MadGraph 5 – The All-New Matrix Element generator for Everything

Johan Alwall
Fermilab

For the MadGraph 5 team:
J.A. Fabio Maltoni (Louvain)
Olivier Mattelaer (Louvain) Tim Stelzer (UIUC)
and Michel Herquet

Fermilab Theory Seminar, March 31, 2011

The LHC is on track!

but where are we heading...?

What will be needed for the LHC?

NLO

Exp-TH communication

Very exotic models

Exotic models

Effective theories

DECAY CHAINS

MATRIX ELEMENTS

Advanced analysis techniques

Multi-jet samples

Cluster/Grid computing

Merging ME/PS

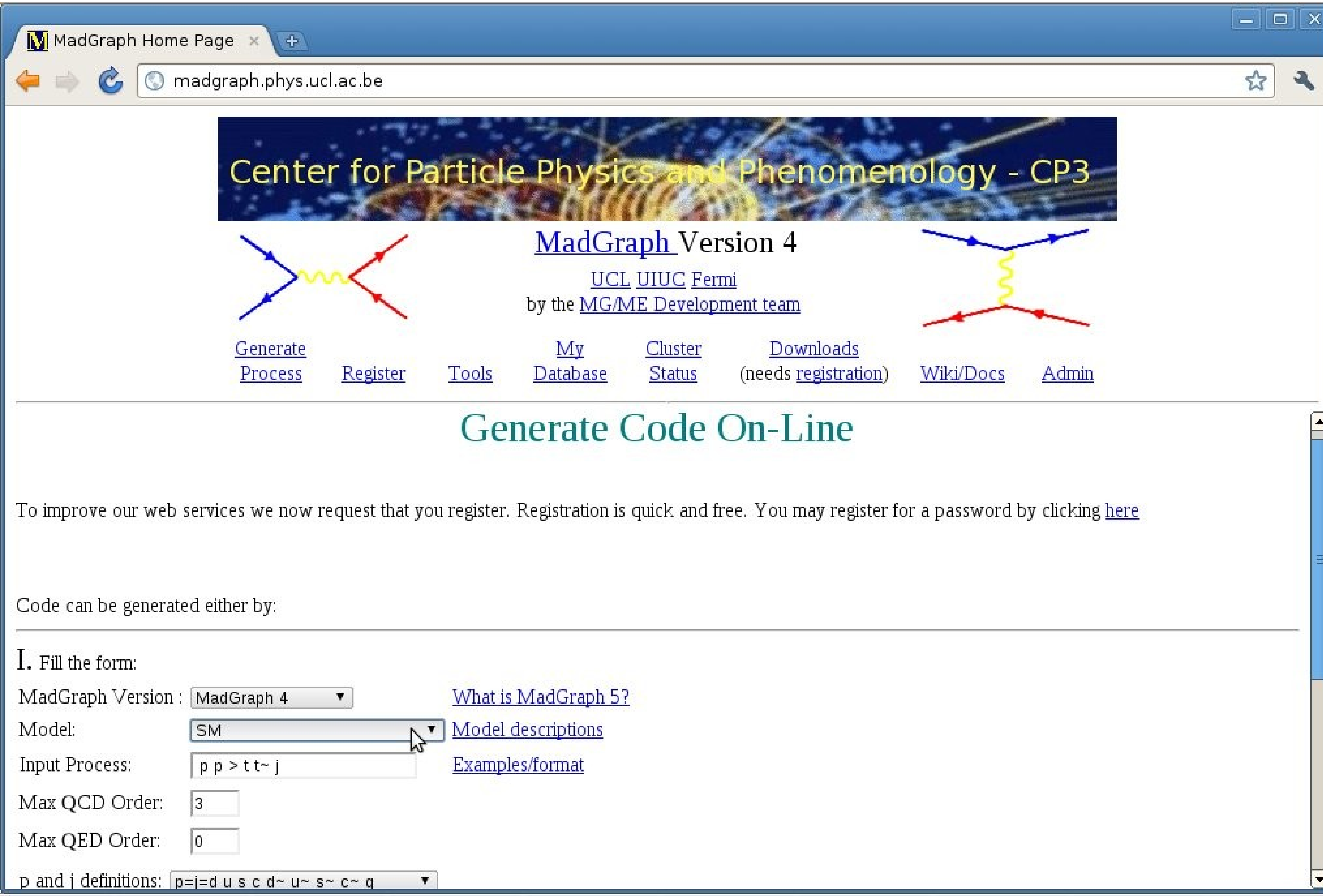
DECAY PACKAGES

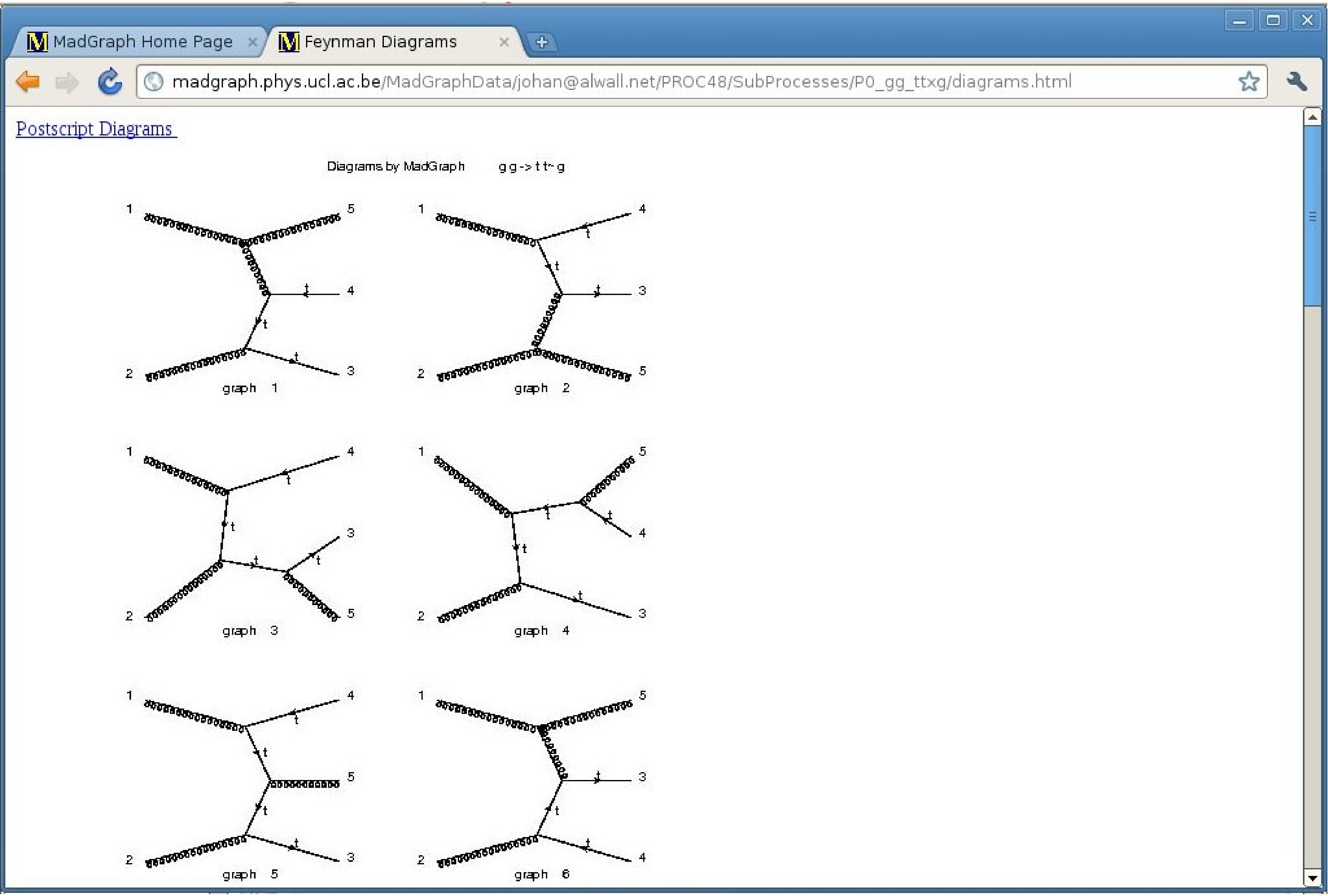
Testing / robustness

User friendliness

MadGraph/MadEvent 4

- One of the most widely used automatized matrix element generators
  - Specify any process using simple syntax
  - > 1500 registered users (+ CDF/D0/CMS/ATLAS/...)
- Originally written by Tim Stelzer in 1994
- Phase space integrator/event generator MadEvent by F. Maltoni and T. Stelzer in 2002
- MadGraph/MadEvent v. 4 in 2006


MadGraph Home Page
madgraph.phys.ucl.ac.be
Center for Particle Physics and Phenomenology - CP3
MadGraph Version 4
UCL UIUC Fermi
by the MG/ME Development team
Generate Process
Register
Tools
My Database
Cluster Status
Downloads (needs registration)
Wiki/Docs
Admin
Generate Code On-Line
To improve our web services we now request that you register. Registration is quick and free. You may register for a password by clicking here
Code can be generated either by:
I. Fill the form:
MadGraph Version : MadGraph 4
What is MadGraph 5?
Model: SM
Model descriptions
Input Process: p p > t t~ j
Examples/format
Max QCD Order: 3
Max QED Order: 0
p and j definitions: p=j=d u s c d~ u~ s~ c~ g


MadGraph Home Page
Feynman Diagrams
madgraph.phys.ucl.ac.be/MadGraphData/johan@alwall.net/PROC48/SubProcesses/P0_gg_ttxg/diagrams.html
Postscript Diagrams
Diagrams by MadGraph g g -> t t~ g
graph 1
graph 2
graph 3
graph 4
graph 5
graph 6

MadGraph Home Page | Online Event Genera...

madgraph.phys.ucl.ac.be/MadGraphData//johan@alwall.net/PROC48/HTML/crossx.html

Results for p p > t t~ j in the sm

Available Results

Links	Events		Tag	Run	Collider	Cross section (pb)	Events	
results plots banner	Parton-level	LHE rootfile	fermi	run_01	p p 7000 x 7000 GeV	.60019E+03	10000	Remove run
	Hadron-level (Pythia)	STDHEP LHE rootfile (LHE)						Remove Pythia
	Reco. Objects. (PGS)	LHCO rootfile						Remove PGS

Main Page

Why new MadGraph?

- First version of core code from 1994
  - Never intended for the present variety of use cases and periferal tools
- Written in Fortran 77
  - Fixed array sizes
  - No recursion
  - Difficult to modularize, modify or extend (no OO, dynamic libraries,...)
- Becoming a roadblock for new developments

MadGraph 5

- Development started November 2009
  - First public full MG functionality version in May 2010
  - Release version 1.0.0 upcoming (including revamped MadEvent and much much more)
- Programming language: Python
  - (Very) high level, modern programming language
  - Easy to learn/write/maintain/extend/structure/...
- Modular program structure
  - Central collection of tools for any matrix element-based implementations
- New algorithms for every part of the code
  - Diagram generation, color, decay chains, ...

Speed benchmarks

Full MadEvent subprocess directory output, including diagram drawing

Computer: Sony Vaio TZ

Process	MadGraph 4	MadGraph 5	Subprocesses	Diagrams
$pp \to jjj$	29.0 s	54.4 s	34	307
$pp \to jjl^+l^-$	341 s	258 s	108	1216
$pp \to jjje^+e^-$	1151 s	654 s	141	9012
$u\bar{u} \to e^+e^-e^+e^-e^+e^-$	772 s	175 s	1	3474
$gg \to ggggg$	2788 s	1049 s	1	7245
$pp \to jj(W^+ \to l^+\nu_l)$	146 s	70 s	82	304
$pp \to t\bar{t}$+full decays	5640 s	22 s	27	45
$pp \to \tilde{q}/\tilde{g}\,\tilde{q}/\tilde{g}$	222 s	286 s	313	475
7 particle decay chain	383 s	5.2 s	1	6
$gg \to (\tilde{g} \to u\bar{u}\tilde{\chi}_1^0)(\tilde{g} \to u\bar{u}\tilde{\chi}_1^0)$	70 s	5.5 s	1	48
$pp \to (\tilde{g} \to jj\tilde{\chi}_1^0)(\tilde{g} \to jj\tilde{\chi}_1^0)$	$\gg$ 1 year	551 s	144	11008

$$gg \to (\tilde{g} \to u(\bar{\tilde{u}}_l \to \bar{u}(\tilde{\chi}_2^0 \to Z\tilde{\chi}_1^0)))(\tilde{g} \to u\hat{\bar{\tilde{d}}}\tilde{\chi}_1^-)$$

Speed of ME calculation

Computer: Sony Vaio TZ

Process	Amplitudes	Wavefunctions		Run time	
		MG 4	MG 5	MG 4	MG 5
$u\bar{u} \to e^+e^-$	2	6	6	$< 6\mu s$	$< 6\mu s$
$u\bar{u} \to e^+e^-e^+e^-$	48	62	32	0.22 ms	0.14 ms
$u\bar{u} \to e^+e^-e^+e^-e^+e^-$	3474	3194	301	46.5 ms	19.0 ms
$u\bar{u} \to d\bar{d}$	1	5	5	$< 4\mu s$	$< 4\mu s$
$u\bar{u} \to d\bar{d}g$	5	11	11	27 μs	27 μs
$u\bar{u} \to d\bar{d}gg$	38	47	29	0.42 ms	0.31 ms
$u\bar{u} \to d\bar{d}ggg$	393	355	122	10.8 ms	6.75 ms
$u\bar{u} \to u\bar{u}gg$	76	84	40	1.24 ms	0.80 ms
$u\bar{u} \to u\bar{u}ggg$	786	682	174	35.7 ms	17.2 ms
$u\bar{u} \to d\bar{d}d\bar{d}$	14	28	19	84 μs	83 μs
$u\bar{u} \to d\bar{d}d\bar{d}g$	132	178	65	1.88 ms	1.15 ms
$u\bar{u} \to d\bar{d}d\bar{d}gg$	1590	1782	286	141 ms	34.4 ms
$u\bar{u} \to d\bar{d}d\bar{d}d\bar{d}$	612	758	141	42.5 ms	6.6 ms

Speed of ME calculation

Computer: Sony Vaio TZ

Factor ~10 less function calls
→ factor ~2.5 speed improvement!

Process	Amplitudes	Wavefunctions		Run time	
		MG 4	MG 5	MG 4	MG 5
$u\bar{u} \to e^+e^-$	[illegible]	[illegible]	[illegible]	[illegible]	$< 6\mu s$
$u\bar{u} \to e^+e^-e^+e^-$	48	62	32	0.22 ms	0.14 ms
$u\bar{u} \to e^+e^-e^+e^-e^+e^-$	3474	3194	301	46.5 ms	19.0 ms
$u\bar{u} \to d\bar{d}$	1	5	5	$< 4\mu s$	$< 4\mu s$
$u\bar{u} \to d\bar{d}g$	5	11	11	27 μs	27 μs
$u\bar{u} \to d\bar{d}gg$	38	47	29	0.42 ms	0.31 ms
$u\bar{u} \to d\bar{d}ggg$	393	355	122	10.8 ms	6.75 ms
$u\bar{u} \to u\bar{u}gg$	76	84	40	1.24 ms	0.80 ms
$u\bar{u} \to u\bar{u}ggg$	786	682	174	35.7 ms	17.2 ms
$u\bar{u} \to d\bar{d}d\bar{d}$	14	28	19	84 μs	83 μs
$u\bar{u} \to d\bar{d}d\bar{d}g$	132	178	65	1.88 ms	1.15 ms
$u\bar{u} \to d\bar{d}d\bar{d}gg$	1590	1782	286	141 ms	34.4 ms
$u\bar{u} \to d\bar{d}d\bar{d}d\bar{d}$	612	758	141	42.5 ms	6.6 ms

Test suite

Unit (= method) tests, Acceptance tests and Parallel tests

```
[alwall@alwall-laptop madgraph5] python tests/test_manager.py
...
Test a complete decay chain process gp>jg,j>jjj,j>jjj ... ok
Test a complete decay chain process pp>jj,j>jj with QED=2, QCD=2 ... ok
Test the functions for checking equal decay chains ... ok
Test the HelasMultiProcess with the processes uu~>uu~ ... ok
Setting up and testing pp > nj based on multiparticle lists, ... ok
Test decay chain with majorana particles e+e->n1n1 ... ok
Test a multistage decay g g > d d~, d > g d, d~ > g d~, g > u u~ g ... ok
Test error raising in HelasWavefunction __init__, get and set ... ok
Test filters for wavefunction properties ... ok
Test the color basis building for uu~ > aggg (3! elements) ... ok
Test the color flow decomposition of various color strings ... ok
Test the colorize function for uu~ > gg ... ok
Test the colorize function for uu~ > ggg ... ok
Test the Nc power restriction during color basis building ... ok
Test index fixing for immutable color string ... ok
----------------------------------------------------------------------
Ran 415 tests in 34.098s
```

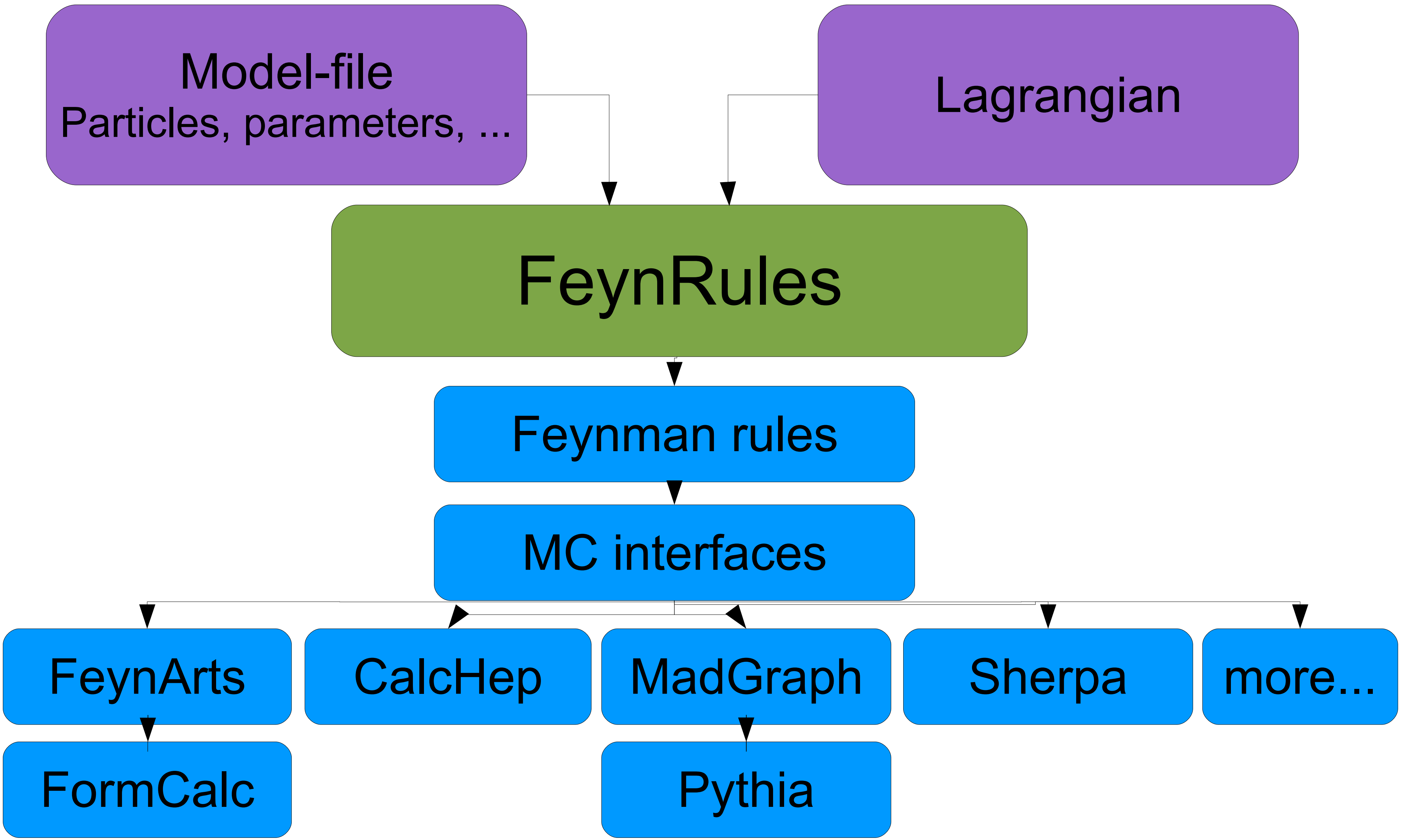
~30500 lines of tests, cf. ~29000 lines of code in MadGraph 5

The big news

- Automatic writing of HELAS routines for vertices with any Lorentz structure
- Effective vertices with any number of particles (including multi-fermion vertices)
- New color structures (color sextets and ε^{ijk})
- Checks to validate model implementations
- Output of C++ matrix elements for Pythia 8
- New, compact organization of subprocesses and integration channels in MadEvent
- Really user-friendly command line interface

Sidenote: FeynRules

[Christiansen, Duhr, arXiv:0806.4194]



Automatic HELAS routines

with P. de Aquino, C. Duhr, W. Link

- Universal FeynRules Output (UFO)
  - Includes color and Lorentz structure
  - Allows for complete specification of effective/non-renormalizable vertices
  - Allows for automatic output of model parameter calculations for any model and language
- Automatic Language-independent Output of Helicity Amplitudes (ALOHA)
  - Automatic generation the necessary helicity amplitude code for any new model (including effective theories, multi-fermion vertices,...) in Fortran/C++/Python/...

Universal FeynRules Output (UFO)

particles.py:

```
G = Particle(pdg_code = 21,
         name = 'G',
         antiname = 'G',
         spin = 3,
         color = 8,
         mass = 'ZERO',
         width = 'ZERO',
         texname = 'G',
         antitexname = 'G',
         line = 'curly',
         charge = 0,
         LeptonNumber = 0,
         GhostNumber = 0)
```

lorentz.py:

```
VVV1 = Lorentz(name = 'VVV1',
         spins = [ 3, 3, 3 ],
         Structure =
                 'P(3,1)*Metric(1,2) -
                 P(3,2)*Metric(1,2) -
                 P(2,1)*Metric(1,3) +
                 P(2,3)*Metric(1,3) +
                 P(1,2)*Metric(2,3) -
                 P(1,3)*Metric(2,3)')
```

couplings.py:

```
GC_4 = Coupling(name = 'GC_4',
         value = '-G',
         order = {'QCD':1})
```

vertices.py:

```
V_2 = Vertex(name = 'V_2',
         particles = [ P.G, P.G, P.G ],
         color = [ 'f(1,2,3)' ],
         lorentz = [ L.VVV1 ],
         couplings = {(0,0):C.GC_4})
```

ALOHA output

Fortran

```
      SUBROUTINE VVV1_0(V1,V2,V3,C,VERTEX)
      IMPLICIT NONE
      DOUBLE COMPLEX V1(6)
      DOUBLE COMPLEX V2(6)
      DOUBLE COMPLEX V3(6)
      DOUBLE COMPLEX C
      DOUBLE COMPLEX VERTEX
      DOUBLE PRECISION P2(0:3),P3(0:3),P1(0:3)

      P2(0) =  DBLE(V2(5))
      P2(1) =  DBLE(V2(6))
      P2(2) =  DIMAG(V2(6))
      P2(3) =  DIMAG(V2(5))
      P3(0) =  DBLE(V3(5))
      P3(1) =  DBLE(V3(6))
      P3(2) =  DIMAG(V3(6))
      P3(3) =  DIMAG(V3(5))
      P1(0) =  DBLE(V1(5))
      P1(1) =  DBLE(V1(6))
      P1(2) =  DIMAG(V1(6))
      P1(3) =  DIMAG(V1(5))

VERTEX = C*( (V3(1)*( (V1(1)*( (V2(2)*( (0, -1)*P1(1)+(0, 1)
     $ *P3(1)))+( (V2(3)*( (0, -1)*P1(2)+(0, 1)*P3(2)))+(V2(4)*( (0,
     $ -1)*P1(3)+(0, 1)*P3(3))))))+( (V2(1)*( (V1(2)*( (0, 1)*P2(1)
     $ +(0, -1)*P3(1)))+( (V1(3)*( (0, 1)*P2(2)+(0, -1)*P3(2)))
     $ +(V1(4)*( (0, 1)*P2(3)+(0, -1)*P3(3))))))+( (P1(0)*( (0, 1)
     $ *(V2(2)*V1(2))+(0, 1)*(V2(3)*V1(3))+(0, 1)*(V2(4)*V1(4))))
     ...
```

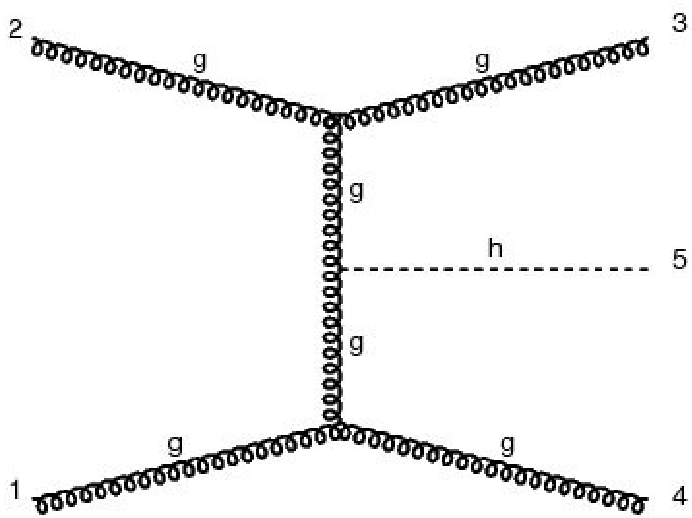
ALOHA output

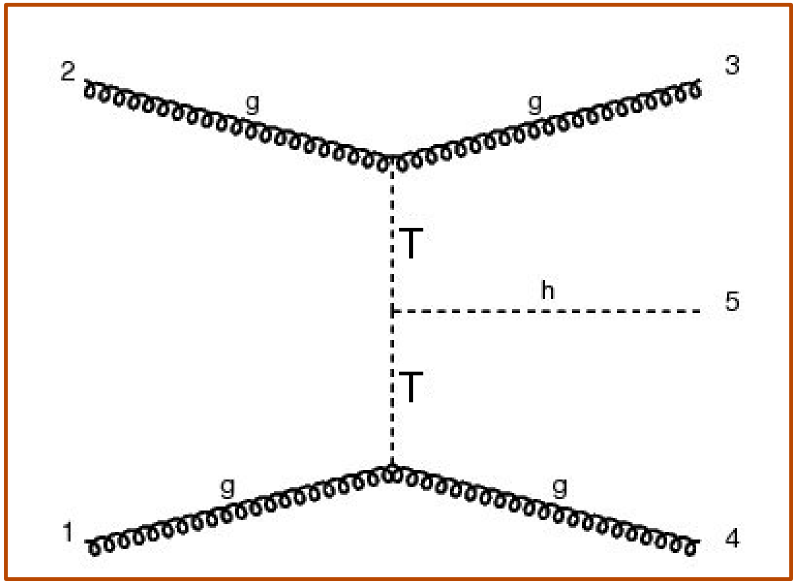
C++

```
void VVV1_0(complex<double> V1[], complex<double> V2[], complex<double> V3[],
   complex<double> C, complex<double> & vertex)
{
  double P2[4], P3[4], P1[4];
  P2[0] = V2[4].real();
  P2[1] = V2[5].real();
  P2[2] = V2[5].imag();
  P2[3] = V2[4].imag();
  P3[0] = V3[4].real();
  P3[1] = V3[5].real();
  P3[2] = V3[5].imag();
  P3[3] = V3[4].imag();
  P1[0] = V1[4].real();
  P1[1] = V1[5].real();
  P1[2] = V1[5].imag();
  P1[3] = V1[4].imag();
  vertex = C * ((V3[0] * ((V1[0] * ((V2[1] * (complex<double> (0., -1.) * P1[1]
     + complex<double> (0., 1.) * P3[1])) + ((V2[2] * (complex<double> (0.,
     -1.) * P1[2] + complex<double> (0., 1.) * P3[2])) + (V2[3] *
     (complex<double> (0., -1.) * P1[3] + complex<double> (0., 1.) *
     P3[3]))))) + ((V2[0] * ((V1[1] * (complex<double> (0., 1.) * P2[1] +
     complex<double> (0., -1.) * P3[1])) + ((V1[2] * (complex<double> (0., 1.)
     * P2[2] + complex<double> (0., -1.) * P3[2])) + (V1[3] * (complex<double>
     (0., 1.) * P2[3] + complex<double> (0., -1.) * P3[3]))))) + ((P1[0] *
     (complex<double> (0., 1.) * (V2[1] * V1[1]) + complex<double> (0., 1.) *
     (V2[2] * V1[2]) + complex<double> (0., 1.) * (V2[3] * V1[3]))) + (P2[0] *
     (complex<double> (0., -1.) * (V2[1] * V1[1]) + complex<double> (0., -1.)
     * (V2[2] * V1[2]) + complex<double> (0., -1.) * (V2[3] * V1[3])))))) +
     ...
```

Multi-particle vertices

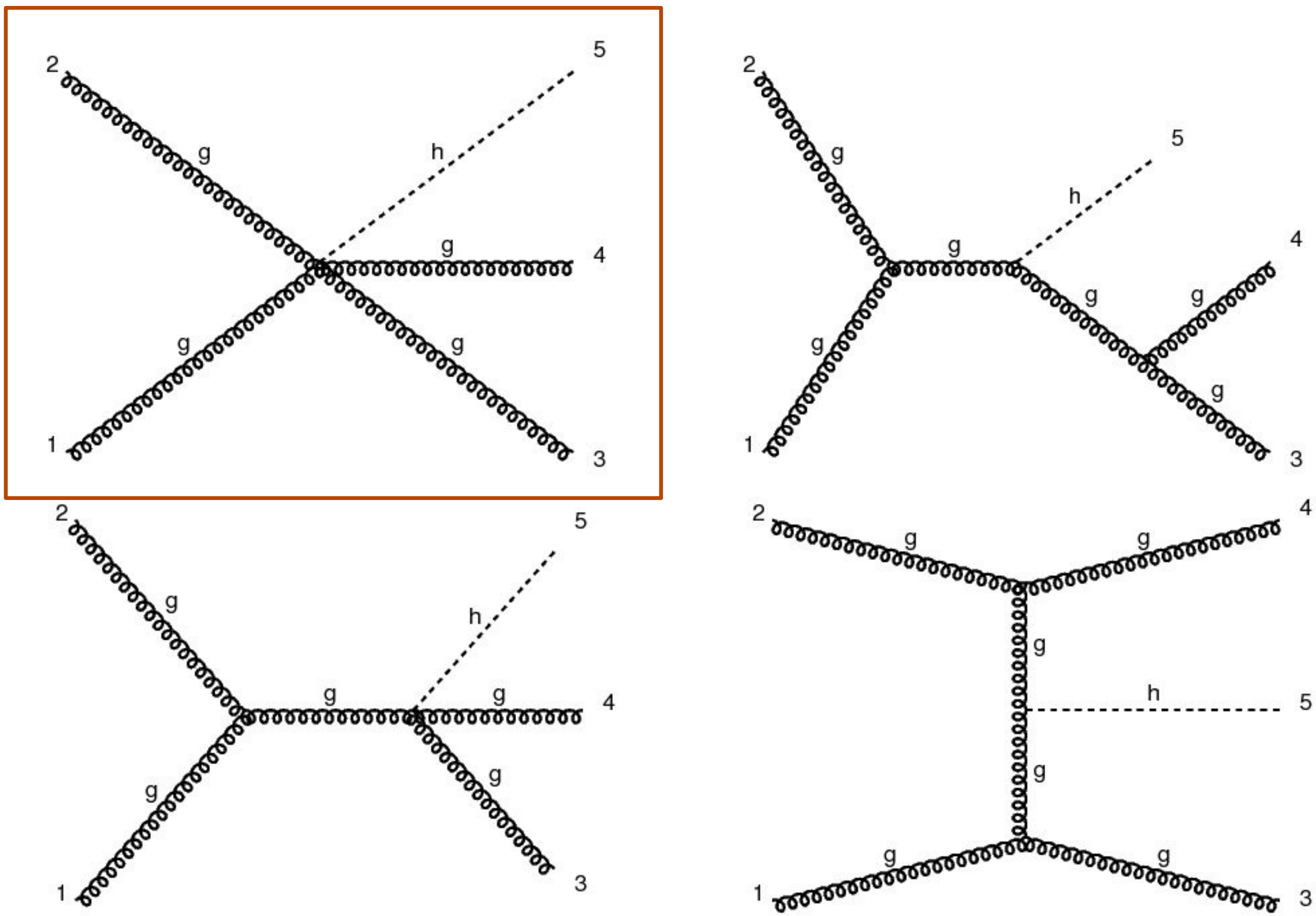
- Higgs effective couplings to gluons (through top loop)
  - From non-commutativity of QCD, get H + 2-, 3- and 4-gluon vertices
  - In MG4: 5-particle vertex handled by using non-propagating tensor particles





Effective multi-particle vertices

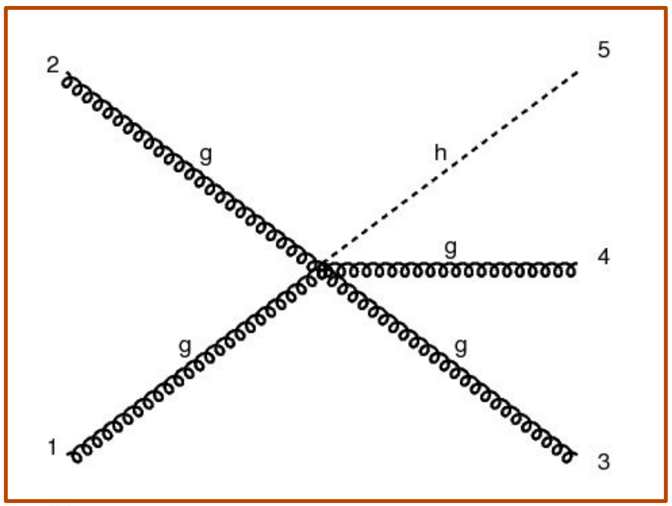
- In MG5: Use 5-particle vertex directly!

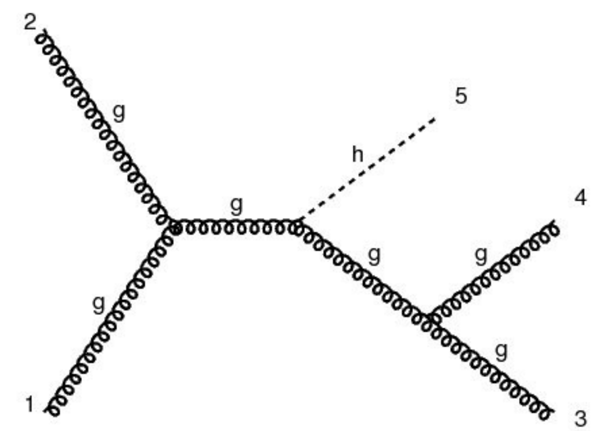


(Thoroughly tested against MG4)

Effective multi-particle vertices

- In MG5: Use 5-particle vertex directly!





lorentz.py

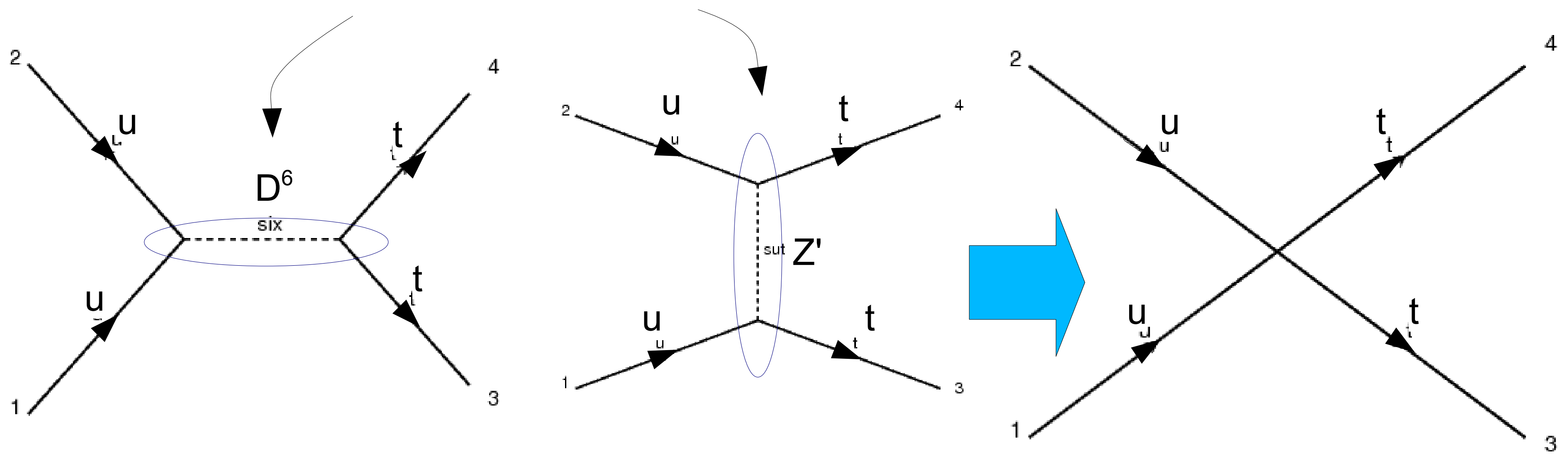
```
VVVVS1 = Lorentz(name = 'VVVVS1',
             spins = [ 3, 3, 3, 3, 1 ],
             structure = 'Metric(1,4)*Metric(2,3) - Metric(1,3)*Metric(2,4)')

VVVVS2 = Lorentz(name = 'VVVVS2',
             spins = [ 3, 3, 3, 3, 1 ],
             structure = 'Metric(1,4)*Metric(2,3) - Metric(1,2)*Metric(3,4)')

VVVVS3 = Lorentz(name = 'VVVVS3',
             spins = [ 3, 3, 3, 3, 1 ],
             structure = 'Metric(1,3)*Metric(2,4) - Metric(1,2)*Metric(3,4)')
```

Multi-fermion vertices

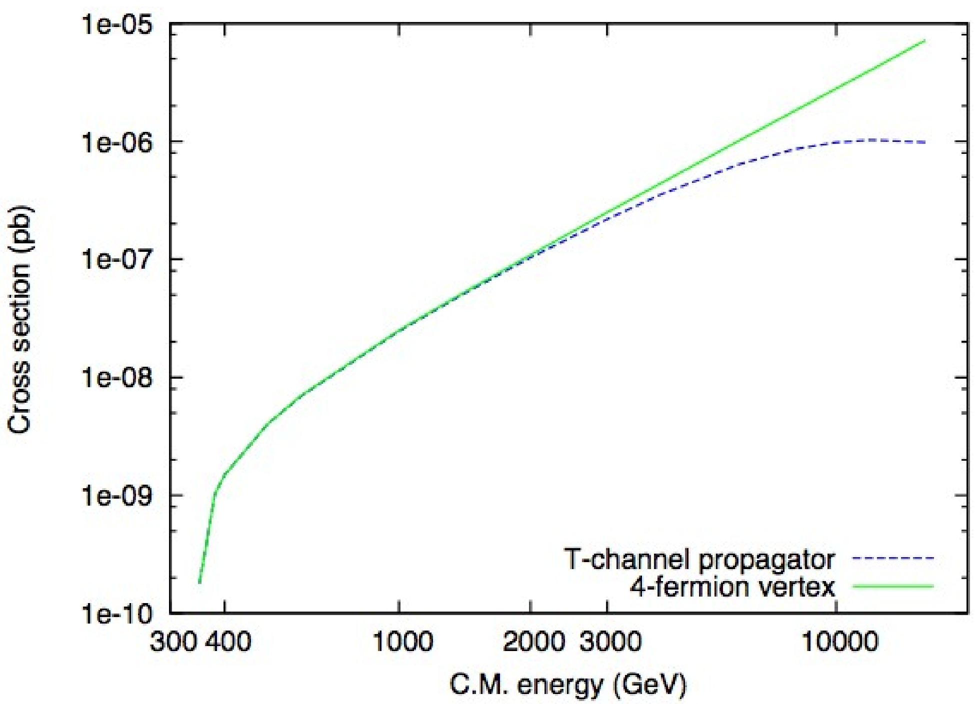
- Extra difficulty in multi-fermion vertices: Fermion flow
  - Using convention based on particle ordering as IOIOIO... (incoming-outgoing)
  - Example: $\bar{u}\ \bar{u}\ t\ t \neq \bar{u}\ t\ \bar{u}\ t$ (from MG point of view*)



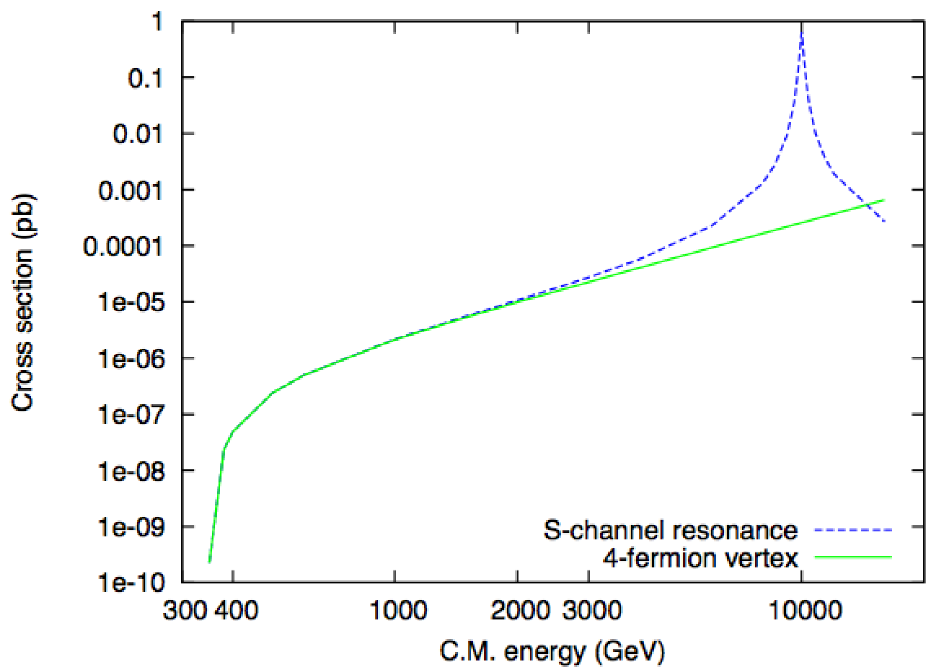
* Can interchange using Fierz identities

Multi-fermion vertices

Comparisons between explicit propagators and 4-fermion vertex



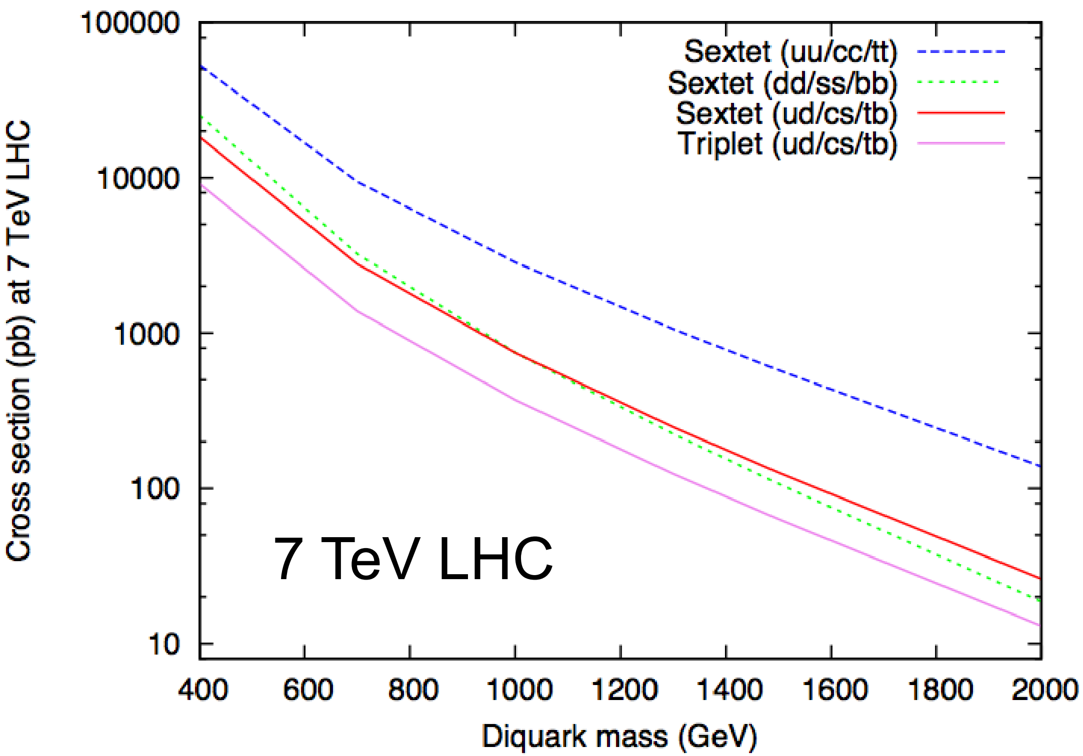
t-channel u u > t t



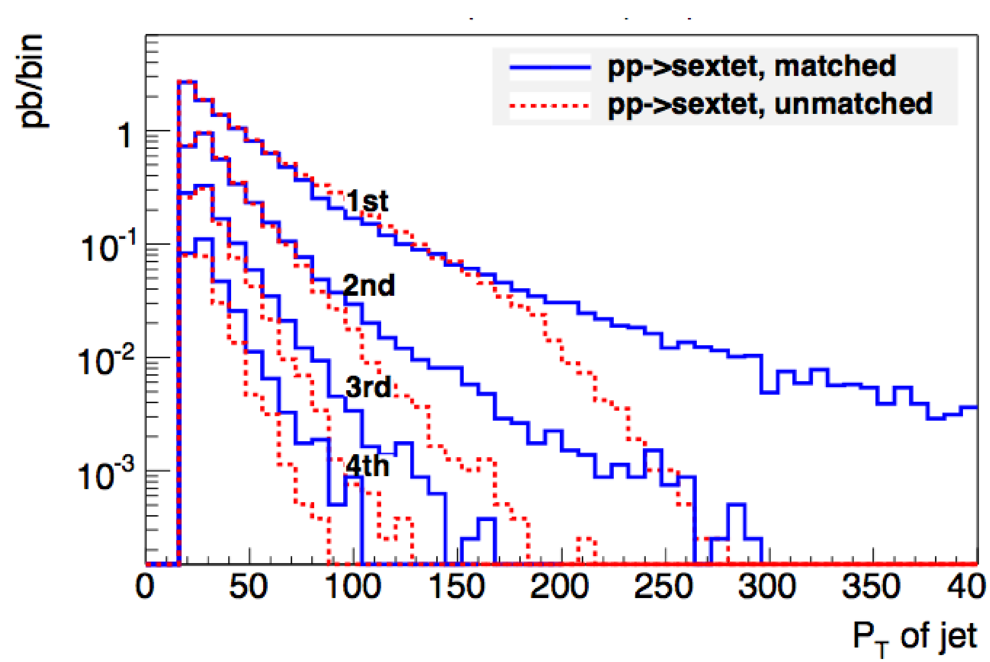
s-channel u u > t t

New color structures

- Implementation of diquark models q q > D
- Require 3 x 3 = 6 + $\bar{3}$, i.e. color sextet or ε^{ijk}
- Important also for RPV SUSY



Diquark cross sections with coupling 0.01



Jet p_T:s, fully matched
pp → D + 0,1,2 jets

Pythia 8 matrix elements

Sigma_sm_qq_ttx.h

- Automatic generation of Pythia 8 processes by MadGraph 5
- Creates process library of .h and .cc files in format used by Pythia, together with example main file
- Run as standard internal Pythia process
- Allows using Pythia for any (2→1,2,3) process in any model at the push of a key!

```
#include "SigmaProcess.h"
#include "Parameters_sm.h"
using namespace std;
namespace Pythia8
{
//==========================================
// A class for calculating the matrix elements for
// Process: u u~ > t t~ QED=0 @1
// Process: c c~ > t t~ QED=0 @1
// Process: d d~ > t t~ QED=0 @1
// Process: s s~ > t t~ QED=0 @1
//--------------------------------------------------------------------

class Sigma_sm_qq_ttx : public Sigma2Process
{  public:

   // Constructor.
   Sigma_sm_qq_ttx() {}
  // Initialize process.
   virtual void initProc();
   // Calculate flavour-independent parts of cross section.
   virtual void sigmaKin();
   // Evaluate sigmaHat(sHat).
   virtual double sigmaHat();
   // Select flavour, colour and anticolour.
   virtual void setIdColAcol();

...
```

Process	MG5-Pythia (mb)	Pythia (mb)	Rel. difference
g g > g g	5.92E-001	5.92E-001	0.0000
g g >q qbar	1.54E-002	1.55E-002	0.0036
g q > g q	3.22E-001	3.22E-001	0.0003
q qbar > g g	3.87E-004	3.85E-004	0.0018
q q > q q	3.24E-002	3.22E-002	0.0026
g g > b bbar	3.61E-003	3.59E-003	0.0026
(q q > b bbar	7.67E-005	7.90E-005	0.0149)
g g > t tbar	5.38E-007	5.38E-007	0.0003
q qbar > t tbar	8.15E-008	8.29E-008	0.0086
q b> t q' (t-channel)	1.97E-007	1.83E-007	0.0364
q qbar' > t b (s-channel)	8.71E-009	8.47E-009	0.0137
q g > q gamma	1.82E-004	1.82E-004	0.0008
q qbar > g gamma	1.26E-005	1.28E-005	0.0063
q qbar > gamma gamma	8.55E-008	8.58E-008	0.0019
g g > gluino gluino	4.55E-009	4.55E-009	0.0000
q qbar > gluino gluino	2.02E-010	2.03E-010	0.0037

MG5 multiparton processes

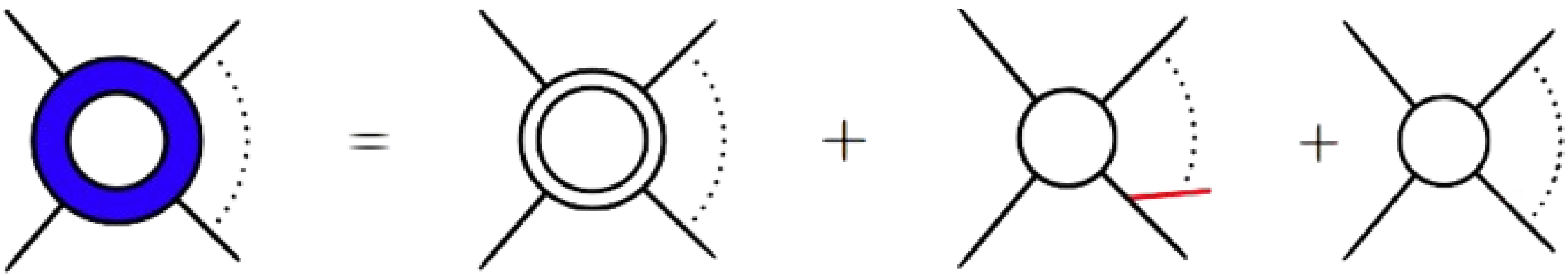
- New scheme for organizing MadEvent processes
  - Combine all processes and channels with same pole structures
  - p p → l^+l^- j j j: 5 subprocess directories, 368 integration channels (factor 11 fewer than MG/ME4)
    p p → W j j j j: factor 15 fewer channels than MG/ME4
  - Optimizations to phase space integration → reduced phase space integration time by factor 3
- Ongoing: Multi-parton matrix element evaluation by recursion relations
  - Should allow up to W/Z+6 jets
  - Challenge: MadEvent-style phase space integration

MG5 multiparton processes

Process	Subprocess directories		Channels for survey		Directory size	
	ME 4	ME 5	ME 4	ME 5	ME 4	ME 5
$pp \to W^+j$	6	2	12	4	79 MB	35 MB
$pp \to W^+jj$	41	4	138	29	438 MB	64 MB
$pp \to W^+jjj$	73	5	1164	184	842 MB	110 MB
$pp \to W^+jjjj$	296	7	15029	1327	3.8 GB	352 MB
$pp \to l^+l^-j$	12	2	48	8	149 MB	44 MB
$pp \to l^+l^-jj$	54	4	586	58	612 MB	83 MB
$pp \to l^+l^-jjj$	86	5	5408	368	1.2 GB	151 MB
$pp \to l^+l^-jjjj$	235	7	63114	2500	5.3 GB	662MB
$pp \to t\bar{t}$	3	2	5	4	49 MB	39 MB
$pp \to t\bar{t}j$	7	3	45	25	97 MB	56 MB
$pp \to t\bar{t}jj$	22	5	417	188	274 MB	98 MB
$pp \to t\bar{t}jjj$	34	6	3816	1300	620 MB	209 MB

MadGraph@NLO

NLO | Virtual | Real | Born



$$\sigma^{\mathrm{NLO}} = \int_m d^{(d)}\sigma^V + \int_{m+1} d^{(d)}\sigma^R + \int_m d^{(4)}\sigma^B$$

MadGraph@NLO

[Frederix, Gehrmann, Greiner;
Frederix, Frixione, Maltoni, Stelzer]

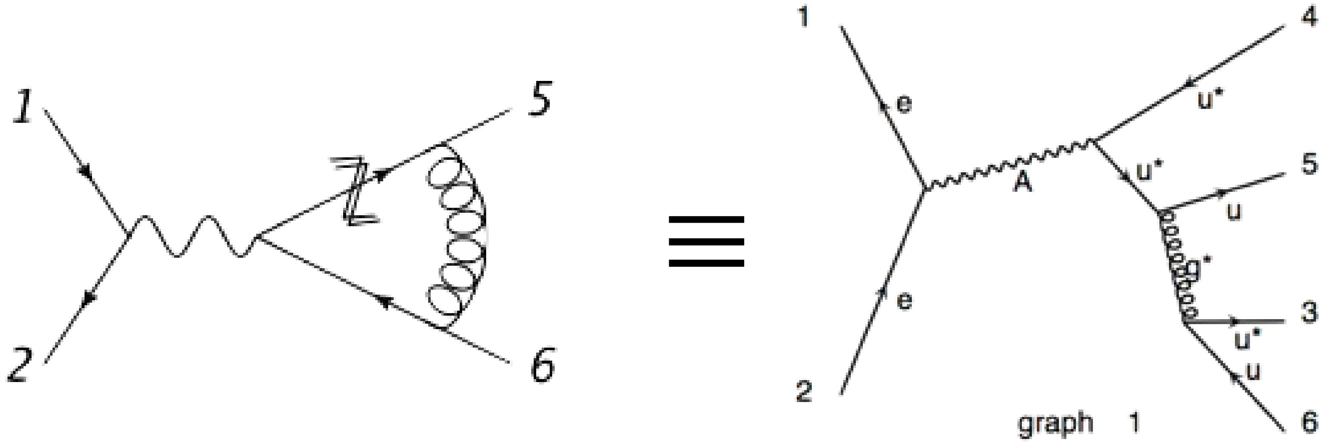
- Real contributions:
  - MadDipole: Catani-Seymour dipole substraction scheme, including integrated dipoles
  - MadFKS: Frixione-Kunszt-Signer substraction scheme (used for MC@NLO and POWHEG implementations). Including phase space integration for complete NLO calculation (with loop corrections from any package through Binoth-LH accord)
- Both: Handles both SM and BSM processes, and massless and massive external particles

MadGraph@NLO

- MadNLO virtuals

[Hirschi, Frederix, Frixione, Garzelli, Maltoni, Pittau arXiv:1103.0621]

  - Uses MadGraph to generate amplitudes for n+2 process, build loop amplitude using CutTools (OPP technique, by Ossola, Papadopoulos, Pittau)
  - Any SM process already possible, NLO for any BSM model in principle possible (see next point)
  - Work to automatize generation of necessary “R2” and UV counter terms for any model yet to be started



MadGraph@NLO

	Process	μ	n_{lf}	Cross section (pb) LO	NLO
a.1	$pp \to t\bar{t}$	m_{top}	5	123.76 ± 0.05	162.08 ± 0.12
a.2	$pp \to tj$	m_{top}	5	34.78 ± 0.03	41.03 ± 0.07
a.3	$pp \to tjj$	m_{top}	5	11.851 ± 0.006	13.71 ± 0.02
a.4	$pp \to t\bar{b}j$	$m_{top}/4$	4	25.62 ± 0.01	30.96 ± 0.06
a.5	$pp \to t\bar{b}jj$	$m_{top}/4$	4	8.195 ± 0.002	8.91 ± 0.01
b.1	$pp \to (W^+ \to)e^+\nu_e$	m_W	5	5072.5 ± 2.9	6146.2 ± 9.8
b.2	$pp \to (W^+ \to)e^+\nu_e\, j$	m_W	5	828.4 ± 0.8	1065.3 ± 1.8
b.3	$pp \to (W^+ \to)e^+\nu_e\, jj$	m_W	5	298.8 ± 0.4	300.3 ± 0.6
b.4	$pp \to (\gamma^*/Z \to)e^+e^-$	m_Z	5	1007.0 ± 0.1	1170.0 ± 2.4
b.5	$pp \to (\gamma^*/Z \to)e^+e^-\, j$	m_Z	5	156.11 ± 0.03	203.0 ± 0.2
b.6	$pp \to (\gamma^*/Z \to)e^+e^-\, jj$	m_Z	5	54.24 ± 0.02	56.69 ± 0.07
c.1	$pp \to (W^+ \to)e^+\nu_e b\bar{b}$	$m_W + 2m_b$	4	11.557 ± 0.005	22.95 ± 0.07
c.2	$pp \to (W^+ \to)e^+\nu_e t\bar{t}$	$m_W + 2m_{top}$	5	0.009415 ± 0.000003	0.01159 ± 0.00001
c.3	$pp \to (\gamma^*/Z \to)e^+e^- b\bar{b}$	$m_Z + 2m_b$	4	9.459 ± 0.004	15.31 ± 0.03
c.4	$pp \to (\gamma^*/Z \to)e^+e^- t\bar{t}$	$m_Z + 2m_{top}$	5	0.0035131 ± 0.0000004	0.004876 ± 0.000002
c.5	$pp \to \gamma t\bar{t}$	$2m_{top}$	5	0.2906 ± 0.0001	0.4169 ± 0.0003
d.1	$pp \to W^+W^-$	$2m_W$	4	29.976 ± 0.004	43.92 ± 0.03
d.2	$pp \to W^+W^-\, j$	$2m_W$	4	11.613 ± 0.002	15.174 ± 0.008
d.3	$pp \to W^+W^+\, jj$	$2m_W$	4	0.07048 ± 0.00004	0.1377 ± 0.0005
e.1	$pp \to HW^+$	$m_W + m_H$	5	0.3428 ± 0.0003	0.4455 ± 0.0003
e.2	$pp \to HW^+\, j$	$m_W + m_H$	5	0.1223 ± 0.0001	0.1501 ± 0.0002
e.3	$pp \to HZ$	$m_Z + m_H$	5	0.2781 ± 0.0001	0.3659 ± 0.0002
e.4	$pp \to HZ\, j$	$m_Z + m_H$	5	0.0988 ± 0.0001	0.1237 ± 0.0001
e.5	$pp \to Ht\bar{t}$	$m_{top} + m_H$	5	0.08896 ± 0.00001	0.09869 ± 0.00003
e.6	$pp \to Hb\bar{b}$	$m_b + m_H$	4	0.16510 ± 0.00009	0.2099 ± 0.0006
e.7	$pp \to Hjj$	m_H	5	1.104 ± 0.002	1.036 ± 0.002

[Hirschi, Frederix, Frixione, Garzelli, Maltoni, Pittau arXiv:1103.0621]

MadGraph@NLO

- Automatic MC@NLO [Frederix, Frixione, Torrielli]
  - Inclusion of counterterms for parton shower allows automatization of MC@NLO within the MadFKS framework
  - Work recently started, very promising results so far
- Move to MadGraph 5 [J.A., Frederix, Hirschi, Zaro]
  - Work started March 2011 (JA + V. Hirschi)
  - Huge optimizations, more straightforward implementations
  - Take advantage of upcoming improvements in matrix element speed (recursion relations)
  - Improved/facilitated organization of produced code
  - Full BSM functionality of MG5

FAQ

- "Do I need to learn Python now?"
  - Not unless you want to make additions to the core code of MadGraph. The standard MadGraph output is still Fortran (or C++), and MadEvent works as before.
- "Can I use jet matching with my favorite new physics model?"
  - Yes you can! Matching between MadEvent and Pythia works as before, for any model.
- "Can I use my old MadGraph models with MG5?"
  - Yes you can! MadGraph 5 is fully backward compatible for both models and cards.

FAQ (cont.)

- "Will there be a MadEvent in Python?"
  - No, MadEvent stays Fortran, but is continuously developed to make optimal use of new functionality of MadGraph 5.
- "Where can I get all this amazing stuff?"
  - At https://launchpad.net/madgraph5
    (or simply google for MadGraph 5)
  - Release (as tar ball) and development code (using the Bazaar versioning system)

Conclusions

- MadGraph 5 is now a full-fledged replacement for older MadGraph versions, with important and unprecedented improvements in all directions.
- MadGraph 5 v. 0.7 available, 1.0.0 (including new MadEvent) released in the coming days
- Stay tuned for more exciting stuff in the near future (more NLO, fast multijet, BSM decay package, FeynRules developments, …)

Don't miss upcoming MadGraph developer workshop at Fermilab (May 3-6, 2011)

Backup slides

Process checks

Suite of powerful checks on internal consistency of model implementation (including UFO/ALOHA/MG5) internally in MG5

- Generation of ME using multiple different orders of the wavefunction calls
- Gauge invariance for massless vector bosons
- Lorentz boost invariance of ME

Decay chains

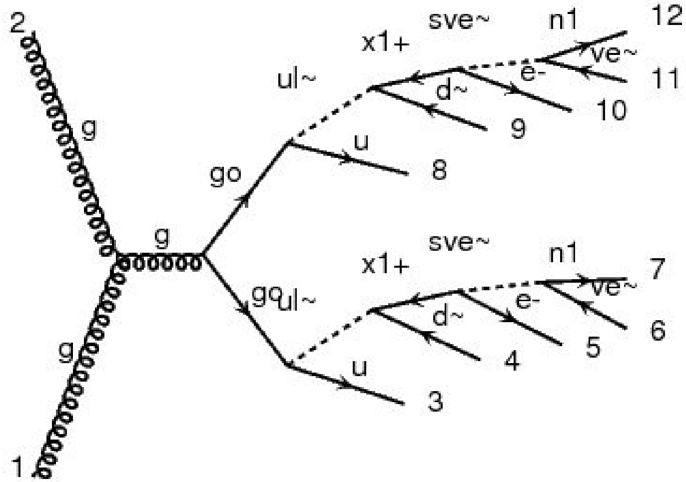


diagram 1

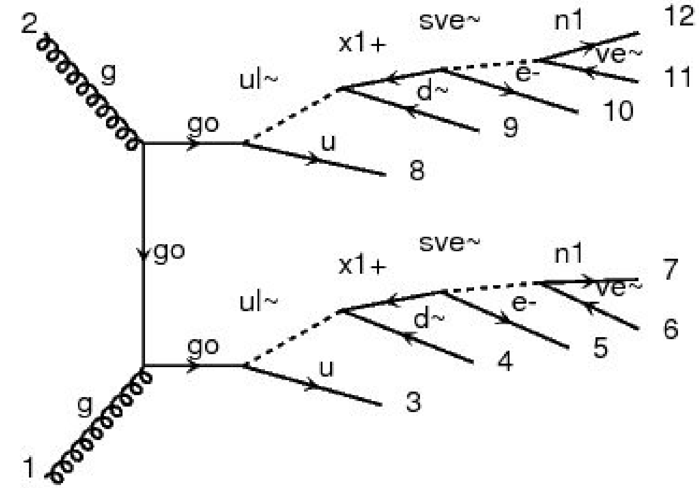


diagram 2

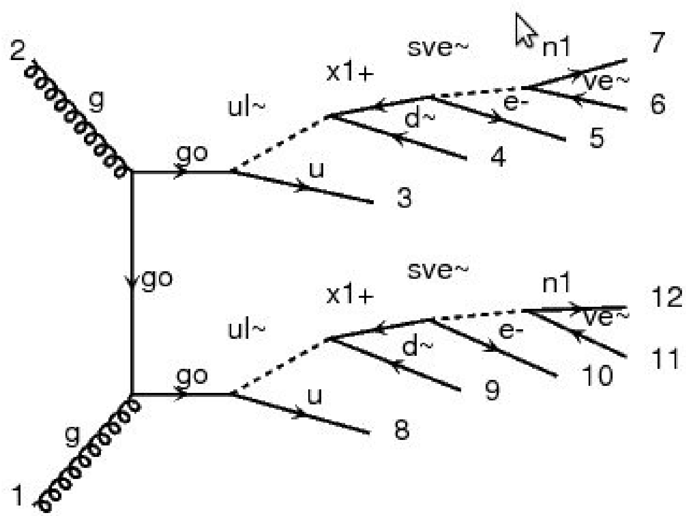


diagram 3

Process: g g > go go
  Decay: go > u ul~
    Decay: ul~ > d~ x1-
      Decay: x1- > e- sve~
        Decay: sve~ > ve~ n1
  Decay: go > u ul~
    Decay: ul~ > d~ x1-
      Decay: x1- > e- sve~
        Decay: sve~ > ve~ n1
(10 FS particles. Generation time: 5 s)