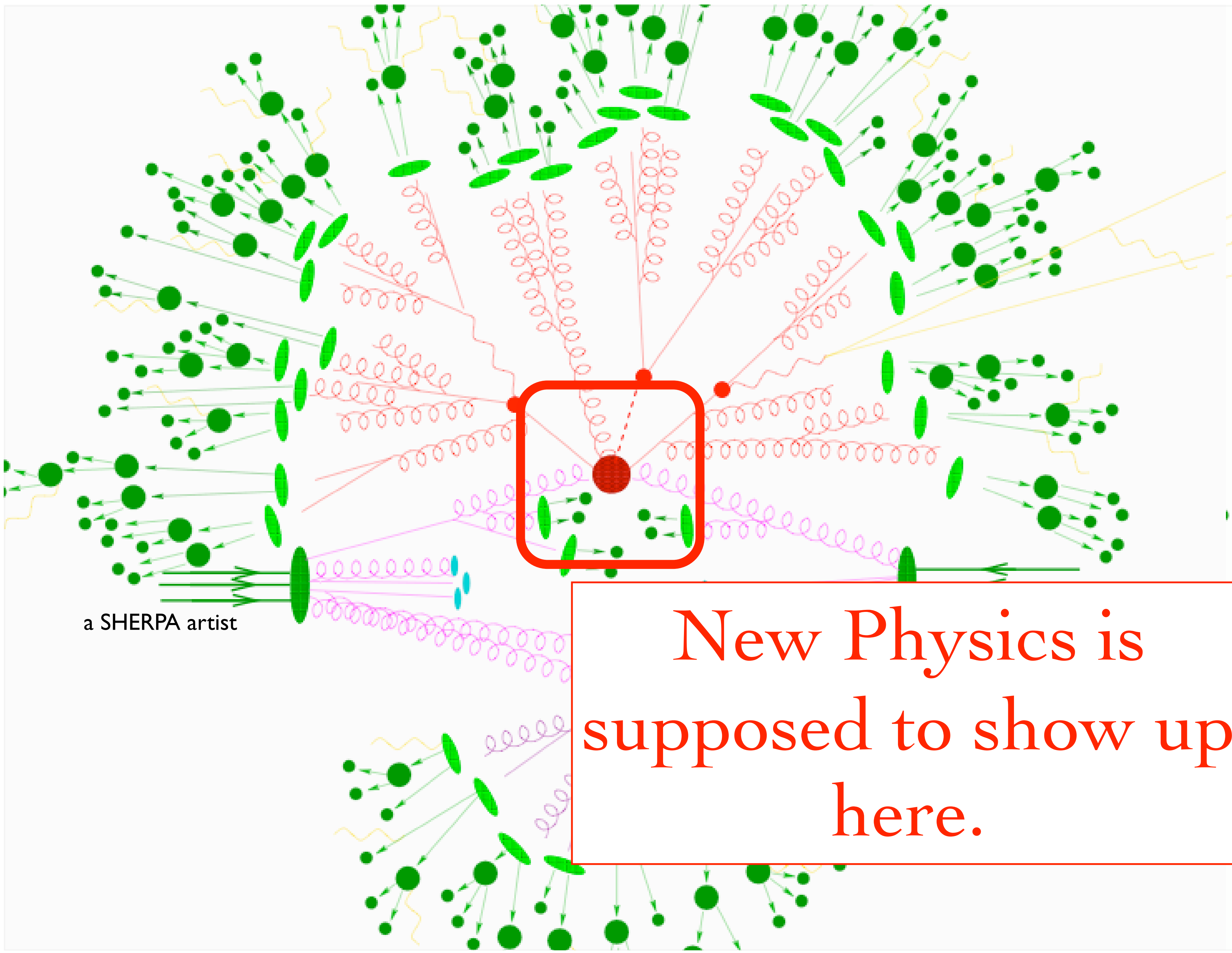
Going Beyond SM



Going Beyond SM

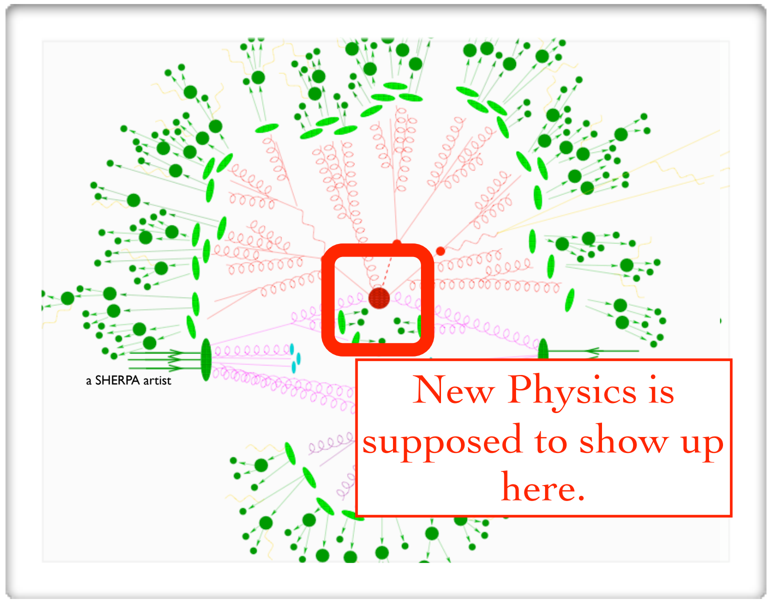


Going Beyond SM



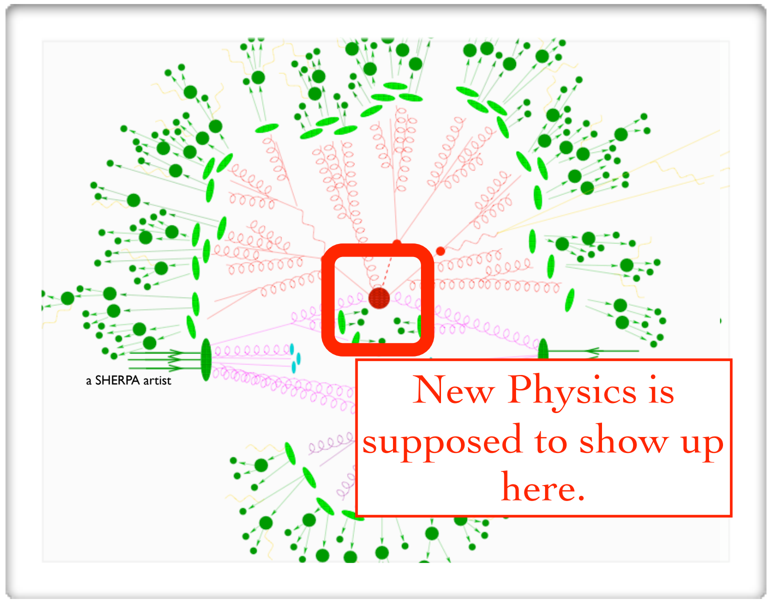
New Physics is supposed to show up here.

Going Beyond SM



- Parton Shower Monte Carlo Codes
  - ➡ Herwig
  - ➡ Pythia
  - ➡ Sherpa
- Multi-purpose LO matrix element generators (parton level)
  - ➡ CalcHep / CompHep
  - ➡ MadGraph / MadEvent
  - ➡ Sherpa (AMEGIC++, Comix)
  - ➡ Whizard / Omega

Going Beyond SM



- Parton Shower Monte Carlo Codes
  - ➡ Herwig
  - ➡ Pythia
  - ➡ Sherpa
- Multi-purpose LO matrix element generators (parton level)
  - ➡ CalcHep / CompHep
  - ➡ MadGraph / MadEvent
  - ➡ Sherpa (AMEGIC++, Comix)
  - ➡ Whizard / Omega

Going Beyond SM

- A BSM model can be defined via
  - ➡ The particles appearing in the model.
  - ➡ The values of the parameters ('Benchmark point').
  - ➡ The interactions among the particles, usually dictated by some symmetry group, and quantified in the Lagrangian of the model.
- All this information needs to be implemented into the MC codes, usually in the form of text files that contain the definitions of the particles, the parameters and the vertices.

Going Beyond SM

- This can be a very tedious exercise.
- Most of these codes have only a very limited amount of models implemented by default (~ SM and MSSM).
- However, still these codes do not work at the level of Lagrangians, but need explicit vertices.
- The process of implementing Feynman rules can be particularly tedious and painstaking:
  - ➡ Each code has its own conventions (signs, factors of i, ...).
  - ➡ Vertices need to be implemented one at the time.
- Most codes can only handle a limited amount of color and / or Lorentz structures (~ SM and MSSM)

Going Beyond SM

- Example: SUSY model

$$\mathcal{L} = \Phi^\dagger e^{-2gV}\Phi_{|_{\theta^2\bar\theta^2}} + \frac{1}{16g^2\tau_{\mathcal{R}}}\mathrm{Tr}(W^\alpha W_\alpha)_{|_{\theta^2}} + \frac{1}{16g^2\tau_{\mathcal{R}}}\mathrm{Tr}(\bar W_{\dot\alpha}\bar W^{\dot\alpha})_{|_{\bar\theta^2}}$$
$$+ W(\Phi)_{|_{\theta^2}} + W^\star(\Phi^\dagger)_{|_{\bar\theta^2}} + \mathcal{L}_{\mathrm{soft}}$$

- Very easy 'theory description'
  - ➡ Choose a gauge group (+ additional internal symmetries).
  - ➡ Choose the matter content (= chiral superfields in some representation).
  - ➡ Write down the most general superpotential.
  - ➡ Write down the soft-SUSY breaking terms.
  - ➡ (+ check validity of the model)

Going Beyond SM

- Example: SUSY model

$$\mathcal{L} = \Phi^\dagger e^{-2gV}\Phi_{|_{\theta^2\bar{\theta}^2}} + \frac{1}{16g^2\tau_{\mathcal{R}}}\mathrm{Tr}(W^\alpha W_\alpha)_{|_{\theta^2}} + \frac{1}{16g^2\tau_{\mathcal{R}}}\mathrm{Tr}(\bar{W}_{\dot{\alpha}}\bar{W}^{\dot{\alpha}})_{|_{\bar{\theta}^2}}$$
$$+ W(\Phi)_{|_{\theta^2}} + W^\star(\Phi^\dagger)_{|_{\bar{\theta}^2}} + \mathcal{L}_{\rm soft}$$

- ‘Monte Carlo description’
  - ➡ Express superfields in terms of component fields.
  - ➡ Express everything in terms of 4-component fermions (beware of the Majoranas!).
  - ➡ Express everything in terms of mass eigenstates.
  - ➡ Integrate out D and F terms.
  - ➡ Implement vertices one-by-one (beware of factors of *i*, *etc*!)

Going Beyond SM

- The aim of this lecture is to present a code that automatizes all these steps, and allows to implement the model in MC codes starting directly from the Lagrangian.
- Workflow:
    - ➡ Define your particles and parameters.
    - ➡ Enter your Lagrangian.
    - ➡ Let the code compute the Feynman rules.
    - ➡ Output all the information in the format required by your favorite MC code.

Plan of the Lecture

- A quick overview of FeynRules
- Getting started:
  - ➡ ϕ^4 theory
  - ➡ Adding gauge interactions (scalar QCD)
- Towards LHC phenomenology: Extending the SM
- Time permitting: Some advanced topics

N.B.: Tutorial this afternoon!

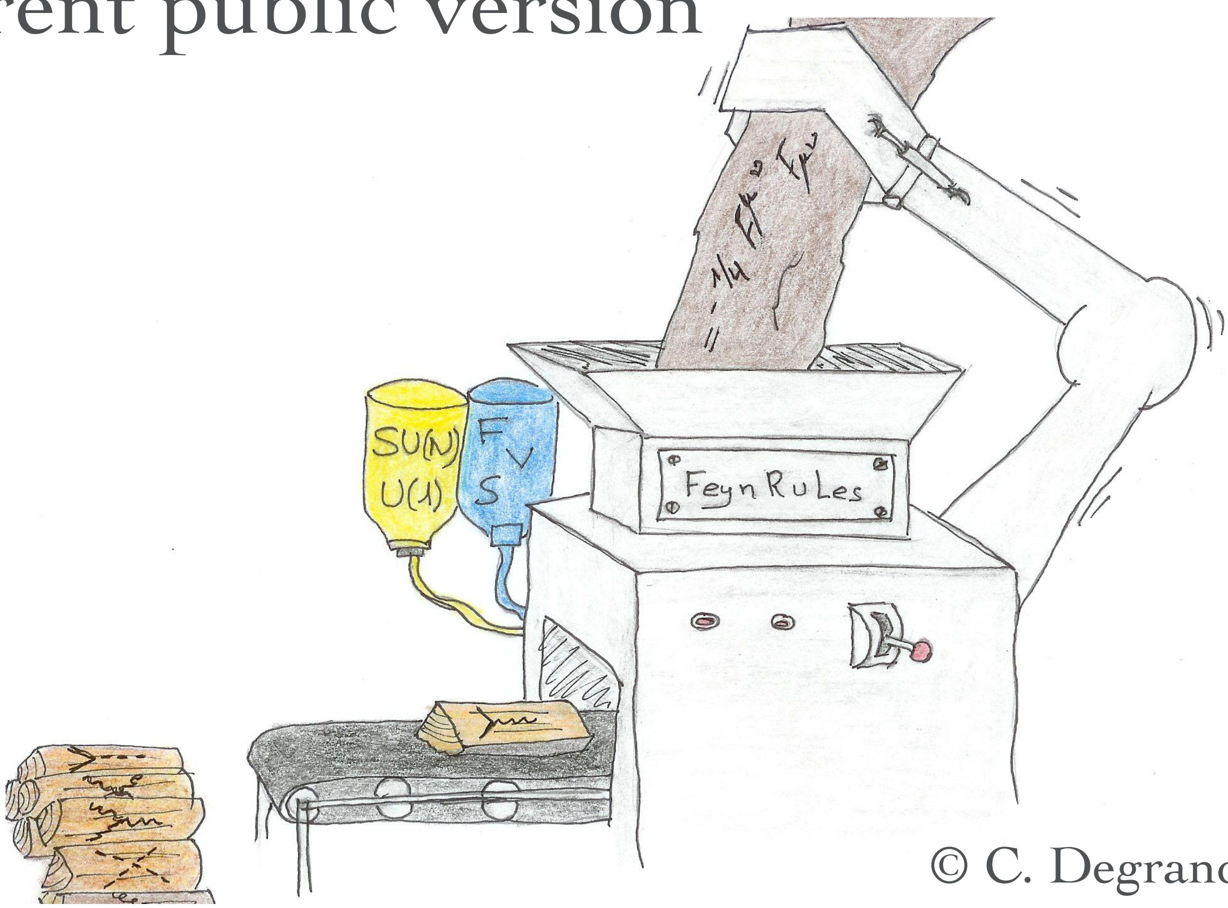
FeynRules: a quick overview

- FeynRules is a Mathematica package that allows to derive Feynman rules from a Lagrangian.

- The only requirements on the Lagrangian are:
  - ➡ All indices need to be contracted (Lorentz and gauge invariance)
  - ➡ Locality
  - ➡ Supported field types: spin 0, 1/2, 1, 2 & ghosts

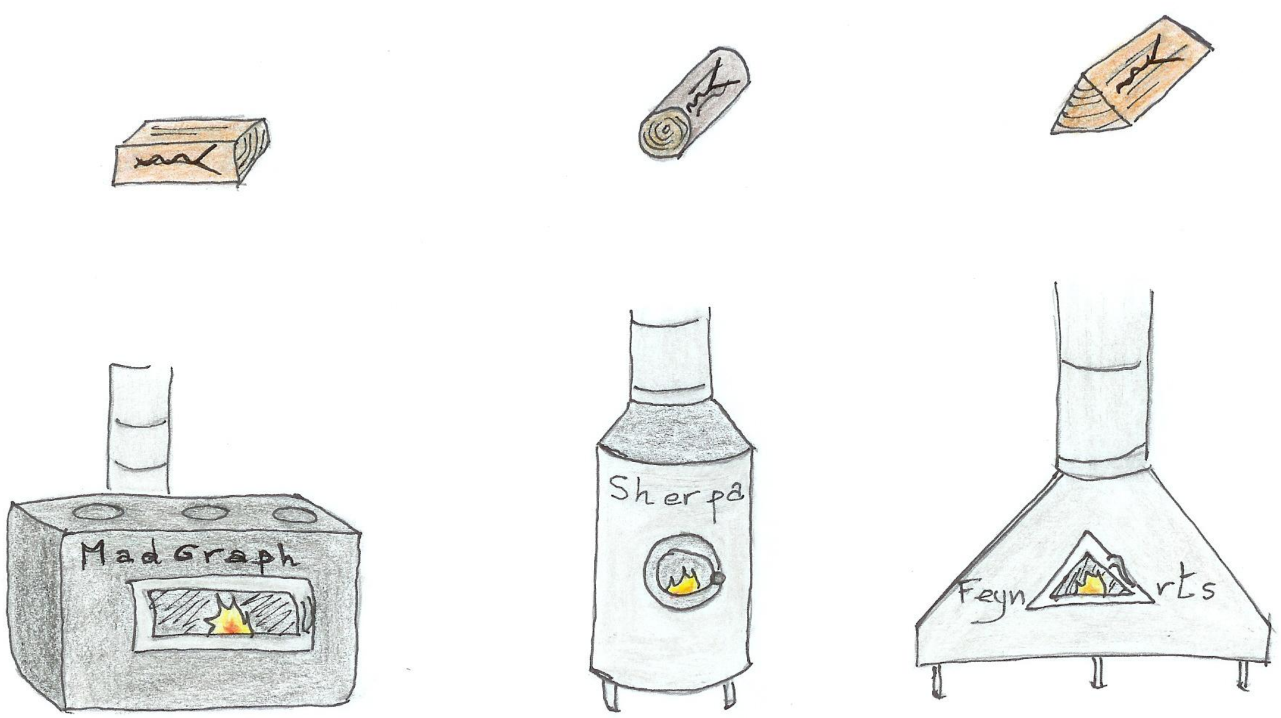
FeynRules: a quick overview

- FeynRules comes with a set of interfaces, that allow to export the Feynman rules to various matrix element generators.
- Interfaces coming with current public version
  - ➡ CalcHep / CompHep
  - ➡ FeynArts / FormCalc
  - ➡ MadGraph 4 & 5
  - ➡ Sherpa
  - ➡ Whizard / Omega



FeynRules: a quick overview

- FeynRules comes with a set of interfaces, that allow to export the Feynman rules to various matrix element generators.
- Interfaces coming with current public version:
  - ➡ CalcHep / CompHep
  - ➡ FeynArts / FormCalc
  - ➡ MadGraph 4 & 5
  - ➡ Sherpa
  - ➡ Whizard / Omega



FeynRules: a quick overview

- The input requested form the user is twofold.

- The Model File:
  Definitions of particles and parameters (e.g., a quark)

```
F[1]  ==
  {ClassName      ->   q,
   SelfConjugate ->  False,
   Indices           ->  {Index[Colour]},
   Mass              ->  {MQ,  200},
   Width             ->  {WQ, 5}   }
```

- The Lagrangian:

$$\mathcal{L} = -\frac{1}{4} G^a_{\mu\nu}\, G^{\mu\nu}_a + i\bar{q}\,\gamma^\mu\, D_\mu q - M_q\,\bar{q}\,q$$

```
L =
-1/4 FS[G,mu,nu,a] FS[G,mu,nu,a]
+ I qbar.Ga[mu].del[q,mu]
- MQ qbar.q
```

FeynRules: a quick overview

- Once this information has been provided, FeynRules can be used to compute the Feynman rules for the model:

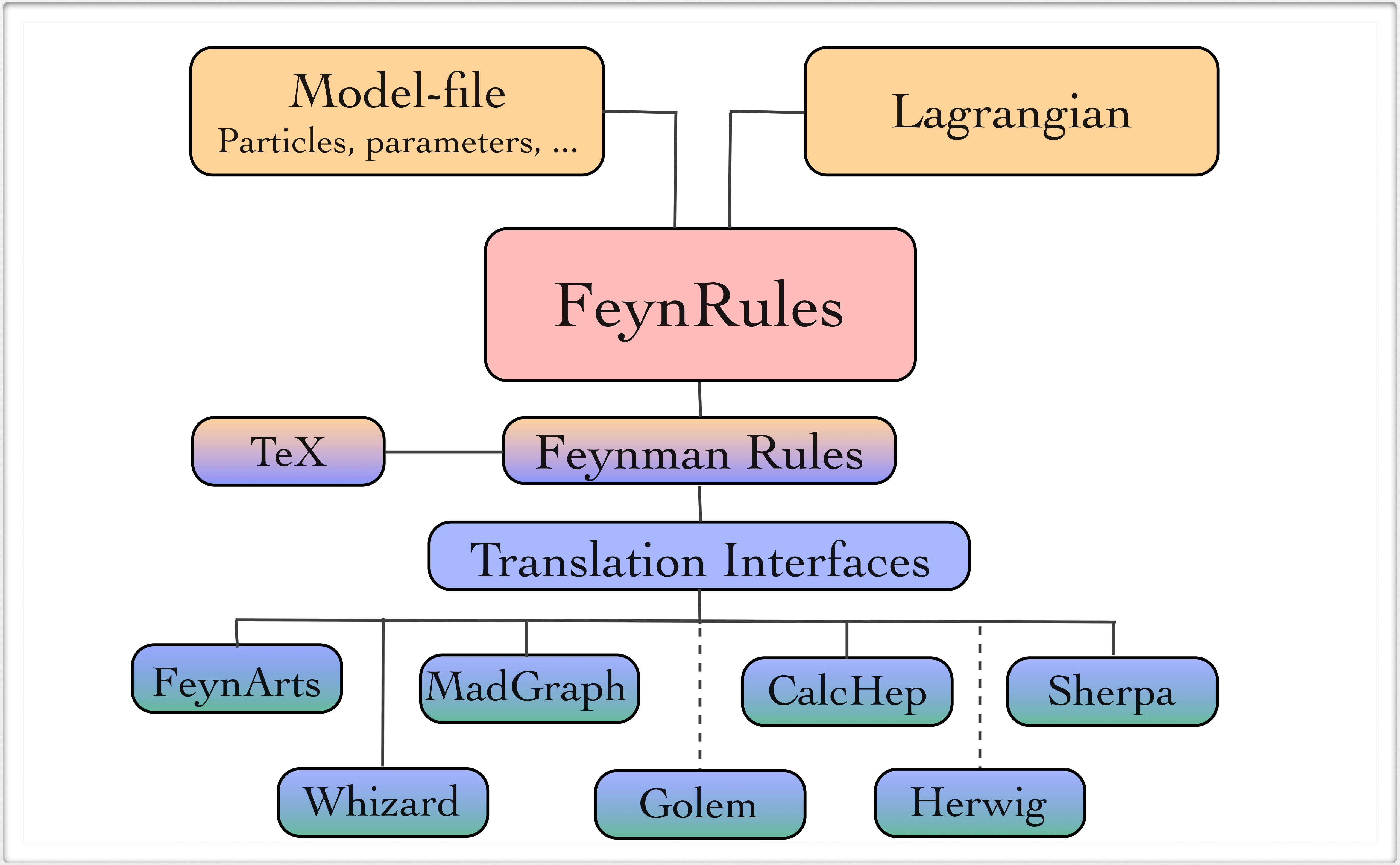
FeynmanRules[L]

- Equivalently, we can export the Feynman rules to a matrix element generator, e.g., for MadGraph 4,

WriteMGOutput[L]

- This produces a set of files that can be directly used in the matrix element generator ("plug 'n' play").

FeynRules: a quick overview



Getting Started: phi4 theory

phi4 theory

- Let us consider a model consisting of two complex scalar fields, interacting with each other:

$$\mathcal{L} = \partial_\mu \phi_i^\dagger \partial^\mu \phi_i - m^2 \phi_i^\dagger \phi_i + \lambda (\phi_i^\dagger \phi_i)^2$$

- We need to implement into a FeynRules model file
  - ➡ The two fields ϕ_1 and ϕ_2, or rather one field carrying an index.
  - ➡ The two new parameters m and λ .
- In a second step, we need to implement the Lagrangian into Mathematica.

How to write a model file

- A model file is simply a text file (with extension *.fr*).
- The syntax is Mathematica.
- General structure:

Preamble

(Author info, model info, index definitions, ...)

Particle Declarations

(Particle class definitions, spins, quantum numbers, ...)

Parameter Declarations

(Numerical Values, ...)

Preamble of the model file

- The preamble allows to 'personalize' the model file, and define all the indices that are carried by the fields
  - ➡ In our case we have one index, taking the values 1 or 2.

```
M$ModelName =  "Phi_4_Theory";


M$Information = {Authors -> {"C. Duhr"},
                 Version -> "1.0",
                 Date -> "09. 09. 2011"};


IndexRange[ Index[Scalar] ] = Range[2];
IndexStyle[ Scalar, i];
```

Particle Declaration

- Particles are defined as 'classes', grouping together particles with similar quantum numbers, but different masses (~multiplet).

```
M$ClassesDescription = {
   S[1] == {
         ClassName -> phi,
         ClassMembers -> {phi1,phi2},
         SelfConjugate -> False,
         Indices -> {Index[Scalar]},
         FlavorIndex -> Scalar,
         Mass -> {MS, 100}
     }
};
```

Particle Declaration

- Particles are defined as ‘classes’, grouping together particles with similar quantum numbers, but different masses (~multiplet).

```
M$ClassesDescription = {
  S[1] == {
          ClassName -> phi,
          ClassMembers -> {phi1,phi2},
          SelfConjugate -> False,
          Indices -> {Index[Scalar]},
          FlavorIndex -> Scalar,
          Mass -> {MS, 100}
      }
};
```

Spin (S, F, V, U, T)

Particle Declaration

- Particles are defined as 'classes', grouping together particles with similar quantum numbers, but different masses (~multiplet).

```
M$ClassesDescription = {
   S[1] == {
         ClassName -> phi,
         ClassMembers -> {phi1,phi2},
         SelfConjugate -> False,
         Indices -> {Index[Scalar]},
         FlavorIndex -> Scalar,
         Mass -> {MS, 100}
      }
};
```

Symbol used for the particle in the Lagrangian. Antiparticle called `phibar`.

Particle Declaration

- Particles are defined as 'classes', grouping together particles with similar quantum numbers, but different masses (~multiplet).

```
M$ClassesDescription = {
   S[1] == {
          ClassName -> phi,
          ClassMembers -> {phi1,phi2},
          SelfConjugate -> False,
          Indices -> {Index[Scalar]},
          FlavorIndex -> Scalar,
          Mass -> {MS, 100}
      }
};
```

The field is complex, i.e., there is an antiparticle.

Particle Declaration

- Particles are defined as 'classes', grouping together particles with similar quantum numbers, but different masses (~multiplet).

```
M$ClassesDescription = {
   S[1] == {
         ClassName -> phi,
         ClassMembers -> {phi1,phi2},
         SelfConjugate -> False,
         Indices -> {Index[Scalar]},
         FlavorIndex -> Scalar,
         Mass -> {MS, 100}
      }
};
```

Symbol for the mass used in the Lagrangian, + numerical value in GeV.

Parameter Declaration

- Parameter classes are defined in a similar way to the particle classes.
  - ➡ In our case, we have two parameters, the mass m and the coupling λ .
  - ➡ The mass was already defined with the particle, no need to define it a second time.

```
M$Parameters = {
   lam == {
          Value -> 0.1
      }
};
```

The Mathematica session

- We now run FeynRules to obtain the Feynman rules of the model
    - ➡ This is done in a Mathematica notebook.

- Step 1: Load FeynRules into Mathematica

```
In[1]:= $FeynRulesPath = SetDirectory["~/FeynRules-SVN/feynrules-current"];

In[2]:= << FeynRules`
```

The Mathematica session

- We now run FeynRules to obtain the Feynman rules of the model
  - ➡ This is done in a Mathematica notebook.
- Step 1: Load FeynRules into Mathematica

```
In[1]:= $FeynRulesPath = SetDirectory["~/FeynRules-SVN/feynrules-current"];

In[2]:= << FeynRules`

   – FeynRules –

  Authors: C. Duhr, N. Christensen, B. Fuks


  Please cite: Comput.Phys.Commun.180:1614–1641,2009 (arXiv:0806.4194).

  http://feynrules.phys.ucl.ac.be
```

The Mathematica session

- Step 2: Load the model file

```
In[3]:= SetDirectory["~/FeynRules-SVN/trunk/models/Phi_4_Theory"];

In[4]:= LoadModel["Phi_4_Theory.fr"]
```

The Mathematica session

- Step 2: Load the model file

```
In[3]:= SetDirectory["~/FeynRules-SVN/trunk/models/Phi_4_Theory"];

In[4]:= LoadModel["Phi_4_Theory.fr"]
```

This model implementation was created by

C. Duhr

Model Version: 1.0

For more information, type ModelInformation[].

The Mathematica session

- **Step 3:** Enter the Lagrangian

$$\mathcal{L} = \partial_\mu \phi_i^\dagger \partial^\mu \phi_i - m^2 \phi_i^\dagger \phi_i + \lambda(\phi_i^\dagger \phi_i)^2$$

```
In[5]:= L = del[phibar[i], mu] del[phi[i], mu] - MS^2 phibar[i] phi[i] +
          lam (phibar[i] phi[i]) (phibar[j] phi[j])
```

Out[5]= $\text{lam}\,\text{phi}_i\,\text{phi}_j\,\text{phi}_i^\dagger\,\text{phi}_j^\dagger + \text{MS}^2\,(-\text{phi}_i)\,\text{phi}_i^\dagger + \partial_{\text{mu}}(\text{phi}_i)\,\partial_{\text{mu}}(\text{phi}_i^\dagger)$

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

The Mathematica session

- Step 4: Computing the Feynman rules

In[6]:= `FeynmanRules[L]`

Starting Feynman rule calculation.

Collecting the different structures that enter the vertex...

Found 1 possible non zero vertices.

Start calculating vertices...

1 vertex obtained.

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

(* *)

Vertex 1

Particle 1 : Scalar , phi

Particle 2 : Scalar , phi

Particle 3 : Scalar , $\text{phi}^\dagger$

Particle 4 : Scalar , $\text{phi}^\dagger$

Vertex:

$$2\, i\, \text{lam}\, \delta_{i_1,i_4}\, \delta_{i_2,i_3} + 2\, i\, \text{lam}\, \delta_{i_1,i_3}\, \delta_{i_2,i_4}$$

(* *)

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

(* *)

Vertex 1

Particle 1 : Scalar , phi

Particle 2 : Scalar , phi

Particle 3 : Scalar , $\text{phi}^\dagger$

Particle 4 : Scalar , $\text{phi}^\dagger$

Vertex:

$$2\, i\, \text{lam}\, \delta_{i_1,i_4}\, \delta_{i_2,i_3} + 2\, i\, \text{lam}\, \delta_{i_1,i_3}\, \delta_{i_2,i_4}$$

(* *)

Feynman rule for the particle class!

The Mathematica session

- Step 4: Computing the Feynman rules

```
In[7]:= FeynmanRules[L, FlavorExpand → True]
```

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[7]:= FeynmanRules[L, FlavorExpand → True]
```

(* *)

Vertex 1

Particle 1 : Scalar , phi1

Particle 2 : Scalar , phi1

Particle 3 : Scalar , $\text{phi1}^\dagger$

Particle 4 : Scalar , $\text{phi1}^\dagger$

Vertex:

$4\, i\, \text{lam}$

(* *)

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[7]:= FeynmanRules[L, FlavorExpand → True]
```

(* *)

Vertex 2

Particle 1 : Scalar , phi1

Particle 2 : Scalar , $\text{phi1}^\dagger$

Particle 3 : Scalar , phi2

Particle 4 : Scalar , $\text{phi2}^\dagger$

Vertex:

$2\, i\, \text{lam}$

(* *)

The Mathematica session

- Step 4: Computing the Feynman rules

```
In[7]:= FeynmanRules[L, FlavorExpand → True]
```

(* *)

Vertex 3

Particle 1 : Scalar , phi2

Particle 2 : Scalar , phi2

Particle 3 : Scalar , $\text{phi2}^\dagger$

Particle 4 : Scalar , $\text{phi2}^\dagger$

Vertex:

$4\, i\, \text{lam}$

(* *)

Summary

- We have now fully implemented our model, and have obtained the Feynman rules.
- We also have all the information to implement the model into a matrix element generator.
- This can be done automatically using the FeynRules interfaces.
  - ➡ Will discuss this a bit later.
  - ➡ Let's first learn a bit more how to implement models.

Getting Started: Gauging our model

Gauging phi4 theory

- Let us gauge our model, say the scalar is in the adjoint of SU(3) (QCD octet).
- The change in the Lagrangian is very minor:
  - ➡ add field strength tensor
  - ➡ replace derivative by covariant derivative.

$$\mathcal{L} = -\frac{1}{4} F^a_{\mu\nu} F^{\mu\nu}_a + D_\mu \phi_i^\dagger D^\mu \phi_i - m^2 \phi_i^\dagger \phi_i + \lambda (\phi_i^\dagger \phi_i)^2$$

$$D_\mu = \partial_\mu - i g_s T^a G^a_\mu$$

- Technically speaking, we just added two new objects to our model:
  - ➡ a new particle: the gluon *G*.
  - ➡ a new parameter: the gauge coupling g_s.

Preamble of the model file

- The fields now carry an index in the adjoint index.
  - ➡ Need to define this new index in the preamble.

```
M$ModelName = "Phi_4_Theory_Octet";

M$Information = {Authors -> {"C. Duhr"},
                 Version -> "1.0",
                 Date -> "09. 09. 2011"};

IndexRange[ Index[Scalar] ] = Range[2];
IndexStyle[ Scalar, i];
IndexRange[ Index[Gluon] ] = Range[8];
IndexStyle[ Gluon, a];
```

Particle Declaration

- The scalar is now an octet.

```
M$ClassesDescription = {
   S[1] == {
         ClassName -> phi,
         ClassMembers -> {phi1,phi2},
         SelfConjugate -> False,
         Indices -> {Index[Scalar], Index[Gluon]},
         FlavorIndex -> Scalar,
         Mass -> {MS, 100}
      }
};
```

Particle Declaration

- We also need to define the gluon field.

```
M$ClassesDescription = {
   S[1] == {...},

   V[1] == {
         ClassName -> G,
         SelfConjugate -> True,
         Indices -> {Index[Gluon]},
         Mass -> 0
      }
};
```

Parameter Declaration

- We also need to define the gauge coupling.

```
M$Parameters = {
   lam == {
           Value -> 0.1
      },

   gs == {
           Value -> 1.22
      }
};
```

Gauge groups

- We have now defined the gauge coupling and the gauge boson.
- To gauge the theory we need however more:
  - ➡ Structure constants.
  - ➡ Representation matrices.
  - ➡ ...
- FeynRules allows to define gauge group classes in a similar way to particle and parameter classes.

Gauge groups

- FeynRules allows to define gauge group classes in a similar way to particle and parameter classes.

```
M$GaugeGroups = {

  SU3C == {
        Abelian -> False,
        GaugeBoson -> G,
        StructureConstant -> f,
        CouplingConstant -> gs
      }
}
```

- Could add other representations via
  `Representation -> {T, Colour}`

The Mathematica session

- Step 1: Load FeynRules into Mathematica
- Step 2: Load the model file
- Step 3: Enter the Lagrangian

$$\mathcal{L} = -\frac{1}{4} F^a_{\mu\nu} F_a^{\mu\nu} + D_\mu \phi_i^\dagger D^\mu \phi_i - m^2 \phi_i^\dagger \phi_i + \lambda (\phi_i^\dagger \phi_i)^2$$

```
In[9]:= L = -1 / 4 FS[G, mu, nu, a] FS[G, mu, nu, a] +
    DC[phibar[i, a], mu] DC[phi[i, a], mu] - MS ^2 phibar[i, a] phi[i, a] +
    lam (phibar[i, a] phi[i, a]) (phibar[j, b] phi[j, b])
```

Out[9]=

$$\left(\partial_{\mathrm{mu}}(\mathrm{phi}_{i,a}) - i\, \mathrm{gs}\, G_{\mathrm{mu,a\$979}}\, \mathrm{phi}_{i,\mathrm{i\$979}}\, \mathrm{FSU3C}^{\mathrm{a\$979}}_{a,\mathrm{i\$979}}\right)\left(\partial_{\mathrm{mu}}(\mathrm{phi}^{\dagger}_{i,a}) + i\, \mathrm{gs}\, G_{\mathrm{mu,a\$978}}\, \mathrm{FSU3C}^{\mathrm{a\$978}}_{\mathrm{i\$978},a}\, \mathrm{phi}^{\dagger}_{i,\mathrm{i\$978}}\right) + \mathrm{lam}\, \mathrm{phi}_{i,a}\, \mathrm{phi}_{j,b}\, \mathrm{phi}^{\dagger}_{i,a}\, \mathrm{phi}^{\dagger}_{j,b} - \frac{1}{4}\left(\mathrm{gs}\, G_{\mathrm{mu,bb\$976}}\, G_{\mathrm{nu,cc\$976}}\, f_{a,\mathrm{bb\$976},\mathrm{cc\$976}} - \partial_{\mathrm{nu}}(G_{\mathrm{mu},a}) + \partial_{\mathrm{mu}}(G_{\mathrm{nu},a})\right)\left(\mathrm{gs}\, G_{\mathrm{mu,bb\$977}}\, G_{\mathrm{nu,cc\$977}}\, f_{a,\mathrm{bb\$977},\mathrm{cc\$977}} - \partial_{\mathrm{nu}}(G_{\mathrm{mu},a}) + \partial_{\mathrm{mu}}(G_{\mathrm{nu},a})\right) + \mathrm{MS}^2\,(-\mathrm{phi}_{i,a})\,\mathrm{phi}^{\dagger}_{i,a}$$

The Mathematica session

- Step 4: Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

(* *)

Vertex 1

Particle 1 : Vector , G

Particle 2 : Vector , G

Particle 3 : Vector , G

Vertex:

$$\text{gs}\, p_1^{\mu_3} f_{a_1,a_2,a_3} \eta_{\mu_1,\mu_2} - \text{gs}\, p_2^{\mu_3} f_{a_1,a_2,a_3} \eta_{\mu_1,\mu_2} - \text{gs}\, p_1^{\mu_2} f_{a_1,a_2,a_3} \eta_{\mu_1,\mu_3} + \text{gs}\, p_3^{\mu_2} f_{a_1,a_2,a_3} \eta_{\mu_1,\mu_3} + \text{gs}\, p_2^{\mu_1} f_{a_1,a_2,a_3} \eta_{\mu_2,\mu_3} - \text{gs}\, p_3^{\mu_1} f_{a_1,a_2,a_3} \eta_{\mu_2,\mu_3}$$

(* *)

The Mathematica session

- **Step 4:** Computing the Feynman rules

In[6]:= `FeynmanRules[L]`

(* *)

Vertex 2

Particle 1 : Vector , G

Particle 2 : Vector , G

Particle 3 : Vector , G

Particle 4 : Vector , G

Vertex:

$$i\,\mathrm{gs}^2\,\eta_{\mu_1,\mu_4}\,\eta_{\mu_2,\mu_3}\,f_{a_1,a_3,\mathrm{al}}\,f_{a_2,a_4,\mathrm{al}} + i\,\mathrm{gs}^2\,\eta_{\mu_1,\mu_4}\,\eta_{\mu_2,\mu_3}\,f_{a_1,a_2,\mathrm{al}}\,f_{a_3,a_4,\mathrm{al}} + i\,\mathrm{gs}^2\,\eta_{\mu_1,\mu_3}\,\eta_{\mu_2,\mu_4}\,f_{a_1,a_4,\mathrm{al}}\,f_{a_2,a_3,\mathrm{al}} - i\,\mathrm{gs}^2\,\eta_{\mu_1,\mu_3}\,\eta_{\mu_2,\mu_4}\,f_{a_1,a_2,\mathrm{al}}\,f_{a_3,a_4,\mathrm{al}} - i\,\mathrm{gs}^2\,\eta_{\mu_1,\mu_2}\,\eta_{\mu_3,\mu_4}\,f_{a_1,a_4,\mathrm{al}}\,f_{a_2,a_3,\mathrm{al}} - i\,\mathrm{gs}^2\,\eta_{\mu_1,\mu_2}\,\eta_{\mu_3,\mu_4}\,f_{a_1,a_3,\mathrm{al}}\,f_{a_2,a_4,\mathrm{al}}$$

(* *)

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

(* *)

Vertex 3

Particle 1 : Vector , G

Particle 2 : Scalar , phi

Particle 3 : Scalar , $\mathrm{phi}^{\dagger}$

Vertex:

$$\mathrm{gs}\,\mathrm{p}_3^{\mu_1}\, f_{\mathrm{a}_3,\mathrm{a}_1,\mathrm{a}_2}\, \delta_{\mathrm{i}_2,\mathrm{i}_3} - \mathrm{gs}\,\mathrm{p}_2^{\mu_1}\, f_{\mathrm{a}_3,\mathrm{a}_1,\mathrm{a}_2}\, \delta_{\mathrm{i}_2,\mathrm{i}_3}$$

(* *)

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

(* *)

Vertex 4

Particle 1 : Vector , G

Particle 2 : Vector , G

Particle 3 : Scalar , phi

Particle 4 : Scalar , $\text{phi}^\dagger$

Vertex:

$$i\,\text{gs}^2\,\eta_{\mu_1,\mu_2}\,\delta_{\text{i}_3,\text{i}_4}\,f_{\text{a}_1,\text{a}_4,\text{al}}\,f_{\text{a}_2,\text{a}_3,\text{al}} + i\,\text{gs}^2\,\eta_{\mu_1,\mu_2}\,\delta_{\text{i}_3,\text{i}_4}\,f_{\text{a}_1,\text{a}_3,\text{al}}\,f_{\text{a}_2,\text{a}_4,\text{al}}$$

(* *)

The Mathematica session

- **Step 4:** Computing the Feynman rules

```
In[6]:= FeynmanRules[L]
```

(* *)

Vertex 5

Particle 1 : Scalar , phi

Particle 2 : Scalar , phi

Particle 3 : Scalar , $\text{phi}^\dagger$

Particle 4 : Scalar , $\text{phi}^\dagger$

Vertex:

$$2\, i\, \text{lam}\, \delta_{a_1,a_4}\, \delta_{a_2,a_3}\, \delta_{i_1,i_4}\, \delta_{i_2,i_3} + 2\, i\, \text{lam}\, \delta_{a_1,a_3}\, \delta_{a_2,a_4}\, \delta_{i_1,i_3}\, \delta_{i_2,i_4}$$

(* *)

Towards LHC phenomenology: Extending the SM

Extending the SM

- So far we have only considered our model standalone.
- For LHC phenomenology, one usually wants a BSM model that is an extension of the SM.
- FeynRules offers the possibility to start form the SM model, and to add/change/remove particles and operators.
- For this, it is enough to load our new model together with the SM implementation:

```
LoadModel[ “SM.fr“, “Phi_4_Gauged“ ];
```

N.B.: In the SM implementation, the gluon and the QCD gauge group are already defined, so no need to redefine them.

Running Interfaces

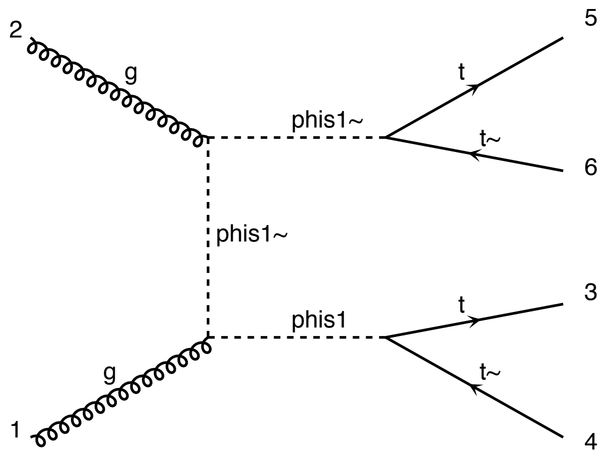
- We are now ready to do phenomenology!
- FeynRules contains interfaces to the following codes:
  - ➡ CalcHep / CompHep
  - ➡ FeynArts / FormCalc
  - ➡ MadGraph 4 & 5
  - ➡ Sherpa
  - ➡ Whizard / Omega
- Each interface produces a set of text files that can be read into the existing generators.

Running Interfaces

- The interfaces are called via the Mathematica commands

```
WriteCHOutput[ LSM, L ];              (* CalcHep *)
WriteFeynArtsOutput[ LSM, L ];        (* FeynArts/FormCalc *)
WriteMGOutput[ LSM, L ];              (*MadGraph 4 *)
WriteUFO[ LSM, L ];                   (* UFO / MadGraph 5 *)
WriteSHOutput[ LSM, L ];              (* Sherpa *)
WriteWOOutput[ LSM, L];               (* Whizard / Omega *)
```

- The files produced by FeynRules can then be processed by the matrix element generators.



Running Interfaces

- Some interfaces require/admit additional options that were not discussed.
- E.g., the SM input parameters should be named following some conventions that assure that, e.g., the strong coupling is recognized as such by the generator.
- Some interfaces to some generators have the colour and / or Lorentz structures hardwired:

	Spins	Lorentz	Colour
CalcHep	0,1/2,1,2	~all	1,3,8 (limited)
FeynArts	0,1/2,1	all	all*
MadGraph 4	0,1/2,1	MSSM - like	1,3,8 (limited)
MadGraph 5	0,1/2,1,2	all	1,3,6,8
Sherpa	0,1/2,1	SM - like	1,3,8
Whizard	0,1/2,1,2	MSSM - like	1,3,8

Other available models

- The same procedure can be used to extend any other models.
- Many models can be downloaded from the FeynRules web page, and can serve as a start to implement new models (http://feynrules.irmp.ucl.ac.be/).
  - ➡ SM (+ extensions: 4th generation, diquarks, See-saw...).
  - ➡ MSSM, NMSSM, RPV-MSSM.
  - ➡ Extra dimensions: UED, LED, Higgsless, HEIDI.
  - ➡ Minimal walking Technicolor.

Advanced topics

Complicated models

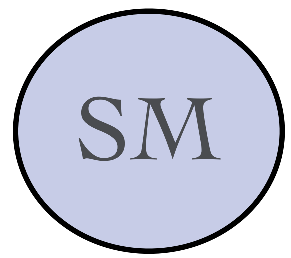
- The procedure described so far requires the Lagrangian to be written explicitly in terms of scalar, vector and 4-component fermion fields.
- For some models, this is not the most convenient way to write the Lagrangian:
  - ➡ Supersymmetric models are very compact in terms of superfields.
  - ➡ Extra-dimensional models naturally live in a D > 4 dimensional space.
- FeynRules, together with the underlying Mathematica engine, allows to write down compact Lagrangians, even for complicated models.

Model Restrictions

- A *model restriction* is a model that is obtained from a bigger model by putting some of its parameters to zero (or 1, etc.).
- Example:

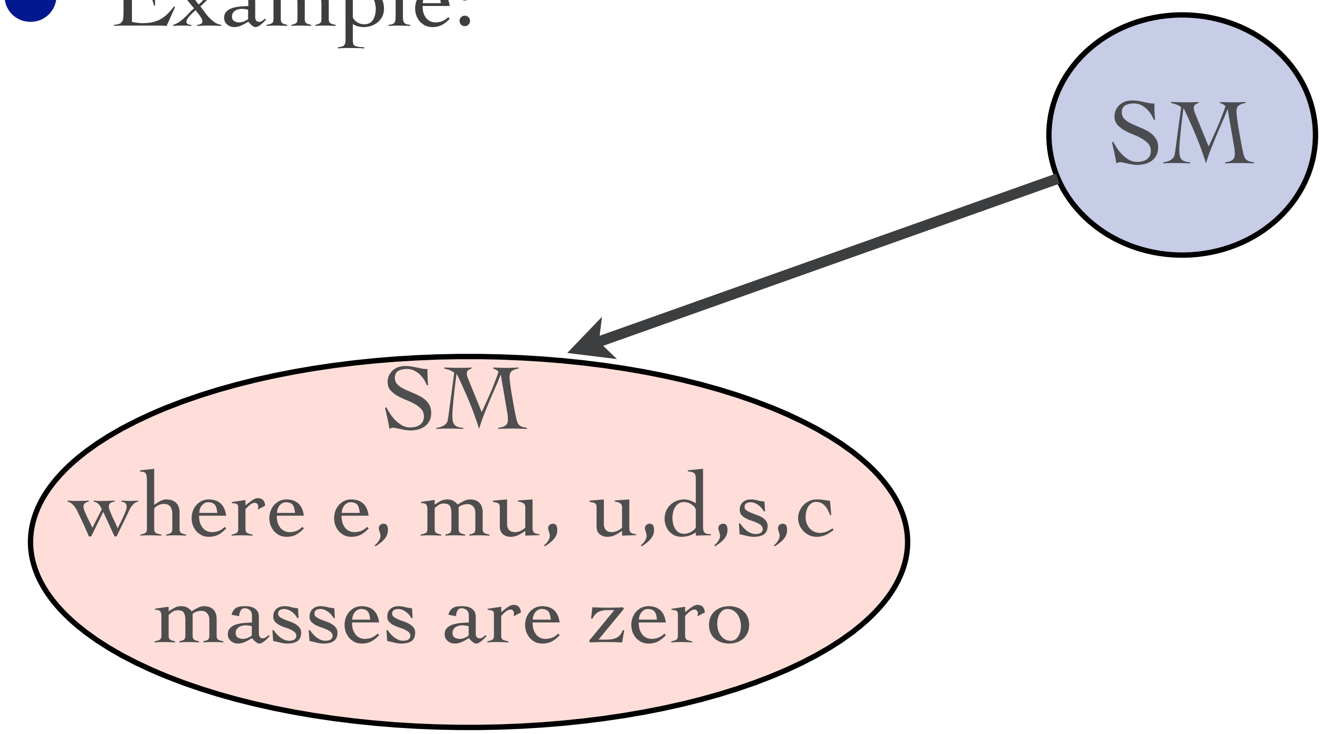
Model Restrictions

- A *model restriction* is a model that is obtained from a bigger model by putting some of its parameters to zero (or 1, etc.).
- Example:



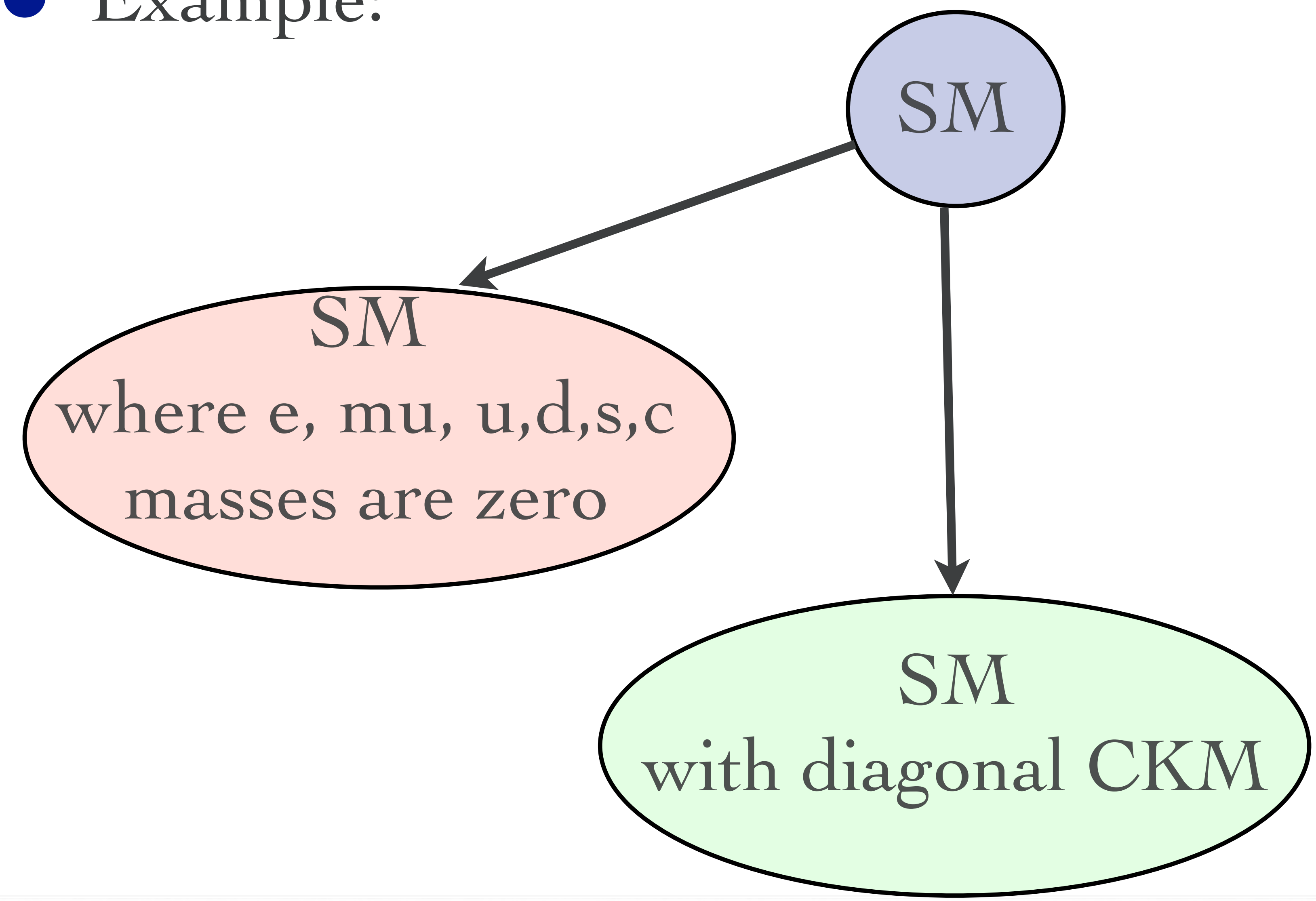
Model Restrictions

- A *model restriction* is a model that is obtained from a bigger model by putting some of its parameters to zero (or 1, etc.).
- Example:



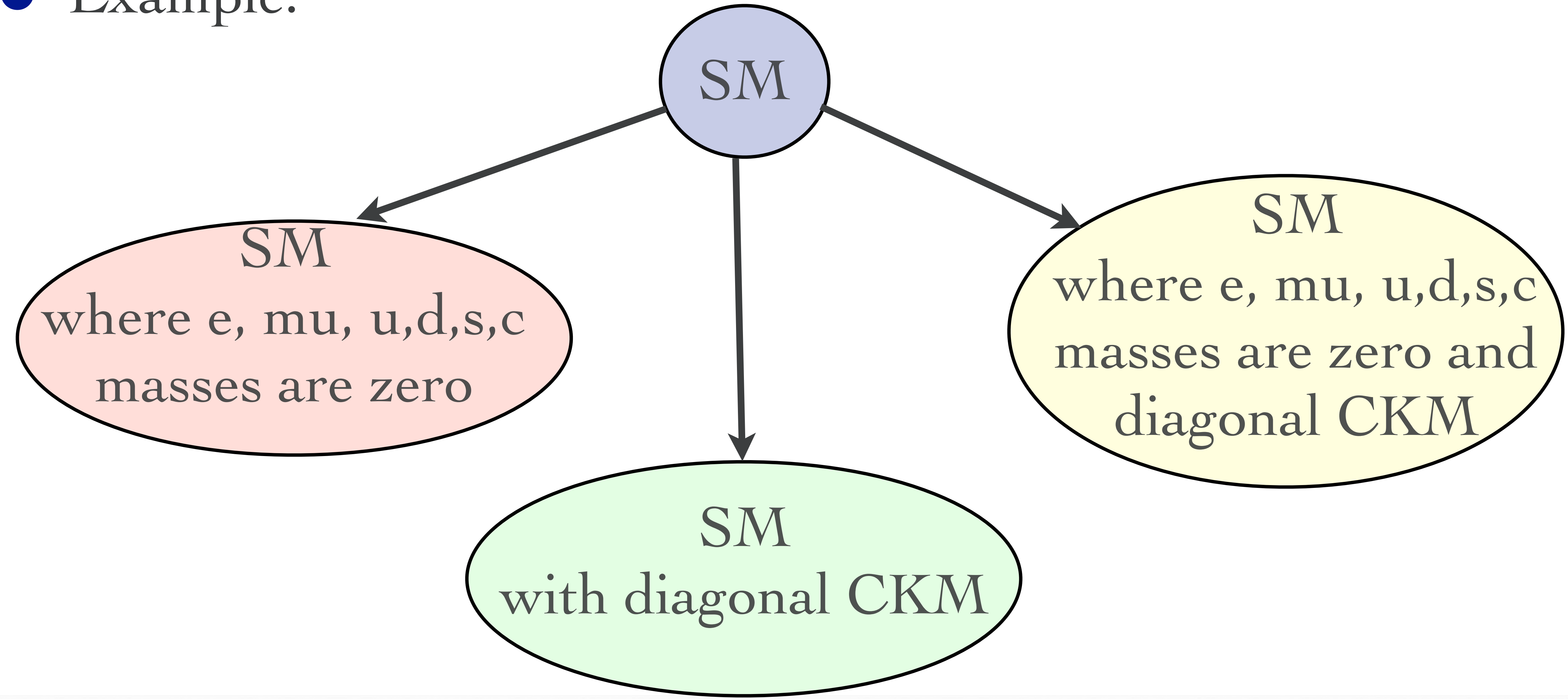
Model Restrictions

- A *model restriction* is a model that is obtained from a bigger model by putting some of its parameters to zero (or 1, etc.).
- Example:



Model Restrictions

- A *model restriction* is a model that is obtained from a bigger model by putting some of its parameters to zero (or 1, etc.).
- Example:

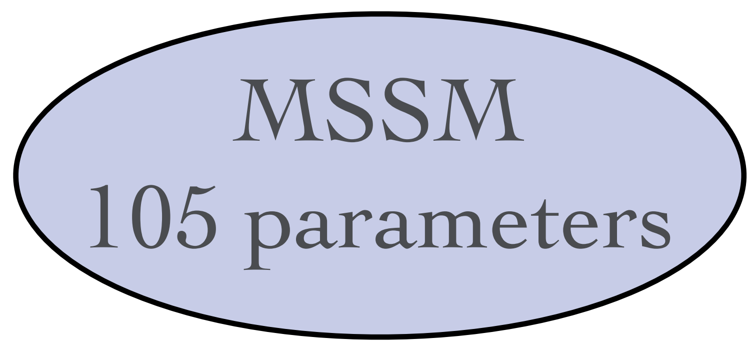


Model Restrictions

- A *model restriction* is a model that is obtained from a bigger model by putting some of its parameters to zero (or 1, etc.).
- Example:

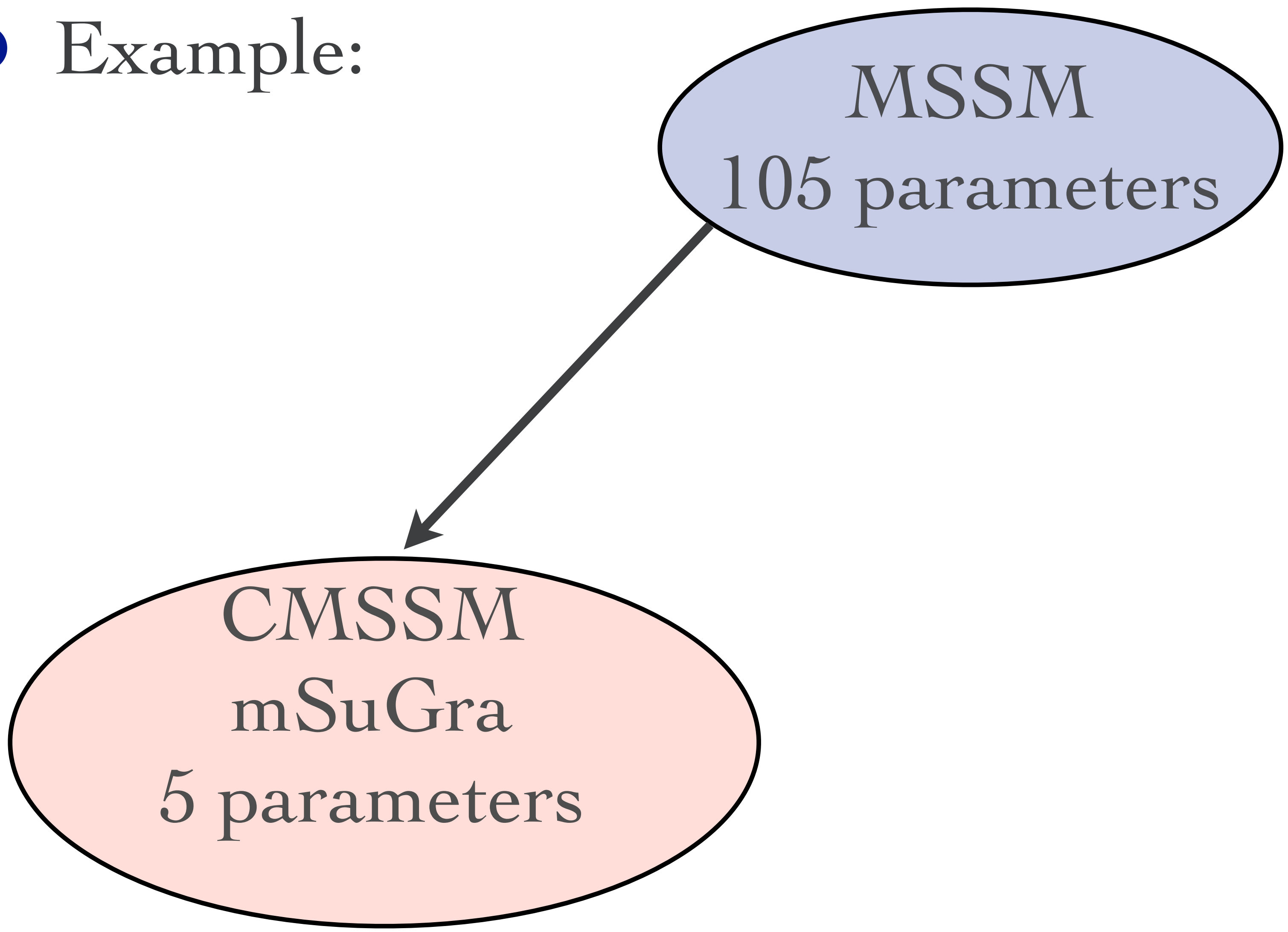
Model Restrictions

- A *model restriction* is a model that is obtained from a bigger model by putting some of its parameters to zero (or 1, etc.).
- Example:



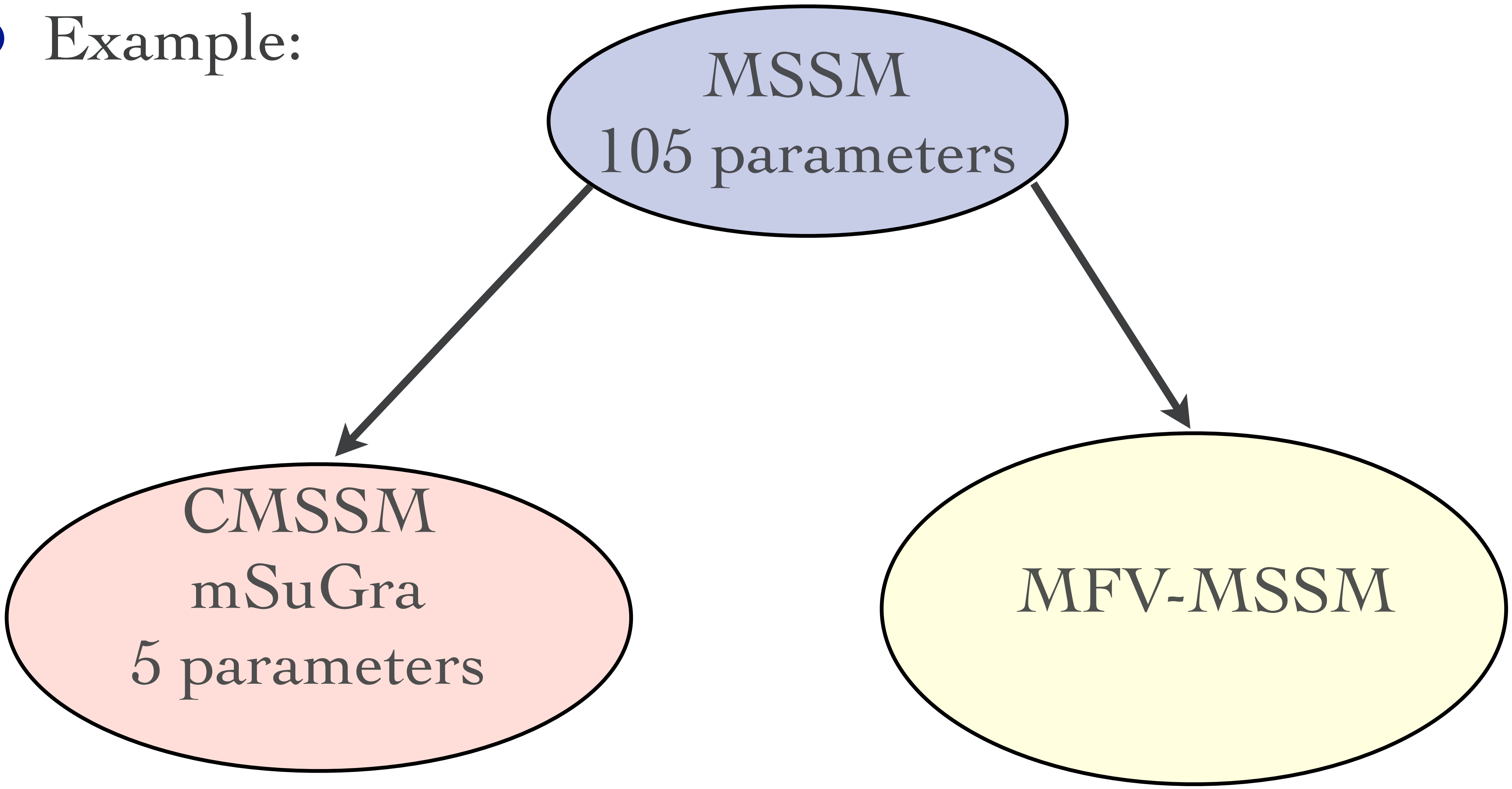
Model Restrictions

- A *model restriction* is a model that is obtained from a bigger model by putting some of its parameters to zero (or 1, etc.).
- Example:



Model Restrictions

- A *model restriction* is a model that is obtained from a bigger model by putting some of its parameters to zero (or 1, etc.).
- Example:



Model Restrictions

- In phenomenological applications one generally does not need the *full* model, but only a subset.

- Keeping the full model is ok, but it might make the MC unnecessarily slow.
  - ➡ Example: for generic CKM, lots of flavor-violating vertices, that lead to diagrams that are numerically subleading.

- We want a way to get rid of the 'undesired' vertices!

Model Restrictions

- Restriction files allow to achieve this by using simple Mathematica replacement rules.

```
M$Restrictions = {
        CKM[i_,i_] -> 1,
        CKM[i_?NumericQ, j_?NumericQ] :> 0 /; (i =!= j),
}
```

- If one or more restrictions are loaded after loading a model file, the corresponding replacement rules are applied at runtime when computing the vertices.

```
LoadRestriction[ “DiagonalCKM.rst“ ];
```

Supersymmetric models

- FeynRules allows to use the superfield formalism for supersymmetric theories.
- The code then
  - ➡ expands the superfields in the Grassmann variables and integrates them out.
  - ➡ Weyl fermions are transformed into 4-component spinors.
  - ➡ auxiliary fields are integrated out.
- As a result, we obtain a Lagrangian that can be exported to matrix element generators!

Supersymmetric models

- Example: SUSY QCD
  - ➡ 1 octet vector superfield $V^a = (\tilde{g}^a, G^a_\mu, D^a)$
  - ➡ 1 triplet left-handed chiral superfield $Q^i_L = (\tilde{q}^i_L, \chi^i, F^i_L)$
  - ➡ 1 triplet right-handed chiral superfield $Q^i_R = (\tilde{q}^i_R, \bar{\xi}^i, F^i_R)$
- The physical spectrum contains
  - ➡ a gauge boson, the gluon
  - ➡ two complex triplet scalars
  - ➡ an octet Majorana fermion
  - ➡ a triplet Dirac fermion, the quark $q^i = (\chi^i, \bar{\xi}^i)$

Supersymmetric models

- Interactions (almost) entirely fixed by SUSY

$$Q_L^\dagger\, e^{-2g_sV}\, Q_L + Q_R^\dagger\, e^{-2g_sV}\, Q_R + \frac{1}{8g_s^2}\mathrm{Tr}(W^\alpha W_\alpha) + \frac{1}{8g_s^2}\mathrm{Tr}(\overline{W}_{\dot\alpha}\overline{W}^{\dot\alpha})$$
$$+W(Q_L, Q_R^\dagger) + W^\star(Q_L^\dagger, Q_R)$$

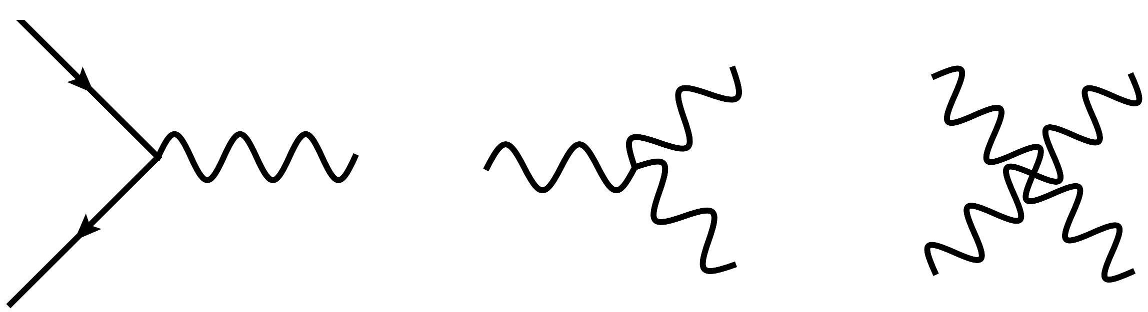
- The gauge sector is already rather complicated in terms of component fields...

Supersymmetric models

- Interactions (almost) entirely fixed by SUSY

$$Q_L^\dagger\, e^{-2g_s V}\, Q_L + Q_R^\dagger\, e^{-2g_s V}\, Q_R + \frac{1}{8g_s^2}\mathrm{Tr}(W^\alpha W_\alpha) + \frac{1}{8g_s^2}\mathrm{Tr}(\overline{W}_{\dot\alpha}\overline{W}^{\dot\alpha})$$
$$+W(Q_L, Q_R^\dagger) + W^\star(Q_L^\dagger, Q_R)$$

- The gauge sector is already rather complicated in terms of component fields...

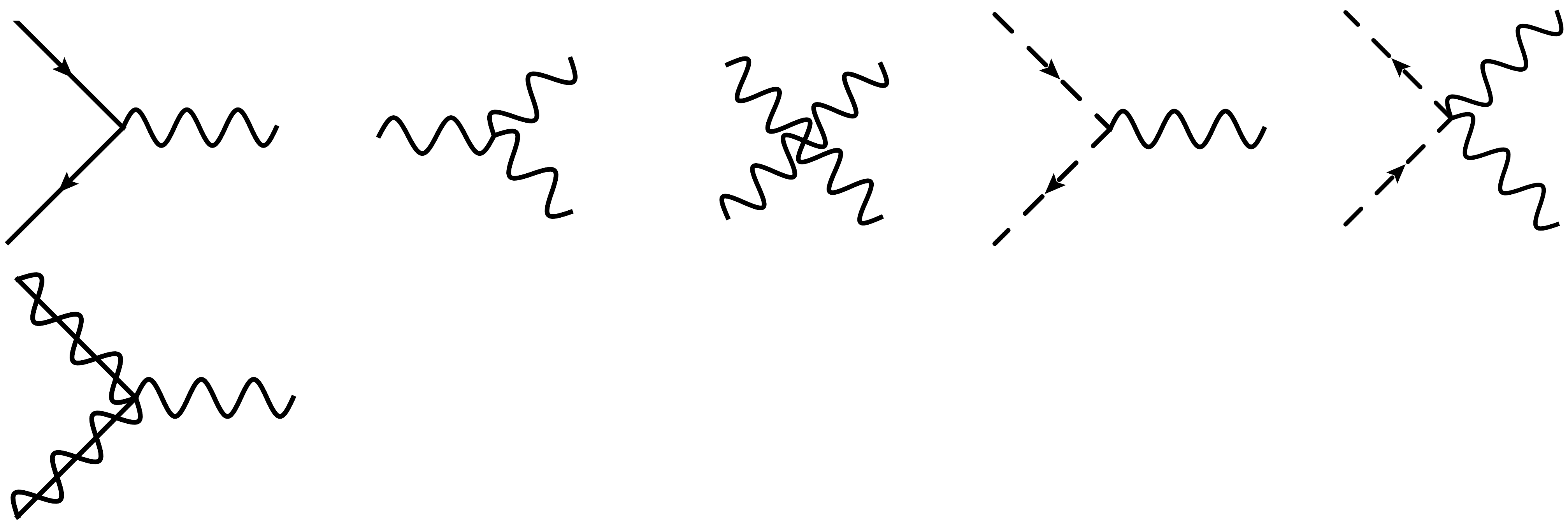


Supersymmetric models

- Interactions (almost) entirely fixed by SUSY

$$Q_L^\dagger\, e^{-2g_s V}\, Q_L + Q_R^\dagger\, e^{-2g_s V}\, Q_R + \frac{1}{8g_s^2}\mathrm{Tr}(W^\alpha W_\alpha) + \frac{1}{8g_s^2}\mathrm{Tr}(\overline{W}_{\dot\alpha}\overline{W}^{\dot\alpha})$$
$$+W(Q_L, Q_R^\dagger) + W^\star(Q_L^\dagger, Q_R)$$

- The gauge sector is already rather complicated in terms of component fields...

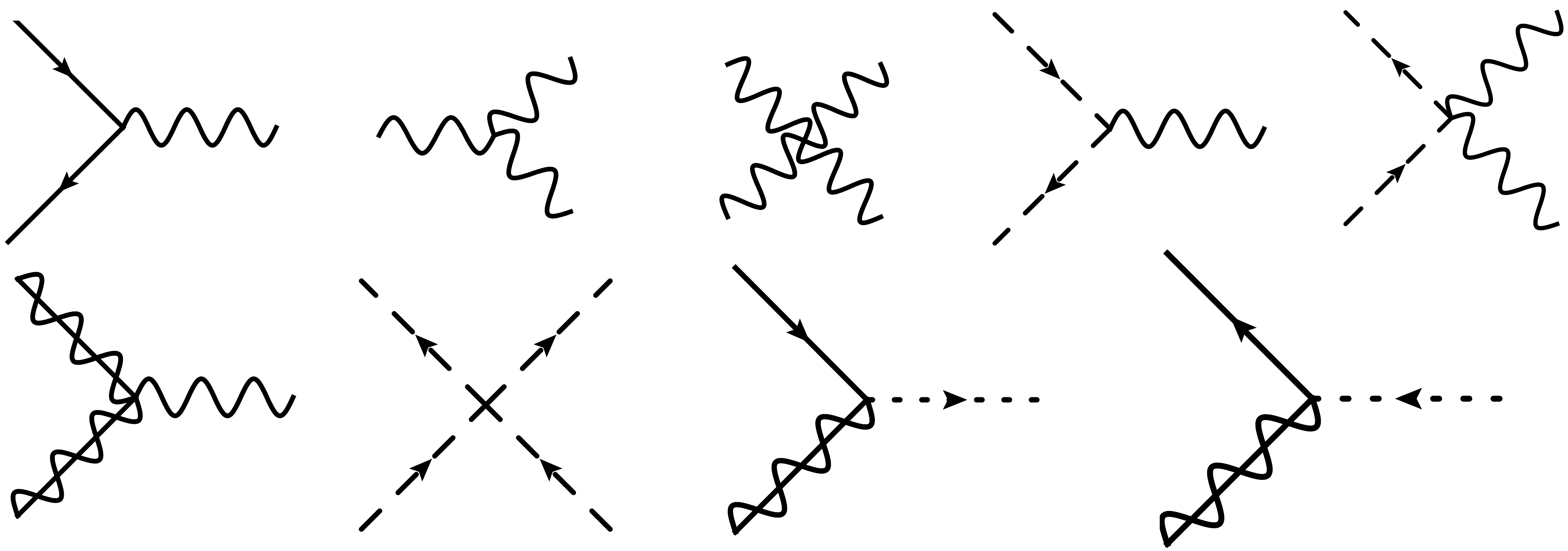


Supersymmetric models

- Interactions (almost) entirely fixed by SUSY

$$Q_L^\dagger\, e^{-2g_s V}\, Q_L + Q_R^\dagger\, e^{-2g_s V}\, Q_R + \frac{1}{8g_s^2}\mathrm{Tr}(W^\alpha W_\alpha) + \frac{1}{8g_s^2}\mathrm{Tr}(\overline{W}_{\dot\alpha}\overline{W}^{\dot\alpha})$$
$$+W(Q_L, Q_R^\dagger) + W^\star(Q_L^\dagger, Q_R)$$

- The gauge sector is already rather complicated in terms of component fields...



Defining superfields

```
VSF[1] == { ClassName   -> GSF,
            GaugeBoson  -> G,
            Gaugino     -> gow,
            Indices     -> {Index[Gluon]}},

CSF[1] == { ClassName    -> QL,
             Chirality   -> Left,
             Weyl        -> qLw,
             Scalar      -> QLs,
             Indices->{Index[Colour]}},
```

$$V^a = (\tilde{g}^a, G^a_\mu, D^a)$$

$$Q^i_L = (\tilde{q}^i_L, \chi^i, F^i_L)$$

- The component fields are defined separately.
- Auxiliary F and D fields could be added, but can be left out, and are created on the fly.

Using superfields

```
WS = ...
SL = VSFKineticTerms[] + CSFKineticTerms[] + WS + HC[WS];
```

- A set of functions allows to transform the superspace action into a component field Lagrangian.
  - ➡ SF2Components: expansion in the Grassmann parameters
  - ➡ ThetaThetabarComponent etc.: selects the desired coefficient in the Grassmann expansion.
  - ➡ SolveEqMotionF/SolveEqMotionD: solves the equations of motion for the F and D terms.
  - ➡ WeylToDirac: Transforms Weyl fermions into 4-component fermions.

Using superfields

$$-4 f_{Gluon\$1,Gluon\$2,Gluon\$3} f_{Gluon\$3,Gluon\$4,Gluon\$5} G_{mu\$1,Gluon\$1} G_{mu\$1,Gluon\$4} G_{mu\$2,Gluon\$2} G_{mu\$2,Gluon\$5} gs^4 +$$

$$16 \partial_{mu\$2}(G_{mu\$1,Gluon\$1}) f_{Gluon\$1,Gluon\$2,Gluon\$3} G_{mu\$1,Gluon\$2} G_{mu\$2,Gluon\$3} gs^3 + 8 i \bar{go}_{r\$28685,Gluon\$2} \cdot go_{r\$28698,Gluon\$1} f_{Gluon\$1,Gluon\$2,Gluon\$3} G_{mu\$1,Gluon\$3} \gamma^{mu\$1} P_{-,r\$28685,r\$28698} gs^3 -$$

$$8 i \bar{go}_{r\$28698,Gluon\$1} \cdot go_{r\$28685,Gluon\$2} f_{Gluon\$1,Gluon\$2,Gluon\$3} G_{mu\$1,Gluon\$3} \gamma^{mu\$1} P_{+,r\$28698,r\$28685} gs^3 - 8 \partial_{mu\$2}(G_{mu\$1,Gluon\$1})^2 gs^2 +$$

$$8 \partial_{mu\$2}(G_{mu\$1,Gluon\$1}) \partial_{mu\$1}(G_{mu\$2,Gluon\$1}) gs^2 + G_{mu\$1,Gluon\$1} G_{mu\$1,Gluon\$2} QLs_{Colour\$1} QLs^{\dagger}_{Colour\$2} T^{Gluon\$1}_{Colour\$2,Colour\$3} T^{Gluon\$2}_{Colour\$3,Colour\$1} gs^2 +$$

$$G_{mu\$1,Gluon\$1} G_{mu\$1,Gluon\$2} QRs_{Colour\$1} QRs^{\dagger}_{Colour\$2} T^{Gluon\$1}_{Colour\$2,Colour\$3} T^{Gluon\$2}_{Colour\$3,Colour\$1} gs^2 + 4 i \bar{go}_{r\$28679,Gluon\$1} \cdot \partial_{mu\$1}(go_{r\$28692,Gluon\$1}) \gamma^{mu\$1} P_{-,r\$28679,r\$28692} gs^2 -$$

$$4 i \partial_{mu\$1}(\bar{go}_{r\$28682,Gluon\$1}) \cdot go_{r\$28695,Gluon\$1} \gamma^{mu\$1} P_{-,r\$28682,r\$28695} gs^2 - 4 i \partial_{mu\$1}(\bar{go}_{r\$28692,Gluon\$1}) \cdot go_{r\$28679,Gluon\$1} \gamma^{mu\$1} P_{+,r\$28692,r\$28679} gs^2 +$$

$$4 i \bar{go}_{r\$28695,Gluon\$1} \cdot \partial_{mu\$1}(go_{r\$28682,Gluon\$1}) \gamma^{mu\$1} P_{+,r\$28695,r\$28682} gs^2 - i \partial_{mu\$1}(QLs^{\dagger}_{Colour\$1}) G_{mu\$1,Gluon\$1} QLs_{Colour\$2} T^{Gluon\$1}_{Colour\$1,Colour\$2} gs +$$

$$i \sqrt{2}\, \bar{q}_{r\$28675,Colour\$1} \cdot go_{r\$28688,Gluon\$1} P_{+,r\$28675,r\$28688} QLs_{Colour\$2} T^{Gluon\$1}_{Colour\$1,Colour\$2} gs + i \partial_{mu\$1}(QRs^{\dagger}_{Colour\$1}) G_{mu\$1,Gluon\$1} QRs_{Colour\$2} T^{Gluon\$1}_{Colour\$1,Colour\$2} gs -$$

$$i \sqrt{2}\, \bar{q}_{r\$28689,Colour\$1} \cdot go_{r\$28676,Gluon\$1} P_{-,r\$28689,r\$28676} QRs_{Colour\$2} T^{Gluon\$1}_{Colour\$1,Colour\$2} gs + i \partial_{mu\$1}(QLs_{Colour\$1}) G_{mu\$1,Gluon\$1} QLs^{\dagger}_{Colour\$2} T^{Gluon\$1}_{Colour\$2,Colour\$1} gs -$$

$$i \sqrt{2}\, \bar{go}_{r\$28677,Gluon\$1} \cdot q_{r\$28690,Colour\$1} P_{-,r\$28677,r\$28690} QLs^{\dagger}_{Colour\$2} T^{Gluon\$1}_{Colour\$2,Colour\$1} gs - i \partial_{mu\$1}(QRs_{Colour\$1}) G_{mu\$1,Gluon\$1} QRs^{\dagger}_{Colour\$2} T^{Gluon\$1}_{Colour\$2,Colour\$1} gs +$$

$$i \sqrt{2}\, \bar{go}_{r\$28691,Gluon\$1} \cdot q_{r\$28678,Colour\$1} P_{+,r\$28691,r\$28678} QRs^{\dagger}_{Colour\$2} T^{Gluon\$1}_{Colour\$2,Colour\$1} gs + \bar{q}_{r\$28687,Colour\$2} \cdot q_{r\$28700,Colour\$1} G_{mu\$1,Gluon\$1} T^{Gluon\$1}_{Colour\$2,Colour\$1} \gamma^{mu\$1} P_{-,r\$28687,r\$28700} gs -$$

$$\bar{q}_{r\$28699,Colour\$1} \cdot q_{r\$28686,Colour\$2} G_{mu\$1,Gluon\$1} T^{Gluon\$1}_{Colour\$1,Colour\$2} \gamma^{mu\$1} P_{+,r\$28699,r\$28686} gs + \frac{1}{2} \partial_{mu\$1}(QLs_{Colour\$1}) \partial_{mu\$1}(QLs^{\dagger}_{Colour\$1}) +$$

$$\frac{1}{2} \partial_{mu\$1}(QRs_{Colour\$1}) \partial_{mu\$1}(QRs^{\dagger}_{Colour\$1}) - \frac{1}{4} \partial_{mu\$1}(\partial_{mu\$1}(QLs^{\dagger}_{Colour\$1})) QLs_{Colour\$1} - \frac{1}{4} \partial_{mu\$1}(\partial_{mu\$1}(QLs_{Colour\$1})) QLs^{\dagger}_{Colour\$1} - \frac{1}{4} \partial_{mu\$1}(\partial_{mu\$1}(QRs^{\dagger}_{Colour\$1})) QRs_{Colour\$1} -$$

$$\frac{1}{4} \partial_{mu\$1}(\partial_{mu\$1}(QRs_{Colour\$1})) QRs^{\dagger}_{Colour\$1} + \frac{1}{32} QLs_{Colour\$1\$28637} QLs_{Colour\$1\$28639} QLs^{\dagger}_{Colour\$2\$28637} QLs^{\dagger}_{Colour\$2\$28639} T^{Gluon\$1}_{Colour\$2\$28637,Colour\$1\$28637} T^{Gluon\$1}_{Colour\$2\$28639,Colour\$1\$28639} +$$

$$\frac{1}{32} QLs_{Colour\$1\$28639} QLs^{\dagger}_{Colour\$2\$28639} QRs_{Colour\$1\$28638} QRs^{\dagger}_{Colour\$2\$28638} T^{Gluon\$1}_{Colour\$2\$28638,Colour\$1\$28638} T^{Gluon\$1}_{Colour\$2\$28639,Colour\$1\$28639} +$$

$$\frac{1}{32} QLs_{Colour\$1\$28637} QLs^{\dagger}_{Colour\$2\$28637} QRs_{Colour\$1\$28640} QRs^{\dagger}_{Colour\$2\$28640} T^{Gluon\$1}_{Colour\$2\$28637,Colour\$1\$28637} T^{Gluon\$1}_{Colour\$2\$28640,Colour\$1\$28640} +$$

$$\frac{1}{32} QRs_{Colour\$1\$28638} QRs_{Colour\$1\$28640} QRs^{\dagger}_{Colour\$2\$28638} QRs^{\dagger}_{Colour\$2\$28640} T^{Gluon\$1}_{Colour\$2\$28638,Colour\$1\$28638} T^{Gluon\$1}_{Colour\$2\$28640,Colour\$1\$28640} -$$

$$\frac{1}{16} QLs_{Colour\$1} QLs_{Colour\$1\$28641} QLs^{\dagger}_{Colour\$2} QLs^{\dagger}_{Colour\$2\$28641} T^{Gluon\$1}_{Colour\$2,Colour\$1} T^{Gluon\$1}_{Colour\$2\$28641,Colour\$1\$28641} -$$

$$\frac{1}{16} QLs_{Colour\$1} QLs^{\dagger}_{Colour\$2} QRs_{Colour\$1\$28642} QRs^{\dagger}_{Colour\$2\$28642} T^{Gluon\$1}_{Colour\$2,Colour\$1} T^{Gluon\$1}_{Colour\$2\$28642,Colour\$1\$28642} -$$

$$\frac{1}{16} QLs_{Colour\$1\$28643} QLs^{\dagger}_{Colour\$2\$28643} QRs_{Colour\$1} QRs^{\dagger}_{Colour\$2} T^{Gluon\$1}_{Colour\$2,Colour\$1} T^{Gluon\$1}_{Colour\$2\$28643,Colour\$1\$28643} -$$

$$\frac{1}{16} QRs_{Colour\$1} QRs_{Colour\$1\$28644} QRs^{\dagger}_{Colour\$2} QRs^{\dagger}_{Colour\$2\$28644} T^{Gluon\$1}_{Colour\$2,Colour\$1} T^{Gluon\$1}_{Colour\$2\$28644,Colour\$1\$28644} + \frac{1}{2} i \bar{q}_{r\$28680,Colour\$1} \cdot \partial_{mu\$1}(q_{r\$28693,Colour\$1}) \gamma^{mu\$1} P_{-,r\$28680,r\$28693} -$$

$$\frac{1}{2} i \partial_{mu\$1}(\bar{q}_{r\$28683,Colour\$1}) \cdot q_{r\$28696,Colour\$1} \gamma^{mu\$1} P_{-,r\$28683,r\$28696} - \frac{1}{2} i \partial_{mu\$1}(\bar{q}_{r\$28694,Colour\$1}) \cdot q_{r\$28681,Colour\$1} \gamma^{mu\$1} P_{+,r\$28694,r\$28681} + \frac{1}{2} i \bar{q}_{r\$28697,Colour\$1} \cdot \partial_{mu\$1}(q_{r\$28684,Colour\$1}) \gamma^{mu\$1} P_{+,r\$28697,r\$28684}$$

Summary

- Implementing a New Physics into a matrix element generator can be a tedious and error-prone task.
- FeynRules tries to remedy this situation by providing a Mathematica framework where a new model can be implemented starting directly from the Lagrangian.
- There are no restrictions on the model, except
    - ➡ Lorentz and gauge invariance
    - ➡ Locality
    - ➡ Spins: 0, 1/2, 1, 2, ghosts (3/2 to come in the future)

- Try it out on your favorite model!
  http://feynrules.irmp.ucl.ac.be/