New models in WHIZARD with FeynRules

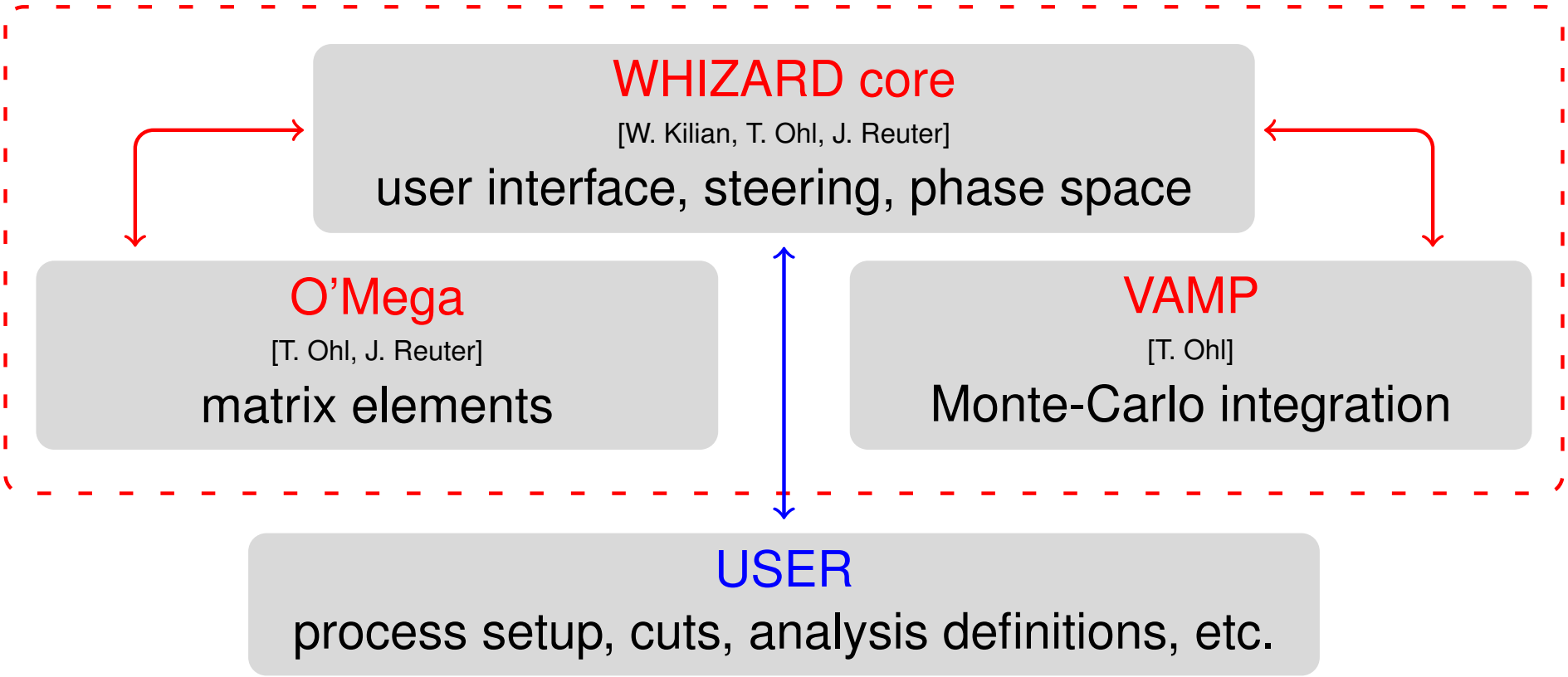
Christian Speckner

Universität Freiburg

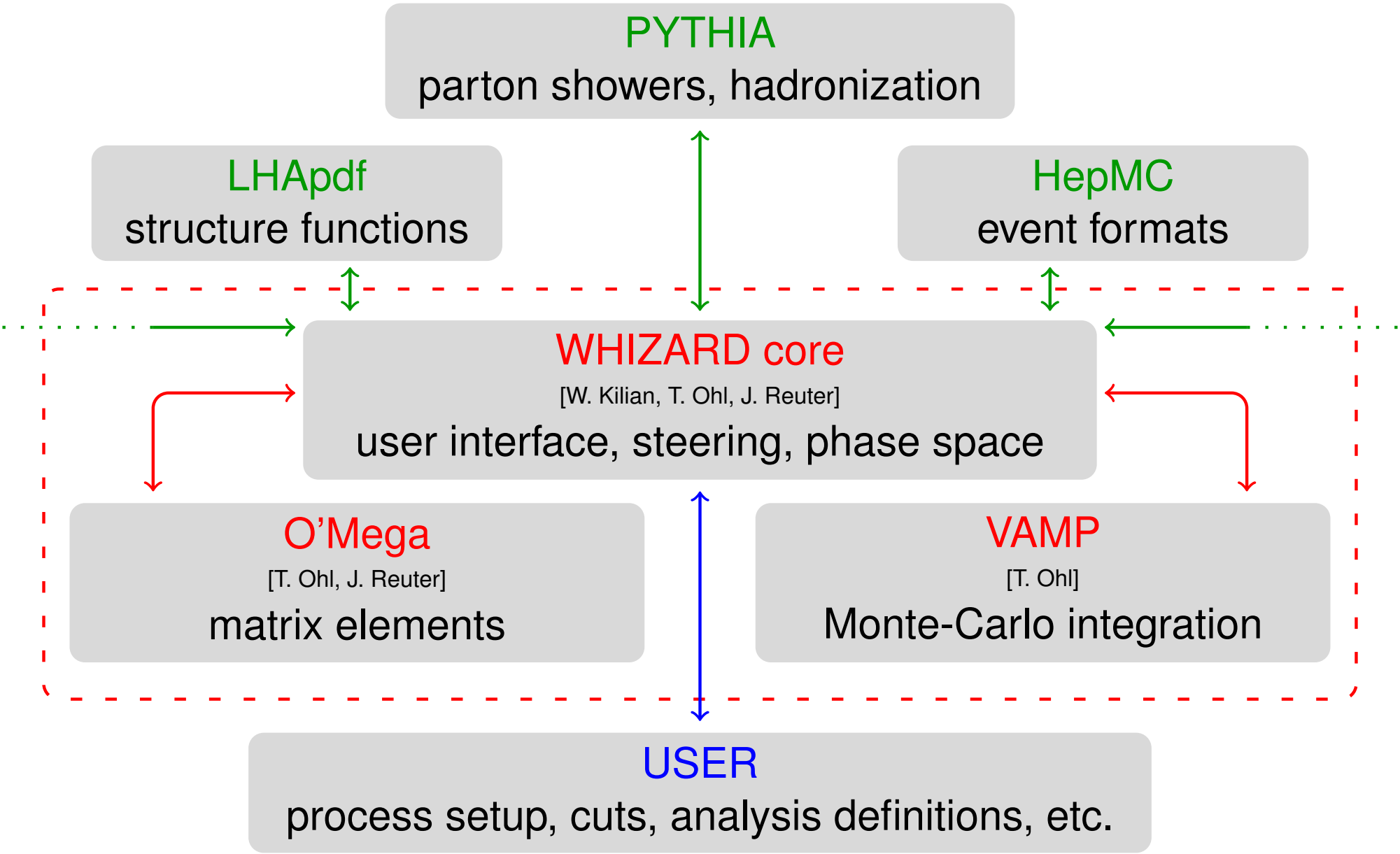
MC4BSM, April 14th, 2010

Outline:

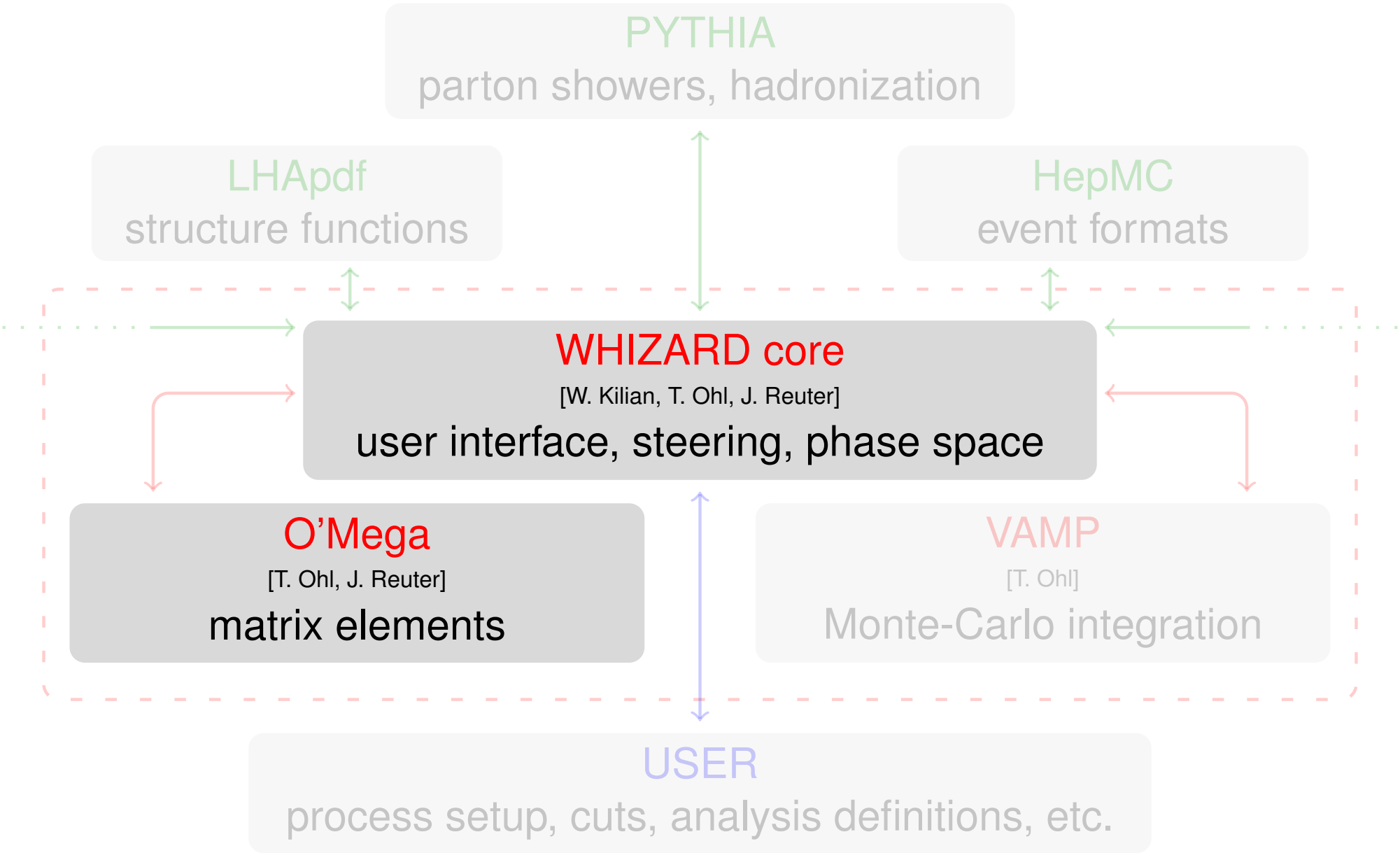
How is WHIZARD structured?



How is WHIZARD structured?



Model-dependent components:



New models in O'Mega

In O'Mega, a model is a O'Caml module

- Defining the particles in the model
- " their couplings
- Providing functions for translating particles and couplings to plain text identifiers (for code generation and user interface)
- Compiled and linked to the rest of the O'Mega framework, yielding an executable matrix element compiler

New models in O'Mega

In O'Mega, a model is a O'Caml module

- Defining the particles in the model
- " their couplings
- Providing functions for translating particles and couplings to plain text identifiers (for code generation and user interface)
- Compiled and linked to the rest of the O'Mega framework, yielding an executable matrix element compiler

Pro:

- Full power of an elegant functional programming language
- Well structured and readable model files

New models in O'Mega

In O'Mega, a model is a O'Caml module

- Defining the particles in the model
- " their couplings
- Providing functions for translating particles and couplings to plain text identifiers (for code generation and user interface)
- Compiled and linked to the rest of the O'Mega framework, yielding an executable matrix element compiler

Pro:

- Full power of an elegant functional programming language
- Well structured and readable model files

Con:

- Need to learn O'Caml
- Physics interleaved with a lot of technical infrastructure

Code examples:

The good...

Particle definitions:

```
type generation = Gen0 | Gen1 | Gen2

type csign = Pos | Neg
type isospin = Iso_up | Iso_down
type fermion =
  | Lepton of (csign × generation × isospin)
  | Quark of (csign × generation × isospin)
type boson =
  | W of csign
  | Z | A | G
type flavor = Fermion of fermion
  | Boson of boson
```

$A\bar{q}q$ type vertex definitions

```
let vertices_aqq =

let vgen kk gen iso = (
  (Fermion (Quark (Neg, gen, iso)),
   Boson A,
   Fermion (Quark (Pos, gen, iso)))
 ,
   FBF(1, Psibar, V, Psi)
 ,
   G_a_quark iso
 )
in loop_iso (loop_gen [vgen])
```

Code examples:

The good...

Particle definitions:

```
type generation = Gen0 | Gen1 | Gen2

type csign = Pos | Neg
type isospin = Iso_up | Iso_down
type fermion =
  | Lepton of (csign × generation × isospin)
  | Quark of (csign × generation × isospin)
type boson =
  | W of csign
  | Z | A | G
type flavor = Fermion of fermion
  | Boson of boson
```

$Aq\bar{q}q$ type vertex definitions

```
let vertices_aqq =

let vgen kk gen iso = (
  (Fermion (Quark (Neg, gen, iso)),
   Boson A,
   Fermion (Quark (Pos, gen, iso)))
 ,
  FBF(1, Psibar, V, Psi)
 ,
  G_a_quark iso
 )
in loop_iso (loop_gen [vgen])
```

...and the bad

Particle indentifiers:

```
let flavor_to_string =

let postfix = function
  | Fermion (Lepton (_, cs, _, Iso_down)) →
   (match cs with Pos → "-" | Neg → "+" )
  | Fermion (Quark (_, Neg, _, _))
  | Fermion (Lepton (_, Neg, _, Iso_up)) → "bar"
  | Boson (W (_, cs)) →
   (match cs with Pos → "+" | Neg → "-" )
  | _ → ""
in let rump = function
  | Fermion (Lepton desc) → (match desc with
    | (_, _, Gen0, Iso_up) → "nue"
    | (_, _, Gen0, Iso_down) → "e"
    | (_, _, Gen1, Iso_up) → "numu"
    | (_, _, Gen1, Iso_down) → "mu"
    | (_, _, Gen2, Iso_up) → "nutau"
    | (_, _, Gen2, Iso_down) → "tau" )
  | Fermion (Quark desc) → (match desc with
    | (_, _, Gen0, Iso_up) → "u"
    | (_, _, Gen0, Iso_down) → "d"
    | (_, _, Gen1, Iso_up) → "c"
    | (_, _, Gen1, Iso_down) → "s"
    | (_, _, Gen2, Iso_up) → "t"
    | (_, _, Gen2, Iso_down) → "b" )
  | Boson (W _) → "W" | Boson Z → "Z"
  | Boson A → "A" | Boson G → "gl"
in function x → (rump x) ^ (postfix x)
```

What constitutes the WHIZARD part of a model?

1. A model file (custom syntax) declaring
   - External and derived parameters

```
parameter GF    = 1.16639E-5                    # Fermi constant
derived   v     = 1 / sqrt (sqrt (2.) * GF)     # v (Higgs vev)
```

   - Particles and -properties

```
particle D_QUARK 1  parton
  spin 1/2  charge -1/3   isospin -1/2  color 3
  name d down
  anti dbar D "d~"
  tex_anti "\bar{d}"
```

   - Yet another vertex list (for phasespace parametrization)

```
vertex D d A
vertex U u A
vertex S s A
```

What constitutes the WHIZARD part of a model?

1. A model file (custom syntax) declaring
   - External and derived parameters

```
parameter GF    = 1.16639E-5                    # Fermi constant
derived   v     = 1 / sqrt (sqrt (2.) * GF)     # v (Higgs vev)
```

   - Particles and -properties

```
particle D_QUARK 1  parton
  spin 1/2  charge -1/3   isospin -1/2  color 3
  name d down
  anti dbar D "d~"
  tex_anti "\bar{d}"
```

   - Yet another vertex list (for phasespace parametrization)

```
vertex D d A
vertex U u A
vertex S s A
```

2. A piece of FORTRAN 95 glue to
   - Calculate couplings from parameters
   - Setup the environment for the matrix element evaluation code
   - Take care of updating the couplings when α_S is evolved

Consequence for the BSM phenomenologist

Implementing a new model into WHIZARD is possible (W2 even has infrastructure to accomodate third-party models without hacking or recompiling WHIZARD), but not straightforward, and programming skills are indispensable.

Consequence for the BSM phenomenologist

Implementing a new model into WHIZARD is possible (W2 even has infrastructure to accomodate third-party models without hacking or recompiling WHIZARD), but not straightforward, and programming skills are indispensable.

$\longrightarrow$ Enter the FeynRules interface!

What is FeynRules?

New physics as seen by different physicists:

- Theorist: Fields and Lagrangians
- Experimentalist: Event distributions

How to go from one to the other? Monte Carlo Generators!

What is FeynRules?

New physics as seen by different physicists:

- Theorist: Fields and Lagrangians
- Experimentalist: Event distributions

How to go from one to the other? Monte Carlo Generators!

Problem for the phenomenologist:

- There are many programs around...
- ...cross-checking results between those is highly desirable...
- ...but each has its own unique model file format
  $\longrightarrow$ implementing a new model is a tedious and error-prone task!

What is FeynRules?

New physics as seen by different physicists:

- Theorist: Fields and Lagrangians
- Experimentalist: Event distributions

How to go from one to the other? Monte Carlo Generators!

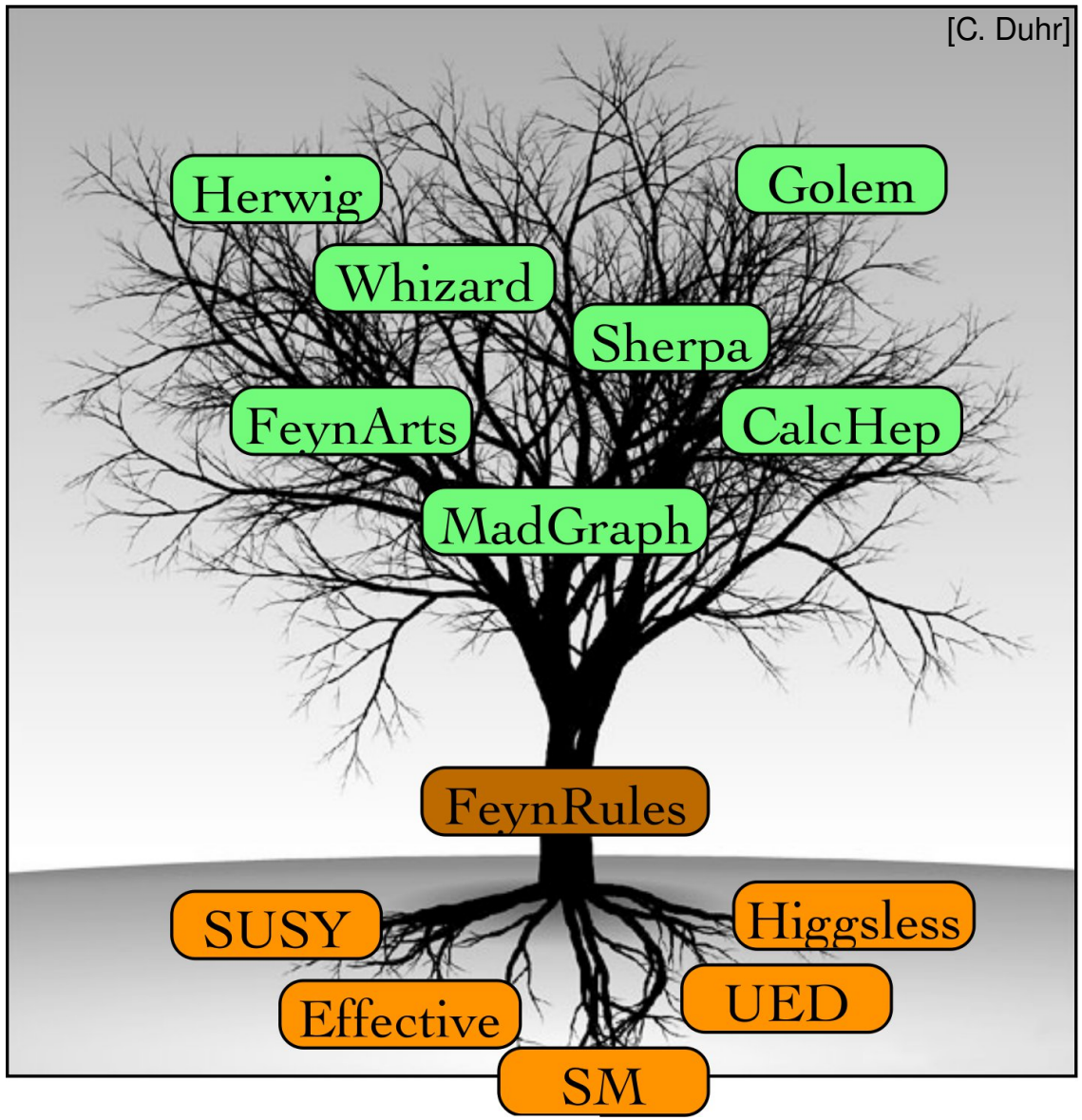
Problem for the phenomenologist:

- There are many programs around...
- ...cross-checking results between those is highly desirable...
- ...but each has its own unique model file format
  $\longrightarrow$ implementing a new model is a tedious and error-prone task!

Enter FeynRules [N. Christensen, C. Duhr, B. Fuks et al.]:

- Mathematica package, calculates Feynman rules from Lagrangian
- Output: model files for different generators
- Allows to go directly from the Lagrangian to the event generator

Available models and supported Codes:



Generating a model with FeynRules:

1. Write the model file
   - Contains particle and parameter definitions + Lagrangian
   - Syntax for particle definitions resembles FeynArts

```
F[1] == { ClassName     -> q,
          SelfConjugate -> False,
          Indices       -> {Index[Color]},
          Mass          -> {MQ, 10} }
```

   - Lagrangian $\mathcal{L}$ is entered directly

```
L = - 1/4 * FS[A, mu, nu] * FS[A, mu, nu]
    + I qbar . Ga[mu] . DC[q, mu] - MQ * qbar . q
```

Generating a model with FeynRules:

1. Write the model file
   - Contains particle and parameter definitions + Lagrangian
   - Syntax for particle definitions resembles FeynArts

```
F[1] == { ClassName     -> q,
          SelfConjugate -> False,
          Indices       -> {Index[Color]},
          Mass          -> {MQ, 10} }
```

   - Lagrangian $\mathcal{L}$ is entered directly

```
L = - 1/4 * FS[A, mu, nu] * FS[A, mu, nu]
    + I qbar . Ga[mu] . DC[q, mu] - MQ * qbar . q
```

2. Feynman rules can be calculated from $\mathcal{L}$

```
FeynmanRules[L, FlavorExpand -> True]
```

3. Generate output suitable for event generator (e.g. MadGraph)

```
WriteMGOutput[L]
```

Generating a model with FeynRules:

1. Write the model file
   - Contains particle and parameter definitions + Lagrangian
   - Syntax for particle definitions resembles FeynArts

```
F[1] == { ClassName     -> q,
          SelfConjugate -> False,
          Indices       -> {Index[Color]},
          Mass          -> {MQ, 10} }
```

   - Lagrangian $\mathcal{L}$ is entered directly

```
L = - 1/4 * FS[A, mu, nu] * FS[A, mu, nu]
    + I qbar . Ga[mu] . DC[q, mu] - MQ * qbar . q
```

2. Feynman rules can be calculated from $\mathcal{L}$

```
FeynmanRules[L, FlavorExpand -> True]
```

3. Generate output suitable for event generator (e.g. MadGraph)

```
WriteMGOutput[L]
```

(Some) features:

- Supports fields of spins 0, $\frac{1}{2}$, 1 and 2
- Can handle higher-dimensional operators
- Performs sanity checks like hermicity and charge conservation

What does it do?

- Integrates into the FeynRules framework
- Translates the FeynRules data structures
- Emits all components required a working WHIZARD model
  - O'Mega model code
  - WHIZARD model file
  - FORTRAN glue code
  - A build system for compiling and installing

What does it do?

- Integrates into the FeynRules framework
- Translates the FeynRules data structures
- Emits all components required a working WHIZARD model
  - O'Mega model code
  - WHIZARD model file
  - FORTRAN glue code
  - A build system for compiling and installing

How to get it?

- Checkout from the WHIZARD SVN:

```
svn checkout http://svn.hepforge.org/whizard/branches/speckner/FeynRulesInterface
```

- Install into FeynRules (current public version is supported) via

```
cd FeynRulesInterface; ./install.sh /path/to/FeynRules
```

- Documentation is available at

  `https://server06.fynu.ucl.ac.be/projects/feynrules/wiki/WhizardInterface`

- Will be part of the next FeynRules release

How to use it:

1. Minimal Mathematica input:

```
$FeynRulesPath = SetDirectory["."];
<<FeynRules`;
LoadModel["SM/SM.fr"];
WriteWOOutput[LSM, WOModelName->"fr_sm", Output->"WO-fr_sm"];
```

2. Run through Mathematica kernel (or type into notebook):

```
math < input.m
```

How to use it:

1. Minimal Mathematica input:

```
$FeynRulesPath = SetDirectory["."];
<<FeynRules`;
LoadModel["SM/SM.fr"];
WriteWOOutput[LSM, WOModelName->"fr_sm", Output->"WO-fr_sm"];
```

2. Run through Mathematica kernel (or type into notebook):

```
math < input.m
```

3. Compile and install the model

```
cd WO-fr_sm
./configure WO_CONFIG=/path/to/whizard/binaries
make install
```

   - Default destination: `${HOME}/.whizard`
   - Other destinations can be selected via `--prefix=...`

How to use it:

1. Minimal Mathematica input:

```
$FeynRulesPath = SetDirectory["."];
<<FeynRules`;
LoadModel["SM/SM.fr"];
WriteWOOutput[LSM, WOModelName->"fr_sm", Output->"WO-fr_sm"];
```

2. Run through Mathematica kernel (or type into notebook):

```
math < input.m
```

3. Compile and install the model

```
cd WO-fr_sm
./configure WO_CONFIG=/path/to/whizard/binaries
make install
```

   - Default destination: `${HOME}/.whizard`
   - Other destinations can be selected via `--prefix=...`

4. The model is now ready for use, e.g.

```
model = fr_sm
process ezz = "e+", "e-" => Z, Z
compile
sqrts = 500 GeV
integrate (test)
show (results)
```

Features:

- Supports WHIZARD 2 + legacy versions > 1.92
- Fully validated with: SM + Three-Site Model (N. Christensen), MSSM (B. Fuks)
- Handles all fields supported by FeynRules
- Can do unitarity, Feynman and R_ξ gauges
- Supports all gauge invariant dimension 4 operators (+ some higher-order ones)
- Code output is formatted, commented and readable

Features:

- Supports WHIZARD 2 + legacy versions > 1.92
- Fully validated with: SM + Three-Site Model (N. Christensen), MSSM (B. Fuks)
- Handles all fields supported by FeynRules
- Can do unitarity, Feynman and R_ξ gauges
- Supports all gauge invariant dimension 4 operators (+ some higher-order ones)
- Code output is formatted, commented and readable

Plans for the future:

- Extend the range of supported operators
  - Make full use of what is currently supported in O'Mega
  - Extend O'Mega to allow arbitrary operators
- Extend the range of validated models
- Reliability of the interfaces crucial $\longrightarrow$ more sanity checks

Conclusions:

- Implementing models in WHIZARD
  - Involves three steps — O'Mega, WHIZARD and FORTRAN glue
  - Requires some programming skills (O'Caml and FORTRAN)
  - Is tedious for those no technically inclined
- FeynRules
  - Is a powerful package for the calculation of Feynman rules
  - Automatically generates model files for numerous event generators
  - Takes care of the pecularities and conventions of the different codes $\longrightarrow$ user can concentrate on the physics
- The WHIZARD $\longleftrightarrow$ FeynRules interface
  - Generates the code necessary to implement a model into WHIZARD
  - Supports most of the features offered by FeynRules
  - Has been validated with several models
  - Eases the implementation of new models considerably