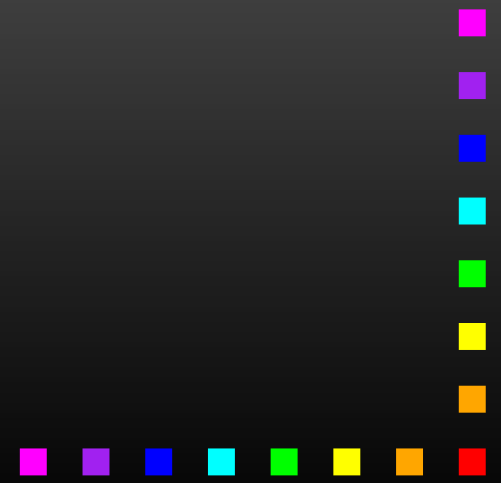
# FormCalc 6

Thomas Hahn

Max-Planck-Institut für Physik
München





# The FeynSystem

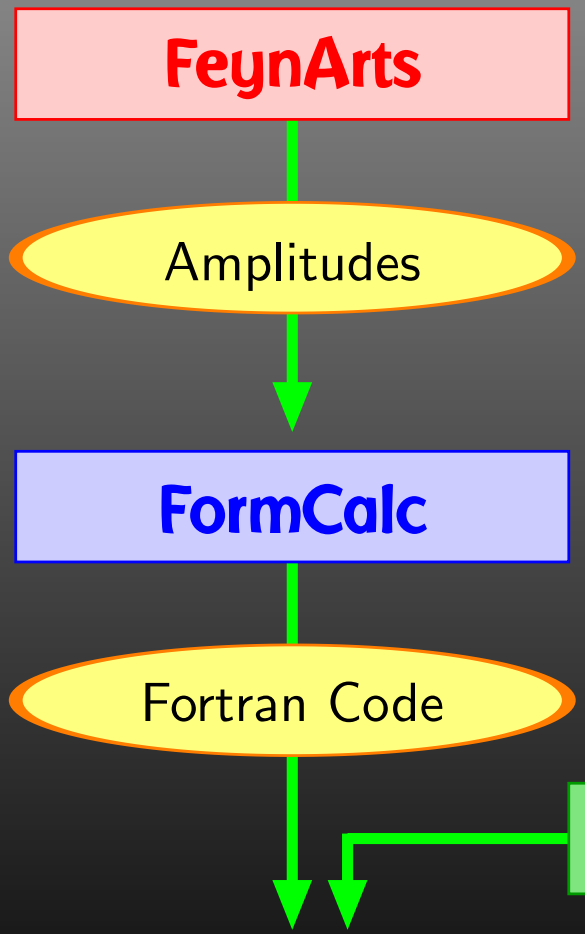



**Diagram Generation:**

- Create the topologies
- Insert fields
- Apply the Feynman rules
- Paint the diagrams

**Algebraic Simplification:**

- Contract indices
- Calculate traces
- Reduce tensor integrals
- Introduce abbreviations

**Numerical Evaluation:**

- Convert Mathematica output to Fortran code
- Supply a driver program
- Implementation of the integrals

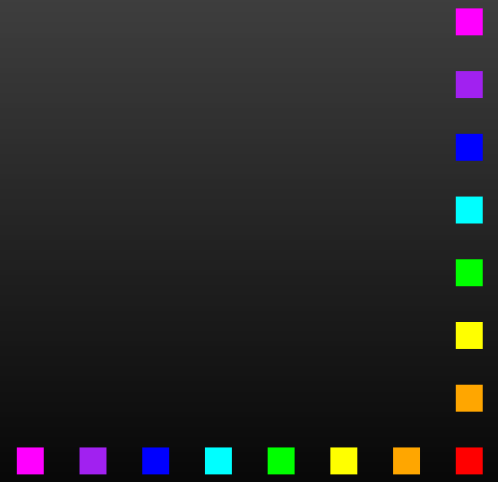
Symbolic manipulation (Computer Algebra) for the structural and algebraic operations.

Compiled high-level language (Fortran) for the numerical evaluation.

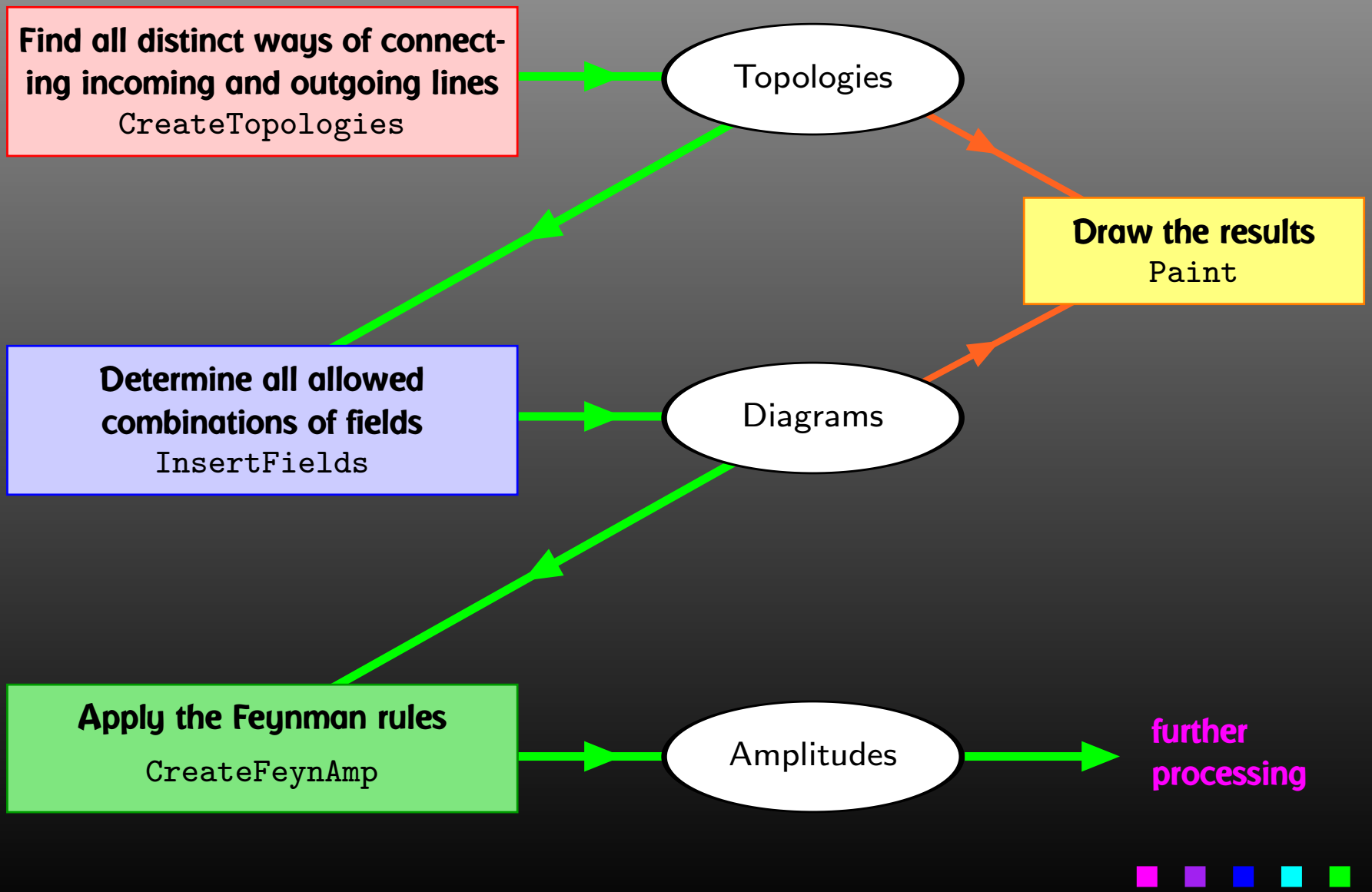
$|\mathcal{M}|^2 \longrightarrow$ Cross-sections, Decay rates, ...

# New Features in Version 6

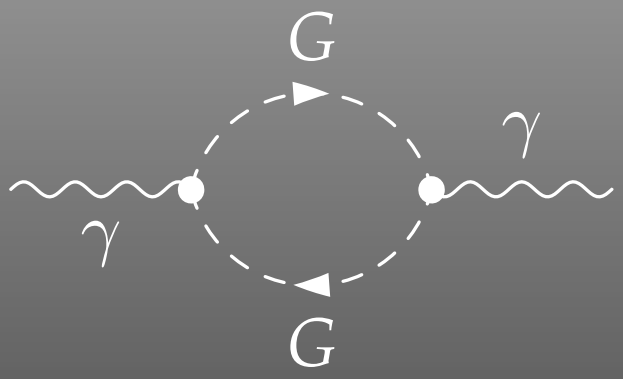
- Alternate-channel communication between FORM and Mathematica.
- Code-generation for CutTools.
- Improvements in Dirac chains (4D) for analytical purposes.
- New functions to re-use previously introduced abbreviations and subexpressions in a new session.



# FeynArts


Find all distinct ways of connecting incoming and outgoing lines
CreateTopologies
Topologies
Draw the results
Paint
Determine all allowed combinations of fields
InsertFields
Diagrams
Apply the Feynman rules
CreateFeynAmp
Amplitudes
further processing

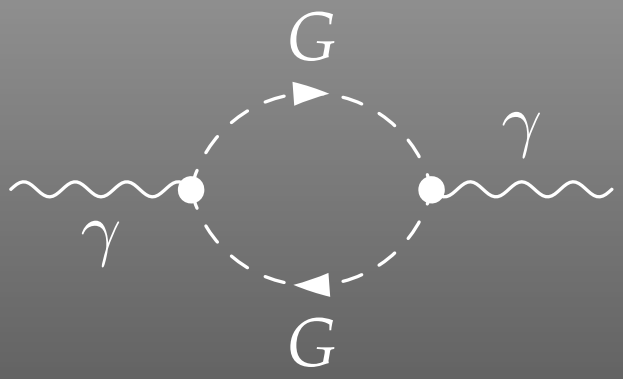
# CreateFeynAmp output




= FeynAmp[ *identifier* ,
*loop momenta* ,
*generic amplitude* ,
*insertions* ]

```
GraphID[Topology == 1, Generic == 1]
```

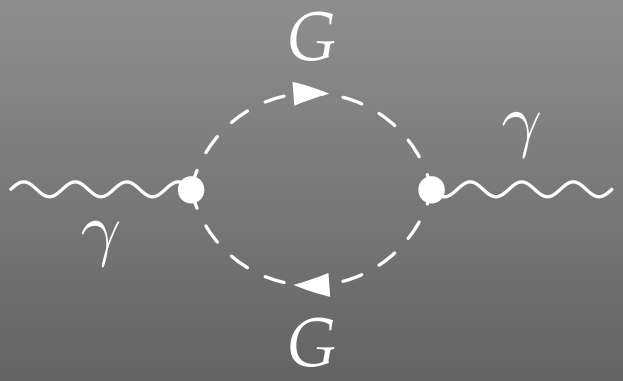
# CreateFeynAmp output




= FeynAmp[ *identifier*,
*loop momenta*,
*generic amplitude*,
*insertions* ]

```
Integral[q1]
```

# CreateFeynAmp output




= FeynAmp[ *identifier*,
*loop momenta*,
*generic amplitude*,
*insertions* ]

$\frac{\texttt{I}}{\texttt{32 Pi}^4}$ RelativeCF ..... *prefactor*

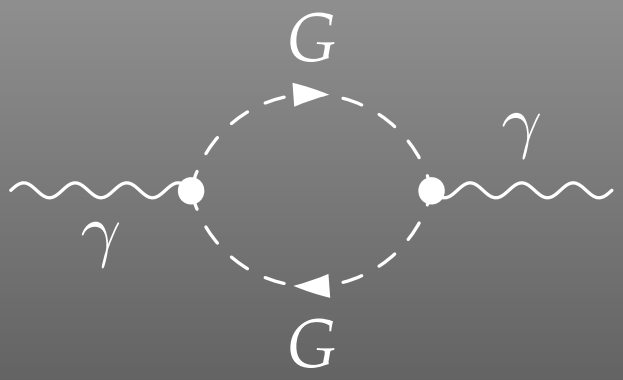
FeynAmpDenominator[$\frac{1}{\texttt{q1}^2 - \texttt{Mass[S[Gen3]]}^2}$, $\frac{1}{(\texttt{-p1 + q1})^2 - \texttt{Mass[S[Gen4]]}^2}$] ..... *loop denominators*

(p1 - 2 q1)[Lor1] (-p1 + 2 q1)[Lor2] ..... *kin. coupling structure*

ep[V[1], p1, Lor1] ep*[V[1], k1, Lor2] ..... *polarization vectors*

$\texttt{G}^{(0)}_{\texttt{SSV}}$[(Mom[1] - Mom[2])[KI1[3]]]
$\texttt{G}^{(0)}_{\texttt{SSV}}$[(Mom[1] - Mom[2])[KI1[3]]] ..... *coupling constants*

# CreateFeynAmp output




= FeynAmp[ *identifier*,
*loop momenta*,
*generic amplitude*,
*insertions* ]

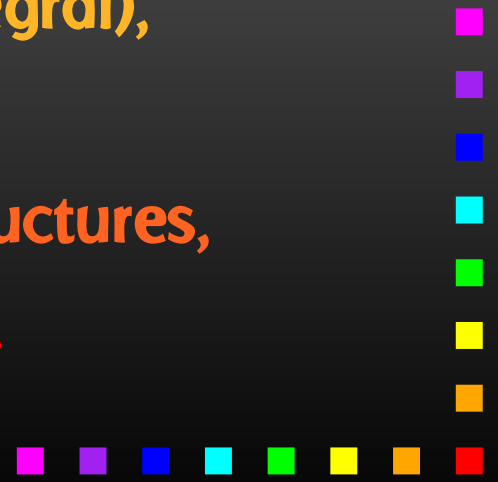
```
{ Mass[S[Gen3]],
  Mass[S[Gen4]],
  G_SSV^(0)[(Mom[1] - Mom[2])[KI1[3]]],
  G_SSV^(0)[(Mom[1] - Mom[2])[KI1[3]]],
  RelativeCF } ->
Insertions[Classes][{MW, MW, I EL, -I EL, 2}]
```

# Algebraic Simplification

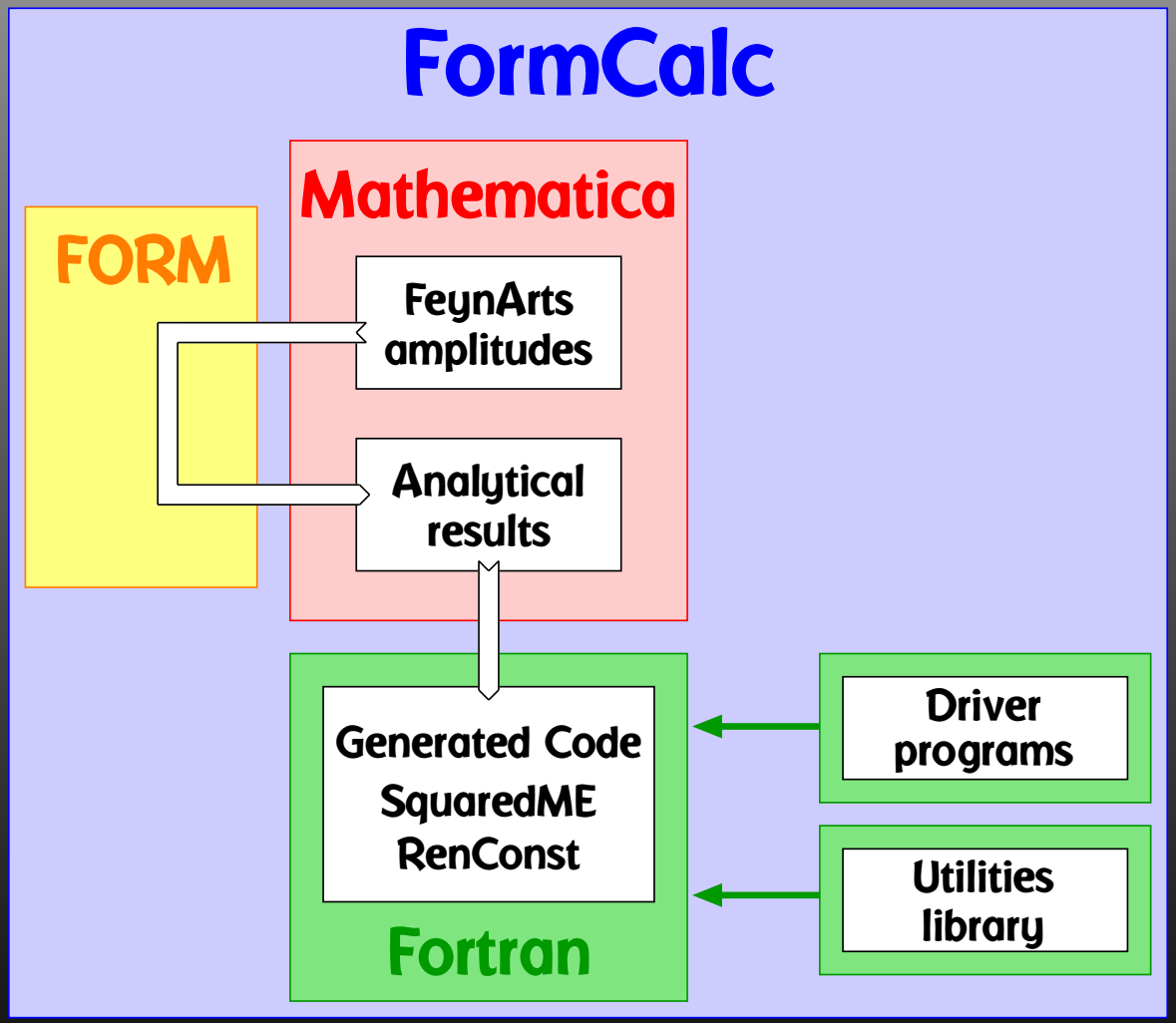
The amplitudes of `CreateFeynAmp` are in no good shape for direct numerical evaluation.

A number of steps have to be done analytically:

- contract indices as far as possible,
- evaluate fermion traces,
- perform the tensor reduction,
- add local terms arising from D·(divergent integral),
- simplify open fermion chains,
- simplify and compute the square of SU(N) structures,
- "compactify" the results as much as possible.



# FormCalc Internals



# FormCalc Output

A typical term in the output looks like

```
C0i[cc12, MW2, MW2, S, MW2, MZ2, MW2] *
  ( -4 Alfa2 MW2 CW2/SW2 S AbbSum16 +
    32 Alfa2 CW2/SW2 S^2 AbbSum28 +
    4 Alfa2 CW2/SW2 S^2 AbbSum30 -
    8 Alfa2 CW2/SW2 S^2 AbbSum7 +
    Alfa2 CW2/SW2 S(T-U) Abb1 +
    8 Alfa2 CW2/SW2 S(T-U) AbbSum29 )
```

= loop integral

= kinematical variables

= constants

= automatically introduced abbreviations

# Abbreviations

**Outright factorization is usually out of question.**
**Abbreviations are necessary to reduce size of expressions.**

```
AbbSum29 = Abb2 + Abb22 + Abb23 + Abb3

Abb22 = Pair1 Pair3 Pair6

Pair3 = Pair[e[3], k[1]]
```

**The full expression corresponding to `AbbSum29` is**

```
Pair[e[1], e[2]] Pair[e[3], k[1]] Pair[e[4], k[1]] +
Pair[e[1], e[2]] Pair[e[3], k[2]] Pair[e[4], k[1]] +
Pair[e[1], e[2]] Pair[e[3], k[1]] Pair[e[4], k[2]] +
Pair[e[1], e[2]] Pair[e[3], k[2]] Pair[e[4], k[2]]
```

# Categories of Abbreviations

- Abbreviations are recursively defined in several levels.
- When generating Fortran code, FormCalc introduces another set of abbreviations for the loop integrals.

In general, the abbreviations are thus costly in CPU time.
It is key to a decent performance that the abbreviations are separated into different Categories:

- Abbreviations that depend on the helicities,
- Abbreviations that depend on angular variables,
- Abbreviations that depend only on $\sqrt{s}$.

Correct execution of the categories guarantees that almost no redundant evaluations are made and makes the generated code essentially as fast as hand-tuned code.

# The Abbreviate Function

The `Abbreviate` **Function** allows to introduce abbreviations for arbitrary (sub-)expressions and extends the advantage of categorized evaluation. Example:

```
abbrexpr = Abbreviate[expr, 5]
```

The second argument, 5, determines the **Level** below which abbreviations are introduced, i.e. how much of expression is 'abbreviated away.' Abbreviationing has the 'side effect' that **duplicate expressions are replaced by the same symbol.**

The subexpressions are **retrieved with** `Subexpr[]`.

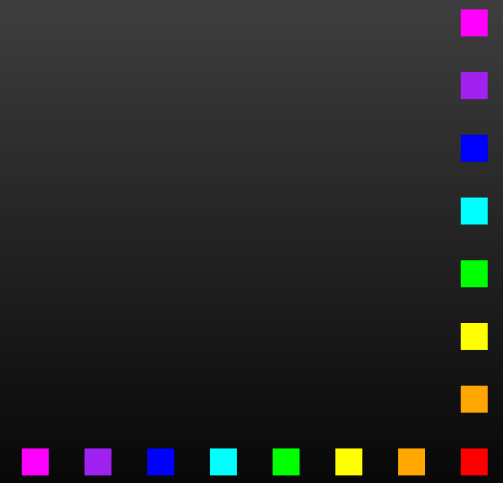
Typical speed-ups are a **factor 3** in MSSM calculations at typical execution times of `Abbreviate` of 30 sec.

# Re-using Abbreviations

Abbreviations were so far restricted to one FormCalc session, e.g. one could not save intermediate results involving abbreviations and resume computation in a new session.

FormCalc 6 adds two functions to 'register' abbreviations and subexpressions from an earlier session:

```
RegisterAbbr[abbr]
RegisterSubexpr[subexpr]
```



# Alternate Link between FORM and Mathematica

FORM is renowned for being able to handle very large expressions. To produce (pre-)simplified expressions, however, terms have to be wrapped in functions, to avoid immediate expansion. The number of terms in a function is severely limited in FORM: on 32-bit systems to 32767.

Dilemma: FormCalc gets more sophisticated in pre-simplifying amplitudes while users want to compute larger amplitudes. Thus, users have recently seen many 'overflow' messages from FORM.

Solution: Pre-simplified generic amplitude is sent to Mathematica intermediately for introducing abbreviations. Result: significant reduction in size of intermediate expressions.

Tentukov, Vermaseren 2006

# Effect on Intermediate Amplitudes

## FORM $\longrightarrow$ Mathematica:

part of $uu \longrightarrow gg$ @ tree level

```
+Den[U,MU2]*(
  -8*SUNSum[Col5,3]*SUNT[Glu3,Col5,Col2]*SUNT[Glu4,Col1,Col5]*mul[Alfas*Pi]*
  abb[fme[WeylChain[DottedSpinor[k1,MU,-1],6,Spinor[k2,MU,1]]]*ec3.ec4
    -1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec3,ec4,Spinor[k2,MU,1]]]
    +fme[WeylChain[DottedSpinor[k1,MU,-1],7,Spinor[k2,MU,1]]]*ec3.ec4
    -1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec3,ec4,Spinor[k2,MU,1]]]]*MU
  -4*SUNSum[Col5,3]*SUNT[Glu3,Col5,Col2]*SUNT[Glu4,Col1,Col5]*mul[Alfas*Pi]*
  abb[fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec3,ec4,k3,Spinor[k2,MU,1]]]
    -2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec4,Spinor[k2,MU,1]]]*ec3.k2
    -2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,k3,Spinor[k2,MU,1]]]*ec3.ec4
    +fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec3,ec4,k3,Spinor[k2,MU,1]]]
    -2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec4,Spinor[k2,MU,1]]]*ec3.k2
    -2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,k3,Spinor[k2,MU,1]]]*ec3.ec4]
  +8*SUNSum[Col5,3]*SUNT[Glu3,Col5,Col2]*SUNT[Glu4,Col1,Col5]*mul[Alfas*MU*Pi]*
  abb[fme[WeylChain[DottedSpinor[k1,MU,-1],6,Spinor[k2,MU,1]]]*ec3.ec4
    -1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec3,ec4,Spinor[k2,MU,1]]]
    +fme[WeylChain[DottedSpinor[k1,MU,-1],7,Spinor[k2,MU,1]]]*ec3.ec4
    -1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec3,ec4,Spinor[k2,MU,1]]]] )
```

## Mathematica $\longrightarrow$ FORM:

```
-4*Den(U,MU2)*SUNSum(Col5,3)*SUNT(Glu3,Col5,Col2)*SUNT(Glu4,Col1,Col5)*
  AbbSum5*Alfas*Pi
```

# CutTools

Tensor loop integrals have in FormCalc so far been treated by Passarino-Veltman reduction only, e.g.

$$\frac{q_\mu q_\nu}{D_0 D_1} = g_{\mu\nu}\,\mathtt{B00}(p, m_1, m_2) + p_\mu p_\nu\,\mathtt{B11}(p, m_1, m_2)$$

where `B00` and `B11` are provided by LoopTools.

CutTools implements the cutting-technique-inspired OPP (Ossola, Papadopoulos, Pittau) method. It needs the numerator as a function of $q$ which it can sample:

$$\mathtt{Bcut}(2, \mathtt{num1}, \mathtt{num2}, p, m_1, m_2)$$

where $\mathtt{num1} = q_\mu q_\nu$ and $\mathtt{num2} = 0$ (coeff. of $D - 4$).

Independent way of checking LoopTools results.
Performance?

# CutTools Validation

**Present status of validation of CutTools results with HELAC:**

- **Use CutTools for cut-constructible and R1 terms.**

  Ossola, Papadopoulos, Pittau 2007

- **R2 rational terms through Feynman rules, all in 4 dimensions.**

  Ossola, Papadopoulos, Pittau 2008

- **van Hameren's code for loop functions (IR).**

  van Hameren, Vollinga, Weinzierl 2005

- **All colour connections - full colour results.**

  Kanaki, Papadopoulos 2000

- **Ward Identities checked for all color connections. Comparison for $n = 4, 5, 6$ gluons with known results.**

# CutTools Validation

## Comparison of six-gluon Amplitude:

| Helicity | Method | $1/\epsilon^2$ | $1/\epsilon$ | 1 |
|---|---|---|---|---|
| $+\ +\ +\ +\ +\ +$ | BDDK | 0 | 0 | 0.5298064836 43855 |
| | EGZ | $1.060660419488780 \times 10^{-14}$ | $3.813284749527035 \times 10^{-14}$ | 0.529806483661295 |
| | OPP-HELAC | $2.665666018358549 \times 10^{-16}$ | $3.712705224857268 \times 10^{-16}$ | 0.529806483661295 |
| $-\ +\ +\ +\ +\ +$ | BDDK | $1.011255761241711 \times 10^{-11}$ | $6.753625348984687 \times 10^{-10}$ | 3.2599670 4351899 |
| | EGZ | $2.377895374324842 \times 10^{-13}$ | $8.549005883762705 \times 10^{-13}$ | 3.25996705427236 |
| | OPP-HELAC | $4.873074271795793 \times 10^{-16}$ | $4.562297672115668 \times 10^{-15}$ | 3.25996705427234 |
| $-\ -\ +\ +\ +\ +$ | BDDK | 170.947689902659 | 614.590878376396 | 1373.74753500854 |
| | EGZ | 170.947689902659 | 614.590878376397 | 1373.74753500828 |
| | OPP-HELAC | 170.947689902659 | 614.590878376397 | 1373.74753500852 |
| $-\ +\ -\ +\ -\ +$ | BDDK | 18.8322923700467 | 67.7058293474830 | 151.043950328960 |
| | EGZ | 18.8322923700485 | 67.7058292869577 | 151.043950337947 |
| | OPP-HELAC | 18.8322923700484 | 67.7058292869573 | 151.043950337954 |
| $+\ -\ +\ -\ +\ -$ | BDDK | 18.8322923700554 | 67.7058292857048 | 153.780101529836 |
| | EGZ | 18.8322923700485 | 67.7058292869577 | 153.780101415986 |
| | OPP-HELAC | 18.8322923700484 | 67.7058292869573 | 153.780101615242 |

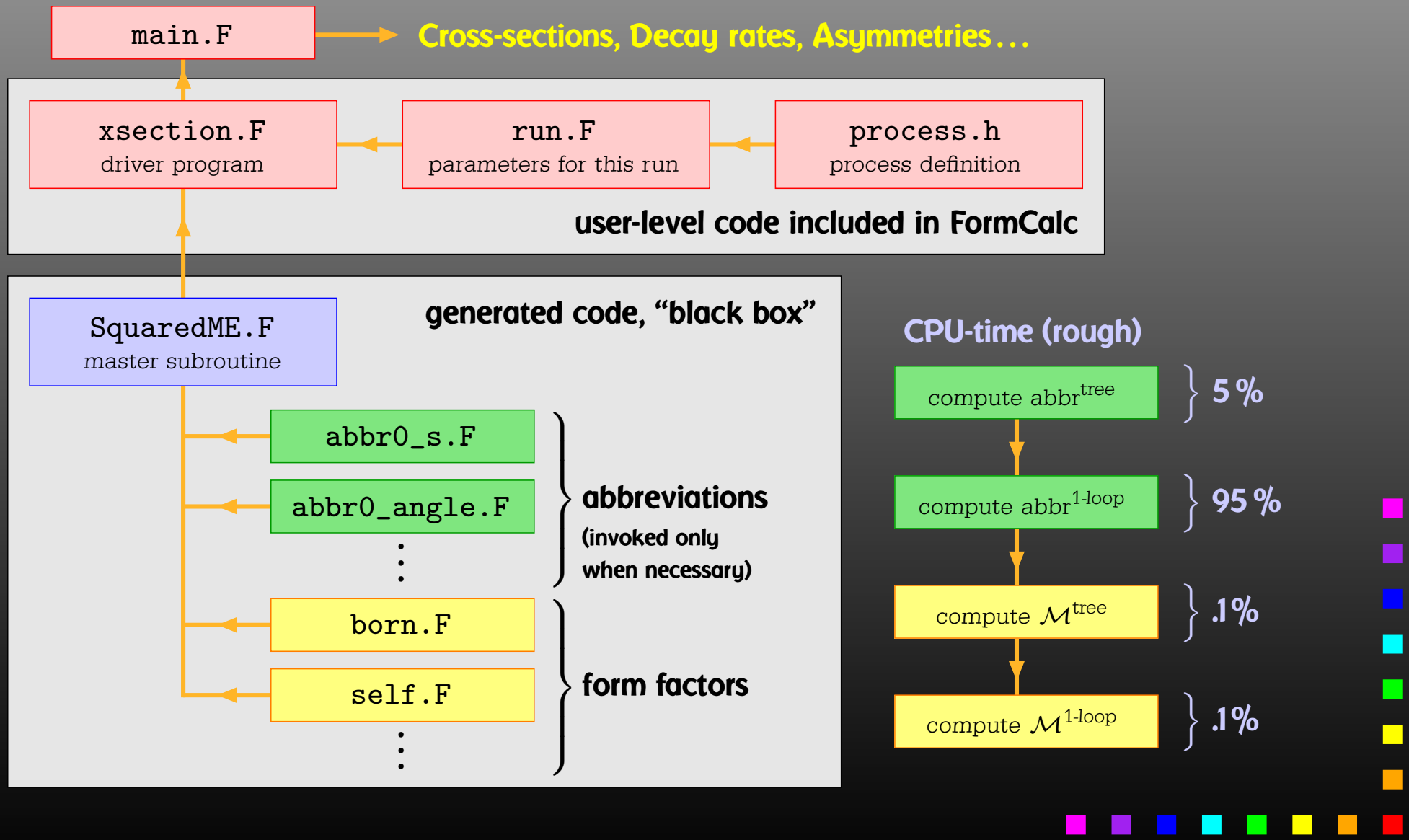
Bern, Dixon, Dunbar, Kosower 1994; 1995
Ellis, Giele, Zanderighi 2006; Giele, Zanderighi 2008

# Dirac Chains in 4D

As numerical calculations are done mostly using Weyl-spinor chains, there has been a paradigm shift for Dirac chains to make them better suited for analytical purposes, e.g. the extraction of Wilson coefficients.

- Already in Version 5, Fierz methods have been implemented for Dirac chains, thus allowing the user to force the fermion chains into almost any desired order.
- Version 6 further adds the `Colour` method to the `FermionOrder` option of `CalcFeynAmp`, which brings the spinors into the same order as the external colour indices.
- Also new in Version 6: completely antisymmetrized Dirac chains, i.e. $\texttt{DiracChain}[1, \mu, \nu] = \sigma_{\mu\nu}$.

# Numerical Evaluation in Fortran 77



# Features of the Generated Code

- **Modular:** largely autonomous pieces of code provide
  - kinematics,
  - model initialization,
  - convolution with PDFs.
- **Extensible:** default code serves (only) as an example. Other 'Frontends' can be supplied, e.g. HadCalc, sofox.
- **Re-usable:** external program need only call `ProcessIni` (to set up the process) and `ParameterScan` (to set off the calculation).
- **Interactive:** Mathematica interface provides Mathematica function for cross-section/decay rate.
- **Parallel:** built-in distribution of parameter scans.

# Coming: FeynHiggs as a SUSY Frontend

On the technical level, FeynHiggs provides an interface for SUSY parameters with

- several input methods (direct, file, SLHA, Mathematica),
- parameter-scan capable.

Easy I/O with `SLHARead`, `SLHAWrite`, `FHReadRecord`, etc.
Routines to extract parameters from FeynHiggs:

- `FHGetPara`, retrieve computed parameters (masses etc.),
- `FHRetrievePara`, retrieve input parameters,
- `FHRetrieveSMPara` for SM,
- `FHRetrieveNMFV` for non-minimal flavour-violation,

The `FHRetrieveXY` functions have the same invocation as `FHSetXY`.

# MultiCore Package

**MultiCore is a new package to distribute Mathematica computations (numeric and symbolic) on a cluster of PCs.**

**Provides `MapCore` that substitutes `Map` (/@):**

```
<< MultiCore`

AddCore[10, << myprogram.m]

tb = Range[5, 50, .5];
sig = MapCore[sigma, tb]

ListPlot[Thread[{tb, sig}]]
```

load long definitions in remote session

take 10 free hosts from .submitrc file

replace `Map` by `MapCore`

**Need to set up hosts once in `.submitrc` file:**

```
pc123   2
pcboss  4
pcsven
```

# Summary and Outlook

New Features in FormCalc Version 6:

- Intermediate FORM expressions get sent to Mathematica for abbreviations – significant size reduction.
- Code can be generated for the CutTools library.
- New Functions for registering abbreviations and subexpressions allow to 'resume' sessions.
- Improvements in 4D Dirac chains for analytical purposes (choice of ordering, antisymmetrization).
- Version 6 will shortly be available: http://www.feynarts.de/formcalc

New MultiCore Package parallelizes Mathematica evaluations:

- Requires one-time setup of `.submitrc` for cluster.
- Replaces `Map` by `MapCore`.
- Will be released soon: http://www.feynarts.de/mapcore